OFDM with SDR
Wireless Communication

(B-KUL-JLI1CG)

Gilles Callebaut

February 18, 2024

Contents

Chapter 1

Abbreviations

ADC analog-to-digital converter. 10, 19, 20

API application program interface. 22

AWGN additive white Gaussian noise. 6, 9, 15, 17

BER bit error rate. 16–18

BPSK binary phase-shift keying. 16, 17

CFO carrier frequency offset. 10, 11

CP cyclic prefix. 8

CPU central-processing unit. 21

DAB Digital Audio Broadcasting. 5

DAC digital-to-analog converter. 19, 20

DFT discrete Fourier transform. 5, 7, 8, 10, 11, 14

DSL digital subscriber line. 5

DSP digital signal processing. 19

DVB Digital Video Broadcasting. 5

FFT fast Fourier transform. 5

FIFO First In, First Out. 23

GPL GNU General Public License. 22

GPS Global Positioning System. 21

GRC GNU Radio Companion. 23, 29

ICI intercarrier interference. 10, 12

IDFT inverse discrete Fourier transform. 7, 8

IF intermediate-frequency. 19–21

ISI intersymbol interference. 5, 8

LO local oscillator. 10

LTE Long Term Evolution. 5, 15, 21

OFDM orthogonal frequency-division multiplexing. 5–9, 13–16

OS operating system. 25

PCIe Peripheral Component Interconnect Express. 20

PSK phase shift keying. 8, 16, 17

QAM quadrature amplitude modulation. 8, 17, 18

QPSK quadrature phase-shift keying. 16, 17

RF radio frequency. 5, 19–22

SDR software-defined radio. 5, 6, 19–22

SER symbol-error rate. 17

SMA SubMiniature version A. 27

SNR signal-to-noise ratio. 9, 13, 17, 18

TO timing offset. 13, 14

USRP universal software radio peripheral. 1, 5, 22, 28

VNA vector network analyzer. 27

WiMAX Worldwide Interoperability for Microwave Access. 5, 18

WLAN wireless LAN. 5, 18

ZF zero-forcing. 14

Chapter 2

Introduction

Current broadband wireless standards are based on orthogonal frequency-division multiplexing (OFDM), a multi-carrier modulation scheme which provides strong robustness against intersymbol interference (ISI) by dividing the broadband channel into many orthogonal narrowband subchannels in such a way that attenuation across each subchannel stays flat. Orthogonalization of subchannels is performed with low complexity by using the fast Fourier transform (FFT), an efficient implementation of discrete Fourier transform (DFT), such that the serial high-rate data stream is converted into multiple parallel low-rate streams, each modulated on a different subcarrier.

Many of the currently used wireless standards are using OFDM for the physical layer, e.g., wireless LAN (WLAN), Worldwide Interoperability for Microwave Access (WiMAX), Digital Audio Broadcasting (DAB), Digital Video Broadcasting (DVB), digital subscriber line (DSL), Long Term Evolution (LTE), etc. Beside the existing systems there is active research on future systems enhancing the existing standards to improve system performance. The investigation and assessment of information theoretic concepts for wireless resource management of those new systems in real-world scenarios requires flexible testbeds with a wide range of reconfigurable parameters. This functionality is currently offered in software-defined radio (SDR) technology based on general purpose hardware only.

We designed a modular, SDR based and reconfigurable framework which treats the OFDM transmission link as a black box. The given framework contains transmitter and receiver nodes that are composed of a host commodity computer and a general purpose radio frequency (RF) hardware, namely USRP. Baseband OFDM signal processing at host computers is implemented in the GNU Radio framework, an open source, free software toolkit for building SDRs [1].

The control and feedback mechanisms provided by the given framework allow for reconfigurable assignments of predefined transmission parameters at the input and estimation of link quality at the output. High flexibility, provided by a large set of reconfigurable parameters, which are normally static in real systems, enables implementation and assessment of different signal processing and resource allocation algorithms for various classes of system requirements.

During this lab exercises the SDR concept will be studied. Insight into the high flexibility in system design offered by SDR or comparable systems and corresponding architectural constraints will be

This document is highly insipred by and containts work done by Michael Reyer and RWTH Aachen University. The author was allowed to alter and distribute the content provided by them.

gained. Within the framework, high reconfigurability of transmission parameters allows for easy assessment and evaluation of OFDM system performance in real wireless channel conditions and for comparison with theoretically derived results.

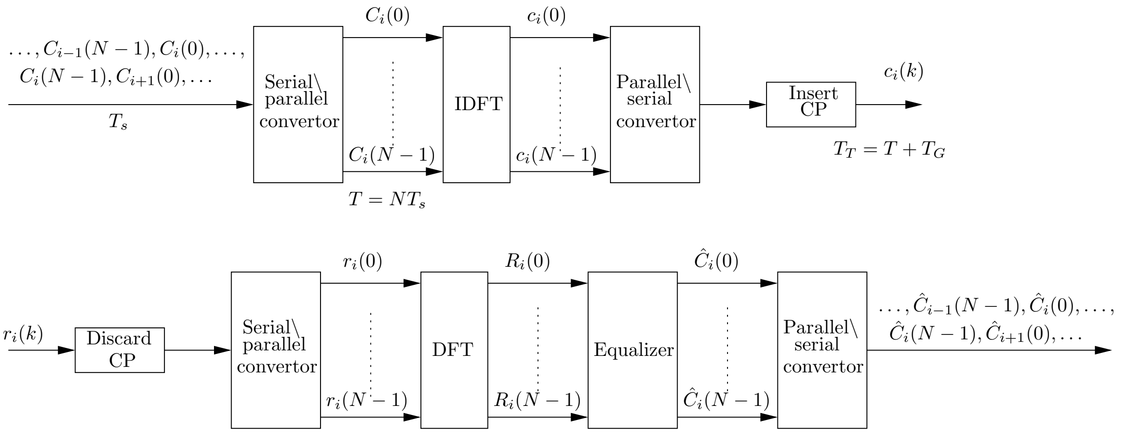
This script is organized as follows. An introduction to basic OFDM system's characteristics is given in Chapter 3. In Section 3.1, a corresponding discrete-model is introduced and applied for analytical assessment of the influence of system impairments which are discussed in Section 3.2. A short survey of coherent modulation techniques commonly used in OFDM systems and their performance evaluation in additive white Gaussian noise (AWGN) channels are presented in Section 3.3. Basic principles, architectural concepts of SDR and an introduction to GNU Radio framework are pictured in Chapter 4. In Section 4.1, system benefits and practical limitations of SDR are addressed. Deeper insight into GNU Radio architecture and an example of wireless channel simulation within a given framework can be gained in Section 4.2. In Chapter **??**, a detailed system description of the SDR framework, which will be used for lab exercises, is given. Finally, the preparatory and lab exercises are described, and corresponding tasks are depicted in Chapters **??** and **??**.

Chapter 3

OFDM Basics

In this chapter the basic principles of OFDM baseband signal processing are given and an appropriate discrete-time OFDM system model is introduced. In the following section impact and prevention of synchronization errors and equalization are explained. Finally, digital modulations commonly used in wireless transmission standards are described in Section 3.3.

OFDM is a multi-carrier modulation scheme that is widely adopted in many recently standardized broadband communication systems due to its ability to cope with frequency selective fading [2]. The block diagram of a typical OFDM system is shown in Fig. 3.1. The main idea behind OFDM is to divide a high-rate encoded data stream (with symbol time T_S) into N parallel substreams (with symbol time $T = NT_S$) that are modulated onto N orthogonal carriers (referred to as subcarriers). This operation is easily implemented in the discrete time domain through an N-point inverse discrete Fourier transform (IDFT) unit and the result is transmitted serially. At the receiver, the information is recovered by performing a DFT on the received block of signal samples. The data transmission



$\ldots, C_{i-1}(N-1), C_i(0), \ldots,$
$C_i(N-1), C_{i+1}(0), \ldots$
T_s
Serial\ parallel convertor
$C_i(0)$
$C_i(N-1)$
$T = NT_s$
IDFT
$c_i(0)$
$c_i(N-1)$
Parallel\ serial convertor
Insert CP
$c_i(k)$
$T_T = T + T_G$
$r_i(k)$
Discard CP
Serial\ parallel convertor
$r_i(0)$
$r_i(N-1)$
DFT
$R_i(0)$
$R_i(N-1)$
Equalizer
$\hat{C}_i(0)$
$\hat{C}_i(N-1)$
Parallel\ serial convertor
$\ldots, \hat{C}_{i-1}(N-1), \hat{C}_i(0), \ldots,$
$\hat{C}_i(N-1), \hat{C}_{i+1}(0), \ldots$

Figure 3.1: Block diagram of a typical OFDM system

in OFDM systems is accomplished in a symbolwise fashion, where each OFDM symbol conveys N (possibly coded) complex data symbols. As a consequence of the time dispersion associated with the frequency-selective channel, contiguous OFDM symbols may partially overlap in time-domain.

This phenomenon results into inter symbol interference ISI, with ensuing limitations of the system performance. The common approach to mitigate ISI is to introduce a guard interval of appropriate length among adjacent symbols. In practice, the guard interval is obtained by duplicating the last N_G samples of each IDFT output and, for this reason, is commonly referred to as cyclic prefix (CP). As illustrated in Fig. 3.2, the CP is appended in front of the corresponding IDFT output. This results into an extended OFDM symbol consisted of $N_T = N + N_G$ samples which can totally remove the ISI as long as N_G is properly designed according to the channel delay spread.

Referring to Fig. 3.1, it can be seen that the received samples are divided into adjacent segments of length N_T, each corresponding to a different transmitted OFDM symbol. Without loss of generality, lets concentrate on the ith OFDM symbol at the receiver. The first operation is the CP removal, which is simply accomplished by discarding the first N_G samples of the considered segment. The remaining N samples are fed to a DFT and the corresponding output is subsequently passed to the channel equalizer. Assuming that synchronization has already been established and the CP is sufficiently long to eliminate the ISI, only a one-tap complex-valued multiplier is required to compensate for the channel distortion over each subcarrier, which will be further described in Section 3.2.3. To better understand this fundamental property of OFDM, however, we need to introduce the mathematical model of the communication scheme depicted in Fig. 3.1.

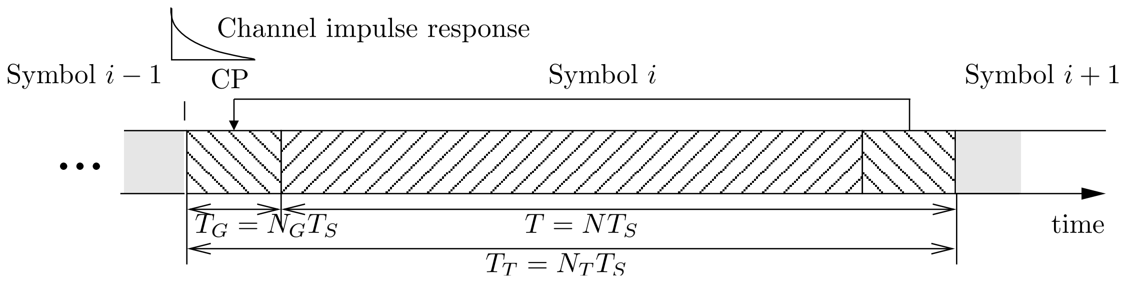


Figure 3.2: Structure of an OFDM symbol

3.1 Discrete-time OFDM System Model

Since OFDM is a block based communication model, a serial data stream is converted into parallel blocks of size N and IDFT is applied to obtain time-domain OFDM symbols. Complex data symbols $C_i(n)$, for $n = 0, \dots, N-1$, within the ith OFDM symbol are taken from either a phase shift keying (PSK) or quadrature amplitude modulation (QAM) constellation. Then, time domain representation of the ith OFDM symbol after IDFT and CP insertion is given by

$$c_i(k) = \begin{cases} \sum_{n=0}^{N-1} C_i(n) e^{j2\pi kn/N}, & -N_G \le k \le N-1 \\ 0, & \text{else} \end{cases}, \tag{3.1}$$

where N_G is the length of the CP which is an important design parameter of the OFDM system that defines the maximum acceptable length of channel impulse response. Furthermore, the transmitted signal can be obtained by concatenating OFDM symbols in time domain as

$$c(k) = \sum_i c_i(k - iN_T). \tag{3.2}$$

In wireless communication systems, transmitted signals are typically reflected, diffracted, and scattered, arriving at the receiver along multiple paths with different delays, amplitudes, and phases as illustrated in Fig 3.3. This leads to an overlapping of different copies of the same signal on the receiver side differing in their amplitude, time of arrival and phase. A common model to describe the wireless channel makes use of the channel impulse response, written as $h(l) = \alpha(l)e^{j\theta(l)}$, for $l = 0, \ldots, L-1$, where L presents the total number of received signal paths, while $\alpha(l)$ and $\theta(l)$ are attenuation and phase shift of the lth path, respectively. The differences in the time of arrival are eliminated by the cyclic prefix, which is described in the next section. In addition to multipath

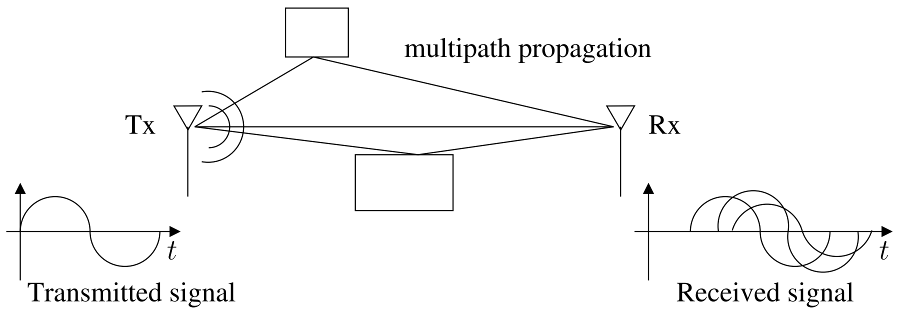


Figure 3.3: The basic principle of multipath propagation

effects, additive noise is introduced to the transmitted signal. The main sources of additive noise are thermal background noise, electrical noise in the receiver amplifiers, and interference [3]. The noise decreases the signal-to-noise ratio (SNR) of the received signal, resulting in a decreased performance. The total effective noise at the receiver of an OFDM system can be modeled as AWGN with a uniform spectral density and zero-mean Gaussian probability distribution. The time domain noise samples are represented by $w(k) \sim SCN(0, \sigma_w^2)$, where σ_w^2 denotes the noise variance and zero the mean of the circular symmetric complex distribution. Therefore, the discrete-time model of received OFDM signals can be written as

$$y(k) = \sum_{l=0}^{L-1} h(l)c(k-l) + w(k). \tag{3.3}$$

Multipath propagation and additive noise affect the signal significantly, corrupting the signal and often placing limitations on the performance of the system.

3.2 OFDM System Impairments

Since timing and frequency errors in multi-carrier systems destroy orthogonality among subcarriers which results in large performance degradations, synchronization of time and frequency plays a major role in the design of a digital communication system. Essentially, this function aims at retrieving some reference parameters from the received signal that are necessary for reliable data detection. In an OFDM system, the following synchronization tasks can be identified [2]:

- *sampling clock synchronization*: in practical systems the sampling clock frequency at the receiver is slightly different from the corresponding frequency at the transmitter. This produces intercarrier interference (ICI) at the output of the receiver's DFT with a corresponding degradation of the system performance. The purpose of a sampling clock synchronization is to limit this impairment to a tolerable level.

- *timing synchronization*: the goal of this operation is to identify the starting point of each received OFDM symbol in order to find the correct position of the DFT window. In burst-mode transmissions timing synchronization is also used to locate the start of the frame (frame synchronization) which is a collection of OFDM symbols.
- *frequency synchronization*: a frequency error between the local oscillators at the transmitter and receiver results in a loss of orthogonality among subcarriers with ensuing limitations of the system performance. Frequency synchronization aims at restoring orthogonality by compensating for any frequency offset caused by oscillator inaccuracies.

The block diagram of the receiver is depicted in Fig. 3.4. In the analog frontend, the incoming waveform $r_{RF}(t)$ is filtered and down-converted to baseband using two quadrature sinusoids generated by a local oscillator (LO). The baseband signal is then passed to the analog-to-digital converter (ADC), where it is sampled with frequency $f_s = 1/T_s$. Due to Doppler shifts and/or oscillator instabilities, the frequency f_{LO} of the LO is not exactly equal to the received carrier frequency f_c. The difference $f_d = f_c - f_{LO}$ is referred to as carrier frequency offset (CFO), or shorter frequency offset, causing a phase shift of $2\pi k f_d$[1]. Therefore, the received baseband signal can be expressed as

$$r(k) = y(k)e^{j2\pi\varepsilon k/N} \tag{3.4}$$

where

$$\varepsilon = N f_d T_s \tag{3.5}$$

is the frequency offset normalized to subcarrier spacing $\Delta f = 1/(NT_s)$. In addition, since the time scales at the transmitter and the receiver are not perfectly aligned, at the start-up the receiver does not know where the OFDM symbols start and, accordingly, the DFT window will be placed in a wrong position. As it will be shown later, since small (fractional) timing errors do not produce any degradation of the system performance, it suffices to estimate the beginning of each received OFDM symbol within one sampling period. Let Δk denotes the number of samples by which the

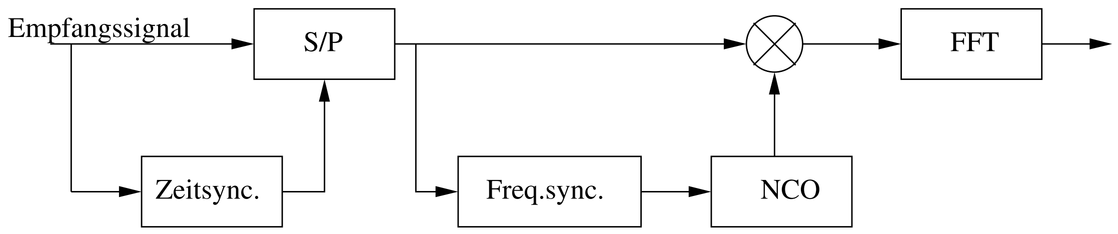


Figure 3.4: Block diagram of a basic OFDM receiver

receive time scale is shifted from its ideal setting. The samples from ADC are thus expressed by

$$r(k) = e^{j2\pi\varepsilon k/N} y(k - \Delta k) + w(k). \tag{3.6}$$

[1]Notice that this phase shift is not time dependent (if f_d is constant) and is different for each subcarrier k.

Replacing (3.2) and (3.3) in (3.6), samples are given as

$$r(k) = e^{j2\pi\varepsilon k/N} \sum_{i} \sum_{l=0}^{L-1} h(l) c_i(k - l - \Delta k - iN_T) + w(k). \tag{3.7}$$

The frequency and timing synchronization units shown in Fig. 3.4 employ the received samples $r(k)$ to compute estimates of ε and Δk, noted as $\hat{\varepsilon}$ and $\hat{\Delta k}$. The former is used to counter-rotate $r(k)$ at an angular speed $2\pi\hat{\varepsilon}k/N$ (frequency correction) using CFO, while the timing estimate is exploited to achieve the correct position of the received signal within the DFT window (timing correction). Specifically, the samples $r(k)$ with indices $iN_T + \Delta k \leq k \leq iN_T + \Delta k + N - 1$ are fed to the DFT device and the corresponding output is used to detect the data symbols conveyed by the ith OFDM block.

3.2.1 Effects of Frequency Offset

In order to assess the impact of a frequency error on the system performance, we assume ideal timing synchronization and let $\Delta k = 0$ and $N_g \geq L - 1$. At the receiver, the DFT output for the ith OFDM symbol is computed as

$$R_i(n) = \frac{1}{N} \sum_{k=0}^{N-1} r(k + iN_T) e^{-j2\pi kn/N}, \; 0 \leq n \leq N - 1 \tag{3.8}$$

Substituting (3.7) into (3.8) we get

$$\begin{aligned}
R_i(n) &= \frac{1}{N} \sum_{k=0}^{N-1} \left[e^{j2\pi\varepsilon(k+iN_T)/N} \sum_{l=0}^{L-1} h(l) c_i(k-l) + w(k) \right] e^{-j2\pi kn/N} \\
&= \frac{1}{N} e^{j\varphi_i} \sum_{k=0}^{N-1} e^{j2\pi k(\varepsilon-n)/N} \sum_{l=0}^{L-1} h(l) \sum_{m=0}^{N-1} C_i(m) e^{j2\pi(k-l)m/N} + W_i(n) \\
&= \frac{1}{N} e^{j\varphi_i} \sum_{m=0}^{N-1} \left\{ \sum_{l=0}^{L-1} h(l) e^{j2\pi lm/N} \right\} C_i(m) \sum_{k=0}^{N-1} e^{-j2\pi k(m+\varepsilon-n)/N} + W_i(n) \\
&= \frac{1}{N} e^{j\varphi_i} \sum_{m=0}^{N-1} H(m) C_i(m) \sum_{k=0}^{N-1} e^{j2\pi k(m+\varepsilon-n)/N} + W_i(n)
\end{aligned} \tag{3.9}$$

where $\varphi_i = 2\pi i\varepsilon N_T/N$, $W_i(n)$ is Gaussian distributed thermal noise with variance σ_w^2 derived as

$$W_i(n) = \frac{1}{N} \sum_{k=0}^{N-1} w(k) e^{-j2\pi kn/N}, \; 0 \leq n \leq N - 1 \tag{3.10}$$

and $H(m)$ is channel frequency response defined as DFT of channel impulse response given as

$$H(m) = \sum_{l=0}^{L-1} h(l) e^{-j2\pi lm/N}, \; 0 \leq m \leq N - 1. \tag{3.11}$$

Performing standard mathematical manipulations (3.9) is derived to

$$R_i(n) = e^{j\varphi_i} \sum_{m=0}^{N-1} H(m)C_i(m)f_N(\varepsilon + m - n)e^{j\pi(N-1)(\varepsilon+m-n)/N} + W_i(n), \tag{3.12}$$

where

$$\begin{aligned} f_N(x) &= \frac{\sin(\pi x)}{N\sin(\pi x/N)} \\ &\approx \frac{\sin(\pi x)}{\pi x} = \mathrm{si}(\pi x) \end{aligned} \tag{3.13}$$

can be derived using the standard approximation $sin(t) \approx t$ for small values of argument t.

In the case when the frequency offsetis a multiple of subcarrier spacing Δf, i.e., ε is integer-valued, (3.12) reduces to

$$R_i(n) = e^{j\varphi_i} H(|n-\varepsilon|_N)C_i(|n-\varepsilon|_N) + W_i(n), \tag{3.14}$$

where $|n-\varepsilon|_N$ is the value of $n-\varepsilon$ reduced to interval $[0, N-1)$. This equation indicates that **an integer frequency offset does not destroy orthogonality among subcarriers and only results into a shift of the subcarrier indices by a quantity** ε. In this case the nth DFT output is an attenuated and phase-rotated version of $C_i(|n-\varepsilon|_N)$ rather than of $C_i(n)$. Otherwise, when ε is not integer-valued the subcarriers are no longer orthogonal and ICI does occur. In this case it is convenient to rewrite (3.12) like

$$R_i(n) = e^{j[\varphi_i + \pi\varepsilon(N-1)/N]} H(n)C_i(n)f_N(\varepsilon) + I_i(n,\varepsilon) + W_i(n), \tag{3.15}$$

where $I_i(n,\varepsilon)$ accounts for ICI and is given as

$$I_i(n,\varepsilon) = e^{j\varphi_i} \sum_{m=0, m\neq n}^{N-1} H(m)C_i(m)f_N(\varepsilon + m - n)e^{j\pi(N-1)(\varepsilon+m-n)/N}. \tag{3.16}$$

From (3.15) it follows that non-integer normalized frequency offsetε influences the received signal on nth subcarrier twofold. Firstly, received signals on all subcarriers are **equally** attenuated by $f_n^2(\varepsilon)$ and phase shifted by $(\varphi_i + \pi\varepsilon(N-1)/N)$, while the second addend in (3.15) presents interference from other subcarriers (ICI).

Letting $E\left\{|H(n)|^2\right\} = 1$ and assuming independent and identically distributed data symbols with zero mean and power $S = E\left\{|C_i(n)|^2\right\} = 1$, the interference term $I_i(n,\varepsilon)$ can reasonably be modeled as a Gaussian zero-mean random variable with variance (power) defined as

$$\sigma_i^2(\varepsilon) = E\left\{|I_i(n)|^2\right\} = S \sum_{\substack{m=0 \\ m\neq n}}^{N-1} f_N^2(\varepsilon + m - n). \tag{3.17}$$

Under assumption that all subcarriers are used and by means of the identity

$$\sum_{m=0}^{N-1} f_N^2(\varepsilon + m - n) = 1, \tag{3.18}$$

which holds true independently of ε, interference power (3.17) can be written as

$$\sigma_i^2(\varepsilon) = S\left[1 - f_N^2(\varepsilon)\right]. \tag{3.19}$$

A useful indicator to evaluate the effect of frequency offset on the system performance is the loss in SNR, which is defined as

$$\gamma(\varepsilon) = \frac{SNR^{(ideal)}}{SNR^{(real)}}, \tag{3.20}$$

where $SNR^{(ideal)}$ is the SNR of a perfectly synchronized system given as

$$SNR^{(ideal)} = S/\sigma_w^2 = E_S/N_0, \tag{3.21}$$

where E_S is the average received energy over each subcarrier while $N_0/2$ is the two-sided power spectral density of the ambient noise, while

$$SNR^{(real)} = Sf_N^2(\varepsilon)/\left[\sigma_w^2 + \sigma_i^2(\varepsilon)\right], \tag{3.22}$$

is the SNR in the presence of a frequency offset ε. Substituting (3.21) and (3.22) into (3.20), it becomes

$$\gamma(\varepsilon) = \frac{1}{f_N^2(\varepsilon)}\left[1 + \frac{E_S}{N_0}(1 - f_N^2(\varepsilon))\right]. \tag{3.23}$$

For small values of ε (3.23) can be simplified using the Taylor series expansion of $f_N^2(\varepsilon)$ around $\varepsilon = 0$, resulting in

$$\gamma(\varepsilon) \approx 1 + \frac{1}{3}\frac{E_S}{N_0}(\pi\varepsilon)^2. \tag{3.24}$$

It can be seen that the SNR loss is approximately proportional to the square of the normalized frequency offsetε.

3.2.2 Effects of Timing Offset

In order to assess the performance of the OFDM system in the presence of small timing offset (TO) let assume perfect frequency synchronization, i.e., $\varepsilon = 0$ and consider only the effect of TO when it is smaller than the uncorrupted part of the cyclic prefix, i.e., $\Delta k \leq N_g - (L+1)$.

Under these assumptions, (3.8) is derived to

$$\begin{aligned} R_i(n) &= \frac{1}{N}\sum_{k=0}^{N-1}\left[\sum_{l=0}^{L-1} h(l)c_i(k-l-\Delta k) + w_i(k)\right] e^{-j2\pi kn/N} \\ &= \frac{1}{N}\sum_{k=0}^{N-1}\sum_{l=0}^{L-1} h(l)\sum_{m=0}^{N-1} C_i(m)e^{j2\pi(k-l-\Delta k)m/N}e^{-j2\pi kn/N} + W_i(n) \\ &= \frac{1}{N}\sum_{m=0}^{N-1}\left[\sum_{l=0}^{L-1} h(l)e^{-j2\pi lm/N}\right] C_i(m)\left[\sum_{k=0}^{N-1} e^{-j2\pi k(m-n)/N}\right] e^{-j2\pi\Delta km/N} + W_i(n) \\ &= \sum_{m=0}^{N-1} H(m)C_i(m)\delta(m-n)e^{-j2\pi\Delta km/N} + W_i(n) \\ &= H(n)C_i(n)e^{-j2\pi\Delta km/N} + W_i(n), \end{aligned} \tag{3.25}$$

where $W_i(n)$ and $H(m)$ are defined in (3.10) and (3.11), respectively, while $\delta(m-n)$ presents the Kronecker delta defined as

$$\delta(m-n) = \begin{cases} 1, & m = n \\ 0, & m \neq n. \end{cases} \tag{3.26}$$

From expression above it can be seen that small TO causes only a linear phase rotation across the DFT outputs and can be compensated by the channel equalizer, which can not distinguish between phase shifts introduced by the channel and those derived from the TO. Timing offset does not destroy the orthogonality of the carriers and the effect of timing error is a phase rotation which linearly changes with subcarrier order.

3.2.3 Equalization

Channel equalization is the process through which a coherent receiver compensates for any distortion induced by frequency-selective fading. For the sake of simplicity, ideal timing and frequency synchronization is considered throughout this subsection. The channel is assumed static over each OFDM block, but can vary from block to block. Under these assumptions, and assuming that the receiver is perfectly synchronized, i.e, $\varepsilon = 0$ and $\Delta k = 0$, the output of the receiver's DFT unit during the ith symbol is given by

$$R_i(n) = H_i(n)C_i(n) + W_i(n), \; 0 \leq n \leq N-1 \tag{3.27}$$

where $C_i(n)$ is the complex data symbol and $W_i(n)$ as well as $H(m)$ are defined in (3.10) and (3.11), respectively. An important feature of OFDM is that channel equalization can independently be performed over each subcarrier by means of a bank of one-tap multipliers. As shown in Fig. 3.5, the nth DFT output $R_i(n)$ is weighted by a complex-valued coefficient $P_i(n)$ in order to compensate for the channel-induced attenuation and phase rotation. The equalized sample $Y_i(n) = P_i(n)R_i(n)$ is then subsequently passed to the detection unit, which delivers final decisions $\hat{C}_i(n)$ on the transmitted data. Intuitively, the simplest method for the design of the equalizer coefficients, is to perform

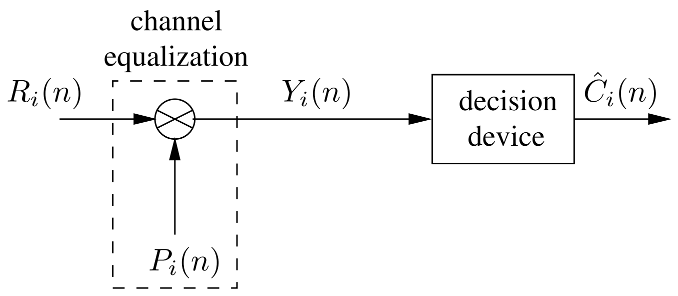


Figure 3.5: Block diagram of an OFDM receiver

a pure channel inversion, know as zero-forcing (ZF) criterion. The equalizer coefficients are then given by

$$P_i(n) = \frac{1}{H_i(n)}, \tag{3.28}$$

while the DFT output takes the form

$$Y_i(n) = \frac{R_i(n)}{H_i(n)} = C_i(n) + \frac{W_i(n)}{H_i(n)}, \; 0 \leq n \leq N-1. \tag{3.29}$$

From (3.29) it can be noticed that ZF equalization is capable of totally compensating for any distortion induced by the wireless channel. However, the noise power at the equalizer output is given by $\sigma_w^2/|H_i(n)|^2$ and may be excessively large over deeply faded subcarriers characterized by low channel gains.

Inherent system requirement for ZF equalizer is the knowledge of the channel transfer function $H_i(n)$. Therefore, in many wireless OFDM systems, sequence of data symbols is preceded by several reference OFDM symbols (preambles) known to the receiver, forming the **OFDM frame**. Typical frame structure is shown in Fig. 3.6 where preambles are typically used for **synchronization** and/or **channel estimation** purposes. In typical fixed wireless standards as WLAN, it can be assumed that

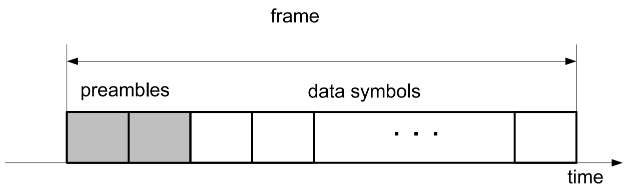


Figure 3.6: Frame structure

the channel remains static over frame duration, i.e., $H_i(n) = H(n)$ for $i = 1, \dots, I$, where I is the total number of OFDM symbols within one frame. Then, channel estimates obtained from the preambles can be used to coherently detect the entire payload.

Assuming that the OFDM frame has one preamble with index $i = p = 1$, the output of the DFT block (3.27) can be written as

$$R_p(n) = H(n)C_p(n) + W_p(n), \; 0 \le n \le N-1 \tag{3.30}$$

where $C_p(n)$ are complex data symbols **known** to the receiver. Then, estimates of the channel frequency response $\hat{H}(n)$ can be obtained as

$$\hat{H}(n) = \frac{R_p(n)}{C_p(n)} = H(n) + \frac{W_p(n)}{C_p(n)}, \; 0 \le n \le N-1. \tag{3.31}$$

On the other hand, in applications characterized by relatively high mobility as those envisioned by the LTE standard, the channel response undergoes significant variations over one frame and must continuously be tracked to maintain reliable data detection. In this case, in addition to initial reference blocks, known symbols called pilots are normally inserted into the payload section of the frame at some convenient positions. These pilots are scattered in both time and frequency directions (i.e., they are positioned over different blocks and different subcarriers), and are used as reference values for channel estimation and tracking.

In order to assess and compare the influence of system impairments on different data rates supported in OFDM systems, a short survey of commonly used coherent modulation techniques and their performance evaluation in AWGN channel are given in the next section.

3.3 Digital Modulations Used in OFDM Systems

Consider some digital information that is given by a finite bit sequence. To transmit this information over a physical, analog channel by a passband signal we need a mapping rule between the set of bit sequences and the set of possible signals or constellation points on the complex plane, as shown in Fig. 3.7. Such a mapping rule is called a digital modulation scheme. A linear digital modulation scheme is characterized by the complex baseband signal [4]

$$C(t) = \sum_i C_i g(t - kT), \tag{3.32}$$

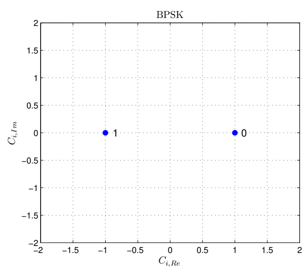
where C_i is a given constellation point and $g(t)$ is a pulse shape used for transmission. Since mapping is usually performed in digital domain we will keep discrete domain representation of modulated complex symbols for further simplification. In the following we will resume some of the coherent modulation schemes typically used in OFDM systems.

3.3.1 Phase Shift Keying (PSK)

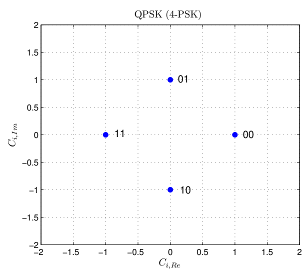
PSK or Multiple PSK (M-PSK) modulation, where M is the number of constellation points, is characterized that all signal information is put into the phase of the transmitted signal, preserving constant envelope property. The M-PSK complex symbol C_i can be written as

$$C_i = \sqrt{S}e^{j(\frac{2\pi m}{M}+\theta_0)}, \; m = 0, 1, \ldots, M-1, \tag{3.33}$$

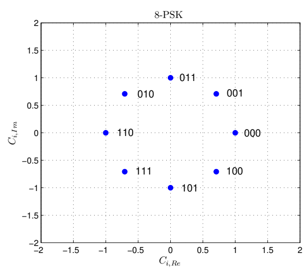
where S is the average signal power and θ_0 is an arbitrary constant phase. Constellation diagrams for $M = 2, 4, 8$, i.e., binary phase-shift keying (BPSK), quadrature phase-shift keying (QPSK) or 4-PSK and 8-PSK, respectively, when $\theta_0 = 0$, are shown in Fig. 3.7. The simplest PSK modulation



(a) BPSK constellation diagram



(b) QPSK(4-PSK) constellation diagram



(c) 8-PSK constellation diagram

Figure 3.7: Gray-coded M-PSK constellation diagrams

format is BPSK, where a logical „1“ is encoded as 0 phase, and a logical „0“ is coded as a phase of π. Then, the modulated symbol, defined in (3.33), can be written as

$$C_i = \pm\sqrt{S} \tag{3.34}$$

with constellation diagram shown in Fig. 3.7a. MPSK constellation diagram for 4-PSK (2 bits mapped into $4 = 2^2$ phases) and 8-PSK (3 bits mapped into $8 = 2^3$ phases), are shown in Fig. 3.7b and Fig. 3.7c, respectively. Note that they are optimized to minimize the bit error rate (BER), resulting in the gray-coded M-PSK constellation, i.e., adjacent constellation points differ in one bit as in Fig. 3.7. The BER is defined as the ratio between the number of successfully received to the number of total transmitted information bits and is usually taken as a measure of modulation quality. For BPSK in AWGN it is given as [5]

$$p_{b,BPSK} = Q\left(\sqrt{\frac{2E_b}{N_0}}\right), \tag{3.35}$$

where $\frac{E_b}{N_0}$ is the SNR per bit and $Q(x)$ is defined as

$$Q(x) = \frac{1}{2}\operatorname{erfc}\left(\frac{x}{\sqrt{2}}\right), \tag{3.36}$$

where

$$\operatorname{erfc}(x) = \frac{2}{\sqrt{\pi}}\int_x^\infty e^{-y^2}dy \tag{3.37}$$

is the complementary error function (erfc). For higher order M-PSK, where $M > 4$, the symbol error rate (SER) can be expressed as

$$p_{s,M-PSK} = 2Q\left(\sqrt{\frac{2E_b\log_2 M}{N_0}}\sin\frac{\pi}{M}\right), \tag{3.38}$$

where

$$\frac{E_s}{N_0} = \frac{E_b\log_2 M}{N_0}$$

is the SNR per symbol. For Gray-coded modulations the BER in the high SNR regime for each modulation is approximately

$$p_{b,M-PSK} \approx \frac{p_{s,M-PSK}}{\log_2 M}.$$

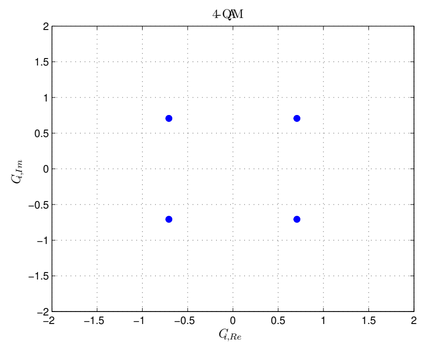
3.3.2 Quadrature Amplitude Modulation (QAM)

QAM is a bandwidth efficient signaling scheme that, unlike M-PSK does not possess a constant envelope property, thus offering higher bandwidth efficiency, i.e., more bits per second (bps) can be transmitted in a given frequency bandwidth. QAM modulated signals for M constellation points can be written as

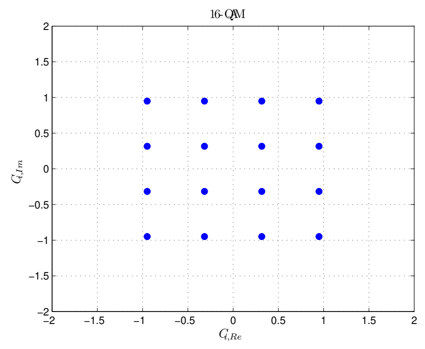
$$C_i = \sqrt{S}K(X_i + jY_i),$$

where $X_i, Y_i \in \left\{\pm1, \pm3, \ldots, \sqrt{M}-1\right\}$ and K is a scaling factor for normalizing the average power for all constellations to S. The K value for various constellations is shown in Table 3.1. Corresponding QAM constellation diagrams for 4-QAM (2 bits mapped into $4 = 2^2$ points), 16-QAM (4 bits mapped into $16 = 2^4$ points), 64-QAM (6 bits mapped into $64 = 2^6$ points), and 256-QAM (8 bits mapped into $16 = 2^8$ points), are shown in Fig. 3.8. It can be noticed that 4-QAM corresponds to QPSK with constant phase shift $\theta_0 = \pi/4$. The symbol-error rate (SER) for QAM modulations can be expressed as

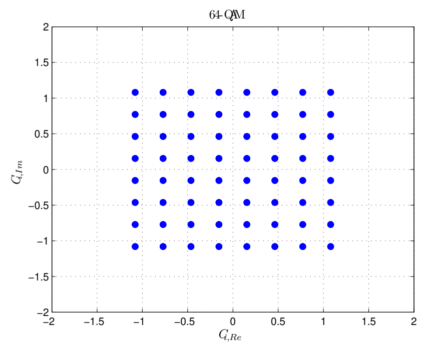
$$p_{s,M-QAM} = 1 - \left(1 - 2\left(1 - \frac{1}{\sqrt{M}}\right)Q\left(\sqrt{3\frac{E_b\log_2 M}{(M-1)N_0}}\right)\right)^2.$$



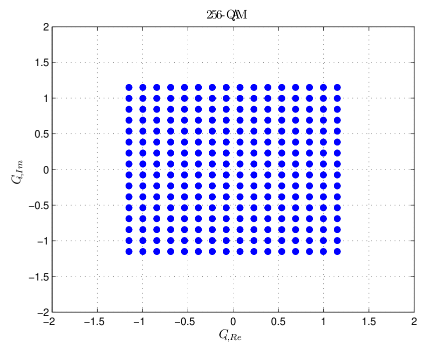
(a) 4-QAM constellation diagram



(b) 16-QAM constellation diagram



(c) 64-QAM constellation diagram



(d) 256-QAM constellation diagram

Figure 3.8: QAM constellation diagrams

where $Q(x)$ is defined in (3.36). For Gray-coded modulations the BER in the high SNR regime for each modulation is, as for M-PSK case, approximately

$$p_{b,M-QAM} \approx \frac{p_{s,M-QAM}}{\log_2 M}. \tag{3.39}$$

QAM schemes like 4-QAM (QPSK), 16-QAM and 64-QAM are used in typical wireless digital communications specifications like WLAN and WiMAX [6].

Modulation	Number of bits m	K
4-QAM	2	$1/\sqrt{2}$
16-QAM	4	$1/\sqrt{10}$
64-QAM	6	$1/\sqrt{42}$
256-QAM	8	$1/\sqrt{170}$

Table 3.1: Modulation dependent parameters

Chapter 4

Software Defined Radio and GNU Radio Framework

In this chapter a general introduction to the SDR concept is given. Additionally, advantages of SDRs and given hardware limitations are addressed. Here, the GNU Radio SDR framework is presented giving insight into the basic architectural features.

An SDR is a radio that is built entirely or in large parts in software, which runs on a general purpose computer. A more extensive definition is given by Joseph Mitola, who established the term Software Radio [7]:

"A software radio is a radio whose channel modulation waveforms are defined in software. That is, waveforms are generated as sampled digital signals, converted from digital to analog via a wideband digital-to-analog converter (DAC) and then possibly upconverted from intermediate-frequency (IF) to RF. The receiver, similarly, employs a wideband ADC that captures all of the channels of the software radio node. The receiver then extracts, downconverts and demodulates the channel waveform using software on a general purpose processor. Software radios employ a combination of techniques that include multi-band antennas and RF conversion; wideband ADC and DAC; and the implementation of IF, baseband and bitstream processing functions in general purpose programmable processors. The resulting software defined radio (or software radio) in part extends the evolution of programmable hardware, increasing flexibility via increased programmability."

This means, that instead of using analog circuits or a specialized digital signal processing (DSP) to process radio signals, the digitized signals are processed by architecture independent, and high level software running on general purpose processors. The term radio designates any device, that transmits and/or receives radio waves. While most modern radios contain firmware that is written in some kind of programming language, the important distinction in a software radio is that it is not tailored to a specific chip or platform, and it is therefore possible to reuse its code across different underlying architectures [8].

4.1 Ideal Software Defined Radio and Practical Limitations

In the ideal case, the only hardware that is needed besides a computer is an antenna and an ADC for the receiver, as well as a DAC for the transmitter. An SDR would thus look as depicted in Fig. 4.1. In the receiver, a transmitted radio signal is picked up by an antenna, and then fed into an ADC to sample it. Once digitized, the signal is sent to some general purpose computer (e.g. an embedded PC) for processing. The transmitter looks very similar, except that the signal is sent in the reverse direction, and a DAC is used instead of an ADC. In a complete transceiver, the processing unit and the antenna may be shared between receiver and transmitter.

While the approach presented in the previous section is very simple and (in the ideal case) extremely versatile, it is not practical, due to limitations in real hardware. However, various solutions have been suggested to overcome these problems. A quick look at the different hardware limitations is given below. For better readability, only the receiving side is discussed. The transmitting side is constructed symmetrically.

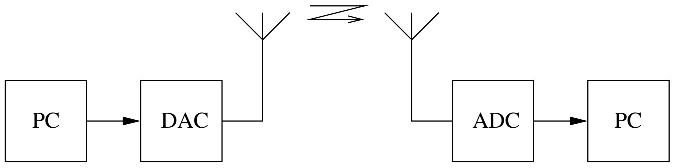


Figure 4.1: Ideal SDR transmission

- *Analog-to-digital converter*: According to the Nyquist sampling theorem, the sampling rate of the ADCs must be at least twice as high as the bandwidth of received signal which limits the maximum bandwidth of the received signal. Conventional ADCs are capable of sampling rates in the area of 500 Msps, which translates to a bandwidth of 250 MHz. While this bandwidth is enough for most current applications, the carrier frequency is usually higher than 250 MHz. In practice, a RF frontend is therefore usually required, to convert the received signal to an IF. Notably, as of 2018 there exists 12-bit ADCs which can sample at 6.4 Gs/s. This allows to directly digitize[1] signals at RF frequencies and achieve sufficient dynamic range for modern communications. However, this technology is costly and not energy-efficient.

  The second parameter, the ADC resolution influences the dynamic range of the receiver. As each additional bit doubles the resolution of the sampled input voltage, the dynamic range can be roughly estimated as $R = 6dB \times n$ where R is the dynamic range and n the number of bits in the ADC. As ADCs used for SDR usually have a resolution of less than 16 bits, it is important to filter out strong interfering signals, such as signals from mobile phones, before the wideband ADCs. This is usually done in the RF frontend.

- *Bus Speed*: Another problem lies in getting the data from the ADC to the computer. For any practical bus, there is a maximum for the possible data rate, limiting the product of sample rate and resolution of the samples. The speed of common buses in commodity PCs ranges from a few Mbps to several Gbps as an example, the Peripheral Component Interconnect Express (PCIe) Gen 5 bus has a theoretical maximum speed of 64 GB/s (in one direction).

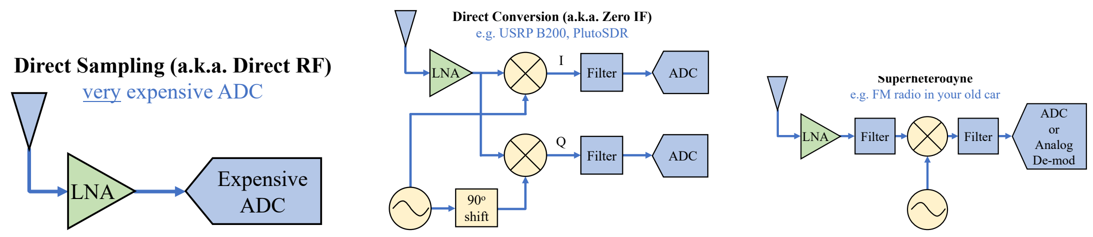
[1]Therefore it is often called direct-RF sampling.

However, the speed is limited by the connections on your PC. For USB-C has a theoretical transfer speed of 20 Gbit/s, which would not support transfering direct-RF samples.

- *Performance of the Processing Unit*: For real-time processing, the performance of the central-processing unit (CPU) and the sample rate limit the number of mathematical operations that can be performed per sample, as samples must be processed as fast as they arrive. In practice, this means that fast CPUs, clever programming and possibly parallelization is needed. If this does not suffice, a compromise must be found, to use a less optimal but faster signal processing algorithm.

- *Latency*: Since general purpose computers are not designed for real-time applications, a rather high latency can occur in practical SDRs. While latency is not much of an issue in transmit-only or receive-only applications, many wireless standards, such as LTE require precise timing, and are therefore very difficult to implement in an SDR.

Hence, we need to reduce the number of samples. This is done by first down-converting the analog RF signal to an intermediate-frequency (superheterofyne receiver) or directly to baseband and perform quadrature sampling (direct conversion or zero-IF).



(a) Direct RF (b) Zero IF (c) Superheterodyne

Figure 4.2: RF Architectures (from pysdr.org)

Because of the use of general purpose processing units, an implementation of a given wireless application as an SDR is likely to use more power and occupy more space than a hardware radio with analog filtering and possibly a dedicated signal processor. Because an SDR contains more complex components than a hardware radio, it will likely be more expensive, given a large enough production volume.

Nevertheless, SDR concepts carry the flexibility of software over to the radio world and introduces a number of interesting possibilities. For example, very much the same way as someone may load an alternative word processor or Internet browser on a PC, depending on the task at hand, a SDR could allow its user to load a different configuration, depending on whether the user wants to listen to a broadcast radio transmission, place a phone call or determine the position via Global Positioning System (GPS).Since the same hardware can be used for any application, a great reuse of resources is possible. Another interesting possibility enabled by SDR is the creation of a cognitive radio, which is aware of its RF environment and adapts itself to changes in the environment. By doing this, a cognitive radio can use both the RF spectrum and its own energy resources more efficiently. As a cognitive radio requires a very high degree of flexibility, the concept of SDR is very convenient for its practical realization.

4.2 GNU Radio Architecture

GNU Radio is an open source, free software toolkit for building SDRs [1]. It is designed to run on commodity computers combined with minimal hardware, allowing the construction of simple software radios [8]. The project was started in early 2000 by Eric Blossom and has evolved into a mature software infrastructure that is used by a large community of developers. It is licensed under the GNU General Public License (GPL), thus anyone is allowed to use, copy and modify GNU Radio without limits, provided that extensions are made available under the same license. While GNU Radio was initially started on a Linux platform, it now supports various Windows, MAC and various Unix platforms.

GNU Radio architecture consists of two components. The first component is the set of numerous building blocks which represents C++ implementations of digital signal processing routines such as (de)modulation, filtering, (de)coding and I/O operations such as file access, for further information about C++ programming see for example [9]–[13]. The second component is a framework to control the data flow among blocks, implemented as Python scripts enabling easy reconfiguration and control of various system functionalities and parameters, for further studies on Python see e.g., [14]. By *wiring* together such building blocks, a user can create a software defined radio, similar to connecting physical RF building blocks to create a hardware radio. An RF interface for GNU Radio architecture is realized by USRP boards, a general purpose RF hardware, which performs computationally intensive operations as filtering, up- and down-conversion. The USRP B210s connected via USB, which we will be using in this lab, are controlled through a robust application program interface (API) provided by GNU Radio.[2]

4.2.1 Gnu Radio Framework

A data flow among different blocks is abstracted by **flowgraph**, a directed acyclic graph in which the vertices are the GNU Radio blocks and the edges corresponds to data **streams**, as shown in Fig. 4.3. Generally, GNU Radio blocks, shown in Fig. 4.4 operate on continuous streams of data.

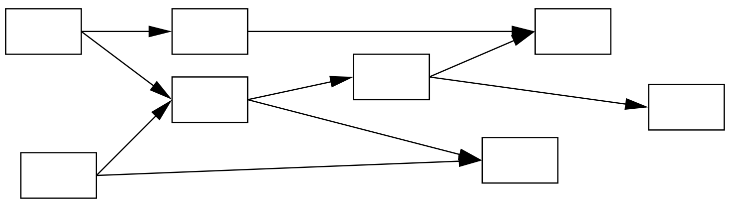


Figure 4.3: An example of **flowgraph**

Most blocks have a set of input and/or output ports, therefore, they consume data from input streams and generate data for their output streams. Special blocks called sources and sinks only consume or produce data, respectively. Examples of sources and sinks are blocks that read and write, respectively, from USRP receive ports, sockets, and file descriptors. Each block has an input and output signature (IO signatures) that defines the minimum and maximum number of input

[2]Note that the USRPs can also be programmed with the UHD library from Ettus. This provides a more low-level access to the device, and is therefore out of scope of this lab.

and output streams it can have, as well as the size of the data type on the corresponding stream. Examples of supported types are

- `c` - complex interleaved floats (8 byte each),
- `f` - floats (4 byte),
- `s` - short integers (2 byte) and
- `b` - byte integers (1 byte).

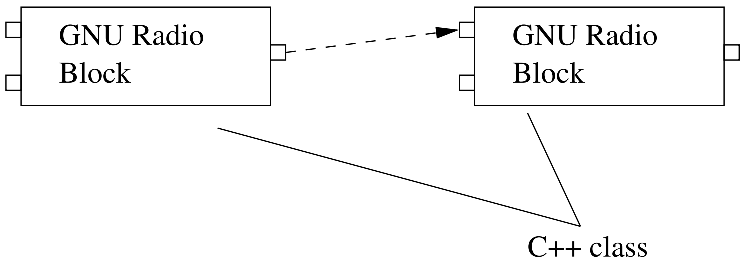


Figure 4.4: GNU Radio blocks

A full list of the supported data types can be found in GNU Radio Companion (GRC) by clicking on `Help > Types`.

Each block defines a `general_work()` function that operates on its input to produce output streams. In order to help the scheduler decide when to call the work function, blocks also provide a `forecast()` function that returns the system runtime, the number of input items it requires to produce a number of output items and how many output items it can produce given a number of input items. At runtime, blocks tell the system how many input (output) items they consumed (produced). Blocks may consume data on each input stream at a different rate, but all output streams must produce data at the same rate. The input and output streams of a block have buffers associated with them. Each input stream has a read buffer, from which the block reads data for processing. Similarly, after processing, blocks write data to the appropriate write buffers of its output streams. The data buffers are used to implement the edges in the flowgraph: the input buffers for a block are the output buffers of the upstream block in the flowgraph. GNU Radio buffers are single writer, multiple reader First In, First Out (FIFO) buffers. Several blocks are connected in Python

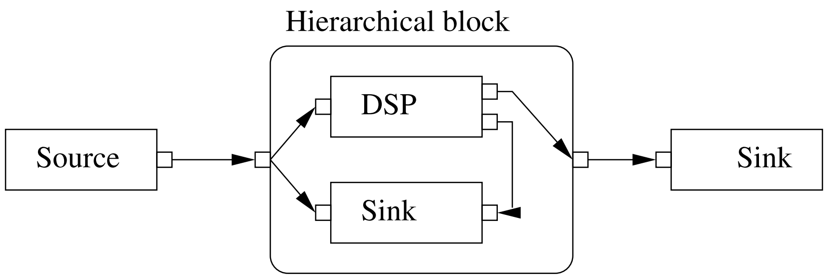


Figure 4.5: An example of a flowgraph with a hierarchical block

forming a flowgraph using the **connect** function, which specifies how the output stream(s) of a processing block connects to the input stream of one or more downstream blocks. The flowgraph mechanism then automatically builds the flowgraph; the details of this process are hidden from the user. A key function during flowgraph construction is the allocation of data buffers to connect neighboring blocks. The buffer allocation algorithm considers the input and output block sizes used by blocks and the relative rate at which blocks consume and produce items on their input and output streams. Once buffers have been allocated, they are connected with the input and output streams of the appropriate block.

Several blocks can also be combined in a new block, named **hierarchical** block, as shown in Fig. 4.5. **Hierarchical** blocks are implemented in Python and together with other blocks can be combined into new **hierarchical** blocks. Input and output ports of hierarchical blocks have the same constraints as those of terminal blocks.

The GNU Radio scheduler executes the graph that was built by the flowgraph mechanism. During the execution, the scheduler queries each block for its input requirements, and it uses the above-mentioned forecast functions to determine how much data the block can consume from its available input. If sufficient data is available in the input buffers, the scheduler calls the block's work function. If a block does not have sufficient input, the scheduler simply moves on to the next block in the graph. Skipped blocks will be executed later, when more input data is available. The scheduler is designed to operate on continuous data streams.

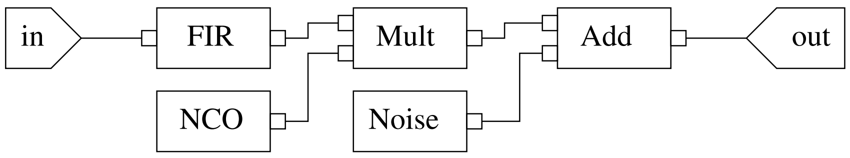


Figure 4.6: Wireless communication channel simulation model

4.2.2 An Example: Wireless Channel Simulation

It will be shown how a model for a static wireless channel can be implemented as a GNU Radio **hierarchical** block. The channel is affected by multipath propagation, frequency offset and additive noise. Fig. 4.6 shows a model with internal blocks and corresponding ports [15].

Multipath effects are modeled using a FIR-filter where complex filter coefficients are taken from an arbitrary channel model, e.g., Rayleigh channel model. The signal from an input port is derived to the corresponding GNU Radio block **gr.fir_filter_ccc**. The suffix ccc denotes that the input stream, output stream and filter coefficients are of complex data types.

According to (3.4), the frequency offset is modeled as a sinus wave with fixed frequency and is multiplied with the incoming signal. The corresponding GNU radio blocks are the complex sine signal source **gr.sig_source_c** and the multiplicator with complex inputs and outputs **gr.multiply_cc**, respectively.

Finally, complex additive Gaussian noise generated by **gr.noise_source_c** is added to the incoming signal in the **gr.add_cc** block and the result is directed to the output port.

The initial parameters of a given hierarchical block, named **simple_channel**, are additive noise standard variance, frequency offset normalized to sampling frequency and complex FIR-filter coefficients. IO signatures of input and output ports are identical, and in the framework there is minimum one port and maximum one port for both input and output.

During runtime, internal blocks are initialized and connected to the flowgraph. The corresponding python script is shown below.

Program 1 Python script for simulation of a wireless communication channel

```
class simple_channel(gr.hier_block2):
  def __init__(self, noise_rms, frequency_offset, channel_coefficients):
    gr.hier_block2.__init__(self, "simple_channel", # Blocktype Identifier
        gr.io_signature(1,1,gr.sizeof_gr_complex),  # incoming
        gr.io_signature(1,1,gr.sizeof_gr_complex))  # outgoing

    # for example channel_coefficients = [0.5+0.1j, 0.2-0.01j]
    multipath_sim = gr.fir_filter_ccc(1, channel_coefficients)

    # frequency_offset normalized to sampling frequency
    # amplitude = 1.0, DC offset = 0.0
    offset_src = gr.sig_source_c(1, gr.GR_SIN_WAVE, frequency_offset, 1.0, 0.0)
    mix = gr.multiply_cc()

    # noise_rms -> var(noise) = noise_rms**2
    noise_src = gr.noise_source_c(gr.GR_GAUSSIAN, noise_rms/sqrt(2))
    add_noise = gr.add_cc()

    # describe signal paths
    self.connect(self, multipath_sim)       # incoming port
    self.connect(multipath_sim, (mix,0))
    self.connect(offset_src, (mix,1))
    self.connect(mix, (noise_add,0))
    self.connect(noise_src, (noise_add,1))
    self.connect(noise_add, self)           # outgoing port
```

4.3 Installing GNU Radio

GNU Radio can be run on multiple operating systems (OSs), although the most reliable platform is a Linux-based environment, e.g., Ubuntu or Mint. For Windows and macOS Radioconda can be installed, which bundles a collection of cross-platform installers for numerous open-source software radio packages.

More instructions can be found at https://wiki.gnuradio.org/index.php?title=InstallingGR. **Do not** follow the instructions under *Other Installation Methods*. If not clear, contact your lab assistant.

Chapter 5

Assignments

5.1 Tune your own antenna

In this section, you'll be customizing your own antenna. Begin by attaching the SubMiniature version A (SMA) connector to the provided patch antenna under the guidance of the lab instructor (see first procedure). Once this is completed, carefully trim small sections from the corners to fine-tune the antenna for operation at 917 MHz.[1]

Procedure connector:

1. Remove the excess coating on the feed pin of the SMA connector with a cutter knife. **Before beginning, seek additional details from your lab instructor on the cutting process. Additionally, exercise caution as sharp tools are involved—cut away from the body (and fingers).**
2. Enlarge the hole of the feed point of the antenna by drilling (0.5 mm).
3. Solder the connector on the antenna, two ground planes of the connector and the feed point at the front of the antenna. See Fig. 5.1.

Procedure antenna:

1. Calibrate the NanoVNA (`nanovna.com`) to the desired frequency band.
2. Solder the SMA connector to the patch antenna.
3. Connect your antenna to the NanoVNA, using `CH0`.
4. Set the vector network analyzer (VNA) to reflect (S11) mode: `DISPLAY > CHANNEL > CH0 REFLECT`.

[1]Why don't we buy a ready-made antenna? The answer is sustainability and learning by experience. The antennas used in this lab session were manufactured with the wrong substrate altering the expected specifications of the antenna, i.e., it is de-tuned from it's designed frequency. Rather than throwing them in the bin, we reuse them in this lab. Similarly, the SMA connectors had to be altered before use.

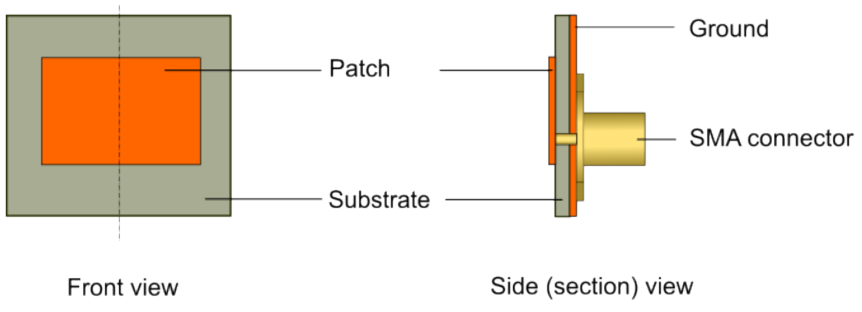


Figure 5.1: Illustration of a microstrip patch antenna on a finite ground using a pin feed (from Altair Community)

5. Refer to the S-parameter details at `www.antenna-theory.com/definitions/sparameters.php`. If clarification is needed, consult your lab assistant.
6. The antenna is tuned when the S11 curve reaches its minimum at 917 MHz. Aim for an antenna bandwidth of at least 1 MHz (S11 below −10 dB around the target frequency).
7. Plot the S11 curve for your antenna, identify the frequency at which it radiates most effectively, and determine the antenna bandwidth.
8. Extract the S11 curve via the Python API `https://github.com/ttrftech/NanoVNA/tree/master/python`
9. Carefully trim small sections from the copper corners of the patch antenna to decrease the patch size and adjust the radiating frequency of the antenna. **Before beginning, seek additional details from your lab instructor on the cutting process. Additionally, exercise caution as sharp tools are involved—cut away from the body (and fingers) and ensure the antenna is placed flat on the table.**
10. Iterate the process of reading S11 and making cuts until the desired target frequency is achieved (steps 7-9).
11. Show your results to the lab assistant before continuing to the next assignment.

5.2 Construct your own USRP housing

To not damage the delicate electronics, the B210 should be sealed via a casing. See Fig. 5.2 for an exploded view of the casing developed by Dramco for the USRP B210.

Sequence of instructions:

1. Place the bolts in the holders (11, 21, 22 and 12)
2. Place the holders on the sides of the casings, in the outer sides of the side (1,2, 3 and 4)
3. Assemble the top and the sides (1 and 3), not the front (4) or back (2)

4. Place the B210 inside the casing, aligned with holes of the holders
5. Complete the assembly by mounting the front and back, encapsulating the B210.

5.3 Getting started with GNU Radio

Read Chapter 4 thoroughly before installing GRC.

Follow the *Beginner Tutorials* on `https://wiki.gnuradio.org/index.php?title=Tutorials`. It is imperative for the remainder of the lab sessions that you will carefully and thoroughly go through each of those tutorials. Additionally, follow the introduction tutorial on complex signals if you are not yet familier with concepts such as complex base band and pass band (`https://wiki.gnuradio.org/index.php?title=IQ_Complex_Tutorial`).

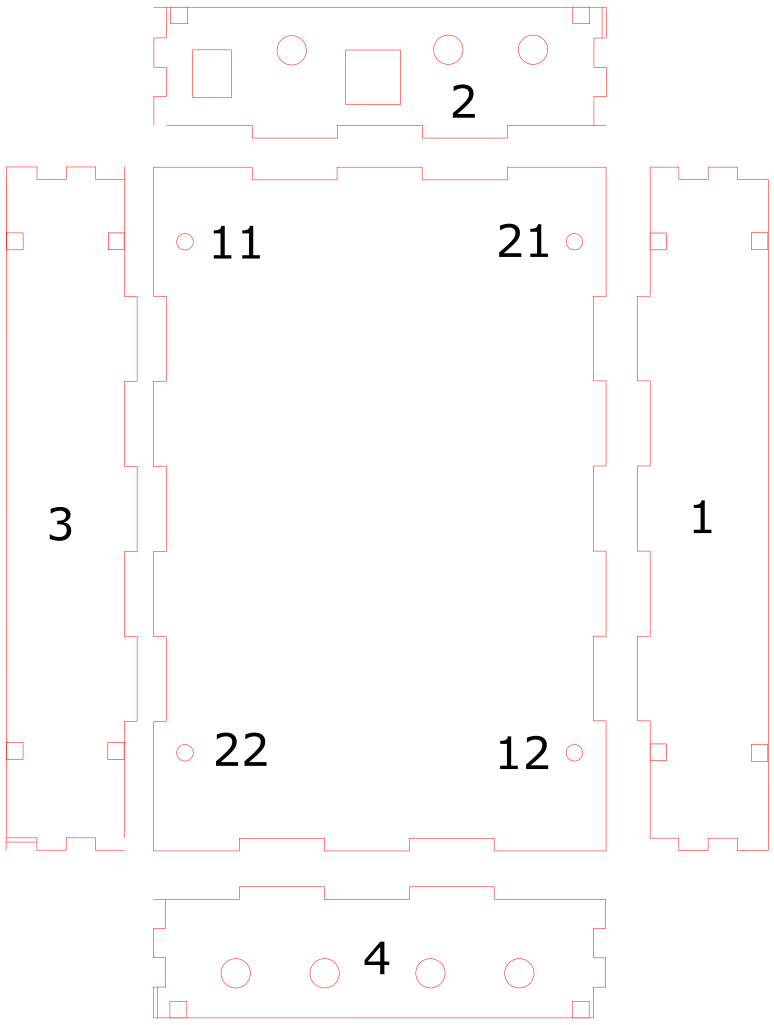


Figure 5.2: Exploded view of the B210 casing. The numbers indicate which component should be placed where.

Chapter 6

Evaluation

Evaluation of this lab session will consider students' work attitude throughout the session and a final test to assess their acquisition of requisite skills and knowledge.

Students have the option to collaborate in groups of up to two. However, it's essential to note that assignments must be completed individually. This means each student should be capable of repeating the lab session independently, as their performance will be evaluated during the final test. Additionally, when the students are referred to external sources during the assignment, this must be also known for the final test. Further instructions will be given during the lab sessions.

Bibliography

[1] *"GNU Radio"*, http://gnuradio.org/trac. [Online]. Available: http://gnuradio.org/trac.

[2] M.-o. Pun, M. Morelli, and C. C. J. Kuo, *Multi-Carrier Techniques For Broadband Wireless Communications: A Signal Processing Perspectives*. London, England, UK: Imperial College Press, 2007, isbn: 1860949460, 9781860949463.

[3] Y. Li and G. L. Stuber, *Orthogonal Frequency Division Multiplexing for Wireless Communications (Signals and Communication Technology)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006, isbn: 0387290958.

[4] T. J. Rouphael, *RF and Digital Signal Processing for Software-Defined Radio: A Multi-Standard Multi-Mode Approach*. Newton, MA, USA: Newnes, 2008, isbn: 0750682108, 9780750682107.

[5] A. Goldsmith, *Wireless Communications*. New York, NY, USA: Cambridge University Press, 2005, isbn: 0521837162.

[6] M. Ergen, *Mobile Broadband - Including WiMAX and LTE*. Springer, 2009, isbn: 0387681892, 9780387681894.

[7] J. I. Mitola, *Cognitive Radio Architecture: The Engineering Foundations of Radio XML*. New York, NY USA: Wiley-Interscience, 2006, isbn: 0471742449.

[8] A. Mueller, *DAB, Software Receiver Implementation*. ETH Zurich: Semester Thesis, 2008.

[9] *Effektiv C++ programmieren*.

[10] *Mehr Effektiv C++ programmieren*.

[11] S. Meyers, *Effective STL*. Addison Wesley, 2001.

[12] *Die C++ Programmiersprache*.

[13] *C++ Coding Standards*.

[14] A. Martelli, *Python in a Nutshell. A Desktop Quick Reference*, 1st ed. institute: O'Reilly & Associates, 2003.

[15] D. Auras, *GNU Radio Implementierungen zur Synchronisierung von OFDM-Systemen*, German. RWTH Aachen: Studienarbeit, 2009.