Reduced Complexity Decision Feedback Channel Equalizer using Series Expansion Division

S. Yassin, H. Tawfik
Department of Electronics and Communications
Cairo University Faculty of Engineering
Cairo, Egypt
syassin@eece.cu.edu.eg, Hazim.Tawfik@eece.cu.edu.eg

***Abstract*—Decision Feedback Equalizers (DFE) are used to eliminate the effect of Inter Symbol Interference (ISI) in band limited channels. The Recursive Least Squares (RLS) is one of the algorithms that are used to update the coefficients of the equalizer due to its fast convergence. However, its complexity is in the order of $O(N^2)$ multiplication per iteration. Fast fixed order algorithms are used to solve the RLS algorithm using explicit set of equations. Therefore, their complexity is in the order of $O(N)$. However, divisions are needed to perform these algorithms, making it difficult to choose the one with lower computational cost. In this paper a new metric is proposed to calculate a *weight score* for any algorithm. To verify the validity of the proposed metric, the algorithms are compared by a recent study of RLS based on Decent Coordinate Descent (DCD) iterations using computer simulation. In addition, a new method to optimize the weight score is proposed, by using multiplicative techniques to perform divisions required for the fast fixed algorithms. Results have been verified by implementing a DFE on Field Programmable Gate Array (FPGA).**

Keywords*-*reduced complexity; channel equalization; signal processing; division; multiplication.

I. Introduction

ISI is a common phenomenon encountered when recovering band limited channels. ISI occurs if modulation bandwidth increases beyond the channel coherence bandwidth [1]. Channel equalization is used to compensate for ISI at the receiver to decrease the bit errors. In the family of DFE equalizers the previous output decisions influence the current estimated symbol [2]. Therefore, DFE has better tracking performance than the family of linear equalizers when the channel has severe distortion and many nulls in the pass band. Both linear and non-linear equalizers are identified by a structure and an algorithm as shown in Figure 1. In the following paragraph, we will describe the structure and algorithm of DFE briefly.

The structure of DFE may be linear transversal or lattice. Both types will be treated in a similar approach to evaluate their complexity. The choice of an algorithm to update the equalizer weights is of great importance. Rate of convergence, misadjustment, computational complexity, and numerical properties are used to evaluate different algorithms [1]. The main focus of this paper is to evaluate the computational cost or complexity for DFE that is generic enough to be used for DFE with any algorihm and structure. A comprehensive survey on DFE can be found in [2].

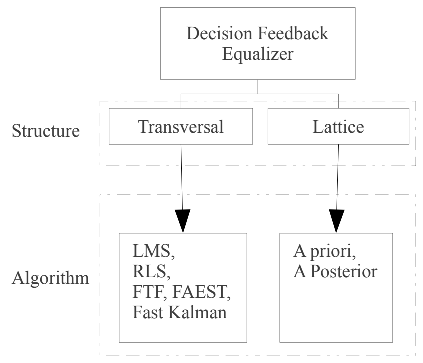
Decision Feedback Equalizer
Structure
Transversal
Lattice
Algorithm
LMS, RLS, FTF, FAEST, Fast Kalman
A priori, A Posterior

Fig. 1. DFE is classified by a structure and an algorithm

Related work such as [3], [4], and [5] are focused on the complexity reduction of DFE, because reduction in complexity leads to reduction in power consumption. This is desirable for applications with battery operated devices, where power consumption is main concern. In [3], the complexity is reduced by observing that in communication systems the input data has shift structure. In [4], spectral factorization is used to calculate the DFE coefficients in fast and efficient way. In [5], a new Dichotomous Coordinate Descent (DCD) algorithm is used to solve the RLS equations. Related work propose different solutions, but they are all have one thing in common. The common is the reduction of the total number of required mathematical operations.

The work in this paper is two fold. First, a new metric to evaluate the complexity of different DFE. This would decrease the effort to choose a suitable algorithm to be implemented. Second, DFE algorithms containing division operations consume large chip area, and it is advised to avoid them for implementation in programmable devices [6]. This work proposes a new method to implement division process. Hence, it is now permitted to use DFE containing division operations under reduced complexity constraints.

The rest of this paper is organized as follows; in Section II, candidate DFE algorithms are compared from computational cost point of view as reported in the literature. Then a new metric is proposed to evaluate the computational cost between those algorithms, and the comparison is repeated to validate the proposed metric. In Section III, it is proposed to use multiplicative techniques to perform division operations involved in DFE algorithm. Therefore, the DFE weight score will be reduced, where a smaller weight score corresponds to less complexity. In Section IV, results of comparison using the proposed metric are verified using computer simulations. In addition, to verify that the weight score of division is reduced, one algorithm is implemented into FPGA using hardware description language. Resource utilization for the number of used cells in the FPGA is also reported to show the number of used cells by a DFE. Conclusion is presented in Section V.

II. Computational Cost of DFE-RLS

A. Complexity as Number of operations

In the literature of adaptive filtering such as [2] and [4], the computational cost of DFE is reported as the number of four basic mathematical operations in addition to square root. To reduce complexity it is advised neither to use divisions nor square root, and as few multiplications as possible [5]. In this paper the use of division will be permitted by reducing its weight score as will be discussed in Section III.

Let N be the total number of forward and backward taps of DFE and consider two algorithms A and B. Algorithm A has $O(7N)$ multiplications and two divisions, while algorithm B has $O(8N)$ multiplications and one division. Note that $O(7N)$ represents that the total number of an operation is dominated by a term that is seven times the number of taps in DFE. It is not straightforward to determine which algorithm has lower computational cost. Hence, it is required to find a metric or *weighting score* for each basic arithmetic operation to be able to determine the overall computational cost. The value of the weight score will represent the overall complexity of DFE. A lower weight score corresponds to reduced complexity.

In previous work where the complexity of DFE is addressed, the weight score is reported as the number of basic mathematical operations [4], [5], and [6]. The complexity of DFE is usually reported for real data and linear filtering. These results are extended to include complex data, which is the general case for communication systems [7] as shown in Table I. In [8], it was shown that the computational cost of DFE is the same as linear filtering for the same total number of taps. Hence, the calculations are not only applied for linear filtering but also for the DFE problem.

Candidate algorithms will be chosen to represent the families of conventional Least Mean Square (LMS), conventional RLS, fast fixed order, lattice, and the family of DCD. Each candidate algorithm is generic enough to represent its family. The same procedure can be used to calculate the weight score for any other algorithm. Due to the desirable feature of fast fixed order family [3] with the complexity in the order of $O(N)$, three members of this family will be evaluated; namely Fast Transversal Filter (FTF), Fast Aposterior Error Sequential Technique (FAEST) and Fast Kalman. In the rest of this subsection, each family of algorithms will be discussed briefly.

(LMS) algorithm has the lowest computational cost [2]. Although LMS has the slowest convergence rate, it will be used as a reference scenario for the proposed metric.

TABLE I
Computational Cost of DFE using RLS Algorithm for general complex data where N is the total number of taps

Algorithm	$\times$	$+$	$\div$
Conventional	$4N^2+16N+1$	$4N^2+12N-1$	1
Apriori Lattice	$64N$	$32N$	$32N$
FAEST	$28N+6$	$28N+2$	5
FTF	$28N+10$	$28N+1$	3
Fast Kalman	$36N+2$	$32N+1$	2
ERLS-DCD-16	$12N$	$134N$	0
LMS	$8N+2$	$8N$	0

Fast fixed order algorithms are based on RLS; namely FTF, FAEST and Fast Kalman. Their complexity is in the order of $O(N)$ and their rate rate of convergence is similar to the conventional RLS, which is considered fast. Therefore, they will be suitable for systems needing short iterations to reach optimum weights of the channel to reduce transmission overhead [1].

The lattice RLS algorithms have a lot of desirable features such as improved numerical properties and modularity [6]. However, lattice filters have higher computational requirements and can not be used in all applications [5]. They will be compared here with other families, to show the effectiveness of using the proposed metric. It will be shown in Section IV how the proposed metric simplifies the comparison between different families.

A recent study to reduce the complexity of RLS algorithm is based on DCD iterations [5]. The RLS is expressed in terms of auxiliary normal equations with respect to increments of the filter weights. Auxiliary equations are solved using line search methods. These methods have more than one solution for conventional RLS problem. One of these solutions is chosen, which has the least computational cost among its family namely; Exponentially Weighted RLS with 16 iterations per sample (ERLS-DCD-16). However, the ERLS-DCD offer reduced complexity at the expense of low convergence time.

After describing the candidate algorithms briefly, it can be observed from Table I that it is not straightforward to determine which algorithm has the least computational cost. Moreover, it is impossible to arrange the rows in Table I in a descending order according to computational cost. Therefore, we found a need to develop a new metric to evaluate the weight score of DFE. To accomplish this goal, it is proposed to normalize all operations to a weight score as will be discussed in Subsection II-B.

B. Complexity Weight Score

As stated in [9] the simplest mathematical operation is the binary addition. Therefore, all operations should be mapped to a finite number of additions. Then, the overall weight score of DFE will be a linear summation of the weight score of all used operations. In order to accomplish the normalization,

TABLE II
MULTIPLICATION USING ADD AND SHIFT METHOD

Multiplicand	$\dots\dots\dots x_3\ x_2\ x_1\ x_0$
Multiplier	$\dots\dots\dots y_3\ y_2\ y_1\ y_0$
Partial Products	$\dots x_3y_0\ x_2y_0\ x_1y_0\ x_0y_0$ $\dots x_3y_1\ x_2y_1\ x_1y_1\ x_0y_1$ $\dots x_3y_2\ x_2y_2\ x_1y_2\ x_0y_2$ $\dots\dots\dots\dots\dots\dots\dots\dots\dots$
Final Product	$Z_{M_c+M_l-1}\dots\dots\dots\dots\dots\dots\dots\ Z_2\quad Z_1\quad Z_0$

we propose to analyse the low level details of each operation involved in DFE. In the following, the normalization will be developed for the multiplication operation. This normalization enables one to compare fairly between different families of algorithms.

As described in Table II the binary multiplication may be performed by adding the multiplier to the multiplicand and storing the result. Then the multiplier is shifted one bit to the left and added to the previous result [9]. To illustrate the multiplication of X and Y Let M_l and M_c be the data width for multiplier and multiplicand respectively. The first partial product is the binary multiplication of first bit in the multiplier y_0 and each bit in the multiplicand x_i where i is in the range 0 to M_{c-1}. Recalling that binary multiplication is the logical *"AND"* operation, the first bit of the first partial product equals x_0y_0, and the second bit of the first partial product equals x_1y_0, up to $x_{M_c-1}y_0$. The same procedure is repeated for each bit in the multiplier until all of the partial products are generated.

This method is the simplest from complexity point of view because it can be implemented using only one adder and one shifter. However, from processing time point of view it is considered the slowest to store the final result.

Another multiplication method is the Wallace Tree [9]. It is based on parallel generation of the required number of partial products. Afterwards this number of partial products is reduced. It is the fastest multiplier scheme at the expense of increased computational cost. A method that is considered a good compromise between processing time and computational cost is the iterative array of cells [9]. Hence, the iterative array multiplication will be used in the rest of this paper. However, the new proposed metric can be calculated for all multiplication methods as will be discussed in the following paragraph. In general, the iterative array method is used for short data lengths, which is the case for communication receiver [10], because its delay increases with operand length [9].

Multiplication consists of a finite number of cells or building blocks. To perform one multiplication, a finite number of building blocks are needed. let G be the number of building blocks needed to perform one multiplication. In order to calculate the number of required building blocks G, the following

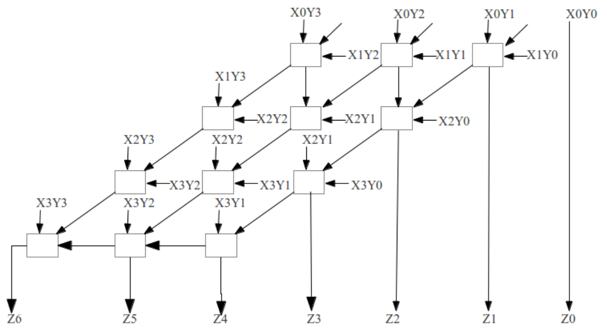

Fig. 2. Structure of 4 x 4 multiplier using full adder cells

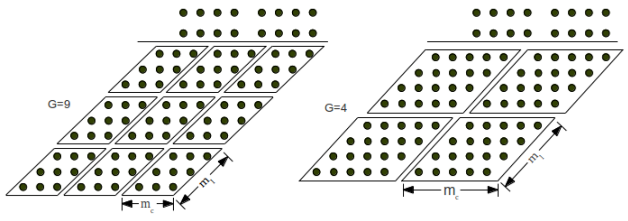

Fig. 3. Compromise between G and $m_l \times m_c$. Each black circle is used to represent one adder cell.

relation is used [9]

$$G = \lceil \frac{M_l \times M_c}{m_l \times m_c} \rceil \tag{1}$$

Where and m_l and m_c are the data width of the building block used. For example, using $M_l = M_c = 12$ and $m_l = m_c = 4$ will result in $G = 9$. Therefore, nine building blocks will be needed to construct one multiplier and each building block is $m_c \times m_l$ multiplier.

To complete the normalization, each building block is composed of finite number of 1-bit full adder cell as shown in Figure 2. Hence, one multiplier building block is composed of $[(m_l - 1) \times (m_c) + 1]$ 1-bit full adder blocks. Therefore, the weight cost of multiplication M_{score} can be rewritten as

$$M_{score} = ((m_l - 1) \times m_c + 1) \times G \tag{2}$$

Equation 2 explains how to calculate the weight score of multiplication operation in terms of 1-bit full adder cells. In Section III, the weight score of division operation will be calculated in terms of adder cells similarly.

It can be observed that there exist an inverse proportional relation between the data width of the building block and the number of building blocks G. As $m_l \times m_c$ increases, the number of required building blocks G decreases as shown in Figure 3. From complexity point of view, the weight score of multiplier is independent of the number of used building blocks G. Intuitively it does not matter whether to use one large building block or many small building blocks. In both situations the total number of 1-bit full adder cells are the same as shown in Figure 3. However, from processing time point of view it is favoured to use smaller building blocks. This is due to the fact that the processing time to have a result is proportional to the width of the building block [11].

It can be observed, from Figure 3, that using 3×3 building block generates an extra partial product because the data width is not integer multiple of the building block data width. This extra partial product will be zeros and may be skipped using a multiplexer [11]. In general, for any $M_l \times M_c$ data width, and any $m_l \times m_c$ building block width the extra partial products will not affect the final weight score. This is because the weight score is used for comparison based on the number of used 1-bit full adder cells and not on the number of building blocks.

III. Division using series expansion

Division algorithms can be divided into two categories; the first category is based on iteration of subtraction and the second is based on iteration of multiplication [9]. The first category is performed as the normal pencil and paper division. At each iteration, there is remainder R, divisor D, and quotient Q. The ith bit of quotient q_i can be calculated using the following equation

$$R(i) = R(i+1) - q_i * D * 10^i \tag{3}$$

For example, the first iteration of division of 4000 over 3 will be $4000 - 1 \times 3 \times 1000$ where $R(i+1) = 4000$, $q_3 = 1$, $D = 3$ and $10^i = 10^3$. Each iteration q_i is chosen to be 0 or 1 according to the negative or positive value of $R(i)$ respectively. This method is considered slow because the delay is proportional to the ratio between the divisor and the dividend. Therefore, we will consider another category of division algorithms, which has less processing time.

The second category of division algorithms is based on the use of Maclaurin series expansion [9]. The division $\frac{a}{b}$ will be obtained as the multiplication of a and $\frac{1}{b}$. Note that, according to the floating point standard [12], numbers are represented in either 32 or 64 bits with the format $1.x \times radix^{exponent}$ where x is a fraction. Therefore, one can use $b = 1 + x$ and ignore higher orders of x, depending on the required accuracy, in the familiar series expansion

$$\frac{1}{b} = \frac{1}{1+x} = \underbrace{(1-x)(1+x^2)(1+x^4)(1+x^8)}_{Memory location Address} \tag{4}$$

All possible values of the reciprocal are stored in a memory element with its length proportional to the data width. This would replace the need to true division with a simple memory allocation. In typical communication systems [10] the data widths are in the range 8, 10, 12 or 14 bits, because a memory block of size 2^c is needed to store all values of a digital word with data width c bits. Hence, the memory size needed is approximately 0.25, 1, 4 or 16 kbits.

To form one divider a finite number of adders and multipliers is needed to form the memory address. This number varies according to the desired data accuracy. For 8-bit data accuracy, six multipliers and four adders are needed to implement one divider and the calculation of the memory address is shown in Figure 4. To calculate the memory element address we need to calculate the powers x^2, x^4, and x^8 depending on the desired

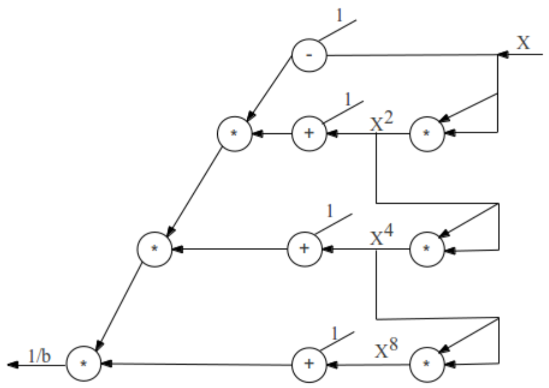

Fig. 4. Division using series expansion for 8-bit accuracy.

accuracy. For 8-bit accuracy, the weight score of one division D_{score} can be calculated according to the following relation

$$D_{score} = 6 * M_{score} + 4 \tag{5}$$

To achieve higher accuracy for the division operation, higher powers of x should be considered. For increased accuracy, such as 12-bit, x^{12} can be calculated by multiplying x^4 by x^8, which have been calculated earlier as shown in Figure 4. Consequently, one extra multiplier and two adders are needed. The weight score can be modified accordingly to be

$$D_{score} = 7 * M_{score} + 6 \tag{6}$$

According to equations 5 and 6 the division is normalized into a finite number of 1-bit full adder cells. In the following section results of weight score for different algorithms are presented.

IV. Simulation Results

In this section, results obtained by computer simulation are presented. The complexity weight score of different algorithms is plotted against filter order N. First, the weight score of each operation is calculated according to equations 2 and 5. For each algorithm the total computational cost is calculated according to Table II. Then the number of multiplications and divisions in the table is multiplied by M_{score} and D_{score} respectively. In this manner the computational cost is compared fairly. The total weight score of each algorithm is the linear summation between the weight score of additions, multiplications and divisions included in that algorithm. This procedure is repeated for different values of equalizer order N as shown in Figure 5.

The conventional RLS was not plotted due to its high weight score that would compress all other graphs at the bottom of the y-axis. The weight score of conventional RLS is almost $5.5k$ at $N = 10$. However, its complexity increases exponentially afterwards until it reaches $20k$ at $N = 20$.

It is observed that the LMS has the least weight score over all values of N. This result was expected as was mentioned in Section II-A. The ERLS-DCD-16 has slightly higher complexity than LMS. This result conforms to the results in [5],

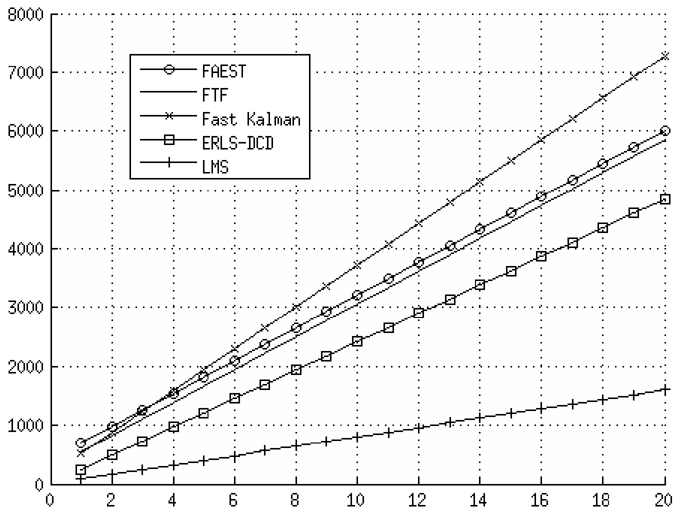

Fig. 5. Complexity of DFE versus the total number of taps

TABLE III
FPGA RESOURCES FOR THE PROPOSED ENHANCEMNT FOR FTF

Resources	$N = 8$	$N = 12$	$N = 16$
Slice	127(1%)	177(1%)	206(1%)
D-FF	136(1%)	186(1%)	226(1%)
LUT-4	206(1%)	313(1%)	403(1%)
BRAM	1(1%)	2(2%)	4(4%)
DSP48	2(2%)	3(3%)	3(3%)

which states that LMS complexity is proportional to $2N$, while ERLS-DCD-16 complexity is proportional to $3N$. Both FTF and FAEST have almost the same weight score over N. It is important to observe that fast Kalman has higher weight score than FASET. This result was not obvious before using the weight score metric because FAEST has 5 divisons while Fast Kalman has 2 divisions only.

In order to verify the usage of the proposed division method experimentally, the resource utilization for the Xilinx FPGA "Spartan3A-DSP1800" [13] is shown in Table III. Results are obtained using Xilinx development suite $ISE12.1$. Using the automatic synthesis procedure, the synthesizer fails to design the division process and ends with error messages. This result was expected because the implementation of division operation into FPGA is problematic [14]. Different division implementations such as restoring, non restoring, and Xilinx Intellectual Property(IP) consumes large area of the FPGA chip. Afterwards, divisions are implemented as proposed using memory elements, which is called Block RAM (BRAM). The synthesis process succeeded with low chip utilization . Note that the multiplication and the tapped delay line of the DFE are implemented using the DSP blocks of the FPGA [13].

V. CONCLUSION

DFE equalizers are used to eliminate ISI phenomena. Many algorithms are used to adopt the coefficients of equalizers such as conventional RLS. However, RLS complexity is high and in the order of $O(N^2)$. Other algorithms have lower complexity such as fast RLS algorithms with computational cost in the order of $O(N)$, and ERLS-DCD-16. However, it is not straightforward to compare fairly between different algorithms. In this paper a new metric, *weight score*, is proposed to measure the complexity of each algorithm. Each algorithm is normalized as a function of the number of 1-bit full adder units needed to construct the DFE. Comparison results were presented using computer simulation. In addition, a new method was proposed to decrease the weight score of the division operation. This is accomplished by using the iterative array division, which is based on Maclaurin series expansion. To verify the reduction of weight score experimentally one algorithm, (FTF), is implemented in FPGA and resource utilization results were presented. It was shown that division operation can be implemented into FPGA with low chip area utilization.

ACKNOWLEDGMENT

This work is supported by the Egyptian National Telecommunication Regulatory Authority (NTRA).

REFERENCES

[1] T. Rappaport, Wireless Communication: Priciples and Practices, 2nd ed., New Jersey: Prentice Hall, 2001, pp. 308-318
[2] J. Proakis, "Adaptive Equalization for TDMA Mobile Radio", IEEE Transactions on Vehicular Technology, vol. 40, no. 2, May. 1991, pp. 333-341
[3] J. Cioffi and T. Kailath, "Fast, Recursive Least-Squares Transversal Filters for Adaptive Filtering", IEEE Transactions on Acoustics, Speech, and Signal processing, vol. 32, Apr. 1984, pp. 304-337
[4] N. Al-Dhahir, J. Cioffi, "Fast computation for Channel-Estimate based Equalizers in Packet Data Transmission", IEEE Transactions on Signal Processing, vol. 43, no. 11, Nov. 1995, pp. 2462-2473
[5] Y. Zakharov, G. White, and J. Liu, "Low-Complexity RLS Algorithms Using Dichotomous Coordinate Descent Iterations", IEEE Transactions on Signal Processing, vol. 56, no. 7, July. 2008, pp. 3150-3161
[6] H. Sayed, Fundamentals of Adaptive Filtering, 2nd ed., New Jersey: Wiley-IEEE Press, 2003, pp. 600-620
[7] B. Bjerke, J. Proakis, K. Martin Lee, and Z. Zvonar, "A Comparison of GSM Receivers for Fading Multipath Channels with Adjacent- and Co-Channel Interference", IEEE Journal on selected areas in communications, vol. 18, Nov. 2000, pp. 2211-2219
[8] Y. Yang, X. Gao, Z. Gao, and X. Wang, "An Classification-based Adaptive Decision Feedback Equalizer for Rayleigh Multipath Channel", Journal of Computational Information Systems, vol. 8, no. 2, Jan. 2012, pp. 869-876
[9] M.J. Flynn, and S.F. Oberman, Advanced Computer Arithmetic Design", 2nd ed., New York: John Wiley and Sons, 2001, pp. 113-125
[10] O. Dabeer, and U. Madhow "Channel Estimation with Low-Precision Analog-to-Digital Conversion", IEEE International Conference on Communications (ICC 10), vol. 2, May. 2010, pp. 23-27
[11] R. Singh, P. Kumar, and B. Singh, "Performance Analysis of 32-Bit Array Multiplier with a Carry Save Adder and with a Carry Look-Ahead Adder", International Journal of Recent Trends in Engineering, vol. 2, no. 6, Nov. 2009, pp. 83-86
[12] IEEE Standard for Floating-Point Arithmetic,"IEEE Std 754-2008", pp.2-7, Aug. 2008, doi: 10.1109/IEEESTD.2008.4610935.
[13] Xilinx Inc., "XtremeDSP DSP48A for Spartan-3A DSP FPGAs User Guide", UG431 (v1.3), July. 2008, pp.32-34
[14] N. Sorokin, "Implementation of high-speed fixed-point dividers on FPGA", Journal of Computer Science and Technology, vol. 6 no. 1, May. 2006, pp. 8-11