Design and Implementation of an OFDM WLAN Synchronizer

by

Joseph Pierri

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2007

Abstract

With the advent of OFDM for WLAN communications, as exemplified by IEEE 802.11a, it has become imperative to have efficient and reliable synchronization algorithms for OFDM WLAN receivers. The main challenges with synchronization deal with the delay spread and frequency offset introduced by the wireless channel. In this work, rigorous research is done into OFDM WLAN synchronization algorithms, and a thorough synchronizer implementation is presented. This synchronizer performs packet detection, frequency offset estimation, and time synchronization. Competing time synchronization algorithms are compared under a variety of channel conditions, with varying delay spreads, frequency offsets, and channel SNR. The metrics used to select between competing algorithms are statistical variance, and incremental hardware complexity. The time synchronization algorithms chosen are a basic, or dropoff detection, algorithm for coarse time synchronization, and a quantized cross-correlator with a maximum detector for fine time synchronization.

Acknowledgements

I would like to thank my supervisor, Prof. Amir K. Khandani, for his unwavering support during the course of my graduate studies, for sharing his technical expertise with me, and for helping me produce this thesis. In addition, I would like to thank Dr. Shahram Talakoub for providing thoughtful analysis and feedback on the topic of my research.

Finally, I would like to thank my parents, Edward and Mary Pierri, for their encouragement and their belief in me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This work documents the design and implementation of the synchronizer for an IEEE 802.11a [IEE99b] receiver. This receiver was built for academic research use at the University of Waterloo. The goal for this receiver implementation was to have a system that functioned with a high degree of reliability, was reconfigurable, and was efficient in terms of hardware costs.

1.1 Contributions of Thesis

While there are several works which examine IEEE 802.11a receiver synchronization algorithms in the literature, a complete and rigorous Field Programmable Gate Array (FPGA) implementation of a synchronizer has not yet been documented. This work offers a thorough investigation into many of the existing synchronization algorithms, and complete documentation of the design and implementation of a synchronizer which uses these algorithms. It is hoped that this work can be used as a reference point for any team that wants to build a similar Orthogonal Frequency Division Multiplexing (OFDM) Wireless Local Area Network (WLAN) synchronizer in the future.

This work develops comprehensive metrics for selecting among possible time synchronization algorithms, on the basis of performance and hardware optimality, and uses these metrics in selecting between competing implementation options. This work looks at three different auto-correlation algorithms for coarse time synchronization, and four different cross-correlation implementations for fine time synchronization. It examines whether or not fine time synchronization is necessary for this design, and which implementation should be chosen.

This work also introduces several important new synchronizer features, and implements other features which have hereto been described only theoretically in the literature. Overall this work examines existing algorithms and adds several important modifications, producing a novel synchronizer which has excellent performance on all counts.

1.2 Organization of Thesis

The second chapter of this work begins by examining the problems associated with synchronization in an OFDM system. It looks at the specific challenges involved with IEEE 802.11a systems, and how the structure of the packet preamble can be used for synchronization. It examines existing algorithms in the literature, and techniques for implementing these algorithms in hardware.

The third chapter examines the metrics used to compare synchronization algorithms, and the algorithms chosen for implementation. It also looks at the simulation and implementation environment, including the channel models used for simulation, the input sequences, and the software and hardware used. The fourth chapter gives a complete description of the implementation process and the results derived from simulation. This dissertation concludes with an analysis of the experiment results, and recommendations for future research.

Chapter 2

Background Information

WLANs have become increasingly pervasive in recent years. The IEEE 802.11 standard is currently one of the most widely-used standards for implementing short-range WLANs [IEE99a]. The standard was designed to provide wireless connectivity and efficient data transfer for users who are either stationary or are moving with limited velocity. The first amendment to the standard, IEEE 802.11b, used Direct-Sequence Spread Spectrum (DSSS) modulation for transmitting data [IEE99c]. Later amendments (specifically IEEE 802.11a and 802.11g) make use of OFDM, a modulation method which offers several important benefits.

2.1 OFDM Systems

In an OFDM system, the available bandwidth, B, of the system is divided into N subcarriers, each with a bandwidth of $B_N = \frac{B}{N}$ (in IEEE 802.11a, $N = 52$ and $B_N = 312.5\,\text{kHz}$). The value N is chosen to ensure that $B_N >> B_C$, where B_C is the coherence bandwidth of the channel [Uys06]. This means that the frequency-selective fading channel can be treated instead as N frequency-flat fading channels, and consequently, the system is much less susceptible to Inter-Symbol Interference (ISI) [Fla03] due to multi-path fading.

The term "Orthogonal" refers to the fact that each of the subcarriers, or tones, is orthogonal in the frequency domain. That is, at the centre frequency of one tone, all other tones have amplitude zero [Fla03]. This orthogonality ensures that there will not be Inter-Carrier Interference (ICI) between subcarriers. This is in contrast with non-orthogonal frequency division multiple access (FDMA) systems in which guard bands are inserted between carriers to ensure that no ICI occurs.

In an OFDM system, bits are assigned to the N subcarrier frequencies at the transmitter. This frequency domain data is then passed through an Inverse Fast Fourier Transform (IFFT) to yield a time domain representation, which is called an OFDM symbol. On the receiver end, the time domain OFDM symbol is received, and a Fast Fourier Transform (FFT) is performed to yield the frequency domain data.

At the receiver, the FFT is performed on M OFDM symbol samples which have been received (in IEEE 802.11a, $M = 64$). For the FFT to be calculated correctly, all M samples must be from the same OFDM symbol. However, if the start of the packet is inaccurately calculated due to poor time synchronization, samples from adjacent OFDM symbols may also be present at the input to the FFT. To remedy this, a guard interval is introduced between OFDM symbols [Fla03]. This guard interval is composed of redundant data from the end of the OFDM symbol, effectively lengthening each OFDM symbol and increasing the probability that the FFT will be calculated correctly. Note that the FFT calculation will yield an identical result regardless of which M of the symbol samples are used, provided that all M samples are consecutive.

A major benefit of OFDM systems is their low cost of implementation, due to the relatively simple components required. An OFDM system requires Digital Signal Processing (DSP) hardware for the implementation of the IFFT and FFT transformations. These processors are inexpensive and widely available, either as stand-alone components or on FPGA hardware [Fla03]. An OFDM receiver requires only only one modulator and demodulator, in contrast with other multi-carrier systems [Uys06].

2.2 Synchronization Issues with OFDM

With any data communications system, a critical component is the ability for the receiver to detect the transmission of a packet. This can be complicated by the effects of the wireless channel, so it is important to have packet detection algorithms which can account for channel effects.

The wireless channel also affects the orthogonality of the subcarriers. If there is an offset between the subcarrier frequencies at the transmitter and the subcarrier frequencies at the receiver, the tones will no longer be orthogonal, and this can cause significant degradation in system performance. To maintain this orthogonality, the transmitter and receiver must be precisely synchronized in terms of frequency. This requires accurate frequency offset calculation at the receiver.

Another important issue which must be dealt with in OFDM systems is ISI, which can occur in a multipath fading wireless channel. OFDM systems are typically resistant to significant ISI because of the presence of the guard interval at the beginning of each OFDM symbol. The guard interval length in IEEE 802.11a, for example, is 800 ns for each OFDM data symbol [IEE99b]. As the length of the channel multipath delay spread is usually 200 ns or less, this length is usually sufficient for preventing ISI in the system. Despite the presence of the guard interval, it is still possible for ISI to occur in OFDM systems. To prevent ISI from occurring, there must be accurate time synchronization at the receiver.

2.3 The IEEE 802.11a Preamble

The IEEE 802.11a standard provides mechanisms for dealing with the synchronization problems noted above. OFDM symbols are transmitted over a channel as part of a packet. The size of this packet can vary, but it is generally several orders of magnitude longer than a single OFDM symbol. The packet consists of a packet header, followed by a data payload. The packet header contains information about the packet that the receiver will require, including information about the packet duration and the transmission rate [IEE99b]. It also contains a preamble which is used for synchronization purposes.

The preamble consists of ten repeated short symbols (forming the Short Training Sequence, or STS), and two repeated long symbols (forming the Long Training Sequence, or LTS) [IEE99b]. See Figure 2.1 for a diagram of the preamble. Each of the short symbols are composed of 16 samples, and each of the long symbols are composed of 64 samples. A guard interval is inserted before the long symbols. This guard interval is composed of samples repeated from the end of the LTS.

The preamble design includes repeated symbols because this repetition makes synchronization easier for the receiver. The receiver will be able to recognize the presence of an incoming packet because the received symbols will be similar to each other, irrespective of

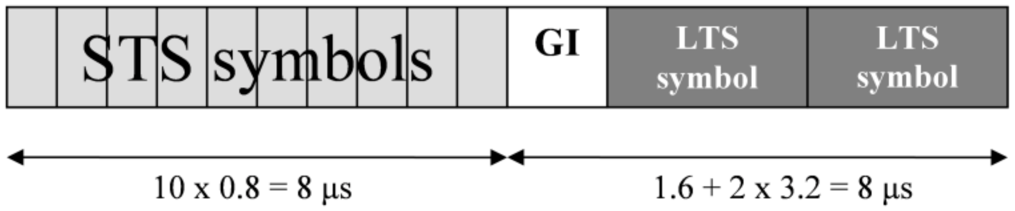


Figure 2.1: IEEE 802.11a Preamble

the effects of the channel.

The standard specifies [IEE99b] that the first seven short symbols of the STS should be used for signal detection, automatic gain control (AGC), and diversity selection (for Multiple Input Multiple Output systems). The last three symbols of the STS should be used for Coarse Frequency Offset (CFO) calculation, and Time Synchronization. The LTS is designed to be used for Channel Estimation and Fine Frequency Offset (FFO) calculation. It can also be used to refine the time synchronization estimates.

After calculating the time-domain preamble sequences, $r_{\text{SHORT}}(t)$ and $r_{\text{LONG}}(t)$ (please see Appendix A for mathematical details), the sequences are appended together, and placed at the beginning of the packet to be transmitted. The preamble is transmitted over the channel without undergoing coding or interleaving.

2.4 Synchronization Algorithms for IEEE 802.11a

Since the development of the IEEE 802.11a standard, many algorithms have been developed for synchronization which exploit the structure of the preamble. In the following pages, a few of these algorithms are outlined briefly.

2.4.1 Packet Detection

The first challenge for the receiver is to detect the packet. One possible algorithm to use is packet detection based on power level [HT01]. That is, the presence of a packet can be inferred when the signal power exceeds a specific threshold. However, packet detection cannot be done in this manner in wireless systems because of the channel noise and multipath fading, which cause the received power to vary.

Another proposal is to use signal auto-correlation [HT01], taking advantage of the repetition in the preamble, and correlating the received sequence samples with a delayed copy of the sequence, with the delay being equivalent to the length of one symbol. A moving average of this correlation can be taken over a range of one symbol. This is represented mathematically in Equation 2.1:

$$R(d) = \sum_{m=0}^{L-1} \left(r^*_{d+m} r_{d+m+L} \right) \tag{2.1}$$

Note that r_d represents the value of the dth incoming sample, r^*_d represents the conjugate of r_d, and that in the case of the STS symbols, $L = 16$ samples. Because of the variance of the incoming signal power, it is not possible to use a detection algorithm based on $R(d)$ alone [HT01]. A method for packet detection was developed in which auto-correlation is normalized by a moving sum of the received power [SC97], as shown in Equations 2.2 and 2.3:

$$P(d) = \sum_{m=0}^{L-1} |r_{d+m+L}|^2 \tag{2.2}$$

$$M(d) = \frac{|R(d)|^2}{(P(d))^2} \tag{2.3}$$

This value is then compared with a threshold, th, and a packet is said to have been detected if $M(d) > \text{th}$. The values of $M(d)$ should fall between 0 and 1. The value th should be chosen to minimize the incidence of false positive detection, and also the incidence of undetected packets, which occur when the receiver is unable to detect the training sequence.

Other possible packet detection schemes use Maximum Likelihood (ML) detection or Minimum Mean-Squared Error Criterion (MMSE) methods [HG02]. In a comparison of packet detection approaches, it was shown that there are fewer instances of false alarms using the auto-correlation method [HG02]. The ease of implementation is another important advantage for the normalized auto-correlation method seen in [SC97]

2.4.2 Frequency Offset Estimation

Methodologies for estimating the carrier frequency offset rely on the fact that a phase difference is introduced as a consequence of any frequency offset between the transmitter and receiver. That is, if two identical samples are transmitted over the channel, the phase difference between them at the receiver is proportional to the frequency offset, and also proportional to the separation between the two transmission times. Specifically, for a frequency offset of Δf the magnitude of the phase offset is:

$$\phi = 2\pi t \Delta f \tag{2.4}$$

This phase difference can be calculated by observing incoming samples separated by one symbol length. For the STS, the phase difference can be extracted from the auto-correlation value $R(d)$ in Equation 2.1. $R(d)$ can be expressed as:

$$R(d) = e^{-\mathrm{j}2\pi L\Delta f} \sum_{m=0}^{L-1} |r_{d+m}|^2 \tag{2.5}$$

The relationship between the ϕ value in Equation 2.4 and the term $R(d)$ in Equation 2.5 is given by:

$$\phi = \angle R(d) \tag{2.6}$$

In this case, $L = 16$ samples, for a total time difference of $16T_s$, where T_s is the sample period, 50 ns. Thus, if the phase difference value can be determined, an estimate of the frequency offset can be calculated as:

$$\Delta f = \frac{\angle R(d)}{2\pi \times 16T_s} \tag{2.7}$$

The value $\angle R(d)$ will fall between π and $-\pi$, and thus the range of possible frequency offset values is:

$$-625\,\mathrm{kHz} \leq \Delta f \leq 625\,\mathrm{kHz}$$

It is possible to calculate Equations 2.1 and 2.5 using the LTS rather than the STS, in which case $L = 64$. When $L = 64$, the precision improves by a factor of 4, and the range of possible offset estimates is:

$$-156.25\,\mathrm{kHz} \leq \Delta f \leq 156.25\,\mathrm{kHz}$$

Because of the improved precision, performing the calculation with a 64 sample auto-correlation is referred to as fine frequency offset estimation, while the method using a 16 sample auto-correlation is referred to as coarse frequency offset estimation. Because of the limited range in the 64 sample case, the frequency offset is best estimated in two passes, first using the STS, and then using the LTS. The IEEE 802.11a standard states that the maximum tolerance for the central frequency is ±20 parts per million (ppm), which corresponds to a maximum possible frequency offset of 200 kHz, when the carrier frequency is 5 GHz [IEE99b].

2.4.3 Time Synchronization

Several methods exist for determining the offset between the assume packet start and the actual packet start, referred to as the timing offset. The time synchronization methods proposed in [SC97] are based on the timing metric in Equation 2.3. In one method, the maximum value of $M(d)$ is found, and also the sample index, d_{max}, at which this maximum value occurs. The timing offset can be taken as the difference between the actual location of d_{max} and its expected location. In the second method, two points at which $M(d)$ is 90% of $M(d_{\text{max}})$ are found, and the midpoint between these two samples is used to estimate the offset. The problem with these approaches is the lack of precision in the first case, and the difficulty of hardware implementation in the second case.

In [LL04], and in other approaches, the time synchronization calculation is done by finding the index, d_{dropoff}, at which $M(d)$ falls below half of its peak value. This dropoff point will occur after the final STS symbol, during the guard interval which precedes the LTS symbols.

Another approach was introduced by [KFST04]. This method relies on calculating $R(d)$, and then calculating another auto-correlation sequence, this time with a sample separation of $2L$:

$$R_2(d) = \sum_{m=0}^{L-1} \left(r^*_{d+m} r_{d+m+2L} \right) \tag{2.8}$$

The difference between these two sequences is then calculated:

$$R_{\text{diff}}(d) = R(d) - R_2(d) \tag{2.9}$$

This difference sequence typically has a triangular peak during the LTS guard interval, and the index, d_{diffmax} of this peak can be used to calculate the timing offset. This algorithm promises improved performance, and has relatively low hardware complexity [KSF04].

A third scheme for auto-correlation was introduced by [FE03]. This method calculates the sum of the incoming sequence delayed by L, and the same sequence delayed by $2L$, and this sum is correlated with the undelayed sequence.

$$R_3(d) = \sum_{m=0}^{L-1} \left(r^*_{d+m} \times \left(r_{d+m+L} + r_{d+m+2L} \right) \right) \tag{2.10}$$

Once again, a detector can be designed to determine the index, d_{sumdrop}, at which $R_3(d)$ drops off to half of its peak value.

Instead of correlating the incoming sequence with delayed signal samples, it is possible to correlate the incoming sequence with the original preamble sample values. This approach is referred to as cross-correlation, and it uses the following calculation:

$$\Lambda(d) = \sum_{m=0}^{L-1} c_m^* r_{d+m} \tag{2.11}$$

The c_m^* terms are the complex conjugates of the preamble sample values, L is symbol length, and r_d is the received sequence. In the case where the LTS is used for cross-correlation, $L = 64$, and the c_m^* terms are taken from the original LTS.

Auto-correlation and cross-correlation methods are contrasted in [FWD+03]. The cross-correlation algorithm uses the LTS, and several detectors which can be used for determining the timing point are compared. The first of these detectors simply finds the maximum value of $\Lambda(d)$:

$$d_{\text{xcmax}} = \arg\max_{d} \left(|\Lambda(d)| \right) \tag{2.12}$$

The second detector adds the absolute values of N successive cross-correlation results, and attempts to maximize the sum:

$$d_{\text{xcsum}} = \arg\max_{d} \left(\sum_{p=0}^{N-1} |\Lambda(d+p)| \right) \tag{2.13}$$

Finally, a third detector looks to find the first instance at which $\Lambda(d)$ exceeds a chosen threshold, th, where th is a percentage of the observed maximum value. That is:

$$d_{\text{xcp}} = \arg\min_{d} \mid |\Lambda(d)| \geq \text{th} \times |\Lambda(d_{\text{xcmax}})| \tag{2.14}$$

Comparing an auto-correlation algorithm and a cross-correlation algorithm, it has been shown that the cross-correlation algorithm outperforms, and that the first and third detectors have much better performance than the second detector [FWD+03].

The STS can also be used in cross-correlation algorithms. If the original STS symbols are correlated with the incoming symbol sequence, there should be peaks in $\Lambda(d)$ every 16 samples. The timing offset can be calculated from the point at which these cross-correlation peaks stop occurring [YCK06].

A few algorithms exist which combine the auto-correlation and cross-correlation approaches. In [NG03], $\Lambda(d)$ is used to identify the point at which $R(d)$ drops off, and this dropoff point is used to calculate the timing offset value. A similar approach is used in [KP07]. In this case the coarse timing point is determined using auto-correlation, and the fine timing point is determined using cross-correlation.

Another type of algorithm which combines both auto-correlation and cross-correlation methods was proposed [ZS04]. In this case, the auto-correlator output is compared to a threshold, as in other methods, and a moving sum of the cross-correlator output is taken, and compared with another threshold. The timing offset is calculated at the moment at which the auto-correlator output is less than the first threshold, and the moving sum is greater than the second threshold.

Rather than limiting the calculations to the STS and LTS, another possibility is to include the guard interval in the calculations, taking advantage of its known structure. In [YCK06], an algorithm is proposed in which the last STS symbol, t_{10}, is united with the guard interval to form a 48 sample symbol. Because of the longer symbol length, this symbol can offer a more precise timing offset estimate, while reserving the LTS for later use. The downside of this approach is that the correlation result has prominent sidelobes which interfere with the estimation process. These sidelobes result from the correlation of the STS with the incoming sequence. To account for these sidelobes, the correlation of the

STS and the incoming sequence is calculated, and subtracted from the 48 sample symbol correlation.

When doing synchronization, an essential consideration is AGC, which, as per the IEEE 802.11a standard, is done using the STS. In [FE03] an interface is proposed which allows the detection system to be disabled while the gain value is being calculated, and is then re-enabled once it has been calculated.

2.5 Implementation Algorithms and Techniques

One of the important algorithms used for calculating Equations 2.1 and 2.2 is the following from [SC97]:

$$R(d+1) = R(d) + (r^*_{d+L} r_{d+2L}) - (r^*_d r_{d+L}) \tag{2.15}$$

This iterative algorithm allows for an efficient hardware implementation of L-sample averaging. Another algorithm was introduced in [LL04] which has a further hardware saving modification for this correlation calculation. In this algorithm:

$$R(d+1) = R(d) + Re\{(r^*_{d+L} r_{d+2L}) - (r^*_d r_{d+L})\} \tag{2.16}$$

By considering only the real values, this algorithm reduces the hardware complexity, and lowers the noise level of the correlation sequence.

Another contribution is the observation that the division operation required in Equation 2.3 can be avoided by choosing a metric threshold level which is a power of 2 [CVA+04a]. For instance, choosing a threshold value of 0.5 will allow the calculation to implemented using a bit shift operation rather than a division operation. The advantage is hardware savings, and in [CVA$^+$04a] it was shown that the value of 0.5 is actually a very good threshold choice. In [YNW02], the choice of 0.5 is shown to be a good choice for Rician fading channels as well, for a variety of different delay values.

For calculating the frequency offsets a CORDIC (COordinate Rotation DIgital Computer) processor can be used to calculate the value of the angle in Equation 2.6. The CORDIC takes two input values, the real and imaginary components of the calculated auto-correlation value from Equation 2.1, and returns the angle corresponding to these values. In [CVA$^+$04b], a CORDIC processor is incorporated into a receiver hardware design, and it is reused for both the coarse and fine frequency offset estimate. More details about the CORDIC algorithm and implementation are given in later chapters.

One of the limitations of the cross-correlation schemes for time synchronization is the hardware complexity required. If bit-shifting operations could replace multiplication operations, the hardware savings would be significant. In [HLK03], the c_m^* terms are quantized to powers of 2, allowing bit-shifting to be used in place of multiplication.

In the implementation in [HLK03], the hardware savings are almost 90%, while the performance does not suffer greatly. In fact, in the case where the quantization range is between -4 and +4, the performance is very comparable to the case where no quantization is used. However, using a quantization range smaller than [-4, +4] would result in a degradation in performance [HLK03].

In [MKB06] the structure of the LTS is expressed as a function of conjugates, as seen in Equation 2.17. Using this conjugate property, it is possible to reduce the computations required for calculating the cross-correlation sum by half, resulting in hardware savings.

$$r_{\text{LONG}}(t) = [y_A(k) y_B(k) y_A(k) y_B(k)] \tag{2.17}$$

where:

$$y_B(k) = y_A^*(32 - k) \tag{2.18}$$

An overall system architecture is demonstrated in [CVA+04b], showing how the detection, synchronization and compensation blocks work together at the system level. Another overall system architecture is given in [MS04]. In this paper, the block diagram for a Xilinx FPGA implementation is given, including the circuitry used for the synchronization block.

Chapter 3

Implementation Considerations and Simulation Environment

3.1 Comparison Metrics

One of the questions which this work attempts to answer is: how should a designer choose between competing synchronization algorithms? In this section, metrics for comparing various algorithms are described.

In quantifying the accuracy of an algorithm, a useful metric is the variance of the estimates produced by the synchronizer algorithm. The simulation iterations will produce a range of slightly different channel conditions, and this will result in different estimates for the time and frequency offset. The variance within a series of estimates is indicative of how well an algorithm can withstand the random noise introduced by the channel. The variance should be non-zero, because the timing offset will not be identical in all cases. A smaller variance is very desirable.

With this investigation, the synchronizer was implemented on an FPGA. The efficiency of a synchronizer design in terms of hardware can be estimated by the quantity of resources that it takes up on the FPGA. Specifically, with the Altera Stratix II, the resources of interest are Adaptive Look-up Tables (ALUTs), registers, and DSP elements. The resource counts are taken after hardware synthesis, but before placement and routing, because placement and routing algorithms often sacrifice area to improve circuit speed.

Thus the two metrics which are used in this work to evaluate and compare synchronization algorithms are the variance of simulation iterations, and the total hardware footprint of the synchronizer design, measured in FPGA resource counts.

3.2 Algorithm Evaluation

Packet detection synchronization can be done using the Equations 2.1, 2.2 and 2.3. No ML methods are considered in this work, because, as detailed in Section 2.4, the auto-correlation algorithms offer superior performance and lower variance. The hardware required to do packet detection consists of a block for calculating auto-correlation, one for calculating power, a bit-shifter, and a comparator.

The coarse frequency offset estimate can be calculated using Equations 2.6 and 2.7. This requires the addition of a CORDIC block, as well some additional hardware blocks for calculating the frequency value from the angle. Fine frequency offset calculations can be done with the same hardware, although it requires another auto-correlator for performing the 64 sample auto-correlation calculation. The question for the time synchronization implementation is: what additional hardware should be added to the design, and how much incremental performance benefit is derived from this addition of hardware.

As mentioned in Section 2.4, the proposals from [SC97] for time synchronization are unattractive candidates, and are not considered for implementation. The "dropoff detection" method introduced in [LL04] is straight-forward to implement in hardware. This algorithm takes the 16 sample auto-correlator output and determines the point at which it drops below a threshold value, outputting the index value.

The other two auto-correlation algorithms considered are the "difference" method presented in [KFST04], and the "sum" method presented in [FE03]. Implementing the former algorithm requires an additional auto-correlator and a peak detector, and implementing the latter algorithm requires an additional adder, delay elements, and the same dropoff detector mentioned earlier. Implementing a cross-correlation algorithm will require more hardware, but with the promise of much better precision for time synchronization

One important decision is how to determine the timing offset point using a cross-correlator. The two detectors under consideration are the ones presented in [FWD$^+$03]. Specifically, the max detector, as seen in Equation 2.12, and the minimum threshold detector given by Equation 2.14. A final decision when implementing a cross-correlation algorithm is whether to use fully precise cross-correlation coefficients, or whether to round those coefficients to the nearest powers of 2, as seen in [HLK03].

Overall, three auto-correlation algorithms are considered for coarse time synchronization, and four different cross-correlation implementations are considered for fine time synchronization. Based on the metrics introduced in Section 3.1, these algorithms are compared and evaluated, and then a final synchronizer design is decided upon.

3.3 Software and Hardware

Once the algorithms to be implemented are known, the next step is setting up an appropriate simulation environment for evaluating these algorithms. The implementation and simulation of all algorithms is done using the Simulink environment. The simulation environment provides pre-built channel models, mechanisms for specifying input sequences and storing output results. The Altera DSP Builder software provides hardware building blocks which can be used to construct the synchronizer circuit. This circuit can be simulated in Simulink and MATLAB, compiled into netlists, and finally implemented directly onto an FPGA.

The chosen FPGA is the Altera Stratix II EP2S180. This FPGA is large enough to accommodate the entire receiver design, including the synchronizer, and offers a large quantity of DSP resources. The chosen synthesis tool is Altera's Quartus Native Synthesis, used within the DSP Builder environment.

To generate simulation results, a series of MATLAB scripts were authored to automate the process of executing simulations. The input for the Simulink circuit was taken from input files, and is detailed in Section 3.5. The wireless channel effects are simulated using a channel model, detailed in Section 3.4. Output from the circuit is captured in the MATLAB environment, and saved to files. It is also fed to MATLAB scripts for data analysis and graphical display.

3.4 Channel Model

The channel model used in this investigation follows the tapped-delay model. It incorporates the effects of multi-path fading and Additive White Gaussian Noise (AWGN). The block diagram for the channel model used is given in Figure 3.1.

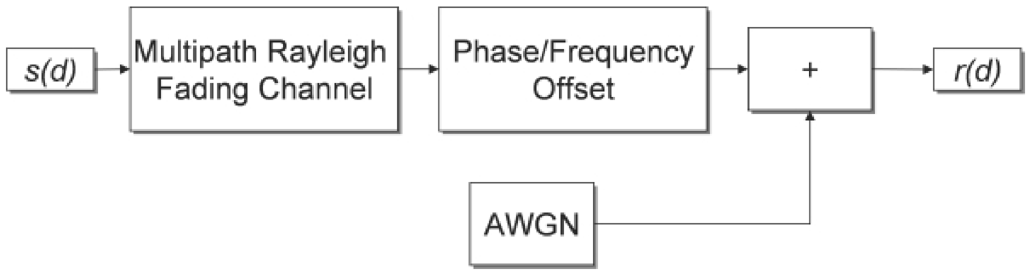


Figure 3.1: Channel Block Implementation

In building the tapped-delay model for the channel, this investigation relied upon industry standard channel models. In particular the European Telecommunications Standards Institute (ETSI) channel models [Ber01].

The ETSI channel model parameters for two types of channels are given in Tables 3.1 and 3.2. The first type of channel simulates the indoor environment of an office, and has a delay spread of 50 ns. The second type of channel simulates a large open space environment, and has a delay spread of 150 ns.

Tap number	Delay (ns)	Relative Power (dB)
1	0	0.0
2	10	-0.9
3	20	-1.7
4	30	-2.6
5	40	-3.5
6	50	-4.3
7	60	-5.2
8	70	-6.1
9	80	-6.9
10	90	-7.8
11	110	-4.7
12	140	-7.3
13	170	-9.9
14	200	-12.5
15	240	-13.7
16	290	-18.0
17	340	-22.4
18	390	-26.7

Table 3.1: Delay Profile in ETSI A Channel Model

Tap number	Delay (ns)	Relative Power (dB)
1	0	-3.3
2	10	-3.6
3	20	-3.9
4	30	-4.2
5	50	0.0
6	80	-0.9
7	110	-1.7
8	140	-2.6
9	180	-1.5
10	230	-3.0
11	280	-4.4
12	330	-5.9
13	400	-5.3
14	490	-7.9
15	600	-9.4
16	730	-13.2
17	880	-16.3
18	1050	-21.2

Table 3.2: Delay Profile in ETSI C Channel Model

In addition to the two delay models used, various frequency offsets were used in simulations. These were chosen to simulate channel conditions in the best, moderate and absolute worst cases. They are given in Table 3.3.

Frequency offset values (kHz)
0
100
200

Table 3.3: Frequency Offset Values Used in Simulation

3.5 Input Sequences

To test the ability of the circuit to detect the presence of a packet sent over the channel, it was necessary to generate two types of input sequences. A sequence in which a packet preamble is present, and sequences in which there is no preamble present. The specific preamble sequence sent over the channel is given in Table G.24 in [IEE99b].

It is necessary to send a large quantity of packets over the channel, while changing the random channel parameters in each case. In this way, the performance of the synchronizer can be averaged over a series of trials. In this work, 200 simulations are run for each test case, and in each simulation the "seed" parameters for the channel model are selected randomly, between 0 and 100. A selection of seeds within this range ensures an adequate amount of channel condition variation. There are two seed parameters (one for the AWGN component, and one for the Multipath Fading component), and the generation of random seeds is done through MATLAB.

Chapter 4

Implementation of the Synchronizer

4.1 Packet Detection

Implementing the Packet Detection hardware involved building the auto-correlator, the power calculator, and the comparator required for implementing Equations 2.1, 2.2 and 2.3, deciding on a threshold value, and ensuring correct functionality during simulations.

In accordance with the analysis given in Section 2.4, it was decided that the threshold should be set at $0.5 \times P(d)$. A choice of 0.5 allows for the use of a bit shift operation instead of a division operation.

The input to the packet detection circuit is the output from the AGC circuitry. However, to isolate and study the synchronization algorithms of interest in this work, the AGC is ignored, and the input comes directly from the sampled channel output. The output of the packet detection circuit includes a control signal indicating when a packet has been detected, as well as the auto-correlation and power values, which are also used elsewhere in the synchronizer. The packet detection output control signal should only become non-zero after the AGC has completed.

To account for the problem of momentary spikes in the auto-correlation value, an averaging circuit has also been introduced. The averaging circuit will output a non-zero value only if the packet detection signal has been non-zero for a certain number M of the past N clock periods. Values of $M = 8$ and $N = 32$ were used for the simulations. The block diagram for the packet detector is given in Figure 4.1.

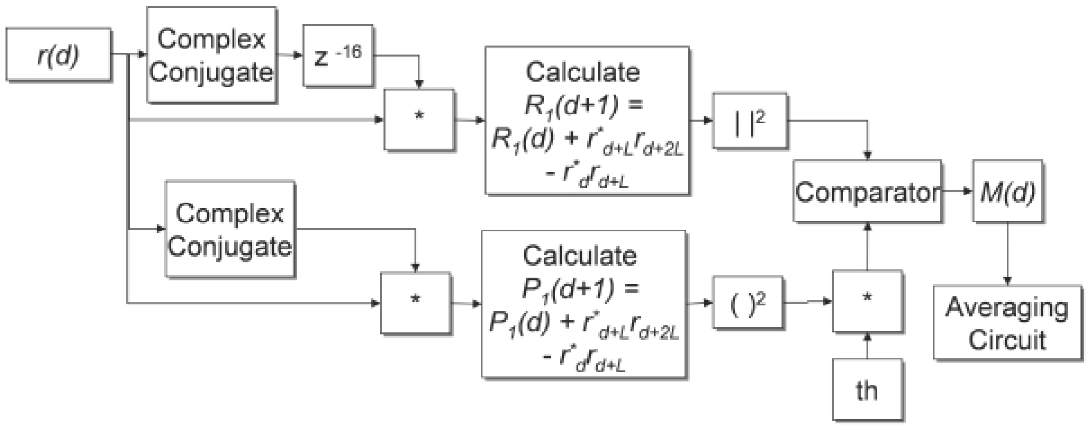

r(d)
Complex Conjugate
z -16
*
Calculate R1(d+1) = R1(d) + r*d+L rd+2L - r*d rd+L
| |2
Comparator
M(d)
Complex Conjugate
*
Calculate P1(d+1) = P1(d) + r*d+L rd+2L - r*d rd+L
( )2
*
th
Averaging Circuit

Figure 4.1: Packet Detector Block Diagram

Figure 4.2 shows the behaviour of the packet detection circuit when the input sequence contains a preamble. In this figure, the top graph shows the 16 sample auto-correlation, the second graph shows the power output (multiplied by 0.5), the third graph shows the comparator output, and the fourth graph shows the averaged comparator output. This graph shows the effect that the averager has on the circuit, smoothing the comparator output, but delaying the signal by several samples.

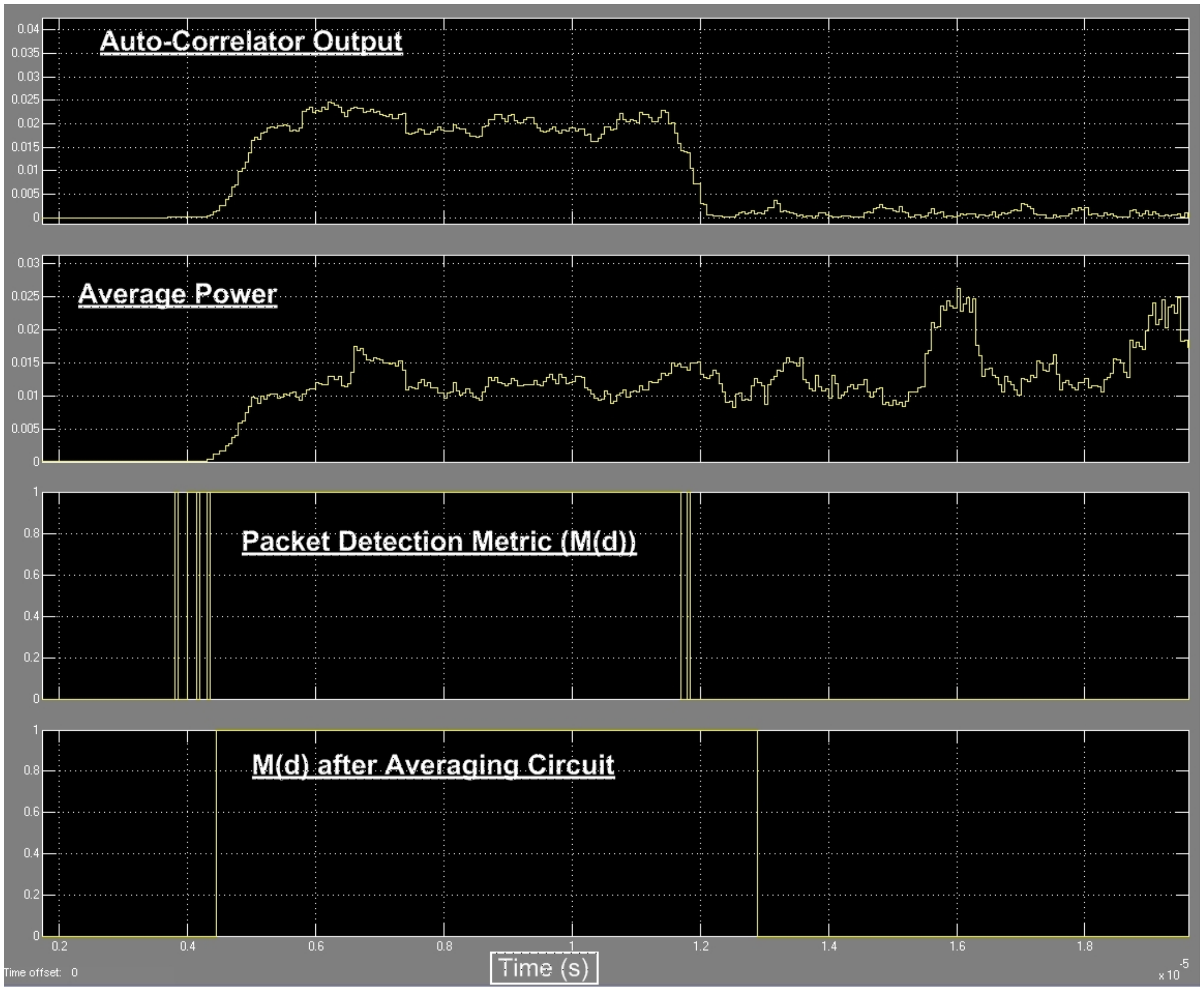


Figure 4.2: Typical Packet Detector Output

4.2 Frequency Offset Calculation

The frequency offset is estimated in two steps, first using the 16 sample auto-correlator output, and then subsequently using 64 sample auto-correlator output. The hardware requirements for this part of the synchronizer overlap with the requirements for the Packet Detector. In particular, the auto-correlator hardware can be reused.

The frequency offset estimation block should take as its inputs the auto-correlation of the incoming samples and control signals for the circuit. The outputs of the circuit should include control signals, and a frequency offset estimate. The inputs and outputs to the frequency offset estimation block are shown in Figure 4.3. The output of the estimation block can be sampled at any time during the STS, but it must be after the AGC has completed.

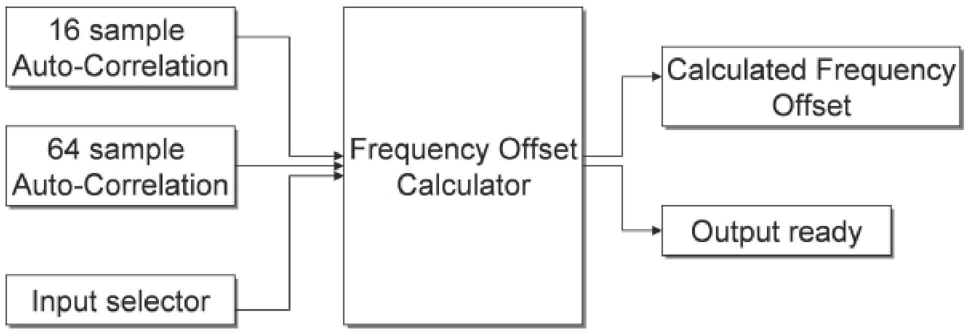

16 sample Auto-Correlation
64 sample Auto-Correlation
Input selector
Frequency Offset Calculator
Calculated Frequency Offset
Output ready

Figure 4.3: Frequency Offset Estimation Block

Inside the block, the CORDIC is the engine used for calculating the angle of the sample auto-correlation, as per Equation 2.6. To calculate this angle, the CORDIC executes the algorithm given in Figure 4.4.

```
for count <- 1 to N
  diff_x <- x / power (2, count - 1)
  diff_y <- y / power (2, count - 1)
  diff_angle <- atan (count)
  if y < 0 then
    x <- x - diff_y
    y <- y + diff_x
    angle <- angle - diff_angle
  else
    x <- x + diff_y
    y <- y - diff_x
    angle <- angle + diff_angle
  end if
end for
```

Figure 4.4: CORDIC Algorithm

This algorithm can be visualized in the following way. The input is the vector (x, y) and the algorithm seeks to rotate this vector to the x-axis. At the end of the algorithm's execution, the angle is calculated and stored in *angle*, and the hypotenuse value is stored in the variable x. The absolute value of the y variable will continue to decrease with each iteration. The number of iterations N is up to the implementer, and the algorithm will improve its precision with more iterations. The atan(*count*) constants are calculated prior to the execution of the algorithm, and are stored in memory.

A value of 20 was chosen for N, and the (x, y) vector is composed of the real and imaginary parts of the auto-correlation value, with each being 20 bits in width. The CORDIC block takes integer inputs, so the incoming real and imaginary values are scaled by a predetermined scaling factor.

The value of $N = 20$ requires 20 clock cycles for the output value to be calculated. During the course of this 20 cycle period, the estimation output value will vary. To ensure that the estimation value can be sampled at the output, a small circuit is required to hold the "settled" value. In addition to storing the output, this circuit performs the calculation given in Equation 2.7, and also divides by the scaling factor.

Another functionality required for the frequency offset estimation block is the ability to compensate for angles which are outside of the first quadrant (i.e., $x > 0, y > 0$), as the CORDIC will only calculate angles between 0 degrees and 90 degrees. This hardware takes the sign bit from the incoming real and imaginary parts of the input, and uses these to choose the quadrant of the angle.

The hardware implementation of the frequency estimation block involved several steps. The CORDIC required by the circuit was adapted from an existing CORDIC implementation which was used for calculating $\sin(x)$ and $\cos(x)$ values. The original implementation is available as a demo application in Altera's DSP Builder software. The adaptation required restructuring the CORDIC by modifying the decision signal "*decision_zsign*", and adding some control circuitry to the shift registers.

Please refer to Figure 4.5 for a diagram of internals of the frequency offset estimation block. Figure 4.6 shows the typical behaviour of the frequency offset estimation circuitry when the input sequence contains a packet preamble. In this figure, the top graph shows the output from the CORDIC block, and the lower graph shows the coarse frequency offset estimate.

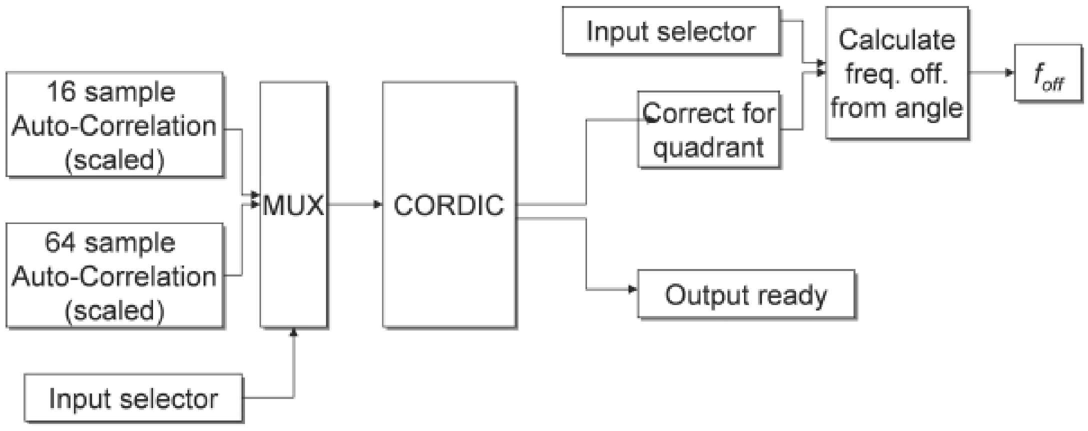

Input selector
Calculate freq. off. from angle
f_{off}
16 sample Auto-Correlation (scaled)
MUX
CORDIC
Correct for quadrant
64 sample Auto-Correlation (scaled)
Output ready
Input selector

Figure 4.5: Internals of the Frequency Offset Estimation Block

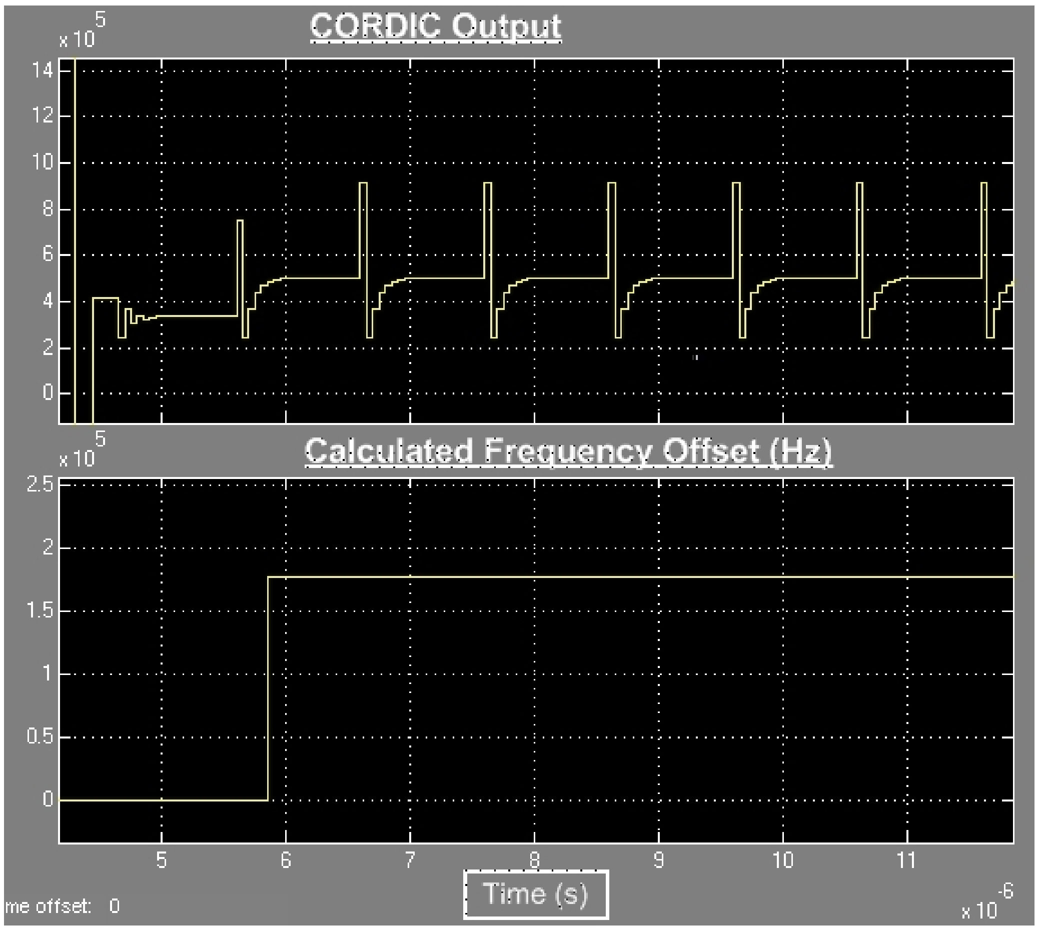


Figure 4.6: Typical Frequency Offset Estimator Output

To test the estimator, and to analyze its behaviour in varying channel conditions, a series of tests were performed, using a series of different parameters. The SNR was kept constant at 10 dB, and the estimator performance was examined in the face of various delay spreads and frequency offset values. The results of these trials are summarized in Table 4.1. These results show that the accuracy of the estimator is relatively good, as all of the estimates are quite close to the actual frequency offset introduced by the wireless channel.

Freq. Offset (kHz)	**Delay Spread (ns)**	**Max. Abs. Calculation Error (kHz)**
0	50	6.33
0	150	6.33
100	50	3.67
100	150	3.67
200	50	22.13
200	150	22.13

Table 4.1: Frequency Offset Estimator Results

4.3 Coarse Time Synchronization

This section covers each of the coarse time synchronization possibilities outlined in Section 3.2. The three algorithms under consideration are the “Basic Auto-Correlation” method, which refers to the method outlined in [LL04], the “Auto-Correlation Difference” method, which refers to the method outlined in [KFST04], and the “Auto-Correlation Sum” method, which refers to the method outlined in [FE03].

4.3.1 Basic Auto-Correlation Method

In the first case, the auto-correlation calculating hardware is reused. The $R_1(d)$ value calculated in Figure 4.1 is used as the input to a detector. This detector includes a counter, which is initiated when the Packet Detector asserts that a packet has been detected, and continues counting while the output of the auto-correlator exceeds a particular threshold. The threshold used is once again 50% of the incoming signal power.

The detector takes as input the auto-correlator output as well as some control signals, and outputs the counter output as well as a control signal, and the counter output is held for a certain number of samples. The circuit includes the counter, a simple state machine, and some additional control circuitry. Without considering AGC, the minimum output value for the circuit would be 136 samples (from the first preamble sample), and the maximum possible would be 176 samples.

Figure 4.7 shows the typical behaviour of the circuitry when the input sequence contains a packet preamble. In this figure, the top graph shows the output of the counter, and the lower graph shows the control signal which notifies the circuit that the time synchronization has been completed.

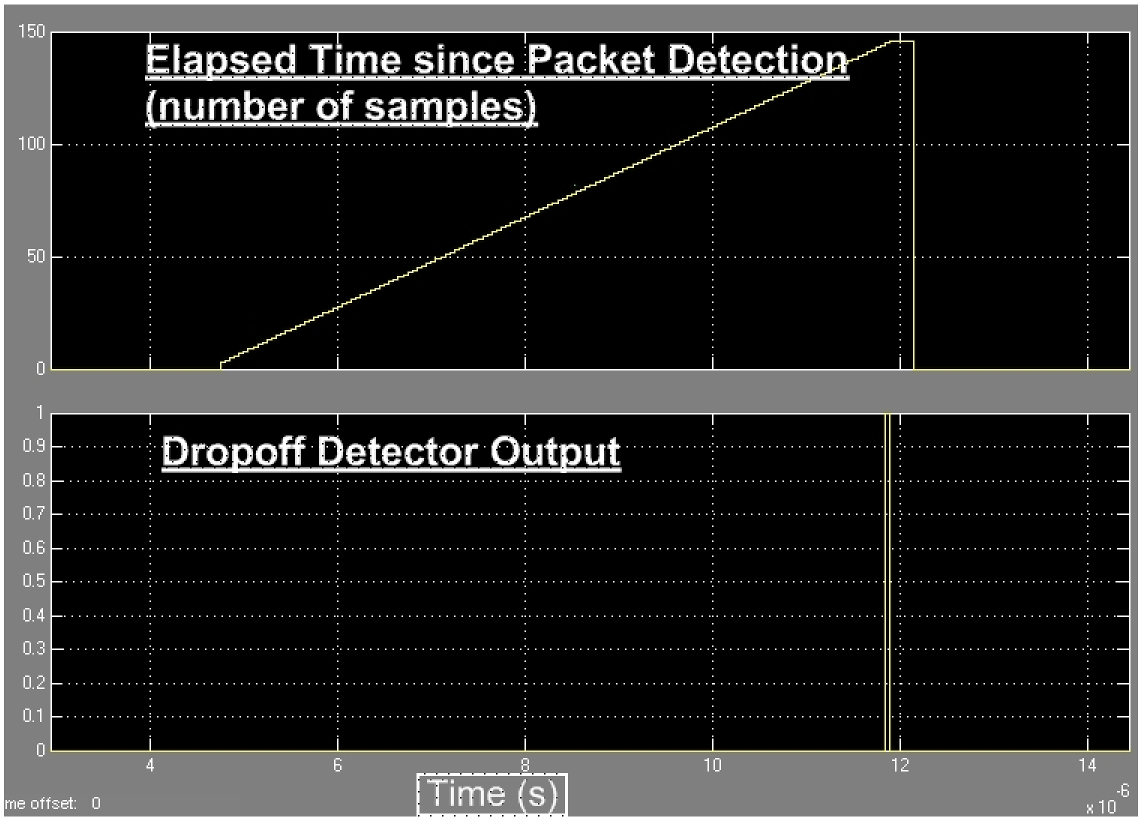
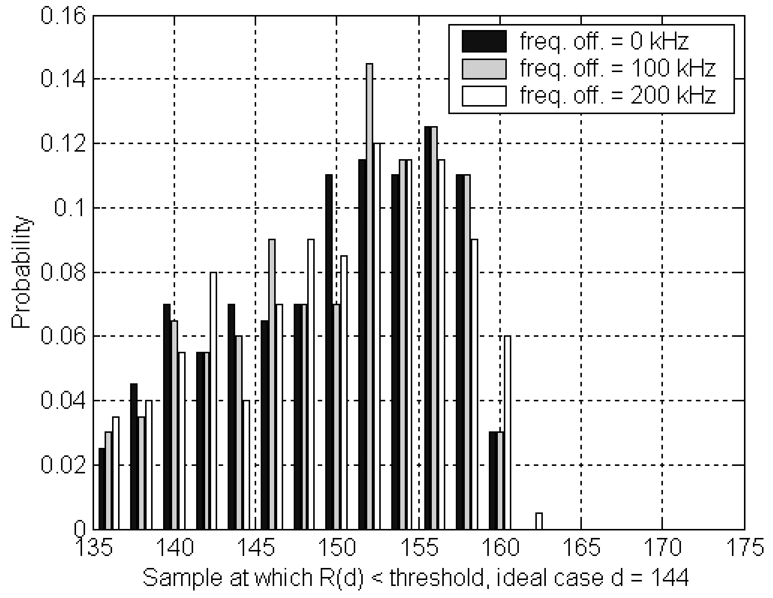


Figure 4.7: Typical Basic Auto-Correlator Time Synchronization Output

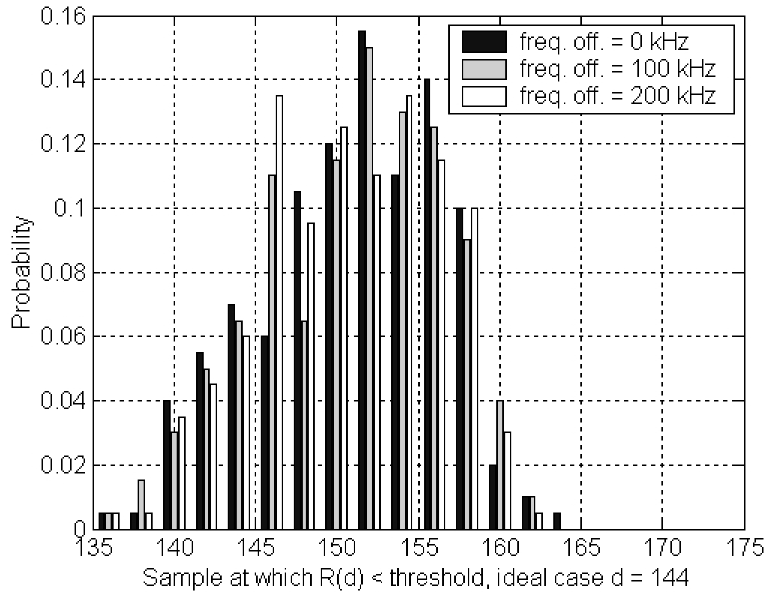
To test the validity and performance of this algorithm, the circuit was tested under varying delay spread, frequency offset, and SNR conditions, as detailed in Section 3.4. In each set of conditions, the circuit was tested 200 times, with different random seeds, meaning that in each trial the output from the channel was different.

If there were perfect channel conditions, the auto-correlator output would drop below the threshold somewhere around sample 144. However, because of the noise, and the effects of the multi-path fading, the sample value at which the output drops below the threshold varies, and can be different in each case.

Figure 4.8 shows the behaviour of the time synchronization circuitry under delay spreads of 50 ns and 150 ns respectively, displaying the sample value at which the auto-correlator output drops below the threshold. What is evident in these figures is that the variance is quite high with this method, mainly because the noise and the delay spread can affect the timing point significantly. However, this method preforms similarly irrespective of the frequency offset.



(a) Delay Spread of 50 ns



(b) Delay Spread of 150 ns

Figure 4.8: Distribution of Time Synchronization Calculations for SNR=10 dB, Using the Basic Auto-Correlation Method

4.3.2 Auto-Correlation Difference Method

In this method, the 16 sample $R_1(d)$ calculation from Figure 4.1 is reused, and a 32 sample auto-correlator is introduced. The outputs from these two correlators are subtracted, and the output of this subtractor is fed to a peak detector. The location of this peak is taken to be the timing offset point.

The hardware requirements for this method include a second auto-correlator, with a delay value of 32 samples rather than 16, a subtractor, and a peak detector. The peak detector includes an index counter, which starts when a packet is first detected and concludes when the peak of the subtracter output is detected. It also includes a state machine, as well as several other pieces of control circuitry. The peak detector must be able to ignore spikes that occur in the subtracter output, and produce a time synchronization estimate only when a genuine peak has occurred. The block diagram for the Auto-Correlation Difference Algorithm is given in Figure 4.9.

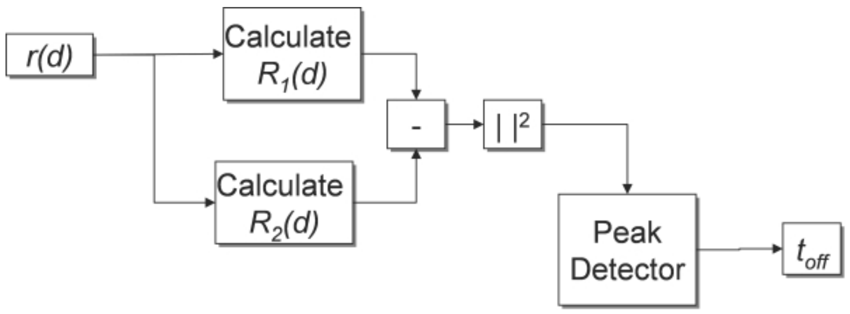
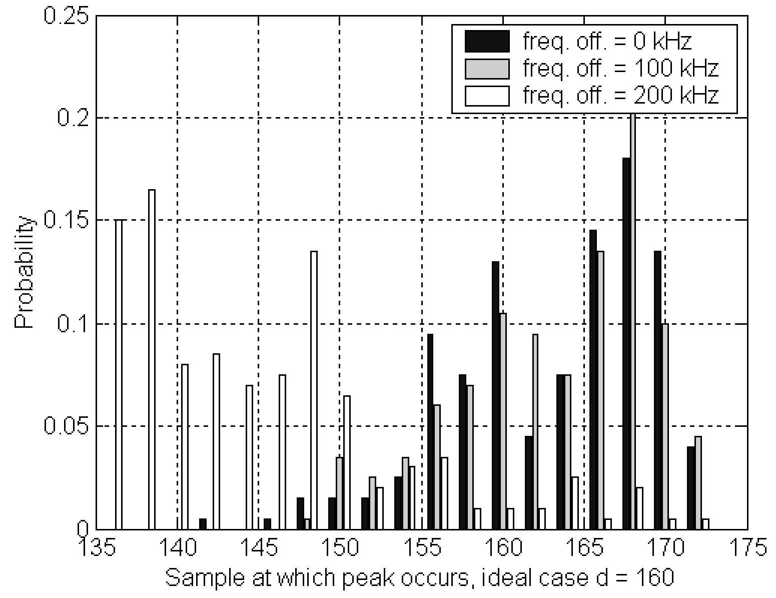


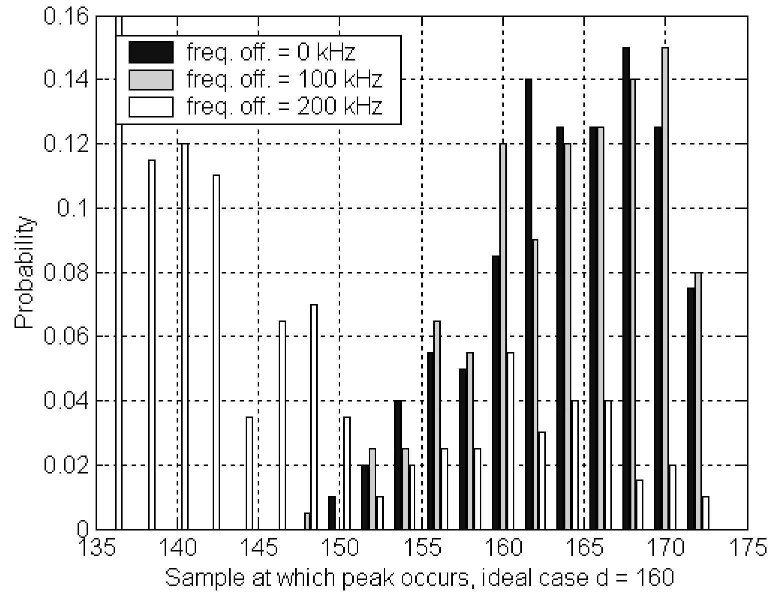
Figure 4.9: Block Diagram for the Auto-Correlation Difference Algorithm

The same tests which were performed on the basic auto-correlator method circuit were performed on this auto-correlator subtracter circuit. Without noise or channel effects, the Difference method would produce a peak around sample 160. However, due once again to channel effects and noise, the sample at which the peak will occur varies.

Figure 4.10 shows the behaviour of the time synchronization circuitry under delay spreads of 50 ns and 150 ns, displaying the sample value at which the peak in the difference signal occurs. As evidenced by these figures, the variance of the Auto-Correlation Difference method is quite comparable with the basic method at lower frequency offsets, however, at a frequency offset of 200 kHz, the performance is far worse, and the resulting estimates have a very large variance. The main weakness in this algorithm seems to be that a well-defined peak is difficult to locate under extreme channel conditions.



(a) Delay Spread of 50 ns



(b) Delay Spread of 150 ns

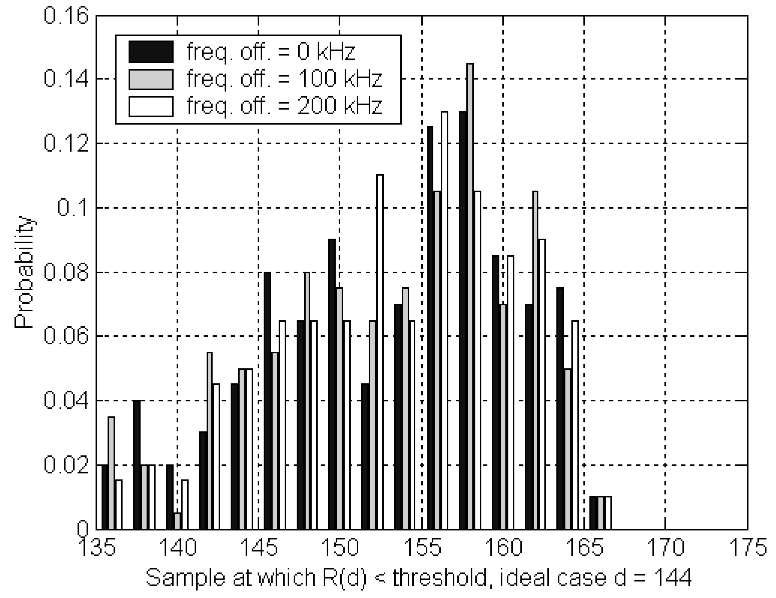
Figure 4.10: Distribution of Time Synchronization Calculations for SNR=10 dB, Using the Auto-Correlation Difference Method

4.3.3 Auto-Correlation Sum Method

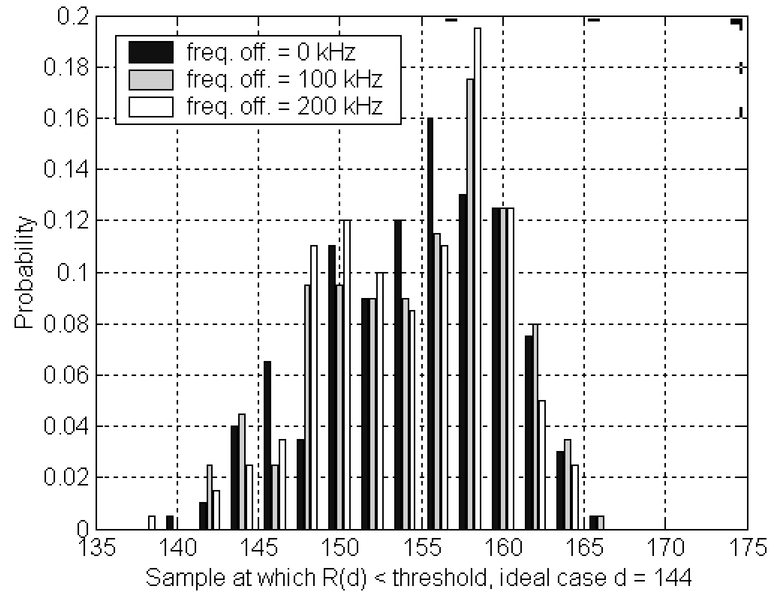
In this method, the calculation of $R_1(d)$ given in Figure 4.1 is reused, with the addition of a delay element, which delays the incoming samples by 32 clock cycles.

As in the Basic Auto-Correlation Estimator, a detector is required to observe the point at which the auto-correlation value falls below a particular threshold, and this detector is identical to the one in the Basic Auto-Correlator. The same tests which were performed on the basic auto-correlator method circuit were performed on this circuit. This method should produce a drop below the threshold level around sample 144, just as in the first case.

Figure 4.11 shows the behaviour of the time synchronization circuitry under delay spreads of 50 ns and 150 ns respectively, displaying the sample value at which the auto-correlator output drops below the threshold value. This algorithm, like the Basic Auto-Correlation Algorithm, has a large variance due to the effects of the channel and noise. The performance is similar to that of the basic method in all cases.



(a) Delay Spread of 50 ns

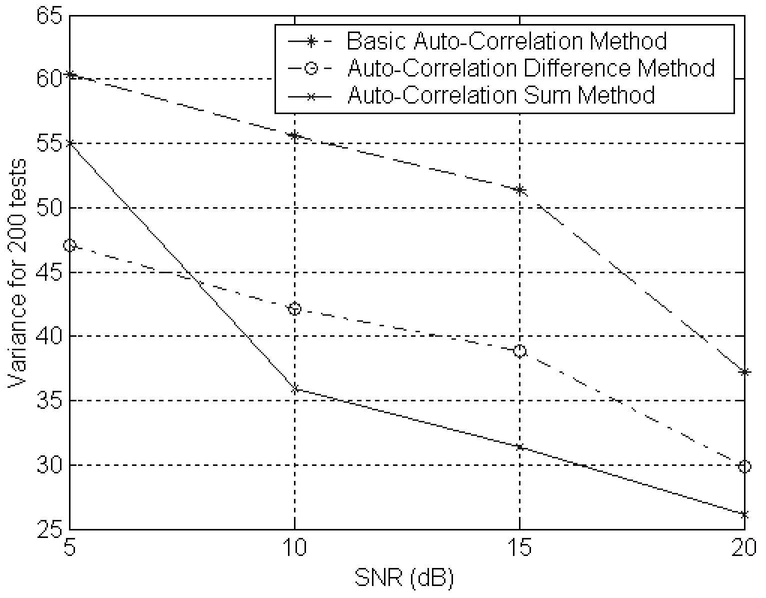


(b) Delay Spread of 150 ns

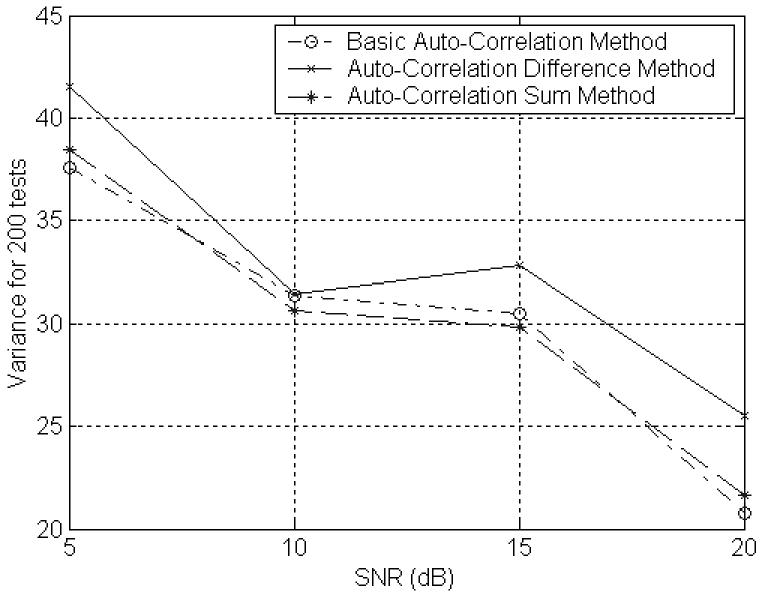
Figure 4.11: Distribution of Time Synchronization Calculations for SNR=10 dB, Using the Auto-Correlation Sum Method

4.3.4 Performance of Estimators under Varying Channel SNR

In order to compare the coarse time synchronizers, the variance of their estimates under various channel conditions is compared. The variance is examined under a set of SNR values, with a constant frequency offset of 100 kHz, and for delay spreads of 50 ns and 150 ns, respectively. The results can be seen in Figure 4.12. It can be seen that all three algorithms have large variance, and are very susceptible to channel noise. It is clear that greater precision will be needed for correct synchronization.



(a) Delay Spread of 50 ns



(b) Delay Spread of 150 ns

Figure 4.12: Distribution of Time Synchronization Calculations for Varying SNR Values, Frequency Offset of 100 kHz

4.4 Fine Time Synchronization

After calculating a coarse timing offset value using the STS, fine time synchronization can be calculated using the LTS. This involves adding more hardware to the circuit to implement a cross-correlation calculator. It also involves selecting an appropriate detection metric, among those discussed in Section 2.4.

4.4.1 Cross-Correlation Calculation

In a cross-correlator, the incoming sequence is simply multiplied by constant values which are identical to the original synchronization sequence which was submitted over the channel, as per Equation 2.11. The circuity required for this cross-correlator is composed of multipliers and adders. The architecture for one of these multiply-accumulate blocks is given in Figure 4.13. The total circuit would be composed of 64 copies of this circuit. The $c(m)$ values are loaded from the MATLAB simulation environment, and thus can be reprogrammed for testing purposes.

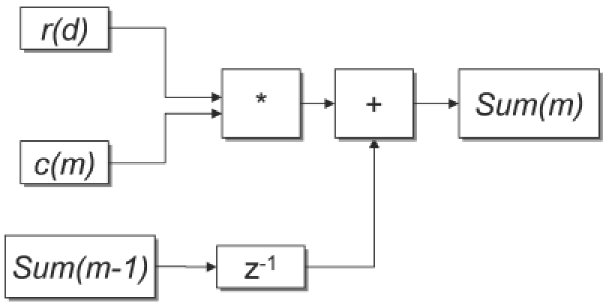


Figure 4.13: Multiply-Accumulator Circuit for Cross-Correlator

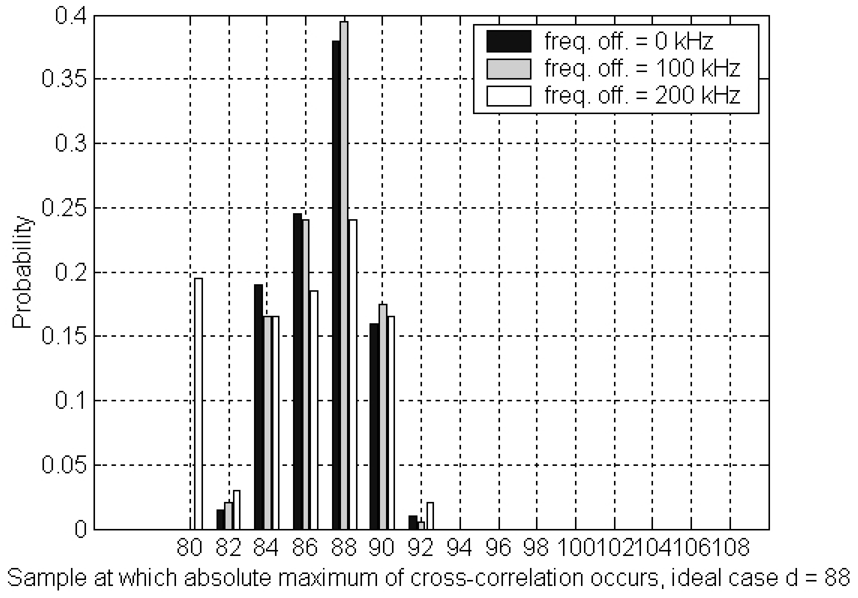
The inputs to this cross-correlator block are the signal sequence received over the channel, and the control signal which enables the circuit. The output of this cross-correlator goes to a peak detector. This peak detector can either search for the maximum cross-correlation value, or it can find the first index value at which the cross-correlation exceeds a particular threshold, corresponding to the two detection metrics given by Equations 2.12 and 2.14.

The hardware required for implementing these detectors is fairly straightforward. The maximum detector searches for the index of the maximum value which occurs within the expected sample range. The other detector, the "Minimum Threshold Detector", searches for the index of the first sequence sample which exceeds a predetermined threshold value. Note that both of these detectors search for a peak within the first 100 samples after they are enabled. This is due to the fact that the cross-correlation peak ideally should occur at sample 96 of the LTS. However, there is a minimum of an 8 sample delay before the cross-correlation detector is activated, so the timing point would occur at sample 88 if there was no timing offset present.

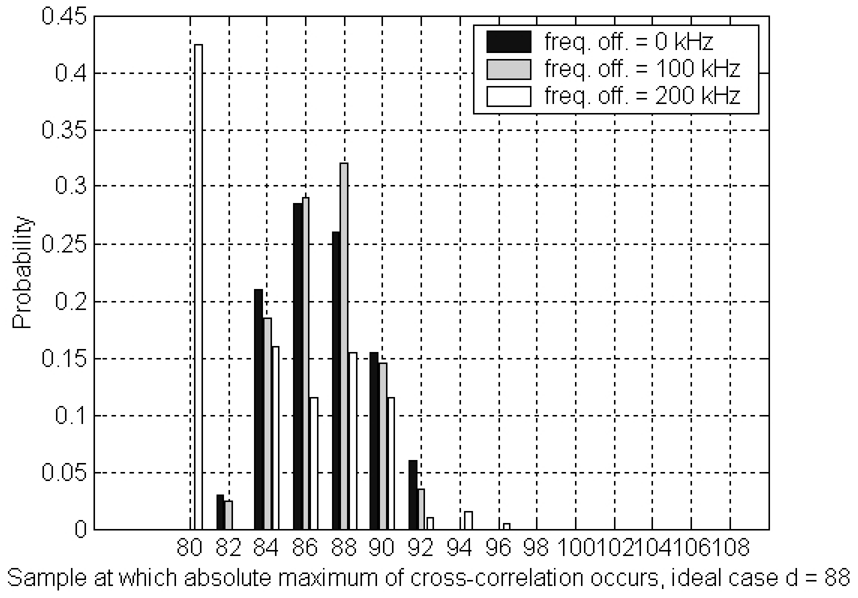
To test the behaviour of the cross-correlator and the two detectors, a series of tests were performed, with varying delay spreads, frequency offsets, and SNR values. The performance of cross-correlator with the two detection methods, at an SNR of 10 dB, with varying frequency offset values, and with delay spreads of 50 ns and 150 ns, is displayed in Figures 4.14 and 4.15. The x-axes in these plots show the sample at which the peak occurs, indicating the timing offset point.

The charts show that the cross-correlator performs extremely well with both detectors. The only exception is the higher variance which occurs at frequency offsets of 200 kHz.

The charts show a spike at the sample value of 80, but this indicates the cumulative sum of all instances which occur at sample values less than or equal to 80. With both detectors, the variance is more pronounced with a delay spread of 150 ns than with a delay spread of 50 ns.

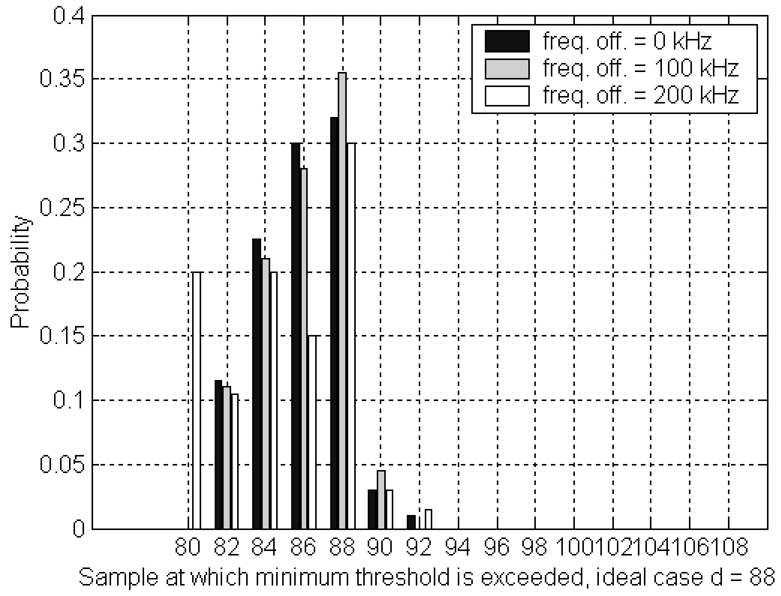


(a) Delay Spread of 50 ns

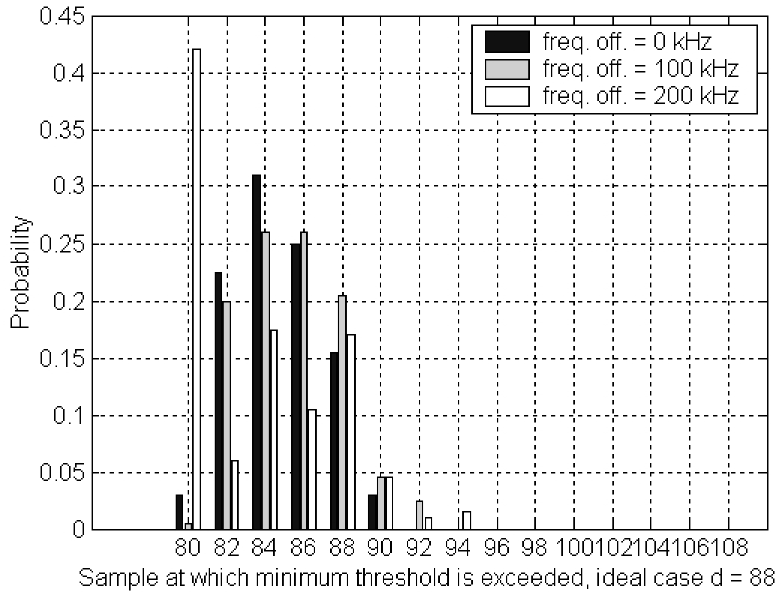


(b) Delay Spread of 150 ns

Figure 4.14: Distribution of Fine Time Synchronization Calculations for SNR=10 dB, Using the Cross-Correlator with the Maximum Detector



(a) Delay Spread of 50 ns



(b) Delay Spread of 150 ns

Figure 4.15: Distribution of Fine Time Synchronization Calculations for SNR=10 dB, Using the Cross-Correlator with the Minimum Threshold Detector

4.4.2 Quantized Cross-Correlator

In the quantized version of the cross-correlator the implementation of the multiply-accumulate circuitry is modified to reduce hardware complexity. Implementing this quantized cross-correlation involved replacing the multipliers in the original cross-correlation circuit with bit-shifters. It also involved taking the constant values that are used in the cross-correlator and replacing them with quantized values, all of which are powers of 2.

The quantization algorithm takes the cross-correlation coefficients, $c^*(m)$, and assigns quantization levels, $q^*(m)$, in their place [HLK03]:

$$q^*(m) = Q_i\left[\frac{2^i c^*(m)}{\max\{c^*(m)\}}\right] \tag{4.1}$$

where the function $Q(x)$ is given by:

$$Q(x) = \begin{array}{ll} 2^{\lfloor \log_2 x \rfloor}, & x > 0 \\ -2^{\lfloor \log_2 x \rfloor}, & x < 0 \\ 0, & x = 0 \end{array} \tag{4.2}$$

The cross-correlation value, $\Lambda(d)$ can then be calculated as follows:

$$\Lambda(d) = \sum_{m=0}^{M-1} \operatorname{sgn}(m) \times [r(d+m) << l(m)] \tag{4.3}$$

where:

$$l(m) = \begin{array}{ll} \log_2 |q^*(m)|, & q^*(m) \neq 0 \\ 0, & q^*(m) = 0 \end{array} \tag{4.4}$$

$$\text{sgn}(m) = \begin{cases} +1, & q^*(m) > 0 \\ \text{-1}, & q^*(m) < 0 \\ 0, & q^*(m) = 0 \end{cases} \tag{4.5}$$

The choice of $i = 3$ in Equation 4.1 corresponds to quantization level of [-8, -4, -2, -1, 0, 1, 2, 4, 8]. Quantization to these levels ensures a sharp reduction in hardware complexity, while maintaining a high level of accuracy in the calculation of $\Lambda(d)$. The architecture for one of quantized shift-accumulate blocks is given in Figure 4.16. The total circuit would be composed of 64 copies of this circuit. The $l(m)$ values are loaded from the MATLAB simulation environment, and thus can be reprogrammed for testing purposes.

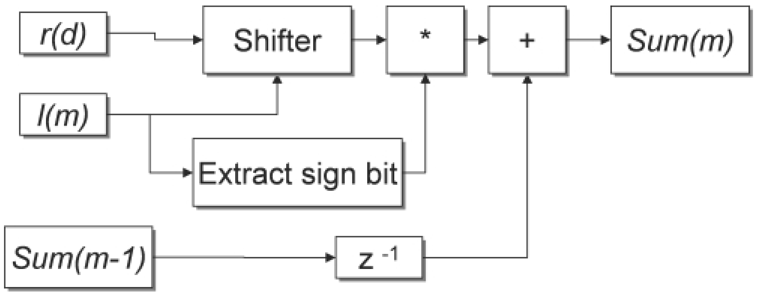
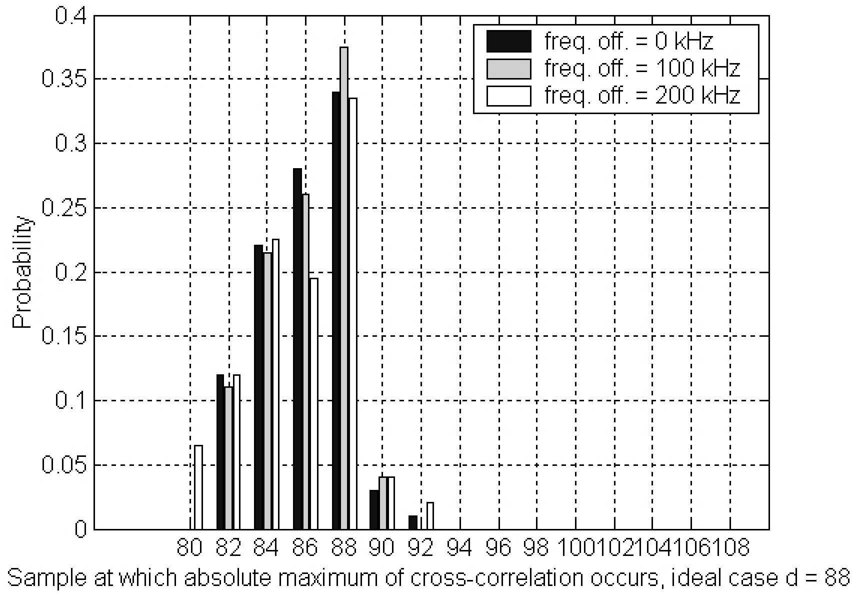


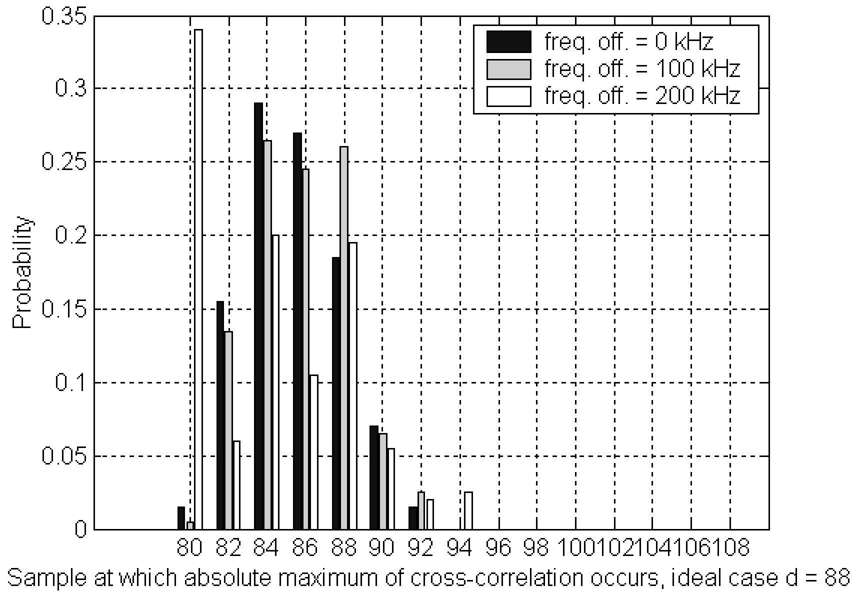
Figure 4.16: Diagram of the Multiply-Accumulate Circuit Used in the Quantized Version of the Cross-Correlator

This quantized cross-correlation hardware was tested using both types of peak detectors, in each case under varying delay spreads, frequency offsets, and SNR values. Once again, in the case of perfect time synchronization, the sample value at which the cross-correlation would be maximized would be sample 88.

The results for the quantized cross-correlator with the Maximum Detector and delay spreads of 50 ns and 150 ns are given in Figure 4.17, while the results with the Minimum Threshold Detector with delay spread values of 50 ns and 150 ns are given in Figure 4.18. Once again, the performance is excellent for both detectors. In all cases, the variance is higher with a frequency offset of 200 kHz, however, the variance with a delay spread of 50 ns is much better with the quantized cross-correlator and the Maximum Detector than in the case of the non-quantized cross-correlator.

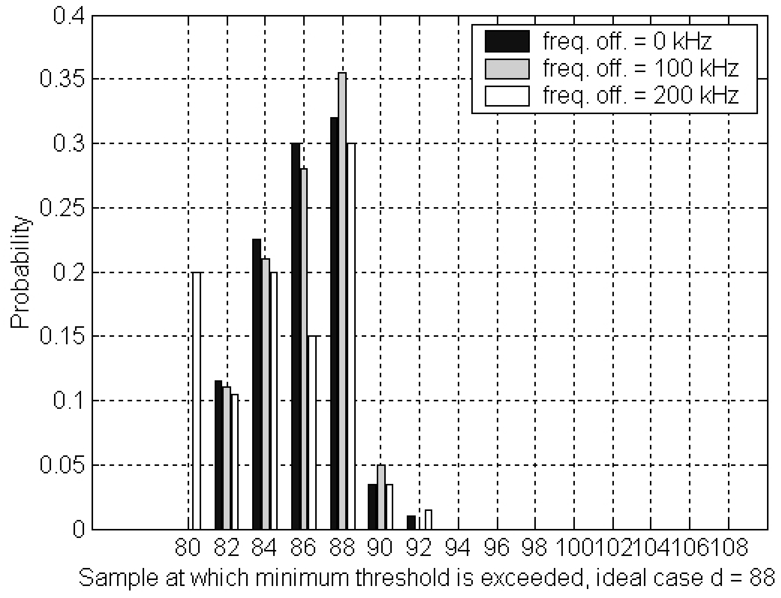


(a) Delay Spread of 50 ns

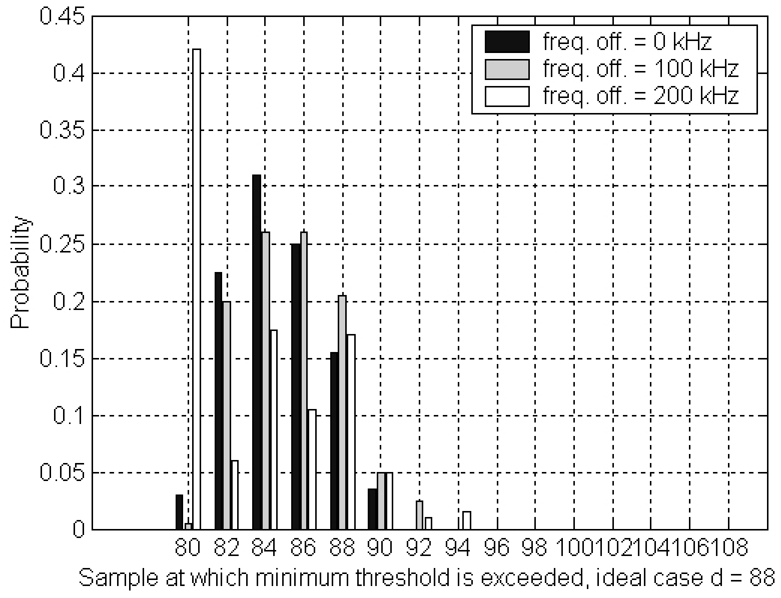


(b) Delay Spread of 150 ns

Figure 4.17: Distribution of Fine Time Synchronization Calculations for SNR=10 dB, Using the Quantized Cross-Correlator with the Maximum Detector



(a) Delay Spread of 50 ns

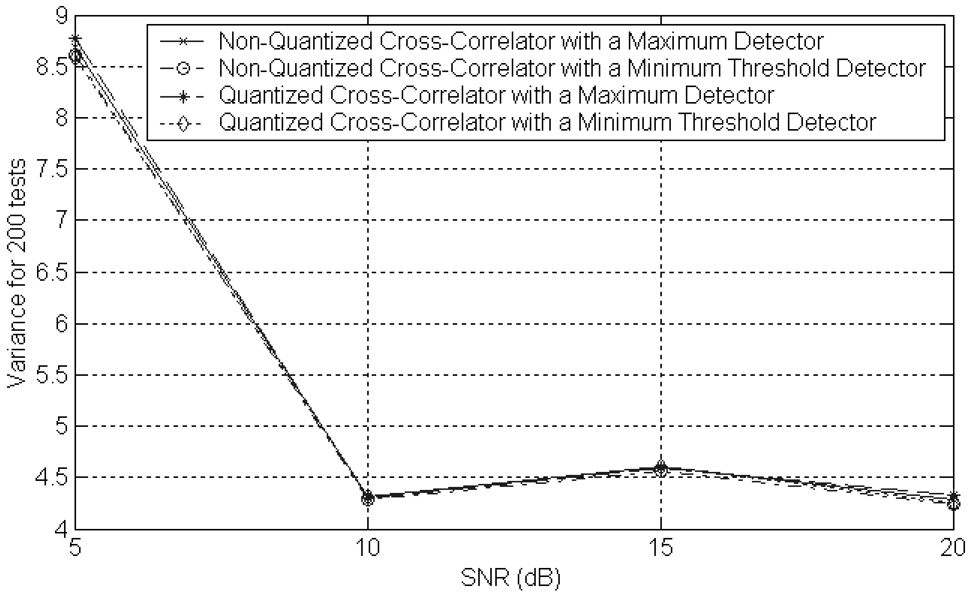


(b) Delay Spread of 150 ns

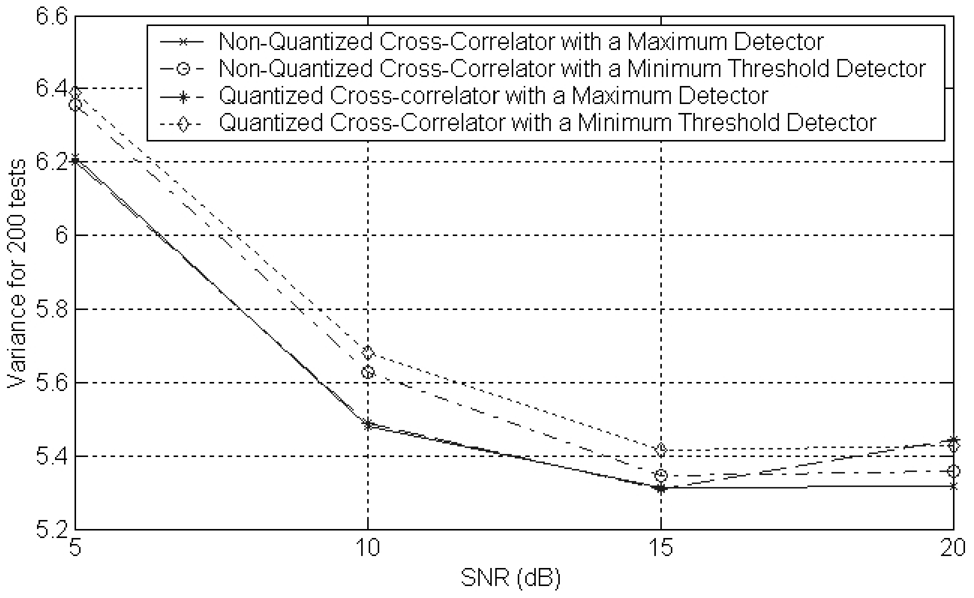
Figure 4.18: Distribution of Fine Time Synchronization Calculations for SNR=10 dB, Using the Quantized Cross-Correlator with the Minimum Threshold Detector

4.4.3 Performance of Cross-Correlation Estimators under Varying Channel SNR

To compare the performance of the quantized and non-quantized cross-correlators, using the two sets of detectors, a series of tests were performed in which the SNR value was varied, with constant delay spread and frequency offset parameters. The results for delay spreads of 50 ns and 150 ns are given in Figure 4.19.



(a) Delay Spread of 50 ns



(b) Delay Spread of 150 ns

Figure 4.19: Distribution of Fine Time Synchronization Calculations for Varying SNR Values, Frequency Offset of 100 kHz

These charts indicate that the performance at a delay spread of 50 ns are quite comparable, while at a delay spread of 150 ns, the Maximum Detector tends to outperform the Minimum Threshold Detector, for both the quantized and non-quantized cases. Despite the stark differences in implementations, the performance of both the quantized and non-quantized cross-correlators is remarkably similar.

4.5 Algorithm Analysis

In accordance with the methodology proposed in Section 3.1, the two metrics of interest are the variance of the timing offset calculations, and the hardware complexity. In Table 4.2, the time synchronization algorithms are compared on the basis of variance.

Algorithm Name	**Var. at 10 dB**	**Var. at 20 dB**
Basic Auto-Correlation method	42.18	29.81
Auto-Correlation Difference method	35.95	26.14
Auto-Correlation Sum method	55.52	37.19
64 sample cross-correlation method, with maximum detection metric	4.32	4.29
64 sample cross-correlation method, with minimum threshold detection metric	4.28	4.24
Quantized 64 sample cross-correlation method, with maximum detection metric	4.31	4.33
Quantized 64 sample cross-correlation method, with minimum threshold detection metric	4.32	4.26

Table 4.2: Summary of Time Synchronization Algorithms, with Delay Spread of 50 ns and Frequency Offset of 100 kHz

The hardware complexity metric for each algorithm is based on the resource requirements for that algorithm. Once again, these are the values output by the synthesis program after the Simulink circuit had been compiled into HDL files.

The hardware resources used by an implementation include the number of ALUTs, the number of registers, and the number of DSP block elements. A summary of the results for each type of algorithm can be seen in Table 4.3. Keep in mind that all of the options listed also include the frequency offset estimation and packet detection circuitry. Also, in the case of the fine time synchronization circuits, the Basic Auto-Correlation circuitry is included for coarse time synchronization.

Algorithm Name	**ALUTs** [1]	**Registers** [2]	**DSP** [3]
Basic Auto-Correlation method	2084	2083	24
Auto-Correlation Difference method	3656	3656	96
Auto-Correlation Sum method	3638	3638	96
64 sample cross-correlation method, with maximum detection metric	9216	4362	34
64 sample cross-correlation method, with minimum threshold detection metric	9230	4363	34
Quantized 64 sample cross-correlation method, with maximum detection metric	4692	4428	34
Quantized 64 sample cross-correlation method, with minimum threshold detection metric	4706	4429	34

Table 4.3: Summary of Hardware Complexity of Synchronization Algorithms

From Table 4.2, it can be seen that the coarse time synchronizer which offers the best performance in terms of variance is the Auto-Correlation Difference estimator. However the estimator which requires the lowest incremental hardware cost is the Basic Auto-

[1]Total ALUTs on Stratix II EP2S180F1020C3: 143520

[2]Total Registers on Stratix II EP2S180F1020C3: 143520

[3]Total DSP Blocks on Stratix II EP2S180F1020C3: 96

Correlation method from [LL04]. On the basis of the lower hardware costs, and also on the basis of the lower variance in channel conditions with high frequency offsets (compare Figures 4.8 and 4.10), the Basic Auto-Correlator seems to be the better choice for coarse time synchronization.

It can also be seen that the fine time synchronizers all have very similar performance in terms of variance, but that the hardware complexity is much lower for the quantized versions of the estimators. The maximum detector has very comparable performance in most of the cases, and seems to perform better at extreme frequency offsets. On the basis of these points, the quantized 64 sample cross-correlator with a maximum detector seems to be the best option for fine time synchronization.

4.6 Final Implementation

Given that the final implementation of the synchronizer must include both the packet detection and frequency offset estimator, the decisions the designer must make are in regards to time synchronization. On the basis of performance capability and low hardware complexity, the Basic Auto-Correlator is preferred for coarse time synchronization. The next decision is whether or not to include a fine time synchronization. On the basis of the low incremental cost and the vast performance increase, the quantized fine time estimator is preferred. The additional size required by the quantized estimator is not a big issue considering the large size of the FPGA being used. The detector can be either the maximum detector or the minimum threshold detector, but on the basis of the better performance at high frequencies, the maximum detector is preferred.

The block diagram for the final synchronizer is shown in Figure 4.20.

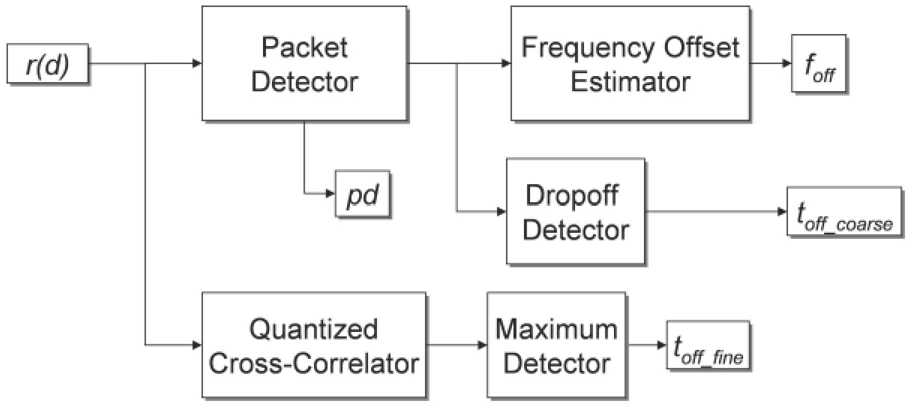


Figure 4.20: Final Synchronizer Implementation

4.7 Total Hardware Requirements

The hardware requirements for the combination of the packet detector, the frequency offset estimation circuitry, the Basic Auto-Correlator, and the quantized 64 sample cross-correlator with the maximum detector are given in Table 4.4:

ALUTs	**Registers**	**DSP elements**
4692	4428	34

Table 4.4: Hardware Complexity for the Final Synchronizer Design

Chapter 5

Conclusions and Future Work

With the widespread adoption of IEEE 802.11 as a standard for Wireless LANs, and the higher data rates afforded by OFDM based implementations, the importance of good receiver algorithms for this class of systems is paramount. This work attempted to examine one class of algorithms, synchronization algorithms, for the particular case of the IEEE 802.11a WLAN standard. The primary goal was the implementation of reliable, efficient, and reconfigurable hardware.

This work proposed a complete examination of the decision-making, design, and implementation engineering phases for an OFDM synchronizer. It contributed a sound methodology for choosing between competing time synchronization algorithms, based on statistical variance and incremental hardware complexity.

This work examined three different auto-correlation algorithms for coarse time synchronization, and four different cross-correlation algorithms for fine time synchronization. It was determined that coarse time synchronization alone would not be enough because of the large variance produced during algorithm simulations. Fine time synchronization, on the other hand, offered much lower variance.

After a careful analysis of competing algorithms, it was decided that the best choice for time synchronization was to use the Basic Auto-Correlation estimator. It was also decided that the quantized 64 sample cross correlator, in conjunction with the maximum detector, would be used for fine time synchronization.

This work also introduced several new features not yet seen in the literature, such as packet detection averaging, an implementation of a quantized cross-correlation circuit, and compensation for angles outside of the first quadrant during frequency offset calculation. The main difference separating this work from previous works is the thorough documentation of a synchronizer implementation. The final synchronizer design was shown, and the total hardware complexity of the circuit was calculated.

An obvious area for future research is in the arena of MIMO-OFDM systems. The research done in this work could be built upon and extended to the case of multiple antennae systems. In particular, the Simulink work done during the course of this research can be configured with very little incremental effort. With the advent of IEEE 802.11n as a standard for MIMO-OFDM, this research would be very useful in coming years.

A few areas not considered in great detail in this work were the optimal fixed point precision for the synchronizer circuit, and how the circuit would be affected by the presence of an AGC component. Future works should look into these aspects for a more complete consideration of the synchronizer implementation problem. Another possibility is for ASIC researchers to use the research done in this work as the basis for an investigation into the optimal implementation possibilities for OFDM receiver synchronization on ASIC hardware rather than FPGAs.

In summary, the accomplishments of this dissertation include a novel approach to algorithm analysis, the introduction of new features not yet seen in the literature, and the complete documentation of a synchronizer design. In the end, the synchronizer design was a success, as it consumed an relatively low quantity of hardware resources, and produced excellent results for packet detection, frequency offset estimation, and time synchronization.

Appendix A

Structure of the IEEE 802.11a Preamble

A short training symbol has the following subcarrier assignment:

$$S_{-26,26} = \sqrt{(13/6)} \times \begin{bmatrix} 0, & 0, & 1+j, & 0, & 0, & 0, & -1-j, & 0, & 0, \\ 0, & 1+j, & 0, & 0, & 0, & -1-j, & 0, & 0, & 0, \\ -1-j, & 0, & 0, & 0, & 1+j, & 0, & 0, & 0, & 0, \\ 0, & 0, & 0, & -1-j, & 0, & 0, & 0, & -1-j, & 0, \\ 0, & 0, & 1+j, & 0, & 0, & 0, & 1+j, & 0, & 0, \\ 0, & 1+j, & 0, & 0, & 0, & 1+j, & 0, & 0 & \end{bmatrix} \tag{A.1}$$

This is brought into the time domain by performing an IFFT, yielding $r_{\mathrm{SHORT}}(t)$, the transmitted STS:

$$r_{\mathrm{SHORT}}(t) = w_{\mathrm{T_{SHORT}}}(t) \sum_{k=-N_{ST}/2}^{N_{\mathrm{ST}}/2} S_k e^{k\mathrm{j}2\pi\Delta f t} \tag{A.2}$$

Where $T_{\mathrm{SHORT}} = 8\,\mu$s is the total length of the STS, $N_{\mathrm{ST}} = 52$ is the number of subcarriers, and $\Delta_F = 312.5\,\text{kHz}$, which is the subcarrier frequency spacing [IEE99b]. The $w(t)$ term is a time-windowing function, used to smooth the transition between OFDM symbols [IEE99b]:

$$w_{\mathrm{T}}(nT_{\mathrm{s}}) = \begin{array}{ll} 1 & 1 \leq n \leq 79 \\ 0.5 & n = 0, 80 \\ 0 & \text{otherwise} \end{array} \tag{A.3}$$

Where T_{s} is the sample period, which is 50 ns in 802.11a.

A long training symbol has the following subcarrier assignment:

$$L_{-26,26} = \begin{bmatrix} 1, & 1, & \text{-1}, & \text{-1}, & 1, & 1, & \text{-1}, & 1, & \text{-1}, \\ 1, & 1, & 1, & 1, & 1, & 1, & \text{-1}, & \text{-1}, & 1, \\ 1, & \text{-1}, & 1, & \text{-1}, & 1, & 1, & 1, & 1, & 0, \\ 1, & \text{-1}, & \text{-1}, & 1, & 1, & \text{-1}, & 1, & \text{-1}, & 1, \\ \text{-1}, & \text{-1}, & \text{-1}, & \text{-1}, & \text{-1}, & 1, & 1, & \text{-1}, & \text{-1}, \\ 1, & \text{-1}, & 1, & \text{-1}, & 1, & 1, & 1, & 1 & \end{bmatrix} \tag{A.4}$$

Performing the IFFT yields $r_{\mathrm{LONG}}(t)$, the transmitted LTS:

$$r_{\mathrm{LONG}}(t) = w_{\mathrm{T}_{\mathrm{LONG}}}(t) \sum_{k=-N_{\mathrm{ST}}/2}^{N_{\mathrm{ST}}/2} S_k e^{k\mathrm{j}2\pi\Delta f(t-T_{\mathrm{G12}})} \tag{A.5}$$

Where $T_{\mathrm{LONG}} = 8\,\mu\text{s}$ is the total length of the LTS, and $T_{\mathrm{G12}} = 1.6\,\mu\text{s}$ is the length of the guard interval inserted before the first long training symbol. Precise preamble sample values are specified in Tables G.4 and G.6 of the IEEE 802.11a standard [IEE99b].

Bibliography

[Ber01] A. Berno. Time and Frequency Synchornization Algorithms for Hiperlan/2. Master's thesis, Universita Degli Studi di Padova, 2001.

[CVA$^+$04a] M.J. Canet, F. Vicedo, V. Almenar, J. Valls, and E.R. de Lima. A Common FPGA Based Synchronizer Architecture For Hiperlan/2 and IEEE 802.11a WLAN Systems. In *PIMRC 2004: 15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications.*, pages 531–535, September 2004.

[CVA$^+$04b] M.J. Canet, F. Vicedo, V. Almenar, J. Valls, and E.R. de Lima. Hardware Design of a FPGA-Based Synchronizer for Hiperlan/2. In *FPL 2004 : Field-Programmable Logic and Applications*, 2004.

[FE03] A. Fort and W. Eberle. Synchronization and AGC Proposal for IEEE 802.11a Burst OFDM Systems. In *GLOBECOM*, pages 1335–1338, 2003.

[Fla03] Flarion Technologies Inc. *OFDM for Mobile Data Communications*, 2003.

[FWD$^+$03] A. Fort, J. Weijers, V. Derudder, W. Eberle, and A. Bourdoux. A Performance and Complexity Comparison of Auto-Correlation and Cross-Correlation for OFDM Burst Synchronization. In *ICASSP '03*, pages II 341–344, April 2003.

[HG02] D.Y. Huang and D.Y.L. Guek. Performance Analysis of OFDM Burst Synchronization Algorithms. In *ICCS 2002: The 8th International Conference on Communication Systems, 2002.*, pages 245–249, November 2002.

[HLK03] T. Ha, S. Lee, and J. Kim. Low-complexity Correlation System for Timing Synchronization in IEEE 802.11a Wireless LANs. In *RAWCON '03: Radio and Wireless Conference, 2003.*, pages 51–54, August 2003.

[HT01] J. Heiskala and J. Terry. *OFDM Wireless LANs: A Theoretical and Practical Guide*. Sams, 2001.

[IEE99a] IEEE Standard 802.11-1999. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE, 1999.

[IEE99b] IEEE Standard 802.11a-1999. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: High-Speed Physical Layer in the 5 GHz Band*. IEEE, September 1999.

[IEE99c] IEEE Standard 802.11b-1999. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: High-Speed Physical Layer in the 2.4 GHz Band*. IEEE, September 1999.

[KFST04] K.Wang, M. Faulkner, J. Singh, and I. Tolochko. Timing Synchronization for 802.11a WLANs under Multipath Channels. In *Proc. ATNAC 2003*, 2004.

[KP07] T. Kim and S.C. Park. A New Symbol Timing and Frequency Synchronization Design for OFDM-based WLAN Systems. In *ICACT '07*, pages 1669–1672, February 2007.

[KSF04] K.Wang, J. Singh, and M. Faulkner. FPGA Implementation of an OFDM-WLAN Synchronizer. In *DELTA 2004: Second IEEE International Workshop on Electronic Design, Test and Applications, 2004.*, pages 89–94, January 2004.

[LL04] J. Liu and J. Li. Parameter Estimation and Error Reduction for OFDM-Based WLANs. *IEEE Transactions on Mobile Computing*, 3(2), April 2004.

[MKB06] S.K. Manusani, R.S. Kshetrimayum, and R. Bhattacharjee. Robust Time and Frequency Synchronization in OFDM based 802.11a WLAN systems. In *Annual India Conference, 2006*, pages 1–4, September 2006.

[MS04] F. Manavi and Y. Shayan. Implementation of OFDM modem for the Physical Layer of the IEEE 802.11a Standard Based on Xilinx Virtex-II FPGA. In *IEEE 59th Vehicular Technology Conference, 2004*, pages 1768–1772, May 2004.

[NG03] S. Nandula and K. Giridhar. Robust Timing Synchronization for OFDM-based Wireless LAN System. In *TENCON*, pages 1558–1561, 2003.

[SC97] T. Schmidl and D. Cox. Robust Frequency and Timing Synchronization for OFDM. *IEEE Transactions on Communications*, 45(12), December 1997.

[Uys06] M. Uysal. ECE 414 Course Notes, 2006.

[YCK06] J. Yang, K. Cheun, and J. Kim. Symbol Timing Synchronization Algorithm for Wireless LAN Systems in Multipath Channels. In *APCC '06. Asia-Pacific Conference on Communications, 2006.*, pages 1–5, August 2006.

[YNW02] K. Yip, T. Ng, and Y. Wu. Impacts of Multipath Fading on the Timing Synchronization of IEEE 802.11a Wireless LANs. *Proc. ICC 2002*, pages 517–521, 2002.

[ZS04] L. Zhou and M. Saito. A New Symbol Timing Synchronization for OFDM based WLANs Under Multipath Fading Channels. In *PIMRC 2004: 15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, 2004.*, pages 1210–1214, September 2004.