SYNCHRONIZER IMPLEMENTATION FOR AN IEEE 802.11a COMPLIANT ASIC

ALFONSO TROYA, KOUSHIK MAHARATNA, MILOŠ KRSTIĆ, ECKHARD GRASS and ROLF KRAEMER
Wireless Communication Systems Department, IHP
Im Technologiepark 25, D-15236 Frankfurt (Oder)
Germany

ABSTRACT

In OFDM transmissions, synchronization arises to be one of the most critical operations. The reason for that is the preservation of *orthogonality*: timing offsets and frequency offsets destroy easily this orthogonality, leading to Inter-Symbol Interference (ISI) as well as Inter-Carrier Interference (ICI).
This paper is focused on the implementation of a synchronizer for the IEEE 802.11a standard [1], which is based on the OFDM transmission scheme. Furthermore, during this year the first chips compliant with the standard are to be deployed, for that reason we also present a comparison of our solution, which is based on the one presented in [2] with two other ones proposed in [4] and [5].

KEY WORDS: OFDM, Synchronization, IEEE 802.11a, ASIC design.

1. INTRODUCTION

The standard IEEE 802.11a [1] has been devoted devoted to be used in LAN networks, where a burst transmission takes place. At the beginning of each frame a number of *preamble* symbols are transmitted in order to estimate the synchronization parameters (time and frequency offsets), which are going to be used throughout the reception of that frame (Figure 1). Such estimators are commonly referred as Data-Aided (DA) estimators, and their final implementation is obviously very dependent on the preamble structure, which normally shows some kind of periodic pattern. Nevertheless, a number of different architectures could be found if some restrictions in terms of core area or power consumption are to be attained.

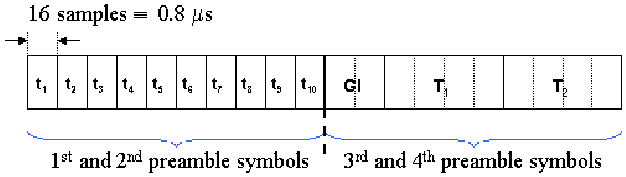

Fig.1. IEEE 802.11a preamble structure.

In the present work core area and power consumption are the main concerns and their minimization is going to be the main goal. The paper is divided into 6 sections. After the introduction is section 1, we present the basic ideas behind frequency and timing estimation in the in the case of the IEEE 802.11a standard and general solutions are derived in sections 2 and 3. Section 4 is devoted to give a detailed explanation of the proposed solution as well as its hardware implementation. Furthermore, a comparison with two other solutions is given in section 5. Finally section 6 addresses some interesting conclusions.

2. FREQUENCY OFFSET ESTIMATION

The structure of the preamble symbols has typically a periodic pattern, leading to a solution for the synchronization based on *autocorrelations* and *crosscorrelations* of the received signal. To show this, we initially consider the frequency offset problem. The main reason for this offset is a mismatch in the frequency during RF down-conversion. After sampling, the signal has the following form

$$y_{OFF}(t)\big|_{t=nT_S} = y(n)\cdot e^{j2\pi\alpha\frac{f_{Ch}}{f_S}n} \qquad (1)$$

where $y(n)$ is the original signal and T_S is the sampling time. In this expression we have also included α, which is the normalized frequency offset and is defined as $\alpha = f_{OFF} / f_{Ch}$; where f_{OFF} is the frequency offset and f_{Ch} is the frequency spacing between two adjacent subcarriers in the OFDM symbol.
The input signal $y_{OFF}(n)$ is applied to the autocorrelator, whose structure is sketched in Figure 2. Mathematically, the function $J(k)$ is expressed as

$$J(k) = \sum_{l=0}^{N_{avg}-1} y^*_{OFF}(l-k)\cdot y_{OFF}(l-k-N_d) = e^{-j2\pi\alpha\frac{f_{Ch}}{f_S}N_d}\cdot\sum_{l=0}^{N_{avg}-1} y^*(l-k)\cdot y(l-k-N_d) \qquad (2)$$

If $y(n)$ is a periodic signal with a period of N_d samples (i.e. $y(n) = y(n\text{-}N_d)$), then (2) can be simplified to finally get

$$J(k) = e^{-j2\pi\alpha\frac{f_{Ch}}{f_S}N_d} \cdot \sum_{l=0}^{N_{avg}-1} \left| y(l-k) \right|^2 \qquad (3)$$

In (3) it is straightforward to see that the phase of $J(k)$ is only due to α, and so α could be found as follows

$$\alpha = \frac{f_S}{2\pi \cdot N_d \, f_{Ch}} \tan^{-1}\left(J^*(k)\right) \qquad (4)$$

Nonetheless, there are several factors which destroy the periodicity, making $y(n) \neq y(n\text{-}N_d)$. The most important ones are the AGC settling time and the channel impulse response. The noise also contributes, but its effect can be greatly compensated by the averaging N_{avg}. In addition, when the periodicity is preserved, $|J(k)|^2$ shows a *plateau* that is also helpful during timing estimation (Figure 3).

We may want for α a symmetric range of possible values, say $[-\alpha_{max}, \alpha_{max})$. This range depends on the parameter N_d as follows

$$-\pi \leq 2\pi\alpha \frac{f_{Ch}}{f_S} N_d < \pi \qquad (5)$$

For the special case of the IEEE 802.11a standard, we find that f_S = 20 MHz and f_{Ch} = 312.5 kHz, giving for α the different ranges listed in Table 1 depending on the selected value for N_d.

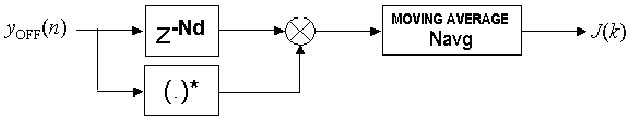

Fig.2. General autocorrelator scheme.

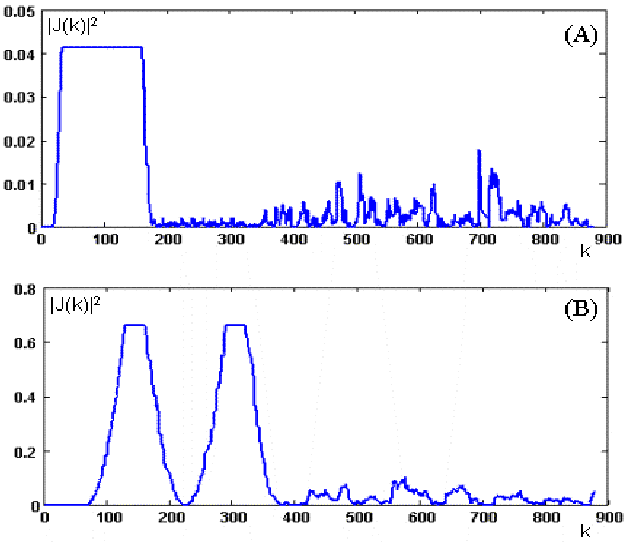

Fig.3. Autocorrelation applied to the IEEE 802.11a preambles: (A) Nd=16, Navg=16; (B) Nd=64, Navg=64.

N_d	$[-\alpha_{max}, \alpha_{max})$
16	[-2, 2)
32	[-1, 1)
64	[-0.5, 0.5)

Table 1. Range of α vs. N_d.

3. TIMING OFFSET ESTIMATION

The timing synchronization is solved by performing a crosscorrelation between the input signal and a known reference signal. This operation is going to be greatly affected by the frequency offset if the length of the crosscorrelation is excessively long, for that reason it should be decided whether the timing offset will be estimated before or after the signal has been corrected for the frequency offset.

From Figure 1, two possible reference signals could be taken for the crosscorrelation: t_1 or T_1. The results for these crosscorrelations are shown in Figure 4, where it can also be seen the dependence of the crosscorrelation with respect to α. When using t_1 as reference (Figure 4.a) ten peaks are obtained, with a distance between peaks of 16 samples. In Figure 4.c, where the reference is T_1, only two peaks separated by 64 samples are obtained, but they are much sharper. Nevertheless, these two peaks are undistinguishable in Figure 4.d, where a value for $\alpha = 0.7$ is considered.

4. IMPLEMENTATION

The solution we propose is sketched in Figure 4, and is mainly based on the one proposed in [2]. The value of the frequency offset is in the range ±1.5 (80 ppm @ 5.8 GHz) and is separated into integer and fractional parts, being α the fractional part, i.e. $\alpha \equiv \pm 0.5$. The integer part (β) is calculated in the frequency domain, once the input stream has been corrected for α, and could take one of the following discrete values: +1, 0, –1. As an example, if the overall normalized frequency offset is 0.85, then the frequency estimator should ideally estimate α to be 0.15 and β to be +1; if it were 1.23, α would be –0.23 and β again +1.

The determination of α comes out naturally considering the scheme in Figure 2 and the selected value for N_d in Figure 5. The determination of β is achieved through a *crosscorrelator* in the frequency domain. The input signal $y_{OFF}(n)$ is firstly corrected for its fractional frequency offset (α) using a Numerically Controlled Oscillator (NCO), whose actual implementation is based on the CORDIC algorithm. Then we perform the FFT with 64 samples of the signal $s_{LONG}(n)$ (Figure 6), which is a 32-sample shifted version of the original signal $T_1(n)$. This shift is compensated at the output of the FFT by a simple phase correction of $(-1)^k$.

As $T_1(n)$ is to be known, a timing estimation should be available. Any error in this estimation will be seen as a linear phase in the frequency domain. To avoid this effect during estimation of β, we do not perform directly the crosscorrelation of the FFT output signal and the reference signal, but we first *differentially decode* this

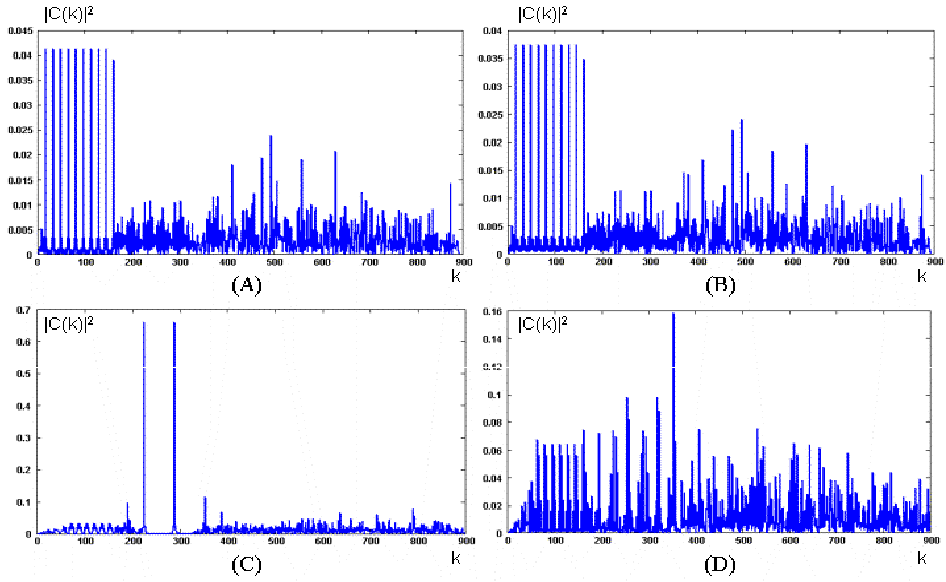

Fig.4. Crosscorrelation results: (A) with t_1 and α=0.0; (B) with t_1 and α=0.7; (C) with T_1 and α=0.0; (D) with T_1 and α=0.7.

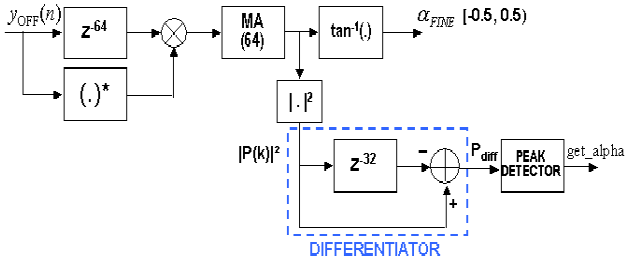

Fig.5. Proposed scheme for the synchronizer: fine frequency offset estimation and timing estimation.

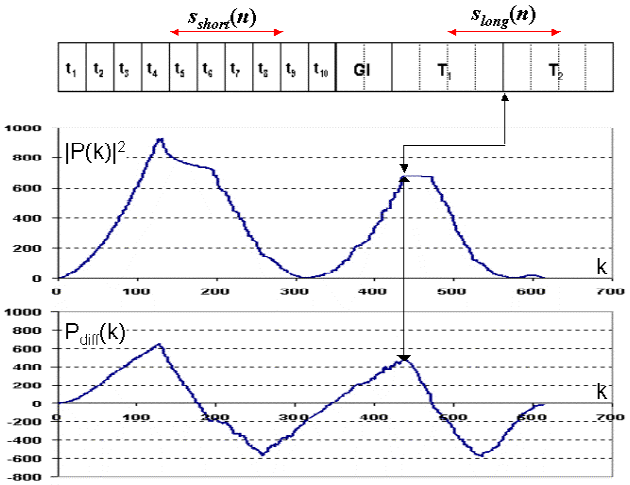

Fig.6. Timing estimation using a differenciator.

output. Because of the restricted values for β, the input data stream is corrected for its integer frequency offset in the frequency domain using a shift register.

For the **timing estimation** we make no use of crosscorrelators, but we try to determine the beginning of the second plateau in Figure 3.b. This *plateau detector* is mainly composed of two blocks: *differentiator* and *peak detector* (Figure 5). The peak detector is implemented as a relatively "smart" circuit, that is able to find absolute peaks in the differentiated data stream, and also is capable to avoid instantaneous peaks caused by the presence of noise in the system.

Under an implementation point of view, the synchronizer needs to perform several computation intensive tasks. The primary computation tasks involved are complex autocorrelation, FIR filtering, absolute magnitude computation for the incoming complex vectors, plateau detection, arctangent calculation, NCO correction, 64-point FFT and complex crosscorrelation. Thus, the overall hardware organization of the synchronizer is extremely complex. However, from the computational perspective, three of the above listed functionalities are mainly time as well as power consuming. These are the arctangent calculation for generation of fractional frequency offset, the NCO correction of the data belonging to the same frame in continuous manner using this fractional frequency offset and the 64-point FFT. The first two of them can be performed by using a single CoOrdinate Rotation DIgital Computer (CORDIC) processor by operating it in *vectoring* and *rotation* mode respectively. The design of the 64-point FFT is much more complex and has tighter time constraints. Thus, these two blocks require utmost care during the hardware design phase.

The overall hardware organization of the synchronizer is shown in Figure 7. For better controllability over the design, we partitioned the entire design in several blocks shown by the dotted boundary in Figure 7. In the next few subsections we discuss the architectural organizations of the different blocks.

Control strategy: It is clear from Figure 7 that controlling such a huge complicated structure using a

central controller is not a trivial job. Thus, in this case, we adopted distributed control system and "token flow" approach. Each of the partitioned blocks are controlled by a separate controller which also sends a signal (token) to the next controller about its status. Depending on this token signal, the next controller controls the block it is responsible for. In addition to this, the modules belonging to the same block generate another token signal for its immediately next module on completion of its work. Depending on this signal the module accepts the data from the previous one and processes it. This distributed control strategy enables us to control the complete system in much elegant and effective fashion.

Autocorrelator: The autocorrelator consists of a FIFO of length 64, a complex conjugation circuitry and a complex multiplier. The complex conjugation circuitry is nothing but simple sign reversal arrangement for the imaginary part. The autocorrelator accepts 10-bit input data and after the complex multiplication retains the most significant 16-bits for the real and imaginary part.

The FIR filter: The FIR filter (Moving Average block is Figure 5) is 64 stages long and accepts 16-bit complex inputs. For our purpose, the filter coefficients are 1 and thus, in effect it gets reduced to an accumulator that makes addition of the most recent 64 samples. We realized this circuit using only one adder and one subtractor. In every cycle the oldest sample is subtracted and the most recent sample is added to the current result. The real and imaginary parts of these data are 16-bit wide and are used for calculating the square of absolute magnitude of the particular vector as well as the arctangent.

Plateau detector: After the squaring operation, synchronization data is fed into the plateau detector. The goal of this detection is to provide the information of the symbol frame timing and to signal the start of the fractional frequency offset (α) calculation. Input stream of 32-bit squared magnitude data is shown in Figure 6 for the case of E_b/N_0=50dB and α=0.235. In order to find the beginning of the second plateau in the input data stream we perform the differentiation of the input data. With this operation, the plateau detection is substituted with much simpler peak detection, which is performed by the peak detector. The peak detector is implemented as a relatively "smart" circuit that is able to find absolute peaks in the differentiated data stream, and it is also capable to avoid instantaneous peaks caused by the present noise in the system. Structure of the plateau detector is shown in Figure 5.

CORDIC processor: The CORDIC processor accepts 16-bit inputs and generates 16-bit outputs. The processor is used in two modes *viz.* vectoring and rotation. In the vectoring mode the CORDIC takes the real and imaginary parts the FIR filter output and evaluates arctangent of that. The result of this operation is the generation of the term $-2\pi\alpha$ where α is the fractional frequency offset. The second job for the CORDIC processor is to make the correction of the incoming data (both for long preamble symbol as well as data symbols) by the phasor $\exp\{-j2\pi n\alpha/N\}$, where N = 64. The rotation mode of CORDIC elegantly performs this operation.

The CORDIC processor we use here is a novel adaptive CORDIC processor that eliminates the requirement of post processing scale factor and is adaptive in nature. Thus, choosing appropriate iteration steps adaptively it converges to the final result at faster rate. Using a technique which we term as "domain folding", the convergence range of the processor is spanned over the entire co-ordinate space. The processor is implemented in simple pipeline fashion where each section of the pipeline transfers the control data token to the next stage. Accordingly the successive stages operate either in vectoring or in rotation mode. Additionally, an input data valid and an output data valid signal are provided to indicate valid data at the input and output. Detailed discussion about the algorithm and implementation of the CORDIC processor is currently out of scope of this paper because of space restriction.

α-Accumulator: This circuit is intended for generating the term $-2\pi n\alpha/64$. The α-accumulator accepts the term $-2\pi\alpha$ generated by the *vectoring* operation of the CORDIC and divides it by 64. This operation is nothing but 6-bit right shift of the binary result coming out from the CORDIC. After that, at successive cycles it accumulates the term $-2\pi\alpha/64$ to itself in order to provide appropriate correction value to the data symbol. In practice, the accumulator transfers these results to the CORDIC processor (now in *rotation* mode) in every cycle and the *n*th data undergoes a rotation operation by $-2\pi n\alpha/64$ angle in the CORDIC processor. Since the CORDIC processor has a convergence range over the entire co-ordinate space no questions of angle overflow arise here.

64-point FFT: The 64-point FFT processor is the most time critical block in the synchronizer. The timing constraint says that the 64-point FFT should be performed in 4 μsec for every data symbol which can be met using some specialized structure for 64-point FFT. Excessive parallelism or serial construction for this module has their demerits in terms of area, power and computation time respectively. Thus, it is necessary to design the FFT processor in a specialized way to satisfy all the constraints.

We reformulated the 64-point FFT in terms of 2D 8-point FFT. The computation of 8-point FFT can be implemented using only addition, subtraction and shift-and-add principle. This enables us to implement the basic 8-point FFT in parallel manner using only combinatorial logic. In the real implementation, we used two such

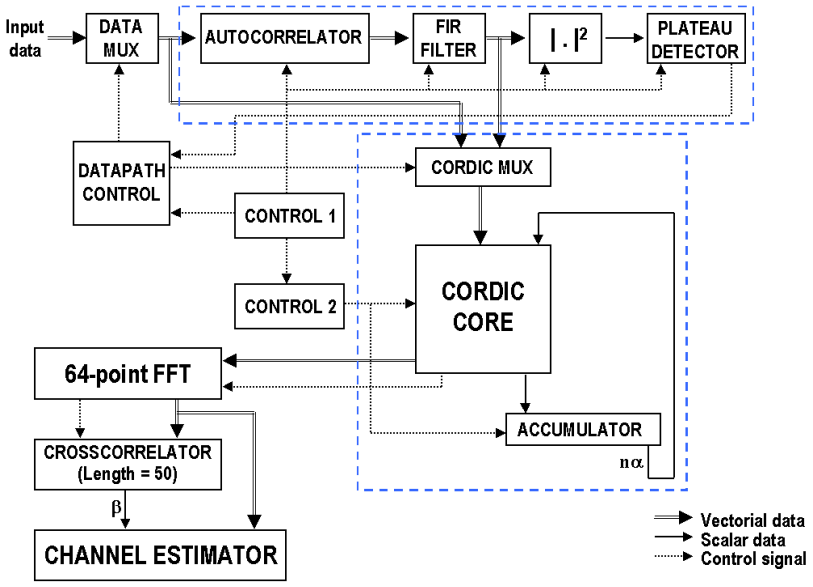

Fig.7. Main hardware components for the proposed synchronizer.

8-point FFT modules. The overall 64-point FFT processor works in the following way: the serial incoming data are first stored in a register from where they are directed in appropriate manner to the first 8-point FFT module. The result of the 8-point FFT on every data set is stored in another internal register from where they are shuffled in appropriate manner to the second 8-point FFT. The result of the second 8-point FFT is once again reshuffled in appropriate manner to generate a serial output data. A simple binary counter is provided for controlling all the operations of the different modules. In addition, an input and output data valid signals are provided for the complete processor which indicate the arrival of valid input data and output results, respectively. In our design, the input data valid signal of the FFT is tied with the output data valid signal of the CORDIC whereas, the output data valid signal of the FFT is tied to the input data valid signal of the crosscorelator. Detail discussion of the FFT architecture is beyond the scope of this paper because of page restriction. However, interested readers are referred to [3].

Crosscorrelator: The crosscorrelator implemented in the synchronizer is a simplified version of the standard complex crosscorrelator. Complex crosscorrelator is usually "weak" point in modern communication designs because of its computation complexity and need for large silicon area. Having this in mind, in this implementation we are using a simplified scheme for the crosscorrelator, with simple XNOR 1-bit multipliers, which substitute the commonly used complex multipliers. Instead of multiplication of 16-bit complex numbers, a XNOR multiplier performs only multiplication of sign bits of the input complex values. The result of a XNOR multiplication is a 2-bit complex value, so the rest of the crosscorrelator is significantly simplified. Later calculation is based on the 2-bit input values, instead of 32-bit values, which are the results of complex multiplication. The designed simple crosscorrelator has the complexity more than 3-times less than standard crosscorrelator. It is important to notice that functionality remains the same, as it will be with standard complex crosscorrelator.

In addition to the above modules, we also incorporated one data-path controller that takes care of switching the data corresponding to the data-path symbol to a separate FIFO. After the correction of α for the Long Preamble Symbol, the CORDIC applies the same correction on the data-path symbols stored in this FIFO. In this case the data-path controller module generates the relevant enable signals for the CORDIC.

The entire synchronizer is coded in VHDL. To design the entire system in VHDL, we adopted a hierarchical approach. We first coded every single block and finally instantiated them in the final design. For testing of the design we used the Modelsim™ simulator from Mentor Graphics. The VHDL simulation of the complete synchronizer is then cross-checked with the results from the original SPW™ simulation that shows correct functional behavior of the VHDL coded system.

5. BRIEF COMPARISON

There are other solutions for the synchronization in the IEEE 802.11a standard present in the literature. The solution presented in [3] (Berkeley) considers frequency offsets in the range ±100 ppm @ 5.8 GHz (580 kHz; $\alpha \equiv \pm 1.856$), while the timing estimation is obtained using the short preamble symbols (t_1 in Figure 1). The scheme also makes use of two autocorrelation operations with different delays (N_{d1}=64, N_{d2}=16). Each of them provides

an estimation for α (α_{FINE} and α_{COURSE}) which should be "combined" afterwards. As the expected value for α is bigger than the value we can estimate through α_{FINE}, the combination of α_{FINE} and α_{COURSE} is not just an addition.
The result of the crosscorrelation with t_1 was already shown in Figure 4.a. In [4] this crosscorrelation is calculated and furthermore multiplied by the output of the autocorrelator with N_d=16. This output will give the result shown in Figure 4.a, where the plateau of 160 samples is exactly centered at the positions of these first ten peaks, thus enhancing them to simplify the timing estimation.

The second solution to be mentioned is the one presented in [5] (Nokia). In this case the frequency offsets are in the range ±40 ppm @ 5.8 GHz (232 kHz; $\alpha \equiv \pm 0.7424$). The way to estimate α is fairly similar to the previous one: two different estimations for α are also calculated (named α_1 and α_2), but in this case their respective ranges are ±2 and ±1. As the actual value for α is supposed to be below these two values, its estimation is directly their average.
Nokia's architecture is especially interesting for the solution they propose for the frame detection, although it is quite hardware consuming. The timing estimation itself is done after correcting for α the input data stream, and is based on a crosscorrelator with the signal T_1 in Figure 1 as reference signal. The output of this crosscorrelator is the one already shown in Figure 4.c. A very simple peak detector is then needed.

6. CONCLUSION

Clearly indicate advantages, limitations and possible applications.

REFERENCES

Proceedings Papers:

[1] IEEE P802.11a/D7.0, *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High Speed Physical Layer in the 5 GHz Band* (Piscataway, NJ, July 1999).

[2] B. Stantchev, G. Fettweis, Burst Synchronization for OFDM-Based Cellular Systems with Separate Signaling Channels, *Proc. 48th Annual Vehicular Technology Conference (VTC'98)*, Ottawa, Canada, 1998, 758-762.

[3] K. Maharatna, E. Grass, U. Jagdhold, A Low-Power 64-point FFT/IFFT Architecture for Wireless Broadband Communication, *Proc. 7th Int'l. Conference on Mobile Multimedia Communication (MoMuC 2000)*, Tokyo, Japan, 2000, 2A-2-1 – 2A-2-4.

[4] Yun Chiu et al., OFDM Receiver Design, Final Report 12/12/2000. *Downloaded from http://bwrc.eecs.berkeley.edu*.

[5] L. Schwoerer, H. Wirz, VLSI Implementation of IEEE 802.11a Physical Layer, *Proc. 6th Int'l. OFDM Workshop (InOWo 2001)*, Hamburg, Germany, 2001, 28.1 – 28.4.