International Journal of Research in Engineering and Science (IJRES)
ISSN (Online): 2320-9364, ISSN (Print): 2320-9356
www.ijres.org Volume 2 Issue 5 ǁ May. 2014 ǁ PP.14-20

A High Speed Transposed Form FIR Filter Using Floating Point Dadda Multiplier

Dhivya.V.M[1], Sridevi.A[2]

[1](ECE, SNS College of Technology/Anna University, Coimbatore, India)
[2](ECE, SNS College of Technology/Anna University, Coimbatore, India)

ABSTRACT*: There is a huge demand in high speed area efficient parallel FIR filter using floating point dadda algorithm, due to increase performance of processing units. Area and spped are usually confictiong constraints so that improving speed results mostly in large areas. In our research we will try to determine the best solution to this problem by comparing the results of different multipliers. Different sized of two algorithm for high speed hardware multipliers were studied and implemented ie.dadda and booth multipliers. The working of these two multipliers were studied and implementing each of them separately in VHDL. The results of this research will help us to choose the better option between multipliers for floating point multiplier for fabricating different system.*

KEYWORDS - *Booth multiplier, Dadda multiplier, FIR Filter, Floating point multiplier, VHDL*

I. INTRODUCTION

FIR filters are one of two primary types of digital filters used in Digital signal Processing (DSP) applications. Main approach is to implement high throughput FIR filter .multiplier block reduction has attempted to minimize various cost functions such as number of adders and multipliers block area by sub expression elimination. Attention has been paid to identify ways to reduce power consumption of high throughput FIR implementation based on FPGA. Implementations based on FPGA hardware are better suited because of the likely need for frequent modifications. To increase the effective throughput of original filter or to reduce the power consumption of original filter parallel or block processing has been applied to digital FIR filter. In many design situation the overhead hardware incurred by parallel processing cannot be tolerated due to design area limitation. Therefore it is beneficial to realize parallel FIR filter structure that consumes less area than the traditional one. Sub modules used will be Floating point multiplier and Error tolerant adder (ETA). ETA includes sub modules i.e modified XOR gate and Ripple Carry Adder. Ripple Carry Adder is used because it requires less hardware. ETA cause less power consumption due to the elimination of carry propagation to a large extent.

1.1. Applications

FIR filters are used as a fundamental processing element in any DSP system. FIR filters are used in DSP applications ranging from video and image processing to wireless communications[2]. In the application of video processing, the FIR filter circuit must be able to operate at high frequencies, while in other applications, such as cellular telephony, the FIR filter circuit must be a low-power circuit, capable of operating at moderate frequencies.

1.2 FIR Filters

FIR Filters are the backbone of DSP system. FIR means ―Finite Impulse Response. If impulse is input, that is, a signal 1 sampled followed by many 0 samples, zeroes will come out after the 1 sample has made its way through the delay line of filter. The impulse response is finite because there is no feedback in the FIR. However, if feedback is employed yet the impulse response is finite, the filter still is a FIR. Example is the Moving Average Filter, in which the Nth prior sample is feed back then each time a new sample comes in. This filter has a finite impulse response even though it uses feedback : after N samples of an impulse, the output will always be zero. Alternative to FIR filters are ―Infinite Impulse Response (IIR). IIR filters use feedback, so when you input an impulse the output theoretically rings indefinitely. The advantages of FIR filters outweigh the disadvantages, so they are used much more than IIRs.

1.3 Floating Point Multiplier

Multipliers are the major components of high performance systems used extensively in digital electronics such as microprocessors, digital signal processors and FIR Filters etc. The performance of any

system is determined by the performance of multipliers because they are the slowest part in the system. Moreover, they require greater area than other components. Therefore, optimizing the speed and area of the multipliers is the foremost issue. As area and speed are both conflicting constraints this means that for greater speed we need larger area. A number of algorithms are proposed and used to design multipliers and the actual implementation is mostly some little refinements and variations of the few basic algorithms presented here. In addition to choosing those algorithms for addition, subtraction, multiplication etc an architect must make other decisions like how exceptions should be handled and what precisions should be implemented. We have designed a type of multiplier floating point .Our discussion on floating point will focus almost exclusively on the IEEE floating-point standard because of its rapidly increasing acceptance. Although floating point arithmetic involves manipulating exponents and shifting fractions, the bulk of the time in floating point is spent operating on fractions using integer algorithms. Thus, after our discussion of floating point we will take a more detailed look at efficient algorithms and architectures. When two binary numbers, multiplicand "N" and multiplier "M" are multiplied the algorithm utilizes distributive property of multiplication. The multiplication of M*N consists of partial products which represents a single component of the total product . A binary multiplier can be decomposed in to a sum of partial products. It can be done by selection of a partial product value, and some shifting that is independent of the election value. Opting a group length for multiplication limit us to make a trade-off between the number of partial products and the complexity of computing them.

II. SYSTEM MODEL

2.1 Booth Multiplier

Booth algorithm provides a procedure for multiplying binary integers in signed-2's complement representation [1]. According to the multiplication procedure, strings of 0's in the multiplier require no addition but just shifting and a string of 1's in the multiplier from bit weight 2k to weight 2m can be treated as 2k+1 - 2m. Booth algorithm involves recoding the multiplier first. In the recoded format, each bit in the multiplier can take any of the three values: 0, 1 and -1.Suppose we want to multiply a number by 01110 (in decimal 14). This number can be considered as the difference between 10000 (in decimal 16) and 00010 (in decimal 2). The multiplication by 01110 can be achieved by summing up the following products:

- 24 times the multiplicand(24 = 16)
- 2's complement of 21 times the multiplicand (21 = 2).

In a standard multiplication, three additions are required due to the string of three 1's.This can be replaced by one addition and one subtraction. The above requirement is identified by recoding of the multiplier 01110 using the following rules summarized in table 1.

Table 1: Radix 2 recoding rules

Qn	Qn+1	Recoded bits	Operation performed
0	0	0	Shift
0	1	+1	Add M
1	0	-1	Subtract M
1	1	0	Shift

To generate recoded multiplier for radix-2, following steps are to be performed:

- Append the given multiplier with a zero to the LSB side
- Make group of two bits in the overlapped way

Recode the number using the above table. Consider an example which has the 8 bit multiplicand as 11011001 and multiplier as 011100010.

```
        Multiplicand 1 1 0 1 1 0 0 1
        Multiplier   0 1 1 1 0 0 0 10
Recoded multiplier +1 0 0 -10 0+1-1
                     0 0 0 1 0 0 1 1 1
                     1 1 1 0 1 1 0 0 1
                     0 0 0 0 0 0 0 0 0
                     0 0 0 0 0 0 0 0 0
                     0 0 0 1 0 0 1 1 1
                     0 0 0 0 0 0 0 0 0
                     1 1 1 0 1 1 0 0 1
        Product 0000001001001001
```

III. PROPOSED FLOATING-POINT DADDA MULTIPLIER

3.1 IEEE Standard 754 Floating Point Numbers

Floating point describes a method of representing real numbers in a way that can support a wide range of values. The base for scaling is normally 2, 10, 16. The representation is exactly have the form:

Significant digits × base ^ (exponent)

The term floating point refers to the fact that the radix point (decimal point or in computers, binary point) can ―float□ ; thatis, it can be placed anywhere relative to the significant digits of the number. The advantage of floating point representation over fixed-point and integer representation is that it can support a much wide range of values eg. Fixed-point representation has seven decimal digits with two decimal places can represent the number 34565.67, 145.12 , 1.45 and so on, whereas a floating point representation (such as IEEE 754decimal 32 format) with seven decimal digits could in addition represent 1.324765, 234516.7, 0.00003455687, 4327651000000000, and so on. The floating-point format needs slightly more storage (in order to encode the position of the radix point). So when stored in the same space, floating-point numbers achieve their greater range at the expense of precision. The Fig. 2 shows the block diagram of floating point multiplier. Initially the input values are unpacked into sign-bit, exponents and significands and applied to the sub-modules. The sign of the multiplication result is obtained by EX-OR operation on the input sign-bits. The exponents of the inputs are added and bias value is

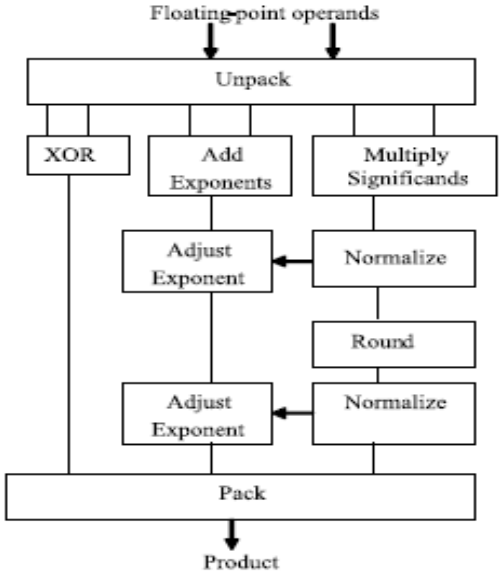

Fig. 2. Block diagram of Floating-point Multiplier.

subtracted from the result (in order to make the final exponent result in biased format). This resultant exponent is adjusted in the subsequent normalization steps and checked for overflow or underflow before final exponent is obtained. The *24×24* significand multiplication is the main block in which redundant computations take place frequently and is the main obstacle in achieving high-performance and low-power consumption in the filter design. The redundant computations can be reduced by identifying common computations and sharing them among filter taps. Since each alphabet is represented with *4*-bits, for the *24×24* significand multiplication we need six select & shift units. We have implemented an IEEE-754 compliant Floating point dadda based on the proposed architecture. Also, we have implemented booth multiplier for comparison purpose.

1) Result Sign: The result sign is obtained by XOR operation of the sign bits of the input operands.

2) Exponent Addition: We have used Carry-Look Ahead adders for the exponent addition and for the subtraction of bias (valued 127) from the added result

.

IV. DADDA MULTIPLIER IMPLEMENTATION

The Dadda algorithm reduces the tree by reducing columns instead of rows. The goal of the algorithm is to use the least amount of elements as possible. To accomplish this, the algorithm adds elements as late as possible. In the dot diagram notation developed by Dadda each partial product is represented by a dot[7]. The

dot diagram for an 8 by 8 Dadda multiplier is shown in Fig. 3. The eight rows of eight dots each at the top of the figure represent the partial product matrix formed by the AND gates.

Dadda multiplier use a minimal number of (3,2) and (2,2) counters at each level during the compression to achieve the required reduction. The reduction procedure for Dadda compression trees is given by the following recursive algorithm.

1. Let $d_1=2$ and $d_{j+1}= [1.5* d_j]$. D_j is the height of the matrix for the j^{th}stage. Repeat until the largest j^{th} stage is reached in which the original N height matrix contains at least one column which has more than d_j dots.
2. In the j^{th} stage from the end, place (3,2) and (2,2) counters as required to achieve a reduced matrix. Only columns with more than d_j dots or which will have more than d_j dots as they receive carries from less significant (3,2) and (2,2) counters are reduced.
3. Let j=j-1 and repeat step 2 until a matrix with a height of two is generated. This should occur when j-1

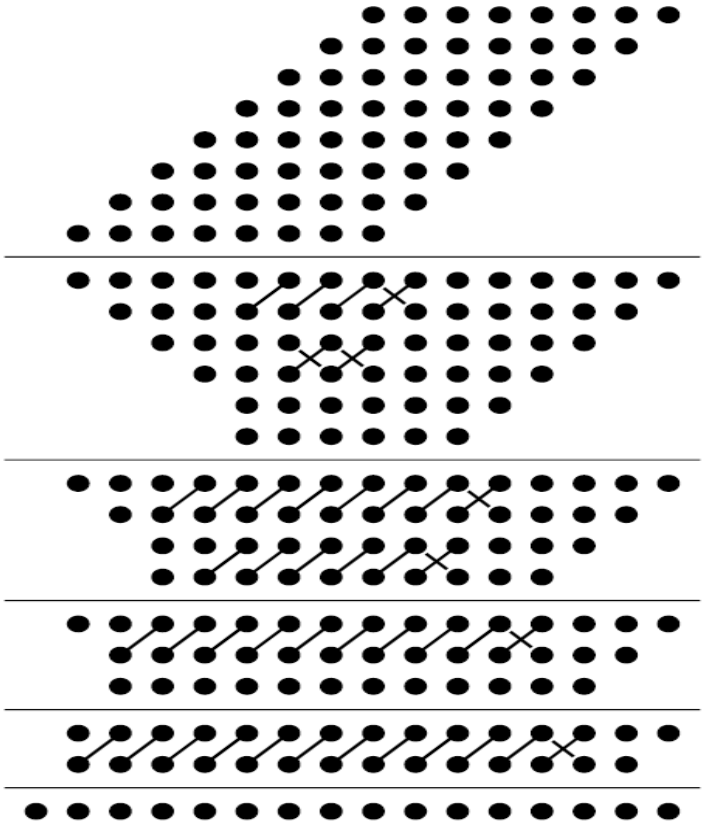

Fig. 3 Dot diagram for 8 by 8 Dadda multiplier

V. FIR FILTER IMPLEMENTATION USING FLOATING POINT DADDA MULTIPLIER

The N-tap FIR filter in Z domain is given in equation 1

$$\begin{aligned} Y_{(Z)} &= H_{(Z)}X_{(Z)} = \sum_{n=0}^{N-1} h(n)Z^{-n} \sum_{n=0}^{\infty} x(n)\,Z^{-n} \\ Y_{(Z)} &= Y_{0(Z^4)} + Z^{-1}Y_{1(Z^4)} + Z^{-2}Y_{2(Z^4)} + Z^{-3}Y_{3(Z^4)} \\ &= (H_0 + Z^{-1}H_0 + Z^{-2}H_2 + Z^{-3}H_3)\ (X_0 + Z^{-1}X_1 + Z^{-2}X_2 \\ &\quad + Z^{-3}X_3) \end{aligned} \tag{1}$$

Traditional 4- parallel FIR filter Equations shown in equation 2[7]

$$
\begin{aligned}
Y_{0(z^4)} &= X_{0(z^4)}H_{0(z^4)} + Z^{-4}X_{1(z^4)}H_{3(z^4)} \\
&\quad + Z^{-4}X_{2(z^4)}H_{2(z^4)} + Z^{-4}X_{3(z^4)}H_{1(z^4)} \\
Y_{1(z^4)} &= X_{0(z^4)}H_{1(z^4)} + X_{1(z^4)}H_{0(z^4)} + Z^{-4}X_{2(z^4)}H_{3(z^4)} \\
&\quad + Z^{-4}X_{3(z^4)}H_{2(z^4)} \\
Y_{2(z^4)} &= X_{0(z^4)}H_{2(z^4)} + X_{1(z^4)}H_{1(z^4)} + X_{2(z^4)}H_{0(z^4)} \\
&\quad + Z^{-4}X_{3(z^4)}H_{3(z^4)} \\
Y_{3(z^4)} &= X_{0(z^4)}H_{3(z^4)} + X_{1(z^4)}H_{2(z^4)} + X_{2(z^4)}H_{1(z^4)} \\
&\quad + X_{3(z^4)}H_{0(z^4)}
\end{aligned}
\tag{2}
$$

Requires 16 multiplications, 12 additions, 6 delay elements.[7]

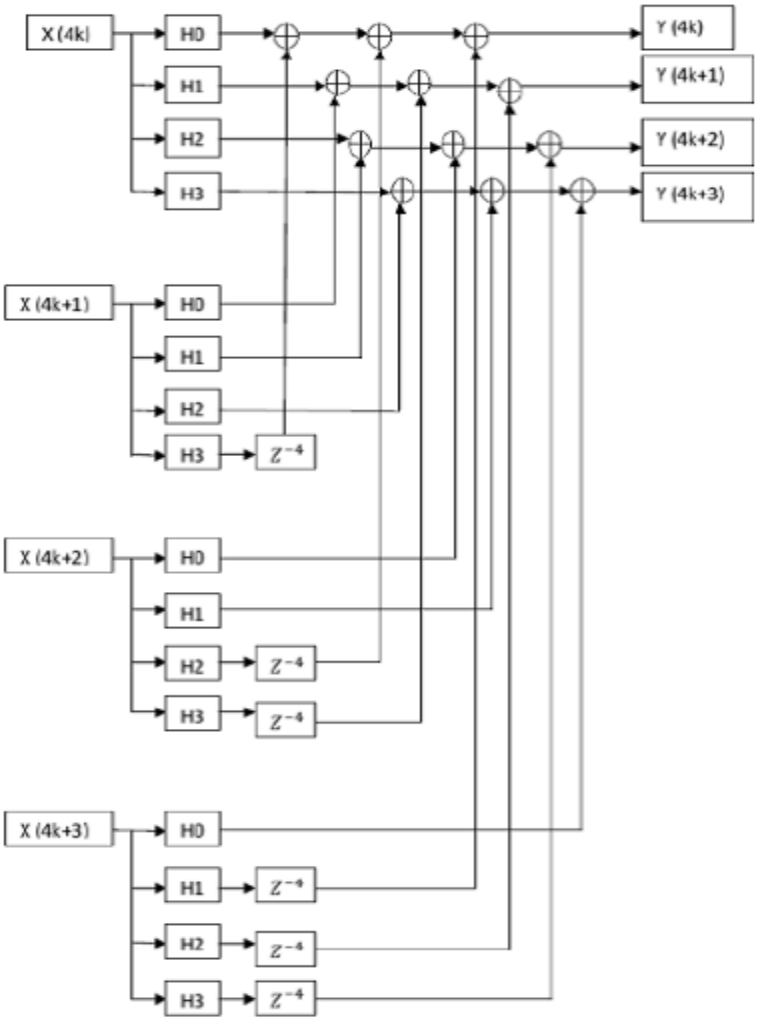

Fig.5 shows the proposed structure of the parallel FIR filter using floating point dadda multiplier.

The proposed structure of the parallel FIR filter using floating point Dadda multiplier shown in fig. 5. Since the precomputer lies on the critical path of the Floating point Dadda algorithm one pipeline stage is introduced after the precomputation block. Because of this the latency of the proposed FIR filter increases by one clock cycle. If conventional multipliers (Wallace multiplier, Booth-encoded multiplier etc.) are used for filter implementation, flip-flops for pipelining should be placed in every tap of the filter. However, pipelining of the filter using Floating point dadda algorithm can be simply done by placing flip-flops right after the precomputation block, irrespective of the filter size, due to computation sharing and reuse. Therefore, the cost of

pipelining (the number of flip-flops) is much smaller than using conventional multipliers. IEEE-754 complaint Floating-point adder is designed as explained in the previous section for the adder elements in the filter.

VI. RESULTS AND DISCUSSIONS

We have described the proposed floating-point Dadda multiplier architecture and floating-point booth multiplier architecture using Verilog Hardware Description Language. To demonstrate the application of the proposed floating-point Dadda algorithm, we have implemented the floating-point Dadda algorithm architecture into a 10-tap programmable parallel FIR filter with transposed direct form. Also, to show the effectiveness of the proposed floating-point Dadda algorithm scheme, a 10-tap FIR filter with programmable coefficients is designed based floating-point booth multiplier implemented. Both the multiplier scheme and parallel FIR filter structures using both floating-point Dadda multiplier and floating-point booth multiplier are modeled using Verilog HDL and simulated using Xilinx . After functional validation, the structures are synthesized using RTL Compiler. The results of the filter implemented based on floating-point Dadda algorithm and floating-point booth multipliers are tabulated in Table I.

TABLE I. COMPARISON RESULTS

Components		**FIR Filter-Floating Point Dadda Multiplier**	**FIR-Filter Floating Point Booth Multiplier**
Delay	Clock(ns)	44	47
Power	Switching power(mw)	41.77	69.46
	Net power(mw)	26.92	37.36
	Internal power(nw)	14.85	32.09
	Leakage power(nw)	21.12	22.69
	Total power(mw)	83.75	138.93
Power-Delay product(PDP)		3685.05	6529.84
Area	No.of Cells	46521	36703
	Cell Area(μm^2)	475442	464179

Parallel FIR filter based floating-point dadda algorithm structure achieves 39.7% savings in Power and an improvement of 6.38% in terms of speed and 43.56% savings in terms of PDP (Power Delay product) compared to the parallel FIR filter based on floating-point booth multiplier with 2.43% area overhead. This is achieved by computation sharing which reduces the computational redundancy in the filtering operation.

VII. CONCLUSION

We have implemented parallel FIR filter based on floating-point dadda algorithm and booth Multiplier. In floating-point dadda algorithm scheme, the precomputed values are shared among the multipliers in the filter. Since redundant computations are removed in the 24×24 significand multiplication, the floating-point dadda algorithm technique results in Low-power and high-performance compared to booth multiplier. The proposed parallel FIR filter based on floating-point dadda algorithm architecture can be used in the design of adaptive filter and signal equalizers.

References

[1] B. Jeevan, S. Narender, Dr C.V. Krishna Reddy and Dr K. Sivani "A High Speed Binary Floating Point Multiplier Using Dadda Algorithm," *IEEE Transaction on* VLSI 978-1-4673-5090-7/13/$31.00 ©2013 IEEE.

[2] Sivanantham .S, Jagannadha Naidu. K, Bala murugan. S & Bhuvana Phaneendra .D "Low Power Floating Point Computation Sharing Multiplier for Signal Processing Applications" International Journal of Engineering and Technology (IJET). vol. 5, no. 2,pp 975-985 Apr-May 2013.

[3] Muhammad K & Roy K, "Reduced computational redundancy implementation of DSP algorithms using computation sharing vector scaling," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol.10, no.3, pp.292-300, June 2002

[4] Jongsun Park, Muhammad K & Roy K, "High-performance FIR filter design based on sharing multiplication," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol.11, no.2, pp.244-253, April 2003.

[5] Carl Hamacher, Zvonko Vranesic & Safwat Zaky, "Computer Organization," Mc Graw Hill, Intl. Edition, 2002.

[6] R. I. Hartley, "Sub expression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II*, vol. 43, pp.677–688, Oct.1996.

[7] M. Potkonjak, M. Srivastava & A. P. Chandrakasan, "Multiple constant multiplications: Efficient and versatile framework and algorithms for exploring common subexpression elimination," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 151–165, Feb. 1996.

[8] N. Sankarayya, K. Roy & D. Bhattacharya, "Algorithms for low power high speed FIR filter realization using differential coefficients," *IEEE Trans. Circuits Syst. II*, vol. 44, pp. 488–497, June 1997.

[9] K. Muhammad & K. Roy, "A graph theoretic approach for synthesizing very low-complexity high-speed digital filters," *IEEE Trans. Computer-Aided Design*, vol. 21, pp. 204–216, Feb. 2002.

[10] Jongsun Park, Woopyo Jeong, Mahmoodi-Meimand H, Yongtao Wang, Choo H & Roy K, "Computation sharing programmable FIR filter for low-power and high-performance applications," *IEEE Journal of Solid-State Circuits*, vol.39, no.2, pp. 348- 357, Feb. 2004.

[11] Praveen Kumar M V, Sivanantham S, Balamurugan S & Mallick P.S, "Low power reconfigurable multiplier with reordering of partial products," *Proc International conference on Signal Processing, Communication, Computing and Networking Technologies (ICSCCN)*, 2011, vol., no., pp.532-536, 21-22 July 2011.

[12] Balamurugan, S., Sneha Ghosh, S. Balakumaran, R. Marimuthu, and P. S. Mallick. "Design of low power fixed-width multiplier with row bypassing." *IEICE Electronics Express* 9, no. 20 (2012): 1568-1575.

[13] Sivanantham,S., Padmavathy, M., Divyanga, S. & Anitha Lincy, P. V. 2013 "System-On-a-Chip Test Data Compression and Decompression with Reconfigurable Serial Multiplier", *International Journal of Engineering and Technology*, vol. 5, no. 2, pp. 973-978.

[14] Hunsoo Choo, K Muhammad & K Roy, "Two's Complement Computation Sharing Multiplier and Its Applications to High Performance DFE," *IEEE Trans. on Signal processing,* Vol.51, No.2, Feb'03.

[15] Mohamed Al-Ashrfy, Ashraf Salem and Wagdy Anis "An Efficient implementation of Floating Point Multiplier" *IEEE Transaction* on VLSI 978-1-4577-0069-9/11@2011 IEEE, Mentor Graphics.

[16] B. Fagin and C. Renard, "Field Programmable Gate Arrays and Floating Point Arithmetic," IEEE Transactions on VLSI, vol. 2, no. 3, pp. 365-367, 1994.

[17] N. Shirazi, A. Walters, and P. Athanas, "Quantitative Analysis of Floating Point Arithmetic on FPGA Based Custom Computing Machines," Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'95), pp.155-162, 0-8186-7086-X/9$50 4.00 *0* 1995 *IEEE*.

[18] L. Louca, T. A. Cook, and W. H. Johnson, "Implementation of IEEE Single Precision Floating Point Addition and Multiplication on FPGAs," Proceedings of 83 the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'96), pp. 107-116, 1996.

[19] Jaenicke and W. Luk, "Parameterized Floating-Point Arithmetic on FPGAs", Proc. of IEEE ICASSP, 2001, vol. 2, pp.897-900.

[20] Whytney J. Townsend, Earl E. Swartz, "A Comparison of Dadda and Wallace multiplier delays". Computer Engineering Research Center, The University of Texas.

[21] Xilinx13.4, Synthesis and Simulation Design Guide", UG626 (v13.4) January 19, 2012.