A SOPC Architecture for Data-dependent Superimposed Training Channel Estimation

Fernando Martín del Campo, René Cumplido, Roberto Perez-Andrade
Computer Science Department
National Institute of Astrophysics, Optics and Electronics
C.P. 72840, Puebla, Mexico

A. G. Orozco-Lugo
Section of Communications
CINVESTAV-IPN
C.P. 07360, Mexico City, Mexico
aorozco@cinvestav.mx

E-mail: {fmartin,rcumplio, j_roberto_pa}@inaoep.mx

Abstract

Channel estimation in wireless communication systems is usually accomplished by inserting, along with the information, a series of known symbols, whose analysis is used to define the parameters used by filters that remove the distortion of the data. Nevertheless, a part of the available bandwidth has to be used by these symbols. Until now, no alternative solution has demonstrated to be fully satisfying for commercial use, but one technique that looks promising is superimposed training (ST). This work describes a hybrid software-hardware FPGA implementation of a recent algorithm that belongs to the ST family, known as Data-dependent Superimposed Training (DDST), which does not need extra bandwidth for its training sequences (TS) as it adds them arithmetically to the data. DDST also adds a third sequence known as data-dependent sequence, that destroys the interference caused by the data over the TS. As DDST's computational burden is to high for the commercial processors used in mobile systems, a SOPC (System on a Programmable Chip) approach is used in order to solve the problem.

Keywords

Wireless Communication Systems, Data-dependent Superimposed Training, SOPC, Hardware Coprocessors.

1 Introduction

The air is inherently noisy and its nature can contribute to the presence of different kinds of interference, as the one known as Intersymbol Interference or ISI, in which the energy of the message symbols is spread in such way that a part of each symbol overlaps with that of the neighboring symbols. The ISI can, in fact, make almost impossible to the detector inside the receiver to differentiate between a symbol and the spread energy of consecutive ones. Nevertheless, this channel can be modeled as a linear system, whose effects can be reverted in the receiver, if one knows its parameters with enough precision.

To obtain these parameters, the majority of the wireless communication systems use sequences of known symbols (training sequences) that, after a certain analysis, allow the estimation of the channel. Once this is done, the original information can be extracted, using well known techniques for data recovery.

The most extended technique to integrate the training sequences to the information is known as *time-division multiplexed channel estimation or time multiplexed training* where some of the transmission slots are used for the pilots or training symbols [1]. The performance of this approach is very high, but it has the disadvantage of needing part of the available bandwidth to accommodate the extra data.

Even though several options have been proposed, none of them has demonstrated to be more feasible than the usual training.

One of the most promising techniques that has not yet been implemented physically is *Superimposed Training (ST)*, where the training sequence is arithmeticly added to the information, saving the necessity of more bandwidth at the expense of a little power loss on the information signal [2], [3].

The method named *Data-dependent Superimposed Training* goes beyond ST adding another sequence (the data-dependent training sequence) to the information. When estimating the channel, what is analyzed is the training sequence so, from this point of view, the original data can be considered additive noise that distorts the object of study. The data-dependent sequence cancels the contribution of the input signal at the frequency bins where the training sequence has energy, improving the channel estimates over ST [4].

The problem with DDST, talking about its possible implementation, is its high computational complexity, that renders the commercial mobile and low power demanding processors useless for this purpose, at least taking into account the speeds demanded by the systems in which it could be used.

This work will describe a combined software/hardware implementation of the DDST algorithm using a SOPC approach, highlighting the most challenging issues that have arisen, and the way in which they have been tackled.

2 DDST Algorithm Review

Figure 1 shows a general block diagram of a digital communication receiver based on DDST.

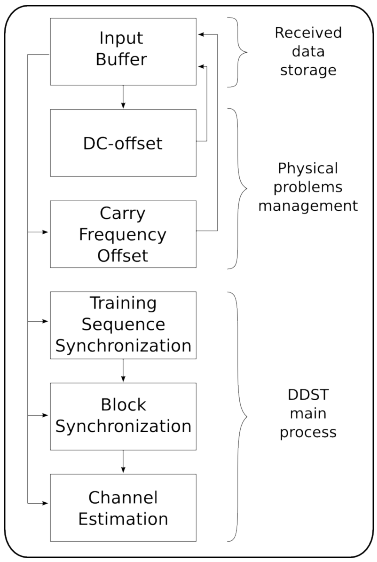

Figure 1. DDST general block diagram

Before describing each block, it is important to note that they cannot be executed in a parallel fashion, that is, to start each block processing, it is absolutely necessary that all the previous stages have already finished their own function. The lines with the arrow markers indicate from where each block receives its input and to where it feeds its output.

It is also important to mention that the DC-offset, along with the two synchronization steps and the channel estimation itself exploit the *cyclostationarity* that is induced in the transmitted signal when superimposed training is employed [2], [3].

This work does not focuses on the mathematical meaning of the formulas used to solve the different steps of the DDST approach, but in the computational burden that they present and in the procedure followed to implement them in the system.

2.1 Input Buffer

To begin with the steps of the algorithm, it is necessary to store the input data samples as they are being received, because it is not possible to correct them ”on the fly“. As shown in figure 1, the input data samples are corrected several times as the different stages of the DDST channel estimation are fulfilled.

Before going further into the process, it is fair to mention that all data samples are complex valued quantities, so all operations performed in the following steps involve the computation of both a real and an imaginary part of several numbers.

2.2 DC-offset estimation and correction

Practical systems commonly face a physical problem resulting from the building techniques used: their output, seen as voltage levels, presents an unwanted constant value that is added to the expected signal. Even though this value is almost always very small, it must be considered as the method works with first order statistics [2].

This block involves the reshaping of a vector formed by an N size subset of the original input data into a matrix of size $\|P\ x\ (N/P)\|$, whose rows are then summed to form a vector of length P. Each element of the vector is then multiplied by 1/(N/P) so, in fact, each element of the output vector corresponds to the arithmetic mean of each of the rows of the matrix mentioned above.

Once this vector has been obtained, the process reaches an iterative phase that involves matrix multiplications, norm of a vector (the square root of the sum of the squares of the real and imaginary parts of each element of the vector) several other multiplications, and a division. All of these operations have a high computational complexity, where the square root and the matrix multiplication are the more challenging. Once the DC-offset has been obtained, it is removed from the input data by simply subtracting it from each input element.

2.3 Carry Frequency Offset estimation and correction

Due to the lack of perfect oscillators and because of Doppler shifts, receivers in practical pass-band systems always experiment a carry frequency offset [6]. Among the mathematical operations found in a DDST based digital communications receiver, CFO estimation has the more complex of all, both in time and resource usage. This block requires several summations, Fast Fourier Transforms and vector norms, but even these operations result insignificant when compared with the real problem of this stage: to obtain a CFO estimate, a simulation run, for example, needed to calculate 36864 complex exponentials. If they are solved using the Euler’s Formula, 73728 trigonometric operations have to be performed.

As the CFO estimation is an iterative process, the complexity is not the only problem, because low data resolutions lead to fast growing errors, that in the end can result in a very inaccurate estimate. Once this process is complete, the CFO is removed from the input data by multiplying each input sample by one complex exponential term.

2.4 Training Sequence Synchronization Estimation

In practical applications, it is usually impossible to suppose a perfect synchronization between the transmitter and the receiver at the training sequence level, so channel estimation must consider this issue.

When there is no perfect synchronization, the estimate is just a circular shifted version of the real channel estimation that would have been obtained under ideal conditions. Using the ciclostationarity of the signal, one of the possible permutations in the circular array is equal to the estimate supposing perfect synchronization, so the problem is reduced to obtain the knowledge of the correct permutation. Mathematical operations in this stage are almost identical to those performed for the dc-offset estimation [5].

2.5 Block Synchronization Estimation

As with training sequence estimation, block synchronization is also based on the particular structure of the channel output's cyclic mean vector, and can be achieved even in the presence of a DC-offset. Due to its special characteristics, in DDST it is not enough to locate the start of a training sequence period, because it is also necessary to find the start of each received block. Only the vector encompassing a full DDST block will provide a cyclostationary mean vector independent from the data sequence, with a reduced "data" noise compared to the rest of the estimates. This procedure is achieved through a specific cost function, that will give a minimum value only with the right version of the cyclostationary mean vector [5]. Even though this block is in concept very different from the Training Sequence Synchronization estimation, the necessary operations for the Block Synchronization Estimation also include matrix multiplications, norm of a vector and other multiplications.

2.6 Channel Estimation

Once the two synchronization estimations have been obtained, the channel estimation stage can be tackled with very similar operations to those that have already been used. In fact, this easy step only needs a vector reshape, the mentioned vector obtained from the arithmetic mean of that reshaped matrix, and a matrix multiplication. At the end we obtain a vector whose complex elements correspond to the values of each tap of the estimated channel.

3 DDST and its hardware implementation

At this point, it can be inferred that the difficulty for implementing the algorithm in hardware is caused by two main reasons: the complexity of the operations (square root and trigonometric functions are far from an optimal hardware solution) and the amount of data that is used, transformed and updated constantly. The first of these issues leads to the generation of huge hardware components, that usually need several multipliers, units that are scarce in mid-range FPGAs. The second one requires a very complex control unit and the need of a high amount of memory accesses.

Moreover, the huge amount of data dependencies makes it very difficult to use techniques such as parallel processing and pipeline implementation. Even in those few cases when they are possible, its performance gain versus a software only implementation is small, and they usually require a considerable FPGA area.

This implementation tackles the problem by using a hybrid software/hardware approach. A set of C language programs running over a *NIOS II soft processor* spare the need of a complex control, while dedicated hardware coprocessors perform the most time and resource demanding operations (like the FFTs), also allowing operations with non-standard data lengths (48 bits, for example) that are hidden to the C programs of the system, making easier to control, modify and extend the software section of the SOPC.

4 Hardware Architecture

Figure 2 depicts the DDST hardware architecture. It has been designed to run in an Altera Stratix II FPGA as a NIOS II controlled system. As it can be seen, the architecture resembles that of a common computer system, with the difference that it presents several dedicated memories for fast data fetching and processing, and a set of special hardware accelerators that interact directly with the rest of the system. The processor only passes them certain parameters and activates them (through the use of their slave ports), but they execute its processing in a stand alone fashion. This means that they can read and write, using the master ports, all the memories in the system (although they almost always interact with the dedicated ones) and, while one of them is working, neither the processor nor do the other accelerators need to be interrupted. The control for no resource competition is performed by the software section of the system.

4.1 NIOS II Soft-core Embedded Processor

The NIOS II from Altera is a soft-core microprocessor that, for the DDST implementation, presents several advantages, like its low power consumption and its small required FPGA area, along with its easiness for interacting with custom hardware accelerator modules.

4.2 Dedicated On-chip Memories

The on-chip memories are structures that allow the transparent management and use of the memory blocks contained inside the FPGAs. They have the smallest latency (1) of all the available memories in the Altera boards. Low latency reduces the number of cycles needed to obtain, operate, and store a datum or a group of them. Fixed latency means that the system does not need to access the memory sequentially to achieve the highest throughput.

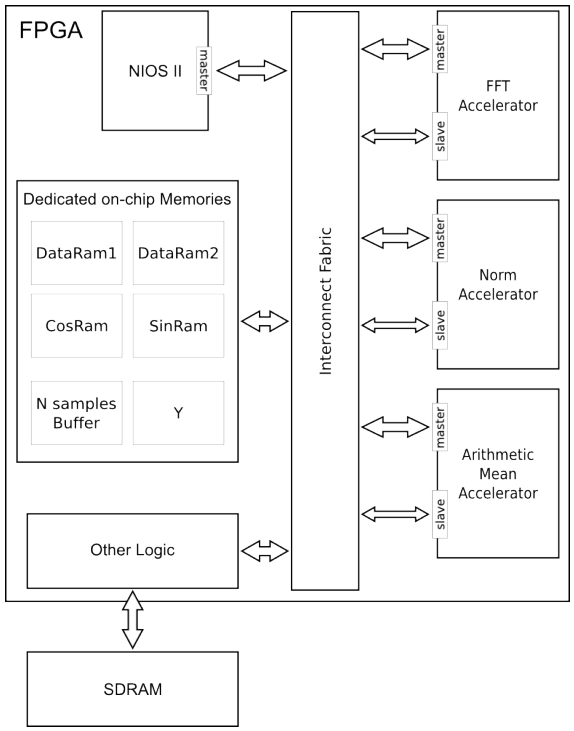

Figure 2. Hardware Architecture of the system

In the Altera NIOS II IDE, on-chip memories can be used as if they were part of the general data memory by just using a pointer to its assigned address, inside a typical C language program, that then will be compiled for the soft processor architecture.

Dedicated memories DataRAM1, DataRAM2, CosRAM, and SinRAM are used by the FFT accelerator, while the N Samples Buffer and Y are accessed by the other two coprocessors. The NIOS II can access all of them.

4.3 Interconnet Fabric

Contrary to the majority of *Systems on a Chip* and common computer systems, Altera does not use a conventional bus scheme. Their systems feature a switch interconnect fabric which bypasses bus contention in most applications and gives a higher-performance pipe between processors and peripherals. This improves the DDST implementation execution time and prevents the necessity of extra control in both the software and hardware parts of the SOPC.

4.4 FFT Accelerator

The FFT coprocessor is based on an original design of Altera that uses a tool named C2H to convert C language instructions to hardware elements directly. There are other options (to use an IP core or a custom hardware design) that can achieve a better performance than this solution, but it shows an interesting possibility of the system being built: if the DDST algorithm is modified to have better

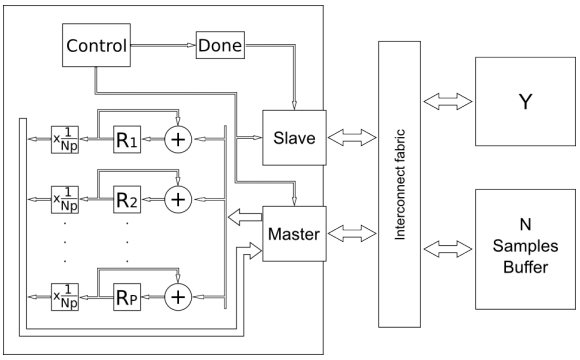

Figure 3. Arithmetic mean hardware accelerator.

performance or to reduce the computational complexity of some step, it is possible to implement such change in the architecture by just modifying some lines of code in a C program, instead of redesigning a full hardware accelerator. As it is, the FFT coprocessor executes its function with acceptable speed and area consumption.

4.5 Arithmetic Mean Accelerator

As mentioned in section 2, an operation that is performed constantly along the DDST execution is the vector reshape, followed by the arithmetic mean of the rows of the resulting matrix. This implies several memory accesses and a series of summations and divisions. At the end, the result is a vector of P elements, where P is the length of the training sequence.

This coprocessor, whose block diagram is shown in figure 3, performs the summations over the vector to reshape in parallel, sending the result to a dedicated memory that corresponds to the vector of P elements. In this way the step of the reshaping can be bypassed and the execution time is reduced considerably.

As P is a parameter that has to be known before computing, then its inverse can also be previously known and the P necessary divisions can be performed as multiplications, accelerating the process a little more.

Summarizing, this coprocessor reads directly from the *N samples buffer* dedicated on-chip memory, **P** data, each one of 32 bits , accumulating their respective values to **P** 32 bits wide registers. At the end of the process, they are multiplied by 1/(N/P), so now they contain the arithmetic means of the rows from the reshaped matrix. Finally, the results are stored in another on-chip memory.

As it will be seen in the results section, this accelerator outperforms a software only version by more than 30 times, giving also an smaller error as it works with higher resolutions during the multiplication stage.

4.6 Norm or Magnitude Accelerator

The norm of a complex number *a+bi*, is calculated as in (1) and it represents the magnitude of that number.

$$\sqrt{a^2 + b^2} \tag{1}$$

The computational burden of this operation is high, as it does not only need to multiply the real and imaginary part by themselves, but also to obtain a square root. This last operation is difficult to implement with the required speed both in software and hardware, and the algorithms used are iterative and involve the use of multiplications, substractions and comparisons.

This problem is solved by using an accelerator that has several advantages over the software-only version: it fetches both the real and the imaginary part of the FFT elements each time it performs a read operation; it works with 64 bit arithmetic, so there is no loss in the accuracy of the result. Moreover, it accumulates the calculated magnitudes as it works, so at the end of the process we will have the summation of all of them, a parameter needed for the iterative part of the CFO block. Figure 4 shows the block diagram of the accelerator. The operation of the square root block will be explained later.

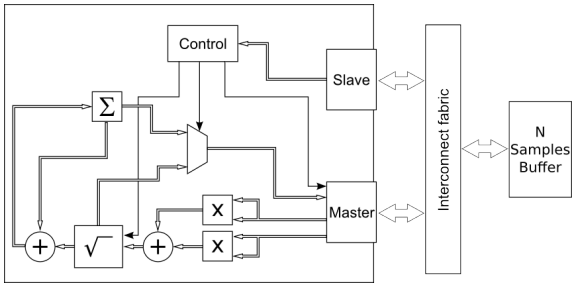

Figure 4. Magnitude hardware accelerator module

As it can be seen, it is possible to send to the N Samples Buffer the result of each of the magnitudes as they are obtained, or their total summation. These decisions are fed to the module by the software program, along with the address of the memory and the amount of complex numbers to process (1024 numbers resulting from the FFT, for example).

4.6.1 Square root hardware submodule

The square root is solved by a submodule with an operation based on the non-restoring algorithm implementation like the one of [7], but with two main differences: first, the 8 more significant bits of the root are obtained from a look-up table. This could be considered an approximated root, that then can be *fine tunned*. For example, the square root of 5 is ≈ 2.236. The look-up table would give a value of 2, so only the decimal part of the result has to be calculated. This increases drasticly the speed of the coprocessor while still using very little FPGA area. Second, each iteration calculates, in parallel, 4 bits of the root, and not only one. This system is depicted in figure 5.

The approximated root is obtained from the look-up table using as index the most significant bits of the radicand. Then this value is appended to a set of possible roots that are squared and compared to the original radicand. A comparator tree evaluates all the results and decides which of the possible roots gave the smallest error. This value is then updated as the new approximated root and the next four bits are calculated. With each iteration, the approximated root of the possible set of roots grows four bits, until it reaches the least significant bit.

Another advantage of the coprocessor is that it stops its operation as soon as it finds an exact root of an introduced number, so not all entries take the same amount of cycles to be calculated. For example, if we try to find the root of 14.0625, the process will stop as soon as it realizes that 3.75 is its exact square root (on the first iteration), even if the original number is represented as a 64 bits array, that usually requires 6 iterations for full resolution or 5 iterations for a maximum error of $\approx 7.15x10^{-7}$.

5 Results

The hybrid architecture was compared against an optimized software-only version of the DDST algorithm implementation. The system for this non-hardware implementation is very similar to the accelerated version, but it does not have the coprocessors or the dedicated on-chip memories.

Both systems were tested with a set of 3765 complex data, with 32 bits for the real and imaginary part, respectively. In fact, lower resolutions (16 bits for example) degrade the performance so drasticly that the obtained results are far to different from the expected ones (obtained from a Matlab simulation using double precision float data). Execution time was measured using the Altera module *performance counter*, that physically times a group of code lines and outputs both seconds consumed and cycles taken for the operation to finish.

Table 1 shows an abstract of the synthesis report for the software-only and hybrid systems. Space percentages refer to an estimate that the synthesizer makes according to the total available resources. The equal frequencies are expected as all the coprocessors run at higher speeds than the NIOS II, which determines the maximum performance of both designs. Because of the large amount of performed multiplications along the algorithm, it can be seen that the total of DSP blocks are used by the hardware accelerated implementation.

Implementation	Software	Hardware
Max. Frequency	238.72MHz	238.72MHz
Space (Total)	29%	68%
ALUTs	15%	60%
Registers	13%	24%
DSP Blocks	10%	100%

Table 1. Synthesis Summary

On the other hand, table 2 reports the time and cycles taken to complete some specific operations in the software program and its hybrid architecture counterpart.

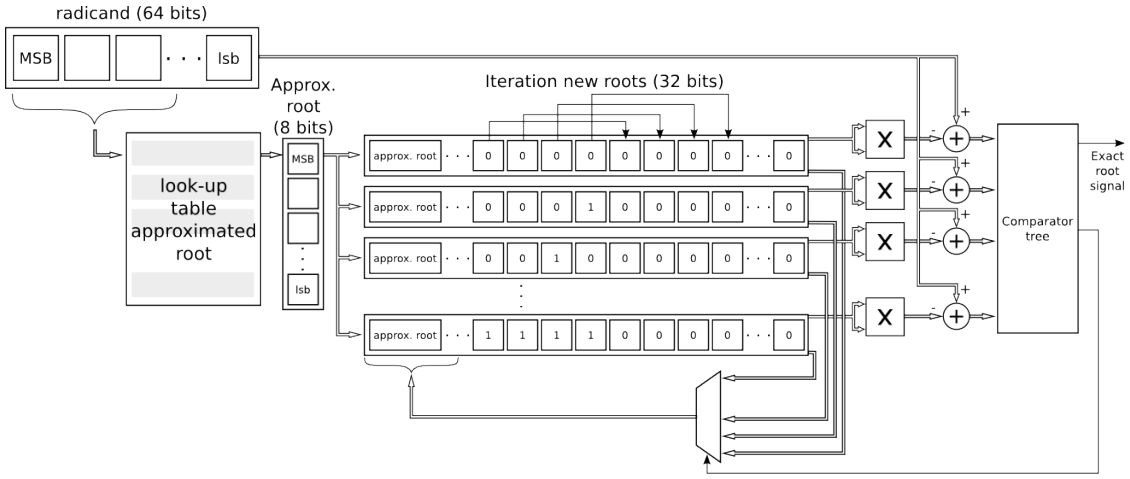

Figure 5. Square root submodule

Operation	Implementation				Performance increment
	Software only		Hardware accelerated		
	cycles	time [ms]	cycles	time [ms]	
Vector reshape and arithmetic mean	74824	0.75	2238	0.02	37.5x
1024 points FFT	1591929	15.92	56743	0.57	27.92x
Norm of all the output data from the FFT	3144061	31.44	138511	1.39	22.61x
CFO iterative process	354248859	3.54249	319750003	3.1975	1.1x

Table 2. Comparisons between software-only and hardware optimized operations

All of the arithmetic and manipulation operations of the accelerated SOPC outperform their software-only version, not only in speed, but also in precision, thanks to the non standard data lengths that are used in the intermediate results. This cannot be done with such efficiency in a common program due to the fixed data lengths that have to be used. For example, a series of 14 bits wide data have to be stored in 16 bits wide variables, and their multiplication in variables of 32 bits, unless a data resolution loss can be tolerated. On the other hand, the hardware solution can take up 14 bits registers and store the products in 28 bits structure, both if they are send to general purpose memories or to registers inside the FPGA. If we consider the fact that the input data of the system are 32 bits wide, this characteristic gets more importance, as 64 bit resolution in the software-only program is more difficult to use, and it would be a necessity since the first performed multiplication.

As it can be seen from table 2, the only operation in the system that still needs to be optimized is the CFO iterative section. This is expected as the hybrid implementation in the SOPC also uses the *sinf* and *cosf* functions from the Altera version of the *math.h* library, that calculates the sine and cosine functions of simple precision floating point inputs. To find a way to accelerate these sine and cosine calculations, used for the complex exponential operations, the use of look-up tables and a CORDIC generator [8], are being tested. These experiments consider the trade-offs between precision (so the final estimate has enough accuracy), speed (for a solution that can compete with the performance of the other coprocessors) and occupied FPGA area.

6 Conclusions

An alternative solution that uses both a hardware and a software approach was developed to allow the implementation of a digital communications receiver based on Data-dependent Superimposed Training. It was shown that it is possible to analyze a software-only code to detect the critical sections that can be translated into faster and more accurate hardware coprocessors, which can both be managed by the central microprocessor and operate independently, accessing the memories in the system without the necessity of interrupting the other components. The problem of the complex exponentials has yet to be solved. As a solution using mathematical series is not suitable for FPGAs, a cordic generator and a hybrid approach using small look-up tables is being analyzed. In addition, to improve the performance of the whole system, the use of a DMA module is also under study, in order to reduce the time consumed in memory accesses.

With this work it was possible to detect the most problematic stage of a receiver based on DDST (the Carry Frequency Offset Estimation). Nevertheless, the

use of FFTs and a *look-up table / parallel / iterative* norm accelerators make the calculation of trigonometric functions (sines and cosines) the only obstacle left in order to obtain a practical system for DDST.

The hybrid software-hardware approach demonstrated to be very versatile and flexible, allowing fast implementation of several kinds of algorithms and their fast modification, from a small change in the input parameters values to the alteration of a full stage of the process.

References

[1] Min Dong, Lang Tong, B.M. Sadler: *Optimal insertion of pilot symbols for transmissions over time-varying flat fading channels*, Signal Processing, IEEE Transactions on, Vol. 52, No. 5, May 2004, pp. 1403-1418.

[2] Aldo G. Orozco-Lugo, M. Mauricio Lara, and Des C. McLernon: *Channel Estimation Using Implicit Training, IEEE Transactions on Signal Processing*, Vol. 52, no. 1, January 2004.

[3] E. Alameda-Hernandez, D. C. McLernon, M. Ghogho, A. G., Orozco-Lugo and M. Lara: *Improved Synchronisation for Superimposed Training Based Channel Estimation*, IEEE/SP 13th Workshop on Statistical Signal Processing ,2005

[4] Mounir Ghogho, Des McLernon, Enrique Alameda-Hernandez and Ananthram Swami, Channel Estimation and Symbol Detection for Block Transmission Using Data-Dependent Superimposed Training, IEEE Signal Processing Letters, Vol. 12, No. 3, March 2005, pp. 226-229.

[5] Enrique Alameda-Hernandez, Desmond C. McLernon, Aldo G. Orozco-Lugo, Mauricio Lara and Mounir Ghogho: Synchronization and DC-Offset Estimation for Channel Estimation Using Data-Dependent Superimposed Training, EUSIPCO 2005, Turkey, September 2005.

[6] Orozco-Lugo, A.G.; Lara, M.M.; Alameda-Hernandez, E.; Moosvi, S.; McLernon, D.C.: *Frequency Offset Estimation and Compensation Using Superimposed Training*, Electrical and Electronics Engineering, 2007. ICEEE 2007. 4th International Conference on, Vol. 5, No. 7, Sept. 2007, pp. 118 - 121.

[7] Piromsopa, K, Aportewan, C. and Chongstitvatana, P.: An FPGA implementation of a fixed-point square root operation, Inter. Symposium on Communications and Information Technology, Vol. 14-16, Thailand, 2001, pp. 587-589.

[8] Ray Andraka: A survey of CORDIC algorithms for FPGA based computers, ACM/SIGDA sixth international symposium on Field programmable gate arrays, United States, 1998, pp. 191 - 200.