DIGITAL IMPLEMENTATION OF ETSI OFDM
SYMBOL SYNCHRONIZER BASED ON SLIDING CORRELATION

by

RIZA DÖNMEZ

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
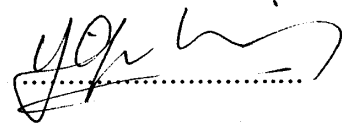the requirements for the degree of Master Science
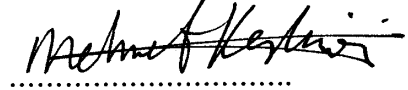

Sabancı University
May 2003

DIGITAL IMPLEMENTATION OF ETSI OFDM
SYMBOL SYNCHRONIZER BASED ON SLIDING CORRELATION

APPROVED BY:

Assoc. Prof. Dr. Yaşar GÜRBÜZ
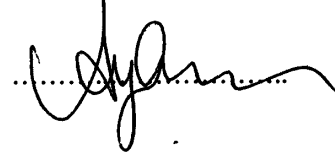(Thesis Advisor)

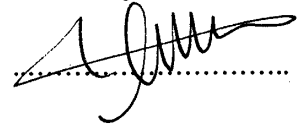Asist. Prof. Dr. Mehmet KESKİNÖZ
(Thesis Co-Advisor)

Asist. Prof. Dr. Meriç ÖZCAN
(Jury Member)

Asist. Prof. Dr. Ayhan BOZKURT
(Jury Member)

Asist. Prof. Dr. İbrahim TEKİN
(Jury Member)

DATE OF APPROVAL: 17.05.2003

ABSTRACT

This thesis presents the design, implementation, verification and synthesis of a digital hardware, which performs OFDM symbol synchronization using short training symbols (STS) defined in European Telecommunications Standards Institute (ETSI) HiperLan/2 Physical Layer specifications. Designed ETSI OFDM Symbol Synchronizer IP was synthesized in CMOS 0.13µm technology using Virtual Silicon Technology (VST) Standard Cell Libraries.

In this thesis, we first explain OFDM and OFDM systems in detail. Synchronization problems occurring in OFDM systems are classified and techniques used to overcome these problems are presented. Then a digital ETSI OFDM Symbol Synchronizer IP, which performs OFDM symbol synchronization task based on the correlation of the received symbols, is proposed. Proposed architecture has been designed using VHDL (VHSIC Hardware Description Language) in the implementation part of the thesis. Designed IP has been verified functionally first, then synthesized in CMOS 0.13µm technology. Gate-level verification has been also performed after synthesis of the IP.

Like other communication systems, synchronization is a critical problem to be solved in OFDM systems. One of the arguments against OFDM is that it is highly sensitive to synchronization errors. Before an OFDM receiver can demodulate the sub-carriers, it has to perform at least two synchronization tasks: First, it has to find out where the symbol boundaries are. Second, it has to estimate and correct the carrier frequency offset of the received signal and clock offset between transmitter and receiver because any offset introduces Inter-carrier interference (ICI) and Inter-symbol interference (ISI). This work aims to review OFDM and synchronization issues in OFDM systems and to design a digital symbol synchronizer hardware that performs the detection of OFDM symbols, which is the first synchronization task mentioned above.

ETSI HiperLAN/2 standard has been used in this work as the reference for all parameters needed and used in the hardware implementation of ETSI OFDM Symbol Synchronizer. Although the needed sampling frequency of OFDM receiver is 20 MHz in the ETSI standards, the designed IP can be run up to 50 MHz. It can be easily adapted to any changes in the standard, such as the increase in speed.

The generically designed ETSI OFDM STS Symbol Synchronizer IP can be integrated to other modules easily and used as part of the whole synchronizer block in ETSI OFDM receivers.

ÖZET

Bu tez Avrupa Telekomünikasyon Standartları Enstitüsü (ETSI), Fiziksel Katman tarifinde açıklanan STS (Short Training Symbols - STS) sembollerini kullanarak OFDM sembol senkronizasyonunu gerçekleyen bir sayısal devrenin tasarımı, uygulanması, sınanması ve sentezlenmesi aşamalarından oluşmuştur. Tasarlanan ETSI OFDM (Orthogonal Frequency Division Multiplexing) Sembol Senkronizasyon devresi, Virtual Silicon Technology (VST) Standart Hücre Kütüphaneleri kullanılarak 0.13 μm sayısal CMOS teknolojisinde sentezlenmiştir.

Bu tezde, öncelikle OFDM ve OFDM sistemleri detaylı olarak açıklanmıştır. OFDM sistemlerinde karşılaşılan senkronizasyon problemleri sınıflandırılarak, bu problemlerin çözümünde kullanılan senkronizasyon teknikleri sunulmuştur. Bunların ardından, alıcıya gelen sembollerin korelasyonuna dayalı OFDM senkronizasyon işlemini gerçekleştiren ETSI OFDM Sembol Senkronizasyon devresi önerilmiştir. Önerilen mimari, tezin uygulama bölümünde VHDL (Çok Yüksek Hızlı Entegre Devre Donanım Tanımlama Dili) kullanılarak gerçeklenmiştir. Bu devre ilk önce işlevsel olarak sınanmış, ardından 0.13 μm sayısal CMOS teknolojisinde sentezlenmiştir. Devrenin sentezi sonrasında ,kapı düzeyinde işlevselliği yeniden test edilmiştir.

Diğer haberleşme sistemlerinde olduğu gibi, senkronizasyon, OFDM sistemlerinde de çözümlenmesi gereken kritik bir sorundur. OFDM senkronizasyon hatalarına çok duyarlı bir yapıya sahiptir. Bir OFDM alıcısı, OFDM alt-taşıyıcılarını demodüle etmeden önce, en azından iki senkronizasyon işlevini yerine getirmek zorundadır: İlki, alıcıya gelen OFDM sembol sınırlarını, bir başka deyimle OFDM sembolünün ne zaman başladığını tespit etmek zorundadır. İkincisi, alınan sinyaldeki taşıyıcı frekans ofsetini ve alıcı ve verici arası saat ofsetini tahmin etmeli ve düzeltmelidir. Zira, herhangi bir ofset taşıyıcılar arası ve semboller arası girişime neden olmaktadır.

Bu çalışma, OFDM ve OFDM sistemlerindeki senkronizasyon olgularını ele almayı ve yukarıda bahsedilen ilk senkronizasyon işlevi olan, alıcıda OFDM sembollerinin saptamasını gerçekleyen bir sayısal sembol senkronizasyon devresi tasarlamayı hedeflemektedir.

ETSI OFDM Sembol Senkronizasyon devresinin uygulamasında ETSI HiperLAN/2 Standardı, tüm parametreler için referans olarak alınmıştır. ETSI standardında, OFDM alıcısının örnekleme frekansı 20 MHz olmasına karşın, tasarlanan devre 50 MHz hıza kadar çalışabilmektedir. Devre, ETSI standardında örnekleme frekansında ileride meydana gelebilecek değişikliği, 50 MHz hıza kadar destekleyebilir.

Jenerik olarak tasarlanan ETSI OFDM STS Sembol Senkronizasyon devresi, diğer modüllerle kolaylıkla birleştirilip, ETSI OFDM alıcılarında tüm senkronizasyonu sağlayan bloğun bir parçası olarak kullanılabilir.

To my wife Handan,
our son Can
and
to my parents.

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

LIST OF FIGURES

xiii

LIST OF TABLES

LIST OF ABBREVIATIONS

ACF	Association Control Function
ACH	Access Feedback Channel
AFC	Automatic Frequency Control
AP	Access Point
ARP	Antenna Reference Point
ARQ	Automatic Repeat Request
AWGN	Additive White Gaussian Noise
ATM	Asynchronous Transfer Mode
BCH	Broadcast Channel
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
BRAN	Broadband Radio Access Networks
CC	Central Controller
CFO	Carrier Frequency Offset
CL	Convergence Layer
CO	Clock Offset
CP	Cyclic Prefix
DAB	Digital Audio Broadcasting
DCC	DLC Connection Control
DFT	Discrete Fourier Transform
DiL	Direct Link
DLC	Data Link Control
DLCC	DLC Connection
DLCC-ID	DLC Connection Identifier
DM	Direct Mode

DUT	Device Under Test
EC	Error Control
EIRP	Effective Isotropic Radiated Power
ETSI	European Telecommunications Standards Institute
FCH	Frame Channel
FEC	Forward Error Correction
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GI	Guard Interval
HIPERLAN	High Performance Local Area Network
HIPERLINK	High Performance Radio Link
Hz	Hertz
ICI	Inter Carrier Interference
IDFT	Inverse Discrete Fourier Transform
IFI	Inter Frame Interference
IFFT	Inverse Fast Fourier Transform
IP	Intellectual Property
ISI	Inter Symbol Interference
LAN	Local Area Network
LCH	Long Transport Channel
LLC	Logical Link Control
LTS	Long Training Symbol
MAC	Medium-Access Controller
ML	Maximum Likelihood
MT	Mobile Terminal
OFDM	Orthogonal Frequency Division Multiplexing
PDU	Protocol Data Unit
PHY	Physical
PPM	Parts Per Million
RCH	Random Channel
RLC	Radio Link Control
RRC	Radio Resource Control
QAM	Quadrature Amplitude Modulation
QoS	Quality of Service

QPSK	Quadrature Phase Shift Keying
RF	Radio Frequency
SCH	Short Transport Channel
SDF	Standard Delay File
SIR	Signal - to - (ISI + ICI) Ratio
SNR	Signal to Noise Ratio
STS	Short Training Symbol
TC	Transport Channel
USAP	User Service Access Point
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
WLAN	Wireless Local Area Network

1. INTRODUCTION

1.1. Motivation

Over the last decade, the market for wireless service has grown at an unprecedented rate. The industry has grown from cellular phones and pagers to broadband and ultra-broadband wireless services that can provide voice, data, and full-motion video in real time. Wireless communications systems are playing currently a major role and expected to play a more important role in providing portable access to future information services.

Within the wide variety of wireless communication systems, there are many modulation techniques in current use. A very important modulation technique, OFDM, is currently of great interest by the researchers in the Universities and research laboratories all over the world since it provides data transmission in a bandwidth-efficient way. Multi-carrier or Orthogonal frequency-division multiplexing (OFDM) systems have gained an increased attention during the last years. It is used in the European digital broadcast radio system. OFDM has already been accepted for the new wireless local area network standards from IEEE 802.11, High Performance Local Area Network type 2 (HIPERLAN/2) and Mobile Multimedia Access Communication (MMAC) Systems.

Like other communication systems, synchronization is a critical problem to be solved in OFDM systems. One of the arguments against OFDM is that it is highly sensitive to synchronization errors. Before an OFDM receiver can demodulate the sub-carriers, it has to perform at least two synchronization tasks. First, it has to find out where the symbol boundaries are and what the optimal timing instants are to minimize the effects of inter-carrier interference (ICI) and inter-symbol interference (ISI). Second, it has to estimate and correct the carrier frequency offset of the received signal because any offset introduces ISI. This work aims to review OFDM and synchronization issues and implement a synchronizer hardware that realizes the first synchronization task

based on sliding correlation. During the course of study, ETSI standards are considered for the design.

This work analyzes OFDM and the synchronization problems in OFDM systems; implements a European Telecommunications Standards Institute (ETSI) OFDM Short Training Symbols (STS) Symbol Synchronizer. This work can be taken as a basis of a doctoral research for implementing a complete OFDM receiver.

1.2.   Thesis Organization

The goal of this thesis is to research OFDM and synchronization problems existing in OFDM systems and design and implement the OFDM STS Symbol Synchronization system based on ETSI standards.

The thesis is organized as follows:

Chapter 2 gives an overview of OFDM. This chapter considers the basic OFDM receiver and transmitter structure and mathematical modeling of the blocks.

Chapter 3 contains synchronization issues in OFDM systems. Symbol and frequency synchronization problems are mentioned in detail followed by the descriptions of the sensitivity of OFDM to synchronization errors and different synchronization techniques.

Chapter 4 covers the design of ETSI OFDM STS Symbol Synchronizer IP including the design pre-study before the implementation, simulation and synthesis. First preamble and correlation characteristics are explained, and then the general information of STS is given with the generated reference OFDM preamble example. Sliding and cross correlation techniques are explained and compared with each other, followed by a discussion on why sliding correlation method is more useful than the cross one for ETSI STS synchronizer. After a short description of the LTS part of preamble, the pre-study section is completed. The proposed architecture for the symbol synchronizer is explained in detail then the achieved results at the end of implementation of ETSI OFDM STS Symbol Synchronizer are presented with simulation and synthesis. Amplitude and phase outputs of the designed symbol synchronizer are compared to the reference matlab model graphically. Results of two syntheses realized with CMOS 0.13μm for 20 MHz and 50 MHz operation frequencies are compared to each other in terms of area and power consumption estimations.

Finally, conclusions are drawn for the study and based on these assessments some possible future research topics are suggested in chapter 5.

2. INTRODUCTION TO OFDM (Orthogonal Frequency Division Multiplexing)

Multi-carrier transmission is the principle of transmitting data by dividing the data stream into several parallel bit streams, each of which has a much lower bit rate [4]. Orthogonal Frequency Division Multiplexing (OFDM) with densely spaced subcarriers and overlapping spectra is a special form of multi-carrier transmission. To obtain a high spectral efficiency, the sub-carrier center frequencies are selected to have minimum values to maintain orthogonality; hence the name OFDM is used.

OFDM is a special case of multi-carrier transmission, where a single data stream is transmitted over a number of lower rate sub-carriers. One of the main advantages to use OFDM is to increase the robustness against distortion caused by frequency selective channel or narrowband interference. In a single carrier system, a single fade or interferer can cause the entire link to fail, but in a multi-carrier system, only a small percentage of the sub-carriers will be affected. Error correction coding can then be used to correct for the few erroneous sub-carriers.

The concept of using parallel data transmission and frequency division multiplexing was published in the mid-1960s [1, 2]. The history of OFDM dates back to the mid 60's, when R. W. Chang published his paper on the synthesis of band-limited signals for multi-channel transmission [1]. He presented a principle for transmitting messages simultaneously through a linear band-limited channel without inter-channel (ICI) and inter-symbol (ISI) interference.

In a classical parallel data system, the total signal frequency band is divided into N non-overlapping frequency sub-channels. Each sub-channel is modulated with a separate symbol and then the N sub-channels are frequency-multiplexed. It is good to avoid spectral overlap of channels to eliminate inter-channel interference. However, this leads to inefficient use of the available spectrum. To cope with the inefficiency, the ideas proposed from mid-1960s were to use parallel data and Frequency Division Multiplexing (FDM) with overlapping sub-channels. Figure 2.1 illustrates the difference between the conventional non-overlapping multi-carrier technique and overlapping

4

multi-carrier modulation technique. As shown in Figure 2.1, by using the over-lapping multi-carrier modulation technique, we save almost 50 % of bandwidth. To realize the overlapping multi-carrier technique, however we need to reduce crosstalk between sub-carriers, which means that we want orthogonality between the different modulation carriers.

The main idea behind OFDM is to split the data stream to be transmitted into *N* parallel streams of reduced data rate and to transmit each of them on a separate sub-carrier. These carriers are made orthogonal by appropriately choosing the frequency spacing between them to obtain a high spectral efficiency. Therefore, spectral overlapping among sub-carriers is allowed, since the orthogonality ensure that the receiver can separate the OFDM sub-carriers and a better spectral efficiency can be achieved than by using simple frequency division multiplex. The word orthogonality here indicates that there is a precise mathematical relationship between the frequencies of the carriers in the system. In a normal frequency-division multiplex system, many carriers are spaced apart in such a way that the signals can be received using demodulators. In such receivers, guard bands are introduced between the different carriers and in the frequency domain, resulting in a lowering of spectrum efficiency. It is possible, however, to arrange the carriers in an OFDM signal so that the sidebands of the individual carriers overlap and the signals are still received without adjacent carrier interference. To do this the carriers must be mathematically orthogonal. Figure 2.2 shows spectra of orthogonal OFDM sub-carriers.
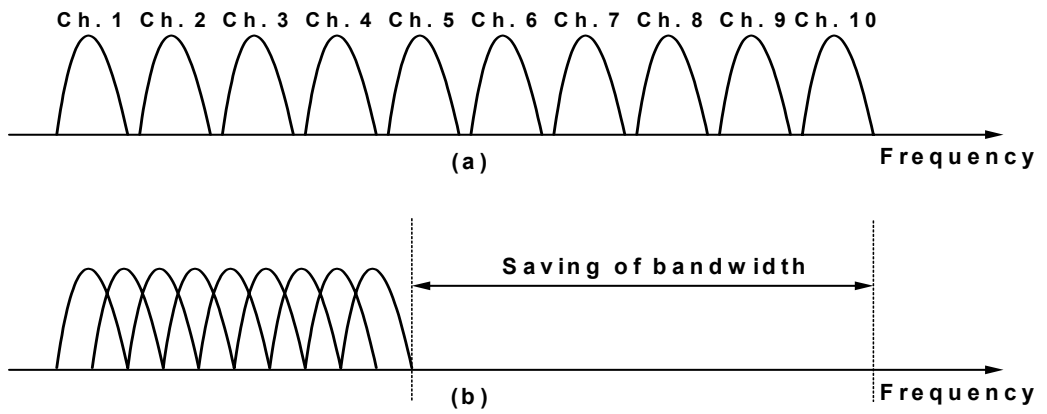
Figure 2.1 Concept of OFDM signal: (a) Conventional multi-carrier technique, (b) Orthogonal multi-carrier modulation technique
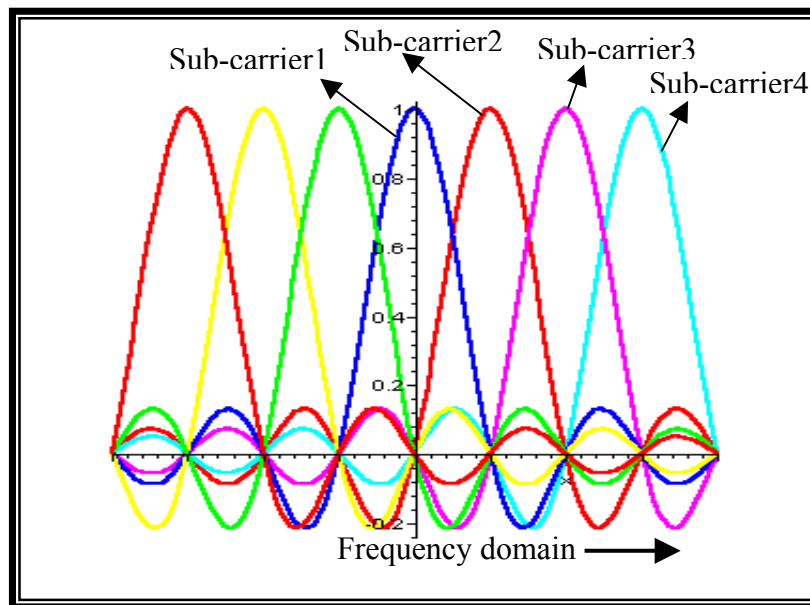
Figure 2.2 Spectra of individual sub-carriers

The general block diagram of an OFDM transceiver is illustrated in Figure 2.3. In the transmitter path, binary input data is encoded. After interleaving, the binary values are converted into QAM values: Each *n*-bit group is assigned to an appropriate complex symbol having a signal constellation according to the used digital modulation technique (QAM). The bits in each group determine the constellation point according to the selected sub-carrier modulation. At this point we have a complex data. After QAM mapping, pilot insertion is realized to facilitate coherent reception. To make the system robust to multi-path propagation, a cyclic prefix is added. Further, windowing is applied to attain a narrower output spectrum. After this step, the digital output signals can be converted to analog signals, which are then up-converted to broadcasting band, amplified and transmitted through an antenna.

The OFDM receiver basically performs the reverse operations of the transmitter, together with additional training tasks. First, the receiver has to estimate symbol timing and frequency offset, using special training symbols in the preamble. Then it can do an FFT for every symbol to recover the QAM values of all sub-carriers. The training symbols and pilot sub-carriers are used to correct the channel response as well as remaining phase drift. The QAM values are then demapped into binary values, after which a Viterbi decoder can decode the information bits.
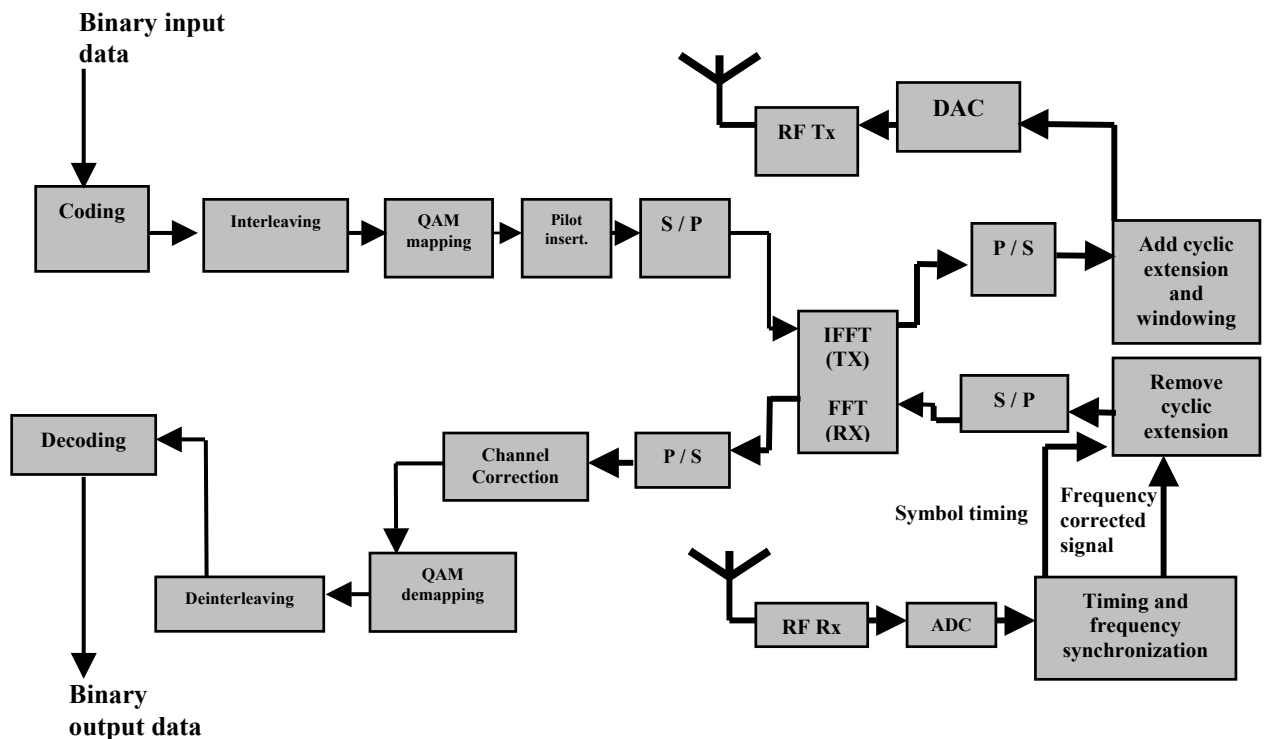
Figure 2.3 Basic OFDM communication system

2.1. OFDM Signal

2.1.1. Generation of Sub-carriers Using IFFT

As illustrated in Figure 2.4, an OFDM signal consists of a sum of sub-carriers that are modulated by using quadrature amplitude modulation (QAM) or phase shift keying (PSK). In its most general form, the low-past equivalent OFDM signal can be written as a set of modulated carriers transmitted in parallel, as follows [5]:

$$s(t) = \sum_{n=-\infty}^{\infty} \left[\sum_{k=0}^{N-1} C_{n,k} g_k (t - nT_s) \right] \qquad (2.1)$$

with

$$g_k(t) = \begin{bmatrix} e^{j2\pi f_k t} & t \in [0, T_s) \\ 0 & \text{otherwise} \end{bmatrix} \qquad (2.2)$$

and

$$f_k = f_0 + \frac{k}{T_s} \quad , k = 0.....N\text{-}1 \qquad (2.3)$$

where

- $C_{n,k}$ is the QAM modulated data (symbol transmitted on the kth sub-carrier in the nth signaling interval, each of duration is T_s).
- N is the number of OFDM sub-carriers
- f_k is the kth sub-carrier frequency, with f_0 being the lowest frequency to be used.

The nth OFDM frame can be defined as the transmitted signal for the nth signaling interval of duration equal to one symbol period T_s, and denote it by $F_n(t)$ in Equation (2.1) instead of the term in parenthesis which corresponds to the nth OFDM frame, the relation can be rewritten as

$$s(t) = \sum_{n=-\infty}^{\infty} F_n(t) \qquad (2.4)$$

and thus, $F_n(t)$ corresponds to the set of symbols $C_{n,k}$, k = 0…N-1, each transmitted on the corresponding sub-carriers f_k.

Demodulation is based on the orthogonality of the carriers $g_k(t)$, namely:

$$\int_R g_k(t)g_l(t)dt = T_s.\delta(k-l) \qquad (2.5)$$

where δ is kronecker delta function and R indicates data rate.

Therefore, by assuming no interference and noise in the channel, the demodulator will produce transmitted symbol as:

$$C_{n,k} = \frac{1}{T_s}.\int_{nT_s}^{(n+1)T_s} s(t)g_k^*(t)dt \qquad (2.6)$$

The block diagram of an OFDM modulator is given in Figure 2.4, while the demodulator is shown in Figure 2.5, where, for simplicity, the impulse response of communications systems has been ignored.
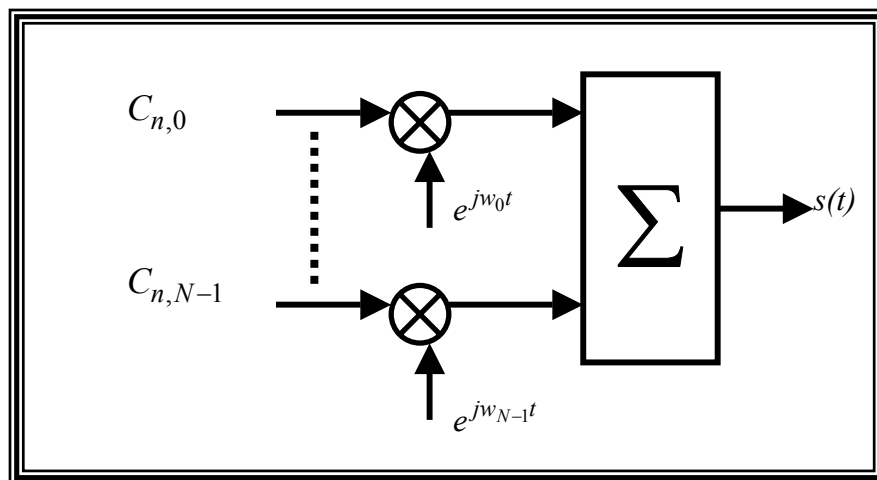
Figure 2.4 OFDM modulator
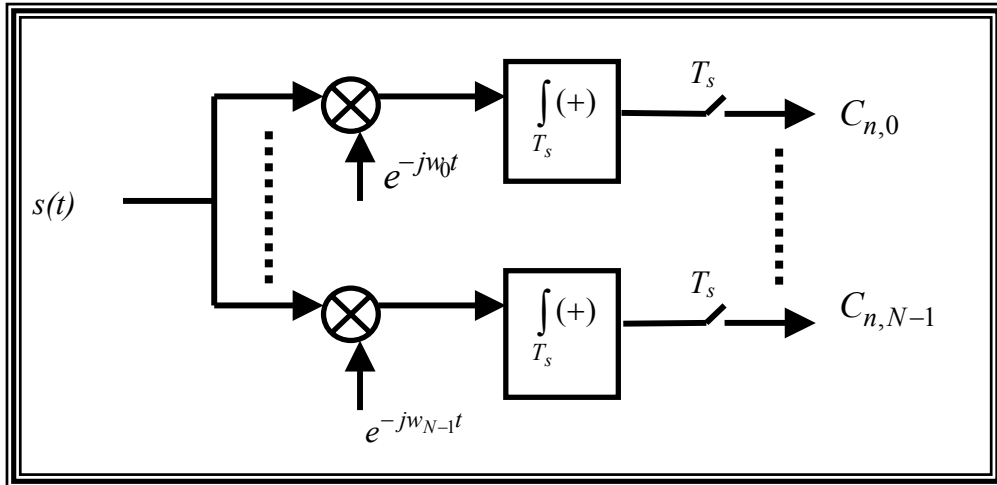
Figure 2.5 OFDM Demodulator

As an example, Figure 2.6 shows four sub-carriers from one OFDM signal in time domain. In this example, all sub-carriers have the same phase and amplitude. But in practice the amplitudes and phases may be modulated differently for each sub-carrier. Each sub-carrier has exactly an integer number of cycles in the interval T_s and the number of cycles between adjacent sub-carriers differs by exactly one. This property accounts for the orthogonality between the sub-carriers.

———— Sub-carrier1
— — — Sub-carrier2
············ Sub-carrier3
— · — Sub-carrier4

Time domain ⟶
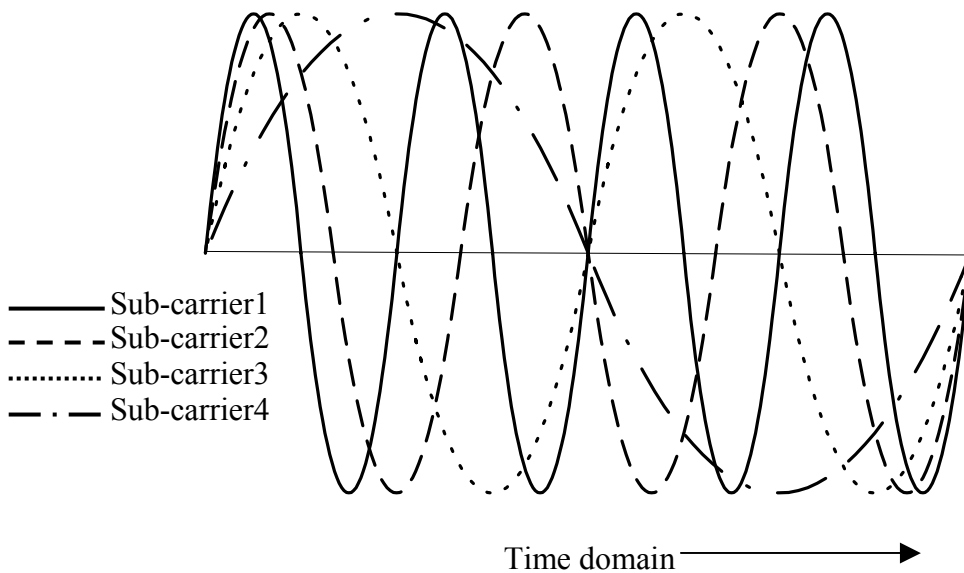
Figure 2.6 Example of four sub-carriers within one OFDM symbol

The orthogonality of the different OFDM sub-carriers can also be demonstrated in another way. According to Equations (2.1), (2.2) and (2.3), each OFDM symbol contains sub-carriers that are nonzero over a T_s-second interval. Hence, the spectrum of a single symbol is a convolution of a group of dirac pulses located at the sub-carrier frequencies with the spectrum of the square pulse that is one for a T_s-second period and zero otherwise. The amplitude spectrum of the square pulse is equal to sinc($\pi f T_s$), which has zeros for all frequencies f that are an integer multiple of $\frac{1}{T_s}$. This effect is shown in Figure 2.2, which shows the overlapping sinc spectra of individual sub-carriers. At the maximum of each sub-carrier spectrum, all other sub-carrier spectra are zero. Because an OFDM receiver essentially calculates the spectrum values at those points that correspond to the maximum of individual sub-carriers, it can demodulate each sub-carrier free from any interference from the other sub-carriers if synchronization is perfect and no channel distortion and noise exist.

The complex base-band OFDM signal as defined by Equation (2.4) is in fact nothing more than the inverse Fourier transform of N QAM input symbols. The time discrete equivalent is the inverse discrete Fourier (IDFT), which is given by Equation (2.8). By sampling the low pass equivalent signal of Equation (2.1) and Equation (2.4) at a rate N times higher than the symbol rate $\frac{1}{T_s}$, and assuming $f_0 = 0$ (that is the carrier frequency is equal to the lowest sub-carrier frequency), the OFDM frame can be expressed as:

$$F_n(m) = \sum_{k=0}^{N-1} C_{n,k} g_k(t - nT_s)\Bigg|_{t=\left(n+\frac{m}{N}\right)T_s}, m = 0....N-1 \qquad (2.7)$$

which yields

$$F_n(m) = e^{j2\pi f_0 T_s \frac{m}{N}}\left[\sum_{k=0}^{N-1} C_{n,k} e^{j2\pi k \frac{m}{N}}\right] = N.IDFT\{C_{n,k}\} \qquad (2.8)$$

In practice, this transform can be implemented very efficiently by the inverse fast fourier transform (IFFT).

To point out the difference between OFDM and (Frequency Division Multiplexing) FDM, the power spectrum density for the two systems with binary phase shift keying (BPSK) data on all carriers is considered in Figure 2.7, illustrating the two spectra indicating the occupied bandwidth W as function of the number of carriers N. Note that here R indicates data rate.
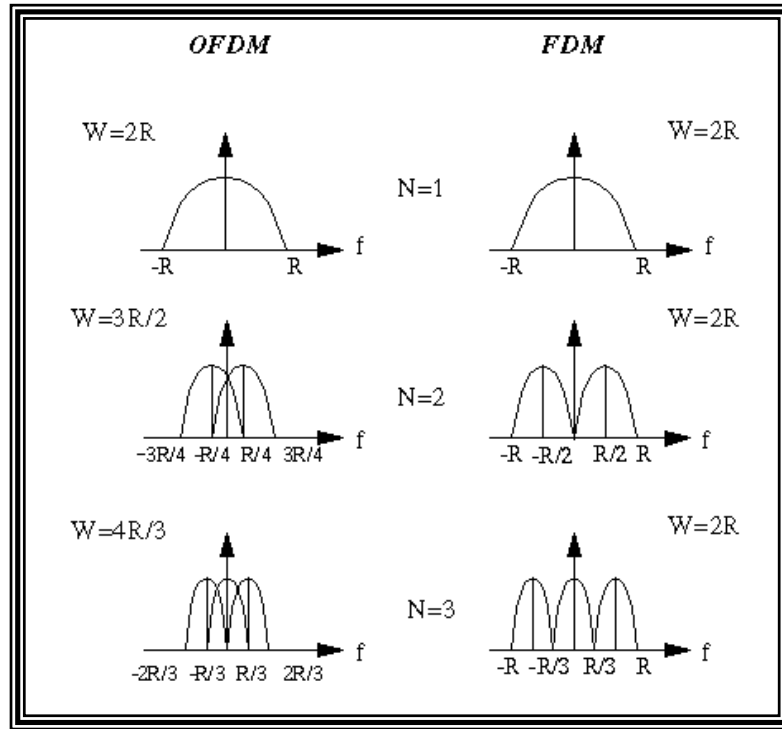
Figure 2.7 OFDM versus FDM power spectrum density

From Figure 2.7, one can see that the OFDM signal requires less bandwidth as the number of carriers is increased, and in the limit we have:

$$\lim_{N \to \infty} W = \lim_{N \to \infty} \frac{N+1}{N}.R = R = \frac{N}{T_S} \tag{2.9}$$

This is possible since there is spectral overlapping, which is resolved making use of the orthogonality of the sub-carriers.

By performing the sampling as indicated, the OFDM signal is subject to no loss since the two-sided bandwidth of the low-pass equivalent OFDM signal (neglecting side-lobes due to the outer sub-carriers) is $W = N/T_S$. Then, the sampling rate of N/T_S

is exactly the corresponding Nyquist rate, and hence there will be no frequency domain aliasing.

2.1.2. Guard Time and Cyclic Extension

One of the most important reasons to use OFDM is the efficient way to deal with interference due to multi-path. By dividing the input data-stream in N sub-carriers, the symbol duration is made N times smaller, which also reduces the relative multi-path delay spread, relative to the symbol time, by the same factor. An OFDM signal retains its sub-carrier orthogonality property when transmitted through a non-dispersive channel. Most channels of interest, however, contain significant time and/or frequency dispersion. These impairments introduce inter symbol interference (ISI) and inter carrier interference (ICI), and can destroy the orthogonality of the sub-carriers. A major advantage of OFDM, mentioned before, is the ability to enhance the basic signal in ways that overcome channel impairments.

There are two aspects of the multi-path channel that need attention:

- The delay spread, which produces an impulse response extended in time
- The arrival at the receiver of delayed versions of the transmitted signal causing interference manifests itself as frequency-selective fading.

To protect against time dispersions including multi-path, a guard interval equal to the length of the channel impulse response is introduced between successive OFDM symbols. The guard interval is commonly implemented by the cyclic extension of the IFFT output [36]. The problem of ICI is illustrated in Figure 2.8. In this figure, a sub-carrier1 and a delayed sub-carrier2 are shown. When an OFDM receiver tries to demodulate the first sub-carrier, it will encounter some interference from the second sub-carrier, because within the FFT interval, there is no integer number of cycle difference between sub-carrier 1 and 2. At the same time, there will be cross talk from the first to the second sub-carrier for the same reason.

Part of sub-carrier #2 causing
ICI on sub-carrier #1

Sub-carrier #1

Delayed sub-carrier #2

Guard time   FFT integration time = 1/Carrier spacing
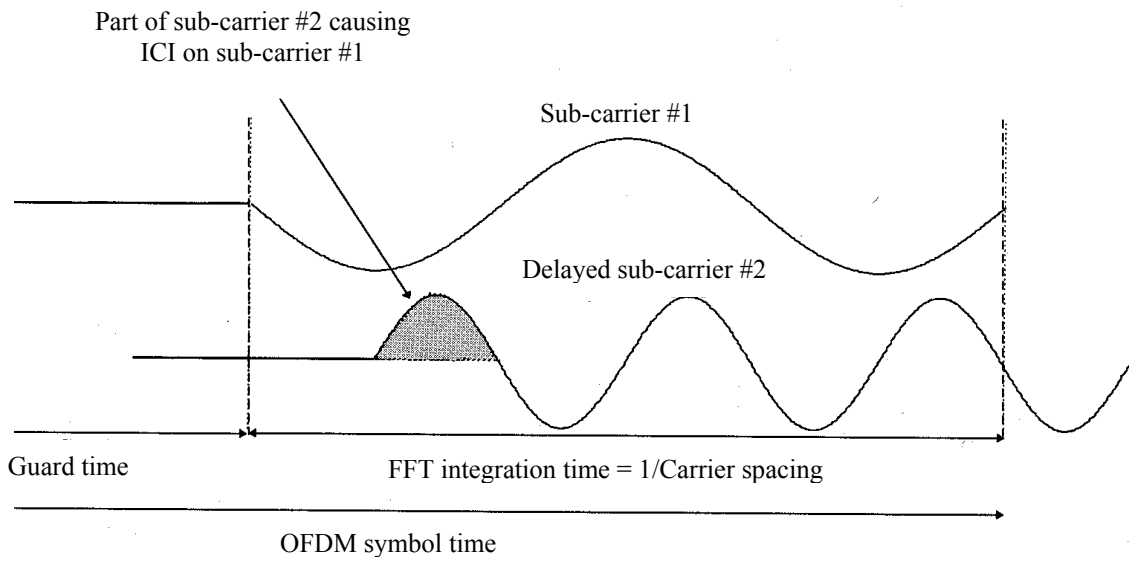
OFDM symbol time

Figure 2.8 Effect of multi-path with zero signal in the guard time

To eliminate ICI, the OFDM symbol is cyclically extended in the guard time, as shown in Figure 2.9 [36]. This ensures that delayed replicas of the OFDM symbol always have an integer number of cycles within the FFT interval, as long as the delay is smaller than the guard time. As a result, multi-path signals with delays smaller than the guard time don't cause ICI.

Sub-carrier #1

Sub-carrier #2

Sub-carrier #3

Guard time / cyclic prefix   FFT integration time = 1/carrier spacing
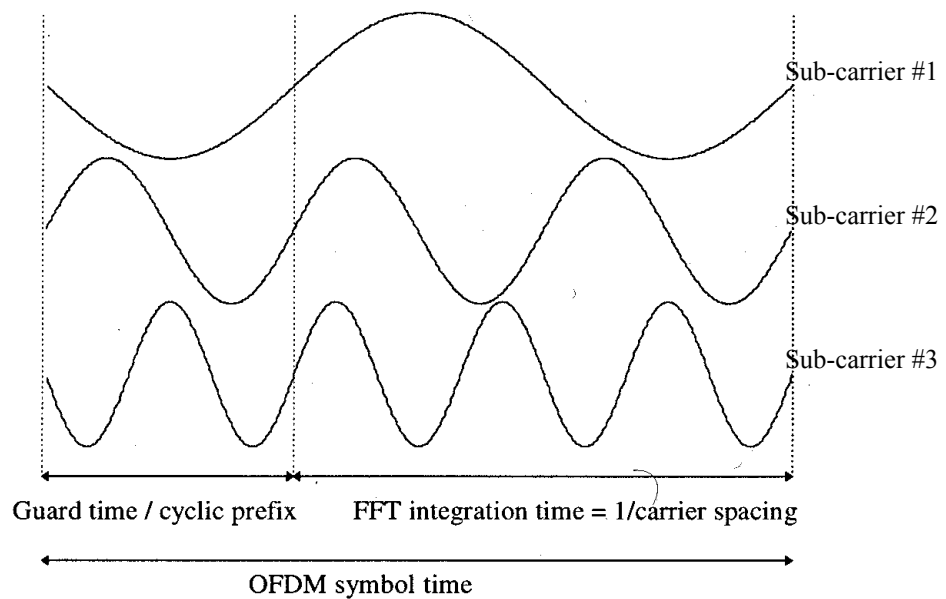
OFDM symbol time

Figure 2.9 OFDM symbol with cyclic extension

14

Figure 2.10 illustrates how multi-path affects OFDM symbol [36]. This figure shows received signals for the channel as solid lines; the dotted curve is a delayed replica of the solid curve. Three separate sub-carriers are shown during three symbol intervals. In reality, an OFDM receiver only sees the sum of all these signals, but showing the separate components facilitates to see clearly what the effects of multi-path are. From the figure, it can be seen that the OFDM sub-carriers are BPSK modulated, which means that there can be 180-degree phase jumps at the symbol boundaries. For the dotted curve, these phase jumps occur at a certain delay after the first path. In this particular example, this multi-path delay is smaller than the guard time, which means there are no phase transitions during the FFT interval. Hence, an OFDM receiver "sees" the sum of pure sine waves with some phase offsets. This summation does not destroy the orthogonality between the sub-carriers; it only introduces a different phase shift for each sub-carrier. The orthogonality will be lost if the multi-path delay becomes larger than the guard time. In that case, the phase transitions of the delayed path fall within the FFT interval of the receiver. The summation of the sine waves of the first path added with the phase modulated waves of the delayed path no longer gives a set of orthogonal pure sine waves, resulting in a certain level of interference.
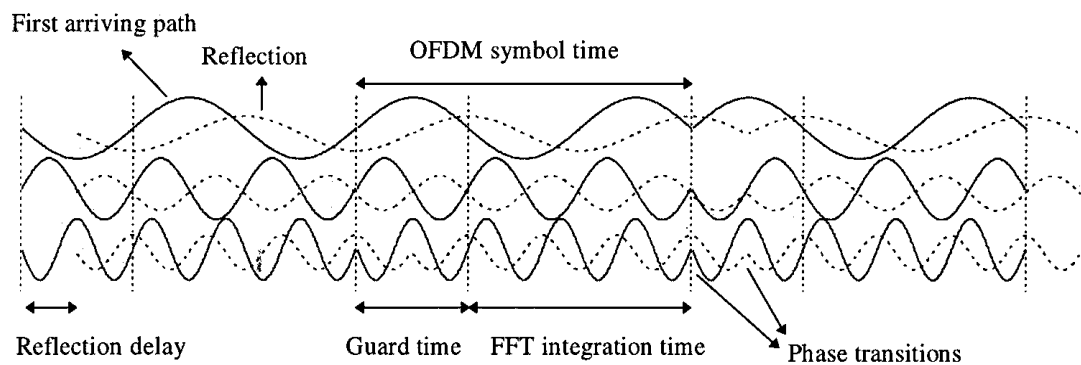
Figure 2.10 Example of an OFDM signal with three sub-carriers in a channel; the dashed line represents a delayed multi-path component.

The ratio of the guard interval to useful symbol duration is application dependent. Since the insertion of guard interval will reduce data throughput, the guard (cyclic prefix) interval T_{guard} is usually less than $T/4$ (see Table D.6. T_{guard} is represented by T_{CP}). T represents here the FFT integration time.

When a signal $s(t)$, which is sent over a channel with impulse response $h(t)$, the received signal is given by the convolution:

$$r(t) = h(t) * s(t) \qquad (2.10)$$

and if the channel is not ideal, i.e. $h(t) = \delta(t)$, there will be inter symbol interference (ISI). It is convenient to view the OFDM signal in terms of data frames, so we can anticipate that the channel will produce ISI within the frame, and will also produce inter frame interference (IFI) among adjacent frames [5]. Considering the discrete-time equivalent signal and the channel h_i, $i = 0,.....,L$, with L being the delay spread of the channel, equation (2.10) becomes

$$r_m = \sum_{i=0}^{L} h_i . s_{m-i} = h_0 . s_m + \underbrace{\sum_{i=1}^{L} h_i . s_{m-i}}_{\text{ISI}} \qquad (2.11)$$

Figure 2.11 shows this convolution sum for the particular case of L=2. Here, $s_{n,N-1}$ represents the OFDM signal carried by $(N-1)^{\text{th}}$ sub-carrier in the nth frame. From this graphical representation it can be seen that the introduction of a guard interval of length equal to the delay spread L of the channel between two adjacent frames will "absorb" the channel delay and hence remove IFI.
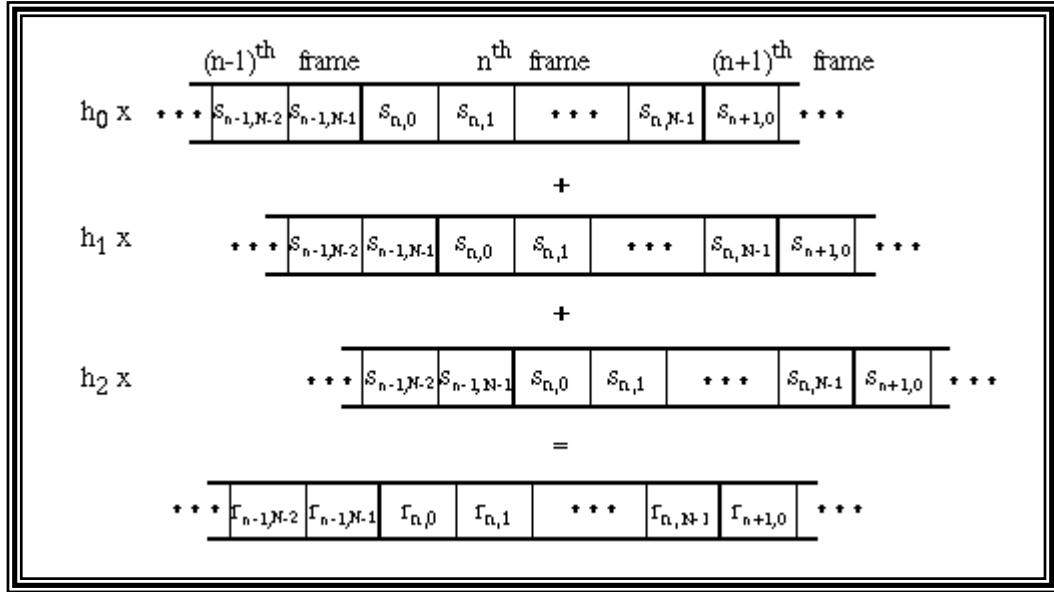
Figure 2.11 Inter Frame Interference in OFDM systems.

This may be accomplished by inserting L leading zeros in each frame at the transmitter and removing them at the receiver. However, in order to also eliminate ISI from within the frame, it is better to use a cyclic prefix instead of an all zero guard interval. In this case, after dumping the prefix at the receiver, one would get the periodic (cyclic) convolution of the transmitted data frame and the channel. The cyclically extended frame can then be written as [5]

$$F_n^t(m) = \begin{bmatrix} F_n(N+m), & m = -L....-1 \\ F_n(m), & m = 0...N-1 \end{bmatrix} \tag{2.12}$$

where

$$F_n(m) = \sum_{k=0}^{N-1} C_{n,k} e^{j2\pi k \frac{m}{N}}, m = 0.....N-1 \tag{2.13}$$

After discarding the prefix, the received frame becomes

$$\hat{F}_n(m) = \sum_{i=0}^{N-1} F_n(m-i)_N \cdot h_i \qquad (2.14)$$

where $(m-i)_N$ represents the modulo N subtraction. After DFT demodulation we get

$$\hat{C}_{n,k} = \frac{1}{N} \cdot \sum_{m=0}^{N-1} \hat{F}_n(m) e^{-j2\pi k \frac{m}{N}} = C_{n,k} \cdot H_k \qquad (2.15)$$

where $k = 0........N-1$ and H_k is the channel's transfer function at the sub-carrier frequency f_k from Equation (2.3). Therefore, by using a cyclic prefix, the effect of the channel is transformed into a complex multiplication of the data symbols with the channel coefficients H_k, and all ISI and IFI is removed.

2.1.3. Useful Symbol Duration

The useful symbol duration T (FFT integration period) affects the carrier spacing and coding latency. To maintain the data throughput, longer useful symbol duration results in an increase of the number of carriers and the size of FFT (assuming that the signal constellation is fixed). The number of carriers corresponds to the number of complex points being processed in FFT. In practice the carrier offset and phase stability may affect spacing between carriers.

2.1.4. Number of Carriers

"Less than one quarter" rule of thumb and the use of an FFT algorithm in turn drive the selection of the number of carriers, and hence the transform size for a particular application [6]. The first-order design of an OFDM scheme for an application

using this approach begins by considering the channel delay-spread, which dictates the duration of the guard interval. The number of sub-carriers that both maintains the information rate needed for the application (also satisfies the channel bandwidth constraints) and meets the "less than 1/4 symbol" rule of thumb can be determined. The carriers are spaced by the reciprocal of the useful symbol duration. The number of carriers corresponds to the number of complex points being processed in FFT.

2.2. Properties of OFDM

After introducing the OFDM signaling scheme, we can list its major advantages and disadvantages as follows:

- OFDM makes efficient use of the spectrum by allowing overlap.
- By dividing the channel into narrowband flat fading sub-channels, OFDM is more resistant to frequency selective fading than single carrier systems are.
- ISI and IFI are eliminated through via cyclic prefix.
- Using adequate channel coding and interleaving, one can recover symbols lost due to the frequency selectivity of the channel
- Channel is simpler than using adaptive equalization techniques with single carrier systems.
- OFDM is computationally efficient by using FFT techniques to implement the modulation and demodulation functions. Also, for multiple communication channels, as is the case in digital audio broadcasting (DAB) systems, partial FFT algorithms can be used in order to implement program selection and decimation.

The disadvantages can be listed as follows:

- The OFDM signal has a noise like amplitude with a very large dynamic range, therefore it requires RF power amplifiers with a high peak to average power ratio.

- OFDM is more sensitive to carrier frequency offset and phase offsets than single carrier systems are.

2.3. Choice of OFDM Parameters

The choice of various OFDM parameters is a tradeoff between various, often conflicting requirements. Usually, there are three main requirements as follows:

- Bandwidth
- Bit rate
- Delay spread

The delay spread directly dictates the guard time. As a rule, the guard time should be about two to four times the root-mean-squared delay spread (see chapter 2.1.2). This value depends on the type of coding and QAM modulation. Higher order QAM (like 64-QAM) is more sensitive to ICI and ISI; while heavier coding obviously reduces the sensitivity to such interference.

Since the guard time has been set, the symbol duration can be fixed. To minimize the signal-to-noise ratio (SNR) loss caused by the guard time, it is desirable to have the symbol duration much larger than the guard time. It cannot be arbitrarily large, however, because larger symbol duration means more sub-carriers with a smaller sub-carrier spacing, a larger implementation complexity, and more sensitivity phase offset and frequency offset [11], as well as an increased peak-to-average power ratio.

After the symbol duration and guard time are fixed, the number of sub-carriers can be determined by inverse of the useful symbol duration (symbol duration-guard time). Alternatively, the number of sub-carriers may be also determined by the required bit rate divided by the bit rate per sub-carrier. The bit rate per sub-carrier is defined by the modulation type (e.g. 64-QAM), coding rate and symbol rate. An additional

20

requirement that can affect the chosen parameters is the demand for an integer number of samples both within the FFT/IFFT interval and in the symbol interval.

To see the relation between these three requirements mentioned above, let's assume we want to design a system with the following requirements:

- Bit rate: 24 Mbps
- Tolerable delay spread: 200 ns
- Bandwidth: <16 MHz

First, we can set the guard time to a safe value using the given value for the delay-spread requirement: Delay spread should be smaller than guard time (see 2.1.2). Let's take the guard time 800 ns, which is four times delay-spread. By choosing the OFDM symbol duration 5 times (4.0 µs = guard time (0.8 µs) + useful symbol part duration (3.2 µs)) the guard time according to ETSI HiperLan/2 standard (see Table D.6), we are now ready to find the number of sub-carriers and sub-carrier spacing. The sub-carrier spacing is the inverse of 4.0 – 0.8 = 3.2 µs, which gives 312.5 kHz. To determine the number of sub-carriers needed, we can look at the ratio of the required bit rate and the OFDM symbol rate. To achieve 24 Mbps, each OFDM symbol has to carry 96 bits of information (96/4.0 µs = 24 Mbps). To do this, there are several options. One is to use 16-QAM together with ½ coding rate to get 2 bits per carrier in a symbol. In this case, 48 sub-carriers are needed to get the required 96 bits per symbol. Another option is to use QPSK with rate ¾ coding rate, which gives 1.5 bits per sub-carrier in a symbol. In this case, 64 sub-carriers are needed to reach the 96 bits per symbol. However, 64 sub-carriers means a bandwidth of 64 * 312.5 kHz = 20 MHz, which is larger than the target bandwidth. To achieve a bandwidth smaller than 16 MHz, the number of sub-carriers needed to be equal to or smaller than 50. Hence, the first option with 48 sub-carriers and 16-QAM fulfills all the requirements.

In this section, we reviewed the OFDM, compared it to FDM in terms of advantages and drawbacks. We saw how the basic OFDM signal is formed using IFFT and adding a cyclic extension. We explained how OFDM avoids the problem of inter-symbol interference by transmitting a number of narrowband sub-carriers together with using a guard time. We gave an example to a basic OFDM communication system and summarized the functionality of its sub-blocks. Choice of OFDM parameters for communication system was explained with an example. We mentioned an important

term for OFDM, i.e. orthogonality. After this introduction, we will see the synchronization issues that should be taken care of in OFDM receivers in the next chapter.

3.   SYNCHRONIZATION

One of the arguments against OFDM is that it is highly sensitive to synchronization errors, in particular, to frequency errors. Before an OFDM receiver can demodulate the sub-carriers, it has to perform at least two synchronization tasks:

- Symbol (frame) timing synchronization
- Carrier frequency synchronization (carrier frequency offset) and sampling frequency synchronization (clock offset)

An OFDM receiver first, has to find out where the symbol boundaries are and what the optimal timing instants are to minimize the effects of inter-carrier interference (ICI) and inter-symbol interference (ISI). Symbol (Frame) timing synchronization means finding an estimate where the symbol starts. Second, it has to estimate and correct for the carrier frequency offset of the received signal, because any offset introduces ICI. Notice that these two synchronization tasks are not the only training required in an OFDM receiver. For coherent receivers, except for the frequency, the carrier phase also needs to be synchronized. Further, a coherent QAM receiver needs to learn the amplitudes and phases of all sub-carriers to find out the decision boundaries for the QAM constellation of each sub-carrier [9, 14, 16, 17, 19].

3.1.   Introduction

In an OFDM link, the sub-carriers are perfectly orthogonal only if transmitter and receiver use exactly the same frequencies. Any frequency offset immediately results in ICI. A related problem is the phase noise; a practical oscillator does not produce a carrier at exactly one frequency, but rather a carrier that is phase modulated by random

phase jitter. As a result, the frequency, which is the time derivative of the phase, is never perfectly constant, thereby causing ICI in an OFDM receiver. For single-carrier systems, phase noise and frequency offsets only give degradation in the received signal-to-noise ratio (SNR) rather than introducing interference. This is the reason that the sensitivity to phase noise and frequency offset are often mentioned as disadvantages of OFDM in respect to single-carrier systems.

3.2.   Symbol Synchronization

3.2.1. Sensitivity To Timing Errors

In OFDM systems, a great deal of attention is given to symbol synchronization. Finding the symbol timing for OFDM systems means finding an estimate of the symbol start point. So the objective is to detect the start point of OFDM symbol. However, by using a cyclic prefix, the timing requirements are relaxed somewhat. There is usually some tolerance for symbol timing errors since a cyclic prefix is used to extend the symbol. A timing offset gives rise to a phase rotation of the sub-carriers. This phase rotation is largest on the edges of the frequency band. If a timing error is small enough to keep the channel impulse response within the cyclic prefix, the orthogonality is maintained. In this case a symbol timing delay can be viewed as a phase shift introduced by the channel. Then the phase rotations can be estimated by a channel estimator. If a time shift is larger than the cyclic prefix and the receiver's FFT interval extends over a symbol boundary, ISI will occur. Hence, OFDM demodulation should be quite insensitive to timing offsets. To achieve the best possible multi-path robustness, however, there exists an optimal timing instant. Any deviation from this timing instant means that the sensitivity to delay spread increases, so the system can handle less delay spread than the value it was designed for. To minimize this loss of robustness, the system should be designed such that the timing error is small compared with the guard interval.
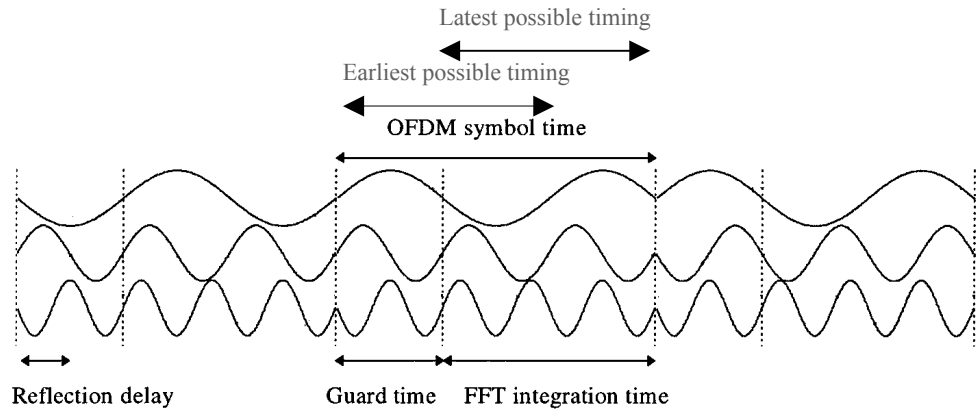
Figure 3.1 Example of an OFDM signal with three sub-carriers, showing the earliest and latest possible symbol timing instants that do not cause ISI or ICI.

An interesting relationship exists between symbol timing and the demodulated sub-carrier phases [20]. Looking at Figure 3.1, it can be seen that as the timing changes, the phases of the sub-carriers change. The relation between the phase, φ_i, of sub-carrier, i, and the timing offset, τ, is given by

$$\varphi_i = 2\pi f_i \tau \qquad (3.1)$$

where, f_i is the frequency of the ith sub-carrier before sampling. For an OFDM system with N sub-carriers and a sub-carrier spacing of $1/T$, a timing delay of one sampling interval of T/N causes a significant phase shift of $2\pi(1-1/N)$ between the first and last sub-carrier. T represents here useful symbol duration. These phase shifts add to any phase shifts that are already present because of multi-path propagation. In a coherent OFDM receiver, channel estimation is performed to estimate these phase shifts for all sub-carriers [9, 14, 16, 19, 21].

3.2.2. Sensitivity To Phase Noise

Carrier phase noise is caused by imperfections in the transmitter and receiver oscillators. Phase noise basically has two effects. First, it introduces a random phase variation that is common to all sub-carriers. If the oscillator line width is much smaller than the OFDM symbol rate, which is usually the case, then the common phase error is strongly correlated from symbol to symbol; so tracking techniques or differential detection can be used to minimize the effects of this common phase error. The second and more disturbing effect of phase noise is that it introduces ICI, because the sub-carriers are no longer spaced at exactly $1/T$ in the frequency domain. The amount of ICI is calculated and translated into a degradation in SNR that is given as [11]

$$D_{phase} \cong \frac{11}{6\ln 10}\left(4\pi N \frac{\beta}{W}\right)\frac{E_s}{N_o} \tag{3.2}$$

where, β is the -3 dB one-sided bandwidth of the power density spectrum of the carrier, W is the bandwidth and E_s / N_o is the symbol energy per noise spectral density. Note that the degradation increases with the number of sub-carriers and the phase noise degradation is proportional to $\beta.T$, which is the ratio of the line-width and sub-carrier spacing $1/T$.

3.3. Frequency Synchronization

3.3.1. Sampling Frequency Synchronization

The received continuous-time signal is sampled at instants determined by the receiver clock. There are two types of methods of dealing with the mismatch in sampling frequency. In synchronized-sampling systems a timing algorithm controls a voltage-controlled crystal oscillator in order to align the receiver clock with the transmitter clock. The other method is non-synchronized sampling, where the sampling rate remains fixed, requiring post-processing in the digital domain. The effect of a clock frequency offset is that the useful signal component is rotated, attenuated and, also ICI is introduced. The bit-error rate performance of a non-synchronized sampling systems are much more sensitive to a frequency offset, compared with a synchronized-sampling system [11]. For non-synchronized sampling systems, it was shown that the degradation (in dB) due to a frequency sampling offset depends on the square of the carrier index and the square of relative frequency offset.

3.3.2. Carrier Frequency Synchronization

Frequency offsets are created by differences in oscillators in transmitter and receiver, Doppler shifts or phase noise introduced by non-linear channels. There are two destructive effects caused by a carrier frequency offset in OFDM systems:

- One is the reduction of signal amplitude (the sinc functions are shifted and no longer sampled at the peak) and the other is the introduction of ICI from the other carriers, as illustrated in Figure 3.2 and Figure 3.3.
- The latter is caused by the loss of orthogonality between the sub-channels. Pollet analytically evaluates the degradation of the BER caused by the presence of carrier frequency offset and carrier phase noise for an AWGN channel [11]. It is found that a multi-carrier system is much more sensitive than a single-carrier system. If we denote the normalized relative frequency offset, by the sub-carrier spacing with $\Delta f = \dfrac{\Delta F}{W / N}$ (ΔF is the frequency offset and N the number of sub-carriers), the degradation D in SNR (in dB) can then be approximated by

$$D\,(\text{dB}) \approx \frac{10}{3\ln 10}(\pi \Delta f)^2 \frac{E_s}{N_o} = \frac{10}{3\ln 10}\left(\pi\,\frac{N \cdot \Delta F}{W}\right)^2 \frac{E_s}{N_o} \quad (3.3)$$

Note that the degradation (in dB) increases with the square of the number of sub-carriers, if ΔF and W are fixed.

Moose derives the signal-to-interference-ratio (SIR) on a fading and dispersive channel [12]. The SIR is defined as the ratio of the power of the useful signal to the power of the interference signal (ICI and additive noise).
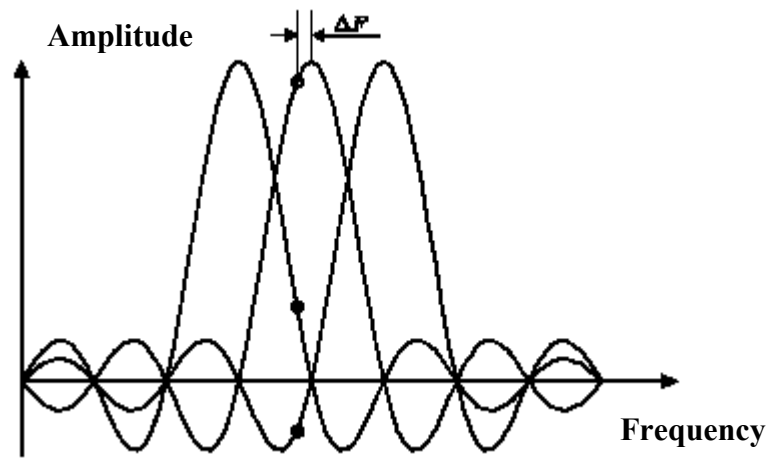
Figure 3.2 Effects of a frequency offset ΔF: reduction in signal amplitude (o) and inter-carrier interference (•)

$$\Delta f_c = \frac{1}{NT_s} \quad W = \frac{1}{T_s} \Rightarrow \Delta f_c = \frac{W}{N}$$

W: bandwidth
N: # of sub-carriers
T$_S$: sample period
Δfc: sub-carrier spacing

RF carrier (or DC)

Δfc

W
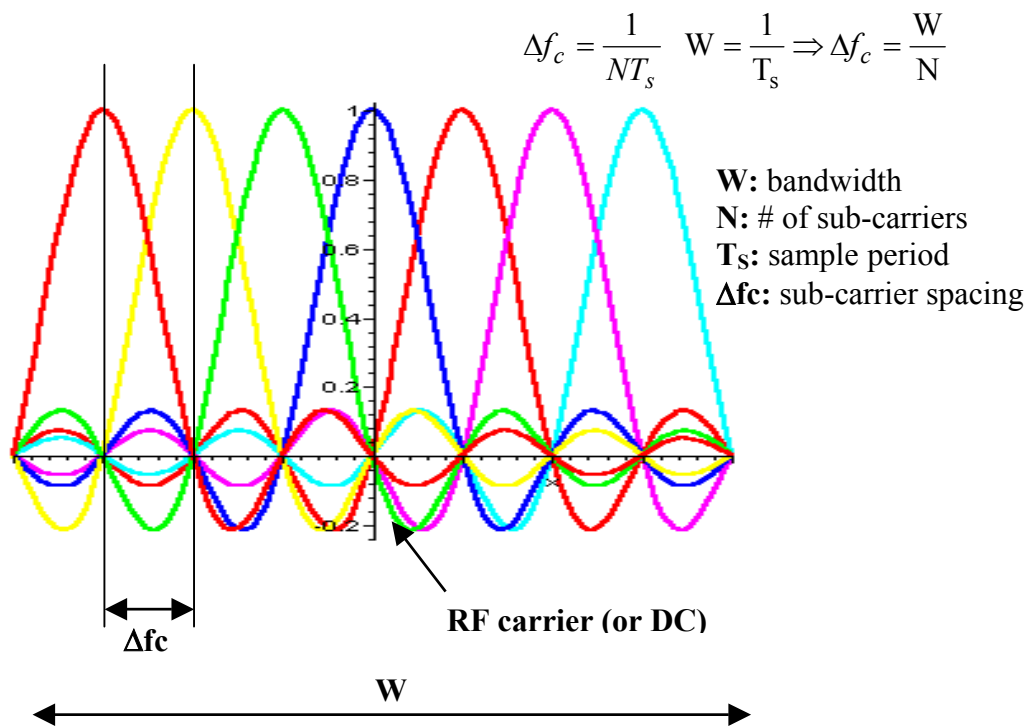
Figure 3.3 Sub-carrier spacing

He assumed that all channel attenuations h_k have the same power, $E\{|h_k^2|\}$. An upper bound on the degradation is [12]

$$D(dB) \le 10\log_{10}\left(\frac{1 + 0.5947\dfrac{E_s}{N_0}\sin^2 \pi\Delta f}{\sin c^2 \Delta f}\right) \qquad (3.4)$$

where $\text{sinc}x \equiv (\sin \pi x)/(\pi x)$. The factor 0.5947 is found from a lower bound of the summation of all interfering sub-carriers. In Figure 3.4 the degradation is plotted as a function of the normalized frequency offset Δf, i.e. relative to the sub-carrier spacing [12].
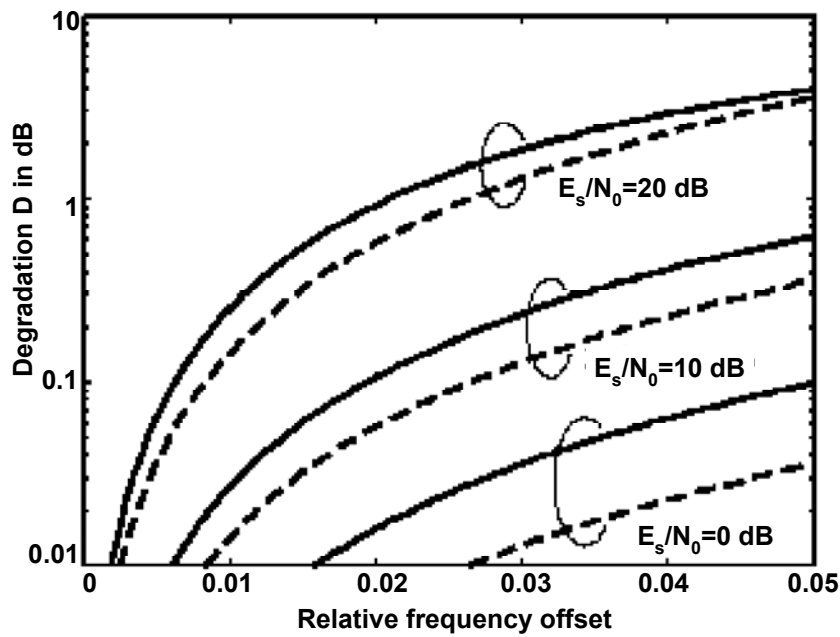
Figure 3.4 Degradation in SNR due to a frequency offset (normalized to the sub-carrier spacing). Analytical expression for AWGN (dashed) and fading channels (solid).

3.4. Synchronization Techniques

3.4.1. Synchronization Using The Cyclic Extension

Because of the cyclic prefix, the first T_G (guard time) seconds part of each OFDM symbol is identical to the last part. This property can be exploited for both timing and frequency synchronization by using a synchronization system like depicted in Figure 3.5. Basically, this device correlates a T_G long part of the signal with a part that is T seconds delayed [18, 19]. The correlator output can be written as

$$x(t) = \int_0^{T_G} r(t-\tau)r(t-\tau-T)d\tau \qquad (3.5)$$
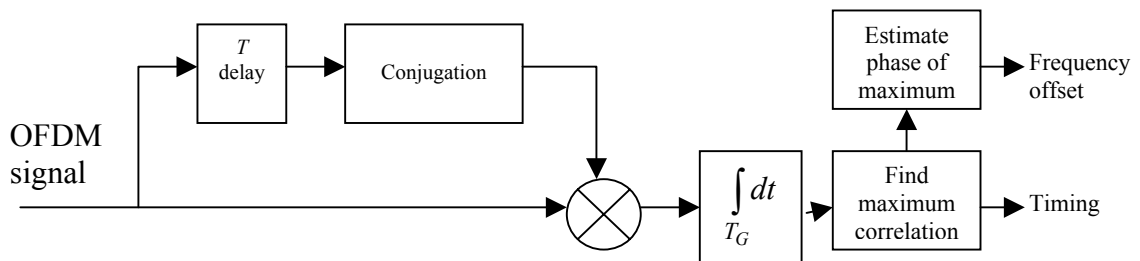
Figure 3.5 Synchronization using the cyclic prefix

Two examples of the correlation output are shown in Figure 3.6 and Figure 3.7 for eight OFDM symbols with 192 and 48 sub-carriers, respectively [19, 36]. These figures illustrate a few interesting characteristics of the cyclic extension correlation method. First, both figures clearly show eight peaks for the eight different symbols but the peak amplitudes show a significant variation. The reason for this is that although the average power for a T seconds interval of each OFDM symbol is constant, the power in the guard time can substantially vary from this average power level. Another effect is the level of the undesired correlation side-lobes between the main correlation peaks. These side-lobes reflect the correlation between two pieces of the OFDM signal that belong partly or totally to two different OFDM symbols. Because different OFDM symbols contain independent data values, the correlation output is a random variable, which may reach a value that is larger than the desired correlation peak. The standard deviation of the random correlation magnitude is related to the number of independent samples over which the correlation is performed. The larger the number of independent samples means the smaller the standard deviation. In the extreme case, where the correlation is performed over only one sample, the output magnitude is proportional to the signal power, and there is no distinct correlation peak in this case. In the other extreme case, where the correlation is performed over a very large number of samples, the ratio of side-lobes-to-peak amplitude will go to zero. Because the number of independent samples is proportional to the number of sub-carriers, the cyclic extension correlation technique is only effective when a large number of sub-carriers are used, preferably more than 100. An exception to this is the case where instead of random data symbols, specially designed training symbols are used [13]. In this case, the integration can be done over the entire symbol duration instead of the guard time only. The level of undesired correlation side-lobes could be minimized by a proper selection of the training symbols.
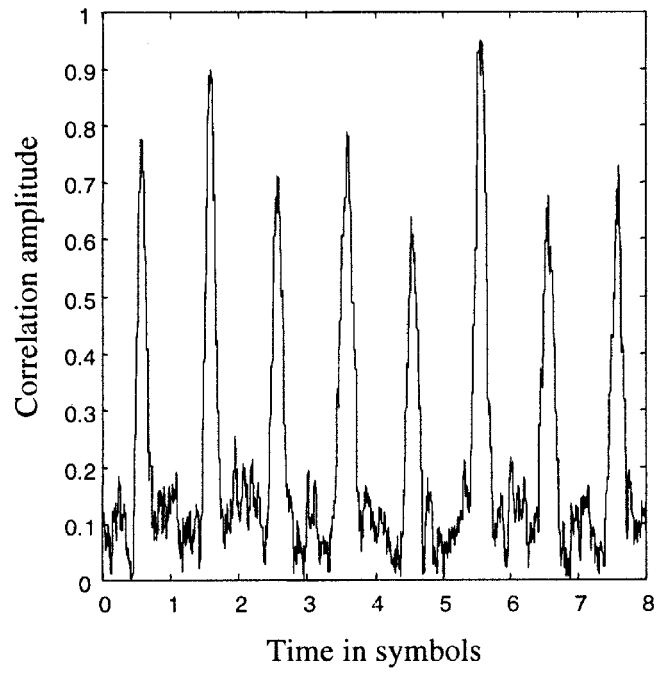
Figure 3.6 Example of correlation output amplitude for eight OFDM symbol with 192
sub-carriers and a 20% guard time
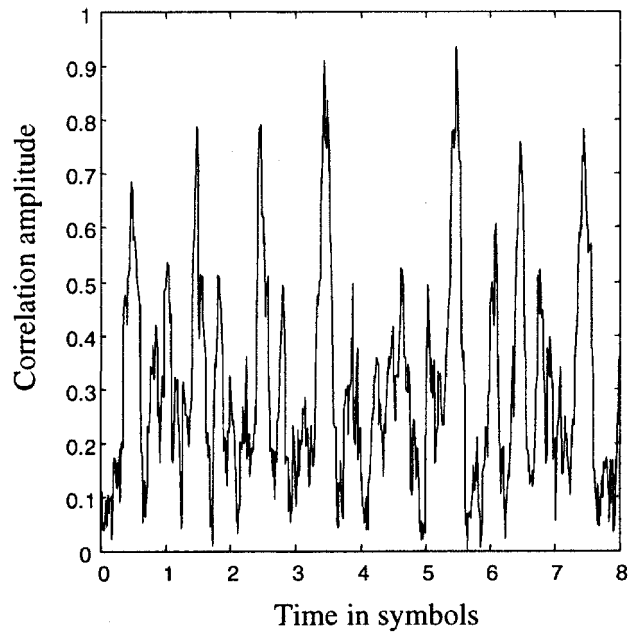
Figure 3.7 Example of correlation output amplitude for eight OFDM symbols with 48
sub carriers and a 20%guard time

We know that the undesired correlation side-lobes only create a problem for symbol timing. But they do not play a role for frequency offset estimation. Once symbol timing is known, the cyclic extension correlation output can be used to estimate the frequency offset. The phase of the correlation output is equal to the phase drift between samples that are T seconds apart. Hence, the frequency offset can simply be found as the correlation phase divided by $2\pi T$. This method works up to a maximum absolute frequency offset of half the sub-carrier spacing. To increase this maximum range, shorter symbols can be used, or special training symbols with different PN sequences on odd and even sub-carriers frequencies to identify a frequency offset of an integer number of sub-carrier spacing [9].

The noise performance of the frequency offset estimator is now determined for an input signal $r(t)$ that consists of an OFDM signal $s(t)$ with power P and additive Gaussian noise $n(t)$ with a one – sided noise power spectral density of N_0 within the bandwidth of the OFDM signal:

$$r(t) = s(t) + n(t) \tag{3.6}$$

The frequency-offset estimator multiplies the signal by a delayed and conjugated version of the input to produce an intermediate signal $y(t)$ given by [9, 36]

$$y(t) = r(t)r^*(t-T) = \left\|s(t)^2\right\|\exp(j\varphi) + n(t)s^*(t-T) + n^*(t-T)s(t) + n(t)n^*(t-T)$$

$$\tag{3.7}$$

The first term in the right – hand side of Equation (3.7) is the desired output component with a phase equal to the phase drift over a T – second interval and a power equal to the squared signal power. The next two terms are products of the signal and the Gaussian noise. Because the signal and noise are uncorrelated and because noise samples separated by T seconds are uncorrelated, the power of the two terms is equal to twice the product of signal power and noise power. Finally, the power of the last term of Equation (3.7) is equal to the squared noise power. If the input SNR is much larger than one, the power of the squared noise component becomes negligible compared with the power of the other two noise terms. For practical OFDM systems, the minimum input SNR is about 6 dB, so the signal power is four times smaller than the power of the two signal – noise product terms.

The frequency offset is estimated by averaging $y(t)$ over an interval equal to the guard time T_G and then the phase of $y(t)$ is estimated. Because the desired output component of Equation (3.7) is a constant vector, averaging reduces the noise that is added to this vector. Assuming that the squared noise component may be neglected, the output SNR is approximated as [36, 37]

$$SNR_0 \cong \frac{P^2}{2PN_0/T_G} = \frac{PT_G}{2N_0} \tag{3.8}$$

Figure 3.8 shows a vector representation of the phase estimation, where the noise is divided into in phase and quadrature components, both having a noise power of N_0/T_G.
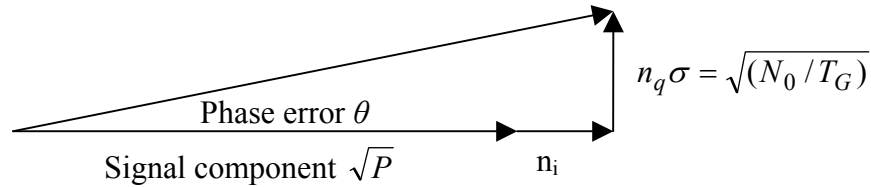
Figure 3.8 Vector representation of phase drift estimation

The phase error θ is given by Equation (3.9), where the approximation has been made that n_i and n_q are small compared with the signal amplitude $\sqrt{P}$ [37].

$$\theta = \tan^{-1}(\frac{n_q}{\sqrt{P}+n_i}) \cong \frac{n_q}{\sqrt{P}} \qquad (3.9)$$

Because the frequency offset estimation error is equal to the phase error θ divided by $2\pi T$, the standard deviation of the frequency error is given by [36, 37]

$$\sigma_f \cong \frac{1}{2\pi T}\sqrt{\frac{N_0}{PT_G}} = \frac{1}{2\pi T}\sqrt{\frac{1}{E_s/N_0}\frac{T_G}{T_S}} \qquad (3.10)$$

where, T_s is the symbol interval and E_s / N_0 is the symbol - to - noise energy ratio, defined as

$$\frac{E_s}{N_0} = \frac{PT_s}{N_0} \qquad (3.11)$$

E_s/N_0 is equal to the bit energy - to – noise density E_b/N_0 multiplied by the number of bits per symbol. Because OFDM typically has a large number of bits per symbol and E_b/N_0 is larger than 1 for successful communications, typical E_s/N_0 values are much larger than 1. For instance, with 48 sub-carriers using 16-QAM and rate ½ coding, there are 96 bits per OFDM symbol. In this case, E_s/N_0 is about 20 dB larger than E_b/N_0. Typical E_b/N_0 value is about 10 dB, typical E_s/N_0 value is about 30 dB.

If the required E_s/N_0 value for an acceptable frequency error level is too large, then averaging the vector y(t) in Equation (3.7) over multiple OFDM symbols can be used to increase the effective signal - to - noise ratio. For averaging over K symbols, the frequency error standard deviation becomes

$$\sigma_f \cong \frac{1}{2\pi T}\sqrt{\frac{1}{KE_s/N_0}\frac{T_G}{T_s}} \qquad (3.12)$$

3.4.2. Synchronization Using Special Training Symbols

The synchronization technique based on the cyclic extension is particularly suited to tracking or to blind synchronization in a circuit-switched connection, where no special training signals are available. For packet transmission, however, there is a drawback because an accurate synchronization needs an averaging over a large (>10) number of OFDM symbols to attain a distinct correlation peak and a reasonable SNR. For high-rate packet transmission, the synchronization time needs to be as short as possible, preferably a few OFDM symbols only. To achieve this, special OFDM training symbols can be used for which the data content is known to the receiver [9, 12, 14]. In this way, the entire received training signal can be used to achieve synchronization, whereas the cyclic extension method only uses a fraction of each symbol.

Figure 3.9 shows a block diagram of a matched filter that can be used to correlate the input signal with the known OFDM training signal. Here, T is the sampling interval and C_i are the matched filter coefficients, which are the complex conjugates of the known training signal. From the correlation peaks in the matched filter output signal, both symbol timing and frequency offset can be estimated. The matched filter correlates with the OFDM time signal before performing a FFT in the receiver.
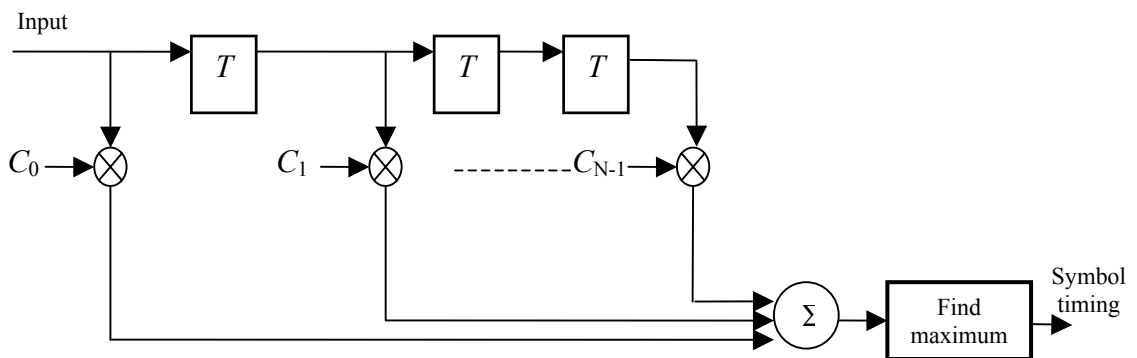
Figure 3.9 Matched filter that is matched to a special OFDM training symbol

3.4.3. Optimal Timing In The Presence Of Multi-path

The task of OFDM symbol timing is to minimize the amount of ISI and ICI. This type of interference is absent when the FFT is taken over the flat part of the signaling window, which is shown in Figure 3.10. This window is the envelope of the transmitted OFDM symbols. Within the flat part of the window, all sub-channels maintain perfect orthogonality. In the presence of multi-path, however, orthogonality is lost if the multi-path delays exceed the effective guard time, which is equal to the duration of the flat window part minus the FFT period.
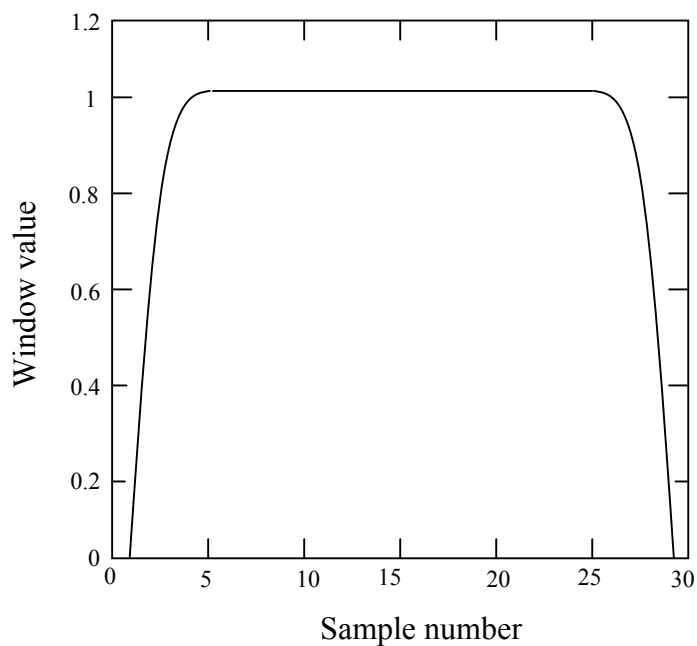
Figure 3.10 Raised cosine window

The effect of multi-path propagation on ISI and ICI is illustrated in Figure 3.11. It shows the windowing envelopes of three OFDM symbols. The radio channel consists of two paths with a relative delay of almost half of a symbol and relative amplitude of 0.5. The receiver selects the FFT timing such that the FFT is taken over the flat envelope part of the strongest path. Because the multi-path delay is larger than the guard time, however, the FFT period cannot at the same time cover a totally flat envelope part of the weaker signal. As a result, the non-flat part of the symbol envelope causes ICI. At the same time, the partial overlap of the previous OFDM symbol in the FFT period causes ISI.

The solution to the timing problem is to find the delay window with a width equal to the guard time. This contains maximum signal power. The optimal FFT starting time, then, is equal to the following equation:

The starting delay of the found delay window, plus the delay that occurs between a matched filter peak output from a single OFDM pulse and the delay of the last sample on the flat part of the OFDM signal envelope, minus the length of the FFT interval.
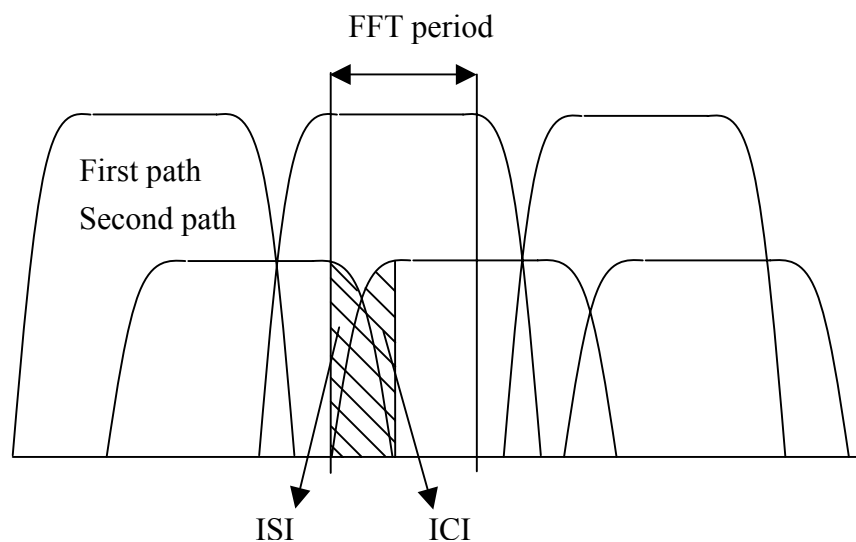
Figure 3.11 ISI/ICI caused by multi-path signals

Figure 3.12 shows the OFDM symbol structure, where T is the time needed by the FFT. If a multi-path signal is introduced with a relative delay (relative to the delay of the shown reference OFDM signal) exceeding T_{g1}, it will cause ISI and ICI. Similarly, multi-path signals with relative delays less than $-T_{g2}$ cause ISI and ICI. The timing problem is now to choose T_{g1} and T_{g2} such that the amount of ICI and ISI after the FFT is minimized.
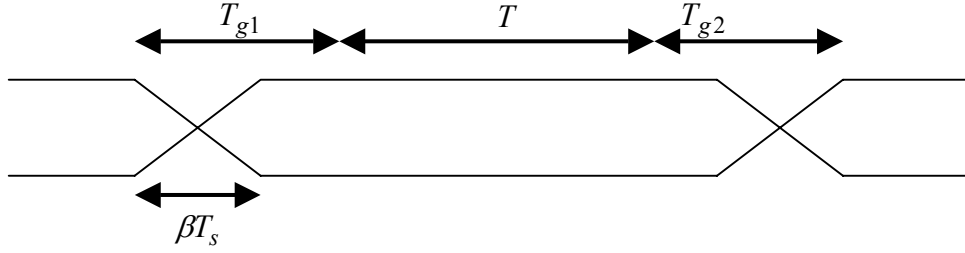
Figure 3.12 OFDM symbol structure

It is clear in Figure 3.12 that ISI and ICI are caused by all multi-path signals, which delays fall outside a window of $T_g = T_{g1} + T_{g2}$. All multi-path signals within this delay window contribute to the effectively used signal power. Hence, the optimal timing circuit maximizes the signal - to - (ISI + ICI) ratio (SIR), given by [36]

$$SIR = \frac{S_u}{S_t - S_u}, \; S_u = \int_{T_0}^{T_0+T_g} \|h(\tau)\|^2 d\tau, \; S_t = \int_{-\infty}^{\infty} \|h(\tau)\|^2 d\tau \qquad (3.13)$$

where, $T_0 = -T_{g2}$ is the timing offset of the guard time window T_g. S_t denotes the total received signal power and S_u is the useful signal power. Because only S_u depends on the timing offset T_0, the SIR is maximized by maximizing S_u; that is, choosing the T_0 value that contains the largest power of $h(\tau)$ in the interval $\{T_0, T_0 + T_g\}$.

In this chapter, we reviewed and classified synchronization problems existing in OFDM systems and we provided general overview about synchronization techniques used in OFDM receivers. In Chapter 4, the implementation part of our thesis can be seen. We proposed and designed a digital synchronizer hardware, which realizes the timing synchronization of ETSI OFDM symbol (frame) using sliding correlation method. OFDM symbol synchronizer that we designed uses the preambles defined in

ETSI HiperLan / 2 standard to detect the OFDM symbol (please see Appendix D for detailed information about ETSI standard). We can remember from previous chapter, timing synchronization means to find out where the OFDM symbol boundaries are. Notice that, in our implementation we assumed a perfect media. That means CO, CFO, AWGN and phase offset do not exist. Hence these issues were not considered by our synchronizer.

4. SYNCHRONIZATION PRACTICE: SYNCHRONIZATION DETECTION USING SHORT TRAINING SYMBOLS (STS)

In the previous chapters, we reviewed OFDM, generation of OFDM signal. We analyzed synchronization issues that should be solved in OFDM receivers. This chapter presents digital design and hardware implementation of ETSI OFDM symbol synchronizer, which detects ETSI OFDM symbols at the receiver using sliding correlation method. The goal of this implementation is not to design a whole OFDM synchronizer that realizes all synchronization tasks including CFO and CO compensations. The first synchronization task that a receiver should perform is to find out where OFDM symbols starts. Our aim in this implementation is to design a generic digital hardware that can be used in OFDM receivers, which performs the detection of ETSI OFDM symbols referenced in Appendix D. In the first part of this chapter, we have analyzed preambles and correlation characteristics to be used in our implementation. We have given the assumptions and parameters used in the implementation. First sliding and cross correlation methods used to solve synchronization problems have been described, then they have been compared to each other according to matlab OFDM model's correlation outputs to determine the suitable one for our implementation. In the second part, we have explained the details of the OFDM symbol synchronizer that we proposed and designed, followed by simulation and synthesis results achieved in the implementation.

4.1. Preamble and Correlation Characteristics

One of the purposes of the preamble preceding every OFDM packet is to allow start-of-symbol detection. Using the fact that it is based on well-known patterns, which the receiver can recognize. The beginning of the preamble is based on Short Training Symbols ("STS", 16 samples long instead of 64) while the end is based on Long

Training Symbols ("LTS", having the normal length of 64 samples). The reason why the short training symbols are only 16 samples long is due to the frequency domain sequence on which they are based, where every fourth carrier carries data while all the others do not. The result of the IFFT of such a sequence is that the 64 time domain samples can be split in 4 identical sub-symbols or 4 STS's. Figure 4.1 illustrates an example of the ETSI UP LONG preamble, where the short training symbols are the consecutive B's and the IB, while the long training symbols are both final C's:
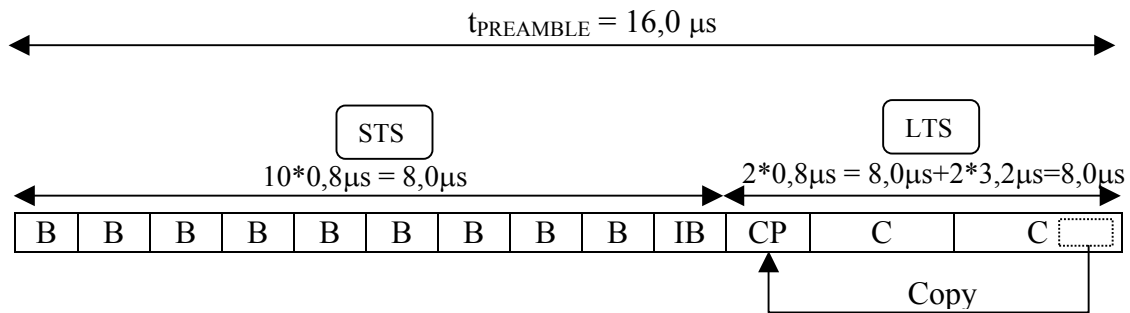
Figure 4.1 ETSI UP LONG preamble

Here, IB short OFDM symbol is sign-inverted copy of B short OFDM symbol (please see Appendix D).

In this thesis, we refer to Appendix D.1.5 for a complete description of all the available preamble structures in the HiperLAN/2 physical layer standards. We consider numerical values seen in Table D.6.

4.1.1. Short Training Symbols (STS)

We consider the preambles defined by ETSI, which contain an IB symbol at the end of the STS section. Since the ETSI preambles are based upon defined symbols, it makes sense to compute a sliding correlation over 16 samples of successive received STS with each other (see the example in Figure 4.2) in order to find the ending point of the STS section and the start of the LTS part. The main idea behind sliding correlation is to correlate the successive received OFDM samples to each other within a proper correlation window. In our implementation, the correlation window length is 16 because received STS has 16 samples length each, which means at each clock cycle, two 16 bits-

long digital inputs are correlated each other. The received complex OFDM symbols are sampled at receiver and converted to digital samples, so that real and imaginary parts are separated. The digital samples are shifted through a shift register block to synchronizer module. This performs correlation process. Figure 4.2 illustrates this sliding correlation process of received digital samples. Since the received samples are shifted through shift registers and delayed at each clock cycle, it is possible to compute a sliding correlation over 16-samples correlation window, which means the newest 16 samples are correlated to the previous 16 samples. The zoomed view of sliding correlation process of two successively received symbols is depicted in Figure 4.3.

Figure 4.4 illustrates the shape of the amplitude and phase of the 16 samples sliding correlation process of the short training symbols contained inside the ETSI BROADCAST. The schematic of the ETSI BROADCAST Preamble and reference STS data for ETSI BROADCAST Preamble dumped from matlab simulink model is shown in Figure 4.5. The goal for the synchronizer is to detect the IB section based upon the sequence SB to find out where the OFDM symbol starts. In the ETSI BROADCAST case, the correlation amplitude is the same for both SA and SB based section (please see D.1.5.4.7.1 for details of SA and SB). However the correlation phase transition (from high to low or from low to high) allows us to distinguish one from the other.

Figure 4.4 reflects the ideal conditions, meaning that there is no AWGN (Additive White Gaussian Noise) injected, no perturbation induced by the channel (the channel impulse response is a single tap in the time domain), no CFO (Carrier Frequency Offset) and no CO (Clock Offset) exist.
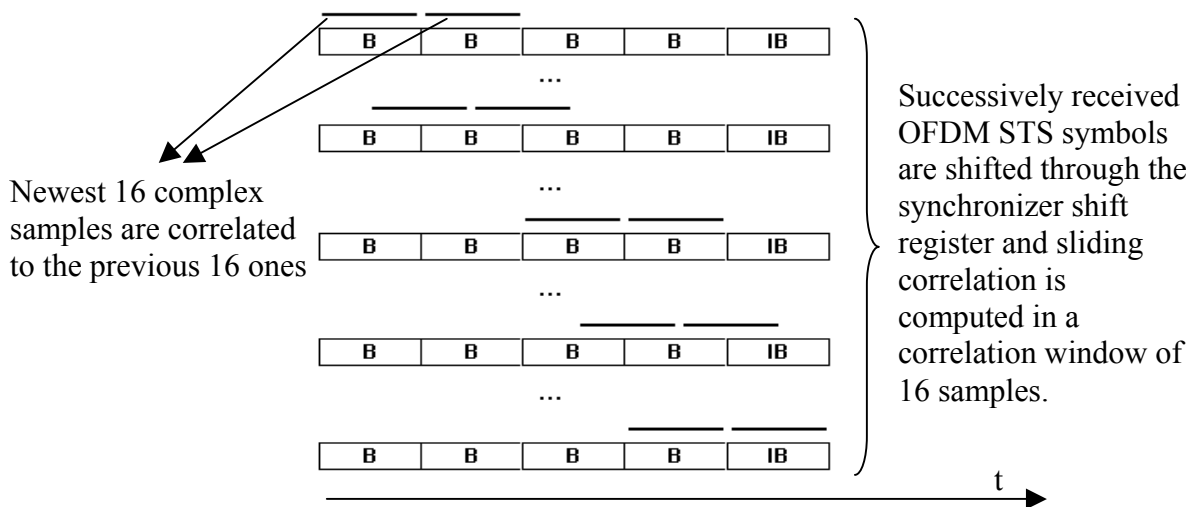
Figure 4.2 Illustration of Sliding Correlation of Received STS

STS1 (i.e. B)

SlidRLength (here 16 samples)

STS2 (i.e. IB)

SlidRLength (here 16 samples)

Samples

Shift register

Shift register

Multiplication sample by sample

Addition of all values

I, Q

•"R" represents "Correlator".
•"Slid" represents "Sliding".
•SlidRLength is determined by the length of STS symbols.
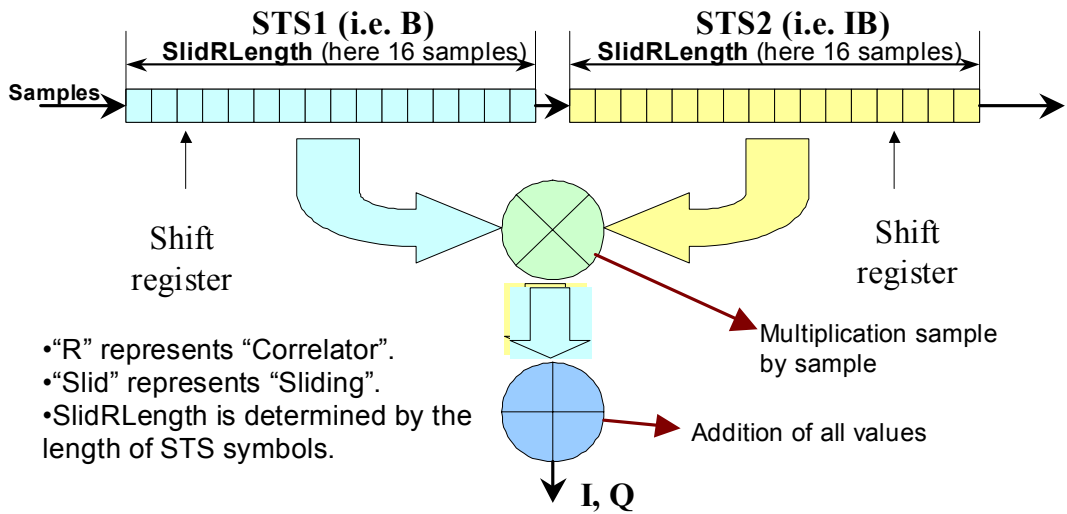
Figure 4.3 Sliding correlation of two received STS symbols over a 16 samples correlation window

Correlation phase makes sharp transition from high to low at the end of SA based section.

Correlation phase makes sharp transition from low to high at the end of SB based section.

Correlation amplitude

Correlation phase

SA

SB

Sample

Sample

(a)

(b)
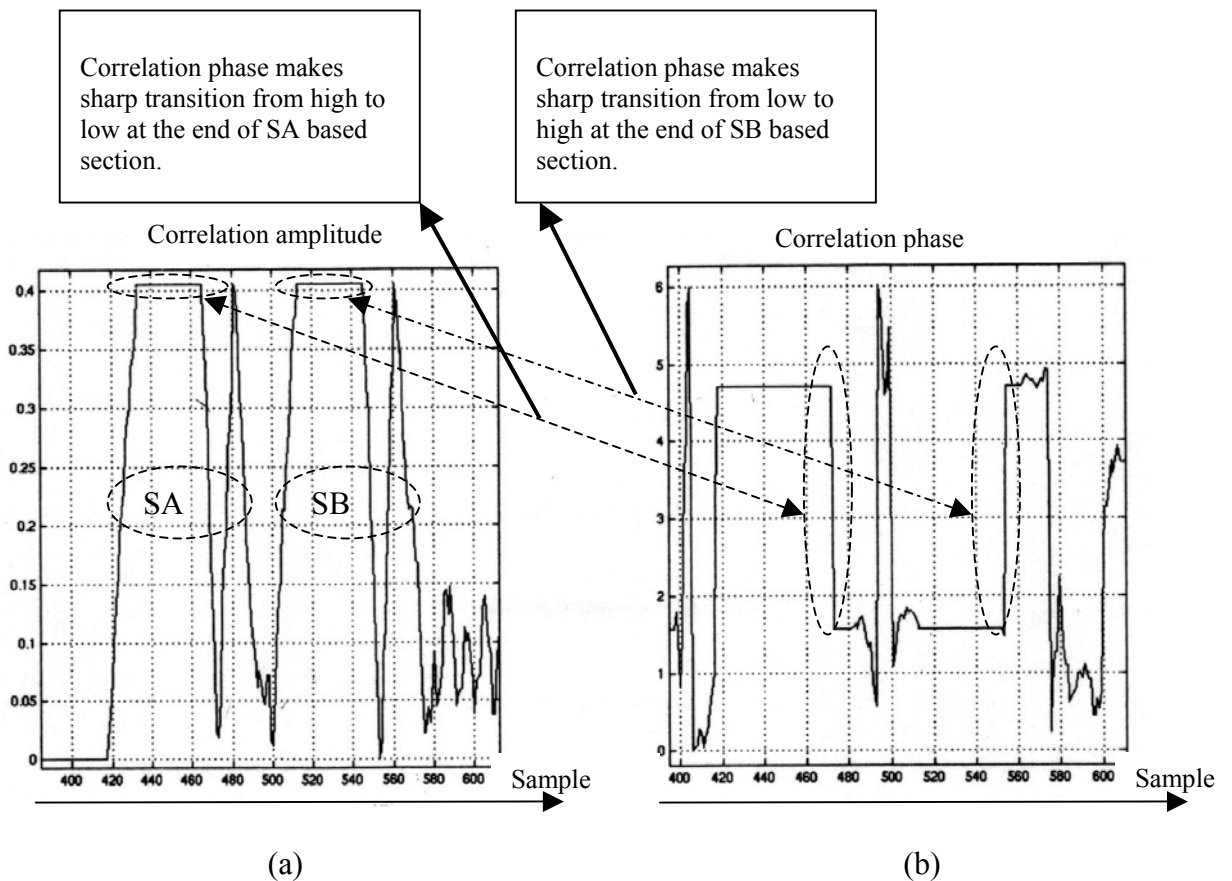
Figure 4.4 Sliding Correlation of the ETSI BROADCAST Preamble: (a) Correlation Amplitude. (b) Correlation Phase

45

SHORT TRAINING SYMBOL (STS)

LONG TRAINING SYMBOL (LTS)

A | IA | A | IA | IA | B | B | B | B | IB | CP | C | C

Copy

ETSI BROADCAST PREAMBLE

A
IA
B
IB
} SHORT OFDM SYMBOLS (EACH 16 SAMPLES)

IA } SIGN-INVERTED COPY OF PRECEDING SHORT SYMBOL   A

IA = - A

IB } SIGN-INVERTED COPY OF PRECEDING SHORT SYMBOL   B

IB = - B

C } NORMAL LENGTH OFDM SYMBOL (64 SAMPLES)

CP } CYCLIC PREFIX (COPY OF 32 LAST SAMPLES OF C SYMBOLS)
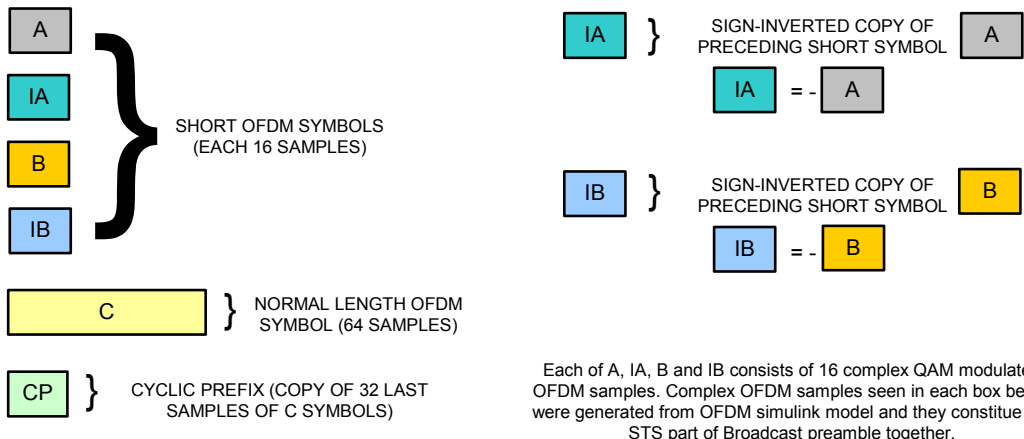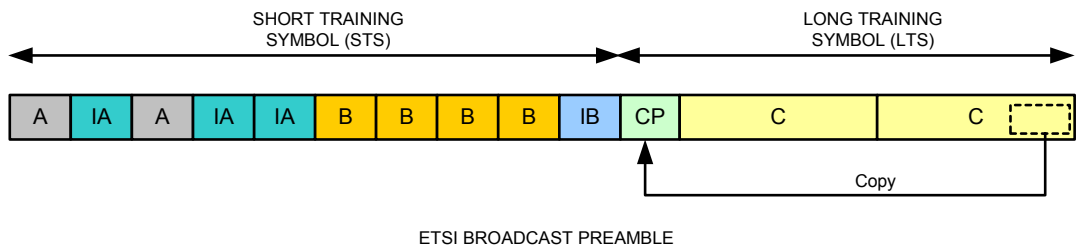
Each of A, IA, B and IB consists of 16 complex QAM modulated OFDM samples. Complex OFDM samples seen in each box below were generated from OFDM simulink model and they constitue the STS part of Broadcast preamble together.

A	IA	B	IB
-0.0920 + 0.0000i	0.0920 - 0.0000i	0.0460 + 0.0460i	-0.0460 - 0.0460i
-0.0944 - 0.0414i	0.0944 + 0.0414i	-0.1324 + 0.0023i	0.1324 - 0.0023i
-0.1202 - 0.0352i	0.1202 + 0.0352i	-0.0135 - 0.0785i	0.0135 + 0.0785i
0.0137 + 0.1179i	-0.0137 - 0.1179i	0.1428 - 0.0127i	-0.1428 + 0.0127i
0.1301 - 0.0000i	-0.1301 + 0.0000i	0.0920 + 0.0000i	-0.0920 - 0.0000i
0.0137 - 0.1179i	-0.0137 + 0.1179i	0.1428 - 0.0127i	-0.1428 + 0.0127i
-0.1202 + 0.0352i	0.1202 - 0.0352i	-0.0135 - 0.0785i	0.0135 + 0.0785i
-0.0944 + 0.0414i	0.0944 - 0.0414i	-0.1324 + 0.0023i	0.1324 - 0.0023i
-0.0920 - 0.0000i	0.0920 + 0.0000i	0.0460 + 0.0460i	-0.0460 - 0.0460i
-0.1413 + 0.0277i	0.1413 - 0.0277i	0.0023 - 0.1324i	-0.0023 + 0.1324i
-0.0498 - 0.0850i	0.0498 + 0.0850i	-0.0785 - 0.0135i	0.0785 + 0.0135i
0.0691 - 0.0234i	-0.0691 + 0.0234i	-0.0127 + 0.1428i	0.0127 - 0.1428i
-0.0000 + 0.1301i	0.0000 - 0.1301i	0.0000 + 0.0920i	-0.0000 - 0.0920i
-0.0691 - 0.0234i	0.0691 + 0.0234i	-0.0127 + 0.1428i	0.0127 - 0.1428i
0.0498 - 0.0850i	-0.0498 + 0.0850i	-0.0785 - 0.0135i	0.0785 + 0.0135i
0.1413 + 0.0277i	-0.1413 - 0.0277i	0.0023 - 0.1324i	-0.0023 + 0.1324i

A | IA | B | IB

Figure 4.5 ETSI BROADCAST Preamble and STS Data dumped from OFDM simulink model.

Instead of a sliding correlation, we could also compute the cross correlation, i.e. correlate the received data with the ideal transmitted symbol (which is not altered by noise, channel etc.). The principle is sketched in Figure 4.6. First difference between the sliding correlation and the cross correlation is that cross correlation is realized between the received STS and the ideal transmitted symbol while sliding correlation occurs between consecutive received STS symbols. Second difference is that two inputs of sliding correlator changes at each clock cycle while just one input of cross correlator changes at each clock cycle. This is because the transmitted ideal symbol is correlated with the newest received 16 samples. The zoomed view of cross correlation process of the successively received symbols with the ideal transmitted symbol is depicted in Figure 4.7.
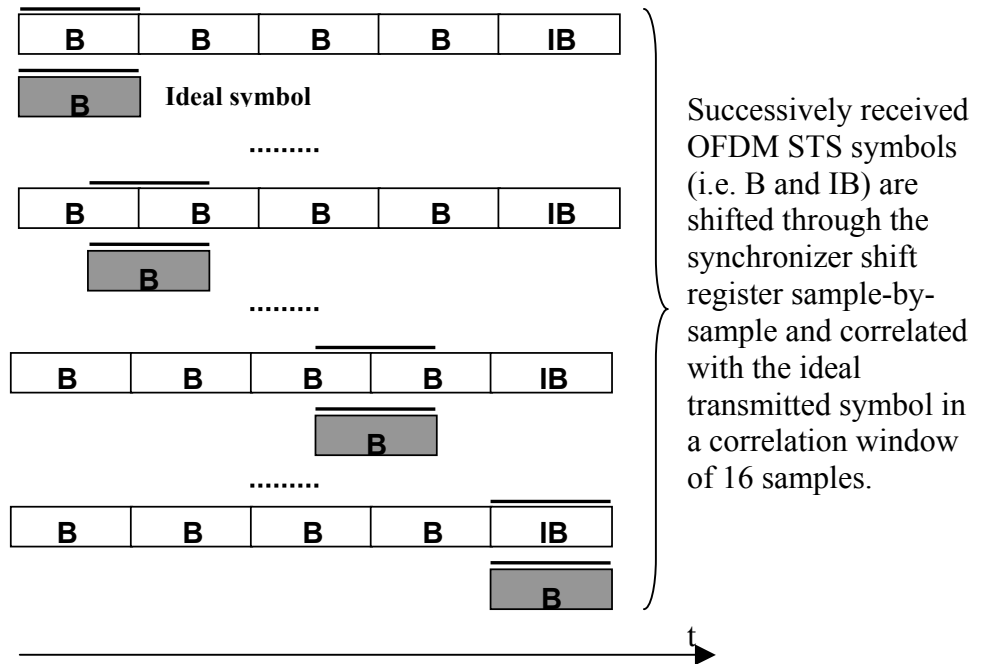
Figure 4.6 Example of Cross Correlation of Received STS

Samples **STS (i.e. B)**

Transmitted ideal B symbol
CrossRLength (here 16 samples)

Shift register

•CrossRLength is determined by the length of STS.

Register (DOES NOT shift)

Complex multiplication sample by sample

Addition of all values

I, Q
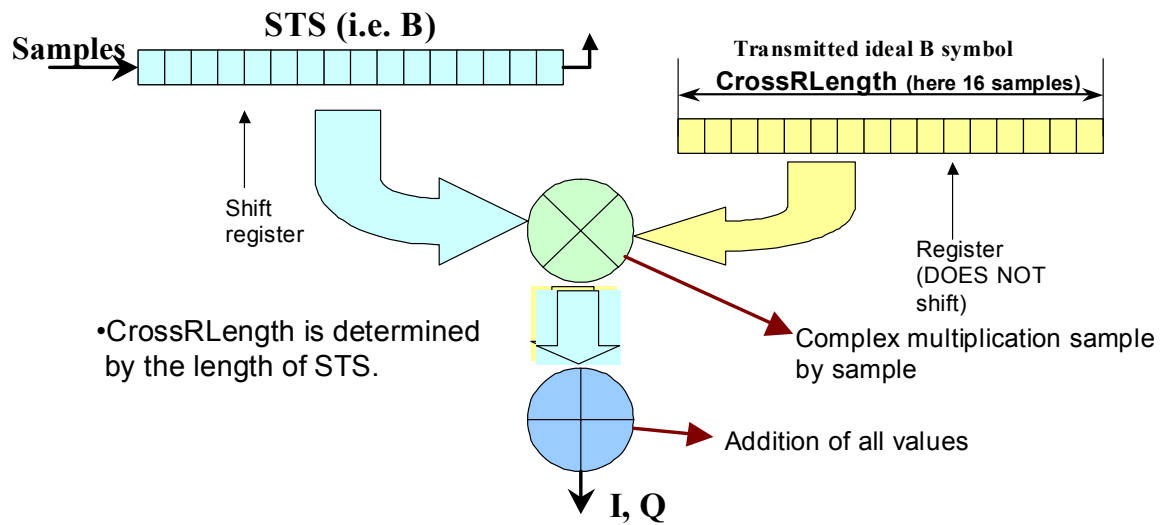
Figure 4.7 Cross correlation of the received STS symbols with the transmitted ideal symbol in a 16 samples correlation window

Figure 4.8 represents the amplitude and the phase of the ideal cross correlation (without any perturbation-no channel effect) of the transmitted symbol with the received data. As seen in cross correlation phase graph, the phase of the cross correlation does not presents consecutive stable values. The phase jump therefore is not easily detectable. The 4 peaks seen in cross correlation amplitude graph correspond to the match between received and ideal B symbols, while the last one is due to the correlation between the received IB and the ideal B. Synchronization on ETSI preambles is best performed by using a sliding correlation of the received samples so that the pattern is more easily detectable. This is the main reason why sliding correlation method is used for synchronization of ETSI preambles in this thesis.
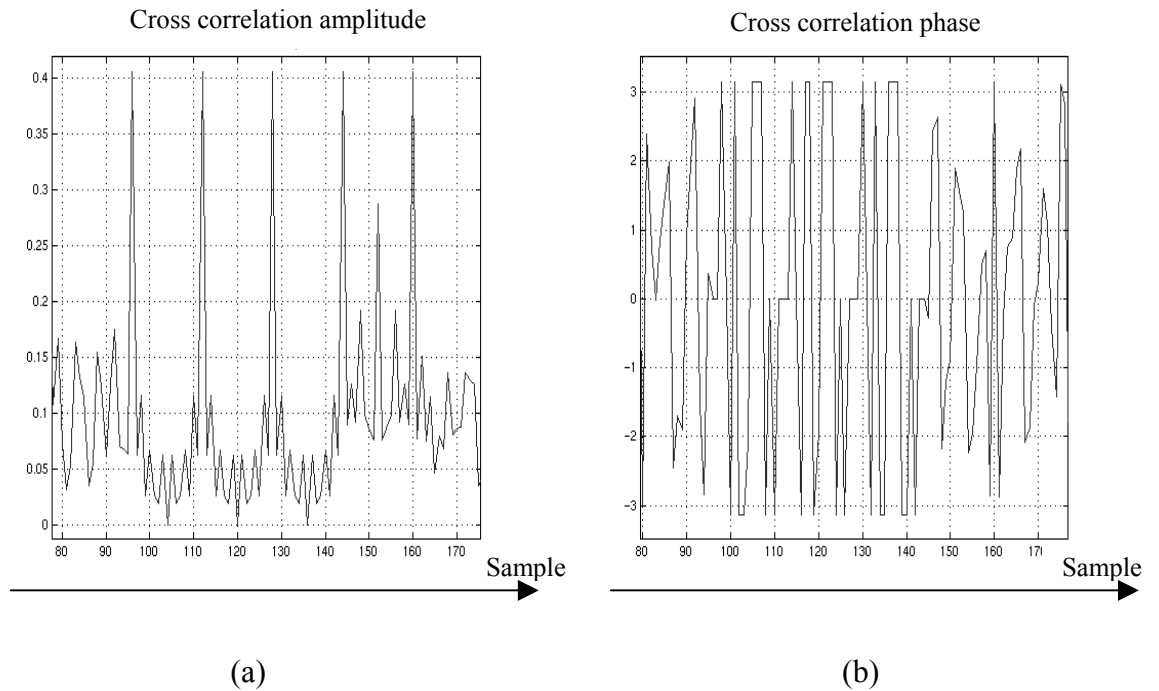
Figure 4.8 ETSI Ideal Cross Correlation: (a) Correlation Amplitude. (b) Correlation Phase

All the previous figures assume perfect conditions. In realistic situations, various effects have to be taken into account that impairs the previous results. Since we assume the ideal conditions, we will not analyze these effects.

4.1.2. Long Training Symbols (LTS)

All the Physical Layer burst structures (please see D.1.5.4.7) have a preamble containing two specific OFDM symbols (C) of normal length (64 samples, hence called 'long training symbols'), preceded by a cyclic prefix of the symbols copying the last 32 samples of the C symbols. One might compute the sliding correlation over 32 samples of the LTS.

4.2. Digital Design and Hardware Implementation of ETSI OFDM STS Synchronizer Using Sliding Correlator

4.2.1. Top-level Architecture

As mentioned before, the synchronization in OFDM systems is accomplished using correlation methods. Since the synchronization pattern of sliding correlation is more easily detectable than the one of cross correlation, we have selected the sliding correlation method for our synchronization application. The top-level block representation of the STS synchronizer is depicted in Figure 4.9.
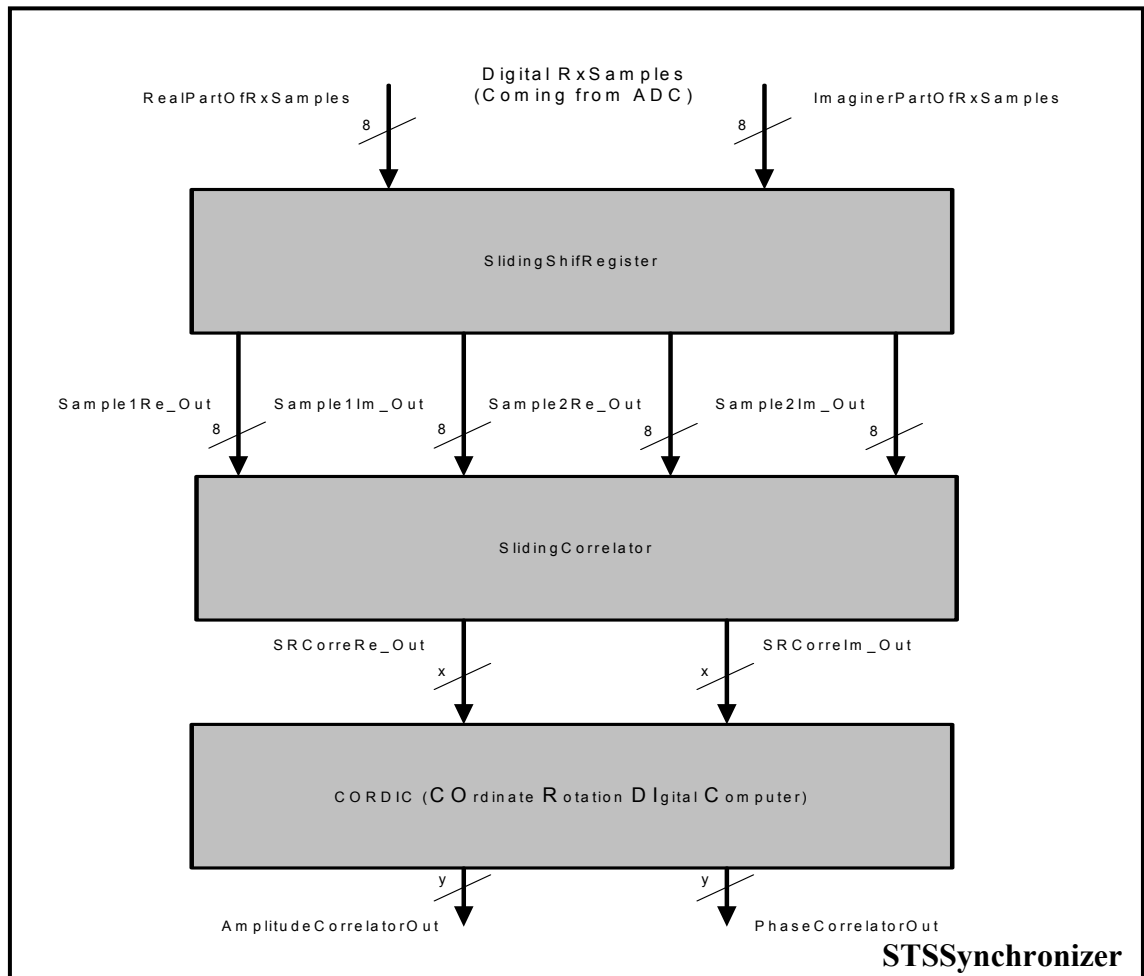
Figure 4.9 Top-level Block Diagram of ETSI OFDM STS Synchronizer

In our hardware implementation we assume the followings:

- Transmitted OFDM symbols are received successively by the antenna at the OFDM receiver and sampled at 20 MHz sampling clock frequency.
- The received complex analog OFDM samples (sub-carriers which carries QAM modulated complex data) are already converted to digital data; real and imaginary parts are separated each other
- Our synchronizer takes these digital samples; real and imaginary parts are coming separately.

STS Synchronizer design consists of 3 sub-modules:

1. Sliding Shift Register module: It gets digital samples coming from the previous module (Analog to Digital Converter (ADC) Shifter) and stores the newest 16 samples. It provides the next block with correct data (Sample1 and Sample 2) to be correlated.
2. Sliding Correlator module: This module realizes sliding correlation process of the inputs provided by Sliding Shift Register module.
3. CORDIC module: This module outputs the amplitude and phase characteristics of the correlated OFDM data.

All sub-blocks are handled in more details in the next sections.

In our implementation, we consider numerical values seen in Table D.6. In the simulink OFDM model that we used, the modulation type was selected as 64QAM, preamble type was chosen as ETSI_BROADCAST, CO was set to 0 Hz, CFO was set to 0 ppm and SNR was set to 210 dB that provided perfect conditions. Transmitted complex OFDM data was dumped from this model and used as the input stimuli of our hardware implementation. Real and imaginary parts of the generated input stimuli were separated each other and converted to 8-bits digital samples each.

4.2.1.1.Sliding Correlator Shift Register Unit (SlidingShiftRegister)

The function of the Sliding Correlator Shift Register Unit is to store the newest 16 samples and to provide SlidingCorrelator block with correct data to be correlated. At rising edge of clock, new sample is registered and oldest sample is discarded.

In SlidingShiftRegister Block there are 17x8x2-Bit shift register to store the most recent 16 samples. Each sample consists of real and imaginary parts, stored in 8-Bit precision.

The detailed architecture of SlidingShiftRegister block is depicted in Figure 4.10. It simply gets the digital real and imaginary parts of received OFDM samples and it stores newest 16 samples. Then it outputs newest sample and 16-clock cycles delayed sample to Sliding Correlator module, next block. The reason of this functionality is that the length of correlation window is for 16 samples (STS i.e. B, A, IA and IB each consists of 16 samples to be correlated). The correct data to be correlated by the next block should be the newest one and the 16-clock cycles delayed one.
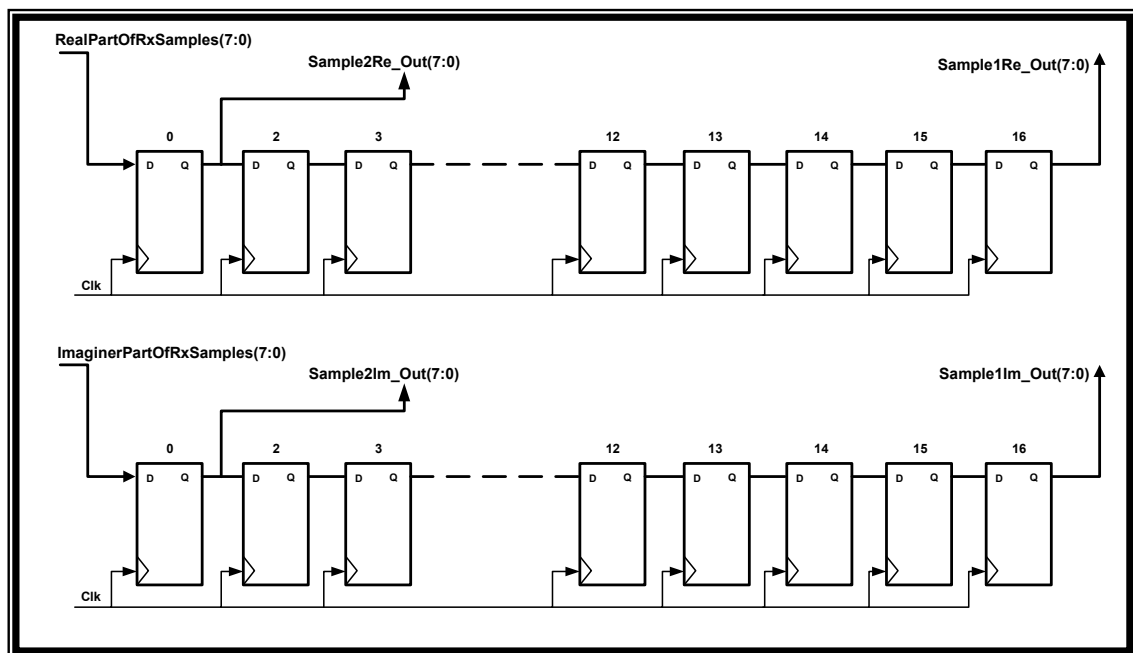
Figure 4.10 Architecture of SlidingShiftRegister Block

4.2.1.2.Sliding Correlator Unit

The general operation of a correlator is multiplying its input signals "sample-by-sample" and adding the product of each multiplication. If the signals to be correlated are of complex samples, i.e. a+jb and c+jd, then the multiplication occurs between one input signal and the "conjugate" of the other input signal. For instance, (a+jb) is multiplied by (c-jd), the conjugate of c+jd.

The implementation of "Sliding Correlator" depends on a serial approach, which means at each clock cycle, it's enough to multiply 2 samples with each other and add the product to previous one to take the correlation of two successively received STS symbols. This structure requires just one "complex multiplier". Instead of this, all 16 samples of two received STS can be multiplied each other in parallel, then all multiplication products can be accumulated to perform the sliding correlation of two successive received OFDM symbols. This approach requires (# of samples in a STS) complex multiplier units. This increases the area.

Sliding Correlator consists of 2 sub-blocks as shown in Figure 4.11:

1. SRCorrComplexMultiplier: Complex Multiplier Block
2. SRCorrAccumulator: Accumulator Block

Sliding Correlator basically receives real and imaginary samples from Sliding Correlator Shift Register and Complex Multiplier Sub-block in Sliding Correlator multiplies each new sample with the conjugate of a sample that has been received 16 samples before. At each clock cycle, new product of input samples is added to the previous sum and the oldest product of samples is subtracted from this sum in Accumulator Block. The output of this accumulator is the output of SlidingCorrelator block.
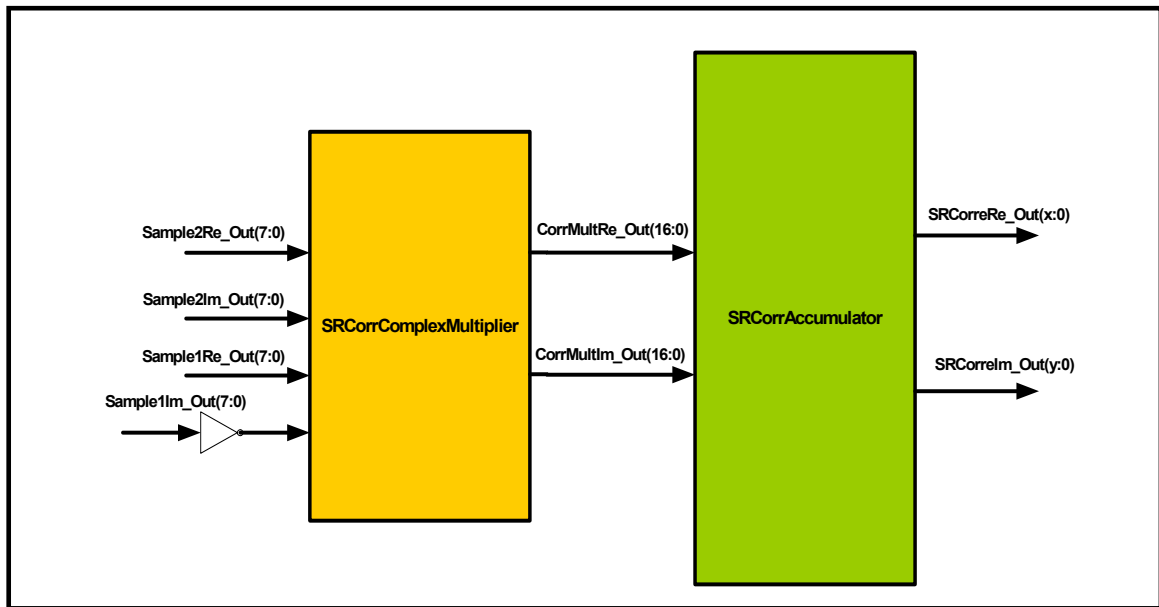
Figure 4.11 The top-level block diagram of Sliding Correlator

A complex multiplier seen in Figure 4.12 is used in SlidingCorrelator block. This complex multiplier works on the simple principle of 4 multiplications as explained below:

Let two complex numbers that are going to be multiplied be $(A + jB)$ and $(C + jD)$ and the product $(P + jQ)$. We have,

$$P = (A \times C) - (B \times D) \text{ and}$$
$$Q = (A \times D) + (B \times C) \tag{4.1}$$

Each of the inputs of the complex multiplier is 8 bits wide. The width of the both real and imaginer output is 17.
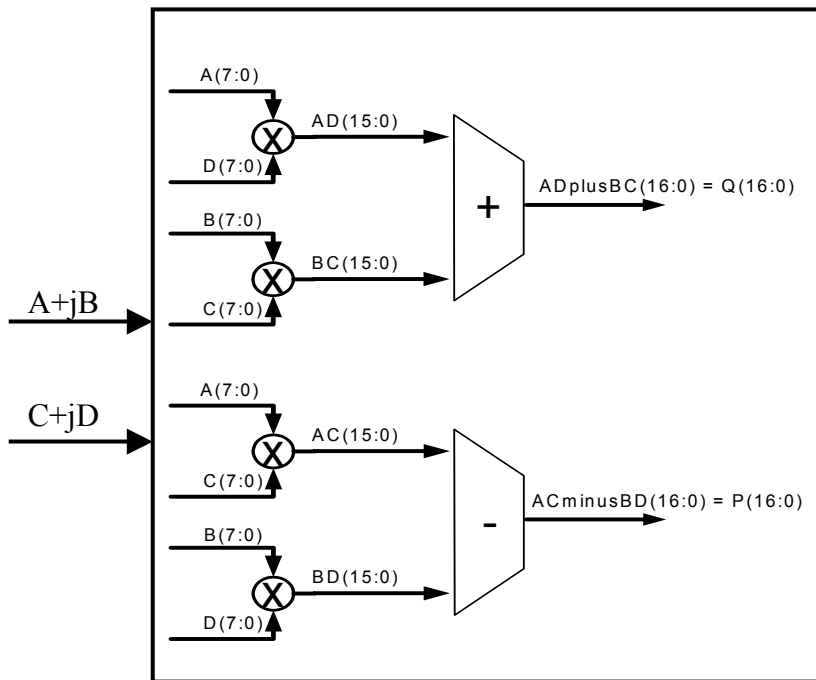
Figure 4.12 Complex Multiplier Structure

As seen in Figure 4.13, the real and imaginary outputs of the SRCorrComplexMultiplier are the inputs of the SRCorrAccumulator. There are two separate shift registers for both real and imaginary parts. At each clock cycle, a new multiplied value for both real and imaginary parts are sent into these internal shift registers, which are 16 samples long.

Real and imaginary shift registers are both 17 bits long.

The methodology of our sliding correlation implementation can be simply explained as follows:

- The first rule of correlation implementation is to multiply the first complex sample (a+jb) with the conjugate of the second complex sample (c-jd). This is done by SRCorrComplexMultiplier.
- The second rule is to accumulate every new product of the SRCorrComplexMultiplier and to subtract the 16 clock cycles delayed product from this accumulation. To understand this explained process let's imagine 17 clock cycles later from the beginning of STS sliding correlation process. At this time the first sample of first B symbol shown in Figure 4.2 is at the output of 17th flip flop of sliding shift register in Figure 4.10 and the first sample of second B symbol is at the output of first flip flop of sliding correlator in Figure 4.10. This is the critical time for the correlation process because the meaningful data begins from this point. The first sample of first B is multiplied by the first sample of second B and the product is sent into SRCorrAccumulator. 16 clock cycles later, all samples of both first B and second B are multiplied each other and the accumulation is done. At each new clock cycle, a new product of complex multiplier should be added to the accumulation while the oldest product is subtracted. This is needed because our STS symbols have 16 samples each so the "correlation window" width is just for 16 samples.

The outputs of sliding correlator are registered at the rising edge of Clk. The reset signal is not shown in all figures for clarity.

Figure 4.11, Figure 4.12 and Figure 4.13 show the detailed diagrams of the SlidingCorrelator block with the widths of input, internal and output signals. As seen in these figures, the input samples of SlidingCorrelator are 8 bits for real or imaginary parts but the internal samples are of 23 bits for real or imaginary parts. This is because inside the correlator a much bigger precision than 8 bits is needed in order not to loose precision during the computations.
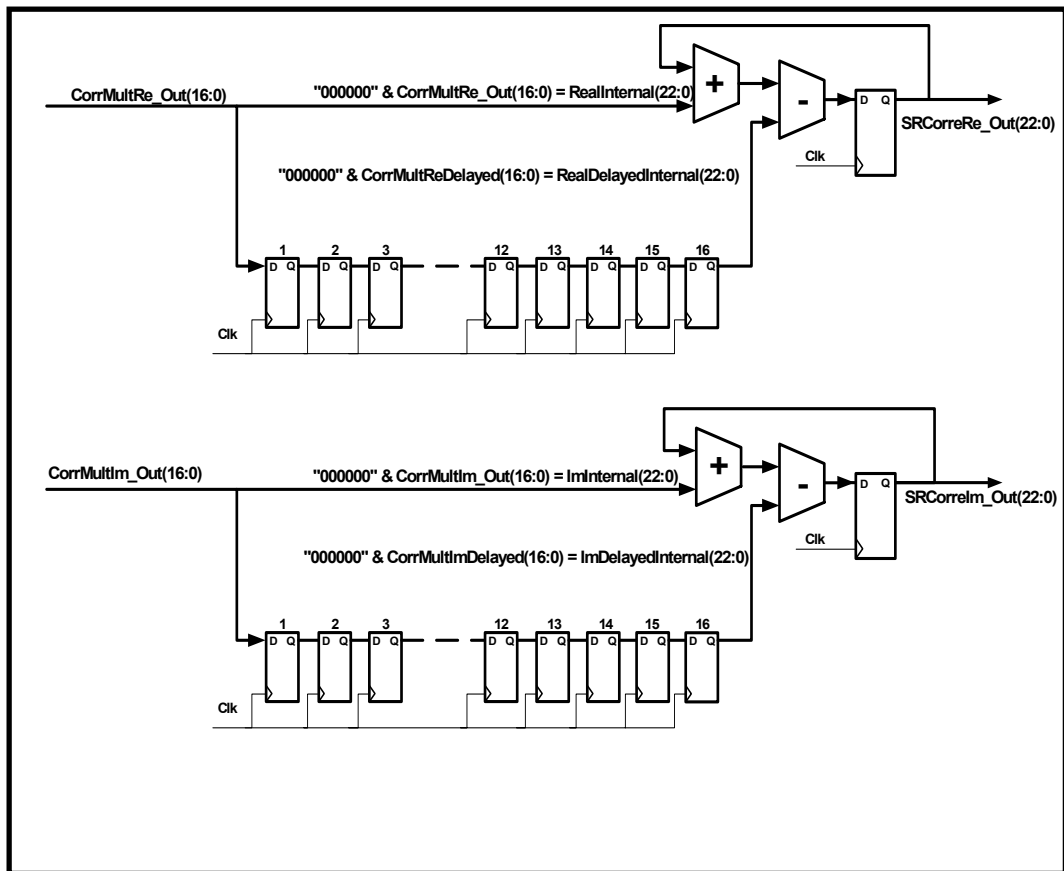
Figure 4.13 Detailed architecture of SRCorrAccumulator block

4.2.1.3.CORDIC (COrdinate Rotation DIgital Computer) Unit

In Figure 4.9, at the output of SlidingCorrelator block, we have real and imaginary samples of correlated OFDM symbols. But we still need to find out the amplitude and the phase values of correlated OFDM samples to implement the OFDM synchronizer. Amplitude and phase characteristics should be analyzed to detect the OFDM symbol at the receiver. An algorithm called "CORDIC" has been developed for such kind of operations, i.e. computing trigonometric functions that are based on vector rotations. The CORDIC algorithm provides an iterative method of performing vector rotations by arbitrary angles using only shifts and adds which facilitates the digital design of this [28]. This is why the CORDIC algorithm was chosen in our thesis.

57

4.2.1.3.1. Functional Description: Cordic Theory

CORDIC is a hardware-efficient algorithm that brings an iterative solution for trigonometric functions and uses only shifts and adds to perform.

All of the trigonometric functions can be computed or derived from functions using vector rotations. Vector rotation can also be used for polar to rectangular and rectangular to polar conversions.

The CORDIC algorithm provides an iterative method of performing vector rotations (see Figure 4.14) by arbitrary angles using only shifts and adds. The rotation is derived from the general rotation transform as follows:
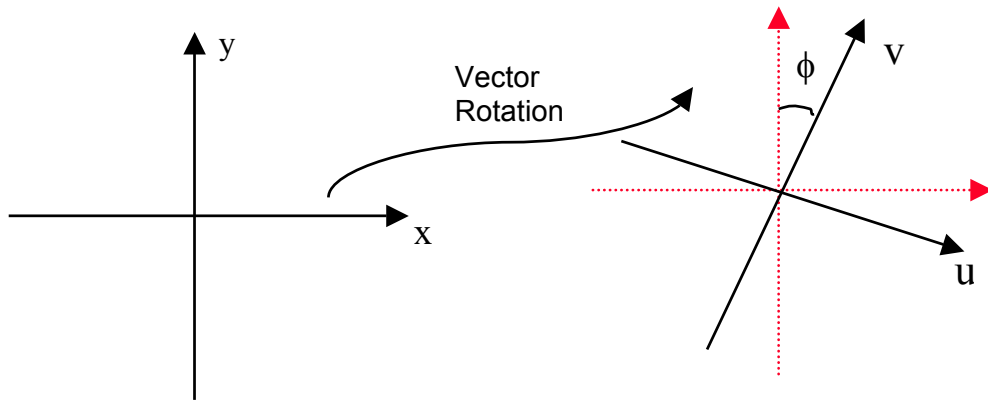
Figure 4.14 Vector Rotation

$$u = x\cos\phi - y\sin\phi$$
$$v = y\cos\phi + x\sin\phi \tag{4.2}$$

which rotates a vector in a Cartesian plane by the angle ϕ. These can be rearranged so that:

$$u = \cos\phi.[x - y\tan\phi]$$
$$v = \cos\phi.[y + x\tan\phi] \tag{4.3}$$

So far, nothing is simplified. However, if the rotation angles are restricted so that $\tan \Delta\phi_i = \pm 2^{-i}, i = 0.....N$, the multiplication by the tangent term is reduced to simple shift operation. Arbitrary angles of rotation are obtainable by performing a series of successively smaller elementary rotations. If the decision at each iteration, i, is which direction to rotate rather than whether or not to rotate, then the $\cos(\delta_i)$ term becomes a constant (because $\cos(\delta_i) = \cos(-\delta_i)$). The iterative rotation can now be expressed as:

$$\tan \Delta\phi_i = \pm 2^{-i}, \quad i = 0.......N.$$

$$u_0 = x, v_0 = v$$

$$u_{i+1} = K_i\left[u_i - v_i.d_i.2^{-i}\right]$$

$$v_{i+1} = K_i\left[v_i + u_i.d_i.2^{-i}\right]$$

$$z_{i+1} = z_i - d_i \tan^{-1}(2^{-i}) \qquad (4.4)$$

where

$$K_i = \cos(\tan^{-1} 2^{-i}) = \frac{1}{\sqrt{1 + 2^{-2i}}}$$

$$d_i = \pm 1$$

i	0	1	2	3
$\tan^{-1}2^{-i}$	45°	26.56°	14.03°	7.12°
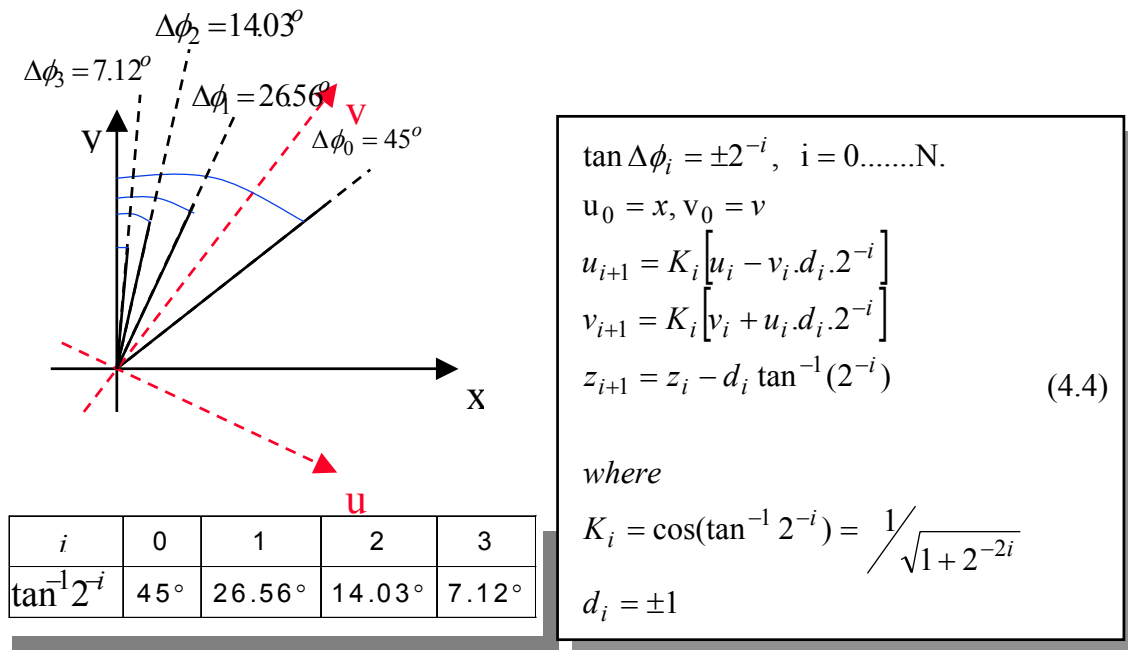
Figure 4.15 Iterative Rotation Solution

Removing the scale constant from the iterative equations yields a shift-add algorithm for vector rotation. The product of the K_i's can be applied elsewhere in the system or treated as part of a system processing gain. That product approaches 0.60725 as the number of iterations goes to infinity. Therefore, the rotation algorithm has a gain, A_n, of approximately 1.647. The exact gain depends on the number of iterations and the relation

$$A_N = \prod_N \sqrt{1 + 2^{-2i}}$$ (4.5)

The angle of a composite rotation is uniquely defined by the sequence of the directions of the elementary rotations. That sequence can be represented by a decision vector. The set of all possible decision vectors is an angular measurement system based on binary arctangents. Conversions between this angular system and any other can be accomplished using a look-up table. A better conversion method uses an additional adder-subtractor that accumulates the elementary rotation angles at each iteration. The elementary angles can be expressed in any convenient angular unit. Those angular values are supplied by a small lookup table (one entry per iteration) or are hardwired, depending on the implementation. The angle accumulator adds a third difference equation to the CORDIC algorithm:

$$z_{i+1} = z_i - d_i . \tan^{-1}(2^{-i})$$ (4.6)

Obviously, in cases where the angle is useful in the arctangent base, this extra element is not needed.

The CORDIC rotator is normally operated in one of two modes. The first, called rotation rotates the input vector by a specified angle (given as an argument). The second mode, called vectoring, rotates the input vector to the x-axis while recording the angle required to make that rotation.

Rotation Mode:

In rotation mode, the angle accumulator is initialized with the desired rotation angle. The rotation decision at each iteration is made to diminish the magnitude of the residual angle in the angle accumulator. The decision at each iteration is therefore based on the sign of the residual angle after each step. Naturally, if the input angle is already expressed in the binary arctangent base, the angle accumulator may be eliminated. For rotation mode, the CORDIC equations are:

$$\hat{u}_{i+1} = \hat{u}_i - \hat{v}_i.d_i.2^{-i}$$
$$\hat{v}_{i+1} = \hat{v}_i + \hat{u}_i.d_i.2^{-i}$$
$$z_{i+1} = z_i - d_i.\tan^{-1}(2^{-i}) \tag{4.7}$$
where
$$d_i = -1 \text{ if } z_i < 0, +1 \text{ otherwise}$$

After N iterations,

$$u_N = K_N.\hat{u}_N = K_N\left[x_0 \cos z_o - y_0 \sin z_0\right]$$
$$v_N = K_N.\hat{v}_N = K_N\left[y_0 \cos z_o + x_0 \sin z_0\right]$$
$$z_N = 0 \tag{4.8}$$
where
$$K_N = \prod_N 1\bigg/\sqrt{1 + 2^{-2i}}$$

Vectoring Mode:

In the vectoring mode, the CORDIC rotator rotates the input vector through the angle necessary to align the result vector with the x-axis. The result of the vectoring operation is a rotation angle and the scaled magnitude of the original vector (the x component of the result). The vectoring function works by seeking to minimize the y component of the residual vector at each rotation. The sign of the residual y component is used to determine which direction to rotate next. If the angle accumulator is initialized with zero, it will contain the traversed angle at the end of the iterations. In the vectoring mode, the CORDIC equations are [28]:

$$\hat{u}_{i+1} = \hat{u}_i - \hat{v}_i.d_i.2^{-i}$$
$$\hat{v}_{i+1} = \hat{v}_i + \hat{u}_i.d_i.2^{-i}$$
$$z_{i+1} = z_i - d_i.\tan^{-1}(2^{-i}) \tag{4.9}$$
where
$$d_i = +1 \text{ if } \hat{v}_i < 0, -1 \text{ otherwise}$$

After N iterations,

$$u_N = K_N \cdot \hat{u}_N = K_N \sqrt{x_0^2 + y_0^2}$$
$$v_N = K_N \cdot \hat{v}_N = 0$$
$$z_N = z_0 + \tan^{-1}(y_0/x_0) \qquad (4.10)$$
where
$$K_N = \prod_N 1/\sqrt{1 + 2^{-2i}}$$

The CORDIC rotation and vectoring algorithms as stated are limited to rotation angles between $-\pi/2$ and $\pi/2$. This limitation is due to the use of 2^0 for the tangent in the first iteration. For composite rotation angles larger than $\pi/2$, an additional rotation shown in Equation (4.11) is required. An initial rotation of either π or 0 can be made avoiding reassignment of the x and y components to the rotator elements. This gives the correction iteration. There is no growth due to this initial rotation.

$$x' = d.x$$
$$y' = y$$
$$z' = z \text{ if } d = 1, \text{ or } \pi - z \text{ if } d = -1 \qquad (4.11)$$
$$d = -1 \text{ if } x < 0, +1 \text{ otherwise}$$

This reduction forms a modulo 2π representation of the input angle. In our implementation we take this initial rotation into consideration. The CORDIC rotator described is usable to compute several trigonometric functions directly and others indirectly. Judicious choice of initial values and modes permits direct computation of sine, cosine, arctangent, vector magnitude and transformations between polar and Cartesian coordinates. In this thesis since we need to find just the vector magnitude (amplitude of the sliding correlator output) and the arctangent (phase of the sliding correlator output) of the complex output vector of sliding correlator block, only these two direct computations of CORDIC rotator are explained in detail below. Note that Cartesian to Polar coordinate transformation also consists of finding the magnitude and phase angle of the input vector provided by the vectoring mode CORDIC rotator.

Arctangent:

The arctangent, $\phi = A\tan(y/x)$, is directly computed using the vectoring mode CORDIC rotator if the angle accumulator is initialized with zero. The argument must be provided as a ratio has the advantage of being able to represent infinity (by setting x = 0). Since the arctangent result is taken from the angle accumulator, the CORDIC rotator growth does not affect the result.

$$z_N = z_0 + \tan^{-1}(y_0/x_0) \qquad (4.12)$$

Vector Magnitude:

The vectoring mode CORDIC rotator produces the magnitude of the input vector as a byproduct of computing the arctangent. After the vectoring mode rotation, the vector is aligned with the x-axis. The magnitude of the vector is therefore the same as the x component of the rotated vector. This result is apparent in the result equations for the vector mode rotator:

$$u_N = K_N \cdot \hat{u}_N = K_N \sqrt{x_0^2 + y_0^2} \qquad (4.13)$$

where $K_N = \prod_N 1 / \sqrt{1 + 2^{-2i}}$. The magnitude result is compensated by multiplying with K_N. Note that this product approaches 0.60725 as the number of iterations goes to infinity.

4.2.1.3.2. Structure Overview

In our ETSI STS synchronizer digital design, the CORDIC algorithm is limited to rotation angles between $-\pi/2$ and $\pi/2$. CORDIC outputs must be compensated by multiplying with K=0.60725 (excluding the angle).

CORDIC module we designed consists of three blocks seen in Figure 4.16:

1. PRE_CORDIC
2. CORDIC_CORE
3. POST_CORDIC

PRE_CORDIC receives real and imaginary parts of correlator data and produces the initial values of CORDIC_CORE.

CORDIC_CORE evaluates the algorithm for Cartesian to Polar conversion (namely vectoring mode CORDIC rotator) and outputs the results of desired iteration. These results are led to POST_CORDIC.

Finally, POST_CORDIC compensates the amplitude and corrects the phase and then gives them out.
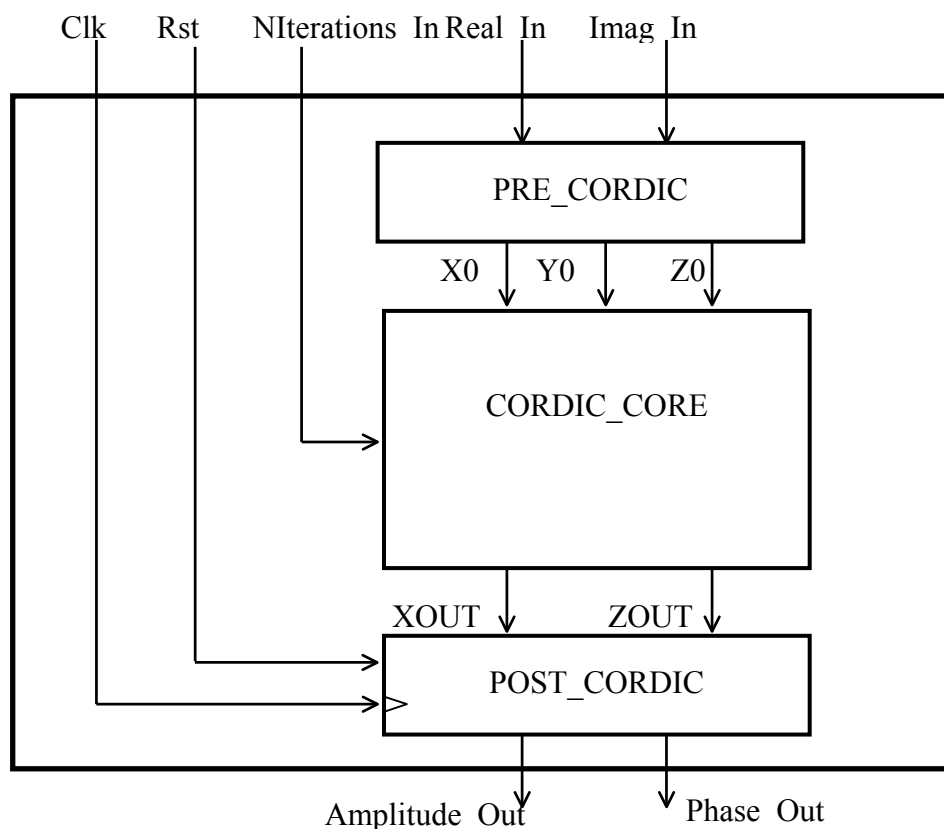
Figure 4.16 Top-level block representation of CORDIC

In our design, CORDIC algorithm works only at $[-\pi/2, \pi/2]$ interval, so a glue logic is required in case there is a data out off this region at the input. PRE_CORDIC is designed to represent this data in the region CORDIC algorithm works. Phase information is represented in $[0, 2\pi]$ interval, and this is evaluated in POST_CORDIC block. Amplitude is compensated in this block as well.

In PRE_CORDIC block (see Figure 4.17), after Real_In and Imag_In enter the CORDIC module, the region of coming data is checked. If Real_In is less than zero, X0 is fed by the inverse signed of Real_In. Imag_In feeds Y0 and Z0 (initial phase) is set to 0. These operations are performed to allocate the data in the region CORDIC_CORE works.

If Real_In is greater than zero, this means that the coming data is already in the region CORDIC_CORE works. So, Real_In and Imag_In are led directly to X0 and Y0, respectively. And, initial phase is set to zero.

XOut output of CORDIC_CORE represents the amplitude of the data coming to CORDIC module, but differs from exact amplitude by a constant. To compensate this difference XOut is multiplied by this constant (K=0.60725) and divided by SQRT (2) for normalization to [0,1] region in POST_CORDIC block depicted in Figure 4.18. Also in this block, a switching operation is applied to phase information in order to provide that phase is between 0 and 2π at the output. Outputs are registered. These registers are updated at rising edge of clock, Clk, and reset with asynchronous active low signal, Rst.

In CORDIC_CORE, depicted in Figure 4.19, N=10 fold ADD/SUB + SHIFTER (The shifters are each a fixed shift, which means that they can be implemented in the wiring) level is put cascaded in order to perform the CORDIC Algorithm. Outputs of each level are led to a mux controlled by NIterations signal (see Figure 4.19). So that, it is available to take the outputs of desired iteration step which is NIterations (NIterations $\leq$ 10). During the implementation of CORDIC block, it has been seen that 10 iteration for the CORDIC Algorithm is quite satisfactory to get the desired result and more iteration step requires more logic in terms of preserving accuracy in arithmetic operations; more iteration is done more hardware is required. Shifter is performing arithmetic shifting to right. The most important point here is to determine all constants to be used for the calculation of z_i's. Constants in CORDIC Algorithm can be calculated as follows:

If we remember that we use vectoring mode CORDIC rotator, our equations are

$$\hat{u}_{i+1} = \hat{u}_i - \hat{v}_i.d_i.2^{-i}$$
$$\hat{v}_{i+1} = \hat{v}_i + \hat{u}_i.d_i.2^{-i}$$
$$z_{i+1} = z_i - d_i.\tan^{-1}(2^{-i}) \qquad (4.14)$$
where
$$d_i = +1 \text{ if } \hat{v}_i < 0, -1 \text{ otherwise}$$

Constants seen in Figure 4.18 to be calculated are determined by

$$\tan^{-1}(2^{-i}) \qquad (4.15)$$

where i = 0, 1, ……, N-1 and N = 10 in our exercise.

So, constants needed are calculated and inserted in Table 4.1.

i.e. Constant0 = $\tan^{-1}(2^0)$ = 45°.

	Constant = $\tan^{-1}(2^i)$		
i	Constant i	Degree	Radian
0	Constant 0	45°	0.78539
1	Constant 1	26.356°	0.46364
2	Constant 2	14.036°	0.24497
3	Constant 3	7.125°	0.12435
4	Constant 4	3.576°	0.06241
5	Constant 5	1.789°	0.03123
6	Constant 6	0.895°	0.01562
7	Constant 7	0.447°	0.05549
8	Constant 8	0.223°	0.00390
9	Constant 9	0.1119°	0.00195

Table 4.1 Constants used in CORDIC_CORE block
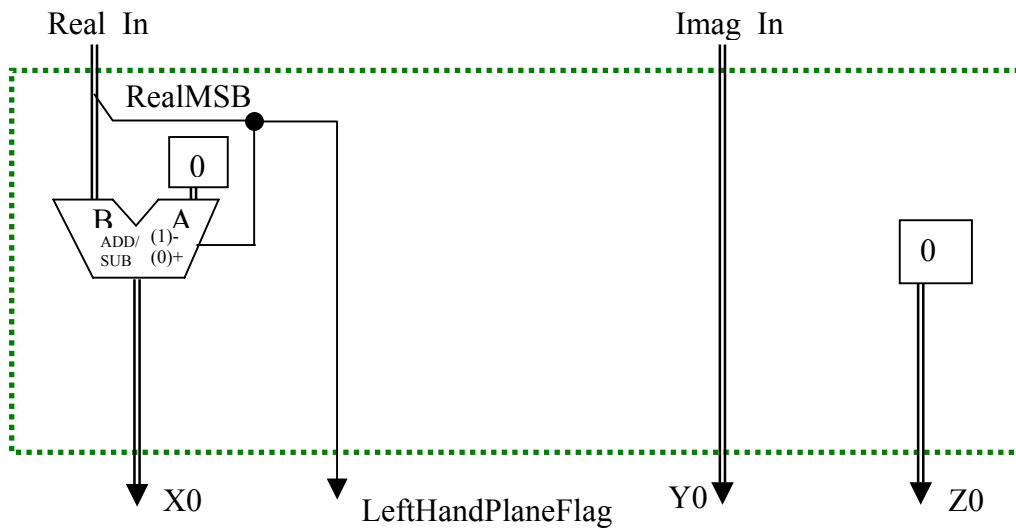
Figure 4.17 PRE_CORDIC Structure
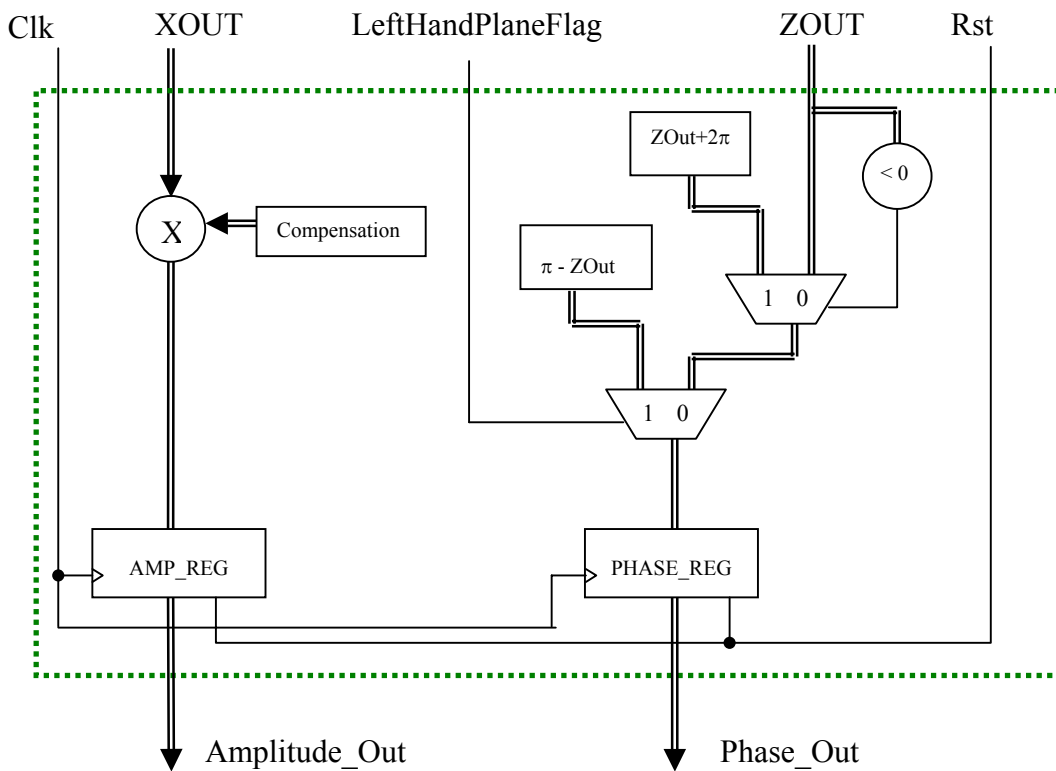
Figure 4.18 POST_CORDIC Structure
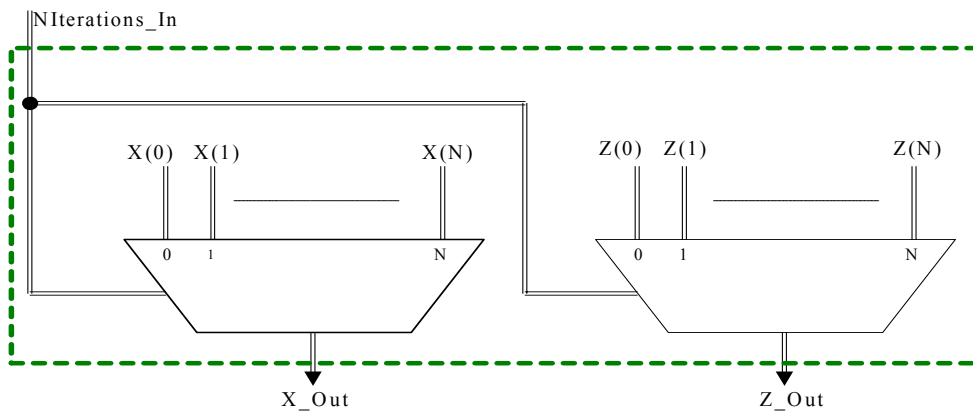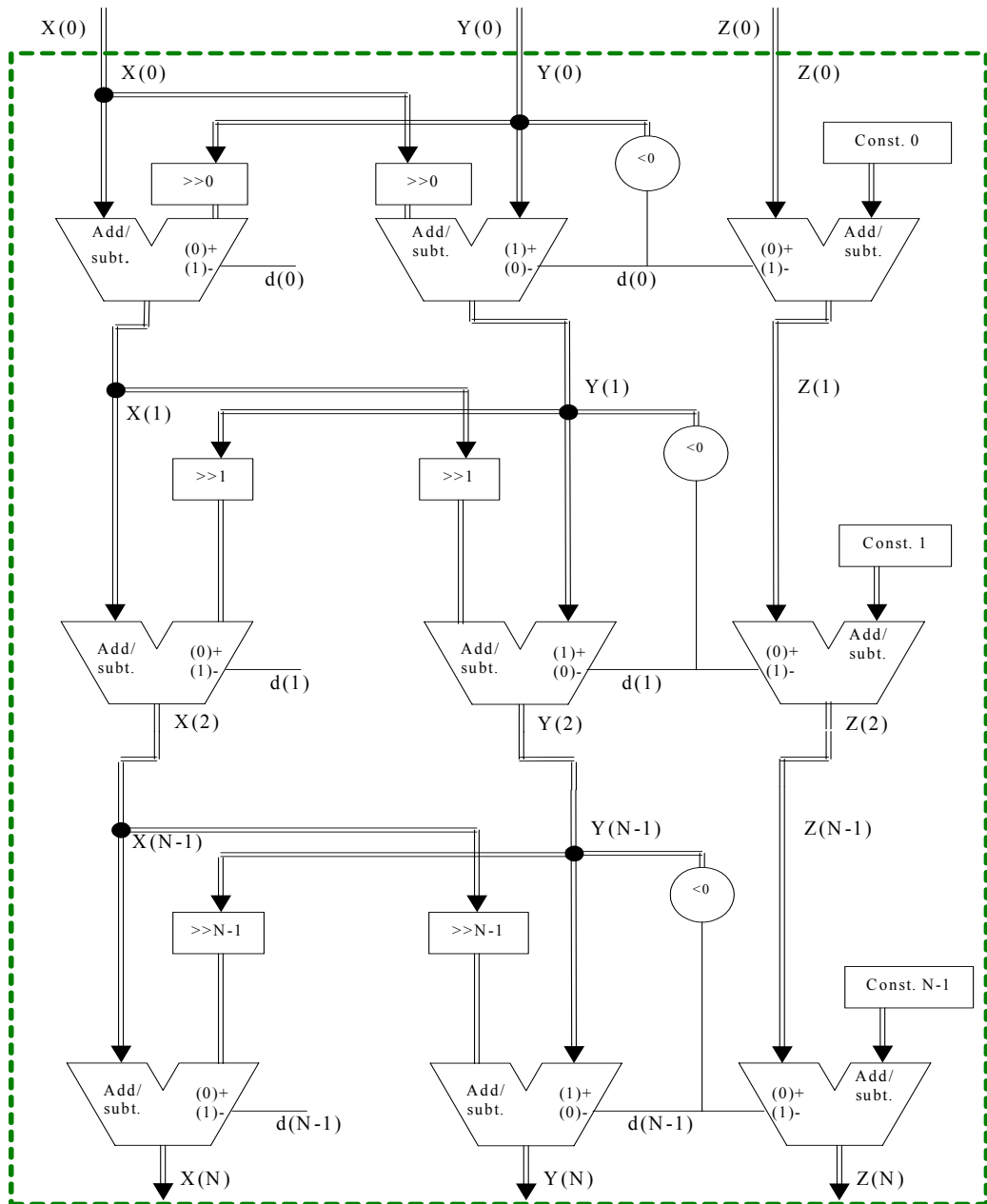
Figure 4.19 CORDIC_CORE Structure

4.3. Hardware Design of Generic ETSI OFDM STS Synchronizer

4.3.1. Coding of ETSI OFDM STS Synchronizer

ETSI OFDM STS Synchronizer seen in Figure 4.9 is coded using VHDL digital hardware description language.

Three sub-modules of ETSI OFDM STS Synchronizer are coded separately. These modules are connected to each other in STSSynchronizer, which is the top-level module. The SRCorrComplexMultiplier block is located inside the SlidingCorrelator module.

All modules are coded generically so that ETSI OFDM STS Synchronizer gains flexibility, which will allow us in the future to use it in a whole synchronizer block of an OFDM receiver.

VHDL codes of ETSI OFDM STS Synchronizer can be seen in Appendix B and Appendix C.

4.3.2. Simulation of ETSI OFDM STS Synchronizer

ETSI OFDM STS Synchronizer seen in Figure 4.9 was tested and simulated using Cadence Affirma NC VHDL Simulation Tool.

For each of the three sub-modules, separate functional simulations were performed. Simulation sections of each module can be seen in Figure 4.23, Figure 4.24, Figure 4.25 and Figure 4.26 respectively. After sub-modules were tested and verified individually, the following step, which is the top-level verification plan was realized for whole system.

Before giving a start to top-level functional verification, verification environment was constructed as follows:

1. Forming of input stimuli to be forced to ETSI OFDM STS Synchronizer:

An OFDM input stimulus was dumped to a matlab file from simulink OFDM model. The dumped OFDM stimuli have the characteristics summarized

in Table 4.2. As seen in this table, the dumped data is under perfect conditions. The STS part of this stimuli can be seen in Figure 4.5.

OFDM Stimuli Characteristics	
Modulation type	64QAM
Preamble type	ETSI_BROADCAST
SNR	210.0 dB
CO	0 ppm
CFO	0 Hz

Table 4.2 Input stimuli characteristics

2. **Writing matlab scripts to convert the data from real-complex format to binary format:**

As seen in Figure 4.5, the dumped OFDM data consists of real-complex numbers. This data had to be converted to binary format. Real and imaginary parts also had to be separated from each other in order to be ready for being forced to ETSI OFDM STS Synchronizer.

Separation of real and imaginary parts of OFDM stimuli was realized in matlab environment without using a script. The conversion of real data to binary format was realized with a simple matlab script so that real and imaginary parts were stored separately. OFDM input stimuli were written into a file. Since the only STS part of all OFDM stimuli was enough for testing the ETSI OFDM STS Synchronizer, the stimuli other than STS and some part of LTS samples was deleted from file. Input stimuli consist of 250 real and imaginary digital samples.

3. **Writing top-level test-bench**

In order to test ETSI OFDM STS Synchronizer, a top-level test-bench was written. It reads the input stimuli from a file, then applies these stimuli to ETSI OFDM STS Synchronizer and writes the outputs of the system into an output file.

4. Writing simple matlab script which plots amplitude and phase graphs of ETSI OFDM STS Synchronizer in matlab environment

In order to transfer outputs of the top-level simulations, a simple script was written. This script reads the outputs from the output file generated by top-level test-bench and then plots the amplitude and phase graphs of ETSI OFDM STS Synchronizer.

4.3.2.1.Top-level Functional Simulation Results of ETSI OFDM STS Synchronizer

At the end of top-level functional simulations, we could get the desired results in terms of amplitude and phase characteristics in comparison with the graphs seen in Figure 4.4. Owing to the fact that the generated OFDM stimuli change each time the OFDM simulink model runs (a random data generator generates all data), it is observed that the dumped stimuli used for hardware simulations were different from the matlab model data. Because of this reason, there have been small differences between the graphs of our results and the matlab model.

As explained in chapter 4.1.1, the goal is to detect the IB short OFDM symbol at the end of the preamble section based upon the sequence SB. In the ETSI BROADCAST case (see chapter D.1.5.4.7.1 and Figure 4.4), the correlation amplitude is the same for both SA and SB based section. However the correlation phase transition (from high to low or from low to high) allows distinguishing them.

Figure 4.20 represents the results achieved in our hardware implementation when the number of iterations realized in CORDIC block is equal to 10. When it's compared to Figure 4.4, it can be easily seen that the desired results for both amplitude and phase characteristics could be achieved. The first plateau and then first peak seen in Figure 4.20 a, correspond to SA based section while the second ones are related to SB based section respectively. As the first plateau seen in Figure 4.20 a continues, the phase also preserves its value (see Figure 4.20 b). Whenever the first plateau finishes, the phase of ETSI OFDM STS Synchronizer makes a sharp transition from high-to-low, which is what we expect (see Figure 4.4). Then, after the first peak seen in Figure 4.20 a, the second plateau begins; while the phase characteristic again preserves its value. The

phase jump is realized this time from low-to-high whenever the second plateau finishes. This is also same as what we expect (see Figure 4.4). These phase transitions allow us to distinguish SA and SB based sections from each other. So amplitude and phase characteristics should be processed and analyzed together to detect the short IB symbol at the end of STS section.

First ~180 output OFDM samples are the outputs for the ETSI BROADCAST STS input samples while first ~18 output samples are zero since the meaningful correlation begins after 17 clock cycles later from the beginning of STS sliding correlation process (see 4.2.1.2). After ~180th sample, ETSI OFDM STS Synchronizer begins to output the LTS part related results, which we are not interested in. Note that the amplitude and phase outputs for LTS part of preamble section are not so meaningful since the correlation window width of ETSI OFDM STS Synchronizer is only for 16 samples. Anyway the LTS part is not used to detect where the OFDM symbol boundaries are.

As mentioned before, CORDIC block runs basically a combinational iterative algorithm to implement the needed vector rotations. As the number of iterations realized in CORDIC block increases, the achieved results go better in comparison with the previous iteration. For instance, Figure 4.21 represents the amplitude and phase characteristics at the output of ETSI OFDM STS Synchronizer for NRIterations = 2, while Figure 4.22 represents the ones for NRIterations = 5. Since the results achieved for 10 iterations were quite satisfactory, the number of iterations was limited to 10 in CORDIC block of our implementation. Note that our design does not support more than 10 iterations and it's needed to add more logic for this.

Figure 4.23, Figure 4.24, Figure 4.25, Figure 4.26 and Figure 4.27 show the simulation sections of each block and the top-level of ETSI OFDM STS Synchronizer.
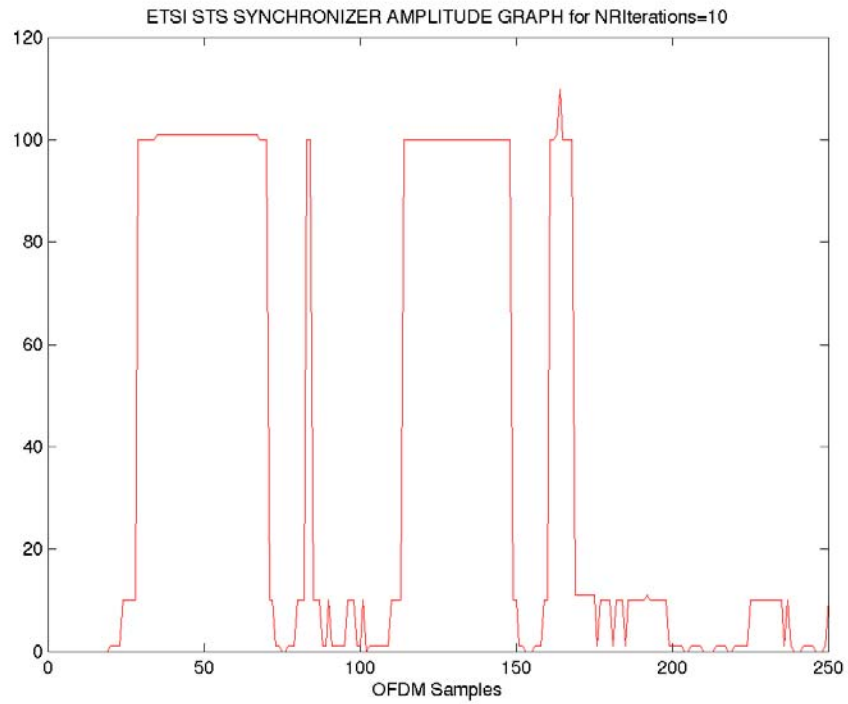
(a)
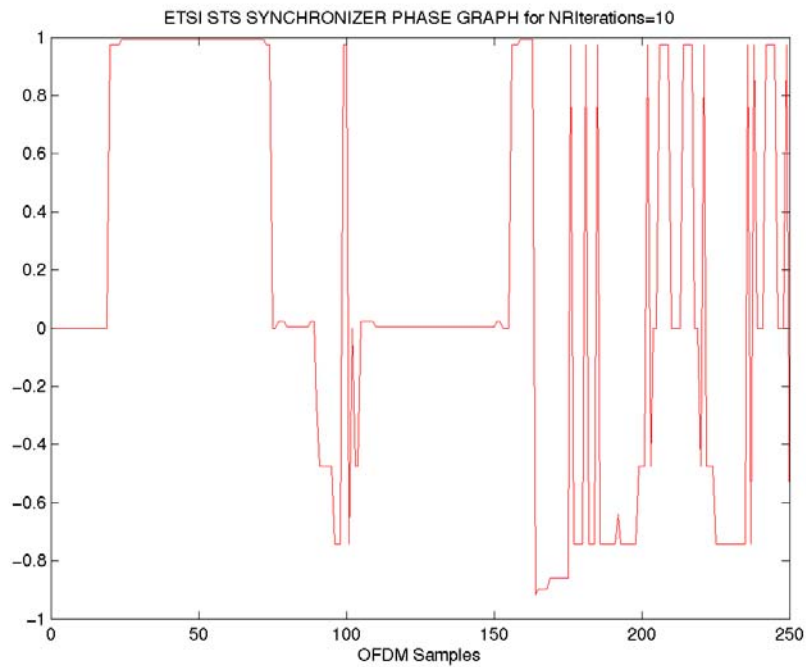
(b)

Figure 4.20 ETSI OFDM STS Synchronizer output graphs for ETSI BROADCAST
Preamble for NRIterations=10 in CORDIC block: (a) Amplitude (b) Phase
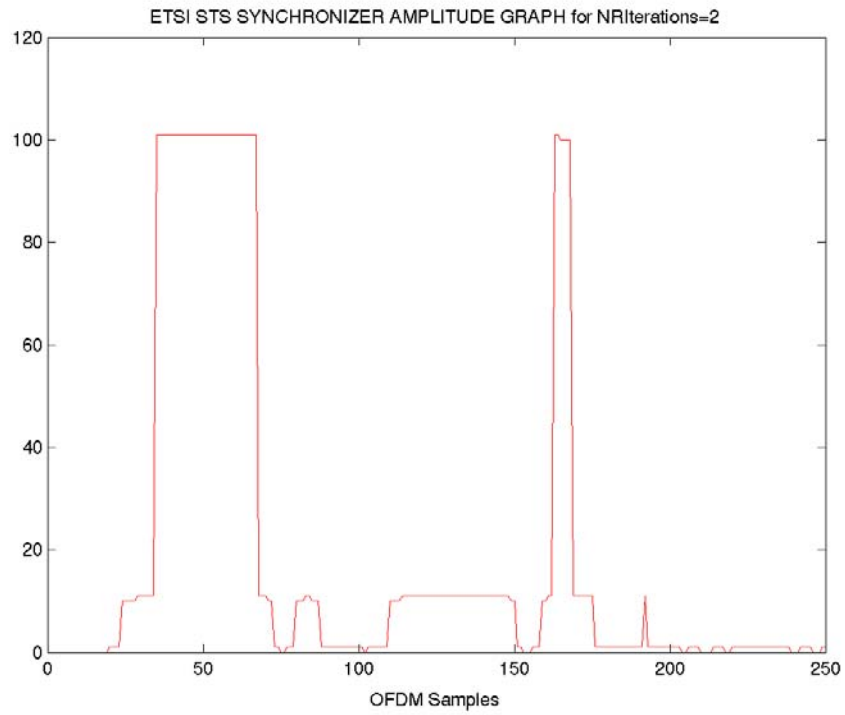
(a)
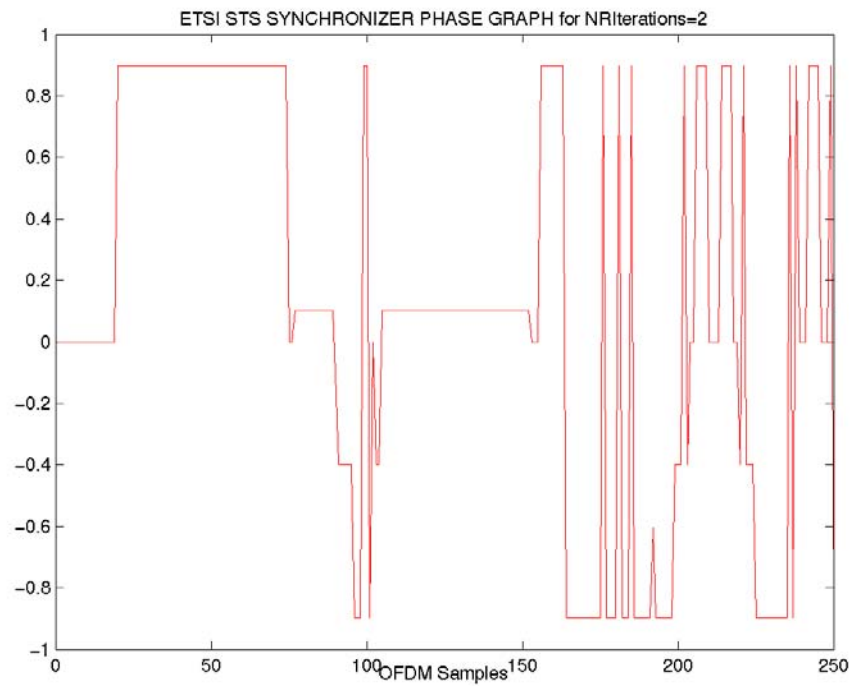
(b)

Figure 4.21 ETSI OFDM STS Synchronizer output graphs for ETSI BROADCAST
Preamble for NRIterations = 2 in CORDIC block: (a) Amplitude (b) Phase
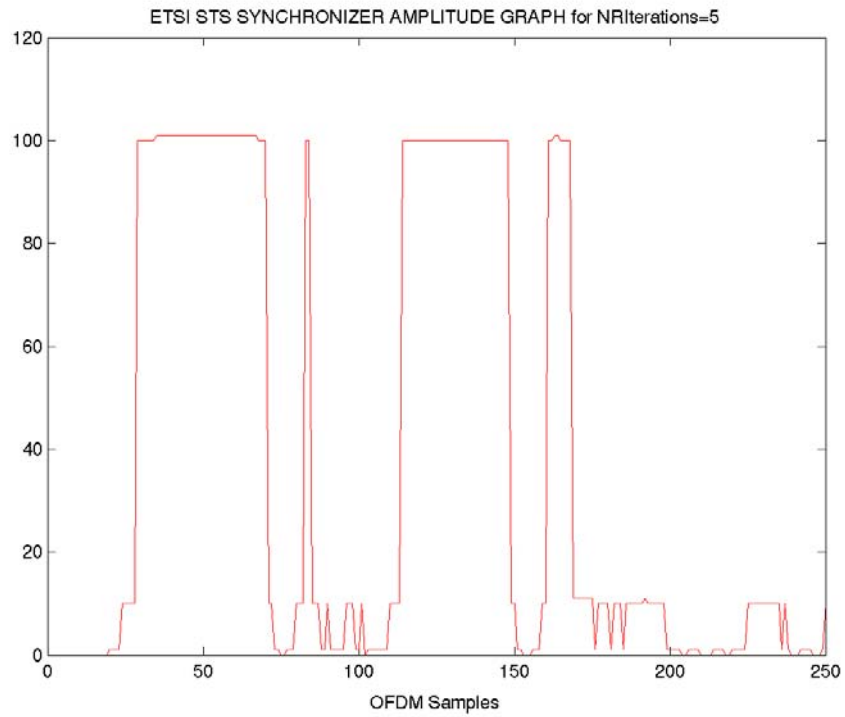
(a)
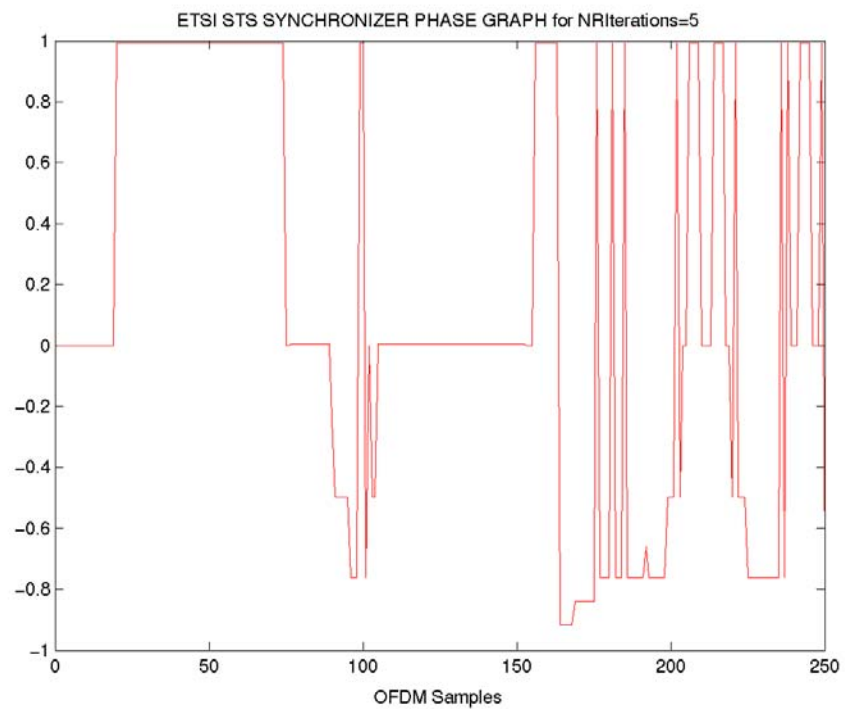
(b)

Figure 4.22 ETSI OFDM STS Synchronizer output graphs for ETSI BROADCAST
Preamble for NRIterations = 5 in CORDIC block: (a) Amplitude (b) Phase
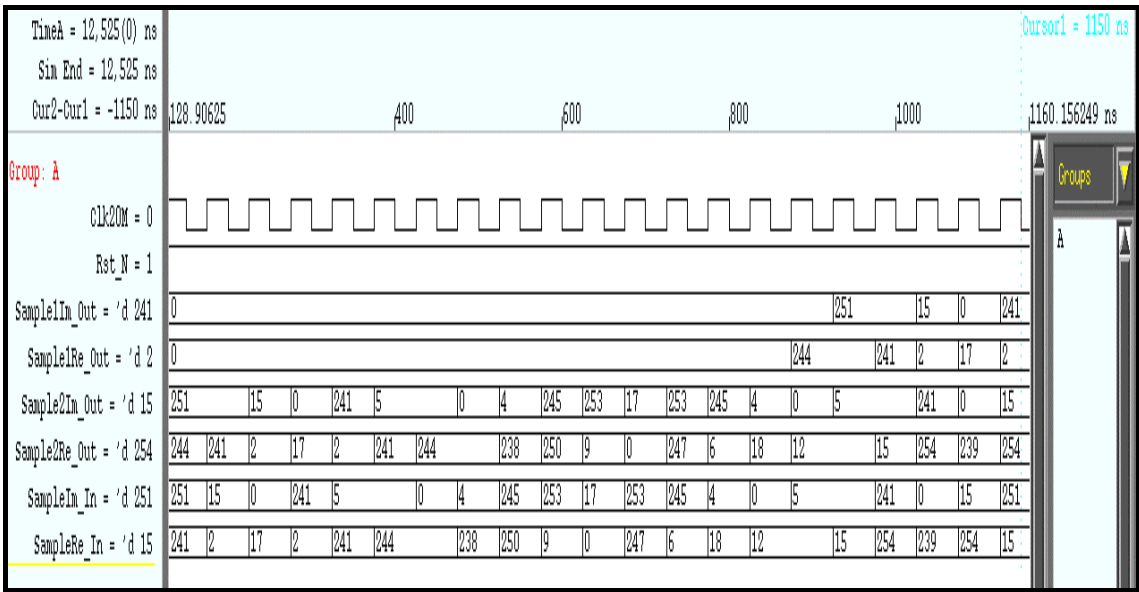
Figure 4.23 Simulation section of SlidingShiftRegister block
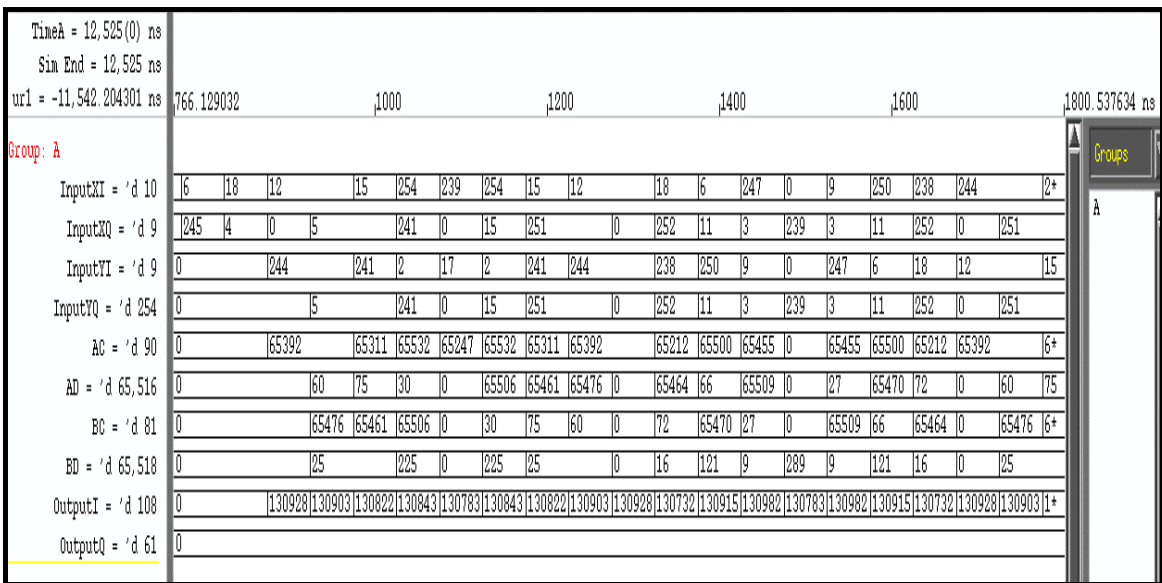
Figure 4.24 Simulation section of SRCorrComplexMultiplier sub-block in SlidingCorrelator
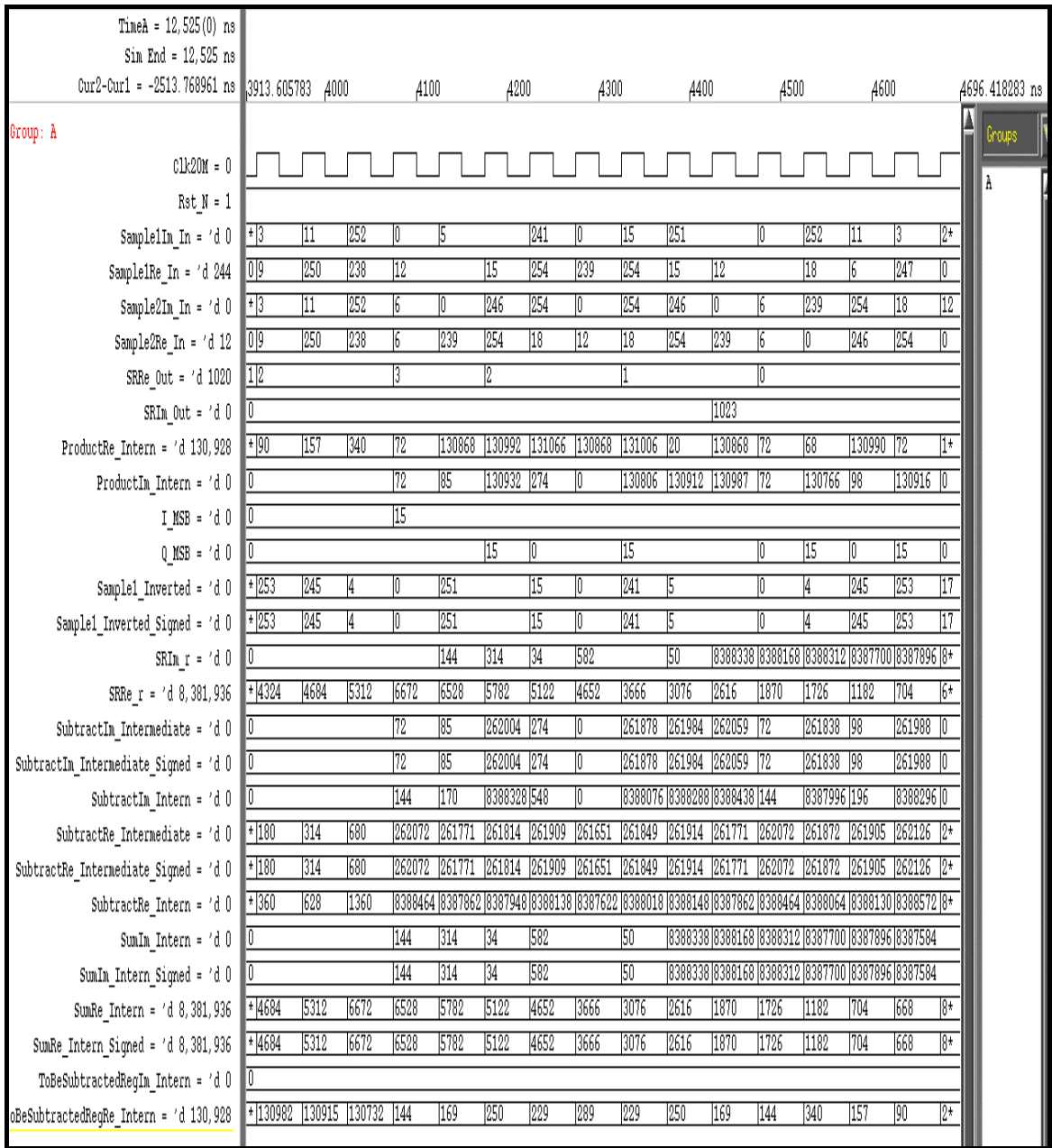
Figure 4.25 Simulation section of SlidingCorrelator block
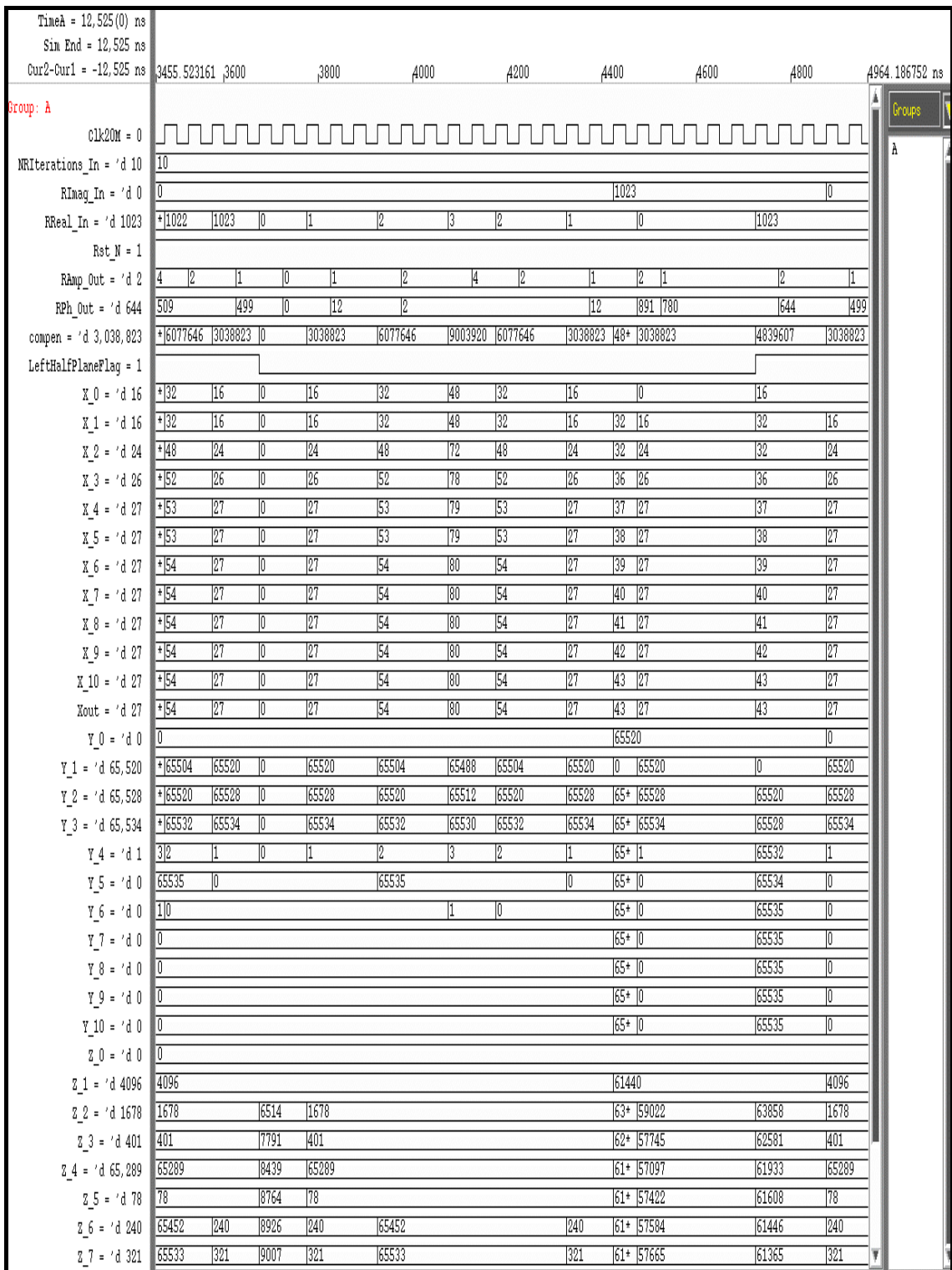
Figure 4.26 Simulation section of CORDIC block
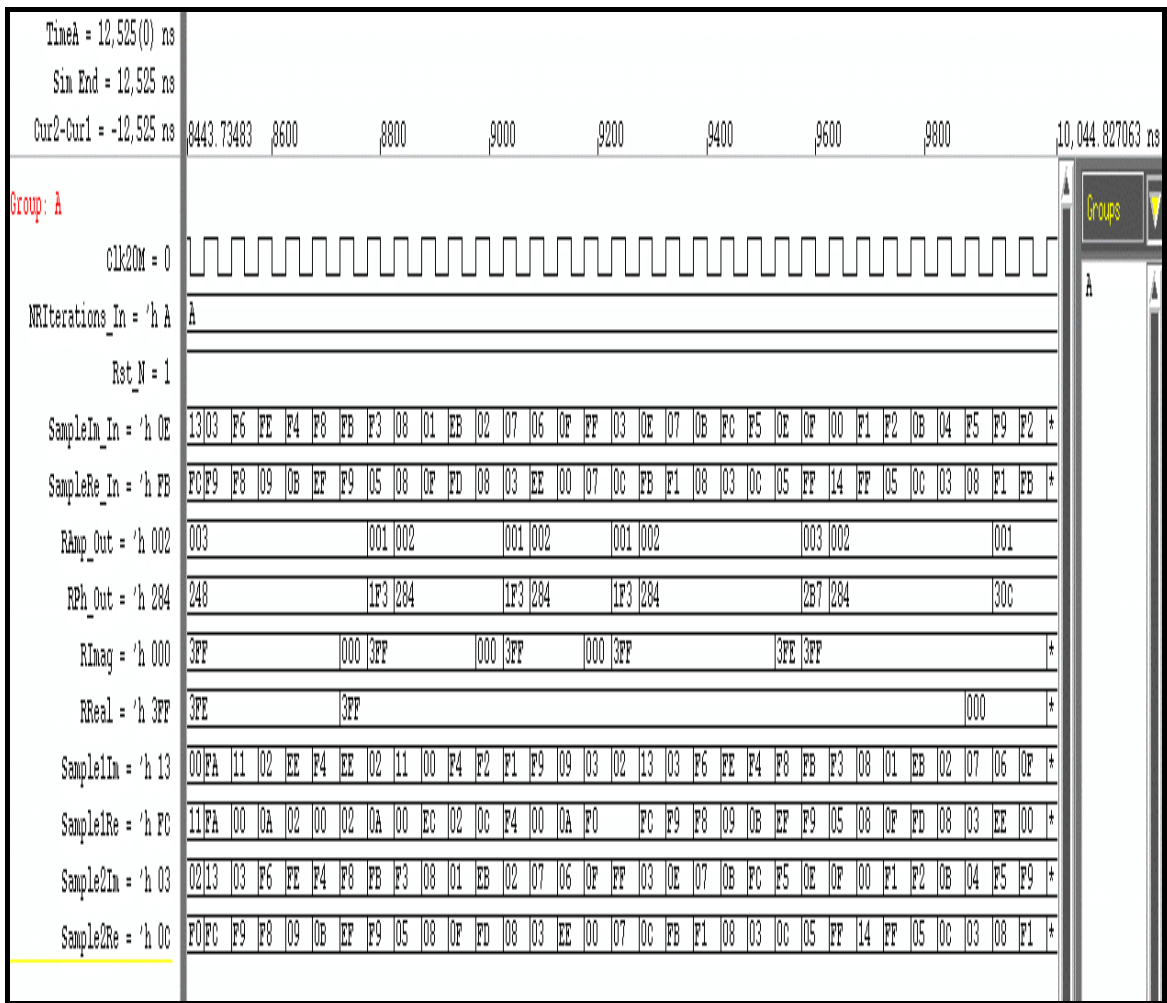
Figure 4.27 Top-level simulation section of STSSynchronizer

4.3.3. Synthesis of ETSI OFDM STS Synchronizer IP and Gate-level Simulations

4.3.3.1. Synthesis

As mentioned before, ETSI OFDM STS Synchronizer was not manufactured and resulted as a generic "IP", which means it's ready to be synthesized, adapted and used in an OFDM receiver. But although it was not manufactured, it was necessary to synthesize it to see whether it exists any problems or not in terms of static timing analysis and also to see its maximum processing speed and the area which it covers.

Synthesis was realized with Synopsys Design Analyzer Synthesis tool in CMOS 0.13μm technology using Virtual Silicon Technology (VST) library cells. It was written a synthesis script, which included all necessary constraints for the synthesis. In order to get more efficient results in terms of high-level optimization of both timing and area, "Synopsys DesignWare Foundation Synthetic Library" components that are a collection of reusable intellectual property blocks were used by adding necessary constraints into the synthesis script.

As a synthesis methodology, "top-down" synthesis way was used since it provides a push-button approach and our design is not so large. All constraints were applied to STSSynchronizer, which is the top-level block.

Since it was run a preliminary synthesis, it was not constrained a wire load model and used the default one that was assigned by the synthesis tool. Top-level schematic view of synthesized ETSI OFDM STS Synchronizer IP is shown in Figure 4.28 and schematic views of each of sub-modules can be seen in Figure 4.29, Figure 4.30, Figure 4.31, Figure 4.32, Figure 4.33 respectively.
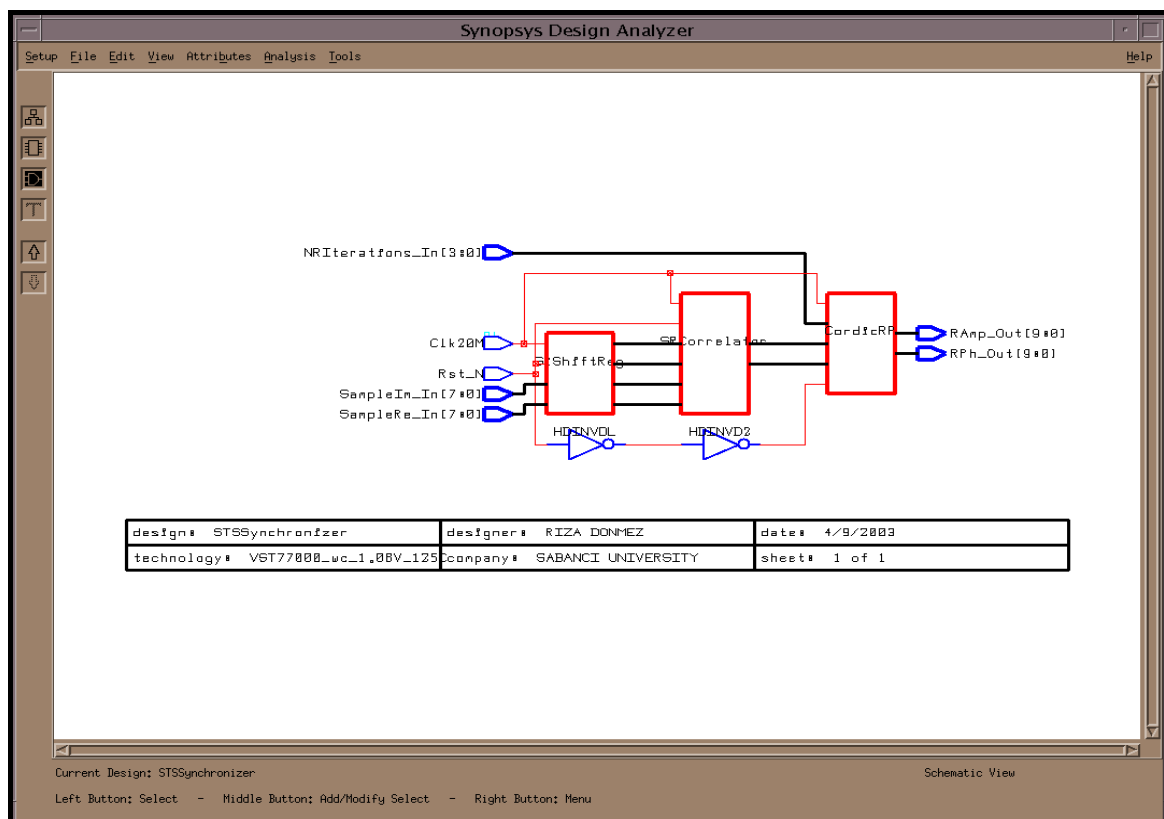
Figure 4.28 Top-level schematic view of synthesized ETSI OFDM STS Synchronizer
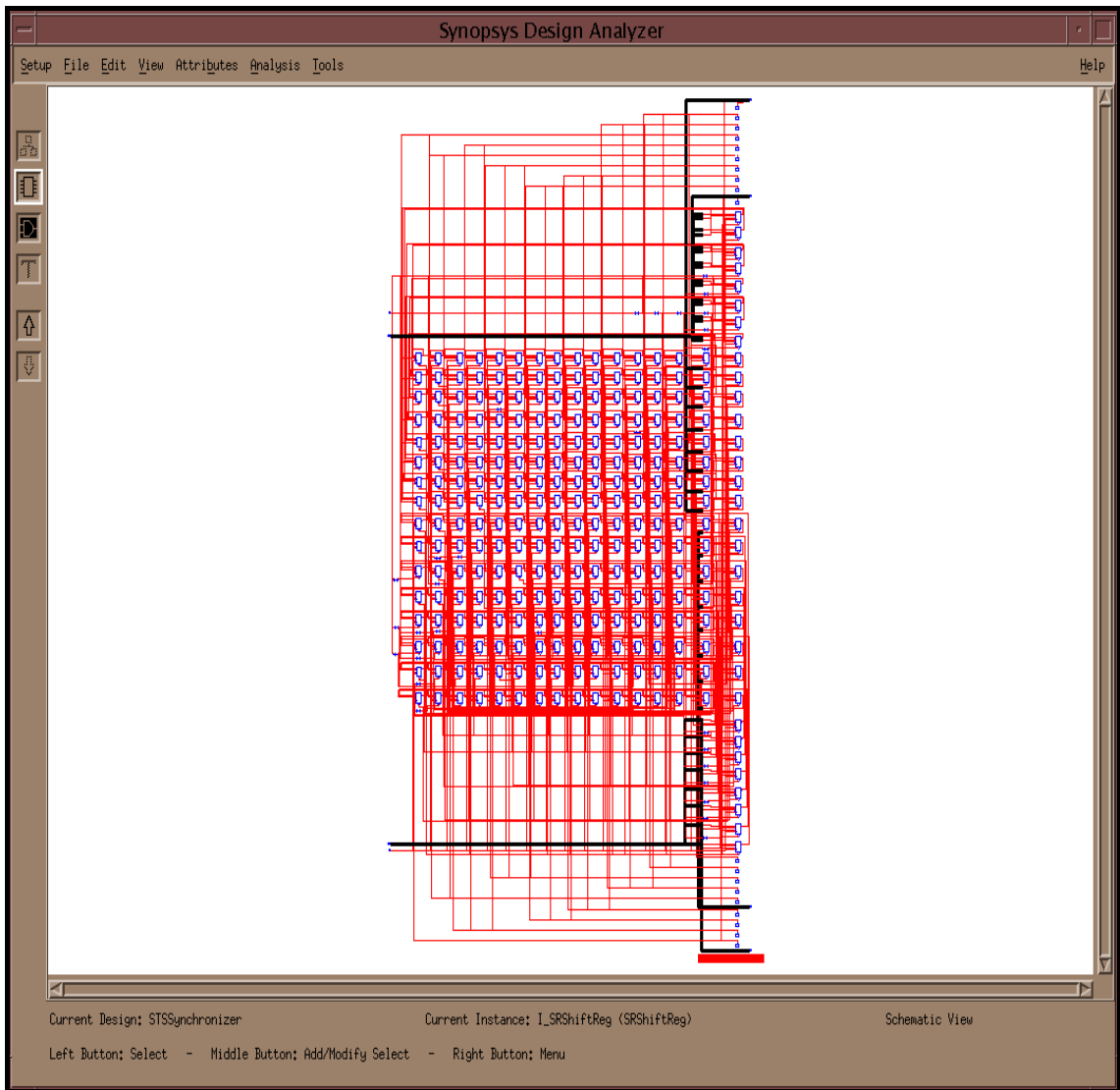
Figure 4.29 Schematic view of synthesized SlidingShiftRegister block
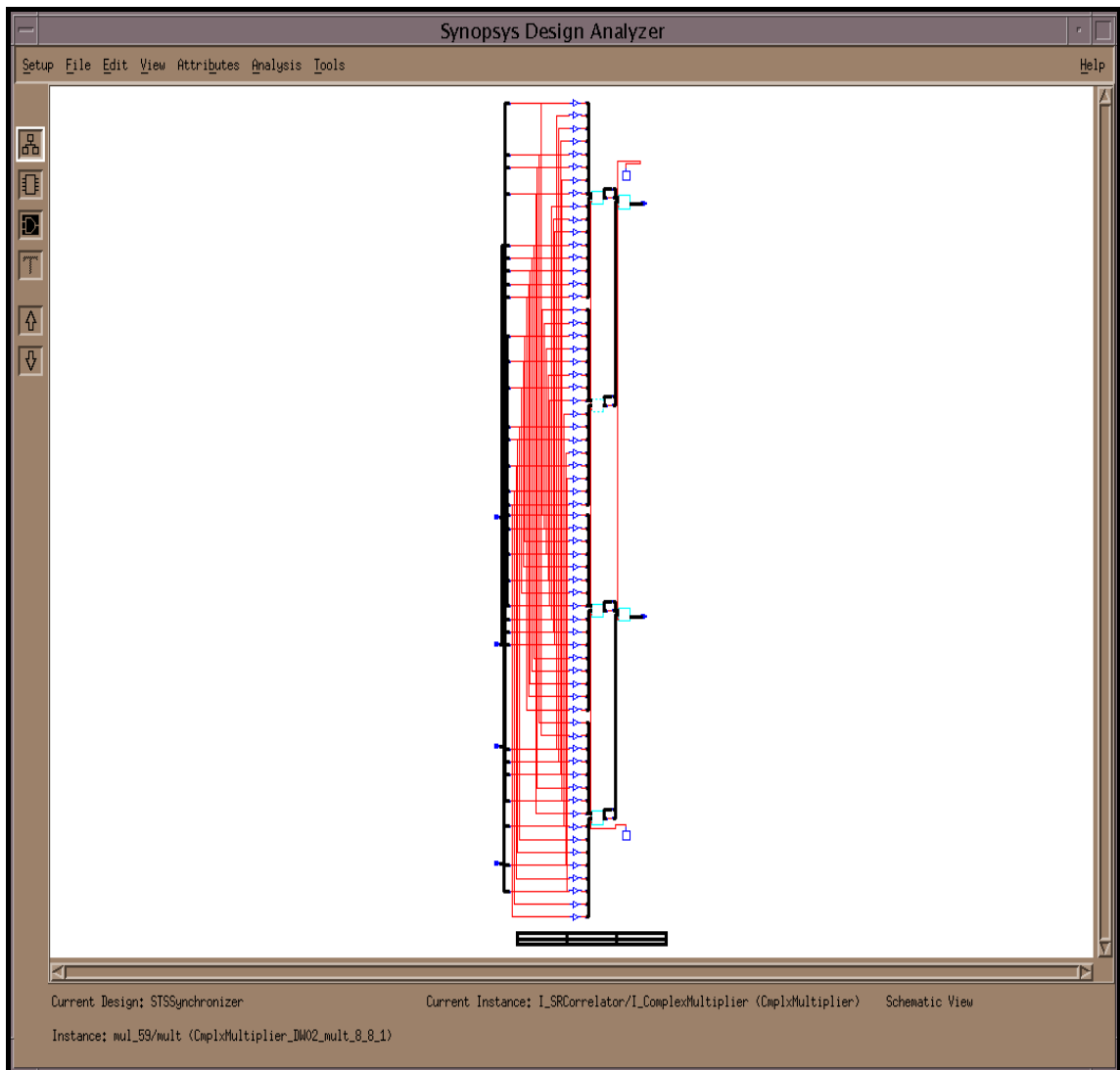
Figure 4.30 Schematic view of synthesized SRCorrComplexMultiplier block
instantiated in SlidingCorrelator block
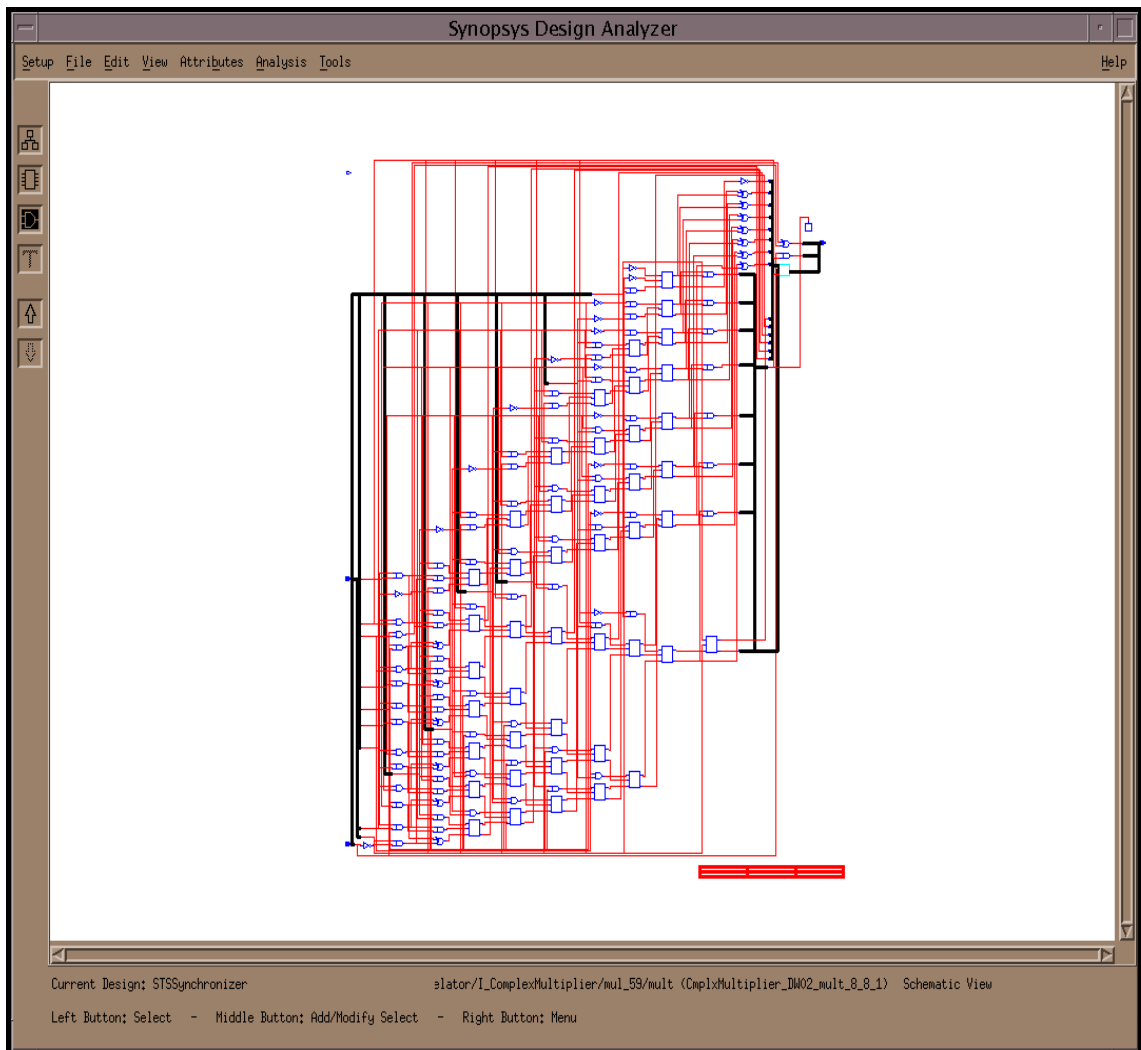
Figure 4.31 Schematic view of a DesignWare multiplier component instantiated in SRCorrComplexMultiplier block
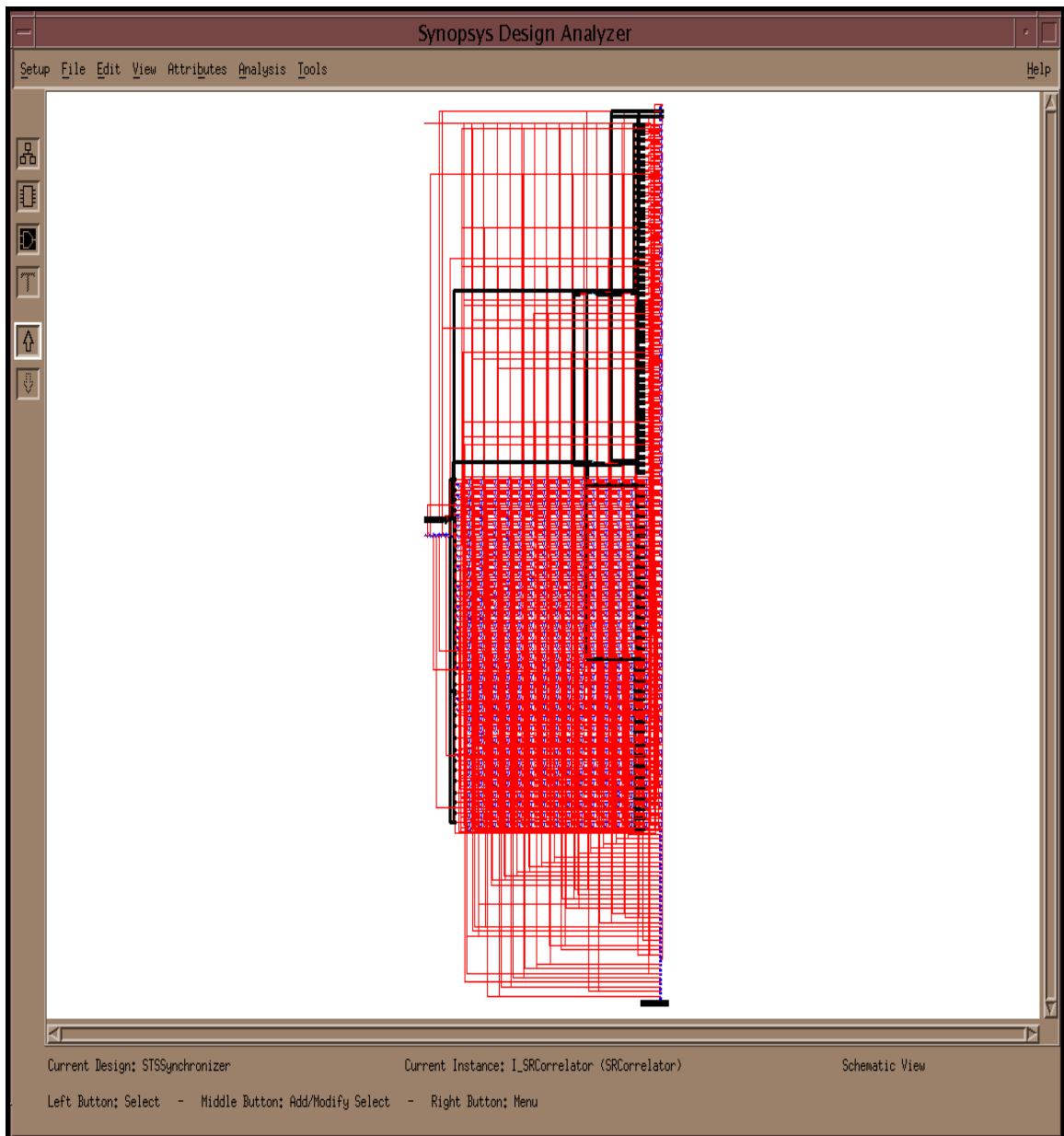
Figure 4.32 Schematic view of synthesized SlidingCorrelator block
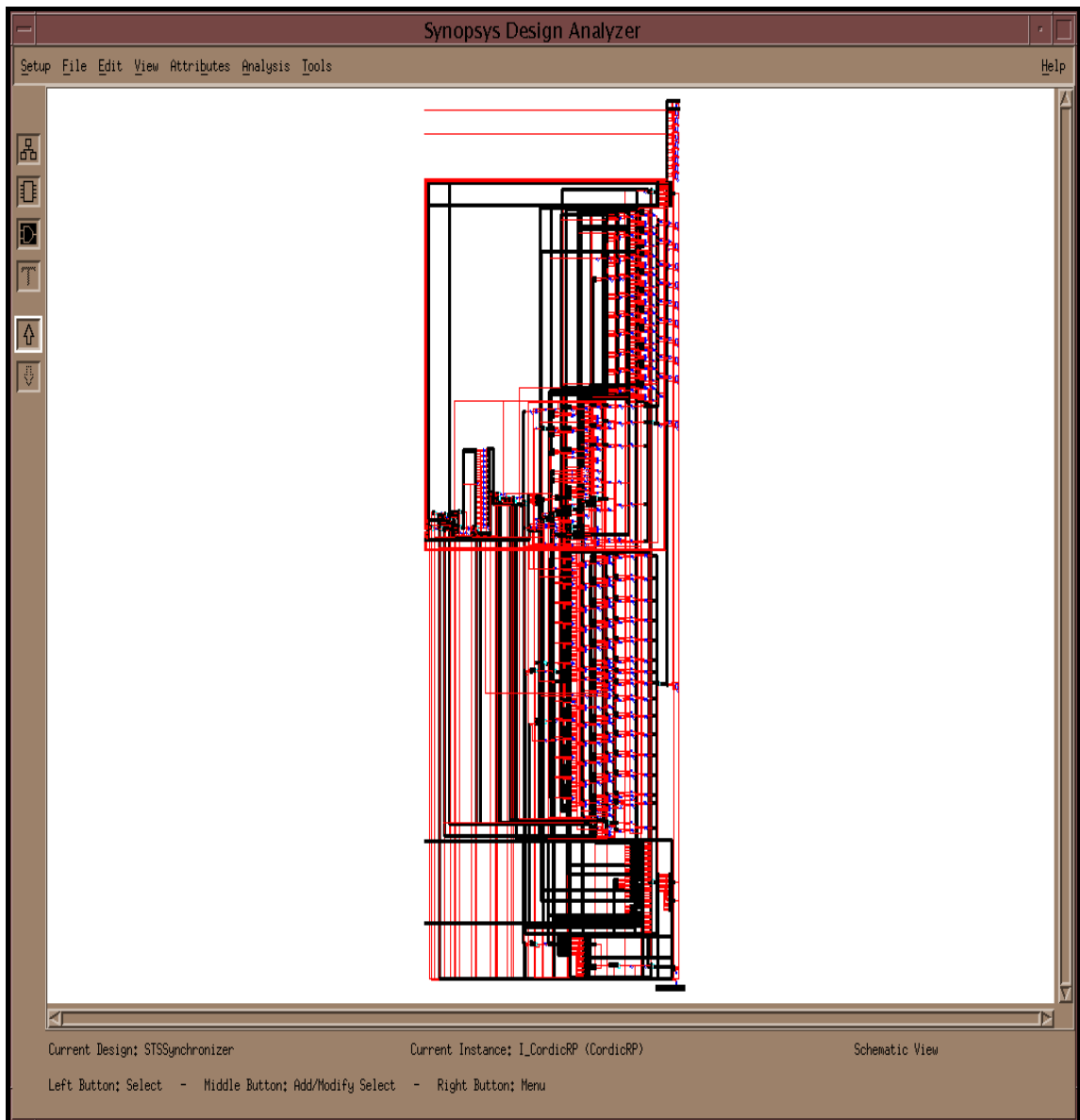
Figure 4.33 Schematic view of synthesized CORDIC block

Since the ETSI Hiperlan/2 PHY layer standards requires a 20 MHz sampling frequency at receiver (please see Table D.6), synthesis was run first setting the system operation frequency to 20 MHz. It was seen that the system had not any problem at this operation frequency in terms of critical timing issues and we did not get any violations. Area results achieved for this synthesis can be seen in Table 4.3. Then, as the second step, it was aimed to catch the maximum operation frequency of this IP. After several synthesis trials with "low effort" constraint, at the end of static timing analysis, it was seen that the maximum operation speed for our IP was 50 MHz and CORDIC block was at the critical path since its algorithm was implemented using mostly combinational logic. After this clock frequency, system begins to produce setup time violations. This result is exactly same as what we expect. To reach speeder frequencies than 50 MHz, it's necessary to implement a pipelined architecture inside the CORDIC block, which increases both latency and throughput of the system. This is not aimed to reach the possible maximum speeds in our implementation since we accept ETSI Hiperlan/2 OFDM standard and parameters. But IP is ready to run up to 50 MHz speed, maybe more after a serious synthesis trials with correct constraints. Synthesis results for 50 MHz in terms of area are in Table 4.4. Power estimation reports given by Synopsys Design Analyzer for 20MHz and 50MHz-operating frequencies are in Table 4.5 and in Table 4.6 respectively. Equation to calculate the approximate power consumption in the CMOS 0.13μm technology-VST77000 databook is given by

$$P_{diss} = (E_{rise} + E_{fall} + (C_{fanout}.V^2)).F_{switching} + P_{static} \qquad (4.16)$$

where:

- P_{diss} is the power dissipation of the gate (in pW).
- E_{rise} is the energy for the rising transition (in pJ).
- E_{fall} is the energy for the falling transition (in pJ).
- C_{fanout} is the output load capacitance (in pF); the number of loads multiplied by the value for a standard load.
- V is the supply voltage.
- $F_{switching}$ is the switching frequency of the transition (in mHz).
- P_{static} is the static power dissipation of the library cell (in pW).

Block		SlidingShiftRegister	SlidingCorrelator	CORDIC	STSSynchronizer (Total)
Combinational Area	μ^2	177.984009	14020.992188	23499.062500	37706.683594
	Gates	34	2707	4533	7273
Noncombinational Area	μ^2	12690.431641	27527.037109	933.119995	41150.585938
	Gates	2173	5310	180	7938
Total Cell Area	μ^2	12868.416016	41548.031250	24432.181641	78857.265625
	Gates	2207	8017	4713	15211
Number of Flip-Flops		272	590	20	882

Table 4.3 Area results of synthesis of ETSI OFDM STS Synchronizer for 20MHz operation frequency

Block		SlidingShiftRegister	SlidingCorrelator	CORDIC	STSSynchronizer (Total)
Combinational Area	μ^2	177.984009	14033.087891	53122.183594	67341.890625
	Gates	34	2707	10247	12990
Noncombinational Area	μ^2	12690.431641	27359.132812	933.119995	41162.683594
	Gates	2173	5278	180	7940
Total Cell Area	μ^2	12868.416016	41572.218750	54055.304688	108504.578125
	Gates	2207	7985	10427	20930
Number of Flip-Flops		272	590	20	882

Table 4.4 Area results of synthesis of ETSI OFDM STS Synchronizer for 50MHz operation frequency

Operating Conditions		wc_1.08V_125C			
Global Operating Voltage (V)		1.08			
Library		VST77000_wc_1.08V_125C			
Power Consumption Estimation					
Block		SlidingShiftRegister	SlidingCorrelator	CORDIC	STSSynchronizer (Total)
Cell Internal Power	µW	99.1853	209.6462	10.8812	319.7526
Net Switching Power	µW	6.9607	20.1876	2.9840	73.8889
Total Dynamic Power = CIP + NSP	µW	106.1460	229.8338	13.8652	393.6414
Cell Leakage Power (Static Power)	µW	14.0214	61.3368	52.4857	127.8588

Table 4.5 Power consumption estimation for 20MHz operation frequency

Operating Conditions		wc_1.08V_125C			
Global Operating Voltage (V)		1.08			
Library		VST77000_wc_1.08V_125C			
Power Consumption Estimation					
Block		SlidingShiftRegister	SlidingCorrelator	CORDIC	STSSynchronizer (Total)
Cell Internal Power	µW	248.4968	523.9656	28.2263	800.7877
Net Switching Power	µW	17.4481	50.3458	7.3146	184.5302
Total Dynamic Power = CIP + NSP	µW	265.9449	574.3114	35.5408	985.3179
Cell Leakage Power (Static Power)	µW	14.0473	60.8577	109.4740	184.3936

Table 4.6 Power consumption estimation for 50MHz operation frequency

4.3.3.2. Gate-level Simulations

After synthesis of the IP, the net-list of the synthesized design was saved in verilog format; then it was generated the sdf (standard delay file) file needed for the gate-level simulations. As mentioned before, our design was resulted as an IP and no back-end activities were done such place-and-route process. This is why we did not have a real sdf file, which is normally produced after the place-and-route operation. So gate-level simulations were realized using the sdf file generated by Synopsys Design Analyzer, which included the estimated timings for each of library elements.

Gate-level simulation results seen in Figure 4.35 a and Figure 4.35 b are same as the ones we got during functional simulations (see Figure 4.20). A section of gate-level simulations of STSSynchronizer is shown in Figure 4.34.
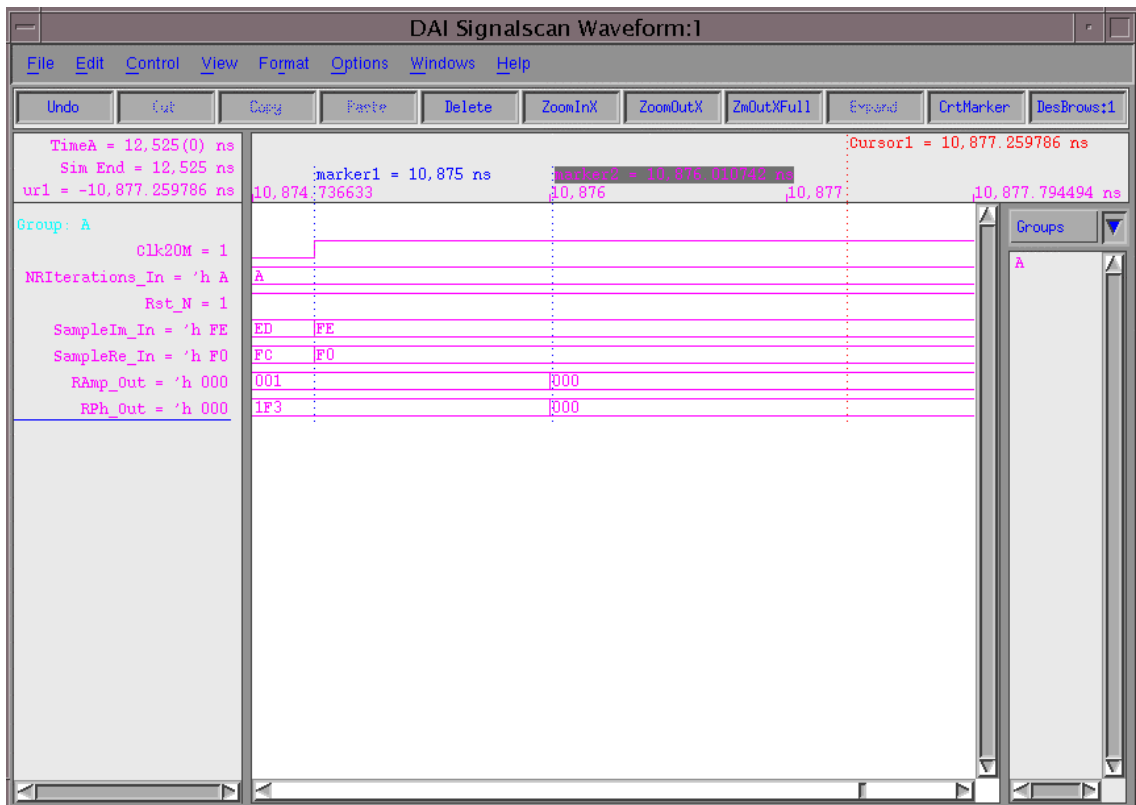
Figure 4.34 Gate-level simulation section of STSSynchronizer
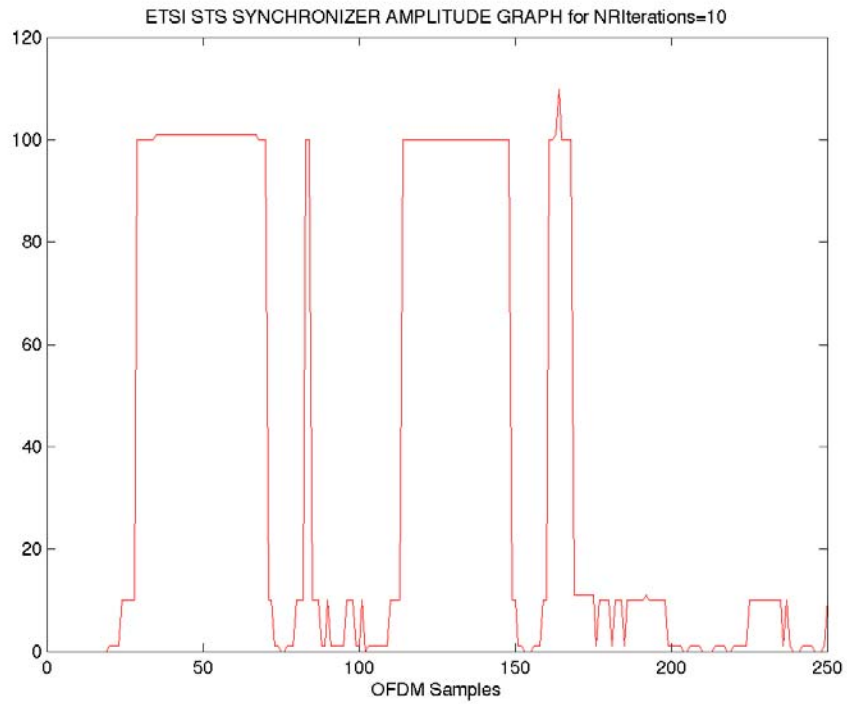
(a)
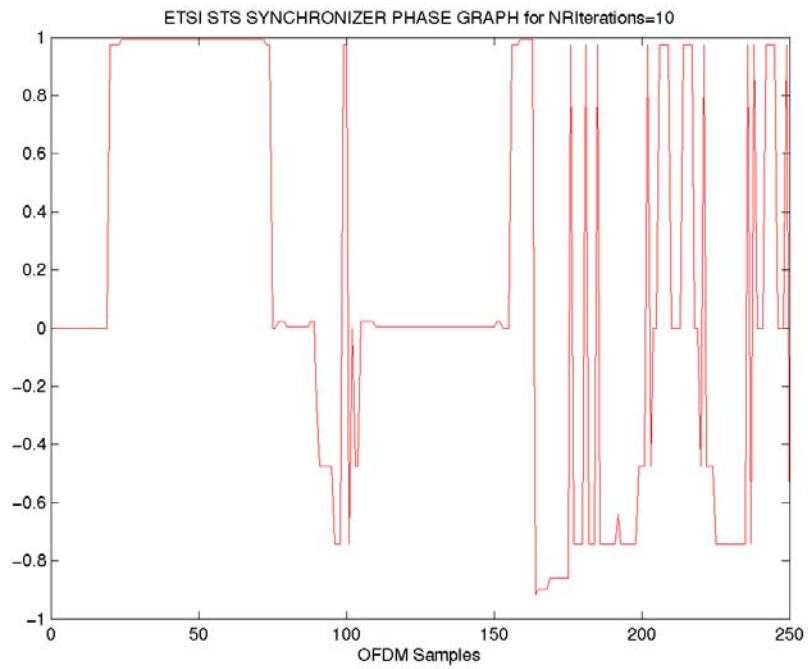
(b)

Figure 4.35 ETSI OFDM STS Synchronizer output graphs (gate-level) for ETSI BROADCAST
Preamble for NRIterations=10 in CORDIC block: (a) Amplitude (b) Phase

5.   CONCLUSIONS

This thesis has presented the design, digital implementation, functional and gate-level verification and synthesis of ETSI OFDM STS Synchronizer IP in digital CMOS 0.13μm technology using VST libraries. Physical realization of the Symbol Synchronizer has not been performed, but it is ready to be integrated as a part of whole synchronizer, which implements all needed synchronization tasks in an ETSI OFDM receiver.

The architecture of the ETSI OFDM STS Synchronizer is based on sliding correlation methodology. A serial approach is reflected to whole design instead of parallel, which decreases the total area reducing the number of arithmetic functional blocks used in the design like multipliers.

The design consists of three main blocks: Sliding Shift Register block, which provides the Sliding Correlator block with the correct data to be correlated; Sliding Correlator block, which realizes the main functionality of the IP, sliding correlation of OFDM samples and includes the Complex Multiplier block; CORDIC block, which provides the amplitude and phase values of correlated OFDM samples.

At the end of functional and gate-level verifications of symbol synchronizer we designed, we could achieve very satisfactory results: Amplitude and phase characteristics of the slightly correlated received samples were very similar to simulink matlab model's ones. At the output of designed synchronizer, amplitude and phase characteristics of the correlated received samples allowed us to detect the OFDM symbol. Amplitude and phase transitions of the correlated received STS symbols were the same as what we expected. As a result, ETSI OFDM symbols can be easily detectable by the hardware we proposed and designed in this thesis.

Although the current standard requires 20 MHz operation frequency, ETSI OFDM STS Synchronizer IP is capable to work up to 50 MHz. This means that it can be easily adapted to the future designs up to this speed. The CORDIC block is at the critical path

in terms of design timing since it has a huge combinational logic to implement the iterative CORDIC algorithm. CORDIC block should be redesigned with a pipelined architecture in order to increase the operation frequency higher than 50 MHz.

To summarize, the proposed and digitally designed ETSI OFDM STS Symbol Synchronizer IP is capable to correlate received ETSI OFDM symbols correctly and to find out where ETSI OFDM symbol boundaries are. The achieved results are satisfactory and can be used as a starting point for possible future works.

Based on the finding of this thesis, for future works, the following issues may be proposed:

- In our work, we made our design considering a perfect media and we did not consider impairments caused by CFO (Carrier Frequency Offset), CO (Clock Offset), AWGN, phase errors and channel effects. First, our design can be tested under these effects, then taking these effects into account, it can be developed and designed a whole ETSI OFDM Synchronizer that deals with all of these impairments, including the ETSI OFDM STS Symbol Synchronizer IP as well.
- A complete digital OFDM receiver can be designed, implemented and produced including the whole synchronizer block.

APPENDIX A: SCHEMATICS OF WHOLE IP
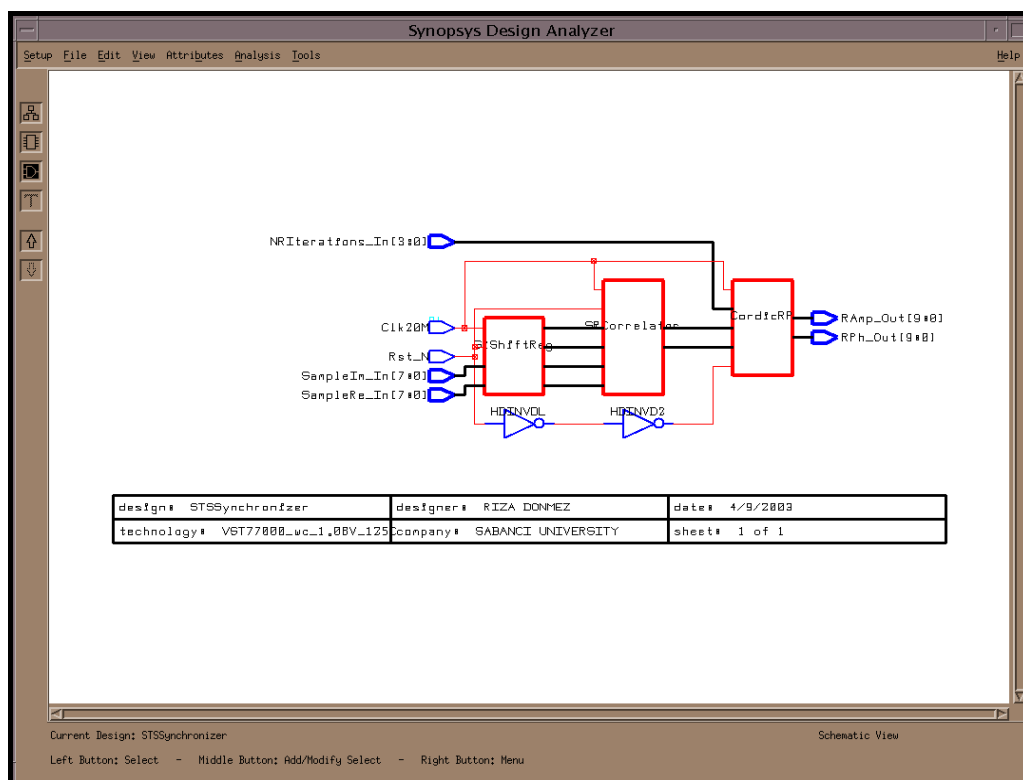
Figure A.1 Top-level schematic view of synthesized ETSI OFDM STS Synchronizer
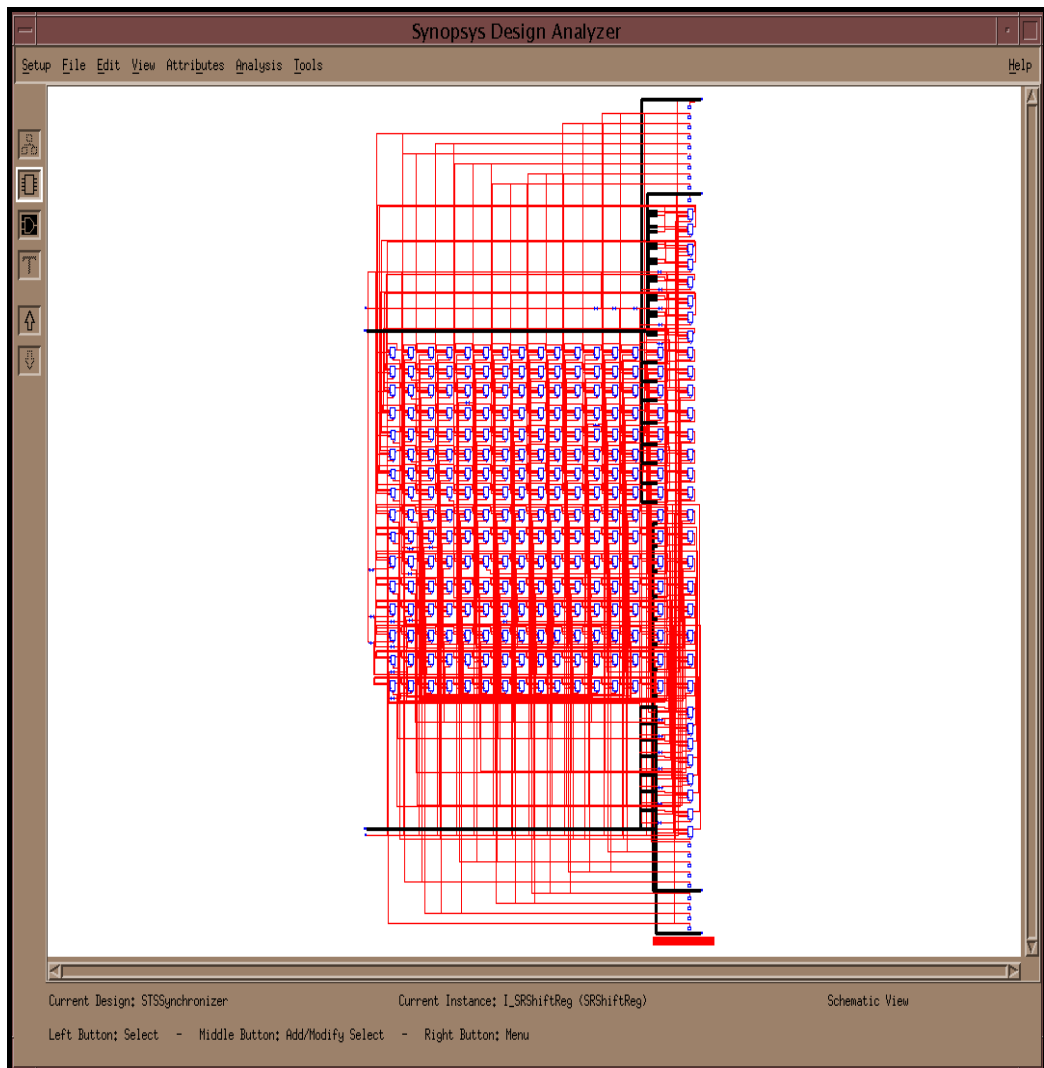
Figure A.2 Schematic view of synthesized SlidingShiftRegister block
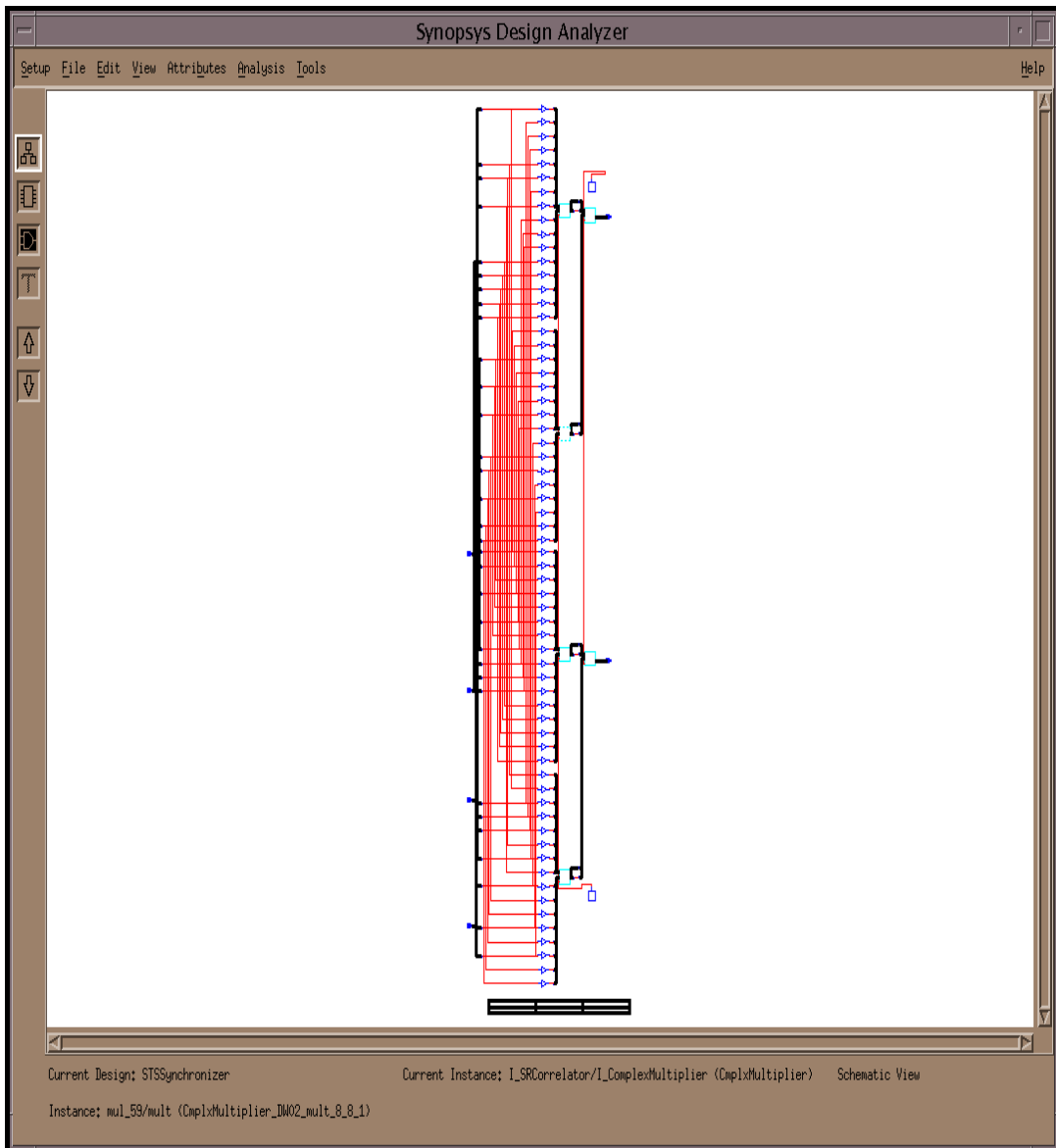
Figure A.3 Schematic view of synthesized SRCorrComplexMultiplier block instantiated in SlidingCorrelator block
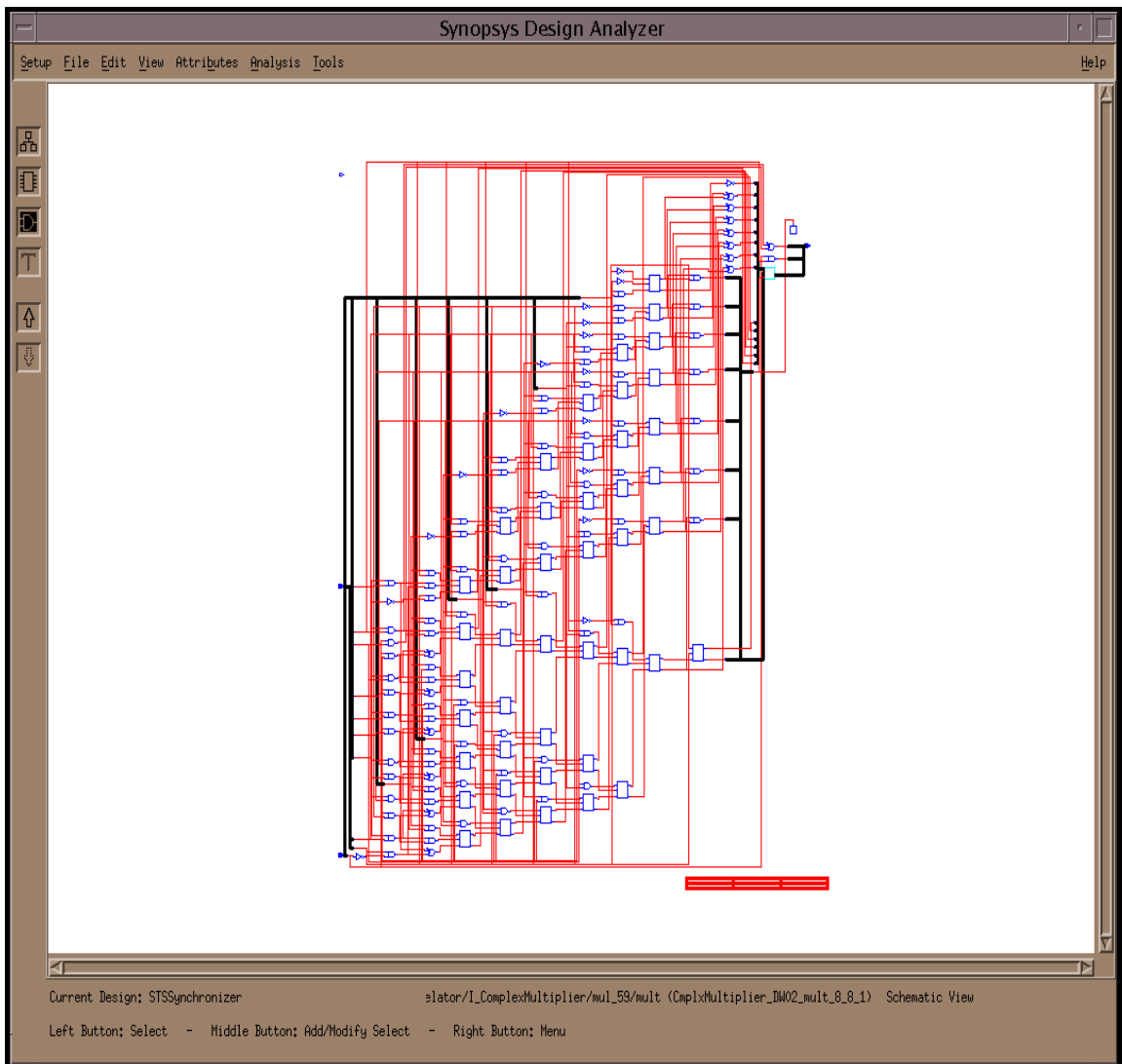
Figure A.4 Schematic view of a DesignWare multiplier component instantiated in SRCorrComplexMultiplier block
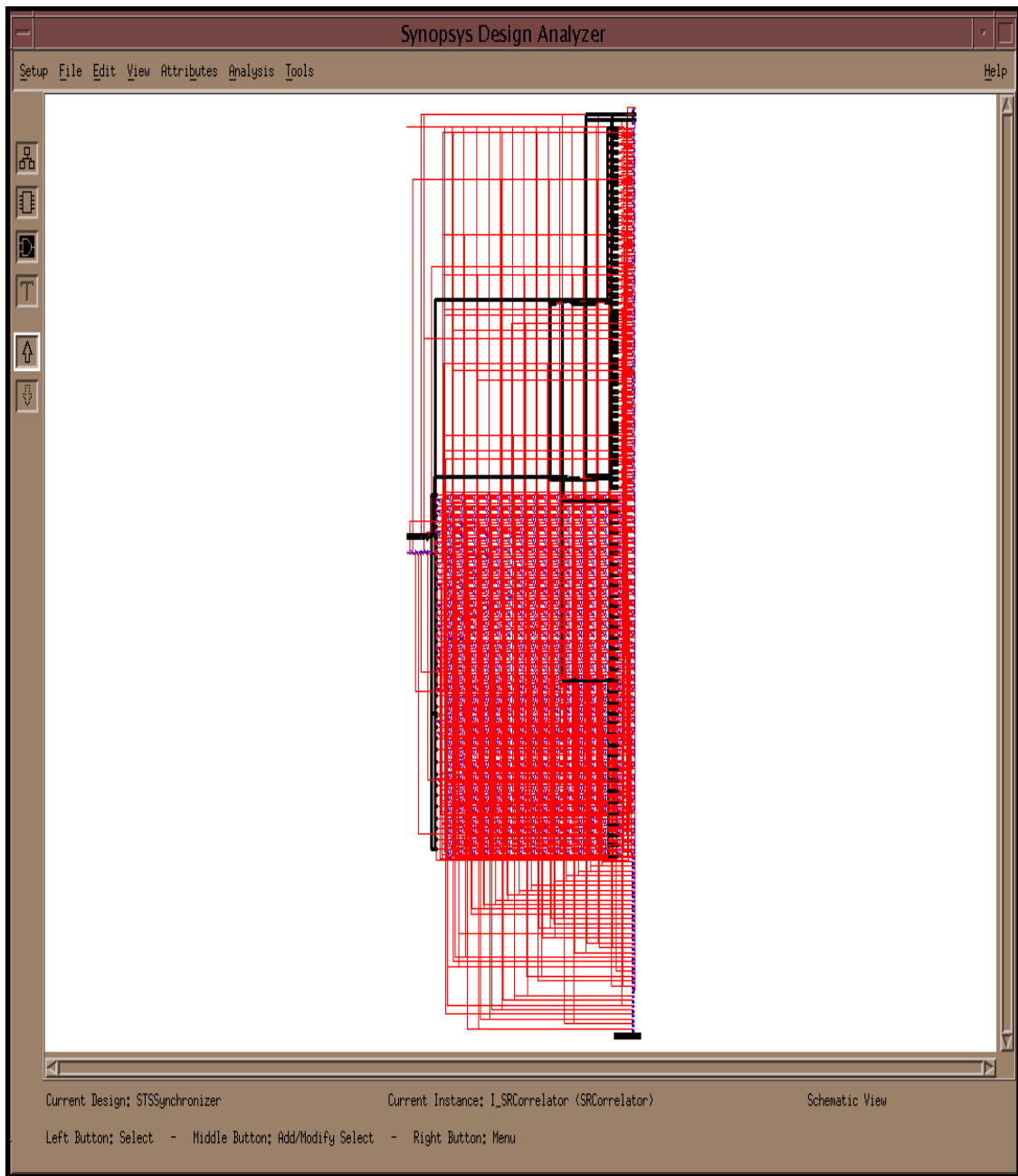
Figure A.5 Schematic view of synthesized SlidingCorrelator block
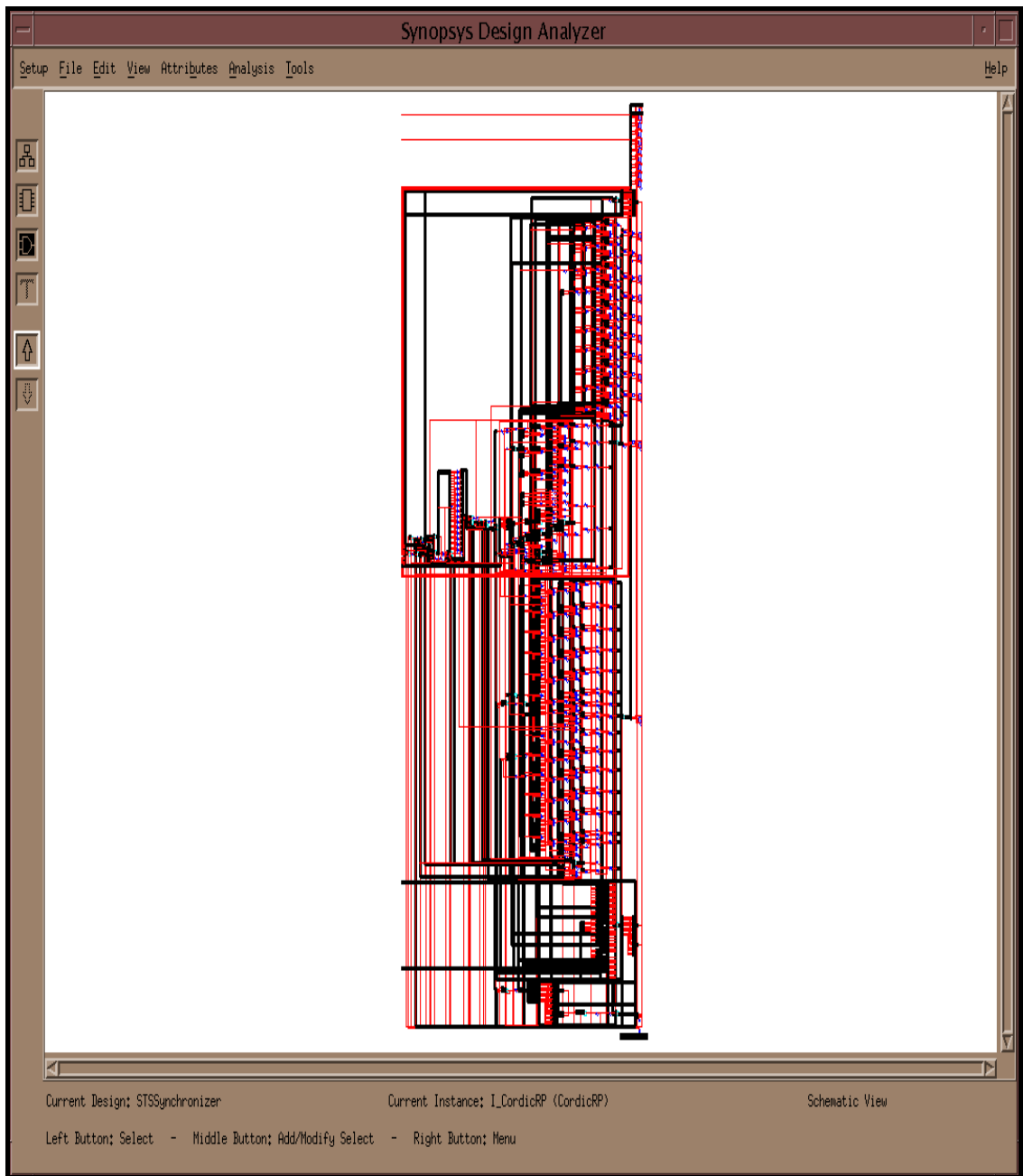
Figure A.6 Schematic view of synthesized CORDIC block

APPENDIX B: FUNCTIONAL VHDL CODES

1. STSSynchronizerConstants.Package.vhd

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

package STSSynchronizerConstants is


--SlidingShiftRegConstants

        constant SR_SHIFTREG_LENGTH     : integer := 17;
        constant SAMPLE_WIDTH           : integer := 8;
        constant NR_BITS                : integer := 8;

--SRCorrelatorConstants
        constant m                      : integer := 8;
        constant n                      : integer := 8;
        constant CORRELATION_LENGTH     : integer := 16;
        constant SAMPLE_IN_WIDTH        : integer := m;      --8
        constant SAMPLE_INTERN_WIDTH    : integer := m+n+7; --23
        constant SAMPLE_OUT_WIDTH       : integer := 10;

--CordicRPConstants

        constant D_CORDIC_SIGNED_wl     : integer := 10;
        constant D_CORDIC_UNSIGNED_wl   : integer := NR_BITS + 2;
        constant D_CORDIC_INTERN_wl     : integer := NR_BITS + 8;
        constant D_CORDIC_INTERN_iwl    : integer := 3;

        constant WIDTH                  : integer := D_CORDIC_SIGNED_wl;
        constant WIDTH_INTERN           : integer := D_CORDIC_INTERN_wl;
        constant IWIDTH_INTERN          : integer := D_CORDIC_INTERN_iwl;


end STSSynchronizerConstants;
```

2.   SRShiftRegComponent.Package.vhd

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

library lib_sts;
use lib_sts.STSSynchronizerConstants.all;

package SRShiftRegComponent is

 component SRShiftReg

 port (

 Clk20M             : in std_logic;
 Rst_N              : in std_logic;
 SampleRe_In        : in std_logic_vector(SAMPLE_WIDTH-1 downto 0);
 SampleIm_In        : in std_logic_vector(SAMPLE_WIDTH-1 downto 0);
 Sample1Re_Out      : out std_logic_vector(SAMPLE_WIDTH-1 downto 0);
 Sample1Im_Out      : out std_logic_vector(SAMPLE_WIDTH-1 downto 0);
 Sample2Re_Out      : out std_logic_vector(SAMPLE_WIDTH-1 downto 0);
 Sample2Im_Out      : out std_logic_vector(SAMPLE_WIDTH-1 downto 0)

 );
     end component;

end SRShiftRegComponent;
```

3.   SRShiftReg.Entity.vhd

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;


library lib_sts;
use lib_sts.STSSynchronizerConstants.all;


entity SRShiftReg is

  port(
    Clk20M         : in std_logic;
    Rst_N          : in std_logic;
    SampleRe_In    : in std_logic_vector(SAMPLE_WIDTH-1 downto 0);
    SampleIm_In    : in std_logic_vector(SAMPLE_WIDTH-1 downto 0);
    Sample1Re_Out  : out std_logic_vector(SAMPLE_WIDTH-1 downto 0);
    Sample1Im_Out  : out std_logic_vector(SAMPLE_WIDTH-1 downto 0);
    Sample2Re_Out  : out std_logic_vector(SAMPLE_WIDTH-1 downto 0);
    Sample2Im_Out  : out std_logic_vector(SAMPLE_WIDTH-1 downto 0));

end SRShiftReg;
```

4.   SRShiftReg.rtl.vhd


```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

library lib_sts;
use lib_sts.STSSynchronizerConstants.all;

architecture rtl of SRShiftReg is

  type t_SamplesShiftReg is array(SR_SHIFTREG_LENGTH-1 downto 0) of
std_logic_vector(SAMPLE_WIDTH-1 downto 0);

  signal SampleRe_r            : t_SamplesShiftReg;
  signal SampleIm_r            : t_SamplesShiftReg;

begin

p_SRShiftRegister: process(Clk20M, Rst_N, SampleRe_In, SampleIm_In,
SampleRe_r, SampleIm_r)
begin
  if (Rst_N = '0') then
    SampleRe_r <= (others=> (others => '0'));
    SampleIm_r <= (others=> (others => '0'));
  elsif (Clk20M'event and Clk20M = '1') then
    SampleRe_r <= SampleRe_r(SR_SHIFTREG_LENGTH-2 downto 0) &
SampleRe_In;
    SampleIm_r <= SampleIm_r(SR_SHIFTREG_LENGTH-2 downto 0) &
SampleIm_In;
  end if ;
end process p_SRShiftRegister;

  Sample1Re_Out <= SampleRe_r(SR_SHIFTREG_LENGTH-1);
  Sample1Im_Out <= SampleIm_r(SR_SHIFTREG_LENGTH-1);
  Sample2Re_Out <= SampleRe_r(0);
  Sample2Im_Out <= SampleIm_r(0);

end rtl;
```

5. CmplxMultiplierComponent.Package.vhd

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

library lib_sts;
use lib_sts.STSSynchronizerConstants.all;

package CmplxMultiplierComponent is


    component CmplxMultiplier

        port (

          InputXI                 : in std_logic_vector(m-1 downto 0);
          InputXQ                 : in std_logic_vector(m-1 downto 0);
          InputYI                 : in std_logic_vector(n-1 downto 0);
          InputYQ                 : in std_logic_vector(n-1 downto 0);
          OutputI                 : out std_logic_vector(m+n downto 0);
          OutputQ                 : out std_logic_vector(m+n downto 0)

              );
    end component;

end CmplxMultiplierComponent;
```

6. CmplxMultiplier.Entity.vhd

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

library lib_sts;
use lib_sts.STSSynchronizerConstants.all;

entity CmplxMultiplier is

     port(

    InputXI                 : in std_logic_vector(m-1 downto 0);
    InputXQ                 : in std_logic_vector(m-1 downto 0);
    InputYI                 : in std_logic_vector(n-1 downto 0);
    InputYQ                 : in std_logic_vector(n-1 downto 0);
    OutputI                 : out std_logic_vector(m+n downto 0);
    OutputQ                 : out std_logic_vector(m+n downto 0)

        );
end CmplxMultiplier;
```

7.   CmplxMultiplier.rtl.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

library lib_sts;
use lib_sts.STSSynchronizerConstants.all;


-- Two complex numbers are multiplied: InputX = A+jB, InputY = C+jD
-- Output = P + jQ where P = A*C - B*D, Q = A*D + B*C.


architecture rtl of CmplxMultiplier is

signal AD,BC,AC,BD      : std_logic_vector(m+n-1 downto 0);
-- AD=A*D, BC=B*C, AC=A*C, BD=B*D

begin

  AD <= CONV_STD_LOGIC_VECTOR(signed(InputXI) * signed(InputYQ),m+n);
  BC <= CONV_STD_LOGIC_VECTOR(signed(InputXQ) * signed(InputYI),m+n);
  AC <= CONV_STD_LOGIC_VECTOR(signed(InputXI) * signed(InputYI),m+n);
  BD <= CONV_STD_LOGIC_VECTOR(signed(InputXQ) * signed(InputYQ),m+n);

  OutputI <= CONV_STD_LOGIC_VECTOR((signed(AC(m+n-1) & AC) -
signed(BD(m+n-1) & BD)),m+n+1);
  OutputQ <= CONV_STD_LOGIC_VECTOR((signed(AD(m+n-1) & AD) +
signed(BC(m+n-1) & BC)),m+n+1);


end rtl;
```

8. SRCorrelatorComponent.Package.vhd

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

library lib_sts;
use lib_sts.STSSynchronizerConstants.all;

package SRCorrelatorComponent is


 component SRCorrelator

   port (

     Clk20M        : in std_logic;
     Rst_N         : in std_logic;
     Sample1Re_In  : in std_logic_vector(SAMPLE_IN_WIDTH-1 downto 0);
     Sample1Im_In  : in std_logic_vector(SAMPLE_IN_WIDTH-1 downto 0);
     Sample2Re_In  : in std_logic_vector(SAMPLE_IN_WIDTH-1 downto 0);
     Sample2Im_In  : in std_logic_vector(SAMPLE_IN_WIDTH-1 downto 0);
     SRRe_Out      : out std_logic_vector(SAMPLE_OUT_WIDTH-1 downto 0);
     SRIm_Out      : out std_logic_vector(SAMPLE_OUT_WIDTH-1 downto 0)

          );
     end component;

end SRCorrelatorComponent;
```

9. SRCorrelator.Entity.vhd

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

library lib_sts;
use lib_sts.STSSynchronizerConstants.all;

entity SRCorrelator is

  port(
   Clk20M        : in std_logic;
   Rst_N         : in std_logic;
   Sample1Re_In  : in std_logic_vector(SAMPLE_IN_WIDTH-1 downto 0);
   Sample1Im_In  : in std_logic_vector(SAMPLE_IN_WIDTH-1 downto 0);
   Sample2Re_In  : in std_logic_vector(SAMPLE_IN_WIDTH-1 downto 0);
   Sample2Im_In  : in std_logic_vector(SAMPLE_IN_WIDTH-1 downto 0);
   SRRe_Out      : out std_logic_vector(SAMPLE_OUT_WIDTH-1 downto 0);
   SRIm_Out      : out std_logic_vector(SAMPLE_OUT_WIDTH-1 downto 0));

end SRCorrelator;
```

10. SRCorrelator.rtl.vhd

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

library lib_sts;
use lib_sts.STSSynchronizerConstants.all;
use lib_sts.CmplxMultiplierComponent.all;

architecture rtl of SRCorrelator is



  type t_CorrRegRe is array(CORRELATION_LENGTH-1 downto 0) of
std_logic_vector(m+n downto 0);
  type t_CorrRegIm is array(CORRELATION_LENGTH-1 downto 0) of
std_logic_vector(m+n downto 0);

  signal CorrRegRe_r                      : t_CorrRegRe;
  signal CorrRegIm_r                      : t_CorrRegIm;
  signal ProductRe_Intern              : std_logic_vector(m+n downto 0);
  signal ProductIm_Intern              : std_logic_vector(m+n downto 0);
  signal ToBeSubtractedRegRe_Intern  : std_logic_vector(m+n downto 0);
  signal ToBeSubtractedRegIm_Intern  : std_logic_vector(m+n downto 0);
  signal SubtractRe_Intermediate    : std_logic_vector(m+n+1 downto 0);
  signal SubtractRe_Intermediate_Signed : signed(m+n+1 downto 0);
  signal SubtractIm_Intermediate    : std_logic_vector(m+n+1 downto 0);
  signal SubtractIm_Intermediate_Signed : signed(m+n+1 downto 0);
  signal I_MSB                        :
std_logic_vector(SAMPLE_INTERN_WIDTH-(m+n+2)-2 downto 0); -- 4 bits
  signal Q_MSB                        :
std_logic_vector(SAMPLE_INTERN_WIDTH-(m+n+2)-2 downto 0); -- 4 bits
  signal Sample1_Inverted            :
std_logic_vector(SAMPLE_IN_WIDTH-1 downto 0);
  signal Sample1_Inverted_Signed : signed(SAMPLE_IN_WIDTH-1 downto 0);
  signal SubtractRe_Intern            :
std_logic_vector(SAMPLE_INTERN_WIDTH-1 downto 0);
  signal SubtractIm_Intern            :
std_logic_vector(SAMPLE_INTERN_WIDTH-1 downto 0);
  signal SumRe_Intern                 :
std_logic_vector(SAMPLE_INTERN_WIDTH-1 downto 0);
  signal SumRe_Intern_Signed : signed(SAMPLE_INTERN_WIDTH-1 downto 0);
  signal SumIm_Intern                 :
std_logic_vector(SAMPLE_INTERN_WIDTH-1 downto 0);
  signal SumIm_Intern_Signed : signed(SAMPLE_INTERN_WIDTH-1 downto 0);
  signal SRRe_r : std_logic_vector(SAMPLE_INTERN_WIDTH-1 downto 0);
  signal SRIm_r : std_logic_vector(SAMPLE_INTERN_WIDTH-1 downto 0);

begin

I_ComplexMultiplier : CmplxMultiplier port map(
    InputXI   => Sample2Re_In,
    InputXQ   => Sample2Im_In,
    InputYI   => Sample1Re_In,
    InputYQ   => Sample1_Inverted,
    OutputI   => ProductRe_Intern,
    OutputQ   => ProductIm_Intern);
```

```vhdl
Sample1_Inverted_Signed <= - signed(Sample1Im_In);
Sample1_Inverted <=
CONV_STD_LOGIC_VECTOR(Sample1_Inverted_Signed,SAMPLE_IN_WIDTH);


ToBeSubtractedRegRe_Intern <= CorrRegRe_r(CORRELATION_LENGTH-1);
--CORRELATION_LENGTH-1=15
ToBeSubtractedRegIm_Intern <= CorrRegIm_r(CORRELATION_LENGTH-1);
--CORRELATION_LENGTH-1=15


SubtractRe_Intermediate_Signed <= signed(ProductRe_Intern(m+n) &
ProductRe_Intern) - signed(ToBeSubtractedRegRe_Intern(m+n) &
ToBeSubtractedRegRe_Intern);
--SubtractRe_Intermediate is 18-bits-wide ((m+n+1) = (m+n) + (m+n))
SubtractRe_Intermediate <=
CONV_STD_LOGIC_VECTOR(SubtractRe_Intermediate_Signed,(m+n+2));
I_MSB <= (others => SubtractRe_Intermediate(m+n+1));
--I_MSB is 4-bits-wide
SubtractRe_Intern <= I_MSB & SubtractRe_Intermediate & '0';
--SubtractRe_Intern is 23-bits-wide


SubtractIm_Intermediate_Signed <= signed(ProductIm_Intern(m+n) &
ProductIm_Intern) - signed(ToBeSubtractedRegIm_Intern(m+n) &
ToBeSubtractedRegIm_Intern);
--SubtractIm_Intermediate is 18-bits-wide ((m+n+1) = (m+n) + (m+n))
SubtractIm_Intermediate <=
CONV_STD_LOGIC_VECTOR(SubtractIm_Intermediate_Signed,(m+n+2));
Q_MSB <= (others => SubtractIm_Intermediate(m+n+1));
--Q_MSB is 4-bits-wide
SubtractIm_Intern <= Q_MSB & SubtractIm_Intermediate & '0';
--SubtractIm_Intern is 23-bits-wide

SumRe_Intern_Signed <= signed(SRRe_r) + signed(SubtractRe_Intern);
SumRe_Intern <= CONV_STD_LOGIC_VECTOR(SumRe_Intern_Signed, (m+n+7));

SumIm_Intern_Signed <= signed(SRIm_r) + signed(SubtractIm_Intern);
SumIm_Intern <= CONV_STD_LOGIC_VECTOR(SumIm_Intern_Signed, (m+n+7));


SRRe_Out <= SRRe_r(20 downto 11);
SRIm_Out <= SRIm_r(20 downto 11);


p_CorrelatorRegister: process(Clk20M, Rst_N, CorrRegRe_r, CorrRegIm_r,
ProductRe_Intern, ProductIm_Intern)
begin
-- Input Samples are being shifted.
  if (Rst_N = '0') then
    CorrRegRe_r <= (others => (others => '0'));
    CorrRegIm_r <= (others => (others => '0'));
  elsif (Clk20M'event and Clk20M = '1') then
    CorrRegRe_r <= CorrRegRe_r(CORRELATION_LENGTH-2 downto 0) &
ProductRe_Intern ;
    CorrRegIm_r <= CorrRegIm_r(CORRELATION_LENGTH-2 downto 0) &
ProductIm_Intern ;
  end if;
end process p_CorrelatorRegister;
```

```
p_Accumulator: process(Clk20M, Rst_N)
begin
  if (Rst_N = '0') then
    SRRe_r <= (others => '0');
    SRIm_r <= (others => '0');
  elsif (Clk20M'event and Clk20M = '1') then
    SRRe_r <= SumRe_Intern;
    SRIm_r <= SumIm_Intern;
  end if ;  --Clk
end process p_Accumulator;


end rtl;
```

11.   CordicRPComponent.Package.vhd


```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

library lib_sts;
use lib_sts.STSSynchronizerConstants.all;

package CordicRPComponent is

  component CordicRP

    port (

        Clk20M                  : in std_logic;
        Rst_N                   : in std_logic;
        NRIterations_In         : in std_logic_vector(3 downto 0);
        RReal_In                : in std_logic_vector(WIDTH-1 downto 0);
        RImag_In                : in std_logic_vector(WIDTH-1 downto 0);

        RAmp_Out                : out std_logic_vector(WIDTH-1 downto 0);
        RPh_Out                 : out std_logic_vector(WIDTH-1 downto 0)

          );
    end component;


end CordicRPComponent;
```

12. CordicRP.Entity.vhd

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

library lib_sts;
use lib_sts.STSSynchronizerConstants.all;


entity CordicRP is

    port(
        Clk20M              : in std_logic;
        Rst_N               : in std_logic;
        NRIterations_In     : in std_logic_vector(3 downto 0);
        RReal_In            : in std_logic_vector(WIDTH-1 downto 0);
        RImag_In            : in std_logic_vector(WIDTH-1 downto 0);
        RAmp_Out            : out std_logic_vector(WIDTH-1 downto 0);
        RPh_Out             : out std_logic_vector(WIDTH-1 downto 0)
          );

end CordicRP;
```

13. CordicRP.rtl.vhd

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

library lib_sts;
use lib_sts.STSSynchronizerConstants.all;

architecture rtl of CordicRP is

signal RReal_Ext : std_logic_vector(WIDTH_INTERN-1 downto 0);
signal RImag_Ext : std_logic_vector(WIDTH_INTERN-1 downto 0);

signal X_0 ,Y_0, Z_0, Xshft_0, Yshft_0      : signed(WIDTH_INTERN-1
downto 0);
signal X_1 ,Y_1, Z_1, Xshft_1, Yshft_1      : signed(WIDTH_INTERN-1
downto 0);
signal X_2 ,Y_2, Z_2, Xshft_2, Yshft_2      : signed(WIDTH_INTERN-1
downto 0);
signal X_3 ,Y_3, Z_3, Xshft_3, Yshft_3      : signed(WIDTH_INTERN-1
downto 0);
signal X_4 ,Y_4, Z_4, Xshft_4, Yshft_4      : signed(WIDTH_INTERN-1
downto 0);
signal X_5 ,Y_5, Z_5, Xshft_5, Yshft_5      : signed(WIDTH_INTERN-1
downto 0);
signal X_6 ,Y_6, Z_6, Xshft_6, Yshft_6      : signed(WIDTH_INTERN-1
downto 0);
signal X_7 ,Y_7, Z_7, Xshft_7, Yshft_7      : signed(WIDTH_INTERN-1
downto 0);
```

```vhdl
signal X_8 ,Y_8, Z_8, Xshft_8, Yshft_8    : signed(WIDTH_INTERN-1
downto 0);
signal X_9 ,Y_9, Z_9, Xshft_9, Yshft_9    : signed(WIDTH_INTERN-1
downto 0);
signal X_10 ,Y_10, Z_10                   : signed(WIDTH_INTERN-1
downto 0);


signal LeftHalfPlaneFlag        : std_logic;
signal RRealSigned              : signed (WIDTH_INTERN-1 downto 0);
signal RImagSigned              : signed (WIDTH_INTERN-1 downto 0);
signal NRIterationsUnsigned     : unsigned(3 downto 0);
signal NRIterationsint          : integer range 0 to 15;
signal X0                       : signed(WIDTH_INTERN-1 downto 0);
signal Y0                       : signed(WIDTH_INTERN-1 downto 0);
signal Z0                       : signed(WIDTH_INTERN-1 downto 0);
signal Xout                     : signed(WIDTH_INTERN-1 downto 0);
signal Zout                     : signed(WIDTH_INTERN-1 downto 0);
signal compen                   : std_logic_vector(2*WIDTH_INTERN-1+1
downto 0);
signal RPh_nxt                  : signed(WIDTH_INTERN-1 downto 0);
signal RPh_nxt_left             : unsigned(WIDTH-1 downto 0);
signal RPhUnsigned              : unsigned(WIDTH_INTERN-1 downto 0);
constant ZERO                   : signed(WIDTH_INTERN-1 downto 0) :=
"0000000000000000";
constant ONE                    : signed(WIDTH_INTERN-1 downto 0) :=
"0010000000000000";
constant TWO                    : signed(WIDTH_INTERN-1 downto 0) :=
"0100000000000000";
constant FOUR                   : signed(WIDTH_INTERN-1 downto 0) :=
"1000000000000000";
constant COMPENSATION           : signed(WIDTH_INTERN-1 downto 0) :=
"0001001101101110" ;
-- 0.607253321089875 --0.607177734375
constant COMPENSATION_SQRT2 : unsigned(WIDTH_INTERN-1+1 downto 0) :=
"11011011110100101" ;   -- 0.607253321089875 / SQRT(2)

--constant StepPhase0        : integer := 4096; -- "0001000000000000"
--constant StepPhase1        : integer := 2418; -- "0000100101110010"
--constant StepPhase2        : integer := 1277; -- "0000010011111101"
--constant StepPhase3        : integer := 648;  -- "0000001010001000"
--constant StepPhase4        : integer := 325;  -- "0000000101000101"
--constant StepPhase5        : integer := 162;  -- "0000000010100010"
--constant StepPhase6        : integer := 81;   -- "0000000001010001"
--constant StepPhase7        : integer := 40;   -- "0000000000101000"
--constant StepPhase8        : integer := 20;   -- "0000000000010100"
--constant StepPhase9        : integer := 10;   -- "0000000000001010"
constant StepPhase0         : signed(WIDTH_INTERN-1 downto 0) :=
"0001000000000000";
constant StepPhase1         : signed(WIDTH_INTERN-1 downto 0) :=
"0000100101110010";
constant StepPhase2         : signed(WIDTH_INTERN-1 downto 0) :=
"0000010011111101";
constant StepPhase3         : signed(WIDTH_INTERN-1 downto 0) :=
"0000001010001000";
constant StepPhase4         : signed(WIDTH_INTERN-1 downto 0) :=
"0000000101000101";
constant StepPhase5         : signed(WIDTH_INTERN-1 downto 0) :=
"0000000010100010";
constant StepPhase6         : signed(WIDTH_INTERN-1 downto 0) :=
"0000000001010001";
```

```vhdl
constant StepPhase7          : signed(WIDTH_INTERN-1 downto 0) :=
"0000000000101000";
constant StepPhase8          : signed(WIDTH_INTERN-1 downto 0) :=
"0000000000010100";
constant StepPhase9          : signed(WIDTH_INTERN-1 downto 0) :=
"0000000000001010";

procedure CordicCore(   X_pre    : in signed(WIDTH_INTERN-1 downto 0);
                        Y_pre    : in signed(WIDTH_INTERN-1 downto 0);
                        Z_pre    : in signed(WIDTH_INTERN-1 downto 0);
                        X_shift  : in signed(WIDTH_INTERN-1 downto 0);
                        Y_shift  : in signed(WIDTH_INTERN-1 downto 0);
                        signal X : out signed(WIDTH_INTERN-1 downto
0);
                        signal Y: out signed(WIDTH_INTERN-1 downto 0);
                        signal Z: out signed(WIDTH_INTERN-1 downto 0);
                        StepPhase : in signed(WIDTH_INTERN-1 downto 0)
                        ) is
                    begin
                        if Y_pre < 0 then
                            X <= X_pre-Y_shift;
                            Y <= Y_pre+X_shift;
                            Z <= Z_pre-StepPhase;
                        else
                            X <= X_pre+Y_shift;
                            Y <= Y_pre-X_shift;
                            Z <= Z_pre+StepPhase;
                        end if;
end CordicCore;

procedure ShiftRight(   X    : in signed(WIDTH_INTERN-1 downto 0);
                        Y    : in signed(WIDTH_INTERN-1 downto 0);
                        iteration: in integer;
                        signal X_shift: out signed(WIDTH_INTERN-1
downto 0);
                        signal Y_shift: out signed(WIDTH_INTERN-1
downto 0)
                        ) is
                variable X_int : signed(WIDTH_INTERN-1 downto 0);
                variable Y_int : signed(WIDTH_INTERN-1 downto 0);
                    begin

                    X_int := X;
                    for I in 1 to iteration loop
                        X_int(WIDTH_INTERN-I) := X(WIDTH_INTERN-1);
                    end loop;
                    X_int(WIDTH_INTERN-iteration-1 downto 0) :=
X(WIDTH_INTERN-1 downto iteration);
                    X_shift <= X_int;
                    Y_int := Y;
                    for I in 1 to iteration loop
                        Y_int(WIDTH_INTERN-I) := Y(WIDTH_INTERN-1);
                    end loop;
                    Y_int(WIDTH_INTERN-iteration-1 downto 0) :=
Y(WIDTH_INTERN-1 downto iteration);
                    Y_shift <= Y_int;

end ShiftRight;
```

```
begin

RReal_Ext <= RReal_In(WIDTH-1) & RReal_In(WIDTH-1) & RReal_In &
"0000";
RImag_Ext <= RImag_In(WIDTH-1) & RImag_In(WIDTH-1) & RImag_In &
"0000";

RRealSigned <= signed(RReal_Ext);
RImagSigned <= signed(RImag_Ext);

NRIterationsUnsigned    <= unsigned(NRIterations_In);
NRIterationsint         <= conv_integer(NRIterationsUnsigned);

------------------------------------------------------------------------

PreCordicR2PProcess_PROC:process(RRealSigned, RImagSigned)

begin
if RRealSigned < 0 then
  X0 <= ZERO - RRealSigned;
  LeftHalfPlaneFlag <= '1';
else
  X0 <= RRealSigned;
  LeftHalfPlaneFlag <= '0';
end if;
Y0 <= RImagSigned;
Z0 <= ZERO;
end process;

X_0 <= X0;
Y_0 <= Y0;
Z_0 <= Z0;

------------------------------------------------------------------------

Xshft_0 <= X_0;
Yshft_0 <= Y_0;

CordicCore ( X_0, Y_0, Z_0, Xshft_0, Yshft_0, X_1, Y_1, Z_1,
StepPhase0 );
ShiftRight (X_1, Y_1, 1, Xshft_1, Yshft_1 );

CordicCore ( X_1, Y_1, Z_1, Xshft_1, Yshft_1, X_2, Y_2, Z_2,
StepPhase1 );
ShiftRight (X_2, Y_2, 2, Xshft_2, Yshft_2 );

CordicCore ( X_2, Y_2, Z_2, Xshft_2, Yshft_2, X_3, Y_3, Z_3,
StepPhase2 );
ShiftRight (X_3, Y_3, 3, Xshft_3, Yshft_3 );

CordicCore ( X_3, Y_3, Z_3, Xshft_3, Yshft_3, X_4, Y_4, Z_4,
StepPhase3 );
ShiftRight (X_4, Y_4, 4, Xshft_4, Yshft_4 );

CordicCore ( X_4, Y_4, Z_4, Xshft_4, Yshft_4, X_5, Y_5, Z_5,
StepPhase4 );
ShiftRight (X_5, Y_5, 5, Xshft_5, Yshft_5 );

CordicCore ( X_5, Y_5, Z_5, Xshft_5, Yshft_5, X_6, Y_6, Z_6,
StepPhase5 );
ShiftRight (X_6, Y_6, 6, Xshft_6, Yshft_6 );
```

```
CordicCore ( X_6, Y_6, Z_6, Xshft_6, Yshft_6, X_7, Y_7, Z_7,
StepPhase6 );
ShiftRight (X_7, Y_7, 7, Xshft_7, Yshft_7 );

CordicCore ( X_7, Y_7, Z_7, Xshft_7, Yshft_7, X_8, Y_8, Z_8,
StepPhase7 );
ShiftRight (X_8, Y_8, 8, Xshft_8, Yshft_8 );

CordicCore ( X_8, Y_8, Z_8, Xshft_8, Yshft_8, X_9, Y_9, Z_9,
StepPhase8 );
ShiftRight (X_9, Y_9, 9, Xshft_9, Yshft_9 );

CordicCore ( X_9, Y_9, Z_9, Xshft_9, Yshft_9, X_10, Y_10, Z_10,
StepPhase9 );

------------------------------------------------------------------------

SelectIteration_PROC:
process(RRealSigned,RImagSigned,NRIterationsint,X_0,Z_0,X_1,Z_1,X_2,Z_
2,X_3,Z_3,X_4,Z_4,X_5,Z_5,X_6,Z_6,X_7,Z_7,X_8,Z_8,X_9,Z_9,X_10,Z_10)

begin

if RRealSigned = 0 and RImagSigned = 0 then
  Xout <= ZERO;
  Zout <= ZERO;
else
 case NRIterationsint is
     when 0 =>
          Xout <= X_0;
          Zout <= Z_0;
     when 1 =>
          Xout <= X_1;
          Zout <= Z_1;
     when 2 =>
          Xout <= X_2;
          Zout <= Z_2;
     when 3 =>
          Xout <= X_3;
          Zout <= Z_3;
     when 4 =>
          Xout <= X_4;
          Zout <= Z_4;
     when 5 =>
          Xout <= X_5;
          Zout <= Z_5;
     when 6 =>
          Xout <= X_6;
          Zout <= Z_6;
     when 7 =>
          Xout <= X_7;
          Zout <= Z_7;
     when 8 =>
          Xout <= X_8;
          Zout <= Z_8;
     when 9 =>
          Xout <= X_9;
          Zout <= Z_9;
     when 10 =>
          Xout <= X_10;
```

```vhdl
            Zout <= Z_10;
        when others =>
            Xout <= ZERO;
            Zout <= ZERO;
 end case;
end if;
end process;


------------------------------------------------------------------------

PostCordicCom_PROC: process (Zout, LeftHalfPlaneFlag)
begin
if LeftHalfPlaneFlag = '1' then
     RPh_nxt <= TWO - Zout;
elsif Zout < 0 then
     RPh_nxt <= FOUR + Zout;
else
     RPh_nxt <= Zout;
end if;
end process;

RPhUnsigned <= conv_unsigned(RPh_nxt, WIDTH_INTERN);
RPh_nxt_left <= RPhUnsigned(WIDTH_INTERN-2 downto WIDTH_INTERN-2-
WIDTH+1);      -- RPh_nxt / 4

compen <= conv_unsigned(Xout, WIDTH_INTERN) * COMPENSATION_SQRT2;

------------------------------------------------------------------------

PostCordicSeq_PROC: process (Rst_N, Clk20M)
begin
if Rst_N = '0' then
  RAmp_Out <= (others => '0');
  RPh_Out  <= (others => '0');
elsif Clk20M'event and Clk20M='1' then

  RAmp_Out <= compen((2*WIDTH_INTERN-IWIDTH_INTERN+1) downto
(2*WIDTH_INTERN-IWIDTH_INTERN-WIDTH+1+1)); -- (29 downto 20)
  RPh_Out <= std_logic_vector(RPh_nxt_left);
end if;
end process;

end rtl;
```

14. STSSynchronizerComponent.Package.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

library lib_sts;
use lib_sts.STSSynchronizerConstants.all;

package STSSynchronizerComponent is


component STSSynchronizer

 port (
Clk20M                  : in std_logic;
Rst_N                   : in std_logic;
NRIterations_In         : in std_logic_vector(3 downto 0);
SampleRe_In             : in std_logic_vector(SAMPLE_WIDTH-1 downto 0);
SampleIm_In             : in std_logic_vector(SAMPLE_WIDTH-1 downto 0);
RAmp_Out                : out std_logic_vector(WIDTH-1 downto 0);
RPh_Out                 : out std_logic_vector(WIDTH-1 downto 0));

 end component;

end STSSynchronizerComponent;
```

15. STSSynchronizer.Entity.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

library lib_sts;
use lib_sts.STSSynchronizerConstants.all;

entity STSSynchronizer is

port(

    Clk20M          : in std_logic;
    Rst_N           : in std_logic;
    NRIterations_In : in std_logic_vector(3 downto 0);
    SampleRe_In     : in std_logic_vector(SAMPLE_WIDTH-1 downto 0);
    SampleIm_In     : in std_logic_vector(SAMPLE_WIDTH-1 downto 0);
    RAmp_Out        : out std_logic_vector(WIDTH-1 downto 0);
    RPh_Out         : out std_logic_vector(WIDTH-1 downto 0));

end STSSynchronizer;
```

16. STSSynchronizer.rtl.vhd

```vhdl
library lib_sts;
use lib_sts.STSSynchronizerConstants.all;
use lib_sts.SRShiftRegComponent.all;
use lib_sts.SRCorrelatorComponent.all;
use lib_sts.CordicRPComponent.all;


architecture rtl of STSSynchronizer is


signal Sample1Re: std_logic_vector(SAMPLE_WIDTH-1 downto 0);
signal Sample1Im: std_logic_vector(SAMPLE_WIDTH-1 downto 0);
signal Sample2Re: std_logic_vector(SAMPLE_WIDTH-1 downto 0);
signal Sample2Im: std_logic_vector(SAMPLE_WIDTH-1 downto 0);
signal Rreal    : std_logic_vector(SAMPLE_OUT_WIDTH-1 downto 0);
signal Rimag    : std_logic_vector(SAMPLE_OUT_WIDTH-1 downto 0);

begin

I_SRShiftReg : SRShiftReg port map(
    Clk20M              =>   Clk20M,
    Rst_N               =>   Rst_N,
    SampleRe_In         =>   SampleRe_In,
    SampleIm_In         =>   SampleIm_In,
    Sample1Re_Out       => Sample1Re,
    Sample1Im_Out       => Sample1Im,
    Sample2Re_Out       => Sample2Re,
    Sample2Im_Out       => Sample2Im);

I_SRCorrelator :   SRCorrelator port map(
    Clk20M              =>   Clk20M,
    Rst_N               =>   Rst_N,
    Sample1Re_In        => Sample1Re,
    Sample1Im_In        => Sample1Im,
    Sample2Re_In        => Sample2Re,
    Sample2Im_In        => Sample2Im,
    SRRe_Out            => RReal,
    SRIm_Out            => RImag);

I_CordicRP    :    CordicRP port map(
    Clk20M              =>   Clk20M,
    Rst_N               =>   Rst_N,
    NRIterations_In     => NRIterations_In,
    RReal_In            => RReal,
    RImag_In            => RImag,
    RAmp_Out            => RAmp_Out,
    RPh_Out             => RPh_Out);

end rtl;
```

17. TB_STSSynchronizer.rtl.vhd

```vhdl
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.std_logic_arith.all;
use std.textio.all;
use ieee.std_logic_textio.all;

library lib_sts;
use lib_sts.STSSynchronizerConstants.all;
use lib_sts.STSSynchronizerComponent.all;

entity TBE_STSSynchronizer_rtl is
end TBE_STSSynchronizer_rtl;

architecture TBA_STSSynchronizer_rtl of TBE_STSSynchronizer_rtl is


signal Clk20M              : std_logic:='0';
signal Rst_N               : std_logic:='0';
signal NRIterations_In     : std_logic_vector(3 downto 0):="1010";
--N = 10
signal SampleRe_In         : std_logic_vector(SAMPLE_WIDTH-1 downto
0):="00000000";
signal SampleIm_In         : std_logic_vector(SAMPLE_WIDTH-1 downto
0):="00000000";
signal RAmp_Out            : std_logic_vector(9 downto 0);
signal RPh_Out             : std_logic_vector(9 downto 0);
signal FirstChar           : string(1 to 1);
signal SimEnd              : boolean := false;
constant c_Period20Mhz     : time:= 50 ns;

begin

  I_STSSynchronizer : STSSynchronizer port map (
                  Clk20M            => Clk20M,
                  Rst_N             => Rst_N,
                  NRIterations_In   => NRIterations_In,
                  SampleRe_In       => SampleRe_In,
                  SampleIm_In       => SampleIm_In,
                  RAmp_Out          => RAmp_Out,
                  RPh_Out           => RPh_Out
                  );


Rst_N <= '1' after 10 ns;

p_ClkGenerator: process
    begin
      Clk20M <= '0';
      wait for c_Period20Mhz/2;
      while not SimEnd loop
        Clk20M <= '1';
        wait for c_Period20Mhz/2;
        Clk20M <= '0';
        wait for c_Period20Mhz/2;
      end loop;
    end process p_ClkGenerator;
```

```vhdl
 p_apply_stimuli : process (Clk20M, Rst_N)
     file InputFile : text is in "./Test_Data_Dir/ETSIStimuli.txt";
         variable InputVector   : line;
         variable Command_Col   : string(1 to 1);
         variable Temp_In       : std_logic_vector(7 downto 0);

         begin

             if (Rst_N = '0') then
                     --Do nothing
             elsif (Clk20M'event and Clk20M = '1') then
-- Stimuli on the positive edge

                 if not endfile(InputFile) then

                 readline(InputFile, InputVector);

         --SampleRe_In
                     read(InputVector, Temp_In);
                     SampleRe_In <= Temp_In;

         --SampleIm_In
                     read(InputVector, Temp_In);
                     SampleIm_In <= Temp_In;

                 else
                     SimEnd <= true;
                     assert false report " End of Simulation"
severity failure;
                 end if;

             end if;
         end process p_apply_stimuli;

--
-- writing on falling edge
--
p_write_outputs: process(Clk20M, Rst_N)

file OutputFile1: text is out "./Test_Data_Dir/RAmp_Out_thesis.txt";
file OutputFile2: text is out "./Test_Data_Dir/RPh_Out_thesis.txt";

    variable OutVector1             : line;
    variable OutVector2             : line;

    begin
        if (Rst_N = '0') then
            -- Do nothing
        elsif Clk20M'event and Clk20M='0' then
            write(OutVector1,RAmp_Out);
            write(OutVector2,real(conv_integer(signed(RPh_Out))) /
512.0 );
            writeline(OutputFile1,OutVector1);
            writeline(OutputFile2,OutVector2);

        end if;

        end process p_write_outputs;

end TBA_STSSynchronizer_rtl;
```

C. APPENDIX C: GATE-LEVEL VHDL CODES

1. STSSynchronizer.Shell.vhd

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

library lib_sts;
use lib_sts.STSSynchronizerConstants.all;

entity STSSynchronizer is

port(

    Clk20M              : in std_logic;
    Rst_N               : in std_logic;
    NRIterations_In     : in std_logic_vector(3 downto 0);
    SampleRe_In         : in std_logic_vector(SAMPLE_WIDTH-1 downto 0);
    SampleIm_In         : in std_logic_vector(SAMPLE_WIDTH-1 downto 0);
    RAmp_Out            : out std_logic_vector(WIDTH-1 downto 0);
    RPh_Out             : out std_logic_vector(WIDTH-1 downto 0));

end STSSynchronizer;

architecture verilog of STSSynchronizer is
        attribute foreign of verilog: architecture is "VERILOG(event)
lib_sts.STSSynchronizer:v";
begin
end;
```

2. TB_STSSynchronizer_GAT.rtl.vhd

It is same as TB_STSSynchronizer.rtl.vhd.

APPENDIX D: ETSI BRAN HIPERLAN TYPE 2 STANDARD

The increasing demand for "anywhere, anytime" communications and the merging of voice, video and data communications create a demand for broadband wireless networks. ETSI has created the BRAN project to develop standards and specifications for broadband radio access networks that cover a wide range of applications and are intended for different frequency bands. This range of applications covers systems for licensed and license exempt use.

The categories of systems covered by the BRAN project are summarized as follows:

- **HIPERLAN/1** provides high-speed (20 Mbit/s typical gross data rate) radio local area network communications that are compatible with wired LANs based on Ethernet and Token Ring standards. Restricted user mobility is supported within the local service area only. The technical specification for HIPERLAN/1, ETS 300 652, was published by ETSI in 1996 (last revised version published as EN 300 652). HIPERLAN/1 systems are intended to be operated in the 5 GHz band.

- **HIPERLAN/2** is a standard for a high-speed radio communication system with typical data rates from 6 Mbit/s to 54 Mbit/s. It connects portable devices with broadband networks that are based on IP, ATM and other technologies. Centralized mode is used to operate HIPERLAN/2 as an access network via a fixed access point. In addition a capability for direct link communication is provided. This mode is used to operate HIPERLAN/2 as an ad-hoc network without relying on a cellular network infrastructure. In this case a central controller (CC), which is dynamically selected among the portable devices, provides the same level of QoS support as the fixed access point. HIPERLAN/2 is capable of supporting

multi-media applications by providing mechanisms to handle QoS. Restricted user mobility is supported within the local service area; wide area mobility (e.g. roaming) may be supported by standards outside the scope of the BRAN project. HIPERLAN/2 systems are intended to be operated in the 5 GHz band.

- **HIPERLINK** provides very high-speed (up to 155 Mbit/s data rate) radio links for static interconnections and is capable of multi-media applications; a typical use is the interconnection of HIPERACCESS networks and/or HIPERLAN access points into a fully wireless network. It should be noted that for HIPERLINK the intended operation frequency is 17 GHz - this in view of the very limited EIRP allowed in CEPT/ERC TR/22-06.

Since HIPERLAN/2 is used as the standard for the implementation part (ETSI OFDM STS Synchronizer Hardware Design) of this thesis, only HIPERLAN/2's parameters and specifications are mentioned below.

D.1. HIPERLAN/2 Services and Functions

D.1.1. Introduction

A HIPERLAN/2 network for business environment consists typically of a number of APs (access point) each of them covers a certain geographic area. Together they form a radio access network with full or partial coverage of an area of almost any size. The coverage areas may or may not overlap each other, thus simplifying roaming of terminals inside the radio access network. Each AP serves a number of MTs, which have to be associated to it. In the case where the quality of the radio link degrades to an unacceptable level, the terminal may move to another AP by performing a handover. For home environment, HIPERLAN/2 network is operated as an ad-hoc LAN, which can be put into operation in a plug-and-play manner. The HIPERLAN/2 home system

share the same basic features with the HIPERLAN/2 business system by defining the following equivalence between both systems:

- A subnet in the ad-hoc LAN configuration is equivalent to a cell in the cellular access network configuration.
- A central controller in the ad-hoc LAN configuration is equivalent to the access point in the cellular access network configuration. However, the central controller is dynamically selected from HIPERLAN/2 portable devices and can be handed over to another portable device, if the old one leaves the network.
- Multiple subnets in a home are made possible by having multiple CCs (central controller) operating at different frequencies.

HIPERLAN/2 supports two basic modes of operation:

- **Centralized mode**: In this mode, an AP is connected to a core network, which serves the MTs (mobile terminal) associated to it. All traffic has to pass the AP, regardless of whether the data exchange is between an MT and a terminal elsewhere in the core network or between MTs belonging to this AP. The basic assumption is that a major share of the traffic is exchanged with terminals elsewhere in the network. This feature is mandatory for all MTs and APs.
- **Direct mode**: In this mode, the medium access is still managed in a centralized manner by a CC. However, user data traffic is exchanged between terminals without going through the CC. It is expected that in some applications (especially, in home environment), a large portion of user data traffic is exchanged between terminals associated with a single CC. This feature is intended for use within home environment, and hence, is mandatory in DLC (data link control)-home extensions.

NOTE 1: A central controller may also be connected to a core network and, thus, shall be able to operate in both direct and centralized mode.

The HIPERLAN/2 basic protocol stack on the AP/CC side and its functions are shown in Figure D.1. The convergence layer (CL) offers service to the higher layers that are out of the scope of this document.

The physical layer delivers a basic data transport function by providing means of a base-band modem and a RF part. The base-band modem will also contain a forward error correction function.

The DLC layer consists of the Error Control (EC) function, the Medium Access Control (MAC) function and the Radio Link Control (RLC) function. It is divided in the user data transport functions and the control functions, located mainly on the right hand side and on the left-hand side of Figure D.1, respectively.

The user data transport function is fed with user data packets from higher layers via the User Service Access Point (USAP). This part contains the EC that performs an ARQ (Automatic Repeat Request) protocol. The DLC protocol operates connection oriented, which is shown by multiple connection end points in the USAP. One EC instance is created for each DLC connection. In case the higher layer is connection oriented, DLC connections can be created and released dynamically. In case the higher layer is connectionless, at least one DLC connection has to be set up which handles all user data.

The left part contains the RLC Sub-layer, which delivers a transport service to the DLC Connection Control (DCC), the Radio Resource Control (RRC) and the Association Control Function (ACF).

NOTE 2: Only the RLC is standardized which defines implicitly the behavior of the DCC, ACF and RRC. One RLC instance needs to be created per MT.

The CL on top is also separated in a data transport and a control part. The data transport part provides the adaptation of the user data format to the message format of the DLC layer (DLC SDU). In case of higher layer networks other than ATM, it contains a segmentation and re-assembly (SAR) function. The control part can make use of the control functions in the DLC, e.g. when negotiating CL parameters at association time.
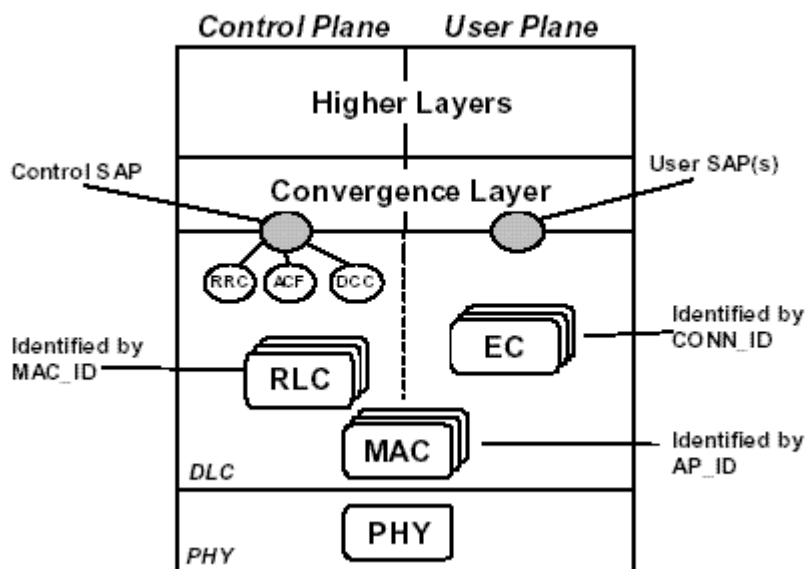
Figure D.1 HIPERLAN/2 Protocol Stack and Functions

D.1.2. HIPERLAN/2 DLC Functions

The HIPERLAN/2 DLC functions are divided in data transport and data link control functions and will be described in two sub-clauses in the following.

D.1.2.1. Medium Access Control

The medium access control is a centrally scheduled TDMA/TDD scheme. Centrally scheduled means that the AP/CC controls all transmissions over the air. This concerns uplink, downlink and direct mode phase equally.

The basic structure on the air interface generated by the MAC is shown in Figure D.2. It consists of a sequence of MAC frames of equal length with 2 ms duration. Each MAC frame consists of several phases:

- Broadcast (BC) phase: The BC phase carries the BCCH (broadcast control channel) and the FCCH (frame control channel). The BCCH contains general announcements and some status bits announcing the appearance of more detailed broadcast information in the downlink phase (DL). The

FCCH carries the information about the structure of the ongoing frame, containing the exact position of all following emissions, their usage and content type. The messages in the FCCH are called resource grants (RG).

- Downlink (DL) phase: The DL phase carries user specific control information and user data, transmitted from AP/CC to MTs. Additionally, the DL phase may contain further broadcast information which does not fit in the fixed BCCH field.
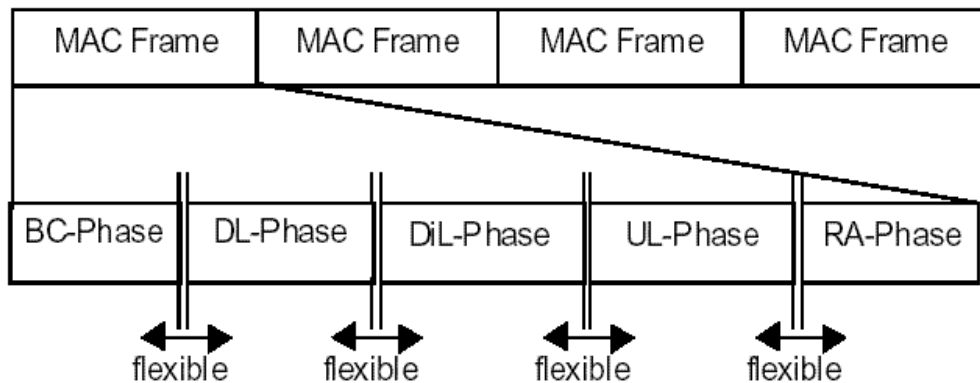
Figure D.2 MAC Frame Format for Sectored Antennas

- Uplink (UL) phase: The UL phase carries control and user data from the MTs to the AP/CC. The MTs have to request capacity for one of the following frames in order to get resources granted by the AP/CC.
- Direct Link (DiL) phase: The DiL phase carries user data traffic between MTs without direct involvement of the AP/CC. However, for control traffic, the AP/CC is indirectly involved by receiving Resource Requests from MTs for these connections and transmitting Resource Grants in the FCCH.

NOTE 1: The DiL phase is mandatory in home environments.

- Random access (RA) phase: The RA phase carriers a number of RCH (random access channels). MTs to which no capacity has been allocated in the UL phase use this phase for the transmission of control information.

Non-associated MTs use RCHs for the first contact with an AP/CC. This phase is also used by MTs performing handover to have their connections switched over to a new AP/CC.

The structure is slightly different when the AP/CC has a sectored antenna as shown in Figure D.3. The solution chosen distributes the available MAC frame duration over the sectors. In this case, each phase is repeated, in time, one for each sector.

NOTE 2: The use of DiL with sectored antennas is not specified.
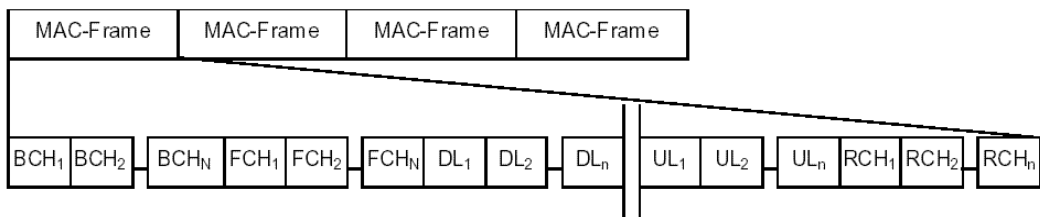
Figure D.3 MAC Frame Format for Sectored Antennas

The DL, DiL and UL phases consist of two types of PDUs: long PDUs and short PDUs. The long PDUs have a size of 54 bytes and contain control or user data, see Figure D.4. The DLC SDU, which is passed from or to the DLC layer via the U-SAP, has a length of 49.5 bytes. The remaining 4.5 bytes are used by the DLC for a PDU type field, a sequence number (SN) and a cyclic redundancy check (CRC). The purpose of the CRC is to detect transmission errors and is used, together with the SN, by the EC.

The short PDUs with a size of 9 bytes contain only control data and are always generated by the DLC. They may contain resource requests in the uplink, ARQ messages like acknowledgements and discard messages or RLC information.

The same size of 9 bytes is also used in the RCH. The RCH can only carry RLC messages and resource requests. The access method to the RCH is a slotted aloha scheme. The collision resolution is based on a binary back-off procedure, which is controlled by the MTs. The AP/CC can decide dynamically how many RCH slots it provides per MAC frame.
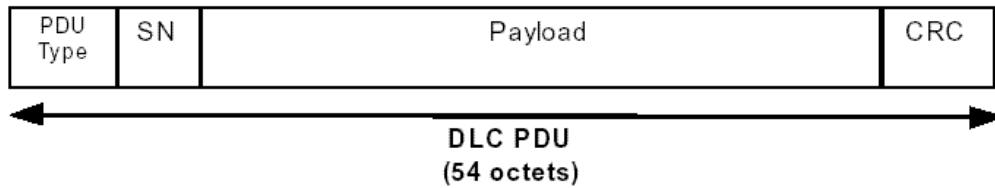
Figure D.4 Format of the Long PDUs

D.1.2.2. Error Control

The EC is based on an ARQ (Automatic Repeat Request) scheme. Additional forward error correction and the EC are complementary but do not collaborate.

The ARQ scheme is based on a selective repeat mechanism. It requires a very careful transmission window handling in both transmitter and receiver. Therefore the receiver has to notify the transmitter about the sequence number below, which all messages have been received correctly (bottom of window) and which messages out of the received ones were not correct. Moreover, the transmitter may want to discard messages, e.g. because they have exceeded their maximum lifetime.

D.1.3. Radio Link Control Functions

NOTE: The control functions are closely related to the protocols defined in the RLC. Only the RLC will be specified, the control functions themselves are out of the scope of the standard. In the explanations below, the control functions and the actual RLC will be handled synonymously.

D.1.3.1.   Association Control Function

A Terminal intending to communicate with an AP/CC has always to be associated to this AP/CC. The reasons are:

- The AP/CC always has to create some resources for each MT associated, e.g. the RLC connection and a MAC ID.
- The MAC protocol is centrally controlled by the AP/CC, regardless of whether it operates in centralized or in direct mode.

The tasks of the association control are:
- Association: The first step is the allocation of a MAC ID to a terminal, followed by the negotiation of the link capabilities. These comprise the selected CL and other features. AP/CC and MT decide in this step whether encryption and / or authentication are performed or not and which encryption and authentication mechanisms are used, respectively.
- Encryption key exchange: This step is performed after the link capability negotiation and is optional. It is based on the Diffie-Hellmann key exchange procedure. The Diffie-Hellmann secret and public values are used by both AP/CC and MT to generate and refresh the session key.
- Authentication: This step is performed after the encryption key exchange and is optional. The authentication affects both MT and AP/CC, i.e. they perform a mutual authentication.
- Beacon Signaling in the AP/CC: The beacon signaling provides basic information about essential features and properties of the AP/CC, which are broadcast in each MAC frame. The ACF provides some of the values that are broadcast.
- Encryption key refresh: This feature is optional. It can happen periodically and is requested by the AP/CC.
- Disassociation: This feature shall be performed by the MT if possible.

NOTE: This may not be possible if the MT power drops suddenly.

D.1.3.2. Radio Resource Control

The radio resource control (RRC) is responsible for the surveillance and efficient use of available frequency resources.

The functions of the RLC for the support of the RRC are:

- Dynamic Frequency Selection: HIPERLAN/2 will operate in a "Plug-and-Play" manner and will not require frequency planning. The decision on the selection of a frequency channel is, in the first step when no MTs are associated, based on the AP/CC's own measurements. During operation, the situation may change and the AP/CC has to switch to a different frequency channel. However, each terminal has a specific interference situation, which may make it impossible for one or more MTs to communicate with the AP/CC efficiently. Therefore, the decision when to perform a frequency change and to which frequency has to be based on both measurements of the AP/CC and the associated MTs. The DFS supporting functions of the RLC allow for:

  - Measurements of MTs and AP/CC: The terminal may do measurements on its own or on different channel, either based on its own decision or ordered by the AP/CC;
  - Reporting of the obtained measurements from MTs to the AP/CC;
  - Frequency change of the AP/CC and its associated MTs.

- MT alive procedure: In order to make sure that the AP/CC does not reserve resources unnecessarily for an MT, the AP/CC may request it to report if it is still alive.
- MT absence function: The MT may want to scan for a different frequency channel in order to find out whether it shall perform a handover and to which new AP/CC it shall change. This function is triggered by the MT.
- Power saving function: Many MTs will be battery driven. Therefore, HIPERLAN/2 will support an efficient scheme to support the conservation of battery power. The mechanism will be based on sleep intervals after which the terminal listens periodically whether the AP/CC wants it to

receive data. If no data are pending in DL, or DiL, the MT remains in sleep modus without communication with the AP/CC in centralized mode or with another MT in direct mode. The length of the sleep intervals can be negotiated between AP/CC and MT. This function is triggered by the AP/CC; the selection of the sleep interval is done by the AP/CC.

- Transmit Power Control: AP/CC and MT will support means to adapt their transmission power to the current requirements of the radio link.

- Handover: The handover function will be restricted to business and public applications and will not be supported in home networks in the first phase. The RRC will decide when to perform a handover and support its execution.

D.1.3.3. DLC Control Function

The DLC connection control (DCC) is responsible for set up and release of user connections. The relation to a higher layer connection set up procedure can be created by a call reference identifier in the DLC connection set up request message. If any kind of QoS support is required by a higher layer, the necessary parameters have to be provided by the higher layers. Since the scheduler will not be specified, the specification of these parameters is out of the scope of HIPERLAN/2. The only DLC related parameters to be exchanged are a DLC Connection ID and ARQ related values like maximum window size and number of allowed retransmissions.

The functions of DCC are:

- DLC connection set up: This feature comprises set up procedures for centralized mode, direct mode and multicasts, all of which can be originated either by the AP/CC or the MT.

- DLC connection release: This feature comprises release procedures for centralized mode, direct mode and multicasts, all of which can be originated either by the AP/CC or the MT.

- DLC connection modify: This feature comprises modify procedures for centralized mode, direct mode and multicasts all of which can be

originated either by the AP/CC or the MT. The modification refers to the DLC specific connection parameters, which are described above.

- Multicast join and leave: These features allow a terminal to join already existing multicast groups and leave one it belongs to.

D.1.4. Convergence Layer

The convergence layers (CL) adapt the core network to the HIPERLAN/2 DLC layer. The CL provides all functions needed for connection set-up and support mobility in the core network. For each supported core network a special CL is designed. Support for packet based networks like Ethernet (IEEE 802.3), IP, PPP and IEEE 1394 (Fire-wire) as well as cell based networks like ATM and UMTS will be available.

The convergence layers available at the AP/CC are announced via broadcast. MT and AP/CC negotiate one of them during association. In combination with the QoS functions of HIPERLAN/2 it shall be possible to support various QoS schemes. Among others IP like RSVP, Differentiated Services or priority scheduling according to IEEE 802.1D.

The packet based convergence layer is used to integrate HIPERLAN/2 into existing packet-based networks. To support the different technologies used nowadays and to be open for future technologies, the Packet CL is structured hierarchically into a common part and a number of service specific convergence sub-layers (SSCS). The common part mainly contains a SAR function to fit the packets into the fixed length of a HIPERLAN/2 packet. The first SSCS to be specified is the Ethernet SSCS, which is followed by IEEE 1394, IP, and PPP SSCSs in the course of the year 2000. For each part a specification will be created.

The ATM CL also consists of a common part and SSCSs. The common part shall not contain a SAR function because ATM cells basically fit into the HIPERLAN/2 DLC SDU. Nevertheless, a compression of the ATM cell header is necessary, transmitting only its most important parts.

D.1.5. HIPERLAN/2 Physical Layer

D.1.5.1. Transport Channels and PDU Trains

The radio subsystem provides a set of transport channels describing the message format over the air interface. Transport channels are used as basic elements in constructing PDU (Protocol Data Unit) trains. The PDU trains that consist of a sequence of transport channels represent the interface between the DLC protocol and the PHY layer. DLC specifies six different PDU train types:

1. Broadcast PDU train;
2. FCH (Frame CHannel) and ACH (Access Feedback CHannel) PDU train;
3. Downlink PDU train;
4. Uplink PDU train with short preamble;
5. Uplink PDU train with long preamble;
6. Direct link PDU train.

D.1.5.2. Reference Configuration

For the purpose of elaborating the specification of physical layer functions, a reference configuration of the transmission chain is used as shown in Figure D.5. It should be noted that only the transmission part is specified.
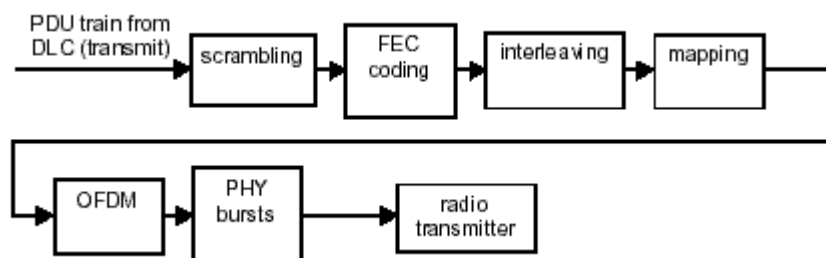
Figure D.5 Reference Configuration of Transmitter

D.1.5.3.   PHY Layer Functional Entities

The PHY layer of HIPERLAN/2 offers information transfer services to the DLC of HIPERLAN/2. For this purpose, it provides for functions to map different DLC PDU trains into framing formats called PHY bursts appropriate for transmitting and receiving management and user information between an AP/CC and an MT in the centralized mode or between two MTs in the direct mode. This includes the following functional entities at transmitter:

- Configuring the transmission bit rate by choosing appropriate PHY mode based on the link adaptation mechanism.
- Scrambling the PDU train content.
- Encoding the scrambled bits according to the forward error correction set during PHY layer configuration.
- Interleaving the encoded bits at the transmitter by using the appropriate interleaving scheme for the selected PHY layer mode.
- Sub-carrier modulation by mapping the interleaved bits into modulation constellation points.
- Producing the complex base-band signal by OFDM modulation.
- Inserting pilot sub-carriers, appending appropriate preamble to the corresponding PDU train at the transmitter and building the PHY layer burst.
- Performing radio transmission by modulating the radio frequency carrier with the complex base-band signal at transmitter.

D.1.5.4.   Physical Layer

D.1.5.4.1. Introduction

The PHY layer of HIPERLAN/2 is based on the modulation scheme Orthogonal Frequency Division Multiplexing (OFDM). In order to improve the radio link capability due to different interference situations and distance of MTs to the access point, a multi-rate PHY layer is applied, where the "appropriate" mode will be selected by a link adaptation scheme. The data rate ranging from 6 Mbit/s to 54 Mbit/s can be varied by using various signal alphabets for modulating the OFDM sub-carriers and by applying different puncturing patterns to a mother convolutional code.

BPSK, QPSK, 16QAM are used as mandatory modulation formats, whereas 64QAM is applied as an optional one for both AP and MT. The mode dependent parameters are listed in the Table D.1.

Modulation	Coding Rate R	Nominal Bit Rate [Mbit/s]	Coded Bits Per Sub-Carrier N_{BPSC}	Coded Bits Per OFDM Symbol N_{CBPS}	Data Bits Per OFDM Symbol N_{DBPS}
BPSK	1/2	6	1	48	24
BPSK	3/4	9	1	48	36
QPSK	1/2	12	2	96	48
QPSK	3/4	18	2	96	72
16QAM	9/16	27	4	192	108
16QAM	3/4	36	4	192	144
64QAM	3/4	54	6	288	216

Table D.1 Mode Dependent Parameters

D.1.5.4.2. Data Scrambling

The content of each PDU train (NBPDU bits) from the DLC shall be scrambled with a length-127 scrambler. The scrambler uses the generator polynomial *S(x)* as given by:

$$S(x) = X^7 + X^4 + 1 \qquad\qquad (D.1)$$

and is illustrated in Figure D.6. The same scrambler shall be used to scramble transmit data and to descramble receive data. All PDU trains belonging to a MAC frame are transmitted by using the same initial state for scrambling. The initialization shall be performed as follows:

- Broadcast PDU train in case AP uses one sector: scrambler initialized at the 5th bit of BCH (Broadcast CHannel), at the 1st bit of FCH and at the 1st bit of ACH;

- Broadcast PDU train in case AP uses one sectors: scrambler initialized at the 5th bit of BCH;

- FCH -and -ACH PDU train transmitted only in the case of a multiple sector AP: scrambler initialized at the 1st bit of FCH and at the 1st bit of ACH;

- Downlink PDU train, Uplink PDU train with short preamble, Uplink PDU train with long preamble and Direct link PDU train: Scrambler initialized at the 1st bit of the PDU train.

The initial state shall be set to a pseudo random non-zero state, which is determined by the Frame counter field in the BCH at the beginning of the corresponding MAC frame. The Frame counter field consists of the first four bits of BCH, represented by $(n_4 n_3 n_2 n_1)_2$, and shall be transmitted unscrambled. n_4 shall be transmitted first. The initial state shall be derived by appending $(n_4 n_3 n_2 n_1)_2$ to the fixed binary number $(111)_2$ in the form $(111\ n_4 n_3 n_2 n_1)_2$.

As an example if the Frame counter is given as $(0100)_2$, the initial state of the scrambler shall be $(111\ 0100)_2$. The transport channel content starting with $(10011101\ 000...)_2$ shall be scrambled to $(00111110\ 011...)_2$.
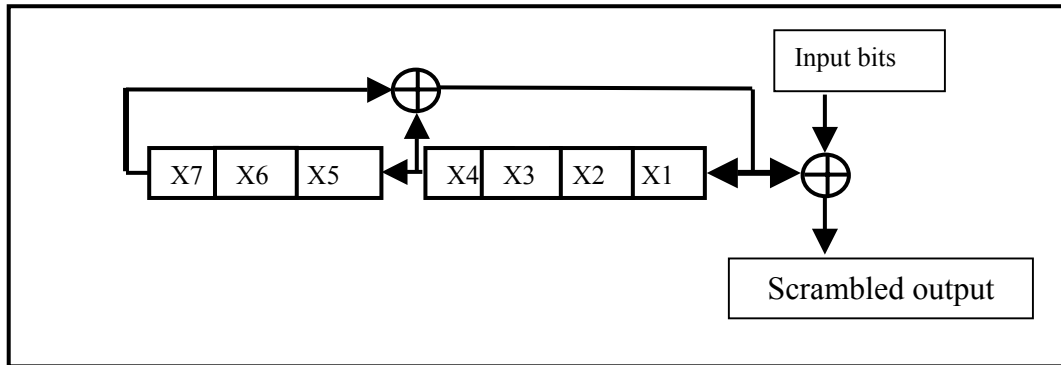
Figure D.6 Scrambler Schematic Diagram

D.1.5.4.3. FEC (Forward Error Correction) Coding

The scrambled PDU train of NBPDU bits shall be encoded by a channel encoder unit. The mandatory encoder is described in this clause and depicted in Figure D.7. It consists of four consecutive operational blocks: code termination, encoding, code rate independent puncturing (P1) and code rate dependent puncturing (P2). It should be noted that this sequence of operation indicates a logical operation of the encoding process, but not a specific implementation.
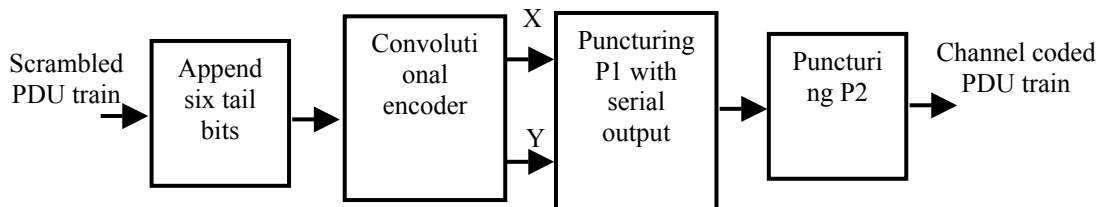
Figure D.7 Functional blocks of FEC coder

The codetermination, encoding, and puncturing P1 shall be performed depending on the PDU train type as follows:

- Broadcast PDU train in omni-antenna case: tail bits shall be appended and puncturing P1 shall be performed individually to BCH, FCH and ACH. The encoder shall be initialized at the 1st bit of BCH, at the 1st bit of FCH and at the 1st bit of ACH;
- Broadcast PDU train in sector-antenna case: tail bits shall be appended and puncturing P1 shall be performed to BCH. The encoder shall be initialized at the 1st bit of BCH;
- FCH and ACH PDU train: tail bits shall be appended and puncturing P1 shall be performed separately to FCH and ACH. The encoder shall be initialized at the 1st bit of FCH, at the 1st bit of ACH without priority, and at the 1st bit of ACH with priority;
- Downlink PDU train, Uplink PDU train with short preamble, Uplink PDU train with long preamble, and Direct link PDU train: Tail bits shall be appended and puncturing P1 shall be performed once for the PDU train. The encoder shall be initialized at the 1st bit of the PDU train.

Puncturing P2 shall be performed equally to all the PDU train types as described in clause D.1.5.4.3.2.

D.1.5.4.3.1. Code Termination, Encoding, P1 Puncturing

D.1.1.5.4.3.1. Downlink PDU Train, Uplink PDU Train with Short and Long Preambles and Direct Link PDU Train

Four of the PDU train types (Downlink PDU train, Uplink PDU train with short preamble, Uplink PDU train with long preamble, and Direct link PDU train) are processed by the encoder as a whole. Tail bits are added once and the respective tail bit puncturing, P1, shall be performed once for the PDU train. The encoder shall also be initialized once at the beginning of the PDU train.

In the first phase six non-scrambled zero ('0') bits are appended to the input data for codetermination purposes. These bits, denoted as tail bits, return the convolutional encoder to "zero state". The resulted (NBPDU + 6) bits shall be coded with a convolutional encoder of code rate 1/2 with 64 states. The generator polynomials of the mother code are G1 = 133OCT for X output and G2 = 171OCT [ITU reference for G1 and G2] for Y output (see Figure D.8). The encoder shall be set to "zero state" before the encoding process.
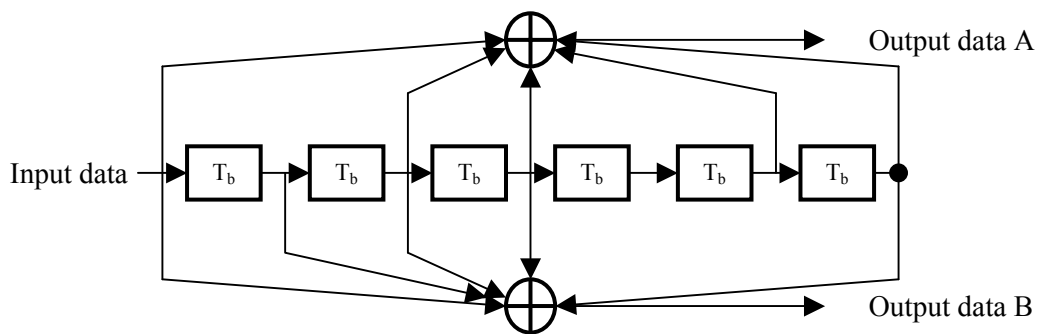
Figure D.8 The mother convolutional code of rate ½

The first puncturing scheme P1 will be applied independently from the code rate. The puncturing shall be applied always to the first SCH-PDU (Short Transport CHannel) of the last DLC Connection of the PDU train to be transmitted over the air interface. If there is no such an SCH-PDU in the last DLC Connection, P1 shall be applied to the first LCH-PDU (Long Transport CHannel) of the last DLC Connection of the PDU train. Four examples of the position of the P1 puncturing inside a PDU train are illustrated in Figure D.9 as informative information.

The first 156 bits of the PDU, which the P1 puncturing is applied to, are punctured differently from the rest of the encoded bit stream. The puncturing patterns are given in Table D.2. In this table X and Y refer to the two outputs of the convolutional encoder (see Figure D.8) where X1 is sent first.

PDU-wise bit numbering	Puncturing pattern	Transmitted sequence (after parallel-to-serial conversion)
0-155	X: 1111110111111 Y: 1111111111110	$X_1Y_1X_2Y_2X_3Y_3X_4Y_4X_5Y_5X_6Y_6X_8Y_7X_9Y_8X_{10}$ $Y_9X_{11}Y_{10}X_{12}Y_{11}X_{13}Y_{12}$
>156	X: 1 Y: 1	X_1Y_1

Table D.2 Puncturing pattern P1 and transmitted sequence after parallel-to-serial conversion
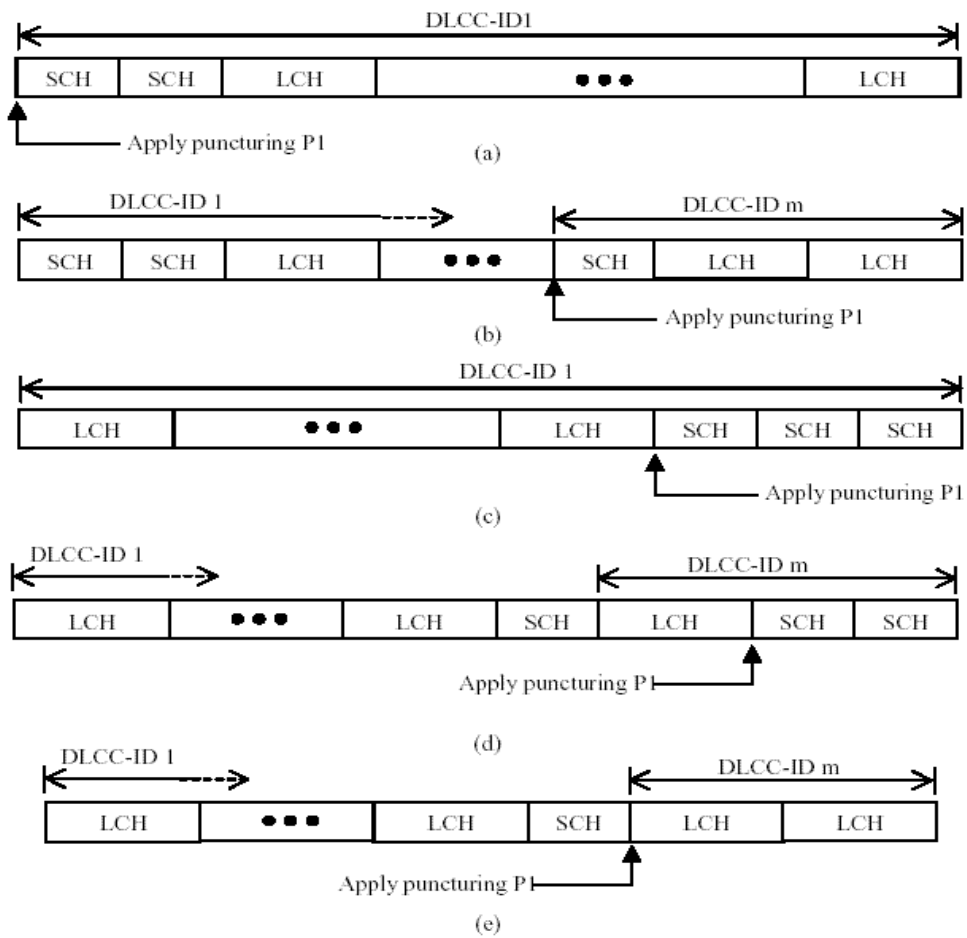
Figure D.9 Position of Puncturing P1 in cases of,

(a) one DLC Connection (DLCC-ID 1) in a downlink PDU train.
(b) two (or more) DLC Connections (DLCC-ID 1…DLCC-ID m) in a downlink PDU train,
(c) one DLC Connection (DLCC-ID 1) in an uplink PDU train,
(d) two (or more) DLC Connections (DLCC-ID 1…DLCC-ID m) in an uplink PDU train, two (or more) DLC Connections (DLCC-ID 1…DLCC-ID m) in an uplink PDU train when no SCH in the last DLC connection

D.1.1.5.4.3.2. Broadcast PDU Train, FCH-and-ACH PDU train

Two of the PDU train types, i.e. Broadcast PDU train and FCH-and-ACH PDU train in the case of a multiple sector AP, are processed transport channel by transport channel. Tail bits shall be appended and additional puncturing shall be performed individually to each transport channel. The encoder shall be also initialized once at the beginning of each transport channel, i.e. at the 1st bit of BCH, FCH and ACH.

In the first phase six non-scrambled zero ('0') bits are appended to each transport channel for codetermination purposes. These bits, denoted as tail bits, return the convolutional encoder to "zero state". The resulted (NBPDU + 6) bits shall be coded with a convolutional encoder of coding rate 1/2 with 64 states. The generator polynomials of the mother code (G1 = 133OCT for X output and G2 = 171OCT for Y output) are the same as used with other PDU train types shown in Figure D.8. The encoder shall be set in "zero state" before the encoding process at the beginning of each transport channel.

The first puncturing scheme P1 will be applied independently from the code rate. The puncturing shall be applied always to all the transport channels in the PDU train equally. The first 156 bits of the transport channel, which the P1 puncturing is applied to, are punctured differently from the rest of the encoded bit stream. The puncturing patterns are given in Table D.2. In this table X and Y refer to the two outputs of the convolutional encoder (see Figure D.8) where X1 is sent first.

D.1.5.4.3.2.Code Rate Dependent Puncturing P2

Puncturing P2 is to provide code rates of 9/16 and 3/4 and it is applied to bits from puncturing P1. It shall be performed equally to all the PDU train types. The input is de-multiplexed into 2 sub-streams. The de-multiplexing is defined as a mapping of the input bits x_{di} onto the output bits $b_{e,do}$ (see Figure D.10):

$$b_{di(mod)2, \; di(div)2} = x_{di} \qquad\qquad (D.2)$$

where di is the input bit number, do is the output bit number in each sub-stream, mod is the integer modulo operator, and div is the integer division operator.
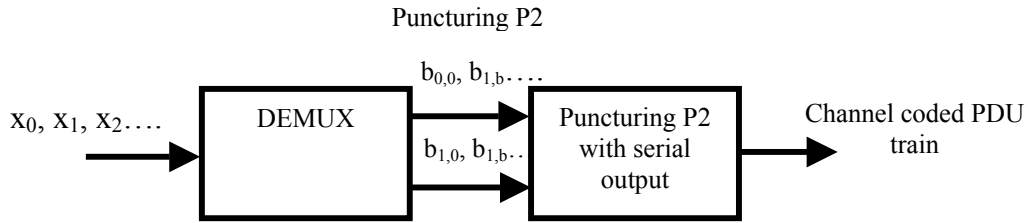
Puncturing P2

Figure D.10 Code Rate Dependent Puncturing P2

Puncturing P2 is applied to the two bit sub-streams $b_{0,do}$ and $b_{1,do}$ as given in Table D.3. The result is parallel-to-serial converted into a coded and punctured bit stream from which $b_{0,0}$ is sent first.

Code Rates r	Puncturing pattern	Transmitted sequence (after parallel-to-serial conversion)
1/2	$b_{0,d0}$: 1 $b_{1,d0}$: 1	$b_{0,0}\ b_{1,0}$
9/16	$b_{0,d0}$: 1 1 1 1 1 1 1 1 0 $b_{1,d0}$: 1 1 1 1 0 1 1 1 1	$b_{0,0}\ b_{1,0}\ b_{0,1}\ b_{1,1}\ b_{0,2}\ b_{1,2}\ b_{0,3}$ $b_{1,3}\ b_{0,4}\ b_{0,5}\ b_{1,5}\ b_{0,6}\ b_{1,6}\ b_{0,7}$ $b_{1,7}\ b_{1,8}$
3/4	$b_{0,d0}$: 1 1 0 $b_{1,d0}$: 1 0 1	$b_{0,0}\ b_{1,0}\ b_{0,1}\ b_{1,2}$

Table D.3 Puncturing pattern P2 and transmitted sequence after parallel-to-serial conversion for the possible code rates

D.1.5.4.4. Data Interleaving

All encoded data bits shall be interleaved by a block interleaver with a block size corresponding to the number of bits in a single OFDM symbol, N_{CBPS}. The interleaver is defined by a two-step permutation. It should be noted that this sequence of permutations is for the ease of the mathematical representation of the interleaving process, but not a specific implementation. The first ensures that adjacent coded bits are mapped onto nonadjacent sub-carriers. The second permutation ensures that adjacent coded bits are mapped alternately onto less and more significant bits of the constellation, and by this long runs of low reliability bits are avoided.

k shall be the index of the coded bit before the first permutation; i shall be the index after the first and before the second permutation and j shall be the index after the second permutation, just prior to modulation mapping.

The first permutation, is defined by the rule:

$$i = (N_{CBPS}/16)(k \bmod 16) + floor(k/16), \; k=0,1,..., N_{CBPS}\text{-}1 \qquad (D.3)$$

The function *floor(.)* denotes here the largest integer not exceeding the parameter, and mod is the integer modulo operator.

The second permutation is defined by the rule:

$$j = s \cdot floor(i/s) + (i + N_{CBPS} - floor(16 \cdot i/N_{CBPS})) \bmod s, \; i = 0,1, \; N_{CBPS} - 1 \quad (D.4)$$

The value of *s* is determined by the number of coded bits per sub-carrier, N_{BPSC}, according to:

$$s = max(N_{BPSC}/2, 1) \qquad (D.5)$$

D.1.5.4.5. Signal Constellations and Mapping

HIPERLAN/2 PHY layer uses Orthogonal Frequency Division Multiplex (OFDM) transmission. The OFDM sub-carriers shall be modulated by using BPSK, QPSK, 16QAM or 64QAM modulation depending on the PHY mode selected for data transmission. The interleaved binary serial input data is divided into groups of N_{BPSC} (1, 2, 4 or 6) bits and converted into complex numbers representing BPSK, QPSK, 16QAM or 64QAM constellation points. The conversion shall be performed according to Gray coded constellation mappings, illustrated in Figure D.11, with the input bit b1 being the earliest in the stream. Additionally, Table D.4 illustrates encoding from input bits to the I and Q values for all the modulations. The output values d are formed by multiplying the resulting $(I + jQ)$ value by a normalization factor K_{MOD}:

$$d = (I + jQ) \ x \ K_{MOD} \qquad\qquad (D.6)$$

The normalization factor K_{MOD} depends on the modulation as prescribed in Table D.4. Note that the modulation type can vary inside a PDU train from one PDU to another while inside one PDU only one modulation type is used. The purpose of the normalization factor is to achieve the same average power for all mappings. The normalization factor K_{MOD} should indicate this fact and no implementation rule.

Modulation	K_{MOD}
BPSK	1
QPSK	$1/\sqrt{2}$
16QAM	$1/\sqrt{10}$
64QAM	$1/\sqrt{42}$

Table D.4 Modulation Dependent Normalization Factor KMOD

BPSK		
Input bit b_1	I-out	Q-out
0	-1	0
1	1	0

QPSK			
Input bit b_1	I-out	Input bit b_2	Q-out
0	-1	0	-1
1	1	1	1

16QAM			
Input bit b_1b_2	I-out	Input b_3b_4	Q-out
00	-3	00	-3
01	-1	01	-1
11	1	11	1
10	3	10	3

64QAM			
Input bit $b_1b_2b_3$	I-out	Input $b_4b_5b_6$	Q-out
000	-7	000	-7
001	-5	001	-5
011	-3	011	-3
010	-1	010	-1
110	1	110	1
111	3	111	3
101	5	101	5
100	7	100	7

Table D.5 Encoding Tables for BPSK, QPSK, 16QAM and 64QAM

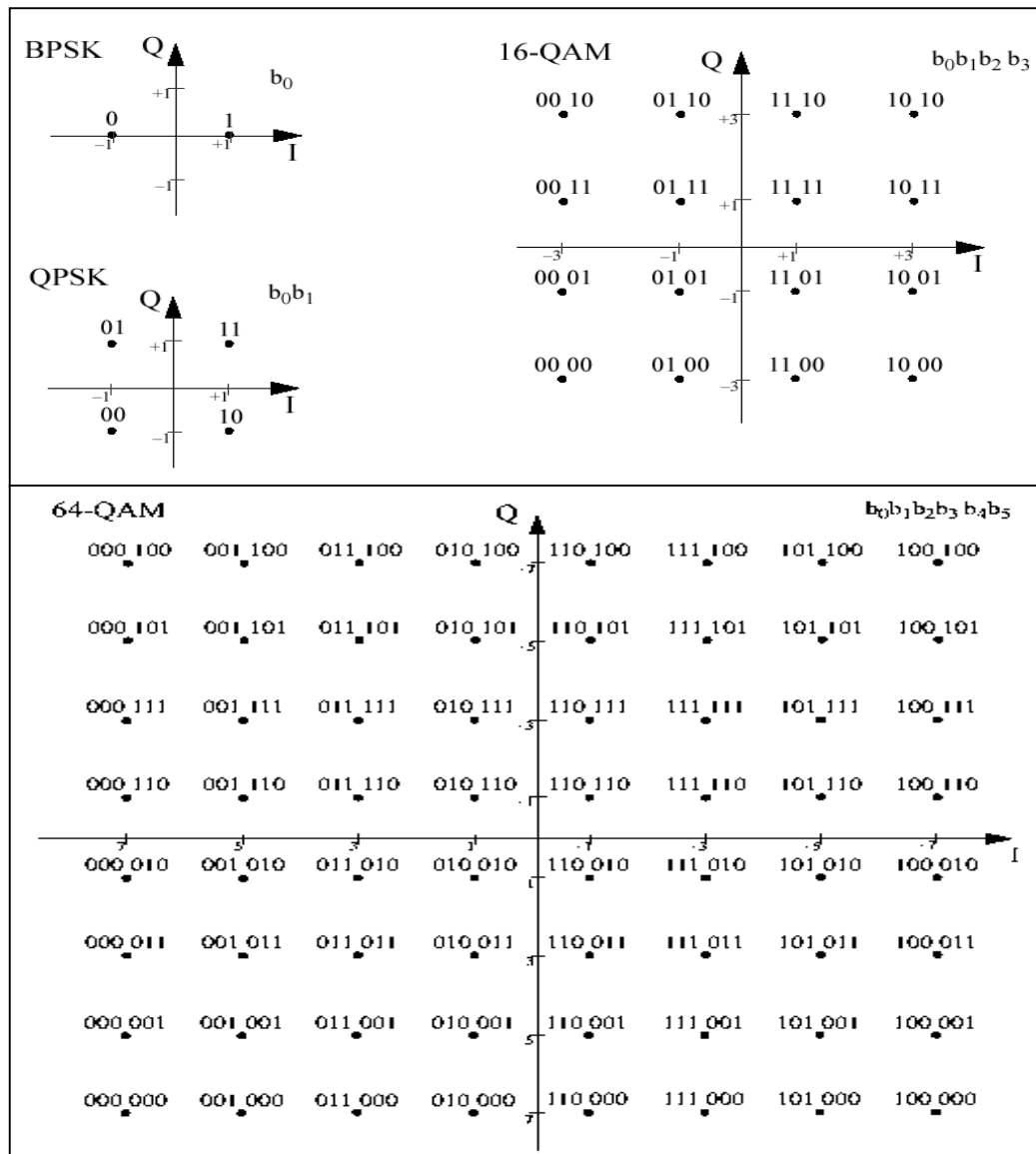Figure D.11 BPSK, QPSK, 16QAM and 64QAM constellation bit encoding

D.1.5.4.6. Modulation Technique

The stream of complex valued sub-carrier modulation symbols at the output of mapper, denoted by d_n, shall be divided into groups of $N_{SD} = 48$ complex numbers:

$$Dn \ (mod) \ 48, \ n \ (div) \ 48 = dn \tag{D.7}$$

where mod is the integer modulo operator and div is the integer division operator.

Each group $D_{m,n}$ shall be transmitted in an OFDM symbol. All data OFDM symbols contain data in data carriers and reference information in pilot carriers. For data there are N_{SD} = 48 carriers and for pilots N_{SP} = 4 carriers in each symbol. Thus, each symbol is constituted by a set of N_{ST} = 52 carriers and transmitted with a duration T_S. Two parts compose this symbol interval: a useful symbol part with duration T_U and a cyclic prefix with duration T_{CP}. The cyclic prefix consists of a cyclic continuation of the useful part, T_U, and it is inserted before it. Thus the cyclic prefix is a copy of the last T_{CP}/T samples of the symbol part sent in front of the symbol part.

The length of the useful symbol part is equal to 64 samples and its duration is T_U = 3,2 μs. For the cyclic prefix length T_{CP} there are two possible values in the HIPERLAN/2 system: mandatory 800 ns and optional 400 ns.

Numerical values for the OFDM parameters are given in Table D.6. The symbol format is shown in Figure D.12 in which CP stands for cyclic prefix followed by a useful part, Data n, of the symbol.

Parameter	Value	
Sampling rate $f_s = 1/T$	20 MHz	
Symbol part duration T_U	64*T 3,2 μs	
Cyclic prefix duration T_{CP}	16*T 0,8 μs (mandatory)	8*T 0,4 μs (optional)
Symbol interval T_S	80*T 4,0 μs (T_U+T_{CP})	72*T 3,6 μs (T_U+T_{CP})
Number of data sub-carriers N_{SD}	48	
Number of pilot sub-carriers N_{SP}	4	
Total number of sub-carriers N_{ST}	52 (N_{SD}+N_{SP})	
Sub-carrier spacing Δ_f	0,3125 MHz (1/T_U)	
Spacing between the two outmost sub-carriers	16,25 MHz (N_{ST}*Δ_f)	

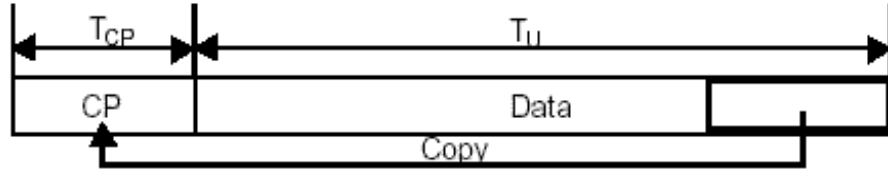Table D.6 Numerical Values for the OFDM Parameters

Figure D.12 Illustration of an OFDM Symbol with Cyclic Prefix

Base-band format of a transmitted OFDM symbol is:

$$r_n(t) = \sum_{l=-N_{ST}/2}^{N_{ST}/2} C_{l,n} \cdot \Psi_{l,n}(t)$$
(D.8)

where:

$$\Psi_{l,n}(t) = \begin{cases} e^{j2\pi l\Delta_f(t-T_{CP}-nT_S)} & ,nT_S \leq t \leq (n+1)T_S \\ 0 & ,\text{else} \end{cases}$$
(D.9)

where:

n denotes the OFDM symbol number;

l denotes the sub-carrier number;

$C_{l,n}$ is complex symbol (data or pilot) for carrier l of the OFDM symbol no. n.

The carriers used for data transmission are:

$$-26 \leq l \leq -22, -20 \leq l \leq -8, -6 \leq l \leq -1, 1 \leq l \leq 6, 8 \leq l \leq 20, 22 \leq l \leq 26$$

and the pilot carriers for reference signal transmissions are:

$$l = -21, -7, 7, 21$$

The sub-carrier falling at D.C. (0-th sub-carrier, $l = 0$) is not used.

The mapping from an individual data symbol group $D_{m,n}$ into symbols $C_{l,n}$ is defined as:

$$C_{l,n} = \begin{cases} D_{l+26,n}, -26 \leq l \leq -22 \\ D_{l+25,n}, -20 \leq l \leq -8 \\ D_{l+24,n}, -6 \leq l \leq -1 \\ D_{l+23,n}, 1 \leq l \leq 6 \\ D_{l+22,n}, 8 \leq l \leq 20 \\ D_{l+21,n}, 22 \leq l \leq 26 \end{cases} \tag{D.10}$$

The reference signal transmitted in the pilot carriers is defined as:

$$C_{l,n} = \begin{cases} + p_n, l = -21 \\ + p_n, l = -7 \\ + p_n, l = 7 \\ - p_n, l = 21 \end{cases} \tag{D.11}$$

where p_n is a sequence to randomize the reference signal transmitted. The sequence p_n is a cyclic extension of the 127-element sequence given by:

$p_{0...126}$ = {1, 1, 1, 1, -1, -1, -1, 1, -1, -1, -1, -1, 1, 1, -1, 1, -1, -1, 1, 1, -1, 1, 1, -1, 1, 1, 1, 1, 1, 1, -1, 1, 1, 1, -1, 1, 1, -1, -1, 1, 1, 1, -1, 1, -1, -1, -1, 1, -1, 1, -1, -1, 1, -1, -1, 1, 1, 1, 1, 1, -1, -1, 1, 1, -1, -1, 1, -1, 1, -1, 1, 1, -1, -1, -1, 1, 1, -1, -1, -1, -1, 1, -1, -1, 1, -1, 1, 1, 1, 1, -1, 1, -1, 1, -1, 1, -1, -1, -1, -1, -1, 1, -1, 1, 1, -1, 1, -1, 1, 1, 1, -1, -1, 1, -1, -1, -1, 1, 1, 1, -1, -1, -1, -1, -1, -1, -1}

The sequence p_n can be generated with the polynomial $S(x)$ used in data scrambling (see Figure D.6):

$$S(x) = X^7 + X^4 + 1 \qquad \text{(D.12)}$$

when the "all ones" (1111111) initial state is used, and by replacing all '1's with -1 and all '0's with 1. Each sequence element is used for one OFDM symbol. This scrambler shall be initialized at the beginning of all PDU trains.

The mapping from data and pilot complex symbols into the sub-carrier frequencies is shown in Figure D.13.
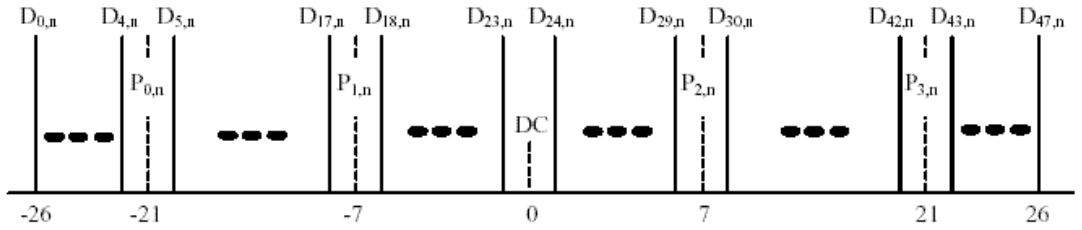

Figure D.13 Sub-carrier Frequency Allocation

The resulted N_{SYM} OFDM symbols are concatenated as:

$$r_{PAYLOAD}(f) = \sum_{n=1}^{N_{SYM}} r_n(t - nT_s) \qquad \text{(D.13)}$$

to result the base-band format of the PDU train, called payload. The structure of the payload section is illustrated in Figure D.14. It consists of variable number (N_{SYM}) of OFDM symbols required to transmit the PDU train payload.

The following relation relates the actual transmitted signal to the complex base-band signal:

$$r_{RF}(t) = \sqrt{2} \, \text{Re}\left\{ r_{BURST}(t) e^{j2\pi f_c t} \right\} \qquad \text{(D.14)}$$

where Re(.) stands for the real part of complex variable, f_c denotes the carrier center frequency, and $r_{BURST}(t)$ is base-band format of a PHY burst composed of payload and preamble and is defined in the following clause.
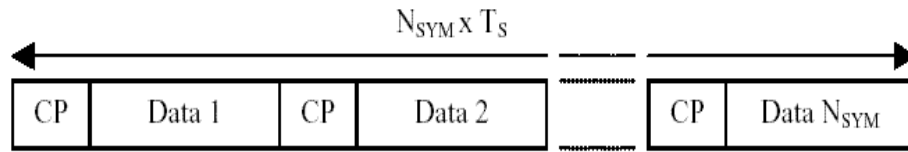
148

Figure D.14 PDU Train Payload ($r_{PAYLOAD}$) format

D.1.5.4.7. PHY Bursts

System has five different kinds of PHY bursts:

1. Broadcast burst;
2. Downlink burst;
3. Uplink burst with short preamble;
4. Uplink burst with long preamble;
5. Direct link burst (optional).

The PDU trains delivered by DLC are mapped onto the PHY bursts as depicted below depending on the number of sectors used by AP.

a. Number of sectors per AP=1.

In this case, the Broadcast PDU train shall be concatenated to FCH-and-ACH PDU train and the resulting Broadcast PDU train is mapped onto the Broadcast burst.
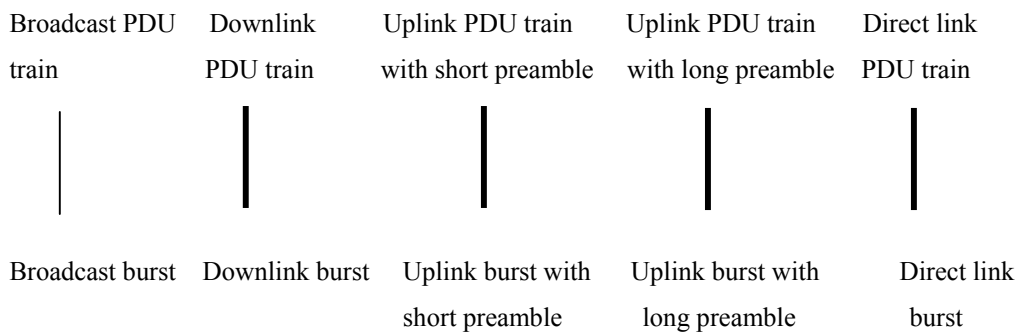
b. Number of sectors per AP>1

149

In this case only the Broadcast PDU train shall be mapped onto the Broadcast burst. The FCH-and-ACH PDU train shall be mapped onto a downlink burst.

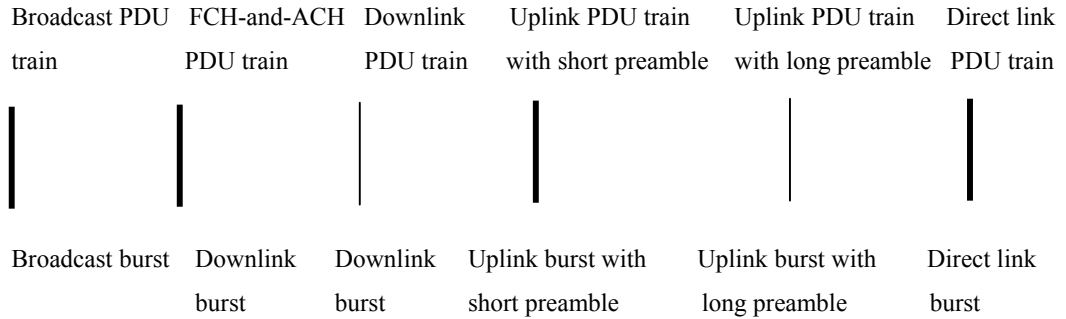| Broadcast PDU train	FCH-and-ACH PDU train	Downlink PDU train	Uplink PDU train with short preamble	Uplink PDU train with long preamble	Direct link PDU train
Broadcast burst	Downlink burst	Downlink burst	Uplink burst with short preamble	Uplink burst with long preamble	Direct link burst

Independently of the burst type each burst consists of two sections: preamble and payload. Each burst is started with a preamble section, $r_{PREAMBLE}$, which is followed by a payload section, $r_{PAYLOAD}$, and its base-band format is:

$$r_{BURST}(t) = r_{PREAMBLE}(t) + r_{PAYLOAD}(t - t_{PREAMBLE}) \qquad (D.15)$$

The time-offset $t_{PREAMBLE}$ determines the starting point of the payload section of the burst and depends on the burst type. The basic structure of a PHY burst is illustrated in Figure D.15.

Preamble $r_{PREAMBLE}$	Payload $r_{PAYLOAD}$

Figure D.15 PHY burst format

D.1.5.4.7.1.Broadcast burst

Broadcast burst consists of a preamble of length $t_{PREAMBLE}$ = 16,0 μs and a payload section of length N_{SYM} x T_S. Structure of the broadcast burst preamble is illustrated in Figure D.16.
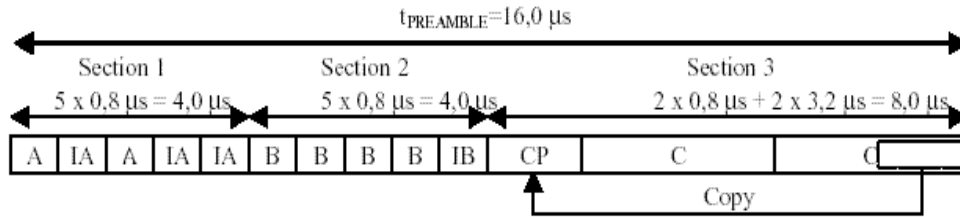

Figure D.16 Broadcast Burst Preamble

In below the term "short OFDM symbol" refers only to its length that is 16 samples instead of a regular OFDM symbol of 64 samples used in HIPERLAN/2 system.

The broadcast burst preamble is composed of three sections: section 1, section 2 and section 3.

Section 1 consists of 5 specific short OFDM symbols that are denoted in Figure D.16 by A and IA. The first 4 short OFDM symbols in section 1 (A, IA, A, IA) constitute a regular OFDM symbol consisting of 12 loaded sub-carriers (±2, ±6, ±10, ±14, ±18, and ±22) given by the frequency-domain sequence SA:

$$SA_{-26...26} = \sqrt{(13/6) \times \begin{cases} 0,0,0,0,-1+j,0,0,0,1+j,0,0,0,1-j,0,0,0,-1-j,0,0,0,-1+j, \\ 0,0,0,-1-j,0,0,0,-1+j,0,0,0,-1-j,0,0,0,-1+j,0,0,0,-1-j, \\ 0,0,0,1-j,0,0,0,1+j,0,0,0,0 \end{cases}}$$

The last short symbol in section 1 (IA) is a repetition of preceding 16 time-domain samples.

Section 2 consists of 5 specific short OFDM symbols that are denoted in Figure D.16 by B and IB. The first 4 short OFDM symbols in section 2 (B, B, B, B) constitute

a regular OFDM symbol consisting of 12 loaded sub-carriers (±4, ±8, ±12, ±16, ±20, and ±24) given by the frequency-domain sequence SB:

$$SB_{-26\ldots26} = \sqrt{(13/6) \times \left\{ \begin{array}{l} 0, 0, 1+j, 0, 0, 0, -1-j, 0, 0, 0, 1+j, 0, 0, 0, -1-j, 0, 0, 0, -1-j, 0, 0, \\ 0, 1+j, 0, 0, 0, 0, 0, 0, 0, -1-j, 0, 0, 0, -1-j, 0, 0, 0, 1+j, 0, \\ 0, 0, 1+j, 0, 0, 0, 1+j, 0, 0, 0, 1+j, 0, 0 \end{array} \right\}}$$

The last short symbol in section 2 (IB) is a sign-inverted copy of the preceding short symbol B, i.e. IB = -B.

Section 3 consists of two OFDM symbols (C) of normal length preceded by a cyclic prefix (CP) of the symbols. All the 52 sub-carriers are in use and they are modulated by the elements of the frequency-domain sequence SC given by:

$SC_{-26\ldots26}$ = {1, 1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, 1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 0, 1,

-1, -1, 1, 1, -1, 1, -1, 1, -1, -1, -1, -1, -1, 1, 1, -1, -1, 1, -1, 1, -1, 1, 1, 1, 1}

The cyclic prefix CP is a copy of the 32 last samples of the C symbols and is thus double in length compared to the cyclic prefix of the normal data symbols.

The broadcast burst is formed by concatenating the above-described preamble with the data payload. The resulted broadcast burst is as illustrated in Figure D.21 a.

D.1.5.4.7.2.Downlink Burst

Downlink burst consists of a preamble of length = 8,0 μs and a payload section of length $N_{SYM} x T_S$. Structure of the downlink burst preamble is illustrated in Figure D.17.
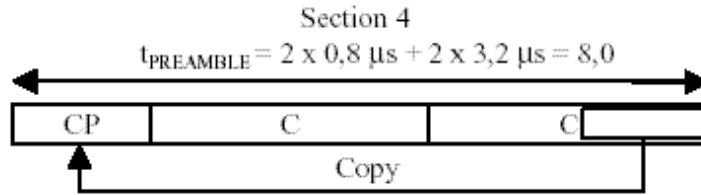
Figure D.17 Downlink Burst Preamble

The downlink burst preamble is equal to the section 3 of the broadcast burst preamble. It is composed of two OFDM symbols (C) of normal length preceded by a cyclic repetition (CP) of the symbols. All the 52 sub-carriers are in use and they are modulated by the elements of the frequency-domain sequence SC given by:

$$SC_{-26\ldots26} = \{1, 1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, 1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 0, 1,$$
$$-1, -1, 1, 1, -1, 1, -1, 1, -1, -1, -1, -1, -1, 1, 1, -1, -1, 1, -1, 1, -1, 1, 1, 1, 1\}$$

The cyclic repetition CP is a copy of the 32 last samples of the C symbols and is thus double in length compared to the cyclic prefix of the normal data symbols.

The downlink burst is formed by concatenating the above - described preamble with the data payload. The resulted downlink burst is as illustrated in Figure D.21 b.

D.1.5.4.7.3.Uplink Burst with Short Preamble

It consists of a preamble of length $t_{PREAMBLE} = 12,0$ μs and a payload section of length $N_{SYM} x T_S$. Structure of the short preamble for uplink bursts is illustrated in Figure D.18.
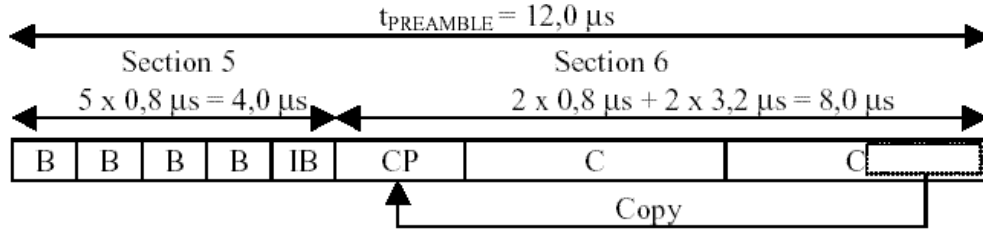
Figure D.18 Short Preamble for Uplink Bursts

In below the term "short OFDM symbol" refers only to its length that is 16 samples instead of a regular OFDM symbol of 64 samples used in HIPERLAN/2 system.

The short preamble for uplink bursts is composed of two sections: section 5 and section 6. The sections are equal to the latter two sections of the broadcast burst preamble: section 5 = section 2, section 6 = section 3.

Section 5 consists of 5 specific short OFDM symbols denoted in Figure D.18 by B and IB. The first 4 short OFDM symbols in this section (B, B, B, B) constitute a regular OFDM symbol consisting of 12 loaded sub-carriers (± 4, ± 8, ± 12, ± 16, ± 20, and ± 24) given by the frequency-domain sequence SB:

$$SB_{-26....26} = \sqrt{(13/6) \times \begin{cases} 0,0,1+j,0,0,0,-1-j,0,0,0,1+j,0,0,0,-1-j,0,0,0,-1-j,0,0, \\ 0,1+j,0,0,0,0,0,0,0,-1-j,0,0,0,-1-j,0,0,0,1+j,0, \\ 0,0,1+j,0,0,0,1+j,0,0,0,1+j,0,0 \end{cases}}$$

The last short symbol in section 5 (IB) is a sign-inverted copy of the preceding short symbol B, i.e. IB = -B.

Section 6 consists of two OFDM symbols (C) of normal length preceded by a cyclic repetition (CP) of the symbols. All the 52 sub-carriers are in use and they are modulated by the elements of the frequency-domain sequence SC given by:

$$SC_{-26...26} = \{1, 1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, 1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 0, 1,$$
$$-1, -1, 1, 1, -1, 1, -1, 1, -1, -1, -1, -1, -1, 1, 1, -1, -1, 1, -1, 1, -1, 1, 1, 1, 1\}$$

154

The cyclic repetition CP is a copy of the 32 last samples of the C symbols and is thus double in length compared to the cyclic prefix of the normal data symbols.

The uplink burst with short preamble is formed by concatenating the above - described preamble with the data payload. The resulted uplink burst is as illustrated in Figure D.21 c.

D.1.5.4.7.4.Uplink Burst with Long Preamble

It consists of a preamble of length $t_{PREAMBLE}$ = 16,0 µs and a payload section of length $N_{SYM} x T_S$. Structure of the long preamble for uplink bursts is illustrated in Figure D.19.
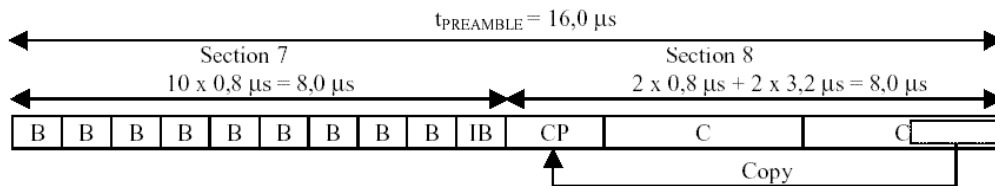

Figure D.19 Long Preamble for Uplink Bursts

In below the term "short OFDM symbol" refers only to its length that is 16 samples instead of a regular OFDM symbol of 64 samples used in HIPERLAN/2 system.

The long preamble for uplink bursts is composed of two sections: section 7 and section 8.

Section 7 consists of 10 specific short OFDM symbols denoted in figure 15 by B and IB. The first 4 short OFDM symbols in this section (B, B, B, B) constitute a regular OFDM symbol consisting of 12 loaded sub-carriers (±4, ±8, ±12, ±16, ±20, and ±24) given by the frequency-domain sequence SB:

$$SB_{-26...26} = \sqrt{(13/6) \times \begin{cases} 0,0,1+j,0,0,0,-1-j,0,0,0,1+j,0,0,0,-1-j,0,0,0,-1-j,0,0, \\ 0,1+j,0,0,0,0,0,0,0,-1-j,0,0,0,-1-j,0,0,0,1+j,0, \\ 0,0,1+j,0,0,0,1+j,0,0,0,1+j,0,0 \end{cases}}$$

The last short symbol in section 7 (IB) is a sign-inverted copy of the preceding short symbol B, i.e. IB = -B.

Section 8 consists of two OFDM symbols (C) of normal length preceded by a cyclic repetition (CP) of the symbols. All the 52 sub-carriers are in use and they are modulated by the elements of the frequency-domain sequence SC given by:

$SC_{-26...26}$ = {1, 1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, 1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 0,

1, -1, -1, 1, 1, -1, 1, -1, 1, -1, -1, -1, -1, -1, 1, 1, -1, -1, 1, -1, 1, -1, 1, 1, 1, 1}

The cyclic repetition CP is a copy of the 32 last samples of the C symbols and is thus double in length compared to the cyclic prefix of the normal data symbols. Thus the section 8 is equal to the section 3, section 4, and section 6.

The uplink burst with long preamble is formed by concatenating the above - described preamble with the data payload. The resulted uplink burst is as illustrated in Figure D.21 d.

D.1.5.4.7.5. Direct Link Burst

Direct link burst is optional. It consists of a preamble of length $t_{PREAMBLE}$ = 16,0 μs and a payload section of length $N_{SYM} \times T_S$. Structure of the preamble for direct link bursts is illustrated in Figure D.20.
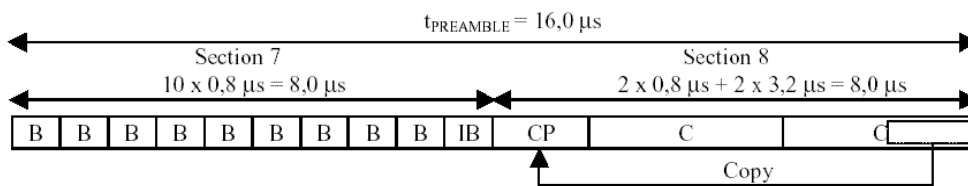

Figure D.20 Preamble for Direct Link Bursts

In below the term "short OFDM symbol" refers only to its length that is 16 samples instead of a regular OFDM symbol of 64 samples used in HIPERLAN/2 system.

The preamble for direct link bursts is composed of two sections: section 7 and section 8.

Section 7 consists of 10 specific short OFDM symbols denoted in Figure D.20 by B and IB. The first 4 short OFDM symbols in this section (B, B, B, B) constitute a regular OFDM symbol consisting of 12 loaded sub-carriers 6(±4, ±8, ±12, ±16, ±20, and ±24) given by the frequency sequence SB:

$$
SB_{-26...26} = \sqrt{(13/6) \, x \begin{cases} 0,0,1+j,0,0,0,-1-j,0,0,0,1+j,0,0,0,-1-j,0,0,0,-1-j,0,0, \\ 0,1+j,0,0,0,0,0,0,0,-1-j,0,0,0,-1-j,0,0,0,1+j,0, \\ 0,0,1+j,0,0,0,1+j,0,0,0,1+j,0,0 \end{cases}}
$$

Section 8 consists of two OFDM symbols (C) of normal length preceded by a cyclic repetition (CP) of the symbols. All the 52 sub-carriers are in use and they are modulated by the elements of the frequency-domain sequence SC given by:

$$
SC_{-26...26} = \{1, 1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, 1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 0, \\
1, -1, -1, 1, 1, -1, 1, -1, 1, -1, -1, -1, -1, -1, 1, 1, -1, -1, 1, -1, 1, -1, 1, 1, 1, 1\}
$$

The cyclic repetition CP is a copy of the 32 last samples of the C symbols and is thus double in length compared to the cyclic prefix of the normal data symbols. Thus the section 7 is equal to the section 3, section 4, and section 6.

The direct link burst is formed by concatenating the above - described preamble with the data payload. The resulted direct link burst is as illustrated in Figure D.21 e.
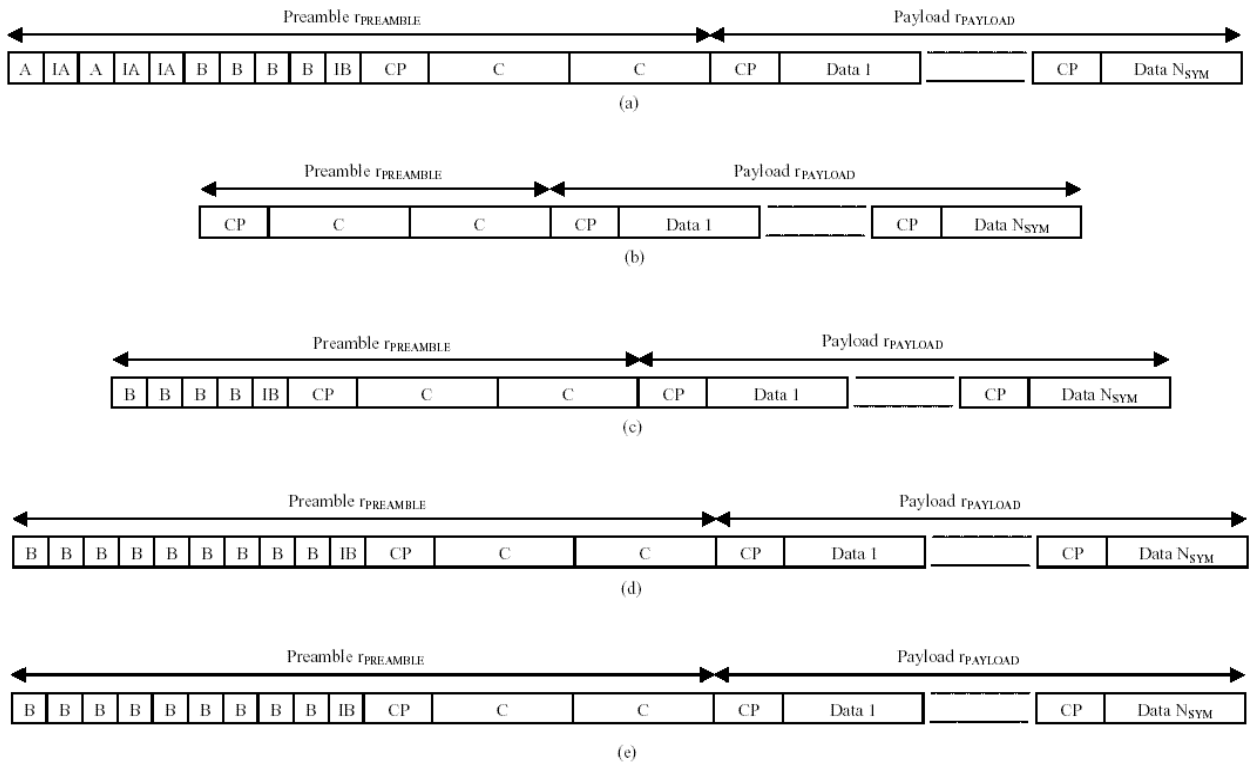
Figure D.21 PHY burst structures: (a) Broadcast burst, (b) Downlink burst, (c) Uplink burst with short preamble, (d) Uplink burst with long preamble, (e) Direct link burst

APPENDIX E: TOOLS THAT WERE USED

During this thesis, the tools listed in Table E.1 have been used.

Tool	Version
Cadence Affirma NC Simulator	3.0
Synopsys Design Analyzer	1999.10-4
Matlab	6.1
Simulink	4.1
MS Word	2000

Table E.1 Tools that were used

REFERENCES

1. R. W. Chang, *Synthesis of Band Limited Orthogonal Signals for Multi-channel Data Transmission*, Bell Syst. Tech. J., Vol. 45, pp. 1775-1796, Dec. 1966.

2. B. R. Salzberg, *Performance of an efficient parallel data transmission system*, IEEE Trans. Comm., Vol. COM-15, pp. 805-813, Dec. 1967.

3. R. Prasad, *Wireless Broadband Communication Systems*, IEEE Comm. Mag., Vol. 35, pp. 18, Jan. 1998

4. A. John, C. Bingham, *Multi-carrier Modulation for Data Transmission: An Idea Whose Time Has Come,* IEEE Communication Magazine, 37:5–14, May 1990.

5. http://www.sce.carleton.ca/~Laszlo.Hazy/OFDM/ofdm.html

6. W. Y. Zhou, Y. Wu, *COFDM: an overview,* IEEE Transactions on Communications, Vol. 41, No: 1, pp. 1-8, March 1995.

7. ETSI, *Broadband Radio Access Networks (BRAN); HIPERLAN TYPE 2 Technical Specification; Physical (PHY) Layer*, 2001.

8. M. Speth, A. Fechtel, G. Fock and H. Meyr, *Optimum Receiver Design for Wireless Broadband Systems Using OFDM-Part1*, IEEE Transactions on Communications, Vol. 47, No. 11, pp. 1668-1676, November 1999.

9. T. M. Schmidl and D. C. Cox, *Robust Frequency and Timing Synchronization for OFDM*, IEEE Transactions on Communications, Vol. 45, No. 12, pp. 1613-1621, December 1997.

10. F. Adachi, *OFDM for the New 5 GHz Wireless LAN standards*, IEEE Communications Magazine, pp. 84-86, December 1999.

11. T. Pollet, M. van Bladel and M. Moeneclaey, *BER Sensitivity of OFDM Systems to Carrier Frequency Offset and Wiener Phase Noise*, IEEE Transactions on Communications, Vol. 43, No. 2/3/4, pp. 191-193, Feb.-Apr. 1995.

12. P. H. Moose, *A technique for Orthogonal Frequency Division Multiplexing Frequency Offset Correction*, IEEE Transactions on Communications, Vol. 42, No. 10, pp. 2908-2914, Oct. 1994

13. R. Böhnke and T. Dölle, *Preamble Structures for HiperLAN Type 2 Systems*, ETSI BRAN Document No. HL13SON1A, Apr. 7, 1999

14. W. D. Warner and C. Leung, *OFDM/FM Frame Synchronization for Mobile Radio Data Communication*, IEEE Transactions on Communications, Vol.42, No. 3, pp. 302-313, Aug. 1993.

15. L. Tomba, *On the Effects of Wiener Phase Noise in OFDM systems*, IEEE Transactions on Communications, Vol.46, No. 5, pp. 580-583, May 1998.

16. Fredrik Tufvesson and Ove Edfors, *Preamble-based Time and Frequency Synchronization for OFDM systems*, submitted to IEEE Journal on Selected Areas in Communications, January 2000.

17. J. Armstrong, *Analysis of new and existing methods of reducing inter-carrier interference due to carrier frequency offset in OFDM*, IEEE Transactions on Communications, Vol.47, No. 3, March 1999.

18. J. J. Van de Beek, M. Sandell, M. Isaksson and P. O. Börjesson, *Low-Complex Frame Synchronization in OFDM Systems*, Proceedings of International Conference on Universal Personal Communications ICUP' 95, Nov. 1995.

19. M. Sandell, J. J. Van de Beek and P. O. Börjesson, *Timing and Frequency Synchronization in OFDM Systems Using the Cyclic Prefix*, Proceedings of International Symp. On Synchronization, Saalbau, Essen, Germany, 1995, pp. 16-19, Dec. 14-15, 1995.

20. T. N. Zogakis and J. M. Cioffi, *The effect of Timing Jitter on the Performance of a Discrete Multitone System,* IEEE Transactions on Communications, Vol. Com-33, NO. 6**,** June 1985.

21. U. Lambrette, M. Speth and H. Meyr, *OFDM Burst Frequency Synchronization by Single Carrier Training Data*, IEEE Communication Letters, Vol. 44, No. 7, pp. 799-808, July 1996.

22. Ali Zamanian, *Orthogonal Frequency Division Multiplex Overview*, Microwave Journal, October 2001

23. *Broadband Radio Access Networks, HIPERLAN Type 2, Physical (PHY) Layer,* ETSI TS 101 475 Technical Specification, Apr. 2000.

24. W. Eberle, M. Badaroglu, V. Derudder, S. Thoen, P. Vandenameele, L. Van der Perre, M. Vergara, B. Gyselinckx, M. Engels, and I. Bolsens, *A digital 80 Mb/s OFDM transceiver IC for wireless LAN in the 5 GHz band*, in Proc. IEEE Int. Solid State Circuits Conf. (ISSCC), pp. 74-75, Feb. 2000.

25. W. Eberle, V. Derudder, G. Vanwijnsberghe, M. Vergara, L. Deneire, L. Van der Perre, Marc G. E. Engels, I. Bolsens, and Hugo De Man, *80-Mb/s QPSK and 72-Mb/s 64-QAM Flexible and Scalable Digital OFDM Transceiver ASICs for Wireless Local Area Networks in the 5-GHz Band*, IEEE Journal of Solid-State Circuits, Vol. 36, No. 11, NOVEMBER 2001.

26. Baoguo Yang, Khaled Ben Letaief, Roger S. Cheng, and Zhigang Cao, *Timing Recovery for OFDM Transmission,* IEEE Journal on Selected Areas In Communications, Vol. 18, No. 11, Nov. 2000.

27. L. M. Leibowitz, *Multiplexing Techniques for Digital Correlator Speed Improvement*, IEEE Transactions on Communications, Vol. Com-33, NO. 6**,** June 1985.

28. R. Andraka, *A Survey of CORDIC Algorithms for FPGA based computers*, Andraka Consulting Group, In 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays.

29. S. Johansson, D. Landström and Peter Nilsson, *Hardware Implementation of an OFDM Synchronizer*, Department of Applied Electronics, Lund, Sweden, July 1999.

30. Randy Roberts, *Technical Tricks, February 1993: Correlators*, RF/SS Consulting

31. Yun Hee Kim, *An efficient frequency offset estimator for timing and frequency synchronization in OFDM system*, IEEE Pacific Rim Conference On Communications, Computers and Signal Processing, pp 580–583, 1999

32. Jan-Jaap van de Beek, M. Sandell, and P.O. Börjesson, *ML estimation of time and frequency offset in OFDM systems*, IEEE Trans. on Signal Proc., 45(7): 1800–1805, July 1997.

33. Richard Herveille, *Cordic Core Specification*, http://www.opencores.org

34. S. Haykin, *Communication Systems*, 3rd edition, New York, NY: Wiley, 1994.

35. http://www.etsi.org/bran/bran.html

36. Richard Van Nee, Ramjee Prasad, *Wireless Multimedia Communications*, Artech House Universal Personal Communication Library, 2000.

37. Robertson, P., and S. Kaiser, *Analysis of the Effects of Phase-Noise in OFDM Systems*, Proceedings of IEEE VTC'95, pp. 1652-1657.

38. J. Proakis, *Digital Communications, 3rd edition*, McGrawHill, 1995.

39. http://www.hiperlan2.com/web/