ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH IN ELECTRICAL, ELECTRONICS, INSTRUMENTATION AND CONTROL ENGINEERING
Vol. 3, Issue 3, March 2015

Efficient FPGA Implementation of Variable precision floating point Arithmetic using VHDL Language

Mr. Varun[1], Dr. Vinod Kapse[2]

M Tech (VLSI Design), GGITS, Jabalpur[1]

Head EC, GGITS, Jabalpur[2]

Abstract*:* Computers were originally built as fast, reliable and accurate computing machines. It does not matter how large computers get, one of their main tasks will be to always perform computation. Most of these computations need real numbers as an essential category in any real world calculations. Real numbers are not finite; therefore no finite, representation method is capable of representing all real numbers, even within a small range. Thus, most real values will have to be represented in an approximate manner. The scope of this paper includes study and implementation of Adder/Subtractor and Multiplication ,Division and Square root functional units using HDL for computing arithmetic operations and functions suited for hardware implementation. In this paper, we present pipelined architecture to implement Variable bit floating point Arithmetic in Field Programmable Gate Array (FPGA). Specially we designed square root of floating point numbers using modified non restoring square root algorithm. This algorithm has been optimized by eliminating a number of elements without compromising the precision of the square root and the remainder. The algorithms are coded in VHDL and validated through extensive simulation. These are structured so that they provide the required performance i.e. speed and gate count. It is an improvement over non restoring algorithm as it uses only subtract operation and append 01 instead of add operation and append 11. Here the basic building block is Controlled Subtract Multiplex (CSM). By using this module, the algorithm can be designed for any number of input bits. This strategy offers an efficient use of hardware resources. The modified non restoring algorithm is designed using VHDL and implemented on ALTERA cyclone II FPGA. The implementation results show reduced area in terms of logic elements when compared to restoring algorithm.

Keywords: ALTERA cyclone II FPGA, CSM, VHDL

I. INTRODUCTION

Nowadays, floating point Arithmetic is very important for several applications in digital signal and image processing, computer graphics and scientific computing . But division and square root is the most time consuming operation among four arithmetic operations.

Designing a high-speed reciprocal unit is very useful for division operation because the division can be replaced as the following method: the reciprocal of divisor is computed at first, and then it is used as the multiplier in a subsequent multiplication with the dividend.

If several divisions by the same divisor need to be performed, this method is particularly efficient, since once the reciprocal of divisor is found for the first division, each subsequent division involves just one additional multiplication.

The square root function is widely used in computer graphics, image and signal processing, statistics, communications and scientific calculation applications.

Due to complications involved in implementation of square root algorithms, its design in digital system has always been a bottleneck. The basic operations like addition and subtraction are easy to implement in an FPGA because synthesis tools have optimized addition/subtraction units based on FPGA architecture. Multiplication, division and square root are complex operations; square root in particular, is computationally intensive as it involves convergence and approximation techniques.

Many algorithms/methods have been developed to implement it on FPGAs. But there is a need of an algorithm which should be more efficient in terms of time, speed and on-chip area.

During the recent years field programmable gate arrays (FPGA's) have become the dominant form of programmable logic. In comparison to previous programmable devices like programmable array logic (PAL) and complex programmable logic devices (CPLD's), FPGA's can implement far larger logic functions.

FPGA's supports sufficient logic to implement complete systems and sub-systems. FPGA exploit the increasing capacity of integrated circuits to provide designers with reconfigurable logic that can be programmed on application-specific basis.

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH IN ELECTRICAL, ELECTRONICS, INSTRUMENTATION AND CONTROL ENGINEERING
Vol. 3, Issue 3, March 2015

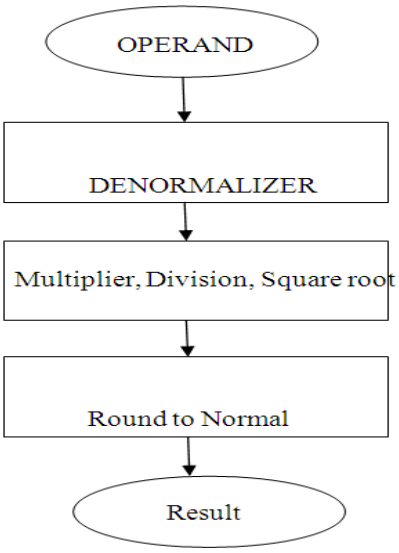

Fig 1. Flow graph of Proposed design

In this paper, we study restoring and modified non restoring square root algorithm and implement them using pipelined architecture in VHDL. This approach uses CSM as a basic building block. The block has been modified for implementing restoring algorithm. These blocks have been optimized to reduce the number of logic cells utilized. The performance is compared based on logic elements and power consumption using ALTERA Stratix-II FPGA.

Variable Length Floating Point Addition/subtraction Module

Floating Point Addition is one of the complex unit in the floating point arithmetic operations. Addition / subtraction is the most basic arithmetic operation. The hardware implementation of this arithmetic for floating point numbers is a complicated operation due to the requirement of normalization. A proposed implementation method of Variable bit floating point adder/subtractor has been shown here. The flowchart for Floating Point adder/subtractor is shown in Fig. 1. Here the term *addition* is used to refer to both *addition* and *subtraction* as the same hardware is used in both cases.

The steps for computing addition of two floating point numbers proceeds as follows,

1. Compare exponents and mantissa of both numbers. Decide large exponent & mantissa and small exponent & mantissa.
2. Right shift the mantissa associated with the smaller exponent, by the difference of exponents.
3. Add both mantissa if signs are same else subtract smaller mantissa from large one.
4. Do the rounding of the result after mantissa addition.
5. If the subtraction results in loss of most significant bit (MSB), then the result must be normalized. To do this, the most significant non-zero entry in the result mantissa must be shifted until it reaches the front. This is accomplished by a "Leading one detector (LOD)" followed by a shift.
6. Do normalization and adjust large exponent accordingly.
7. Final result includes sign of larger number, normalized exponent and mantissa.

Implementation of Floating Point Multiplier

Floating Point Multiplication Algorithm

Multiplying two numbers in floating point format is done as follows.

1. Adding the exponent of the two numbers then subtracting the bias from their result.
2. Multiplying the significand of the two numbers.
3. Calculating the sign by XORing the sign of the two numbers.

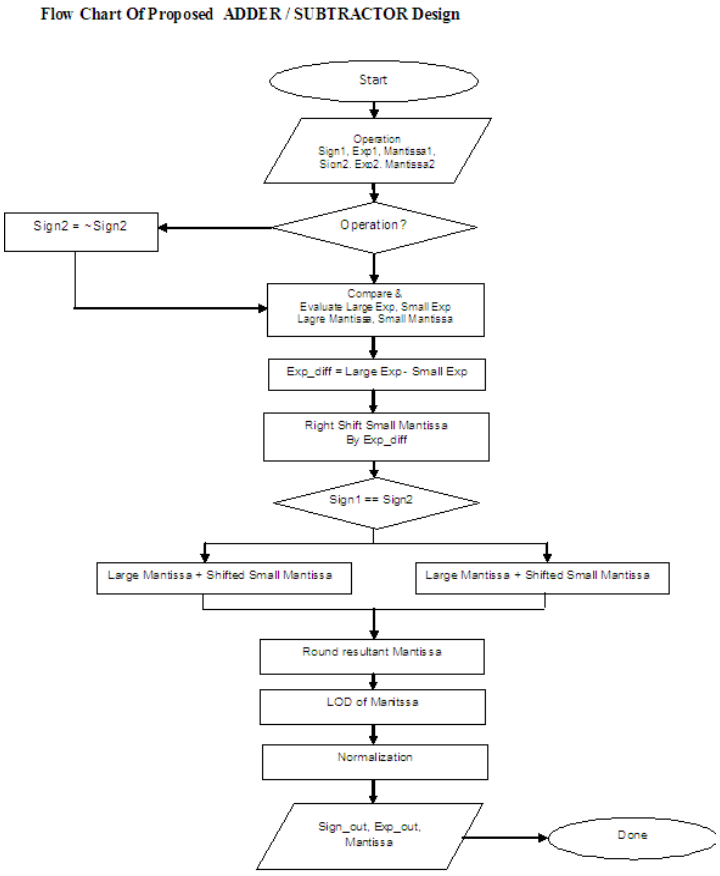

Fig. 2 Flow Chart of Variable Length Floating Point Addition/ subtraction Module

In order to represent the multiplication result as a normalized number there should be 1 in the MSB of the result (leading one).The following steps are necessary to multiply two floating point numbers.

1. Multiplying the significand, *i.e.*, (1.M1*1.M2) .
2. Placing the decimal point in the result .
3. Adding the exponents, *i.e.*, (E1 + E2 – *Bias*) .
4. Obtaining the sign i.e. s1 xor s2 .
5. Normalizing the result, *i.e.*, obtaining 1 at the MSB of the results "significand" .
6. Rounding the result to fit in the available bits .
7. Checking for underflow/overflow occurrence.

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH IN ELECTRICAL, ELECTRONICS, INSTRUMENTATION AND CONTROL ENGINEERING
Vol. 3, Issue 3, March 2015

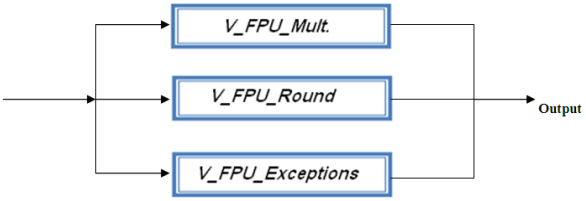

Fig. 3 Multiplier structure with rounding and exceptions

To Perform the Division Operation in VHDL

The algorithm for floating point multiplication is explained through flow chart in Figure 4. Let N1 and N2 are normalized operands represented by S1, M1, E1 and S2, M2, E2 as their respective sign bit, mantissa (significand) and exponent. If let us say we consider x=N1 and d=N2 and the final result q has been taken as "x/d".

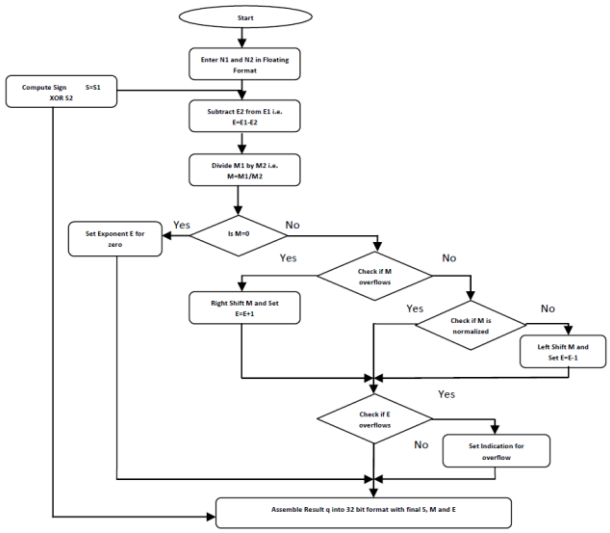

Fig 4. Flow Chart of Division Operation

Again the following four steps are used for floating point division.

1. Divide signifcands, subtract exponents, and determine sign
M=M1/M2
E=E1-E2
S=S1XORS2
2. Normalize Mantissa M (Shift left or right by 1) and update exponent E
3. Rounding the result to fit in the available bits
4. Determine exception flags and special values

The sign bit calculation, mantissa division, exponent subtraction (no need of bias subtraction here), rounding the result to fit in the available bits and normalization is done in the similar way as has been described for multiplication.

Pipelining is a well known technique for achieving faster clock rates while sacrificing latency. Pipeling offers an economic way to realize temporal parallelism in digital systems. To achieve pipelining one must subdivide the input process into a sequence of subtasks each of which can be executed by specialized hardware stage that operates concurrently with other stages in the pipeline. In the present method, five stage pipelining is incorporated for faster performance.

The square root algorithm

A small modification in non restoring algorithm makes calculation faster. It uses only subtract operation and appends "01". It uses n stage pipelining to find square root of 2n bit number. The following algorithm describes the modified non restoring square root algorithm.

Step 1: Start
Step 2: Initialize the radicand (M) which is 2n bit number.
Divide the radicand in two bits beginning at decimal
point in both directions.
Step 3: Beginning on the left (most significant), select the first
group of one or two bits. (If n is odd then first group
has one bit, else two bits.)
Step 4: Select the first group of bits and subtract" 01" from it.
If borrow is zero, result is positive then quotient is 1
otherwise it is 0.
Step 5: Append 01(to be subtracted next two bits of dividend)
and quotient to subtract from remainder of previous
stage.
Step 6: If result of subtraction is negative, write previous
remainder as it is and quotient is considered as 0, else
write the difference as remainder and quotient as 1.
Step 7: Repeat step 5 and step 6 until end group of two bits.
Step 8: End.

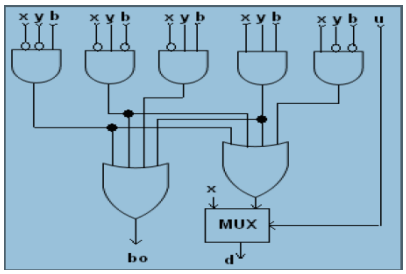

Fig 5. Internal structure of CSM block

From Fig. it is clear that Controlled Subtract Multiplex(CSM) is a combination of full subtractor and 2:1 multiplexer. Multiplexer is used to select one of the inputs based on one bit quotient which acts as a select line for multiplexer.

From the algorithm, it can be concluded that if the result of subtraction is negative (which will set the borrow bit to 1), quotient "u" is selected as 0 which ultimately selects the input "*x*" for the next iteration. Also if the result of subtraction is positive which gives 0 as a borrow, quotient

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH IN ELECTRICAL, ELECTRONICS, INSTRUMENTATION AND CONTROL ENGINEERING
Vol. 3, Issue 3, March 2015

is selected as 1. This sets select line to 1 which gives difference "d" as input for next iteration.

Inputs to the CSM block are *x, y, b* and u whereas *d* and *b0* (borrow) are outputs.

If $b0 = 0$,

then $d <= x\text{-}y\text{-}b$ else $d <= x$.

As CSM includes full sub tractor having *x*, *y* and *b* as its inputs, it works as follows:

1) It performs subtraction : $x\text{-}y\text{-}b$

2) If the result of subtraction is positive, we get $b0 = 0$, $u = 1$ and $d = x\text{-}y\text{-}b$

3) If the result of subtraction is negative, we get $b0 = 1$,

The generalization of simple implementation of modified non restoring digit by digit algorithm for unsigned 6 bit square root is shown in Fig. For 6 bit input number, 3 bit quotient(u2u1u0) is obtained as answer.

Each row of the circuit executes one iteration of non restoring digit by digit square root algorithm, where it only uses subtract operation and appends „01". In the pipelined structure using CSM block, some input patterns are fixed. So , the design can be optimized by minimizing the Boolean equations of b0 and d. It can be implemented by modifying CSM block

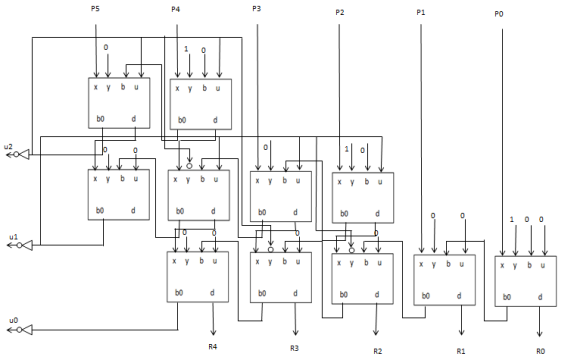

Fig. 6. Pipelined structure of 6 bit unsigned square root number.

EXPERIMENTAL RESULT

The proposed architecture was implemented using VHDL and synthesized on a Altera Stratix II FPGA with simulation on the Quartus-II . Some sample result is shown in Tab1.

Table 4.1: Area, Latency and Throughput for Floating-Point Division

F P Format	8(2,5)	16(4,11)	24(6,17)	32(8,23)	40(10,29)
slices	66 (1%)	115 (1%)	281 (1%)	361 (1%)	617 (1%)
Block RAMs	1 (1%)	1 (1%)	1 (1%)	7 (4%)	62 (43%)
embdd multipliers	2 (1%)	2 (1%)	8 (5%)	8 (5%)	8 (5%)
clock period (ns)	4.9	6.8	7.8	7.7	8.0
max freq. (MHz)	202	146	129	129	125
latency (clk cycle)	10	10	14	14	14
latency (ns)	49	68	109	108	112

Table 4.1: Area, Latency and Throughput for Floating-Point Square root.

F P Format	8(2,5)	16(4,11)	24(6,17)	32(8,23)	48(10,38)
slices	85 (1%)	172 (1%)	308 (1%)	351 (1%)	779 (2%)
Block RAMs	3 (2%)	3 (2%)	3 (2%)	3 (2%)	13 (9%)
embdd multiplier	4 (2%)	7 (4%)	9 (6%)	9 (6%)	16 (11%)
clock period (ns)	6.1	7.2	7.8	8.0	8.8
max freq. (MHz)	165	139	129	125	114
latency (clk cle)	9	12	13	13	16
latency (ns)	55	86	101	104	140

Comparison of our Floating-Point Divide and Square Root and Xilinx Floating-Point IP Cores

The algorithms we use are not digit-recurrence as are most other divider and square root implementations. As a result, the latency does not increase linearly as the data bitwidth grows. Our goal is to keep the clock period relatively constant over a wide range of bitwidths and formats. Therefore we have to add more pipeline stages for wider bitwidth formats as a compromise. This results in a slightly increasing latency with the increasing bitwidth. Thus, the latency of our floating-point divide and square root is very short compared to most other divider and square root implementations

CONCLUSION

This paper presents restoring and modified non restoring algorithm for variable bits arithmetic include addition, subtraction, multiplication, division and square root calculation. The proposed design result will be an accurate as far as the the output is concern. I try to optimized non restoring algorithm to reduces on chip area and pipelining will increases the speed performance. The result will be extended for square root implementation of 64 bit floating point number and also it can be expanded to larger numbers to solve complicated square root problem in FPGA implementation.

REFERENCES

[1] P. C. Diniz and G. Govindu. Design of Field-Programmable Dual-precision Floating-Point Arithmetic Units. In Proceedings of the 16th international conference on field-programmable logic and applications (FPL'06), pages 733–736, August 2006.

[2] J. Janhunen, P. Salmela, O. Silv´en, and M. Juntti,"Fixed- versus floating-point implementation of MIMOOFDM detector," in Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing, Prague, Czech Republic,May 22–27 2011, pp. 3276–3279.

[3] T. M. Bruintjes, K. H. G. Walters, S. H. Gerez, B. Molenkamp, and G. J. M. Smit, "Sabrewing: A lightweight architecture for combined floating-point and integer arithmetic," ACM Trans. Archit. Code Optim.vol. 8, no. 4, pp. 41:1–41:22, Jan. 2012.

[4] IEEE Standards Board and ANSI. *IEEE Standard for Binary Floating-Point Arithmetic*.IEEE Press, 1985. IEEE Std 754-1985.

[5] G. Govindu, L. Zhuo, S. Choi, and V. Prasanna, "Analysis of High-*Parallel and Distributed.Processing Symp.*, pp. 149b, April 2004.

[6] M. K. Jaiswal and R. C. Cheung. High performance reconfigurable Reconfigurable Computing:Architectures, Tools and Applications, pages 302–313. Springer, 2012.

[7] Altera Corp. Stratix v website. http:// www.altera.com/devices/fpga/stratix-fpgas/stratix-v/stxv-index.jsp.