

# OFDM with SDR Wireless Communication

(B-KUL-JLI1CG)

Gilles Callebaut

March 18, 2024

# Contents

<b>1</b>	<b>Abbreviations</b>	<b>3</b>
<b>2</b>	<b>Introduction</b>	<b>7</b>
<b>3</b>	<b>OFDM Basics and Impairments</b>	<b>9</b>
3.1	The basics (in laymen terms) . . . . .	9
3.1.1	Key features . . . . .	10
3.1.2	Benefits of OFDM . . . . .	10
3.2	The OFDM System . . . . .	10
3.2.1	The Wireless Channel – A short summary . . . . .	11
3.3	Discrete-time OFDM System Model . . . . .	12
3.4	OFDM System Impairments . . . . .	13
3.4.1	Effects of Frequency Offset . . . . .	15
3.4.2	Effects of Timing Offset . . . . .	17
3.4.3	Equalization . . . . .	17
3.5	Digital Modulations Used in OFDM Systems . . . . .	19
3.5.1	Phase Shift Keying (PSK) . . . . .	20
3.5.2	Quadrature Amplitude Modulation (QAM) . . . . .	21
<b>4</b>	<b>Software Defined Radio and GNU Radio Framework</b>	<b>23</b>
4.1	Ideal Software Defined Radio and Practical Limitations . . . . .	24
4.2	GNU Radio Architecture . . . . .	26
4.2.1	Gnu Radio Framework . . . . .	26
4.3	Installing GNU Radio . . . . .	28

<b>5</b>	<b>OFDM in GNU Radio</b>	<b>29</b>
5.1	Conventions and Notations . . . . .	29
5.1.1	FFT Shifting . . . . .	29
5.1.2	Carrier Indexing . . . . .	29
5.1.3	Carrier and Symbol Allocation . . . . .	29
5.2	Transmitting . . . . .	30
5.2.1	OFDM Transmitter . . . . .	31
5.3	Receiver . . . . .	32
5.3.1	Detection and Synchronization . . . . .	32
<b>6</b>	<b>Assignments</b>	<b>35</b>
6.1	Tune your own antenna . . . . .	35
6.2	Construct your own universal software radio peripheral (USRP) housing . . . . .	37
6.3	Getting started with GNU Radio . . . . .	37
6.4	Getting to know OFDM . . . . .	39
<b>7</b>	<b>Evaluation</b>	<b>41</b>

# Chapter 1

## Abbreviations

**ADC** analog-to-digital converter. 14, 23, 24

**API** application program interface. 26

**AWGN** additive white Gaussian noise. 8, 13, 19, 39

**BB** base-band. 19, 20, 37, 41

**BER** bit error rate. 21

**BPSK** binary phase-shift keying. 20, 32

**BW** bandwidth. 31

**CFO** carrier frequency offset. 14

**CP** cyclic prefix. 12, 13, 31, 33

**CPU** central-processing unit. 25

**DAB** Digital Audio Broadcasting. 7

**DAC** digital-to-analog converter. 23, 24

**DC** direct current. 29

**DFT** discrete Fourier transform. 7, 11–15, 17, 18

**DSL** digital subscriber line. 7

**DSP** digital signal processing. 23

**DVB** Digital Video Broadcasting. 7

**FDE** frequency domain equalizer. 12

**FDM** frequency-division multiplexing. 10

- FFT** fast Fourier transform. 7, 29
- FIFO** First In, First Out. 27
- GPL** GNU General Public License. 26
- GPS** Global Positioning System. 25
- GRC** GNU Radio Companion. 27, 37, 39, 41
- ICI** intercarrier interference. 13, 16
- IDFT** inverse discrete Fourier transform. 11, 12
- IF** intermediate-frequency. 23–25
- IQ** in-phase and quadrature. 41
- ISI** intersymbol interference. 7, 11, 12, 41
- LO** local oscillator. 14
- LTE** Long Term Evolution. 7, 19, 25
- OBW** occupied bandwidth. 31
- OFDM** orthogonal frequency-division multiplexing. 2, 7–13, 17–19, 29, 39, 41, 42
- OFDMA** orthogonal frequency-division multiple access. 10
- OS** operating system. 28
- PCIe** Peripheral Component Interconnect Express. 24
- PSK** phase shift keying. 12, 20, 21
- QAM** quadrature amplitude modulation. 10, 12, 21, 22, 41
- QPSK** quadrature phase-shift keying. 20, 22, 32
- RF** radio frequency. 7, 19, 23–26
- SDR** software-defined radio. 7, 8, 23–26
- SMA** SubMiniature version A. 35, 36
- SNR** signal-to-noise ratio. 13, 16, 17, 39
- TO** timing offset. 17
- USRP** universal software radio peripheral. 2, 7, 26, 37, 41

**VNA** vector network analyzer. [36](#), [41](#)

**WiMAX** Worldwide Interoperability for Microwave Access. [7](#)

**WLAN** wireless LAN. [7](#)

**ZF** zero-forcing. [18](#)



# 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 inspired by and contains 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 3.3, a corresponding discrete-model is introduced and applied for analytical assessment of the influence of system impairments which are discussed in Section 3.4. 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 3.5. 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 and Impairments

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 3.5.

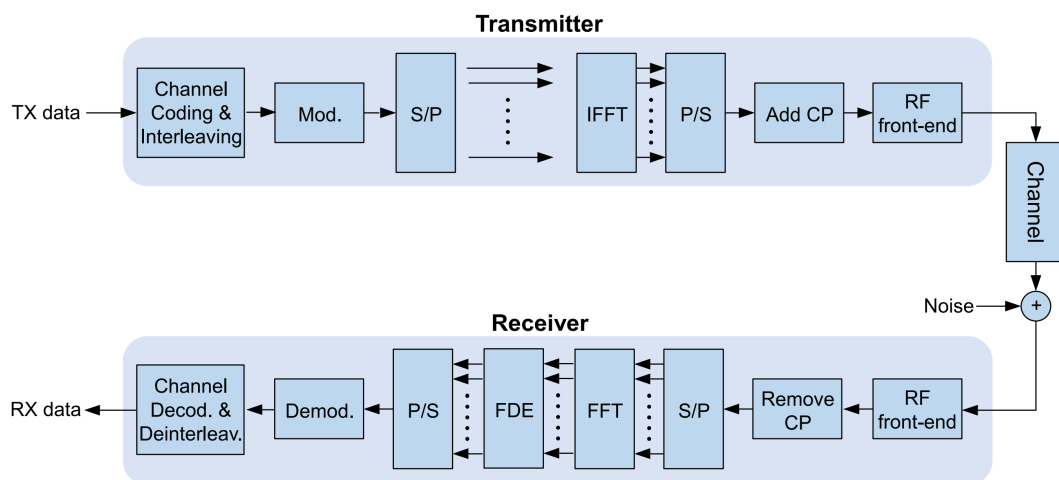


Figure 3.1: Block diagram of a typical **OFDM** system (image from [2])

### 3.1 The basics (in laymen terms)

Imagine you have a fast internet connection, and you want to send a large file to a friend. Instead of sending the entire file in one go, you decide to break it into smaller chunks and send them separately. This makes it easier to manage and lessens the chance of losing all the data if something goes wrong during transmission. This is the main operating principle of **OFDM**.

### 3.1.1 Key features

#### Frequency Division Multiplexing

OFDM combines the principles of multiplexing and modulation. Multiplexing is the technique of combining multiple signals into one signal for transmission, while modulation involves modifying the properties of a carrier signal to encode information. In OFDM, each subchannel is modulated independently using a technique like quadrature amplitude modulation (QAM) (see Section 3.5), and then all the modulated subchannels are combined for transmission.

#### Orthogonality

In frequency-division multiplexing (FDM), the subchannels are typically close to each other in frequency, which can lead to interference. OFDM employs a clever technique called orthogonalization<sup>1</sup>, where the subchannels are spaced apart at precise intervals, ensuring that they don't interfere with each other. It's akin to having lanes on a highway spaced far enough apart that cars in one lane don't disrupt those in adjacent lanes.

#### Multiplexing

After orthogonalization, multiple streams of data can be transmitted simultaneously over the different subchannels. Each subchannel acts as a separate communication path, allowing for efficient use of the available spectrum. If the subchannels are used by multiple users, this technique is called orthogonal frequency-division multiple access (OFDMA).

### 3.1.2 Benefits of OFDM

#### Spectral Efficiency

By dividing the available bandwidth into smaller subchannels, OFDM allows for efficient use of the spectrum. This enables higher data rates compared to traditional single-carrier modulation techniques.

#### Robustness

OFDM is inherently robust against frequency-selective fading [3], which occurs when different frequencies in the signal experience different levels of attenuation or delay. Since OFDM divides the signal into multiple subchannels, even if some subchannels are affected by fading, others may still carry useful data.

#### Flexibility

OFDM can adapt to changing channel conditions by adjusting parameters such as subchannel spacing and modulation scheme. This flexibility makes it suitable for a wide range of communication systems.

## 3.2 The OFDM System

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

---

<sup>1</sup>In mathematics, two functions are orthogonal if their inner product (the integral of their product over a given interval) is zero.

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 in **OFDM** systems is accomplished in a symbolwise fashion, where each OFDM symbol conveys  $N$  (possibly coded) complex data symbols.

### 3.2.1 The Wireless Channel – A short summary

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. 3.2. 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

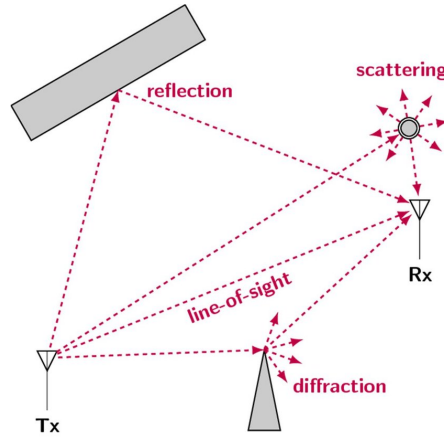


Figure 3.2: The basic principle of multipath propagation (image form [4])

the wireless channel makes use of the *channel impulse response* (time domain), written as,

$$h(l) = \alpha(l)e^{j\theta(l)}, \quad (3.1)$$

for  $l = 0, \dots, L - 1$ , where  $L$  presents the total number of received signal paths, while  $\alpha(l)$  and  $\theta(l)$  are attenuation and phase shift of the  $l$ th path, respectively. The differences in the time of arrival are eliminated by the cyclic prefix, which is described in the next section. One of the effects of the reception of multiple overlapping copies, i.e., multipath, is frequency selective fading. From the signal's perspective, the signal experiences different attenuation for different frequencies, complicating mitigation of these unwanted effects. This is the reason why **OFDM** is used, as it "splits" the spectrum in several subchannels, allowing to use single-tap equalizers to resolve frequency-selective fading. See Section 3.4.3 for more details.

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 **ISI**, with ensuing limitations of the system performance. The common approach to mitigate **ISI** is to introduce a guard interval of appropriate length between adjacent symbols. In practice, the guard interval is obtained by duplicating the last part of the **OFDM** symbol and, for this reason, is commonly referred

to as **cyclic prefix (CP)**. As illustrated in Fig. 3.3, the **CP** is appended in front of the corresponding **IDFT** output. This results into an extended **OFDM** symbol consisted of the **CP** and original **OFDM** symbol. The **ISI** can be removed as long as the **CP** length is properly designed according to the channel delay spread.

Referring to Fig. 3.1, let's concentrate on 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 length of the **CP**). The remaining  $N$  samples are fed to a **DFT** and the corresponding output is subsequently passed to the **frequency domain equalizer (FDE)**. 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 3.4.3. 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.

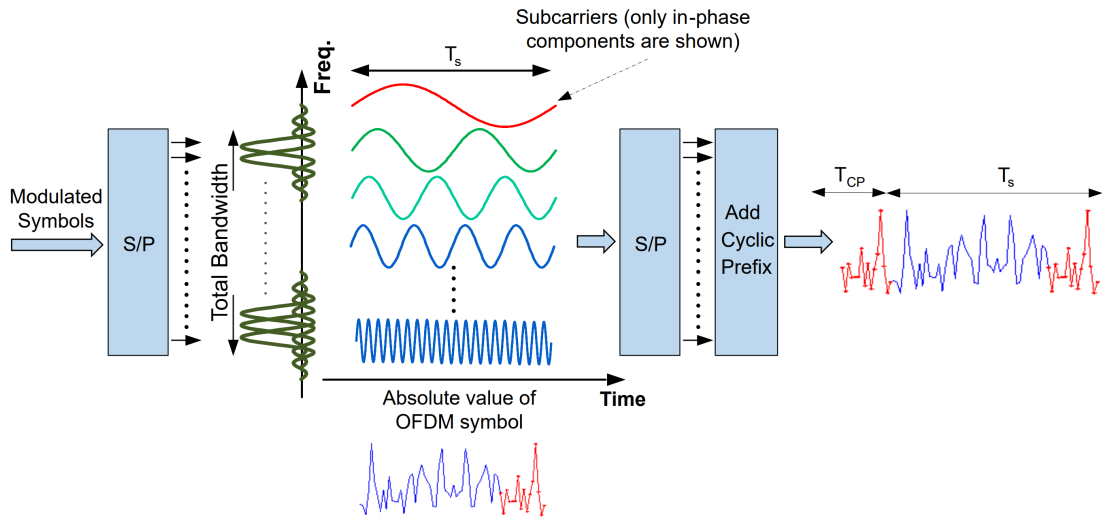


Figure 3.3: Structure of an OFDM symbol (image from [2])

### 3.3 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  $i$ th **OFDM** symbol are taken from either a **phase shift keying (PSK)** or **QAM** constellation. Then, time domain representation of the  $i$ th **OFDM** symbol after **IDFT** and **CP** insertion is given by

$$c_i(k) = \begin{cases} \sum_{n=0}^{N-1} C_i(n) e^{j2\pi kn/N}, & -N_G \leq k \leq N-1 \\ 0, & \text{else} \end{cases}, \quad (3.2)$$

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

$$c(k) = \sum_i c_i(k - iN_T). \quad (3.3)$$

In addition to multipath 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 [5]. 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  $w(k) \sim CN(0, \sigma_w^2)$ , where  $\sigma_w^2$  denotes the noise variance and 0 the mean of the circular symmetric complex distribution. Therefore, the discrete-time model of received OFDM signals can be written as,

$$y(k) = \sum_{l=0}^{L-1} h(l)c(k-l) + w(k). \quad (3.4)$$

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

### 3.4 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 [3]:

- *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  $2\pi k f_d T_s$ <sup>2</sup>. Therefore, the received baseband signal can be expressed as

$$r(k) = y(k)e^{j2\pi\epsilon k/N} \quad (3.5)$$

where

$$\epsilon = N f_d T_s \quad (3.6)$$

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  $\Delta k$  denotes the number of samples by which the

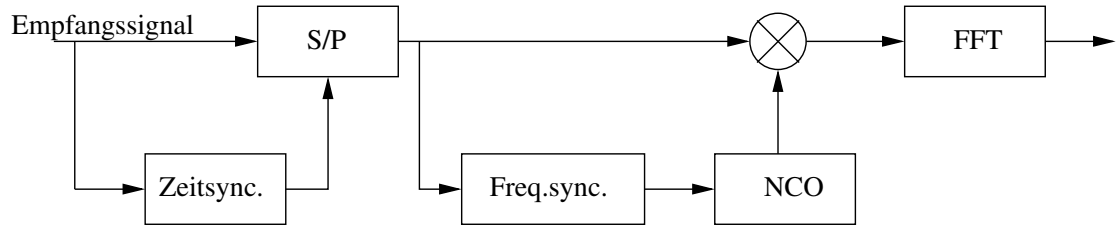


Figure 3.4: Block diagram of a basic OFDM receiver

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

$$r(k) = e^{j2\pi\epsilon k/N} y(k - \Delta k) + w(k). \quad (3.7)$$

Replacing (3.3) and (3.4) in (3.7), samples are given as

$$r(k) = e^{j2\pi\epsilon k/N} \sum_i \sum_{l=0}^{L-1} h(l) c_i(k - l - \Delta k - iN_T) + w(k). \quad (3.8)$$

The frequency and timing synchronization units shown in Fig. 3.4 employ the received samples  $r(k)$  to compute estimates of  $\epsilon$  and  $\Delta k$ , noted as  $\hat{\epsilon}$  and  $\hat{\Delta k}$ . The former is used to counter-rotate  $r(k)$  at an angular speed  $2\pi\hat{\epsilon}k/N$  (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  $i$ th OFDM block.

<sup>2</sup>Notice that this phase shift is not time dependent (if  $f_d$  is constant) and is different for each subcarrier  $k$ .

### 3.4.1 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  $i$ th OFDM symbol is computed as

$$R_i(n) = \frac{1}{N} \sum_{k=0}^{N-1} r(k + iN_T) e^{-j2\pi kn/N}, \quad 0 \leq n \leq N-1 \quad (3.9)$$

Substituting (3.8) into (3.9) we get

$$\begin{aligned} R_i(n) &= \frac{1}{N} \sum_{k=0}^{N-1} \left[ e^{j2\pi\epsilon(k+iN_T)/N} \sum_{l=0}^{L-1} h(l) c_i(k-l) + w(k) \right] e^{-j2\pi kn/N} \\ &= \frac{1}{N} e^{j\varphi_i} \sum_{k=0}^{N-1} e^{j2\pi k(\epsilon-n)/N} \sum_{l=0}^{L-1} h(l) \sum_{m=0}^{N-1} C_i(m) e^{j2\pi(k-l)m/N} + W_i(n) \\ &= \frac{1}{N} e^{j\varphi_i} \sum_{m=0}^{N-1} \left\{ \sum_{l=0}^{L-1} h(l) e^{j2\pi lm/N} \right\} C_i(m) \sum_{k=0}^{N-1} e^{-j2\pi k(m+\epsilon-n)/N} + W_i(n) \\ &= \frac{1}{N} e^{j\varphi_i} \sum_{m=0}^{N-1} H(m) C_i(m) \sum_{k=0}^{N-1} e^{j2\pi k(m+\epsilon-n)/N} + W_i(n) \end{aligned} \quad (3.10)$$

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

$$W_i(n) = \frac{1}{N} \sum_{k=0}^{N-1} w(k) e^{-j2\pi kn/N}, \quad 0 \leq n \leq N-1 \quad (3.11)$$

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

$$H(m) = \sum_{l=0}^{L-1} h(l) e^{-j2\pi lm/N}, \quad 0 \leq m \leq N-1. \quad (3.12)$$

Performing standard mathematical manipulations (3.10) is derived to

$$R_i(n) = e^{j\varphi_i} \sum_{m=0}^{N-1} H(m) C_i(m) f_N(\epsilon + m - n) e^{j\pi(N-1)(\epsilon+m-n)/N} + W_i(n), \quad (3.13)$$

where

$$\begin{aligned} f_N(x) &= \frac{\sin(\pi x)}{N \sin(\pi x/N)} \\ &\approx \frac{\sin(\pi x)}{\pi x} = \text{si}(\pi x) \end{aligned} \quad (3.14)$$

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

In the case when the frequency offset is a multiple of subcarrier spacing  $\Delta f$ , i.e.,  $\epsilon$  is integer-valued, (3.13) reduces to

$$R_i(n) = e^{j\varphi_i} H(|n - \epsilon|_N) C_i(|n - \epsilon|_N) + W_i(n), \quad (3.15)$$



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  $\varepsilon$** . In this case the  $n$ th DFT output is an attenuated and phase-rotated version of  $C_i(|n - \varepsilon|_N)$  rather than of  $C_i(n)$ . Otherwise, when  $\varepsilon$  is not integer-valued the subcarriers are no longer orthogonal and **ICI** does occur. In this case it is convenient to rewrite (3.13) like

$$R_i(n) = e^{j[\varphi_i + \pi\varepsilon(N-1)/N]} H(n)C_i(n)f_N(\varepsilon) + I_i(n, \varepsilon) + W_i(n), \quad (3.16)$$

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

$$I_i(n, \varepsilon) = e^{j\varphi_i} \sum_{m=0, m \neq n}^{N-1} H(m)C_i(m)f_N(\varepsilon + m - n)e^{j\pi(N-1)(\varepsilon+m-n)/N}. \quad (3.17)$$

From (3.16) it follows that non-integer normalized frequency offset  $\varepsilon$  influences the received signal on  $n$ th 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 (3.16) presents interference from other subcarriers (**ICI**).

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

$$\sigma_i^2(\varepsilon) = E\{|I_i(n)|^2\} = S \sum_{\substack{m=0 \\ m \neq n}}^{N-1} f_N^2(\varepsilon + m - n). \quad (3.18)$$

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

$$\sum_{m=0}^{N-1} f_N^2(\varepsilon + m - n) = 1, \quad (3.19)$$

which holds true independently of  $\varepsilon$ , interference power (3.18) can be written as

$$\sigma_i^2(\varepsilon) = S [1 - f_N^2(\varepsilon)]. \quad (3.20)$$

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

$$\gamma(\varepsilon) = \frac{SNR^{(ideal)}}{SNR^{(real)}}, \quad (3.21)$$

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

$$SNR^{(ideal)} = S/\sigma_w^2 = E_S/N_0, \quad (3.22)$$

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

$$SNR^{(real)} = Sf_N^2(\varepsilon) / [\sigma_w^2 + \sigma_i^2(\varepsilon)], \quad (3.23)$$

is the **SNR** in the presence of a frequency offset  $\varepsilon$ . Substituting (3.22) and (3.23) into (3.21), it becomes

$$\gamma(\varepsilon) = \frac{1}{f_N^2(\varepsilon)} \left[ 1 + \frac{E_S}{N_0} (1 - f_N^2(\varepsilon)) \right]. \quad (3.24)$$

For small values of  $\varepsilon$  (3.24) can be simplified using the Taylor series expansion of  $f_N^2(\varepsilon)$  around  $\varepsilon = 0$ , resulting in

$$\gamma(\varepsilon) \approx 1 + \frac{1}{3} \frac{E_S}{N_0} (\pi\varepsilon)^2. \quad (3.25)$$

It can be seen that the SNR loss is approximately proportional to the square of the normalized frequency offset  $\varepsilon$ .

### 3.4.2 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, (3.9) is derived to

$$\begin{aligned} R_i(n) &= \frac{1}{N} \sum_{k=0}^{N-1} \left[ \sum_{l=0}^{L-1} h(l) c_i(k-l-\Delta k) + w_i(k) \right] e^{-j2\pi kn/N} \\ &= \frac{1}{N} \sum_{k=0}^{N-1} \sum_{l=0}^{L-1} h(l) \sum_{m=0}^{N-1} C_i(m) e^{j2\pi(k-l-\Delta k)m/N} e^{-j2\pi kn/N} + W_i(n) \\ &= \frac{1}{N} \sum_{m=0}^{N-1} \left[ \sum_{l=0}^{L-1} h(l) e^{-j2\pi lm/N} \right] C_i(m) \left[ \sum_{k=0}^{N-1} e^{-j2\pi k(m-n)/N} \right] e^{-j2\pi \Delta k m/N} + W_i(n) \\ &= \sum_{m=0}^{N-1} H(m) C_i(m) \delta(m-n) e^{-j2\pi \Delta k m/N} + W_i(n) \\ &= H(n) C_i(n) e^{-j2\pi \Delta k n/N} + W_i(n), \end{aligned} \quad (3.26)$$

where  $W_i(n)$  and  $H(m)$  are defined in (3.11) and (3.12), respectively, while  $\delta(m-n)$  presents the Kronecker delta defined as

$$\delta(m-n) = \begin{cases} 1, & m = n \\ 0, & m \neq n. \end{cases} \quad (3.27)$$

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.

### 3.4.3 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  $i$ th symbol is given by

$$R_i(n) = H_i(n)C_i(n) + W_i(n), \quad 0 \leq n \leq N - 1 \quad (3.28)$$

where  $C_i(n)$  is the complex data symbol and  $W_i(n)$  as well as  $H(m)$  are defined in (3.11) and (3.12), 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  $n$ th 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

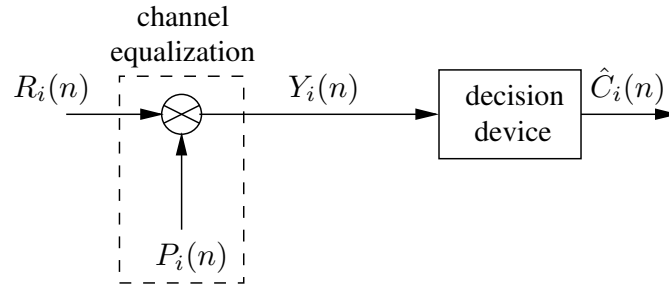


Figure 3.5: Block diagram of an OFDM receiver

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

$$P_i(n) = \frac{1}{H_i(n)}, \quad (3.29)$$

while the DFT output takes the form

$$Y_i(n) = \frac{R_i(n)}{H_i(n)} = C_i(n) + \frac{W_i(n)}{H_i(n)}, \quad 0 \leq n \leq N - 1. \quad (3.30)$$

From (3.30) 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 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 (3.28) can be written as

$$R_p(n) = H(n)C_p(n) + W_p(n), \quad 0 \leq n \leq N - 1 \quad (3.31)$$

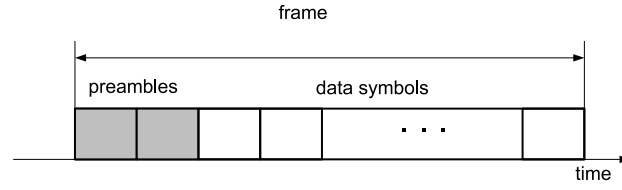


Figure 3.6: Frame structure

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

$$\hat{H}(n) = \frac{R_p(n)}{C_p(n)} = H(n) + \frac{W_p(n)}{C_p(n)}, \quad 0 \leq n \leq N - 1. \quad (3.32)$$

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.

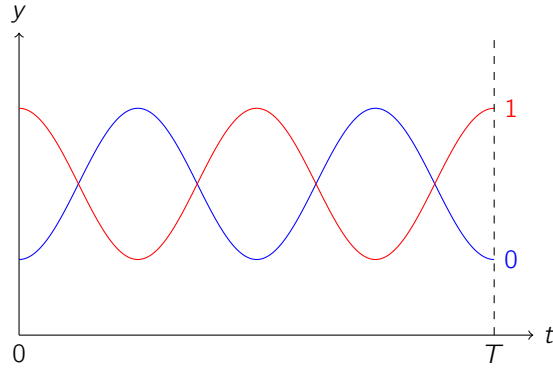
### 3.5 Digital Modulations Used in OFDM Systems

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 (on the **RF** frequency) 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. 3.8. Such a mapping rule is called a digital modulation scheme. An example is shown in Fig. 3.7b, where the red signal represent a logical 1 and the blue signal a logical 0. The signal frequency is the carrier frequency, i.e., it is the passband signal. In **base-band**, this would be represented by one complex symbol per passband signal. To go from the passband to the **base-band (BB)**, the carrier frequency is removed from the passband signal, thus the only thing remaining is the amplitude and phase contributions, i.e., the complex symbol.

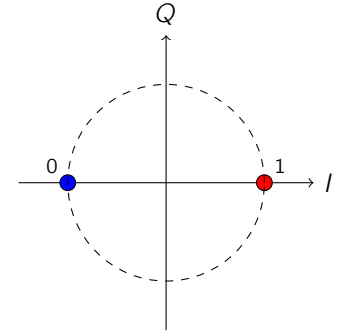
A linear digital modulation scheme is characterized by the complex baseband signal [6]

$$C(t) = \sum_i C_i g(t - kT), \quad (3.33)$$

where  $C_i$  is a given constellation point (complex) and  $g(t)$  is a pulse shape used for transmission. In Fig. 3.7b, the pulse shape is a rectangular window in the time domain, meaning that the signal is transmitted over a finite time, the symbol time  $T$ . 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.



(a) BPSK passband signal representation.



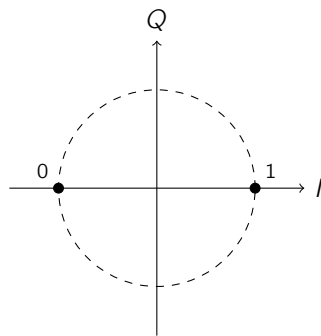
(b) BPSK complex BB signal representation.

### 3.5.1 Phase Shift Keying (PSK)

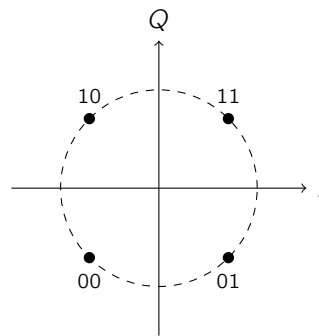
PSK or Multiple PSK (M-PSK) modulation, where  $M$  is the number of constellation points, is characterized that all signal information is put into the phase of the transmitted signal, preserving constant envelope property (the amplitude between symbols are constant). The M-PSK complex symbol  $C_i$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

$$C_i = \sqrt{S} e^{j(\frac{2\pi m}{M} + \theta_0)}, \quad m = 0, 1, \dots, M-1, \quad (3.34)$$

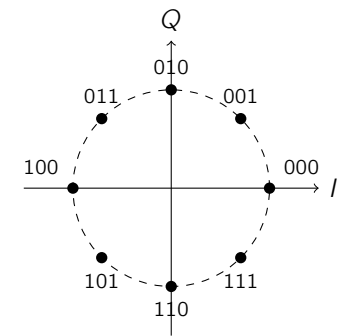
where  $S$  is the average signal power and  $\theta_0$  is an arbitrary constant phase. Constellation diagrams for  $M = 2, 4, 8$ , i.e., binary phase-shift keying (BPSK), quadrature phase-shift keying (QPSK) or 4-PSK and 8-PSK, respectively, when  $\theta_0 = 0$ , are shown in Fig. 3.8.



(a) BPSK constellation diagram



(b) QPSK(4-PSK) constellation diagram



(c) 8-PSK constellation diagram

Figure 3.8: Gray-coded constellation diagrams

The simplest PSK modulation format is BPSK, where a logical „1“ is encoded as 0 phase, and a logical „0“ is coded as a phase of  $\pi$ . Then, the modulated symbol, defined in (3.34), can be written as

$$C_i = \pm\sqrt{S} \quad (3.35)$$

with constellation diagram shown in Fig. 3.8a. M-PSK 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. 3.8b and Fig. 3.8c, 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. 3.8. 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.

### 3.5.2 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.

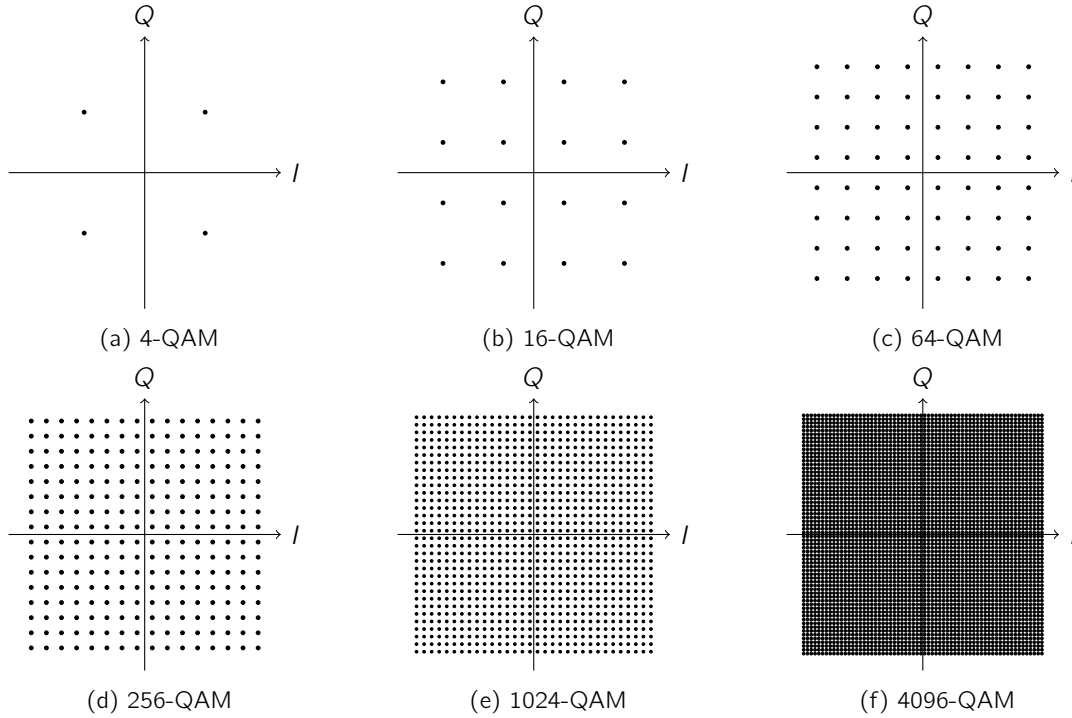


Figure 3.9: QAM constellation diagrams

QAM modulated signals for  $M$  constellation points can be written as

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

where  $X_i, Y_i \in \{\pm 1, \pm 3, \dots, \sqrt{M} - 1\}$  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. 3.9. It can be noticed that 4-QAM corresponds to QPSK with constant phase shift  $\theta_0 = \pi/4$ .

QAM schemes are used in several typical wireless digital communications specifications for a long time. A trend is the drastic increase in modulation depth as can be seen in Table 3.2.

Table 3.1: Modulation dependent parameters

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 3.2: Overview of maximum modulation depth in Wi-Fi. [7]

Wi-Fi standard	IEEE standard	Modulation Depth	bits per symbol
Wi-Fi 5	802.11ac	256-QAM	8
Wi-Fi 6(E)	802.11ax	1024-QAM	10
Wi-Fi 7	802.11be	4096-QAM	12
Wi-Fi 8	802.11bn	8192-QAM	13

## 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 [8]:

“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 [9].



## 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.

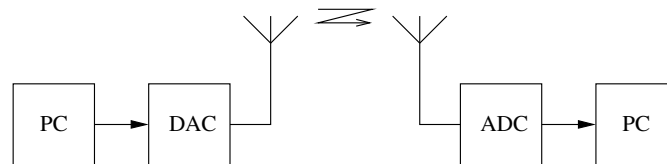


Figure 4.1: Ideal SDR transmission

- **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<sup>1</sup> 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).

<sup>1</sup>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 transferring 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).

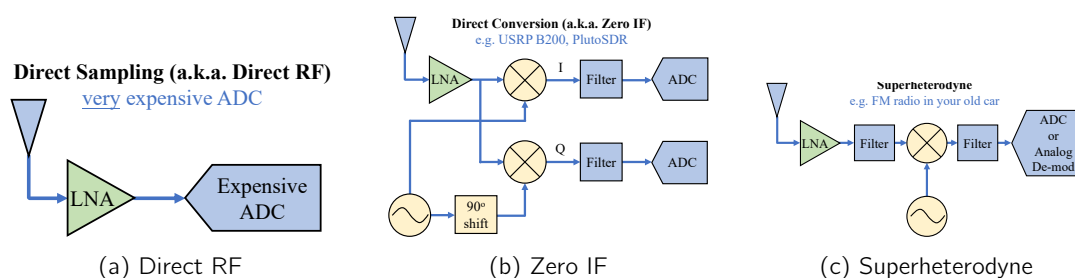


Figure 4.2: RF Architectures (from [pysdr.org](http://pysdr.org))

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 [9]. 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 [10]–[14]. 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., [15]. 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.<sup>2</sup>

### 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.

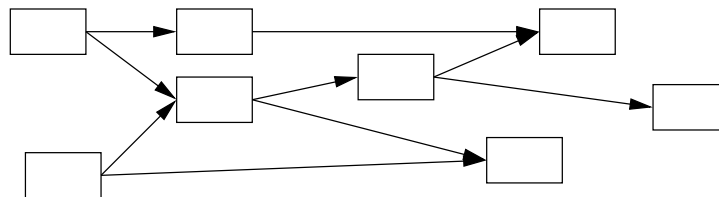


Figure 4.3: An example of **flowgraph**

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

<sup>2</sup>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).

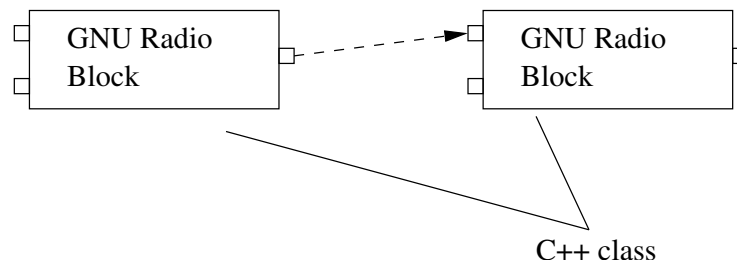


Figure 4.4: GNU Radio blocks

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

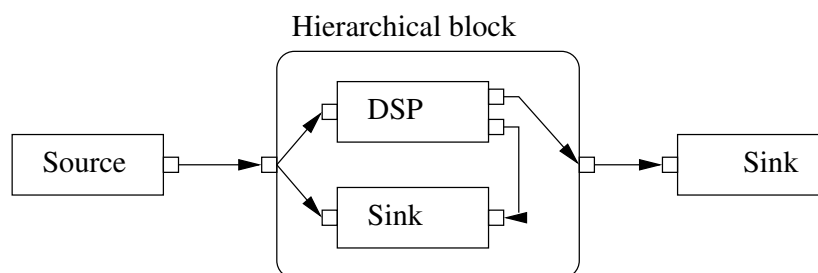


Figure 4.5: An example of a flowgraph with a hierarchical block

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.

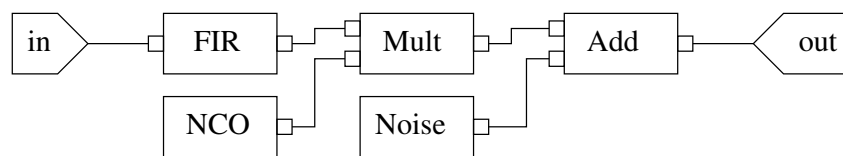


Figure 4.6: Wireless communication channel simulation model

## 4.3 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

## OFDM in GNU Radio

This chapter is an extension of the [OFDM](#) tutorial of the GNU Radio Wiki<sup>1</sup>.

### 5.1 Conventions and Notations

#### 5.1.1 FFT Shifting

In all cases where [OFDM](#) symbols are passed between blocks, the default behaviour is to [FFT](#)-Shift these symbols, i.e., that the [direct current \(DC\)](#) carrier is in the middle (to be precise, it is on carrier floor( $N/2$ ) where  $N$  is the [FFT](#) length and carrier indexing starts at 0). This means that when viewing [OFDM](#) symbols, [FFT](#)-shifted symbols are in their natural order, i.e., as they appear in the pass band.

#### 5.1.2 Carrier Indexing

Carriers are always index starting at the DC carrier, which has the index 0 (you usually don't want to occupy this carrier). The carriers right of the DC carrier (the ones at higher frequencies) are indexed with 1 through  $N/2 - 1$  ( $N$  being the FFT length again).

The carriers left of the DC carrier (with lower frequencies) can be indexed  $-N/2$  through  $-1$  or  $N/2$  through  $N - 1$ . Carrier indices  $N - 1$  and  $-1$  are thus equivalent. *The advantage of using negative carrier indices is that the FFT length can be changed without changing the carrier indexing.* For this reason, usage of negative carrier indices are recommended.

#### 5.1.3 Carrier and Symbol Allocation

Many blocks require knowledge of which carriers are allocated, and whether they carry data or pilot symbols. GNU Radio blocks uses three objects for this, typically called `occupied_carriers` (for

---

<sup>1</sup>[https://wiki.gnuradio.org/index.php?title=Basic\\_OFDM\\_Tutorial](https://wiki.gnuradio.org/index.php?title=Basic_OFDM_Tutorial)

the data symbols), `pilot_carriers` and `pilot_symbols` (for the pilot symbols). We advise you to follow these naming conventions.

Every one of these objects is a vector of vectors. `occupied_carriers` and `pilot_carriers` identify the position within a *frame* where data and pilot symbols are stored, respectively.

`occupied_carriers[0]` identifies which carriers are occupied on the first OFDM symbol, `occupied_carriers[1]` does the same on the second OFDM symbol etc.

Here's an example:

```
occupied_carriers = ((-2, -1, 1, 3), (-3, -1, 1, 2))
pilot_carriers = ((-3, 2), (-2, 3))
```

Every OFDM symbol carries 4 data symbols. On the first OFDM symbol, they are on carriers -2, -1, 1 and 3. Carriers -3 and 2 are not used, so they are where the pilot symbols can be placed. On the second OFDM symbol, the occupied carriers are -3, -1, 1 and 2. The pilot symbols must thus be placed elsewhere, and are put on carriers -2 and 3.

If there are more symbols in the OFDM frame than the length of `occupied_carriers` or `pilot_carriers`, they wrap around (in this example, the third OFDM symbol uses the allocation in `occupied_carriers[0]`). We do not advise you to make use of this wrap around function, but rather recommend to properly define the carrier allocations.

But how are the pilot symbols set? This is a valid parameterization:

```
pilot_symbols = ((-1, 1j), (1, -1j), (-1, 1j), (-1j, 1))
```

The position of these symbols are those in `pilot_carriers`. So on the first OFDM symbol, carrier -3 will transmit a -1, and carrier 2 will transmit a 1j. Note that `pilot_symbols` is longer than `pilot_carriers` in this example— this is valid, the symbols in `pilot_symbols[2]` will be mapped according to `pilot_carriers[0]`. Again, we do not advise you to make use of wrap around functionality.

## 5.2 Transmitting

It is assumed that the input is a stream of complex scalars with a length tag, i.e., the transmitter will work on one frame at a time. An example flow graph can be found on: [https://github.com/gnuradio/gnuradio/blob/main/gr-digital/examples/ofdm/ofdm\\_loopback.grc](https://github.com/gnuradio/gnuradio/blob/main/gr-digital/examples/ofdm/ofdm_loopback.grc). This block is defined in [https://github.com/gnuradio/gnuradio/blob/main/gr-digital/python/digital/ofdm\\_txx.py](https://github.com/gnuradio/gnuradio/blob/main/gr-digital/python/digital/ofdm_txx.py).

First, the input scalars are allocated according to the OFDM Carrier Allocator block. This sorts the incoming complex scalars onto OFDM carriers, and also places the pilot symbols onto the correct positions. There is also the option to pass OFDM symbols which are prepended in front of every frame (i.e. preamble symbols). These can be used for detection, synchronization, and channel estimation.

The carrier allocator outputs OFDM symbols (i.e. complex vectors of FFT length). These must be converted to time domain signals before continuing, which is why they are piped into the FFT

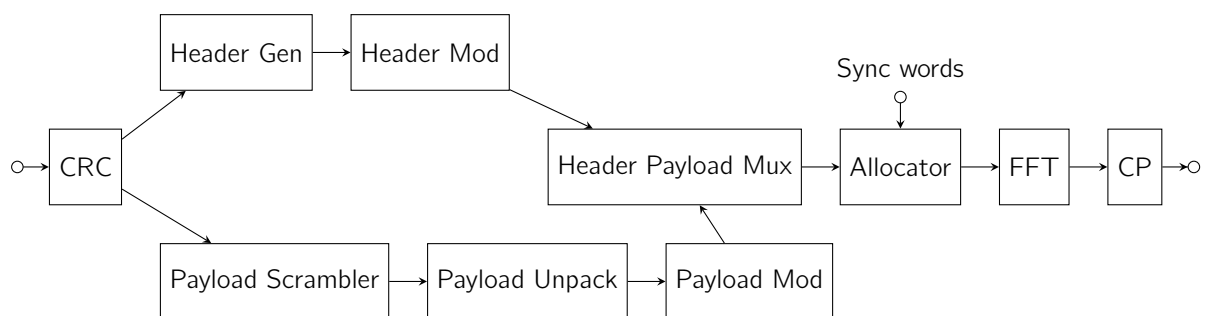
FFT Length	64
Data subcarriers	52 (-26 to 26)
Pilot subcarriers (BPSK)	4 (-21, -7, 7, 21)
DC subcarrier	Null
Bandwidth (BW)	20 MHz
Occupied bandwidth (OBW)	16.6 MHz
Subcarrier Frequency Spacing	312.5 kHz (20 MHz/64)

Table 5.1: 802.11a OFDM

block. Note that because all the OFDM symbols are treated in the shifted form, the IFFT block (i.e., reversed FFT block) must be shifting as well.

Finally, the cyclic prefix is added to the OFDM symbols, through the `OFDM Cyclic Prefixer`.

Next to this, the `OFDM Transmitter` block performs prepending two sync words, header, and payload modulation. For the remainder of the lab session, make use of the transmitter block, instead of separate blocks.



### 5.2.1 OFDM Transmitter

#### Parameters

- FFT Length**  
 The number of sub-carriers used per OFDM symbol.
- Cyclic Prefix Length**  
 The maximum possible length of multi-paths in their time dispersion. Conventionally, the **CP** length is equal to  $\text{FFT}/4$ .
- Packet Length Tag Key**  
 The name of the tag giving packet length at the input. This is propagated through the flow graph in order for all blocks to know how long each packet is.
- Occupied Carriers**  
 A vector of vectors describing which OFDM carriers are occupied with data symbols.



`occupied_carriers[0]` identifies the carriers that are used for the first OFDM symbol, and so on. See Section 5.1 for details and conventions.

- **Pilot Carriers**

A vector of vectors describing which OFDM carriers are occupied with pilot symbols. See Section 5.1 for details and conventions.

- **Pilot Symbols**

A vector of vectors indicating the pilot symbols. See Section 5.1 for details and conventions.

- **Sync Word 1**

The first sync preamble symbol. This has to be with zeros on alternating carriers, e.g., (0., 1.536, 0., -1.536, 0., ...). Used for fine and coarse frequency offset and timing estimation. Length of sync sequence must be the same as the value of FFT length. If none is passed a random sync sequence is created for fine frequency offset and timing estimation. This is the first of typically two sync preamble symbols for the Schmidl & Cox sync algorithm. The relevant feature of these symbols is that every second sub-carrier is zero. In the time domain, this results in two identical halves of the OFDM symbols. Symbols are always BPSK symbols. Carriers are scaled by  $\sqrt{2}$  to keep total energy constant. Note, that the default sync word only contains symbols on the `occupied_carriers` indices.

- **Sync Word 2**

The second sync preamble symbol. This has to be filled entirely. Also used for coarse frequency offset and channel estimation. Length of sync sequence must be the same as the value of FFT length. If None is used, it creates a random sync sequence for coarse frequency offset and channel estimation. This is the second of typically two sync preamble symbols for the Schmidl & Cox sync algorithm. Symbols are always BPSK symbols. No scaling is applied, as all carriers (`occupied_carriers`) are used.

- **Header Modulation**

It has two options:

- binary phase-shift keying (BPSK)
- quadrature phase-shift keying (QPSK)

- **Payload Modulation**

It has three options:

- binary phase-shift keying (BPSK)
- quadrature phase-shift keying (QPSK)
- 8-PSK (Eight Phase Shift Keying)

## 5.3 Receiver

### 5.3.1 Detection and Synchronization

Before anything happens, an OFDM frame must be detected, the beginning of OFDM symbols must be identified, and frequency offset must be estimated. The effects of these synchronization and

timing offsets are detailed in Section 3.4. Here, we will focus on the Schmidl & Cox method [16]. This is implemented in the GNU Radio block *Schmidl & Cox OFDM synch*.<sup>2</sup>

The evaluation of the coarse frequency offset is not done in this block. Also, the initial equalizer taps are not calculated here. This means the block performs a fine-grained frequency offset estimation and sync word detection, which you can use later for timing offset correction, i.e., start-of-frame detection.

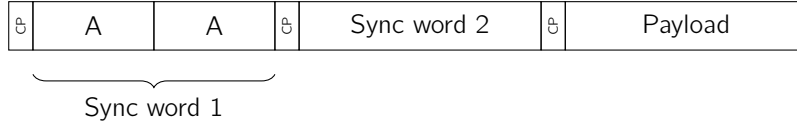


Figure 5.1: Illustration of OFDM frame in GNU Radio.

The OFDM frame is illustrated in Fig. 5.1. The first part comprises a CP and the sync word 1, which in turn consists of two identical sequences. Often the sync word 1 is constructed by only populating the carriers every second carrier, i.e., upsampling in the frequency domain, this yields two identical sequences in the time domain (A in Fig. 5.1). The sequences are designed such that their autocorrelation function has a sharp peak, being close to the dirac function. In order to thus find the start of frame, you need to find the sample offset where the correlation is highest.

The correlation metric  $P(d)$  for the received samples  $z[n]$  can be written as,

$$P(d) = \sum_{i=0}^{N/2-1} z \left[ \left( d + \frac{N}{2} \right) + i \right] z^*[d+i]. \quad (5.1)$$

In other words, this function starts at index  $d$  and computes the correlation of a sample at index  $d+i$  with a sample  $\frac{N}{2}$  samples further. Note, that the correlation is computed by taking the product of the conjugate of one sample with another sample. This function effectively takes two sequential windows of size  $N/2$  and computes their correlation. If perfectly aligned, both windows contain the sequence A and the correlation function yields the highest value. This method would work perfectly in ideal cases, but unfortunately, reality is never ideal.

In real wireless systems, the signal will experience different attenuation at different time instances, resulting in a different magnitude of the received sequences (A), making the above referenced correlation method insufficient. To combat this, the metric  $P(d)$  can be normalized by the energy of the samples within the correlation window.

In [16], the energy of the second part is computed only,

$$R(d) = \sum_{i=0}^{N/2-1} \left| z \left[ \left( d + \frac{N}{2} \right) + i \right] \right|^2.$$

This differs from the computation in GNU Radio which computes the energy of both halves. This avoids spurious detects at the end of a burst, when the energy level suddenly drops.

The new timing synchronization metric  $M(d)$  is defined as,

<sup>2</sup>[https://wiki.gnuradio.org/index.php/Schmidl\\_%26\\_Cox\\_OFDM\\_synch](https://wiki.gnuradio.org/index.php/Schmidl_%26_Cox_OFDM_synch).

$$M(\epsilon_0) = \frac{|P(d)|^2}{R(d)^2}. \quad (5.2)$$

And the optimal timing instant is given as,

$$\hat{d} = \max_d M(d) \quad (5.3)$$

GNU Radio uses a plateau detector<sup>3</sup> to estimate  $\hat{d}$ . This detector outputs a 1 when the magnitude exceeds a configurable threshold for more than one sample. More extensive and complex detectors can be used, e.g., matched filtering.

---

<sup>3</sup>[https://github.com/gnuradio/gnuradio/blob/main/gr-blocks/lib/plateau\\_detector\\_fb\\_impl.cc](https://github.com/gnuradio/gnuradio/blob/main/gr-blocks/lib/plateau_detector_fb_impl.cc)

# Chapter 6

## Assignments

### 6.1 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.<sup>1</sup>

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 (1.5 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. [6.1](#).

Procedure antenna:

1. Calibrate the NanoVNA ([nanovna.com](https://nanovna.com)) to the desired frequency band (procedure from <https://nanorfe.com/nanovna-v2-user-manual.html>).
  - Calibration must be performed whenever the frequency range to be measured is changed. When calibration is activated, the left side of the screen should show Cx and D.
  - Changing the frequency sweep range always clears the active calibration, if any.
  - The calibration procedure is as follows:

---

<sup>1</sup>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 its 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.

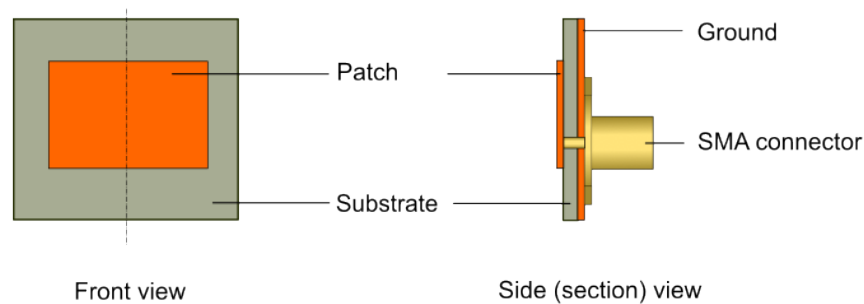


Figure 6.1: Illustration of a microstrip patch antenna on a finite ground using a pin feed (from Altair Community)

- Reset current calibration state. Select menu item CAL RESET and then CALIBRATE.
  - Attach a SMA coaxial cable to port 1.
  - Connect OPEN standard to port 1 cable and click OPEN. Wait for menu item highlight.
  - Connect SHORT standard to port 1 cable and click SHORT. Wait for menu item highlight.
  - Connect LOAD standard to port 1 cable and click LOAD. Wait for menu item highlight.
  - Click DONE.
  - Specify the dataset number (0 to 4) and save. e.g. SAVE 0.
  - Note that there is no need to wait for the plots to fully update after connecting a calibration standard. Clicking any of the OPEN, SHORT, LOAD, THRU calibration menu items will perform a full sweep with 2x averaging. Once the sweep is complete the corresponding menu item will become highlighted, and you may proceed to the next calibration standard.
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.
  5. Refer to the S-parameter details at [www.antenna-theory.com/definitions/sparameters.php](http://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.

## 6.2 Construct your own USRP housing

To not damage the delicate electronics, the B210 should be sealed via a casing. See Fig. 6.2 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.

## 6.3 Getting started with GNU Radio

Read Chapter 4 thoroughly before installing GRC.

Follow the following sections on <https://wiki.gnuradio.org/index.php?title=Tutorials>:

- Beginner Tutorials
  - Introducing GNU Radio
  - Flowgraph Fundamentals
- Intermediate/Advanced Tutorials
  - Miscellaneous
    - \* Using GNU Radio With SDRs
    - \* IQ and Complex Signals
    - \* Understanding Sample Rate

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 familiar with concepts such as complex BB and pass band ([https://wiki.gnuradio.org/index.php?title=IQ\\_Complex\\_Tutorial](https://wiki.gnuradio.org/index.php?title=IQ_Complex_Tutorial)).

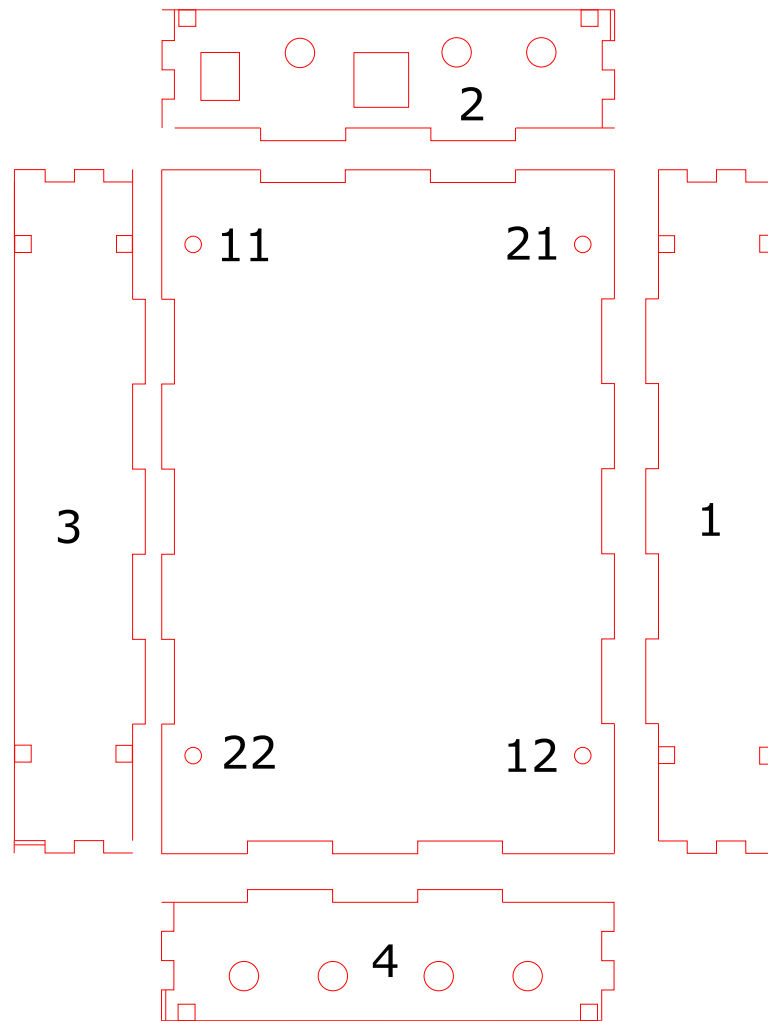


Figure 6.2: Exploded view of the B210 casing. The numbers indicate which component should be placed where.

## 6.4 Getting to know OFDM

Start by reading Chapter 3 with a special focus on Section 3.1, 3.2, 3.4.3, 3.5. After this the building blocks of OFDM should be clear.

After completing this, consult the *Basic OFDM Tutorial* of GR ([https://wiki.gnuradio.org/index.php?title=Basic\\_OFDM\\_Tutorial](https://wiki.gnuradio.org/index.php?title=Basic_OFDM_Tutorial)). Create the *Example loopback flowgraph*. After this, and grasping all the required knowledge to thoroughly understand this flowgraph, run this flowgraph and ask your teaching assistant to evaluate your progress.

Now, that you are familiar with the basics of OFDM, we will dive deeper into the details of the implementation in GNU Radio. Read Chapter 5 and construct an OFDM frame in your programming language of choice. Compute  $M(d)$  and demonstrate that you are able to find the correct time sample under AWGN. Use a parameter SNR to define the noise level.

Following the OFDM transmitter structure, find the structure of the receiver, as defined in [https://github.com/gnuradio/gnuradio/gr-digital/python/digital/ofdm\\_txxr.py](https://github.com/gnuradio/gnuradio/gr-digital/python/digital/ofdm_txxr.py). To do this, write down all block connections (`self.connect`) and find search for the details regarding each block and function call. Examples are: `ofdm_frame_equalizer_vcvc`, `ofdm_sync_sc_cfb` and `channelest`. You can use GRC to help replicate the internal flow graph if the OFDM receiver.





# Chapter 7

## 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.

Things you need to know (non-exhaustive list):

- What is meant by **base-band (BB)**?
- How does the B210 get **in-phase and quadrature (IQ)** samples?
- What was the relation between the patch antenna area and the carrier frequency of the antenna?
- What does the S11-parameter tell you about your antenna?
- How does *pack K bits* block work in **GNU Radio Companion (GRC)**?
- What is the *throttle* block, and when do you (not) need it?
- What is the difference between streams and vectors in GNU Radio?
- How do you calibrate a **vector network analyzer (VNA)** and what does each step do?
- Which RF architecture is used by the **universal software radio peripheral (USRP)** B210?
- What is frequency selective fading?
- How is **intersymbol interference (ISI)** mitigated in **orthogonal frequency-division multiplexing (OFDM)** systems?
- What is the relation between a **N-quadrature amplitude modulation (QAM)** (with N the number of constellation points) and the number of bits per symbol?

- What technique is used in OFDM to go from a set of parallel symbols to one OFDM symbol.
- Why does the *Schmidl & Cox Synchronization* block does not need to a-priori know the sync word symbols?

# Bibliography

- [1] "GNU Radio", <http://gnuradio.org/trac>. [Online]. Available: <http://gnuradio.org/trac>.
- [2] H. Arslan and E. Baar, *Flexible and cognitive radio access technologies for 5G and beyond*. Institution of Engineering and Technology, 2020.
- [3] 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.
- [4] V. Milosevic, B. Jokanovic, O. Boric-Lubecke, and V. M. Lubecke, "Key microwave and millimeter wave technologies for 5g radio," in *Research Anthology on Developing and Optimizing 5G Networks and the Impact on Society*, IGI global, 2021, pp. 770–798.
- [5] 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.
- [6] T. J. Roupheal, *RF and Digital Signal Processing for Software-Defined Radio: A Multi-Standard Multi-Mode Approach*. Newton, MA, USA: Newnes, 2008, isbn: 0750682108, 9780750682107.
- [7] Wikipedia contributors, *IEEE 802.11 — Wikipedia, the free encyclopedia*, [https://en.wikipedia.org/w/index.php?title=IEEE\\_802.11&oldid=1210657347](https://en.wikipedia.org/w/index.php?title=IEEE_802.11&oldid=1210657347), [Online; accessed 2-March-2024], 2024.
- [8] J. I. Mitola, *Cognitive Radio Architecture: The Engineering Foundations of Radio XML*. New York, NY USA: Wiley-Interscience, 2006, isbn: 0471742449.
- [9] A. Mueller, *DAB, Software Receiver Implementation*. ETH Zurich: Semester Thesis, 2008.
- [10] *Effektiv C++ programmieren*.
- [11] *Mehr Effektiv C++ programmieren*.
- [12] S. Meyers, *Effective STL*. Addison Wesley, 2001.
- [13] *Die C++ Programmiersprache*.
- [14] *C++ Coding Standards*.
- [15] A. Martelli, *Python in a Nutshell. A Desktop Quick Reference*, 1st ed. institute: O'Reilly & Associates, 2003.
- [16] T. Schmidl and D. Cox, "Robust frequency and timing synchronization for ofdm," *IEEE Transactions on Communications*, vol. 45, no. 12, pp. 1613–1621, 1997. doi: [10.1109/26.650240](https://doi.org/10.1109/26.650240).