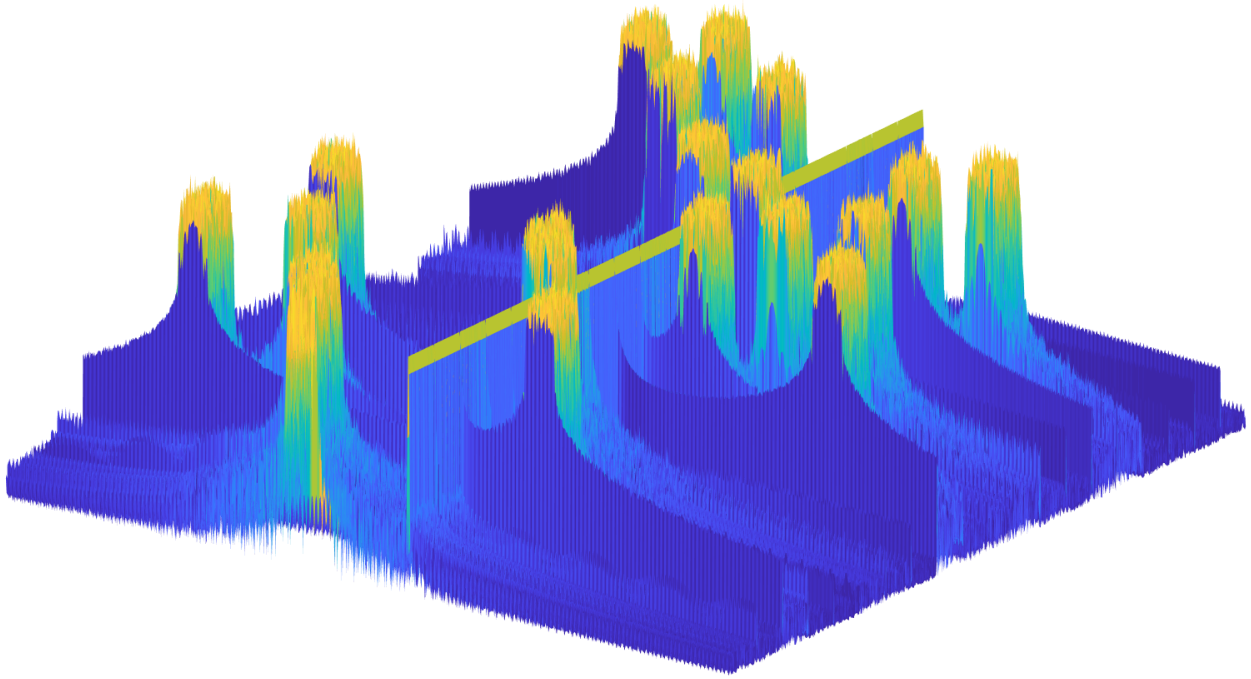
CHALMERS
UNIVERSITY OF TECHNOLOGY



Design and Evaluation of an Interference-resistant Wireless Link

using Frequency-Hopping Spread-Spectrum

Master's thesis in Communication Engineering

SEBASTIAN SAHLIN and RICKARD LARSSON

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019

Design and Evaluation of an Interference-resistant Wireless Link

using Frequency-Hopping Spread-Spectrum

SEBASTIAN SAHLIN and RICKARD LARSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2019

Design and Evaluation of an Interference-resistant Wireless Link
using Frequency-Hopping Spread-Spectrum

Supervisor: Mohammad Hossein Moghaddam, Department of Electrical Engineering
Examiner: Erik Ström, Department of Electrical Engineering

Master's Thesis 2019
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: PSD of a jammer interfering with an FHSS-system

Typeset in LaTeX
Gothenburg, Sweden 2019

Design and Evaluation of an Interference-resistant Wireless Link
using Frequency-Hopping Spread-Spectrum
Department of Electrical Engineeringy
Chalmers University of Technology

Abstract

Frequency-Hopping Spread-Spectrum (FHSS) is a common strategy for implementing interference- and jamming-resistant wireless links. In this work a proposal for a FHSS link has been evaluated by means of simulation in MATLAB. In addition, the possibility of implementing some aspects of the system on a Software Defined Radio (SDR) platform, consisting of the GNU Radio toolbox and USRP B200 software radio, has been studied. Results show that while the proposed FHSS link improves BER performance by 8 dB over a single-carrier link in a jamming scenario, the overall anti-jamming capability is still unsatisfactory. In addition, more work is needed to properly verify the capabilities of the SDR platform.

Keywords: FHSS, DPSK, SDR, USRP, GNURadio, Anti-jam, Jamming, TDL, Pseudorandom.

Acknowledgements

We would like to thank our examiner Erik Ström, and supervisor at Chalmers Mohammad Hossein Moghaddam, for providing valuable feedback and support over the span of this thesis project. Furthermore we'd like to thank SAAB, in particular our supervisors Morgan Andersson and Niclas Ylvén, for making us feel welcome and for providing us with the opportunity to apply our skills and deepen our knowledge in the interesting subject that is Spread Spectrum technology.

Abbreviations

API	Application Programming Interface
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
PSK	Phase Shift Keying
DPSK	Differential Phase Shift Keying
DSP	Digital Signal Processing
DSSS	Direct Sequence Spread Spectrum
FFH	Fast Frequency Hopping
FHSS	Frequency-Hopping Spread Spectrum
LOS	Line-of-sight
MPC	Multipath Component
SFH	Slow Frequency Hopping
TDL	Tactical Data Link
PSD	Power Spectral Density
SS	Spread spectrum
SNR	Signal-to-Noise ratio
SIR	Signal-to-Interference ratio
SDR	Software Defined Radio
RS	Reed Solomon
RF	Radio Frequency
LPI	Low Probability of Intercept
TADIL	Tactical Digital Information Links
WLAN	Wireless Local Area Network

Contents

List of Figures

List of Tables

1
Introduction

A military conflict is a hostile environment for people and technology alike. Because wireless communication has ubiquitous tactical uses on a battlefield, such as enabling coordination between military platforms on land, at sea and in the air, an adversary will often attempt to eavesdrop on or disable these communications. In the latter case, different strategies are employed for causing deliberate interference (known as *jamming*) on frequencies of interest. Therefore, in turn, communication links operating in these conditions must employ strategies to circumvent or resist this jamming (i.e., employ *anti-jamming*).

1.1 Background

Spread spectrum (SS) technology has long been a common building block in wireless communication links wishing to overcome interference, whether intentional or not. It is the act of deliberately using (or spreading) far more bandwidth (or spectrum) than what is physically dictated to achieve reliable communications. Spread spectrum is typically divided into two overarching types: Direct-Sequence Spread-Spectrum (DSSS) systems and Frequency-Hopping Spread-Spectrum (FHSS) systems, the latter famously patented during the Second World War by actress Hedy Lamarr and composer George Antheil using piano music rolls as the basis for a hopping sequence [1]. Ever since, FHSS has generally been affiliated with the military domain, although in later years some commercial link standards such as Bluetooth has also adopted FHSS as a way to combat interference, stemming from its use of a shared and crowded spectrum resource on 2.4 GHz. FHSS is also used in some multiple-access schemes.

Today, a number of standards for military communication links (referred to as TDL or TADIL, Tactical Data Links/Tactical Digital Information Links) are in use throughout the world, most of which employ some kind of anti-jamming strategy [2]. However, detailed specifications on the physical-, link- and network layers etc of these links are not publicly available.

SAAB AB has developed a proposal for the physical layer of a TDL employing FHSS as the main interference mitigation technique, along with specifications for frame structure, channel coding and more[1]. The proposal is the basis for this thesis work.

[1]Details on the specification are presented in Section 4.1.

1.2 Aims

The aim of this thesis work is to investigate the feasibility and performance of the proposed communication link under deliberate interference conditions. This will mainly be done by means of simulation, however an important future goal for SAAB is to have a working (though likely scaled-down) implementation on an SDR (Software Defined Radio) platform. What this means in practise is that the simulation chain should be designed with certain hardware constraints in mind. There are three aims:

- Design and simulate the data link based on the given specifications
- Evaluate the performance of the simulation under deliberate interference conditions
- Determine the feasibility of implementing the link on an SDR platform

1.3 Scope

The focus of the project is on the performance characteristics of the overall link (such as bit error rate, resistance to jamming etc) as specified, in the simulation. It does not concern detailed investigation into any particular components of the link, such as channel coding, modulation or other related parameters, other than what is required to meet the intended specifications.
While a rudimentary link protocol will be implemented to fulfil the frame structure and timing specification, considerations outside the physical layer of the link is not within the scope of this project.

1.4 Methodology

The project started by researching various techniques and algorithms for frequency hopping and synchronization of links built on this technique. Thereafter a simulation chain of the system was designed using MATLAB with the goal of analyzing the performance of the link. This included simulating bit error rate (BER) curves in static AWGN channels and applying different jamming waveforms to the link to evaluate the anti-jamming capabilities. Another part of the project involved investigating the feasibility of implementing a FHSS system on a particular SDR platform.

1.5 Societal, ethical and ecological aspects

The data link described in this work is not designed for a specific application or product as it is merely experimental. However, one should still consider the potential future applications that can be based on this work. The given specifications regarding range and relative velocity suggest that the system shall be capable of operating in airborne vehicles. This includes anything from commercial aircraft to military-grade products such as fighter aircraft, UAVs etc, of which the latter are of

particular interest to SAAB.

There are no ecological nor environmental impacts of this thesis work. A data link interacts with the physical world through electromagnetic radiation. Naturally, any such experimentation was conducted in accordance with the recommendations and legislation given by Strålsäkerhetsmyndigheten (Swedish Radiation Safety Authority) [3].

2

Digital Communication Theory

In this chapter a foundation is built for understanding the analysis and design of communication links. It deals with concepts applicable to communication links in general, with focus on concepts particularly important to the data link developed in this thesis work.

2.1 Channel coding

All forms of communication over a noisy and unreliable channel need some sort of error control. The basic idea that all coding techniques have in common is that redundant data is added to the transmitted message to enable error control at the receiving side.

In coding theory, a distinction is made between error-detecting and error-correcting codes. The more intuitive of the two, error detecting codes, usually consist of checksums or hash functions known to both the transmitter and receiver. The simplest one is arguably the parity bit code where a single bit is added to each message to either ensure the total number of 1s is odd or even. The receiver can then detect if a single bit error has occurred but is not able to specify which bit is erroneous, hence the name error *detecting* code. On the other hand, error correcting codes can both detect and correct errors introduced by the communication channel. An interesting property (or performance indicator) of a code is its error-correcting capability. A code with error-correcting capability t can correct all error patterns with t or fewers bit errors in a codeword, but not all error patterns with $t+1$ or more errors.

Coding theory is a large field itself with countless coding schemes and decoding algorithms. Since channel coding is not the main interest of this thesis, the focus will only lie on the codes implemented in our data link, namely convolutional codes and Reed-Solomon codes.

2.1.1 Convolutional error-correcting codes

Convolutional codes were first introduced by Peter Elias in 1955 as an alternative to the standard linear block codes. His ideas were published in a paper that later became to be regarded as one of the most influential papers in information theory [4]. Its popularity increased further when Andrew Viterbi in 1967 presented a relatively low complex algorithm for maximum likelihood-decoding convolutional codes

using trellises [5, pp. 249–255]. Since its development in the 1960s until today, convolutional codes have been used extensively in a wide range of standards and applications such as satellite communications, mobile communications, radio, television, WLAN etc. Often in combination with other coding schemes.

Block codes encodes the information bits block-wise. In other words, a given data block is always encoded into the corresponding codeword regardless of previous inputs to the encoder. The code is said to be *memoryless*. Contrary to linear block codes, convolutional codes have a *memory* meaning that the output of the encoder is not only dependent on current inputs but also previous ones. Thus it can be seen as a continuous, stream-oriented encoder that encodes an infinitely long stream of data into an infinitely long codeword. The most fundamental metric for convolutional codes is the code rate R_c which is the ratio of the number of input bits to the number of output bits in an encoder. Therefore it describes the amount of redundancy introduced. Intuitively, a lower code rate increases the error correction capability of the code at the expense of spectral efficiency.

A convolutional encoder consists of memory cells (containing previous inputs) and modulo-2 adders (for *binary* convolutional codes). It is usually denoted as a (n, k, L)-code where n, k and L represent the number of input bits, number of output bits and number of memory cells, respectively. A simple $(1, 2, 2)$-code is depicted as a block diagram in Figure 2.1.

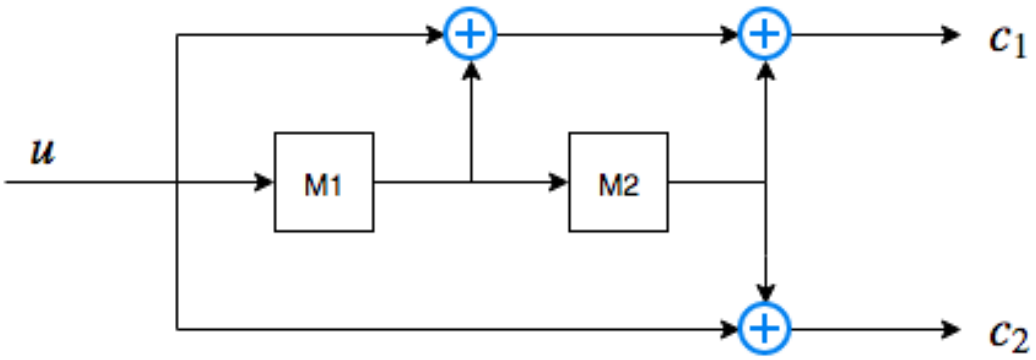


Figure 2.1: Block diagram of a convolutional code with $L = 2$ and $R_c = 1/2$.

Assume that the encoder is initialised in the all-zero state, i.e., M1 and M2 equal zero, and the input sequence $u = [1011000...]$ is to be encoded. For the upper branch in Figure 2.1 representing c_1, the first input bit is added to the memory of the first cell and thereafter added to the memory of the second cell. Whilst the lower branch representing c_2 only adds the input bit to the memory of the second cell. After the input bit at time instance t has been encoded the memory cells are updated. M1 and M2 are updated to contain input bits $u[t-1]$ and $u[t-2]$, respectively. Thus, the output of the encoder can be written as,

$$\begin{aligned} c_1[t] &= u[t] \oplus u[t-1] \oplus u[t-2] \\ c_2[t] &= u[t] \oplus u[t-2] \end{aligned} \tag{2.1}$$

which makes the reason for the name *convolutional* code obvious.

Table 2.1 shows the full procedure of encoding the given input sequence and the resulting outputs c_1 and c_2. After encoding, the outputs are multiplexed (read row-wise left to right) to create a single data sequence ready for modulation. Note that the encoder initializes and terminates in the all-zero state. Because of this, the branches of a standard convolutional encoder are often interpreted as FIR (finite impulse response) filters.

Table 2.1: Encoding $u = [1011000...]$ with the convolutional encoder in Figure 2.1.

t	**u**	**M1**	**M2**	c_1	c_2
1	1	0	0	1	1
2	0	1	0	1	0
3	1	0	1	0	0
4	1	1	0	0	1
5	0	1	1	0	1
6	0	0	1	1	1
7	0	0	0	0	0

2.1.2 Reed-Solomon error-correcting codes

Reed-Solomon codes are classified as memoryless and polynomial linear block codes and were developed by Irving S. Reed and Gustave Solomon [6]. To fully comprehend the algorithms in its entirety a deep understanding of finite field arithmetic and Galois field theory is needed. Instead of a step-by-step explanation, a simplified version of the code and its performance parameters will be discussed.

A finite field is a field (or set) with a finite number of elements. And finite field arithmetic dictates the rules of the elementary mathematical operations addition, subtraction, multiplication and division. A property of finite fields, that Reed-Solomon codes take advantage of, is that all mathematical operations applied to two elements in the field results in another element in the field. In other words, overflow shall be avoided. This is achieved by calculating the $\mod(2^n)$ of the result after a standard operation in the field.

A carefully designed irreducible generator polynomial is used for generating the code. A polynomial representation of the input data is divided by the generator polynomial and the remainder, given by $mod(2^n)$, is appended to the input data. In contrast to convolutional codes, the input remains unchanged in the code and an entire block of data is encoded at a time. The code is characterised by its block length n, message length k and alphabet size q, which usually equals $n + 1$, see Figure 2.2 . For instance, a RS$(n = 255, k = 223)$-code encodes 223 symbols and appends $n - k = 32$ parity symbols to the block, where each symbol consists of 8 bits (since $2^8 = 256 = n + 1$). Its error correction capability t naturally increases with the number of parity symbol and is given by $t = 0.5 \cdot (n - k)$.

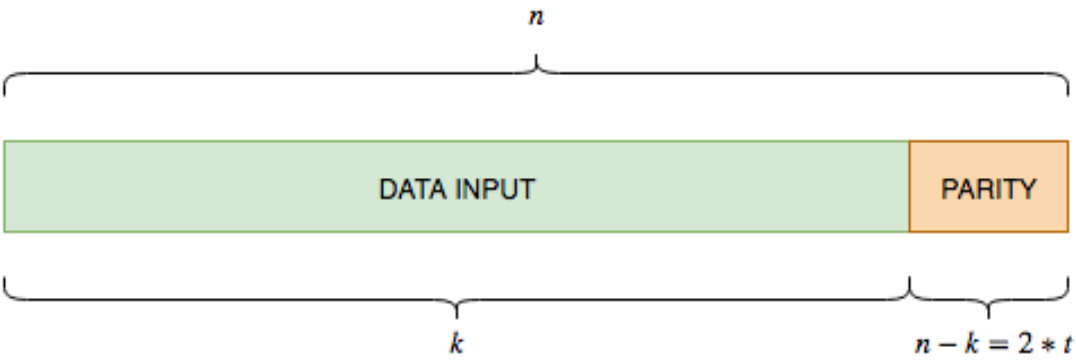


Figure 2.2: Structure of Reed-Solomon encoded block

The well-defined structure of the constructed field and the resulting laws of arithmetic are what enables the decoder to detect abnormalities of the field, caused by noise, and subsequently correct errors.
siunitx

2.2 Digital modulation

In digital communication, modulation is the process of modulating a discrete data signal onto a periodic waveform referred to as a *carrier wave*. The main performance metrics of modulation schemes are their spectral efficiency R and their quality degradation in the presence of noise, often illustrated with BER vs. E_b/N_0 plots. The spectral efficiency describes the amount of information that can be transmitted over a finite bandwidth and is usually measured in bit/s/Hz or simply bit/symbol in the discrete case.

Assume that the carrier wave is a complex exponential given by

$$x(t) = A \cdot e^{-j(2\pi f \cdot t + \phi)}. \tag{2.2}$$

To convey information using the complex exponential it is needed to adjust its properties over time in a controlled fashion that is known to the receiver. The carrier wave has three parameters that can be changed. Its amplitude A, frequency f and

phase ϕ. Simple modulation techniques such as ASK (Amplitude Shift Keying) and PSK (Phase Shift Keying) modulates the information by varying only one parameter of the carrier wave whilst keeping the other two constant. Of course, a combination of the parameters can be modulated as done in QAM (Quadrature Amplitude Modulation) where both the phase and amplitude are modulated.

2.2.1 Binary Phase Shift Keying

To exemplify, let us look at the resulting carrier wave in time domain when modulating a bit sequence using BPSK (Binary Phase Shift Keying). As the name suggests, BPSK is changing (keying) the phase of the carrier wave from 0 radians to π radians whilst maintaining a constant amplitude and frequency.

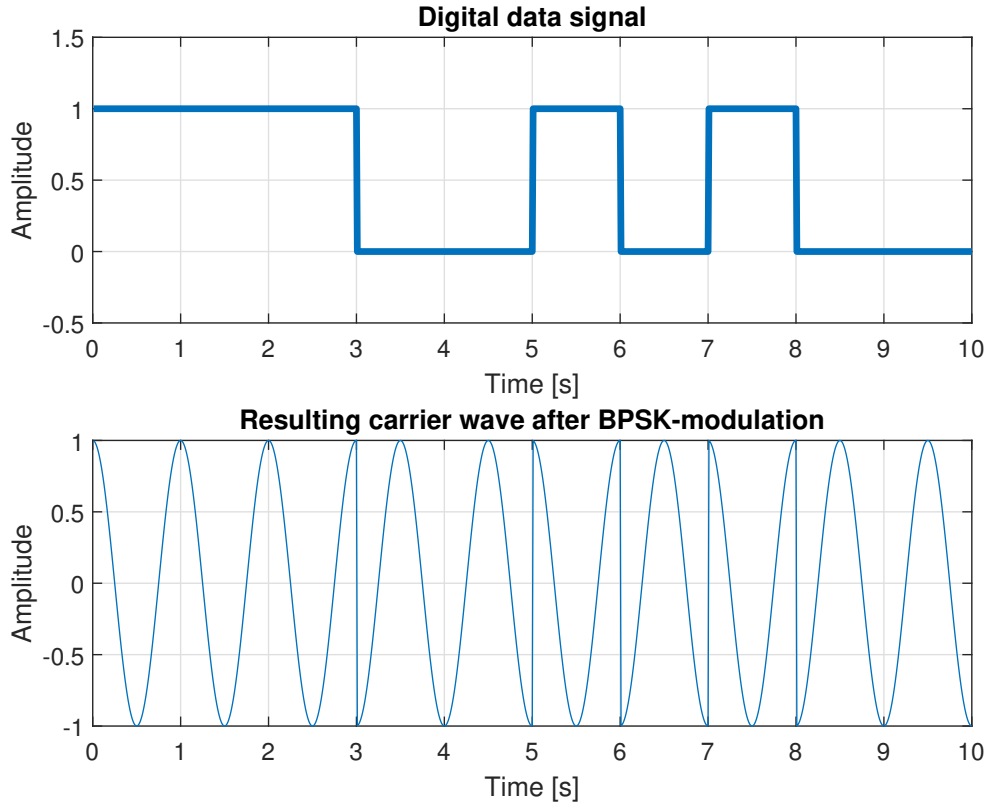


Figure 2.3: BPSK-modulating a bit sequence

The upper plot in Figure 2.3 shows a discrete-time digital signal representing a sequence of ten bits. In BPSK, a 1-bit corresponds to $\phi = \pi$ and a 0-bit corresponds to $\phi = 0$. In the time interval $t \in [0, 2]$, three consecutive 1-bits are modulated. Thus, the phase of the carrier wave in the lower plot in Figure 2.3 has a constant phase of $\phi = \pi$. However, at time instances $t = [3, 5, 6, 7, 8]$, there is a π radians phase shift of the carrier wave since the discrete-time digital signal jumps from representing a 1-bit to a 0-bit, and vice versa.

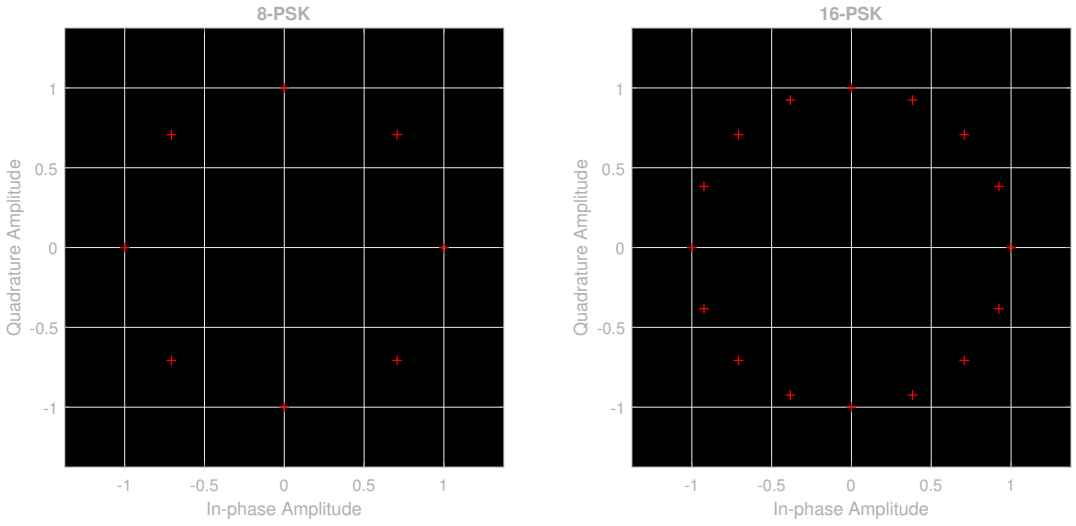
2.2.2 M-ary Phase Shift Keying

In BPSK, each bit is independently modulated with a phase $\phi = (0, \pi)$. In other words, each BPSK-symbol represents one data bit and hence its spectral efficiency is $R = 1$ bit/symbol. To increase the spectral efficiency, groups of bits are modulated at a time.

Let M be the number of symbols in a modulation scheme, also known as the *modulation order*. The number of bits per modulated symbol is then given by $\log_2(M)$. In M-PSK, the phases of these M symbols are chosen as Equation (2.3). A higher modulation order increases the spectral efficiency which in turn increases the data rate of the system. However, a higher modulation order decreases the distance between constellation points (symbols) which increases the system's vulnerability to distortion of the symbols caused by noise.

$$\phi_m = \frac{(2m-1)\pi}{M}, \quad m = 0, 1, \ldots, M \tag{2.3}$$

Figure 2.4 shows constellation diagrams of PSK with different modulation orders. The diagram to the right, with $M = 16$, contains four bits per symbol whilst the one to the left, with $M = 8$, contains three bits per symbol. A phase shift larger than $\frac{\pi}{16}$ caused by a noisy channel would cause a demodulation error for 16-PSK since the distance between adjacent constellation points is $\frac{\pi}{8}$. However, a phase shift of $\frac{\pi}{16}$ in 8-PSK would not result in an error since the distance between adjacent constellation points is more than twice as large as the phase shift.



(a) 8-PSK **(b)** 16-PSK

Figure 2.4: Constellation diagrams of M-PSK

2.2.3 M-ary Differential Phase Shift Keying

DPSK (Differential Phase Shift Keying) is a modulation technique very similar to BPSK and M-PSK, however there is a key difference that affects the implementation as well as the performance. It introduces memory, somewhat akin to convolutional codes in Section 2.1.1. Instead of shifting the phase of the carrier wave according to a fixed constellation, it applies a relative phase that is given by the phase difference between consecutive symbols. This permits the receiver to be implemented without a dedicated algorithm, such as a PLL (Phased Locked Loop), for tracking an absolute phase reference of the carrier. In literature, receivers of this sort is called *non-coherent receivers*. Their design allows for cheaper and lower complexity implementations, at the expense of increased sensitivity to noise, compared to traditional coherent receivers.

A DPSK modulation process is illustrated step-by-step in Table 2.2. Assume modulation order $M = 2$ or in other words binary DPSK (BDPSK) and the bit sequence $b[n]$ that is to be modulated equals [101110]. Also, assume the same bit mapping as for BPSK described in Chapter 2.2.1, i.e., a 1-bit corresponds to $\phi = \pi$ and a 0-bit corresponds to $\phi = 0$. The key difference though, is that a 1-bit in DPSK signifies a phase *transition* of π radians whilst in BPSK it signifies an absolute phase of π radians.

Table 2.2: Binary DPSK-modulating a bit sequence

Time [n]	**Input bit b[n]**	**Phase transition [rad]**	**Symbol s[n]**
0	-	-	$A \cdot e^{j\pi}$
1	1	π	$A \cdot e^{j\cdot(\pi+\pi)} = A$
2	0	0	$A \cdot e^{j\cdot(0+0)} = A$
3	1	π	$A \cdot e^{j\cdot(0+\pi)} = A \cdot e^{j\pi}$
4	1	π	$A \cdot e^{j\cdot(\pi+\pi)} = A$
5	1	π	$A \cdot e^{j\cdot(0+\pi)} = A \cdot e^{j\pi}$
6	0	0	$A \cdot e^{j\cdot(\pi+0)} = A \cdot e^{j\pi}$

Finally, since the symbol $s[n]$ depends on symbol $s[n-1]$, an assumption of the phase of symbol $s[0]$ needs to be made. In this example $s[0] = A \cdot e^{j\pi}$. In the case of a non-coherent receiver (also called differential receiver), the first symbol in a block of DPSK symbols is discarded and not used for conveying actual information bits.

2.2.4 Performance in the presence of noise

As briefly mentioned, the most common metric for evaluating the performance of a modulation scheme is its resulting bit errors caused by noise. The simplest and most commonly used noise model is complex AWGN (Additive White Gaussian Noise) which is defined by its properties

- **Additive:** the noise is added (not multiplied) to the transmitted signal, $y(t) = s(t) + n(t)$
- **White:** the PSD of the noise if flat, i.e., same power for all frequencies
- **Gaussian:** the noise has a complex Gaussian distribution

$y(t)$, $s(t)$ and $n(t)$ are the received signal, transmitted signal and noise signal, respectively. To make a fair comparison of different modulation schemes their spectral efficiency needs to be taken into account. The ratio E_b/N_0, often called the *normalized signal-to-noise ratio*, is commonly used as a measure of the signal quality. It is also related to the ratio of the average powers of $s(t)$ to $n(t)$. The energy per bit, E_b, can be defined as the average received power P divided by the bit rate R_b (number of transmitted information bits per second), while N_0 is the noise power spectral density in W/Hz [7].

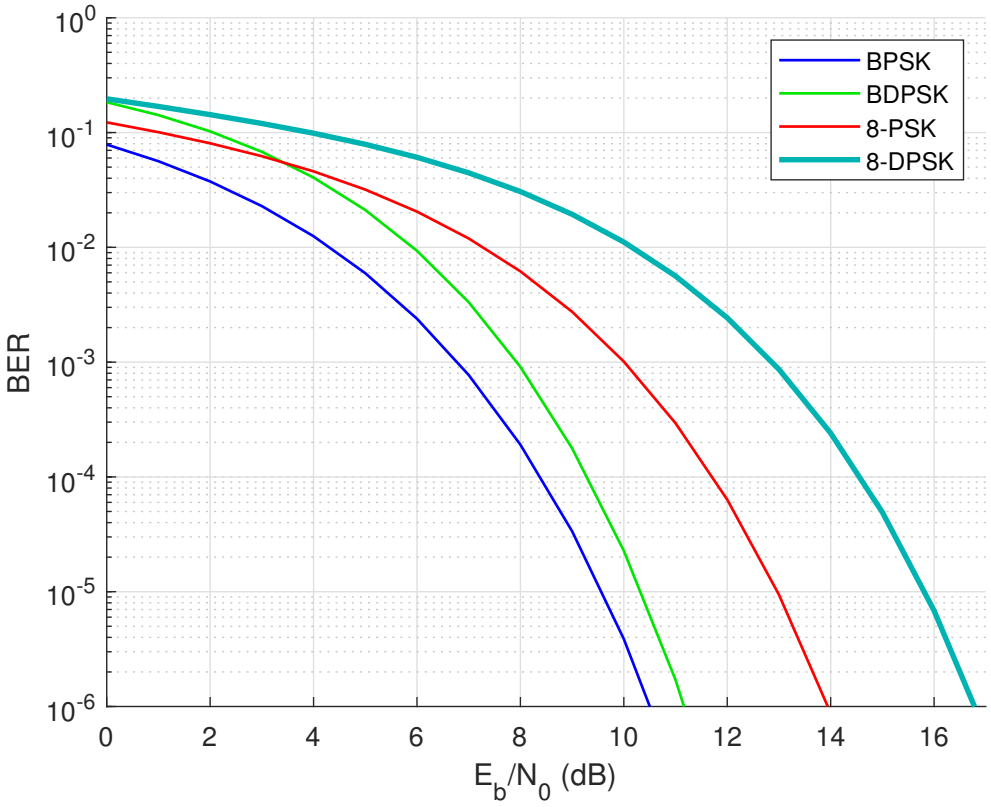


Figure 2.5: BER vs. E_b/N_0 for PSK and DPSK of different modulation orders

Figure 2.5 shows theoretical BER vs. E_b/N_0 plots of PSK and DPSK with different

modulation orders. As common practice, the y-axis is plotted in logarithmic scale. The curves should be interpreted as a marker for what relationship between the signal and noise power is needed to achieve a certain bit error rate. For instance, to achieve a bit error rate of 10^{-5}, or in other words, on average one bit error for every 100000 transmitted bit, an $E_b/N_0 \approx 9.5$ dB is required for BPSK. In the introduction to this chapter, it was suggested that a higher modulation order increases the bit error rate because the distance between constellation points decrease with an increasing modulation order. This is evident in Figure 2.5 where there is approximately a 3.5 dB difference in E_b/N_0 between 8-PSK and 2-PSK (BPSK) at BER = 10^{-5}. Furthermore, there is a relatively large difference (≈ 3 dB) between 8-DPSK and 8-PSK as well.

3

Frequency-Hopping as an Interference Mitigation Strategy

In this chapter the main interference rejecting mechanisms of FHSS are covered briefly. Because jamming is the act of intentionally causing interference, some common jamming strategies are also described in this chapter.

3.1 Frequency-Hopping Spread-Spectrum

In a spread-spectrum (SS) system the total spread-spectrum or transmit bandwidth B_s is greater than the baseband signal bandwidth B [5, pp. 337-339]. This results in increased resistance to intersymbol interference (ISI) and resistance to narrowband interference (i.e., interference occupying a fraction of the transmit bandwidth). As will become clear, the latter property is useful for systems wanting to achieve jamming resistance. The two dominating SS techniques are *direct-sequence* (DS) and *frequency-hopping* (FH). If B is increased by a *spreading* or *diversity factor* K to the instantaneous bandwidth B_s, i.e., the spreading is applied directly to the data signals, this is denoted DS. This is achieved by mixing the data signal with a pseudorandom *spreading sequence* for which the symbol length is approximately $1/K$ times the data symbol length. The pseudorandom nature of the sequence means that, ideally, the signal will appear as white Gaussian noise to any receiver monitoring B_s. Only when the receiver has access to the same spreading sequence can the original data be retrieved.
In a FH system the instantaneous transmit bandwidth is not different from baseband. Instead the carrier frequency of the system is periodically changed or 'hopped' in a pattern pseudorandomly generated from a *hopset* with K frequencies $\{f_1, f_2, ..., f_K\}, f_1 < f_2 < \cdots < f_K$ [8]. Each frequency in the hopset is at the center of a *channel* with bandwidth B, and the total SS bandwidth or *hop bandwidth* B_s is approximately equal to KB. B_s is typically contiguous but it does not have to be; in some cases it is intentionally made non-contiguous for improved interference rejection. Similarly to DS, the receiver needs access to the same pseudorandom pattern as the transmitter in order to retrieve or *dehop* the hopped baseband signal.

3.1.1 Some terminology

It is useful to further highlight some terms that will be used several times throughout the text relating to FHSS. As mentioned, the hop set is the set of K frequencies

among which the carrier frequency of the system is periodically hopped. A specific configuration of the hopset is called the *frequency-hopping pattern*. The rate at which the carrier frequency is hopped is called the *hop rate*, and the time between two hops is the *hop interval*. The length of the hop interval is the *hop duration*. The hop duration consists of the *dwell time*, which is the time during which useful data can be transferred, and the *switching time*, which is the time it takes for the frequency synthesizer circuitry to change the carrier frequency. This is also known as the *dead* or *idle time* due to the fact that no passband signal is transmitted nor observed during this time. During switching time no transmission or reception of data symbols is possible, which negatively impacts the data transmission rate of a FHSS system compared to a single-carrier system.
FHSS can be further divided into *fast frequency hopping* (FFH) and *slow frequency hopping* (SFH) [8]. FFH occurs when the hop rate is equal to or larger than the data symbol rate, and SFH when the opposite is true, i.e., several data symbols are transmitted during a hop interval. FFH is very demanding on frequency synthesis hardware as it must be able to synthesize and apply a new carrier frequency multiple times per symbol, which is challenging at high[1] symbol rates. An example of a standardised TDL using FHSS with a relatively high hop rate of 77 kHz is Link 16, used by NATO among others [9].

3.1.2 Interference mitigation properties

As mentioned, SS systems in general exhibit ISI- and narrowband interference rejection properties. In the case of FH, the main interference rejection principle is avoidance (as opposed to spectrum spreading), because the system will eventually hop away from a frequency channel in which interference is present [8]. Additionally, the fast switching of carrier frequencies and the use of large hop bandwidths makes detection of FH signals difficult, which is desirable in applications where the involved parties want their communications to remain hidden.

3.1.2.1 Narrowband interference rejection

Figure 4.7 illustrates a FHSS system experiencing strong interference (jamming) in one of its channels. The bits sent over the jammed channel will have a much lower SNR than those sent over other channels, and are thus more likely to be decoded in error. Assuming that the interference remains in one channel, its impact on the system can be reduced in a number of ways, one of which is to remove the channel from the hopset (a procedure known as *spectral notching*). However, barring this strategy (which, among other things, requires a bidirectional link) the obvious choice is to increase the number of channels K available for hopping, which would expose fewer bits to the jammed channel.
Recalling that $B_s \approx KB$, we can increase K either by increasing the available hop bandwidth B_s or by reducing the baseband bandwidth B to e.g. B_n. However,

[1]Because newer hardware and software will invariably change what constitutes fast frequency switching, it is impossible to define "high" in this context, however 10's of Msymb/s may be a likely number at this point in time.

assuming that the jammer maintains its bandwidth $B_J \approx B > B_n$, the system will not improve its anti-jam capability since more than one channel will be jammed in this situation, or in other words, the same number of bits are interfered with. The remaining option is to increase the hop bandwidth B_s.
This fact is reflected in the definition[2] of *processing gain* (PG) of a SS system:

$$PG_{dB} = 10 \log_{10} \left(\frac{B_s}{B} \right) \tag{3.1}$$

PG is a measure of the SS system's ability to reject narrowband interference compared to a non-SS system [1]. Another way to view PG is to consider that, with the jamming example above, the system is jammed $1/K$ of the time and thus receives only $1/K$ of the interfering signal power [5, pp. 337–339].
Taking this reasoning further, it is conceivable that a jammer may attempt to jam the entire hop band, i.e., generate interference over B_s (a strategy denoted *barrage* or *wideband noise* jamming). Since all channels now experience interference, the SS system performance is equivalent to a non-SS system in AWGN where the single-sided noise power density N_J is governed by the jammer power.

3.1.2.2 ISI rejection

Any wireless communication system operating in the wideband regime (i.e., whenever the signalling or symbol time T is smaller than the channel delay spread τ) is susceptible to ISI caused by frequency-selective fading, SS systems included. While a SS system can utilize any of the usual methods to combat ISI (e.g. channel equalization), FH systems have a distinct advantage; if the system is able to change carrier frequency before the first interfering multipath component has arrived, ISI is essentially eliminated [5, pp. 337–339]. In other words, when the hop interval is shorter than the delay spread of the channel (assuming a two-ray model) the system is immune to frequency-selective fading. This property is generally reserved for FFH systems due to the high hop rates involved, meaning that SFH systems generally perform the same as single-carrier systems in this regard.

3.1.2.3 Low Probability of Intercept

Low probability of intercept (LPI) is the ability of a communication system (or other RF systems such as radar) to operate with reduced risk of being detected by an adversary [1]. Because detecting RF signals usually entails measuring the amount of energy present on a band of interest (i.e., SNR), the aim is to reduce the ability to detect this energy by various methods. FH systems provide LPI mainly by

- using large hop bandwidths. Aside from the previous discussion on the advantages of PG, using large portions of spectrum is beneficial because it makes it potentially more difficult for a third party to monitor the entire bandwidth at once.

[2]There exists an alternate definition defined as B_s/R_b, where R_b is the uncoded bit rate of the link.

- occupying a given frequency band for a very short amount of time. As mentioned, most detectors operate on the principle of measuring SNR, and this typically includes an integrator operating over some time T or equivalently some number of samples N. The integrator output is then put through a threshold operation to determine whether a signal is present. Very short bursts of energy makes this process more difficult to do while maintaining low false alarm probability.
- changing frequency in a seemingly random manner (Section 3.1.3 will expand on this)

Other LPI strategies may include using antennas with high sidelobe attenuation, to not radiate power in an unintended direction; powerful channel codes to reduce the required output power and/or lower the risk of packet re-transmission; using uni-directional protocols to reduce the possibility of discovering the intended target of a transmission, and many more. LPI may even entail limiting the visibility of physical hardware such as antennas and radomes.

3.1.3 Pseudorandom sequences and synchronization

Pseudorandomness (PR) is a term meaning 'random in appearance, but reproducible by deterministic means' [1]. While the strict mathematical definition is more involved, the aforementioned is largely sufficient as a base for understanding the use of PR in SS applications. A pseudorandom sequence, then, is a sequence (e.g. a stream of numbers) that appears random but is exactly (re)producible by an algorithm or generator. Such a generator is often called a *pseudonoise (PN) generator* since its objective is to generate sequences that are indistinguishable from random noise to a casual observer [10]. The use of PR in FH was briefly mentioned in a previous section; it is used to generate the hopping pattern of the system. Since the definition of PG, Equation (3.1), is indifferent to how B is spread over B_s, one could be led to believe that using a static pattern should be sufficient. In the presence of an intelligent jammer, however, this is not the case. In fact, a common assumption in the analysis of anti-jam systems is that the jammer has complete knowledge of the system, *except* that it does not have the key necessary to generate the pseudorandom sequence [1]. This will be further discussed in Section 3.2.

A key feature of any practical PN generator is that its output is wholly determined by an initial state, known as a seed or key [10]. In software the key is typically the starting value of an algorithm, and in hardware it is typically a specific configuration of a series of shift registers. This makes it possible to synchronize a locally generated PN sequence with a received sequence in e.g. a communication system.

While synchronization of PN generators in SS systems is a wide topic with many possible approaches, in FH it is typically carried out as a two-part dehopping process: FH *acquisition* (coarse synchonization) and FH *tracking* (fine synchronization) [1]. Acquisition is the act of aligning the local PN sequence to within a fraction of a hop interval of the received sequence. This is typically done by exploiting the very good autocorrelation properties of PN sequences; even very small phase shifts will yield very low correlation [10]. By continuously correlating the sequences and adjusting the local PN generator, coarse alignment will eventually be reached. Any residual

misalignment is then handled by a tracking algorithm that attempts to bring the phase difference to zero (typically implemented as a PLL).

3.2 Jamming strategies and waveforms

Jamming is the deliberate use of RF signals to disrupt or disable wireless communications. While there are a number of "standard" waveforms that can be utilized, some of which are presented below, there is no single optimal jammer waveform or strategy that causes the largest damage to all systems. Conversely, there is no single SS system that can be optimally applied against all jammer waveforms [1]. Generally speaking, the problem posed to a jammer becomes: for a given jammer power J and bandwidth B_J, how shall the power be distributed in both time and frequency and what waveform shall be used to cause maximal damage to the target system?

3.2.1 Wideband noise jammer

Wideband or barrage jamming is the act of generating interference over the entire hop band of a SS system [1]. This is achieved by spreading Gaussian noise over B_s with a resulting noise power density according to

$$N_J = \frac{J}{B_s} \tag{3.2}$$

in W/Hz where J is the total available jammer power and B_s is the total hop bandwidth.
It was previously mentioned that an intelligent jammer can be assumed to know every aspect of the SS system except the key to the PN generators; in this case however, it is a simple brute-force jammer that does not exploit *any* knowledge of the SS system aside from B_s. It is consequently the least effective jammer, and is considered the *baseline* to which other jamming strategies are compared. An anti-jam system that performs close to or better than the baseline case in *any* jamming situation is considered an effective anti-jam communication system [1].

3.2.2 Partial-band noise jammer

A closely related jammer to the wideband jammer is the partial-band noise jammer. As the name suggests, the noise jammer spreads Gaussian noise over a bandwidth smaller than the hop bandwidth B_s. Consequently the power spectral density is higher and knowledge of the targeted system's channel bandwidth is exploited. A smaller bandwidth also implies that the targeted system experiences a sort of frequency-selective noise where jammed channels have a lower equivalent E_b/N_0 than others. The power spectral density of a partial band jammer can be expressed as

$$N_J^{'} = \frac{N_J}{\rho_B}, \quad \rho_B = \frac{B_J}{B_s} \leq 1 \tag{3.3}$$

where N_J is the PSD of a barrage jammer and ρ_B is the ratio of jamming bandwidth B_J to total hop bandwidth B_s. For instance, a 10 MHz partial band jammer targeting a system of total hop bandwidth 100 MHz yields $\rho_B = 0.1$. In consequence the PSD is 10 times larger than an equivalent barrage jammer, given the same available power J.

3.2.3 Pulse jammer

A pulse jammer has an arbitrary bandwidth B_J in the interval $(0, B_s)$. Therefore it can be seen as either a pulsed barrage jammer or pulsed partial band jammer, although the waveform can be arbitrary. The key difference is that a pulse jammer distributes its power both in frequency and time by on/off-keying. Assuming a pulsed barrage jammer, the PSD N_J is scaled by ρ_T, which is the ratio of the time duration the jammer is "on" to the time duration it is "off". The PSD of the pulsed barrage jammer can then expressed as

$$N_{PJ} = \min\left(\frac{N_J}{\rho_T}, \frac{J_{\text{Max}}}{B_s}\right), \quad \rho_T = \frac{T_{\text{On}}}{T_{\text{Off}}} \leq 1 \tag{3.4}$$

where N_{PJ} saturates at J_{Max}/B_s when $\rho_T \to 0$. Important to note is that the instantaneous power of the pulse jammer is larger than a continuous jammer given that the time averaged power J is identical. This assumption was made to compare the different jamming strategies. Because pulse jamming causes effects similar to small-scale fading, i.e., burst errors, this can to an extent be mitigated by the use of interleaving.

3.2.4 CW/Multi-tone jamming

A continuous-wave (CW) or tone jammer is quite simply a a sinusiod with power J placed at a carrier frequency somewhere inside B_s [1]. In multi-tone jamming this takes on the form

$$J(t) = \sum_{l=1}^{N_t} \sqrt{2J/N_t} cos(\omega_l t + \theta_l) \tag{3.5}$$

where N is the number of equal-power tones, ω_l is the carrier frequency and θ_l an arbitrary phase over $[0, 2\pi)$. M-DPSK modulation has notably worse performance under multi-tone jamming than partial-band jamming [1].

3.2.5 Repeat-back jammer

FHSS systems with low hop rates (typically SFH systems) can be vulnerable to so called repeat-back jammers. If the hop rate is sufficiently low (from the jammer's perspective) there may be time for the jammer to record, analyse and re-transmit the intercepted signal, during a single dwell time of the FHSS system. The idea is that by using the FHSS system's own signal, it can cause additional damage. This type of jammer is sometimes referred to as a frequency-following jammer since it is continuously "following" the FHSS system's hop pattern.

4

System Design and Simulation

This chapter describes the process of adapting the specification into a system suitable for simulation and testing. The design process was mainly concerned with interpreting system parameters, conceptualizing functions as block diagrams, and writing MATLAB code.

4.1 System specification

Table 4.1 lists a selection of relevant parameters from the specification. While most parameters have been adapted without modification into the simulated system, others have been adjusted to either simplify the simulation or to improve anti-jamming capabilities.

Table 4.1: Communication link specification

Maximum relative velocity	600 m/s
Symbol length	200 ns
Frequency Hopping rate	1991 hops/s
Frame duration	10.24 ms + 640 μs guard
Capacity	8192 kbit/s
Target BER	10^{-5}
Modulation scheme	8-DPSK
Uncoded frame size	88440 bits
Number of hops per frame	20
Outer channel coding	Reed-Solomon (255, 201)
Inner channel coding	3/4 - Convolutional code w/ $d_{min} = 8$
Frame preamble	255 BPSK symbols
Block preamble	13 BPSK symbols

The inner convolutional code has nine states and generator matrix

$$\begin{pmatrix} 3 & 6 & 10 & 15 \\ 0 & 16 & 3 & 13 \\ 16 & 5 & 2 & 17 \end{pmatrix}$$

4.2 System design

In this section the design and simulation process is described. The section begins with a flowchart view of the simulated system and then goes into detail on the individual components.
From a practical standpoint the "simulated system" is a collection of MATLAB functions, however conceptually the system was designed with the implementation stage in mind, where it needs to function on an SDR platform. This has implications mainly in the up- and downsampling stages at passband, which is handled in hardware by the SDR, but needs to be simulated in software.

4.2.1 System Overview

Figure 4.1 shows a simplified representation of the simulation chain, which starts with the transmitter (TX) stage processing raw data bits into frequency-hopped passband samples. The samples are passed through a channel, has a jamming waveform added, and are then passed to the receiver (RX) which essentially performs the TX operations in reverse. A BER is then calculated.

While the specification suggests using BPSK preambles for synchronisation purposes, and these are introduced after symbol mapping, perfect synchronisation has been assumed in the simulation. This includes both PN generator- and frame-synchronisation, since these are implementation-specific issues. For this reason no synchronisation stage is present in the receiver, and the frame- and block-level preambles are simply discarded before BER calculation.

4.2.2 Frame Shaping

This first stage of the simulation chain generates a single frame of information bits and formats them for the channel coding stage. In accordance with the specification a bit vector with 88440 bits is generated and reshaped into a matrix with 20 rows, where each row represents a data block to be sent over its own FH channel. This number is equivalent to the number of hops per frame. Also note that, since the number of bits (and subsequently symbols) is several magnitudes larger than the number of hops, this FHSS system is of the SFH type.

4.2.3 Channel coding and interleaving

The information bits are then passed to the coding and interleaving block. A concatenated codes is used, an outer Reed-Solomon block code and an inner convolutional code, meaning that the coding is carried out in two steps. First, the encoder objects from the Communication Toolbox are created. RS-encoding is performed in one step on the entire matrix containing the information bits. The codeword length $N = 255$ and message length $K = 201$ from Table 4.1 are used. This generates an overhead of $(255 - 201)/201 = 26.9\%$ parity symbols that are appended to the frame. Thus, the total frame size is increased from (20x4422) bits to (20x5610) bits.

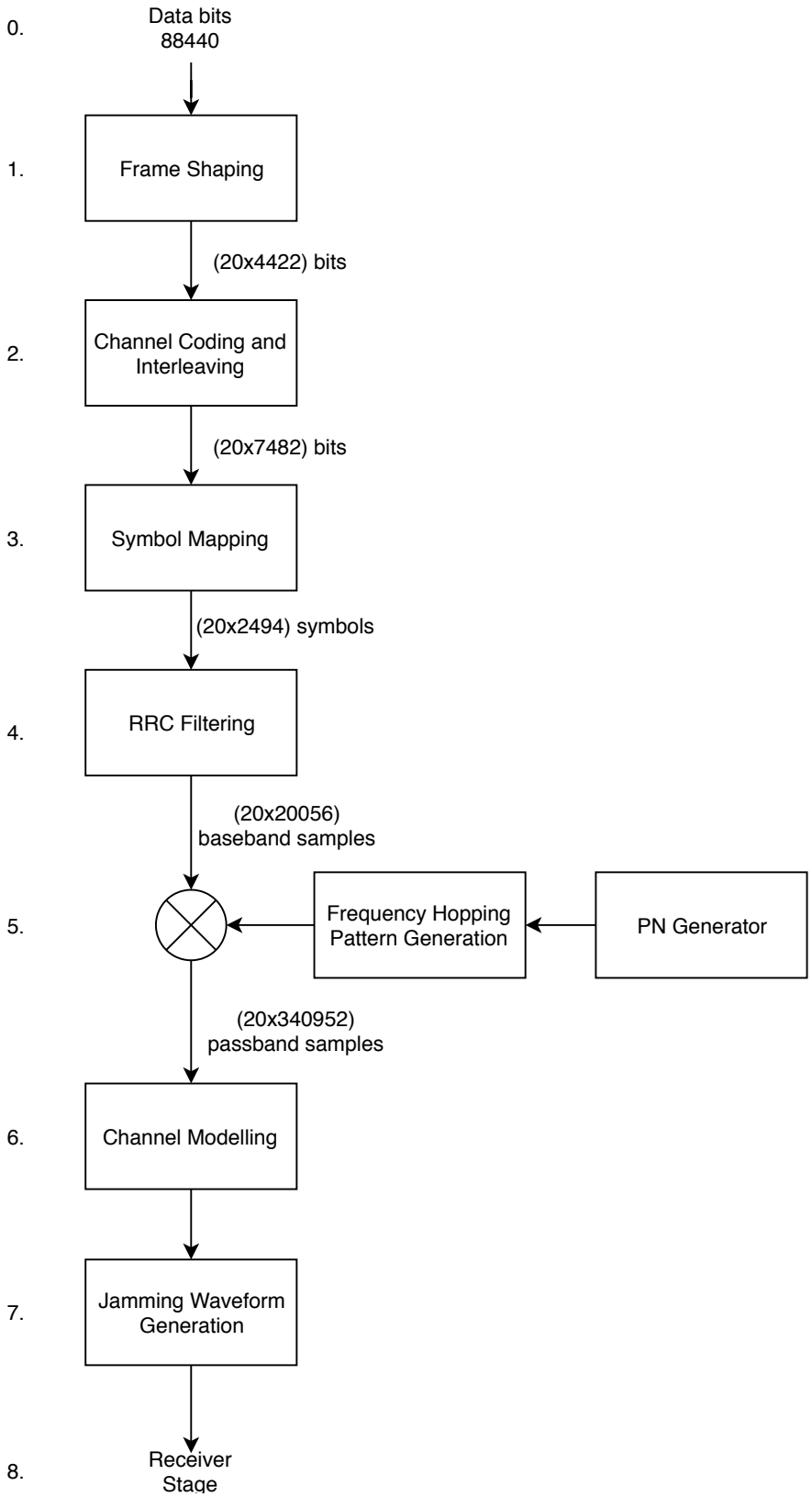


Figure 4.1: Overview of simulated system in MATLAB

The frame is then interleaved to spread the bits over all 20 rows. The interleaver is based on the Mersenne Twister algorithm that generates a uniform pseudorandom permutation of the input [11]. This is done to decrease the system's vulnerability to both burst errors caused by fading as well as jamming. An interleaving factor of 20 means an approximate spreading factor of 95%, that is, 95% of the bits belonging to one block are spread to the other 19 blocks.
The RS-encoded frame is thereafter passed to the 3/4-convolutional code that is implemented with a generator matrix and Trellis structure. The convolutional encoding is performed block-wise with the tail-biting method, meaning that the memory cells of the code initialises and terminates in the same state for every block. The tail bits that would be produced by the encoder for a zero-tail Trellis termination are avoided this way, at the expense of increased complexity of the decoder [12].
Since the coding is done for each block, the convolutional decoding in the receiver can be performed continuously without the need of buffering an entire frame. This potentially reduces latency since the decoding of one block can be carried out concurrently with the reception of the next block (though for de-interleaving and RS decoding, the entire frame is still needed). Finally, since the code encodes three information bits in four codeword bits, the frame size is increased by 25% resulting in (20x7480) total frame size.

4.2.4 Symbol mapping

At this stage the frame consisting of encoded bits is almost ready for symbol mapping. Before doing so, the frame size must be adjusted to be compatible with the chosen modulation scheme. 8-DPSK is used which means that groups of three ($log_2(8)$) bits are modulated at a time. Every block in the frame is therefore padded with two zeros, to make the block length divisible by the number of bits per symbol. This results in the total frame size (20x7482). The 8-DPSK-mapping is carried out block-wise since a 13-bit Barker code is prepended to each block. Thus the relative phase between consecutive 8-DPSK-symbols belonging to different blocks is uncorrelated. Thus, the starting DPSK symbol of each block is lost, which leads to 20 lost symbols per frame. However, the lost symbol contains only one information bit and two zero-padded bits. Therefore, the total number of lost information bits is 20. The decision not to pad with an entire extra 8-DPSK symbol was due to the constraints of the specification.
The resulting frame after symbol mapping is illustrated in figure 4.2. Each block contains 13 BPSK symbols representing a Barker sequence followed by 2494 8-DPSK symbols carrying the payload. As previously stated, the preambles are discarded in the receiver since the simulation assumes perfect synchronisation. In a practical implementation, however, the Barker code has two main purposes: to mark the start of a block, and to potentially aid in PN sequence tracking in the receiver.

4.2.5 RRC filtering

Up to this point, the frame has consisted of discrete complex samples representing 8-DPSK-symbols. To make the frame suitable for transmitting in a real commu-

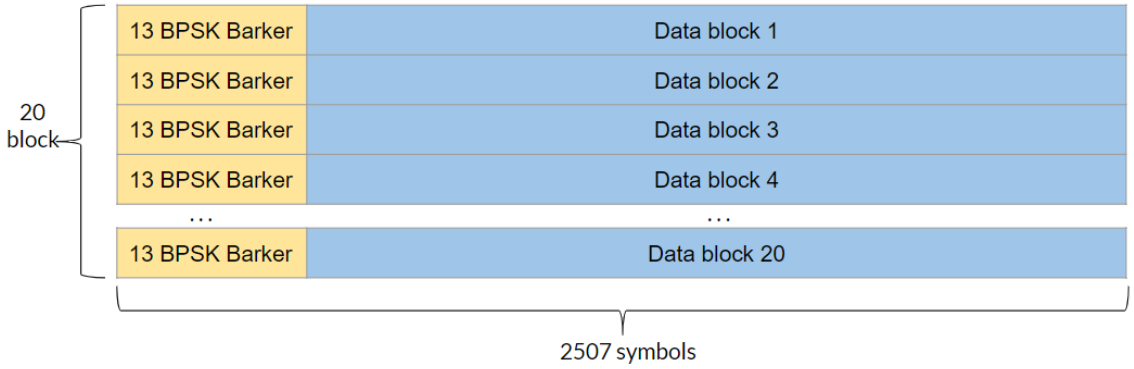


Figure 4.2: Frame structure. Proportions not to scale.

nication channel, the transmitted signal needs to be band-limited. In this system a root-raised cosine filter, in combination with a matched filter in the receiver, is used. This ensures a band-limited signal with large sidelobe suppression (relative to a rectangular pulse), where the excess bandwidth is dependent on the roll-off factor β, and minimises the intersymbol and adjacent channel interference [5, pp. 154–156]. Furthermore, the matched filter in the receiver works as a noise reducing filter and is the optimal receiver filter in an AWGN channel [13].
The design of the pulse takes the link specification as well as the hardware constraints into account. The symbol length T_{symb} is set to 200 ns which is equivalent to a symbol bandwidth B_{symb} of 5 MHz. Setting the roll-off factor, $\beta = 0.1$, generates 500 kHz of excess bandwidth and is calculated as

$$\Delta B = \frac{\beta}{T_{\text{symb}}}. \tag{4.1}$$

Therefore, the total bandwidth B_{tot} after filtering with the RRC can be calculated as

$$\begin{aligned} B_{tot} &= B_{\text{symb}} + \Delta B \\ &= B_{\text{symb}}\Big(1 + \beta\Big) \end{aligned} \tag{4.2}$$

which equals 5.5 MHz for $\beta = 0.1$ and $T_{\text{symb}} = 200$ ns. Moreover, the upsampling factor of the signal dictates how large sample rate is required for achieving $B_{tot} = 5.5$ MHz. It is known that the hardware has a maximal sample rate f_s in the 50 MHz range and that a minimum upsampling factor L_{up} of two is required for time synchronisation purposes in the receiver. The relationship between sample rate f_s and symbol bandwidth B_{symb} can be expressed as

$$f_s = B_{\text{symb}} \cdot L_{up}. \tag{4.3}$$

The values of f_s and L_{up} can be arbitrarily chosen as long as $L_{up} \geq 2$, $11 \leq f_s \leq 50$ MHz and Equation 4.3 holds true. A summary of chosen parameters and their valid ranges is composed in Table 4.2.

Table 4.2: Simulation parameters related to RRC filtering

Parameter	Notation	Value	Valid range
Roll-off factor	β	0.1	[0,1]
Symbol length	T_{symb}	200 ns	Fixed
Symbol bandwidth	B_{symb}	5 MHz	Fixed
Sample rate	f_s	44 MHz	[11, 50] MHz
Upsampling factor	L_{up}	8	[2, ∞)
Pulse span	N/A	10 symbols	[1, ∞)

4.2.6 Frequency hopping pattern generation

After generating the baseband frame, through RRC filtering, it is time to generate the hopset of carrier frequencies representing the FH channels. To accurately simulate an FHSS system with anti-jamming capabilities it is crucial to assign a unique center frequency to each block. That is, more than one block within a given frame shall not be transmitted with the same center frequency.

The specification states that the number of blocks, N_{block}, shall be 20. Also, the total bandwidth, B_{tot}, after RRC filtering was calculated to 5.5 MHz. For this reason a spread spectrum bandwidth of 110 MHz is required to ensure unique center frequencies and non-overlapping channels. The frequency band 220-330 MHz was chosen.

The PN Generator performs a random permutation of a vector with N_{block} elements. The seed input to the PN Generator is simply a constant known to both TX and RX. Naturally using a constant permutation would be a major weakness in a real implementation but for simulation purposes it has no importance. See Section 3.1.3 for more details on PN Generators.

4.2.7 Baseband to passband modulation

The frequency hopping pattern has now been generated but the baseband signals have not yet been upconverted to passband. A new sample rate, f_{master}, is defined. According to Nyquist's sampling theorem, the sample rate f_{master} needs to be at least twice the highest frequency components of the signal to ensure perfect reconstruction. The upper limit of the used frequency band is 330 MHz, and thus $f_{master} \geq 660$ MHz is required.

The baseband signal is interpolated (upsampled and low-pass filtered) to increase the sample rate of the signal. The interpolation factor L_{interp} is given by

$$L_{interp} = \frac{f_{out}}{f_{in}} \tag{4.4}$$

where f_{out} and f_{in} are the output and input sample rates, respectively. A master sample rate of 748 MHz was chosen which yielded $L_{interp} = 17$. Important to note is that the signal duration in continuous time nor the bandwidth are altered when

interpolating.

The signal is now ready for passband modulation by multiplying the signal in time domain with a complex exponential as follows:

$$x_{PB}(t) = \Re\Big(x_{BB}(t) \cdot e^{j2\pi f_c t}\Big) \tag{4.5}$$

where $x_{BB}(t)$ is the baseband signal after interpolation and f_c is the center frequency. Each block in the frame is processed separately with a unique frequency of the complex exponential. A PSD of a frame after passband modulation is shown in figure 4.3. The blocks reside in the 220-330 MHz band and are plotted separately solely for illustration purposes. In other words, a receiver would in reality "see" the sum of of the power distributed in the band. Figure 4.4 introduces time in the PSD of a point-to-point link transmitting one frame. Note how every block has a unique center frequency and that blocks are separated in time, i.e transmitted one by one.

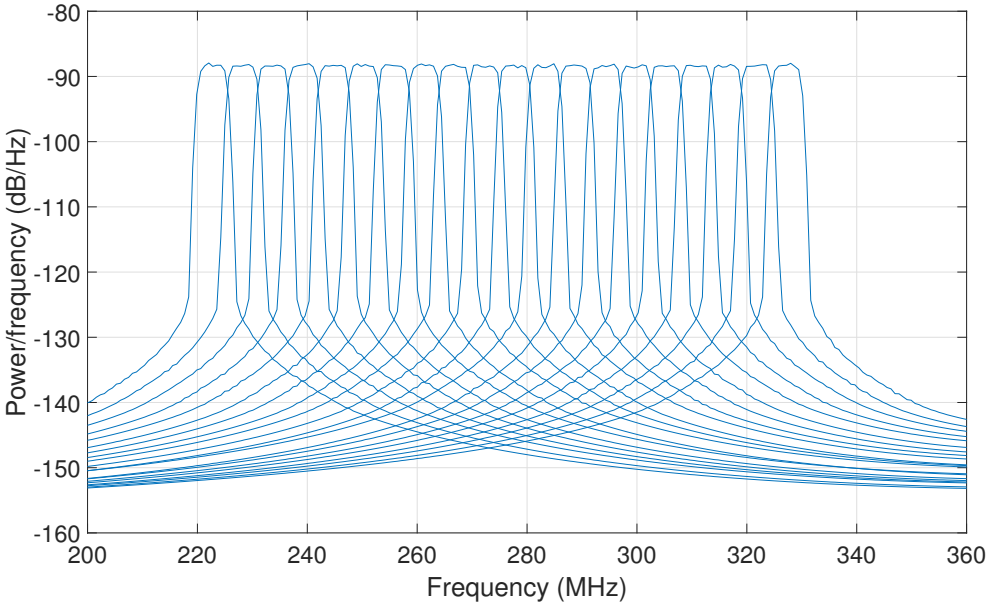


Figure 4.3: PSD of blocks after upconverting to passband, noise-free

4.3 System simulation

A simulation chain was created for evaluating the system's performance in noisy environments and jamming situations. The simulation procedure is identical regardless of which channel model or jammer is simulated. It can be summarised in 5 steps:

1. Defining performance metric to be simulated, i.e., E_b/N_0 or SIR.
2. Generating frame as figure 4.1 describes
3. Adding channel and/or jamming effects

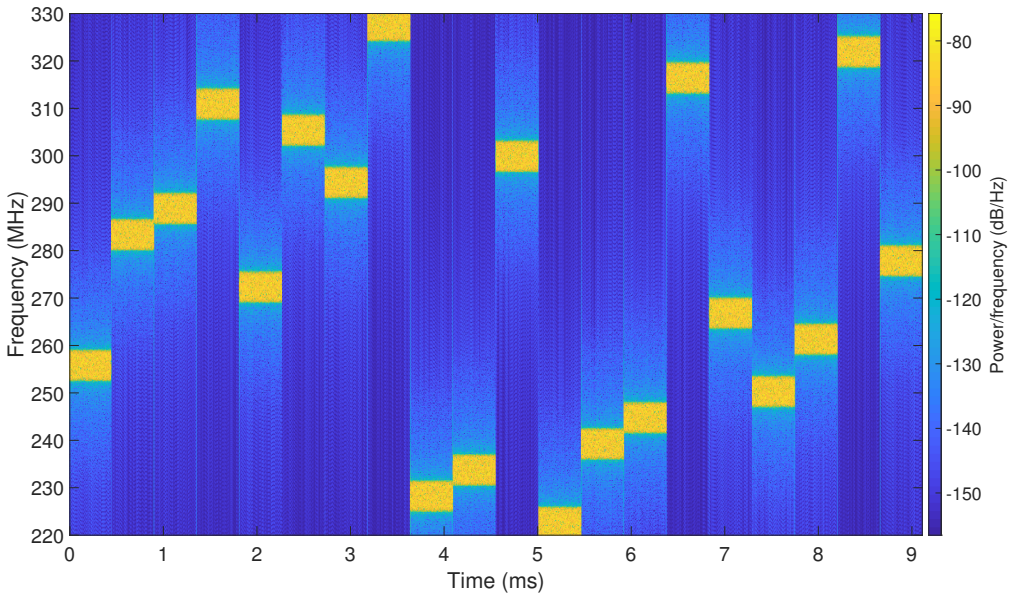


Figure 4.4: PSD of blocks versus time.

4. Decoding frame and count bit errors
5. Iterating steps 2-4 for every element in the performance metric vector defined in step 1

To ensure statistically significant results from the simulation a conditional statement on the minimum number of bit errors or the minimum number of bits transmitted is implemented. That is, for every element in the performance metric vector the simulation continues transmitting frames until the conditional statement is fulfilled. The simulation chain also has a number of soft parameters that can be adjusted to simulate different scenarios and system functions. For instance, decoding mode (soft/hard), encoding (on/off), interleaving (on/off), number of blocks per frame, carrier frequency and jammer bandwidth.

4.3.1 Channel models

A function for simulating the system in an AWGN-channel was implemented and later used both for BER vs. E_b/N_0 simulations and BER vs. SIR in various jamming scenarios. As common practice, E_b/N_0 is the used SNR-metric since it is independent of channel coding, symbol mapping etc., which allows for making fair comparisons to other systems.

To establish a definition of E_s/N_0 the definition of (transmit) power is first examined. It can be defined as

$$P = E_s \cdot R_s = E_b \cdot R_b \tag{4.6}$$

where E_s and E_b are the energy per symbol and bit, respectively. R_s and R_b denotes the data rates. It is also known that the bit rate R_b can be expressed as

$$R_b = \log_2 M \cdot R_s \tag{4.7}$$

where M is the modulation order (8-DPSK) and $\log_2(M)$ results in the number of bits per 8-DPSK symbol. Combining Equation 4.6 and 4.7 yields an expression for E_s:

$$E_s = \frac{E_b \cdot R_b}{R_s} = \frac{E_b \cdot \log_2 M \cdot R_s}{R_s} = E_b \cdot \log_2 M \tag{4.8}$$

Dividing by N_0 yields the sought for expression:

$$\frac{E_s}{N_0} = \frac{E_b}{N_0} \cdot \log_2 M \tag{4.9}$$

However, thus far the channel coding is not taken into account. The coding introduces redundant or overhead bits which in turn decreases the energy per *information* bit. And therefore, it needs to be scaled by the code rate R_c which finally yields

$$\frac{E_s}{N_0} = \frac{E_b}{N_0} \cdot \log_2 M \cdot R_c. \tag{4.10}$$

The code rate is the total channel code rate of the system calculated as

$$R_c = \frac{n_{RS}}{k_{RS}} \cdot \frac{n_{Conv}}{k_{Conv}} \tag{4.11}$$

Thus, the code rate is a measurement on the amount of redundancy introduced by the channel coding. The specified Reed-Solomon and convolutional codes yield $R_c = 0.59$. The unit of R_c can be interpreted as information bits per transmitted bit. In other words, transmitting one *information* bit requires transmitting $1/R_c = 1.69$ bits [1]. For more details on channel coding see Chapter 2.1.

The SNR per symbol, E_s/N_0, takes symbol mapping as well as channel coding into account. To convert the metric to SNR per sample, the upsampling and passband modulation also need to be accounted for. SNR per sample is simply denoted as

$$\text{SNR} = \frac{E_s}{N_0} - L_{up} + 3 \quad [dB] \tag{4.12}$$

where L_{up} is the total samples per symbol (due to RRC-filtering and interpolation). The additional 3 dB is added to the SNR since it was upconverted from (complex) baseband to (real) passband, according to Equation 4.5.

The validity of the conversion from E_b/N_0 to equivalent passband SNR was verified by comparing the resulting BER-curves from a passband simulation to a baseband simulation where no RRC-filtering, upsampling nor passband modulation were used.

[1] Naturally there is no such thing as fractions of a bit. It is simply an interpretation of the definition of code rate.

4.3.2 Channel decoding

Two different decoding approaches, referred to as *soft* and *hard* decoding, were examined. The Berlekamp-Massey algorithm was used for decoding Reed Solomon in both cases. What differentiated soft from hard decoding was the demodulation and decoding of the convolution code. In hard decoding the DPSK-symbols were demodulated to bits and thereafter decoded using the Viterbi algorithm. Contrarily, in soft decoding the DPSK-symbols were not demodulated in the conventional manner. Instead of outputting bits, a log-likehood ratio of the bits were calculated. This approach improved the bit error rate at the expense of complexity.

4.3.3 Jamming waveform generation

The jammer function was designed to be as flexible as possible in order to simulate different scenarios with the same function by altering input parameters. The available parameters that are tuneable are listed in Table 4.3.

Table 4.3: Tuneable jammer parameters

Parameter	Notation	Accepted range
Center frequency	f_J	220-330 MHz
Jammer bandwidth	B_J	0-330 MHz
Duty cycle	D_J	0-100 %
Jammer distribution	X_J	Any distribution

In simulation, the noise samples representing a jamming signal is first generated according to the specified distribution, which in most cases are white Gaussian noise (WGN) samples. Since the bandwidth of WGN is per definition equal to the bandwidth of the system, all generated jamming signals are initially wideband jammers. A bandpass Butterworth filter of order 100 is then used to adjust the jammer's bandwidth, B_J, and center frequency f_J. The PSD, calculated with a Hamming window, of a jamming signal before and after bandpass-filtering is shown in figure 4.5. A Butterworth filter was chosen due to its flat frequency response in the passband, and the filter's high order was chosen to maximise the side-lobe suppression which in turns prevents a jammer from unintentionally leaking into adjacent channels.

Before adding the jamming signal to the passband frame containing the payload, a metric describing the relationship between signal power and jamming power needs to be introduced. Signal-to-interference ratio, SIR, is used and defined as

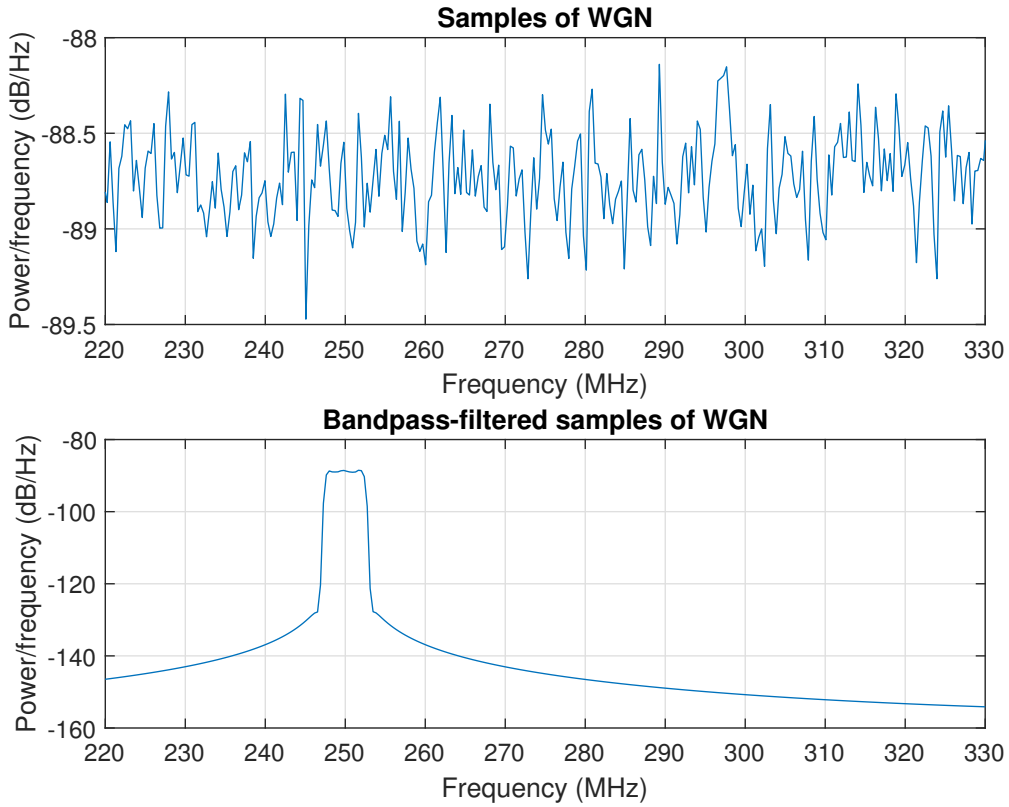


Figure 4.5: PSD of a WGN-jammer before and after bandpass filtering

$$\begin{aligned} \text{SIR} &= \frac{\overline{P}_{Signal}}{\overline{P}_{Jam}} \\ &= \frac{\frac{1}{N}\sum_{n=0}^{N-1}|x_{Sig}[n]|^2}{\frac{1}{N}\sum_{n=0}^{N-1}|x_{Jam}[n]|^2}. \end{aligned} \tag{4.13}$$

Since the signals generated in simulation are discrete and of finite length it can be seen as the sample averaged power. After calculating $\overline{P}_{Signal}$, the jamming signal is scaled by a factor according to SIR-level defined by the user. As mentioned before, it is important to assume a total finite amount of available jamming power $\overline{P}_{Jam}$ that is distributed in both time and frequency.

Figure 4.6 shows the PSD of the passband frame in two jamming situations: a partial band jammer of 4 MHz in the upper plot and a wideband jammer in the lower plot. SIR is set to 0 dB (i.e., $\overline{P}_{Signal} = \overline{P}_{Jam}$) in both cases but the PSDs in dB/Hz differ. Thus, the partial band jammer has concentrated the available power in a smaller bandwidth than the wideband jammer, yielding a larger spectral density.
Note that the center frequency of the jammer is more than 100 MHz lower than the center frequency of the signal and thus would not interfere at all. This is only for illustration purposes, to clearly visualise the spectral densities. In a real jamming

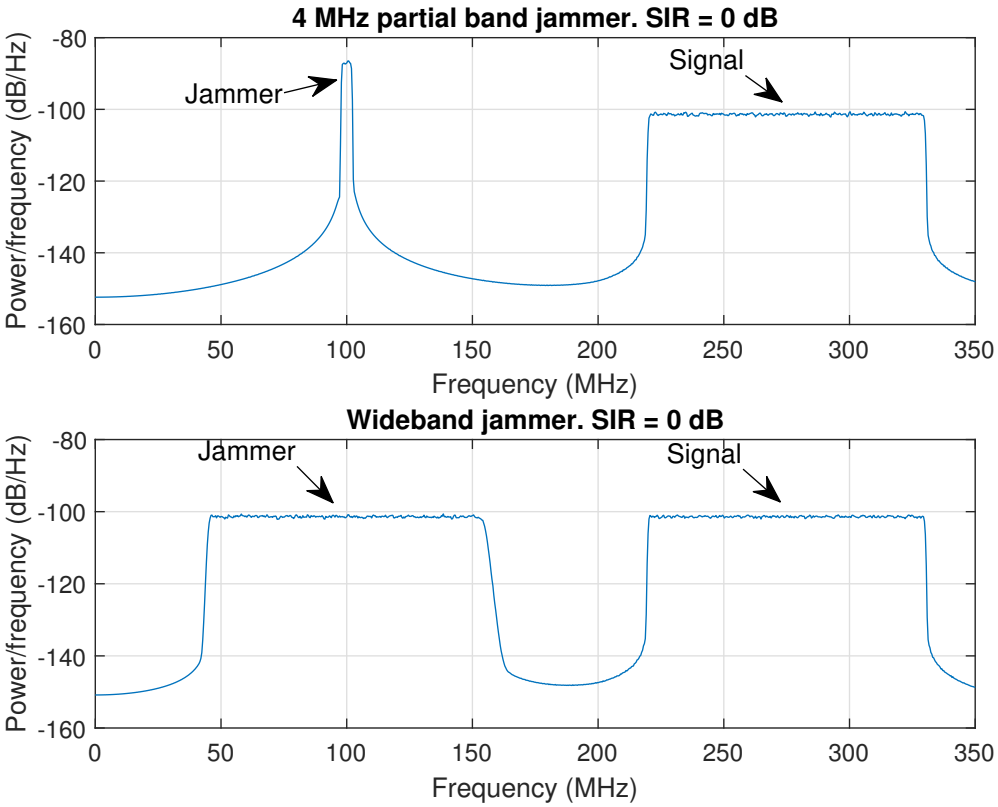


Figure 4.6: PSD of passband frames with partial band jamming (upper) and wideband jamming (lower)

situation, the center frequency of the jammer would reside inside the signal's frequency range. The PSD over time when a 4 MHz partial band jammer is located within the hop band is shown in Figure 4.7.

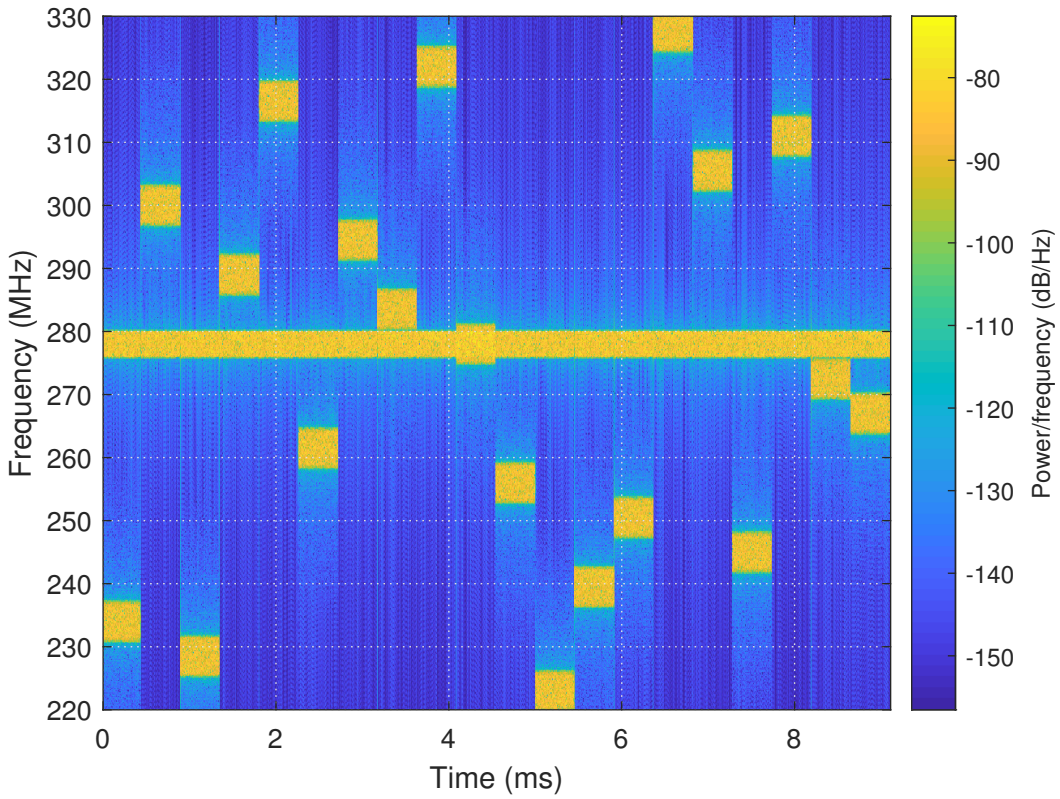


Figure 4.7: PSD over time for a partially jammed passband frame

5

System Implementation Feasibility Study

This chapter outlines the steps taken to determine the feasibility of using a particular SDR setup to implement and evaluate the aforementioned FHSS system, with the aim of aiding future work in this area.
The platform used in this study consist of a pair of USRP B200 units along with the GNU Radio DSP toolkit. Of particular interest is whether the B200 is capable of the fast frequency synthesis necessary to generate a hop rate near 2 kHz and how, if possible, FHSS functionality can be implemented using GNU Radio.

5.1 USRP B200

The Universal Software Radio Peripheral (USRP) line of SDRs from Ettus Research is a commonly used SDR testbed in research and academia. The USRP B200 used in this project can either be run as a "headless" unit by programming the built-in FPGA, or by programming the unit over USB. The latter strategy allows for interfacing with DSP software on e.g. a computer for processing quadrature signals.

5.1.1 Specifications

Some relevant parameters of the B200 units are summarised in Table 5.1. Notably, the units have a continuous tuneable range of 70 MHz-6 GHz which is enough to support extremely wide hop bands in a FHSS application. Up to 56 MHz of instantaneous bandwidth is available, meaning that the 200 ns symbol length required by the system is readily achievable. Furthermore, the unit can be equipped with a GPS-disciplined oscillator (GPSDO) which opens up the possibility to synchronise PN generators via GPS.

Table 5.1: Some features of the USRP B200.

RF coverage	70 MHz - 6 GHz
Interface	USB 3.0
Instantaneous (system) bandwidth	56 MHz
Power Output	$>$ 10 dBm

5.1.2 Tuning process

The B200 has a two-stage frequency tuning process: an RF to intermediate frequency (IF) stage provided by a local oscillator (LO), and a IF to baseband stage handled digitally (known as ”DSP tuning”) [14]. The LO has an associated re-tune delay (LO lock time) since the base LO frequency is fed into a PLL, which has a certain settling time, to produce the desired frequency. The LO lock time poses a fundamental limit to how fast the B200 is able to re-tune, or in other words how fast it can hop between frequencies. The total re-tune latency is guaranteed be be worse than the pure lock time however, mainly due to latency introduced by the host system (i.e., the computer, operating system and software) that is sending the re-tune command [15]. It has been shown that the B200 has an overall average re-tune latency of 3.31 ms (i.e., a maximum frequency hopping rate of 166 Hz) under favourable conditions using LO tuning [14].

There is however a possibility to re-tune the B200 using DSP tuning alone while keeping the LO locked at a certain center frequency. This should provide a near-instant re-tune. The caveat is that with this method the available frequency range surrounding the LO is limited by the clock rate of the unit at 61.44 MHz. (This is further limited by front-end anti-aliasing filters giving the maximum instantaneous bandwidth of 56 MHz as previously mentioned). The range is further limited by the signal bandwidth, which must ”fit” within the Nyquist boundaries of the tuneable range. As an example, if the LO is locked to 1000 MHz, the usable frequency range is 972 - 1028 MHz, meaning that any signal that we wish to sample in its entirety needs to have its frequency support within this range. With a signal bandwidth of 2 MHz, then, the minimum and maximum center frequency will be 973 and 1027 MHz, respectively. Using FHSS terms, we see that as the signal bandwidth grows, the size of the hop set (e.g. the number of available center frequencies along a contiguous set of channels with a certain bandwidth) gets smaller.

5.2 GNU Radio

GNU Radio (GR) is a DSP toolkit similar in nature to LabView or Simulink, though free to download and open-source. It can be used as a stand-alone application either through a command-line interface or through the Gnuradio Companion GUI, which allows for DSP programming in a graphical environment. It can also be embedded into other applications to provide DSP capability. GR comes equipped with many common signal processing functions, however the functionality can be extended through custom Python or C++ code. The possibility of extending GR has been explored as a part of this study.

The following sections describe briefly some details and features of GR that are of particular importance when considering GR as a platform for FHSS applications.

5.2.1 Streaming architecture

GR operates on the principle of a so-called flow graph, which contains at least one data producer (source) and one data consumer (sink), between which data ”flows” in a sample stream [16]. Between the sink and source are function ”blocks” that perform operations on the samples. GR implements a scheduler that attempts to optimise the input/output buffer usage for each block for maximum throughput; this means that the contents of these buffers can vary in size. The reason why this is important is that this strategy or architecture presents challenges for any application in which the buffer size needs to be known, e.g. a communication link working with packetised data or in other words a fixed number of samples. The FHSS link described in this work is such an application; notably the block and frame size are fixed. Another challenge posed by the scheduler and streaming architecture is that loops, e.g. feeding samples back into an ”upstream” block, is forbidden. This makes it more challenging to implement algorithms that are dependent on past input, such as PLLs, or implementing flow control, such as turning off a demodulation stage when the signal power falls below a certain threshold. Loops can, however, be implemented as part of custom function blocks.

5.2.2 Stream tags and message passing

The issue of implementing packet-based applications is partially addressed by the GR feature known as *stream tags*. Stream tags associate information with a certain sample in the sample stream, meaning that the tags move synchronously with the stream. Tags can be written and read inside blocks and actions can be taken based on their contents. As described in Section 5.3.1, this feature was used to mark the boundaries for the data blocks in the FHSS link.

Another feature implementing meta-data capabilities is the *Message passing* API. This API allows information passing between blocks, even upstream. The asynchronous nature of these messages make them less suitable for time-critical functions but simplifies many situations where information needs to be fed back (or forward) to other blocks.

5.2.3 Hardware interfacing

As mentioned, the B200 can be interacted with through an USB interface. This is done by using the Universal Hardware Interface (UHD) driver and API, which has built-in support in GR. The USRP appears as either a source (when in RX mode) or sink (when in TX mode) inside of GR, producing or consuming quadrature samples. Furthermore, with stream tags or message passing as described previously, it is possible to send commands instructing the USRP to change centre frequency, RX or TX gain, and more. Similar interfacing possibilities are available with other SDR hardware.

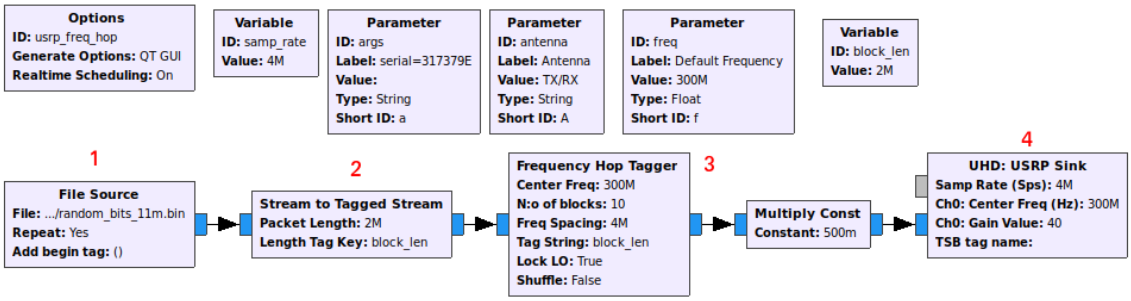


Figure 5.1: Screenshot from GNU Radio showing the transmitter.

5.3 Frequency-Hopping in GNU Radio

GR comes prepared with many common building blocks inherent to any DSP framework, such as FFT functions, filters, graphing tools etc, as well as functions specifically geared towards communication and RF software design, such as modulators, channel coding and more. However, because FH is a somewhat specialised mode of operation, especially in conjunction with specific hardware, there are no ready-made functions for this in GR (as of version 3.7.13). This necessitated the development of a custom function block to perform the frequency-hopping.

5.3.1 Transmitter overview

The theory of operation for the implemented FH transmitter is as follows:

1. Produce a sample stream of baseband samples to be frequency-hopped. This corresponds to step 0 to 4 in Figure 4.1.
2. Tag the sample stream at intervals corresponding to the start of a new block (every 20056 samples if using the previously discussed frame structure).
3. Whenever a block tag is encountered in the stream, attach a new tag with a command instructing the USRP to change carrier frequency.
4. Input the sample stream to the USRP sink which executes the jump. This corresponds to step 5 in Figure 4.1.

The GR flowchart of the very simple transmitter is seen in Figure 5.1. The sample source was in this case a binary file containing pre-generated samples instead of the full symbol generation chain. This reduced the computational load on the host system, which was simultaneously transmitting and receiving samples. The block named ”Frequency Hop Tagger” is the implemented block. The ”Multiply Const” block between 3 and 4 is present to lower the amplitude of the samples going into the USRP to avoid Power Amplifier saturation.

5.3.2 Block implementation and test

The function or block attaching the instructions needs to have the next frequency in the hop pattern available whenever a block tag is encountered. Either the block in

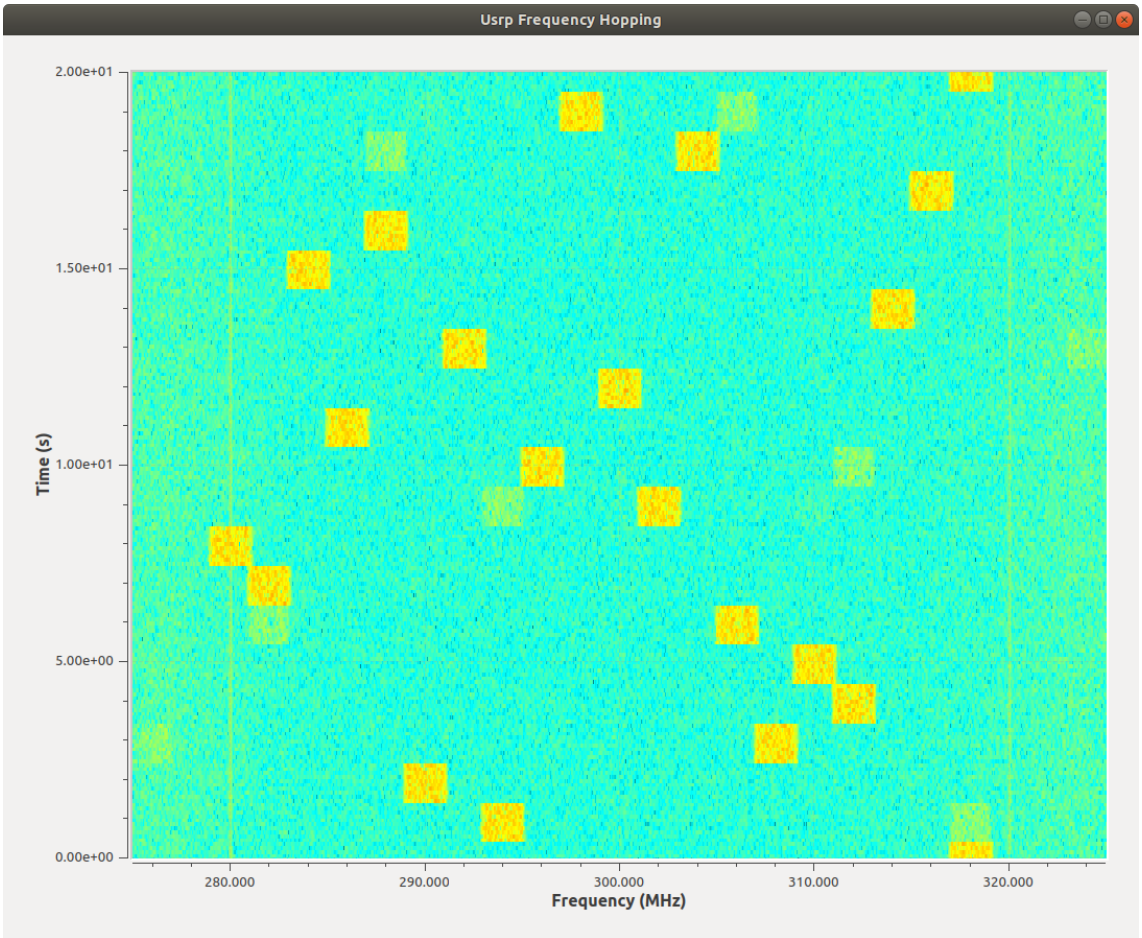


Figure 5.2: Screenshot of a waterfall intensity plot from GNU Radio. The frequency-hopped data blocks are marked in yellow. The observed time window is 20 seconds. Increasing time goes from bottom to top and increasing frequency left to right.

question takes care of generating the hop set or pattern, or it can receive data from an external block through the message API. For simplicity the former was chosen for the implemented block. The block was implemented using Python, for code listing see appendix A. The block can issue commands either to automatically tune both the LO and DSP (the default mode) or to just tune the DSP. In the former case this means setting the absolute frequency of the USRP and in the latter a relative frequency to the LO. Figure 5.2 shows a waterfall plot of the generated signal as received by the second B200 unit. The hop rate is 1 Hz, the signal bandwidth is 2 MHz and the total hop bandwidth is 40 MHz (20 hops per frame). The faint signals alongside some of the blocks are likely aliased copies of the blocks.

6
Results

6.1 System performance in AWGN

The standard performance metric, bit errors caused by additive white Gaussian noise, is shown in Figure 6.1. The system was simulated with all components activated, including channel coding, passband modulation and frequency hopping. Two different decoding algorithms were used; standard hard decoding and the more computationally heavy soft decoding. A coding gain of 4 dB was achieved for hard decoding whilst soft decoding achieved around 3 dB of additional improvement. The gains are relative to the BER-curve for uncoded 8-DPSK modulation, drawn in blue. These results suggest that an E_b/N_0 around 12 dB is required to achieve the target BER of 10^{-5} when using hard decoding. The rest of the results presented in this chapter are based on the hard decoding method.

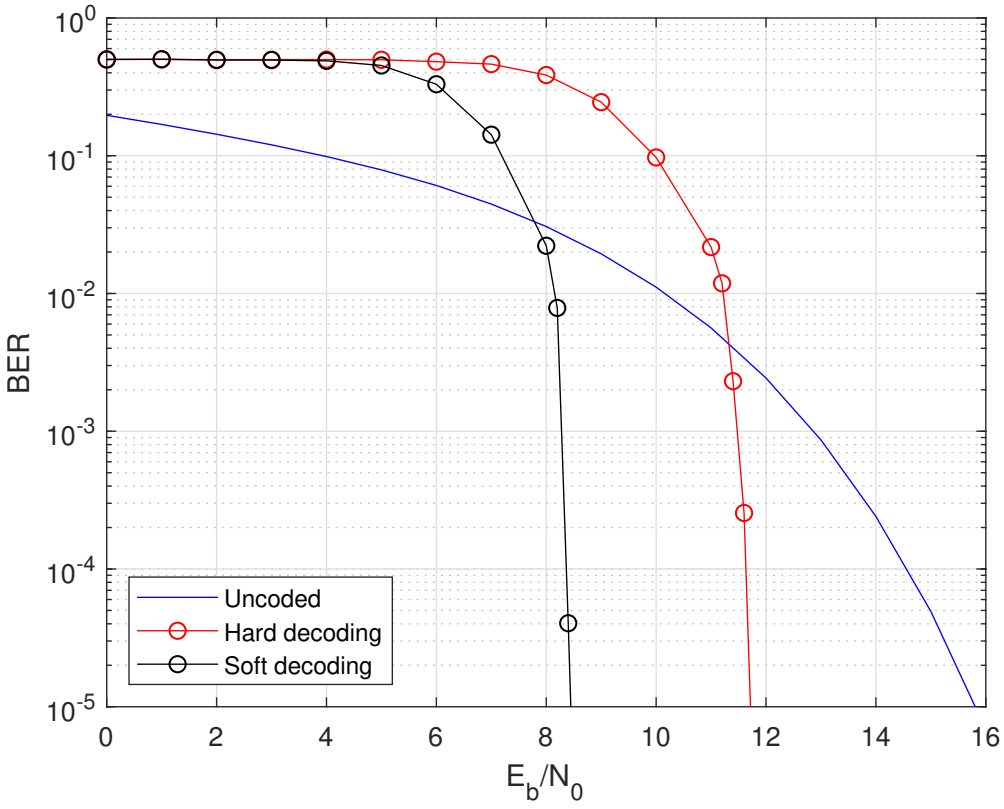


Figure 6.1: BER versus E_b/N_0

6.2 Wideband jamming

The BER for when a wideband jammer, covering the systems total hop bandwidth, is shown in Figure 6.2. The E_b/N_0, due to AWGN-modelling is fixed whilst SIR is incremented along the x-axis. The BER-saturation level (e.g. when the curve settles) is directly given by the corresponding BER in Figure 6.1. The red curve, with $E_b/N_0 = 11.6$ dB, saturates at $\approx 3 \cdot 10^{-4}$ whilst the black curve, with $E_b/N_0 = 11.7$ dB, saturates at $\approx 3 \cdot 10^{-5}$. The saturation region can be seen as the region where the SIR is sufficiently large so that the system is no longer affected by the jammer.

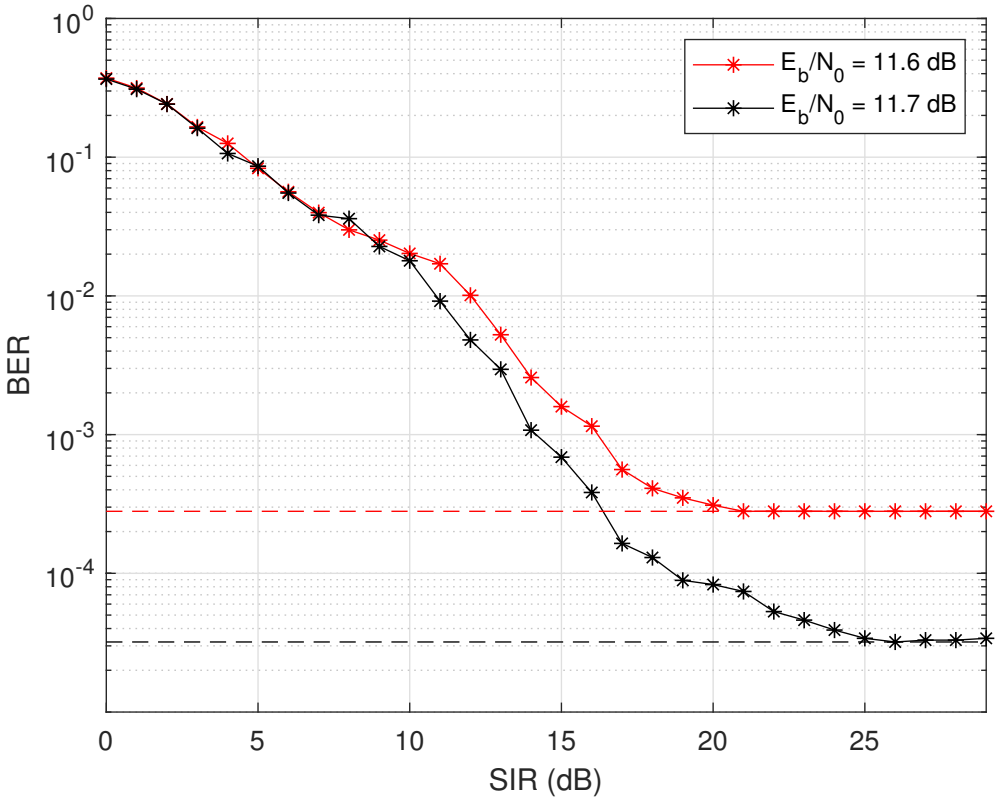


Figure 6.2: BER versus SIR (wideband jammer)

6.3 Partial band jamming

The BER for when a 4 MHz partial band jammer is present, covering one of the 20 hop channels, is shown in Figure 6.3. For low SIR, the red region of the figure, the BER remains constant at $2.5 \cdot 10^{-2}$ (equivalent to 2.5%) which indicates one out of 20 blocks are decoded erroneously. Compare that to the BER for the wideband jammer in Figure 6.2 where the entire frame is decoded erroneously. Furthermore, the high-SIR region marked with green in the figure begins later than that of a wideband jammer. In other words, a larger SIR is required to enter the region where the BER only depends on AWGN.

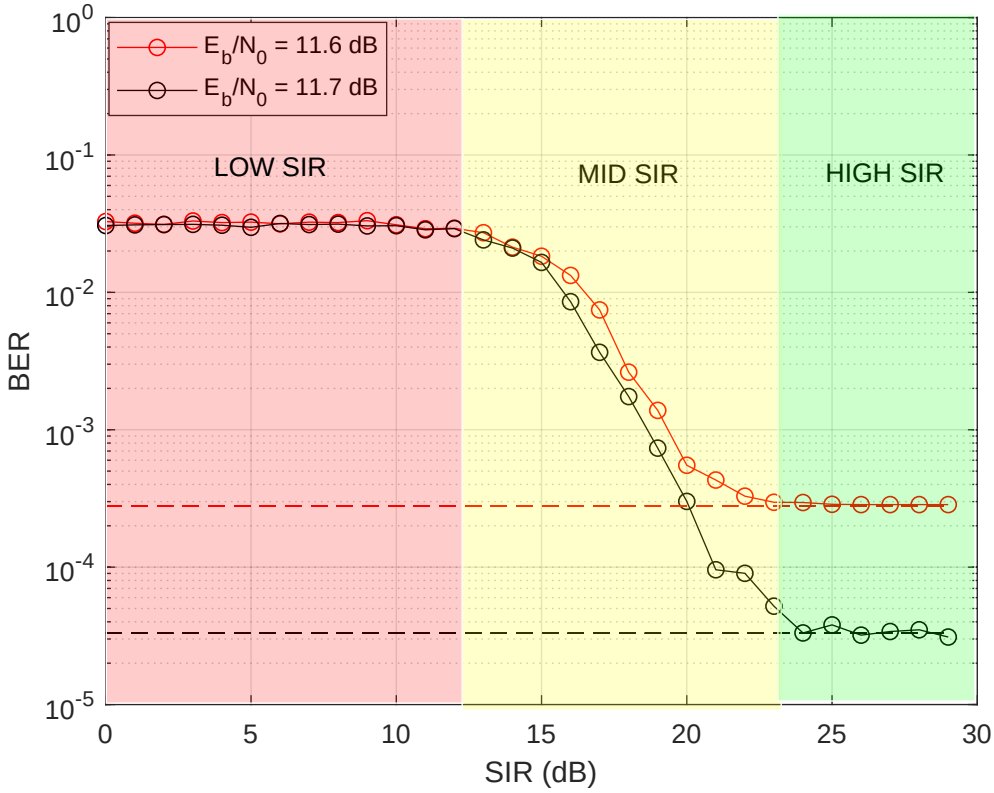


Figure 6.3: BER versus SIR (partial band jammer)

To illustrate one of the advantages of using FHSS, two partial band jamming scenarios were simulated. An identical partial band jammer of 4 MHz was applied to the system when FHSS with 20 hops was activated and when all blocks were modulated onto the same carrier frequency, i.e., single carrier. The resulting BER-curves are shown in Figure 6.4. An 8 dB gain from using FHSS instead of single carrier is seen.

Finally, the effect of doubling the number of blocks (frequency hops) in a partial band jamming scenario was simulated. The BER-curves for $N_{blocks} = 20$ and $N_{blocks} = 40$ are shown in Figure 6.5. A 2 dB gain from using twice as many blocks can be observed.

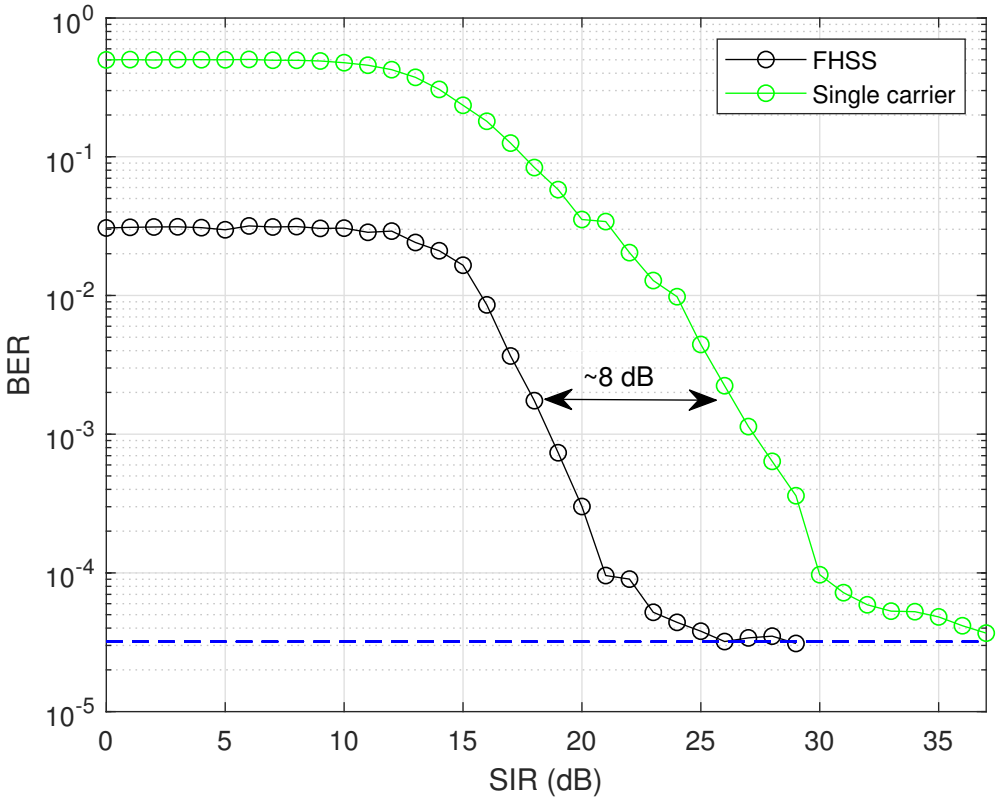


Figure 6.4: BER versus SIR (partial band jammer). $E_b/N_0 = 11.7$ dB.

6.4 Jammer comparison

The simulation results from partial band, tone and wideband jamming are presented in Figure 6.6. As expected, the wideband jammer is the least efficient of the three. The tone and partial band jammer appears to follow the same curve and are approximately 5 dB shifted from the wideband jammer in the mid-SIR region.

Furthermore, a 4 MHz partial band jammer on a single carrier system and a wideband jammer on a FHSS-system was simulated and compared in Figure 6.7. The simulation yielded a gain of 13 dB. The meaning and interpretation of this gain will be explained and clarified in Chapter 7.

6.5 Discrete block jamming on symbol level

This result shows how many percent of symbols of a block the channel coding can correct for, before the block is 'lost'. The impact of the number of hops per frame, N_{blocks}, was investigated by simulating the degradation of BER due to a discrete jammer. In this context, discrete refers to discrete time since no pulse shaping nor passband modulation was used. Thus, the degradation of BER depends only on the channel coding, symbol mapping and the number of blocks (or hops) per frame. Figure 6.8 shows the resulting BER curves for 20, 40 and 50 blocks per frame. The

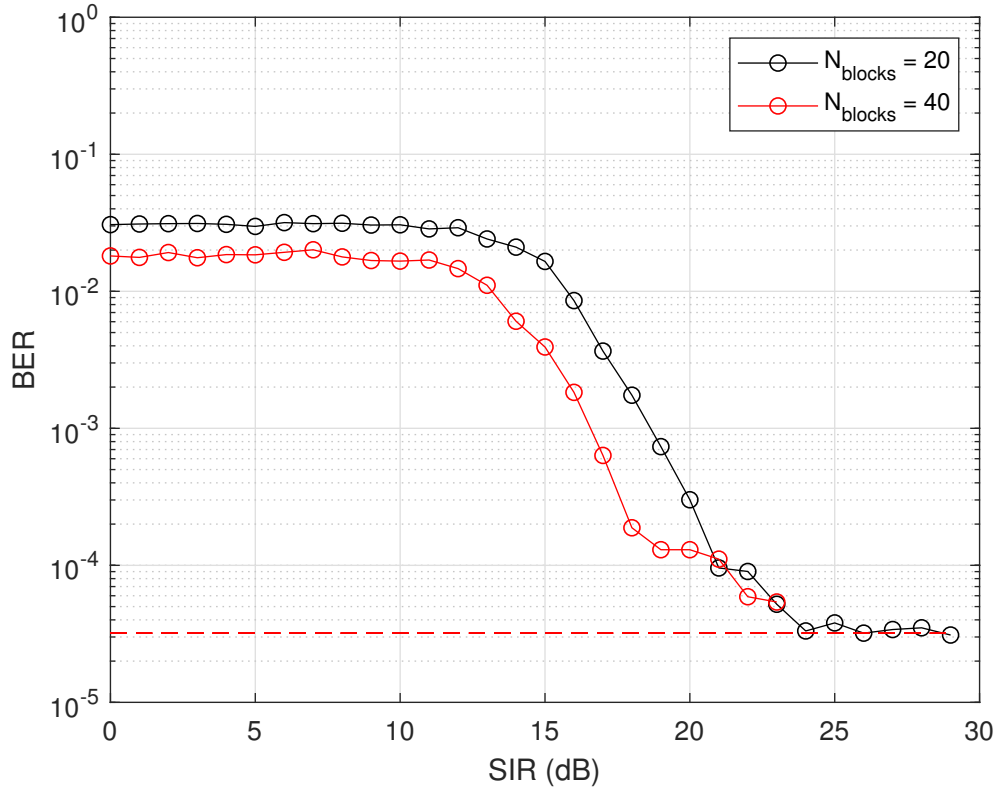


Figure 6.5: Improvement of BER due to increased number of blocks per frame. Partial band jammer present. $E_b/N_0 = 11.7$ dB.

BER is calculated on a frame level and is plotted against the percentage of a block that is jammed. The symbols are consecutively jammed meaning from the beginning of the block to a certain percentage of the total block length. This can be seen as analogous to a pulsed partial band jammer in the time continuous case. Also note that E_b/N_0 is set to be very large in this simulation to exclude the degradation of BER due to noise and instead only focus on the jammer.

The threshold, where the BER goes from zero to non-zero, indicates the upper limit on how many symbols in one block can be received erroneously and still decode the entire frame error free. For 20, 40 and 50 blocks, the limit is 36%, 78% and 96%, respectively. Note how the BER for $N_{blocks} = 20$ saturates at BER=0.025 when the majority of the block is jammed. This indicates that the entire block has been lost and thus the BER on the frame level is given by $1/(2N_{blocks})$.

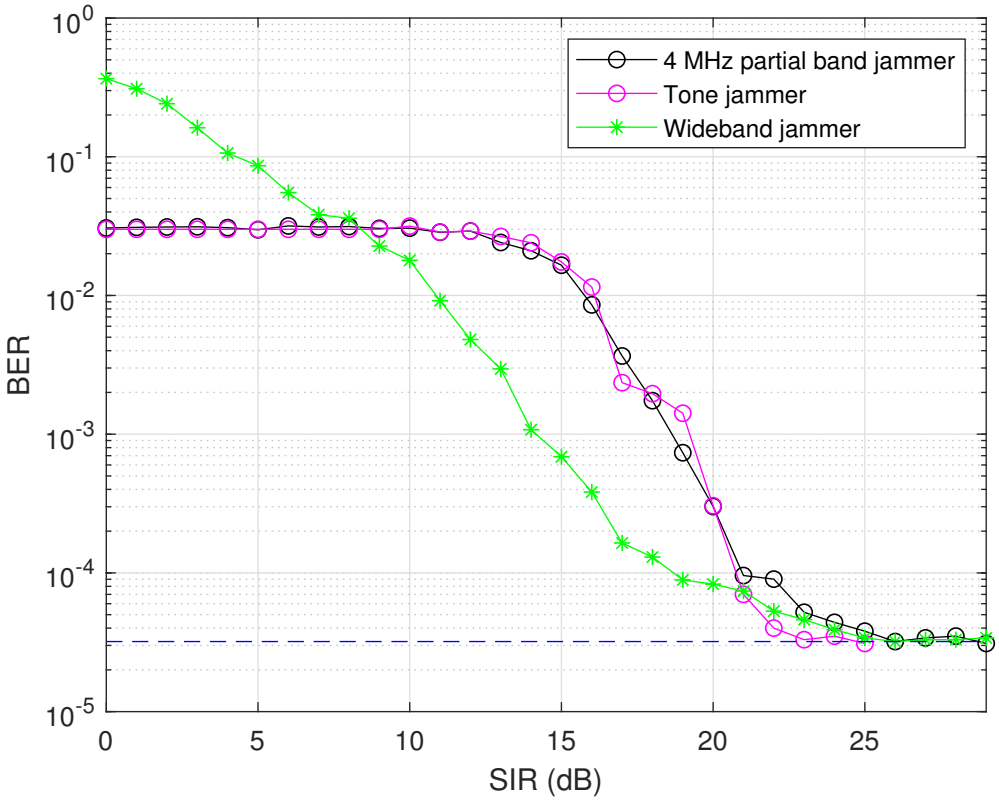


Figure 6.6: Comparing partial band, wideband and tone jammer. $E_b/N_0 = 11.7$ dB

6.6 USRP hop rate

Figure 6.9 shows the received frequency-hopped waveform as observed by the receiving B200 unit. The bandwidth per block is 2 MHz and the total hop bandwidth is 40 MHz. The hop rate is 10 Hz, which was the maximum achieved hop rate without missed hops. The hop set was kept un-randomized to aid in identifying a missed hop, which is a symptom that manifests when the USRP cannot keep up with the requests to change carrier frequency.

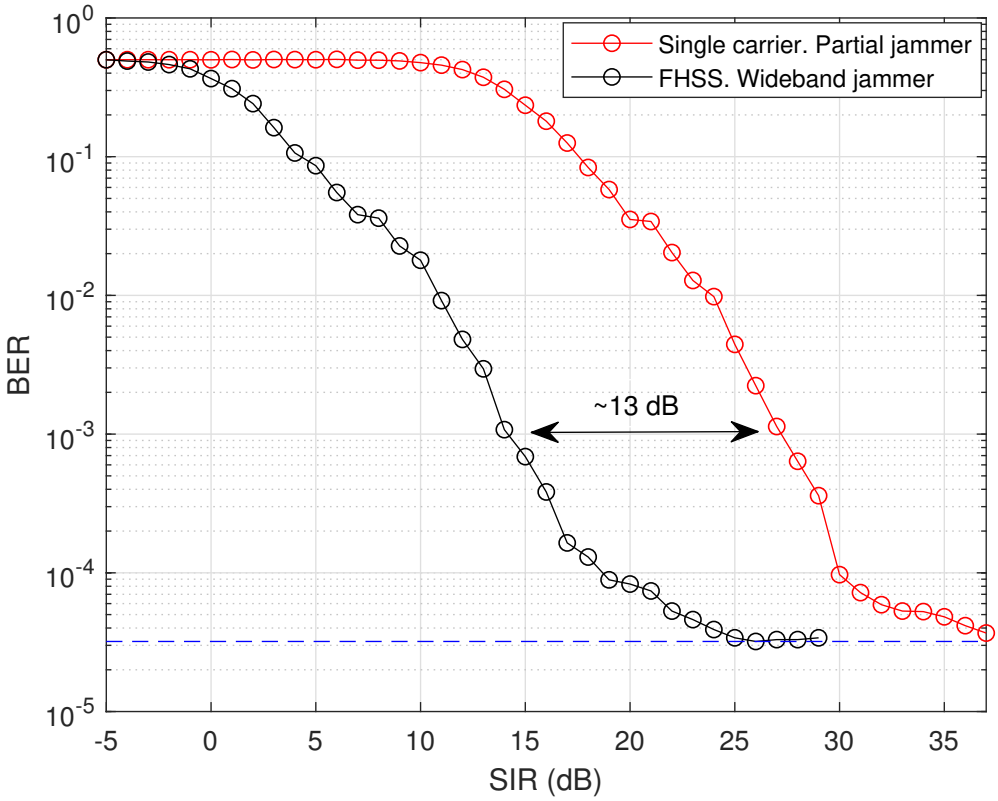


Figure 6.7: Single carrier with partial band jammer versus FHSS with wideband jammer.

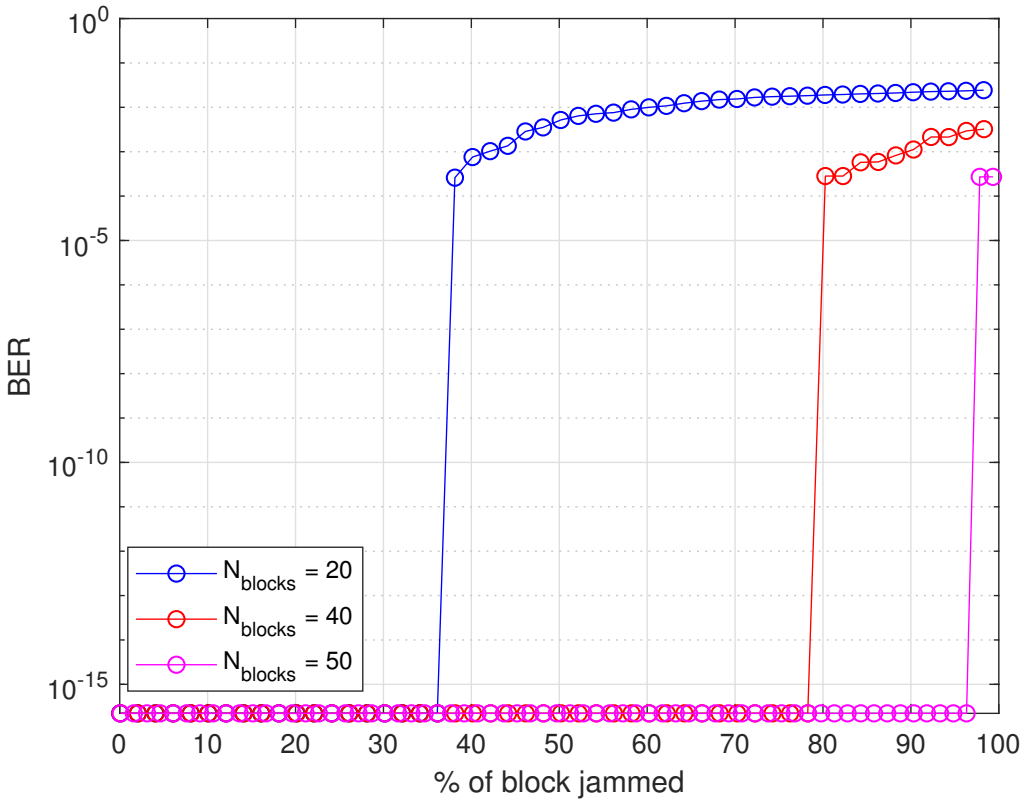


Figure 6.8: BER versus the percentage of symbols in a block being received erroneously

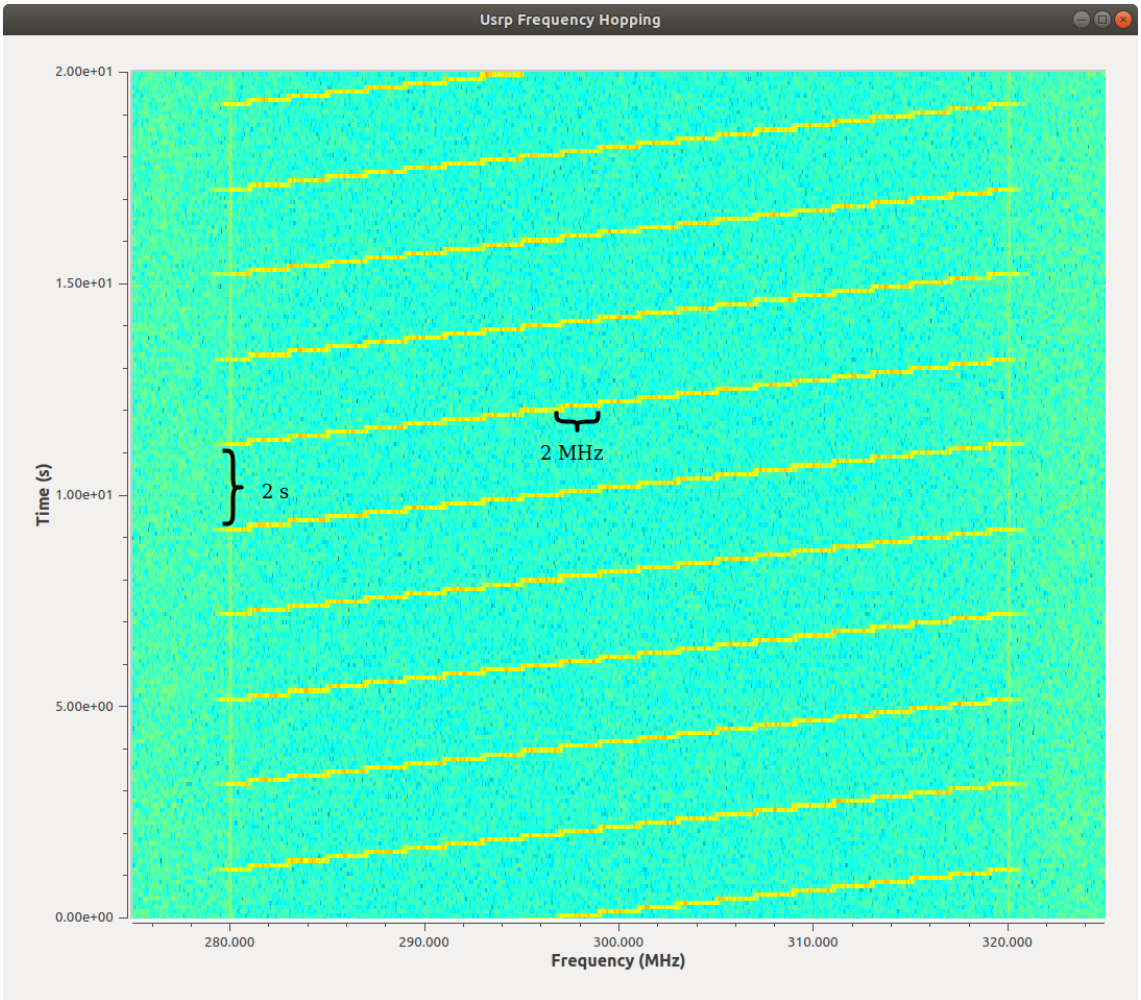


Figure 6.9: Screenshot of waterfall intensity plot from GNU Radio. The hopped data blocks are in yellow. This shows the maximum achieved hop rate with the implemented transmitter.

7
Discussion

7.1 Soft and hard decision decoding

Even though soft decoding outperforms hard decoding with around 3 dB, as shown in Figure 6.1, the former was excluded from further simulation at an early stage of the project. This was decided based on the fact that the soft decoder, implemented as an LLR-approximating algorithm, is far more complex and computationally heavy than the hard decoder using Trellis diagrams. Also, because of its complexity it would not be feasible to implement on a real-time data link without incurring undesirable latency, given our hardware constraints. For subsequent analyses, an E_b/N_0 level of 11.6 - 11.7 dB was used. While this does not correspond exactly to the target BER of 10^{-5} it is quite close, and enabled faster simulation times.

7.2 Optimal anti-jamming capability

In Figure 6.7 we see the processing gain (about 13 dB SIR) that arises from the use of an FHSS system with 20 channels, as compared to a single-carrier link. The scenario playing out in this figure is as follows: an adversary is monitoring the spectrum and intercepts our single-carrier link. This poses no challenge to the adversary, who can design a jammer to efficiently jam our communications by using for instance a partial-band jammer. However, if we switch to an FHSS system the adversary is forced to re-evaluate their strategy. If we have designed the FHSS system efficiently, using high hop rates (to avoid repeat-back jamming) and utilising spectral notching (i.e., removing the jammed channel from the hop set), we can essentially eliminate the interference caused by the partial-band jammer. This follows since the link would remove the jammed hop channel from its FH sequence and therefore be completely unaffected by the jammer in the following transmitted frames. The only remaining strategy for the adversary that guarantees damage to our communications is to revert to wideband jamming. This is what we consider to be an FHSS system with optimal anti-jamming capabilities.

However, the proposed link does not have the spectral notching capability, nor is its hop rate particularly fast. For that reason it is doubtful that we would see this anti-jamming performance from the proposed link in a real-world scenario. In Figure 6.4 we see a comparison with the same single-carrier link, but this time our FHSS system is also being partially jammed. Because we lack the spectral notching capability, we have no choice but to let the jammed channel remain in our hop set.

This yields a performance gain of 8 dB SIR compared to the single carrier link. This gain is 5 dB lower than the gain achieved from the optimal or baseline case in Figure 6.7. Considering the definition of an effective anti-jam system as noted in Section 3.2.1, we can thus conclude that this is not an effective anti-jam system.

7.3 Anti-jamming capability of the link and how to improve it

As we have discussed, partial-band- and tone jamming represents a tough challenge for any FHSS system. We see in Figure 6.6 that in our particular case these jammers impose a 6 dB SIR penalty over wideband jamming in the mid-SIR region. How can we improve this result, despite the lack of ability to remove a jammed channel?

There are a number of factors that come into play when considering the ability to reject partial-band and tone jamming; most importantly the number of bits that are exposed to the jammed channel, the number of blocks per frame and the error-correcting capability of the channel code. In this system, the number of bits that are transmitted per channel is governed by the blocks that make up a frame. Ideally, to reject partial-band jamming the system should have the ability to correct for the loss of an entire block. As shown in Figure 6.8, this is hardly the case for our system. At 20 blocks per frame we can lose about 36% of a single block before the BER becomes non-zero. Doubling the number of blocks dramatically increases the jamming tolerance; Figure 6.5 shows that this results in about 2 dB SIR of improvement. At 50 blocks we can almost handle the loss of an entire block without a single bit error in the frame.

While it is easy to grasp that halving the block size exposes half as many bits to the jammer, the exact relation between the percentage of jammed block and error correcting capability is not trivial; it is an interplay between the interleaver, whose job it is to spread burst errors over the frame, and the Reed-Solomon encoder. With the particular configuration of the frame (20 blocks), bits equivalent to 2.75 RS codewords are placed in each block. 36% corresponds to exactly one RS codeword in this case. In the 40-block case the same figure is 72%, while the error tolerance shown by Figure 6.8 is about 78%. This non-linear relation can perhaps be attributed to the slight error-correcting capability of the convolutional code. Similarly, while it is quite intuitive that the chosen approach to interleaving is beneficial in partial-band jamming, it is not necessarily better performing than a more systematic interleaver. Additionally, the placement of the interleaver, between the convolutional encoder and RS encoder, may not be ideal. This may especially be the case if one considers more complex channel models than what has been investigated in this study.

7.4 Increasing hop bandwidth versus decreasing channel bandwidth

As previously discussed, increasing the number of blocks increases the resistance to partial-band and tone jammers. This can be done in two ways: either the channel bandwidth is decreased whilst the hop bandwidth is kept fixed, or the hop bandwidth is increased whilst the channel bandwidth is kept fixed. However, as noted in Section 3.1.2.1, this is not true if the jammer has a non-zero bandwidth B_J since the jammer would eventually interfere with more than one channel. The case of tone jamming, which has a bandwidth near zero, is perhaps an exception to this rule. We observe in Figure 6.6 that tone jamming has a near-enough identical impact on the system as a partial band jammer. In other words, the tone jammer's impact is "independent" of its bandwidth. Therefore, to combat the effect of the tone jammer, the number of hop frequencies (number of blocks) can be increased, which reduces the number of exposed bits.

A situation that was not simulated is the case of multi-tone jamming; multiple tones distributed over many channels. This would however be a trivial extension of the single-tone case, as the BER would increase with the number of blocked channels. The same strategy would apply as above; increasing the number of blocks.

Ultimately, the choice of whether to increase the total hop bandwidth or reduce the channel bandwidth is a question of the available resources as well as the manner in which the system is being jammed. These changes would, in turn, also have consequences. Reducing the channel bandwidth would lead to less channel coding and/or higher modulation order being needed. Which in turn means that a higher E_b/N_0 would be required to achieve the same BER.

7.5 Bit rate versus anti-jam capability

Increasing the number of hop channels improves the anti-jamming capabilities but it comes with a price. In a real system, every frequency hop introduces a delay (idle time and switching time) caused by the frequency synthesizer circuitry. Thus, there is a trade-off between bit rate and anti-jamming capabilities.

There is also a trade-off between the channel bandwidth and anti-jamming capabilities. A smaller channel bandwidth enables a larger hop set in a given hop bandwidth, but the bit rate is reduced. The proposed link specifies a bit rate of around 8 Mbit/s which is to be considered a high-capacity FHSS-link. If this bit rate is to be maintained while also maintaining a high processing gain or anti-jam capability, the hop bandwidth requirement become unrealistically large. As an example, if a processing gain of 30 dB would be specified for this link, a total hop band of 5.5 GHz (5.5 MHz/channel $\times$ 1000 channels) would be required to maintain the same bit rate.

7.6 GNU Radio and USRP feasibility study

As noted in the results, Figure 6.9 shows the maximum achieved hop rate with the implemented transmitter, which is 10 Hz. This is a disappointing result, especially considering the already low benchmark at 166 Hz as cited in Section 5.1.2. There are a few possible explanations as to why a better result was not achieved. Firstly, the benchmark result was reached by measuring the time for a re-tune command to be sent to and subsequently acknowledged by the USRP, directly interfacing with the UHD API. This as opposed to using the stream tag construct. Because the stream tags adds a few layers of abstraction between the USRP and the re-tune command, it is possible that this incurs a lot of latency in the signal chain. This would then void the use of stream tags as a valid method for constructing a FHSS-transmitter in GR, and instead the API should be used directly.

A second possible (trivial) explanation is ”operator error”, or in other words a mistake in the code. As explained in Section 5.3.2, the USRP accepts commands to re-tune either the entire frequency chain, or just the DSP frequency. However, during development it was found that the command to change DSP frequency was not accepted by the USRP unless a command to set the LO frequency was issued in conjunction. If the issued LO request was set to the same value, this was thought to not re-tune the LO. It is possible however that this command does re-tune the LO, to the same frequency. This would then render the fast DSP tuning pointless. It is unclear whether this is due to a mistake in the code, or a bug in the framework.

8

Conclusion

The proposed FHSS system has been simulated in various jamming scenarios and the feasibility of implementing such a system on an SDR platform has been investigated. The simulation revealed that the overall anti-jamming capability for the given link specification in various jamming environments was unsatisfactory. We conclude that the large channel bandwidth, the high bit rate, the high number of bits per block and ultimately the frame architecture were the reasons for this. However, an improvement over a single-carrier system was still observed. Furthermore, we conclude that our method of implementing an FHSS system was incompatible with the proposed SDR platform. Frequency-hopping was achieved but the specified hop rate at 1991 hops/s was not reached. However, this is likely a software issue and more work is needed in order to properly illustrate the capabilities of the platform.

Our recommendation for an improved link is to increase the number of blocks per frame to ensure that an entire block can be lost without degrading BER performance. This can be achieved either by decreasing the channel bandwidth or increasing the hop bandwidth. Also, a more comprehensive evaluation in terms of jamming scenarios is likely required to verify the system's anti-jamming performance. In addition, link performance in fading channels should be evaluated in order to give a more conclusive picture of the system in a real environment. This should include more rigorous investigation of some system attributes such as the interleaver.

Bibliography

[1] *Spread Spectrum Communications Handbook*. Ser. McGraw-Hill's AccessEngineering. McGraw-Hill, 2002, ISBN: 0-07-144947-7. [Online]. Available: http://proxy.lib.chalmers.se/login?url=http://search.ebscohost.com.proxy.lib.chalmers.se/login.aspx?direct=true&db=edsace&AN=edsace.ccn00214071&site=eds-live&scope=site.

[2] A. Stoica, D. Militaru, D. Moldo, and A. Popa, "Tactical data link - from link 1 to link 22.", *Scientific Bulletin 'Mircea cel Batran' Naval Academy*, vol. 19, no. 2, p. 317, 2016, ISSN: 23928956. [Online]. Available: http://proxy.lib.chalmers.se/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=120573745&site=eds-live&scope=site.

[3] Strålsäkerhetsmyndigheten. (2017). Referensvärden, [Online]. Available: https://www.stralsakerhetsmyndigheten.se/omraden/magnetfalt-och-tradlos-teknik/referensvarden/ (visited on 02/06/2019).

[4] R. G. Gallager. (2008). Peter Elias 1923-2001, A Biographical Memoir, National Academy of Sceiences, [Online]. Available: http://www.nasonline.org/publications/biographical-memoirs/memoir-pdfs/elias-peter.pdf (visited on 03/06/2019).

[5] A. Goldsmith, *Wireless Communications*. Cambridge University Press, 2005, ISBN: 978-0-52183-716-3.

[6] I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields", *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, pp. 300–304, 1960.

[7] E. A. L. John R. Barry and D. G. Messerschmitt, *Digital Communications*. Springer, Boston, MA, 2004, ch. Pulse Amplitude Modulation.

[8] D. Torrieri, *Principles of Spread-Spectrum Communication Systems. [electronic resource]*. Springer International Publishing, 2018, ISBN: 9783319705699. [Online]. Available: http://proxy.lib.chalmers.se/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=cat06296a&AN=clc.b2535682&site=eds-live&scope=site.

[9] G. F. Elmasry, *Tactical Wireless Communications and Networks: Design Concepts and Challenges*. 2012, ch. JTDIS Pulse Structures.

[10] R. N. Mutagi, "Pseudo noise sequences for engineers", *Electronics Communication Engineering Journal*, vol. 8, no. 2, pp. 79–87, Apr. 1996, ISSN: 0954-0695. DOI: 10.1049/ecej:19960205.

[11] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, Jan. 1998, ISSN: 1049-3301. DOI:

10.1145/272991.272995. [Online]. Available: https://doi.org/10.1145/272991.272995.

[12] M. Tomlinson, C. J. Tjhai, M. A. Ambroze, M. Ahmed, and M. Jibril, "Historical convolutional codes as tail-biting block codes", in *Error-Correction Coding and Decoding: Bounds, Codes, Decoders, Analysis and Applications*. Cham: Springer International Publishing, 2017, pp. 289–298, ISBN: 978-3-319-51103-0. DOI: 10.1007/978-3-319-51103-0_10. [Online]. Available: https://doi.org/10.1007/978-3-319-51103-0_10.

[13] V. K. Garg and Y. Wang, *The Electrical Engineering Handbook*. Lucent Technologies, 2005, ch. Transmission of Digital Signals, pp. 965–970, ISBN: 0-12-170960-4.

[14] R. Bell, "Maximum supported hopping rate measurements using the universal software radio peripheral software defined radio", *Proceedings of the GNU Radio Conference*, vol. 1, no. 1, 2016. [Online]. Available: https://pubs.gnuradio.org/index.php/grcon/article/view/2.

[15] N. B. Truong, Y.-J. Suh, and C. Yu, "Latency analysis in GNU radio/USRP-based software radio platforms.", *MILCOM 2013 - 2013 IEEE Military Communications Conference*, p. 305, 2013, ISSN: 9780769551241. [Online]. Available: http://proxy.lib.chalmers.se/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=95079536&site=eds-live&scope=site.

[16] The GNU Radio Foundation. (2019). GNU Radio Manual and C++ API Reference, [Online]. Available: https://www.gnuradio.org/doc/doxygen/page_operating_fg.html (visited on 10/25/2019).

A

Appendix 1: GNU Radio custom block code

```
1  import numpy
2  import pmt
3  import random
4  from gnuradio import uhd
5  from gnuradio import gr
6
7  class frequency_hopping(gr.sync_block):
8      """
9      Apply tags for frequency-hopping to the sample
10     stream based on the detection of tag_name
11     in the stream.
12     """
13     def __init__(self, center_freq, n_hops, freq_spacing,
           tag_name, shuffle, lock_lo):
14         gr.sync_block.__init__(self,
15             name="frequency_hopping",
16             in_sig=[numpy.complex64],
17             out_sig=[numpy.complex64])
18
19         self.center_freq = center_freq
20         self.n_hops = n_hops
21         self.freq_spacing = freq_spacing
22         self.tag_name = pmt.intern(tag_name)
23         self.lock_lo = lock_lo
24         self.count = 0
25         self.hop_set = []
26         # Prepare tag
27         self.freq_tag = gr.tag_t()
28
29         if(self.lock_lo):
30             self.freq_tag.key = pmt.intern('tx_command')
31             self.command = pmt.make_dict()
32             start_freq = n_hops/2 * freq_spacing
33             for n in range(self.n_hops):
```

```
34                  self.hop_set.append(start_freq - n *
                        freq_spacing)
35          else:
36              self.freq_tag.key = pmt.intern('tx_freq')
37              start_freq = center_freq - n_hops/2 *
                    freq_spacing
38              for n in range(self.n_hops):
39                  self.hop_set.append(start_freq + n *
                        freq_spacing)
40          # Randomize hop set?
41          if shuffle:
42              random.shuffle(self.hop_set)
43
44      def work(self, input_items, output_items):
45          in0 = input_items[0]
46          out = output_items[0]
47
48          rel_start = 0
49          rel_end = len(in0)
50
51          tags = self.get_tags_in_window(0, rel_start, rel_end
                , self.tag_name)
52
53          for tag in tags:
54              self.freq_tag.offset = tag.offset
55              if(self.lock_lo):
56                  self.command = pmt.dict_add(self.command,
                        pmt.intern('dsp_freq'), pmt.from_double(
                        self.hop_set[self.count]))
57                  self.freq_tag.value = self.command
58              else:
59                  self.freq_tag.value = pmt.from_float(self.
                        hop_set[self.count])
60
61              self.add_item_tag(0, self.freq_tag)
62              self.count += 1
63
64          if (self.count == self.n_hops):
65              self.count = 0
66
67          out[:] = in0
68          return len(output_items[0])
```