Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Anders Berg Sæther

Investigation of sliding window DFT (sDFT) application for radio

Master's thesis in Electronics Systems Design and Innovation
Supervisor: Lars Magne Lundheim
Co-supervisor: Svein Rypdal Henninen & Morten Paulsen
June 2023

NTNU
Norwegian University of Science and Technology

NORDIC SEMICONDUCTOR

Anders Berg Sæther

Investigation of sliding window DFT (sDFT) application for radio

Master's thesis in Electronics Systems Design and Innovation
Supervisor: Lars Magne Lundheim
Co-supervisor: Svein Rypdal Henninen & Morten Paulsen
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

NTNU
Norwegian University of
Science and Technology

Abstract

This thesis documents a SystemVerilog implementation of a Bluetooth Low Energy (BLE) demodulator and preamble detector, which are based on the sliding window discrete Fourier transform (sDFT). Both systems use matched filters, where the filtering is performed using multiplications in frequency domain. Furthermore, these systems are compared to traditional equivalents of the same demodulator and preamble detector, which use linear convolution to perform the filtering in time domain. The comparison is done both in terms of performance and implementation cost.

It was found that the main advantage of using the sDFT for filtering is that the frequency bins can be computed independently. This means that we do not have to use the entire spectrum to perform filtering, and can therefore save hardware resources by only using the most important frequency bins, with only a small loss in performance. More specifically, the implemented sDFT demodulator can use three out of 16 frequency bins to perform filtering, which reduces the amount of NAND2 equivalents required to synthesize the system by 45% compared to the time domain demodulator. This also increases the bit error rate (BER) by 20% at the most affected value of $\frac{E_b}{N_0}$, which corresponds to a 0.29 dB loss in $\frac{E_b}{N_0}$. Similarly, the implemented sDFT preamble detector can use seven out of 128 frequency bins, which reduces the amount of NAND2 equivalents by 67% compared to the time domain preamble detector. The sDFT implementation also increases number of undetected preambles by 7.8% at the most affected value of $\frac{E_b}{N_0}$, which corresponds to a 0.15 dB loss in $\frac{E_b}{N_0}$.

In short, the sDFT seems to be very efficient when using long filters which's information is mostly contained in a few frequency bins. This way, we only need to use these few bins when filtering in frequency domain, resulting in a large reduction in implementation cost and only a small reduction in performance.

Contents

1 Introduction

Wireless communication is an important part of the modern society, and is something most people encounter many times during their day. For example when talking on the phone, browsing the internet or listening to music with a wireless headset. A simplified block diagram of such a communication system is shown in fig. 1, where a transmitter sends a signal to a receiver through a channel.

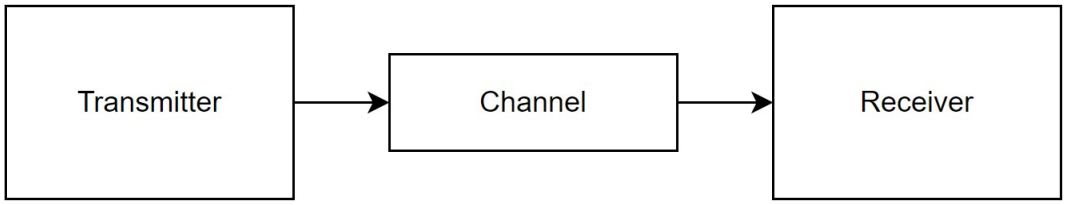


Figure 1: A simplified overview of a general communication system. The transmitter sends a signal through a channel, before the signal is picked up by a receiver.

An important part of the receiver in fig. 1 is the demodulator, which extracts information from the received signal. A demodulator often utilizes matched filters to extract information, and these filters are traditionally implemented in time domain using linear convolution. The same filters can also be implemented in frequency domain using the discrete Fourier transform (DFT), where the filtering is performed using multiplication instead of linear convolution.

This thesis will investigate the use of the sliding window discrete Fourier transform (sDFT) in a radio application, which is an efficient way of computing the DFT by using its recursive nature. More specifically, an sDFT based demodulator will be implemented and compared to a traditional demodulator using their performance and implementation cost. Additionally, the thesis also documents the implementation of an sDFT based preamble detector, which is used to synchronize the demodulator to the received symbols. The preamble detector is also compared to its time domain counterpart.

Apart from this introduction, this thesis is structured into six sections. Section 2 is the first of these, and presents the relevant background and theory needed to understand the implemented demodulator and preamble detector. Furthermore, section 3 and section 4 describes the implementation and verification of the these systems, while section 5 presents and discusses the results found during the verification. Lastly, section 6 describes aspects that can be interesting for further research, while section 7 makes a conclusion based on the results in section 5.

2 Background

This section presents the relevant background and theory needed to understand the implemented demodulator, preamble detector and the communication system they are involved in. Firstly, section 2.1 describes Gaussian frequency shift keying (GFSK) and how a general GFSK transmitter and receiver are built up. Section 2.2 then presents the discrete Fourier transform (DFT), before section 2.3, section 2.4 and section 2.5 present the sliding window

DFT (sDFT), the inverse sDFT and how these can be applied to perform filtering. Furthermore, section 2.6 derives and explains how a matched filter can be used for demodulation. Lastly, section 2.7 presents a way to efficiently estimate the magnitude of a complex number, before section 2.8 describes fixed-point numbers and how they can be used to represent rational numbers in a binary number system.

Note that section 2.2, section 2.3, section 2.4, section 2.5 and section 2.8 have some overlap with the background section in my project thesis, because the topics in these sections are relevant for both theses [1].

2.1 Gaussian frequency shift keying

Gaussian frequency shift keying (GFSK) is a continuous phase modulation technique that builds on regular FSK, where symbols are represented by sinusoidals with different frequencies [2, p. 218]. The difference between FSK and GFSK is that the pulse shaping function in GFSK uses the Gaussian function to reduce transmission bandwidth [3].

2.1.1 GFSK transmitter

A block diagram of a GFSK transmitter is shown in fig. 2. Here, the input signal $x(t)$ is a square wave containing the symbols we want to transmit. The transmitter modulates $x(t)$ and transmits it as $s_{tx}(t)$.

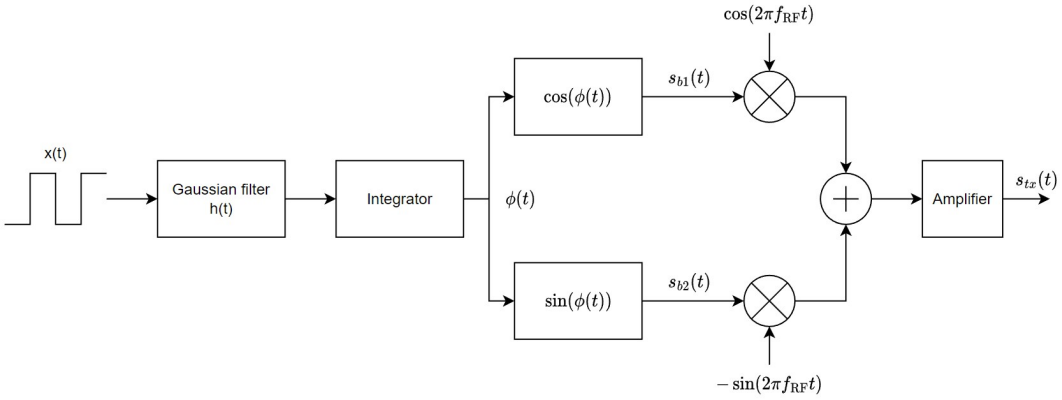


Figure 2: A block diagram of a GFSK transmitter. The input signal $x(t)$ is modulated and transmitted as $s_{tx}(t)$.

The input signal $x(t)$ is given by

$$x(t) = \sum_{n=-\infty}^{\infty} x[n] \frac{\Pi((t - nT)/T)}{T}, \tag{1}$$

where $x[n]$ is a sequence of symbols drawn from the symbol alphabet. In this thesis, we will for simplicity assume a binary symbol alphabet, meaning $x[n] \in \{-1, 1\}$. Additionally, T is the symbol period, and $\Pi(t)$ is the rectangular function, which is defined as

$$\Pi(t) = \begin{cases} 0 & |t| > \frac{1}{2} \\ 1/2 & |t| = \frac{1}{2} \\ 1 & |t| < \frac{1}{2} \end{cases}. \tag{2}$$

Furthermore, the Gaussian filter in fig. 2 smooths the input signal as shown in fig. 3. Here we can see that the transitions between 1 and -1, or the other way around, are instantaneous for the unfiltered signal, while they happen gradually over time for the filtered signal.

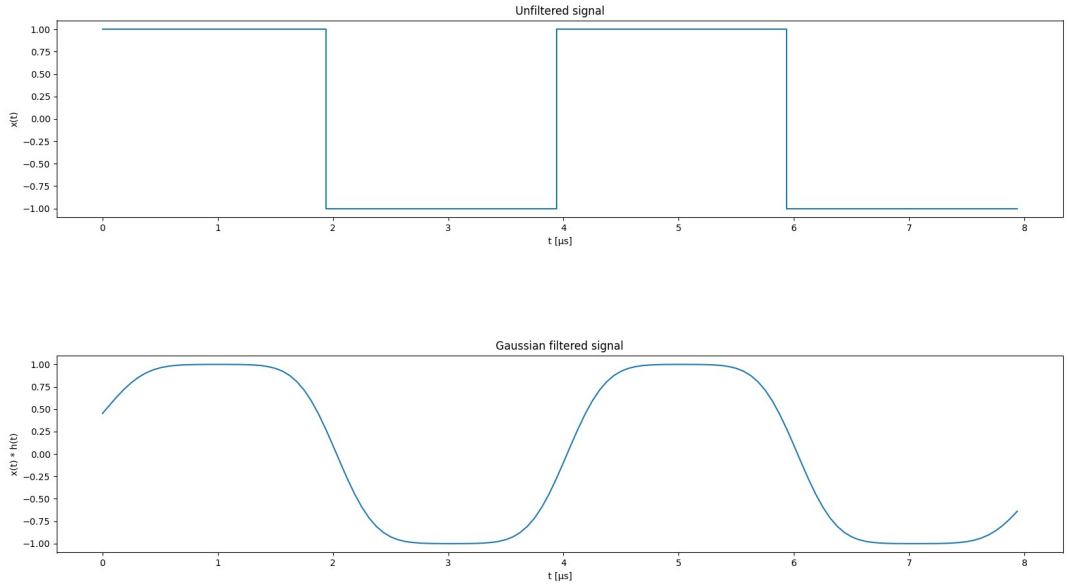


Figure 3: The input signal $x(t)$ before and after the Gaussian filter.

More specifically, the Gaussian filter is defined by its impulse response

$$h(t) = \frac{1}{\sqrt{2\pi}\sigma T} e^{-\frac{1}{2}\left(\frac{t}{\sigma T}\right)^2}, \tag{3}$$

where σT is the standard deviation [3, p. 517]. Note that this definition implies that the filter has an infinite length impulse response, which needs to be limited in a practical application. Additionally, one can express σ using the filter's 3 dB bandwidth B [3, p. 517]. This gives us

$$\sigma = \frac{\sqrt{\ln 2}}{2\pi B T}. \tag{4}$$

As mentioned, GFSK modulates symbols using sinusoidal of different frequencies. This is done using a phase $\phi(t)$ that changes over time, which is computed by integrating the Gaussian filter output, as shown in fig. 2. More specifically, the phase is defined as

$$\phi(t) = h\pi \int_{-\infty}^{t} \sum_{n=-\infty}^{\infty} x[n] g(\tau - nT) d\tau, \tag{5}$$

where h is the modulation index and $g(t)$ is the pulse shaping function, which is defined as

$$\begin{aligned} g(t) &= h(t) * \frac{\Pi(t/T)}{T} \\ &= \int_{-\infty}^{\infty} h(\tau) \frac{\Pi((t-\tau)/T)}{T} d\tau \\ &= \frac{1}{T} \int_{t-T/2}^{t+T/2} \frac{1}{\sqrt{2\pi}\sigma T} e^{-\frac{1}{2}\left(\frac{\tau}{\sigma T}\right)^2} d\tau. \end{aligned} \tag{6}$$

We then use the substitution $u = \frac{\tau}{\sigma T}$, which means $\frac{du}{d\tau} = \frac{1}{\sigma T} \Rightarrow d\tau = \sigma T du$. Inserting this into (6) gives

$$g(t) = \frac{1}{\sqrt{2\pi}T} \int_{u_1}^{u_2} e^{-\frac{u^2}{2}} du, \tag{7}$$

where $u_1 = \frac{t-T/2}{\sigma T}$ and $u_2 = \frac{t+T/2}{\sigma T}$. Finally, we can use the Q-function

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_{x}^{\infty} e^{-\frac{u^2}{2}} du \tag{8}$$

to express (7) as

$$\begin{aligned} g(t) =& \frac{1}{T} \left(Q(u_1) - Q(u_2) \right) \\ =& \frac{1}{T} \left(Q\left(\frac{t - \frac{T}{2}}{\sigma T} \right) - Q\left(\frac{t + \frac{T}{2}}{\sigma T} \right) \right). \end{aligned} \tag{9}$$

Lastly, as shown in fig. 2, the transmitted signal $s_{tx}(t)$ is created by upmixing the baseband signal

$$\begin{aligned} s_b(t) &= s_{b1}(t) + s_{b2}(t) \\ &= \cos(\phi(t)) + \sin(\phi(t)) \end{aligned} \tag{10}$$

to a radio frequency (RF) and then amplifying it. In other words, $s_{tx}(t)$ can be expressed as

$$
\begin{aligned}
s_{tx}(t) &= A\left[s_{b1}(t)\cos(2\pi f_{\mathrm{RF}}t) - s_{b2}(t)\sin(2\pi f_{\mathrm{RF}}t)\right] \\
&= A\left[\cos(\phi(t))\cos(2\pi f_{\mathrm{RF}}t) - \sin(\phi(t))\sin(2\pi f_{\mathrm{RF}}t)\right] \\
&= \frac{A}{2}\left[\cos(2\pi f_{\mathrm{RF}}t + \phi(t)) + \cos(2\pi f_{\mathrm{RF}}t - \phi(t))\right] \\
&- \frac{A}{2}\left[\cos(2\pi f_{\mathrm{RF}}t - \phi(t)) - \cos(2\pi f_{\mathrm{RF}}t + \phi(t))\right] \\
&= A\cos(2\pi f_{\mathrm{RF}}t + \phi(t)),
\end{aligned}
\tag{11}
$$

where A is the amplitude and f_{RF} is the radio frequency. Note that the RF is generally is the low GHz range.

2.1.2 GFSK receiver

This thesis focuses on a demodulator and preamble detector which are a part of a GFSK receiver. This receiver has a block diagram as shown in fig. 4, and downmixes the received signal to an intermediate frequency (IF). Note that the IF is generally in the low MHz range.

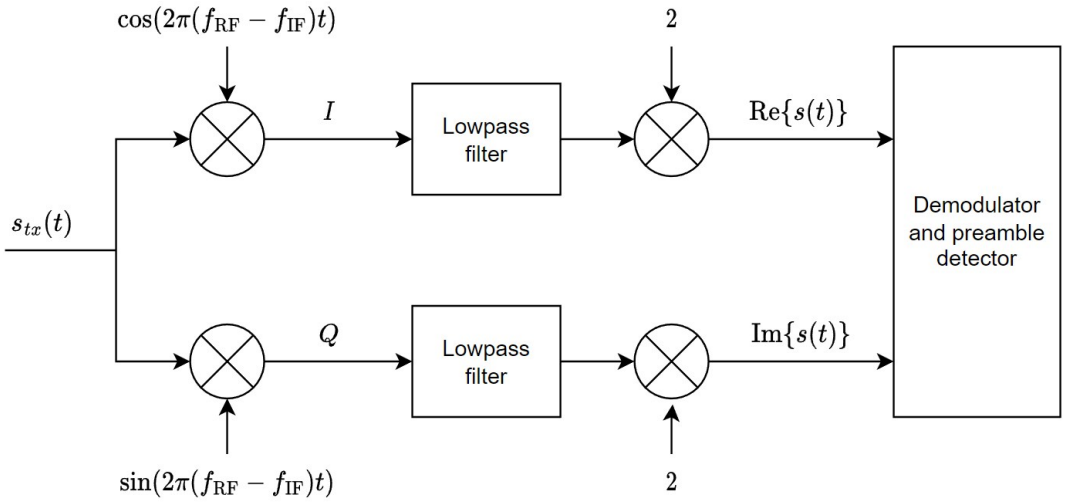


Figure 4: A block diagram of a GFSK receiver. The received signal is downmixed and transformed into an IQ signal which is sent to the demodulator and preamble detector.

The receiver in fig. 4 downmixes using both a sine and a cosine, which creates and IQ signal. We treat the IQ signal as a complex valued signal, and before the lowpass filter, this is given by

$$
\begin{aligned}
s_{\text{IF}}(t) &= I + jQ \\
&= s_{tx}(t) \cdot \cos(2\pi(f_{\text{RF}} - f_{\text{IF}})t) \\
&+ j s_{tx}(t) \cdot \sin(2\pi(f_{\text{RF}} - f_{\text{IF}})t) \\
&= A\cos(2\pi f_{\text{RF}} t + \phi(t)) \cdot \cos(2\pi(f_{\text{RF}} - f_{\text{IF}})t) \\
&+ jA\cos(2\pi f_{\text{RF}} t + \phi(t)) \cdot \sin(2\pi(f_{\text{RF}} - f_{\text{IF}})t) \\
&= \frac{A}{2}\left[\cos(2\pi(2f_{\text{RF}} - f_{\text{IF}})t + \phi(t)) + \cos(2\pi f_{\text{IF}} t + \phi(t))\right] \\
&+ j\frac{A}{2}\left[\sin(2\pi(2f_{\text{RF}} - f_{\text{IF}})t + \phi(t)) + \sin(2\pi f_{\text{IF}} t + \phi(t))\right].
\end{aligned}
\tag{12}
$$

The lowpass filter in fig. 4 then removes the high frequency components from the $s_{\text{IF}}(t)$, meaning $\cos(2\pi(2f_{\text{RF}} - f_{\text{IF}})t + \phi(t))$ and $\sin(2\pi(2f_{\text{RF}} - f_{\text{IF}})t + \phi(t))$. Lastly, the resulting signal is multiplied by 2 in order to preserve the amplitude. This signal is given by

$$
\begin{aligned}
s(t) &= A\left[\cos(2\pi f_{\text{IF}} t + \phi(t)) + j\sin(2\pi f_{\text{IF}} t + \phi(t))\right] \\
&= Ae^{j(2\pi f_{\text{IF}} t + \phi(t))},
\end{aligned}
\tag{13}
$$

and is sent to the demodulator and preamble detector.

2.1.3 Intersymbol interference

One downside with the Gaussian filter used for GFSK is that it introduces intersymbol interference (ISI). One way to visualize this is by comparing the phase trees for FSK and GFSK, which show how $\phi(t)$ from (5) changes depending on the transmitted symbols. For the case of FSK, the pulse shaping function is rectangular, meaning that (5) can be written as

$$
\phi(t) = h\pi \int_{-\infty}^{t} \sum_{n=-\infty}^{\infty} x[n]\Pi(\tau - nT)d\tau, \tag{14}
$$

where $\Pi(t)$ is a rectangular function given by (2).

From (14), we can see that the phase contribution for each symbol is $\pm h\pi$, depending on the sign of $x[n]$. This is shown in the FSK phase tree in fig. 5, where the phase changes by $\pm h\pi$ between each symbol.

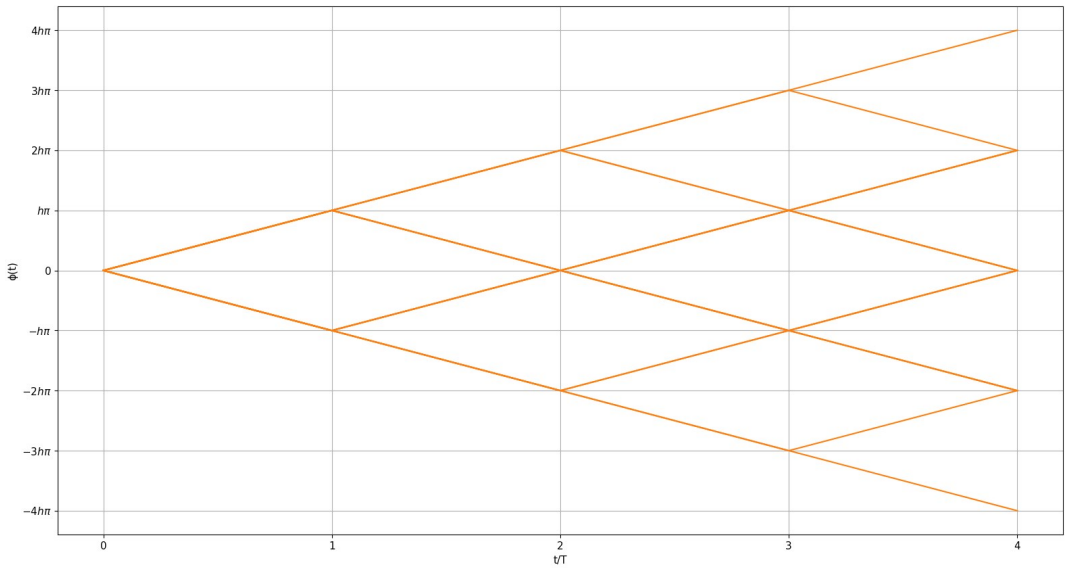


Figure 5: A phase tree for an FSK signal, where the initial phase is set to 0. Following a path upwards represents sending a 1 and downwards a -1. Note that the time axis is normalized by the symbol period.

For GFSK, the pulse shaping function $g(t)$ in (5) is a Gaussian given by (9), which results in the phase change from each symbol to be $\leq h\pi$ in absolute value due to ISI. A phase tree of a GFSK signal is shown in fig. 6, where the length of Gaussian filter $h(t)$ from (3) is limited to span two symbols. In this phase tree, we can clearly see the ISI by the way $\phi(t)$ curves. More specifically, $\phi(t)$ curves when the transmitted symbols alternate, causing the phase shift between two symbols to be $< h\pi$. Otherwise, when two or more identical symbols are transmitted after each other, the phase shift is $\pm h\pi$ and this creates a straight line in the phase tree. Also note that the symbols in fig. 6 are preceded by several ones, which is why the phase initially increases.

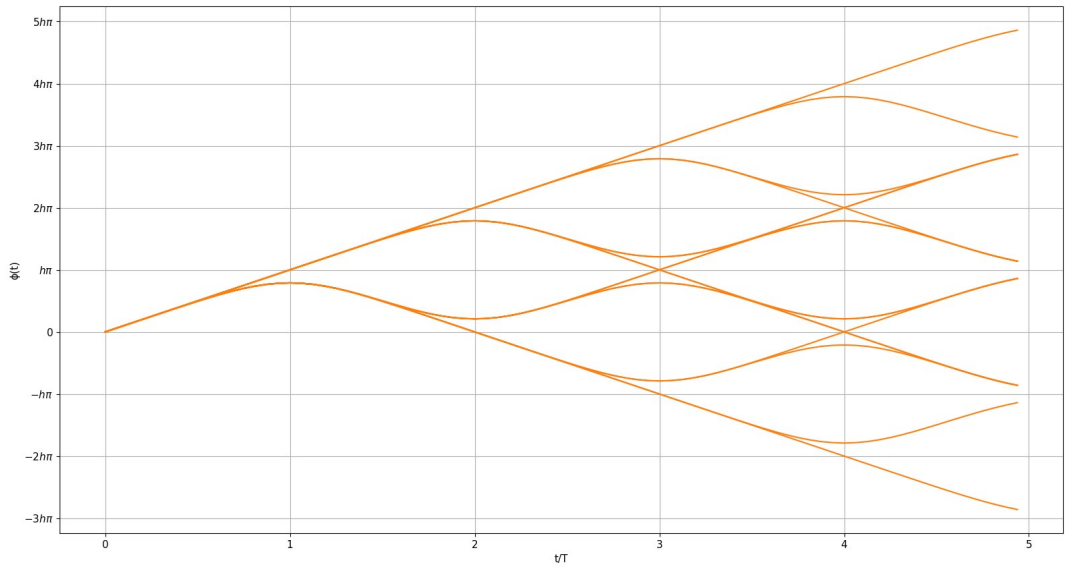


Figure 6: A phase tree for a GFSK signal, where the sequence is preceded by several 1s. Following a path upwards represents sending a 1 and downwards a -1. Note that the time axis is normalized by the symbol period.

2.2 DTFT and DFT

The Discrete Time Fourier Transform (DTFT) is a transform that computes a frequency domain representation of a discrete time signal. Since the frequency axis of the DTFT is continuous, the DTFT can't be implemented in a system and is just an analytical tool.

The Discrete Fourier Transform (DFT) computes a sampled DTFT of a finite length signal, meaning that the frequency axis is discrete. The DFT is given by

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N}, \tag{15}$$

where N denotes the number of samples on the frequency axis, which is greater or equal to the length of $x[n]$ [4, p. 468]. Since the frequency axis spans $[-\frac{f_s}{2}, \frac{f_s}{2}]$, this gives us a frequency resolution of

$$\Delta f = \frac{f_s}{N}. \tag{16}$$

Furthermore, the factor $e^{j2\pi/N}$ is often referred to as twiddle factor and is denoted as

$$W_N = e^{j2\pi/N}.$$

The DFT also has an inverse, which is given by

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j2\pi kn/N}. \tag{17}$$

2.3 Sliding window discrete Fourier transform

The sliding window discrete Fourier transform (sDFT) is an algorithm that uses the recursive nature of the DFT to compute it efficiently [5, p. 1]. This algorithm is found by inspecting a sequence of length M

$$x = \{x[n], x[n-1], \ldots, x[n-M+1]\},$$

and applying the DFT to it. According to (15), this gives us

$$\begin{aligned} X_n[k] &= \sum_{m=0}^{M-1} x[n-M+1+m]e^{-j2\pi km/M} \\ &= \sum_{m=0}^{M-1} x[\hat{n}+m]W_M^{-mk}, \end{aligned} \tag{18}$$

where $\hat{n} = n - M + 1$ and $W_M^{-mk} = e^{-j2\pi km/M}$.

We then do the same again, but with a new sequence that is shifted one sample from the previous. More specifically, n has been replaced with $n+1$, and the DFT of this sequence is denoted $X_{n+1}[k]$. By writing all the terms in the sum in $X_n[k]$ and $X_{n+1}[k]$, we get respectively (19) and (20).

$$X_n[k] = x[\hat{n}] + x[\hat{n}+1]W_M^{-k} + \cdots + x[\hat{n}+M-1]W_M^{-(M-1)k} \tag{19}$$

$$X_{n+1}[k] = x[\hat{n}+1] + x[\hat{n}+2]W_M^{-k} + \cdots + x[\hat{n}+M]W_M^{-(M-1)k} \tag{20}$$

We can see that (19) and (20) are very similar, and the sDFT uses these similarities to efficiently compute the DFT. More specifically, it expresses X_{n+1} as

$$\begin{aligned} X_{n+1}[k] &= x[\hat{n}+1] + x[\hat{n}+2]W_M^{-k} + \cdots + x[\hat{n}+M]W_M^{-(M-1)k} \\ &= X_n[k]W_M^k - x[\hat{n}]W_M^k + x[\hat{n}+M]W_M^{-(M-1)k} \\ &= W_M^k(X_n[k] - x[\hat{n}] + x[\hat{n}+M]W_M^{-Mk}) \\ &= W_M^k(X_n[k] - x[\hat{n}] + x[\hat{n}+M]e^{-j2\pi k}) \\ &= W_M^k(X_n[k] - x[\hat{n}] + x[\hat{n}+M]), \end{aligned} \tag{21}$$

where the final simplification comes from Euler's formula, stating that $e^{-j2\pi k} = 1$ when k is an integer. Lastly, the substitution $\hat{n} = n - M + 1$ is undone and we use $X_n[k]$ and $X_{n-1}[k]$ instead of $X_{n+1}[k]$ and $X_n[k]$. The sDFT is then given by

$$X_n[k] = W_M^k(X_{n-1}[k] + x[n] - x[n-M]). \tag{22}$$

Note that the length M does not have any restrictions, e.g. it does not have to be on the form 2^k like for some algorithms of the Fast Fourier Transform (FFT) [4, p. 531].

As shown in (22), a successive sDFT only requires two complex additions and a complex multiplication per frequency component, regardless of the window size M [5, p. 3]. This means that the entire spectrum requires $2N$ complex additions and N complex multiplications, where N is the number of frequency components. Furthermore, note that a sequence of N samples requires the sDFT to be computed N times, because the window has to be shifted by one sample N times. This is illustrated in fig. 7, where the figure shows how the window is shifted for the first N samples of a sequence. Note that the same concept applies when starting from any sample and computing the sDFT for the next N samples.

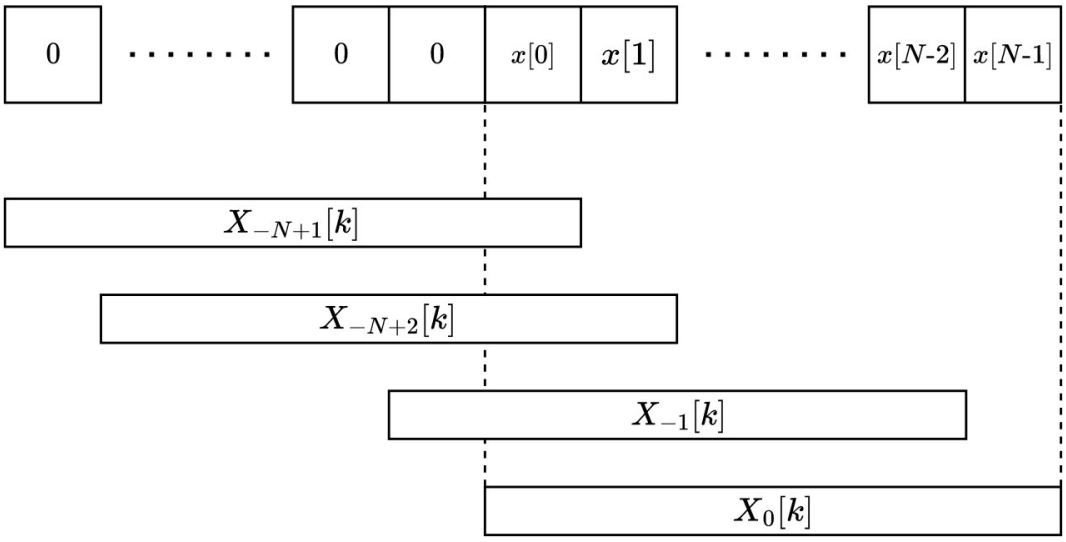


Figure 7: An illustration showing that the sDFT needs to be computed N times to cover N samples. Note that the signal is assumed to be 0 before the N samples.

2.4 Inverse sDFT

The inverse sDFT algorithm used in this thesis is derived from the inverse DFT in (17). We start by adding a time index m to $X_m[k]$, which denotes the DFT computed at time m. This gives us

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X_m[k] e^{j2\pi kn/N}, \tag{23}$$

where N is the size of the window used to compute $X_m[k]$. Furthermore, we note that $n = 0$ results in $e^{j2\pi kn/N} = 1$, which means that the first sample in the window can be computed using only N additions and no multiplications. More specifically, we can reconstruct $x[0]$ using

$$
\begin{aligned}
x[0] &= \frac{1}{N}\sum_{k=0}^{N-1} X_0[k]e^{j2\pi k(0/N)} \\
&= \frac{1}{N}\sum_{k=0}^{N-1} X_0[k].
\end{aligned}
\tag{24}
$$

Since the sDFT uses windows that are just one sample apart, we just need to reconstruct one sample per window in order to reconstruct the entire signal, with the exception of $N-1$ at the end of the signal. This means that (24) can be generalized to

$$
x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X_n[k], \tag{25}
$$

where $x[n]$ is the first sample in the window used to compute $X_n[k]$.

2.5 Filtering with the sDFT

A filter can be used to attenuate unwanted components from a signal, for instance noise. In this thesis, we will consider linear time-invariant (LTI) filters, which can be described by

$$
y[n] = \sum_{k=0}^{M-1} b_k x[n-k] - \sum_{k=1}^{N} a_k y[n-k], \tag{26}
$$

where $y[n]$ is the filter output, $x[n]$ is the input signal and b_k and a_k are the filter coefficients [4, p. 674]. Filtering in time domain is performed using linear convolution, which is given by

$$
y[n] = x[n] * h[n] = \sum_{m=-\infty}^{\infty} x[m]h[n-m], \tag{27}
$$

where $h[n]$ is the sampled impulse response of the filter.

Furthermore, a filter can be described in frequency domain using its frequency response $H(\omega)$, where $\omega = 2\pi f$ is the angular frequency [4, p. 675]. The DTFT of the filter output is given by

$$
Y(\omega) = H(\omega)X(\omega), \tag{28}
$$

where $H(\omega)$ is given by

$$
H(\omega) = \frac{\sum_{k=0}^{M-1} b_k e^{-j\omega k}}{1 + \sum_{k=1}^{N} a_k e^{-j\omega k}}. \tag{29}
$$

As shown in (28), filtering in frequency domain can be performed using the multiplication of two DTFTs. Since the DTFT can't be implemented in a system, we need to use the DFT instead. However, the multiplication of two DFTs is equivalent to circular convolution in time domain, and not linear convolution like we want [4, p. 488].

If we assume that $x[n]$ and $h[n]$ are both of length N, the expected length of $y[n] = x[n] * h[n]$ is $2N - 1$ [4, p. 493]. However, when performed in frequency domain using an N-point DFT,

$$\hat{y}[n] = \text{IDFT}\{H[k]X[k]\} \tag{30}$$

has a length of N. This is a result of circular convolution essentially being an aliased version of linear convolution. More specifically, the aliasing is given by

$$\begin{aligned} \hat{y}[0] &= y[0] + y[N] \\ \hat{y}[1] &= y[1] + y[N+1] \\ &\vdots \\ \hat{y}[N-1] &= y[N-1] + y[2N-1]. \end{aligned} \tag{31}$$

However, note that since $y[n]$ is of length $2N - 1$, $y[2N - 1]$ is not a part of this sequence since this is the $2N$th sample [4, p. 497]. Therefore, $\hat{y}[N - 1]$ is given by

$$\hat{y}[N-1] = y[N-1] + 0 = y[N-1], \tag{32}$$

and does not contain aliasing.

When using the sDFT, we note that the inverse sDFT only needs to construct one sample per window, as explained in section 2.4. Therefore, we can utilize this to only reconstruct the samples that doesn't contain aliasing. The inverse sDFT shown in eq. (25) thus needs to be altered to reconstruct the Nth sample in the window, meaning

$$y[N-1] = \frac{1}{N} \sum_{k=0}^{N-1} \hat{Y}[k] e^{j2\pi k(N-1)/N}, \tag{33}$$

where $\hat{Y}[k] = H[k]X[k]$ is the filtered spectrum.

2.6 Minimum-distance criterion and matched filter

Assume a transmitter sends a single symbol, such that the receiver observes

$$r(t) = as(t) + n(t), \tag{34}$$

where $a \in \mathcal{A}$ is the transmitted symbol, $s(t)$ is the noiseless received signal and $n(t)$ is the noise. For a symbol alphabet $\mathcal{A} = \{-1, 1\}$, a receiver following the minimum-distance strategy will compare the received signal $r(t)$ to the two possible transmitted signals, $s(t)$ and $-s(t)$ [2, p. 154]. This comparison is done by computing the energies

$$\int_{-\infty}^{\infty} |r(t) - s(t)|^2 dt \quad \text{and} \quad \int_{-\infty}^{\infty} |r(t) + s(t)|^2 dt, \tag{35}$$

where the correct symbol will have a smaller energy. This comparison can be generalized to any symbol alphabet

$$\hat{a} = \arg\min_{a \in \mathcal{A}} \int_{-\infty}^{\infty} |r(t) - as(t)|^2 dt, \tag{36}$$

where $\hat{a}$ is the symbol outputted by the receiver. The minimized expression in (36) is called a cost function, and can be rewritten as

$$\begin{aligned} J =& \int_{-\infty}^{\infty} |r(t) - as(t)|^2 dt \\ =& \int_{-\infty}^{\infty} |r(t)|^2 dt - 2\mathrm{Re}\left\{ a^* \int_{-\infty}^{\infty} r(t)s^*(t)dt \right\} + |a|^2 \int_{-\infty}^{\infty} |s(t)|^2 dt \\ =& E_r - 2\mathrm{Re}\{a^* y\} + |a|^2 E_h, \end{aligned} \tag{37}$$

where

$$E_r = \int_{-\infty}^{\infty} |r(t)|^2 dt \quad \text{and} \quad E_h = \int_{-\infty}^{\infty} |s(t)|^2 dt \tag{38}$$

are respectively the energies of $r(t)$ and $s(t)$. Furthermore,

$$y = \int_{-\infty}^{\infty} |r(t)s^*(t)|^2 dt \tag{39}$$

can be interpreted as the correlation between the noiseless signal and received signal with noise [2, p. 156].

We notice that in (37), the first term E_r is independent on the transmitted symbol a, meaning that the receiver can ignore it. Additionally, of the two remaining terms only the first one depends on $r(t)$, and it does so through the correlation y in (39). Therefore, y is a sufficient statistic for making the minimum distance decision [2, p. 156].

This decision can be implemented in two different ways. Either directly as shown in (39) or by using a filter with impulse response $s^*(-t)$ and sample it at $t = 0$, as shown in fig. 8. Such a filter is said to be matched to $s(t)$, and the structure in fig. 8 is called a sampled matched filter.

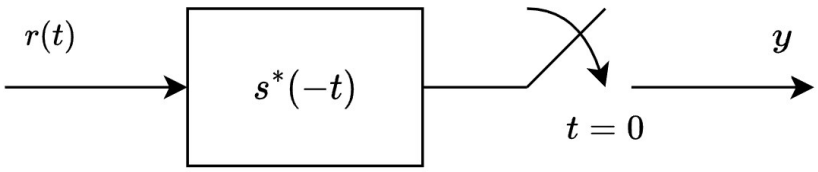


Figure 8: A sampled matched filter implementation of a minimum-distance receiver. Adapted from [2, p. 156].

A demodulator can therefore be made using a filterbank of matched filters that each corresponds to a different symbol. The decision is then made by selecting the symbol corresponding to the filter output with the highest magnitude at the intervals nT, where T is the symbol period.

2.7 Magnitude estimation

The magnitude of a complex number $z = x + jy$ is given by

$$|z| = \sqrt{x^2 + y^2}, \tag{40}$$

and involves computing one addition, two squares and one square root. Especially the latter is expensive to do in hardware, which motivates the need for a cheaper approximation of the magnitude.

One way of doing this is the alpha max plus beta min algorithm, which is given by

$$|z| \approx \alpha \max(|x|, |y|) + \beta \min(|x|, |y|). \tag{41}$$

The optimal values for α and β are

$$\begin{aligned} \alpha_0 &= \frac{2\cos(\frac{\pi}{8})}{1 + \cos(\frac{\pi}{8})} \approx 0.960433870103 \\ \beta_0 &= \frac{2\sin(\frac{\pi}{8})}{1 + \cos(\frac{\pi}{8})} \approx 0.397824734759, \end{aligned} \tag{42}$$

but one can also use other values that are cheaper to implement in hardware at a cost of a larger error [6]. The performance of a few different values are shown in fig. 9. Note that the figure only shows the results for one quadrant of the complex plane, but since the estimator uses the absolute values of x and y, their sign doesn't matter. In other words, the performance is the same in all four quadrants.

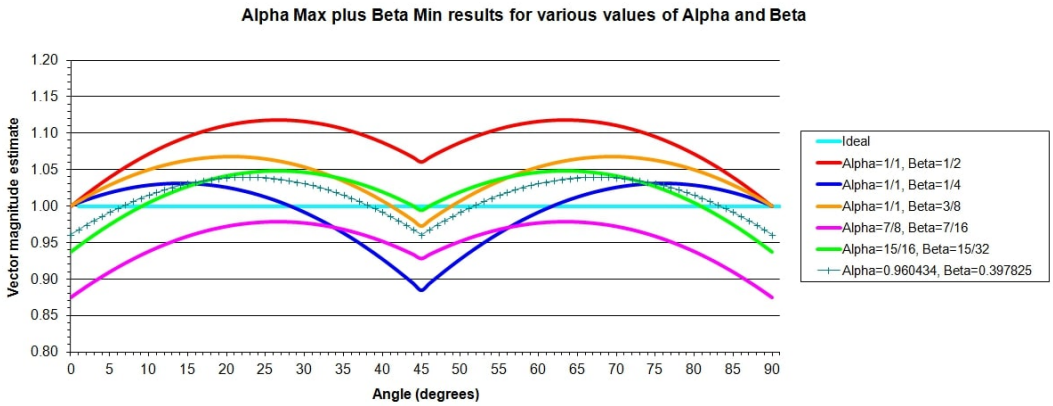


Figure 9: The performance of a few different values of α and β. Figure taken from [6], used under the CC BY-SA 3.0 license [7].

2.8 Fixed-point number representation

Fixed-point numbers can be used to represent rational numbers in a binary number system [4, p. 615]. A number is split into an integer and a fraction part with fixed lengths, as shown in fig. 10.

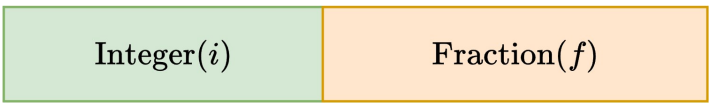


Figure 10: The structure of a fixed-point number, which consists of an integer and a fraction part. Both parts have a fixed number of bits, and these numbers are denoted as respectively i and f.

Two's complement fixed-point numbers are denoted as $Q(i, f)$, where i and f are respectively the number of integer and fraction bits. A $Q(i, f)$ number has a resolution of 2^{-f} and can represent numbers in the range $[-2^{i-1}, 2^{i-1} - 2^{-f}]$.

Note that in a register a $Q(i, f)$ number is just a group of bits, so in order to extract the decimal value of these bits, we can use the formula

$$\text{value}_{10} = v_{\text{i}} \cdot 2^{-f}. \tag{43}$$

In (43), v_i is the integer value of the full number (not just the integer bits), and f is the number of fraction bits. This means that fixed-point numbers are just regular integers, except that they are scaled by 2^{-f}. This scaling factor adds a decimal point f digits from the right.

(43) is often best illustrated with an example, so we will use it to find the value of the $Q(3, 5)$ number 10101100_2. Since this number starts with a 1, we need to first find the two's complement. Inverting the bits and adding 1 gives us $01010011 + 1 = 01010100$. According to (43), the value of this number is then $-(2^2 + 2^4 + 2^6) \cdot 2^{-5} = -2.625_{10}$.

2.8.1 Fixed-point arithmetic

A fixed-point number is essentially just an integer number that is scaled by 2^{-f}, as shown in (43). When doing arithmetic, we can therefore treat fixed-point numbers as integers, as long as we remember that these integers have been upscaled by 2^f from the value they represent.

For addition, the sum is not affected by this upscaling. This can be shown by adding two upscaled numbers

$$A \cdot 2^f + B \cdot 2^f = (A + B) \cdot 2^f,$$

where the result is also upscaled by 2^f. This is as it should be, because when we extract the value using (43), we get $A + B$.

For multiplication, the product is affected by this upscaling and needs to be corrected. If we multiply two upscaled numbers we get

$$A \cdot 2^f \cdot B \cdot 2^f = A \cdot B \cdot 2^{2f},$$

where the result is scaled by 2^{2f}. Therefore, we need to divide by 2^f before extracting the value using (43) in order to get the correct result.

3 Implementation

This section describes the implementation of a Bluetooth Low Energy (BLE) demodulator. The demodulator utilizes a matched filterbank, which uses the sDFT to perform filtering. Furthermore, the demodulator uses a preamble detector to find the optimal time to sample the matched filter outputs.

Firstly, section 3.1 and section 3.2 describes the structure of the packets and signals used by the demodulator and preamble detector. Additionally, section 3.3 and section 3.4 describes how fixed-point numbers and complex adders and multipliers are implemented in SystemVerilog. Furthermore, the implementation of the sDFT based demodulator is described in detail in section 3.5, while section 3.6 describes a way to use the sDFT to save hardware resources with a slight reduction in performance. Section 3.7 then describes the implementation of the preamble detector and how it is connected to the demodulator. Lastly, section 3.8 describes alternative time domain implementations of the demodulator and preamble detector, which are later compared to the sDFT implementations.

Note that section 3.3, section 3.4 and section 3.5.1 builds on the work done in my project thesis, so these sections have some overlap with the implementation section in [1].

3.1 Bluetooth low energy packets

As mentioned, the demodulator and preamble detector implemented in this thesis uses Bluetooth Low Energy (BLE) packets. These packets consist of a preamble, an address, a payload and error correction using cyclic redundancy check (CRC), as shown in fig. 11 [8, p. 3]. In this case, we are using a preamble with a length of 8 symbols, consisting of alternating -1s and 1s. Due to time constraints, and the need to limit the scope of the thesis, the remaining parts of the packets will just be treated as data that needs to be demodulated. In other words, we do not perform error correction nor do we check if the demodulated address matches an actual address. Therefore, the length and shape of the address, payload and CRC fields are not important.

Figure 11: A BLE packet consisting of an 8 bit preamble, an address, a payload and CRC.

3.2 GFSK modulation

When implementing the demodulator, we assume that there already exists a receiver which downmixes the received signal to an intermediate frequency. As described in section 2.1, the downmixed signal is then given by (13), which is repeated here for convenience:

$$s(t) = Ae^{j(2\pi f_{\text{IF}} t + \phi(t))}. \tag{44}$$

In eq. (44), $\phi(t)$ is given by (5), which is also repeated here:

$$\phi(t) = h\pi \int_{-\infty}^{t} \sum_{n=-\infty}^{\infty} x[n] g(\tau - nT) d\tau. \tag{45}$$

In order for the demodulator to work with the signal in (44), the signal is sampled at intervals T_s, giving the sampled signal

$$s[n] = s(nT_s) = Ae^{j(2\pi f_{\text{IF}} nT_s + \phi(nT_s))}. \tag{46}$$

Furthermore, we choose the intermediate frequency to be $f_{\text{IF}} = 1$ MHz. This could technically be chosen as any frequency in the low MHz range, so 1 MHz is an arbitrary choice. We also choose our data rate to be the same as f_{IF}, meaning $f_d = \frac{1}{T} = 1$ MHz. Additionally, we choose to send each symbol over several samples, because this makes it easier to correctly demodulate the symbols in the presence of noise. However, more samples per symbol also requires a higher sampling frequency in order to maintain the same data rate. Therefore, we choose to use 16 samples per symbol, because this makes the system quite robust to noise, while still having a reasonably low sampling frequency. More specifically, the sampling frequency is given by $f_s = \frac{1}{T_s} = 16 f_d = 16$ MHz. Lastly, the modulation index h in (45) is chosen to be 0.5, because for BLE h should be between 0.45 and 0.55 [9, p. 978].

In (45), $g(t)$ is the pulse shaping function, which involves a Gaussian filter. This filter is described by its impulse response $h(t)$ from (3). As mentioned in section 2.1.1, $h(t)$ has an infinite length, meaning that it needs to be limited in order for it to be implemented. In this case, we choose to limit it to two symbol periods, meaning 32 samples. The sampled $h(t)$ is then given by

$$h[n] = h(nT_s) = \frac{1}{\sqrt{2\pi}\sigma T} e^{-\frac{1}{2}\left(\frac{nT_s}{\sigma T}\right)^2}, \tag{47}$$

for $n = \{-16, -15, -14, \ldots, 13, 14, 15\}$. Additionally, σ is given by (4), which is repeated here:

$$\sigma = \frac{\sqrt{\ln 2}}{2\pi BT}. \tag{48}$$

In (48), BT is set to 0.5 and this results in the sampled impulse response shown in fig. 12.

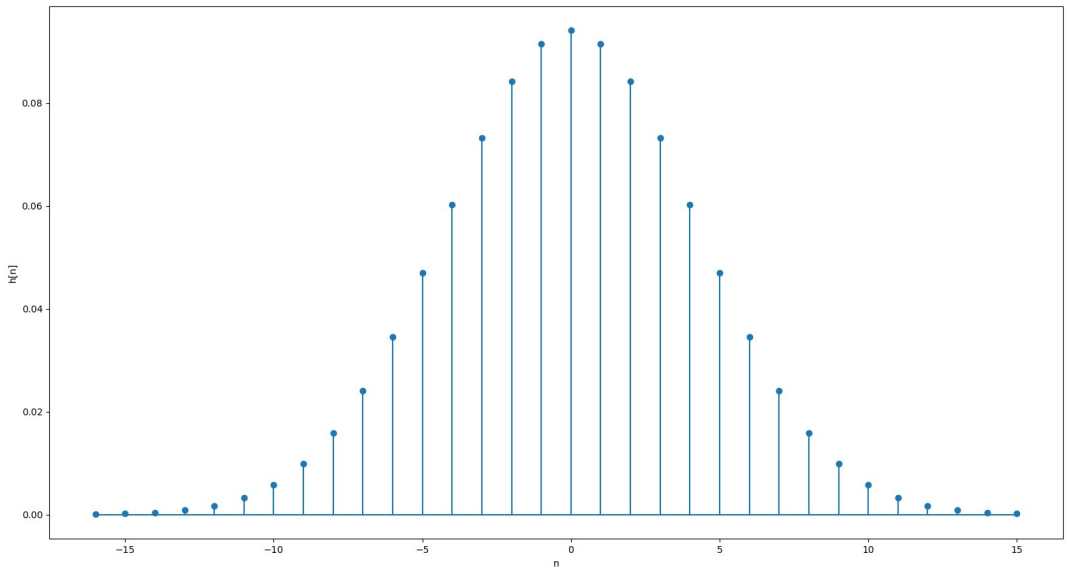


Figure 12: The sampled impulse response of a Gaussian filter. The length of the filter is 32 samples, which corresponds to two symbols.

As mentioned in section 2.1.3, the phase shift per symbol for GFSK is $\leq h\pi$ in absolute value. For $h = 0.5$ and a data rate $f_d = 1$ MHz, this means that the maximum shift in frequency is

$$\begin{aligned} f_{\text{shift}} &= \frac{h\pi}{2\pi} \cdot f_d \\ &= \frac{0.5\pi}{2\pi} \cdot 1 \text{ MHz} \\ &= 0.25 \text{ MHz}. \end{aligned} \tag{49}$$

3.3 Fixed-point numbers in SystemVerilog

The systems implemented in this thesis uses fixed-point numbers. As mentioned in section 2.8, this is a way to represent rational numbers in a binary number system by scaling an integer with 2^{-f}, where f is the number of fraction bits. However, in SystemVerilog, a number is still a register with a set amount of bits, which makes it more convenient to treat them as regular integers. This means that the numbers are upscaled by 2^f, and we need to keep this scaling factor in mind when we perform arithmetic, as described in section 2.8.1.

During my project thesis it was found that for a 16 bin sDFT, the accuracy barely improves for more accurate formats than $Q(4, 12)$ [1, p. 28]. Therefore, we choose to use $Q(4, 12)$ numbers.

3.4 Complex arithmetic

Parts of the systems implemented in this thesis performs arithmetic using complex numbers. For this purpose, complex addition and multiplication are implemented as shown in respectively section 3.4.1 and section 3.4.2.

3.4.1 Complex adder

Complex addition is computed as

$$\begin{aligned} z = x + y &= (x_1 + jx_2) + (y_1 + jy_2) \\ &= x_1 + y_1 + j(x_2 + y_2), \end{aligned}$$

meaning that

$$\begin{aligned} \mathrm{Re}(z) &= \mathrm{Re}(x) + \mathrm{Re}(y) \\ \mathrm{Im}(z) &= \mathrm{Im}(x) + \mathrm{Im}(y). \end{aligned}$$

In SystemVerilog, this is implemented as shown in fig. 13, which requires two real additions.

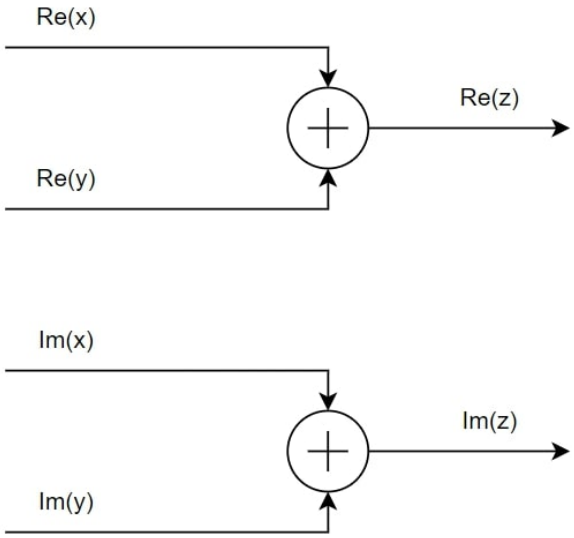


Figure 13: A complex adder computing $z = x + y$.

3.4.2 Complex multiplier

Complex multiplication is computed as

$$\begin{aligned} z = x \cdot y &= (x_1 + jx_2) \cdot (y_1 + jy_2) \\ &= x_1y_1 - x_2y_2 + j(x_1y_2 + x_2y_1), \end{aligned}$$

meaning that that

$$\begin{aligned} \mathrm{Re}(z) &= \mathrm{Re}(x)\mathrm{Re}(y) - \mathrm{Im}(x)\mathrm{Im}(y) \\ \mathrm{Im}(z) &= \mathrm{Re}(x)\mathrm{Im}(y) + \mathrm{Im}(x)\mathrm{Re}(y). \end{aligned}$$

In SystemVerilog, this is implemented as shown in fig. 14, which requires four real multiplications and two real additions/subtractions.

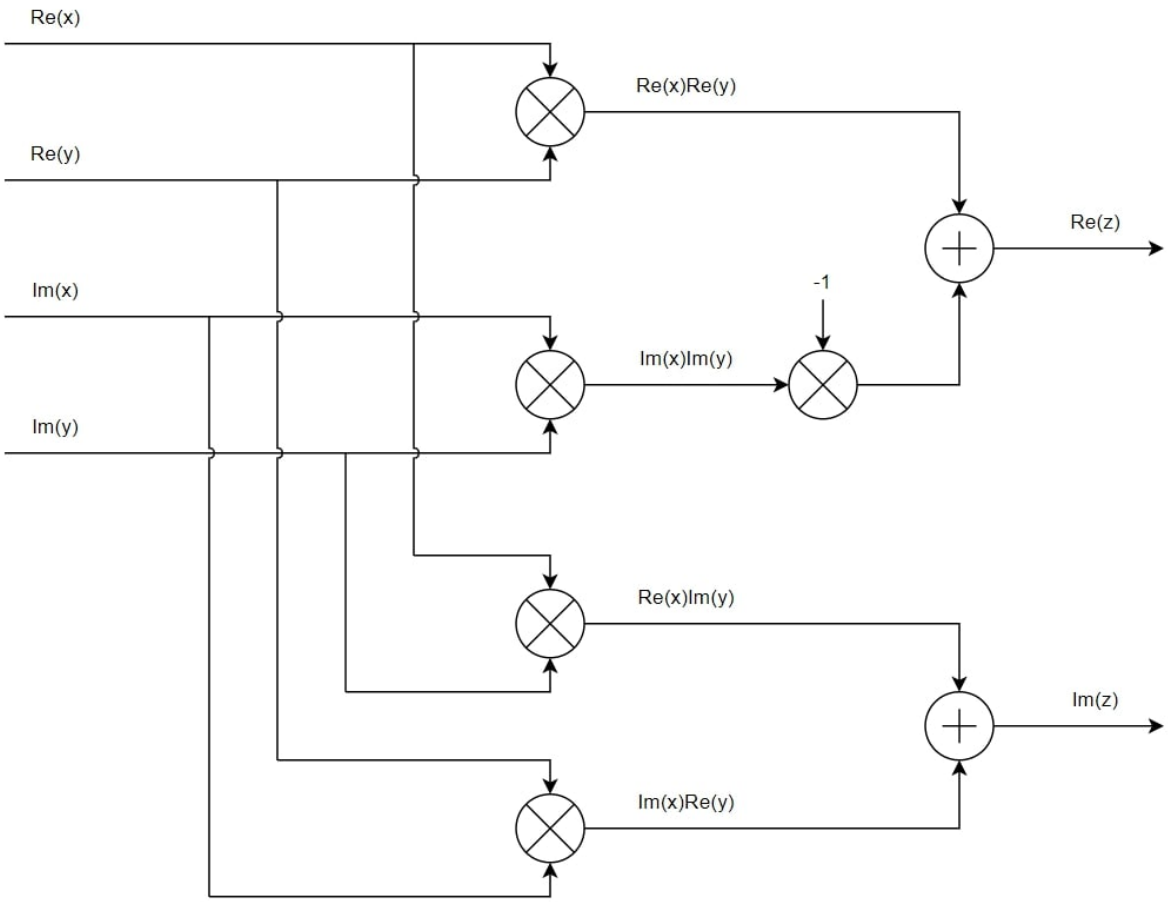


Figure 14: A complex multiplier computing $z = x \cdot y$.

3.5 sDFT demodulator

A block diagram of the implemented sDFT demodulator is shown in fig. 15. It is based around a matched filterbank, where the filtering is performed in frequency domain using the sDFT. The filterbank contains one filter for each symbol in the symbol alphabet, resulting in two filters. The filter outputs are then transformed back into time domain using the inverse sDFT, before they are compared to decide which symbol was received. The sequence of demodulated symbols is denoted $\hat{x}[n]$.

Furthermore, it is assumed that the noise added to the transmitted signal is additive white Gaussian noise. This is why a matched filterbank structure was chosen, because such filters maximizes the SNR in these conditions [2, p. 158].

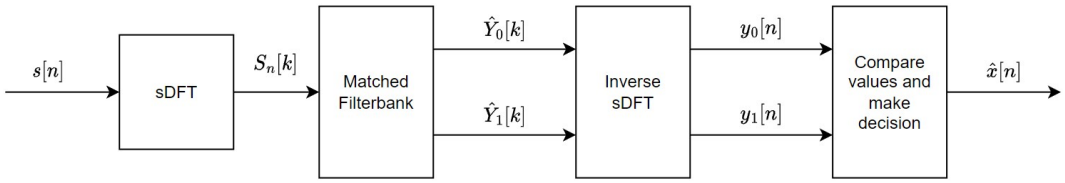


Figure 15: A block diagram of the demodulator. It uses a matched filterbank with one filter for each symbol, where the filtering is performed in frequency domain using the sDFT. The inverse sDFT is then computed, and the outputted samples are compared in order to decide which symbol was received.

3.5.1 sDFT

Figure 16 shows a block diagram of the SystemVerilog implementation of the sDFT, which is an IIR implementation of (22) and builds on the block diagram presented in [5, p. 3]. Note that this block diagram shows the computation of a single frequency bin, and each bin is computed independently of the other bins. In order to compute the entire sDFT, the system in fig. 16 is repeated N times, resulting in N frequency bins. Furthermore, since the sDFT operates on complex numbers, the adders and multipliers in fig. 16 are complex. These are implemented as shown in section 3.4.1 and section 3.4.2.

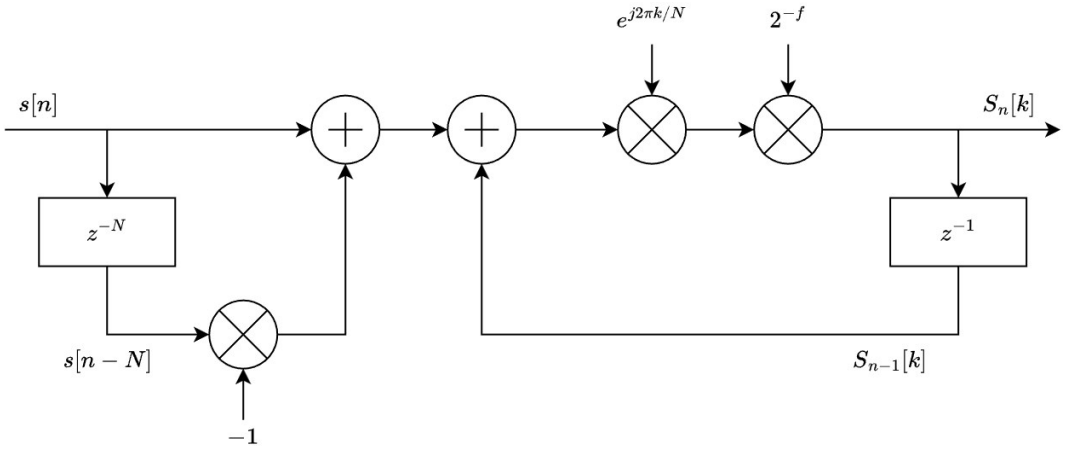


Figure 16: An IIR implementation of a system computing a single bin of the sDFT.

The system in fig. 16 initialises by setting $s[n]$ and $S_n[k]$ to 0, and then needs N samples before the output is ready, where N is the window size. Note that in this implementation the number of frequency bins and the window size are the same. We choose $N = 16$, because this is the number of samples per symbol, and this way, one window can capture exactly one symbol. As shown in (16), this gives a frequency resolution of $\Delta f = \frac{f_s}{N} = \frac{16}{16} = 1$ MHz.

Furthermore, note that the system multiplies by 2^{-f} after multiplying by $e^{j2\pi k/N}$. As explained in section 2.8.1, this is done because we're using fixed-point numbers that are treated as integers. For multiplication, this means that the scaling factor of the resulting product will be 2^{2f}, which needs to be downscaled by 2^{-f} in order to be correct. This downscaling is done by removing f of the least significant bits, which is a cheap way to perform division and will be referred to as bit slicing.

Lastly, the twiddle factors $e^{j2\pi k/N}$ are not computed in the SystemVerilog module. Instead, they are computed in Python and imported as a set of parameters, because they are constants and do not require extra logic to be computed. When computed, the twiddle factors are always rounded down (in absolute value), because their magnitude needs to be less or equal to 1 in order to ensure stability [10].

3.5.2 Matched filterbank

As explained in section 2.6, a matched filterbank consists of filters matched to the different symbols in the symbol alphabet. Furthermore, as mentioned in section 2.1.3, the Gaussian filter in GFSK introduces ISI, meaning that a transmitted symbol is dependent on the symbols before it. Therefore, if we were to create matched filters based on a GFSK signal, we would have to create multiple filters per symbol.

In order to save hardware resources, we will instead create the matched filters based on FSK signals. As shown in section 2.1.3, FSK signals do not contain ISI, meaning that one filter per symbol is enough. Note that matching to FSK signals gives a slightly worse performance than matching to GFSK, but the difference in performance is small.

For the 1 symbol, we are matching to

$$s_1(t) = Ae^{j(2\pi f_{\text{IF}} t + \phi_1(t))}, \tag{50}$$

where

$$\phi_1(t) = h\pi \int_{-\infty}^{t} \sum_{n=-\infty}^{\infty} \Pi(\tau - T)d\tau, \tag{51}$$

and t spans one symbol period, meaning $0 \leq t \leq T$, and $\Pi(t)$ is the rectangular function. Similarly, for the -1 symbol, we are matching to

$$s_0(t) = Ae^{j(2\pi f_{\text{IF}} t + \phi_0(t))}, \tag{52}$$

where

$$\phi_0(t) = h\pi \int_{-\infty}^{t} \sum_{n=-\infty}^{\infty} -\Pi(\tau - T)d\tau. \tag{53}$$

The impulse responses for the matched filters are then given by

$$\begin{aligned} h_1(t) &= s_1^*(-t) \\ h_0(t) &= s_0^*(-t), \end{aligned} \tag{54}$$

where the length of $s_1(t)$ and $s_0(t)$ are limited to one symbol period T.

Furthermore, the impulse responses in (54) are sampled with the sampling frequency $f_s = \frac{1}{T_s} = 16$ MHz, giving

$$\begin{aligned} h_1[n] &= h_1(nT_s) = s_1^*(-nT_s) \\ h_0[n] &= h_0(nT_s) = s_0^*(-nT_s). \end{aligned} \tag{55}$$

The impulse responses $h_1[n]$ and $h_0[n]$ have a length of 1 symbol or 16 samples, and they are shown in fig. 17 and fig. 18. These figures show both the real and imaginary part of $h_1[n]$ and $h_0[n]$, and we can see that the only difference between these parts is a 90° phase shift.

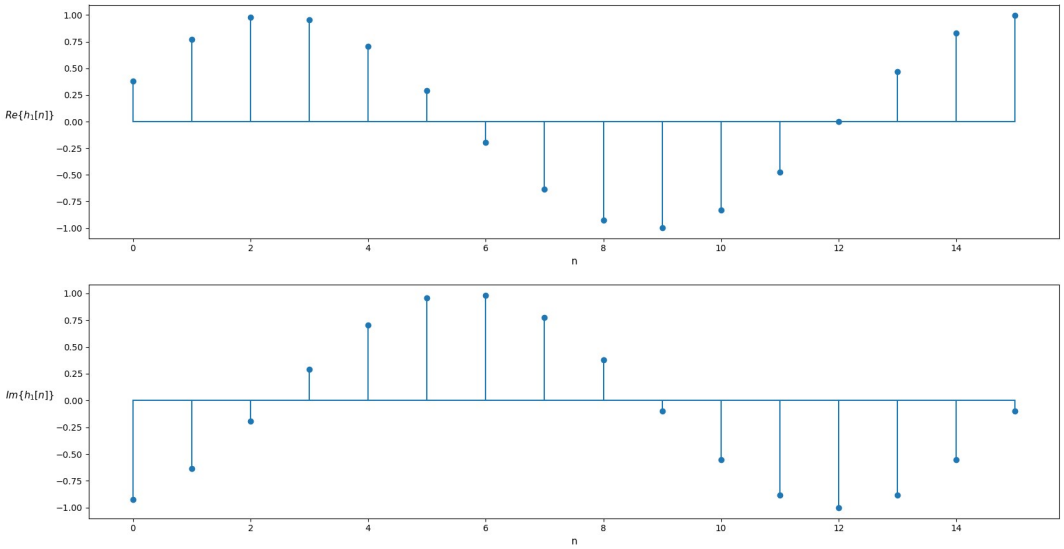


Figure 17: The real part and imaginary part of $h_1[n]$, which is the sampled impulse response of the filter matched to a 1. The length of $h_1[n]$ is 1 symbol or 16 samples.

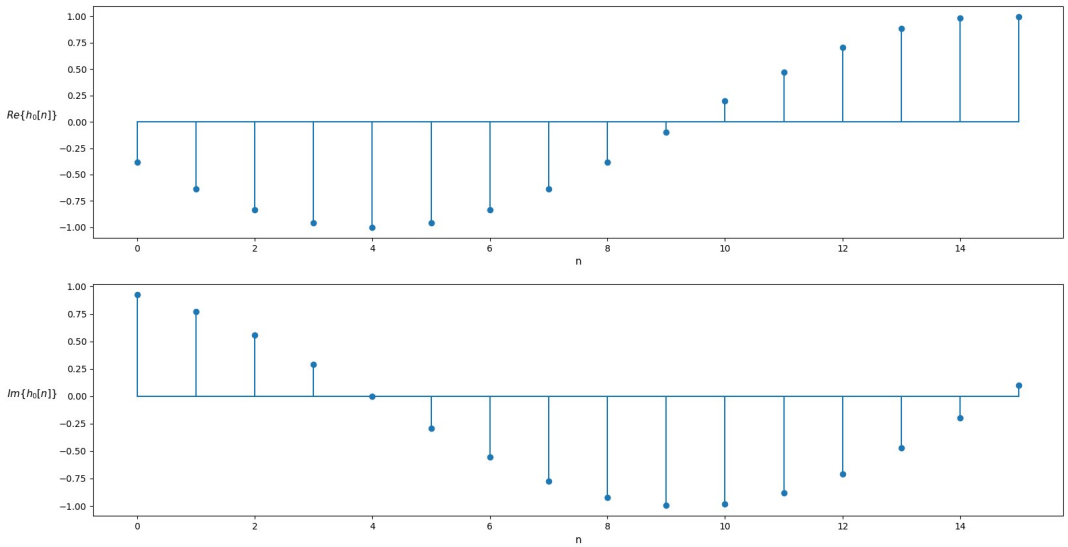


Figure 18: The real part and imaginary part of $h_0[n]$, which is the sampled impulse response of the filter matched to a -1. The length of $h_0[n]$ is 1 symbol or 16 samples.

Furthermore, the magnitude of the frequency responses of $h_1[n]$ and $h_0[n]$ are shown in respectively fig. 19 and fig. 20, and are denoted $|H_1(f)| = |\text{DTFT}\{h_1[n]\}|$ and $|H_0(f)| = |\text{DTFT}\{h_0[n]\}|$. These figures also show how $H_1(f)$ and $H_0(f)$ are sampled when computed using the sDFT. The sampled frequency responses are denoted $H_1[k]$ and $H_0[k]$, and have a length of 16 samples.

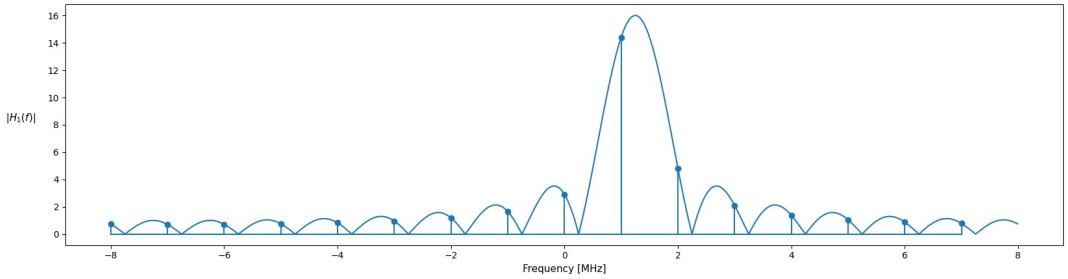


Figure 19: The magnitude of $H_1(f)$, which is the frequency response of the filter matched to a 1. The figure also shows the magnitude of the sampled frequency response $H_1[k]$, which has 16 samples.

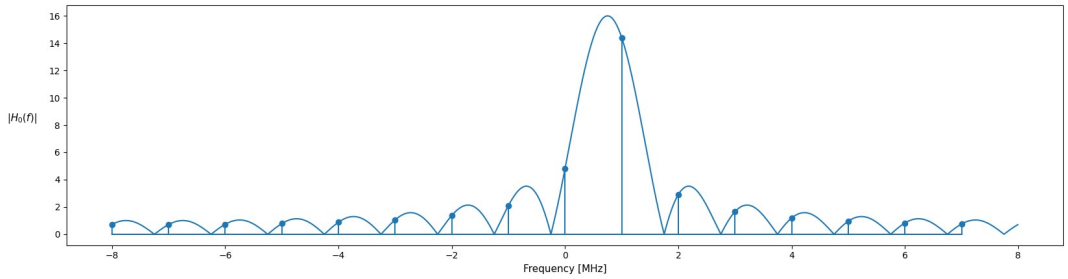


Figure 20: The magnitude of $H_0(f)$, which is the frequency response of the filter matched to a -1. The figure also shows the magnitude of the sampled frequency response $H_0[k]$, which has 16 samples.

The filtering is performed in frequency domain, and as mentioned in section 2.5, this is done by multiplication. This means that the filtered spectrum is given by

$$\hat{Y}_i[k] = H_i[k]S_n[k], \tag{56}$$

where $H_i[k]$ is the sampled frequency response of the matched filter for symbol i and $S_n[k]$ is the sDFT of $s[n]$. Note that we write $\hat{Y}_i[k]$ instead of $Y_i[k]$ to indicate that $y_i[n] \neq \text{IDFT}(\hat{Y}_i[k])$ due to aliasing from circular convolution.

More specifically, each of the filters in the filterbank are implemented as shown in fig. 21, which is mainly just an implementation of (56). However, fig. 21 also includes a division by 2^f in order to ensure proper scaling of the resulting fixed-point numbers, as explained in section 2.8.1.

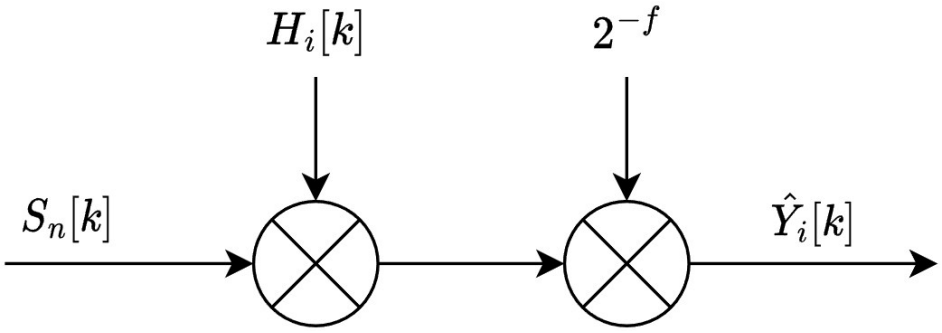


Figure 21: The implementation of the sDFT filter, where the filtering is performed using a multiplication. The product of this multiplication is divided by 2^f in order to ensure proper scaling of the fixed-point numbers.

Note that the filter coefficients $H_i[k]$ are computed in Python and given to the SystemVerilog module as a set of parameters. This is done because the coefficients are constant, so they do not require extra logic to be computed in the SystemVerilog module.

3.5.3 Inverse sDFT

After filtering, the filtered spectra from the two matched filters are sent to the next submodule in fig. 15, which computes the inverse sDFT. As mentioned in section 2.5, the last sample in the filtered sequence does not contain aliasing, so this is the one we want to reconstruct. Therefore, the inverse sDFT is adapted to reconstruct the Nth sample, where N is the window size of the sDFT. This sample is given by

$$y_i[N-1] = \frac{1}{N}\sum_{k=0}^{N-1} \hat{Y}_i[k]e^{j2\pi k(N-1)/N}. \tag{57}$$

In order to save hardware resources, the multiplication by $e^{j2\pi k(N-1)/N}$ is performed in the filter instead of in the inverse sDFT. The filters are then given the coefficients $H_i[k]e^{j2\pi k(N-1)/N}$ instead of just $H_i[k]$. Since the filter coefficients are computed in Python, this reduces the

number of multiplications in the SystemVerilog module by one per frequency bin per filter. The resulting filter output is then given by

$$\hat{Y}_i[k] = S_n[k]H_i[k]e^{j2\pi k(N-1)/N}, \tag{58}$$

which means the inverse sDFT is given by

$$y_i[N-1] = \frac{1}{N}\sum_{k=0}^{N-1}\hat{Y}_i[k]. \tag{59}$$

As a reminder from section 2.4, the sDFT only needs to reconstruct one sample per sDFT computation. This is because the sDFT uses windows with maximum overlap, meaning that each window is only one sample apart. There, $y_i[n]$ can be reconstructed using (59), and this is implemented as shown in fig. 22. Note that this computation introduces a delay of $N-1$ samples.

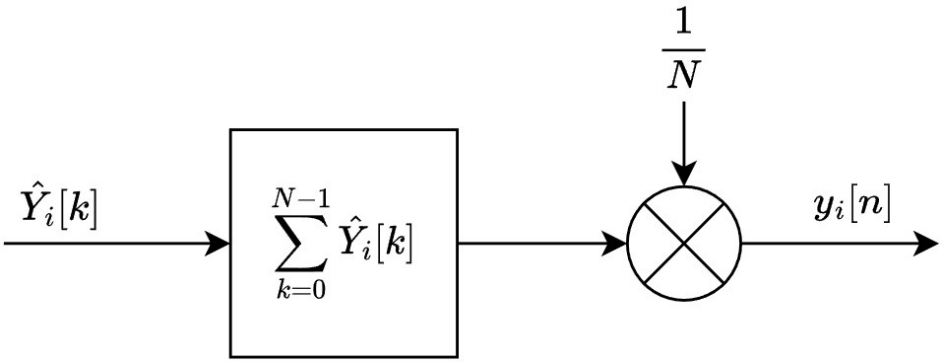


Figure 22: The implementation of the inverse sDFT. The bins of the filtered spectrum given by (58) are summed, before the sum is divided by the number of frequency bins N.

3.5.4 Demodulation decision

As explained in section 2.6, the demodulator can decide which symbol was received by choosing the symbol corresponding to the filter output with the highest magnitude. In this case we only have two symbols, meaning that we can use the difference

$$\Delta y[n] = |y_1[n]| - |y_0[n]|, \tag{60}$$

where $y_0[n]$ and $y_1[n]$, as shown in fig. 15, are the output of the matched filters corresponding to respectively to the symbols -1 and 1. This means that if $\Delta y[n] < 0$, $|y_0[n]| > |y_1[n]|$ and the demodulated symbol will be -1. Similarly, $\Delta y[n] > 0$ means the demodulated symbol will be 1.

Furthermore, $\Delta y[n]$ is averaged over 16 samples in order to improve SNR, where 16 was chosen because this is the number of samples per symbol. Specifically, this means that the averaged $\Delta y[n]$ is given by

$$\overline{\Delta y}[n] = \frac{1}{16} \sum_{m=n-15}^{n} \Delta y[m]. \tag{61}$$

Without noise present, $\overline{\Delta y}[n]$ will look something like in fig. 23. Note the averaging and the way the inverse sDFT is computed introduces a delay, and this delay is compensated for in the figure. Furthermore, the amplitude of $\overline{\Delta y}[n]$ is also scaled in order to give a more readable plot.

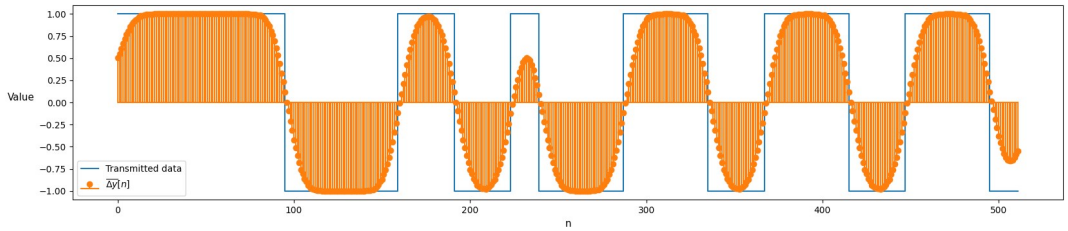


Figure 23: $\overline{\Delta y}[n]$ plotted alongside the transmitted data. Note that $\overline{\Delta y}[n]$ is aligned with the transmitted data and its amplitude is scaled for a more readable plot.

As we can see from fig. 23, the optimal time to make a decision is at the peaks of $\overline{\Delta y}[n]$, because this is where the SNR will be the highest. In a practical application this timing isn't known, but it can be found by using a preamble detector, which is described in section 3.7.

3.5.5 Magnitude estimation

As mentioned in section 2.7, computing the magnitude of a complex number is an expensive operation in hardware. In order to save hardware resources when computing the magnitude of the filter outputs, the demodulator uses the alpha max plus beta min algorithm. This algorithm computes an estimate of the magnitude of a complex number in a way that is much cheaper than computing the true magnitude. More details about this algorithm are described section 2.7.

Based on the results of the tests described in section 4.1, it was found that it is worth testing two different values for α and β for the demodulator. Firstly, the optimal values α_0 and β_0 given by (42) are tested because these generally give the lowest error. For convenience, (42) is repeated here:

$$\begin{aligned} \alpha_0 &= \frac{2\cos(\frac{\pi}{8})}{1+\cos(\frac{\pi}{8})} \approx 0.960433870103 \\ \beta_0 &= \frac{2\sin(\frac{\pi}{8})}{1+\cos(\frac{\pi}{8})} \approx 0.397824734759. \end{aligned} \tag{62}$$

Secondly, $\alpha = 1$ and $\beta = \frac{1}{2}$ are also used since these can be implemented using only additions and bit slicing for division, and therefore require fewer hardware resources than the optimal values.

3.6 Partial spectrum computation

As shown in fig. 16, the bins in the sDFT are computed independently, which means we can save hardware resources by only computing the most important of them. By inspecting the sampled frequency responses of the matched filter in fig. 19 and fig. 20, we see that the magnitude of some of the bins are significantly higher than the rest. More specifically, we see that the five bins highlighted in fig. 24 and fig. 25 contains most of the information for the matched filters. Therefore, one can save hardware resources by only computing these bins, with only a small reduction in performance.

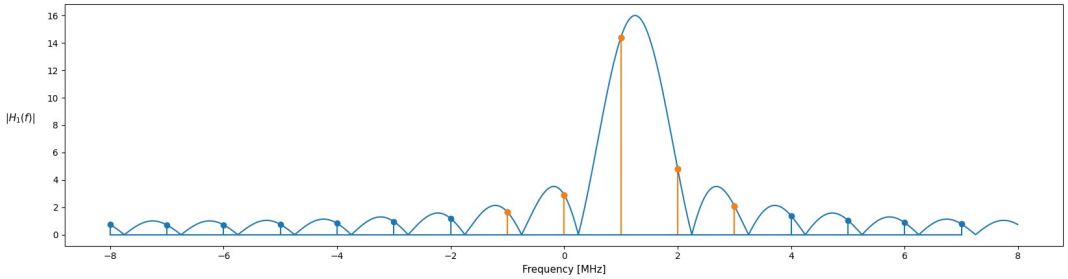


Figure 24: The magnitude of $H_1(f)$, which is the frequency response of the filter matched to a 1. The figure also shows the magnitude of the sampled frequency response $H_1[k]$, and highlights the five bins with highest magnitude.

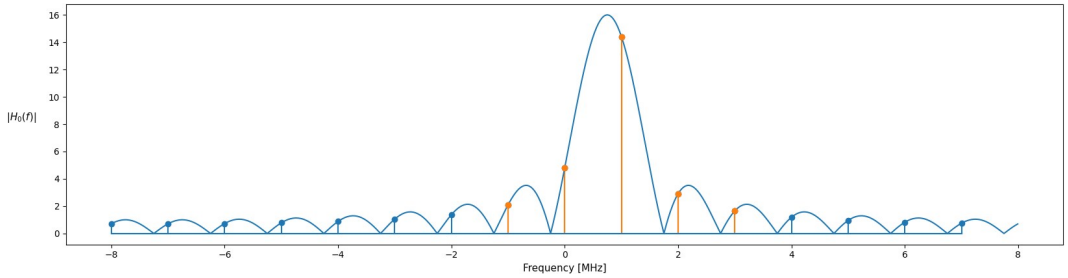


Figure 25: The magnitude of $H_0(f)$, which is the frequency response of the filter matched to a -1. The figure also shows the magnitude of the sampled frequency response $H_0[k]$, and highlights the five bins with highest magnitude.

One can also compute even fewer bins to save even more hardware resources, but at the cost of more performance. One way of doing this is to only use the three bins highlighted in fig. 26 and fig. 27.

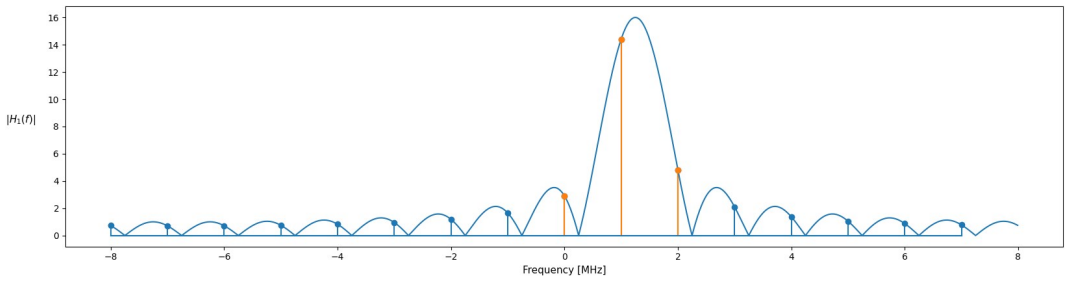


Figure 26: The magnitude of $H_1(f)$, which is the frequency response of the filter matched to a 1. The figure also shows the magnitude of the sampled frequency response $H_1[k]$, and highlights the three bins with highest magnitude.

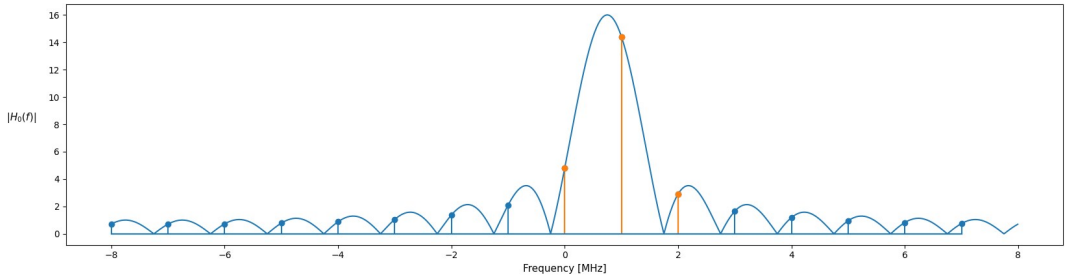


Figure 27: The magnitude of $H_0(f)$, which is the frequency response of the filter matched to a -1. The figure also shows the magnitude of the sampled frequency response $H_0[k]$, and highlights the three bins with highest magnitude.

3.7 Preamble detector

As mentioned in section 3.5.4, the optimal time for the demodulator to make the demodulation decision is at the peaks of $\overline{\Delta y}[n]$. We denote this time as τ, and a preamble detector is implemented to estimate this time as $\hat{\tau}$. The preamble detector is connected to the demodulator as shown in fig. 28.

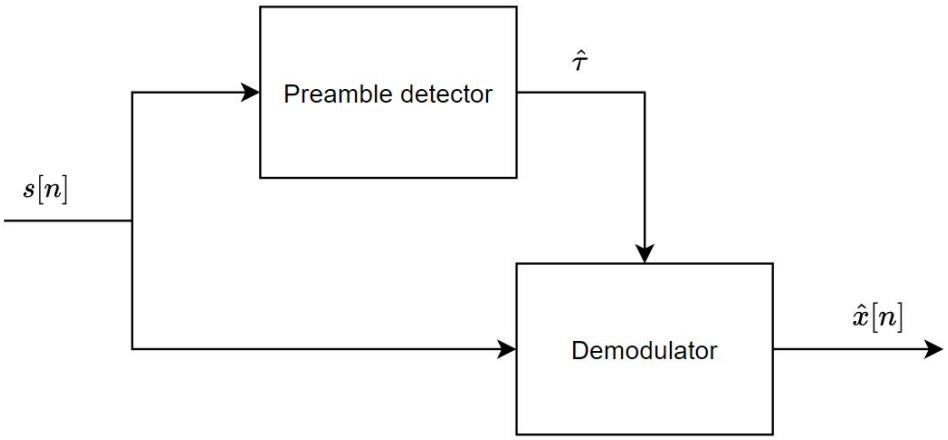


Figure 28: A block diagram of how the implemented preamble detector and demodulator are connected. The preamble detector is used to synchronize the demodulator, which extracts the symbols from $s[n]$.

Furthermore, fig. 29 shows a block diagram of the preamble detector, which computes the estimate $\hat{\tau}$ using a matched filter, averaging and peak detection.

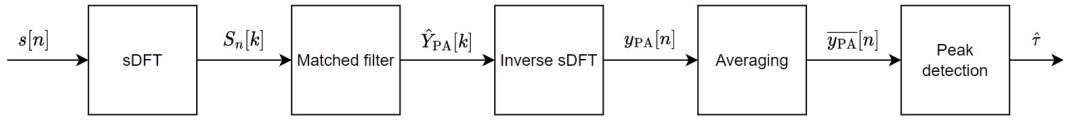


Figure 29: A block diagram of the preamble detector, which computes $\hat{\tau}$ as an estimate for the optimal decision time τ.

The preamble detector builds on a matched filter which is matched to the preamble, where the filtering is done in frequency domain, similar to the matched filters described in section 3.5.2. In other words, the filter is matched to an FSK signal modulating the sequence $x_{\mathrm{PA}}[n] = [-1, 1, -1, 1, -1, 1, -1, 1]$. This FSK signal can then be expressed as

$$s_{\mathrm{PA}}(t) = Ae^{j(2\pi f_{\mathrm{IF}} t + \phi_{\mathrm{PA}}(t))}, \tag{63}$$

where $\phi_{\mathrm{PA}}(t)$ is given by

$$\phi_{\mathrm{PA}}(t) = h\pi \int_{-\infty}^{t} \sum_{n=-\infty}^{\infty} x_{\mathrm{PA}}[n]\Pi(\tau - T)d\tau. \tag{64}$$

The matched filter will then have an impulse response given by

$$h_{\mathrm{PA}}(t) = s^*_{\mathrm{PA}}(-t), \tag{65}$$

which is sampled with the sampling frequency $f_s = \frac{1}{T_s} = 16$ MHz, resulting in

$$h_{\mathrm{PA}}[n] = h_{\mathrm{PA}}(nT_s) = s^*_{\mathrm{PA}}(-nT_s). \tag{66}$$

The sampled impulse response $h_{\mathrm{PA}}[n]$ has a length of 8 symbols or 128 samples, and is shown in fig. 30. Note that this figure shows both the real and imaginary part of $h_{\mathrm{PA}}[n]$, and the only difference between these parts is a 90° phase shift.

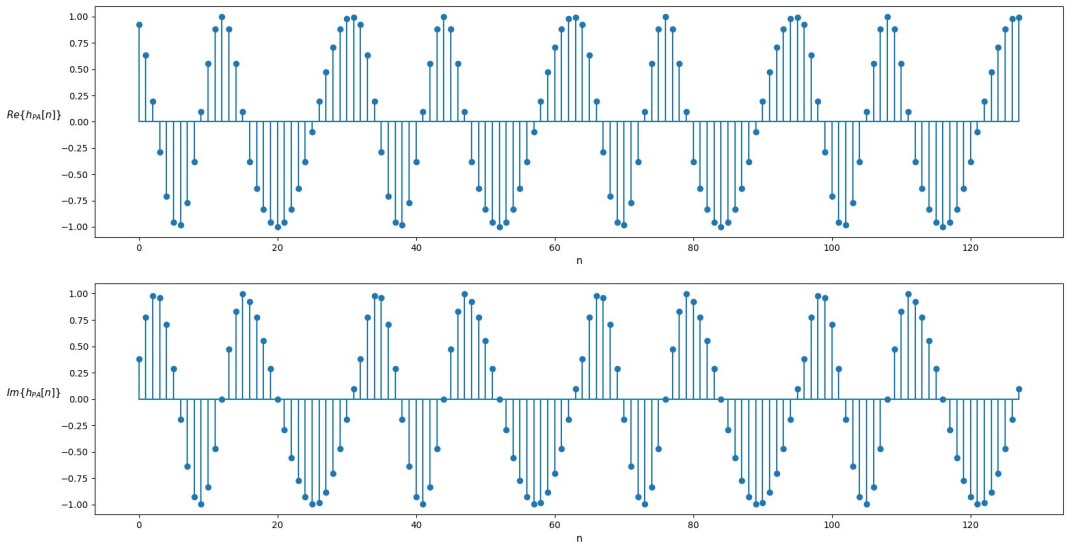


Figure 30: The real and imaginary part of $h_{\text{PA}}[n]$, which is the sampled impulse response of the filter matched to the preamble. The length of $h_{\text{PA}}[n]$ is 8 symbols or 128 samples.

Furthermore, the magnitude of the frequency response of $h_{\text{PA}}[n]$ is shown in fig. 31, and is denoted $|H_{\text{PA}}(f)| = |\text{DTFT}\{h_{\text{PA}}[n]\}|$. This figure also shows how $H_{\text{PA}}(f)$ is sampled for it to be used in a practical application. The sampled frequency response is denoted $H_{\text{PA}}[k]$, which is computed by the sDFT and has a length of 128 samples.

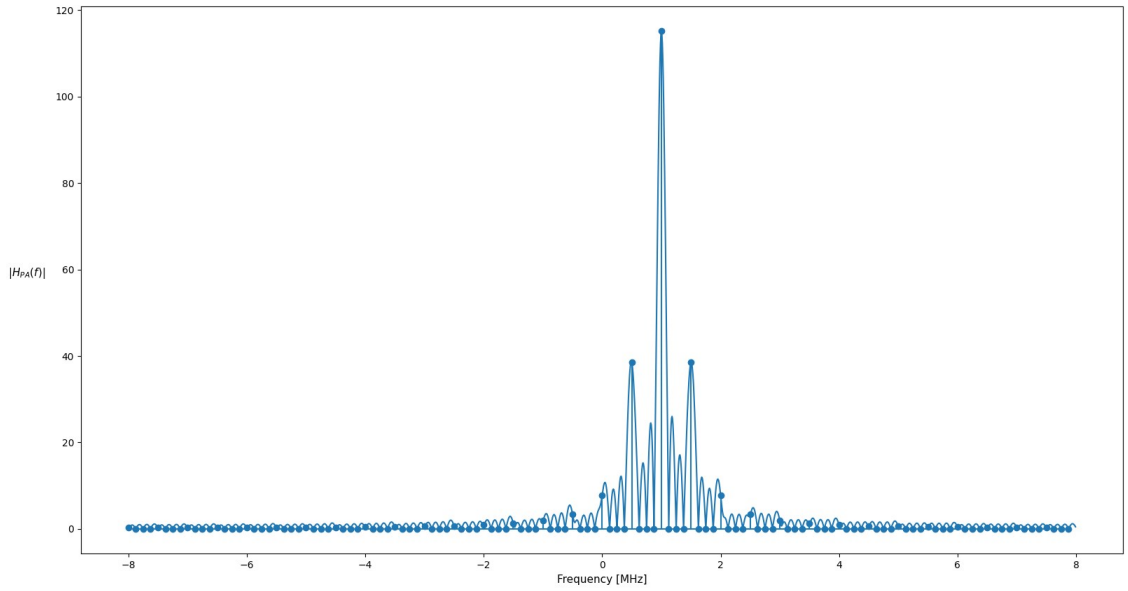


Figure 31: The magnitude of $H_{\text{PA}}(f)$, which is the frequency response of the filter matched to the preamble. The figure also shows the magnitude of the sampled frequency response $H_{\text{PA}}[k]$, which has 128 samples.

As we can see from the frequency response in fig. 31, a few of the frequency bins have a much higher magnitude than the rest. In order to save hardware resources, we only compute

the seven bins highlighted in fig. 32, and thus reducing the number of bins from 128 to seven.

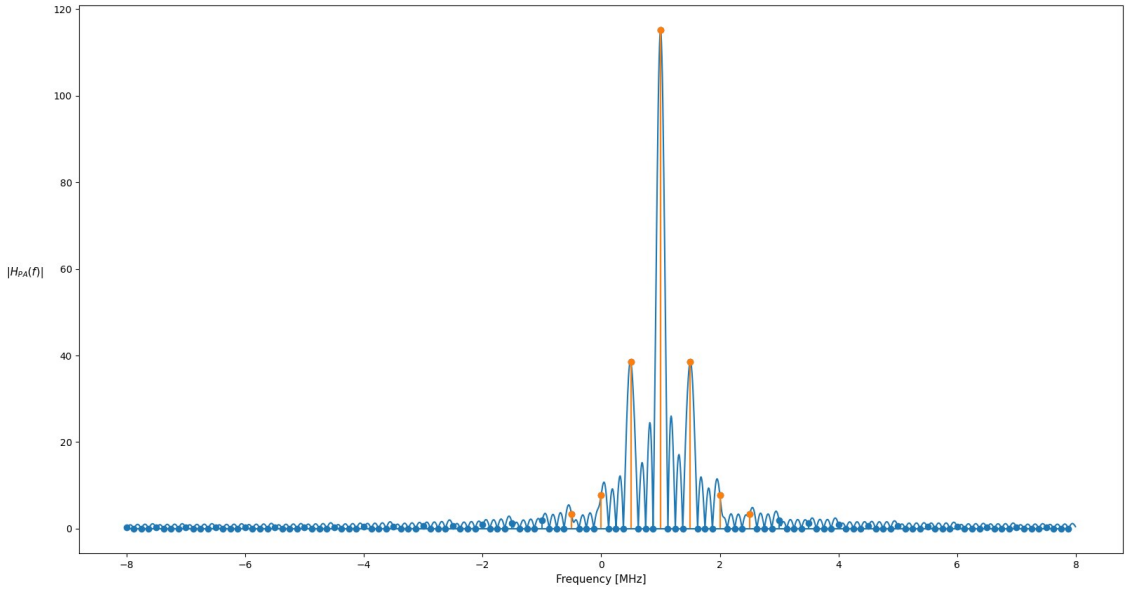


Figure 32: The magnitude of $H_{\text{PA}}(f)$, which is the frequency response of the filter matched to the preamble. The figure also shows the magnitude of the sampled frequency response $H_{\text{PA}}[k]$, and highlights the seven bins with highest magnitude.

After filtering in frequency domain, the inverse sDFT is used to compute the filtered signal in time domain. This signal is denoted $y_{\text{PA}}[n]$, and the magnitude $|y_{\text{PA}}[n]|$ is used to detect the preamble. However, in order to improve the SNR, $|y_{\text{PA}}[n]|$ is averaged over 16 samples. This gives the signal

$$\overline{y_{\text{PA}}}[n] = \frac{1}{16} \sum_{m=n-15}^{n} |y_{\text{PA}}[n]|, \tag{67}$$

where the magnitude is computed using the alpha max plus beta min algorithm described in section 2.7. When no noise is present, $\overline{y_{\text{PA}}}[n]$ looks something like shown in fig. 33.

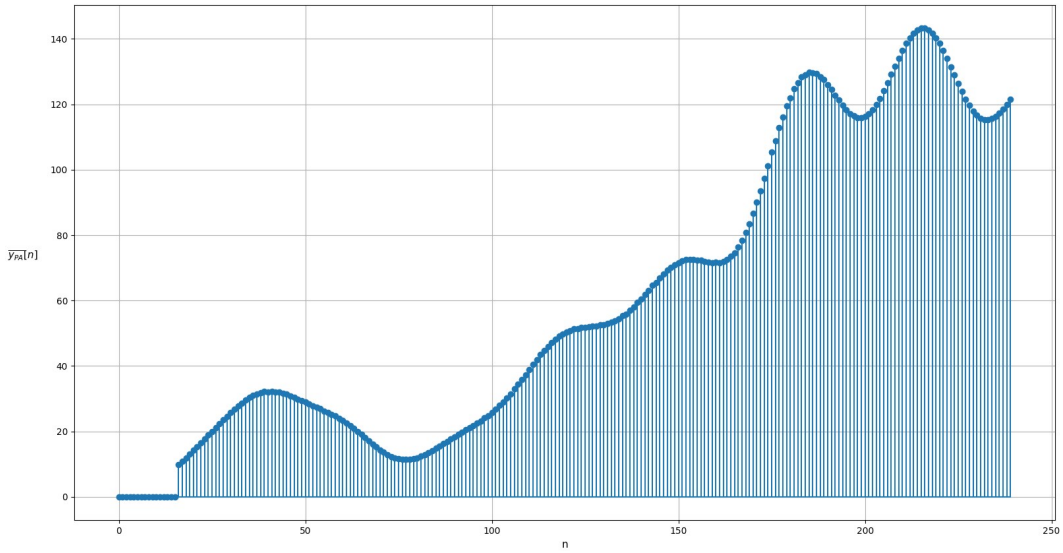


Figure 33: The magnitude of the averaged matched filter output $\overline{y_{\text{PA}}}[n]$.

In fig. 33, the largest peak, which is at around $n = 215$, corresponds to the preamble being detected. However, we also notice that there is another significant peak at around $n = 185$, which is caused by the matched filter detecting a partial preamble. Therefore, it can be quite difficult to separate these two peaks when we do not know when they occur.

Based on this, it is likely easier to detect the slightly smaller peak at around $n = 185$, since there is a steep slope right before it. One way of detecting this peak is by using a threshold that is based on the magnitude of the received signal $s(t)$. Once the $\overline{y_{\text{PA}}}[n]$ gets a value over the threshold, the next peak in $\overline{y_{\text{PA}}}[n]$ will correspond to the preamble.

Through experimenting, it was found that a threshold of $101A$ gives the best result, where A is the magnitude of $s(t)$. This places the threshold in the middle of the slope that precedes the peak at $n = 185$. Since noise can both increase and decrease $\overline{y_{\text{PA}}}[n]$, having the threshold in the middle of the slope gives the most leeway for noise.

Also note that since the value of the threshold depends on the magnitude of the $s(t)$, one would need a module to estimate this magnitude. However, in this thesis we assume that the received signal magnitude is known for the sake of simplicity.

Furthermore, it was found that $\overline{y_{\text{PA}}}[n]$ has quite smooth peaks due to the averaging, even when noise is present. Therefore, it was decided to use a simple peak detection algorithm in order to save hardware resources. This algorithm simply checks when $\overline{y_{\text{PA}}}[n]$ stops increasing after exceeding the threshold, and this point corresponds to the peak.

As a final remark, the preamble detector algorithm presented here might be lacking some details, like for example formulas for the probability of false detections and missed preambles. Since the scope of this thesis focuses on the sDFT, such details falls outside of the scope and are therefore not included. However, should the reader be interested, [11] presents a similar preamble detector where these details are included.

3.8 Time domain implementations

As will be described in more detail in section 4, the sDFT implementations of the demodulator and preamble detector will be compared to time domain equivalents of the same systems. The implementations of the time domain demodulator and preamble detector are described in respectively section 3.8.1 and section 3.8.2.

3.8.1 Time domain demodulator

An overview of the time domain demodulator is shown in fig. 34, which is based around a matched filterbank, just like the sDFT demodulator. The sampled received signal $s[n]$ is filtered using linear convolution, before the filter outputs $y_i[n]$ are compared to decide which symbol was received. Note that $\hat{x}[n]$ denotes the sequence of demodulated symbols.

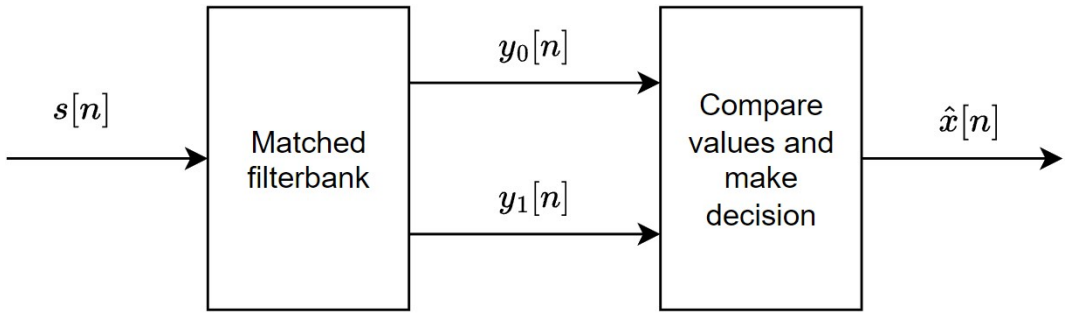


Figure 34: A block diagram of the time domain demodulator. It uses a matched filterbank with one filter for each symbol, where the filtering is performed using linear convolution. The filter outputs are then compared in order to decide which symbol was received.

Figure 34 shows that the only difference from the sDFT demodulator is how the filtering is performed, which is done using linear convolution for the time domain implementation. As shown in (27), this means that the filter output is given by

$$y_i[n] = s[n] * h_i[n] = \sum_{m=-\infty}^{\infty} s[m]h_i[n-m], \tag{68}$$

where $h_i[n]$ is the impulse response of the matched filter corresponding to symbol i. The impulse responses for the filters in the matched filterbank in fig. 34 are shown in fig. 17 and fig. 18.

3.8.2 Time domain preamble detector

An overview of the time domain preamble detector is shown in fig. 35. This implementation is also very similar to its sDFT counterpart, and the only difference is how the filtering is performed. Just like for the time domain demodulator, the filtering is done using linear convolution, as shown in eq. (68). The impulse response of the filter used here is shown in fig. 30.

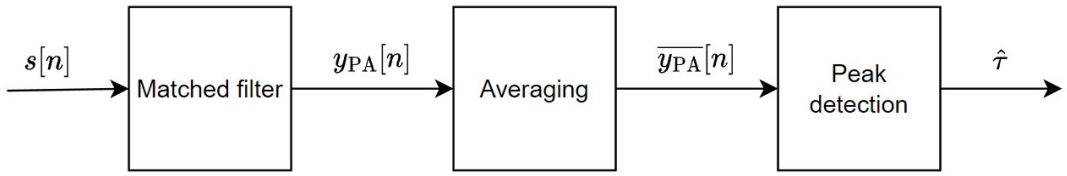


Figure 35: A block diagram of the time domain preamble detector. It is based around a matched filter which is matched to the preamble, and computes the estimate $\hat{\tau}$ using averaging and peak detection.

4 Verification

This section describes the verification of the system implemented in section 3. Firstly, section 4.1 describes how the performance of the alpha max plus beta min algorithm is evaluated. Furthermore, section 4.2 and section 4.3 describes how the demodulator and preamble detector are verified. Lastly, section 4.4 describes how the noise was generated for the test signals, before section 4.5 presents which tools were used for the verification along with their respective versions.

4.1 Magnitude estimation

As described in section 2.7, the magnitude of a complex number can be estimated using the alpha max plus beta min algorithm. This algorithm can use different values for α and β for a trade-off between accuracy and implementation cost. In order to evaluate which values perform better, the values shown in table 1 are tested using a Python implementation of the sDFT demodulator, which uses the entire spectrum to perform filtering. The performance is measured using the BER of the demodulator, and is presented in section 5.1.

Table 1: The evaluated values of α and β. Note that $\alpha_0 = \frac{2\cos(\frac{\pi}{8})}{1+\cos(\frac{\pi}{8})}$ and $\beta_0 = \frac{2\sin(\frac{\pi}{8})}{1+\cos(\frac{\pi}{8})}$ are the optimal values for α and β, as shown in (42).

α	β
α_0	β_0
1	1/2
1	1/4
1	3/8
15/16	15/32

Furthermore, the BER of the different values for α and β are compared to the BER for the true magnitude $|z| = \sqrt{x^2 + y^2}$, because this is the value the alpha max plus beta min algorithm approximates. The BERs are also compared to the BER of the squared magnitude $|z|^2 = x^2 + y^2$, because this is another way of implementing a magnitude estimator in hardware without computing the square root.

In order to include as few sources of error as possible, these tests assume that the optimal decision timing is known. Furthermore, the magnitude estimation is tested using 10000 different GFSK signals, which each consist of 40000 random symbols.

4.2 Demodulator

The performance of the sDFT demodulator described in section 3.5 is evaluated using its BER in two different scenarios, and the results of this are presented in section 5.2. In the first scenario we assume the optimal decision timing is known, meaning that we know the location of the peaks of $\overline{\Delta y}[n]$ from fig. 23. By using this assumption, we get to measure the performance of just the demodulator decision, without any errors from the preamble detector. In this scenario, the demodulator is evaluated when the filter is using the entire sDFT spectrum, and with just five and three bins, as described in section 3.6. Furthermore, the demodulator uses alpha max plus beta min magnitude estimation as described in section 3.5.5, and it is tested for two different values of α and β, namely $(\alpha = \alpha_0, \beta = \beta_0)$ and $(\alpha = 1, \beta = \frac{1}{2})$.

For the second scenario, the demodulator is evaluated using sub-optimal decision timings. This way we get a measurement of how sensitive the demodulator is to errors from the preamble detector. More specifically, the demodulator is tested for

$$\Delta\tau \in [-4, -3, -2, -1, 0, 1, 2, 3, 4].$$

where $\Delta\tau = \hat{\tau} - \tau$ is the difference between used decision time and the optimal one. In this scenario, the filter uses five frequency bins and $(\alpha = 1, \beta = \frac{1}{2})$ for magnitude estimation.

In both scenarios, the demodulator is evaluated using 10000 different GFSK signals, each consisting of 40000 random symbols. The purpose of this is to check how many of the demodulated symbols that are identical to the transmitted ones. Therefore, using random data is sufficient, and we don't need a structure with preamble, address and payload as shown in fig. 11.

Furthermore, the sDFT demodulator is also compared to a time domain implementation of the same demodulator. As described in section 3.8.1, these implementations are identical except for how the filtering is performed, which is done using linear convolution in the time domain implementation and multiplication of two DFTs in the sDFT implementation. The two implementations are compared using both performance in terms of BER, and implementation cost in terms of hardware area. More specifically, the hardware area is measured in the number of flip-flops and NAND2 equivalents required to synthesize the two systems. Note that a specific type of NAND2 gate is used, but which type is not disclosed due to confidentiality.

Lastly, the performance of the sDFT demodulator will also be compared to a Python implementation of a phase-shift discriminator demodulator. The purpose of this is to compare the various sDFT implementations to the performance of a demodulator that is not based around a matched filterbank. In short, the discriminator computes the angle of the received signal $s(t)$ and checks if this angle is increasing or decreasing. Whether it is increasing or decreasing depends on $\phi(t)$ from (5), and this way we can determine if the received symbol was a -1 or 1. Since this thesis focuses on an sDFT matched filterbank demodulator, further details will not be described, but [12, p. 4] offers further details for those interested.

4.3 Preamble detector

The performance of the preamble detector described in section 3.7 is evaluated using two metrics, where the first one is detection error rate, meaning how often a preamble is not

detected. We define a preamble to be undetected when the error $|\Delta\tau| = |\hat{\tau} - \tau| > 8$, where $\hat{\tau}$ is when the preamble is detected and τ is when it is supposed to be. This definition was chosen because there are 16 samples per symbol, so if the preamble detector misses by more than half a symbol, the preamble is considered undetected.

The second metric describes how well the detector estimates the optimal decision time τ in the cases where a preamble is detected. For this we use the mean squared error (MSE) of the estimated decision time and the true one. More specifically, this is given by

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{\tau}_i - \tau_i)^2, \tag{69}$$

where $\hat{\tau}_i$ and τ_i are respectively the estimated and optimal decision time for packet i, and N is the number of detected preambles. The results of both of these metrics are presented in section 5.3.

During the test, the preamble detector is tested by applying packets with a preamble on its input and checking when it detects the preamble on its output. More specifically, the preamble detector is evaluated using GFSK packets consisting of four leading ones, a preamble consisting of eight symbols and a payload of 20 random symbols, as shown in fig. 36. The tests are performed using 10000 versions of this packet, where the noise and payload are randomized differently in each version.

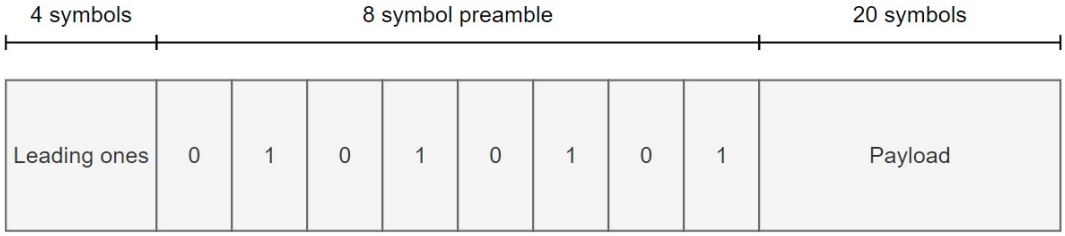


Figure 36: A figure of a single packet used to evaluate the preamble detector. It consists of four leading ones, an eight symbol preamble and a payload with 20 random symbols.

Furthermore, the preamble detector is also compared to a time domain implementation of the same system. As described in section 3.8.2, the only difference between the two implementations is how the filtering is performed, just like for the demodulator. The comparison is also done in the same way as for the demodulator, meaning that both performance and implementation cost is compared. The performance is measured using the metrics described earlier in this section, while the implementation cost is measured using the number of flip-flops and NAND2 equivalents required to synthesize the system.

4.4 Channel noise

To simulate the channel in the communication system, complex white Gaussian noise is added to the signal. The variance of the noise is adjusted to create a signal with a specific $\frac{E_b}{N_0}$, and this is done for

$$\frac{E_b}{N_0} \in [7, 8, 9, 10, 11, 12, 13, 14] \text{ dB}.$$

Furthermore, the noise in the GFSK signals are seeded, meaning that the noise has the same shape but a different magnitude depending on the value of $\frac{E_b}{N_0}$. Note that this applies to each of the 10000 signals used in the tests, meaning that there are 10000 unique noise patterns with a different magnitude depending on $\frac{E_b}{N_0}$. The seeding is done in order to ensure fair and equal test conditions for the different values of $\frac{E_b}{N_0}$.

4.5 Tools for verification

During the verification, various tools are used to perform simulations and tests. More specifically, Python is used to perform the simulations when testing various configurations of alpha max plus beta min magnitude estimation and for the discriminator demodulator. Furthermore, QuestaSim is used to simulate the SystemVerilog implementation of the demodulator and preamble detector, and Synopsys' Design Compiler is used to synthesize and measure the hardware area of these systems. Table 2 shows the version of these tools that were used at the time of verification.

Table 2: The tools used for verification and their respective versions.

Tool	**Version**
Python	3.6.9
QuestaSim	2022.2_1
Synopsys' Design Compiler	1912-sp5-2

5 Results and discussion

This section presents and discusses the results that are used to evaluate the performance of the demodulator and preamble detector described in section 3. Firstly, section 5.1 shows the results related to the performance of the various configurations of the alpha max plus beta min magnitude estimation used in the demodulator. Furthermore, section 5.2 presents the results related to the demodulator, while section 5.3 presents those related to the preamble detector.

5.1 Magnitude estimation

This section presents the results related to how the performance of the magnitude estimation depends on the various values of α and β. As mentioned in section 4.1, this performance is measured using the BER of a Python implementation of the sDFT demodulator which uses the entire spectrum to perform filtering. The BER of this demodulator for various values of α and β is shown in fig. 37. As we can see, the optimal values ($\alpha = \alpha_0, \beta = \beta_0$) gives the lowest error for all values of $\frac{E_b}{N_0}$. Furthermore, the difference in BER between ($\alpha = \alpha_0, \beta = \beta_0$) and the other values shrinks as $\frac{E_b}{N_0}$ increases.

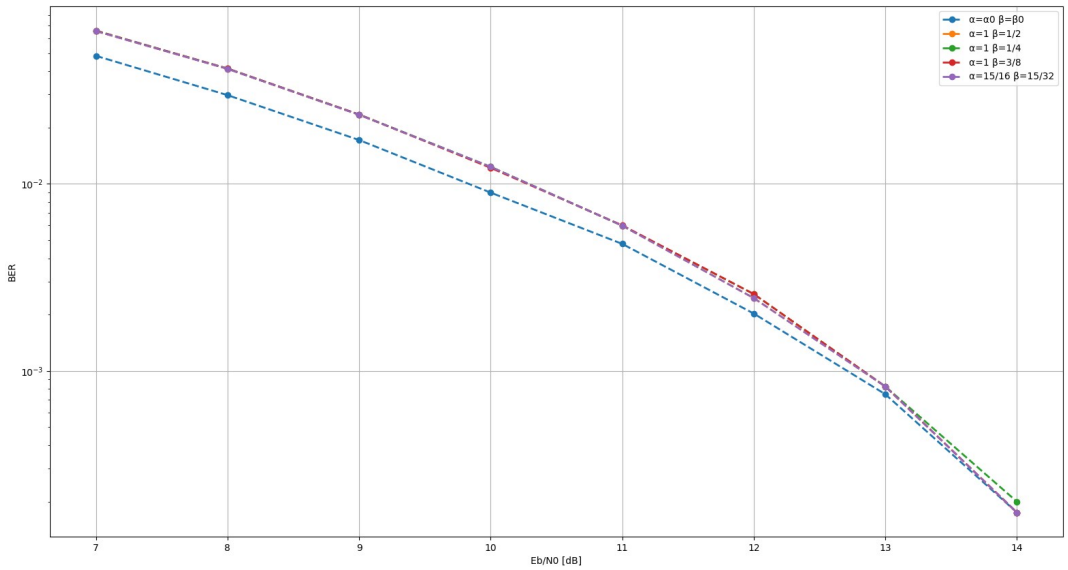


Figure 37: The performance of a Python implementation of the sDFT demodulator using the full spectrum for different values of α and β.

Out of the different values for α and β, $(\alpha = \alpha_0, \beta = \beta_0)$ gives the best performance and are therefore a good choice. Alternatively, if one values hardware resources more, $(\alpha = 1, \beta = \frac{1}{2})$ and $(\alpha = 1, \beta = \frac{1}{4})$ are also good candidates, because dividing by 2 and 4 is cheap in hardware as it can be done using bit slicing. In order to better compare these, fig. 38 shows the BER of these values for α and β, but the y-axis is replaced with

$$\Delta\text{BER} = \text{BER}_{\alpha\beta} - \text{BER}_{|z|},$$

where $\text{BER}_{\alpha\beta}$ is the BER using alpha max plus beta min magnitude estimation and $\text{BER}_{|z|}$ is the BER using the true magnitude $|z| = \sqrt{x^2 + y^2}$. Additionally, fig. 38 also shows the BER of the squared magnitude $|z|^2 = x^2 + y^2$, since this is also a cheap way of computing a magnitude measurement in hardware, and an alternative to the alpha max plus beta min algorithm.

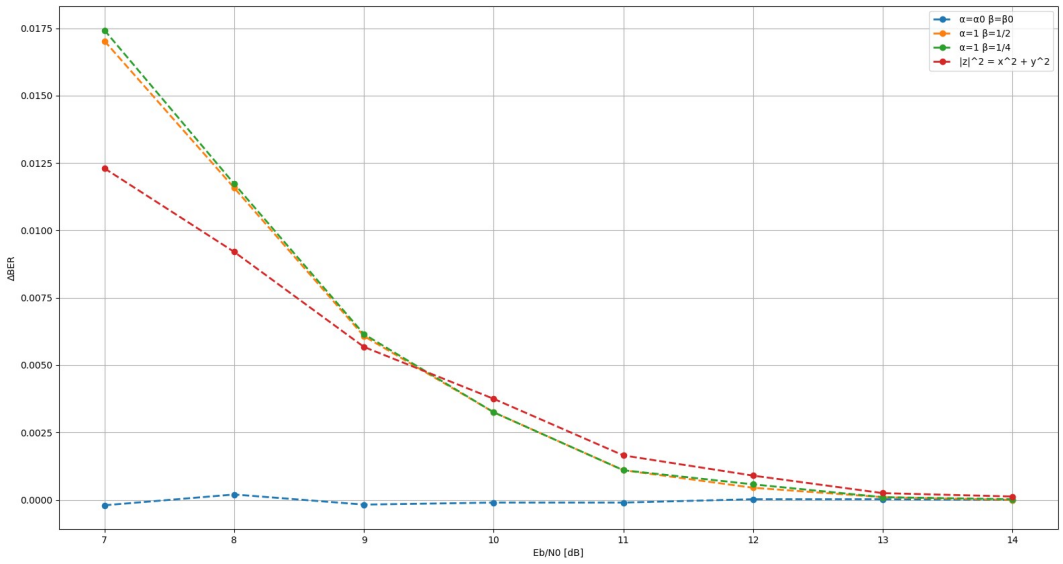


Figure 38: The performance of a Python implementation of the sDFT demodulator using the full spectrum for two values of α and β and the squared magnitude. Note that the y-axis shows the BER relative to the BER using the true magnitude.

As shown in fig. 38, $(\alpha = \alpha_0, \beta = \beta_0)$ gives approximately the same BER as the true magnitude. Furthermore, we see that $(\alpha = 1, \beta = \frac{1}{2})$ and $(\alpha = 1, \beta = \frac{1}{4})$ performs worse than the squared magnitude for low values of $\frac{E_b}{N_0}$ while they perform better for $\frac{E_b}{N_0} > 9$ dB. We also see that the BER of $(\alpha = 1, \beta = \frac{1}{2})$ and $(\alpha = 1, \beta = \frac{1}{4})$ approaches the BER of $(\alpha = \alpha_0, \beta = \beta_0)$ as $\frac{E_b}{N_0}$ increases.

As shown in fig. 9 in section 2.7, the error of $(\alpha = 1, \beta = \frac{1}{2})$ and $(\alpha = 1, \beta = \frac{1}{4})$ is very low when the angle of the complex number we are estimating the magnitude of is close to 90°. Since the system is demodulating an IQ signal, the expected angle is 90° plus some contribution from the noise. As $\frac{E_b}{N_0}$ increases, the contribution from the noise decreases and the angle will be close to 90° more often, which is likely why the BER for all three values of α and β in fig. 38 are roughly the same at high values of $\frac{E_b}{N_0}$.

Based on fig. 38, we can see that $(\alpha = \alpha_0, \beta = \beta_0)$ is the optimal choice when we value performance over hardware resources. This method requires two comparisons, two multiplications and one addition, since the magnitude estimation

$$|z| \approx \alpha \max(|x|, |y|) + \beta \min(|x|, |y|)$$

has to be fully computed. Furthermore, out of $(\alpha = 1, \beta = \frac{1}{2})$ and $(\alpha = 1, \beta = \frac{1}{4})$, $(\alpha = 1, \beta = \frac{1}{2})$ seems to perform slightly better. As mentioned, both of these choices for α and β can perform the multiplication using bit slicing, and therefore only require two comparisons and one addition (in addition to the slicing). Lastly, the squared magnitude $|z|^2$ also has good performance, especially for low values of $\frac{E_b}{N_0}$. However, this method requires two multiplications (squares) and one addition. Therefore, it is just slightly cheaper than $(\alpha = \alpha_0, \beta = \beta_0)$, but has a significantly worse performance.

5.2 Demodulator

This section presents the results related to the SystemVerilog implementation of the demodulator. Section 5.2.1 shows the BER when the optimal decision timing for the sDFT demodulator is known, and compares this to the BER of the time domain implementation. Furthermore, section 5.2.2 presents how the BER is affected by suboptimal decision timings. Lastly, section 5.2.3 shows and compares the synthesis results for both the sDFT and time domain implementations.

5.2.1 BER with known timing

The BER of the SystemVerilog implementation of the sDFT demodulator when the optimal decision timing is known is shown in fig. 39. Note that this figure contains the BER curves for both $(\alpha = \alpha_0, \beta = \beta_0)$ and $(\alpha = 1, \beta = \frac{1}{2})$. For better visibility, fig. 40 and fig. 41 shows the BER for these values of α and β separately.

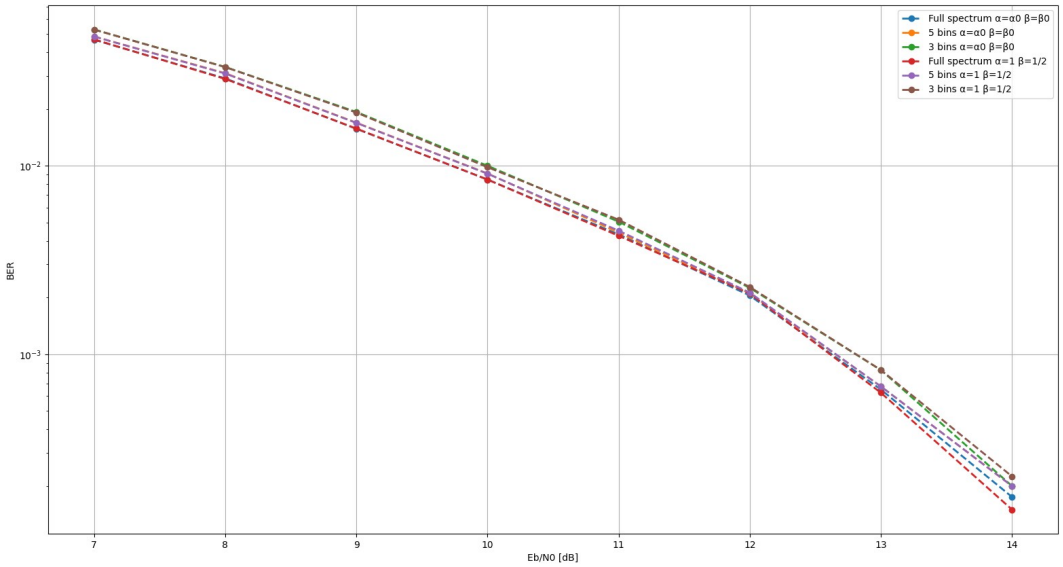


Figure 39: The BER for SystemVerilog implementation of the demodulator when the optimal decision timing is known. The figure shows the BER when the demodulator filters using the full spectrum, five bins and three bins with magnitude estimation using both $(\alpha = \alpha_0, \beta = \beta_0)$ and $(\alpha = 1, \beta = \frac{1}{2})$.

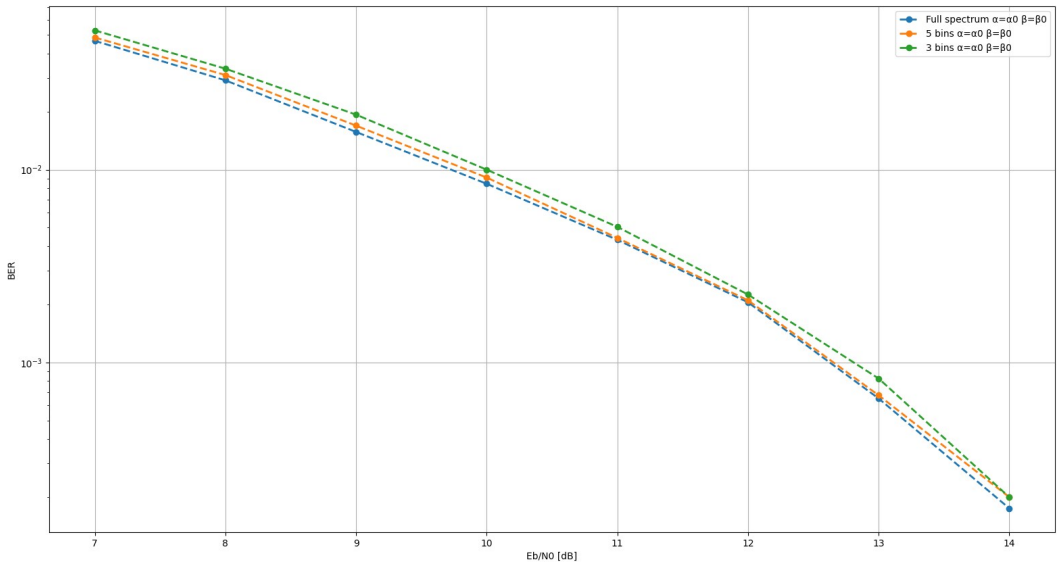


Figure 40: The BER for SystemVerilog implementation of the demodulator when the optimal decision timing is known. The figure shows the BER when the demodulator filters using the full spectrum, five bins and three bins with magnitude estimation using ($\alpha = \alpha_0, \beta = \beta_0$).

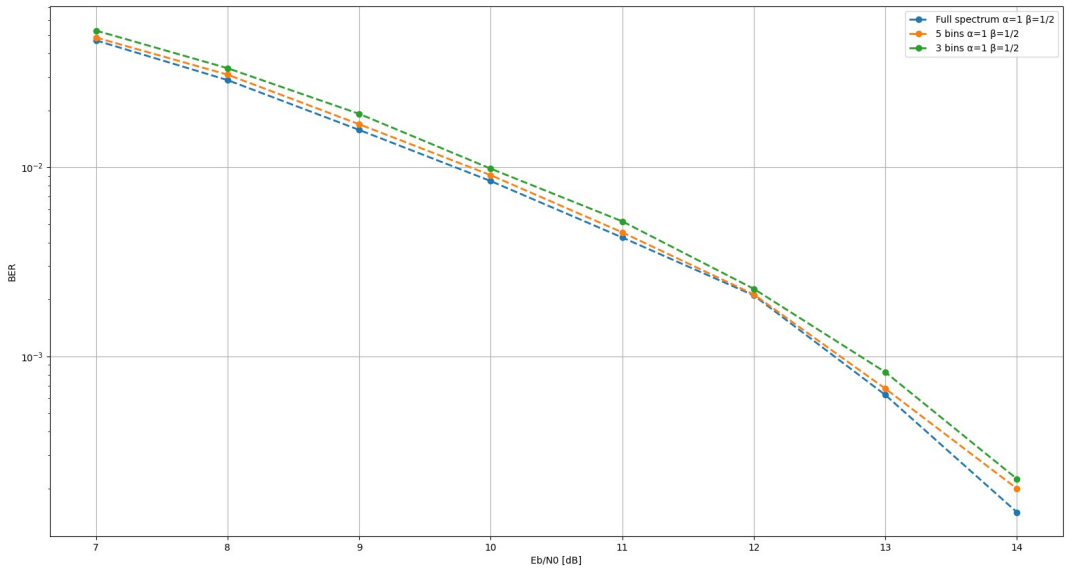


Figure 41: The BER for SystemVerilog implementation of the demodulator when the optimal decision timing is known. The figure shows the BER when the demodulator filters using the full spectrum, five bins and three bins with magnitude estimation using ($\alpha = 1, \beta = \frac{1}{2}$).

From fig. 39, fig. 40 and fig. 41, we see that the BER increases when using five and three bins for filtering compared to using the entire spectrum. This is expected, because using less bins means that we are estimating the ideal matched filter with a slightly suboptimal one. More specifically, using five bins gives an 11% increase in BER for the most affected value of $\frac{E_b}{N_0}$. Alternatively, one can say that the five bin implementation requires 0.19 dB higher $\frac{E_b}{N_0}$ to achieve the same BER as the full spectrum implementation. Similarly, using three bins gives

a 20% increase in BER at the most affected value of $\frac{E_b}{N_0}$, which is equivalent to a 0.29 dB loss in $\frac{E_b}{N_0}$.

Additionally, we can see from fig. 39 that $(\alpha = \alpha_0, \beta = \beta_0)$ and $(\alpha = 1, \beta = \frac{1}{2})$ gives roughly the same performance. In the computation of the sDFT, divisions are implemented using bit slicing, which always rounds down and introduces a rounding error. These rounding errors are likely larger than the inaccuracy introduced by the magnitude estimation, and therefore the choice of α and β does not seem to affect the BER. In the Python implementation of the sDFT demodulator used in section 5.1, there are practically no rounding errors, which is why $(\alpha = \alpha_0, \beta = \beta_0)$ gives a better performance in that case.

Furthermore, fig. 42 also includes the BER of a time domain implementation of the demodulator, in addition to the BER for the sDFT demodulator. All implementations in fig. 42 use $(\alpha = 1, \beta = \frac{1}{2})$ for magnitude estimation.

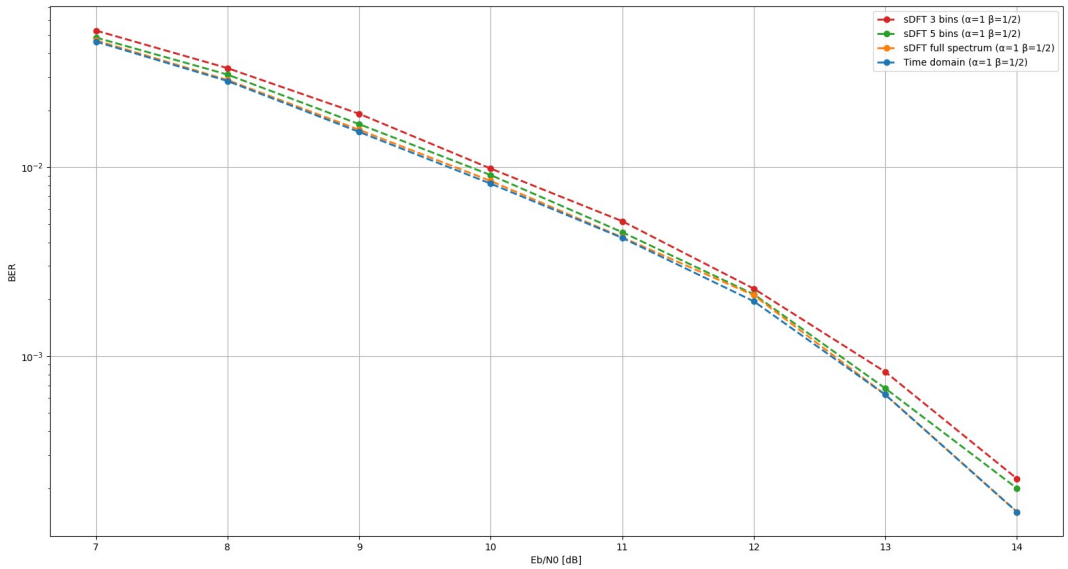


Figure 42: The BER of a time domain implementation of the demodulator plotted alongside the BER of the sDFT demodulator, which filters using the entire spectrum, five bins and three bins. All of the plotted implementations use $(\alpha = 1, \beta = \frac{1}{2})$ for magnitude estimation.

We can see from fig. 42 that the time domain and full spectrum sDFT implementations have approximately the same BER. As explained in section 2.5, performing filtering in time domain using linear convolution is equivalent to performing it in frequency domain using multiplication, as long as one accounts for the aliasing introduced by the multiplication. Therefore, we expect the time domain and full spectrum implementations to have the same BER. The small differnce in BER shown in fig. 42 is therefore likely caused by different amounts of divisions in the two implementations and thus different amounts of rounding errors, since the divisions are done using bit slicing.

Lastly, fig. 43 includes the BER of a Python implementation of a discriminator demodulator. As mentioned in section 4.2, this is an alternative to a matched filter demodulator, and the purpose of including it is to compare the sDFT implementation to a different type of

demodulator. From fig. 43, we can see that the sDFT demodulator performs well compared to the discriminator demodulator. Even when only using three bins, the sDFT demodulator has only a slightly higher BER than the discriminator for $\frac{E_b}{N_0} < 10$ dB, while outperforming it for $\frac{E_b}{N_0} > 10$ dB. Also keep in mind that the discriminator is implemented in Python, meaning that it practically has no rounding errors compared to the SystemVerilog implementation of the sDFT demodulator.

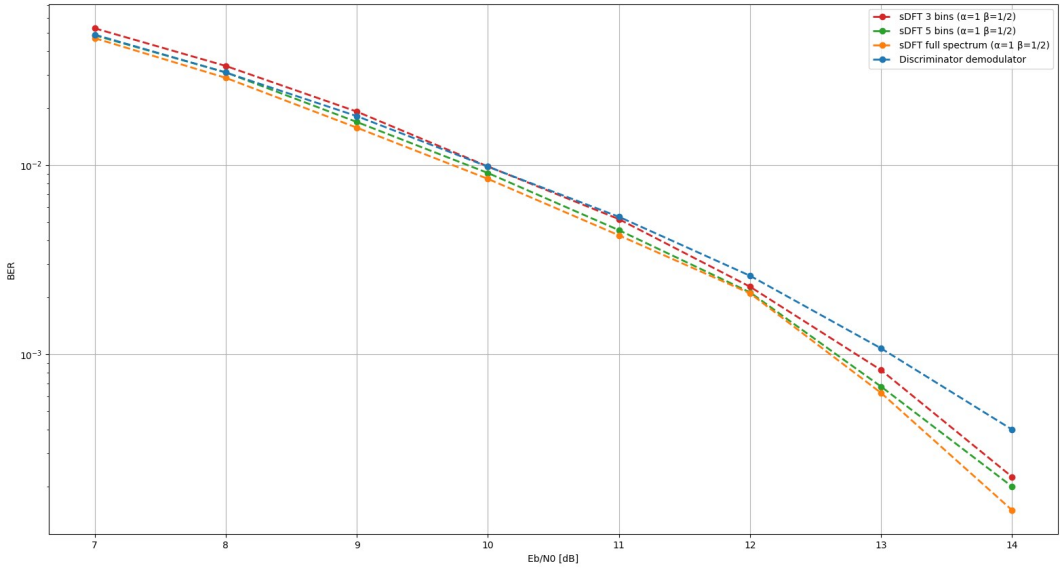


Figure 43: The BER of the sDFT demodulator plotted alongside the BER of a Python implementation of a phase-shift discriminator demodulator. The sDFT demodulator uses $(\alpha = 1, \beta = \frac{1}{2})$ for magnitude estimation.

5.2.2 BER with suboptimal timing

The BER of the SystemVerilog implementation of the demodulator for suboptimal decision timings are shown in fig. 44. The figure shows the BER curves for various values of $\Delta\tau = \hat{\tau} - \tau$, which is the difference between the used and optimal decision time. In this case, the demodulator uses five frequency bins to perform filtering and $(\alpha = 1, \beta = \frac{1}{2})$ for magnitude estimation.

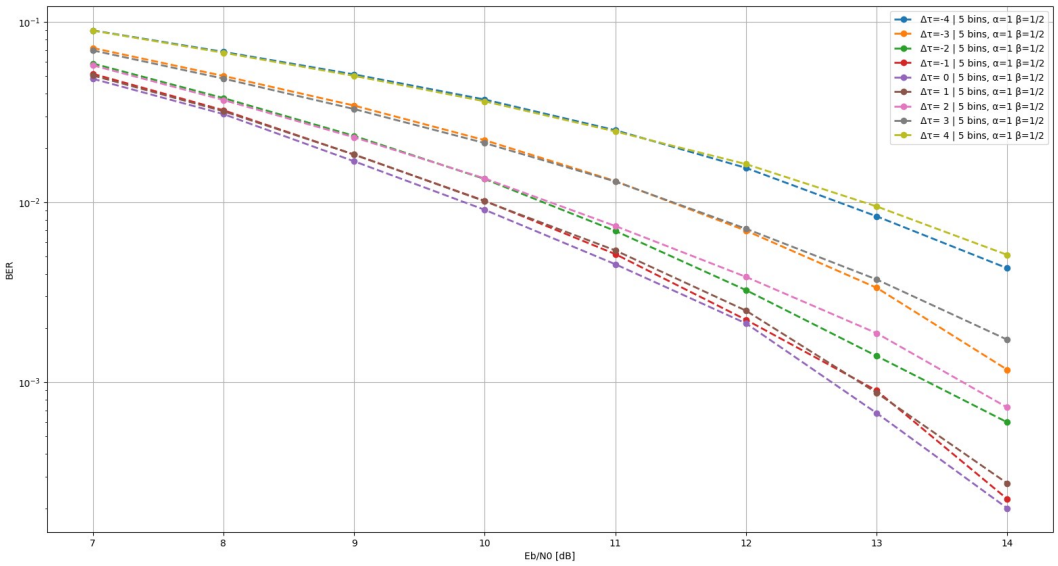


Figure 44: The BER of the SystemVerilog implementation of the demodulator for various suboptimal timings. The demodulator uses five frequency bins to perform filtering and $(\alpha = 1, \beta = \frac{1}{2})$ for magnitude estimation.

As shown in fig. 44, the performance of the demodulator only drops slightly for $\Delta\tau = \pm 1$. However, for offsets that are greater than this, the performance reduces significantly. Therefore, the demodulator is very dependent on the performance of the preamble detector, and can get significant errors if the decision timing estimate from the preamble detector is inaccurate.

Furthermore, we can also see that a suboptimal timing gives a higher increase in BER for higher values of $\frac{E_b}{N_0}$ compared to lower values. This is likely because for high values of $\frac{E_b}{N_0}$, the suboptimal decision timing is the main source of error. In other words, for higher values of $\frac{E_b}{N_0}$, the suboptimal decision timing introduces a higher error relative to the error that is already present.

5.2.3 Synthesis results

This section presents the synthesis results of the sDFT demodulator, where it is synthesized using both the full spectrum and five and three bins for filtering. For comparison, the synthesis results for a time domain version of the same demodulator is also included, where the filtering is performed using linear convolution. The synthesis results are shown in fig. 45 and fig. 46, which respectively show the number of flip-flops and NAND2 equivalents required to synthesize the different implementations. Note that a specific type of NAND2 gate is used, but due to confidentiality, which type is not disclosed.

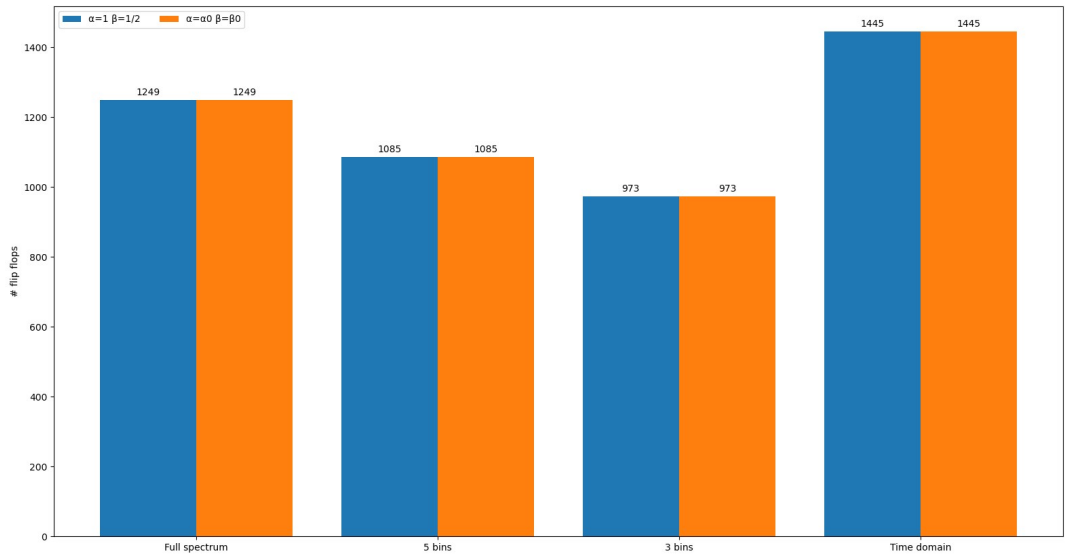


Figure 45: The number of flip-flops required to synthesize the different implementations of the demodulator.

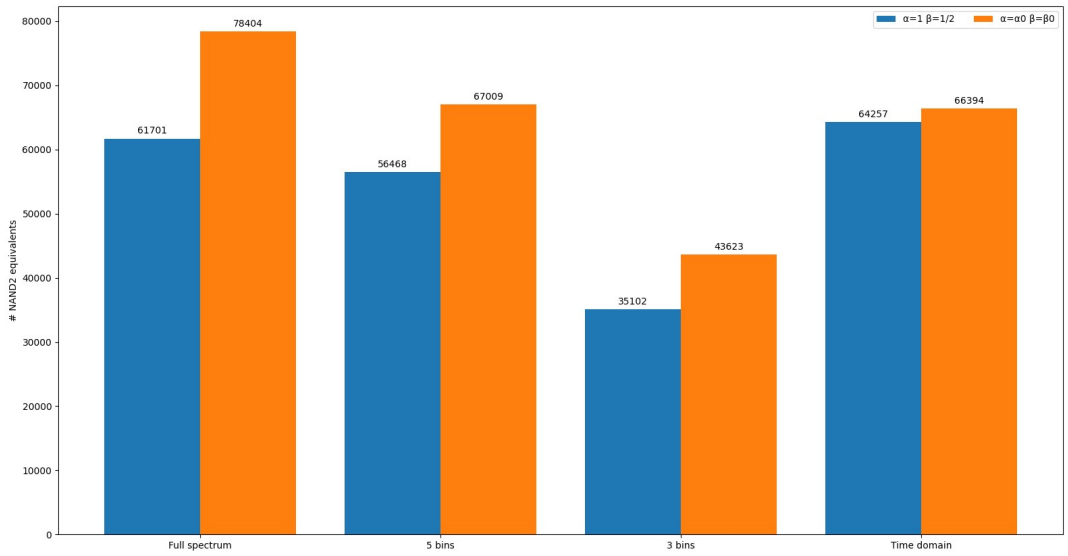


Figure 46: The number of NAND2 equivalents required to synthesize the different implementations of the demodulator.

From fig. 45 we can see that the number of flips-flops depends on the number of bins used for the filtering, since the computed sDFT is stored in a flip-flops, and fewer bins to store means fewer flip-flops. However, the difference in flip-flops between the different implementations is quite small. This is because all implementations use a shift register of the same size to compute the sDFT or perform convolution, and this register is responsible for a significant amount of the required flip-flops. In other words, all four systems have the same baseline cost in terms of flip-flops, caused by this shift register.

We can also see that the value of α and β used in the magnitude estimation does not affect the number of flip-flops. This is because the bit width of the computed magnitude is independent of the values of α of β. Therefore, the values of α and β only affects the combinational logic used to compute the magnitude, and not the registers used to store it.

Furthermore, we can see from fig. 46 that using $(\alpha = \alpha_0, \beta = \beta_0)$ over $(\alpha = 1, \beta = \frac{1}{2})$ for magnitude estimation requires a significantly larger amount of NAND2 equivalents, except for the time domain implementation. Additionally, as discussed in section 5.2.1, the performance gain for using $(\alpha = \alpha_0, \beta = \beta_0)$ is minimal. Based on this, $(\alpha = 1, \beta = \frac{1}{2})$ seems to be the optimal values for magnitude estimation, as they save hardware resources without a notable loss of performance.

Additionally, fig. 46 shows that the sDFT implementations generally require fewer NAND2 equivalents than the time domain implementation. We can also see that there is a small amount of hardware resources saved by using five bins over the full spectrum, and a significant amount saved by using only three bins. In other words, there seems to be a non-linear relationship between the number of frequency bins and the NAND2 equivalents required. This is likely because some of the optimizations the synthesis tool can do require the circuit to be small enough. Therefore we get a significant drop in NAND2 equivalents between five and three bins, as such an optimization is likely possible for three bins, but not for five. However, it is difficult to tell exactly what this optimization is, because this would require inside knowledge of Synopsys' Design Compiler.

5.3 Preamble detector

This section presents the results related to the SystemVerilog implementation of the preamble detector. Section 5.3.1 shows the results related to the performance of both the sDFT and time domain implementations of the preamble detector. Furthermore, section 5.3.2 presents the synthesis results of the same implementations.

5.3.1 Performance

As explained in section 4.3, a preamble is defined as detected when $|\Delta\tau| = |\hat{\tau} - \tau| > 8$, where $\hat{\tau}$ and τ are respectively the estimated and true detection times. The mean squared error for the detected preambles are shown in fig. 47, while the percentage of undetected preambles are shown in fig. 48. Note that fig. 47 and fig. 48 includes both the sDFT and time domain implementations of the preamble detector, where both implementations use $(\alpha = 1, \beta = \frac{1}{2})$ for magnitude estimation. Additionally, the sDFT implementation performs filtering using seven frequency bins.

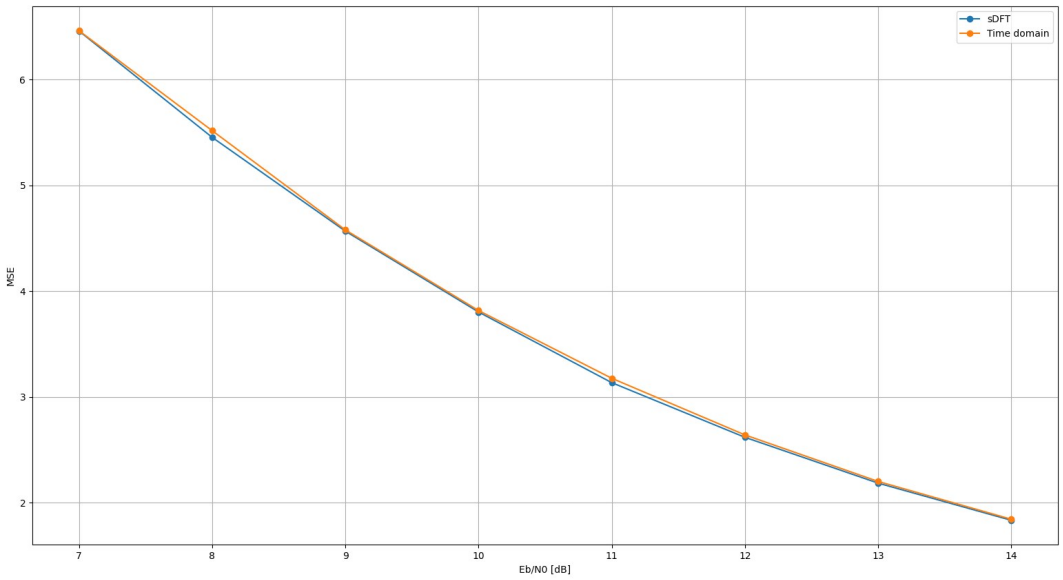


Figure 47: The mean squared error for the detected preambles for the sDFT and time domain implementations of the preamble detector.

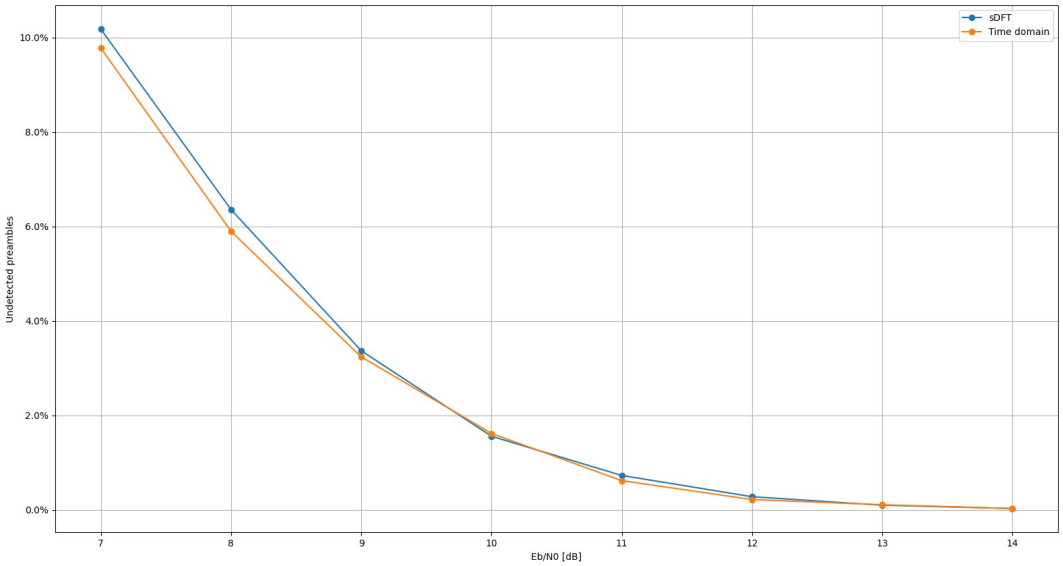


Figure 48: The percentage of undetected preambles for the sDFT and time domain implementations of the preamble detector.

Based on fig. 47 and fig. 48, we can see that the performance of both the sDFT and time domain implementations are almost equivalent. The only notable difference is at the lower values of $\frac{E_b}{N_0}$ in fig. 48, where the number of undetected preambles for the sDFT implementation is increased by 7.8%, relative to the time domain implementation, at most affected value of $\frac{E_b}{N_0}$. This corresponds to a 0.15 dB loss in $\frac{E_b}{N_0}$.

Furthermore, from fig. 47 we can see that the MSE is fairly low for the preambles that are detected, especially for higher values of $\frac{E_b}{N_0}$. This is important, because, as mentioned in

section 5.2.2, the demodulator is very sensitive to suboptimal decision timings, which means the preamble detector needs to be accurate. Also note that the maximum possible MSE is 64, because a preamble is considered undetected, and not included in the MSE calculation, if $|\Delta\tau| > 8$.

However, from fig. 48, we can see that for lower values of $\frac{E_b}{N_0}$, the preamble is undetected for a significant amount of the packets used in the test for both implementations. This is likely a result of the simplicity of the algorithm used to detect the preamble. This algorithm finds the first peak after the filter output exceeds a threshold, and the peak detection used for this is very simple. Therefore, if the noise is strong enough to make the filter output exceed the threshold too early or too late, the peak detection algorithm will still look for the first peak it finds after this happens. To improve this, one might use a more robust type of threshold that is more adaptive than just a fixed value. Alternatively, one might use a more robust peak detection that requires a certain amount of steepness before and after a peak. However, both of these solutions require more hardware resources, so one might also just use the preamble detector as it is, and then accept than one might have to send a packet several times if the noise level is high.

5.3.2 Synthesis results

The synthesis results for the preamble detector are shown in table 3. From this table, we notice that there is only a small difference in the number of flip-flops between the two implementations. Similarly to the demodulator, both implementations use a shift register of the same size to compute the sDFT and perform convolution. This shift register contains 128 $Q(4, 12)$ numbers, which makes it significantly larger than the other parts of the system. Therefore, the majority of the flip-flops that are used in both implementations are used for this shift register, which is why the difference in flip-flops between the implementations is so small.

Table 3: The synthesis results of the preamble detector, showing the number of flip-flops and NAND2 equivalents required to synthesize the module.

Implementation	# flip-flops	# NAND2 equivalents
sDFT	4810	86310
Time domain	4551	258795

We also notice that the sDFT implementation requires about 67% less NAND2 equivalents than the time domain one. As described in section 3.7, the sDFT implementation only uses seven of the 128 frequency bins to perform filtering. The filter used in the time domain implementation has a length of 128 samples, which is why it requires significantly more NAND2 equivalents to synthesize.

6 Future Research

During the thesis, a few interesting aspects had to be skipped due to time constraints. One of these aspects is using a matched filter that is matched to several symbols, for example a filter can be matched to the symbol sequence $x[n] = [-1, 1, 1]$. If the matched filter has a

length of N symbols and the filter outputs are checked each symbol period, each symbol will be filtered N times, meaning there is some overlap. This overlap can be used to perform demodulation more accurately, since the demodulator has N "opportunities" to demodulate each symbol.

However, the downside of this method is that it is more expensive in terms of hardware resources, and some of these extra resources come from the longer filters. Therefore, it would be interesting to test such a system with the sDFT, since a long filter can be implemented using only a few frequency bins, similarly to what is done for the preamble detector described in section 3.7.

Additionally, it would also be interesting to test how the sDFT demodulator performs when there is an offset in the carrier frequency. In a practical application, oscillators are never perfect, so the carrier frequency in the received signal will not be exactly what is expected. In this thesis, it was never tested how the demodulator performs when there is an offset in the carrier frequency, and it would be interesting to quantize how much such an offset affects the BER.

Furthermore, in this thesis it was intentionally chosen to work with signals at the IF to investigate the performance of this. However, most demodulators work with baseband signals, meaning that the receiver contains another mixer than downmixes the received signal from the IF to 0 Hz. It would be interesting to see the difference in performance between operating at the IF and baseband.

Lastly, whether operating at the IF or at the baseband, the mixer contains a lowpass filter to remove an unwanted frequency component generated during the mixing, as shown in fig. 4. Since filtering with the sDFT is performed using multiplications, this lowpass filter can be combined with the matched filters in the demodulator by computing the product of their DFTs. More specifically, the lowpass filter would be removed, and the DFTs of the filters in the matched filterbank of the demodulator in fig. 15 would be multiplied by the DFT of the lowpass filter. This could potentially save hardware resources, since we are removing one filter, and it would be interesting to investigate this further. Note that in this thesis, we assume that the receiver was already implemented. More specifically, the demodulator was given signals from a Python implementation of the receiver, so the lowpass filter in the receiver is not included in the synthesis results.

7 Conclusion

This thesis describes a SystemVerilog implementation of an sDFT based BLE demodulator and preamble detector, and compares these to traditional implementations of the same systems. Both the demodulator and the preamble detector builds on matched filters. For the sDFT implementations, the filtering is done using multiplication in frequency domain, while it is done using linear convolution in time domain for the traditional implementations.

When the sDFT demodulator uses the entire spectrum to perform filtering, it has approximately the same performance as its traditional counterpart. Furthermore, the sDFT demodulator can choose to use only a few bins of the spectrum to perform filtering, in order to reduce the implementation cost with a slight reduction in performance. More specifically, using five out of 16 bins gives approximately a 12% reduction in NAND2 equivalents compared to the

time domain implementation when using ($\alpha = 1, \beta = \frac{1}{2}$) for magnitude estimation. This also increases the BER at the most affected value of $\frac{E_b}{N_0}$ by 11%, which is equivalent to a 0.19 dB loss in $\frac{E_b}{N_0}$. Similarly, using only three out of 16 bins gives approximately a 45% reduction in NAND2 gates, and increases the BER by 20%. This is equivalent to a 0.29 dB loss in $\frac{E_b}{N_0}$. Also note that even when only using 3 bins, the demodulator still has better or approximately equivalent BER to that of a phase-shift discriminator demodulator, which is an alternative to a matched filterbank demodulator, depending on the value of $\frac{E_b}{N_0}$.

The sDFT preamble detector uses a long filter with a length of 128 samples. In frequency domain, this gives 128 frequency bins and only seven of these are used to perform the filtering. This gives a small reduction in performance, but a huge reduction in implementation cost. More specifically, the sDFT preamble detector uses approximately 67% less NAND2 equivalents than the traditional implementation. Additionally, this also increases the number of undetected preambles by 7.8% at most affected value of $\frac{E_b}{N_0}$, which is equivalent to a 0.15 dB loss in $\frac{E_b}{N_0}$.

In short, filtering using the sDFT seems to be more beneficial than filtering using linear convolution, due to the flexibility this offers. More specifically, one can choose to use less frequency bins and potentially save a lot of hardware resources with only a slight loss of performance. This is especially beneficial for longer filters which's information is mainly contained in a few bins in frequency domain, like the filter used for the preamble detector. For filters like this, one can save a lot of hardware resources with only a very small performance loss.

References

[1] A. B. Sæther, "Investigation of sliding window dft (sdft) and comparison with fft," 2022.

[2] J. R. Barry, E. A. Lee, and D. G. Messerschmitt, *Digital Communication*, Third Edition. USA: Kluwer Academic Publishers, 2003. DOI: 10.1007/978-1-4615-0227-2.

[3] D.-C. Chang, "Least squares/maximum likelihood methods for the decision-aided gfsk receiver," *IEEE Signal Processing Letters*, vol. 16, no. 6, pp. 517–520, 2009. DOI: 10.1109/LSP.2009.2016832.

[4] J. G. Proakis and D. K. Manolakis, *Digital Signal Processing*, 4th Editon. Harlow: Pearson Education Limited, 2014.

[5] University of Toronto, *The sliding dft*, https://www.comm.utoronto.ca/~dimitris/ece431/slidingdft.pdf, 2005.

[6] Wikipedia, *Alpha max plus beta min algorithm — Wikipedia, the free encyclopedia*, http://en.wikipedia.org/w/index.php?title=Alpha%20max%20plus%20beta%20min%20algorithm&oldid=1135074387, accessed 21.02.2023.

[7] *Creative commons attribution-sharealike 3.0 unported*, https://creativecommons.org/licenses/by-sa/3.0/deed.en, accessed 17.04.2023.

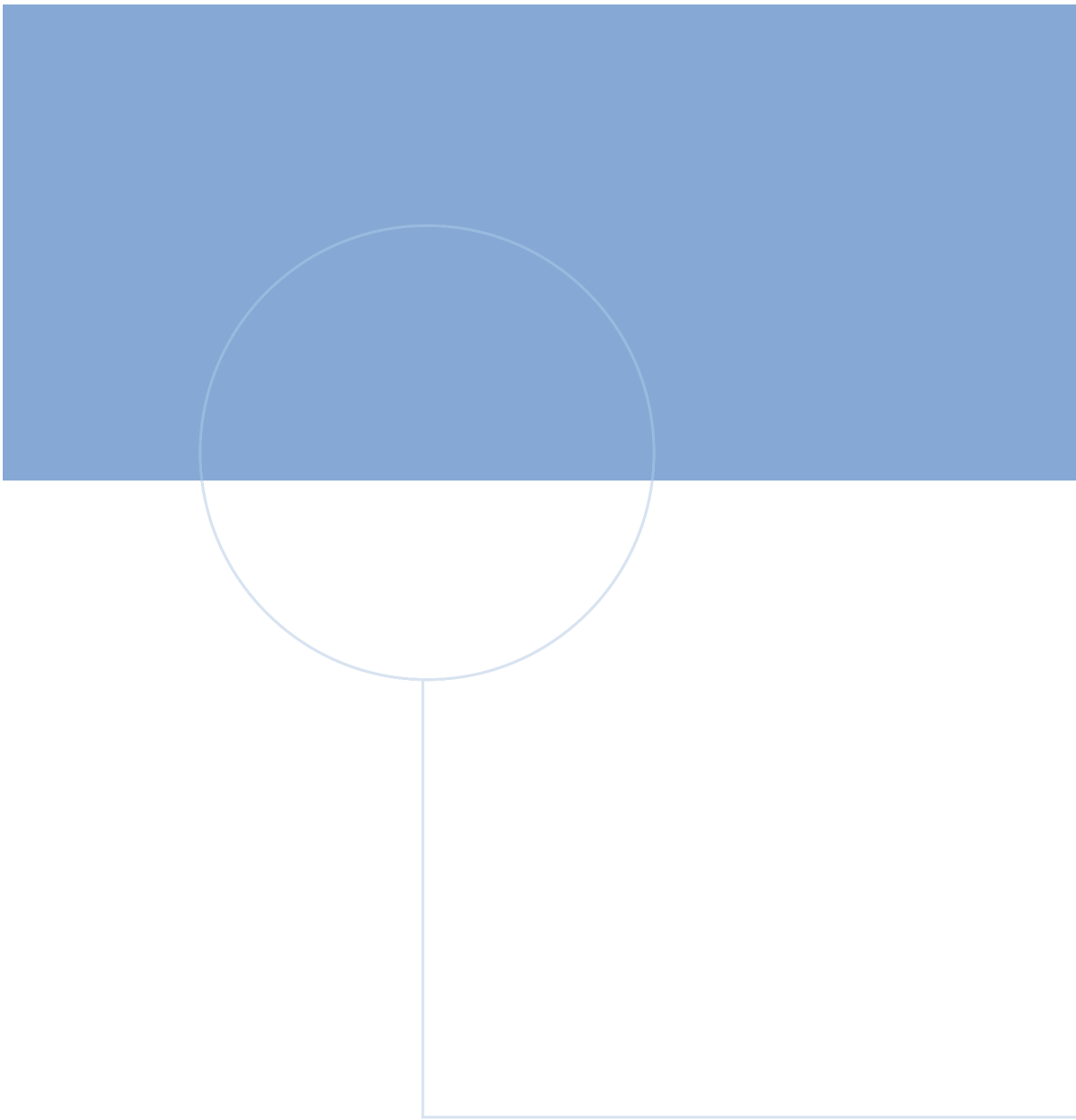
[8] J. Lindh, *Bluetooth low energy beacons*, https://www.ti.com/lit/an/swra475a/swra475a.pdf, 2015.

[9] K. T. Nimisha and P. Biswagar, "Viterbi algorithm based bluetooth low energy receiver for iot," in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, 2017, pp. 978–981. DOI: 10.1109/RTEICT.2017.8256744.

[10] E. Jacobsen, "Understanding and implementing the sliding dft," *Anchor Hill Communications*, 2015. [Online]. Available: https://www.dsprelated.com/showarticle/776.php.

[11] S. Nagaraj, S. Khan, C. Schlegel, and M. V. Burnashev, "Differential preamble detection in packet-based wireless networks," *IEEE Transactions on Wireless Communications*, vol. 8, no. 2, pp. 599–607, 2009. DOI: 10.1109/TWC.2009.071169.

[12] R. Schiphorst, F. Hoeksema, and K. Slump, "Bluetooth demodulation algorithms and their performance," *IEEE Transactions on Circuits and Systems Ii: Analog and Digital Signal Processing*, 2002.



NTNU
Norwegian University of
Science and Technology

NORDIC
SEMICONDUCTOR