Software Defined Radio Laboratory

Chair of Theoretical Information Technology

Summer 2022
TU München

Preface

This document is a manual for the Software Defined Radio Laboratory offered by the Chair of Theoretical Information Technology (Prof. Boche) at Technical University of Munich.

This document was written by Luis Torres-Figueroa, Ullrich Mönich, Emre Durmaz, Hammad Zafar, Anna Frank, Johannes Voichtleitner, Xinyang Li and Vlad Andrei. Please report any mistakes or typos you may find to Vlad Andrei (vlad.andrei@tum.de).

München, October 2022
Vlad Andrei

Version 6.2
January 25, 2023

Contents

1 Introduction

This laboratory focuses on the architecture, design and implementation of a single-antenna, multi-carrier digital transceiver. More complex topics such as MU-MIMO systems, as well as higher-layer aspects, are out of the current scope. In each session, relevant mathematical background behind each experiment will be explained before discussing specific algorithms deployed in widely-available technologies. Furthermore, we will not stay at the algorithmic level, but will progressively implement signal processing blocks at the transmitter and receiver chains with the aid of programmable software defined radios (SDR) and a graphical programming language called LabVIEW NXG.

Each experiment will focus on a different aspect of the communication system and after completing the experiments on this module, you will have implemented a communication system using real hardware, and demonstrated its efficiency by performing data transmission while monitoring key performance indicators, such as the block error rate.

Even though the experiments mainly focus on implementing the given communication system with an SDR, sometimes you will be asked to implement simulations as a preparation to the following tasks. For such tasks, mostly the main block of the simulation will be provided to you and you will be asked to replace various blocks that perform necessary functionalities. The input and output relationships of these blocks will be provided in the form of a table. Besides, for blocks that require a sophisticated algorithm design, the control flows will also be provided.

A complete communication system with several blocks can be seen in Fig. 1.1.

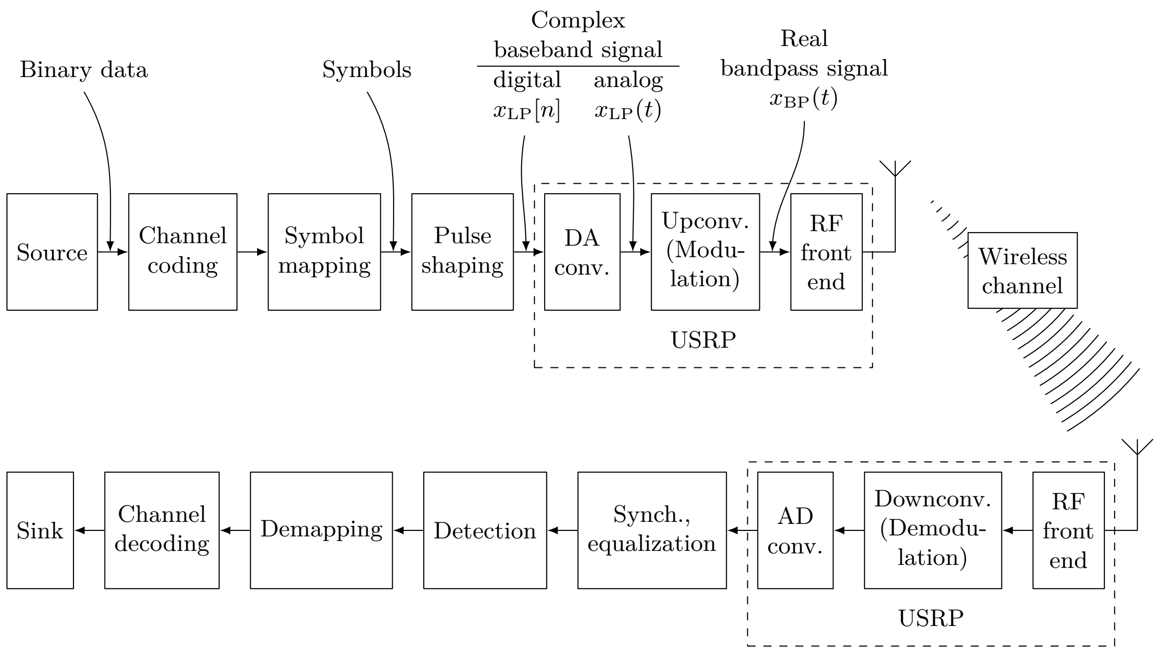


Figure 1.1: Overview of a digital communication system.

1.1 Wireless Transmission

Note that there are strict regulations about the frequencies that can be used for wireless transmission. For teaching purposes in Germany, it is allowed to transmit in the frequency ranges 2.4 GHz–2.5 GHz and 5.725 GHz–5.875 GHz with low transmit power in buildings, when taking care that no other systems are disturbed (“Demonstrationsfunk für Bildungseinrichtungen”). When transmitting, please make sure that you stay within the above frequency ranges.

1.2 Timeline

Day	Date	Topic	Protocol
	20. Oct. 2022	Introduction and Course Overview	
1	27. Oct. 2022	Introduction to the LabVIEW and USRPs, Spectrum Analyzer and FM Receiver	
2	03. Nov. 2022	Symbol Mapping, Pulse Shaping and Matched Filtering	
3	10. Nov. 2022	Digital Receivers and Carrier Synchronization	P. 1 due
4	17. Nov. 2022	Symbol Synchronization	P. 2 due
5	24. Nov. 2022	Frame Synchronization	P. 3 due
	01. Dec. 2022	No Lab (Public Holiday)	
6	08. Dec. 2022	Channel Estimation and Equalization	P. 4 due
7	15. Dec. 2022	Orthogonal Frequency-Division Multiplexing	P. 5 due
	22. Dec. 2022	No Lab	
	29. Dec. 2022	No Lab	
	05. Jan. 2022	No Lab	
8	12. Jan. 2022	Channel Coding	P. 6 due
9	19. Jan. 2022	Physical Layer Security	P. 7 due
10	26. Jan. 2022	Full Transmitter and Receiver Chain	P. 8 due
	02. Feb. 2022	Reserve	P. 9 due
	09. Feb. 2022	Oral Examination	P. 10 due

Table 1.1: Schedule of the software defined radio laboratory.

1.3 Computer Setup

The login to the laboratory computers is done with your LRZ username and password.
user name: your LRZ username
password: your LRZ password

After your first login, make sure that *Ni LabVIEW NXG 4.0* and the *NI-USRP Configuration Utility* are installed.

1.4 General Considerations

You will be working in groups of two and are strongly encouraged to discuss and help each other. Furthermore you should ensure that each of you gets the chance to use the devices, program and write reports equally.

Please be present at the beginning of each day of the course at 13:00, because we will give a short introduction to the topic and provide you with important information. After the introduction you are free to organize your time as you please, that is, there is no obligation for you to be present. You can also work at home (there is a free student version of LabVIEW) for parts where you do not need the hardware or when writing the protocol.

You might also alternatively come to the lab on other days than the lab sessions, if you have not finished the work. Just make sure you inform one of the instructors. Also please verify that your student card can unlock the lab door.

1.5 Lab Protocols

You will document all your experiments and results in a lab protocol. These protocols are graded and 60% of your final grade is based on the protocols. The other 40% of you grade are determined by an oral examination at the end of the course. It is sufficient that one group member uploads the material to Moodle.

The protocol for each experiment/day has to be uploaded the latest on the day shown in Table 1.1. The deadline is 23:59. If for any reason you are unable to return the lab protocol, please contact with your instructor to discuss the next steps. An unexcused late submission of the lab protocol will in any case lead to a lower grade.

All protocols must contain the following points:

- Cover sheet with title, date and time, group name (`group<groupID>`) as well as names and matriculation numbers of both group members.
- Short introduction to the topic (as concretely as possible related to the lab tasks). Please do not copy sections from the script or books.
- Solution of the preparatory tasks.
- Description of the experimental procedure: explanation and justification of all important commands, block diagrams to illustrate functionalities.
- Detailed analysis of the obtained results: Comparison of the results with the theory, discussion of possible errors and their causes.

Furthermore, the following should be noted:

- Clear presentation, meaningful outline, numbered headings.
- Label pictures, sketches, diagrams etc, number and mention them in the main text.
- Pay attention to spelling and grammar!
- The protocol for each lab exercise should contain about 4 to 6 pages of pure text (with 2.5cm margin, 12pt font and simple line spacing), so with graphics you typically get 6 to 10 pages.

Grading of the lab protocols:

- For each protocol you get points.
- At the end you will get one single grade for all protocols together. This grade is determined based on the number of points you achieved, compared to the maximum possible number of points.
- The maximum number of points that you can get for a certain task (pre-lab / experiment) is written in a square box in the section header of the corresponding task.
- Additionally, for each protocol, a maximum of 1 point is awarded for the “overall quality” (depending on the number typos, visual quality, etc.).
- Fractional points, e.g. 2.3 are possible.

1.6 Oral Exam

The date for the oral exam is the 09. February 2022, from 13:00 onwards. The exact schedule will be given some days in advance. If you have any problems (e.g. collisions with other exams) or if you can not participate, do not hesitate to contact us.

2 Introduction to the LabVIEW Environment (Day 1)

In this laboratory you will learn the basic functionalities of the *LabVIEW NXG* 4.0.0 programming software. Further, you will implement your first program.

Note that LabVIEW NXG is a different software than LabVIEW without "NXG". National Instruments calls LabVIEW NXG "the next generation of LabVIEW". For simplicity, we will write LabVIEW throughout this document, even though we are actually referring to LabVIEW NXG.

LabVIEW NXG is still being developed. Some functionalities available in LabVIEW are not yet available in LabVIEW NXG. Therefore, if you need to check available functionalities in your software, make sure you read the most updated resources of LabVIEW NXG.

> "LabVIEW is a graphical programming language developed by National Instruments. The basic building block of LabVIEW is the virtual instrument (VI). Conceptually, a VI is analogous to a procedure or function in conventional programming languages. Each VI consists of a block diagram and a front panel. The block diagram describes the functionality of the VI, while the front panel is a top level interface to the VI. The construct of the VI provides two important virtues of LabVIEW: code reuse and modularity. The graphical nature of LabVIEW provides another virtue: it allows developers to easily visualize the flow of data in their designs. NI calls this Graphical System Design. Also, since LabVIEW is a mature data flow programming language, it has a wealth of existing documentation, toolkits, and examples which can be leveraged in development."
>
> Source: [1]

In Fig. 2.1 the start window of LabVIEW NXG 4.0.0 is shown. The user interface provides easy access to the projects created by the user or the available tutorials. By clicking on the *Projects* tab, recently accessed projects can be displayed or templates related to desired application for a new project can be created. Since LabVIEW is a visual programming language, programming is a bit different than textual programming languages. The *Learning* tab leads to explanations of the LabVIEW programming concepts as well as application oriented examples.

In order to create a new project for your tasks, *VI Project* should be clicked, which is shown marked in the Fig. 2.1.

This introduction mainly aims to save your time by showing you some details of LabVIEW that would otherwise take a lot of time to realize on your own. However, since the LabVIEW software has its own tutorials and exercises at its disposal, a detailed tutorial will not be provided here. The Pre-Lab Task part of this experiment, will guide you to the related tutorials as an exercise. Most of the programming tutorials can be found on *Learning* → *Lessons* → *Programming Basics*. In Fig. 2.2, the folder path to tutorials is marked.

2.1 Programming in LabVIEW

A more detailed tutorial than the one presented in this section is available in *Learning* → *Lessons* → *Getting Started*. After reading this section, please complete all available tutorials there.

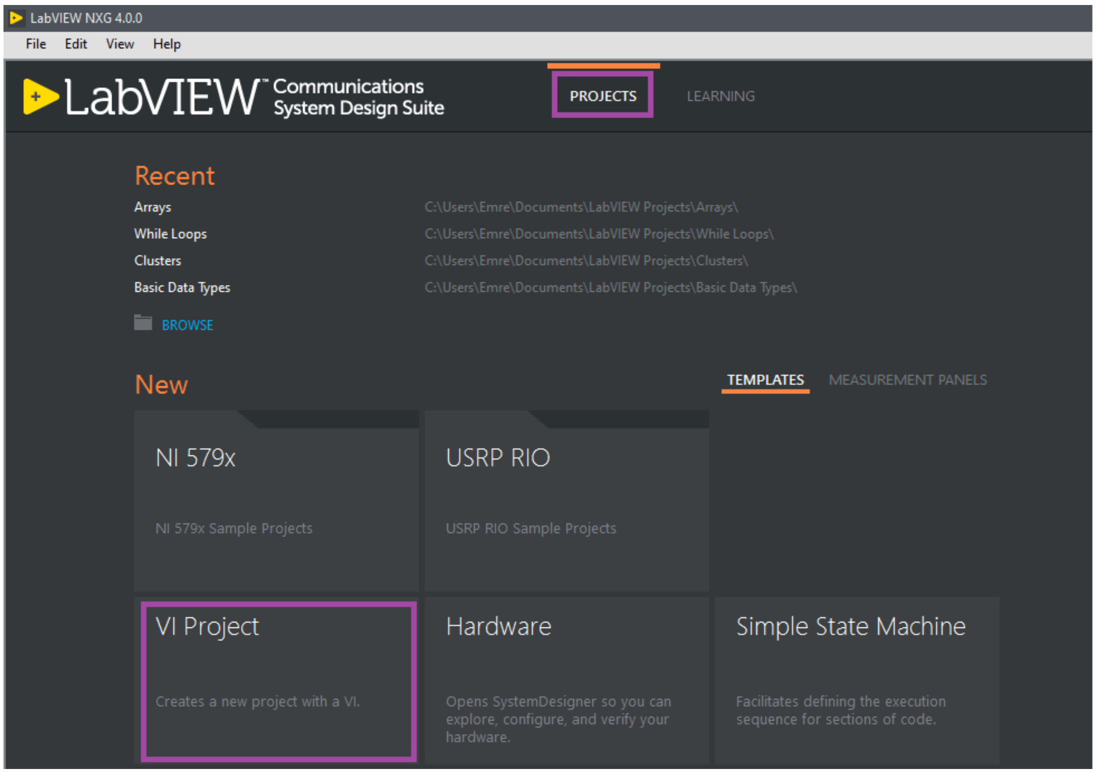

LabVIEW NXG 4.0.0
File
Edit
View
Help
LabVIEW Communications System Design Suite
PROJECTS
LEARNING
Recent
Arrays
While Loops
Clusters
Basic Data Types
C:\Users\Emre\Documents\LabVIEW Projects\Arrays\
C:\Users\Emre\Documents\LabVIEW Projects\While Loops\
C:\Users\Emre\Documents\LabVIEW Projects\Clusters\
C:\Users\Emre\Documents\LabVIEW Projects\Basic Data Types\
BROWSE
New
TEMPLATES
MEASUREMENT PANELS
NI 579x
NI 579x Sample Projects
USRP RIO
USRP RIO Sample Projects
VI Project
Creates a new project with a VI.
Hardware
Opens SystemDesigner so you can explore, configure, and verify your hardware.
Simple State Machine
Facilitates defining the execution sequence for sections of code.

Figure 2.1: Start window of LabVIEW NXG 4.0.0.

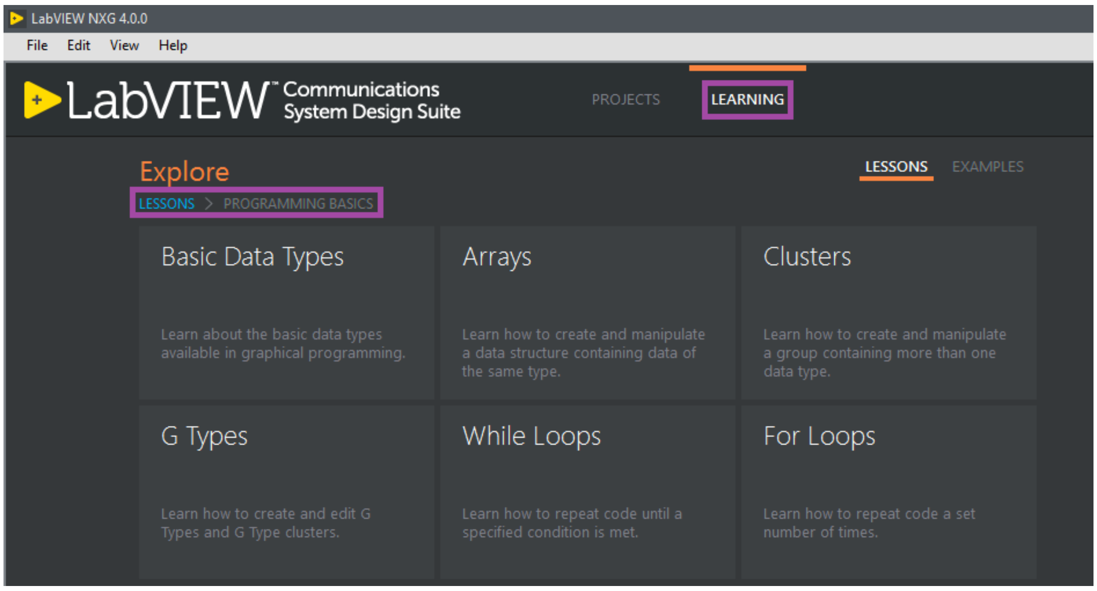

LabVIEW NXG 4.0.0
File
Edit
View
Help
LabVIEW Communications System Design Suite
PROJECTS
LEARNING
Explore
LESSONS > PROGRAMMING BASICS
LESSONS
EXAMPLES
Basic Data Types
Learn about the basic data types available in graphical programming.
Arrays
Learn how to create and manipulate a data structure containing data of the same type.
Clusters
Learn how to create and manipulate a group containing more than one data type.
G Types
Learn how to create and edit G Types and G Type clusters.
While Loops
Learn how to repeat code until a specified condition is met.
For Loops
Learn how to repeat code a set number of times.

Figure 2.2: The learning exercises provided by LabVIEW NXG 4.0.0.

2.1.1 Virtual Instruments (VIs)

LabVIEW program subroutines are termed virtual instruments (VIs). Mainly three components are used in each VI: a *front panel*, which enables you to design the user interface that displays inputs and outputs, a *block diagram*, where the graphical source code is programmed and a *connector pane* which enables you to design input output relationships of the VIs. The connector pane comes into use if you are going to employ the VI you design as a subVI that interacts with other blocks. The inputs and outputs displayed to the user are called *Control* and *Indicator*, respectively.

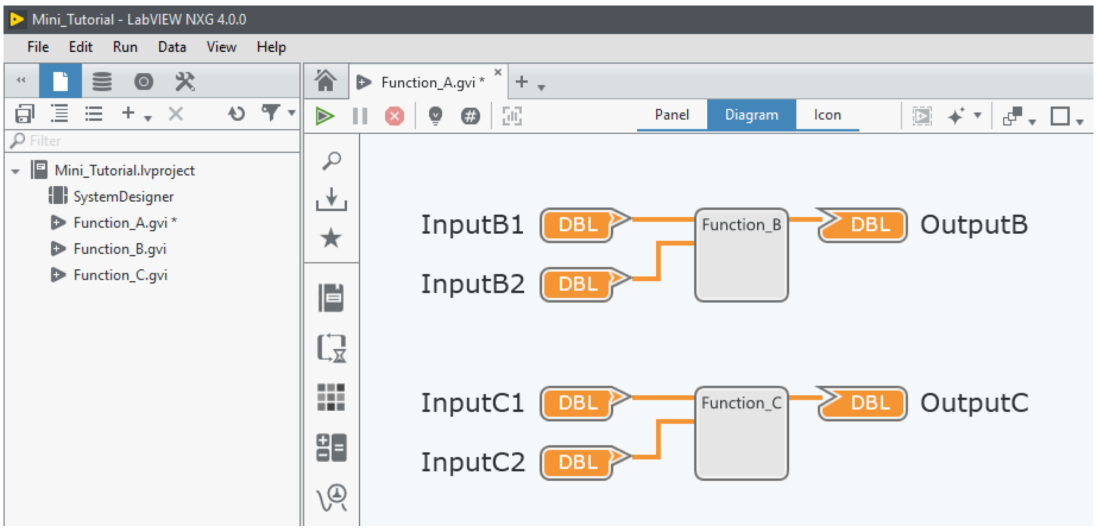
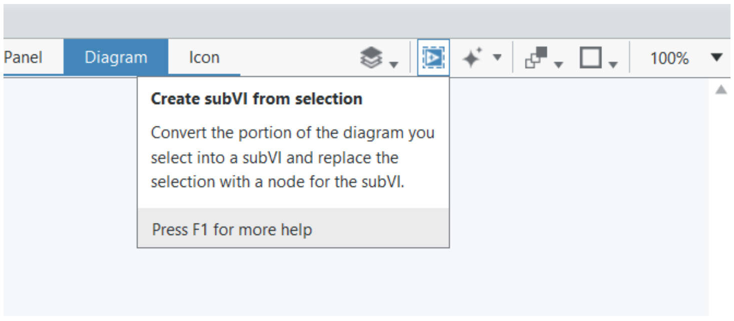


Figure 2.3: A VI that includes two subVIs.



Create subVI from selection

Convert the portion of the diagram you select into a subVI and replace the selection with a node for the subVI.

Press F1 for more help

Figure 2.4: Create subVI from selection button.

Any VI, which is used in the program flow of another VI, is referred as subVI. Thanks to the connector pane, each VI can be defined as a subVI by simply declaring inputs and outputs. In Fig. 2.3, three user defined VIs, namely *Function_A.gvi*, *Function_B.gvi*, and *Function_C.gvi*, can be seen. From the *Diagram*, it can be observed that *Function_B.gvi* and *Function_C.gvi* are being used as subVIs in *Function_A.gvi*.

For the assignments, you will be asked to create your own subVIs. There are two ways to create your subVI. The first method is to simply create another VI from scratch and start using it inside a VI. The second way is to create a subVI from existing code. In order to convert an existing code into a subVI, select a desired section of the code by dragging a frame with the mouse cursor and then declare it as a subVI with the help of *Create subVI from selection* button which is shown in Fig. 2.4.

2.1.2 Saving Graphs and Capturing Data

In order to submit a good protocol, you should use graphs and data you saved during your lab session. Therefore, this section emphasizes the options you have internally with LabVIEW.

In order to save a graph from your *Panel*, make a right click on the graph you would like to save with your mouse, then select *Capture image* → *Save image to file*. Afterwards, you can save your graphs in the desired format.

Sometimes you may want to include your interface from the *Panel* or your source code *Diagram* as a whole. To do this, open the *Panel* or *Diagram* of the VI you would like to capture. Click on *File* → *Print <Desired VI>*. Before applying these steps, make sure the part you would like to capture fits into a page.

If you would like to save every variable in your *Panel*, click on *Capture panel data*. If you would like to save only a specific data, make a right click on any *Indicator*, including graphs. Then click on *Capture Data*.

Every capture you have done is indexed under *Captured Data* tab. You can rename them to use later. In order to process the data with other programs or plot in different programs, a data export option is available.

2.2 Pre-Lab

2.2.1 Getting Familiar With the Basics of LabVIEW

For people who do not have any experience in LabVIEW programming, it is recommended to complete the following tutorials provided by LabVIEW NXG 4.0.0 before coming to the laboratory:

1. Basic Data Types
2. Arrays
3. For Loops
4. While Loops

The tutorials can be found following the points *Learning* → *Lessons* → *Programming Basics* in the start screen of LabVIEW.

2.3 Lab Experiment $\boxed{3}$

2.3.1 Programming a Random Bit Generator

As an assignment, please create a project that can generate random bits. In order to guide you, important steps in your task are stated below:

1. Create a new project, named `Random_Bit_Generator.lvproject`.
2. In your project, you should first implement a subVI, named `Random_Number_Generator.gvi`, which generates real random numbers. This subVI takes no inputs and produces a uniformly distributed random number between 0 and 1 as output. Table 2.1 provides the data types and a description of the input and output variables. There are several ways to generate random numbers in LabVIEW, some of which have already been shown in the tutorials.
3. Create a new VI, named `Main.gvi`. This VI will contain the rest of the code and call the subVI `Random_Number_Generator.gvi`.
4. Round the output of `Random_Number_Generator.gvi` to the nearest integer with the *Round to Nearest* block, which is available in LabVIEW.
5. Place your code inside a for loop. The number of iterations will be equal to the number of bits in the output bit sequence. Now, store the outputs in an array, the size of which is determined by the iteration number of the loop.
6. Drop your input (Number of Bits) and output (Generated Bits) variables to *Panel* and run your project.

Table 2.1: Description of the subVI `Random_Number_Generator.gvi`

Inputs	*Type of Variable*	*Definition*
-	-	-
Outputs	*Type of Variable*	*Definition*
Random number	DBL	A random number sampled from uniform distribution in range of [0,1]

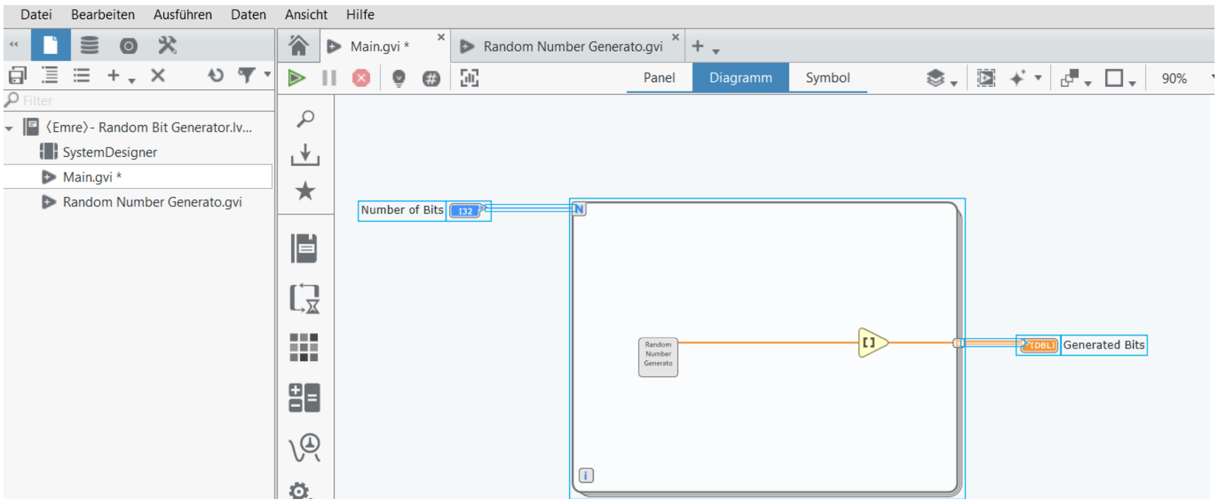


Figure 2.5: Resulting diagram of `main.gvi`.

An example implementation is shown in Fig. 2.5, and the output when the *Number of Bits* is selected as 6 is shown in Fig. 2.6.

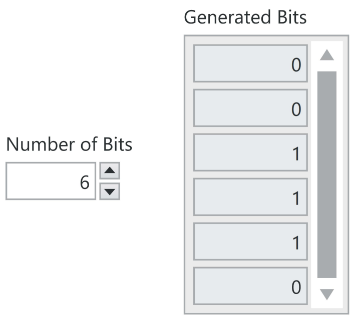


Figure 2.6: Expected output of `Random_Bit_Generator.lvproject`.

2.3.2 Extending Your Random Number Generator

Suppose the random number generator implemented above is a message source $\mathcal{X}$, putting out a string x^N of N bits, i.e. $x^N = \{x_1, x_2, \dots, x_N\}$ with $x_k \in \{0,1\}$, $k = 1, \dots, N$. Now suppose, we want to do a statistical analysis of our source. To this end, create a new subVI called `Compute_Params.gvi`. that computes the following quantities:

- Empirical mean: $m_X = \frac{1}{N}\sum_{k=1}^{N} x_k$
- Empirical variance: $s_X^2 = \frac{1}{N-1}\sum_{k=1}^{N}(x_k - m_X)^2$
- Symbol frequencies: $h_N(l) = \frac{1}{N}|C(l, x^N)|$ for $l = 0, 1$, where $C(l, x^N)$ is the function that counts the number of occurrences of l in the bit string x^N.

Please use only basic blocks (sum, difference, division, square, etc.) and not the available blocks for computing the empirical mean and the empirical variance.

Now run the program `Main.gvi` for $N \in \{10, 10^3, 10^5, 10^7\}$ and observe the output of the subVI `Compute_Params.gvi`. What do you observe? What happens for large N? Did you expect this result?

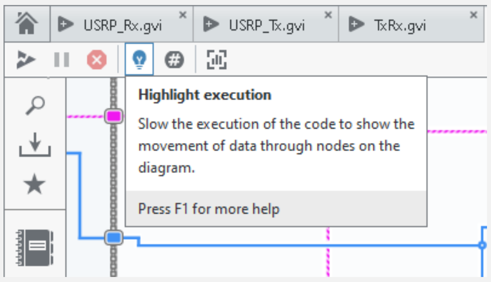
Table 2.2: Description of `Compute_Params.gvi`

Inputs	*Type of Variable*	*Definition*
Bit string	DBL array	Output array of the random bit generator
Outputs	*Type of Variable*	*Definition*
Empirical mean	DBL	see equation in text
Empirical variance	DBL	see equation in text
Symbol frequency	DBL	see equation in text

Tips and Tricks for Debugging

1. A very useful tool for debugging is the “**`probe`**” functionality. If you right click on any wire you can select "add probe" in the context menu. If you do this the probe appears in the debug section on the left hand side of the window. Now each time you run the code, you can see the values on this wire.

2. Using the “**`highlight execution`**” button (light bulb, thee icons right of the "run" button), you can see the timing of the information flow in the diagram. This can be useful for locating errors in the code.



Highlight execution

Slow the execution of the code to show the movement of data through nodes on the diagram.

Press F1 for more help

When designing a program in LabVIEW, there are important things to consider due to visual nature of the language. Even when a VI has the desired functionality and the variables are named properly, a bad written code is mostly hard to follow. Such a code is called *spaghetti code*.
Similar to other programming languages, it is good practice to write modular code, by using subVIs whenever possible. This also increases the reusability of the code.

3 Introduction to the USRPs (Day 1)

3.1 Pre-Lab $\boxed{3}$

3.1.1 Equivalent Lowpass Signals

First, make sure that you are familiar with the concept of the *equivalent lowpass signal.* You can find some of the theoretical background in Appendix A.2. Please read this section.

Roughly speaking, the equivalent lowpass signal x_{LP} of a real bandpass signal x_{BP} is a different representation of x_{BP}. Assume that we have a real bandpass signal x_{BP}, the spectrum of which is concentrated in the passband $[-f_{\mathrm{c}} - B/2, -f_{\mathrm{c}} + B/2] \cup [f_{\mathrm{c}} - B/2, f_{\mathrm{c}} + B/2]$, where f_{c} denotes the center frequency and B the effective bandwidth. Then it is possible to define an equivalent lowpass signal x_{LP} (which is complex in general) that has exactly the same "information content" as the bandpass signal x_{BP} (for an exact definition, see Section A.2). Equally, for a given lowpass signal x_{LP} and center frequency f_{c} there is a uniquely defined real bandpass signal x_{BP}.

In a wireless communication system, the real bandpass signal x_{BP} is transmitted by the antenna. The equivalent lowpass signal x_{LP} is often called *baseband signal* or I/Q signal, where the real part $x_{\mathrm{LP,R}}$ is called in-phase component and the imaginary part $x_{\mathrm{LP,I}}$ quadrature-phase component.

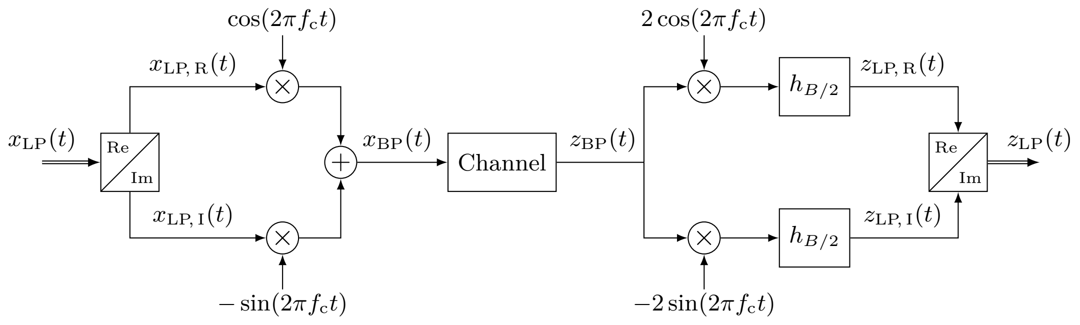


Figure 3.1: Transmission of an I/Q signal.

When using software defined radios, we provide the samples of the complex baseband signal $x_{\mathrm{LP}}[n]$ together with an I/Q rate $r_{\mathrm{IQ}}^{\mathrm{Tx}}$. Then the digital signal is converted into an analog signal according to

$$x_{\mathrm{LP}}(t) = \sum_{n=-\infty}^{\infty} x_{\mathrm{LP}}[n] \operatorname{sinc}(r_{\mathrm{IQ}}^{\mathrm{Tx}} t - n),$$

where

$$\operatorname{sinc}(t) = \frac{\sin(\pi t)}{\pi t}.$$

Note that we have $x_{\mathrm{LP}}[n] = x_{\mathrm{LP}}(n/r_{\mathrm{IQ}}^{\mathrm{Tx}})$. Afterwards, this analog complex baseband signal $x_{\mathrm{LP}}(t)$ is converted into the real bandpass signal $x_{\mathrm{BP}}(t)$, which is transmitted over the channel. At the receiver, the received signal $z_{\mathrm{BP}}(t)$ is first converted into a complex baseband signal $z_{\mathrm{LP}}(t)$. Then, $z_{\mathrm{LP}}(t)$ is sampled at the I/Q rate of the receiver $r_{\mathrm{IQ}}^{\mathrm{Rx}}$, which gives the samples of the complex baseband signal $z_{\mathrm{LP}}[n] = z_{\mathrm{LP}}(n/r_{\mathrm{IQ}}^{\mathrm{Rx}})$. This procedure is illustrated in Fig. 3.1. Fig. 3.1 also shows a way how the bandpass signal $x_{\mathrm{BP}}(t)$ can be generated from the baseband signal $x_{\mathrm{LP}}(t)$, and vice versa. This is process is close to what is done in the USRPs. For a detailed description how the USRPs work, please have a look at Fig. 3.2.

	Transmitter	**Receiver**
Frequency range	70 MHz to 6 GHz	70 MHz to 6 GHz
Frequency step	< 1 kHz	< 1 kHz
Maximum output power	20 dBm	–
Maximum input power	–	−15 dBm
Gain range	−40 to 49.75 dB	−15 to 61 dB
Gain step	0.25 dB	1 dB
Noise figure	–	5 to 7 dB
Frequency accuracy	2.5 ppm	2.5 ppm
Max. instantaneous real-time bandwidth	56 MHz	56 MHz
Maximum I/Q rate (streaming)	15 MS/s	15 MS/s
Digital-to-analog converter (DAC)	12 bits	–
Analog-to-digital converter (ADC)	–	12 bits

Table 3.1: Specifications of the USRP-2901.

3.1.2 Hardware Description

USRP-2901

In the lab we use the USRP-2901 tuneable RF transceiver from National Instruments, operating from 70 MHz to 6 GHz. The maximum instantaneous real-time bandwidth is 56 MHz, and the maximum I/Q sample rate 15 MS/s. The USRP-2901 has a maximum output power of 100 mW (20 dBm). The USRP is connected via USB 3.0. Further specifications of the USRP-2901 device are summarized in Table 3.1.

Note that the output power is not constant over the frequency range, i.e. when transmitting at different frequencies the same setting might lead to a different actual transmit power at the antenna. Further, the maximum output power may also vary from device to device.

In Fig. 3.2 you can see a block diagram that illustrates the functionality of the USRP-2901 devices. The Rx channel of the USRP works as follows. The signal that is fed into the SMA antenna port of the USRP is first amplified and then downconverted by the USRP to the baseband. The downconversion produces two real signal signals (I and Q components), which subsequently are quantized and sampled and then sent forward to the FPGA. Using the FPGA, the digitized signals are downsampled and sent to the computer via USB connection. The Tx channel of the USRP works as follows. The data is received via the USB connection, then after a digital upsampling it is converted into an analog signal. This analog signal is upconverted from the baseband to the desired frequency band, amplified and provided at the SMA antenna port.

The USRP-2901 has two local voltage-controlled oscillators (VCO), one for the transmit (Tx) channels and one for the receive (Rx) channels. While different frequencies can be used for the Tx and Rx channels, the two Tx channels are locked to the same frequency, and the two Rx channels are locked to the same frequency.

Antenna

The antenna is an omni-directional dual band rod antenna for 2.4 GHz to 2.48 GHz and 4.9 GHz to 5.9 GHz, at 3 dBi gain.

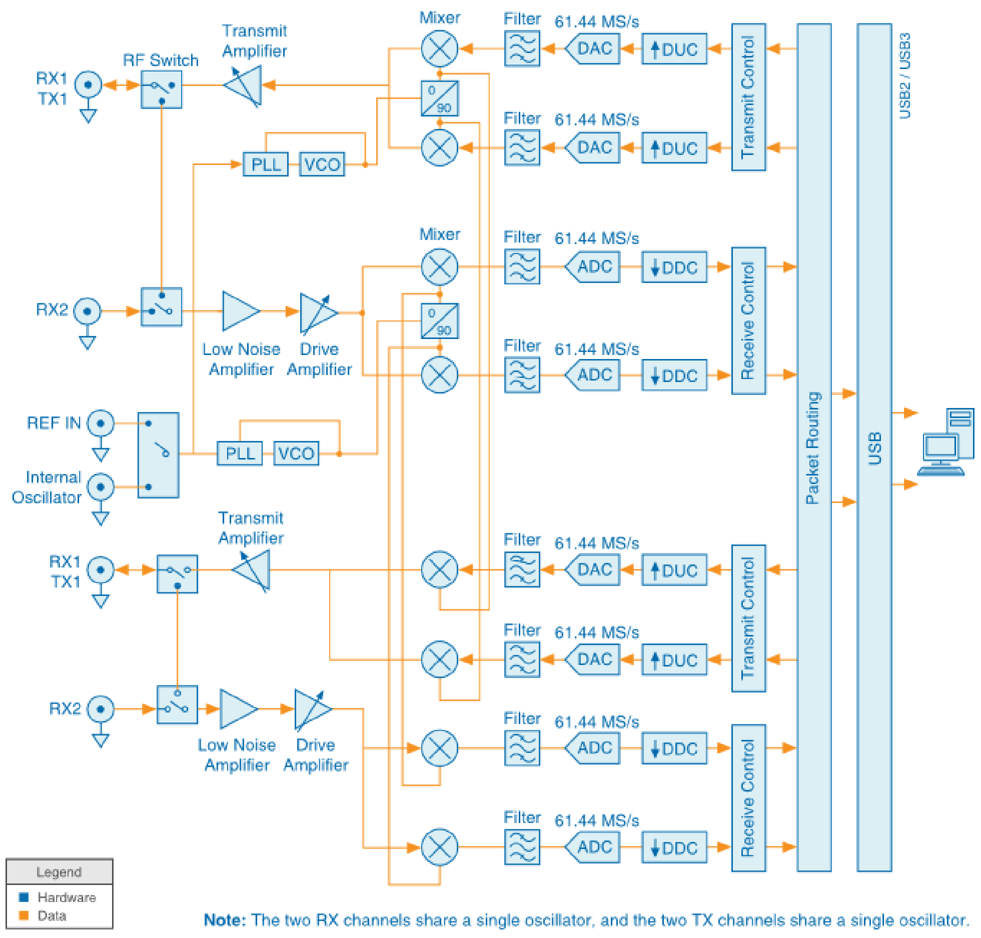


Figure 3.2: Block diagram of the USRP-2901 (Source: National Instruments).

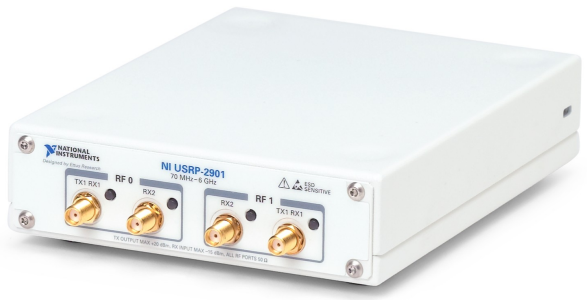


Figure 3.3: Photo of the USRP-2901 (Source: National Instruments).

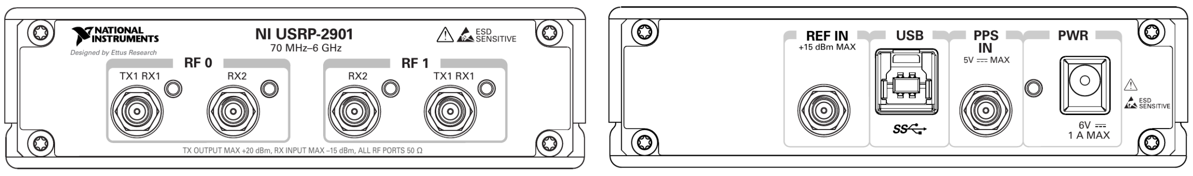


Figure 3.4: Front panel (left) and back panel (right) of the USRP-2901.

3.1.3 Pre-Lab Tasks

1. The USRPs are connected using USB 3.0. What is the maximum data rate (in bit/s) that is supported by this connection? How does this compare to the rate (in bit/s) that is generated when sampling **one** RF channel (which consists of an I and a Q data stream) with the maximum I/Q rate (streaming)? In this scenario, is the data rate of USB 3.0 sufficient to transmit all the data generated by the USRP to the computer? Is this also true if both RF channels are used simultaneously?

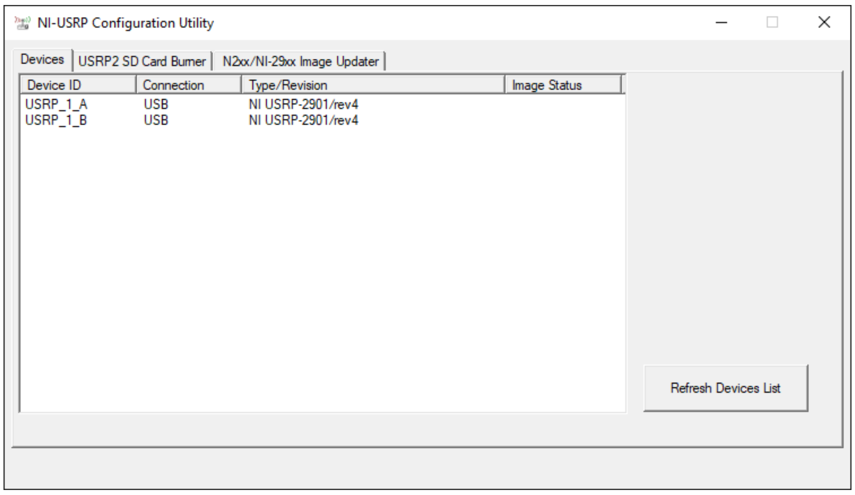
In the lab you will use the parameters for the transmitter that are listed in Tab. 3.2 and the digital baseband signal

$$x_{\mathrm{LP}}[n] = \begin{cases} 1, & n = 1, \dots, 10000, \\ 0, & \text{otherwise.} \end{cases}$$

2. How is the analog baseband signal $x_{\mathrm{LP}}(t)$ that corresponds to this digital baseband signal $x_{\mathrm{LP}}[n]$ computed? Write down the analytical expression of $x_{\mathrm{LP}}(t)$. Insert as many known parameters as possible and simplify the expression as much as possible.
3. What is the "effective" time duration (in seconds) of the signal $x_{\mathrm{LP}}(t)$.
4. Plot $x_{\mathrm{LP}}(t)$ in the range from $-5 \times 10^{-6}\,\mathrm{s}$ to $1.5 \times 10^{-5}\,\mathrm{s}$.
5. Derive the analytical expression of $x_{\mathrm{BP}}(t)$. Insert as many known parameters as possible and simplify the expression as much as possible.
6. What is the period duration of the carrier wave, for the carrier frequency that given in Tab. 3.2?
7. Plot two different regions of the signal $x_{\mathrm{BP}}(t)$. a) Plot $x_{\mathrm{BP}}(t)$ in the range from $0\,\mathrm{s}$ to $5 \times 10^{-9}\,\mathrm{s}$ and b) Plot $x_{\mathrm{BP}}(t)$ in the range from $5 \times 10^{-3}\,\mathrm{s}$ to $5.000\,005 \times 10^{-3}\,\mathrm{s}$.
8. Considering the "effective" time duration of $x_{\mathrm{LP}}(t)$, which parts of $x_{\mathrm{BP}}(t)$ are plotted in a) and b) respectively?
9. How many periods of the carrier wave are visible in each of the plots?

In our experiment we deliberately choose a slightly different carrier frequency at the receiver (see Tab. 3.3).

10. Derive the analytical expression of $z_{\mathrm{LP}}(t)$. For simplicity we assume that there is no channel, i.e., that $z_{\mathrm{BP}}(t) = x_{\mathrm{BP}}(t)$. (This derivation is a little bit longer. Equations (C.2) and (C.3) might be useful.)
11. In the lab you will be provided with a VI that displays the power spectral density of $z_{\mathrm{LP}}(t)$. What would you expect to see?



Device ID	Connection	Type/Revision	Image Status
USRP_1_A	USB	NI USRP-2901/rev4	
USRP_1_B	USB	NI USRP-2901/rev4	

Figure 3.5: NI-USRP Configuration Utility.

3.2 Lab Experiment $\boxed{2}$

In this experiment you will learn how to initialize the URSPs and how to transmit a self-generated signal. The VI for the receiver that you can use to check if your transmitter works correctly, will be provided for this experiment. In the next experiment (Section 4) you will implement a receiver VI by yourself.

3.2.1 Checking the Hardware

First, please check if your equipment is complete. Each group should have:

- 2 USRP-2901 devices
- 2 USB cables
- 1 30 dB attenuator
- 1 SMA(male) to SMA(male) cable
- 2 antennas

3.2.2 Connecting the USRPs

At the beginning of this experiment you will connect the two USRPs to your computer using the USB connector at the rear of the device (see Fig. 3.4). The LED at the rear which is labeled with “PWR” should be blue.

Next you will check if the USRPs are correctly detected by the computer. To this end, start the *NI-USRP Configuration Utility*. There you should see two USRP devices, similar to Fig. 3.5. The USRPs have the naming scheme `USRP_<groupID>_<deviceID>`, where `groupID` is your group ID (number from 1 to 4), and `deviceID` is either A or B. Sometimes it takes Windows several minutes to detect the devices, so please be patient.

3.2.3 Connecting the RF-Cable

For the first experiment we will connect the two USRPs directly using a cable and an 30 dB attenuator. This attenuator is VERY IMPORTANT, since otherwise the radio front-end of the receiving USRP will be destroyed due to the high signal power. Please connect the `RF 0, TX1/RX1` port of device A with the `RF 0, RX2` port of device B (do not forget the attenuator).

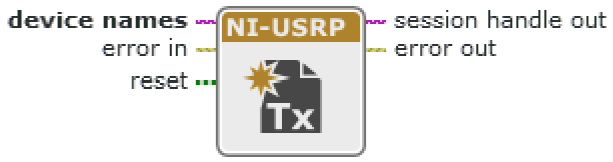
> When attaching an antenna or an SMA connector to one of the ports of the USRP, it is important that you do not turn the antenna/cable itself, but only the connector. Otherwise the SMA port at the USRP-2901 device might be damaged. Your instructor will demonstrate how it is done correctly.

3.2.4 Blocks Used for Initializing and Communicating with the USRPs

In order to implement the transmitter VI you will need the following blocks that are provided by LabVIEW.

niUSRP Open Tx Session

The block `niUSRP Open Tx Session` is used to open a transmit (Tx) session to the device you specify. It returns the instrument session, which you use in all subsequent NI-USRP nodes.

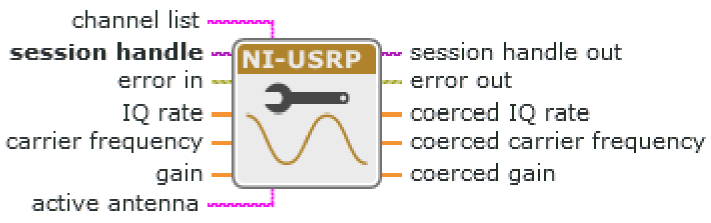


Inputs:
`device name`: Name of the device
(in your case the device name is either `USRP_<groupID>_A` or `USRP_<groupID>_B`).
`error in`: Error conditions that occur before this node runs.
`reset`: Boolean that specifies whether to reset the device to a known initialization state.

Outputs:
`session handle out`: Reference to your instrument session to be passed to the next node.
`error out`: Error information.

niUSRP Configure Signal

The block `niUSRP Configure Signal` configures properties of the transmit (Tx) or receive (Rx) signal. Note, that the USRP devices are not calibrated. The gain value does not represent an absolute gain and does not have linear behavior. Different devices exhibit different gain curves for different carrier frequencies. You may need to experiment to determine the correct gain setting for your application.

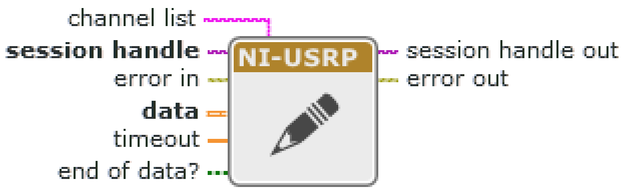


Inputs:
`channel list`: Channels to configure.
`session handle`: Instrument session.
`error in`: Error conditions that occur before this node runs.
`IQ rate`: Rate of the baseband I/Q data (in samples per second S/s).
`carrier frequency`: Carrier frequency (in Hz) of the RF signal.
`gain`: Aggregate gain (in dB) applied to the RF signal.
`active antenna`: Antenna port to use for this channel. Can be either `TX1`, `RX1`, or `RX2`.

Outputs:
`session handle out`: Reference to your instrument session to be passed to the next node.
`error out`: Error information.
`coerced IQ rate`: Actual I/Q rate (in samples per second S/s) for this session, coerced to a value supported by the device.
`coerced carrier frequency`: Actual carrier frequency (in Hz) for this session, coerced to a value supported by the device.
`coerced gain`: Actual gain (in dB) for this session, coerced to a value supported by the device.

niUSRP Write Tx Data

The block `niUSRP Write Tx Data` writes complex, double-precision floating-point data to the specified channels.



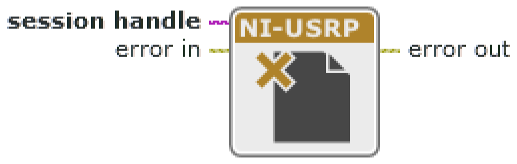
Inputs:
`channel list`: Channel(s) to which to write the data.
`session handle`: Instrument session.
`error in`: Error conditions that occur before this node runs.
`data`: Baseband samples to transmit as an array of complex, double-precision floating-point data. The real and imaginary components of the data correspond to the in-phase (I) and quadrature-phase (Q) data, respectively. Each row of the 2D array corresponds to a separate channel.
The time between samples in the waveform (the sample period) equals 1 divided by the coerced I/Q rate. Determine the coerced I/Q rate by reading the IQ Rate property after you set it or by reading the *coerced IQ rate* output of Configure Signal.
This input accepts complex, double-precision floating-point values whose real and imaginary components range from 1.0 to -1.0. Maintain the maximum complex magnitude to a value less than 1.0 to prevent DSP overflow. Because the DSP frequency response varies over frequency and over I/Q rates, some tones with a complex magnitude less than but close to 1 may cause DSP overflow. Consider reducing the amplitude if you observe unexpected spurs in the spectrum of your generated signal.
`timeout`: Time to wait, in seconds, before returning an error if the requested number of samples have not been generated.
`end of data`: Boolean that determines whether this is the last call to niUSRP Write Tx Data for the current contiguous transmit operation.

Outputs:
`session handle out`: Reference to your instrument session to be passed to the next node.
`error out`: Error information.

Important: The block `niUSRP Write Tx Data` also needs to be configured correctly. In the `Function Configuration` of the block, set “`Single Channel`” and “`Complex Double`”.

niUSRP Close Session

The block `niUSRP Close Session` closes the session handle to the device.



Inputs:
`session handle`: Instrument session.
`error in`: Error conditions that occur before this node runs.

Outputs:
`error out`: Error information.

3.2.5 Programming a VI for Transmitting

In order to transmit data with a URSP the basic steps are:

1. open a Tx Session (`niUSRP Open Tx Session`),
2. configure the USRP (`niUSRP Configure Signal`),
3. write the data to the USRP (`niUSRP Write Tx Data`),
4. close the Session (`niUSRP Close Session`).

Note that steps 1,3, and 4 are similar to writing to a file in a general purpose programming language.

In this experiment you will transmit the same complex I/Q data $x_{\text{LP}}[n]$ that you used in the pre-lab task. That is, you will transmit a sequence of 10000 “ones”. Further, use a loop such that the transmission is repeated until you press the stop button. The parameters for the transmitter are given in Tab. 3.2. You can implement your transmitter using the same structure as given in Fig. 3.6. Some remarks:

- We use a while loop to produce a continuous transmission of our data.
- The while loop is ended if the stop button is pressed.

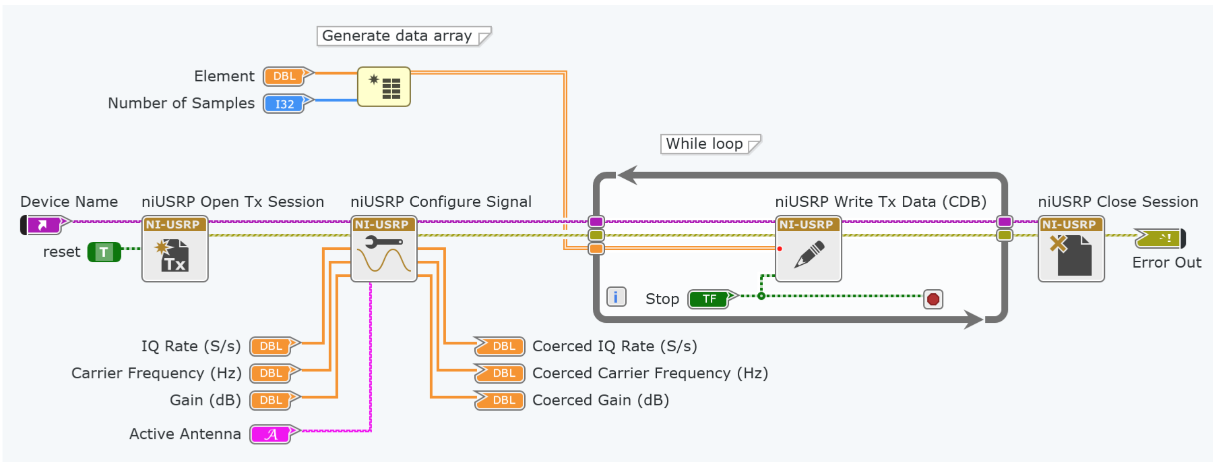


Figure 3.6: Transmit VI.

Parameter	Value
Device name	`USRP_<groupID>_A`
Active antenna	`RF 0, TX1`
Carrier frequency f_c^{tx}	2 GHz
I/Q rate	1 MS/s
Gain	10 dB
Dimension size	10 000 S

Table 3.2: Parameters for the transmitter.

3.2.6 Programming a VI for Receiving

Parameter	Value
Device name	`USRP_<groupID>_B`
Active antenna	`RF 0, RX2`
Carrier frequency f_c^{rx}	2.0002 GHz
I/Q rate	1 MS/s
Gain	1 dB
Number of samples	10 000 S

Table 3.3: Parameters for the receiver.

1. Open the the receiver project `receiver_psd`.
2. Check if the parameters are set correctly, as given in Tab. 3.3.
3. Start the receiver VI.
4. Start your transmitter VI.

> You should always stop a running program with the implemented "Stop" button on the panel. Do not use the red "x" button in the LabVIEW software, because pushing this button will immediately stop the program. This may result in open file handles and undefined buffer stated, because no cleanup operations are performed before stopping the program.

4 Spectrum Analyzer and FM Receiver (Day 1)

The purpose of this lab experiment is to demonstrate the capabilities of USRP using LabVIEW Software and give foresight into what can be achieved using the presently available SDR platform. To this end, a FM demodulator will be developed. Also in this lab, the water spectrum analyzer will be developed to represent the power spectrum of a signal over time.

4.1 Lab Experiment $\boxed{2}$

In this lab experiment you will use a digital receiver which demodulates a frequency modulated signal to the baseband. This receiver can be used to listen to a FM radio station. First, you will implement a waterfall spectrum analyzer that is used to visualize the spectrum of the received signal. The waterfall spectrum analyzer will be used as a subVI in the FM Demodulator. Second, you will run the code to hear radio.

4.1.1 LabVIEW Blocks

The following LabVIEW blocks are important when interacting with the USRPs. You have already encountered some of them in Section 3.

Open Tx Session: This block opens a transmit (Tx) session to the device specified in the device names input and returns session handle as output. IP address or name of the USRP is an input to this block indicating which USRP needs to be activated.

Configure Signal: This block configures Tx or Rx parameters. Sampling rate of the baseband I/Q data, carrier frequency, gain are the input of this block. Active antenna is the antenna port which is being used. The outputs coerced IQ rate, carrier frequency and gain are the actual corresponding values supported by the device.

Write Tx data: This block is responsible to writes complex signed integer data to the specified channel. The real and imaginary components of the data correspond to the in-phase (I) and quadrature-phase (Q) data, respectively.

Close Session: Closes the session handle to the device.

Open Rx Session: This block opens a receive (Rx) session to the device specified in the device name. IP address or name of the USRP is an input to this block indicating which USRP needs to be activated.

Initiate: Starts the Rx acquisition.

Fetch Rx Data: This block fetches complex signed integer data from the specified channel. Number of samples defined the samples to fetch from the acquisition channel.The real and imaginary components of the data correspond to the in-phase (I) and quadrature-phase (Q) data, respectively.

Abort: Stops an acquisition previously started.

4.1.2 Waterfall Spectrum Analyzer

A waterfall spectrum is a representation of the power spectral density of a signal over time. This gives a visual understanding of the frequency content of your signal. For the FM receiver, this spectrum analyzer helps to recognize the radio stations at different frequencies. By looking at the strength of the signal on the spectrum one can easily tune the FM receiver to that particular frequency to listen to the station.

Open the subVI `Waterfall_Spectrum.gvi`. Inside this subVI add the *Power Spectrum* block and input the signal received. Unpack the cluster of the received signal to get the magnitude of the received signal. To show the spectrum of the signal, we need a buffer that can store some values and code to display the data as a plot. Follow the snippet given below to construct a buffer that stores 120 samples and plots the spectrum.

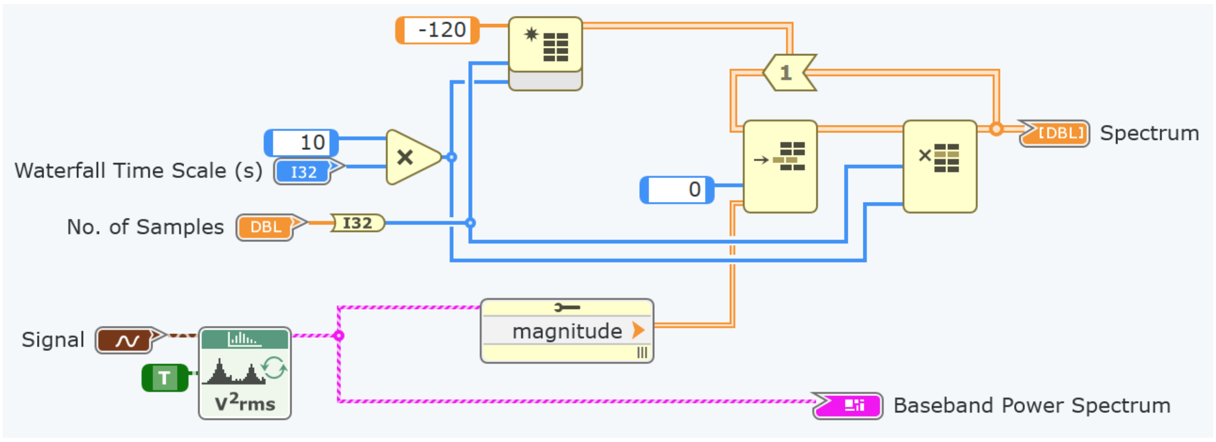


Figure 4.1: Waterfall spectrum analyzer.

Hint: The subVI `Waterfall_Spectrum.gvi` already contains all necessary blocks. They are contained in a `Disable Structure`. The easiest way is to remove the `Disable Structure` and then connect the blocks. You can remove the `Disable Structure` by right-clicking on it and selecting `Remove Disable Structure` from the context menu. Do not cut and paste the blocks, because then the wires in the `FM Demodulator.gvi` VI will be disconnected from the subVI.

4.1.3 FM Demodulator

Once you have implemented the subVI `Waterfall_Spectrum.gvi`, you can run the main program `FM Demodulator.gvi`. The code is a modified version of the example given in the LabVIEW communication design suite. It is recommended that you use `USRP_<groupID>_A` and `RF0, RX2`. In this case you do not have to detach the cable that you have already installed, in order to install the antenna. The sampling rate is configured to be 200 kS/s. Each sample is a complex float. The system is set to tune to 104 MHz, the frequency of a local radio station. The gain is set to be 20 dB.

Figure 4.2 shows the diagram of the FM demodulator. Inside the while loop the receiver, processes the samples that are coming in. The received samples are passed to a FM demodulator block.

Try to tune to different radio stations. Can you identify the structure of an FM signal, as shown in Fig. 4.3, in the spectrum of the demodulated FM signal?

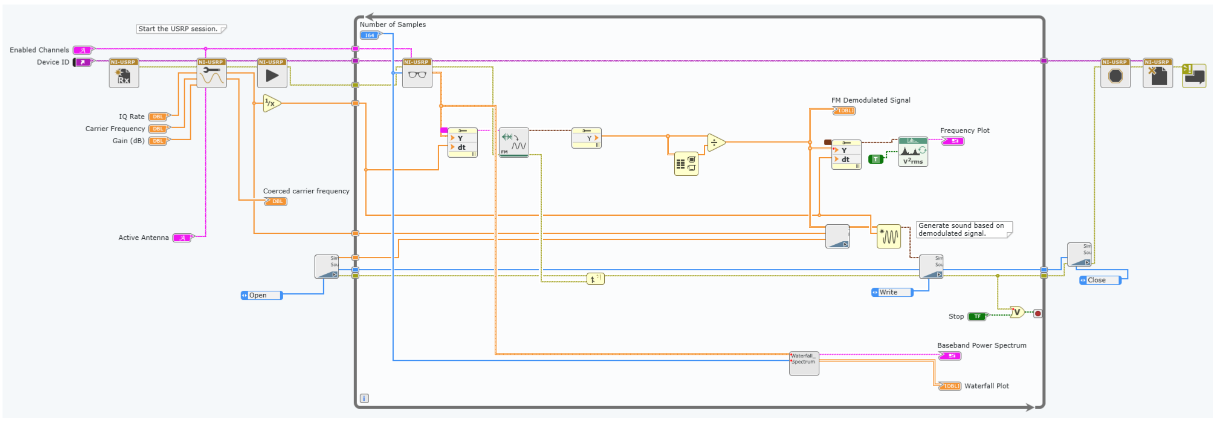


Figure 4.2: FM demodulator.

FM Radio

In Germany, stereo FM radio was established in the 60s. In order to maintain compatibility with the previously existing mono FM radio receivers, a stereo multiplex signal has been introduced, where a sum signal (left+right) and a difference signal (left−right) is transmitted. The left+right signal is limited to a frequency range form 30 Hz to 15 kHz. The left−right signal, which is also limited to a bandwidth of 15 kHz, is amplitude modulated, using a double-side-band modulation with a suppressed carrier at 38 kHz. Further, a 19 kHz pilot tone is transmitted at 8–10% of overall modulation level. The pilot tone is used by the receiver to identify a stereo transmission and to regenerate the 38 kHz carrier with the correct phase. The spectrum of such a stereo multiplex signal is illustrated in Fig. 4.3. In FM radio, the stereo multiplex signal is modulated using an analog frequency modulation. Compared modern digital transmission schemes, such as DAB, the spectrum use of FM radio is relatively inefficient.

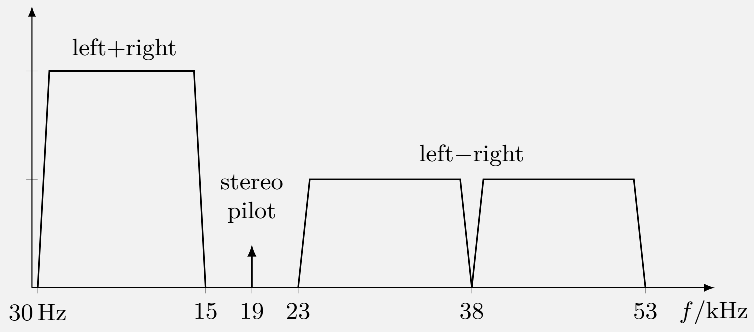


Figure 4.3: Spectrum of a stereo multiplex signal (before the frequency modulation).

5 Symbol Mapping (Day 2)

An essential part of a digital communication system is the process of converting information, represented by bits, into waveforms that can be transmitted over the wireless channel. Important design parameters how to perform this conversion are data rate, bandwidth, and desired reliability of the transmission. The key steps in the conversion of bits into waveforms are

1. symbol mapping
2. pulse shaping with D/A conversion, and
3. upconversion.

This process is illustrated in Fig. 5.1. In the symbol mapping step the bit sequence is transformed into a sequence of complex symbols. To this end, each group of ζ bits is mapped into one complex symbol. The exact number of ζ depends on the employed modulation scheme. In the pulse shaping step the symbol sequence is converted into a waveform. In modern digital communication the pulse shaping is implemented in the digital domain using a digital filter. The digital waveform is then converted in an analog waveform using a D/A converter. This analog waveform is the complex baseband signal $x_{\mathrm{LP}}(t)$. In the final step, the upconversion, the complex baseband signal $x_{\mathrm{LP}}(t)$ is converted into a real bandpass signal $x_{\mathrm{BP}}(t)$. Note that in our case the D/A conversion and the upconversion is done in the SDRs. In Chapter 3 you already learned about the differences between a passband signal and the equivalent lowpass representation of a bandpass signal. For simplicity of exposition, we treat the pulse shaping and D/A conversion as one operation and not as two separate operations.

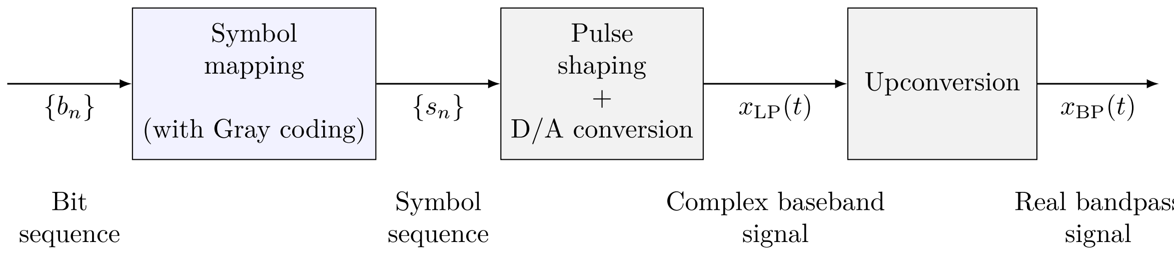


Figure 5.1: The steps in a passband transmission scheme to transmit a coded bit sequence.

The tasks in this experiment focus on the *symbol mapping* step, which is highlighted in Fig. 5.1. Amplitude shift keying (ASK), phase shift keying (PSK), frequency shift keying (FSK), and quadrature amplitude modulation (QAM) are some examples of digital modulation schemes. The next experiment focuses on PSK.

Digital modulation schemes differ by their spectral efficiency, power efficiency, and robustness to multi-path propagation, noise, and interference. While the spectral efficiency is a measure of how efficiently the modulation scheme utilizes the available frequency spectrum, power efficiency is a measure for the power that is needed to transfer a certain number of bits. Power efficiency is especially important for portable devices that are battery dependent [2, p. 48]. In this SDR Laboratory, we do not treat these aspects.

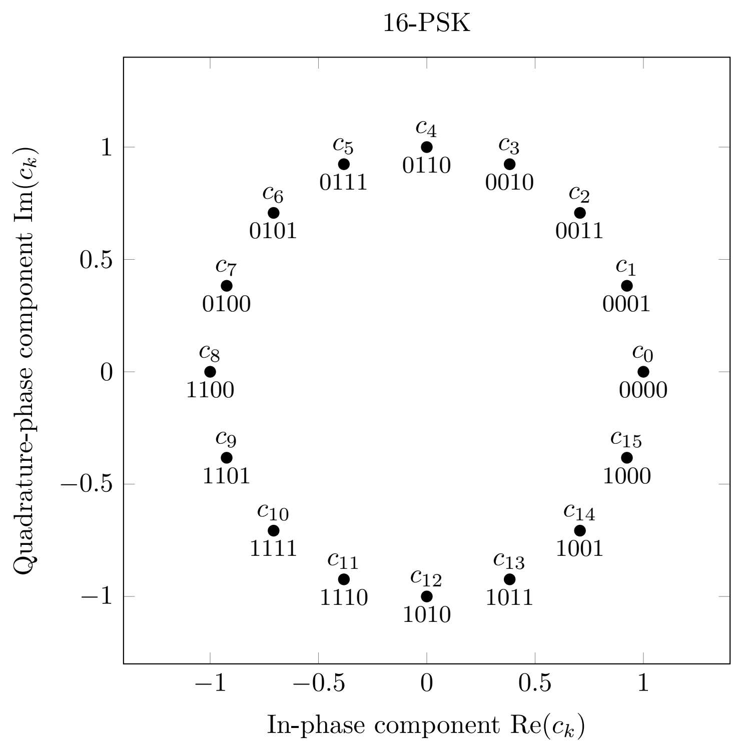


Figure 5.2: Constellation diagram for 16-PSK with Gray labeling.

5.1 Phase Shift Keying (PSK)

PSK transmits data by changing the phase of a constant frequency carrier wave. PSK is used in practice, e.g. in satellite broadcasting, some operation modes of the wireless LAN standard IEEE 802.11b and the Bluetooth standard (to be more precise, PSK is used in the form of differential PSK here).

The number of bits mapped to one symbol is denoted by ζ. The number of symbols that are available in a certain modulation scheme is called modulation order and denoted by $M = 2^\zeta$. Let $\mathcal{C} = \{c_0, c_1, \ldots, c_{M-1}\}$ denote the symbol alphabet, which is also called constellation. The constellation diagram of a 16-PSK modulation scheme is shown in Fig. 5.2.

In many applications, the average symbol energy is normalized to one in order to be able to compare the performance with other modulations scheme. The normalized symbols satisfy

$$\frac{1}{M}\sum_{k=0}^{M-1} |c_k|^2 = 1. \tag{5.1}$$

For PSK scheme, the energy of all symbols in the constellation is the same and the average energy of all symbols is normalized if $|c_k| = 1$, $k = 0, \ldots, M-1$.

5.2 Gray Coding

Similar to natural binary coding, each decimal number corresponds to a binary sequence in Gray coding. Although in the natural binary coding two adjacent values are likely to differ by more than one bit, in Gray coding two adjacent values differ by only one bit. In Table 5.1, Gray coding for 4 bits is given as an example.

Gray coding can be interpreted as an ordering of the binary numeral system such that two successive values differ in only one bit. This helps us to reduce the bit error rate in communication systems for the following reason. Distortions and noise during the transmission sometimes lead to erroneous detection at the receiver. However, due to the nature of the noise and the distortions, often the detected symbol is close to the transmitted symbol. Say for example, we receive c_4 while actually c_5 has been transmitted. The advantage of using a Gray code in the transmission scheme, now is that the erroneously received bit sequence differs from the actual transmitted bit sequence in only one bit.

Table 5.1: 4-Bit Gray coding

Decimal	Binary representation	Gray coded bits	Gray Decimal
0	0000	0000	0
1	0001	0001	1
2	0010	0011	3
3	0011	0010	2
4	0100	0110	6
5	0101	0111	7
6	0110	0101	5
7	0111	0100	4
8	1000	1100	12
9	1001	1101	13
10	1010	1111	15
11	1011	1110	14
12	1100	1010	10
13	1101	1011	11
14	1110	1001	9
15	1111	1000	8

The operation of the symbol mapping block shown in Fig. 5.1 can be conceptually separated into two mappings. The first mapping performs the Gray mapping and the second mapping does the actual symbol mapping. You will implement both mappings in the lab experiment.

5.3 Pre-Lab $\boxed{2}$

Your protocol should include the answers of the questions below.

1. Why Gray coding is used in communication systems?

2. While it is possible to calculate the theoretical symbol error probability P_s, it is not always possible to calculate a theoretical bit error probability P_b. In such cases if Gray coding is applied, the bit error probability can be approximated as $P_b \approx P_s/\zeta$. What might be the problem with calculating P_b and how does Gray coding help to solve this problem?

3. Assuming the bit sequence $\{0010110010101111\}$, what is the corresponding symbol sequence if we use 16-PSK and Gray labeling as given in Fig. 5.2?

4. What is the symbol alphabet (constellation) of 4-QAM? Assume that the symbol are normalized according to (5.1).

5.4 Lab Experiment $\boxed{3}$

Task A: The first goal of this experiment is to have a program `symbol_mapping.gvi` that converts a sequence of bits into a sequence of symbols while using Gray labeling. You will implement a M-PSK modulation, where the modulation order M is a parameter that can be set by the user via the variable ξ. Remind that we have have $M = 2^{\zeta}$. Clearly, the length of the input bit sequence has to be a multiple of ζ bits.

The inputs and outputs of the program `symbol_mapping.gvi` are listed in Table 5.2. The panel of a reference implementation of this VI can be seen in Fig. 5.3.

In the binary representation that we used in Table 5.1, the most significant bit is the leftmost bit in the bit string. The input bit string to your program should be interpreted in the same way, i.e., each chunk of ξ bits has its most significant bit left.

Table 5.2: Description of `symbol_mapping.gvi`

Inputs	*Type of Variable*	*Definition*
zeta	U32	Number of bits per symbol
bit_string	Boolean array	input bit string (most significant bit left)
Outputs	*Type of Variable*	*Definition*
symbol_sequence	CDB array	symbol sequence (M-PSK, Gray labeling)

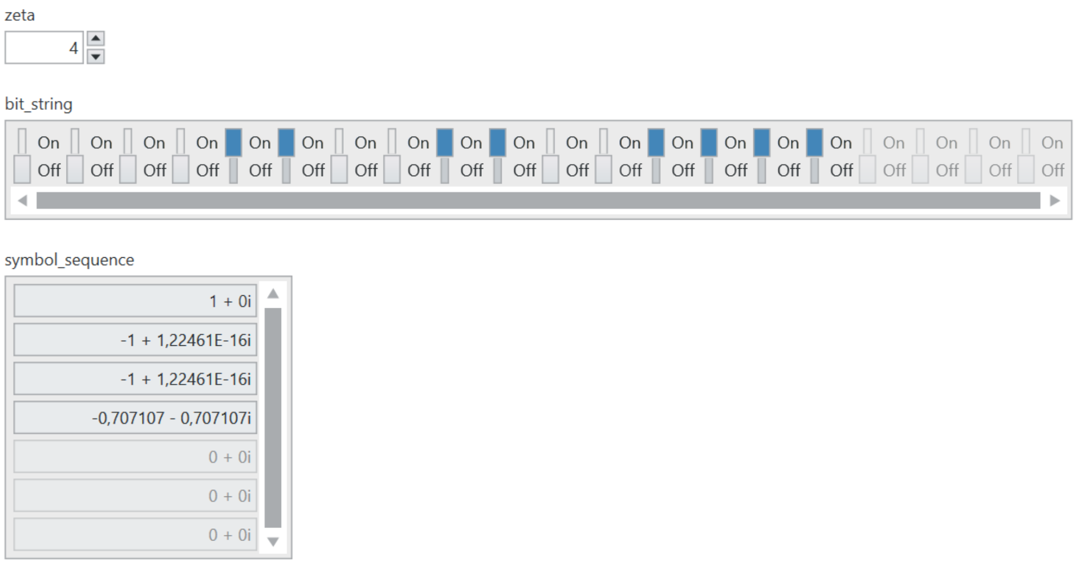
The goal of this lab experiment is that you learn how to create programs and algorithms in LabVIEW, without having a screenshot of the VI or even the needed blocks provided to you. Hence, you are free to implement the VI `symbol_mapping.gvi` however you want. That means you can use all standard functionality of LabVIEW. Please do not use any blocks from the *Modulation Toolkit* (these blocks start with `MT`), because otherwise the task would be too simple. This means that you should seek for a rather "basic" implementation that uses elementary functions and control structures.

Once you finished the implementation, convert the bit sequence from the pre-lab task 3 to a symbol sequence. Check if your output is correct.

Below we provide a "suggested implementation" in case you need hints for your implementation.

> There is a `Boolean` data type in LabVIEW. Internally, LabVIEW stores boolean data as 8-bit values. If the value is zero, the boolean value is FALSE. Any nonzero value represents TRUE. You can use a `Boolean array` to store a bit string.
> To convert an integer to a Boolean array, you can use `Number to Boolean Array`. The converse operation, i.e. converting a Boolean array into an integer is done via `Boolean Array to Number`. Note that in a Boolean array the least significant bit is stored at the first entry of the array, i.e., at array position 0 (little-endian). You can reverse an array by using the block `Reverse 1D Array`.

> Due to the freedom of implementation that you have for this experiment, in addition to the usual protocol, the code of the VI `symbol_mapping.gvi` has to be handed in.



zeta

4

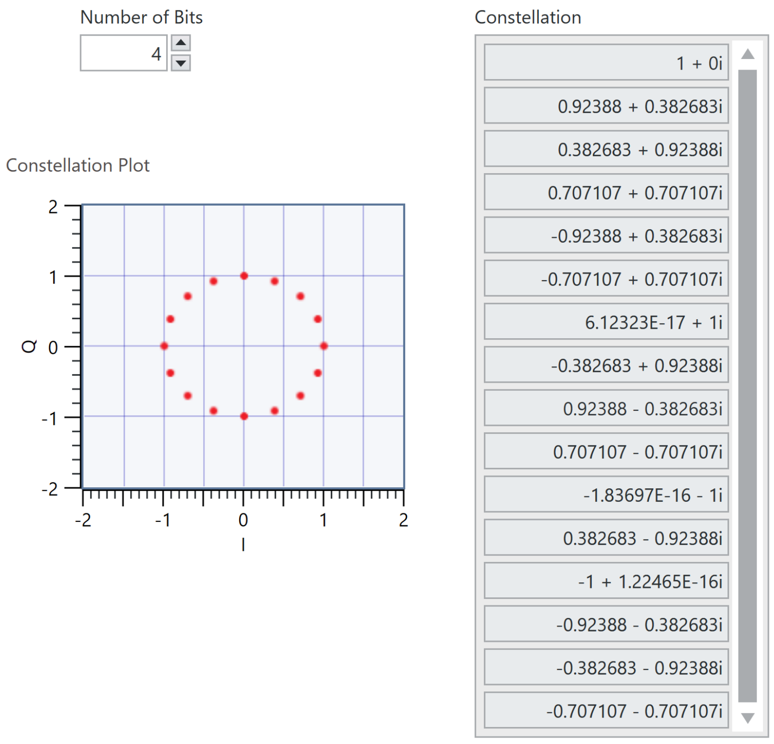
bit_string

symbol_sequence

1 + 0i
-1 + 1,22461E-16i
-1 + 1,22461E-16i
-0,707107 - 0,707107i
0 + 0i
0 + 0i
0 + 0i

Figure 5.3: Expected output of `symbol_mapping.gvi`.

Task B: Additionally, you will implement a second VI `constellation_plotter.gvi` that generates a constellation plot and a list of the constellation symbols (for M-PSK and with Gray labeling), as shown in Figure 5.4. As before, ζ, i.e. the number of bits per symbol should be a variable input of your program. Clearly, for this task, you can reuse much of your code from `symbol_mapping.gvi`.



Number of Bits

4

Constellation

1 + 0i
0.92388 + 0.382683i
0.382683 + 0.92388i
0.707107 + 0.707107i
-0.92388 + 0.382683i
-0.707107 + 0.707107i
6.12323E-17 + 1i
-0.382683 + 0.92388i
0.92388 - 0.382683i
0.707107 - 0.707107i
-1.83697E-16 - 1i
0.382683 - 0.92388i
-1 + 1.22465E-16i
-0.92388 - 0.382683i
-0.382683 - 0.92388i
-0.707107 - 0.707107i

Figure 5.4: Expected output of `constellation_plotter.gvi`.

5.4.1 Suggested Implementation for Task A

In this section, a possible implementation for `symbol_mapping.gvi` is presented. This task is split into several sub-tasks.

1. First, a subVI `M-PSK_constellation.gvi` that generates a list of all possible constellation symbols for M-PSK is implemented. To this end, you can use the equation

   $$c_k = \mathrm{e}^{i2\pi k/M}, \quad k = 0, \dots, M-1.$$

   Check if your subVI creates the correct output. At this point you can already do Task B.

Table 5.3: Description of `M-PSK_constellation.gvi`

Inputs	*Type of Variable*	*Definition*
zeta	U32	Number of bits per symbol
Outputs	*Type of Variable*	*Definition*
constellation	CDB array	List of M-PSK constellation points

2. Next, a subVI `gray_mapping.gvi` that generates the Gray mapping according to Table 5.1. We will implement the gray mapping in the form of a lookup table that is calculated only once. This avoids unnecessary calculations.

   The only input of the subVI `gray_mapping.gvi` is ζ, i.e., the number of bits per symbol. The output of this subVI is an array of integers (U32) of length $M = 2^\zeta$. The entries of this array are specified by columns 1 and 4 of Table 5.1. Think of the value in column 4 as the index of the array, and of the value in column 1 as the value of the array. That is array element 0 should be 0, array element 1 should be 1, array element 3 should be 2, array element 2 should be 3, array element 6 should be 4, and so on. To be specific, for the 4-bit Gray coding example given in Table 5.1, the array would be $[0, 1, 3, 2, 7, 6, 4, 5, 15, 14, 12, 13, 8, 9, 11, 10]$.

   In order to create this array, to the block `Search 1D Array` can be useful for your implementation. Further, the following instructions can be used to compute the Gray coded bits (column 3 in Table 5.1) for a given decimal number (column 1 in Table 5.1).

   a) Convert the decimal number into a binary representation x_1.
   b) Shift the binary representation right by one bit, which gives the binary number x_2.
   c) Modulo-2-addition (XOR) of x_1 and x_2 gives the desired Gray coded bits x_3.

   In short notation the calculation is $x_3 = x_1 \oplus (x_1 \gg 1)$.

   Check if your subVI creates the correct output.

Table 5.4: Description of `gray_mapping.gvi`

Inputs	*Type of Variable*	*Definition*
zeta	U32	Number of bits per symbol
Outputs	*Type of Variable*	*Definition*
Mapping	U32 array	Gray mapping (lookup table)

   In this subVI you will extensively use the blocks `Number to Boolean Array`. and `Boolean Array to Number`.

3. Finally, implement the VI `symbol_mapping.gvi` that was described in Table 5.2.

   Here, the basic idea is as follows: First, generate the Gray mapping lookup table and the list of constellation symbols, using your two subVIs. Second, using a for loop, the input bit string is split into chunks of size ζ. (The block `Array subset` might come in handy for this task.) Still in the for loop, the further steps are:

- Reverse the bits in the boolean array, then convert the result into an integer.
- Use this integer to select the correct element from the Gray mapping lookup table (use `Index Array`).
- Use this value to select the correct symbol from the list of constellation symbols (use `Index Array` again).
- `Tunnel` the symbols outside of the loop in the mode `Auto index values`. If the concepts `Tunnel` and `Auto index values` are unknown to you please take a tutorial about loops.

6 Pulse Shaping and Matched Filtering (Day 2)

The objective of this laboratory session is to introduce the basic concepts of pulse shaping and matched filtering. The order in which these procedures are performed is shown in Fig. 6.1, where the transmitter and receiver block diagrams have been simplified to emphasize the operations discussed in this session. A more detailed discussion of the missing blocks will follow in the upcoming sections.

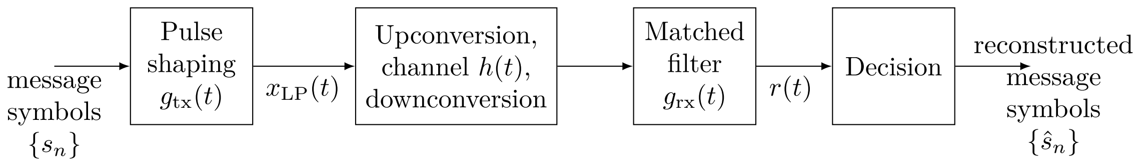

message symbols $\{s_n\}$
Pulse shaping $g_{\text{tx}}(t)$
$x_{\text{LP}}(t)$
Upconversion, channel $h(t)$, downconversion
Matched filter $g_{\text{rx}}(t)$
$r(t)$
Decision
reconstructed message symbols $\{\hat{s}_n\}$

Figure 6.1: Block diagram showing the pulse shaping and matched filtering procedures.

In digital communications, the stream of bits carrying the message that we want to send has to be first converted into an analog signal $x_{\text{BP}}(t)$ before its transmission. When the communication takes place over a wireless channel, there are two main requirements that need to be fulfilled: First, the transmit signal needs to be bandlimited (condition in the frequency domain) and, second, the inter-symbol interference (ISI) needs to be minimized (condition in the time domain).

Both requirements can be satisfied by using a pulse shaping filter at the transmitter that, as its names implies, *shapes* the symbols carrying information so that the outgoing signals comply with the two aforementioned criteria.

Further, at the receiver side, we also use a filter, the matched filter, whose main purpose is to maximize the receiver signal-to-noise ratio (SNR), so that we can then reconstruct the transmitted symbols with as little error as possible.

Both filters, the pulse shaping filter at the transmitter and the receiver filter, have to be jointly designed as we will see.

6.1 Pulse Shaping

As already mentioned, the pulse shaping filter determines the transmit pulse $g_{\text{tx}}(t)$ that is used to convert each symbol $s = s_I + is_Q$ into an analog waveform $s \cdot g_{\text{tx}}(t)$. The complex baseband signal is then given by the sum of all these individual pulses

$$x_{\text{LP}}(t) = \sum_{n=-\infty}^{\infty} s_n g_{\text{tx}}(t - nT), \tag{6.1}$$

where T is the delay between two consecutive transmitted symbols, i.e., the symbol period.

A main trade-off that has to be considered in the design of the impulse response $g_{\text{tx}}(t)$ of the pulse shaping filter is as follows: The more concentrated this signal is in the frequency domain (i.e., the smaller the bandwidth), the more extended the signal will be in the time domain, and vice versa.

Analog signals that have a finite duration in the time domain, have an infinitely extended spectrum in the frequency domain. Rectangle signals normally used for representing bits are a good example of this, as shown in Fig. 6.2. Although sequential symbols shaped with a rectangular signal do not exhibit ISI, their spectrum extends infinitely and it would hypothetically affect communication systems operating in adjacent frequency channels. In general, systems are not allowed to emit out-of-band radiation above strict thresholds set by regulatory entities. Hence, rectangular pulses are almost never used in commercial wireless communication systems.

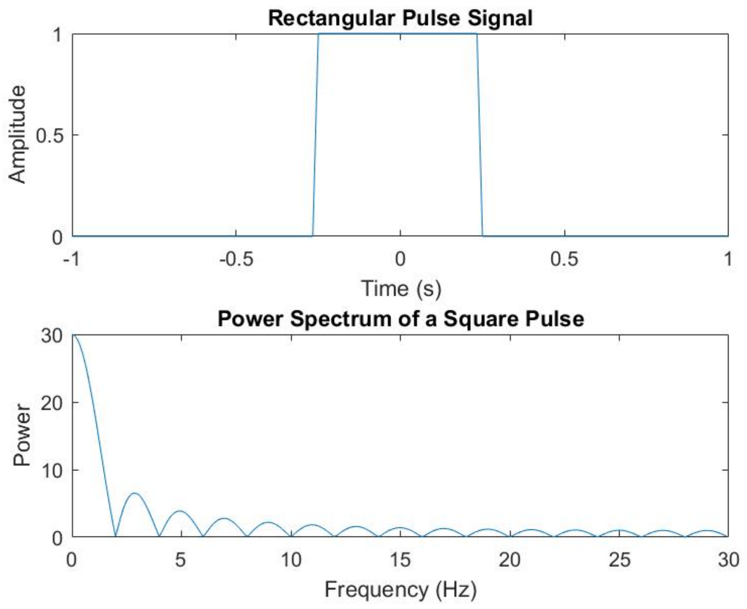


Figure 6.2: A time-limited rectangular pulse signal rect$(2t)$, and its single sided band spectrum.

The other extreme are pulses that have a rectangular shape in the frequency domain. This corresponds to a *sinc*-function shape in the time domain. While theoretically useful, such ideal low pass filters cannot be implemented in practice.

Thus, for practical applications, the pulses need to be bandlimited and also have a smooth shape in the frequency domain (in contrast to the discontinuity of the rectangular function). This can be achieved, for example, by using raised cosine or root raised cosine pulse shaping filters. Although the pulses generated by these filters are quickly decaying in the time domain, they have an effective duration that spans several symbol intervals. It is nevertheless possible to have no inter-symbol interference and to detect the symbols error-free (assuming that there is no additional noise) at the receiver if the system satisfies the first Nyquist criterion.

Note that our discussion so far was restricted to the transmitter pulse shaping filter. A real communication system consists of transmit filter, communication channel, and receive filter. Hence all three components have to be considered when we are interested in an inter-symbol interference free communication system.

First Nyquist Criterion. Let $r(t)$ denote the complex baseband signal at the receiver after downconversion that is received when only one transmit pulse $g_{\text{tx}}(t)$ is transmitted, i.e., we have $x_{\text{LP}}(t) = g_{\text{tx}}(t)$. Note that

$$r(t) = \frac{1}{2}(g_{\text{tx}} * h_{\text{LP}} * g_{\text{rx}})(t),$$

where h_{LP} denotes the equivalent lowpass representation of the channel h. See Appendix B.2 for details.

According to the first Nyquist criterion, each pulse should have no influence on the sample

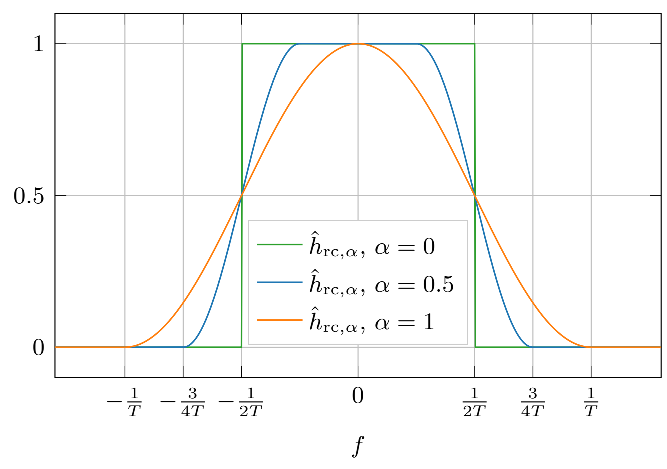


Figure 6.3: Transfer function of the raised cosine filter $\hat{h}_{\mathrm{rc},\alpha}(f)$ for different roll-off factors α.

values at other sample times. This means that the received pulse at the detector r, should satisfy

$$r(kT) = \begin{cases} 1, & \text{if } k = 0, \\ 0, & k \neq 0. \end{cases} \tag{6.2}$$

It can be shown that this time-domain criterion translates into the frequency-domain criterion

$$\sum_{k=-\infty}^{\infty} \hat{r}\left(f - \frac{k}{T}\right) = \text{const.} \tag{6.3}$$

6.1.1 Raised Cosine Filter

A raised cosine filter is commonly used as a pulse shaping filters in communication systems, because it is bandlimited and satisfies the first Nyquist criterion. The transfer function of a raised cosine filter is given by

$$\hat{h}_{\mathrm{rc},\alpha}(f) = \begin{cases} 1, & |f| \leq \frac{1-\alpha}{2T} \\ \frac{1}{2}\left[1 + \cos\left(\frac{\pi T}{\alpha}\left[|f| - \frac{1-\alpha}{2T}\right]\right)\right], & \frac{1-\alpha}{2T} < |f| \leq \frac{1+\alpha}{2T} \\ 0, & \text{otherwise.} \end{cases}$$

Note that the term *raised cosine* refers to the shape of the filter transfer function. The transfer function of a raised cosine filter is plotted in Fig. 6.3 for three values of α, namely, 0, 0.5, and 1. The parameter α is called the *roll-off factor*, and determines the excess bandwidth over the minimum bandwidth of $1/(2T)$. It can be observed that the transfer functions for $\alpha = 0.5$ and $\alpha = 1$ cut off gradually compared to ideal low pass filter with minimum bandwidth (i.e., $\alpha = 0$), and therefore are easier to implement in practice. Note that smaller values of α correspond to a smaller filter bandwidth and hence to an increased effective duration of the filter impulse response.

The impulse response of the raised cosine filter $h_{\mathrm{rc},\alpha}(t)$ for different roll-off factors α is plotted in Fig. 6.4.

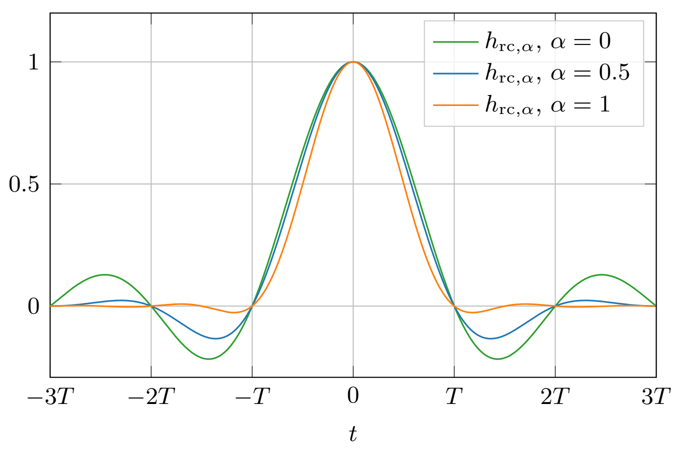


Figure 6.4: Impulse response of the raised cosine filter $h_{\mathrm{rc},\alpha}(t)$ for different roll-off factors α.

6.1.2 Root Raised Cosine Filter

The response of a raised cosine filter satisfies the first Nyquist criterion. However, if we use a raised cosine pulse shaping filter at the transmitter and again a raised cosine filter as its matched filter at the receiver, then the overall response of transmit and receive filter will not satisfy the first Nyquist criterion. Therefore *root raised cosine* filters are used instead at the transmitter and the receiver, giving an overall raised cosine response, which satisfies the first Nyquist criterion. Note that the impulse response of the root raised cosine filter itself does not satisfy the first Nyquist criterion, but the overall response of transmit and receive filters does. The transfer function of the root raised cosine filter is given by

$$\hat{h}_{\mathrm{rrc},\alpha}(f) = \sqrt{\hat{h}_{\mathrm{rc},\alpha}(f)}.$$

In Fig. 6.5 the impulse response of the root raised cosine filter $h_{\mathrm{rrc},\alpha}(t)$ is plotted for different roll-off factors α.

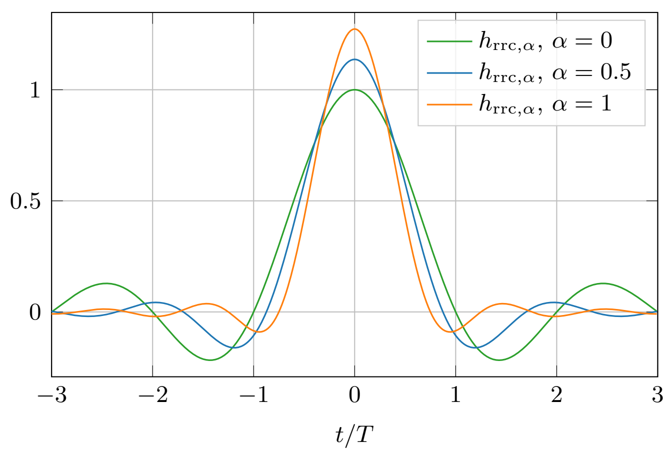


Figure 6.5: Impulse response of the root raised cosine filter $h_{\mathrm{rrc},\alpha}(t)$ for different roll-off factors α.

6.2 Matched Filter

The *matched filter* in communications is a filter that is “matched” to the transmit filter. Let $g_{\mathrm{tx}}(t)$ denote the impulse response of the transmit filter. Then the matched filter is given by

$$g_{\mathrm{m}}(t) = \overline{g_{\mathrm{tx}}(-t)}.$$

That is, the impulse response of matched filter is the time-mirrored and complex conjugated impulse response of the transmit filter. The matched filtered is the linear filter that maximizes the receiver SNR.

If we use a root raised cosine filter as transmit filter $g_{\text{tx}}(t) = h_{\text{rrc},\alpha}(t)$ than the matched filter is the same root raised cosine filter $g_{\text{m}}(t) = h_{\text{rrc},\alpha}(t)$. Further, if we use this matched filter as receive filter $g_{\text{rx}}(t) = g_{\text{m}}(t) = h_{\text{rrc},\alpha}(t)$, then the impulse response of the overall filter $(g_{\text{tx}} * g_{\text{rx}})(t) = (h_{\text{rrc},\alpha} * h_{\text{rrc},\alpha})(t) = h_{\text{rc},\alpha}(t)$ is the impulse response of the raised cosine filter, and hence satisfies the first Nyquist criterion.

6.3 Eye Diagram

The eye diagram is a method widely used for analyzing received analog signals and provides a measure of the quality of their transmission. It is mainly used to identify adverse effects caused either by the channel or the analog signal processing blocks. The eye diagram is constructed by visually overlaying in the time interval $[-T, T]$ multiple parts of the received signals, corresponding to all possible transmit sequences. In Section 6.4 we will explain in more detail how an eye diagram is created. An example of how an eye diagram looks like in an oscilloscope is given in Fig. 6.6.

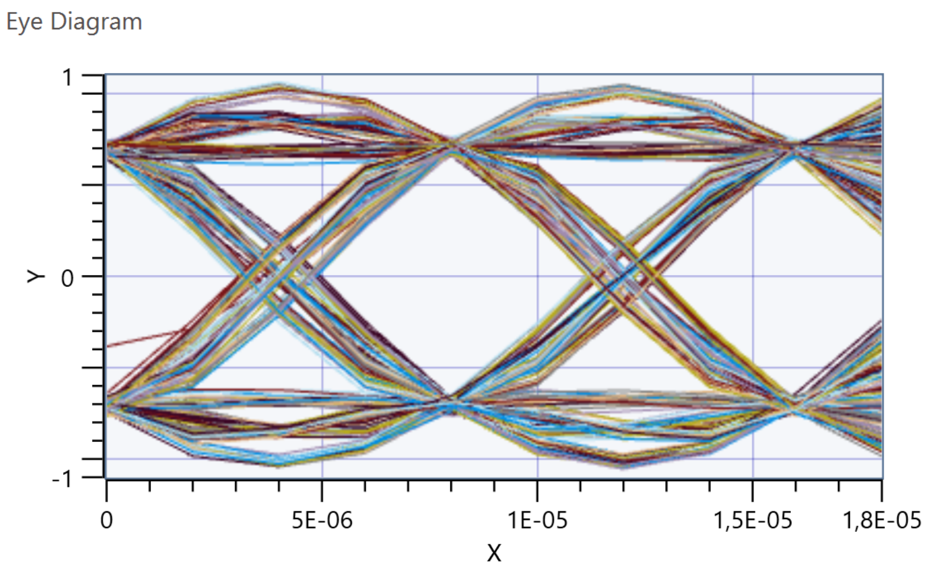


Figure 6.6: Example of an eye diagram.

The vertical opening of the eye diagram helps to analyze the sensitivity of the detector. An eye diagram with a large vertical opening points out to a receiver that is less sensitive to noise. On the other side, the larger the distance between the signals is, thus the smaller the probability of error will be, as it becomes easier to differentiate between signals. If the first Nyquist criterion is fulfilled, then the vertical eye opening is at its maximum.

The horizontal opening of the eye diagram provides a measure of the sensitivity of the system to variations in the sampling instants. An eye diagram with a large horizontal opening is a strong indicator that there will be fewer synchronization errors.

6.4 Example: Pulse Shaping and Eye Diagram

In this section we will give an example that explains step by step how an eye diagram is created. By doing so, also the procedure of pulse shaping will become clearer. For this example we use a binary phase-shift keying (BPSK) modulation with symbol alphabet $\mathcal{C} = \{-1, 1\}$. Having only real-valued signals, makes the illustration simpler.

We use the bit string $\{0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1\}$, which translates to the symbol sequence $\{s_n\} = \{-1, 1, -1, 1, -1, -1, 1, 1, -1, 1, 1, 1\}$. In Fig. 6.7 we see the corresponding baseband signal $x_{\mathrm{LP}}(t)$ (solid curve) that is calculated according to (6.1) using a raised cosine pulse shaping filter $g_{\mathrm{tx}} = h_{\mathrm{rc},\alpha}$ with roll-off factor $\alpha = 0.5$. The dotted gray curves are the wave forms for a single symbol and the solid curve is the sum of all dotted curves. Since $g_{\mathrm{tx}} = h_{\mathrm{rc},\alpha}$ satisfies the first Nyquist criterion, we have no inter-symbol interference. We can see that $x_{\mathrm{LP}}(t)$ (solid curve) perfectly interpolates our symbol sequence, i.e., if we sample $x_{\mathrm{LP}}(t)$ at multiples of T, we recover our symbol sequence: $s_n = x_{\mathrm{LP}}(nT)$, $n = 0, \ldots, 11$.

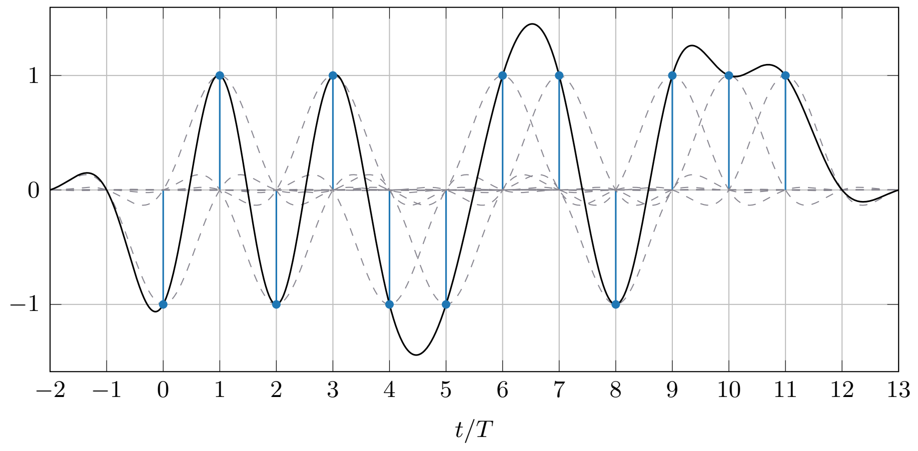


Figure 6.7: Baseband signal $x_{\mathrm{LP}}(t)$ generated using a raised cosine pulse shaping filter $g_{\mathrm{tx}} = h_{\mathrm{rc},\alpha}$ with roll-off factor $\alpha = 0.5$.

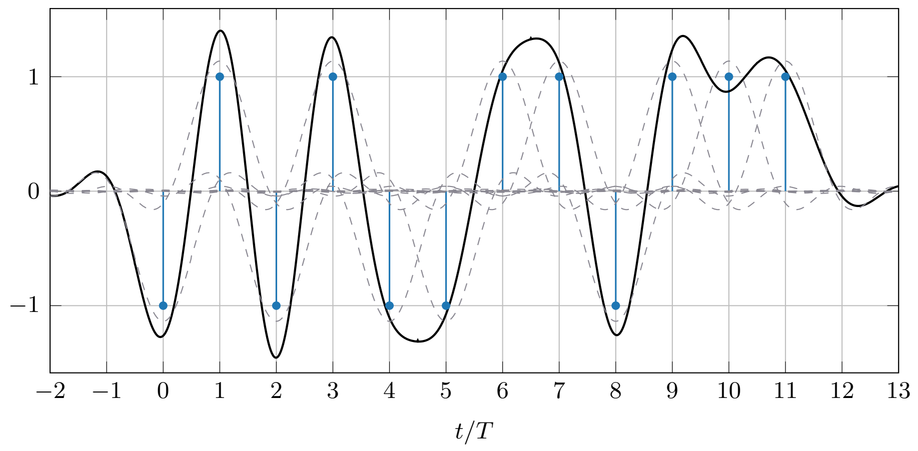
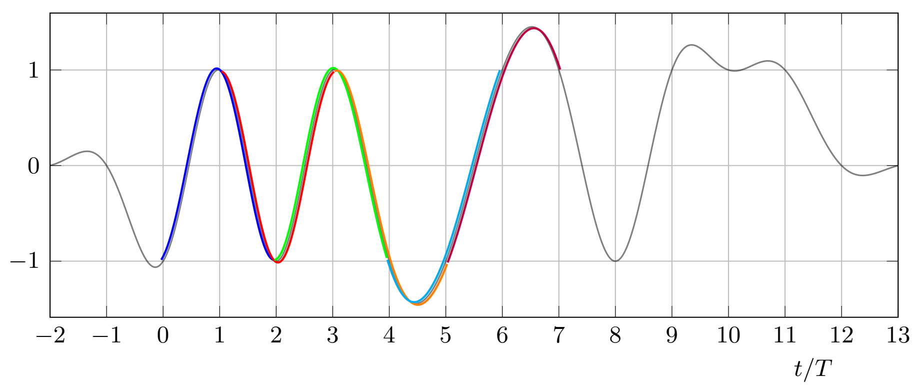


Figure 6.8: Baseband signal $x_{\mathrm{LP}}(t)$ generated using a root raised cosine pulse shaping filter $g_{\mathrm{tx}} = h_{\mathrm{rrc},\alpha}$ with roll-off factor $\alpha = 0.5$.

In Fig. 6.8 we see the same plot, but this time a root raised cosine pulse shaping filter $g_{\mathrm{tx}} = h_{\mathrm{rrc},\alpha}$ with roll-off factor $\alpha = 0.5$ has been used. Here, the transmit filter $g_{\mathrm{tx}} = h_{\mathrm{rrc},\alpha}$ does not satisfy the first Nyquist criterion. And, as a consequence, we have inter-symbol interference.

Next, we illustrate in Fig. 6.9 how the eye diagram is created. We use the signal from Fig. 6.7 that was generated using a raised cosine pulse shaping filter. Segments of the signal, taken from the time intervals $[-T + nT, T + nT]$, $n = 1, \ldots, 10$, are overlayed to obtain the eye diagram (lower panel).



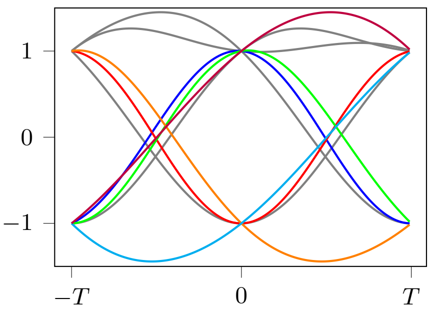


Figure 6.9: Top: Baseband signal $x_{\text{LP}}(t)$ and signal segments of length $2T$ in different colors. Bottom: Eye diagram.

In the rest of this section, we present eye diagrams for signals that were generated with different pulse shaping filters. For these illustration $T = 1$ was selected. In Fig. 6.10 a pulse shaping filter with rectangular impulse response was chosen, in Fig. 6.11 a raised cosine filter ($\alpha = 0.5$), and in Fig. 6.12 a root raised cosine filter ($\alpha = 0.5$).

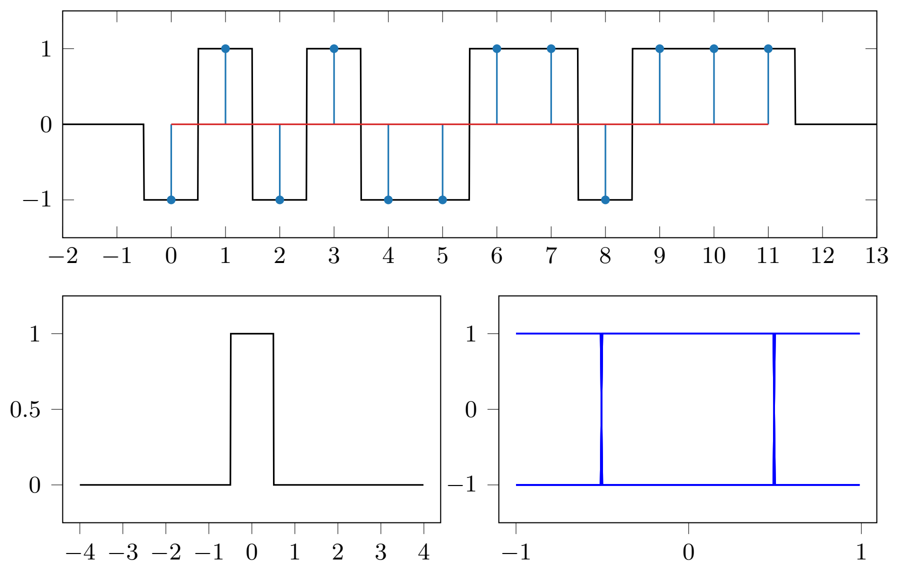


Figure 6.10: Top: Baseband signal $x_{\text{LP}}(t)$ generated using a pulse shaping filter with rectangular impulse response. Bottom left: Impulse response of the pulse shaping filter. Bottom right: Eye diagram.

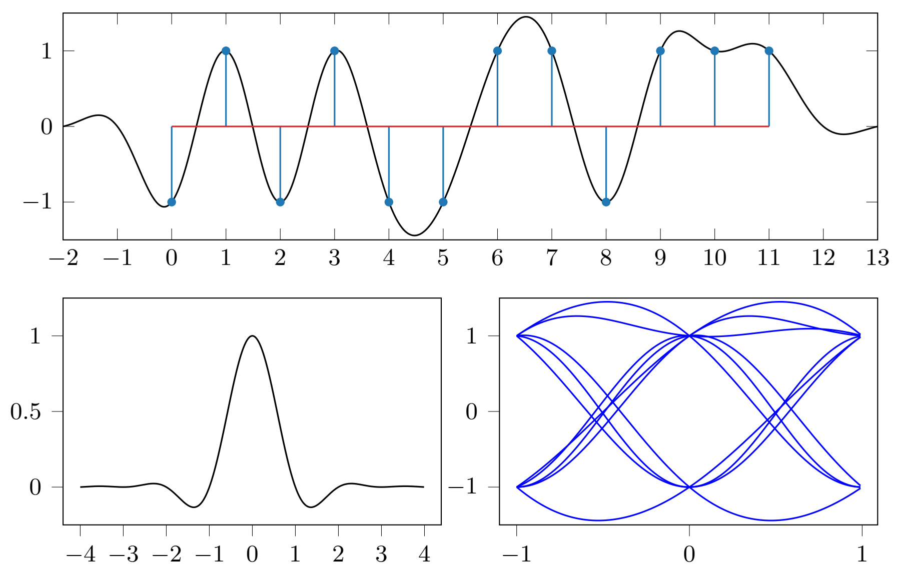


Figure 6.11: Top: Baseband signal $x_{\mathrm{LP}}(t)$ generated using a raised cosine pulse shaping filter $g_{\mathrm{tx}} = h_{\mathrm{rc},\alpha}$ with roll-off factor $\alpha = 0.5$. Bottom left: Impulse response of the pulse shaping filter $h_{\mathrm{rc},\alpha}$. Bottom right: Eye diagram.

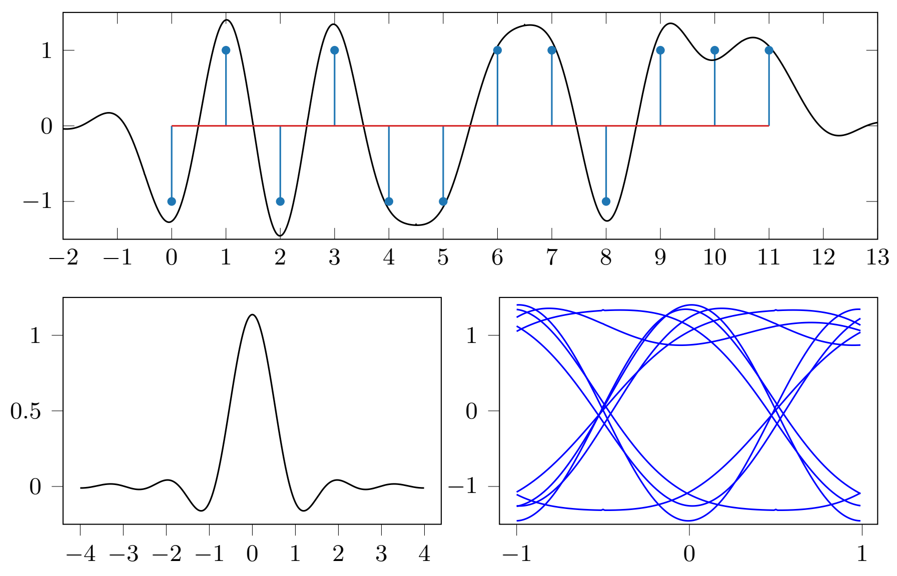


Figure 6.12: Top: Baseband signal $x_{\mathrm{LP}}(t)$ generated using a root raised cosine pulse shaping filter $g_{\mathrm{tx}} = h_{\mathrm{rrc},\alpha}$ with roll-off factor $\alpha = 0.5$. Bottom left: Impulse response of the pulse shaping filter $h_{\mathrm{rrc},\alpha}$. Bottom right: Eye diagram.

Note that in this section we visualized the eye diagram for baseband signal $x_{\mathrm{LP}}(t)$ that is created at the transmitter. In practice it is often more interesting to look at the eye diagram of the baseband signal at the receiver after the matched filter.

6.5 Pre-Lab $\boxed{2}$

Your protocol should include answers to the following questions.

1. Name three causes of inter-symbol interference (ISI), and three countermeasures how to avoid it in a communication system.
2. Similar to Fig. 6.2, plot the time-domain signal and its respective Fourier transform of the triangle function $\Lambda(t)$, defined as

$$\Lambda(t) = \begin{cases} 1 - |t|, & |t| < 1, \\ 0, & \text{otherwise.} \end{cases}$$

   This time show the double side band spectrum graphically.
3. Why does the single side band representation normally suffices to characterize the full spectrum of a signal, even though the spectrum is not fully shown?
4. Consider two baseband signals pulse shaped by the functions rect(t) and $\Lambda(2t)$, i.e., having the same width and the same amplitude. Which of the two transmitted signals will cause higher adjacent-channel interference (ACI)? Elaborate, and (optionally) show the effect graphically.
5. Elaborate on three characteristics that any potential mathematical function should fulfill in order to be eligible for being used for pulse shaping and matched filtering in a communication system.
6. Consider a bit generator that assembles messages of N bits each to be transmitted using a M-ary modulation scheme, where $\frac{N}{log_2 M} \in \mathbb{N}$. We want to transmit one of such messages, thus we first perform upsampling of the symbols at a rate of R samples per symbol before passing this upsampled signal to our pulse shaping filter, which has a length of D (i.e., number of filter taps). How many samples will be transmitted per message? Provide your answer as a function of the given variables.
7. As explained, the objective of the matched filter is to maximize the peak SNR. Explain what does this ratio mean, and why do we want to maximize it? Elaborate how this is done, mathematically.
8. Elaborate on the phenomena that causes the signal properties D_A, M_N, J_T and S_T shown in the eye diagram in Fig. 6.13.

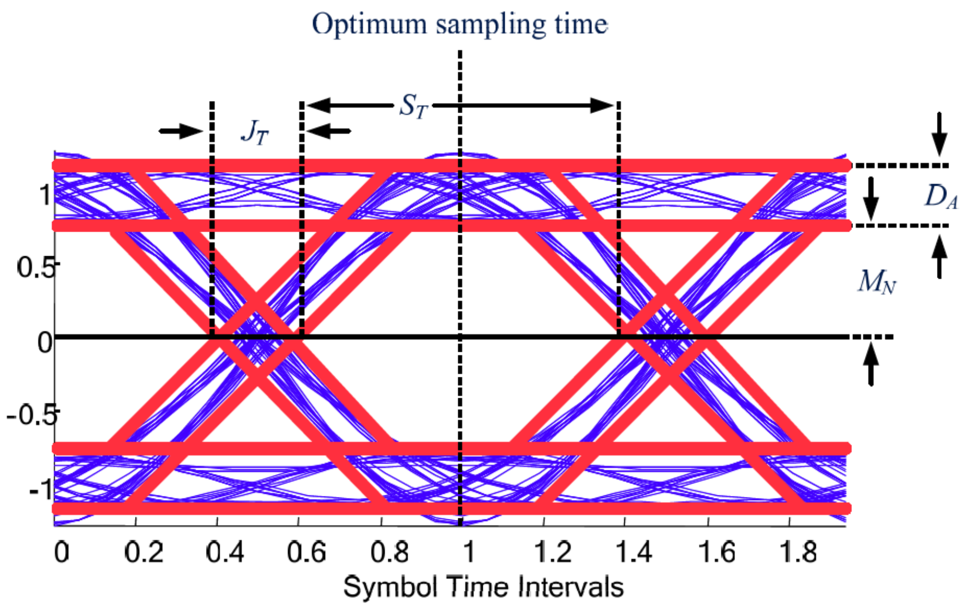


Figure 6.13: Signal distortion effects identifiable from an eye diagram.

6.6 Lab Experiment $\boxed{3}$

In this lab experiment, you will learn how to create an eye diagram. In addition to this, you will learn how to examine the eye diagram and how to analyze the detected pulses just by viewing an eye diagram. Also, you will observe how an eye diagram changes when inter-symbol interference (ISI) is present in the communication channel.

First, you will build a transmitter that generates a PN sequence and then performs pulse shaping to create the transmit waveform. Different filters can be used to shape the pulses. Quadrature amplitude modulation (QAM) is used as a modulation scheme. At the receiver end a QAM demodulator and a matched filter is used to recapture the symbols from the received pulses. Then you will add an AWGN channel to simulate the results.

At the end of this lab experiment you will also transmit and receive the shaped pulses using the USRPs boards.

6.6.1 Transmitter

Open the `TxRx.gvi` file. For the transmission, we will use QAM modulated symbols. You first need to activate the `Inputs` and `Transmitter Side` blocks.

In the `Transmitter Side` block you need to generate the system parameters using the `MT Generate QAM System Parameters (M)` block. This block will generate the `QAM system parameters` Cluster, which is required for modulation of the data bits, and `bits per symbol` (which we will not use).

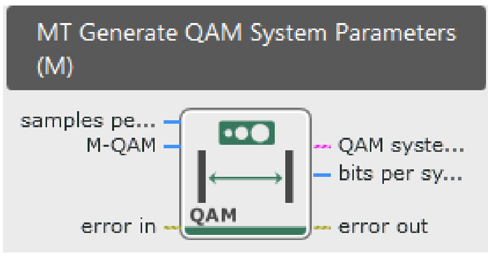
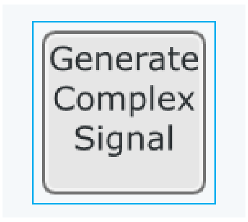


Figure 6.14: Block `MT Generate QAM System Parameters`.

Now you will complete the `Generate Complex Signal` block that performs the pulse shaping and modulation.



Inputs to this block are parameters of pulse shaping filter, PN Sequence Order (i.e., 9), QAM system parameters, symbol rate (divide IQ sampling rate to Samples per Symbols) and error. This block outputs a complex waveform having shaped and modulated pulses.

To generate the required filter coefficients you'll need the `MT Generate Filter Coefficients` block.

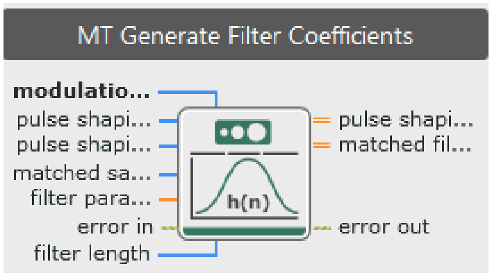
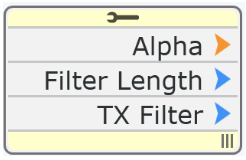


Figure 6.15: Block `MT Generate Filter Coefficients`.

Provide all the required inputs to the `MT Generate Filter Coefficients` block. The required inputs are `pulse shaping filter` (connect to `TX Filter`), `pulse shaping samples per symbol` (connect to `samples per symbol`), `filter parameter` (connect to `Alpha`), `error in` (connect to `error out` of `MT Generate Bits (Galois, PN Order)`), and `filter length` (connect to `Filter Length`). You need to unpack the cluster `Parameters of Pulse shaping filter` to obtain some of these inputs.



Right-click on the wire connected to the `Parameters of Pulse shaping filter`. Select *Cluster* and then *Cluster Properties*. After having connected the inputs, provide the `pulse shaping filter coefficients` to the `MT Modulate QAM` block. This block will generate the complex signal which will be transmitted. The last remaining open connection, which goes to the addition block has to be connected to `Filter Length`.

Now test your code by running `TxRx.gvi`. Initially, set `Samples per Symbol` to 16 and `M-QAM` to 4 as well. Add the `constellation graph` to the panel. Run the simulations for different parameters `Alpha` between 0 and 1, as well as for the *TX filters* `none`, `Raised Cosine`, and `Root Raised Cosine`. How does the constellation graph change?

6.6.2 Receiver

At the receiver side, you need to generate the filter coefficients for the *matched filter*. In the VI `TxRx.gvi` activate the receiver sub-diagram. Provide all the inputs to the `MT Generate`

`Filter Coefficients` block by again unpacking the filter parameters. Filter parameters at the transmitter and receiver side must be the same. After this provide the matched filter coefficients to the `MT Demodulate QAM` block. also provide the `QAM System Parameters` to the `MT Demodulate QAM` block.

Important Remark: The `MT Generate Filter Coefficients` block has the outputs `pulse shaping filter coefficients` and `matched filter coefficients`. However, `matched filter coefficients` are, unlike the name suggest, not the filter coefficients of the matched filter as introduced in Section 6.2.
In the experiment you will plot the filter coefficients and then you will see what the `matched filter coefficients` are. Based on the outcome of this experiment, please answer the question: What would be a better name for these coefficients?

We need to resample the complex waveform before passing it to the `MT Demodulate QAM` block. To this end, we employ the `MT Resample (Complex Cluster)` block. Pass the `output complex waveform` from the *Transmitter Side* directly to the input `input complex waveform` of the `MT Resample (Complex Cluster)` block (that is we use no channel for the moment). The `desired sample rate` of the `MT Resample (Complex Cluster)` block needs the `IQ Sampling Rate [S/sec]` as input. Then pass the `output complex waveform` of the `MT Resample (Complex Cluster)` block to the `input complex waveform` of the `MT Demodulate QAM` block.

Next, pass the `recovered complex waveform` output of the `MT Demodulate QAM` block to the `waveform` input of the `MT Format Eye Diagram (complex)` block. Further connect the input `symbol rate` to an appropriate wire in the diagram, where you already have the symbol rate available.

Now run `TxRx.gvi`. Change the different parameters and observe the eye diagram changing. Use the *TX filters* `none`, `Raised Cosine`, and `Root Raised Cosine`. Do you see a difference between `Raised Cosine` and `Root Raised Cosine`? Why not? Plot the impulse response of the receive filter for the *TX filters*: `none`, `Raised Cosine`, and `Root Raised Cosine`. Which filter is used at the receiver in each case? For each case, please also answer the questions: Is this receive filter the matched filter to the *TX filter*? Do *TX filter* and receive filter together satisfy the first Nyquist criterion?

Further, study the influence of the roll-off factor by choosing different parameters `Alpha` between 0 and 1.

Next, we add an *AWGN channel*. Activate the `AWGN Channel` sub-diagram. Provide all necessary inputs to the `MT Add AWGN` block, and wire the `output complex waveform` from the *Transmitter Side* to the `MT Add AWGN` block. Connect the `output complex waveform` of the `MT Add AWGN` block with the input `input complex waveform` of the `MT Resample (Complex Cluster)` block. How does the eye diagram change in the presence of noise? What it the influence of the parameter `eb/n0 AWGN Noise Power`?

6.6.3 Eye diagram Using USRP

Run the `USRP-Rx.gvi` on one USRP board and for the receiver side run `USRP-Rx.gvi`. As before, you will use the coaxial cable and the attenuator, i.e., you do not need to change the hardware setup. Change the parameters of the filters, filter type, roll-off factor and filter length and observe the changes to the eye diagram.

7 Digital Receivers and Carrier Synchronization (Day 3)

In the early days of receivers' design, digital systems were not as powerful as today, and thus the different synchronization procedures at the receiver were carried out using analog components such as capacitors, inductors, resistances or even vacuum tubes. However, this solution was not trivial since the analog components limited the processing capability. This restricted the receiver to performing a small number of simple operations. Further, analog components created additional problems that were difficult to solve, such as introducing additional sampling delays that could neither be detected nor compensated, or the fact that electronic components drift with time, temperature, and aging.

Thus, the transition to the digital domain was a necessary step in order to avoid dealing with such uncertainties. This is the reason why today most relevant processing is done in the digital domain. Although front-end architectures vary according to specific technologies and manufacturers, the general construction is mostly the same. In Fig. 7.1 it is shown how a typical generic digital receiver looks nowadays, also known as a homodyne receiver or direct conversion receiver. It represents one of the most recent schemes used for RF front-end design that is also implemented in the USRP-2901 used for this laboratory. In this receiver type, the RF signal is directly converted to a baseband signal by using a mixer *coarsely* tuned to a specific frequency channel (e.g., 2.4 GHz or 5 GHz in case of IEEE 802.11n), before being sampled and quantized (i.e., converted to the digital domain).

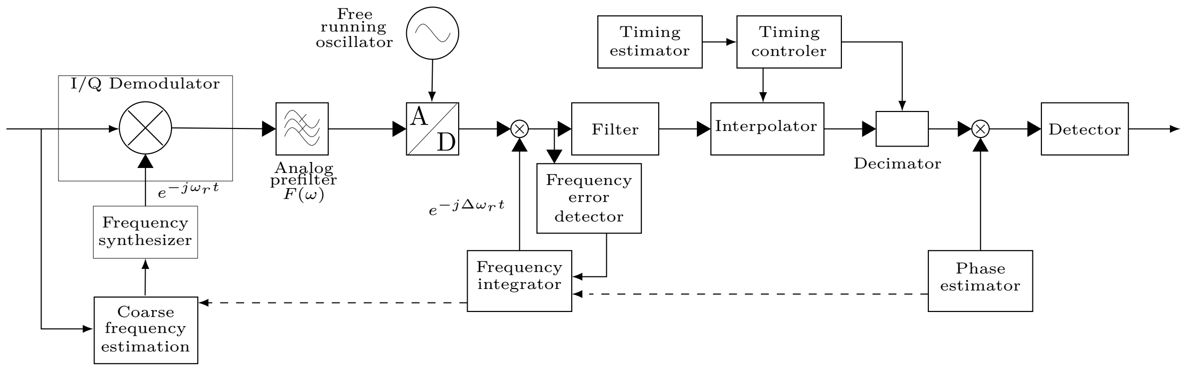


Figure 7.1: Block diagram of a typical digital receiver.

A much older but still popular scheme for RF front-end design is the *superheterodyne receiver*, where the channel selection is done by translating the carrier frequency to a fixed intermediate frequency f_{IF} first using a frequency synthesizer, before the final downconversion to basedband is performed using a second oscillator set to a fixed frequency. One of the main problems of these receivers is, however, that the *image frequency* f'_{IF} created after downconverting the signal using the frequency synthesizer often introduces interference among multiple simultaneous transmissions in different frequency channels, reducing the overall performance in the communication.

In the subsequent sections of this laboratory we will work on the design and implementation of the discrete components of this typical digital receiver for single carrier transmissions, which is well suited for flat fading channels. Thus our main focus will be on implementing the algorithms used for digital signal processing, i.e., after the signal has been downconverted into its in-phase and quadrature (I/Q) components at the RF front-end, and oversampled using an analog-to-digital converter (ADC).

> Flat fading channels, contrary to frequency selective channels, are mathematically characterized by a "flat" transfer function, where all frequency components of the spectrum are equally attenuated. Although this attenuation may change over time, the transfer function remains flat at all times.

7.1 Analog Signal Processing (RF Front-End)

Although in our laboratory all the analog operations are performed by the USRP-2901 itself by default and you will not need to implement them, in this section we will still further elaborate on such operations for the sake of completeness, before moving to the digital signal processing. This will help us understand which channel phenomena have not been compensated in the analog domain and what additional effects these analog components introduce into the received signal, that will need to be dealt with in the digital world later on.

Received signals consist of complicated analog waveforms that are first processed by the RF front-end, which performs analog operations in order to reduce some of the negative effects of the channel before digitizing the signal. With the purpose of placing more emphasis on the analog operations performed at the RF front-end, the typical homodyne receiver depicted in Fig. 7.1 has been redrawn in Fig. 7.2 and its components will be discussed in more detail next.

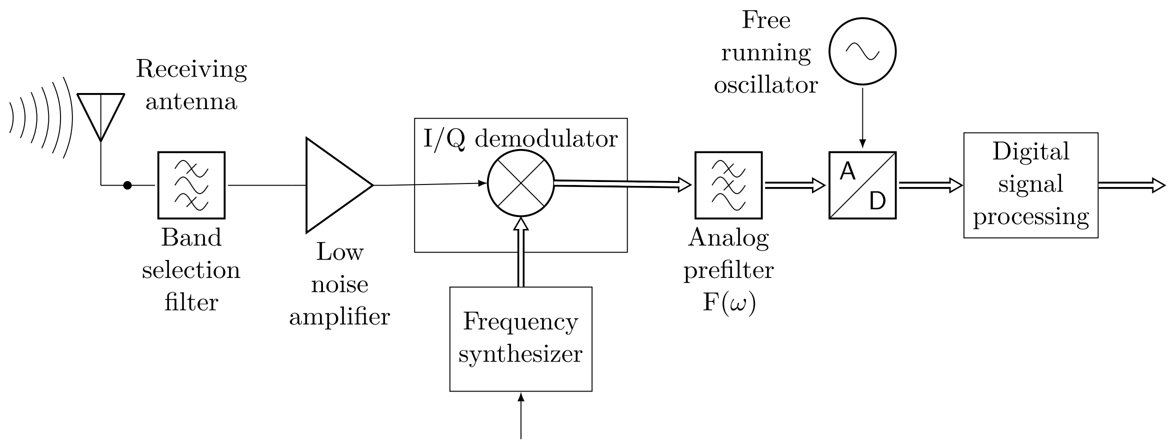


Figure 7.2: Homodyne or direct down conversion receiver.

7.1.1 Receiving antenna

Upon signal detection by the receiving antenna, incoming electromagnetic waves are converted into electric signals. For maximum energy transfer between them (i.e., minimum energy loss), the impedance of the antenna and the outer electronic components of the RF front-end need to be matched. Since this matching involves using resistors, as long as we do not operate at 0 K or ~ -273 °C such passive devices will introduce thermal noise that will be added to the noise introduced by the wireless channel.

Thus, electric signals that are input to the RF front-end already carry in their frequency, phase, and amplitude not only modulated information and adverse channel effects, but also noise introduced by the antenna connection itself.

It is worth mentioning that thermal noise is not one of our main concerns here though. One of the most important things that you have to consider when working with analog components is that they introduce additional noise at each step, and this is unavoidable. So the more analog components we have at the receiver, the higher this *noise factor* will be. Although the efforts on RF front-end design focus on minimizing such noise contribution, electronic components such as mixers and amplifiers, among others, end up accounting for up to 80-90% of the total noise power at the RF front-end. Thus, the noise introduced by any electronic device is taken into account when planing the *link budget*, and it is normally given in its technical specifications. The receiving chain of the USRP-2901 used in this laboratory, for instance, introduces a *noise figure* of the order of $5-7$ dB, as listed in Table 3.1.

7.1.2 Band Selection Filter

This analog filter is responsible of selecting the band of interest and suppressing any out-of-band RF transmissions. The presence of stronger adjacent RF signals can produce negative effects when processed at the RF front-end, e.g. causing our original transmission to be *compressed* and *distorted*.

One of the main advantages of using software-defined radios such as the USRP-2901 used in this laboratory is that this band-pass filter can be controlled electronically, i.e., by selecting the carrier frequency f_c and bandwidth to use for the communication.

7.1.3 Low Noise Amplifier (LNA)

Its main task is raising the signal level without introducing much noise, thus keeping its *noise figure* low. Furthermore, since each amplification step not only increases the signal level but also its noise level, implementing the LNA at the edge of the RF front-end, i.e., before any further analog component is used, additionally ensures that the noise introduced by such components is not amplified as well.

7.1.4 Frequency Synthesizer

For converting the received RF signal directly to baseband we need an oscillator at the receiver that produces a frequency ω_r, which may not perfectly match the frequency ω_t used by the oscillator at the transmitter but needs to be close enough ($\omega_r \approx \omega_t$). Now the question arises: how do we construct an oscillator that generates such frequency? Of course we can use an LC circuit to build an electric resonator that produces ω_r, given by

$$\omega_r = \frac{1}{\sqrt{LC}}.$$

The problem with an LC oscillator is, however, that it is sensitive to temperature changes. Thus, as the temperature rises, the nominal values of both its capacitance and inductance will start changing, causing frequency drifts. Although there are methods to minimize this effect, the use of inductors and capacitors is generally avoided since, given their physical size, it is very difficult to incorporate them into integrated circuits (IC).

A modern solution to this problem involves the use of quartz crystals for frequency synthesis, since they are known to be very precise and less sensitive to temperature changes. For instance, USRP-2901 uses a quartz crystal that provides an accuracy of the order of 2.5 ppm (two and a half parts per million for frequency uncertainty), as listed in Table 3.1. Under this approach the frequency ω_r can be generated with a very high accuracy, but it is nonetheless not completely precise since 2.5 ppm still represents an error of 0.00025%. So, after all, modern frequency synthesizers still generate a frequency offset against the actual wanted frequency. This will be observed in more detail during the laboratory, when the carrier offset will be estimated.

Although Fig. 7.1 as well as Fig. 7.2 depict two different frequencies used for the I/Q demodulation and ADC operation at the RF front-end, in practice their frequency is generated from a unique source composed of a quartz crystal in a phase-locked loop (PLL) circuit for increased stability. This PLL frequency synthesizer is set to a reference frequency that has a very high precision. Thus any frequency needed in the RF front-end is generated as a multiple of such reference frequency thanks to the use of integer-n and fractional-n digital *frequency dividers*. This is what makes it possible for the USRP-2901 to operate at a dynamic frequency range of 70 MHz - 6 GHz, as listed in Table 3.1

7.1.5 I/Q Demodulation

In order to simplify the digital processing of the incoming signal later on, the RF signal is next downconverted and divided into its in-phase and quadrature components by the I/Q demodulator, which uses a *coarsely adjusted* frequency synthesizer set to ω_r in Fig. 7.3. In order to understand what *coarse adjustment* here means, consider for instance the case of a digital satellite receiver, where you have multiple available channels, each with a different central frequency. So what you first do is to set the local oscillator (a.k.a. frequency synthesizer) to approximately the center frequency ω_r of the channel you are interested in, thus after performing I/Q demodulation the transmission in this channel will have been converted to baseband.

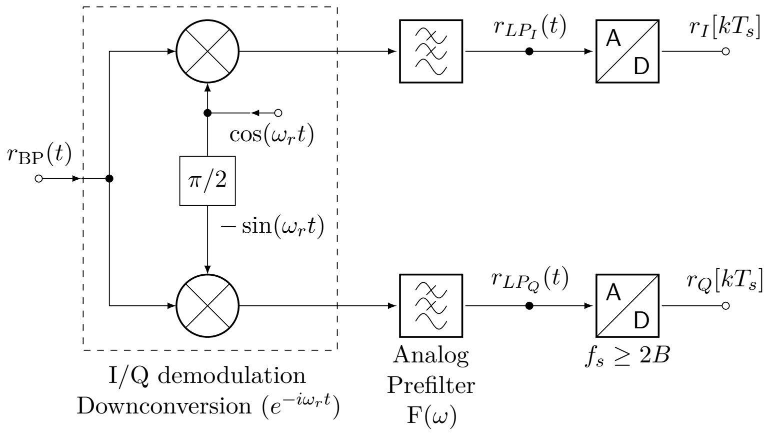


Figure 7.3: I/Q demodulation of an analog bandpass signal.

Analogously, in cellular communications there is a fixed frequency range that is allocated by regulatory entities for commercial use. The specific frequency channel used by a mobile network operator depends nonetheless on where the user is currently located, since radio resource allocation is conditioned by several factors such as population density, cell coverage range, building/vegetation conditions, among others. The eNodeB or base station is ultimately responsible for deciding which frequency channel to use, while the user equipment (UE) or handset monitors a set of pre-configured channels available in its SIM card, looking for active transmissions. Here again, the frequency synthesizer at the UE is *coarsely* set to the center frequency ω_r of the channels of interest, so that at the end the correct bandpass channel is directly converted into baseband (homodyne detection). This, however, implies that any kind of phase offset or Doppler effect caused by the channel is not corrected and will need to be estimated and compensated in the digital domain before attempting doing a detection.

7.1.6 Analog Pre-filter $F(\omega)$

The analog pre-filter $F(\omega)$ fulfills two main purposes. First, the mixer at the I/Q demodulator produces image frequencies and spurious signals as unavoidable by-products of the multiplication operation in the analog domain, which can introduce additional interference if not eliminated. This analog low-pass filter (LPF) placed after the demodulator is therefore responsible for filtering out such unwanted signals, in particular the double carrier frequency component ($2\omega_r$) and the out-of-band noise. Thus it feeds the ADC solely with the wanted baseband signal.

And second, given that the next operation involves sampling, which in the frequency domain translates into a periodic repetition of the spectral density function of the signal for intervals multiple of the sampling frequency f_s, this LPF additionally acts as an anti-alising filter that performs adjacent-channel interference (ACI) suppression as long as the bandwidth of the input signal is smaller or equal than the Nyquist filter, which is bound by $^1/_{2T_s}$, as depicted in Fig. 7.4

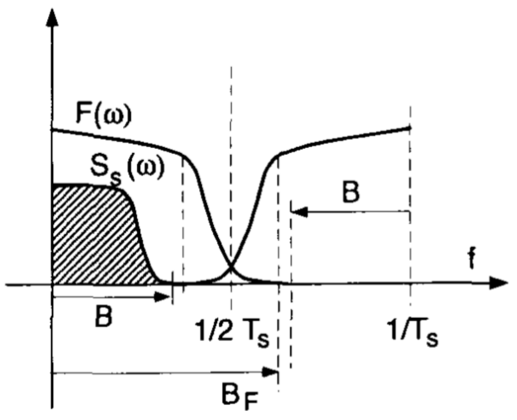


Figure 7.4: Analog prefilter $F(\omega)$ design for *sufficient statistics*. From [3].

7.1.7 Analog-to-Digital Converter (ADC)

One of the reasons why we need to operate at baseband is the sampling rate required by the ADC, since the power consumption increases with the sampling rate. Consider, for instance, that we want to do bandpass sampling of a signal in the *ISM band* at around 2.4 GHz, and consider that we use a free running oscillator with a sampling rate $N/T_s = 10\,\text{GHz}$. The power consumption of an ADC operating under these conditions will be in the order of several Watts, thus producing a prohibitively large amount of heat that hinders any attempt of using it for mobile applications, such as the case for the smartphone manufacturing industry.

Given that the ADC uses a free-running oscillator with a sampling rate $f_s = 1/T_s$ that is fully independent of the symbol rate ($T_s \neq T$), the sampling is of course done at wrong points in time. But this is not a problem since there are methods to estimate and compensate the delay in digital signal processing, as long as the sampling rate f_s is higher than the symbol rate 1/T. So at the end we will end up with several samples per symbol. This will be discussed in more detail in Section 8, dedicated to symbol synchronization.

Keep in mind that all transmissions are band-limited, so as long as our analog pre-filter $F(\omega)$ has a wider bandwidth than the transmitted signal and the sampling rate at the ADC fulfills Nyquist's criterion, then the I/Q samples represent *sufficient statistics* [3], which means that we do not lose any information when converting the signal to the digital domain and all its analog features can be reconstructed from the I/Q samples.

7.2 Digital Signal Processing

In summary, up to this point the analog part has taken care of *coarsely* finding the channel of interest and converting it to baseband. Since for the subsequent mathematical analysis it is more convenient to express real signals using a complex notation, in digital signal processing we follow this approach. Now, remember that it is not possible to generate complex signals in the analog domain. This is the reason why the signal is divided into its in-phase $r_I(t)$ and quadrature $r_Q(t)$ components, so the digital processing can benefit from the complex representation

$$r(t) = r_I(t) + ir_Q(t).$$

7.2.1 Carrier Synchronization

Considering the inaccuracies introduced by the analog components at the RF front-end, in particular the *coarse adjustment* of the central frequency and the imprecision at the frequency synthesizer, the I/Q samples handled to the digital signal processor (DSP) still contain frequency and phase offsets that have not been corrected. Thus the first digital operation that needs to be performed is estimating such frequency and phase offsets in order to do a fine tuning by correcting their remaining error. Such process is called carrier synchronization or carrier recovery. Fig 7.5 shows a communication system where emphasis is given to the *coarsely* adjusted carrier frequency ω_r, and the frequency $\hat{\Omega}$ and phase $\hat{\theta}$ estimators at the receiver.

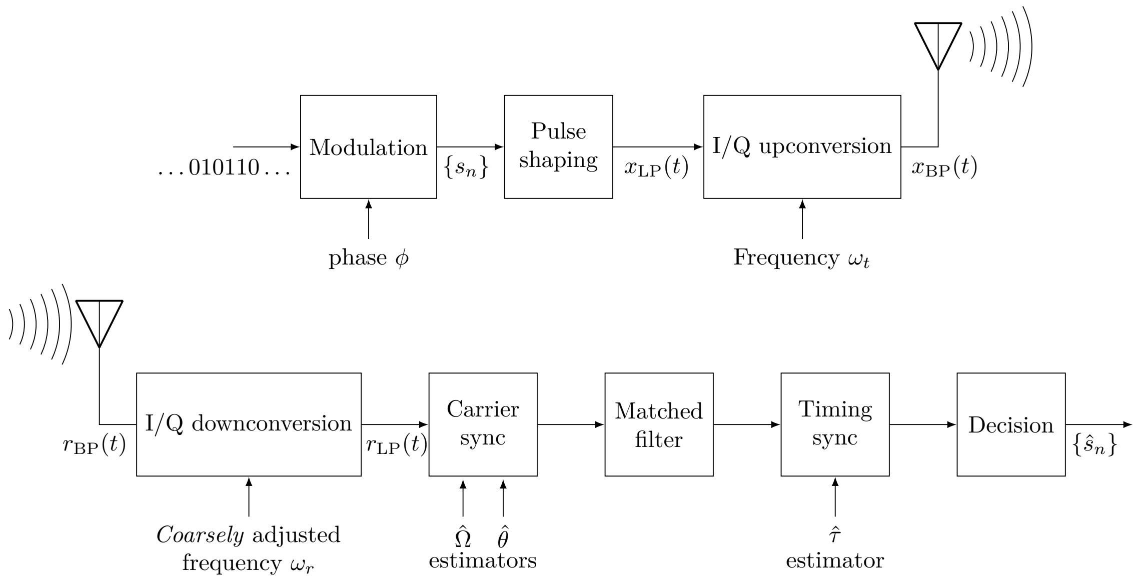


Figure 7.5: Block diagram showing the frequency and phase tracking done by the carrier synchronizer.

The inaccuracy in the analog components is not the only reason why we perform the carrier frequency correction in the digital domain though. Consider the case where the receiver is moving very fast, such as when traveling in a high-speed train or in the highway. Under these conditions the received signal will experience a significant Doppler effect and its central frequency will constantly change as a function of the vehicle's speed. In such a scenario, a finer time-variant tuning needs to be done using a feedback loop in the digital domain, by detecting the remaining carrier frequency error in real time and then using a frequency integrator together with a digital mixer to perform the correction, as shown in Fig. 7.1.

Note: A robust digital receiver design nowadays considers such mobile scenarios. That is why any deviation in the carrier frequency is normally used in a feedback loop for fine-tuning the frequency synthesizer at the RF front-end, as depicted in Fig. 7.1, thus reducing the carrier frequency offset of upcoming RF transmissions. The goal of this feedback loop, nonetheless, is not to perfectly match the local oscillator at the receiver with the one at the transmitter. The goal is instead just to keep the frequency error small, so that the remaining deviation can be more easily compensated in the digital domain. While the USRP-2901 allows implementing such feedback loop between digital and analog domains, our laboratory experiments have been designed for a static setup and aim to counter the effects of flat fading channels, where such feedback is not needed and therefore will not be implemented.

Another reason why the carrier synchronization is the first digital operation performed at the receiver is due to the matched filter. For the matched filter to operate correctly, the remaining frequency offset present in the received signal needs to be very small.

In order to understand the negative effects of not compensating these offsets, consider a system that uses amplitude-modulated (AM) signals to transmit information, represented as

$$x_{\mathrm{BP}}(t) = A(t)\cos(\omega_t t + \phi),$$

As we know, the channel will transform this waveform, introducing frequency and phase aberrations as well attenuation of the received signal depending on the surrounding environment and the receiver's conditions. Thus, the received signal can be represented as

$$r_{\mathrm{BP}}(t) = \tilde{A}(t)\cos(\tilde{\omega}_t t + \tilde{\phi}),$$

where $\tilde{\phi} = \phi + \triangle\phi$ is the phase of the received signal, and $\tilde{\omega}_t = \omega + \triangle\omega$ is the carrier frequency of the received signal. At the receiver, the downconverter will multiply this signal with the *coarsely* adjusted signal

$$c(t) = \cos(\omega_r t + \hat{\theta}),$$

in order to operate at baseband, similar to the procedure shown in Fig. 7.3; however, in AM systems no I/Q decomposition is done. The frequency ω_r is generated using a frequency synthesizer and is *coarsely* adjusted to the clock used at the transmitter. The downconverted signal that will result from this operation will be

$$r_{\mathrm{BP}}(t)\cdot c(t) = \frac{1}{2}\tilde{A}(t)\cos([\tilde{\omega}_t - \omega_r]t + [\tilde{\phi} - \hat{\theta}]) + \frac{1}{2}\tilde{A}(t)\cos([\tilde{\omega}_t + \omega_r]t + [\tilde{\phi} + \hat{\theta}]).$$

The low-pass filter also shown in Fig. 7.3 will suppress the high frequency component, leaving

$$r_{\mathrm{LP}}(t) = \frac{1}{2}\tilde{A}(t)\cos([\tilde{\omega}_t - \omega_r]t + [\tilde{\phi} - \hat{\theta}]).$$

Thus, the effect of the carrier offset is a reduction of signal level in voltage by a factor of

$$\cos([\tilde{\omega}_t - \omega_r]t + [\tilde{\phi} - \hat{\theta}]),$$

and in power by a factor of

$$\{\cos([\tilde{\omega}_t - \omega_r]t + [\tilde{\phi} - \hat{\theta}])\}^2.$$

This means that if not corrected, this will cause signal power loss, and the reduced SNR will increase the number wrong detections, thus increasing the bit error rate (BER).

Finally, also remember that frequency and phase are interrelated, since a fixed frequency offset $\Omega = \triangle\omega$ can be also represented as a linear change in the phase, expressed by

$$\triangle\omega = \frac{d\varphi}{dt}. \tag{7.1}$$

7.2.2 Carrier Synchronization Algorithms

The mathematical framework used to build a carrier recovery algorithm is called *estimation and detection theory*. Although there is a variety of methods for estimating the phase and the frequency of a transmission at the receiver, they can be generally classified in the three categories shown in Fig. 7.6, and described below:

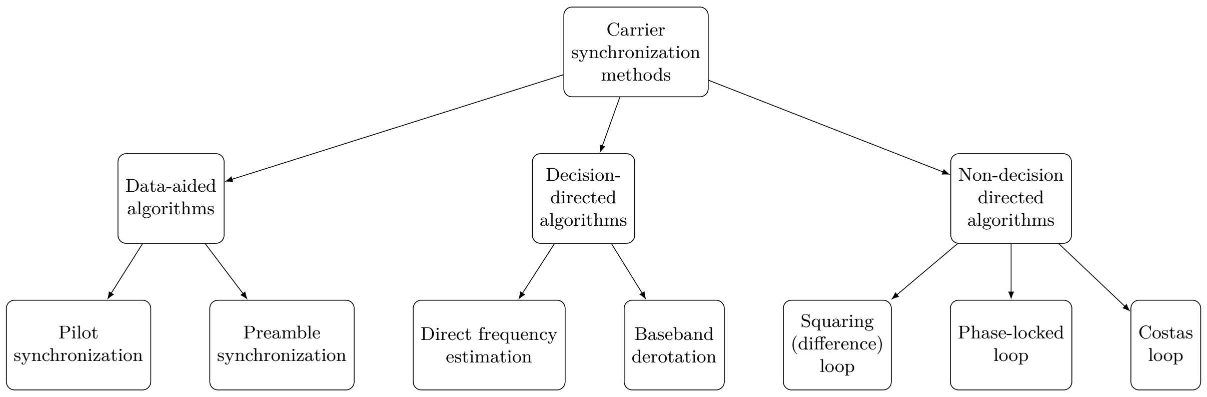

Carrier synchronization methods
Data-aided algorithms
Decision-directed algorithms
Non-decision directed algorithms
Pilot synchronization
Preamble synchronization
Direct frequency estimation
Baseband derotation
Squaring (difference) loop
Phase-locked loop
Costas loop

Figure 7.6: Overview of multiple existing methods for carrier offset estimation.

1. **Data-aided algorithms**, such as the pilot signal method, use dedicated signals such as the preamble or pilot signals as a training sequence for carrier synchronization. The main disadvantage of these methods is that part of the resources are used for signaling purposes, thus decreasing the performance. Furthermore, pilot signals normally add high-power peaks to the spectrum of the signal, which may cause data distortion.

2. **Non-decision directed algorithms**, such as the Costa's loop [4], do not use additional information on the received symbol constellations to obtain an estimation of the phase offset of the original symbols. On the contrary, they just use the incoming I/Q samples to identify aberrations on the received signal's phase.

3. **Decision-directed algorithms**, such as baseband derotation or the Moose algorithm, use knowledge on the positions of the constellation diagram to estimate the phase offset between transmitter and receiver, such as using a preamble with multiple repetitions of the same sequence. Payload symbols are ignored here.

In the next section, we will discuss one of these algorithms that will be implemented in this laboratory using LabVIEW.

7.2.3 Costa's Loop Algorithm (NDD) for PAM (BPSK)

This is a non-decision directed method that operates directly on the received signal $r_{\mathrm{BP}}[kT_s]$ and whose main goal is to maximize the squared form of the downconverted signal $r_{\mathrm{LP}}[kT_s]$ after it has been passed through a low-pass filter. Thus, under this scheme the received signal sampled at passband

$$r_{\mathrm{BP}}[kT_s] = r_{\mathrm{LP}}[kT_s] \cdot \cos(\omega_t kT_s + \phi), \tag{7.2}$$

is I/Q downconverted to baseband using an estimated phase offset θ, then lowpass filtered and finally squared. The value of θ will be adjusted after each iteration until it eventually converges to a constant term.

The aforementioned operations provide the objective function $J_C(\theta)$ that we want to optimize

$$J_C(\theta) = \mathrm{avg}\{(\mathrm{LPF}\{r_{\mathrm{BP}}[kT_s] \cdot \cos(\omega_r kT_s + \theta)\})^2\}.$$

This can be rewritten as

$$J_C(\theta) = \mathrm{avg}\{(\mathrm{LPF}\{r_{\mathrm{LP}}[kT_s] \cdot \cos(\omega_t kT_s + \phi) \cdot \cos(\omega_r kT_s + \theta)\})^2\}.$$

In order to understand its operation, assume that the *coarsely* adjusted frequency synthesis at the receiver leaves a negligible remaining frequency offset error, i.e., $\omega_t \approx \omega_r \approx \omega_0$ and $\omega_t - \omega_r \approx 0$.

Furthermore, assume that the cutoff frequency of the LPF filter is larger than the bandwidth of the transmitted signal $r_{\text{LP}}[kT_s]$, so that

$$\text{LPF}\{r_{\text{LP}}[kT_s]\} = r_{\text{LP}}[kT_s].$$

Then, by using trigonometrical equivalences, we obtain

$$\begin{aligned}
J_C(\theta) &= \text{avg}\{(\text{LPF}\{\frac{1}{2} \cdot r_{\text{LP}}[kT_s] \cdot [\cos((\omega_t - \omega_r)kT_s + \phi - \theta) + \cos((\omega_t + \omega_r)kT_s + \phi + \theta)]\})^2\} \\
&\approx \frac{1}{4}\text{avg}\{(\text{LPF}\{r_{\text{LP}}[kT_s] \cdot [\cos(\phi - \theta) + \cos(2\omega_0 kT_s + \phi + \theta)]\})^2\} \\
&\approx \frac{1}{4}\text{avg}\{(\text{LPF}\{r_{\text{LP}}[kT_s]\} \cdot \cos(\phi - \theta))^2\} \\
&\approx \frac{1}{4}\text{avg}\{r^2_{\text{LP}}[kT_s] \cdot \cos^2(\phi - \theta)\} \\
&\approx \frac{1}{4}\overline{r}^2_{\text{LP}} \cdot \cos^2(\phi - \theta),
\end{aligned}$$

where $\overline{r}^2_{\text{LP}}$ is the (constant) expected value of the square of the received signal $r_{\text{LP}}[kT_s]$. So at the end, the optimization function $J_C(\theta)$ will be proportional to the term $\cos^2(\phi - \theta)$.

This performance function $J_C(\theta)$ for a Costa's loop algorithm with an "unknown" phase offset of $\phi_0 = -0.8$ rad is shown in Fig. 7.7, where it can be observed that the function is maximized when the estimation is perfect, i.e. $\theta = \phi_0$. Since this is a periodic function, other maxima will be also found at $\theta = \phi_0 \pm n\pi$, $\forall n \in \mathbb{N}$.

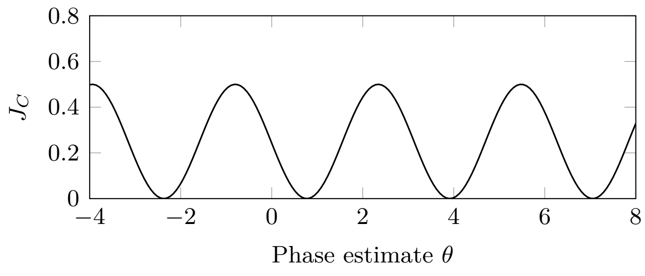


Figure 7.7: Graphical representation of the Costa's loop objective function $J_C(\theta)$ for a phase offset $\phi_0 = -0.8$ rad. $J_C(\theta)$ reaches maximum values at phase estimation values $\theta = -0.8 \pm n\pi$.

Note: The reason why in Fig. 7.7 the maximum value of $J_C(\theta)$ is 1/2 is that by convention the constant $\overline{r}^2_{\text{LP}}/2$ is set to unity, as shown next.
By using the identity $2\cos^2(x) = 1 + \cos(2x)$, the squared expression of (7.2) can be rewritten as

$$r^2_{\text{BP}}[kT_s] = \frac{1}{2}r^2_{\text{LP}}[kT_s] \cdot \{1 + 2\cos(2\omega_t kT_s + 2\phi)\}, \tag{7.3}$$

where $r^2_{\text{LP}}[kT_s]$ can additionally be written as the sum of its positive average value $\overline{r}^2_{\text{LP}}$ and the stochastic variation around this average $v[kT_s] = \text{var}\{r^2_{\text{LP}}[kT_s]\}$

$$r^2_{\text{LP}}[kT_s] = \overline{r}^2_{\text{LP}} + v[kT_s]. \tag{7.4}$$

Replacing (7.4) in (7.3) yields

$$r^2_{\text{BP}}[kT_s] = \frac{1}{2}\{\overline{r}^2_{\text{LP}} + v[kT_s] + \overline{r}^2_{\text{LP}} \cdot \cos(2\omega_t kT_s + 2\phi) + v(kT_s) \cdot \cos(2\omega_t kT_s + 2\phi)\}, \tag{7.5}$$

which after using a narrow bandpass filter centered at near $2\omega_t$, the DC component, the (presumably) lowpass component $v[kT_s]$, and its upconverted signal are suppressed, leaving the approximate expression

$$\tilde{r}^2_{\mathrm{BP}}[kT_s] = \mathrm{BPF}\{r^2_{\mathrm{BP}}[kT_s]\} \approx \frac{1}{2}\bar{r}^2_{\mathrm{LP}} \cdot \cos(2\omega_t kT_s + 2\phi + \psi), \tag{7.6}$$

where ψ is the additional phase shift introduced by the bandpass filtering at frequency $2\omega_t$. Since this function $\tilde{r}^2_{\mathrm{BP}}[kT_s]$ is used by several other optimization approaches such as *CFO estimation via FFT*, *square difference loop* and *phase-locked loop*, for convention its amplitude is set to unity, i.e.,

$$\frac{1}{2}\bar{r}^2_{\mathrm{LP}} = 1, \tag{7.7}$$

which does not affect the operation in the Costa's loop method since the tuning of the algorithm is controlled by μ.

For *implementing* the Costa's loop one approach consists in considering it as a standard adaptive element. The derivative of the objective function, $\frac{dJ_C(\theta)}{d\theta}$ can be thus approximated by performing the derivative before the averaging, using the chain rule, and then swapping the differentiation with the lowpass filtering operation, as in

$$\begin{aligned}
J_C(\theta) &= \mathrm{avg}\{(\mathrm{LPF}\{r_{\mathrm{BP}}[kT_s] \cdot \cos(\omega_r kT_s + \theta)\})^2\} \\
\frac{dJ_C(\theta)}{d\theta} &\approx \mathrm{avg}\{\frac{d\mathrm{LPF}\{r_{\mathrm{BP}}[kT_s] \cdot \cos(\omega_r kT_s + \theta)\}^2}{d\theta}\} \\
&= 2 \cdot \mathrm{avg}\{\mathrm{LPF}\{r_{\mathrm{BP}}[kT_s] \cdot \cos(\omega_r kT_s + \theta)\} \cdot \frac{d\mathrm{LPF}\{r_{\mathrm{BP}}[kT_s] \cdot \cos(\omega_r kT_s + \theta)\}}{d\theta}\} \\
&\approx 2 \cdot \mathrm{avg}\{\mathrm{LPF}\{r_{\mathrm{BP}}[kT_s] \cdot \cos(\omega_r kT_s + \theta)\} \cdot \mathrm{LPF}\frac{\{dr_{\mathrm{BP}}[kT_s] \cdot \cos(\omega_r kT_s + \theta)\}}{d\theta}\} \\
&= -2 \cdot \mathrm{avg}\{\mathrm{LPF}\{r_{\mathrm{BP}}[kT_s] \cdot \cos(\omega_r kT_s + \theta)\} \cdot \mathrm{LPF}\{r_{\mathrm{BP}}[kT_s] \cdot \sin(\omega_r kT_s + \theta)\}\}.
\end{aligned}$$

Accordingly, an *implementable version of the Costa's loop* can be recursively built as

$$\theta[k+1] = \theta[k] + \mu \left.\frac{dJ_C(\theta)}{d\theta}\right|_{\theta=\theta[k]}. \tag{7.8}$$

Replacing the previously obtained expression of the derivative in this equation we obtain

$$\theta[k+1] = \theta[k] - \mu \cdot \mathrm{avg}\{\mathrm{LPF}\{r_{\mathrm{BP}}[kT_s] \cdot \cos(\omega_r kT_s + \theta)\} \cdot \mathrm{LPF}\{r_{\mathrm{BP}}[kT_s] \cdot \sin(\omega_r kT_s + \theta)\}\}.$$

The averaging operation avg$(\cdot)$ can be left off since it is redundant in such small-stepsize update operation, without modifying its nature, leaving the algorithm

$$\theta[k+1] = \theta[k] - \mu \cdot \mathrm{LPF}\{r_{\mathrm{BP}}[kT_s] \cdot \cos(\omega_r kT_s + \theta)\} \cdot \mathrm{LPF}\{r_{\mathrm{BP}}[kT_s] \cdot \sin(\omega_r kT_s + \theta)\}, \tag{7.9}$$

which is graphically shown in Fig. 7.8. Here, the upper path modulates the bandpass signal by a cosine function and then lowpass filters it, while the lower path uses a sine wave for the modulation. The signals are then combined using a mixer, providing the equation update which is integrated using the discrete operator $\Sigma(\cdot)$ to provide a new phase estimation value, that is finally fed back into the oscillators in a recursive manner.

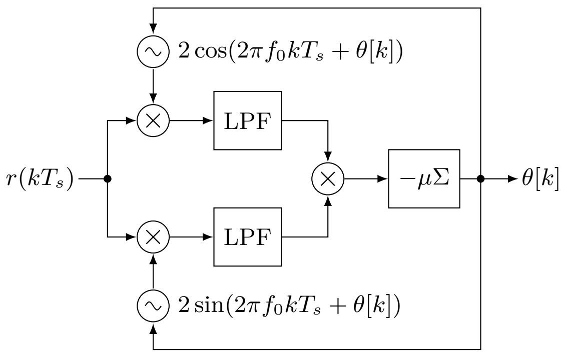

$2\cos(2\pi f_0 kT_s + \theta[k])$
LPF
$r(kT_s)$
$-\mu\Sigma$
$\theta[k]$
LPF
$2\sin(2\pi f_0 kT_s + \theta[k])$

Figure 7.8: Block diagram of a Costa's loop carrier synchronization algorithm.

The expected behavior of the estimated phase offset $\theta[k]$ over time is given in Fig. 7.9, where it is shown how the phase offset estimation will converge to $\phi \pm n\pi$, $\forall n \in \mathbb{N}$ for the case where the carrier frequency is known at the receiver.

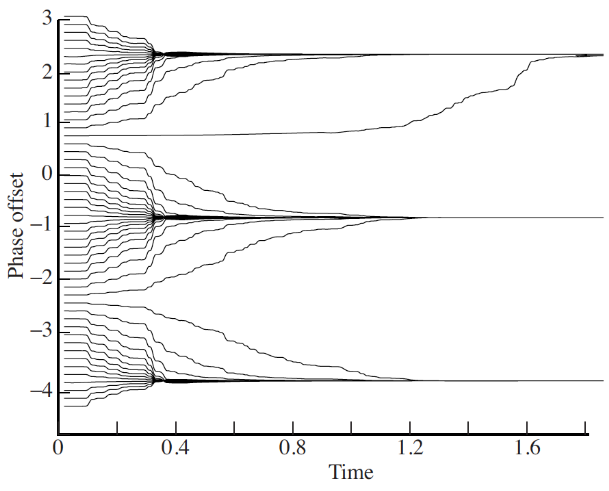


Figure 7.9: Plot showing 50 simulations with different initialization values $\theta[k=0]$ each, and for $\phi = -0.8$ rad. $\theta[k]$'s actual convergence value and pattern depends on where it was initialized; however, it will definitely converge to $\phi \pm n\pi$, $\forall n \in \mathbb{N}$. From [5].

In case the carrier frequency used by the transmitter is not exactly known at the receiver, i.e. $\omega_r \neq \omega_t$, then there will be a fixed frequency offset in the downconverted signal. This can be interpreted as a linear change of the phase offset, as previously discussed in Section 7.2.1. In such case, the Costa's loop algorithm tries to follow the linear change in the phase, as shown in Fig. 7.10, where a simulation shows how the Costa's loop algorithm is used for performing frequency offset Ω estimation, when $\omega_t = 2000\pi$ and $\omega_r = 2000.2\pi$ for 50 simulations with different initialization values $\theta[0]$.

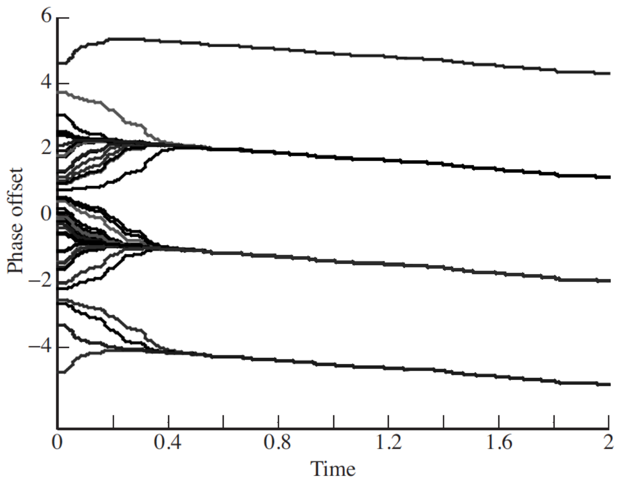


Figure 7.10: Plot showing 50 simulations with different initialization values $\theta[k=0]$ each, for the case of a frequency offset of $\Delta\omega = 0.2\pi$. The plots illustrate how the estimated phase $\theta[k]$'s follows a linear function that has as slight negative slope, which points out to a constant frequency offset. For this simulation, the values $\omega_t = 2000\pi$ and $\omega_r = 2000.2\pi$ were considered. From [5].

7.2.4 Costa's Loop Algorithm (NDD) for QPSK

The Costa's loop examined in the previous section is designed for phase offsets of an integer multiple of π, such as PAM or BPSK modulation schemes. For the case of QPSK, however, a modification of the Costa's loop algorithm needs to be done in order to have a stable convergence to multiples of $\frac{\pi}{2}$, i.e., to match the QPSK constellation diagram.

For such case, the new objective function will be defined as

$$\begin{aligned} J_{CQ}(\theta) &= \cos^2(2\phi - 2\theta) \\ &= \frac{1}{2}(1 + cos(4\phi - 4\theta), \end{aligned} \tag{7.10}$$

where maxima are reached when $\theta = \phi + \frac{\pi}{2}$.

The adaptive element will be constructed analogously to the definition given in Eq. 7.8, and reproduced here

$$\theta[k+1] = \theta[k] + \mu \left. \frac{dJ_C(\theta)}{d\theta} \right|_{\theta=\theta[k]}. \tag{7.11}$$

But now the gradient of the objective function $J_{CQ}(\theta)$ will be instead defined as

$$\frac{dJ_{CQ}(\theta)}{d\theta} = 2 \cdot sin(4\phi - 4\theta). \tag{7.12}$$

In terms of its implementation, a similar approach to Eq. 7.9 is used, where the gradient for the QPSK case is expressed as the multiplication of four lowpass-filtered signals, each with a phase offset of $\pi/4$.

Thus, for a QPSK signal such as

$$r_{\text{QPSK}}(t) = m_1(t) \cdot cos(\omega_c t + \phi) - m_2(t) \cdot sin(\omega_c t + \phi), \tag{7.13}$$

we define four functions:

$$\begin{aligned} x_1(t) &= \text{LPF}\{r(t) \cdot cos(\omega_0 t + \theta\} \\ x_2(t) &= \text{LPF}\{r(t) \cdot cos(\omega_0 t + \theta + \frac{\pi}{4})\} \\ x_3(t) &= \text{LPF}\{r(t) \cdot cos(\omega_0 t + \theta + \frac{\pi}{2})\} \\ x_4(t) &= \text{LPF}\{r(t) \cdot cos(\omega_0 t + \theta + \frac{3\pi}{4})\} \end{aligned} \tag{7.14}$$

whose multiplication yields

$$x_1(t) \cdot x_2(t) \cdot x_3(t) \cdot x_4(t) = 4\nu^2 sin(4\phi - 4\theta) \propto \frac{dJ_{CQ}(\theta)}{d\theta}. \tag{7.15}$$

The block diagram of such implementation is shown in Fig. 7.11, as well as the expected results based on simulations.

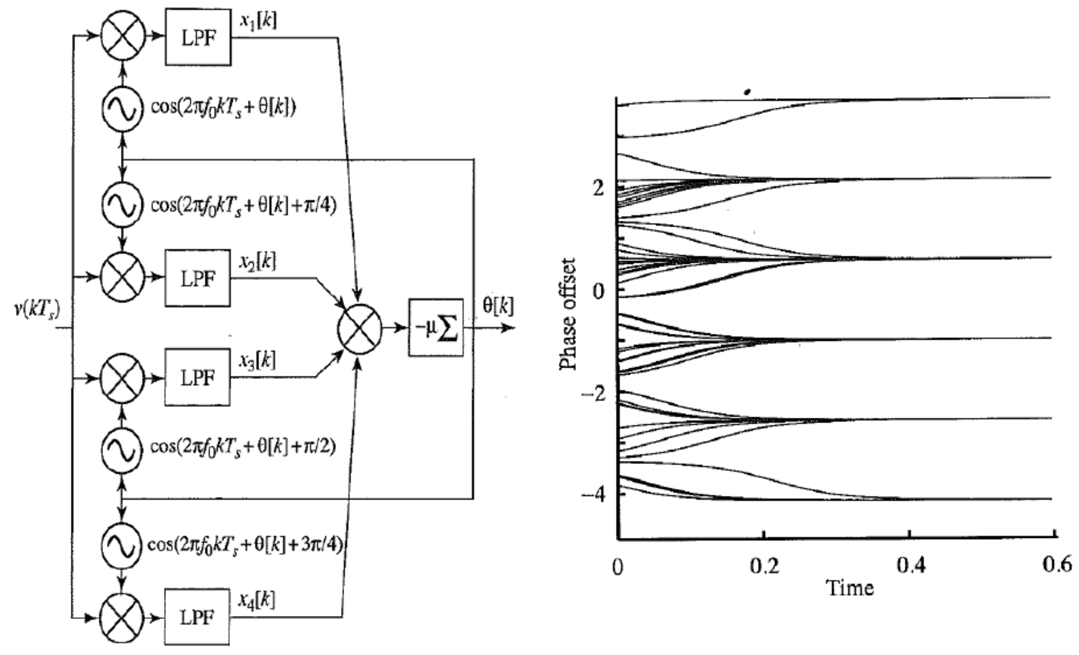

LPF
$x_1[k]$
$\cos(2\pi f_0 kT_s + \theta[k])$
$\cos(2\pi f_0 kT_s + \theta[k] + \pi/4)$
$x_2[k]$
$v(kT_s)$
$-\mu\Sigma$
$\theta[k]$
$x_3[k]$
$\cos(2\pi f_0 kT_s + \theta[k] + \pi/2)$
$\cos(2\pi f_0 kT_s + \theta[k] + 3\pi/4)$
$x_4[k]$
Phase offset
2
0
−2
−4
0
0.2
0.4
0.6
Time

Figure 7.11: Plot showing the block diagram that describes the implementation of the modified Costa's loop algorithm for QPSK (left); and multiple simulations with different initialization values $\theta[k = 0]$ each that converge to a constant value for the case where no frequency offset is present in the received signal (right). From [x].

7.2.5 Baseband Derotation (DD)

In Section 7.2.4 we studied the Costa's loop algorithm for carrier synchronization at *sampling rate*, i.e., before the symbol synchronization had performed the decimation of the oversampled symbols that we will discuss in Section 8. While the narrow bandwidth selected for the single-carrier transmission system studied in this laboratory limits the number of received samples to an amount that is manageable by the generic CPU used for executing such algorithm, when considering systems with wider bandwidth, performing signal processing at *sampling rate* will exert a higher load onto the available computing resources that may end up hindering a real-time operation of the system. Therefore in this section and in Section 7.2.6 we will study two new methods to perform carrier synchronization at *baseband*, thus decreasing the processing load at the receiver.

The objective of this section will be therefore to implement the *baseband derotation algorithm*, which operates at the symbol level, i.e., after symbol synchronization. This decision-directed method takes as input the modulation scheme used in order to know in advance the positions in the constellation diagram where received symbols are expected to be found, and then "inverts the rotation" experienced by such symbols in order to bring them back to one of the appropriate constellation points, thus countering the effect caused by phase and frequency offsets.

The approach that we will follow for implementing our carrier recovery algorithm will employ the principle of adapting elements, where we define the objective function that we want to minimize $J_{BD}(\theta[k])$ as the distance between the received symbol that has already been compensated by the estimated phase offset $\theta[k]$, and the expected symbol positions. Thus, $J_{BD}(\theta[k])$ is defined as

$$J_{BD}(\theta[k]) = \frac{1}{2}(r_{\text{LP}}[k] \cdot e^{i\theta[k]} - s_l) \cdot (r_{\text{LP}}[k] \cdot e^{i\theta[k]} - s_l)^*, \tag{7.16}$$

where $s_l = s_1^l + is_2^l$ represents an element of the set of expected locations in the constellation diagram that will have to be replaced in (7.16) before finding the one that minimizes $J_{BD}^{min}(\theta[k])$. The set $s = \{s_l\}$ is defined according to the modulation scheme used in the communication. That is, for QPSK,

$$s = \frac{1}{\sqrt{2}} \cdot \{1+i,\ -1+i,\ -1-i,\ 1-i\}. \tag{7.17}$$

The value $\tilde{s}[k]$ that minimizes the distance between symbols $J_{BD}(\theta[k])$, expressed as

$$\tilde{s}[k] = \frac{1}{2} \arg\min_{s} \left\{(r_{\text{LP}}[k] \cdot e^{i\theta[k]} - s) \cdot (r_{\text{LP}}[k] \cdot e^{i\theta[k]} - s)^*\right\}, \tag{7.18}$$

will be employed in the adaptive element of the carrier synchronization algorithm, which uses a modified version of the recursive formula given in Eq. 7.8 for the case of baseband derotation, defined as

$$\theta[k+1] = \theta[k] - \mu(\tilde{s}_1[k] \cdot x_2[k] - \tilde{s}_2[k] \cdot x_1[k]), \tag{7.19}$$

where $x[k] = x_1[k] + i \cdot x_2[k]$ is defined as the compensated incoming signal, i.e.,

$$x[k] = r_{\text{LP}}[k] \cdot e^{i\theta[k]}. \tag{7.20}$$

For the implementation take into consideration that the adaptive element can be also expressed as

$$\text{Im}\{\tilde{s}^*[k] \cdot x[k]\} = \tilde{s}_1[k] \cdot x_2[k] - \tilde{s}_2[k] \cdot x_1[k]. \tag{7.21}$$

Fine tuning of the baseband derotation algorithm is done by finding an appropriate value of μ in the range $[-1, 1]$.

7.2.6 Moose Algorithm (DD)

A second decision-directed carrier synchronization algorithm that also operates at *symbol level* is called Moose algorithm, which in contrast to the baseband derotation algorithm does not use the constellation diagram for performing an estimation of the phase offset, but the preamble of the frame instead.

As we have previously discussed in Section 7.2.1, the phase and frequency offsets can be represented as a linear change in the phase over time with a constant slope (Eq. 7.1). This feature, together with the repetitive nature of the preamble used in our experiments, will be exploited by the Moose algorithm in order to track the relative phase rotation of equally spaced symbols extracted from the preamble.

Considering an even number of repetitions of the preamble P, where each preamble sequence has a length M, then the overall accumulated length of the appended preambles will be $N = M \cdot P$. The estimation of the relative phase offset can be then done by dividing the overall preamble in two parts, each of length $N/2$, and estimating the average relative phase rotation between consecutive preamble symbols. Thus, this estimation can be calculated by solving

$$\triangle\hat{\theta} = \frac{1}{N/2}\sum_{n=0}^{N/2-1} \arg(p[n]) - \arg(p[n+N/2]), \tag{7.22}$$

where $p[n], n = 0, \ldots, N/2$ represents preamble symbols.

It is important to notice, however, that the frequency offset may have caused a phase shift between the symbols $p[n]$ and $p[n+N/2]$ that is a multiple of 2π, making the constellation diagram rotate n times. Thus, as part of this task a method for identifying such $2\pi n$ rotations needs to be implemented before the carrier compensation is done, i.e., the second term in the following expression obtained from (7.22)

$$\triangle\hat{\theta} = \triangle\hat{\theta}_0 \pm (2\pi l) \tag{7.23}$$

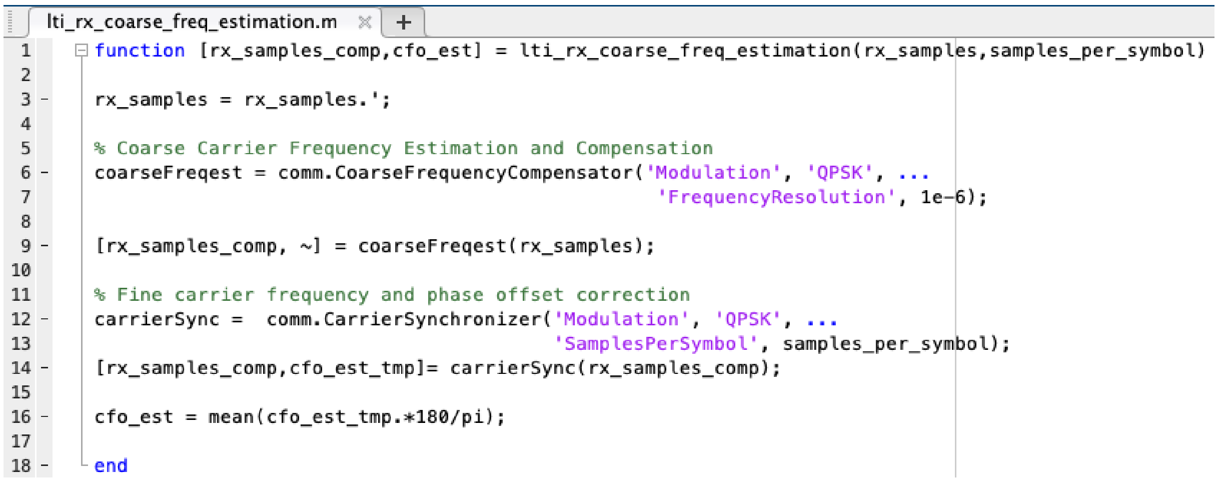
must be eliminated by obtaining $l = \lfloor \frac{\triangle\hat{\theta}}{2\pi} \rfloor$ and then substracting $2\pi l$ from $\triangle\hat{\theta}$ in order to obtain the fundamental carrier offset $\triangle\hat{\theta}_0$.

Once the value $\triangle\hat{\theta}_0$ has been estimated, it will be used in an accumulative manner to compensate the phase offset still present in the received symbol stream.

7.3 Pre-Lab $\boxed{4}$

Your protocol should include answers to the following questions.

1. Elaborate on the effects that the phenomena listed below will have on the constellation diagram of the received signal. Consider the case of QPSK, and show the effects graphically.
   a) Noise
   b) A constant phase offset at the receiver
   c) A constant frequency offset at the receiver
2. Analogous to the analysis presented in Section 7.2.1, show the effect of carrier frequency and phase errors when the modulation scheme used is QPSK (i.e., when I/Q demodulation is used).
3. Name a system that uses pilot signals for carrier synchronization (data-aided algorithm).
4. Indicate under which category the Schmidl-Cox synchronization algorithm can be classified, and by which commercial technology is used.
5. The present laboratory provides the student with the MATLAB script shown in Fig. 7.12 to test the end-to-end system. Elaborate on the methods used by each of the two functions contained in the script.
6. The Costa's loop discussed in this session is used for QPSK. Explain why the same approach cannot be so easily implemented for 16-QAM.
7. Derive an analytical expression for the upper bound for the frequency offset Ω (in radians per sample) as a function of the parameter μ, above which the Costa's loop does not work anymore.



```matlab
function [rx_samples_comp,cfo_est] = lti_rx_coarse_freq_estimation(rx_samples,samples_per_symbol)

rx_samples = rx_samples.';

% Coarse Carrier Frequency Estimation and Compensation
coarseFreqest = comm.CoarseFrequencyCompensator('Modulation', 'QPSK', ...
                                                'FrequencyResolution', 1e-6);

[rx_samples_comp, ~] = coarseFreqest(rx_samples);

% Fine carrier frequency and phase offset correction
carrierSync =  comm.CarrierSynchronizer('Modulation', 'QPSK', ...
                                       'SamplesPerSymbol', samples_per_symbol);
[rx_samples_comp,cfo_est_tmp]= carrierSync(rx_samples_comp);

cfo_est = mean(cfo_est_tmp.*180/pi);

end
```

Figure 7.12: MATLAB implementation of carrier synchronization.

7.4 Lab Experiment $\boxed{6}$

The laboratory experiment consists in implementing the three carrier synchronization algorithms discussed in this section, namely the Costa's loop algorithm for QPSK, baseband derotation, and the Moose algorithm using LabVIEW, that will increase your knowledge on the broad selection of different algorithm implementations.This will prove to be useful in the future when designing new communication systems, where the algorithm choice is normally based on the system's requirements.

For this an end-to-end communication framework also in LabVIEW will be provided, which controls the operation of the transmitting and receiving USRPs. Your task consists in replacing the given algorithm implemented in MATLAB by your own implementation in LabVIEW.

Please take the following information into account when designing your algorithm.

- Directly sampling the received bandpass signal to the digital domain ($r_{\mathrm{BP}}[kT_s]$) is normally not done, since it would require an ADC with a high sampling rate that will introduce computational overhead. Instead, the analog bandpass signal $r_{\mathrm{BP}}(t)$ is directly downconverted to baseband $r_{\mathrm{BP}}(t)$ by the software-defined radio, before being sampled and quantized to obtain the digital signal $r_{\mathrm{LP}}[kT_s]$. Given that nowadays carrier recovery algorithms operate in the digital domain, they also normally operate at baseband.
- Since the operation is done at baseband, the downconversion discussed in Section 7.2.3 is not needed in your implementation.
- In this laboratory you will implement a Costa's loop algorithm for estimating and compensating the phase offset of a QPSK transmission using solely the digital baseband received signal $r_{\mathrm{LP}}[kT_s]$ as input to your implementation.
- The implementation of a Costa's loop carrier synchronizer follows the principle of adaptive elements using the set of equations given in Section 7.2.3, where a predefined objective function is maximized by tracking its gradient.
- The volability of the loop is controlled by a parameter μ, which should be tuned based on its performance regarding the estimation. This input to your system should be defined as a constant value. Normally, $\mu \in [-1, 1], \mu \in \mathbb{R}$.
- The estimator should be initialized with $\theta[k = 0] = 0$, i.e., assuming that there is no phase offset error at the receiver. This value of $\theta[k]$ is then expected to converge to the actual phase offset ϕ.

- Test the sign of your parameter μ. Depending on the way you implement it, your control parameter μ may be positive or negative.
- Plot the results in the *Front Panel* of LabVIEW, principally the objective function $J_C(\theta)$ and $\theta[k]$, and compare it against the expected behavior depicted in Fig. 7.7, Fig. 7.9, and Fig. 7.10.

8 Symbol Synchronization (Day 4)

In this section we will discuss the effects that flat-fading wireless channels and analog components at the RF frontends have on the received signal. Our focus will now lie on the time domain, more specifically, on the propagation and processing delays introduced by these elements, respectively. In an asynchronous communication system such as the one introduced in Section 7, the receiver is responsible of estimating such delays in order to compensate them. Failing to do so will result in I/Q samples taken at arbitrary points in time where adjacent symbols will introduce higher interference levels (ISI), thus decreasing the symbol energy (i.e., lowering the E_b/N_0), eventually affecting the correct demodulation of the signal and leading to a higher bit error rate (BER) in the overall communication.

The digital procedure responsible of this timing estimation and compensation is known by different names: symbol synchronization, clock recovery, timing recovery, or timing synchronization. At the end, however, the objective is the same: To provide a unique digital I/Q sample per symbol that has been sampled at the right point in time.

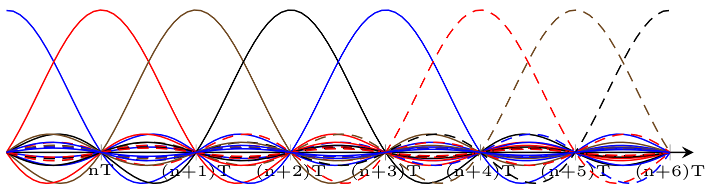
8.1 Background

Fig. 8.1 a) shows graphically how using the pulse shaping function sinc(·) at the transmitter would introduce a non-negligible amount of ISI among adjacent symbols. In contrast, pulse shaping filters that use a root-raised cosine function, such as the one shown in Fig. 8.1 b), exhibit a faster decay over time and therefore will have a lower impact on adjacent symbols. Although this is a design criterion that we discussed when studying the pulse shaper and matched filter, the inter-symbol interference plays an important role when discussing timing offsets. Consider for instance the case where consecutive symbols experience variable delays so that the ISI adds up constructively, thus decreasing the SNR of the received symbol.

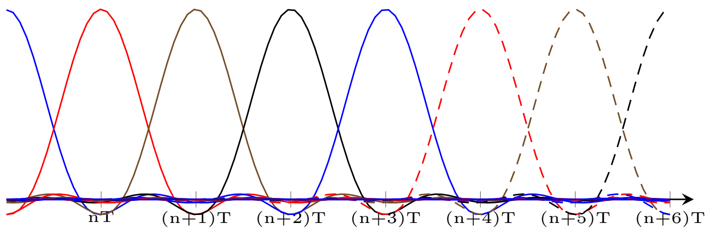
Given that the ultimate goal at the receiver is to recover the transmitted symbols, ideally it should suffice to sample each of them only once. Unfortunately, transmitted symbols undergo time-varying delays when traversing the channel. So, if a single-carrier transmitter sends them periodically every nT seconds (i.e., at a fixed data rate), at the receiver these symbols would need to be sampled at $(n + \varepsilon(t))T$, where $\varepsilon(t)T$ is the time-variant delay introduced by the system, as shown in Fig. 8.1 c).

Furthermore, even considering hypothetically that this delay is known in advance, the receiver would still require a high precision frequency synthesizer that samples the incoming signal at exactly the intended points in time and whose operation is excepted from experiencing clock drifts over time. These two conditions are not only hard to meet, but also end up increasing the manufacturing costs of such devices. Thus, from the hardware implementation perspective, it is unreliable to count on 100% accurate timing of the transmitted and received pulses since, as we have learned in Section 7, clocks at both ends operate with small levels of inaccuracy.

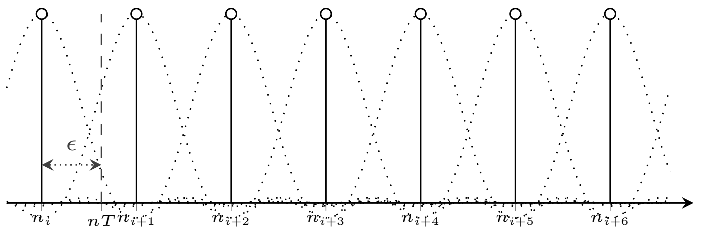
The modern approach to solve this problem consists in using a very loosely time-synchronized receiver that utilizes a free running oscillator to sample each incoming symbol multiple times (oversampling), so the timing offset can be digitally corrected by the *symbol synchronizer* afterwards.



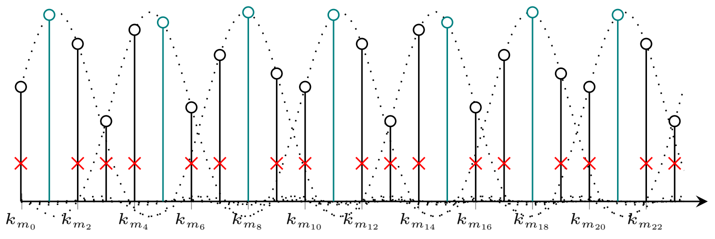
a) Inter-symbol interference when using $\text{sinc}(\frac{\pi}{T}t)$ for pulse shaping.



b) Improved ISI when using a root raised cosine filter with $\alpha = 0.9$



c) Delayed symbols ideally sampled at $\{\epsilon | \max_\epsilon \sum_{i=0}^{N-1} |r_{\text{LP}}^{MF}[n_i \cdot T + \epsilon]|^2\}$.



d) Actual sampling of delayed signal $r_{\text{LP}}^{MF}[k_m T_s]$, before decimation.

Figure 8.1: Main considerations taken into account when designing a timing synchronizer.

The main purpose of such synchronizer is to estimate the delay $\epsilon = \varepsilon(t)T$ indirectly. More than the delay itself, what the receiver is really interested in is to calculate the timing offset relative to its own samples taken at kT_s intervals, i.e., the *sampling phase* or *timing phase*, in order to sample the incoming time-shifted symbols at the point in time where its energy reaches a maximum level. A second parameter that is also estimated by the timing synchronizer, if not already known, is the symbol rate $(1/T)$, and with it the duration of each symbol T. Only once the symbols have been correctly resampled, the *decision* can be performed following the sequential procedures depicted in Fig. 7.5.

After identifying the symbol's sample with the highest energy, the receiver performs a *decimation* of the oversampled symbol by discarding those remaining samples that contain higher ISI, as graphically shown in Fig. 8.1 d), where the discarded samples have been marked with red crosses and the samples with the highest energy per symbol has been colored with green.

Normally an *interpolation* procedure follows in order to estimate the remaining *timing phase*. Although there are several methods for performing this operation, a system designer is normally concerned with the task of finding an adequate trade-off between algorithm complexity and the available computational resources when designing a communication system, and finding the *exact* timing phase is not always necessary. In the case of the implementation of the single carrier receiver done for this laboratory, the number of samples per symbol has been selected to make this error small enough, so that the use of an interpolator is not required for a correct detection. Therefore, the timing phase is assumed to be approximately a multiple of the sampling interval ($\approx m_k \cdot T_s$).

8.1.1 Timing Synchronization Algorithms

Similar to the classification of carrier synchronization algorithms, as shown in Fig. 7.6, timing synchronization algorithms are also classified in these three categories based on the information they use for carrying out their estimation.

1. **Data-aided algorithms**, such as *source recovery error*, rely on signals that are known at both the transmitter and receiver, such as pilot signals, training sequences or preambles, in order to make an estimation.

2. **Non-decision-directed algorithms**, such as *output power maximization* or *early-late gate*, derive the timing offset by extracting information from the received data itself, without relying on any additional source of information. They are also known as non-data-aided algorithms.

3. **Decision directed algorithms**, such as *cluster variance minimization*, use previously correctly detected symbols as reference values for estimating the timing offset of the next symbols.

In this section, we will focus on the non-decision directed *output power maximization* method, which uses solely the received signal $r_{\text{LP}}[kT_s]$ in order to derive the instants at which such signal needs to be resampled.

8.1.2 Output Power Maximization Algorithm

After the received symbol has been processed by the matched filter, its highest energy level is expected to be found at the zero-offset correlation point, i.e. the position where the pulse shape and its matched filter completely overlap and where the SNR is maximized. This property of the matched filter is exploited by the output power maximization algorithm for estimating the timing offset where the sample with the lowest ISI level has been taken.

Initially it is unknown which of the received matched filtered samples carries the transmitted symbol without ISI. Considering a receiver that has already compensated any phase offset, and not considering the effects of attenuation and noise in this analysis, the received sample train $r_{\text{LP}}^{MF}[n]$ at the output of the matched filter can be expressed as:

$$r_{\text{LP}}^{MF}[n] = \sqrt{E_x} \sum_i s[i] \cdot g[(n-i)T - \epsilon], \tag{8.1}$$

where E_x is the energy of a symbol, which for practical purposes is considered to be normalized; $s[n]$ is the original symbol sequence sent by the transmitter; ϵ is the unknown delay introduced by the channel that we want to estimate; T is the symbol duration; and the function $g(t)$ is the convolution of the pulse shaping's and matched filtering's impulse responses, i.e.,

$$g(t) = g_{tx}(t) * g_{rx}(t). \tag{8.2}$$

Assuming that the delay ϵ is shorter than the symbol duration, then (8.1) can be written as

$$r_{\text{LP}}^{MF}[n] = s[n] \cdot g(\epsilon) + \sum_{i \neq n} s[i] \cdot g[(n-i)T - \epsilon], \tag{8.3}$$

where the first term represents the wanted symbol with highest SNR, while the second term adds up all other samples that contain ISI. If we consider the case when $\epsilon = 0$, then due to the Nyquist criterion,

$$r_{\text{LP}}^{MF}[n] = s[n]. \tag{8.4}$$

In order to achieve this result, in which no ISI is present in the sample, a time shift $\hat{\varepsilon}$ is introduced, so that it counters the delay ϵ. In practice, this means that the initial samples before $\hat{\varepsilon}$ are discarded.

The approach used here for estimating the time shift $\hat{\varepsilon}$ takes advantage of the aforementioned property of the pulse shaper and the matched filter, namely that $g(t)$ has its maximum at $g(0)$ and then rapidly decays, as shown in Fig. 8.1 b).

Thus, the output power maximization algorithm defines the following optimization function for a finite train of N symbols, e.g., taken over time interval that a frame lasts,

$$O[\hat{\varepsilon}] = \frac{1}{N} \sum_{n=0}^{N-1} |r_{\text{LP}}^{MF}[n]|^2. \tag{8.5}$$

Introducing the variable $\hat{\varepsilon}$ that optimizes $O[\hat{\varepsilon}]$ in (8.3) and performing a time-shifting operation (compensation), we obtain

$$O[\hat{\varepsilon}] = \frac{1}{N} \sum_{n=0}^{N-1} \left| s[n] \cdot g(\epsilon - \hat{\varepsilon}) + \sum_{i \neq n} s[i] \cdot g[(n-i)T - \epsilon + \hat{\varepsilon}] \right|^2. \tag{8.6}$$

For $\hat{\varepsilon} = \epsilon$, $O[\hat{\varepsilon}]$ takes the value of the average symbol energy, which for matched filtered functions represents the maximum value.

In summary, our algorithm takes samples at the inter-peak distance of the received waveform and finds the timing offset that maximizes the power of the included symbols, i.e. it finds

$$\arg\max_{\hat{\varepsilon}} \sum_{i=0}^{N-1} |r_{\text{LP}}^{MF}[m \cdot i - \hat{\varepsilon}]|^2, \tag{8.7}$$

where N is the total number of symbols before pulse shaping, m is the number of samples per symbol, and $\hat{\varepsilon}$ is the estimated delay. As already mentioned, we assume that $\hat{\varepsilon}$ is a multiple of T_s.

8.1.3 Early-Late Gate Algorithm

This algorithm is slightly more difficult to implement than the *output power maximization* algorithm discussed in Section 8.1.2. In this case the symmetric shape of the signal at the output of the matched filter shown in Fig. 8.1 is exploited. The objective is then to find the minimum-ISI center value, where the difference of the complex-conjugated time-shifted signals

$$s^*[k + d + \epsilon] - s^*[k - d + \epsilon] \tag{8.8}$$

is weighted by the sample at $s[n + \epsilon]$ for all symbols. The real component of these complex multiplications are summed and the value of ϵ that maximized this sum represents the estimated timing offset that is then feed to the downsampler. For the correct estimation the value of d is chosen arbitrarily but sufficiently small, so that $d < m/2$, where m is the number of samples per symbol. For our implementation, d is a multiple of T_s.

Further detailed information on the implementation of the early-late gate algorithm can be found in [4], Section 6.3.2.

8.2 Pre-Lab $\boxed{4}$

Your protocol should include answers to the following questions.

1. What would be the effect on the eye diagram when the timing offset is not correctly estimated by the symbol synchronizer? Similarly, what would be its effect on the constellation demapper?

2. Express mathematically how a time-shifted signal $r(t-\epsilon)$ is represented in the frequency domain in terms of $\mathcal{F}\{r(t)\} = \hat{r}(\omega)$. What is the effect of such delay on the spectrum of the original signal?

3. In this section we have studied the negative effects that time delays have on single-carrier transmission systems. However, such delays can also affect more robust systems, such as Long Term Evolution (LTE) cellular networks where orthogonal frequency-division multiplexing (OFDM) signals are used for downlink transmissions. Indicate which impact will excessive delays have on OFDM signals.

4. When looking for the value $\hat{\varepsilon}$ that maximizes the average signal power, what is the expected interval to which $\hat{\varepsilon}$ is restricted to? What happens if the evaluation interval is selected a) smaller or b) larger than the actual one?

5. As mentioned in Section 8.1.1, early-late gate is another non-decision-directed algorithm that also exploits the matched filter function in its operation. Indicate which feature of such function is exploit, and briefly elaborate how the algorithm works.

8.3 Lab Experiment $\boxed{6}$

This laboratory session consists of two parts to be implemented using LabVIEW NXG 4.0. First, the timing offset will be estimated based on the incoming I/Q samples using the two algorithms studied in this section, i.e., *output power maximization* algorithm elaborated in Section 8.1.2, and the *early-late gate* algorithm discussed in Section 8.1.3. For the case of the *output power maximization* algorithm, the values of the optimization function given in (8.7) are calculated by summing over the signal powers of N samples, trying out different values of $\hat{\varepsilon}$. Once the maximum value of the optimization function is obtained, the value of the timing offset $\hat{\varepsilon}$ that has maximized this function is provided to the downsampler.

And second, the received signal will be downsampled (decimated) by a second LabVIEW VI. That is, the estimated timing offset will be employed to extract one I/Q sample per symbol, i.e., the one with the highest power, and all other samples of the same symbol will be discarded.

Please take into account the following information when designing your algorithm.

- The symbol duration T is assumed to be known by both transmitter and receiver, and the system operates at a constant data rate $1/T$. Thus, your algorithm only needs to take as inputs the value of T in terms of the number of samples per symbol, and the I/Q symbols themselves after having been matched filtered.

- It is assumed that the *timing phase* is a multiple of the sampling intervals T_s. Although this assumption will introduce a small error in the timing estimation, such error level is acceptable for the case of our single-carrier transmission system. This means that there is no need to implement an interpolator as part of your algorithm.

- The delay $\epsilon = \varepsilon(t)T$ is assumed to be constant over the duration of a frame. So, the timing offset correction that will be applied to all symbols in a frame is the same.

- For the second part, you can use the *Decimate VI* available in LabVIEW NXG to down-sample the signal.

9 Frame Synchronization (Day 5)

9.1 Introduction

The purpose of frame synchronization is to identify the start of the sent frame (or frames, if we send more at once), in order to properly apply further signal processing and then decode it. Thus the transmitter does not only send its message, but also a sequence of reference symbols, which we call pilot symbols or synchronization sequence, which aid the receiver in finding the correct start of the frame. At the receiver side, there should be a detector which finds the known reference in the transmitted signal so that subsequent receiver operations can proceed.

In the literature there are multiple approaches to solve the problem of frame synchronization, but in this lab we will consider one approach based on the correlation properties of the pilot symbols.

Consider a received signal passed through the channel as,

$$y\left[n\right] = Hs\left[n-d\right] + v\left[n\right], \tag{9.1}$$

where H is the channel coefficient (assumed to be constant) and d is the delay experienced by the signal, also known as frame offset.

Suppose that a single frame of data is transmitted and consists of N_{sync} pilot symbols $p[n]$ followed by N_{data} data symbols. Note that these pilot symbols are known both at the transmitter and receiver sides. These pilots are defined by the physical layer standard and are chosen to have good auto-correlation property, i.e. the auto-correlation function resembles a Dirac pulse as much as possible.

The main idea behind correlation-based frame synchronization is to first correlate the pilot symbols with the received samples and compute the modulus squared. Formally

$$\eta\left[n\right] = \left|\sum_{k=0}^{N_{\text{sync}}-1} p^{\star}\left[k\right] y\left[n+k\right]\right| \tag{9.2}$$

Then in a second step the estimated frame offset $\hat{d}$ is calculated as

$$\hat{d} = \arg\max_{n}\ \eta\left[n\right], \tag{9.3}$$

i.e., the time-index where the correlation is the strongest is the start of the frame.

Note that since the complex numbers do not have a well defined order relation, we need to take the modulus squared in order to have a real-valued objective in(9.3). Furthermore, one can interpret $\eta\left[n\right]$ as being some kind of energy of the auto-correlation.

9.1.1 Pilot Symbols

Pilot symbols consists of the symbols that are known both at the transmitter and receiver ends. They serve as known reference signals, which can be used for synchronization of the signal at the receiver end. In general, good pilot sequences for synchronization possess strong auto-correlation properties. Sequences that are used in research and commercial wireless systems are Barker sequences, Zadoff–Chu sequences, and Gold sequences, some of which will be used in today's lab.

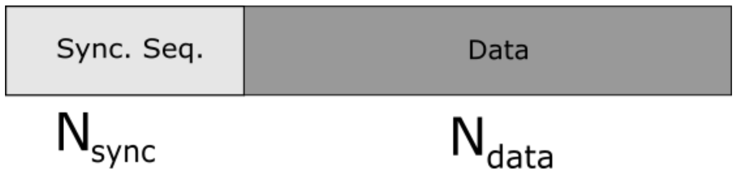


Figure 9.1: Frame structure that we use in this experiment.

9.1.2 Frame structure

In this lab, we insert pilot symbols for synchronization at the beginning of the generated packet (frame). Thus, the frame used in this scheme contains N_{sync} pilot symbols for synchronization at the beginning of a frame followed by the data signal, which is N_{Data} symbols long. Figure 9.1 shows the complete frame structure.

9.2 Pre-Lab $\boxed{4}$

Synchronization Sequences In the lab experiment you will have to implement code to generate synchronization sequences. These exercises are here to familiarize you with the ones that will be used in the lab.

S. 1. Write down the Barker code of length 13, and the formula for generating a Zadoff-Chu sequence of order P and length N_{sync}.

S. 2. What are the auto-correlation properties of these sequences? How do they compare to each other? Optionally support your argument with a plot.

S. 3. What values can the entries of the Barker sequence take? Suppose M-PSK modulation is used. How do you have to scale the entries such that the obtained values belong to the M-PSK constellation? Does this scaling affect the auto-correlation properties of the sequence?

S. 4. Do the entries of the Zadoff-Chu sequence belong to the M-PSK constellation? If no, why could this be a problem?

Phase Ambiguity The coefficient H in (9.1) is in general complex valued, i.e. $H = h\exp(i\phi_h)$. That means that the received pilot/synchronization sequence experiences a phase shift of ϕ_h, which should be compensated for at the receiver, since the symbols will then not be decoded correctly.

P. 1. Consider (9.2). Does this depend on the phase-shift ϕ_h? Support your argument with a calculation. **Hint**: Triangle Inequality.

P. 2. With your intuition from (9.2), do you think that the decision rule in (9.3) can compensate for this phase shift?

P. 3. If you answered the question above with no, how could you compensate for this phase shift? How would you measure this phase if the received samples do not contain noise, i.e. ignoring the $v[n]$-term in 9.1? How would you do it in the noisy case?

9.3 Lab Experiment 1 $\boxed{6}$

In this lab experiment you will learn how to generate a synchronization sequences, append them to the frame to be transmitted. At the receiver side you will implement an algorithm that detects the delay d using the cross-correlation approach presented in (9.3).

The programming will be done in MATLAB. Thus you will also learn how to create an interface between LabView and MATLAB, which is extremely easy to do, and constitutes a powerful tool in designing, prototyping and deploying transceiver algorithms.

9.3.1 Programming

Please locate the folder `matlab_utils`. For the Programming Tasks you will need the files contained in it.

Transmitter Code

1. Open the file `lti_tx_add_sync_preamble.m` and inspect the code. You will see the some conditional branches depending on the variable `sync_type`.
2. In the corresponding branches please implement the generation of the synchronization sequences you studied in the Pre-Lab tasks.

Receiver Code

1. Open the file `detect_frame.m` and inspect the code. You will see the some conditional branches depending on the variable `mode`.
2. In the branch for the case `"custom"`, please implement the correlation algorithm for frame detection. Please ignore the branch for `"prbdet"`.

Hints:

- Look closely at the comments in the code. They guide you towards a possible solution.
- You do not have to reinvent the wheel! Use pre-existing functions if necessary.
- The internet and the MATLAB Communications Toolbox are your best friends.

9.3.2 USRP Experiment

In this part you will mainly have to deal with the files in `usrp_code`.

Preparation Steps

1. Open the `usrp_interfacing.lvproject` file and then look into `LTI_Transmitter_Serial.gvi`. Configure the device and antenna name in the front panel as you did in previous experiments.
2. There you will see a Disable Structure. Inside that Disable Structure, please create a LabView MATLAB interface for the function `lti_tx_add_sync_preamble.m`.

   In order to do this, do the following:

   a) Right click on the project-root, the left click on "New → Interface for MATLAB".
   b) A new window should pop up with a `.mli`-file.
      Please re-name that to `lti_tx_add_sync_preamble.mli`.
   c) Click on "Add Interface Node" and select the path of the MATLAB-file. It should be in the directory `matlab_utils`.
   d) Afterwards configure the inputs and outputs of the `.mli` file according to Table 9.1.
   e) Locate the generated block in `Project Items` and then drag it inside the Disable Structure. Now wire everything properly and remove the Disable Structure.

If you need any help, contact one of the tutors or look at the documentation link available here https://www.ni.com/de-de/support/documentation/supplemental/18/interfacing-labview-communications-to-matlab--software.html.

3. Now have a look at the other `.mli` files. We have already set them up for you to use. The only thing you have to check is that the location of the `.m`-files corresponds to the location of the `matlab_utils`. If that is not the case, please modify the path accordingly.

4. At last, open the file `<`*`your directory`*`>/matlab_utils/params.txt` and set its path properly in the front panel.

Argument Name	Data Type	Behaviour
`sync_type`	String	Input
`sync_len`	DBL	Input
`M`	DBL	Input
`barker_rep`	DBL	Input
`pre_frame`	1-D Array of CDB	Input
`sync_seq`	1-D Array of CDB	Output
`frame`	1-D Array of CDB	Output

Table 9.1: Input and Output Configuration for `lti_tx_add_sync_preamble.mli`.

Experiment

1. Run the VI `LTI_Transmitter_Serial.gvi`, wait for some time and then run the VI `LTI_Receiver_Serial.gvi`

   If all works well, a MATLAB figure showing every signal processing step should pop up and the constellation diagram should contain four blobs centered around the constellation points. If that is not the case, please run the receiver again, as it might be that some noise samples have been fetched first and corrupted the process. Note that all processing steps beyond frame synchronization, aside from symbol demapping and visualization, are switched off.

   If after running it multiple times it still works bad, then you might want to reconsider your implementation or ask one of the tutors for help.

2. In `params.txt` set the field of the variable `sync_type` to `"barker"` and the field of `barker_rep` to `30`. Repeat the experiment multiple times, while varying the `barker_rep`.

   What happens for a small number of repetitions of the Barker sequence, i.e. $1, \ldots, 5$? Based on your observations argue why we need to repeat the Barker sequence.

3. In `params.txt` set the field of the variable `sync_type` to `"zadoff_chu"` and the field of `sync_len` to `129`. Repeat the experiment multiple times, while varying the `sync_len`. Note that since we fixed the order of the Zadoff-Chu Sequence to 40 `sync_len` should be a number which is relatively prime to 40, i.e. the greatest common divisor between the two should be 1.

   What happens for small sequence lengths? What happens for large ones?

4. Which of the two sequences performs better in terms of the bit error rate (BER)? Distinguish the cases when the synchronization sequence is short, and long respectively.

10 Channel Estimation and Equalization (Day 6)

In many communication settings signals are sent from a source to a destination terminal in the form of electromagnetic waves. These waves propagate through a physical medium (channel), which introduces distortions such as damping and delay, among others.

Good news: It is possible to alleviate some of these distortions at the receiver with proper signal processing.

Bad news: In order to do this, we need to know how the medium influences our signals, i.e. to have as much information about the channel as possible.

Deriving the characteristics of the channel directly from the physical reality is only possible in simple and controlled communication scenarios, where we have perfect knowledge about the geometrical and material properties of the medium. In most other settings, we can only guess how the channel behaves, and based on those guesses, we can then try correct the distortions induced. The words for the procedures mentioned in the last sentence are *channel estimation* and *equalization*.

In this lab, you will learn about the basic methods for *pilot-assisted* linear channel estimation and equalization for SISO channels. Furthermore you will get to see, how these methods work in real-world transmissions, by implementing them in USRPs.

The outline is as follows: First, we will talk about the basic channel and data models. In the spirit of digital signal processing (DSP), we will take a discrete-time approach. Then based on that, we will develop simple estimators for the parameters of the channel models and explore some ways to correct the induced distortions. Before going to implement them in hardware, we talk about the integration of these methods into the transmission scheme used in the lab.

10.1 Channel and Data Model

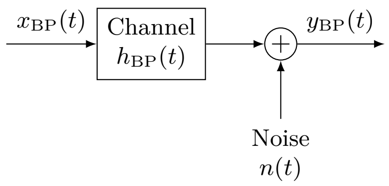


Figure 10.1: Wireless channel as equivalent lowpass LTI system.

Wireless channels are often modeled as linear time-invariant (LTI) systems. Then the propagation of a signal $x_{\mathrm{BP}}(t)$ is represented by a filtering operation with an LTI system with impulse response $h_{\mathrm{BP}}(t)$. Further, a very common way to model noise influences is to add a zero-mean wide-sense stationary Gaussian process $n(t)$ with spectral density $N_0/2$ over the frequency range of interest (white Gaussian noise) after the filtering operation. Then the channel output $y_{\mathrm{LP}}(t)$ is given by

$$y_{\mathrm{BP}}(t) = (x_{\mathrm{BP}} * h_{\mathrm{BP}})(t) + n(t) = \int_{-\infty}^{\infty} x_{\mathrm{BP}}(\tau) h_{\mathrm{BP}}(t-\tau)\, \mathrm{d}\tau + n(t). \tag{10.1}$$

This is illustrated in Fig. 10.1.

Now, we are interested in a discrete-time equivalent lowpass model, so we can apply DSP techniques. As outlined in Appendix B.4, the discrete-time equivalent lowpass representation is given by

$$y[k] = \sum_{m=-\infty}^{\infty} x[m]h[k-m] + n[k], \tag{10.2}$$

where $x[k] = x_{\mathrm{LP}}(2\pi k/W)$, $y[k] = y_{\mathrm{LP}}(2\pi k/W)$, $h[k] = h_{\mathrm{LP}}(2\pi k/W)$, and $\{n[k]\}_{k\in\mathbb{Z}}$ is a sequence of i.i.d. circular symmetric complex Gaussian random variables. W denotes the effective bandwidth of the signal $x_{\mathrm{BP}}(t)$. We assume that both $x[k]$ and $h[k]$ are causal and of finite lengths N_x and N_h, respectively, i.e. that

$$\mathrm{supp}(x) = \{k \in \mathbb{Z}\colon x[k] \neq 0\} \subset \{0, \dots, N_x - 1\}$$

and

$$\mathrm{supp}(h) = \{k \in \mathbb{Z}\colon h[k] \neq 0\} \subset \{0, \dots, N_h - 1\}.$$

Let us *ignore* the noise term $n[k]$ for now. In this case $y[k]$ will be at most $N_x + N_h$ samples long and can be written as

$$y[k] = \sum_{m=0}^{N_x-1} x[m]h[k-m] = \sum_{m=0}^{N_h-1} h[m]x[k-m]. \tag{10.3}$$

This leads to the obvious interpretation, that the (noiseless) channel is just a N_h-tap FIR filter. Thus the problem of channel estimation can be restated as finding the coefficients $h[k]$ of the FIR filter.

Now let us include the noise term again and rewrite (10.3) in a slightly different manner:

$$y[k] = \sum_{m=0}^{N_h-1} h[m]x[k-m] + n[k] = \boldsymbol{x}[k]^\mathsf{T}\boldsymbol{h} + n[k], \tag{10.4}$$

where we defined $\boldsymbol{x}[k] = [x[k], x[k-1], x[k-2], \dots, x[k-N_h+1]]^\mathsf{T}$ and $\boldsymbol{h} = [h[0], \dots, h[N_h-1]]^\mathsf{T}$. Now we collect all observations $y[k]$ from $k = 0$ to $k = N_x + N_h - 1$ into a vector $\boldsymbol{y}$, and (10.4) becomes:

$$\boldsymbol{y} = \begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ \vdots \\ y[N_x + N_h - 1] \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}[0]^\mathsf{T} \\ \boldsymbol{x}[1]^\mathsf{T} \\ \boldsymbol{x}[2]^\mathsf{T} \\ \vdots \\ \boldsymbol{x}[N_x + N_h - 1]^\mathsf{T} \end{bmatrix} \boldsymbol{h} + \begin{bmatrix} n[0] \\ n[1] \\ n[2] \\ \vdots \\ n[N_x + N_h - 1] \end{bmatrix} = \boldsymbol{X}\boldsymbol{h} + \boldsymbol{n} \tag{10.5}$$

Now, with the assumption that $x[n]$ is known at the receiver, which is what pilot-assisted means, the problem of channel estimation can be framed as finding the parameter $\boldsymbol{h}$, that fits the data $\boldsymbol{y}$ best, given the linear model assumption $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{h} + \boldsymbol{n}$. Note that because of the noise $\boldsymbol{n}$, $\boldsymbol{y}$ itself is a random variable.

10.2 Linear Channel Estimation

As stated in the last section, the idea behind channel estimation is to find the parameter $\boldsymbol{h}$ of the linear model in (10.5), given the observations $\boldsymbol{y}$ and the matrix of pilot symbols $\boldsymbol{X}$. We also want the channel estimate $\boldsymbol{h}_{\mathrm{est}}$ to be linear in the observation $\boldsymbol{y}$, i.e.

$$\boldsymbol{h}_{\mathrm{est}} = \boldsymbol{A}\boldsymbol{y} \tag{10.6}$$

for some suitable, complex-valued matrix $\boldsymbol{A}$.

In the following, we will look at things from a stochastic perspective and derive the maximum-likelihood estimator (MLE), and afterwards we will depart this stochastic setting and derive the practical least squares (LS) estimator. We will also see, that under some conditions on the noise statistics, the two are equivalent.

At last, we will give a brief overview about other methods that are employed for channel estimation.

10.2.1 Maximum Likelihood Estimation

The general idea to fitting the model best to the observed data, can be formulated as a maximum likelihood inference problem [6], i.e.:

$$\boldsymbol{h}_{ML} = \arg\max_{\boldsymbol{h}} p_{\boldsymbol{Y}|\boldsymbol{H}}(\boldsymbol{y}|\boldsymbol{h}) \tag{10.7}$$

where $p_{\boldsymbol{Y}|\boldsymbol{H}}(\boldsymbol{y}|\boldsymbol{h})$ denotes the probability density function (pdf) of the observed samples $\boldsymbol{y}$ given the parameters, i.e. channel, $\boldsymbol{h}$.

Since we are conditioning $\boldsymbol{y}$ on $\boldsymbol{h}$, which in general is a fixed realization of a random vector $\boldsymbol{H}$, the only randomness that can appear in $\boldsymbol{y}$ is due to the noise $\boldsymbol{n}$. Expressing the noise in terms of $\boldsymbol{y}$ and $\boldsymbol{h}$, yields following expression for (10.7):

$$\boldsymbol{h}_{ML} = \arg\max_{\boldsymbol{h}} p_{\boldsymbol{Y}|\boldsymbol{H}}(\boldsymbol{y}|\boldsymbol{h}) = \arg\max_{\boldsymbol{h}} p_{\boldsymbol{N}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h}), \tag{10.8}$$

where $p_{\boldsymbol{N}}(\cdot)$ denotes the pdf of the noise vector $\boldsymbol{n}$, which we will talk about next. A widely used assumption is to treat $\boldsymbol{n}$ as a circularly-symmetric complex Gaussian random vector with covariance matrix $\boldsymbol{C}_n$, i.e.,

$$p_{\boldsymbol{N}}(\boldsymbol{n}) = \frac{1}{\pi^M \det \boldsymbol{C}_n} \exp\Big(- \boldsymbol{n}^{\mathrm{H}} \boldsymbol{C}_n^{-1} \boldsymbol{n} \Big), \tag{10.9}$$

where M is the dimensionality of the vector (in this case the number of collected samples $N_x + N_h$) and $(\cdot)^{\mathrm{H}}$ denotes complex conjugation and transposition of a matrix. Note the assumption above is also in accordance to the system model in (10.4), since we are sampling a zero-mean, WSS Gaussian process.

Thus, inserting (10.9) into (10.8) yields

$$\begin{aligned}
\boldsymbol{h}_{ML} &= \arg\max_{\boldsymbol{h}} p_{\boldsymbol{N}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h}) \\
&= \arg\max_{\boldsymbol{h}} \frac{1}{\pi^M \det \boldsymbol{C}_n} \exp\Big(- (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h})^{\mathrm{H}} \boldsymbol{C}_n^{-1} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h}) \Big) \\
&= \arg\max_{\boldsymbol{h}} -(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h})^{\mathrm{H}} \boldsymbol{C}_n^{-1} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h}) \\
&= \arg\min_{\boldsymbol{h}} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h})^{\mathrm{H}} \boldsymbol{C}_n^{-1} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h}).
\end{aligned} \tag{10.10}$$

The second line follows from the fact that scaling a function by a constant factor does not change the location of the extremal points. The third line follows from the fact that $\exp(\cdot)$ is a monotonic function, thus only its the argument is relevant for the maximization procedure. The last line follows from the fact that $\max_\alpha -f(\alpha) = \min_\alpha f(\alpha)$.

Now, in order to solve (10.10) we define $\mathrm{J}(\boldsymbol{h}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h})^{\mathrm{H}} \boldsymbol{C}_n^{-1} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h})$, compute its derivative wrt. $\boldsymbol{h}^*$, where $(\cdot)^*$ denotes complex conjugation, set this to $\mathbf{0}$ and then solve for $\boldsymbol{h}$. Since $J(\boldsymbol{h})$ is a quadratic form in $\boldsymbol{h}$, it is also convex in $\boldsymbol{h}$, thus the local minimum is also global. It follows that

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{h}^*} \mathrm{J}(\boldsymbol{h}) &= \frac{\partial}{\partial \boldsymbol{h}^*} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h})^{\mathrm{H}} \boldsymbol{C}_n^{-1} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h}) \\
&= \frac{\partial}{\partial \boldsymbol{h}^*} \boldsymbol{h}^{\mathrm{H}} \boldsymbol{X}^{\mathrm{H}} \boldsymbol{C}_n^{-1} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h}) \\
&= \boldsymbol{X}^{\mathrm{H}} \boldsymbol{C}_n^{-1} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h}) \overset{!}{=} \mathbf{0}.
\end{aligned}$$

For more on matrix and vector derivatives, we refer to [7].

Thus, the ML channel estimate for the linear model in (10.5) under the assumption of zero-mean Gaussian noise can be given as

$$\boldsymbol{h}_{ML} = (\boldsymbol{X}^{\mathrm{H}}\boldsymbol{C}_n^{-1}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{H}}\boldsymbol{C}_n^{-1}\boldsymbol{y}. \tag{10.11}$$

It can also be shown that the above estimate $\boldsymbol{h}_{ML}$ is also the best unbiased linear estimator (BLUE) [6] for the channel vector $\boldsymbol{h}$, which roughly means that using the ML estimate contains the least uncertainty among all possible linear estimates cf. (10.6), while approaching the true value of the channel vector with increasing number of samples.

The problem now lies in computing the covariance matrix of the noise $\boldsymbol{C}_n$. This can be tackled by either estimating it, or by making some assumptions upon its structure, thus simplifying the problem. In many scenarios, such as a point-to-point, non-time-varying communication scenario, this comes with next to no reduction in performance.

10.2.2 Least Squares Estimation

In the following, we will derive a practical, but also powerful estimate of the channel vector.

The general idea to fitting the model best to the observed data, can also be formulated as a least squares (LS) problem. Namely, we want to find the channel coefficients, which minimize the Euclidean distance between the received samples and the transmitted ones after filtering through $h[k]$, formally,

$$\boldsymbol{h}_{LS} = \arg\min_{\boldsymbol{h}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h}\|_2^2. \tag{10.12}$$

In order to solve the problem in (10.12), we define

$$\mathrm{J}(\boldsymbol{h}) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h}\|_2^2 = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h})^{\mathrm{H}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h}),$$

compute its derivative with respect to $\boldsymbol{h}^*$, set this to $\mathbf{0}$ and then solve for $\boldsymbol{h}$. Since $J(\boldsymbol{h})$ is a quadratic form in $\boldsymbol{h}$, it is also convex in $\boldsymbol{h}$, thus the local minimum is also global. We compute

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{h}^*}\mathrm{J}(\boldsymbol{h}) &= \frac{\partial}{\partial \boldsymbol{h}^*}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h})^{\mathrm{H}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h}) \\ &= \frac{\partial}{\partial \boldsymbol{h}^*}\boldsymbol{h}^{\mathrm{H}}\boldsymbol{X}^{\mathrm{H}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h}) \\ &= \boldsymbol{X}^{\mathrm{H}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h}) \overset{!}{=} \mathbf{0}.\end{aligned}$$

This yields the solution

$$\boldsymbol{h}_{LS} = (\boldsymbol{X}^{\mathrm{H}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{H}}\boldsymbol{y}. \tag{10.13}$$

We can see that the solutions in (10.11) and (10.13) have somewhat similar structure, and thus can be related to each other somehow.

A particular case of interest, is when the noise is white, i.e. $\boldsymbol{C}_n = \sigma^2\boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix of suitable dimensions and σ^2 is the noise variance, which can be measured. In this case the ML estimate can be written as

$$\begin{aligned}\boldsymbol{h}_{ML} &= (\boldsymbol{X}^{\mathrm{H}}\boldsymbol{C}_n^{-1}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{H}}\boldsymbol{C}_n^{-1}\boldsymbol{y} \\ &= (\boldsymbol{X}^{\mathrm{H}}\sigma^{-2}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{H}}\sigma^{-2}\boldsymbol{y} \\ &= (\boldsymbol{X}^{\mathrm{H}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{H}}\boldsymbol{y} \\ &= \boldsymbol{h}_{LS}.\end{aligned}$$

Thus, in the case of white noise the ML and the LS estimates yield the same results.

10.3 Linear Channel Equalization

Now that we know the channel coefficients $\boldsymbol{h}$, we need to find a way to compensate for their effect using a FIR filter.

In this section, we will first have another look at (10.4) and better define what we mean by compensation. Then, we will explore a method for direct equalization and one using exploiting the full information about the channel.

10.3.1 System Model Revisited

For the next discussion, we want to reformulate the channel model, such that the observations are a superposition between a linear function of the transmitted samples and noise. This can be done starting from (10.4) and using the same arguments as in Section 10.1 as follows

$$y[k] = \sum_{m=0}^{N_x-1} h[k-m]x[m] + n[k] = \boldsymbol{h}[k]^\intercal \boldsymbol{x} + n[k], \tag{10.14}$$

where we defined $\boldsymbol{h}[k]^\intercal = [h[k], h[k-1], h[k-2], \ldots, h[k-N_x+1]]$ and $\boldsymbol{x} = [x[0], \ldots, x[N_x-1]]$. As before, collecting all samples $y[k]$ into a vector yields the system model

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{h}[0]^\intercal \\ \boldsymbol{h}[1]^\intercal \\ \boldsymbol{h}[2]^\intercal \\ \vdots \\ \boldsymbol{h}[N_x+N_h-1]^\intercal \end{bmatrix} \boldsymbol{x} + \boldsymbol{n} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{n}. \tag{10.15}$$

Now our goal is to somehow "invert" the matrix $\boldsymbol{H}$ using a FIR filter with impulse response $g[k]$. To this end, we will formulate the filtered signal $d[k]$ also as a matrix-vector multiplication

$$d[k] = \sum_{m=0}^{N_y-1} g[k-m]y[m] = \boldsymbol{g}[k]^\intercal \boldsymbol{y} \tag{10.16}$$

$$= \sum_{m=0}^{N_g-1} y[k-m]g[m] = \boldsymbol{y}[k]^\intercal \boldsymbol{g} \tag{10.17}$$

with $\boldsymbol{g}[k]^\intercal$ and $\boldsymbol{y}$ defined analogously as above. From (10.16) follows with (10.15)

$$\boldsymbol{d} = \boldsymbol{G}\boldsymbol{y} = \boldsymbol{G}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{G}\boldsymbol{n} \tag{10.18}$$

and consequently,

$$\boldsymbol{d} = \boldsymbol{Y}\boldsymbol{g}. \tag{10.19}$$

Now we can see that the "inversion" problem can be seen as trying to find the matrix $\boldsymbol{G}$ and/or the vector $\boldsymbol{g}$, which ensures that the filtered samples $\boldsymbol{d}$ are as close as possible to the transmitted ones $\boldsymbol{x}$. In the next sections, you will see how this can be achieved, using a similar method as in 10.2.2, as well as another one which completely eliminates the inter-symbol-interference (ISI).

10.3.2 Least Squares Equalization

Suppose the pilot symbols $\boldsymbol{x}$ are known at the receiver, which is most of the time the case when employing pilot symbols in a communication scheme. In this case, a practical approach would be to find an equalizer $\boldsymbol{g}$, which maps the received samples as close as possible (in terms of Euclidean distance) to the known transmitted ones $\boldsymbol{x}$. This is the least squares (LS) approach:

$$\boldsymbol{g}_{LS} = \arg\min_{\boldsymbol{g}} \|\boldsymbol{x} - \boldsymbol{d}\|_2^2 = \arg\min_{\boldsymbol{g}} \|\boldsymbol{x} - \boldsymbol{Y}\boldsymbol{g}\|_2^2. \tag{10.20}$$

The solution to (10.20) is then given by

$$\boldsymbol{g}_{LS} = (\boldsymbol{Y}^{\mathrm{H}}\boldsymbol{Y})^{-1}\boldsymbol{Y}^{\mathrm{H}}\boldsymbol{x}. \tag{10.21}$$

Note that when employing this technique, the estimated channel state information (CSI) is not really needed, which makes it practical when channel estimates are hard to obtain and/or are inaccurate. Furthermore, because no channel estimation per-se is needed, the receiver complexity can be reduced.

10.3.3 Zero Forcing Equalization

What if we do not want to employ a LS approach directly but remove the ISI directly. Let us have a look at (10.2) again, without making any assumption about the signals

$$y[k] = \sum_{m=-\infty}^{\infty} x[m]h[k-m] + n[k] = (x * h)[k] + n[k]. \tag{10.22}$$

Taking the $\mathcal{Z}$-transform and using the convolution theorem, we obtain

$$Y(z) = \mathcal{Z}\{y[k]\} = \sum_{k=-\infty}^{\infty} y[k]z^{-k} = H(z)X(z) + N(z). \tag{10.23}$$

Filtering $y[k]$ by $g[k]$ results in a multiplication in the $\mathcal{Z}$-domain

$$D(z) = Y(z)G(z) = G(z)H(z)X(z) + G(z)N(z). \tag{10.24}$$

In order to ensure complete removal of ISI, we require that the "useful" part of $D(z)$, i.e. $G(z)H(z)X(z)$ be exactly $X(z)$. From this we have

$$G(z)H(z) \overset{!}{=} 1. \tag{10.25}$$

The obvious solution to (10.25) is

$$G(z) = \frac{1}{H(z)},$$

from which results, that if $h[k]$ is a FIR filter, then $g[k]$ can be an IIR filter.

On the other hand, assuming $x[k]$ and $h[k]$ are finite length sequences, we can design $g[k]$ to be a finite length FIR filter. This can be done as follows. We first take the inverse $\mathcal{Z}$-transform of (10.25) and obtain

$$\mathcal{Z}^{-1}\{G(z)H(z)\} \overset{!}{=} \mathcal{Z}^{-1}\{1\} \iff c[k] = (g * h)[k] \overset{!}{=} \delta[k] = \begin{cases} 1, & k = 0, \\ 0, & \text{else.} \end{cases} \tag{10.26}$$

We now write the convolution between $g[k]$ and $h[k]$ as a matrix-vector multiplication, by stacking the results from time-steps $k = 0$ to $k = K = N_g + N_h - 1$, which gives

$$\boldsymbol{c} = \begin{bmatrix} c[0] \\ c[1] \\ \vdots \\ c[K] \end{bmatrix} = \begin{bmatrix} \boldsymbol{h}[0]^{\mathsf{T}} \\ \boldsymbol{h}[1]^{\mathsf{T}} \\ \boldsymbol{h}[2]^{\mathsf{T}} \\ \vdots \\ \boldsymbol{h}[K]^{\mathsf{T}} \end{bmatrix} \boldsymbol{g} = \boldsymbol{H}\boldsymbol{g}. \tag{10.27}$$

Since we do not have to consider time-steps beyond $k = K$, we can rewrite the expression above as

$$\boldsymbol{c} = \boldsymbol{H}\boldsymbol{g} \overset{!}{=} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{10.28}$$

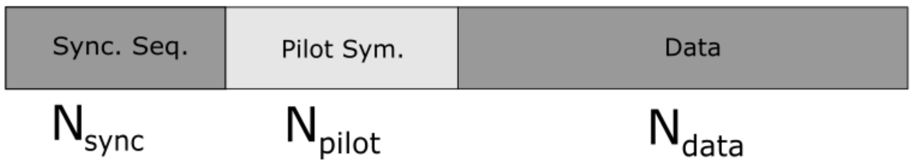


Figure 10.2: Frame structure with pilot symbols for channel estimation/equalization.

Since $\boldsymbol{H}$ is a tall matrix, the system above will always have a solution independent of the number of taps in the equalizer N_g. Furthermore, one may solve the system above using e.g. the Moore-Penrose pseudoinverse of $\boldsymbol{H}$.

One may also see why this filter is called zero-forcing filter, because one "forces" zeros in all other entries beside the first one. Note, that for this procedure we do not really need pilots, only the channel state information.

A last remark, that should be made at this stage is that the zero-forcing condition in equations (10.26) and (10.28) can also be derived from (10.18), and can be stated as $\boldsymbol{GH} = \boldsymbol{I}$. If we take a look at this from the perspective of estimation theory, the ZF condition ensures that $\boldsymbol{d} = \boldsymbol{Gy}$ is an unbiased estimate for $\boldsymbol{x}$, i.e. $\mathbb{E}[\boldsymbol{d}] = \boldsymbol{x}$, $\forall \boldsymbol{x}$, if the noise is zero-mean.

10.4 System Integration

In this section, we will see how to incorporate the methods described above in the communication system for this laboratory. We will first talk about the modifications to the frame structure, such that pilot symbols are included, and then we will talk about the implementation of the estimation and equalization schemes, such that they are in accordance to our system.

10.4.1 Frame Structure

In this lab, we will use the frame structure depicted in Fig. 10.2 for the communication between USRPs. It consists of three sections, namely:

1. Synchronization sequence: This is used to detect the start of the frame, as you learned in the last lab, and to optionally resolve the phase ambigity at the receiver. In this lab we use a Barker sequence of length 13, which we repeat 30 times, resulting in a length $N_{\text{sync}} = 390$ of symbols.

2. Pilot symbols: Next, a block of pilot symbols is inserted for channel estimation and equalization. Cf. our system model in (10.4) they correspond to $x[k]$. In this lab, there are two types of sequences we will use, namely a Walsh–Hadamard sequence and a Gold sequence, whose lengths N_{pilot} can be configured as desired.

3. Data symbols: These are the symbols we actually want to transmit. In this lab, we will use the serialization of the string "Hello World" into bits as our payload.

The above structure is loosely based on the standard IEEE 802.15.3c [8], which also uses a block of pilot symbols for channel estimation. In LTE and other standards, pilot symbols are not sent as a block, but interleaved with the other symbols in the frame. This is done to improve the channel estimates, especially for frequency-selective channels [9].

10.4.2 Receiver Design

In this section we will see how to implement LS channel estimation and the equalization methods presented in 10.3 in our USRP communication system.

After pulse shaping, carrier frequency-offset compensation, timing synchronization and frame detection, a collection of N_{total} samples

$$\boldsymbol{y} = [y[0], \cdots, y[k], y[k-1], \dots y[k-(N_{\text{sync}}+N_{\text{pilot}}+N_{\text{data}}-1)], \dots, y[N_{\text{total}}]]^{\intercal}$$

is delivered from the USRP's buffer.

Since the process of frame detection, gives us the exact time-step where the frame starts and since we are only processing one frame, the channel delay in samples is already compensated for. Thus no further time-alignment is needed. This means that the **observed** sequence of pilot symbols, that passed through the channel

$$\boldsymbol{y_p} = [y[N_{\text{sync}}], y[N_{\text{sync}}+1], \cdots y[N_{\text{sync}}+N_{\text{pilot}}-1]]^{\intercal}$$

has effectively the same length as the transmitted ones

$$\boldsymbol{x_p} = [x[N_{\text{sync}}], x[N_{\text{sync}}+1], \cdots x[N_{\text{sync}}+N_{\text{pilot}}-1]]^{\intercal}.$$

Thus, the system model in (10.5) can be restated by only considering the first N_{pilot} rows of the $\boldsymbol{X}$ matrix, i.e.:

$$\boldsymbol{y_p} = \boldsymbol{X_p h} + \boldsymbol{n} \tag{10.29}$$

where

$$\boldsymbol{X_p} = \begin{bmatrix} x[N_{\text{sync}}] & 0 & 0 & \cdots & 0 \\ x[N_{\text{sync}}+1] & x[N_{\text{sync}}] & 0 & \cdots & 0 \\ x[N_{\text{sync}}+2] & x[N_{\text{sync}}+1] & x[N_{\text{sync}}] & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x[N_{\text{sync}}+N_{\text{pilot}}-1] & x[N_{\text{sync}}+N_{\text{pilot}}-2] & x[N_{\text{sync}}+N_{\text{pilot}}-3] & \cdots & x[N_{\text{sync}}+N_{\text{pilot}}-N_h+1] \end{bmatrix}.$$

Note the zeros appear there, because we cannot and do not want to use data symbols or frame synchronization symbols for the estimation.

The same procedure can be then applied for least squares equalization to get a suitable system model.

At last, after collecting the samples, forming the vector $\boldsymbol{y_p}$ and matrices $\boldsymbol{X_p}$ or $\boldsymbol{Y_p}$ respectively, one can then use equations (10.13), (10.21) and (10.28) to get an answer directly. The implementation of the aforementioned equations can be then done in any programming language of choice. Here we restrict ourselves to LabView and MATLAB.

Further exploration of the nature of the numerical problem to solve gives insights to optimize the solver algorithm, and from there on, one may also start implementing these solvers in memory- or time-constrained systems, e.g. ASICs, FPGAs.

10.5 Pre-Lab $\boxed{4}$

Please answer following questions/solve following problems and write the answers down on your lab report.

Channel Estimation

E. 1. We consider the linear model in (10.5) and suppose the channel coefficient vector $\boldsymbol{h}$ is a complex Gaussian random vector with mean $\boldsymbol{\mu_h}$ and covariance matrix $\boldsymbol{C_h}$. In that case, another approach to estimation would be the maximum a posteriori method, namely:

$$\boldsymbol{h}_{\text{MAP}} = \arg\max_{\boldsymbol{h}} p_{\boldsymbol{H}|\boldsymbol{Y}}(\boldsymbol{h}|\boldsymbol{y}) = \arg\max_{\boldsymbol{h}} p_{\boldsymbol{Y}|\boldsymbol{H}}(\boldsymbol{y}|\boldsymbol{h}) p_{\boldsymbol{H}}(\boldsymbol{h}) \tag{10.30}$$

1. Under which condition on $p_{\boldsymbol{H}}(\boldsymbol{h})$ are the MAP and ML estimators equal?

2. Derive the MAP-estimate $\boldsymbol{h}_{ML}$.

   Hint:

   – Remember
   $$p_{\boldsymbol{H}}(\boldsymbol{h}) = \frac{1}{\pi^M \det \boldsymbol{C}_{\boldsymbol{h}}} \exp\Big(- (\boldsymbol{h} - \boldsymbol{\mu}_{\boldsymbol{h}})^{\mathrm{H}} \boldsymbol{C}_{\boldsymbol{h}}^{-1} (\boldsymbol{h} - \boldsymbol{\mu}_{\boldsymbol{h}})\Big)$$

   – Construct a function $J(\boldsymbol{h})$ similar to the one in Section 10.2.1 compute its derivative wrt. $\boldsymbol{h}^*$, set this to $\boldsymbol{0}$ and then solve for $\boldsymbol{h}$.

E. 2. Let $N_x = 7$ and $N_h = 4$.

   – Write the full matrix $\boldsymbol{X}$ using the vector $\boldsymbol{x}^{\intercal} = [x_0, x_1, \cdots, x_{N_x-1}]$ down.

   – How is this matrix called? What dimensions does it have in this case? What about the general case?

Channel Equalization

Q. 1. Prove (10.21).

Q. 2. What could be a possible disadvantage of the ZF-equalizer, if $H(z) \approx 0$ for $z \in [z_{\min}, z_{\max}]$?

10.6 Lab Experiment 1 $\boxed{6}$

In the following, you will implement the concepts above and familiarize yourself with the interplay between MATLAB and LabView. At first please locate the folders `usrp_code` and `matlab_utils`.

10.6.1 Programming

In this part you will mainly have to do with the files in `matlab_utils`.

Pilot Symbols

1. Familiarize yourself with Walsh-Hadamard and Gold Sequences by reading up on them shortly.
2. Open the file `labview_gen_pilot_seq.m` and inspect the code. You will see conditional branches depending on the variable `pilot_type`.
3. In the corresponding branches please implement the generation of pilot sequences, namely Walsh-Hadamard (WH) and Gold sequences of the length given by `pilot_len`.

Hints:

- Note that for the WH sequences any row of the WH matrix of the correct order is a suitable pilot sequence.
- The entries in the Gold sequence are either 0 or 1. Make sure to apply following mapping after generation, namely:
  $$0 \mapsto 1 \qquad 1 \mapsto -1$$
- The modulation order M is already given. Please apply a phase-shift $\exp\left(i\frac{\pi}{M}\right)$ to the entries of the pilot sequence, to make sure the pilot symbols belong to the constellation.
- The internet and MATLAB Communications Toolbox are your best friends.

Channel Estimation

1. Open the file `labview_channel_estimation.m` and inspect the code.
2. Implement the LS-channnel estimator as given in Eq. 10.13.

Hints:

- First construct the matrix $\boldsymbol{X}_p$, as given in 10.29.
- You can use MATLAB built-in functions if you deem necessary.

Channel Equalization

1. Open the file `labview_channel_equalization.m` and inspect the code. You will also see here conditional branches depending on the variable `eq_mode`.
2. Implement the equalizers given by equations 10.12 and 10.28.

Hints:

- Make sure to construct the matrices $\boldsymbol{Y}$ and $\boldsymbol{H}$ correctly. The former can be done analogously to 10.29.
- The internet and MATLAB Communications Toolbox are your best friends.

10.6.2 USRP Experiment

In this part you will mainly have to do with the files in `usrp_code`.

Preparation Steps

1. Open the `matlab_interfacing.lvproject`.
2. Make sure that all `.mli` files have the correct path set to them, which should be `<`*`your directory`*`>/usrp_code`. If not, set it manually.
3. Now open `LTI_Transmitter_Serial.gvi` and `LTI_Receiver_Serial.gvi`. Configure the device and antenna name in the front panel as you did in previous experiments.
4. At last, open the file `<`*`your directory`*`>/matlab_utils/params.txt` and set its path properly in the front panel.

Experiment

1. Run the VI `LTI_Transmitter_Serial.gvi`, wait for a moment and then run the VI `LTI_Receiver_Serial.gvi`

   If all works well, a MATLAB figure showing every signal processing step should pop up and the constellation diagram should contain four blobs centered around the constellation points. If that is not the case, please run the receiver again, as it might be that some noise samples have been fetched first and corrupted the process.

   If after running it multiple times it still works bad, then you might want to reconsider your implementation or ask one of the tutors for help.

2. In `params.txt` set the field of the variable `eq_type` to `"zf"` and the field of the variable `num_taps_equalizer` to `31`. Repeat 1 for varying number of equalizer taps, as well as the number of channel taps `num_taps_channel`.

   What does the interference measure plot show? Did you expect this result? How does the constellation plot look after equalization? What effect does this have for the SNR? (Set `pilot_type` to `"none"`).

3. Now repeat the experiment for the LS-equalizer. Do this by setting field of the variable `eq_type` to `"ls"`. Answer the questions above again for this type of equalization.

4. For the given pilot length how many taps should the channel and the equalizer have at most? What happens if you increase the number of taps above that level or decrease the length of the pilot sequence in `pilot_len` enough?

5. Now keeping all parameters fixed, try the different pilot sequences out, by setting `pilot_type` to `"gold"` and `"walsh_hadamard"`. Do you observe any difference between the two?

11 Orthogonal Frequency Division Multiplexing (Day 7)

In previous chapters, the information is modulated on the whole signal bandwidth and such transmission scheme is called single-carrier modulation. However, in wireless communication, due to the multipath effect, the frequency-selective channels might lead to severe inter-symbol interference (ISI) and degrade the system performance. To mitigate these problems, the multi-carrier modulation divides the available signal bandwidth into a number of frequency components (subcarriers) such that the transmit signal at each subcarrier will experience flat-fading channel. In this chapter, we introduce the orthogonal frequency division multiplexing (OFDM), which is the most commonly used multi-carrier technique in today's wireless systems.

11.1 Basics of OFDM

11.1.1 OFDM Modulation and Demodulation

Consider a communication system with available bandwidth of B Hz evenly divided into N subcarriers such that the subcarrier spacing (SCS) is $\Delta f = B/N$ Hz, the n-th subcarrier is denoted as $f_n = f_0 + n\Delta f = n\Delta f$ if f_0 is aligned with 0 Hz. We associate the subcarriers with the set of complex exponential signals $\{e^{j2\pi f_n t}\}_{n=0}^{N-1}$ and notice the orthogonality between them:

$$\frac{1}{T_0}\int_0^{T_0} e^{j2\pi f_n t} e^{-j2\pi f_m t} dt = \begin{cases} 1, & n = m \\ 0, & \text{otherwise,} \end{cases}$$

where $T_0 = 1/\Delta f$ is called the OFDM symbol duration. Therefore, by modulating the data symbols (after PSK or QAM) with the orthogonal signal set, one can demodulate them without interference from other subcarriers. Specifically, within the k-th OFDM symbol, or at time t with $kT_0 \le t \le (k+1)T_0$, given a set of N data symbols $\{X_k[n]\}_{n=0}^{N-1}$, the baseband OFDM signal is obtained as:

$$x_k(t) = \sum_{n=0}^{N-1} X_k[n] e^{j2\pi f_n t}, \quad kT_0 \le t \le (k+1)T_0.$$

The inverse processing is

$$X_k[n] = \frac{1}{T_0}\int_{kT_0}^{(k+1)T_0} x_k(t) e^{-j2\pi f_n t} dt.$$

As we are dealing with discrete-time signals, the above signals are sampled with rate $\frac{1}{T_s} = B = N\Delta f$. Therefore, the discrete baseband signal at the l-th sample of the k-th OFDM symbol $x_k[l] = x_k(t)|_{t=kT_0+lT_s}$ is given as

$$x_k[l] = \sum_{n=0}^{N-1} X_k[n] e^{j2\pi f_n (kT_0 + lT_s)}, \quad l = 0, 1, ..., N-1,$$

and can be simplified to

$$x_k[l] = \sum_{n=0}^{N-1} X_k[n] e^{j2\pi nl/N}, \quad l = 0, 1, ..., N-1, \tag{11.1}$$

and $X_k[n]$ is recovered by

$$X_k[n] = \frac{1}{N}\sum_{l=0}^{N} x_k[l]e^{-j2\pi nl/N}, \quad n = 0, 1, ..., N-1. \tag{11.2}$$

In fact, Eq. (11.2) and (11.1) are discrete Fourier transform (DFT) and its inverse version, which can be implemented efficiently using fast Fourier transform (FFT) on modern computers. In the following, we visualize such procedure in terms of OFDM grid, as shown in Fig. 11.1. In convention, the zero subcarrier ($n = 0$) is located in the center row of the grid. The data symbols $X_k[n]$ are first mapped onto the OFDM grid, and the inverse DFT is applied along the subcarrier axis to obtain the time domain signals.

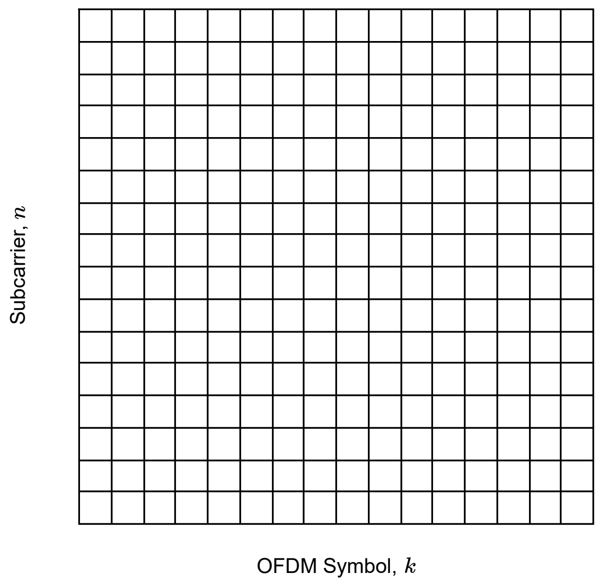


Figure 11.1: OFDM grid.

11.1.2 Cyclic Prefix

The multipath channel that introduces ISI will break the orthogonality of OFDM signals harm the system performance. To overcome this problem, one way is to add guard intervals that are longer than the maximum channel delay spread between adjacent OFDM symbols. The guard intervals can be filled with zeros, which is called zero padding. But the most common way is to fill these intervals with the last samples of the next OFDM symbol, and is called Cyclic Prefix (CP). The benefits of CP can be explained by the properties of DFT. Consider two finite length sequences $x[n]$ and $h[n]$, the circular convolution of them are

$$y[n] = (h \circledast x)[n] = \sum_{m=-\infty}^{\infty} h[m]x[(n-m)_{\text{mod } N_x}] = \sum_{m=0}^{N_h} h[m]x[(n-m)_{\text{mod } N_x}]$$

where N_x, N_h ($N_x \geq N_h$) are the lengths of $x[n]$, $h[n]$ respectively, and $(n-m)_{\text{mod } N_x}$ takes the result of $(n-m)$ modulo N_x. It is also equivalent to periodically extend $x[n]$ to infinite length and convolve it with $h[n]$, thus is called circular convolution. If we denote the DFT of $h[n]$ and $x[n]$ as $H[k]$ and $X[k]$ respectively, the DFT of $y[n]$ will be the product of them, i.e., $Y[k] = H[k]X[k]$.

$$Y[k] = H[k]X[k].$$

We assume further $N_x \geq N_h$ and extend $x[n]$ at the beginning with its last N_g ($N_g \geq N_h$) elements, i.e.,

$$\tilde{x}[n] = [x[N_x - N_g], x[N_x - N_g + 1], ..., x[N_x - 1], x[0], x[1], ..., x[N_x - 1]].$$

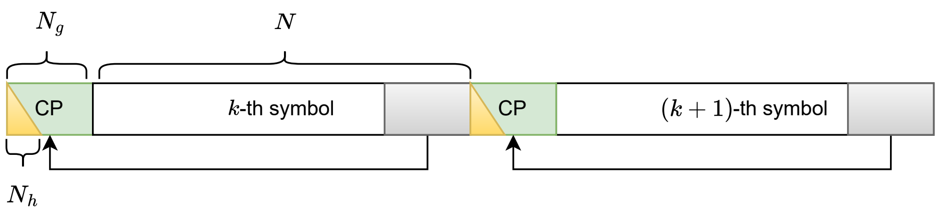
it's easy to show $y[n] = (h * \tilde{x})[n]$, which is the linear convolution between both sequences. With this knowledge, we end up with the idea of CP. Specifically, given the OFDM signal $x_k[l]$ at the k-th OFDM symbol of finite length N, we copy its last N_g samples to its head and obtain

$$\tilde{x}_k[l] = [x_k[N - N_g], x_k[N - N_g + 1], ..., x_k[N - 1], x_k[0], x_k[1], ..., x_k[N - 1]],$$

after the linear convolution with channel impulse response $h[l]$ of length $N_h \leq N_g$ and ignoring the first N_g samples, the received signal $y_k[l]$ is

$$y_k[l] = (h * \tilde{x}_k)[l] = (h \circledast x_k)[l], \quad l = 0, ..., N - 1. \tag{11.3}$$

Taking the DFT of $y_k[l]$ we have $Y_k[n] = H_k[n]X_k[n]$ where $H_k[n]$ is the DFT of $h[l]$ and $X_k[n]$ is just the data symbols that we mapped onto the location (k, n) of OFDM grid.



Consequently, inserting the CP as a guard interval for each OFDM symbol on one hand avoids the ISI and keeps the signals orthogonality, on the other hand, the CP makes the channel effect equivalent to a circular convolution, or simply scalar products in frequency domain, and hence significantly simplifies the following channel estimation and equalization. In practice, the CP length N_g is usually specified as a fraction of the OFDM symbol duration, for instance, a CP fraction of 0.1 indicates that the last 10% part of the OFDM symbol is copied to CP, with length of $0.1T_0$ in seconds or rounded integer of $0.1N$ in number of samples.

11.2 OFDM Synchronization

Similar to the single-carrier case, the carrier frequency offset (CFO) and symbol time offset (STO) are also harmful for OFDM systems as they can destroy the signal orthogonality. However, in OFDM, both CFO and STO are usually jointly estimated and compensated since the alignments in OFDM symbols and subcarriers are relevant to each other. In this section, we will show the impacts of CFO and STO on OFDM signals and how to leverage the signals properties in frequency and time domains to correct those effects.

11.2.1 Effects of CFO

As discussed in Chapter 7, a CFO of f_{offset} will cause a linearly varying phase offset. We define $\epsilon = f_{\text{offset}}/\Delta f = \epsilon_i + \epsilon_f$ with ϵ_i being the integer part and ϵ_i the fractional part of ϵ and it can be shown that the CFO introduces a phase offset of $e^{j2\pi\epsilon l/N}$ for the l-th sample compared to the first sample within each OFDM symbol. It's obvious that the integer frequency offset (IFO) ϵ_i can result in a integer subcarrier shift of $X_k[n]$ in frequency domain which becomes $X_k[n - \epsilon_i]$, as shown in Fig. 11.2. This won't break the signal orthogonality, but the subcarrier disalignment will mislead the subsequent steps. The fractional frequency offset (FFO) ϵ_f, on the other hand, destroys the orthogonality and introduce inter carrier interference (ICI).

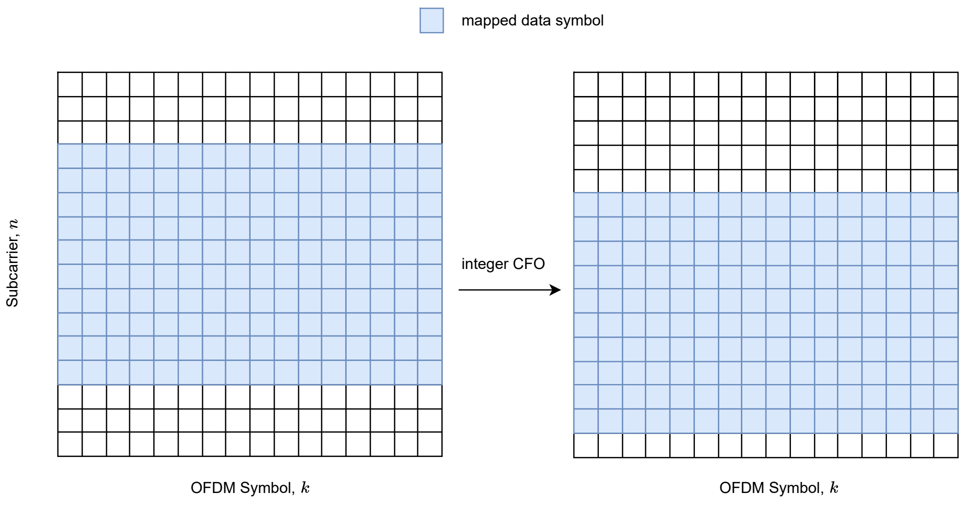


Figure 11.2: Cause of integer CFO

11.2.2 Effects of STO

As we don't use pulse shaping for OFDM signals in this Lab, no upsampling step is applied. Therefore, the STO compensation for OFDM can be understood as frame synchronization, i.e., we want to detect the starting point of the modulated OFDM grid. The effects of STO can also be classified into two cases, depending on whether the synchronized OFDM symbol is overlapped with the adjacent symbols. If the detected OFDM symbol contains the components of others due to, for instance, the lagged channel response, the ISI is introduced and the signal orthogonality won't be maintained. On the other hand, if the detected symbol starting point is located within the CP but after the maximum length of the channel impulse response, the orthogonality remains but a linearly varying phase offset in frequency domain will appear, that is, a STO δ will result in $x_k[l+\delta]$ in time domain and $X_k[n]e^{j2\pi n\delta/N}$ in frequency domain.

11.2.3 Joint Estimation and Compensation

The estimation and compensation of CFO and STO can be performed either in frequency or time domain, and depending on whether training sequence is used, it can also be classified into data-aided and blind techniques. Therefore, the synchronization step in OFDM is so flexible that there are a lot of methods to be proposed such as Schmidle Cox algorithm, null symbol based, and CP based algorithm, etc. In this Lab, we introduce one approach to provide you a insight into it.

The approach in this Lab are based on the OFDM frame structure as shown in Fig. 11.3. The synchronization training sequence is mapped onto the middle of the first OFDM symbol at each frame, and the pilot sequence and data symbols will be discussed later. The training sequence is known to both transmitter and receiver, thus the cross correlation method can be applied.

We first leverage the correlation property of CP to locate the starting point of OFDM symbols. We notice that the CP is the same as the last part of each OFDM symbol. With this knowledge, we can apply two sliding windows of the same size N_g separated by N and compute the cross correlation between the samples in both windows, as illustrated in Fig. 11.4. Specifically, suppose the received signal is $y[l]$, the symbol starting point $\hat{\delta}$ is obtained by

$$\hat{\delta} = \arg\max_{\delta} \left\{ \left| \sum_{i=\delta}^{\delta+N_g-1} y[i]y^*[i+N] \right| \right\}. \tag{11.4}$$

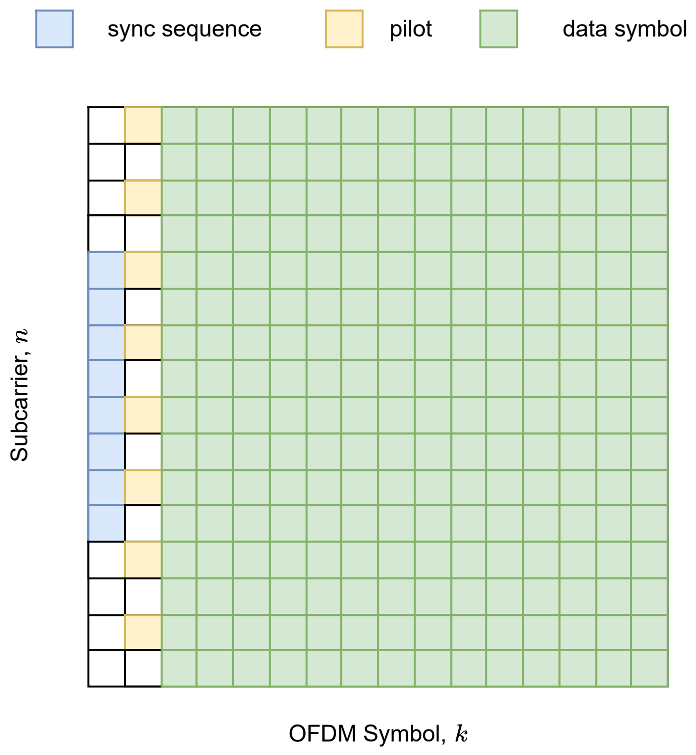


Figure 11.3: OFDM frame structure used in this Lab.

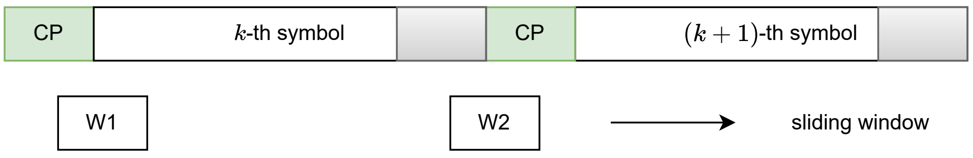


Figure 11.4: OFDM symbol starting point detection based on CP correlation.

It's straightforward to show that such method won't be affected by CFO. To improve the estimation performance, one can utilize more window pairs at once. The next step is to remove the FFO such that no ICI is present when we demodulate the signals. To do it, we leverage the property of CP again. It's obvious that if $y_k[i]$ for $i = 0, ..., N_g - 1$ denotes the CP samples of the k-th OFDM symbol, we have $y_k[i + N] = y_k[i] \exp\{j2\pi N\epsilon/N\} = y_k[i] \exp\{j2\pi\epsilon\}$. Hence, the estimation of ϵ can be obtained by computing the phase difference between both parts. However, the phase computed in this way is within -0.5 and 0.5, thus can only reveal the FFO component. Therefore, the estimation of ϵ_f is given by

$$\hat{\epsilon}_f = \frac{1}{2\pi}\angle\left\{\sum_{i=0}^{N_g-1} y_k^*[i]y_k[i+N]\right\}. \tag{11.5}$$

Also, for a better performance, multiple OFDM symbols can be leveraged.

After OFDM symbol synchronization and FFO compensation, the ISI and ICI are removed and the inverse DFT can be applied to obtain the OFDM grid. The remaining tasks are to compensate the IFO and detect the frame starting symbol. This can be done by correlating the known training sequence with each column of the demodulated OFDM grid. As discussed before, the IFO can shift the symbols with integer subcarriers, and such shift can be obtained by finding the peak of the cross correlation along the frequency dimension. In addition, the frame starting symbol is exactly the column on which the peak is located. To this end, a training sequence with a good cross correlation property is needed and you have already learned some types from the previous chapters.

11.3 Channel Estimation and Equalization

After synchronization, we should be able to extract a OFDM grid with the same structure as Fig. 11.3. The next step is to estimate the channel coefficients with the aid of pilot sequence. As discussed above, with the help of CP, the channel effect in frequency domain is just a complex factor on each symbol, i.e.,

$$Y_k[n] = H_k[n]X_k[n]. \tag{11.6}$$

Therefore, given a set of pilot symbols $X_k^p[n]$ and the received symbols $Y_k^p[n]$, which can be extracted from the predefined locations on the OFDM grid, the channel factors $H_k^p[n]$ are obtained simply by $H_k^p[n] = Y_k^p[n]/X_k^p[n]$. Moreover, due to the high correlation of the channel factors on nearly located symbols, the estimated channel factors can be used to predict those on the other non-pilot positions. In this Lab, the pilot sequence will be allocated to the second OFDM symbol of each frame, and every two neighbour pilot symbols are separated with the same spacing[1], for example one pilot symbol every two subcarriers in Fig. 11.3. After channel estimation for pilot locations, you will interpolate them to predict the channel factors on all subcarriers. We also assume that the channel changes slowly such that the estimated channel factors are valid for all OFDM symbols within a frame, i.e., $H_k[n] = H_{k'}[n]$ for all k, k' and n. Hence, it's then straightforward to equalize the channel and decode the data symbols.

11.4 Practical Concerns

Due to its advantages at compensating frequency-selective channels without impacting the data rate, OFDM techniques have been widely adopted and standardized in many wireless communication systems, such as in cellular systems (4G LTE and 5G NR), wireless LAN (IEEE 802.11) etc. However, there are still some concerns from the perspective of practical deployment. In this section, we introduce two main problems that you need to consider while implementing OFDM on the USRPs.

11.4.1 Guard Carriers

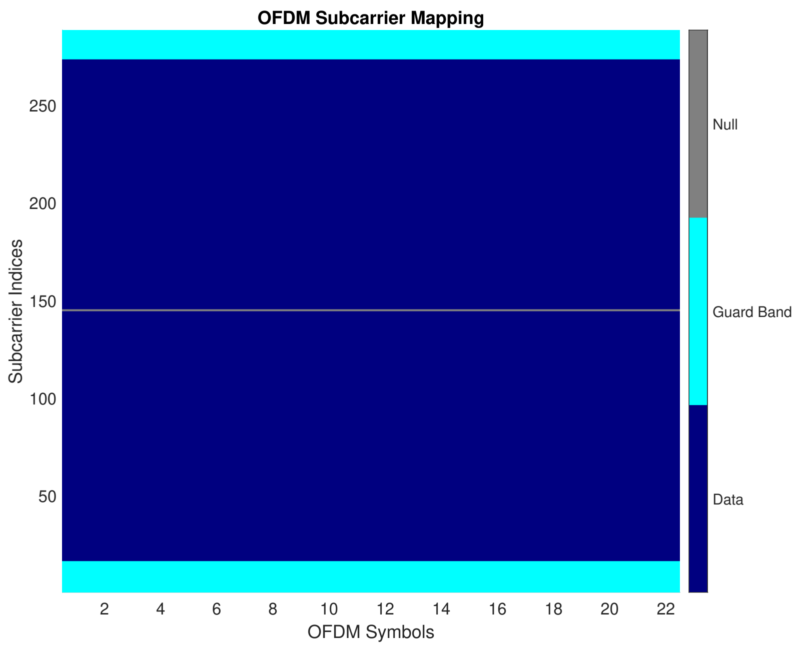
To avoid the out-of-band interference, the out-most subcarriers should be left to 0. Although there are more advance interference suppression techniques such as OFDM pulse shaping, we only focus on the most basic method in this Lab.

11.4.2 DC Component

In communication systems, due to the RF hardware limitation, the noise at the DC component, which corresponds to 0 Hz at baseband, is generally much more significant than other frequency components. This won't be a severe problem for single carrier systems, as each data symbol is carried by the whole bandwidth, but for OFDM, where the subcarriers take on different independent data, the DC subcarrier might be totally corrupted and lose the carried information. For this reason, we usually leave the DC subcarrier to zero, which is then ignored at the receiver after obtaining the OFDM grid.

Consequently, the OFDM grid with guard carriers and null DC subcarrier has the structure as illustrated in the following.

[1] There are two main reasons for that. The first one is to reduce the power consumption, and the second one is for the multi-user multiplexing, where the unallocated locations for one user will be used for the others such that their estimated channel factors are not effected by the pilots of others. But this is not the scope of this Lab.



11.5 Pre-Lab $\boxed{4}$

OFDM Basics

Consider a OFDM system with the settings:

- SCS $\Delta f = 5$ kHz;
- total number of subcarriers $N = 512$ including the guard carriers and the null DC component;
- number of guard carriers 16 on both sides of OFDM grid respectively;
- CP fraction of 0.1.

Please answer the following questions

1. How large is the bandwidth in Hz and the sampling time T_s in seconds?
2. What is the OFDM symbol duration T_0?
3. What is the CP length in seconds and in number of samples N_g (round to the nearest integer), how many samples does a OFDM symbol contains (including CP)?
4. What's the maximum delay should the channel spread such that Eq. (11.6) is guaranteed?
5. How many subcarriers in each OFDM symbol can be allocated by non-zero symbols?
6. How large is the FFT length N_{fft} when you modulate the OFDM grid?

Synchronization

1. Explain the reason that the method in Eq. (11.4) won't be impacted by CFO.
2. Describe briefly the principles of Schmidle Cox algorithm.
3. What training sequences can be used for the synchronization in this Lab?

Channel Estimation and Equalization

1. What could be the problems for the pilot allocation in this Lab when the channel changes dramatically?
2. How will you modify the frame structure to mitigate the problems?

Practical Concerns

1. Besides the introduced two problems, name at least another one problem of OFDM systems.

11.6 Lab Experiment 1 $\boxed{6}$

In this Lab, all the introduced algorithms will be implemented in Matlab. First, open the OFDM LabVIEW project in the folder `ofdm` and you can find three subVIs: `params_file_parser`, `Transmitter`, and `Receiver`. Then specify the parameter file path and all Matlab interfaces, which should be located in the folder `matlab_utils`.

`params_file_parser`

You need first to preprocess the parameters in the file `params_file_parser`, in which the N_{fft}, CP length and the sampling rate are computed based on your input parameter file. Note that the parameter `num_carriers` denotes the number of subcarriers without guard carriers and DC component. The resulting parameters will be incorporated with the input parameters and returned.

`Transmitter`

Open the Matlab file `Transmitter.m` and implement each step according to the instructions.

First, the modulated data symbols are already generated, you need to prepare a OFDM grid called `ofdm_grid` and map the data symbols onto it, from the third to the last OFDM symbol.

Next, you should map the generated pilot sequence onto the second OFDM symbol of the grid. The pilot symbols separated by `pilot_spacing` are saved in the vector variable `mapped_pilot_seq`.

Then, you should generate the synchronization sequence and map it. To do this, you need first to complete the function `gen_sync_preamble`, in which the generated sync sequence should be mapped onto the middle of the vector `mapped_sync_seq`. Then, in `Transmitter.m`, map the returned extended sync sequence onto the OFDM grid.

Finally, modulate the OFDM grid. You might want to use the Matlab object `comm.OFDMModulator`.

`Receiver`

In the last step, you will implement a whole OFDM receiver algorithm, from synchronization to data decoding.

First, please complete the function `joint_cfo_correct_frame_sync` to jointly estimate and correct the CFO and STO using the method introduced in Section 11.2.3.

Then, extract one frame based on the corrected receive samples and demodulate it to a OFDM grid. You might want to use the Matlab object `comm.OFDMDemodulator`.

Next, implement the functions `channel_estimation` and `channel_equalization` to estimate and equalize the channel in frequency domain. For the interpolation in frequency domain, you might want to use the function `interp1` in Matlab.

USRP Experiments

- Connect the USRPs to your computer as before, and run the transmitter and receiver subVIs in turn. Record the resulting plots;
- Replace the wire by antennas, test the performance of your program, record the results;
- Change the `msg_type` to `image` in `params.txt`, rerun the program, do you observe any problem? Can you explain it and how will you improve it (you don't need to implement it)?

12 Channel Coding (Day 8)

Wireless channels are prone to introduce errors to the transmitted signals. In order to satisfy reliability requirements, several error detection and error correction techniques have been developed.

Error correction schemes can be classified into two categories, namely backward error correction and forward error correction. In backward error correction, which is also called automatic repeat request (ARQ), acknowledgments and retransmissions provide reliable transmissions. An acknowledgment is a message sent from receiver to transmitter to confirm that the transmitted frame was received correctly. If the acknowledgment is not received by the transmitter until a specified time has passed, an event which is called timeout, the frame is retransmitted. The forward error correction (FEC) is a method for error control during data transmission, in which the sender transmits reconstruction-data to the receiver that are referred to as *redundancy*. Because FEC does not require a handshake between the transmitter and receiver, it can be used to transmit data to many recipients simultaneously from a single sender.

FEC codes can be classified into classical codes (e.g. Hamming codes, repetition codes, and Reed-Solomon codes) and modern codes (e.g. Turbo codes, LDPC codes, and Polar codes). In order to approach theoretical limits with classical codes, the complexity of the decoder increases exponentially and the design becomes physically impractical. This is the main reason behind the current popularity of modern codes. Modern codes split the decoding process into a number of manageable steps, and this enables powerful codes in a computationally feasible manner. Turbo codes and low-density parity check (LDPC) codes are examples [10, p. 644].

Historically, codes have been divided into block codes and convolutional codes. What makes convolutional codes distinguishable from block codes is existence of memory. The experiment today is going to focus on linear block codes.

12.1 Linear Block Codes

In order to review linear block codes, a few definitions are given next. A linear $[n, k, d]_q$ block code $\mathcal{C}$ is a k-dimensional linear subspace of the vector space $\mathbb{F}_q^n$ with minimum distance d, where q is the alphabet size.

The minimum distance directly determines the error detection and correction capabilities of a code.

A linear code $\mathcal{C}$ is an error-correcting code for which any linear combination of codewords is also a codeword, i.e., if $\mathbf{c}^{(1)}, \mathbf{c}^{(2)} \in \mathcal{C}$ then $a\mathbf{c}^{(1)} + b\mathbf{c}^{(2)} \in \mathcal{C}$, where $a, b \in \mathbb{F}_q$.

Every error correcting code has primary characteristics such as code length, dimension and minimum distance. For an $[n,\ k,\ d]_2$ binary block code we have:

- k information bits which are mapped into n coding bits.
- n encoded bits which is the length of the code.
- $n - k$ redundant bits in each codeword.
- The cardinality of the code $|\mathcal{C}| = 2^k$, that is the number of codewords in a code.
- The ratio of information bits to coding bits $R = k/n$, which is called code rate.

As the code (encoder/decoder) is linear, $\mathcal{C}$ can be defined as the row space of a carefully chosen $k \times n$ binary generator matrix $\mathbf{G}$. That is,

$$\mathcal{C} = \{\mathbf{uG} : \mathbf{u} \in \mathbb{F}_2^k\}.$$

The code $\mathcal{C}$ can also be defined as the null-space of an appropriate $(n-k) \times n$ binary matrix $\mathbf{H}$. That is,

$$\mathcal{C} = \{\mathbf{c} \in \mathbb{F}_2^n : \mathbf{Hc}^T = \mathbf{0}\}.$$

We obtain a systematic code if the generator matrix has the form $\mathbf{G}_s = [\mathbf{I_k} \,|\, \mathbf{A}]$, where $\mathbf{I_k}$ denotes the $k \times k$ identity matrix and $\mathbf{A}$ is a suitable chosen $k \times (n-k)$ matrix.

For a systematic code, having $\mathbf{G} = [\mathbf{I_k} \,|\, \mathbf{A}]$, the parity-check matrix can be defined as $\mathbf{H} = \left[-\mathbf{A}^\mathsf{T} \,|\, \mathbf{I_{n-k}}\right]$.

The minimum Hamming distance of a code says something about its decoding capability. In the following, we define the Hamming weight, Hamming distance, and minimum distance.

For any vector $\mathbf{a}$, the Hamming weight can be defined as the number of non-zero entries

$$wt\left(\mathbf{a}\right) = \sum_{i=1}^{n} \mathbf{1}\left(a_i \neq 0\right).$$

The Hamming distance $d_H\left(\mathbf{a}, \mathbf{b}\right)$ of two vectors $\mathbf{a}$ and $\mathbf{b}$ is defined as the number of dimensions, where the elements of the vectors differ

$$d_H\left(\mathbf{a}, \mathbf{b}\right) = wt\left(\mathbf{a} - \mathbf{b}\right) = \sum_{i=1}^{n} \mathbf{1}\left(a_i \neq b_i\right).$$

For a code $\mathcal{C}$, the minimum distance d can be defined as

$$d = \min_{\substack{\mathbf{c^{(i)}}, \mathbf{c^{(j)}} \in \mathcal{C} \\ i \neq j}} d_H(\mathbf{c^{(i)}}, \mathbf{c^{(j)}}).$$

The parameter d mostly shows how effective a code is in terms of error correction and detection. After the transmission of a codeword $\mathbf{c}$ which belongs to a code $\mathcal{C}$ that has minimum distance of d, the decoder can always *detect* the error with unique decoding if

$$wt\left(\mathbf{e}\right) < d - 1.$$

The number of errors that are guaranteed to *correct* with a distance d code can be given by

$$wt\left(\mathbf{e}\right) < \left\lfloor \frac{d-1}{2} \right\rfloor.$$

Suppose the codeword $\mathbf{c}$ is transmitted over the channel then the receiver observes

$$\mathbf{r} = \mathbf{c} + \mathbf{e},$$

where $\mathbf{e}$ is an error vector with ones at the position where an error has occurred and $+$ denotes a logical XOR operation. If $\mathbf{e}$ is a non-zero vector we obtain the syndrome $\mathbf{s}$ as follows.

$$\mathbf{Hr}^\mathsf{T} = \mathbf{H}(\mathbf{c}^\mathsf{T} + \mathbf{e}^\mathsf{T}) = \mathbf{s}^\mathsf{T}$$

The aim is to find the minimum weight solution of $\mathbf{e}$ for the given $\mathbf{s}$. Syndrome decoding is more efficient than nearest neighbour decoding. However, in this experiment we demonstrate nearest neighbour decoding because it is easier to implement.

Such a decoder chooses the codeword with the smallest Hamming distance to the received vector $\mathbf{r}$. If there exist more than one codeword at the same Hamming distance to $\mathbf{r}$, then the decoder chooses one randomly.

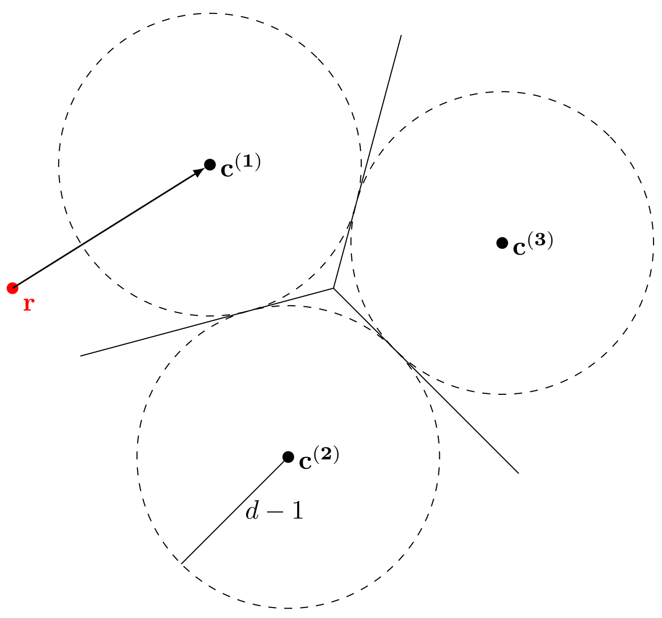


Figure 12.1: Nearest codeword decoding.

We can measure the reliability using the block error rate (BLER) or the bit error rate (BER) depending on the channel state (SNR). BLER is defined as the ratio between the number of erroneous blocks and the total number of transmitted blocks. BER is defined as the ratio between the number of erroneous bits and the total number of transmitted bits.

In Table 12.1, generator and parity-check matrices of simple linear block codes are given. During the first part of the experiment, these matrices will be used to generate the desired codes.

Table 12.1: Generator and parity-check matrices of well known systematic block codes

	Repetition Code $[\mathbf{n},\mathbf{1},\mathbf{n}]_\mathbf{2}$	**Single parity-check** $[\mathbf{n},\mathbf{n-1},\mathbf{2}]_\mathbf{2}$	**Hamming (7, 4)** $[\mathbf{7},\mathbf{4},\mathbf{3}]_\mathbf{2}$
G	$\begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & \dots & 0 & 1 \\ 0 & 1 & \dots & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$
H	$\begin{bmatrix} 1 & 0 & \dots & 0 & 1 \\ 0 & 1 & \dots & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$

In the second part of the experiment, we analyze Reed–Solomon (RS) codes which belong to a subclass of the general class of BCH codes. The RS code is a cyclic and linear block code and is characterized by the block length n, the dimension k and the minimum distance d_{min}. RS codes are suited for detecting and correcting burst data errors. If a symbol (here the symbol is an element of $\mathbb{F}_q$ and can be represented as a bit vector) has errors in more than one bit, that error still counts as one symbol error that can be corrected and this shows that Reed-Solomon codes can correct many bit errors. The RS code achieves the singleton bound.

12.2 Communication Scheme

In the first part of the experiment, we consider an additive channel where we induce an error vector into the channel. Here we consider a transmission system without modulation and demodulation. In the second part of the experiment, we consider a communication scheme as given in Fig. 12.2, where the channel is an AWGN channel. Depending on the choice of the error correcting code the probability of error can be made arbitrarily small.

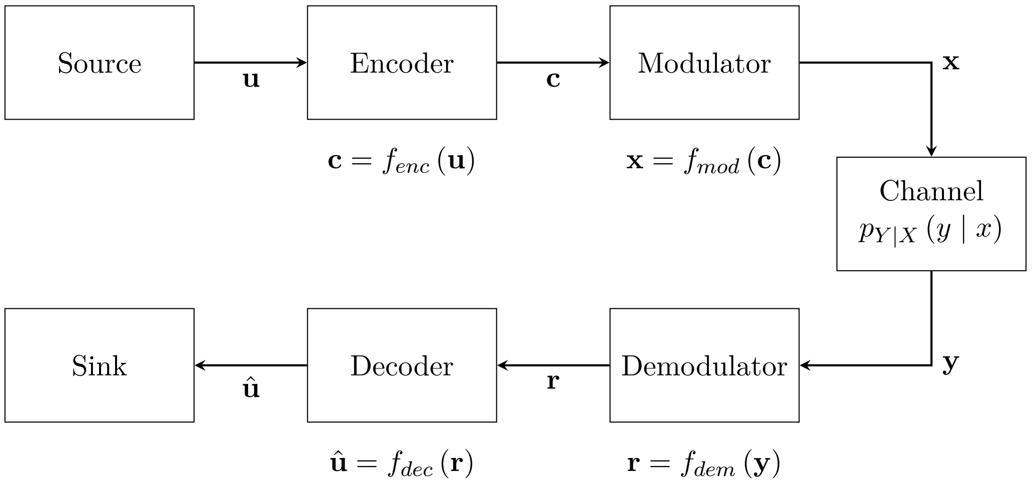


Figure 12.2: Block diagram of the communication system.

12.3 Pre-Lab $\boxed{2}$

Your protocol should include the answers for the following questions. A small overview of finite fields and of Reed-Solomon codes can be found here: https://mathworld.wolfram.com/FiniteField.html and https://www.cs.cmu.edu/~guyb/realworld/reedsolomon/reed_solomon_codes.html.

1. All coding schemes add some redundancy to the actual information symbols to provide a reliable transmission by increasing distance between codewords. What behavior do you expect to see when you increase the block length but leave the minimum distance of the code unchanged during transmission over a binary symmetric channel BSC(p), where p is the crossover probability?
2. How does the generator matrix of a $[7, 3, 5]$ Reed-Solomon code over $\mathbb{F}_{2^3}$ look like? How many errors can it detect and how many correct? Give the elements of the field of the code in power representation, polynomial representation and bit representation.
3. Enter in a block diagram the decoding scheme of the RS decoder.

12.4 Lab Experiment $\boxed{8}$

The lab experiment consists of two parts. The goal of the first part is to become familiar with the coding modules in LabVIEW and to see how the received bits differ from the transmitted bits with and without a channel code, by analyzing the bit error rate (BER) as a function of the SNR. For this we also use an optimal error correcting code, the Reed-Solomon (RS) code. Using MSK/QAM we modulate the encoded data and transmit it over an AWGN channel.

In the second part of the experiment we want to analyze the performance of RS codes when USRPs are used to transmit the data over the air.

12.4.1 Experiment 1

Part 1 First open the project `Coding.lvproject` and consider the functions of the project. In EncDecTest.gvi you can enter any $k \times n$ generator matrix from Table 12.1 and any binary information bit sequence of length $i \cdot k$ with $i \geq 1$ and run the project. You have to enter an error vector even if it is an all-zero vector, i.e. even if a noiseless transmission is to be simulated. Next, you can add any error vector and consider the decoded bit vector.

Task 1: Describe the working principle of `Nearest codeword decoding.gvi` in words or draw a control flow of the decoding structure.

Part 2 Now we want to analyse the BER of a coding scheme where we use $[127, 111]$ and $[255, 239]$ RS codes and an AWGN channel. Open a new project and call it RS-AWGN.

Task 2:
a) Design and implement a communication system as illustrated in Fig. 12.2, where the code is an RS code, the modulation is MSK and the channel is an AWGN channel.
b) To measure the BER we have to compare the transmitted data with the received data. Calculate the BER of your coding scheme.
c) Design a BER calculator in LabVIEW.
d) Plot the curve BER(SNR) when using no coding, an $[127, 111]$ RS code, and an $[255, 239]$ RS code for SNR $= 5, \dots, 15\,\text{dB}$.
e) What can you observe?

For Task 2 use the blocks provided by LabVIEW. You will need the VI's given in figures 12.3, 12.6, 12.4, 12.5, 12.7, 12.8 and the MT Pack Bits and MT UnPack Bits, which packs data to an unsigned or signed integer array and wise versa, respectively. The last two blocks are respectively placed before and after the encoder and decoder. The VI's listed below are not all from the current version. You can find the VI's in the search field in LabVIEW.

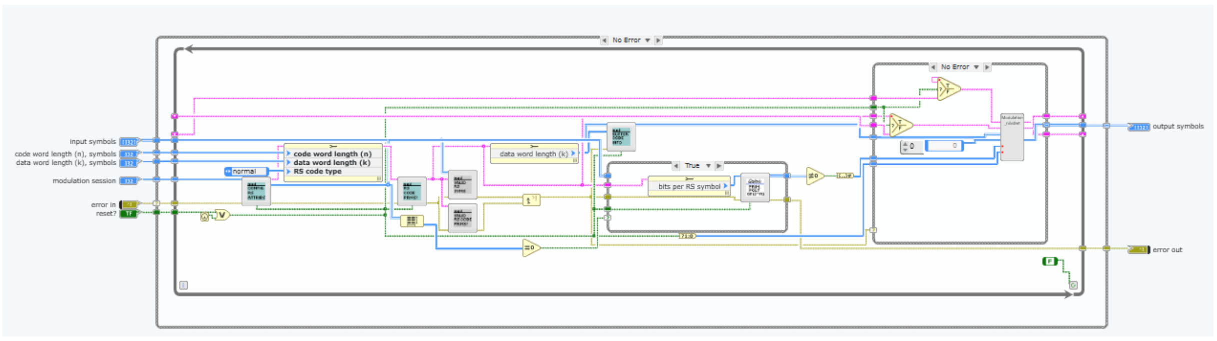


Figure 12.3: RS encoder.

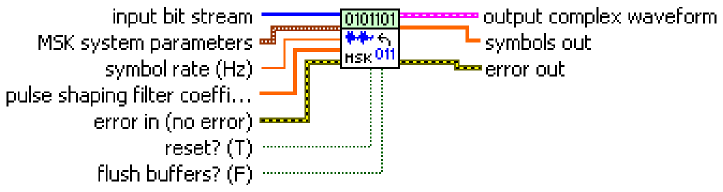


Figure 12.4: MSK modulation.

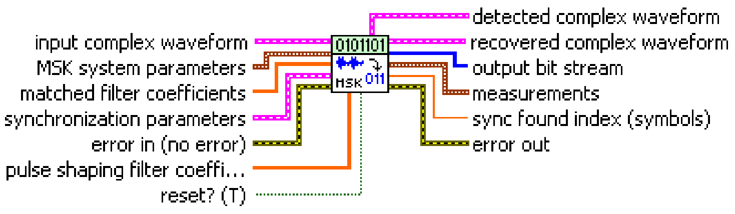

input complex waveform
MSK system parameters
matched filter coefficients
synchronization parameters
error in (no error)
pulse shaping filter coeffi...
reset? (T)
detected complex waveform
recovered complex waveform
output bit stream
measurements
sync found index (symbols)
error out

Figure 12.5: MSK demodulation.

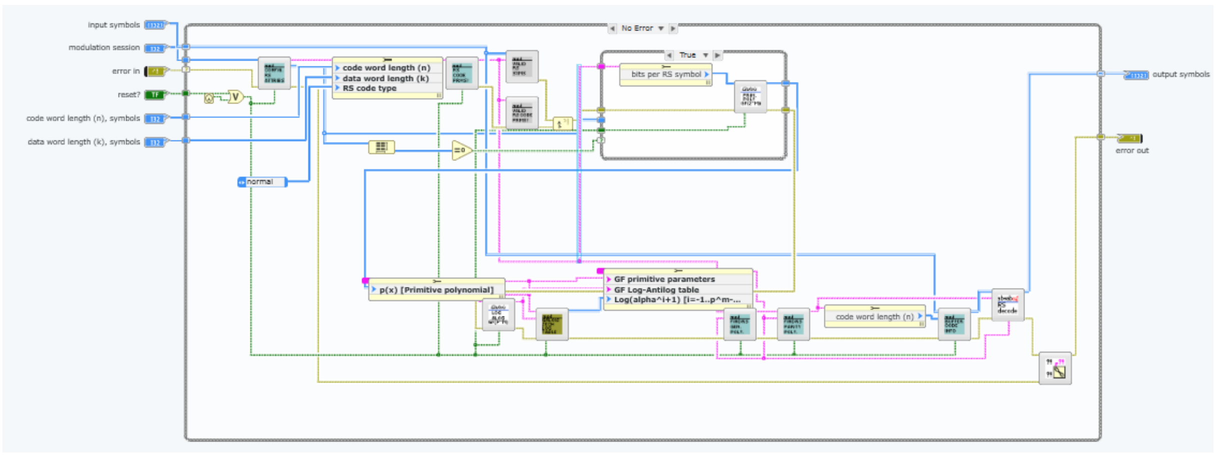


Figure 12.6: RS decoder.

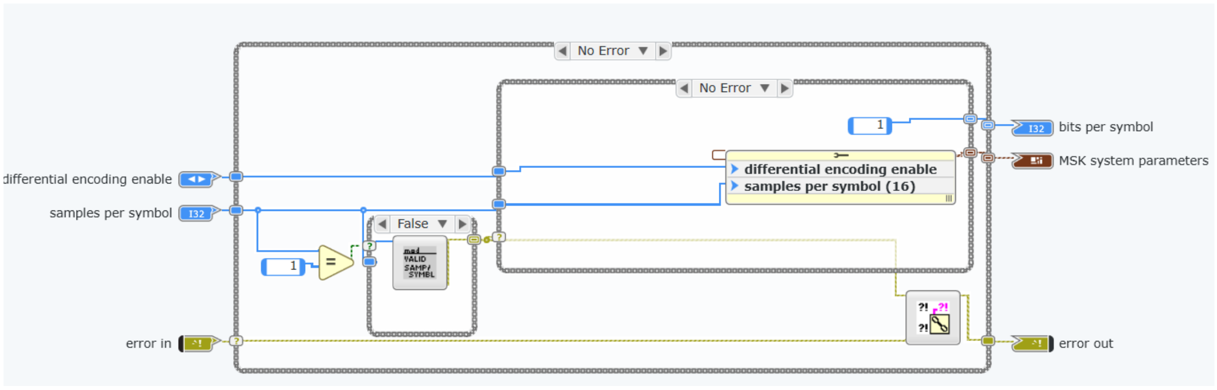


Figure 12.7: Generate MSK system parameters.

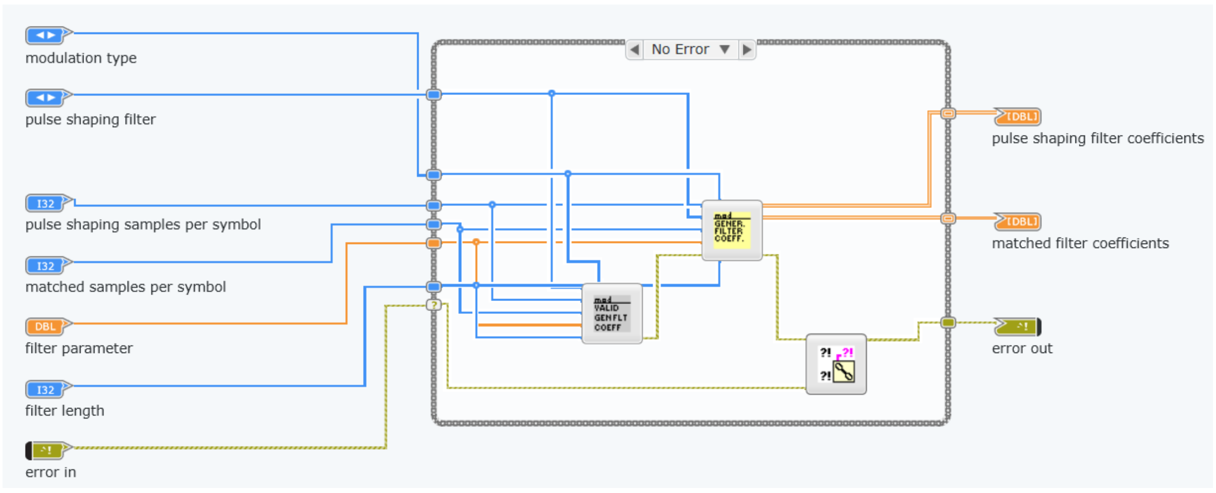


Figure 12.8: Generate filter coefficients.

12.4.2 USRP Experiment

In this part, you will see the difference in performance between uncoded and coded transmission. To this end, you will first have to do the following:

(i) Download the material from moodle and unpack the archive. There you will find two sub-folders `usrp_code` and `matlab_utils`.

(ii) Now open `usrp_code` and start `matlab_interfacing.lvproject`. Make sure that `tx_usrp.gvi` and `rx_usrp.gvi` are open.

(iii) At last make sure that the controls in the LabVIEW front panel of each VI correspond to the values given in Table 12.2.

Parameter	Tx	Rx
`params_file`	`params.txt`	`params.txt`
Device name	`USRP_<groupID>_A`	`USRP_<groupID>_B`
Active antenna	TX1	RX2
Carrier frequency	2 GHz	2 GHz
IQ rate	640 kS/s	640 kS/s
Gain	20	40
ACQ time	-	1 s

Table 12.2: Parameters of the communication system.

Now we are ready to compare the uncoded transmission with the coded one. Now do following steps:

1. Go to the folder `matlab_utils` and open the file `params.txt`.
2. Locate the field `"channel_coder"` and set its value to `"none"`. This implements the USRP transmission without channel coding.
3. Now run the `tx_usrp.gvi`, wait for a second, and then run `rx_usrp.gvi`.
4. After running the two VIs a MATLAB figure should pop up. Look at the bottom right of this figure and locate, where the BER is indicated.

Now repeat the steps 2 to 4, but set the value of the `"channel_coder"` field to `"rs"`. This applies a $(15, 4)$ Reed-Solomon Code to the message bits before transmission.

Task 3:

1. What BER do we achieve in both cases? Which scheme performs better?
2. Now set the transmission gain to a lower value and keep the receive gain fixed, say between 0 and 10. Now do it the other way around. What do you expect to happen in terms of BER? Does this match your observations?
3. Now reset the gain values to their initial values and look at the number of samples transmitted in each scheme. Which one is less bandwidth efficient (i.e. has bigger effective data rate) and why?
4. How can we improve this?

13 Security and Polar Codes (Day 9)

The objectiv of this lab session is on the one hand to introduce the concept of physical layer security and to implement it with the usage of a modular coding scheme. Physical layer security uses the noise statistics of the transmission channels to create security. In contrast to computational based security, no restrictions are set for the attacker. In cryptography, for example, the attacker's computational power is limited. In addition, we will introduce the Polar Code as error correcting code in order to use it in the programming tasks later on.

13.1 Physical Layer Security

13.1.1 Seeded modular Coding Scheme

The wiretap channel model as introduced by Wyner [11] is the underlying information theoretic model we will be studying in this lab. In the wiretap channel model, the sender (Alice) wants to securely transmit the message m to the legitimate receiver (Bob). To achieve this secure transmission, Alice applies randomized encoding to obtain a codeword x from the message m. This codeword is then transmitted over the channel T to Bob. Bob receives the channel output $y = T(x)$ and tries to recover m. The attacker (Eve) also receives a channel output z over the channel E, from which she tries to extract as much information as possible. Note that the channel outputs y and z are different. Thus, there are two code requirements: 1) reliable message transmission from Alice to Bob 2) message transmission should be secure, which means that Eve obtains as little information as possible about the transmitted message.

One possibility to fulfill both requirements is the use of a seeded modular coding scheme, which is shown in Figure 13.1. The advantage of this scheme is that the realization of the requirements takes place in two different blocks. On the one hand, reliable message transmission is implemented in the Coding Layer, where standardized error-correcting codes can be used. On the other hand, the security aspect is realized with the help of a Security Layer.

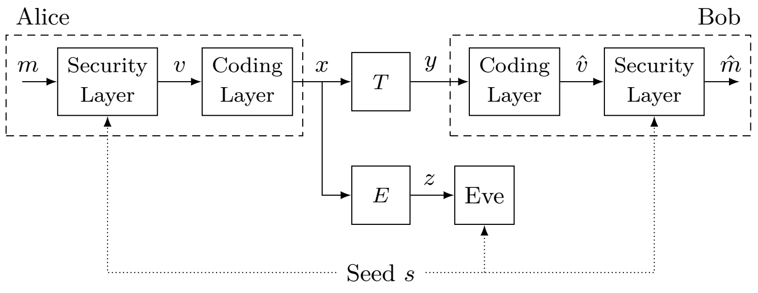


Figure 13.1: Seeded modular coding scheme for the wiretap channel (T, E), consisting of a coding layer and a security layer.

The Security Layer is determined by a function f_s, where $s = (a, t)$ is a randomly chosen seed accessible to all three parties. The two components of the seed, a and t are bit strings of length l and randomly chosen from $\{0,1\}^l \setminus \{0\}^l$ and $\{0,1\}^l$. For a message $m \in \mathcal{M} = \{0,1\}^k$ and a random bit string $r \in \{0,1\}^{l-k}$, $k < l$, we define the mapping $f_s^{-1} : \{0,1\}^k \times \{0,1\}^{l-k} \to \{0,1\}^l$ for the Security Layer on Alice's side as follows:

$$f_s^{-1}(m, r) = a^{-1} * ((m||r) \oplus t), \tag{13.1}$$

where $m||r$ denotes the concatenation of the bit strings m and r, a^{-1} is the inverse, $*$ the multiplication, and $\oplus$ the addition in the corresponding field $GF(2^l)$. At the security layer on Alice's side, for a given message $m \in \mathcal{M}$, we randomly choose a bit string r and compute $v = f_s^{-1}(m, r)$. The bit string v is then further processed by the Coding Layer.

At the Security Layer on Bob's side we have to reverse the action of f_s^{-1}. To this end, we use the mapping $f_s \colon \{0,1\}^l \to \{0,1\}^k$, defined by

$$f_s(\hat{v}) = [(a * \hat{v}) \oplus t]_k \, , \tag{13.2}$$

where $[x]_k$ denotes the operation of selecting the first k bits of x. f_s is applied on the output $\hat{v}$ of the Coding Layer, which results in $\hat{m} = f_s(\hat{v})$. If the transmission over the channel T has been error free, i.e. if $\hat{v} = v$, then we have $\hat{m} = m$ because $f_s(f_s^{-1}(m, r)) = m$ for all s, m, and r.

13.1.2 Performance Evaluation

Eve should learn as little as possible about the message m when examining its channel output. To measure the achieved security, we employ the distinguishing security (DS) metric for a fixed seed s and a message pair $m_1, m_2 \in \mathcal{M}$

$$Adv(s, m_1, m_2) = \max_{\mathcal{A}} 2 \Pr[\mathcal{A}(s, m_1, m_2, z_\Theta(m_1, m_2)) = \Theta] - 1, \tag{13.3}$$

with the set of all attack strategies $\mathcal{A}$, a uniformly distributed random variable $\Theta \in \{1, 2\}$ and the channel output $z_\Theta(m_1, m_2)$, where $\Theta \in \{1, 2\}$ indicates the transmitted message m_1 or m_2. Distinguishing security can be seen as a game. Eve receives the message pair, the seed and the channel output with random bit $\Theta \in \{1, 2\}$. Now, she has to choose a bit $\hat{\Theta} \in \{1, 2\}$ and wins the game if $\Theta = \hat{\Theta}$. The closer the DS metric is to zero, the more secure is the security system.

The distinguishing error rate DER_E is given by

$$DER_E(s, m_1, m_2) = \min_{\mathcal{A}} \Pr[\mathcal{A}(s, m_1, m_2, z_\Theta(m_1, m_2)) \neq \Theta]. \tag{13.4}$$

Since (13.3) is equal to $1 - 2 \cdot DER_E$ means that DER_E close to 1/2 implies "high security". Operationally means DER_E close to 1/2 that Eve cannot reliably distinguish any two messages and the best attack strategy does not perform better than pure guessing.

It is important to stress the difference in the decoding strategies of Eve and Bob. Eve has to decide which one out of two given messages was sent. In contrast, Bob decodes ordinarily without this additional information and tries to determine which message out of the set of all possible messages $\mathcal{M}$ was sent. Hence, the decoding task of Bob in our setting is more intricate than the decoding task of Eve and Bob's block error rate is given by

$$BLER_B = \Pr(\hat{m} \neq m) \, . \tag{13.5}$$

13.2 Polar Codes

Polar Codes were introduced by [12] and are deployed in the 3GPP (3rd Generation Partnership Project) standardization for 5G new radio (5G-NR) for uplink and downlink control channel. The reason herefore is its explicit construction (no ensemble to pick from) and efficient encoding and decoding algorithms. Additionally, polar codes have absence of an error floor and are capacity-achieving over binary input symmetric memoryless channels. The name deduces from its feature to polarize channel bits into good and bad ones.

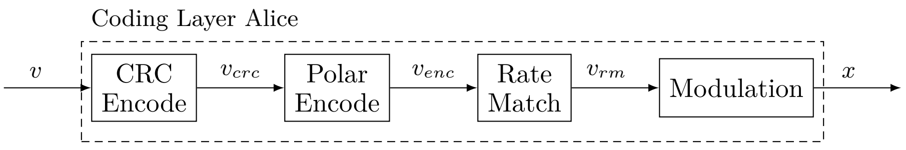


Figure 13.2: Alice's Coding Layer consisting of CRC Encode, Polar Encode, Rate Match and Modulation.

13.2.1 Encoding

Encoding consists of Cyclic Redundancy Check (CRC) Encode, Polar Encode, Rate Match and Modulation as shown in Figure 13.2. The CRC-bits are appended to the l-bits, we want to encode. The additional information from the CRC-bits is used as a codeword selection mechanism. This helps to improve the error correction performance of the successive cancellation list decoder. The decoder output is chosen by the path relative to the smallest path metric, which is additionally fulfilling the CRC constraint. The resulting $v_{crc} \in \{0,1\}^{l_{crc}}$ is the input of the polar encoding step. The polar encoder outputs a vector $v_{enc} \in \{0,1\}^N$ of length N. Since the Polar Code polarizes the bit channels, the best l_{crc} channels are used for the bits of the vector v_{crc} and the worst $N - l_{crc}$ channels are frozen, which means that they are set to zero. The resulting $v'_{crc} \in \{0,1\}^N$ is encoded as follows

$$v_{enc} = v'_{crc} G_N \tag{13.6}$$

with the generator matrix $G_N = F^{\otimes n}$, with $F = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, $n = log_2(N)$ and $\otimes$ be the Kronecker product, respectively.

To illustrate this procedure, we choose a vector $v = \begin{pmatrix} 1 & 1 & 0 & 1 \end{pmatrix}$ and for the sake of simplicity we say $v = v_{crc}$. In the 5G-NR standard exists a table of all bit channels between $1 - 1024$, which are listed from the channel with the worst reliability to the channel with the best reliability. For example the first 16 bit channels have the order $(1, 2, 3, 5, 9, 4, 6, 10, 7, 11, 13, 8, 12, 14, 15, 16)$. While our example is for $N = 8$, the channel order here is $(1, 2, 3, 5, 4, 6, 7, 8)$, where the frozen bits are set to the first $N - l_{crc} = 4$ channels and the bits of the vector v_{crc} to the last $l_{crc} = 4$ bits. After applying (13.6) we get $v_{enc} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$.

We see in the example that the output v_{enc} has a fixed length N to the power of two. If we want to transmit messages with a different length $E \neq N$, we have to change the length N of the vector v_{enc}. This is done with rate matching via interleaving, puncturing or shortening and result in the vector $v_{rm} \in \{0,1\}^E$. In the last encoding step in Alice's Coding Layer, the modulation is applied on v_{rm}, which outputs $x \in \mathbb{C}^c$ with $c = E/j$ and modulation order j.

13.2.2 Decoding

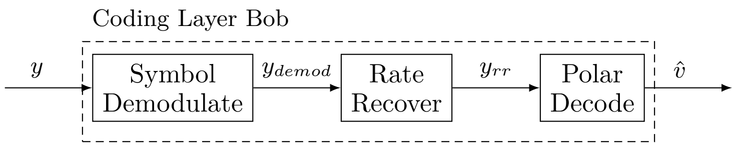


Figure 13.3: Bob's Coding Layer consisting of Symbol Demodulate, Rate Reover and Polar Decode.

Decoding consists of Symbol Demodulate, Rate Recover and Polar Decode as shown in Figure 13.3. The symbol demodulation outputs the log likelihood ratios of the transmitted E bits in the vector $y_{demod} \in \mathbb{R}^E$. Then, y_{demod} is rate recovered, which is the inverse function of rate matching and outputs $y_{rr} \in \mathbb{R}^N$. Finally, the rate recovered y_{rr} is given into the polar decoder.

A well known polar decoding algorithm is called successive cancellation (SC) and is as well proposed in [12]. The downside of the SC-decoder is its mediocre error-correction performance at practical code length. To overcome this issue, [13] introduced a list-based decoding approach for polar codes (SCL). The idea is to run a group of SC-decoders in parallel. Every time the decoder has to make a hard decision, every possibile hard desicion is stored in a list as codeword candidates. For every codeword candidate, a path metric is calculated. If the maximum limit of possible codeword candidates is reached and the decoding algorithm has not finished, the codeword with the highest path metric is discarded. After finishing the algorithm, the codeword with the lowest path metric is displayed. The error-correction performance of SC is increased with this SCL procedure substantially by codewords of moderate length. This increase is especially high when the code is concatenated with a CRC.

13.3 Pre-Lab $\boxed{4}$

Your protocol should include answers to the following questions.

1. What are the code requirements in the Wiretap Channel Model and what is the advantage of the modular coding scheme compared to other schemes?
2. What is the difference in physical layer security and computational based security? If there is a difference, what security would you prefer and why?
3. Does the performance of Bob change with the security layer? Explain your answer. If the performance changes, does it get better or worse?
4. Does the performance of Bob/Eve change if k is changed and all other parameters are fixed?
5. In addition to the Polar Code, which error-correcting code is used in the 5G-NR standard? What could be a disadvantage of the SCL decoder?
6. For the same parameters and $v = \begin{pmatrix}1 & 1 & 0 & 1\end{pmatrix}$ as in the example in Subsection 13.2.1, encode for $N = 16$. Give v'_{crc} and v_{enc} for this case.

13.4 Lab-Experiment $\boxed{6}$

In this lab experiment you will get a virtual machine, where all files are included. With the user profile "student" and the password "sdrlab" you can log in to this machine. MATLAB can be opened via the terminal. In the terminal one enters the commands: "cd", "cd ..", "cd ..", "cd usr/local/MATLAB/R2021a/bin/" and "./matlab". MATLAB R2021a should be opened now.

We will implement the inverse $a^{-1} \in \{0,1\}^l \setminus \{0\}^l$ and multiplication in $GF(2^l)$ efficiently. The algorithms are given for only special values of l since the polynomial

$$\phi(X) = X^l + X^{l-1} + ... + X^2 + X + 1 \tag{13.7}$$

is only an irreducible cyclotomic polynomial in GF(2)[X] if

- $l + 1$ is prime, and
- 2 is a primitive root modulo $l + 1$

Note that in the Implemented algorithms the MSB is on the left side of the vector, i.e. in MATLAB at vector position 1. Additionally we want to note, that the initialized polynomials and the irreducible cyclotomic polynomial are always of degree $l+1$. But since input polynomials are of degree l, they have to be extended to degree $l+1$. To output polynomials of degree l again, the output has to be shortened in the last step.

a	b	c
0 0 0 1	0 0 0 1	0 0 0 1
0 0 1 1	0 0 0 1	0 0 1 1
0 0 0 1	0 0 1 1	0 0 1 1
0 0 1 1	0 0 1 0	0 1 1 0
0 0 1 1	1 0 0 0	0 1 1 1
1 0 0 0	0 0 1 0	1 1 1 1
1 0 0 0	1 0 0 0	0 0 1 0
1 1 1 1	1 1 1 1	1 0 0 0

Table 13.1: Test inputs and outputs for Algorithm 1

13.4.1 Multiplication

Implement Algorithm 1 in "Multiplication.m" and test it with Table 13.1:

Algorithm 1 Multiplication in $GF(2^l)$

INPUT: $a(X), b(X)$ - polynomials of degree l for multiplication, where $a[i], b[i] \in \{0, 1\}$ with $i \in \{0, 1, ..., l-1\}$ refers to the coefficient of X^i

OUTPUT: $c(X) = a(X)b(X) \mod \phi(X)$ -product in $GF(2^l)$

```
 1: $c(X) := 0$                              {Initialization of the polynomial $c$}
 2: $a[l] := 0; b[l] := 0; c[l] := 0$        {Extension to polynomials of degree $l+1$}
 3: for $i = 0$ to $l$ do
 4:   if $a[i] = 1$ then
 5:     $c = c \oplus b$
 6:   end if
 7:   cyclic shift $c$ right 1 bit
 8: end for
 9: if $c[l] = 1$ then
10:   $c = c \oplus \phi$
11: end if
12: remove $c[l]$                            {Reduction to polynomial of degree $l$}
13: if $\sum c = 0$ then
14:   $c[0] = 1$
15: end if
```

13.4.2 Inverse

Implement Algorithm 2 in "Inversion.m" and test it with Table 13.2:

13.4.3 Implementation of the Coding and Security Layers

Matlab Simulation

Implement the Coding and Security Layers in the MATLAB-file "SDR_LTI_Lab.m" (see comments) and use the file to simulate the $BLER_B$ and DER_E for different SNR_B and SNR_E. In case you could not implement the Algorithms 1 or 2 correctly, use the provided functions "c=Multiplication_LTI_SDR_Lab(a,b)" or "b=Inverse_LTI_SDR_Lab(a)". For the Coding Layers use the functions provided by MATLAB based on the 5G standard. The variables you need for the functions are provided in the variable "input_MATLAB_functions". What do you observe?

Algorithm 2 Inversion in $GF(2^l)$

INPUT: $a(X)$ - polynomial of degree l for inversion, where $a[i] \in \{0,1\}$ with $i \in \{0,1,...,l-1\}$ refers to the coefficient of X^i
OUTPUT: $b(X) = a(X)^{-1} \mod \phi(X)$ -inverse vector in $GF(2^l)$

```
1:  $k := 0; b(X) := 0; c(X) := 0;$     {Initialization of the polynomials $b, c$ and the variable $k$}
2:  $b[0] = 1;$
3:  $a[l] := 0; b[l] := 0; c[l] := 0;$     {Extension to a polynomial of degree $l+1$}
4:  $f = a; g = \phi;$     {Initialization of the polynomials $f$ and $g$}
5:  while $f[0] = 0$ do
6:    cyclic shift $f$ right 1 bit
7:    $k = k + 1$
8:  end while
9:  while $(\sum f \neq 0)$ or $(\sum f = 1$ and $f[0] \neq 1)$ do
10:   if degree of $f <$ degree of $g$ then
11:     exchange $f$ and $g$
12:     exchange $b$ and $c$
13:   end if
14:   $f = f \oplus g$
15:   $b = b \oplus c$
16:   while $f[0] = 0$ and $\sum f \neq 0$ do
17:     cyclic shift $f$ right 1 bit
18:     cyclic shift $c$ left 1 bit
19:     $k = k + 1$
20:   end while
21: end while
22: cyclic shift $b$ right $k$ bit
23: if $b[l] = 1$ then
24:   $b = b \oplus \phi$
25: end if
26: remove $b[l]$     {Reduction to polynomial of degree $l$}
```

a	b
0 0 0 1	0 0 0 1
0 0 1 0	1 1 1 1
0 0 1 1	1 0 1 0
0 1 0 0	1 0 0 0
1 0 0 0	0 1 0 0
1 1 0 0	1 1 0 1
1 1 1 1	0 0 1 0

Table 13.2: Test inputs and outputs for Algorithm 2

What changes if you switch the parameters $l = 18$ and $k = 8$ ("n32_l18_k8_LTI_SDR_Lab.mat") to $l = 18$ and $k = 2$ ("n32_l18_k2_LTI_SDR_Lab.mat")? Illustrate your observation in a convenient plot.

Hardware

Implement the Decoding and Security Layer in the MATLAB-file "SDR_LTI_Lab_generate.m" (see comments). Illustrate, that you can transmit and receive with this MATLAB-file. For this, the USRPs have to be connected to the computer and linked to the virtual machine. Right-click on the USB icon at the bottom right of the Virtual Machine and make sure that the USRPs are connected (see Figure 13.4). For decoding we have already received data in the folder "data". Implement Decoding and Security Layer in "SDR_LTI_Lab_decode.m" (see comments) and show the observations in a convenient plot.

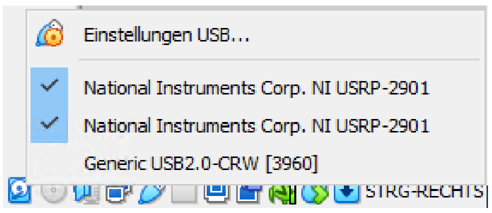

Einstellungen USB...
✓ National Instruments Corp. NI USRP-2901
✓ National Instruments Corp. NI USRP-2901
Generic USB2.0-CRW [3960]
STRG-RECHTS

Figure 13.4: Connecting USRPs to the Virtual Machine.

14 Implementation of Full Transmitter and Receiver Chain (Day 10)

In this experiment we will revisit the sections already studied and look at solving the stated problems from a new perspective. The results obtained after implementing these algorithms are to be compared against the performance obtained with the already-provided solutions and with those previously implemented by the student.

14.1 Building Your Own Communication System using MATLAB $\boxed{5}$

The following experiment is dedicated to piecing together what you have learned in the past sessions into one functioning SISO communication system. You will not have any pre-lab task only the lab experiment.

14.1.1 USRP Setup

First of all create a LabVIEW project containing your `groupID`. Inside, create two VIs called `Tx_USRP.gvi` and `Rx_USRP.gvi`. In the former you will have to implement the transmitter chain, in the latter the receiver chain.

In each of the aforementioned VIs, write LabVIEW code to establish USRP sessions, to write data to and fetch data from the radios, respectively, to configure the signals, etc. For reference, take a look at Section 3, to see what you have to do in order to ensure a minimal working setup.

14.1.2 Transmitter Chain

The transmitter chain should contain following data processing steps:

1. **Payload Generation**
   Here you have to generate the bit string, that you want to transmit through the channel. You are free to choose whatever you want to send, it would be nice if you would use some text, as it can be easily visualized.

2. **Channel Coding** (optional)
   Here you can implement a channel coding if you want. This is not required for the successful completion of the experiment.

3. **Symbol Mapping**
   Choose a suitable modulation scheme and implement it. This should take the (coded) bits and map them to a vector of I/Q samples. We recommend to use M-PSK, but you can experiment with something else.

4. **Frame Synchronization Sequence**
   Choose one type of synchronization sequence that you used in the lab, implement a method to generate it and to append it to the vector of data I/Q samples.

5. **Pilot Sequence** (optional)
   Do the same thing as above for the pilot sequence for channel estimation, if you wish to implement channel estimation and equalization. This is not required for the successful completion of the experiment.

6. **OFDM Modulation** (optional)
   If you wish to implement an OFDM system, at this stage you should map your data, the frame synchronization sequence and everything else needed onto the resource grid, then apply subsequent steps such as IFFT and appending the CP.

7. **Pulse Shaping**
   Filter the I/Q samples with a pulse shaping filter. The design of the pulse shaping filter is up to you. We recommend to use a root raised cosine filter. If you are implementing an OFDM system you may skip this step.

14.1.3 Receiver Chain

If you want to implement a single-carrier system, the receiver chain should contain following data processing steps:

1. **Carrier Synchronization**
   Choose one carrier synchronization method and implement it. You might want to look into what you have done already.

2. **Matched Filtering**
   Implement the matched filter that is matched to the pulse shaping filter in the transmitter. The design is up to you. We recommend to use a root raised cosine filter with the same parameters as the one in the transmitter.

3. **Symbol Synchronization**
   Implement timing synchronization. Make sure that at this step you also downsample the received signal.

4. **Frame Synchronization**
   Implement a correlation-based synchronizer as you did it in the lab. Make sure to either generate the synchronzation sequence on-the-fly or have it somewhere stored.

5. **Channel Estimation and Equalization** (optional)
   Optionally implement channel estimation and equalization. Make sure to have the pilot sequence of your choice somewhere stored or to generate it on-the-fly. Please use the LS channel estimator. For equalization, you can either employ a method discussed in the lab or one which was not discussed, such as LMMSE equalizer.

6. **Symbol Demapping**
   Choose a suitable demodulation scheme and implement it. This should take the vector of I/Q samples and map them to (coded) bits . We recommend to use M-PSK, but you can experiment with something else.

7. **Channel Decoding** (optional)
   Implement the channel decoding algorithm for the one coding in the transmitter if you want. This is not required for the successful completion of the experiment.

8. **Payload Parsing**
   If you choose to transmit a serialized text, make sure that you convert the decoded bits to text again. This part should also contain some code to compute the bit error rate (BER).

If you plan on implementing an OFDM system, your receiver processing should look as follows:

1. **Joint Carrier Frequency Offset Compensation and Frame Synchronization**: Please refer to the experiments in Day 7.
2. **OFDM Demodulation**: Implement OFDM demodulation steps such as CP removal and FFT. Please use as much code from Day 7 as possible.
3. **Channel Estimation and Equalization** (optional): Implement a channel estimator and one equalizer of your choice. You might want to start with what you have implemented already.
4. **Symbol Demapping**
   Choose a suitable demodulation scheme and implement it. This should take the vector of I/Q samples and map them to (coded) bits . We recommend to use M-PSK, but you can experiment with something else.
5. **Channel Decoding** (optional)
   Implement the channel decoding algorithm for the one coding in the transmitter if you want. This is not required for the successful completion of the experiment.
6. **Payload Parsing**
   If you choose to transmit a serialized text, make sure that you convert the decoded bits to text again. This part should also contain some code to compute the bit error rate (BER).

14.1.4 Visualization

Make sure to have a nice visualization of your system. It is not necessary to plot all intermediate signals at each processing step at the transmitter and the receiver, but it is useful for debugging purposes.

If you chose to transmit serialized text, please display it before the transmission in `Tx_USRP.gvi` and after reception in `Rx_USRP.gvi`. A BER measurement should also be included.

14.1.5 General Recommmendations

Here you find some useful tips for completing your experiment as fast as possible:

- Reuse as much code from the previous experiments as possible.
- We recommend you write the transmitter and receiver logic in MATLAB and only use LabView for the hardware setup.

14.2 Testing Your System $\boxed{5}$

For the last laboratory session, your final results must be obtained replacing the coaxial cable used so far by the omnidirectional antennas provided by your instructor, selecting one available 5-MHz channel from the 2.4-GHz ISM band listed in Table 14.1, and evaluating the performance of your implementation by performing a transmission/reception with both ends at approximately 1 m of distance from each other (please provide the exact distance).

Channel	f_c in MHz
1	2412
2	2417
3	2422
4	2427
5	2432
6	2437
7	2442
8	2447
9	2452
10	2457
11	2462
12	2467
13	2472
14	2484

Table 14.1: List of central frequencies f_c in the 2.4-GHz ISM band.

Your final protocol should include:

- A plot of the received signal, including a discussion of the reasons for:
  - your selection of the RF frontend gain values
  - how the sampling rate that you selected defines the bandwidth of the digital signal
  - the potential presence of clipping in the incoming signal
  - avoiding signal clipping in the downconverted digital signal
- The resulting constellation diagram after each procedure described in Fig. 7.1, where the operation carried out by each block is briefly described.
- The final BER achieved by your implementation.
- Elaborate how you would expand this implementation in order to obtain the SNR as part of your KPI measurements (you do not have to implement it).
- Indicate the relationship between SNR and E_b/N_0.

A Signals and Systems Basics

Please note that in this whole section, we use the angular frequency $\omega = 2\pi f$ for notational convenience.

A.1 Fourier Transform

The Fourier transform is an important operator in signal processing. The Fourier transform $\hat{x}$ of a signal x is defined by the equation

$$\hat{x}(\omega) = (\mathcal{F}x)(\omega) = \int_{-\infty}^{\infty} x(t)\, \mathrm{e}^{-i\omega t}\ \mathrm{d}t.$$

The inverse Fourier transform is given by

$$x(t) = (\mathcal{F}^{-1}\hat{x})(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{x}(\omega)\, \mathrm{e}^{i\omega t}\ \mathrm{d}\omega.$$

Some symmetry properties of the Fourier transform are summarized in Tab. A.1, and important transform pairs in Tab. A.2. A particularly important equality given by the Plancherel theorem

$$\int_{-\infty}^{\infty} |x(t)|^2\ \mathrm{d}t = \frac{1}{2\pi}\int_{-\pi}^{\pi} |\hat{x}(\omega)|^2\ \mathrm{d}\omega, \tag{A.1}$$

which allows us to calculate the energy of a signal in the time domain or the frequency domain.

A.2 Analytic Signal and Equivalent Lowpass Signals

A.2.1 Analytic Signal

The analytic signal x^+ of a signal x is defined as

$$x^+(t) = x(t) + i(\mathcal{H}x)(t), \tag{A.2}$$

where $\mathcal{H}$ denotes the Hilbert transform operator. For the spectrum we have

$$\hat{x}^+(\omega) = \begin{cases} 2\hat{x}(\omega), & \omega > 0, \\ \hat{x}(\omega), & \omega = 0, \\ 0, & \omega < 0. \end{cases}$$

Time domain $x(t) = x_\mathrm{R}(t) + i x_\mathrm{I}(t)$	Frequency domain $\hat{x}(\omega) = \hat{x}_\mathrm{R}(\omega) + i\hat{x}_\mathrm{I}(\omega)$
real x	even $\hat{x}_\mathrm{R}$, odd $\hat{x}_\mathrm{I}$; $\overline{\hat{x}(\omega)} = \hat{x}(-\omega)$
real and even x	real and even $\hat{x}$
real and odd x	imaginary and odd $\hat{x}$
imaginary x	odd $\hat{x}_\mathrm{R}$, even $\hat{x}_\mathrm{I}$
imaginary and even x	imaginary and even $\hat{x}$
imaginary and odd x	real and odd $\hat{x}$
even x_R, odd x_I; $\overline{x(t)} = x(-t)$	real $\hat{x}$

Table A.1: Symmetry properties of the Fourier transform.

Time domain	Frequency domain
$x(t)$	$\hat{x}(\omega) = \int_{-\infty}^{\infty} x(t)\,\mathrm{e}^{-i\omega t}\ \mathrm{d}t$
$\hat{x}(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} x(\omega)\,\mathrm{e}^{i\omega t}\ \mathrm{d}\omega$	$\hat{x}(\omega)$
$(x * y)(t)$	$x(\omega)\cdot y(\omega)$
$x(t)\cdot y(t)$	$\frac{1}{2\pi}(\hat{x} * \hat{y})(\omega)$
$x'(t)$	$i\omega\hat{x}(\omega)$
$x(t)\,\mathrm{e}^{i\omega_0 t}$	$\hat{x}(\omega - \omega_0)$
$\mathrm{e}^{i\omega_0 t}$	$2\pi\delta(\omega - \omega_0)$
$\cos(\omega_0 t)$	$\pi[\delta(\omega+\omega_0) + \delta(\omega - \omega_0)]$
$\sin(\omega_0 t)$	$i\pi[\delta(\omega+\omega_0) - \delta(\omega - \omega_0)]$

Table A.2: Fourier transform pairs.

A.2.2 Equivalent Lowpass Signals

In wireless communications, often the communication occurs in a passband $[-\omega_0 - W/2, -\omega_0 + W/2] \cup [\omega_0 - W/2, \omega_0 + W/2]$ with center frequency ω_0 and effective bandwidth W. In order to be able to perform the signal processing, e.g., modulation, demodulation, synchronization, etc., in the baseband, we need a equivalent lowpass representation of a bandpass signal.

Assume we have a real bandpass signal x_{BP} with an effective bandwidth W. The equivalent lowpass signal x_{LP} is defined as

$$x_{\mathrm{LP}}(t) = x_{\mathrm{BP}}^{+}(t)\,\mathrm{e}^{-i\omega_0 t}. \tag{A.3}$$

See Fig. A.1 for a visualization of the spectra. Note that according to this definition, the equivalent lowpass signal has twice the energy of the original bandpass signal. Even though the bandpass signal x_{BP} is real, the equivalent lowpass signal x_{LP} will be complex in general. If and only if

$$\hat{x}_{\mathrm{BP}}(\omega_0 - \omega) = \overline{\hat{x}_{\mathrm{BP}}(\omega_0 + \omega)}$$

and

$$\hat{x}_{\mathrm{BP}}(-\omega_0 - \omega) = \overline{\hat{x}_{\mathrm{BP}}(-\omega_0 + \omega)},$$

for $|\omega| \le W/2$, then the equivalent lowpass signal x_{LP} will be real.

The original bandpass signal x_{BP} can be recovered from x_{LP} via

$$x_{\mathrm{BP}}(t) = \mathrm{Re}[x_{\mathrm{LP}}(t)\,\mathrm{e}^{i\omega_0 t}]. \tag{A.4}$$

This equation can be easily verified by inserting (A.4) and (A.3) into (A.4). If the equivalent lowpass signal x_{LP} is real then (A.4) simplifies to

$$x_{\mathrm{BP}}(t) = x_{\mathrm{LP}}(t)\cos(\omega_0 t),$$

which corresponds to

$$\hat{x}_{\mathrm{BP}}(\omega) = \frac{1}{2}(\hat{x}_{\mathrm{LP}}(\omega - \omega_0) + \hat{x}_{\mathrm{LP}}(\omega + \omega_0)).$$

In the general case, i.e., if the equivalent lowpass signal x_{LP} is not real, we obtain from (A.4) that

$$x_{\mathrm{BP}}(t) = \frac{1}{2}(x_{\mathrm{LP}}(t)\,\mathrm{e}^{i\omega_0 t} + \overline{x_{\mathrm{LP}}(t)}\,\mathrm{e}^{-i\omega_0 t}),$$

which gives in the frequency domain

$$\hat{x}_{\mathrm{BP}}(\omega) = \frac{1}{2}(\hat{x}_{\mathrm{LP}}(\omega - \omega_0) + \overline{\hat{x}_{\mathrm{LP}}(-\omega - \omega_0)}).$$

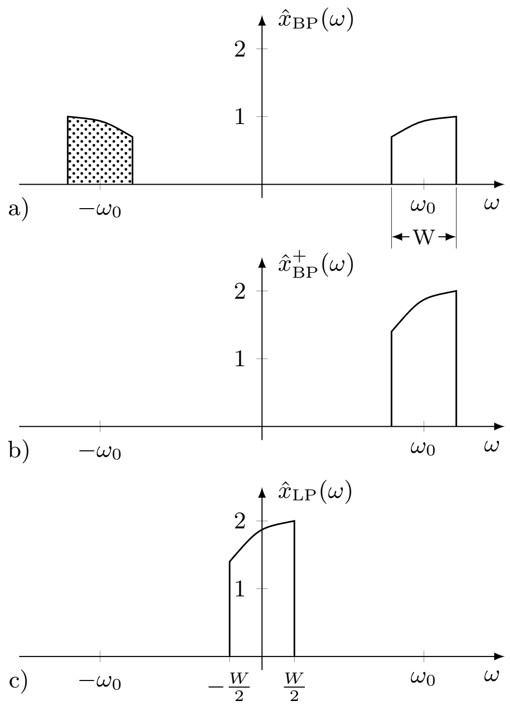


Figure A.1: Equivalent lowpass representation x_{LP} of a real bandpass signal x_{BP} in the frequency domain. a) Spectrum of the real bandpass signal x_{BP} (dotted means complex conjugate). b) Spectrum of the analytic signal x^{+}_{BP}. c) Spectrum of the equivalent lowpass signal x_{LP}.

Further, in the time domain it follows from (A.4) that

$$\begin{aligned} x_{\mathrm{BP}}(t) &= \mathrm{Re}[(x_{\mathrm{LP,R}}(t) + i x_{\mathrm{LP,I}}(t))(\cos(\omega_0 t) + i\sin(\omega_0 t))] \\ &= x_{\mathrm{LP,R}}(t)\cos(\omega_0 t) - x_{\mathrm{LP,I}}(t)\sin(\omega_0 t). \end{aligned} \tag{A.5}$$

Eq. (A.5) shows us how we can compute the real bandpass signal x_{BP} from the real and imaginary parts of the equivalent lowpass signal x_{LP}.

Next, we study how we can recover the complex equivalent lowpass signal x_{LP} from the real bandpass signal x_{BP}. Splitting (A.3) in to real and imaginary part, we obtain

$$\begin{aligned} x_{\mathrm{LP}}(t) &= [x_{\mathrm{BP}}(t) + i(\mathcal{H}x_{\mathrm{BP}})(t)][\cos(\omega_0 t) - i\sin(\omega_0 t)] \\ &= x_{\mathrm{BP}}(t)\cos(\omega_0 t) + (\mathcal{H}x_{\mathrm{BP}})(t)\sin(\omega_0 t) + i\left[(\mathcal{H}x_{\mathrm{BP}})(t)\cos(\omega_0 t) - x_{\mathrm{BP}}(t)\sin(\omega_0 t)\right]. \end{aligned}$$

In fact, using an ideal lowpass filter with bandwidth $W/2$, we can avoid the computation of the Hilbert transform. If we look at the spectra in Fig A.1, we see that

$$\begin{aligned} x_{\mathrm{LP}}(t) &= 2[x_{\mathrm{BP}}(t)\,\mathrm{e}^{-i\omega_0 t}] * h_{W/2}(t) \qquad \text{(A.6)} \\ &= 2[x_{\mathrm{BP}}(t)\cos(\omega_0 t)] * h_{W/2}(t) - i2[x_{\mathrm{BP}}(t)\sin(\omega_0 t)] * h_{W/2}(t), \end{aligned}$$

where

$$h_{W/2}(t) = \frac{W}{2\pi}\,\mathrm{sinc}\left(\frac{Wt}{2\pi}\right) = \frac{\sin(Wt/2)}{\pi t} \tag{A.7}$$

denotes the impulse response of the ideal lowpass filter with bandwidth $W/2$. Hence, it follows that

$$x_{\mathrm{LP,R}}(t) = 2[x_{\mathrm{BP}}(t)\cos(\omega_0 t)] * h_{W/2}(t)$$

and

$$x_{\mathrm{LP,I}}(t) = -2[x_{\mathrm{BP}}(t)\sin(\omega_0 t)] * h_{W/2}(t).$$

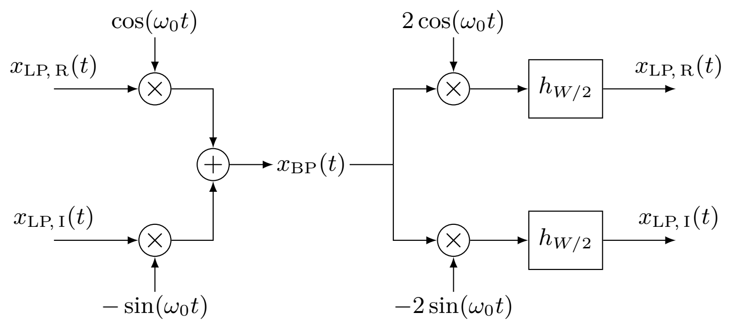


Figure A.2: Computation of the real bandpass signal x_{BP} from the equivalent lowpass signal x_{LP} (left part), and computation of the equivalent lowpass x_{LP} from the real bandpass signal x_{BP} (right part).

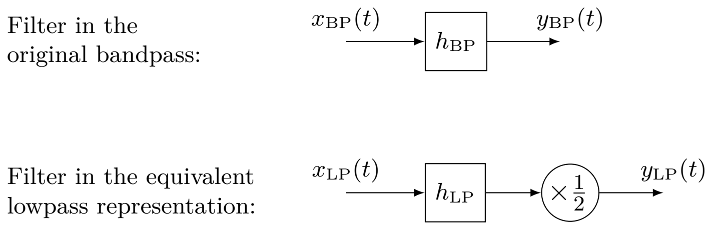


Figure A.3: Filter operation of a LTI system in the original passband (upper) and in the equivalent lowpass representation (lower).

In Fig. A.2 the computation of the real bandpass signal x_{BP} from the equivalent lowpass signal x_{LP} and the computation of the equivalent lowpass signal x_{LP} from the real bandpass signal x_{BP} is visualized.

In wireless communications the signal x_{BP} is the actual signal that is transmitted via the antenna, and the signal x_{LP}, is also called *complex baseband signal*, plays an important role in the mathematical description. The process of computing x_{BP} from x_{LP} is also called *upconversion* and the process of computing x_{LP} from x_{BP} *downconversion*.

A.2.3 Lowpass Representation of Bandpass LTI Systems

In this section we study how we can mathematically describe a bandpass filter operation directly in the baseband. To this end, we essentially apply the same considerations as in Section A.2.2 on the impulse response h_{BP} of an arbitrary bandpass LTI system.

Let x_{BP} be a bandpass signal and h_{BP} the impulse response of an LTI system. Then the output y_{BP} of this system when x_{BP} is the input signal is given by

$$y_{\mathrm{BP}}(t) = (x_{\mathrm{BP}} * h_{\mathrm{BP}})(t) = \int_{-\infty}^{\infty} x_{\mathrm{BP}}(\tau) h_{\mathrm{BP}}(t-\tau)\,\mathrm{d}\tau,$$

or, equivalently, in the frequency domain by

$$\hat{y}_{\mathrm{BP}}(\omega) = \hat{x}_{\mathrm{BP}}(\omega)\hat{h}_{\mathrm{BP}}(\omega).$$

Let ω_0 denote the center frequency of the bandpass system. We define the equivalent lowpass system h_{LP} as

$$h_{\mathrm{LP}}(t) = h^{+}_{\mathrm{BP}}(t)\,\mathrm{e}^{-i\omega_0 t}\,. \tag{A.8}$$

Using this definition of the equivalent lowpass system, we have

$$y_{\mathrm{LP}}(t) = \frac{1}{2}(x_{\mathrm{LP}} * h_{\mathrm{LP}})(t).$$

Note the additional factor of 1/2 in front of the convolution.

B Discrete-Time Equivalent Lowpass Channel

We can use the theory from the previous to derive a discrete-time equivalent lowpass representation of the wireless transmission channel.

B.1 Wireless Channel

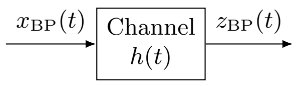


Figure B.1: Wireless channel as LTI system.

Without going into much details (for more details see for example [14]), we will present a simple model of the wireless channel first. We will model the channel as a linear time-invariant (LTI) system with an impulse response $h(t)$, as shown in Fig. B.1. Note that, for a wireless channel, the impulse response $h(t)$ is real valued. Given an input signal $x_{\mathrm{BP}}(t)$, the channel output $z_{\mathrm{BP}}(t)$ can be computed by

$$z_{\mathrm{BP}}(t) = (x_{\mathrm{BP}} * h)(t) = \int_{-\infty}^{\infty} x_{\mathrm{BP}}(\tau)h(t-\tau)\,\mathrm{d}\tau.$$

Clearly, modeling a wireless channel as a time-invariant system is a crude simplification, since a real wireless channel changes over time. However, a real wireless channel often changes only slowly with time. Hence, the channel is approximately constant at least during a short period of time, which is called coherence time. It follows that the LTI model as given here is approximately valid for short channel uses that are no longer than the coherence time.

B.2 Equivalent Lowpass Channel

Assume that we use the channel only in a passband $P = [-\omega_0 - W/2, -\omega_0 + W/2] \cup [\omega_0 - W/2, \omega_0 + W/2]$ with center frequency ω_0 and effective bandwidth W. Let $x_{\mathrm{LP}}(t)$ and $z_{\mathrm{LP}}(t)$ be the equivalent lowpass signals of the channel input $x_{\mathrm{BP}}(t)$ and the channel output $z_{\mathrm{BP}}(t)$, respectively. Our goal is to find a lowpass representation of the channel $h(t)$ such that

$$z_{\mathrm{LP}}(t) = \frac{1}{2}(x_{\mathrm{LP}} * h_{\mathrm{LP}})(t) = \frac{1}{2}\int_{-\infty}^{\infty} x_{\mathrm{LP}}(\tau)h_{\mathrm{LP}}(t-\tau)\,\mathrm{d}\tau. \tag{B.1}$$

As described in Section A.2.3,

$$h_{\mathrm{LP}}(t) = 2[h(t)\,\mathrm{e}^{-i\omega_0 t}] * h_{W/2}(t) \tag{B.2}$$

will give the desired result.

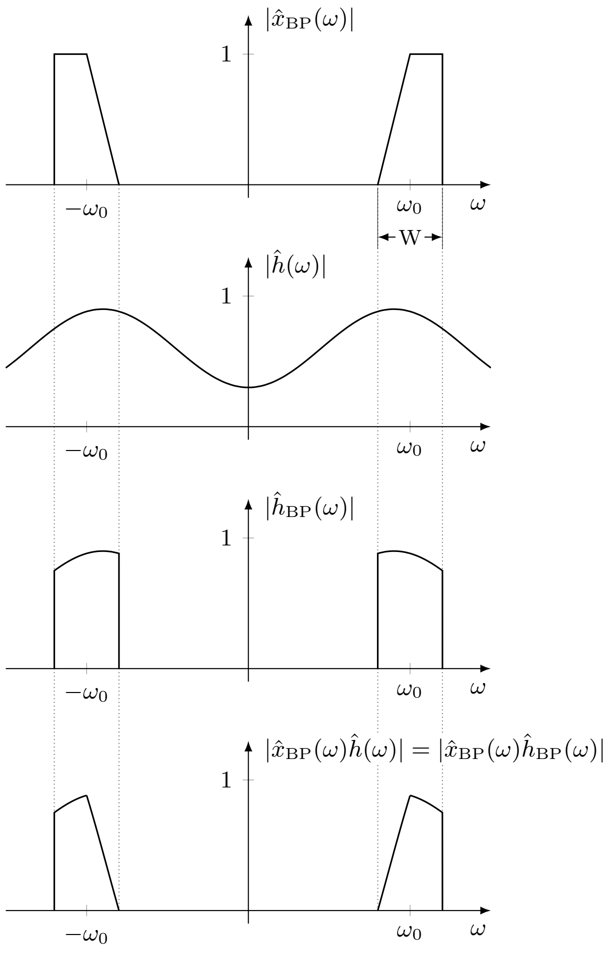


Figure B.2: Illustration of the action of a LTI system h on a bandpass signal x_{BP} in the frequency domain. Only the part of the transfer function that is in the passband of the signal is relevant.

Note that the transfer function of the channel is usually not restricted to the passband $P = [-\omega_0 - W/2, -\omega_0 + W/2] \cup [\omega_0 - W/2, \omega_0 + W/2]$ that we use for the transmission. However, since the transmit signal $x_{\mathrm{BP}}(t)$ has only frequencies in the interval P, it follows that only the corresponding part of the transfer function is relevant. That is, we have

$$z_{\mathrm{BP}}(t) = (x_{\mathrm{BP}} * h)(t) = (x_{\mathrm{BP}} * h_{\mathrm{BP}})(t)$$

where

$$\hat{h}_{\mathrm{BP}}(\omega) = \hat{h}(\omega)\mathbf{1}_P(\omega).$$

$\mathbf{1}_P(\omega)$ denotes the *indicator function* (or characteristic function) of the set P, which is given by

$$\mathbf{1}_P(\omega) = \begin{cases} 1, & \omega \in P, \\ 0, & \text{otherwise.} \end{cases}$$

Clearly, $\mathbf{1}_P(\omega)$ is the transfer function of a bandpass filter with passband P.

B.3 Discretization

Our final step is to create a discrete-time equivalent lowpass model of the of the channel. To this end, we discretize the equivalent lowpass channel that was derived in Section B.2.

Since h_{LP} and x_{LP} are both bandlimited with bandwidth $W/2$ we can express the convolution of h_{LP} and x_{LP} as a convolution summation. That is, we have

$$(h_{\mathrm{LP}} * x_{\mathrm{LP}})(t) = \int_{-\infty}^{\infty} h_{\mathrm{LP}}(\tau) x_{\mathrm{LP}}(t-\tau)\,\mathrm{d}\tau = \frac{2\pi}{W} \sum_{n=-\infty}^{\infty} h_{\mathrm{LP}}\left(\frac{2\pi n}{W}\right) x_{\mathrm{LP}}\left(t - \frac{2\pi n}{W}\right).$$

For a proof of this equality, see for example [15, p. 50, Th. 6.11]. It follows that

$$z_{\mathrm{LP}}(t) = \frac{1}{2}(h_{\mathrm{LP}} * x_{\mathrm{LP}})(t) = \frac{\pi}{W} \sum_{n=-\infty}^{\infty} h_{\mathrm{LP}}\left(\frac{2\pi n}{W}\right) x_{\mathrm{LP}}\left(t - \frac{2\pi n}{W}\right)$$

Sampling the signal z_{LP} at Nyquist rate gives

$$z_{\mathrm{LP}}\left(\frac{2\pi k}{W}\right) = \frac{\pi}{W} \sum_{n=-\infty}^{\infty} h_{\mathrm{LP}}\left(\frac{2\pi n}{W}\right) x_{\mathrm{LP}}\left(\frac{2\pi(k-n)}{W}\right).$$

Using the abbreviations $z_{\mathrm{LP}}[k] = z_{\mathrm{LP}}(2\pi k/W)$, $h_{\mathrm{LP}}[n] = h_{\mathrm{LP}}(2\pi n/W)$, and $x_{\mathrm{LP}}[k] = x_{\mathrm{LP}}(2\pi k/W)$, we obtain

$$z_{\mathrm{LP}}[k] = \frac{\pi}{W} \sum_{n=-\infty}^{\infty} h_{\mathrm{LP}}[n] x_{\mathrm{LP}}[k-n].$$

Further, inserting our expression for the equivalent lowpass channel (B.2) and the impulse response of the ideal lowpass filter (A.7), we see that

$$\begin{aligned} z_{\mathrm{LP}}[k] &= \frac{\pi}{W} \sum_{n=-\infty}^{\infty} \left(\int_{-\infty}^{\infty} 2h(\tau)\,\mathrm{e}^{-i\omega_0\tau}\, h_{W/2}\left(\frac{2\pi n}{W} - \tau\right)\,\mathrm{d}\tau \right) x_{\mathrm{LP}}[k-n] \\ &= \sum_{n=-\infty}^{\infty} \left(\int_{-\infty}^{\infty} h(\tau)\,\mathrm{e}^{-i\omega_0\tau}\,\mathrm{sinc}\left(n - \frac{W\tau}{2\pi}\right)\,\mathrm{d}\tau \right) x_{\mathrm{LP}}[k-n]. \end{aligned}$$

Setting

$$\tilde{h}[n] = \int_{-\infty}^{\infty} h(\tau)\,\mathrm{e}^{-i\omega_0\tau}\,\mathrm{sinc}\left(n - \frac{W\tau}{2\pi}\right)\,\mathrm{d}\tau \tag{B.3}$$

we obtain

$$z_{\mathrm{LP}}[k] = \sum_{n=-\infty}^{\infty} \tilde{h}[n] x_{\mathrm{LP}}[k-n]. \tag{B.4}$$

Note that $\tilde{h}[n] = \frac{\pi}{W} h_{\mathrm{LP}}[n]$. Eq. (B.4) gives us a very simple discrete-time equivalent lowpass channel model. The discrete-time output signal $z_{\mathrm{LP}}[k]$ is just the discrete convolution of the discrete-time input signal $x_{\mathrm{LP}}[k]$ with the discrete-time signal $\tilde{h}[n]$ that represents the channel.

B.4 Noise

In real systems we have, in addition to the distortions introduced by the channel, noise at different places, e.g., thermal noise, noise in the amplifiers, etc. A standard noise model, which is sufficiently good for most applications, is depicted in Fig. B.3.

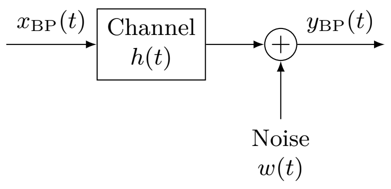


Figure B.3: Wireless channel and additive noise.

In this model, the noise $w(t)$ is white Gaussian noise with power spectral density $N_0/2$, and it is added just before the receiver. The received signal is given by

$$y_{\mathrm{BP}}(t) = (x_{\mathrm{BP}} * h)(t) + w(t).$$

Following the same steps as in Section B.3, it is possible to derive a discrete-time equivalent lowpass model of the channel as follows

$$y_{\mathrm{LP}}[k] = \sum_{n=-\infty}^{\infty} \tilde{h}[n] x_{\mathrm{LP}}[k-n] + \tilde{w}[k],$$

where $\tilde{h}$ is given by (B.3) and $\{\tilde{w}[k]\}_{k \in \mathbb{Z}}$ is a sequence of i.i.d. circularly-symmetric complex Gaussian random variables $\mathcal{CN}(0, \sigma^2)$.

C Some Helpful Mathematical Identities

Trigonometric identities:

$$\sin(x)\sin(y) = \frac{1}{2}[\cos(x-y) - \cos(x+y)] \tag{C.1}$$

$$\cos(x)\cos(y) = \frac{1}{2}[\cos(x-y) + \cos(x+y)] \tag{C.2}$$

$$\sin(x)\cos(y) = \frac{1}{2}[\sin(x-y) + \sin(x+y)] \tag{C.3}$$

Bibliography

[1] Bruce A. Black. *Introduction to Communication Systems — Lab Based Learning with NI USRP and LabVIEW Communications.* National Instruments, 2014.

[2] Jochen Schiller. *Mobile Communications.* Addison-Wesley, 2003.

[3] H. Meyr, M. Oerder, and A. Polydoros. On sampling rate, analog prefiltering, and sufficient statistics for digital receivers. *IEEE Transactions on Communications*, 42(12):3208–3214, December 1994.

[4] Salehi, M and Proakis, J. *Digital communications.* McGraw-Hill Education, 2007.

[5] Johnson Jr, C Richard and Sethares, William A and Klein, Andrew G. *Software receiver design: build your own digital communication system in five easy steps.* Cambridge University Press, 2011.

[6] S. M. Kay. *Fundamentals of Statistical Signal Processing: Estimation Theory.* Prentice Hall, 1997.

[7] K. B. Petersen and M. S. Pedersen. The matrix cookbook, October 2008. Version 20081110.

[8] T. Baykas, C. Sum, Z. Lan, J. Wang, M. A. Rahman, H. Harada, and S. Kato. IEEE 802.15.3c: the first IEEE wireless standard for data rates over 1 Gb/s. *IEEE Communications Magazine*, 49(7):114–121, 2011.

[9] Min Dong and Lang Tong. Optimal design and placement of pilot symbols for channel estimation. *IEEE Transactions on Signal Processing*, 50(12):3055–3069, 2002.

[10] Simon Haykin. *Digital Communication Systems.* Wiley Publishing, 2009.

[11] Aaron D Wyner. The wire-tap channel. *Bell system technical journal*, 54(8):1355–1387, 1975.

[12] Erdal Arikan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on information Theory*, 55(7):3051–3073, 2009.

[13] Ido Tal and Alexander Vardy. List decoding of polar codes. In *2011 IEEE International Symposium on Information Theory Proceedings*, pages 1–5. IEEE, 2011.

[14] David Tse and Pramod Viswanath. *Fundamentals of Wireless Communication.* Cambridge University Press, 2005.

[15] John R. Higgins. *Sampling Theory in Fourier and Signal Analysis – Foundations.* Oxford University Press, 1996.

Index