AALBORG UNIVERSITY
STUDENT REPORT

Master's thesis

Computational Considerations for Ultra-Reliable Low Latency Wireless Networks

Author:
Alexander Korsvang Hagelskjær

Supervisors:
Thomas Arildsen
Gilberto Berardinelli

This report is compiled in LaTeX. Figures are made using TikZ, Inkscape and Matplotlib.

AALBORG UNIVERSITY
STUDENT REPORT

Signal Processing and Acoustics
Aalborg University
http://www.aau.dk

Title:
Computational Considerations for Ultra-Reliable Low Latency Wireless Networks

Theme:
Master's thesis

Project Period:
Spring Semester 2019

Project Group:
Group 1072

Participants:
Alexander Korsvang Hagelskjær

Supervisor:
Thomas Arildsen
Gilberto Berardinelli

Number of Pages: 73

Date of Completion: June 6, 2019

Abstract:

Industrial control networks have traditionally been implemented on wired connections due to latency and reliability constraints. Next-generation cellular networks include services with improvements in these areas, but the improvements are not sufficient for all targeted use cases. Wireless Isochronous Real Time communication (WIRT) is a newly proposed system that targets networked control systems with periodic transmissions and extreme latency and reliability requirements.

This work highlights considerations when implementing the WIRT physical layer. The system architecture is based on Orthogonal Frequency Division Multiplexing (OFDM) for Ultra-WideBand (UWB) spectrum. A prototype implementation is made. The implementation integrates UWB testing equipment, to serve as a testbed for further WIRT development. The implementation is used to estimate which parts of the transceiver introduce the highest latency. The minimum possible latency is determined and the computational complexity of each component is evaluated. Based on this evaluation, architectures for a latency-accurate implementation are discussed.

The decoding of Error Correction Codes (ECCs) is found to be the largest single contributor to latency. Algorithmic alterations are made to reduce the minimum decoding latency to an acceptable level, along with other considerations required for implementation on a reconfigurable logic platform. In the end considerations and further work to prove the feasibility of the system are discussed.

PREFACE

This project is composed as the final thesis for the Signal Processing and Acoustics master at Aalborg University.

For citation the report employs IEEE referencing method. If citations are not present by figures or tables, these are made by the authors of the report. Units are indicated according to the SI system.

The code associated with this work is freely available at [1].

The author would like to thank LitePoint for the equipment used in this project and their support thereof.

Aalborg University, June 6, 2019

Alexander Korsvang Hagelskjær
<ahagel14@student.aau.dk>

TABLE OF CONTENTS

ABBREVIATIONS

ADC . . . . . Analog-to-Digital Converter
AWGN . . . . . Additive White Gaussian Noise
BER . . . . . Bit Error Rate
BLER . . . . . BLock Error Rate
CP . . . . . Cyclic Prefix
CRC . . . . . Cyclic Redundancy Check
DFG . . . . . Data Flow Graph
DFT . . . . . Discrete Fourier Transform
DSP . . . . . Digital Signal Processor
ECC . . . . . Error Correction Code
eMBB . . . . . Enhanced Mobile BroadBand
FFT . . . . . Fast Fourier Transform
FIR . . . . . Finite Impulse Response
FPGA . . . . . Field Programmable Gate Array
FSM . . . . . Finite State Machine
FSMD . . . . . Finite State Machine with Data path
GPU . . . . . Graphics Processing Unit
IDFT . . . . . Inverse Discrete Fourier Transform
IFFT . . . . . Inverse Fast Fourier Transform
LDPC . . . . . Low-Density Parity-Check
LLR . . . . . Log-Likelihood Ratio
LTE . . . . . Long Term Evolution
MAC . . . . . Multiply ACcumulate
mMTC . . . . . Massive Machine-Type Communications
NR . . . . . New Radio
OFDM . . . . . Orthogonal Frequency Division Multiplexing
QAM . . . . . Quadrature Amplitude Modulation
QPSK . . . . . Quadrature Phase Shift Keying
RF . . . . . Radio Frequency
SC . . . . . Successive Cancellation
SCL . . . . . Successive Cancellation List
SCM . . . . . SubCarrier Modulation
SM . . . . . Symbol Mapping
SNR . . . . . Signal to Noise Ratio
SPC . . . . . Single Parity Check
SSC . . . . . Simplified Successive Cancellation
TBCC . . . . . Tail Biting Convolutional Codes
URLLC . . . . . Ultra Reliable Low Latency Communication

UWB . . . . . Ultra-WideBand
VSA . . . . . Vector Signal Analyzer
VSG . . . . . Vector Signal Generator
WIRT . . . . . Wireless Isochronous Real Time communication
ZC . . . . . Zadoff-Chu

1 INTRODUCTION

Industry 4.0 is a term for the next generation of the industrial revolution. One of the major enablers in this is the increased availability of wireless communication and increasing computational power in embedded platforms. The use cases are both extensions to current practices and entirely new options in industrial manufacturing and production [2].

Control systems that connect sensors to actuators through various control algorithms are widely used for industrial purposes. When multiple control loops are connected over a network connection it is known as a *networked control systems* [3]. As networked control systems are generic, they can be used to describe many different systems, from temperature control of supermarket cold storage using mobile networks, through communication internally in automated vehicles, to remote surgery [4, 5]. For temperature control low latency and high reliability in the network is not critical, as the actuators and sensors are slow to respond. In case of remote surgery or intra-vehicle communication however, both low end-to-end latency and very high reliability is critical and potentially even vital [5].

Having both low latency and high reliability is widely recognized as a problem for traditional wireless technologies, where latency and reliability are considered trade-offs [2, 6]. In current Long Term Evolution (LTE) systems the physical layer latency can not be lower than the 1 ms subframe [5, 7]. When including other network delays [8] finds that LTE latencies can barely reach ~30 ms. Additionally, when operating with low latency the reliability is affected, making the networks unsuited for some industrial control networks. To handle different use-cases the next generation cellular network standard 5G introduces three services, Enhanced Mobile BroadBand (eMBB), Massive Machine-Type Communications (mMTC) and Ultra Reliable Low Latency Communication (URLLC) [9]. eMBB aims to improve the broadband experience and mMTC targets communication between a large number of connected devices that are not time sensitive. The third service, URLLC, aims to fulfill the latency and reliability requirements for industrial and other time and reliability sensitive networks [10].

In URLLC the target end-to-end latency is 1 ms. A URLLC system in 5G also requires high reliability, with a packet loss probability of at most $p_{\text{URLLC}} = 10^{-5}$ [9, 10]. One of the limits of the URLLC incorporated in 5G is that it is based on the same general purpose networks. Even though the three services are different, the core technology is the same, and thus not made specially for low latency purposes. The low latency devices considered in URLLC can also include both high and low power devices, devices spread over a large area and both consumer/media and industrial uses [6].

1.1 WIRT

For certain control systems used in industrial networks, the 5G URLLC targets are not stringent enough however. For machine tool operation [5, 8] find that latencies as low as 250 µs are required in addition to packet loss probabilities on the order of 10^{-9}.

[11] introduces a wireless communication system focused on industrial use named Wireless Isochronous Real Time communication (WIRT). In this system the more general purpose nature of 5G is discarded for even tougher latency and reliability requirements. The system aims to provide communication between devices that are close in proximity.

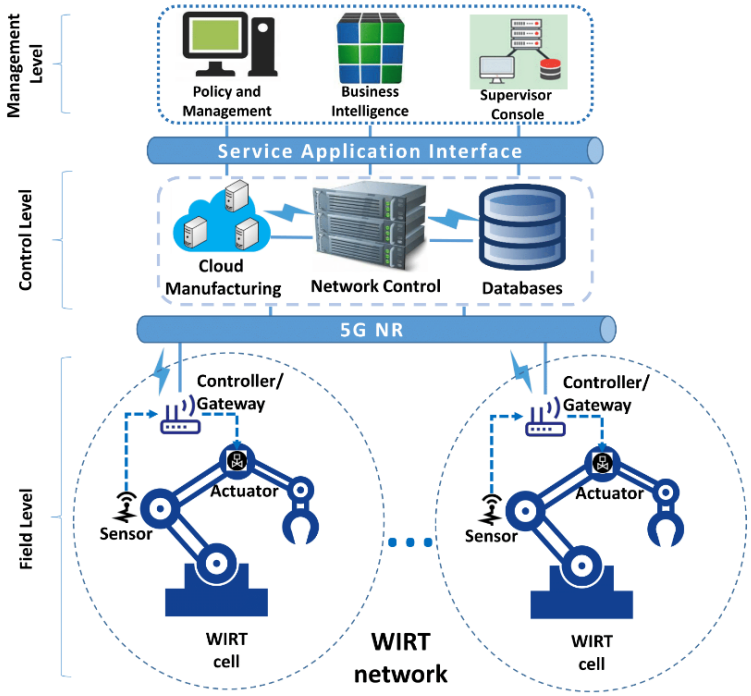


Figure 1.1: WIRTs place in a larger industrial environment. Reproduced from [11].

This is illustrated in Figure 1.1. In the figure WIRT is used to connect local sensors and actuators, while a gateway is connected to a the larger industrial environment using a cellular 5G connection.

To fulfill the role of a networked control system for the next generation industrial networks, WIRT targets packet loss probabilities down to 10^{-9} and end-to-end latency of down to 100 µs. The lower latency and higher reliability are not the only differences between WIRT and URLLC however. Control-loops used for automation are characterized by periodic transmissions. To support this type of traffic WIRT aims to provide connections specifically targeted traffic with deterministic cycles [11], as opposed to the sporadic traffic targeted in URLLC [9].

Another difference is in the spectrum use. 5G networks operate mainly in licensed bands that require centralized gateways with varying coverage. Additionally, spectrum regulation varies between regions, requiring roaming for use-cases where the system may be relocated. To avoid this problem WIRT is based on unlicensed Ultra-WideBand (UWB) communication. UWB communication can use very large bandwidths compared to the more common narrowband system, but is strictly regulated in power spectral density. With low power spectral density, the narrowband systems are not harmed by the UWB communication. Since the UWB communication uses a large bandwidth a narrowband interferer will only disturb a fraction of the used spectrum. The two systems can thus co-exist without mutual harm. This move enables use as a stand-alone radio and provides a large amount of usable spectrum [11]. With a targeted range on the order of 10 m, the low power spectral density is surmountable.

An overview of the key requirements presented in [11] is shown in Table 1.1.

Table 1.1: Key requirements or WIRT. Reproduced from [11].

Parameter	*Requirement/Description*
Application domain	Manufacturing, embedded systems, in-vehicle/aircraft control, etc.
Reliability	10^{-9}
Minimum Latency	0.1 ms
Data payload	32 bytes
Traffic type	Periodic
Spectrum access	Unlicensed
Network topology	Star / Meshed
Nodes per cell	20 - 200
MIMO capability	Limited at the devices, multiple antennas at the gateway
Form factor/power	Small size/low power
Range	In the order of 10 m
Mobility	No
Interference management	Possibly coordinated within cells in the same network, implicit across different networks

1.2 PROJECT OBJECTIVES

Ongoing research on WIRT attempts to map out interference scenarios and further refine the wireless aspects of the system. This work aims to supplement the research by providing the signal processing insight required to realize a prototype WIRT network. As the latency requirement is very harsh, it is necessary to consider all the delays in the system components, both computationally and structurally. As such the goal is to provide an analysis of which architectures and combination of components provides the best latency/reliability trade-off for a WIRT transceiver, backed up by a prototype system that can demonstrate the principles in action. To provide a testbed for future WIRT systems, the prototype system is built to interact with a UWB tester borrowed from LitePoint. This goal can be stated as a research question:

What are the requirements for realization of a WIRT system, from a signal processing point of view?

In order to answer this question a number of related questions must also be answered:

- What components should be part of a WIRT transceiver?
- What is the delay introduced by each component, and which components dominate the total latency?
- How can the system be designed to ensure that the latency and reliability targets are met?
- Which types of software and hardware architectures are appropriate?

Since the project is focused on the signal processing aspects, considerations on the Radio Frequency (RF) hardware required to build a WIRT system are not part of the project. This includes important high-level aspects such as link budgets and interference modeling, but also more specific challenges such as antenna design or Analog-to-Digital Converter (ADC) selection. Furthermore, design of the physical layer frame structures and higher network abstraction layers are also outside the scope of this project.

2 SYSTEM ARCHITECTURE

To understand the WIRT system, first a more generic wireless communication system is described. WIRT is based on multiband Orthogonal Frequency Division Multiplexing (OFDM) with a comparatively large bandwidth compared to LTE and WiFi. Section 2.1 starts with some basic definitions for a wireless communication system. Section 2.2 then describes OFDM and multiband OFDM that the WIRT transiever is based on. The individual components and the parameters that change between implementations are described in Section 2.3. Finally, Section 2.4 describes the configuration of these parameters chosen for WIRT.

The text in this chapter gives an overview of an OFDM system without specific details. In Chapter 3 the implementation is detailed, based on the descriptions in this chapter.

2.1 COMMUNICATION SYSTEMS

The overarching idea of a *communication system* is to transmit information from one point in space to another. In this project wireless communication systems is used to refer to systems that modulate the eletric field in the radio frequencies between two devices that are not physically connected. A basic discrete wireless communication system is seen in Figure 2.1.

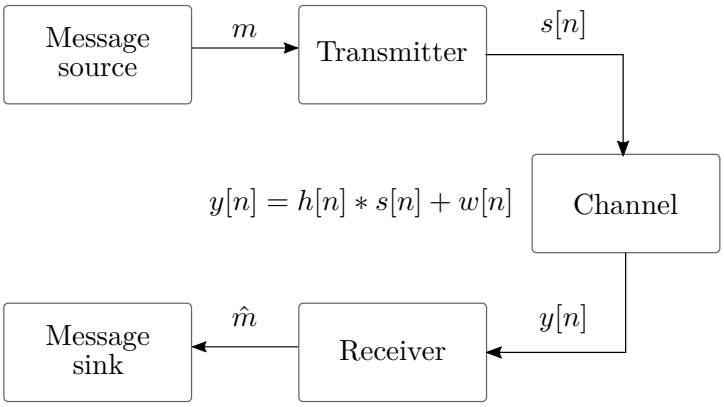


Figure 2.1: Basic communication system in the notation used in this report.

In the figure m is the message to be sent and $\hat{m}$ the receivers estimate of the message respectively. $s[n]$ is the transmitted time domain signal and $y[n]$ is the received signal. h is the channel gain and $w[n]$ is the noise. n represents the discrete time domain index. Convolution is represented with $*$.

The system shown in Figure 2.1 starts with a message m. This message is a sequence of bits that assumed to be binary and have equal probability of 0 and 1. The transmitter block transforms the message bits into an appropriate time domain signal $s[n]$ that can be transmitted over the channel. The signal, now affected by the channel, is received as $y[n]$. The receiver reverses the transformation done by the transmitter and creates an estimate $\hat{m}$ of the original message. The communication is deemed successful if $\hat{m} = m$.

If no noise is present, and it is possible to counteract the effects of the channel, the decoded message should be identical to the transmitted and perfect communication is achieved. This assumption is very strong however and communication is not always

perfect. As seen in the figure, the received signal can be modeled as:

$$y[n] = h[n] * s[n] + w[n]. \tag{2.1}$$

The effect of the channel on the transmitted signal can then be modeled by two parameters, the noise $w[n]$ and the complex channel gain $h[n]$. $h[n]$ is also known as the channel impulse response with corresponding frequency response $H[k]$.

A commonly used channel is the Additive White Gaussian Noise (AWGN) channel where:

$$h[n] = \begin{cases} 1 & \text{for } n = 0 \\ 0 & \text{else,} \end{cases} \tag{2.2}$$

and the noise is distributed as $w[n] \sim \mathcal{N}(0, \sigma_N^2)$. Here σ_N^2 is the variance or noise power. This is a basic model which does not account for multipath fading. In more realistic channel scenarios multipath fading will contribute to $h[n]$ having more than one non-zero value.

2.2 MULTICARRIER MODULATION AND OFDM

In single carrier modulation the message is modulated directly onto a single carrier. In order to increase the data rate of such a system the symbol frequency is increased. The basic idea of multicarrier modulation is to map multiple symbols to multiple subcarriers thereby achieving higher throughput through parallel transmission. The data rate of each individual subcarrier can then be much lower without affecting the system throughput.

Orthogonal Frequency Division Multiplexing (OFDM) refers to a type of multicarrier modulation where the subcarriers are chosen to be orthogonal in frequency. OFDM is widely used in many communication systems, including LTE and WiFi [7]. In multiband OFDM the available spectrum is further split into a number of *channels* that each contain a subsection of the full spectrum. The channels can be used by different systems at the same time, or by a single system to improve reliability. By splitting the full spectrum into channels, the transmitter and receiver complexity is reduced compared to supporting the full bandwidth at once. It is chosen to base the WIRT transceiver on multiband OFDM.

2.3 SYSTEM COMPONENTS

The system shown in Figure 4.1 is a basic communication system. In this section the transmitter and receiver blocks are opened up, and the individual components discussed.

The transceiver architecture for an OFDM system is shown in Figure 2.2. The figures give an overview of respectively the transmitter (Figure 2.2a) and receiver (Figure 2.2b).

The transmission starts with the message m. The message bits are first encoded with an Error Correction Code (ECC) in order to improve the reliability of the transmission. The ECC codewords are mapped to complex symbols to prepare for modulation. The symbols are gathered in a serial to parallel converter and mapped to subcarriers. The Inverse Fast Fourier Transform (IFFT) is performed on the mapped symbols and a known pilot signal. The resulting signal is serialized to a discrete, complex time series. Finally, a cyclic prefix is appended to the series before it is sent to the RF hardware for transmission over a wireless medium.

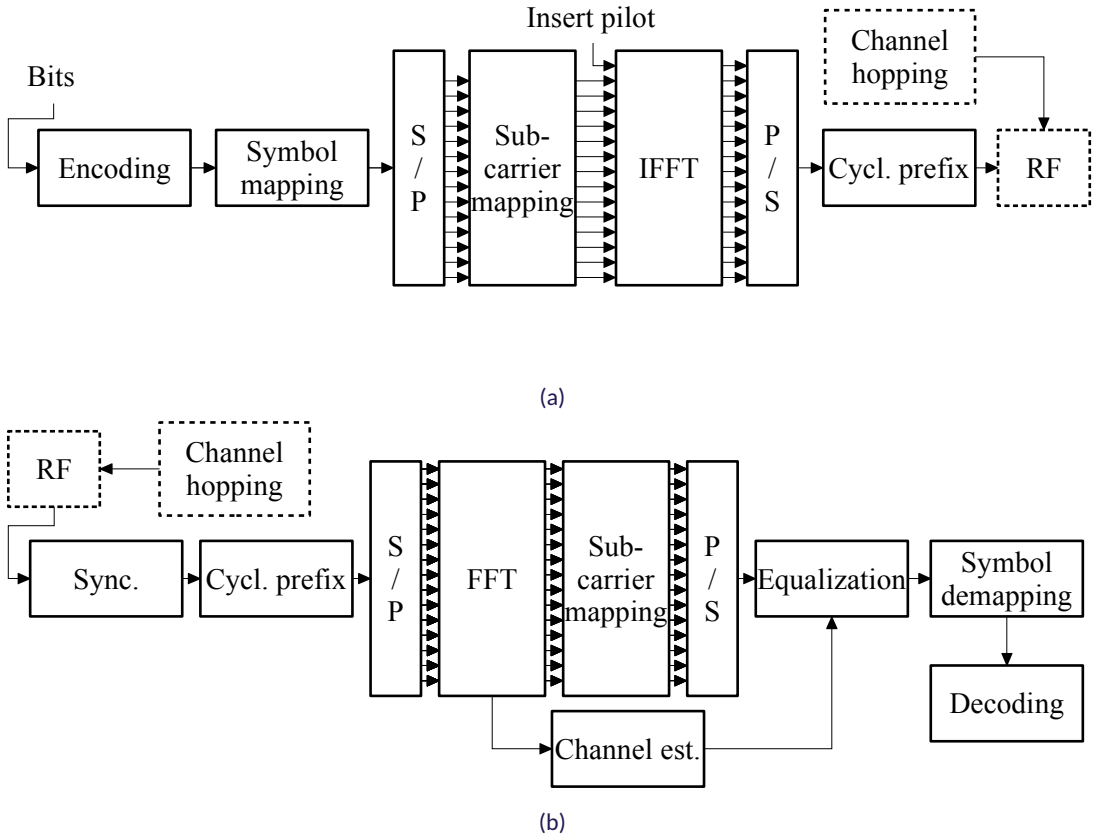


Figure 2.2: The (a) transmitter and (b) receiver chains that form the transceiver architecture for WIRT. P/S and S/P are short for parallel to serial and serial to parallel respectively. Sync. is short for synchronization and channel est. is short for channel estimation. Cycl. is short for cyclic. The dashed boxes are outside the scope of this project.

Reception performs the inverse operations in addition to a few others. First synchronization algorithms are performed to establish correct phase, frequency and time alignment between the transmitter and receiver. The cyclic prefix is then removed and the serial data is parallelized before an Fast Fourier Transform (FFT) is performed to reverse the effect of the IFFT. The pilot carriers are recovered and used in channel estimation while the data carriers are passed through a reverse subcarrier mapping and serialization. The estimated channel is used to equalize the data symbols, which are passed on to the symbol demapper. Finally, the resulting ECC codewords are decoded to produce the most likely transmitted message bits.

The role of each of the components and how they work is described in the remainder of this section.

2.3.1 ERROR CORRECTION CODING

The first and last part of the transceiver chain, encoding and decoding, represent the use of an Error Correction Code (ECC). An ECC is a code used to enhance the robustness of communication systems and improve the communication reliability. This is done by changing the representation of the message in order to spread the information content into more bits than in the original message representation. These extra bits contain redundant information, but can be used to reconstruct the original message completely

if parts of the transmission fail.

In general ECCs are modeled as a transformation of a K-dimension vector to a N-dimension vector where $N > K$. The ratio $R = \frac{K}{N}$ is termed the *rate* of the ECC. One particularly simple type of ECC is *repetition codes*, where the message is simply repeated R times. Since the same message is sent multiple times, all of them have to fail in order for the communications to be unsuccessful.

Repeating packages turns out to be neither the most effective or flexible scheme for introducing redundancy. The field took off with Shannons seminal paper on information theory [12], but ECC research is still ongoing. One of the reasons is that decoding is a computationally complex process and only recently have computers reached a level of performance where wide-spread use of complex ECC is possible. Improving the performance of modern codes such as Low-Density Parity-Check (LDPC), turbo codes and polar codes is still subject of research, both in terms of their ability to correct errors and their computational complexity [13]. Chapter 5 describes ECC in more depth.

The parameters for error correction code are the size of the codeword sent on the channel N, the number of information bits per codeword K and the type of code. Different codes also come with parameters specific to the code.

2.3.2 SYMBOL MAPPING

Symbol mapping is the process of changing the information bits into *symbols* that can be transmitted over a waveform. In single carrier systems this process is also known as *modulation*, but since the term modulation is also used for multicarrier modulation performed for OFDM, so to reduce confusion the operation of mapping bits to symbols is denoted *Symbol Mapping (SM)*.

Many different methods exist that transform bits into different kinds of waveforms, but for wireless communication the most often used types manipulate the phase, frequency and amplitude of a sine and a cosine wave. At the receiver the reverse mapping is then perform to recover the information bits. The SM used with OFDM in LTE are Quadrature Phase Shift Keying (QPSK) and variations of Quadrature Amplitude Modulation (QAM) which encode the information bits onto either just the phase or both the phase and amplitude respectively [14].

The type of SM used in a communication system has a great influence on both the bit error rate and the number of bits represented by each symbol. Modulation types with a large number of bits per symbol can enable high data rates or lower spectral usage but often come with higher probability of bit errors [14].

An example of this is comparing the LTE symbol mappings 16-QAM and QPSK. Because 16-QAM has a lower distance between symbols, each symbol has higher likelihood of failure in the presence of noise. Additionally, since 16-QAM uses both phase and amplitude in the modulation good synchronization must be achieved for both, while QPSK only requires good phase synchronization. The disadvantage of low distance between symbols is countered by the fact that each symbol in an 16-QAM constellation contains $\log_2(16) = 4$ bit while each QPSK symbol contains $\log_2(4) = 2$ bit. 16-QAM modulated signals are therefore able to attain a higher data rate within the same bandwidth, given that the Signal to Noise Ratio (SNR) is sufficient to distinguish the symbols and accurate synchronization is performed.

Having multiple different modulation types available enables a communication system to make a trade-off between data rates and lower failure probabilities. For example, in

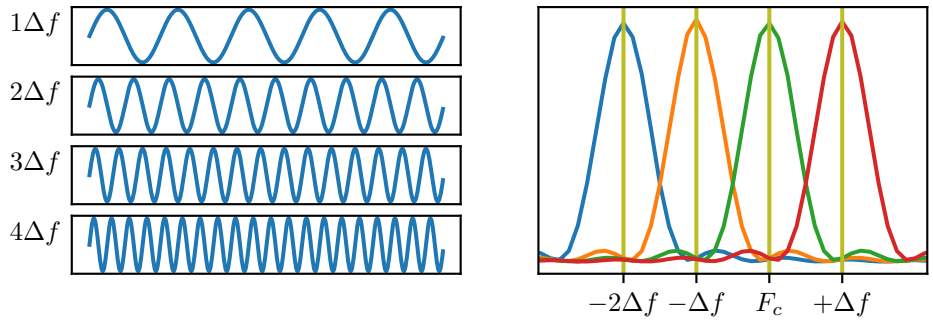


Figure 2.3: Example of how multiple subcarriers in the time domain combine in the frequency domain. Each subcarrier on the left is modulated with data to produce the spectrum on the right. The difference in frequency between individual subcarriers depends on the total bandwidth and the number of subcarriers.

the highly flexible LTE system the user can use a high order QAM when close to a base station and achieve high data rates. If the user moves further from the base station, the handset can transition to QPSK to still maintain connection at a lower data rate. In a real wireless channel small scale fading effects make the situation more complex, but more options for modulation schemes will still make the system more dynamic and robust. Together with a dynamic choice of ECC configuration, this is known as *link adaptation* or *adaptive coding and modulation* [7, Ch. 12].

2.3.3 SUBCARRIER MAPPING AND DFT

After mapping the coded bits to symbols, the symbols must now be modulated onto subcarriers. Each of the complex symbols is mapped to one of $N_{\text{subcarriers}}$ subcarriers. Each subcarrier has a unique frequency, with a spacing of Δf between carriers. The symbol is held on the corresponding carrier for a duration of $T_S = 1/\Delta f$. The subcarriers are then combined into a single with the same length. Since the carriers are chosen orthogonally, the symbols do not interference. Multiple symbols are transmitted at the same time and the symbol frequency can be lowered without lowering the data rate. The process of mapping each symbol onto a subcarrier is known as SubCarrier Modulation (SCM). A visual rendition of this process is shown in Figure 2.3. Input symbols are modulated on each of the subcarriers on the left and each subcarrier is further modulated to the RF carrier frequency F_c. The corresponding spectrum is shown on the right.

Creating each of these subcarriers and mixing them to RF frequency separately requires a bank of as many oscillators as subcarriers, all operating at closely spaced frequencies. The discovery that makes OFDM simple to use in practice is that the subcarriers can be created by an Inverse Discrete Fourier Transform (IDFT) operation on the symbols and then simply mixing the serialized output of the transform to RF [15, Sec. 12.2]. At the decoder this operation can be reversed by computing the Discrete Fourier Transform (DFT). In practice the IDFT and DFT are computing using the IFFT and FFT operations respectively.

The main parameter of the subcarrier mapping is the number of subcarriers $N_{\text{subcarriers}}$, equal to the number of FFT bins N_{FFT}. Another parameter is the *subcarrier spacing* Δf, but given the overall bandwidth of the system and N_{FFT} this parameter is given:

$$\Delta f = \frac{\text{BW}}{N_{\text{FFT}}}. \tag{2.3}$$

2.3.4 CYCLIC PREFIX

The final step in preparing the message bits for transmission is adding a prefix.

The prefix serves two purposes. The first is to serve as a barrier against multipath fading. If the signal travels in multiple different paths and arrive with different delays, the symbols can start to interfere with each other. By adding unused symbols in front of the data carrying symbols, the data symbols are less affected by multipath fading. The severity of the expected fading is the determining factor for the duration of the prefix. For canceling multipath fading the prefix can be any symbol, as it is simply removed in the receiver.

The second purpose is to change the effect of the channel from a linear to a circular convolution in the time domain. As described above the IDFT is used to map the symbols on to separate subcarriers. When the resulting time sequence is sent over the channel, the channel gain h corresponds to a convolution in the time domain. Fortunately, for periodic sequences, convolution in the time domain corresponds to multiplication in the frequency domain [16, sec. 8.6]. Unfortunately, the channel is in general not periodic and neither are the symbols to be sent. By choosing the prefix samples to be equal to the final samples of the message, the new message is now periodic with a period equal to the length of the original message. Since the cyclic prefix is chosen to be longer than the channels impulse response, the convolution of the message with the channel is now a circular one [15, ch. 12]. After the prefix is removed at the receiver, the sampled channel frequency response can be seen as a multiplication with the subcarrier symbols because of the circular convolution. This type of prefix is known as a *cyclic prefix*.

Since the cyclic prefix samples are given by the message, the only free parameter is the duration. The prefix should be long enough to encompass most of the power from a multipath environment, but as it does not carry any data increasing the size also increases the overhead in the communication.

2.3.5 CHANNEL HOPPING

With the message encoded, mapped and modulated onto subcarriers it is now ready for transmission. Channel hopping is the process of choosing a frequency band to send it. Different frequency bands can be affected differently by multipath fading and interference, so changing the center frequency can help potentially increase the reliability of the communication.

Channel hopping can be implemented in two ways; either the IFFT is zero padded in a manner which leaves the intended signal in the correct frequency band, or the RF hardware has the option to change the center frequency to different bands. Performing channel hopping by zero-padding the IFFT would increase the computational complexity of this component by orders of magnitude. Additionally, the RF hardware used to prototype in this project has the capability of changing center frequency to a wide span as described in Section 3.2. Based on these considerations it is chosen to implement the channel hopping by changing the center frequency in the RF hardware. The logic for choosing a frequency band and synchronizing the choice between transmitter and receiver is not considered in this report.

2.3.6 SYNCHRONIZATION

In addition to blocks that reverse the operations above, the receiver contains three blocks extra: Synchronization, channel estimation and equalization.

To reinterpret the bits from received QPSK symbols, the phase of the signal is critical. A problem therefore occurs at the receiver, as the oscillator used in the reception is unlikely to be at the exactly same phase as the oscillator used in the transmitter. Additionally, imperfections in the oscillator hardware can lead to minor differences in frequency between the transmitter and receiver [17, 15]. Static phase errors and small offsets can be cancelled in the equalization, see Section 2.3.7, but if the frequency offset becomes too large, synchronization must be performed in order to successfully receive the information.

Frequency is not the only synchronization that might be required however. The transmitter and receiver do not have a time reference in common and the receiver is unable to know when the transmitter begins a message. The receiver therefore needs a method for detecting a transmission and locating the correct samples for the FFT and further demodulation.

Methods for synchronizing both time and frequency offsets exist, for instance the methods described in [17] and [18]. This tester used in this project uses the same oscillator for both the transmitter and receiver and so has very little to no frequency offset. This is described further in Section 3.2. Since the frequency synchronization is unnecessary it is chosen to use only time synchronization. This is described in more detail in Section 3.3.2.

2.3.7 CHANNEL ESTIMATION AND EQUALIZATION

In order to perform demodulation most wireless systems require an estimate of the channel $h[n]$ and OFDM is no different. In OFDM this estimate is used to perform equalization to cancel out as much of the frequency selectivity of the channel as possible and align the phase of the symbols. In other words, the aim of the equalization is to make the channel frequency response as flat as possible [19, ch. 16].

Based on the representation of an OFDM system as a number of parallel carriers, the channel estimate can also be described as a number of parallel fading channels [19, ch. 19]. By introducing *pilot symbols*, static symbols known at both the transmitter and receiver, the channel response on the k-th subcarrier can be estimated as:

$$H[k] = \frac{x[k]}{c[k]}, \tag{2.4}$$

where k is the subcarrier index, $H[k]$ is the channel frequency response at the subcarrier frequency, $x[k]$ is the received symbol on subcarrier k and $c[k]$ is the expected pilot symbol.

In order to get the best possible estimate of the channel all the subcarriers in the OFDM symbol should be pilot carriers, however then data transmission is not possible. Fortunately, channels closely spaced in time and frequency are likely to have non-zero correlation, making it possible to dedicate only a subset of the carriers to pilot tones and still get a reasonable estimate of the channel. The more subcarriers are allocated as pilot carriers, the better the channel estimate and therefore the equalization, but at the cost of overhead. The amount of spacing possible between the pilot carriers depends the correlation between subcarriers, typically characterized by *coherence bandwidth*. As

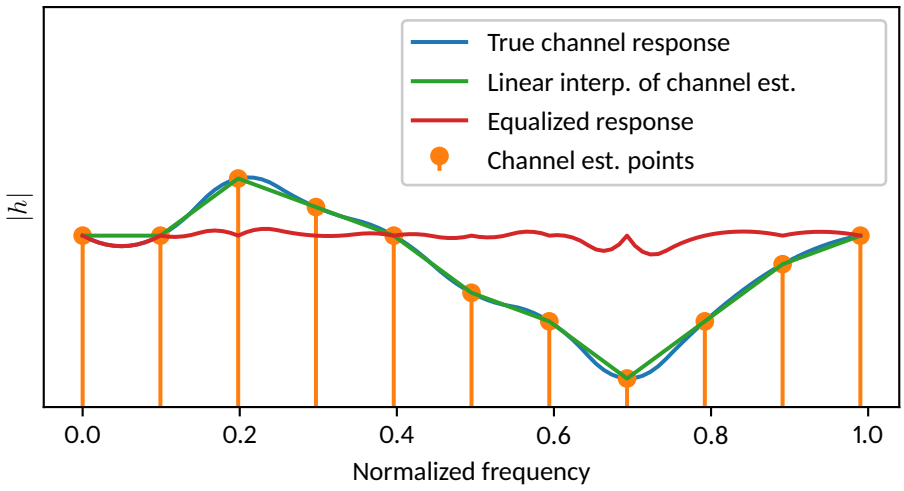


Figure 2.4: Illustration of the effect of channel equlization in a frequency selective channel. The plots shows the absolute value of the channel gain $|H|$ over normalized frequency. The blue line shows the true channel response. The orange stems show the pilot tones and the green line shows the channel estimate when linearly interpolated. The red line shows the equalized response.

long as the coherence bandwidth is larger than the distance in frequency between the subcarriers, they can be considered correlated [15, Sec. 3.3].

Since the channels are correlated in both time and frequency, it is possible to spread out the pilot carriers in either. In practice spreading out in time means that an OFDM symbol is sent with only pilot carriers. The channel estimate is then used for the next couple of symbols that are full of data carriers, thus giving a good resolution in frequency but not in time. The other option is to send a mix of pilot and data carriers in all OFDM symbols. This provides good resolution in time, but worse in frequency.

An illustration of the effect of equalization is shown in Figure 2.4. In this example the second option for channel estimation is used.

2.4 WIRT SPECIFICATIONS

This chapter describes an OFDM system and its components at a high level. Each of the components in the system is described in terms of a series of parameters, enabling description of WIRT using these. Table 2.1 shows the system described in terms of these parameters and three generic ones. Note that the specifications in Table 2.1 are tentative and a number of parameters are not specified.

The specifications leave room for interpretation when implementing. For this work it is chosen to use 400 bit packages. For channel hopping, the rest of the report will assume that enough free channels are available that the system is not limited. The channel estimation requirement specifies a spacing between pilots of 5 MHz. It is chosen to use 124 pilot subcarriers, spread out evenly in frequency, thus leaving 400 subcarriers for data transmission per OFDM symbol. In order to ensure high reliability a low rate ECC should be used, but appropriate codes have not been been researched prior to this work. The choice of an appropriate ECC is handled in Chapter 5.

At the time of writing, several parameters of WIRT system are still under development. Since work is ongoing on charactarizing both wireless channels and interference,

Table 2.1: Specifications for a WIRT transceiver.

Component	*Parameter*	*Requirement*
General	K (information size)	160 - 400 bits ([20, table 5.3.2.1-1])
General	Transmission duration	Min. 20 µs
General	Bandwidth	Min. 500 MHz
Symbol mapping	Type	QPSK
Subcarrier mapping	Subcarrier spacing	960 kHz
Subcarrier mapping	Total subcarriers	Min. 524
Cyclic prefix	Duration	69 µs
Channel hopping	Number of frequency hops	Depends on the available channels
Channel estimation	Spacing between pilots	Approximately 5 MHz

synchronization is not locked down yet. Only the synchronization required for basic operation is implemented, as described in Section 3.3.2. Likewise, requirements for channel estimation are not defined yet. It is therefore again chosen to implement a simple method, described in Section 3.3.3 and delimit the work from more indepth equalization considerations.

Based on the specifications, a WIRT transceiver is implemented.

3 SOFTWARE BASED WIRT IMPLEMENTATION

The system architecture described in Chapter 2 describes the overall architecture of the WIRT transceiver. The next step is creating a prototype of this system. The goal of the prototype is to estimate the computational complexity of the system and establish potential bottlenecks in the operations. A secondary objective of the prototype is to serve as a testbench for future WIRT projects.

An overview of the prototype is given in Section 3.1. The hardware used to perform the RF operations is described in Section 3.2. The more specific considerations and choices made during development are described in Section 3.3. Finally, testing and verification of the software implementation is described in Section 3.4.

3.1 OVERVIEW

The software prototype is split into a number of modules as seen in Figure 3.1. The main blocks contain most of the system functionality. These are split into five submodules that concern WIRT functionality, functionality to interface with the LitePoint instrument and more specific modules for socket utilities, OFDM modulation and polar codes. The misc blocks contain smaller bits of functionality that is used in the WIRT, OFDM and ECC submodules. The testing modules are used to test the functionality of each block individually and, in the case of the WIRT test, the overall system. This chapter describes most of the blocks, except the polar codes which are described in Chapter 5. The module for instrument interfacing is described in Section 3.2. The specific implementation details and choices made for the prototype are detailed in Section 3.3 and testing is described in Section 3.4.

The prototype is developed in Python and depends heavily on the numerical computing library *numpy*. This choice of platform enables fast development and iteration time and access to the large amount of available utilities made for Python development. In an actual implementation it is recommended to move away from Python language to better

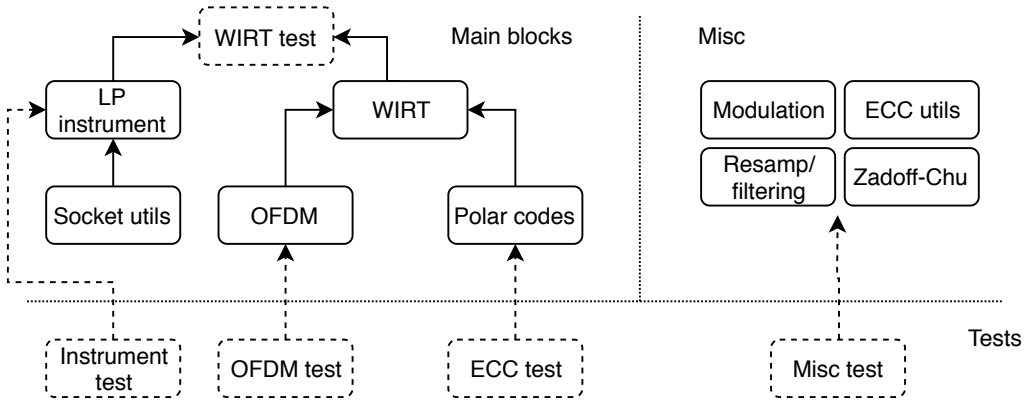


Figure 3.1: An overview of the modules developed for this system. Solid blocks are modules that contain functionality. The relation between blocks is illustrated through arrow connections, e.g. the LP instrument module uses the socket utilities for communication. Dashed blocks and arrows are testing modules and the blocks that they test.

ensure the hard real time deadlines, but for the prototyping scenario it has significant merit.

3.2 UWB TESTER

Compared to traditional narrowband communication, RF development systems for UWB are less common and often expensive. For the purpose of this project an IQgig-UWB tester is borrowed from LitePoint.

The main criterion for the RF hardware to successfully implement the prototype WIRT system, is an instantaneous bandwidth of at least 500 MHz. The Vector Signal Generator (VSG) and Vector Signal Analyzer (VSA), which are used for RF signal generation and sampling respectively, both support 1900 MHz instantaneous bandwidth, more than sufficient [21]. In other specification the tester has an analog-to-digital converter resolution of 12 bit, a noise figure at or below 20 dB and the power is within $\pm$ 1.4 dB across the entire frequency range. Additionally, the VSG and VSA share a single oscillator, ensuring they are synchronized in phase and frequency.

The tester is interfaced over a TCP/IP connection. The tester opens three TCP ports, 24000, 24001 and 24002. The first port is the main communication, with the two other ports used for supporting service interrupts and clear commands. Since these are not required for this use case, the project only uses the main communication port.

All commands and responses are transmitted to and from the tester in ASCII. In order to speed up the transfer of large amounts of data samples, a binary data transfer mode is enabled by sending a `FORM:READ:DATA PACK` command. After the command is received, data is sent in the format

```
#<lenlen><len><data>
```

where `<data>` is the information to be transmitted, packed as binary values, `<len>` is the length in bytes of the data and `<lenlen>` is the number of ASCII characters in the lenght field. The first character `#` represents the start of the header. This format is known as an *Arbitrary block*.

The tester has the ability to play arbitrary I/Q samples from the VSG. To upload I/Q samples for playback they must be packed in a waveform format called *IQVSG*. This format consists of a header and packaged binary data. The header contains a filename, sample rate and total number of samples. The data is formatted as interleaved 4-byte floating point values. The waveforms are sent using the arbitrary block format above.

During testing it is found that the time between the trigger being pulled high, and the samples being ready for download is the largest delay in this system. If a large number of tests using the prototype system are to be performed, it is therefore recommended to pack multiple WIRT packages into a single transfer to the tester. Since the tester has memory for multiple waveforms and can receive data while triggering, the prototype system could be extended to use the wait time to upload new waveforms.

The procedure to setup the tester, upload a single waveform and triggering it is described in Table 3.1.

Table 3.1: The steps performed for initialization of the tester, upload of a waveform and download of the corresponding samples.

Step	*Description*
tester_default_setup	Reset the tester and setup routing and power levels.
tester_set_capturetime	Set the length of a single capture.
upload_waveform	Pack and upload IQ data as a waveform file.
tester_setup_repeat	Setup waveform playlist for triggered playback of the waveform.
tester_setup_ext_triggers	Setup the VSA and VSG to trigger on the external trigger which can be controlled by commands.
tester_arm	Arm the instruments to ready them for triggering.
tester_ext_trigger	Pull the external trigger high to start the transmission.
download_waveform	Download the new samples from the tester.

3.3 SPECIFICS OF THE IMPLEMENTATION

Most of the parameters of transceiver are specified in Section 2.4. Despite this, some implementation details come up during the system implementation. This section describes the choices made in this project for these details.

3.3.1 SYSTEM SAMPLE RATE

The WIRT system targets a bandwidth of 500 MHz per channel. Since the VSG of the tester outputs samples at a rate of 2.4 Gsps, sample rate conversion of the signal is required. Sample rate conversion is the act of changing a digitally sampled signal from one sample rate to another without altering the information content. When the number of samples must increase the operation is known as *interpolation* or *upsampling*, while reducing the number of samples is known as *decimation* or *downsampling*. The naïve way to perform interpolation is to repeat the previous samples. This operation is simple, however as shown in Figure 3.2a, the spectrum of the output signal is widened significantly. The figure shows the frequency response of OFDM signal interpolated 4 times.

A better way to increase the number of samples to insert zeros between the samples to match the new sample rate instead of repeating samples. The signal is then low-pass filtered to create the new interpolated signal. The result of this operation is shown in Figure 3.2b.

For filtering during interpolation an important parameter is how much dampening of the produced images can be achieved. In general more dampening is good, but [22] finds that very sharp filters are not ideal as they lead to intersymbol interference. The parameters for filting are therefore the type of filter and the length. A filter commonly used in communication systems is the root raised cosine filter [23]. These filters are designed specifically to reduce the intersymbol interference by having zeros in their impulse response at multiples of the symbol time. The filter length is a trade-off between requirements from spectrum regulations and increasing interference between symbols.

A disadvantage of filtering is that it introduces a delay in the signal, called a group delay. The introduced group delay depends on the filter type and length. Since a root

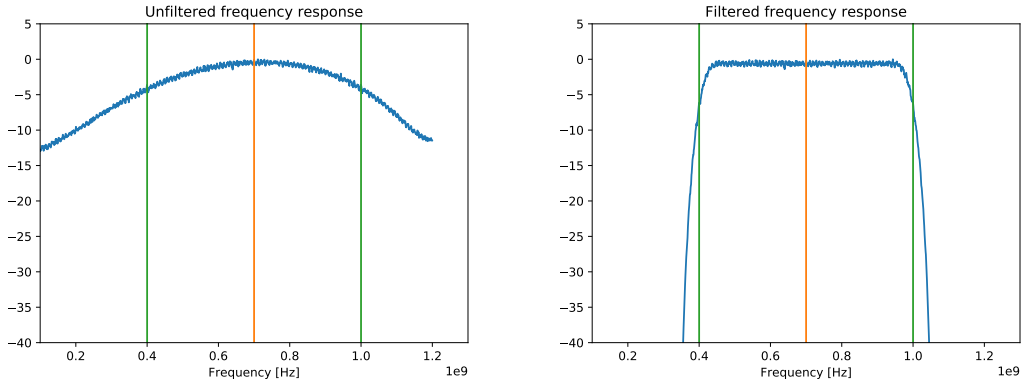


Figure 3.2: Frequency response an OFDM signal with 600 MHz bandwidth interpolated 4 times to 2.4 Gsps. (a) interpolation by repeating samples, (b) interpolation by resampling.

raised cosine filter is a linear phase Finite Impulse Response (FIR) filter, the group delay introduced is a constant time delay for all samples which can be cancelled by simply selecting later samples [24, Ch. 5].

The next question that arises is how much to interpolate. Subcarrier spacing is defined as

$$\Delta f = \frac{B}{N_{\mathrm{FFT}}} = \frac{F_s/R}{N_{\mathrm{FFT}}}, \tag{3.1}$$

where B is the bandwidth of the signal given by the instrument tester sample rate F_s and the resampling rate R and N_{FFT} is the FFT size. Since the VSG has a locked sample rate of 2.4 Gsps the two variables that can be adjusted are the resampling rate and the size N_{FFT}.

The required subcarrier spacing is 960 kHz and 524 subcarriers must be usable, giving the lower limit of the FFT size $N_{\mathrm{FFT}} = 524$. The FFT is most effective in terms of number of computations when the length is a power of 2 so the smallest appropriate size is 1024. Using this FFT size the number of samples per symbol becomes:

$$R = \frac{F_s}{N_{\mathrm{FFT}} \cdot \Delta f} = \frac{2.4\,\mathrm{Gsps}}{1024 \cdot 960\,\mathrm{kHz}} = \frac{625}{256}. \tag{3.2}$$

As it is not possible to reduce this fraction further, the signal must be upsampled 625 times before downsampling 256 times. This requires not only a very large amount of memory, but very long and therefore computational complex filters. With poly-phase filtering the complexity can be reduced by not including the components of the filter that are zero, however a large number of filters are still required, meaning memory, and a lot of complexity to account for switching between filters [24, Ch. 10].

Another option is to not adhere strictly to FFT sizes that are powers of two. The reduction in complexity of the FFT comes from the decomposition of the full N-DFT calculation into two smaller $N/2$-DFT. This is known as radix-2 decomposition, and is what gives raise to the requirement that N_{FFT} is a power of 2. By using decompositions with other radix than two, much more flexibility is achieved while still retaining much of the complexity benefit, especially if the factorization is done into small prime numbers. This is known as the mixed-radix FFT [16, Sec. 9.5.1]. An example of this is the FFT

Table 3.2: Table of the integer FFT sizes following (3.1) for different resampling rates and their prime factorizations. Only contains the FFT sizes larger than the 524, as the size must be larger than the number of subcarriers required.

R	N_{FFT}	*Prime factors*
1	2500	$2^2 \cdot 5^4$
$\frac{5}{4}$	2000	$2^4 \cdot 5^3$
2	1250	$2 \cdot 5^4$
$\frac{5}{2}$	1000	$2^3 \cdot 5^3$
$\frac{25}{8}$	800	$3^3 \cdot 5 \cdot 7$
4	625	5^4

size of 1536 used in LTE [25, pp. 17]. The paper commonly credited for the FFT, [26], also mentions this and notes that the radix-3 decomposition is the most efficient in terms of number of operations but radix-2 has other advantages. From radix-3 and on the efficiency decreases, so as low radix as possible is wanted. The article adds that it is possible to select any radix below 10 without increasing computations more than at most 50 % per decomposition.

An approximate number of operations for a size N, given repeated decomposition in radix r, can be found with the recursion:

$$\text{ops}_r(N) = \begin{cases} N(r + r - 1) + \text{ops}_r(N/r) & \text{for } N > R \\ N(N + N - 1) & \text{for } N \leq R. \end{cases} \tag{3.3}$$

It should be noted that number of operations is not always a good metric for comparing the performance of algorithms. For instance, the software library FFTW does not perform the FFT decomposition all the way to a prime number, but instead to a size that depends on the cache size of the executing machine [27]. Benchmarks of the options in Table 3.2 are shown in Appendix A.

In Table 3.2 all options for integer FFT sizes and their prime factorizations are listed, for which the resampling rate is a a multiple of $\frac{1}{8}$. In comparison, selecting the minimum size $N_{\text{FFT}} = 524$ factorizes into $2^2 \cdot 131$ with a resampling rate of $R = 625/131$ thus not only factorizing worse for the FFT, but also requiring more complex resampling. The number $N_{\text{FFT}} = 625$ can factorize into 5^4 which is a relatively small prime number. Because of the lower memory and computational requirements, it is chosen to use this FFT size. As the VSA also samples with a sample rate of 2.4 GHz, a downsampling operation is added in the receiver, corresponding to the upsampling. Going forward *system sample rate* is used for signals that are not yet upsampled, while *RF sample rate* is used for signals that are.

With $N_{\text{FFT}} = 625$ more subcarriers are introduced compared to the specified 524. Since a bandwidth of 500 MHz is still targeted the unused $625 - 524 = 101$ subcarriers are set to zero. With this configuration a filter length of 31 taps is found to provide good dampening while not affecting performance substantially. Since specific spectral limits are not specified for this project, no further refinement is done to change this size.

3.3.2 SYNCHRONIZATION

Section 2.3.6 describes issue with synchronization that must be dealt with in order to successfully receive a signal. As the LitePoint tester uses the same oscillator for both the VSA and VSG very little frequency offset is observed and no synchronization is required

to compensate. In this prototype the receiver also knows when the transmitter starts a transmission so any time offset that occurs will be from wave propagation delay and inaccuracies in triggering.

It was noted during testing with a wired connection that a steady offset exists, which is to be expected. Furthermore, it was found that the offset changes randomly with 8 samples, requiring time synchronization to correctly locate symbols. It should also be noted that over the air tests will require time synchronization in any case. The distance between the transmit and receive antennas can vary at least between (approximately) zero and 10 m, which is a difference in delay of:

$$T_{\text{ToF}} = \frac{d}{c} \approx 33.36\,\text{ns}. \tag{3.4}$$

Or correspondingly in samples at the system sample rate:

$$S_{\text{ToF}} = T_{\text{ToF}} \cdot 600\,\text{Msps} \approx 20\,\text{samples}, \tag{3.5}$$

where d is the distance between the transmitter and receiver and c is the speed of light. Since this distance is not known, time synchronization is required.

It is chosen to perform this synchronization using a Zadoff-Chu (ZC) sequence. A ZC sequence is the waveform used in multiple places in LTE, for instance the primary synchronization signal[7]. ZC sequences are defined as [28]:

$$x_u(n) = \mathrm{e}^{-j\frac{\pi u n (n+(N_{\text{ZC}} \mod 2))}{N_{\text{ZC}}}} \quad 0 \leq n \leq N_{\text{ZC}}. \tag{3.6}$$

Each sequence is parameterized by two variables, the length of the sequence N_{ZC} and an integer u. If u is selected to be relatively prime to N_{ZC}, then the autocorrelation of the sequence is zero at every non-zero delay [28]. It is also straightforward to see from Equation (3.6) that the sequence has a constant amplitude. It is possible to show that this property continues to be true for variations such as cyclic shifts or additions of constants [28]. Thus, if a ZC sequence is shared between the transmitter and receiver, the receiver can identify time offsets using correlation.

For the prototype a Zadoff-Chu sequence of length $N_{\text{ZC}} = 17$ and the relative prime number $u = 7$ is chosen. This length is chosen arbitrarily to be as high as possible, but lower than the maximum delay in Equation (3.5). Further development should reconsider this length.

3.3.3 CHANNEL EQUALIZATION

Channel estimation is performed using pilots as described in Section 2.3.7. The WIRT specifications in Section 2.4 describe a choice of 124 subcarriers used for pilots per OFDM symbol.

To get an estimate of the channel in the other 400 data carrying symbols, linear interpolation is used. With an estimated channel $H[k]$ at frequency f_k, the interpolation is performed as:

$$\hat{H}[k] = H_0 + (f_k - f_0)\frac{H_1 - H_0}{f_1 - f_0}. \tag{3.7}$$

In the equation f_k is frequency corresponding to interpolated channel estimate $\hat{H}[k]$ and H_0, H_1 correspond to the two estimated channel responses that are closest in frequency

to the point k with lower and higher frequency respectively. The effect of the linear interpolated channel is shown in Figure 2.4. Using this interpolated channel estimate, equalization is performed by:

$$\hat{x}[k] = \frac{x[k]}{\hat{H}[k]}, \tag{3.8}$$

giving the equalized symbols $\hat{x}[k]$ from the unequalized symbol $x[k]$. This type of equalization is known as *zero-forcing*. It is a relatively simple method for equalization and suffers from problems of noise enhancement, especially when the channel has large gaps of very low channel gain at certain frequencies [29]. Other types of equalization exists that are better able to account for the noise power at the cost of increased computational complexity. Equalization is an important part of a real wireless system, however also a large and complex topic in itself. For this project it is therefore chosen to keep the zero-forcing equalizer, but further studies should consider better equalizers.

3.3.4 SOFT DEMODULATION

Soft demodulation is the process of calculating probabilities to go along with the demodulated values at the receiver. For each symbol, the log likelihood that the symbol received x_r is symbol transmitted x_t is:

$$\mathrm{LLR}(x_r) = \log\left(\frac{\Pr(x_r = x_t | r = (I,Q))}{\sum_{\mathrm{x_i} \neq \mathrm{x_t}} \Pr(x_r = x_i | r = (I,Q))}\right), \tag{3.9}$$

where I and Q are the real and complex part of the received sample. Assuming an AWGN channel, the noise is Gaussian and the received signal is described by a normal distribution with mean given by the sent symbol and variance equal to the noise variance.

For QPSK modulation the bits can be compared individually, instead of as part of a symbol, with minimal loss. In other words, if the constellation is chosen correctly, the I and Q components can be decoded individually without significant increase in decoding error [14].

Thus, when considering only either the real or the imaginary part of the received signal, the Log-Likelihood Ratio (LLR) for received value z becomes:

$$\mathrm{LLR}(z) = \log\left(\frac{f(z|\mu = 1/\sqrt{2}, \sigma = \sigma_n)}{f(z|\mu = -1/\sqrt{2}, \sigma = \sigma_n)}\right) = \log\left(\frac{\frac{1}{\sqrt{2\pi\sigma_n^2}} e^{-\frac{1}{2\sigma_n^2}(z-1/\sqrt{2})^2}}{\frac{1}{\sqrt{2\pi\sigma_n^2}} e^{-\frac{1}{2\sigma_n^2}(z-(-1/\sqrt{2}))^2}}\right), \tag{3.10}$$

where $f(z|\mu,\sigma)$ is the PDF of a normally distributed random variable $Z \sim \mathcal{N}(\mu,\sigma)$ with mean μ and variance σ^2. σ_n^2 is the variance of the noise. Equivalently the expression becomes:

$$\begin{aligned}
\mathrm{LLR}(z) &= -\frac{1}{2\sigma_n^2}(z - 1/\sqrt{2})^2 - \left(-\frac{1}{2\sigma_n^2}(z + 1/\sqrt{2})^2\right), \\
&= \frac{1}{2\sigma_n^2}\left((z + 1/\sqrt{2})^2 - (z - 1/\sqrt{2})^2\right), \\
&= \frac{\sqrt{2}}{\sigma_n^2} z.
\end{aligned} \tag{3.11}$$

```python
def qpsk_demodulate_soft(data, ESNO):
    """
    Soft demodulate the QPSK bits based on an AWGN channel.
    Returns the LLRs on the I and Q channels in the real and complex parts of the result
     respectively.
    This is NOT the symbol LLR.
    """
    inv_noisepwr = 1 / (10**(-esno / 10))

    llrs = (np.sqrt(2) * inv_noisepwr) * data

    return np.stack((llrs.imag, llrs.real), axis=-1)
```

Code snippet 3.1: Python code for calculating the LLR values for the complex received symbols in the vector `data`

Note that LLR in this case refers to the likelihood that the original bit is zero compared to the probability it is one. In other words, a positive LLR represents a zero and a negative represents a one. Corresponding Python code for calculation of the LLR values is shown in Code snippet 3.1.

For non-AWGN channels where equalization is used this method can still be used, albeit with higher noise level if the equalization is not perfect.

Soft demodulation requires an estimate of the SNR. An estimate of this number can be found from the channel estimation. For the simple channel estimation used in this project an estimate of the signal power can be found by summing up the energy in the pilots. This can be compared to a reference noise power to find an estimate of the SNR. The noise reference can be measured when no transmissions are present, or found based on used bandwidth and the specifications of the receiver.

3.3.5 MATCHING THE WIRT SPECIFICATION

When a user in the WIRT system is allocated a time slot it will want to use as much of the allocated transmission time as possible to improve the reliability. For the implementation the number of OFDM symbols that can fit in a single transmission on the WIRT system therefore has to be determined.

In the previous section, Section 3.3.1, an FFT size of $N_{\text{FFT}} = 625$ is chosen. This number is also the total number of subcarriers per OFDM symbol and the number of samples in the time domain after the IFFT. For every OFDM symbol, a cyclic prefix of duration $T_{\text{CP}} = 69\,\text{ns}$ is required (specified in Section 2.4). In terms of samples in the system sample rate this corresponds to:

$$S_{\text{CP}} = T_{\text{CP}} \cdot F_s = 69\,\text{ns} \cdot 600\,\text{Msps} = 41.4\,\text{samples}, \tag{3.12}$$

where F_s is the system sample rate before interpolation. Since it is only possible to have integer samples sizes, the total number of samples to be transmitted per OFDM sample is:

$$S_{\text{OFDM}} = \lceil 41.4 \rceil + 625 = 667 \text{ OFDM samples}. \tag{3.13}$$

This is again at the system sample rate before resampling. At resampling rate $R = 4$, the time it takes to transmit a OFDM symbol is given as:

$$T_{\text{OFDM}} = S_{\text{OFDM}} \cdot \frac{R}{F_s} \approxeq 1.11\,\mu\text{s}. \tag{3.14}$$

The WIRT specifications in Section 2.4 specify the minimum slot size as 20 µs. Since this is the minimum duration, the number of OFDM symbols per transmission becomes:

$$N_{\text{OFDM}} = \left\lceil \frac{20\,\mu\text{s}}{T_{\text{OFDM}}} \right\rceil = 18. \tag{3.15}$$

The number of OFDM symbols per WIRT transmission is closely related to the number of bits per transmission. Each OFDM symbol has 400 subcarriers available for data and each subcarrier carries a QPSK symbol. The total number of bits per WIRT transmission is therefore:

$$N_{\text{bits}} = N_{\text{OFDM}} \cdot 800 \frac{\text{bit}}{\text{OFDM symbol}} = 14\,440\,\text{bit}. \tag{3.16}$$

3.4 TESTING, VERIFICATION AND OPTIMIZATION

With the implementation details described, the full WIRT prototype is built.

An important aspect of developing a system is testing that the code works as expected and verifying that the expected code fulfills the requirements. Furthermore, when developing the system even a small amount of optimization can enable faster development iteration.

As described in Section 3.1 the code is developed in several modules, that are subsequently connected to produce the full results. This practice enable individual testing of the modules during development instead of testing the full system at once. Further, when changing a module at a later stage individual tests help minimize software regressions.

Four test modules are used in this project; instrumentation tests, OFDM tests, ECC test and other miscellaneous tests. The instrumentation tests deal with the LitePoint instrument. Different download formats are tested and benchmarked, in addition a test that uploads and redownloads samples to confirm operation. The OFDM test compare the implementation to a similar implementation in MATLAB. These tests are supplemented with logic for creating spectrums to confirm the subcarrier mapping works as expected. The ECC tests evaluate the performance of a number of configurations of the polar codes described in Chapter 5. Finally, a few miscellaneous tests are used for testing the ZC implementation and similar.

In addition to tests, a number of tools were used during development to ease the work and improve the code quality. The first is a debugger. When working with complex chains of logic or highly data dependent modules, a debugger enables temporarily pausing the logic and retrieving the value of all variables. For finding edge case failures a debugger has greatly increased the speed at which bugs are found. The next tool is a profiler. When developing numerically complex software it can be difficult to determine which computations are taking up time. Furthermore, it is not always immediately clear which parts of an application is the bottleneck, as additional unintended operations may be introduced unknowingly. A profiler times and reports the execution time of the application, enabling the developer to get a clear picture of where the time is spent. During development of this prototype a profiler has helped multiple times by identifying seemingly insignificant operations that took up more time than expected. The debugger and profiler used during this project are incorporated in the Spyder development environment.

While Python is a highly productive language from a perspective of programmer time, in terms of computational efficiency it is not optimal. One of the main reason for this is the interpreted nature of the language. This enables fast prototyping and a large number of options for dynamically changing the environment as the program runs, however it also makes automatically compiled optimization difficult. In order to counter this problem Python includes features for extending with functionality written in C or Fortran. Scientific computing packages, such as the NumPy library, make heavy use of this feature to improve the performance.

With the prototype built and integrated with the LitePoint tester, a basic WIRT system is realised.

4 DELAY ESTIMATION

The goal of the project is to evaluate the achievability of the latency requirements for WIRT. In order to reason about the achievability of the very low latency, first a lower bound for the system delay is found. After the bound is found, the proportions of the delays is estimated to get a picture of where the computational complexity is located. Finally, appropriate hardware architectures are discussed.

4.1 STRUCTURAL DELAYS

To provide the lowest latency possible the system should start processing samples as early as possible. From the WIRT specifications in Section 2.4, the RF transmission takes up 20 µs on its own, a full fifth of the total targeted system delay. In order to provide the largest amount of processing time the processing should therefore start while the reception is still occurring. Not all components of the system work on a single sample at a time however. For instance, the FFT operation can not be performed before all N_{FFT} samples have been found and so introduce a delay in the system. The delay incurred here is present even if the computations were performed in zero time, and so provide the lower limit of the latency in the system. Since it is part of the structure and so not dependent on computations, it will be denoted *structural delay*.

4.1.1 TRANSMITTER

The transmitter is first analyzed for structural delays. During the section all data for transmission is assumed available, and the RF-block is assumed to take any samples presented without delay.

The diagrams in Figure 2.2 provided a good logical picture to understand OFDM chains in general. To better argue about the delays in the transmitter however, more specificity is required. In Figure 4.1a a diagram of the transmitter implementation is shown. The setup in the figure transmits one WIRT packet, from the information bits and all the way to RF. The ECC block is the first one, and takes all the information bits and encodes them. The bits are then split into frames that are the size of an OFDM symbols. A total of N_{OFDM} OFDM modulation operations are performed. Together with the ZC sequence used for time synchronization the OFDM symbols are then collected and serialized. Finally, resampling is applied and the sequence is sent to the RF components. Using this diagram, an expression for the delay incurred in the transmitter can be made:

$$D_{\text{transmit}} = D_{\text{ECC}} + D_{\text{split}} + N_{\text{OFDM}} \cdot D_{\text{OFDM mod}} + D_{\text{sync create}} + D_{\text{collect}} + D_{\text{resamp}}. \quad (4.1)$$

This expression is valid if all the operations are executed serially. In a similar manner, a diagram is shown in Figure 4.1b in order to create an expression of the OFDM modulation delay:

$$D_{\text{OFDM mod}} = N_{\text{SM}} \cdot D_{\text{SM}} + D_{\text{SCM}} + D_{\text{IFFT}} + D_{\text{CP}} \quad (4.2)$$

The expressions cover the delay if all the processing is done in a serial fashion. This is not the only option, as it is clear that the OFDM blocks are independent with each other and with the generation of synchronization frame. Assuming unlimited hardware

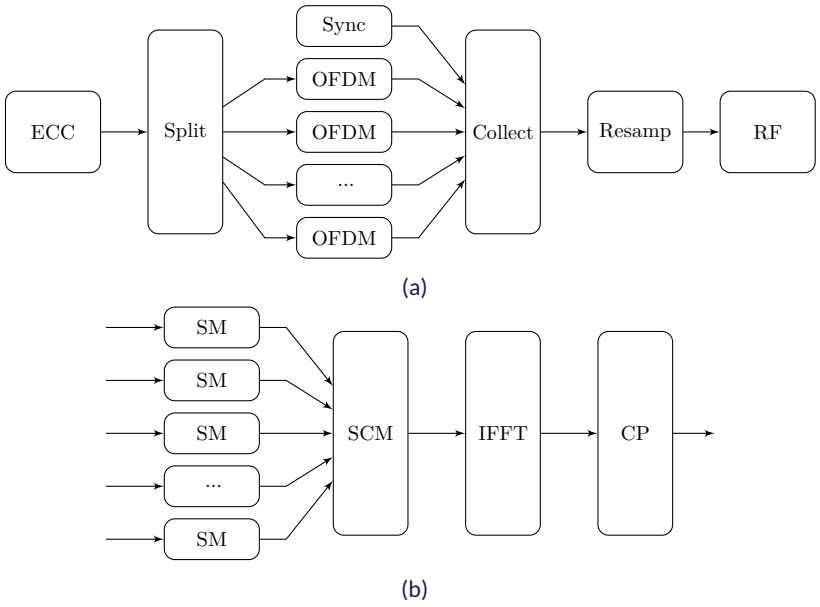


Figure 4.1: The implementation of the transmitter. (a) shows an overview and (b) is a diagram of the operations of the OFDM block itself.

an expression is made for the lowest possible delay:

$$D_{\text{transmit min}} = D_{\text{ECC}} + D_{\text{split}} + D_{\text{OFDM mod min}} + D_{\text{collect}} + D_{\text{resamp}}, \tag{4.3}$$

and

$$D_{\text{OFDM mod min}} = D_{\text{SM}} + D_{\text{SCM}} + D_{\text{IFFT}} + D_{\text{CP}}. \tag{4.4}$$

This delay is the with maximum parallelization, where all the OFDM modulations and SM operations are happening at the same time. Equation (4.3) assumes that $D_{\text{OFDM mod min}} > D_{\text{sync}}$, which is seen later to be a reasonable assumption. These expressions do not cover any further parallelization or speedup that can happen within the operations themselves.

An estimate of the processing delay of each operation is given in the next section. For now note that if all the information bits are available immediately when processing is started, there are no places where delay occurs aside from processing delay.

4.1.2 RECEIVER

In order to assess the delays in the receiver chain a similar diagram is made. Likewise an expression for the overall delay is made:

$$D_{\text{receive}} = D_{\text{resamp}} + D_{\text{sync}} + D_{\text{split}} + N_{\text{OFDM}} \cdot D_{\text{OFDM demod}} + D_{\text{collect}} + D_{\text{ECC}}. \tag{4.5}$$

This expression is very similar to the expression in the transmitter, however there is a noteable difference in the assumptions; the received samples are not all available immediately. Using the $R = 4$ downsampling with the RF sample rate $F_s = 2.4\,\text{GHz}$ of the tester, the time it takes to transmit a single sample is:

$$T_{\text{samp}} = \left(\frac{F_s}{R}\right)^{-1} = 1.667\,\text{ns}. \tag{4.6}$$

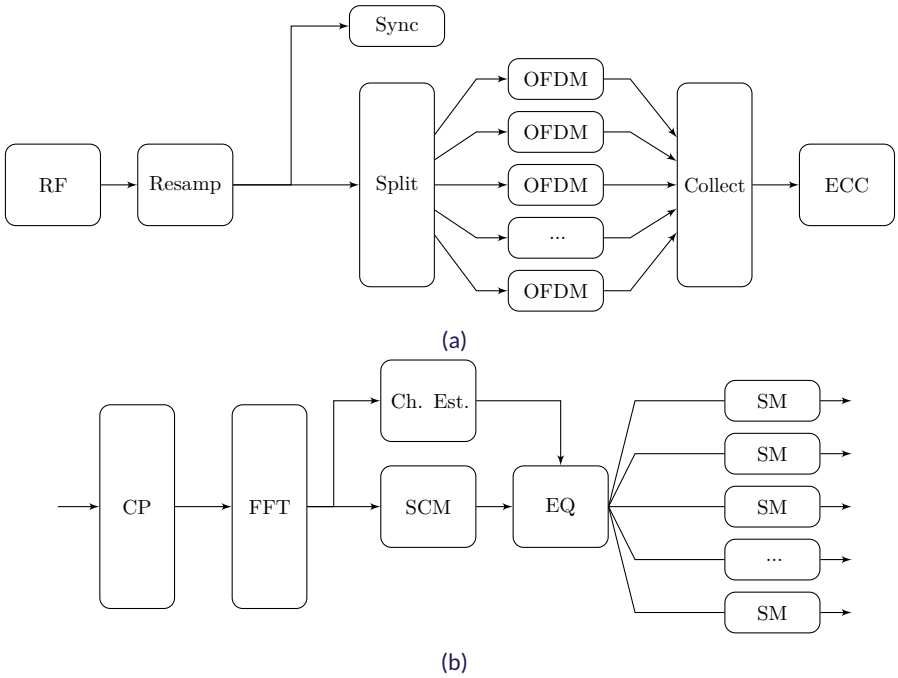


Figure 4.2: The implementation of the receiver. (a) shows an overview and (b) is a diagram of the operations of the OFDM block.

Thus, if $N_{\mathrm{ZC}} = 17$ samples are required before the synchronization can even start, a delay is incurred on the order of:

$$D_{\mathrm{pre-sync}} = N_{\mathrm{ZC}} \cdot T_{\mathrm{samp}} = 28.333\,\mathrm{ns}. \tag{4.7}$$

The same is clearly true for the rest of the system. For instance, demodulation of the first OFDM symbol requires both synchronization to finish and $S_{\mathrm{OFDM}} = 667$ samples to be available. Even if the synchronization is ready as soon as the S_{OFDM} samples are ($D_{\mathrm{sync}} = 0$), demodulation can only begin after:

$$D_{\mathrm{pre-OFDM}} = D_{\mathrm{pre-sync}} + S_{\mathrm{OFDM}} \cdot T_{\mathrm{samp}} = 1.140\,\mu\mathrm{s} \tag{4.8}$$

The maximum delay occurs if all the samples are to be received before processing begins. In this case:

$$D_{\mathrm{structural\ max}} = (N_{\mathrm{ZC}} + N_{\mathrm{OFDM}} \cdot S_{\mathrm{OFDM}})T_{\mathrm{samp}} = 20.038\,\mu\mathrm{s} \tag{4.9}$$

The ECC used in this project uses a low rate and repetitions. In good conditions where SNR is large it is likely that the original message can be recovered before all the repetitions are decoded. If a mechanism is introduced to discern if a packet is correctly decoded or not, the system can stop decoding of further packets thus reducing both decoding time and power used. Further considerations of how this is to be implemented is delegated to furture studies and will not be made here.

Combining the expressions for delay in the receiver and the transmitter the worst case serial latency becomes:

$$D_{\mathrm{max}} = D_{\mathrm{structural\ max}} + D_{\mathrm{transmitter}} + D_{\mathrm{receive}}. \tag{4.10}$$

This is however the worst case delay, as not only is the processing waiting for all samples to arrive before processing, the processing in both the transmitter and the receiver happens serially.

4.2 ESTIMATION OF OPERATIONS

With expressions for the delay established, an estimate is now made in order to determine the feasiblity of upholding the deadline. Since different hardware configurations heavily affect the performance of algorithms the idea is not to identify the exact timings, but instead build an understanding of where the bottlenecks might be. An idea of what the most significant computations are also enables a better choice of which architecture might be most appropriate.

The transmitter delay in Equation (4.1) has six components. First the delays can be split into the ones that occur where computations are performed and ones that occur where data is split or moved in memory. In the transmitter the *Split* and *Collect* blocks represent reinterpretations of bit structures; in the *split* block a single array is split into N_{OFDM} smaller arrays. Since the block simply reinterprets contents that are ready in memory it can be represented as moving a pointer to a memory address. These operations are therefore unlikely to be the largest contributers to latency.

The operations in the transmitter which require computation are the ECC, the OFDM operations, creating the synchronization sequence and resampling the signal. Since a definitive choice of ECC has not been made for WIRT at the time of writing it is hard to estimate the complexity. It is well known however, that the codes that have state of the art error correction performance require a significant amount of computations. It is also common for ECC to have much higher complexity in the decoding process than the encoding process. This is true for most codes such as convolutional codes, turbo codes, LDPC codes and polar codes [30, 13]. In the next chapter a number of different code options are evaluated and a hardware design is made for the selected polar code. For the purpose of the delay estimation it is sufficient to note that the ECC has a very high complexity in the receiver and a comparatively low in the transmitter.

The next computation is the OFDM modulation block. From Figure 4.1b and Equation (4.2) the block can be split into subblocks; the SM blocks, the SCM, the FFT and appending of the Cyclic Prefix (CP). The operations of mapping bits to symbols is straightforward to implement as a look-up table, in addition to being straightforward to parallelize, thus making the resulting complexity negligible. Similarly, as the allocation of subcarriers is constant, the operations in SM consists of placing the data samples in the correct memory locations. Appending the cyclic prefix is also a data copy operation. Compared to the three above steps, the FFT is a large undertaking. For a 625-point FFT, approximately 7020 complex Multiply ACcumulate (MAC) operations must be performed, see Equation (3.3). This block is clearly the dominant factor in the computational complexity of the OFDM block.

The last two computations are creating the synchronization sequence and resampling. Since the synchronization sequence is constant over transmissions it can be precalculated and recalled when required, thus not likely to be a limiting factor in the latency. Resampling requires allocating new memory and filtering with a FIR filter. The complexity of filtering is highly dependent on the length of the filter, but the operation is well studied and structures such as poly-phase filters for resampling reduce the complexity. For a resampling operation with a filter of length L_{filt} and an original sample count of M_{samples}

the number of multiply accumulate operations required is [24, ch. 10]:

$$\text{ops}_{\text{resamp}} = L_{\text{filt}} \cdot M_{\text{samples}}. \tag{4.11}$$

Thus, for the $M = 667$ samples per OFDM symbol, if the filter has more than $L_{\text{filt}} = 10$ then more MACs operations are to be performed here than in the FFT. However, the resampling operation is also highly dependent to the sample rate of the specific RF hardware used.

From the transmitter the most computationally intensive operations are the FFT and the resampling. An qualitative summary of the computations is shown in Table 4.1.

Compared to the transmitter the receiver is more complex. The resampling, cyclic prefix, SCM and FFT operations are similar to the transmitter and carry approximately the same complexity, but more operations are to be performed for soft symbol demodulation, synchronization and channel equalization. Soft demodulation of the symbols requires an estimate of the SNR and a complex multiplication per sample. These are relatively simple computations and not dominant for the delay.

As described in Section 3.3.2 only time synchronization is required for this project. To find the offset the cross-correlation is found between the known sequence and the received samples. The cross-correlation is defined as:

$$C[k] = \sum_{\text{n}=-\infty}^{\infty} S[k+n] \cdot ZC^*[n], \tag{4.12}$$

where $S[n]$ are the received samples, $ZC[n]$ is the ZC sequence and x^* represents the conjugate of a complex variable $x \in \mathcal{C}$. Since the signals are finite in length, for each k only N_{min} samples overlap where $N_{\text{min}} = \min(N_S, N_{\text{ZC}})$ is the length of the shorter sequence. The range of k is chosen within where the sequence is expected to be.

In Section 3.3.2 the maximum offset is determined to be $N_S = 20$ samples at the system sample rate of 600 Msps. To ensure the system continues working in the presence of slightly longer distance or other small delays, an additional 25% extra samples are included in the correlation. The ZC sequence has a length of $N_{\text{ZC}} = 17$ and is thus the smaller of the two. Only the points where the sequences fully overlap are included. The number of MAC operations required for such a cross-correlation is:

$$\text{ops}_{\text{sync}} = N_{\text{ZC}} \cdot (1.25N_S - N_{\text{ZC}} + 1) = 153. \tag{4.13}$$

This is a significantly lower number of computations compared to the FFT, and the synchronization is only performed once at the beginning of reception. The synchronization is therefore not a large part of the delay.

Only time synchronization is required for this project. Other, more extensive, synchronization methods that are required in a real system will likely take more processing to complete. Exactly how much is outside the scope of this report. Given that OFDM systems are well researched and see wide use, this is not considered deal-breaking.

Finally, the channel estimation and equalization are also unique to the receiver. In this project the channel is estimated using a single complex division per pilot subcarrier, followed by interpolation to match the number of data subcarriers. The interpolation can be done in a number of ways, including evaluating Equation (3.7) directly or using the same method as when resampling. Equalization is then performed with a single division per data subcarrier. Each OFDM symbol therefore requires 524 complex divisions and

Table 4.1: A qualitative overview of the approximate computational complexity of the operations in the transmitter.

Symbol	*Description*	*Computation*
D_{ECC}	Error Correction Code	Medium
D_{SM}	Symbol Mapping	Low
D_{SCM}	SubCarrier Modulation	Low
$D_{\text{sync create}}$	Creation of the ZC sequence	Precalculated
D_{IFFT}	Inverse Fast Fourier Transform	High
D_{CP}	Cyclic Prefix	Low
D_{resamp}	Resampling	Medium to high

Table 4.2: A qualitative overview of the approximate computational complexity of the operations in the receiver.

Symbol	*Description*	*Computation*
D_{resamp}	Resampling	Medium to high
D_{sync}	Synchronization	Medium
D_{CP}	Cyclic Prefix	Low
D_{FFT}	Fast Fourier Transform	High
D_{SCM}	SubCarrier Modulation	Low
$D_{\text{Ch. Est.}}$	Channel estimation	Medium
D_{EQ}	Equalization	Medium
D_{SM}	Symbol Mapping	Low
D_{ECC}	Error Correction Code	Very high

400 complex interpolation operations in total. Neither resampling or complex division are trivial operations to perform, but they are not dominant either. Since the focus of this report is not on channel estimation and equalization, they are not considered further. A qualitative overview of the receiver delays is presented in Table 4.2.

4.3 FEASIBILITY

Based on the above analysis and the benchmarking in Table 6.1 it is clear that the operations that introduce the highest latency are the ECC decoder, resampling and the FFT.

The ECC encompasses such a large part of the overall latency that further study is warranted. In Chapter 5 options for suitable ECC are surveyed, one is selected and implemented in the software prototype, and a basic hardware design is made for decoding. While resampling is a large part of the delay, it is chosen to delimit the project from further study of this topic. More information on algorithmic structures for effective resampling can be found in [16, 24, 31, 32]. Finally, the FFT is considered in more depth.

4.3.1 FAST FOURIER TRANSFORM

To get a preliminary estimate of the computational time for the FFT, benchmarks are performed on a development x86 machine. The results of these are shown in Table 4.3 for three different configurations; NumPys built-in FFT based on FFTPACK [33], and the FFTW library called from Python and C respectively [27]. To call the FFTW library from Python the wrapper library *pyfftw* is used.

Table 4.3: Timing results of calculating the FFT with $N_{\mathrm{FFT}} = 625$. The Python results are timed using ipython built-in magic command "%timeit" and given as mean ± std. dev. of 20 runs, 100000 loops each. The C FFTW result is measured with the POSIX `clock_gettime` with the CLOCK_PROCESS_CPUTIME_ID clock as the mean of 10000000 iterations.

FFT library	*Time*
np.fft.fft	16.2 µs ± 348 ns
pyfftw fft	39.6 µs ± 2.61 µs
C FFTW	3.11 µs

In total $2N_{\mathrm{OFDM}} = 36$ FFT operations are run in order to encode and decode. Even with the remarkably fast C code the total computing delay of the FFT alone would be:

$$T_{\mathrm{FFT\ total}} = 2N_{\mathrm{OFDM}} \cdot T_{\mathrm{FFT}} = 111.96\,\mu\mathrm{s}. \tag{4.14}$$

This is higher than the total allowable delay, ruling out the possibility that the full chain can be implemented without careful delay considerations.

4.3.2 DISCUSSION OF POSSIBLE ARCHITECTURES

The next step in improving the performance is therefore to reevaluate the computer architecture used for the project. One option is moving to a Graphics Processing Unit (GPU) based architecture. The system has a large number of data operations, but relatively few control operations. This is suitable for the SIMD based, deeply pipelined GPU architectures. The architectures are suited for high throughput, processing large amounts of data in parallel.

A disadvantage of this move is the relatively low flexibility of GPUs. These processors are made to be used in addition to a general purpose processor, with data transfered back and forth between the processors as required. Additionally, the time it takes to start up a GPU pipeline can be large, resulting in high latencies [34]. GPU computing is inherently designed for high throughput, more than low latency [35].

The next option is to move to a Digital Signal Processor (DSP) based architecture. DSPs are a type of microcontroller specialized for signal processing applications through hardware for accelerating common operations and memory suitable for data streaming. They can provide a large increase in performance compared to ordinary microcontrollers and can be useful for many signal processing applications. They also come with much of the same flexibility of all software programming. However, similarly to ordinary microprocessors they are limited to fast operations using the hardware they carry. While the processors are well-suited for FFT operations, decoders for polar codes are less common. Like a ordinary processor the decoder could be implemented, but the performance might suffer.

An alternative is to move to a hardware reconfigurable architecture, such as an Field Programmable Gate Array (FPGA). FPGAs are highly parallel devices that enable very high throughput of custom operations by direct design of appropriate digital circuits. Since the hardware can be designed to a specific application very low latencies can be achieved at the cost of developer time.

The FFT is a simple comparison of how much can be gained by parallelizing the calculation. FFT operations are highly parallel operations, enabling straightforward trade-off between area and latency. For their lineup of FPGAs Intel has released an FFT

generator supporting powers of two sizes FFT. The generated circuit can perform the transform in the same number of cycles as there are complex samples. In other words an FFT of size $N_{\mathrm{FFT}} = 1024$ has a latency of 1024 cycles. If the FPGA is clocked at 300 MHz the latency becomes 3.41 µs, almost most exactly the same as the measured results from the x86 processor at a much lower frequency. Compared to that processor however, in the FPGA it is also possible to create more than one FFT computation unit, thus resulting in lower latency by letting them work in parallel.

The FFT cores generated by the Intel IP generator are limited to power of two sizes. More flexible designs exist however, such as [36], which also achieves higher throughput and lower latency compared to the Intel core (in the paper denoted Altera).

In this chapter the delays of the WIRT transceiver were estimated to evaluate which parts introduce most latency. It is found in Section 4.1 that if processing is not started while reception is still ongoing more than 20 µs of delay are unavoidable structural delays. If processing of the first OFDM frames is started as soon as adequte samples have arrived, this delay is reduced to 1.44 µs. After the delays Section 4.2 finds that a very large part of the latency comes from decoding of the ECC. In order to hit the latency targets for WIRT, Section 4.3 considers multiple options for computing architectures. It is found that FPGAs are the most suitable target, in order to hit the latency requirements.

5 ANALYSIS OF ERROR CORRECTION CODE

As discussed in the previous chapters, the ECC used has a major impact on the performance of the communication system. Contrary to many of the other parameters of the WIRT system the ECC is not chosen yet. In this chapter a number of codes are evaluated and a suitable option is chosen. Section 5.1 reviews appropriate codes. An implementation of polar codes is described in Section 5.2. Finally, in Section 5.3, the latency of a polar decoder is evaluated and alterations to make the decoder more suitable for reconfigurable architectures are presented.

5.1 SELECTION OF AN APPROPRIATE CODE

A good starting point for modern ECC are the codes considered for 5G, especially the ones used for control channels and URLLC. The codes selected for eMBB are LDPC and Polar codes for data and control information respectively [37, tab. 5.3-1, 5.3-2]. In this context control information is information used for transmission scheduling, link adaptation and similar important, non-data information. Additionally, turbo codes are added to the comparison because of their use in LTE.

Turbo codes are a class of ECC that combine two or more convolutional codes to produce a result that is better than the sum of its parts. Along with convolutional codes themselves, this is one of the two types of ECC used in LTE [7]. The encoding is performed by passing the input through a series of concatenated convolutional codes. The decoder is also based on multiple decoders for the same convolutional codes, however with a twist. Instead of decoding the received signals separately, the two decoders share the information of their individual decodings repeatedly. This gives rise to an iterative decoding process, which is also the origin of the name [13, ch. 7]. Enhanced Turbo codes are in the same family of codes as the turbo codes used in LTE, however with significant improvements especially at coding rates below 1/3 and for high SNR scenarios [38].

LDPC codes are another commonly used type of ECC, but with fame from WiFi instead of LTE [39]. These codes are based on parity checks, where the parity of a bit sequence is sent together with the data. Which information bits each parity bit covers, also known as variable and check nodes respectively, is part of the code specification. Compared to other linear codes the number of information bits that are covered by each parity bit is relatively low, hence the name [13, ch. 5]. Most common LDPC decoders work by finding a solution to each of the sparse parity codes independently and afterwards collecting the results. The process is then repeated with the new parity estimates to give an iterative decoding process [13, ch. 5].

Finally, polar codes are a newer class of ECC originally proposed in [40]. Instead of building on concatenated convolutional codes as the two previous codes, polar codes are based on the concept of *virtual channels*. The encoding procedure maps the information to be sent to a set of virtual channels that are then converted to actual channel uses through recursive application of a short block code. It can be shown that when the mapping is reversed the virtual channels polarize, going to either perfect reliability or complete randomness. Further the fraction of reliable channels is equal to the capacity of the channel [40]. Polar codes are the first ECC to asymptotically achieve the channel

capacity [9, Ch. 11].

For URLLC the debate of coding solutions for URLLC continued until late into the standardization process [9]. It has been chosen for 5G NR to settle on one of the two codes used in eMBB in order to be able to reuse hardware [37]. Substantial evaluation has been performed on both codes for the URLLC scenario [41, 42, 43, 44]. While the sources disagree on which code is best, in the end one of the final documents before standardization of 5G NR [42] observe that polar codes have advantages over LDPC in terms of reliability and latency for small blocks, specifically when the payload is less than $400 \sim 500$ bits. Since this is the case in WIRT, this recommendation is useful.

Before the two codes were settled on others codes were also evaluated. One of the more prominent ones is enhanced turbo codes. These codes are well known from LTE, however improvements have been made since the standardization, including lowering of the mother code rate, improvements to the used convolutional codes, better rate adaptation and improved interleaver [9, Ch. 11]. One of the major advantages of enhanced turbo codes is that the complexity is flat across all rates [38, 45]. For the high rate codes used in eMBB the complexity is higher than corresponding LDPC codes, but when the rate is lowered this picture is reversed, and the LDPC decoders become more complex. This flat complexity can therefore be an advantage for the low rate scenario surveyed in this report.

[46] compares the performance of Tail Biting Convolutional Codes (TBCC) and polar codes at low rates. The findings are that it is possible to create a low complexity polar code encoder and decoder. Furthermore, the polar codes significantly outperforms the TBCC at low coding rates.

Outside of the 5G certification committee code selection for New Radio (NR) has also been discussed. In [47] polar decoders are reviewed in comparison with LTE turbo codes and LDPC codes used in WiFi, WiGig and Ethernet. The review concludes that while polar codes can match the error correction performance of LDPC and Turbo codes, the current hardware implementations do not have the throughput to match. Keeping this in mind, research continues in architectures for polar codes. In [48] many of the same authors of [47] create an implementation of what they call a Fast-SSC-List decoder which has a throughput of 12 Gbps and a latency of 0.54 µs. This is clearly low enough latency for the proposed system.

Comparing the different solutions no single code is the best at every scenario. Most of the codes described above can also be altered, improving their performance in the scenarios specific to WIRT. In this project it is chosen to focus on polar codes due to their novelty and them being the choice for high reliability codes in 5G. Future research should extend the comparison between codes in the WIRT scenarios based on the results of this project.

5.2 POLAR CODES

As described in the previous section, polar codes are a new class of ECC only recently introduced in [40]. The codes are the first to provably reach capacity asymptotically, but for shorter lengths the original decoder performance was not on par with LDPC and Turbo codes. Since then a number of improvements were found, bringing the performance on par and even above other modern codes [9].

In this section polar codes are described in sufficient detail for implementation in this project. Throughout the section, a single instance of encoding and decoding is described

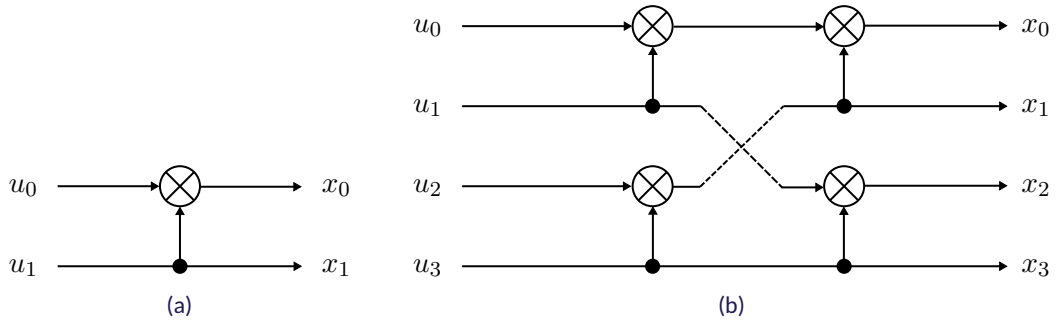


Figure 5.1: An illustration of the construction of polar codes. The round nodes represent XOR operations. (a) illustrates the basic operation, and (b) illustrates how the basic operations are combined for larger N_{polar}.

where N_{polar} is the number of channel usages available for transmission and K is the number of information bits to be sent.

The overall idea is to create a set of virtual channels based on the real channel uses. The real channels are modeled as failing with probability p_{fail}, but which bits fail is unknown and thus all the bits sent have equal likelihood of failing. The virtual channels on the other hand are constructed so the bits do not have equal probability. By encoding the bits together, the virtual channels *polarize* where the channels either become completely reliable with very low p_{fail} or completely unreliable where no information can be passed through. Note that the average amount of bits that can be transfered through N_{polar} transmissions does not change, only the distribution of the failures.

Since the bits can be deterministically ordered by reliability, the K information bits can be sent on the most reliable of the N_{polar} transmissions. The bits that are too unreliable to use are termed the *frozen bits*. While it is not required, they are customarily set to zero. The set of non-frozen (information) bit are denoted as A. Polar encoding transforms a vector of N_{polar} bits, denoted $u = \{u_0, u_1, \ldots, u_N\}$, to another vector of the same size, denoted $x = \{x_0, x_1, \ldots, x_N\}$. The subset of u that contains information is denoted $u_A \in \{0,1\}^K$, while the frozen bits are $u_{\text{A}_\text{c}} \in \{0,1\}^{N_{\text{polar}}-K}$.

As the main goal of this project is to evaluate polar codes for the use in WIRT, the theoretical proof for the effectiveness of channel polarization is not covered. The reader is instead referred to [40] for the original description, and to the significant amount of research on improving the codes later, e.g. [47, 49, 50].

5.2.1 ENCODING

Polar codes are made up of a recursive application of a basic transformation [40]. In Figure 5.1a the basic transformation is seen, where two bits u_0 and u_1 are mapped to two channels x_0 and x_1. Larger codes are created from repeating this transformation, as seen in Figure 5.1b. Here a $N_{\text{polar}} = 4$ code is created from two basic codes. Since all polar codes are created from this recursive application the sizes are limited to powers of two $N_{\text{polar}} = 2^{L_{\text{polar}}}$. Each increase in size adds another *layer* l, for a total of $L_{\text{polar}} = \log_2(N_{\text{polar}})$ layers. Between each layer a shuffling is performed. In Figure 5.1b this is marked with dashed lines.

```python
def polar_transform(u):
    if len(u) == 1:
        x = u
    else:
        u1u2 = np.mod(u[::2] + u[1::2], 2)
        u2 = u[1::2]

        x = np.concatenate((polar_transform(u1u2), polar_transform(u2)))
    return x
```

Code snippet 5.1: Recursive Polar encoder.

```python
def polar_transform_pipelined(u):
    N = len(u)
    N_half = N//2
    n = np.log2(N).astype(int)

    working_bits = u.copy()
    for n_i in range(n):
        u2 = working_bits[1::2].copy()

        working_bits[:N_half] = working_bits[::2] ^ u2
        working_bits[N_half:] = u2

    return working_bits
```

Code snippet 5.2: Non-recursive memory reduced Polar encoder.

The $N_{\text{polar}} = 4$ case shown in Figure 5.1 can also be expressed as a set of equations:

$$\begin{aligned} x_0 &= u_0 \oplus u_1 \oplus u_2 \oplus u_3, \\ x_1 &= u_2 \oplus u_3, \\ x_2 &= u_1 \oplus u_3, \\ x_3 &= u_3, \end{aligned} \tag{5.1}$$

where $\oplus$ represents modulo 2 addition. Looking at this equation it is straightforward to see that the encoding process can be created as a matrix multiplication. This is also how [40] originally describes the procedure. However, performing a matrix multiplication is not the most efficient manner of performing this encoding operation. The complexity is $O(N^2)$ and for large values of N it becomes infeasible. Another, much simpler, method is simply performing the operations recursively as shown in Code snippet 5.1. This method is much less computationally intensive as this time no multiplications are required. A disadvantage is that whenever a function is called in Python (and indeed most languages with recursion), some overhead is required. A better way is therefore to pre-allocate all the required memory and operate on in an in-place manner. Python code to perform this permutation is shown in Code snippet 5.2.

One of the difficulties of polar encoding is choosing the set A, that decides which bits are frozen and which are used for information. [40] proposes a Monte Carlo based sampling method, although the author notes that this method has several disadvantages. One of the disadvantages is the method is highly dependent on the channel. For a specific channel, a code can be constructed at a specific SNR and if the code is used far from this point the performance decreases.

During standardization of polar codes for 5G, another method was found. This so-

called *Polarization Weight* method sorts the virtual channels by their reliabilities and selects the K most reliable bits as the set A [50]. The major discovery was that the reliability of bits was corresponding to the bits set in the index. For an index $i \in \mathcal{Z}$ with bits $B_{\mathrm{n}-1}, B_{\mathrm{n}-2}, \ldots, B_0$, the polarization weight is:

$$W_i = \sum_{\mathrm{j}=0}^{n-1} B_j \cdot \left(\beta^j + \frac{1}{4}\beta^{(1/4)j} \right), \tag{5.2}$$

where β is an adjustable parameter. In [50] this is known as the Higher Order Polarization Weight method. The authors recommend a base of $\beta = 2^{1/4}$, a recommendation which is followed in this project. The set of indices A is created from the K values of i with the largest weight W_i. This method has the major advantage that it is independent of the channel and the channel parameters, in addition to the simplicity.

5.2.2 DECODING

If no noise is present on the channel the decoding can be performed by repeating the encoding process in Equation (5.1) with x and u switched. However, the goal of an ECC is to improve the error correcting performance in the noisy case. Even a single flipped bit would disable the above method, and other decoding algorithms must be used.

In [40] a low complexity Successive Cancellation (SC) decoder is presented. This decoder uses the bit likelihoods from soft demodulation, as described in Chapter 3, and transforms them in a similar manner to the encoder. The effect can be described as running the encoder in Figure 5.1 in reverse, but instead of performing the operation on bits, it is performed on probabilities. Transforming from likelihoods to bits is then a simple case of evaluating which bit $\hat{u}_i$ is most likely.

[40] describes the decoder in terms of probabilities; the probability that a bit is zero versus it is one. To improve numerical stability the same operations can be performed using log likelihoods [51]. As the soft demodulation is already performed in log likelihood domain in Section 3.3.4, this fits well with the rest of the system. When moving in reverse through Figure 5.1, the additions are replaced with an F operation and the branching is replaced by the G operation. In the log likelihood domain these are [51]:

$$F_{\mathrm{LLR}}(\mathrm{LLR}_a, \mathrm{LLR}_b) = 2\tanh^{-1}\left(\tanh(\mathrm{LLR}_a/2)\tanh(\mathrm{LLR}_b/2)\right), \tag{5.3}$$

and

$$G_{\mathrm{LLR}}(\mathrm{LLR}_a, \mathrm{LLR}_b, u) = \begin{cases} \mathrm{LLR}_b - \mathrm{LLR}_a & \text{if } u = 1 \\ \mathrm{LLR}_b + \mathrm{LLR}_a & \text{else.} \end{cases} \tag{5.4}$$

Using the F and G functions, a polar code can be visualized in another way. [52] presents a tree structure for polar codes. An example of this structure is shown in Figure 5.2 for a polar code of size $N_{\mathrm{polar}} = 8$. The tree consists of nodes and branches and branch represents either an F or a G operation. Each node v_l^n has a single parent and two children, and can be identified with a layer l and an index n. The root of the tree, the top node, contains the input LLR values from the channel and is classified as layer 3.

Each node in the tree except the root node gets a list of LLRs from its parent node, performs computations on them and returns a list of bit estimates back to the parent. In Figure 5.3 a single node v is shown. α represents LLRs and β represents bits. The order

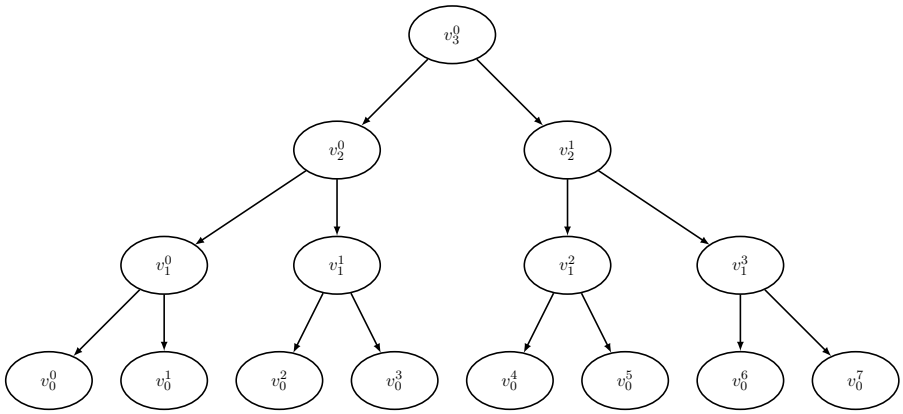


Figure 5.2: A tree structure used to represent the decoding process of polar codes based on message passing. A node u_l^n is the n-th node in layer l. The decoding process moves from the top-most layer and down, left to right. When moving down the LLRs are passed. When moving up the decoded bits are passed.

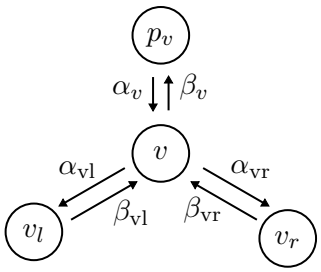


Figure 5.3: A single node in Figure 5.2. Information is passed from the parent node p_v to node v, which passes it on to children v_l and v_r. The bit estimates are passed from the children back up to v, where it is again passed on to p_v. α represents LLRs values and β represents bit estimates.

is as follows: p_v passes α_v to the node v. v performs the F operations to produce α_{vl} which is passed to the left child v_l. When v_l is done processing, it passes the bit estimate β_{vl} back up to v, that uses this estimate to perform the G operation and thereby creates α_{vr}. α_{vr} is passed to the node v_r, which again returns a bit estimate β_{vr}. With the bits collected from v_l and v_r, v computes and returns the final bit estimate β_v back to the parent node.

This process repeats itself at every node to produce the bit estimates. At the root node, α is a vector the size of the polar code. Each time it is passed to the child nodes the vector is reduced to half the size, until it reaches the leafs of the tree. At the leaves the bits are evaluated, immediately returning a bit estimate.

The central recursive function implementing this is shown in Code snippet 5.3. In the first call to the function `av` is the vector of channel LLRs, `all_bits` is a reference to an empty list and `A_set` is a set of indicies describing the positions of the non-frozen information bits. During operation the function fills the output bits into the `all_bits` list, that serves as the output. It is straightforward to show that each layer requres $N/2$ F and G operations, for a total of $\frac{1}{2}N \cdot \log_2(N)$ operations of each.

Many alterations to the original decoder described in [40] exist. This decoder does not achieve ECC performance comparable to LDPC and Turbo codes until code lengths become significantly longer than is common in systems today. To overcome this problem [53, 49] describes a method of improving the ECC performance by extending the SC

```python
def f_op(LLR0, LLR1):
    """
    Perform the F operation in a vectorized manner.
    """

    return 2 * np.arctanh(np.tanh(LLR0/2) * np.tanh(LLR1/2))

def g_op(LLR0, LLR1, u):
    """
    Perform the G operation in a vectorized manner.
    """

    mask = (u == 1)
    LLR0_signed = LLR0
    LLR0_signed[mask] = -LLR0_signed[mask]

    return LLR1 + LLR0


def polar_recurse(av, all_bits, A_set):
    """
    Perform the recursion central to the SC decoding of polar codes.

    av are the log likelihoods from the parent (caller).
    all_bits is list for the final bits.
    A_set is a set of the non-frozen bit locations.
    """
    if len(av) == 1:
        # This is a leaf node
        bit_i = len(all_bits)
        if bit_i in A_set:
            cur_bit = np.array(av < 0, np.int8)
        else:
            cur_bit = np.array([0], np.int8)

        all_bits.extend(cur_bit)
        return cur_bit

    # Not a leaf node
    av_len_half = len(av) // 2
    x = av[:av_len_half]
    y = av[av_len_half:]

    # Left child
    avl = f_op(x, y)
    bvl = polar_recurse(avl, all_bits, A_set)

    # Right child
    avr = g_op(x, y, bvl)
    bvr = polar_recurse(avr, all_bits, A_set)

    # Combine and return
    bv0 = bvl ^ bvr
    bv1 = bvr
    result = np.concatenate((bv0, bv1))

    return result
```

Code snippet 5.3: Python implementation of the F and G operations, and the recursive function that calls them.

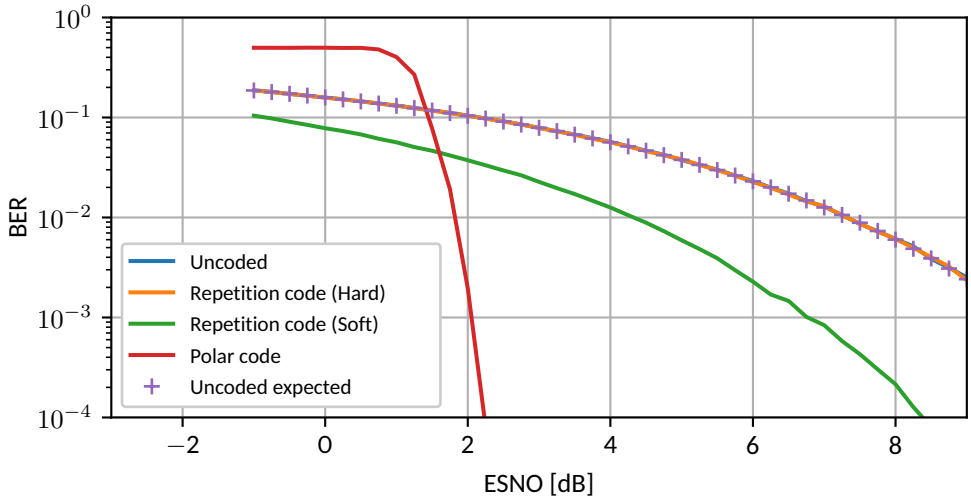


Figure 5.4: Comparison of Rate $R = 1/2$, $N = 2^{13}$ BER performance between no ECC, repetition codes and polar codes. Analytical results for uncoded QPSK are taken from [14]. The repetition codes shows the difference in performance between ECC on hard and soft demodulation outputs.

decoder to calculate multiple paths through the decoding. This algorithm, known as Successive Cancellation List (SCL) is the basis for 5G implementation. Since the SCL is an extension of an SC decoder, it is chosen to focus on the basic decoding algorithm.

5.2.3 VERIFYING THE DECODER

A software implementation of the encoder and decoder above is made to serve as a basis for algorithmic tuning and hardware design. Because of the many augmentations that are possible on polar decoders, the performance is instead evaluated and compared with implementations from literature. The first thing to do however, is to show that the decoder performs better than if no coding, or repetition coding is used. In Figure 5.4 the polar code Bit Error Rate (BER) performance of the polar code is compared to no ECC and repetition codes. The BER is the ratio of bits that are unsuccessfully decoded to the total number of transmitted bits. The repetition code is shown twice, once for combining after hard decisions and once for combining the soft decision LLRs. Additionally, the plot shows the analytically expected BER of uncoded QPSK modulation. It is clear that the polar code corrects significantly more errors than the repetition code is capable off, and is much better than no ECC.

For the WIRT project fairly low ECC rates are required. Figure 5.5 show a comparison between the polar code implementation at different rates. Here the results are given in BLock Error Rate (BLER) instead of BER as a transmission is considered a failure if even a single bits is wrong. It is clear that the lower the rate, the more noise can be tolerated, just as expected.

Finally, the performance is compared to two other polar code results. The results named *MATLAB* are from the MATLAB polar code toolbox at [54]. The results denoted *Tal Vardy* are from the SC decoder used for comparison in [49]. The graph shows that the decoder implementation is within the expected range of performance. The encoder from the MATLAB toolbox is created for 0 dB SNR to compare against the single creation used in this project. Additionally, it is found later that some of the alterations described

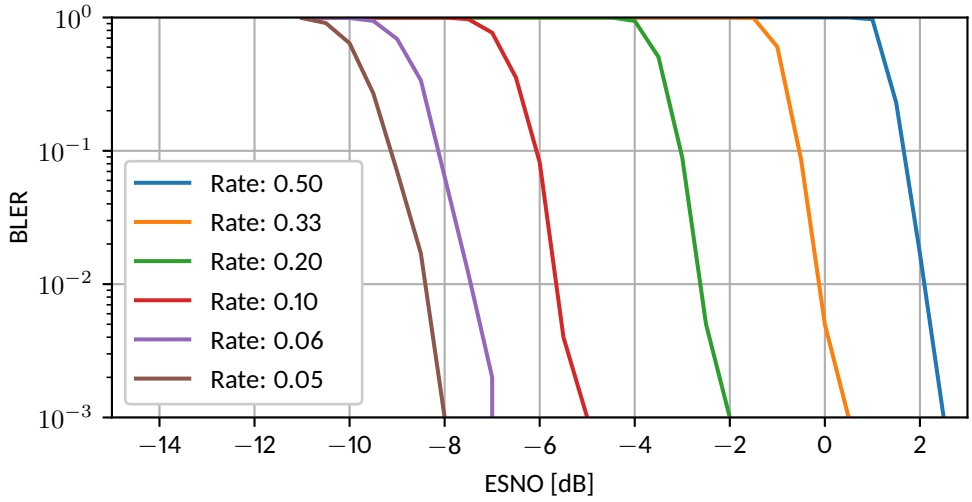


Figure 5.5: A comparison between polar codes showing the increasing ECC performance as the rate decreases. The implementation is shown to scale well with rate.

in the next section can improve the performance of the decoder slightly. On the other hand the results from Tal Vardy show that there are still options to improve the error correcting performance, even before moving to more advanced decoding methods such as SCL.

Figures 5.4 to 5.6 all give credence to the correctness of the implementation. In the interest of time this is deemed sufficient, but any future work based on this prototype should aim to improve the validation using comparsions with more code, analytical bounds and other channel models that more realistically represent real world conditions.

5.3 HARDWARE CONSIDERATIONS

Section 2.4 describes a latency requirement of 100 µs. This requirement is for the full end-to-end latency, including the application device using the communication system. To leave as much time as possible for the application, it is chosen to aim for a transceiver latency of 50 µs. In Chapter 4 decoding of the polar codes is seen to be the largest latency contributer with a large margin, however, it is not the only one. To give a sufficient amount of time for performing the other operations in the chain, it is further chosen to target a maximum latency of 10 µs for the decoder.

FPGA design is typically a trade-off between latency, throughput, area and power [55, 56]. Since the latency target dominates here the focus of this chapter is going to be meeting that goal. An implementation of this design should further consider this trade-off more carefully. Keeping that in mind the polar codes are reconsidered for hardware implementation.

Since FPGAs are typically not locked to a static frequency, it is not useful to express the latency in terms of time. Instead it is chosen to focus on *cycles*, where one cycle corresponds to one period of the clock. The cycles are therefore decoupled from the frequency of the clock itself. When measuring designs in terms of cycles it is important to keep in mind that the maximum frequency of an FPGA depends on the routing requirements of the design. The frequency depends, in part, on the propagation delay of

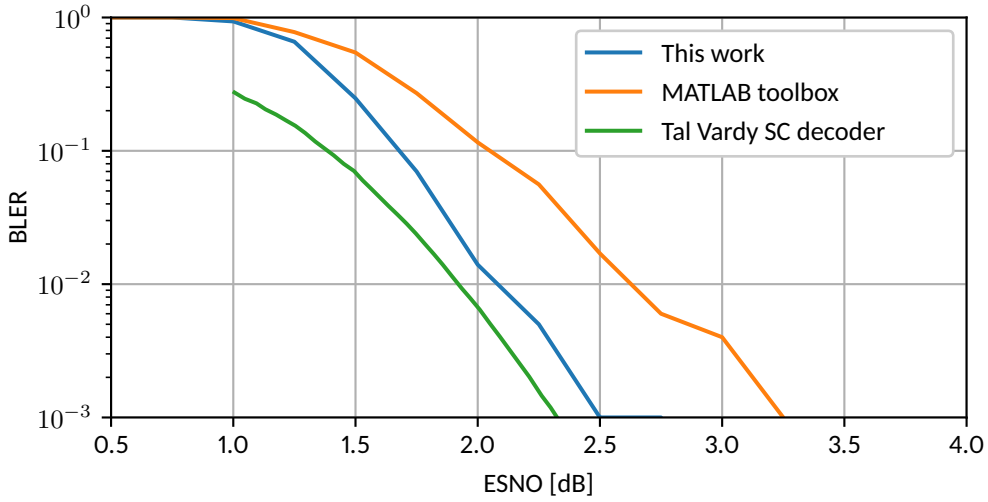


Figure 5.6

the longest chain of combinatorial logic, known as the critical path. For this project the operations are implemented in a single cycle and thus the more complex parts will have a longer combinatorial chain. It is assumed however, that the blocks can be implemented in a manner that they are not in the critical path.

With the specification already created in the form of the software implementation, the next step is to transfer the design to a Data Flow Graph (DFG). The aim of this representation is to visualize the data dependencies in an algorithm, in order to determine which operations are interdependent. The graph is shown in Figure 5.7 for a polar decoder of size $N_{\text{polar}} = 4$. It is pretty clear from already this small graph that the operations are heavily interdependent. This sequential dependency limits how much a circuit implementing the operations can be parallelized, and thereby bring down the latency. The only direct parallelization applicable to this specific graph without alterations is performing the two F operations in F_2 at once and the two G operations in the G_2 block. An illustration of an initial schedule is shown in Figure 5.8. The schedule is created using a *As Soon As Possible* method that finds the minimum latency given unlimited hardware. This parallelization brings the minimum latency to 6 cycles. According to [40, 51] any polar code can be decoded in $2 \cdot N_{\text{polar}} - 2$ cycles, given unlimited parallel hardware, and this result corresponds to that.

The DFG and initial schedule do not represent a full design, however they provide more direct information on the achievability of the latency requirement that is seen in Figure 5.2. The initial schedule also illustrates how the trade-off between power, area and latency is made on several different stages in the design. A general idea of the area can be had based on how many operations are executing concurrently. Each operation occurring concurrently requires more hardware. If the initial schedule is restricted to have only a single operator two more cycles are required for the calculation, but less hardware is required, trading off higher latency for lower area. Other techniques such as pipelining enable higher throughput at the cost of area while generally not affecting the latency.

With this representation a general idea of the latency of the final design is formed. The next step is to determine if the requirements of 10 µs latency can be achieved in

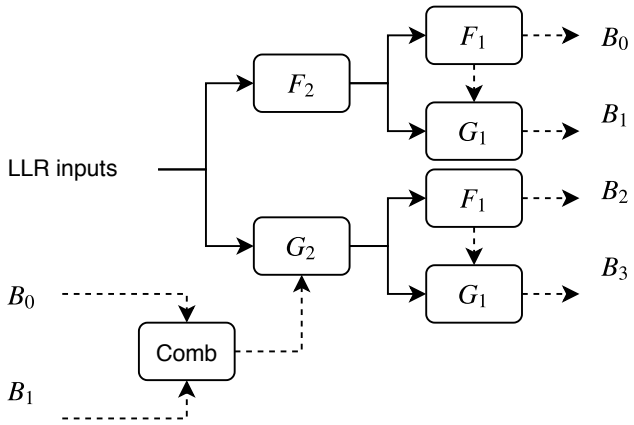


Figure 5.7: A dataflow diagram of the decoding operations in a polar code of size $N_{\text{polar}} = 4$. The solid lines represent LLR values and the dashed lines represent bit values. The output of the decoding process are bits B_0, B_1, B_2, B_3. Each block represents a mathematical operation with a number representing the size of the operation, e.g. the F_2 block takes in four LLR values and outputs two combined value. A line from block A to block B represents a dependency in block B on the values calculated in block A.

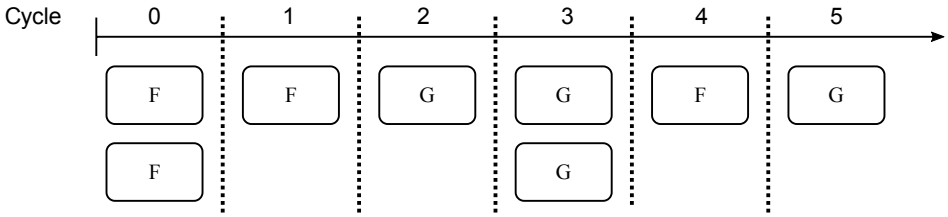


Figure 5.8: A schedule for the execution of the blocks in Figure 5.7.

realistic hardware. Again, this illustrates the iterative procedure of digital design, by revisiting the specifications from earlier. To perform the evaluation the area constraints are first ignored, in order to determine the minimum latency.

In Section 5.1 the total available bits for data transfer is determined to be 14400 bits with 400 bits of information. In order to get the best reliability possible, as many of the 14400 should be used as possible, however the polar codes described here are limited to powers of two $N_{\text{polar}} = 2^n$ for integer n. The approach used to deal with this problem in 5G is called *shortening*. When shortening a number of bits are not sent in order to match the available data rate, at the cost of some error correcting performance [37]. In order to perform shortening the next larger size N_{polar} is selected, before removing excess bits. With 14400 bits available, the next power of two is $N_{\text{polar}} = 2^{14} = 16384$. Considering only the time for decoding this length code has a delay of $2 \cdot 2^{14} - 2 = 32766$ ops. Sequentially decoding this code within 10 µs would require a operating frequency of:

$$\frac{2^{15} - 2}{10\,\mu\text{s}} = 3276.6\,\text{MHz}. \tag{5.5}$$

This frequency is much larger than is commonly achievable with commercial of the shelf FPGAs before considering any of the other components in WIRT or other system elements such as memory transfers.

Shortening is therefore not immediately applicable to this situation and an alternative must be found. Latency requirements are not always as tough as in the case of WIRT.

Table 5.1: The number of used and unused bits for different combinations of N_{polar} and repetitions. The repetitions can be used on different channels.

N_{polar}	*Repetitions*	*Used bits*	*Unused bits*
512	28	14336	64
1024	14	14336	64
2048	7	14336	64
4096	3	12288	2112
8192	1	8192	6208

Examples of designs that perform the same basic algorithm, but instead optimize for area or throughput can for example be found in [51, 57, 58].

For the WIRT use case the latency requirement requires a different solution to be found. One option is instead of creating a single very low rate code, a higher rate code is created and repeated. In addition to lower latency shorter, repeated packages also have other advantages. Having separate, repeated codewords makes the decoding of each codeword independent. Because it is independent it is now possible to start decoding at the moment the first codeword is available instead of waiting for all the data to arrive. Since they are independent the codewords can also be decoded in parallel. The channel is unlikely to change much in the short time of a single package, which is why seperate channels are used. The different channels will have different fading realisations and differing interference. Separate codewords can be split across different channels using channel hopping thus, in the general case, increasing the probability that a good realisation is used.

To select a combination of N_{polar} and repetition rates the powers-of-two between 512 and 8192 are tested. The combinations are shown in Table 5.1. The table shows the amount of bits that are actually used in this combination and how many are not. The first three options waste almost no bits, while in the last two a significant portion is not in use. In Figure 5.9 the performance is compared. It is interesting to see that the performance of $N_{\text{polar}} = 4096$ is approximately as good as $N_{\text{polar}} = 1024$ despite over 2000 bits being unused. The option with the best performance in this scenario is the $N_{\text{polar}} = 2048$ code, repeated 7 times. This option is therefore chosen, as it also represents amble opportunity to perform channel hopping.

The repeated codewords in Figure 5.9 are combined by summation of the received LLRs. Since the test is performed with Gaussian noise, this should be the optimal combination method for combining prior to decoding. To test if the performance improves when the samples are combined after decoding, the implementation is changed to save the LLRs used for bit decision throughout the decoding procedure, providing a soft output of the decoder.

Figure 5.10 shows the performance difference between combining the LLRs prior to decoding (pre-combining) and decoding each received codeword individually and combining their outputs (post-combining). No significant performance difference is seen in this case. In this test all the codewords have equal noise levels. In a more realistic channel, it is likely that the different channels are going to have different realizations and therefore different SNR and interference. If this is the case the two methods of combining are unlikely to be identical in performance.

Based on this information it is chosen to use polar codes of size $N_{\text{polar}} = 2048$, repeated 7 times on different channels. Since the seven repetitions are independent, they can be

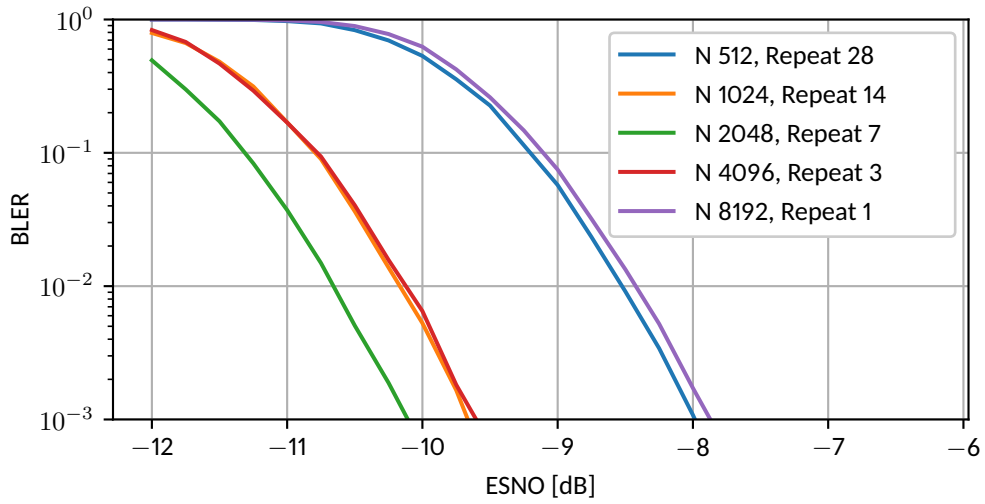


Figure 5.9: Comparison of BLER performance of combinations of polar codes sizes N_{polar} and repetitions that fit in 14400 bits. The $N_{\text{polar}} = 2048$ is the best with $N_{\text{polar}} = 4096$ and $N_{\text{polar}} = 1024$ trailing slightly behind. The $N_{\text{polar}} = 512$ does not perform well, likely because of the high rate. $N_{\text{polar}} = 8192$ also has disappointing performance, because of the large number of unused bits, see Table 5.1.

decoded in parallel. For codes of size $N_{\text{polar}} = 2048$ the required frequency for 10 µs latency is thus 409.4 MHz, an improvement, but still high.

5.3.1 SIMPLIFIED SUCCESSIVE CANCELLATION DECODER

While the number of clock cycles required for decoding of the shorter codes is more feasible, this is achieved without considering other overhead or area constraints. In order to get further improvements, the algorithm is reconsidered.

Consider again the tree representation in Figure 5.2. As described earlier, the leafs at the bottom of the tree represent each bit. When encoding only K of the N available bits are used for information, with the remaining bits being frozen and always set to zero. The tree representation can be extended to illustrate this extra information. The frozen bits are denoted as *rate 0* ($R0$) and the information bits as *rate 1* ($R1$). The branches connected to each leaf gets a rate that is the mean of the leaf nodes $0 \geq R_n \geq 1$. The tree is repeated in Figure 5.11, except now with rate information.

A few things can be learnt from this new representation. First of all, consider the operations that are performed on nodes that are known to be frozen and therefore zero. Since the bit is known to be zero, whatever LLR value is calculated at the leaf node is irrelevant and can be skipped. As a branch connected to multiple $R0$-nodes also becomes zero, calculating the LLR for it is just as irrelevant. Using this knowledge, the tree can be pruned by only keeping the top of an $R0$ chain, thereby reducing the operations. Another, similar structure can be seen a branch only has information bits. In this case the bits can be directly evaluated from the LLR, skipping the F and G operations that are normally required for the subbranches [52].

Both of these reductions work on multiple levels, not only at the leaf nodes. Since they represent small error correction codes on their own, they will be denoted as *subcodes*. The left-most nodes Figure 5.11 represent such as $R0$-subcode. The size of a subcode will be denoted as the number of leaf-nodes that are included, giving the $R0$-subcode a

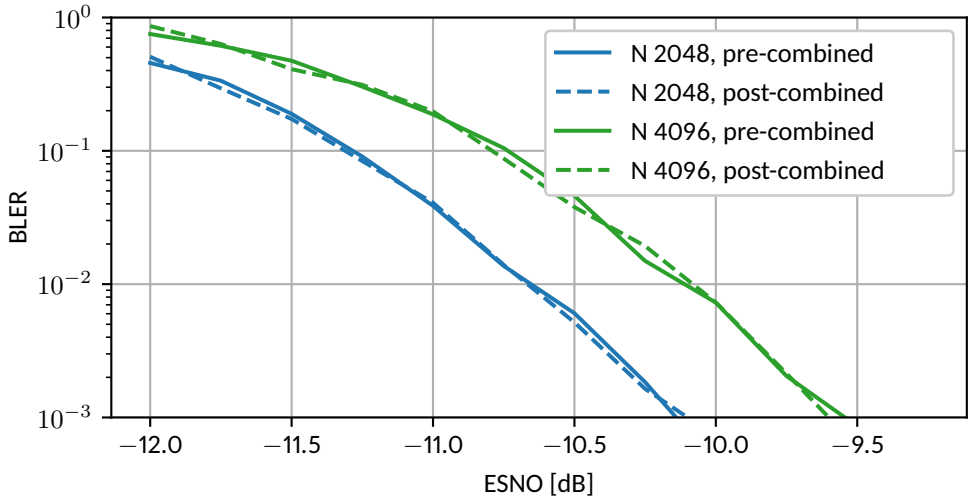


Figure 5.10: Comparison of BLER performance of polar codes when the LLRs are combined at the channel or after decoding.

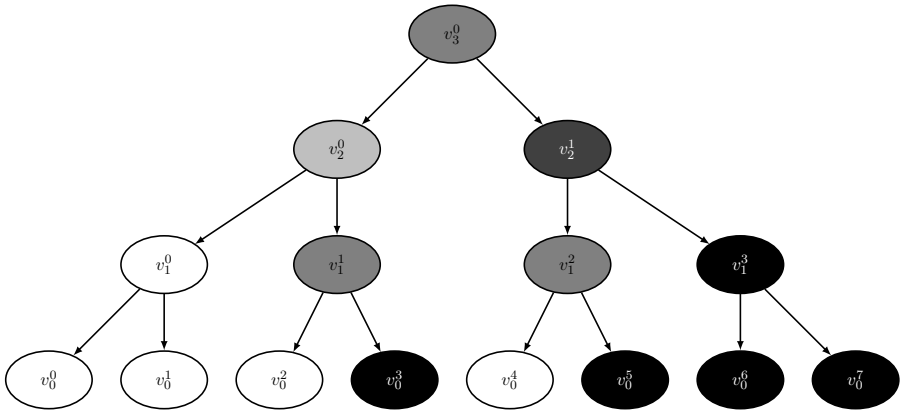


Figure 5.11: The tree structure from Figure 5.2 with rate information incorporated for a $N = 8, K = 4$ code. Information (Rate 1) nodes are colored in black and frozen (Rate 0) nodes in white. The shade of gray in the other nodes illustrate the rate of the node.

size of 2, denoted as a $R0_2$-subcode. The node which forms the top level of a subcode is named after the code, making v_1^0 a $R0_2$-node and so forth.

Especially $R0$-nodes save a lot of computation because the LLR that goes into them does not have to be calculated. The $R1$-nodes also saves operations, but fewer as the LLR values are required to make a hard decision on a bit. Additionally, the same transformation as used in the encoding must also be applied to the bits from an $R1$-node. A decoder implementing $R0$ and $R1$ subcodes is called the Simplified Successive Cancellation (SSC) decoder [52].

In Figure 5.12 the same tree as in Figure 5.11 is shown with trimmed $R0$ and $R1$ nodes. The edges in the graph now illustrate the F and G operations that remain. Note that the $R0$-node v_1^0 does not require LLRs and the F operations are therefore skipped. The original decoder for $N = 8$ required:

$$\text{ops}_{\text{polar 8 full}} = 8 \cdot \log_2(8) = 24 \,\text{operations}, \tag{5.6}$$

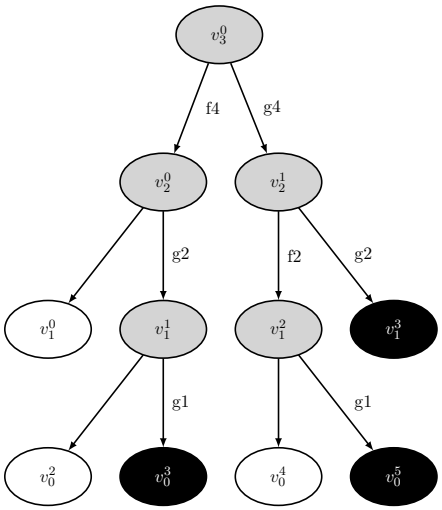


Figure 5.12: The trimmed version of Figure 5.11. The edge labels represent which operation is performed and how many are performed in parallel; e.g. 2 G operations are performed on the LLRs passed from v_2^0 to v_1^2.

which, if executed in parallel, can be completed in:

$$C_{8\ \text{full}} = 2 \cdot 8 - 2 = 14\,\text{cycles}. \tag{5.7}$$

The code rate does not alter this number, making low rate codes inefficient. The trimmed tree of the SSC decoder on the other hand, only requires 6 F operations, 10 G and 2 $R1$ operations for a total of:

$$\text{ops}_{\text{polar 8 ssc}} = 6 + 10 + 2 = 18\,\text{operations}, \tag{5.8}$$

for $K = 4$. If full parallelization is again used, the operations can be completed in

$$C_{8\ \text{ssc}} = 8\,\text{cycles}. \tag{5.9}$$

This almost half the latency of the full decoder.

The effect is even more pronounced with lower rate codes. For an $N_{\text{polar}} = 16, K = 3$ code, a fully parallel decoder without reduction requires 30 cycles while the reduced decoder can perform the computation in just 6. For the $N_{\text{polar}} = 2048, K = 400$ codes that are actually used in this project the full decoder takes at least 4094 cycles, while the reduced decoder can get away with just 472, a more than 8 fold reduction.

This is not the end of the road however. In [59] additional subcodes are introduced in order to reduce the number of total operations even further. In this work only two additional subcodes are included: repetition and Single Parity Check (SPC). Repetition codes are branches that only include a single information node, in the right-most place. An example is shown in Figure 5.13a for a repetition code of size $N_{\text{subcode}} = 4$. The code is named as such because the encoded result will either be all zeros, if the information node is zero, or all ones, if the node is one. The optimal decoding of this code consist of adding the LLRs and making a hard decision based on the sum.

The other subcode, SPC, is shown in Figure 5.13b. This subcode has all the included leafs be information nodes, except the left-most one. This subcode is named after the single bit that calculates parity for all the nodes. Indeed all the possible code words from this subcode have an even number of bits that are one, also known as even parity.

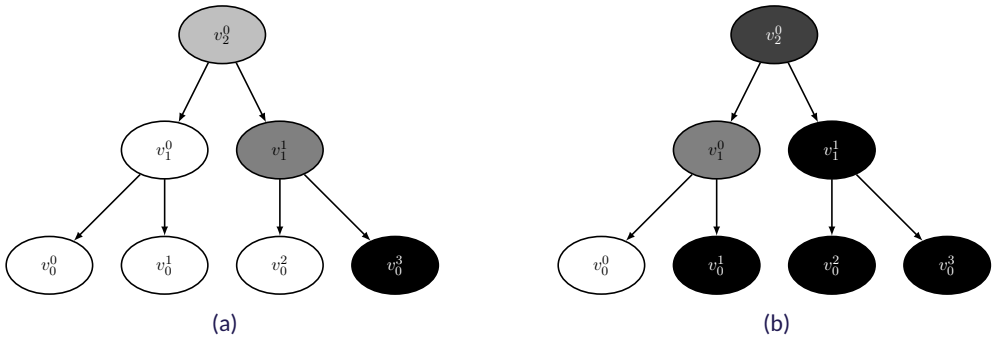


Figure 5.13: Types of subcode nodes size $N_{\text{subcode}} = 4$ introduced from [59]. (a) repetition node and (b) Single Parity Check node.

Decoding this code is slightly more complex than the repetition codes. First a hard decision is made for all the bits, and the parity is calculated. If the resulting parity is odd, the least significant bit is flipped.

These specific subcodes are chosen because they are fairly common in both high [59] and low rate polar codes [60]. For instance, the $N_{\text{polar}} = 8, K = 4$ shown in Figure 5.11 reduces to simply one REP and one SPC node. The resulting number of cycles required when using a decoding that implements the full set of subcodes is:

$$C_{2048\ \text{ssc}} = 354\,\text{cycles}. \tag{5.10}$$

In order to hit the 10 µs target, the required frequency is now just 35.4 MHz. It is likely that further reduction is possible if more subcodes are introduced, but this is sufficient for the purposes of this project.

5.3.2 OTHER CHANGES FOR HARDWARE

The reductions described so far have been useful for reducing the total number of computations. For a hardware design however, two more considerations are made; the F operation in the LLR domain is complex and the currently used floating point representation requires significant area compared to integer representations.

In Section 5.2.2 it is described how the algorithm is transformed into calculations in the LLR regime in order to improve numerical stability. While the operations are numerically stable, this move also introduced complex hyperbolic calculations in the F operation, repeated here:

$$F_{\text{LLR}}(\text{LLR}_a, \text{LLR}_b) = 2\tanh^{-1}\left(\tanh(\text{LLR}_a/2)\tanh(\text{LLR}_b/2)\right). \tag{5.11}$$

Performing this calculation requires relatively complex hardware, taking up area and clock cycles. Fortunately a well-known approximation exists. The approximation is known as *min-sum* and is commonly used in LDPC decoders without considerable error correcting performance loss [51]:

$$F_{\text{LLR}}(\text{LLR}_a, \text{LLR}_b) \approxeq \text{sign}(\text{LLR}_a)\,\text{sign}(\text{LLR}_a)\min(|\text{LLR}_a|, |\text{LLR}_b|). \tag{5.12}$$

This approximation is less complex to perform in hardware and [51] find that no significant loss of error correcting performance is experienced.

Table 5.2: Run time comparison of the effect of the complexity reductions in the polar decoder. The configuration used is $N_{\text{polar}} = 2048, K = 400$. The results are given as the mean run times over 10000 runs.

Alteration	*Run time*
None	29.58 ms
F approximation	23.28 ms
SSC + F approximation	3.71 ms

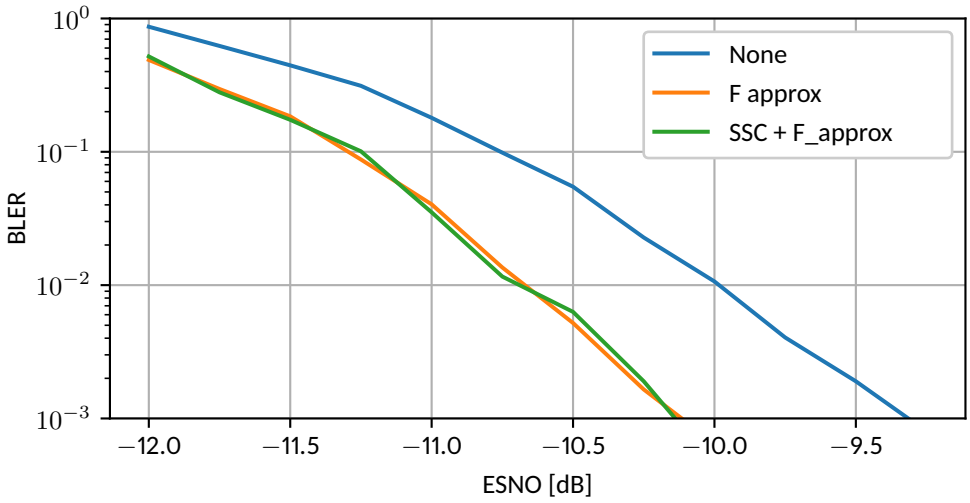


Figure 5.14: Comparison of BLER performance of polar codes with and without the SSC and F approximations.

With the F operation changed, the number representation can be considered. In the software implementation the LLR values passed around are represented as floating point numbers. This representation provides both good accuracy and large dynamic range and modern processors often contain complementary hardware to accelerate performance of floating point calculations. For FPGAs the scenario is different however. Not only are common floating point representations large in terms of bits, with 32 and 64 bit numbers being common, performing even simple mathematical operations such as addition is significantly more complex than with integer presentations. Fortunately in many cases when the system is known, the improved dynamic range and precision of floating point is unnecessary. Instead fixed point representation can be used. As opposed to floating point numbers, fixed point have a constant number of bits assigned to the two parts, integer and fractional. This representation has the same precision regardless of the currently held value. In the case of polar codes, [59] find that for high-rate codes quantization to 4 bits for the channel LLR and 6 bits for internal LLR representations with no fractional bits show performance within 0.1 dB of floating point within their operating range. Adding a single bit for fractions improved the performance to be almost identical with floating point. Through simulations [60] find the same to be the case for low-rate codes. This representation is therefore chosen. Finally, since the absolute value is used in many of the operations, it is chosen to represent the numbers using signed magnitude.

5.3.3 PERFORMANCE OF THE ALTERED DECODER

The alterations are implemented in the software decoder to test the performance.

The error correcting performance of the decoder is compared with and without the SSC reduction and approximation of the F function. The results are shown in Figure 5.14. The performance is seen to improve with the F approximation and not be affected by the SSC reduction.

The run time of the software decoders are shown in Table 5.2. The large reduction in operations from the SSC is clear, with an almost 8 times speed-up compared to the non-altered decoder.

To confirm the number of cycles found in Equation (5.10) a decoder should be implemented and tested on an FPGA. This project is focused on evaluating the full WIRT system, so making a full hardware design was not possible due to time constaints. A smaller $N_{\text{polar}} = 16, K = 3$ decoder is designed and presented in Appendix B to illustrate the design procedure.

This chapter has discussed Error Correction Codes. It is found that selection is not clear-cut, and all the surveyed codes have advantages in some respects. Polar codes are chosen as a focus for this project. The codes are described and a software implementation is made. As expected, it is found that a ECC decoder comes with a significant computational cost. A number of alterations are made to the basic decoder, reducing the complexity and enabling better hardware parallelization. It is found that the decoder can be adequetely reduced in latency to be useful in WIRT.

6 DISCUSSION

This chapter evaluates various decisions made throughout the project. Problems and research for future work are also presented.

The delay in Chapter 4 is estimated based on the number of computations. This is done in part to not confine the runtime to a single ECC implementation and in part because the runtimes on a x86 processor are not always indicative of the runtime on other computing architectures. However, to provide some confidence that the estimates are correct the system implementation is benchmarked. 1000 frames are encoded and decoded using the WIRT prototype in approximately 8.5 s. This number varies more than a second in consecutive trails, illustrating why benchmarking does not provide an appropriate estimate for complexity. Table 6.1 shows the largest contributors to processing time, given as a percentage of the overall execution time. The results align well with the estimates in Tables 4.1 and 4.2. The table is generated from profiling the code snippet shown in Code snippet 6.1.

The BLER performance of the ECC gives an idea of the overall reliability of the system, however it does not provide the full picture. As described in Chapter 2, if synchronization or channel estimation fails the performance is degraded. For this project especially the equalization proves to be a bottleneck. In Figure 6.1 the BLER performance of the full WIRT prototype is shown in an AWGN channel. When the equalization is enabled the noise is amplified so much that the system performs significantly worse.

Equalization can not just be permanently disabled however. In more realistic channel scenarios an estimate of at least the phase is required, as even a simple effect such as a constant phase offset disables the symbol mapping. It is clear that the equalization method used here is not appropriate for low SNR channels, and more advanced methods should be used.

In Chapter 5 it is noted that the original SC polar decoder does not achieve state-of-the-art performance. Alternative implementations achieve this however, with the most common one being SCL. In contrast to SC, SCL does not find the most likely bit separately, but instead the most likely paths. To keep the complexity down only the L most likely paths are kept at every stage. In the end the correct codeword is

Table 6.1: The parts of the software implementation that correspond to over 1% of the total runtime when encoding and decoding a package. 1000 WIRT packages are encoded and decoded for a total runtime of approximately 8.5 s. Notice the IFFT is included in OFDM modulation and the FFT and channel estimation are contained in the OFDM demodulation. Python package loading is not included in the table, but constitute approximately 10 % of the full runtime.

Function	*Description*	*% of runtime*
Polar decode	Polar code decoding	61.2 %
Upsampling	Upsampling of the encoded samples	8.89 %
Downsampling	Downsampling from the RF sample rate	5.68 %
OFDM demodulation	Full OFDM demodulation	5.15 %
OFDM modulation	Full OFDM modulation	2.94 %
Channel estimation	Channel estimation, primarily interpolation	1.25 %
IFFT (total)	IFFT operations	1.23 %
FFT (total)	FFT operations	1.18 %

```
from WIRT import wirt

num_runs = 1000
encoder = wirt()

data = np.random.binomial(1, 0.5, (num_runs, wirt.DATA_SIZE)).astype(np.uint8)
for i in range(num_runs):
    encoded = encoder.encode(data[i])
    decoded = encoder.decode(encoded, 20)
```

Code snippet 6.1: Code used to measure the time it takes for the prototype to encode and decode 1000 WIRT package on a modern x86 machine.

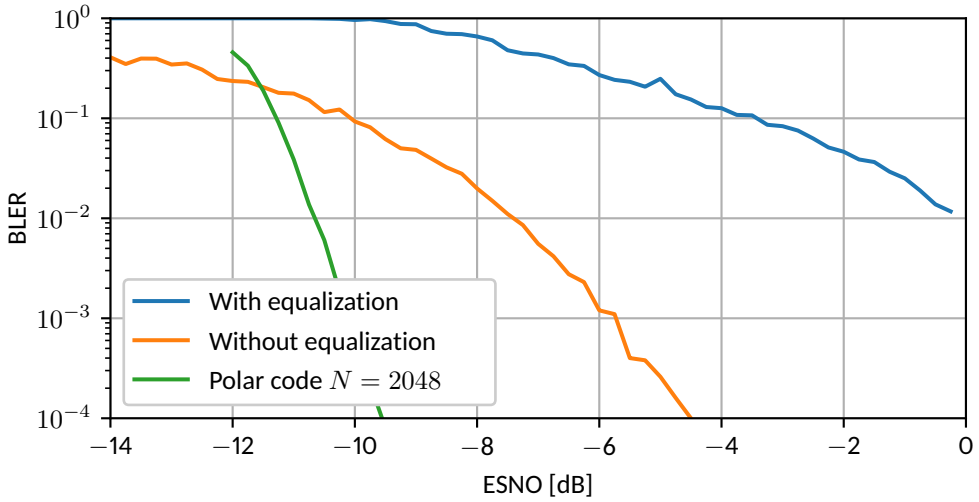


Figure 6.1: Comparison of the overall system performance in an AWGN channel with and without equalization enabled. It is clear that at low signal to noise ratios gain from the equalization procedure affects the noise, greatly decreasing system performance.

identified using a Cyclic Redundancy Check (CRC) among the L candidates [49]. With the repetition codes used in this project adding a CRC would also enable the decoder to stop early in good SNR conditions. This approach is compatible with the SSC reductions in this report and improves the performance, at the cost of higher complexity both in terms of computations performed and implementation difficulty [61, 62]. Research is still active in this area and on other algorithms [63, 64].

This project has largely dealt with the latency requirements while fulfilling the WIRT specification. The performance results in Figure 6.1 show that the configuration increases in reliability as the SNR increases. The results can therefore be used in later work to estimate the approximate SNR required for the reliability targets.

The channel model used for evaluation during the ECC is a simple AWGN channel, that fails to describe many of the effects of a real world wireless channel, including multipath fading and interference. More advanced models would help give more credence to the system, but in the end what matters are the real, over the air results. One of the goals for this project is to create a testbed for future work on WIRT by integrating with a LitePoint tester. This integration is performed, but the system is only tested with a wired connection from the VSG to the VSA. Considerations on the dynamic range of the tester and appropriate measurement equipment are outside the scope of this project,

and so over the air tests are yet to be performed at the time of writing. This is a clear path for future work.

The latency considerations for the reconfigurable logic have been centered around the latency of performing the computations themselves. Another important aspect is memory latency. To get a sense of the effect, consider the specifications of the 64 MB SDRAM chip IS42R16320D [65] which is used on the DE10-Standard FPGA development board [66]. This chip is a few years old, and placed in a cost-optimized segment of the market. For this comparison assume that the SDRAM is already initialized, as that is done at start-up, and that the data is loaded sequentially into memory, enabling a full page burst read. The delay from when the chip is inactive until the first bits are ready on the output bus is up to 6 cycles, at a frequency of 100 MHz for the lowest speed grade unit. When the data transmission is started however, new data is available at every clock cycle making the overall latency for transferring 14400 LLR values:

$$L_{\text{memory}} = \frac{(14440 \cdot 4 \text{ bit})/(16 \text{ bit per transfer})}{100\,\text{MHz}} = 36\,\mu\text{s}. \tag{6.1}$$

Since the clock for the SDRAM might not be synchronized with the system clock, a clock-crossing FIFO can be synthesized, adding a few more clock cycles to the latency.

If the LLRs to be loaded are kept in external memory, the load time is more than three times the delay in the decoding itself, and a system based on this decoder design should account for it. Keeping the decoder fed with data is outside of the scope of this project, but an important consideration for implementations. One potential solution to this problem is to simply load the LLRs into the on-chip memory of the FPGA while receiving. The FPGA used on the same DE10-Standard development board contains 5761 kbit of embedded memory for user designs, more than sufficient to hold all the codewords of a transmission at $14400 \cdot 4\,\text{bit} = 56.25\,\text{kbit}$.

7 CONCLUSION

Wireless communication acts as an enabler for the next generation of industrial automation. Next-generation networks introduce low latency, high reliability services, but for the purposes of critical industrial processes the delay and reliability requirements are even harsher. WIRT is a proposal to solve this problem, while also providing on requested features such as periodic transmissions.

WIRT is still in the research stage, and a number of questions are still open. The objective of this thesis presented in Section 1.2, is the answer the question: *What are the requirements for realization of a WIRT system, from a signal processing point of view?*

The answer to this question is found through a number of related questions. In Chapter 2 the basic components of a multiband OFDM system are presented. The specifications for WIRT are presented in the form of a parameterized multiband OFDM system. In Chapter 3 a software implementation of the WIRT transceiver is presented, describing specific details found during implementation. This implementation is made as a prototype for further WIRT development and integrates with an UWB tester from LitePoint.

The next related question is on the system latency. Chapter 4 first estimates the overall minimum latency, when computational considerations are not included. The approximate scale of the computational complexity is estimated to find bottlenecks in the design. It is found that resampling, FFT and ECC are the major contributors to latency. Since the ECC decoding is a well known contributor to complexity of communication systems, it is researched further. This chapter also considers computational architectures. It is found that in order to provide guarantees that the latency target can be met, a reconfigurable logic platform, such as an FPGA, is most appropriate.

Finally, Chapter 5 delves deeper into which ECCs are approximate for a low delay, high reliability system. While no clear winner is found, polar codes are a good contender. The delay added by the decoder is evaluated based on a software implementation, and indeed found to have a major effect on the overall system delay. In order to justify reaching the latency targets a number of alterations are made to the codes to reduce the number of calculations and simplify the remaining.

This thesis considers WIRT through the lens of signal processing; is it feasible to achieve the latency goals and how can it be done in implementation. The feasibility is evaluated through both considerations on the used algorithms and an implementation. The report presents the requirements found and considerations made, to provide a recommendation for future WIRT work.

BIBLIOGRAPHY

[1] Alexander Korsvang Hagelskjær, "Wirt implementation (github)," Website, 2019. [Online]. Available: https://github.com/Akorsvang/WIRT-implementation

[2] B. Galloway and G. P. Hancke, "Introduction to industrial control networks," *IEEE Communications Surveys Tutorials*, vol. 15, no. 2, pp. 860–880, Second 2013.

[3] G. C. Walsh and Hong Ye, "Scheduling of networked control systems," *IEEE Control Systems Magazine*, vol. 21, no. 1, pp. 57–65, Feb 2001.

[4] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 138–162, Jan 2007.

[5] I. Parvez, A. Rahmati, I. Guvenc, A. I. Sarwat, and H. Dai, "A survey on low latency towards 5g: Ran, core network and caching solutions," *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, pp. 3098–3130, Fourthquarter 2018.

[6] P. Popovski, J. J. Nielsen, C. Stefanovic, E. de Carvalho, E. G. Ström, K. F. Trillingsgaard, A. Bana, D. Kim, R. Kotaba, J. Park, and R. B. Sørensen, "Ultra-reliable low-latency communication (URLLC): principles and building blocks," *CoRR*, vol. abs/1708.07862, 2017.

[7] F. Khan, *LTE for 4G Mobile Broadband: Air Interface Technologies and Performance*. Cambridge University Press, 2009.

[8] P. Schulz, M. Matthe, H. Klessig, M. Simsek, G. Fettweis, J. Ansari, S. A. Ashraf, B. Almeroth, J. Voigt, I. Riedel, A. Puschmann, A. Mitschele-Thiel, M. Muller, T. Elste, and M. Windisch, "Latency critical iot applications in 5g: Perspective on the design of radio interface and network architecture," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 70–78, February 2017.

[9] P. Marsch, O. Bulakci, O. Queseth, and M. Boldi, *5G System Design: Architectural and Functional Considerations and Long Term Research*. Wiley, 2018.

[10] 3GPP, "Study on scenarios and requirements for next generation access technologies," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 38.913, 08 2017, version 14.3.0.

[11] G. Berardinelli, N. H. Mahmood, I. R. Larrad, and P. E. Mogensen, "Beyond 5g wireless irt for industry 4.0: Design principles and spectrum aspects," in *Ieee Globecom Workshops (gc Wkshps)*, 2018.

[12] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, July 1948.

[13] W. Ryan and S. Lin, *Channel Codes: Classical and Modern*. Cambridge University Press, 2009.

[14] S. Haykin, *Communication Systems, 4th Ed.* Wiley India Pvt. Limited, 2006.

[15] A. Goldsmith, *Wireless communications*. New York, NY, USA: Cambridge University Press, 2005.

[16] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing (3rd Edition) (Prentice-Hall Signal Processing Series)*. Pearson, 2009.

[17] T. M. Schmidl and D. C. Cox, "Robust frequency and timing synchronization for ofdm," *IEEE Transactions on Communications*, vol. 45, no. 12, pp. 1613–1621, Dec 1997.

[18] J. J. van de Beek, M. Sandell, and P. O. Borjesson, "Ml estimation of time and frequency offset in ofdm systems," *IEEE Transactions on Signal Processing*, vol. 45, no. 7, pp. 1800–1805, July 1997.

[19] A. Molisch, *Wireless Communications*, ser. Wiley - IEEE. Wiley, 2012.

[20] 3GPP, "Study on Communication for Automation in Vertical domains (CAV)," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 22.804, 12 2018, version 16.2.0.

[21] LitePoint, "IQgig-UWB Technical Specifications," Online, Technical Specifications, 2019.

[22] M. Faulkner, "The effect of filtering on the performance of ofdm systems," *IEEE Transactions on Vehicular Technology*, vol. 49, no. 5, pp. 1877–1884, Sep. 2000.

[23] Michael D. Zoltowski, "Ece 538 digital signal processing i," Website, 2018. [Online]. Available: https://engineering.purdue.edu/~ee538/

[24] R. Lyons, *Understanding Digital Signal Processing*. Prentice Hall, 2011.

[25] J. Zyren and W. McCoy, "Overview of the 3gpp long term evolution physical layer," *Freescale Semiconductor, Inc., white paper*, vol. 7, pp. 2–22, 2007.

[26] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[27] M. Frigo and S. G. Johnson, "The design and implementation of fftw3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, Feb 2005.

[28] D. Chu, "Polyphase codes with good periodic correlation properties (corresp.)," *IEEE Transactions on Information Theory*, vol. 18, no. 4, pp. 531–532, July 1972.

[29] H. Sari, G. Karam, and I. Jeanclaude, "Transmission techniques for digital terrestrial tv broadcasting," *IEEE Communications Magazine*, vol. 33, no. 2, pp. 100–109, Feb 1995.

[30] S. Lin and D. J. Costello, *Error Control Coding*, 2nd ed. Upper Saddle River, NJ 07458: Pearson Education, Inc., 2004.

[31] R. E. Crochiere and L. R. Rabiner, "Interpolation and decimation of digital signals—a tutorial review," *Proceedings of the IEEE*, vol. 69, no. 3, pp. 300–331, March 1981.

[32] F. Harris, *Multirate Signal Processing for Communication Systems*. Prentice Hall PTR, 2004.

[33] The SciPy community, “Discrete fourier transform (numpy.fft),” Website, 2019. [Online]. Available: https://docs.scipy.org/doc/numpy/reference/routines.fft.html

[34] D. Lustig and M. Martonosi, “Reducing gpu offload latency via fine-grained cpu-gpu synchronization,” in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2013, pp. 354–365.

[35] Nvidia, “Cuda c programming guide,” Website, 2019. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[36] J. Greg Nash, “High-throughput programmable systolic array fft architecture and fpga implementations,” in *2014 International Conference on Computing, Networking and Communications (ICNC)*, Feb 2014, pp. 878–884.

[37] 3GPP, “NR; Multiplexing and channel coding,” 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.212, 01 2018, version 15.0.0.

[38] AccelerComm Ltd, “Enhanced turbo codes for URLLC,” Online, 3rd Generation Partnership Project (3GPP), Technical Documentation (TDoc) written contributions R1-1702856, 2017.

[39] “Ieee standard for information technology– local and metropolitan area networks– specific requirements– part 11: Wireless lan medium access control (mac)and physical layer (phy) specifications amendment 5: Enhancements for higher throughput,” *IEEE Std 802.11n-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009)*, pp. 1–565, Oct 2009.

[40] E. Arikan, “Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,” *arXiv e-prints*, p. arXiv:0807.3917, Jul 2008.

[41] Huawei, HiSilicon, “Channel coding for URLLC,” Online, 3rd Generation Partnership Project (3GPP), Technical Documentation (TDoc) written contributions R1-1718374, 2017.

[42] ——, “Channel coding for URLLC,” Online, 3rd Generation Partnership Project (3GPP), Technical Documentation (TDoc) written contributions R1-1801467, 2018.

[43] Tsofun Algorithm, “Channel coding for URLLC,” Online, 3rd Generation Partnership Project (3GPP), Technical Documentation (TDoc) written contributions R1-1718505, 2017.

[44] Samsung, “Discussion on Channel Coding for URLLC Channel,” Online, 3rd Generation Partnership Project (3GPP), Technical Documentation (TDoc) written contributions R1-1700981, 2017.

[45] AccelerComm Ltd, “On the hardware implementation of channel decoders for short block lengths ,” Online, 3rd Generation Partnership Project (3GPP), Technical Documentation (TDoc) written contributions R1-1612306, 2016.

[46] MediaTek Inc., "Examination of NR Coding Candidates for Low-Rate Applications," Online, 3rd Generation Partnership Project (3GPP), Technical Documentation (TDoc) written contributions R1-167533, 2016.

[47] A. Balatsoukas-Stimming, P. Giard, and A. Burg, "Comparison of polar decoders with existing low-density parity-check and turbo decoders," 2017.

[48] P. Giard, A. Balatsoukas-Stimming, T. C. Müller, A. Burg, C. Thibeault, and W. J. Gross, "A multi-gbps unrolled hardware list decoder for a systematic polar code," 2017.

[49] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, May 2015.

[50] Y. Zhou, R. Li, H. Zhang, H. Luo, and J. Wang, "Polarization weight family methods for polar code construction," *CoRR*, vol. abs/1805.02813, 2018.

[51] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, "Hardware architectures for successive cancellation decoding of polar codes," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2011, pp. 1665–1668.

[52] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Communications Letters*, vol. 15, no. 12, pp. 1378–1380, December 2011.

[53] I. Tal and A. Vardy, "List decoding of polar codes," in *2011 IEEE International Symposium on Information Theory Proceedings*, July 2011, pp. 1–5.

[54] Harish Vangala, "Polar codes," Website, 2019. [Online]. Available: http://www.polarcodes.info/

[55] S. A. Khan, *Digital Design of Signal Processing Systems: A Practical Approach*. Wiley, 2011.

[56] F. Vahid, *Digital design, with RTL design, VHDL, and Verilog*. Hoboken, NJ: Wiley, 2011.

[57] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *IEEE Transactions on Signal Processing*, vol. 61, no. 2, pp. 289–299, Jan 2013.

[58] A. Pamuk and E. Arıkan, "A two phase successive cancellation decoder architecture for polar codes," in *2013 IEEE International Symposium on Information Theory*, July 2013, pp. 957–961.

[59] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 946–957, May 2014.

[60] P. Giard, A. Balatsoukas-Stimming, G. Sarkis, C. Thibeault, and W. J. Gross, "Fast low-complexity decoders for low-rate polar codes," *CoRR*, vol. abs/1603.05273, 2016.

[61] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "Llr-based successive cancellation list decoding of polar codes," *CoRR*, vol. abs/1401.3753, 2014.

[62] P. Giard, G. Sarkis, C. Leroux, C. Thibeault, and W. J. Gross, "Low-latency software polar decoders," *CoRR*, vol. abs/1504.00353, 2015.

[63] S. A. Hashemi, M. Mondelli, S. H. Hassani, C. Condo, R. L. Urbanke, and W. J. Gross, "Decoder partitioning: Towards practical list decoding of polar codes," *IEEE Transactions on Communications*, vol. 66, no. 9, pp. 3749–3759, Sep. 2018.

[64] F. Ercan, C. Condo, and W. J. Gross, "Improved bit-flipping algorithm for successive cancellation decoding of polar codes," *IEEE Transactions on Communications*, vol. 67, no. 1, pp. 61–72, Jan 2019.

[65] I. Integrated Silicon Solution, *IS42R16320D Datasheet*, 2011.

[66] Terasic Inc, "De10-standard specifications," Website, 2019. [Online]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=1081&PartNo=2

Appendices

A) BENCHMARKING OF FFT SIZES

To help the choice of FFT, the options in Table 3.2 are compared. Additionally, keeping the N_{FFT} size with the original 524 subcarriers and the power-of-two 1024 are tested. The code used for benchmarking is shown in Code snippet A.1.

```
import numpy as np
import pyfftw
pyfftw.interfaces.cache.enable()

for N in [524, 625, 800, 1000, 1024, 1250, 2000, 2500]:
    print(N)

    a = np.random.rand(N) + 1j * np.random.rand(N)
    b = pyfftw.empty_aligned(N, dtype='complex128')
    b[:] = a.copy()

    %timeit -r20 -n100000 np.fft.fft(a)
    %timeit -r20 -n100000 np.fft.fft(b)
    %timeit -r20 -n100000 pyfftw.interfaces.numpy_fft.fft(a)
    %timeit -r20 -n100000 pyfftw.interfaces.numpy_fft.fft(b)
```

Code snippet A.1: IPython snippet used to estimate execution time of the FFT using numpy and fftw respectively.

Table A.1: Timing results of calculating the FFT for the different N_{FFT} options given in Table 3.2. Run on my laptop, using ipython built-in magic command "%timeit". Given as mean ± std. dev. of 20 runs, 100000 loops each. Array a is allocated as a numpy array and b is allocated aligned using pyfftw.

N_{FFT}	*np.fft.fft(a)*	*np.fft.fft(b)*	*pyfftw fft(a)*	*pyfftw fft(b)*
524	66.7 µs ± 153 ns	66.8 µs ± 432 ns	54 µs ± 103 ns	54 µs ± 107 ns
625	15.8 µs ± 26.1 ns	15.7 µs ± 37.4 ns	37.6 µs ± 63.8 ns	37.8 µs ± 58 ns
800	18.9 µs ± 52.6 ns	18.9 µs ± 18 ns	37.3 µs ± 54.7 ns	37.5 µs ± 238 ns
1000	22.7 µs ± 56.3 ns	22.9 µs ± 282 ns	39.2 µs ± 108 ns	39 µs ± 300 ns
1024	22.1 µs ± 24.3 ns	22 µs ± 29.4 ns	36.4 µs ± 56.6 ns	36.4 µs ± 55.2 ns
1250	27.7 µs ± 25.5 ns	27.8 µs ± 46.3 ns	40.3 µs ± 62.4 ns	40.3 µs ± 66.6 ns
2000	38.2 µs ± 58.9 ns	38.3 µs ± 92.5 ns	43.2 µs ± 148 ns	43.1 µs ± 144 ns
2500	47.5 µs ± 64 ns	47.5 µs ± 107 ns	45.8 µs ± 151 ns	45.7 µs ± 125 ns

B) HARDWARE DESIGN OF A POLAR DECODER

As described in Chapter 5 the decoding procedure is tested in software to confirm the calculations are correct. In order to achive the required latency, an implementation should be done in hardware. To illustrate this design procedure a prototype FPGA implementation is described. To keep the description simple, the decoder is limited in size to $N_{\text{polar}} = 16, K = 3$. This is approximately the same rate as for a $N_{\text{polar}} = 2048$ decoder. Since the control structure of the project is static the prototype is based on a Finite State Machine with Data path (FSMD) [55, 56]. An FSMD is a design structure where the hardware elements of that performs computations data are split from the control structure. These two parts known as the data path and the control path work in tandem to perform computations and the structure acts as a scaffolding when mapping algorithms. A diagram of a FSMD is shown in Figure B.1. The data path contains the hardware elements that perform computations on data, e.g. adders or multipliers, or more specialised structures for specific operations. The control path controls the hardware blocks in the data path to produce the correct series of calculations.

The hardware design is performed in three steps; algorithmic description, data path design and control path design. The first is algorithmic description. The implementation and alterations above describe which computations are to be performed and in what order to produce the desired results. Afterwards the data path is designed by specifying each of the operations in terms of the building blocks available for the targeted hardware. When working with FPGAs the building blocks differ between manufacturers, but common structures are adders, multipliers, registers and look-up tables. The look-up tables provide straightforward implementations for logic gates and small constants. Block diagrams of hardware blocks that can perform the F, G, $R1$ and $C2$ operations are shown in Figures C.1 to C.4. The figure also show suggestions for blocks that can decode SPC and repetition subcodes of size $N_{\text{subcode}} = 4$. The data path is connected to working memory. Since the computations are well specified the exact amount of memory is known well enought that the registers can not only be allocated up front, but also shared. In the context of registers, sharing refers to using the same register for the the

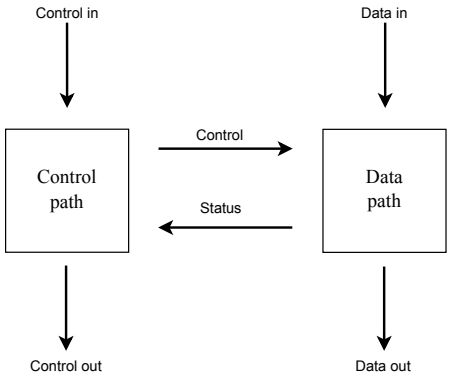


Figure B.1: Diagram of an Finite State Machine with Data path structure. The data path consists of the hardware elements that perform computations, e.g. registers, adders and multipliers. The control path is a Finite State Machine that controls the operations in the data path. The FSMD is a structure that algorithms can be mapped onto.

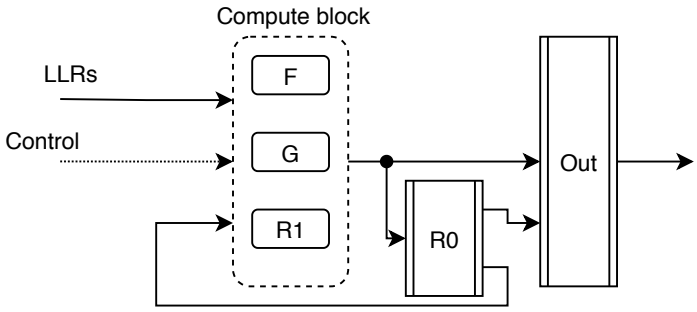


Figure B.2: Data path for hardware design. The design consists of computational blocks on the left and memory registers on the right.

Table B.1: Abstract control path of a $N_{\text{polar}} = 16, K = 4$ polar SSC decoder. The arrows indicate connections from either registers or the input LLRs to the computational blocks and into which register locations. The G block u inputs are zero except state 5 where bits 12 and 13 are used. This is an abstract description shown for understanding, a full control path description would specify which wires are asserted in order to achieve these configurations.

State index	*Input*	*Operation*	*Output*
1	Inputs	G	R0
2	R0[0:7]	G	R0[0:3]
3	R0[0:3]	F	R0[4:5]
4	R0[4:5]	G	R0[4]
4	R0[4]	< 0	B[13]
5	R0[0:3] + B[12:13]	G	R0[0:1]
6	R0[0:1]	$R1_2$	B[14:15]

result of different computations that are not occuring at the same time, in order to save hardware. The data path required for this project is shonw in Figure B.2. In the figure R0 refers to the local working memory, but not neccessarily one register. The register marked *Out* holds the temporary output bits while the calculation progresses. After decoding the output is now the full received codeword and the information bits can be selected with a static mask.

The next step is design of the control path. A Finite State Machine (FSM) is a collection of states and two tables, one that finds the next state of the FSM based on the current state and the input and one that maps a state to a collection of outputs [56]. In the case of the small prototype decoder, the next state is simply the sequentially state after the current one. The mapping from state to output is described in Table B.1. The mappings shown in the table are abstracted to give an overview.

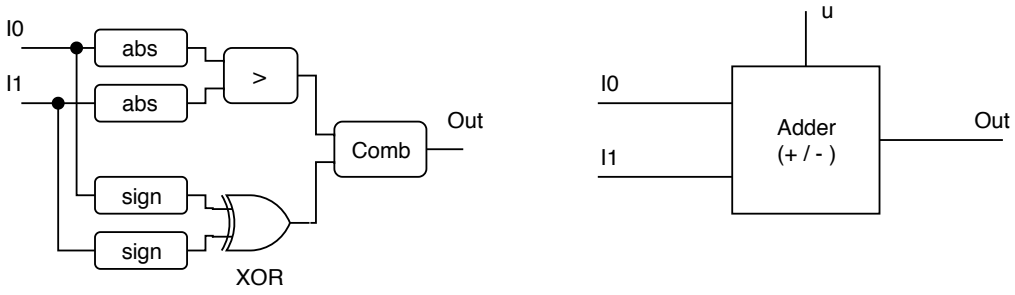
Between the design of the data path and the control path various optimizations, such as retiming, pipelining, operator sharing and unfolding can be performed to manipulate the trade-off between latency, throughput and area [55, Ch. 7]. After the optimization is done, the data and control paths are recreated and the process repeated. This iterative process stops when the design fulfills the requirements.

Based on the data path and control path the design can now be implemented. If an FPGA platform is used, the next step is simulation and verification. Verification that the design performs the correct calculations is done through logic simulation. After the design is logically valid, a timing simulation would show if the design is able to meet

the frequency requirements. Finally, if the design passes the verifications, it can be transfered to FPGA hardware.

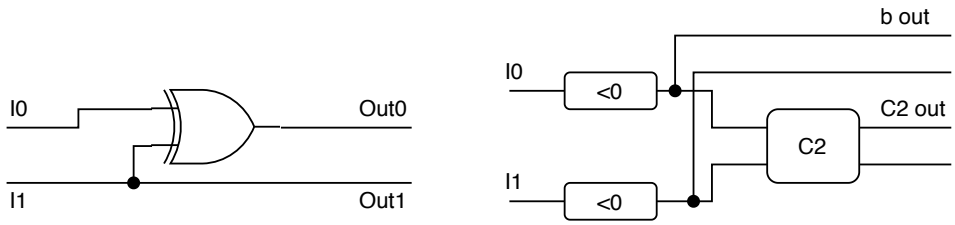
This chapter has covered a choice of ECC for use in a WIRT system. Section 5.1 covers options for modern ECC, ending in a choice of polar codes. The basics of these codes is covered in Section 5.2. Finally, Section 5.3 describes some alterations for the codes that make them more suitable for implementation in hardware, and an example of a small decoder. Using the alterations described in this chapter it is found that it is feasible to reach the latency targets of the project using an FPGA based design.

C) HARDWARE BLOCK DESIGNS



(a) (b)

Figure C.1: The hardware blocks for (a) F_2 and (b) G_2 operations. These blocks can be extended by parallelization.



(a) (b)

Figure C.2: The hardware blocks for (a) C_2 and (b) $R1_2$ operations. These blocks are not able to be parallelized to form larger blocks. The C operation represents an encoder for polar codes.

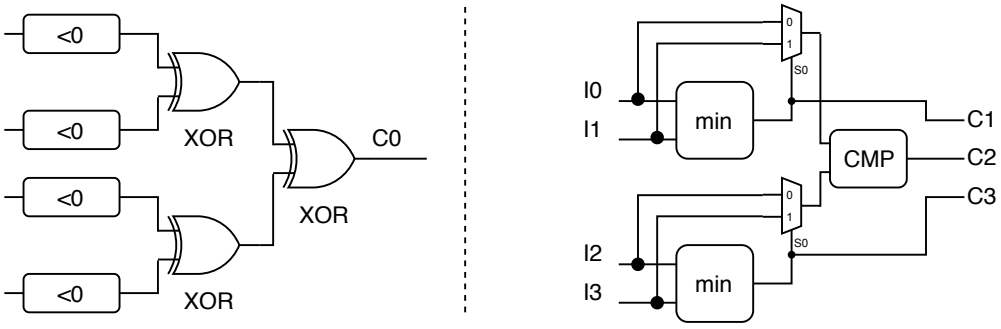


Figure C.3: Two of the three parts of the block that decodes an SPC subcode. The final part is seen in Figure C.4a.

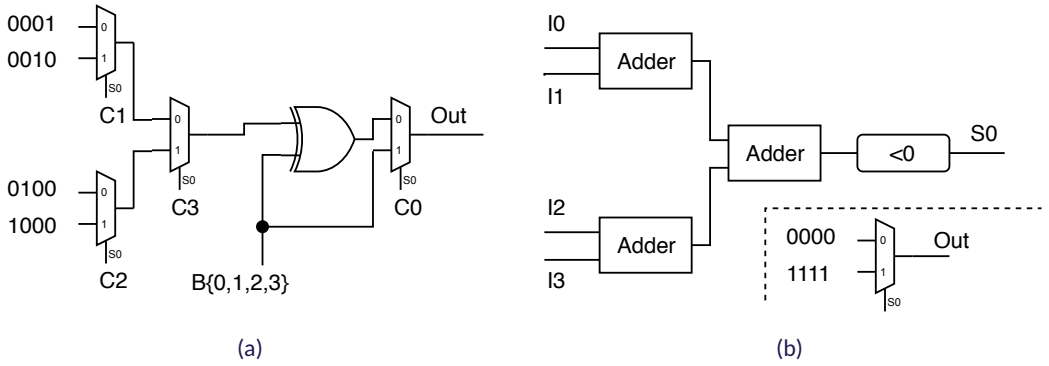


Figure C.4: (a) The final part of the hardware block that decodes the SPC subcode and (b) the circuit for decoding a repetition subcode. The first two parts of (a) are seen in Figure C.3. These blocks are not able to be extended in size by parallelization.

D) CODE OVERVIEW

The code for the implementation is available online at [1]. The implementation is split into seven folders:

1. Instrument control
2. Misc
3. OFDM
4. Other
5. Polar
6. Tests
7. WIRT

"OFDM", "WIRT" and "Polar" correspond to the appropriate modules in Figure 3.1. "Instrument control" contains the socket utilities and the code for interfacing with the LitePoint tester. The "Misc" directory contains the submodules marked as such in Figure 3.1 and all the tests are collected in the "Tests" directory. Finally, the "Other" directory contains minor supplementary code snippets created during development.

In order to execute the code, the root folder must be included in the *PYTHONPATH*, to enable Python to find the submodules. Python version 3.5 and above should work. Additionally the code requires a relatively new version of the Numpy and Scipy libraries.