Modeling and Simulation of an LF-Receiver

Graz University of Technology

Andreas Pedroß

Signal Processing and Speech Communication Laboratory
Graz University of Technology, Austria

Supervisors
Dipl.-Ing. Dr. Univ.-Doz. Klaus Witrisal
Dipl.-Ing. Stefan Mendel
Dipl.-Ing. Walter Schuchter

Diploma Thesis
June, 2009

Abstract

Correct pressure in car tires avoids accidents and reduces CO_2 emissions. Hence the US government enacted the so called TREAD act in 2000. This act determines that in every new car produced after 2006 a *tire pressure monitoring system* (*TPMS*) has to be included. The Infineon SP3x chips series measures the pressure and temperature in the tires. Because of energy saving reasons these chips are in idle mode until they are woken by an external controller. This wakeup signals are Manchester encoded data messages which are sent with an *on-off-keying* modulation scheme at a carrier frequency of 125kHz. The aim of this thesis is to enhance the performance of the digital LF receiver architecture with low-power and low-complexity techniques.

The functionality and the models of the *air interface*, the *analog front end*, and the *digital baseband processor* of the LF receiver are illustrated. The *bit error rate* (*BER*) and the *message success rate* (*MSR*) of the original receiver are evaluated. Various 1bit filters are proposed which are able to increase the performance of the receiver. Additionally a new chip detector and a new Manchester decoder are introduced. These modifications demonstrate a significantly better *BER* and *MSR* performance than the original receiver. The different architectures are compared with respect to their *BER*, *MSR*, and their complexity in contrast to the original receiver. Also a conclusion and an outlook for further research and development is given.

Kurzfassung

Ein korrekter Druck in Autoreifen führt zu einem geringerem Unfallrisiko und zu einer Reduktion von CO_2 Emissionen. Aus diesen Gründen beschloss die US Regierung im Jahr 2000 den sogenannten TREAD-Act. Dieses Gesetz verpflichtet dass jeder Neuwagen ab 2006 ein *Tire Pressure Monitoring System* (*TPMS*) besitzen muss. Die Infineon SP3x Serie sind Mikrochips zur Messung von Druck und Temperatur in den Reifen. Aus Gründen der Energieeffizienz werden diese Chips in einem Ruhe-Modus betrieben und, bei Gebrauch, von einem externen Kontroller aufgeweckt. Dieses Weck-Signale sind Manchester kodierte Daten, welche mit einem *Ein/Aus-* Modulationsschema bei einer Trägerfrequenz von 125kHz übertragen werden. Das Ziel dieser Arbeit ist die Verbesserung der Effizienz des verwendeten digitalen NF-Empfängers durch energie- und platzsparende Maßnahmen.

Die Funktionalität und das Modell der *Luftschnittstelle*, des *Analogen Eingangs* und des *Digitalen Basisband Prozessors* des NF-Empfängers sind dargestellt. Die *Bitfehlerrate* (*BFR*) und die *Nachrichtenerfolgsrate* (*NER*) des originalen Empfänger wurden abgeschätzt. Verschiedene 1bit Filter, welche die Effizienz des Empfängers verbessern, werden gezeigt. Auch ein neuartiger Chip-Detektor und ein neuartiger Manchester-Dekoder werden eingeführt. Diese Modifikationen zeigen ein stark verbessertes *BFR*- und *NER*-Verhalten als der originale Empfänger. Die *BFR*, *NER* und die Komplexität aller neuen Architekturen werden verglichen. Zum Schluss werden die Ergebnisse zusammengefasst und ein Ausblick auf möglich weitere Forschung und Entwicklung wird gegeben.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, June 2009

Andreas Pedroß

For my Mum ...

... and my Dad.

Contents

1. Introduction

1.1. Overview

We all know and read in the news that many car accidents happens every single day. One reason of these accidents is underinflated tires. About three fourth of all cars have at least one underinflated tire. This can create an overheating of the tire and may lead to accidents. Underinflation is a result of natural leakage, temperature changes, and road hazards. A correct pressure in the tires brings the benefit of a better adhesive force for the car and therefore a better behavior in curves, shorter braking distance, less energy consumption, lower CO_2 emission, a longer lifespan of the tires and the entire car, and lower accident risk.

The United States investigated in the year 2000 with the so called TREAD (Transportation Recall Enhancement, Accountability, and Documentation) Act the implementation of a pressure drop warning system on vehicles. Since the beginning of 2006 all new cars and trucks in the US require a *tire pressure monitoring system* (*TPMS*). TMPS is a system which warns the driver if the pressure in the tires is above or below a prescribed limit.

1.2. Tire Pressure Monitoring Systems

There are two general types of *tire pressure monitoring systems* (*TPMS*): the *direct TPMS* and the *indirect TPMS*. *Indirect TPMS* uses the rotation speed of the tires which is provided from the ABS (anti-lock breaking system), to get an estimate of the tire pressure. If all tires have the same pressure, then they have the same radius and therefore the same angular frequency. If one tire is underinflated, then its radius is smaller and his angular frequency rises, and vice versa. The indirect pressure detection is easy to implement, an ABS presupposed, but has the drawback of rather poor accuracy.

The *direct TPMS* detects the pressure in a different way. It measures the pressure in the tires directly. Therefore the pressure sensor has to be in the tire and the collected information has to be passed to the car. This communication has to be done wirelessly, because cables are not recommend for rotating wheels. So also the energy supply has to be inside the tire. Normally a lithium battery is used which endure the lifespan of the tire: 10 years or more are aimed. Consequently energy consumption is the key criterion for developing a direct *TPMS* system. To send collected pressure values, an ISM (Industry, Science, Medicine) band transmitter is typically used. Because it is not very wise concerning the battery lifetime that the transmitter sends his status for example every minute to the monitoring system, a wakeup feature is reasonable.

1.3. Infineon SP3x Series

The Infineon SP3x series (fig. 1.1) provides a set of pressure sensors. They can measure pressures from 100 to 1600 kPa. They are also capable to measure temperature in a range of -40 to +125 °C and acceleration from -12 to 115 g (g is the acceleration of the earth gravitation field on sea level).

Figure 1.1.: The Infineon SP35 TPMS chip.

The data from the physical sensors are collected by a microcontroller, processed and sent to the monitoring system via a 315/434 MHz ASK/FSK (amplitude-shift-keying/frequency-shift-keying) transmitter.

To save energy, the device is normally in a sleep mode. It can be activated with a wakeup command which is transmitted via a 125 kHz ASK channel. To detect and decode this low frequency (LF) signal, the Infineon SP3x series uses a low energy consuming LF ASK receiver architecture.

1.4. The Digital LF-Receiver

The task of the LF-Receiver at the Infineon SP3x series is to power up the entire chip, when needed. Otherwise the chip remains in idle mode for power saving reasons. To meet the stringent requirements, the LF-Receiver itself has to be designed for lowest possible power consumption. The data rates which are used for data transition to the receiver are 3920baud and 4200baud (therefore approximately 4000baud). To reconstruct the clock of the signal a inverted Manchester code is used. The modulation is an OOK (on-off-keying) scheme at 125kHz . The analog architecture of this receiver is based on a limiter structure to demodulate the data with the help of a *received signal strength indicator* signal. A *digital baseband processor* detects the chips and decodes the data.

1.4.1. Inverted Manchester Code

The *Manchester code* is a self clocking code, which means that the clock can be recovered from the received data stream. To do this, the *Manchester code* has transitions at the half of one data bit. Consequently the chip rate is twice the data rate. The ordinary *Manchester code* codes a binary *One* into a chip sequence of an *One* and a *Zero* chip. A binary *Zero* is coded as a sequence of a *Zero* and an *One* chip. The chips of the *inverted Manchester code* are inverted. Therefore a binary *Zero* is coded as a sequence of an *One* and a *Zero* chip and a binary *One* as a sequence of *Zero* and an *One* chip. This is illustrated in fig. 1.2.

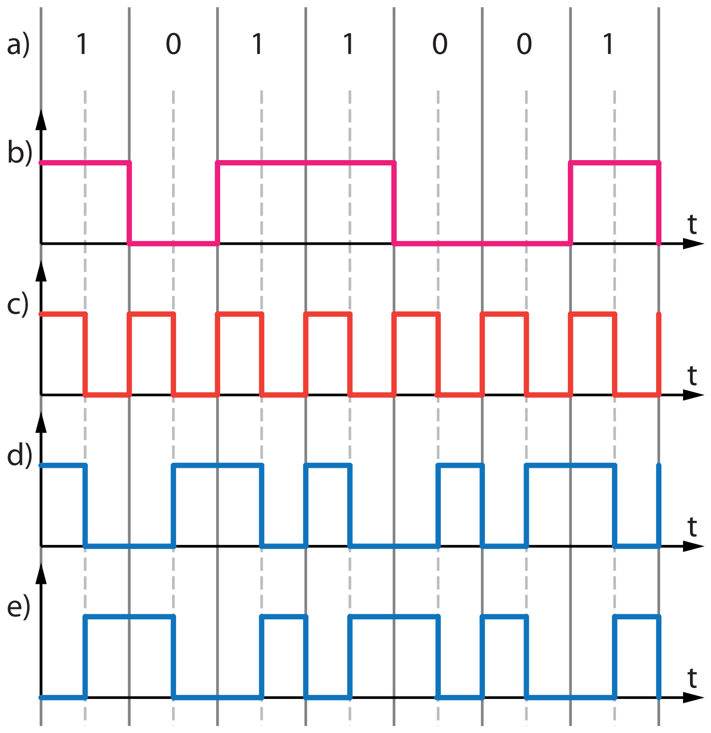


Figure 1.2.: Coding data with the *Manchester code*: a) bits representation of data, b) signal representation of data, c) data clock, d) Manchester coded data, e) inverse Manchester coded.

1.4.2. On-Off-Keying

Amplitude-shift-keying (*ASK*) uses the amplitude of a high frequency signal to modulate data on it [1]. For a set of M symbols exist a set of M amplitudes for modulation. The *ASK* signal can be written as

$$s_i(t) = \sqrt{\frac{2E_i}{T}} \cdot \cos(\omega_0 t + \phi), \tag{1.1}$$

where E_i is the symbol energy of the ith symbol ($i = 1, ..., M$), ω_0 is the carrier frequency, ϕ is the offset phase, and $0 \leq t \leq T$. [1]

The simplest form of a binary *ASK* is called *on-off-keying* (fig. 1.3). One of two symbol energies E_i is set to zero.

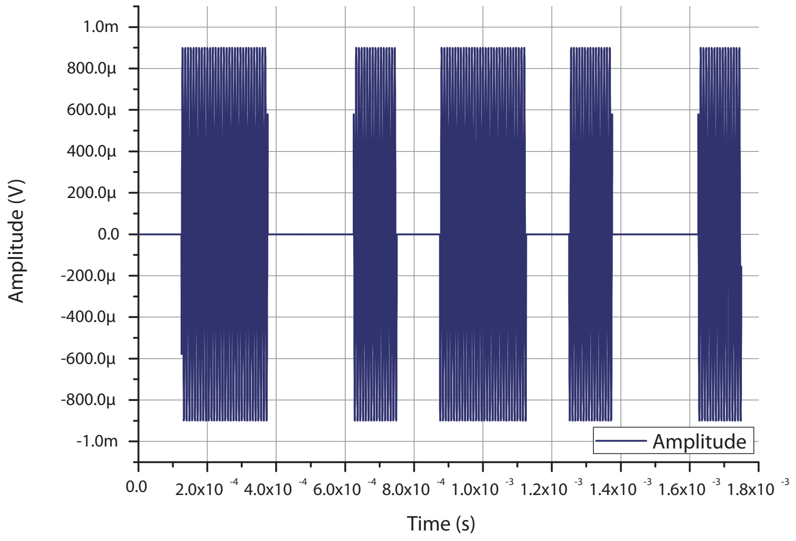


Figure 1.3.: Modulation of data with a data rate of 4kbaud and the *on-off-keying* modulation scheme at a carrier frequency of 125kHz and an amplitude of 0.9mV.

1.4.3. Datagram

A datagram (fig. 1.4) consists of a *preamble*, the *SyncPattern*, and a data message. The *preamble* is a sequence of low-high chips. This is needed to tune all filters within the receiver for a proper operation. The *SyncPattern* is needed, so that the receiver recognize the start of a data transmission. It is a unique chip pattern. Finally the data message is a sequence of Manchester coded bits which are transmitted.

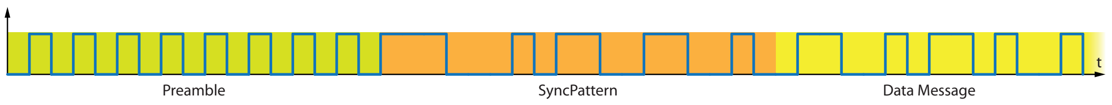


Figure 1.4.: This figure shows the *datagram* of the LF-receiver data link. The green area shows the *preamble*, a sequence of *Zero* and *One* chips. The orange area is the *SyncPattern*, an unique chip sequence that the receiver can recognize a transmission. The yellow area is the Manchester coded data message.

1.5. Objectives

The aim of this thesis is to implement noise filters to increase the performance of the receiver. Various different filters should be found and evaluated. Performance indicators are the *bit error rate* and the *message success rate*. To estimate these indicators the receiver architecture has to be analyzed and a model has to be developed.

1.6. Outline

This thesis is split into five chapters. The first chapter is this introduction. It illustrates the fundamentals of the digital communication link that is used. The second chapter concentrates on the receiver architecture, its functionalities and its models. Also the simulation setting is explained. The third chapter introduces new enhanced detection schemes to improve the performance. Some methods are based on additional filtering. An alternative chip detector, and a novel Manchester decoder are proposed. All approaches enhance the system significantly with the drawback of additional complexity in contrast to the original low-complexity structure. The performance is evaluated and the outcomes are discussed in detail. The fourth chapter summarizes the simulation results and compares them. Also a complexity comparison is done. The fifth chapters concludes this thesis and gives a short outlook for further research and development.

2. Receiver Model

2.1. Overview

The receiver (fig. 2.1) consists of an *air interface*, an *analog front and* (*AFE*) , an *active gain control* (*AGC*) , and a *digital baseband processor* (*DBP*). The *air interface* describes the channel and the antenna of the system. The *analog front end* takes care of the analog signal processing, e.g. the demodulation, and the *digital baseband processor* does the digital signal processing, e.g. chip detection and decoding. In this chapter all these blocks within the receiver processing chain are described in detail.

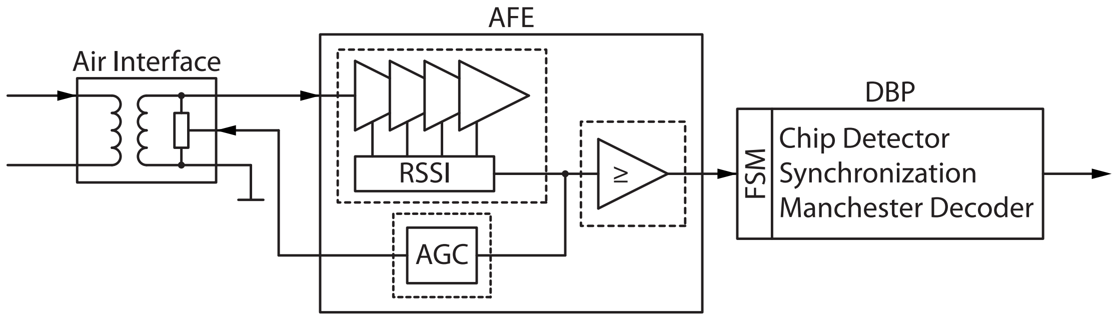


Figure 2.1.: Overall system of the digital receiver.

2.2. Principles of Modeling

Modeling has the aim to describe a real physical system with mathematical tools so that we obtain a set of analytical equations: the mathematical model. A usual system consists of inputs, outputs and functions, which map the input to the corresponding output. In our reality physical signals are always functions of time like temperature, the water-level of a river or the difference between two electric potentials. A system can have a huge range of characteristics: linearity, time variance, memory, and so on. In order to describe a system, various tools are used e.g. transfer functions or systems of ordinary differential equations.

Modeling a system is always a trade-off between the maximal accuracy which is possible and the maximal accuracy which is needed, i.e., the degree of abstraction. It is also important to *understand* the model and its characteristics. Increasing the complexity often leads to systems, which are hard to understand. Sometimes you can describe a system with too much detail which makes the model very complex. But if you don't need such a high degree of detail, it's often better, due to the simulation time or a better understanding of the model, to design a model which just fulfills your needs and not more. On the other hand it is often very difficult, because of lack of information about the real system, to make a model as accurate as you maybe need it.

2.3. Air Interface

2.3.1. Overview

The *air interface* describes the physical channel between the ideal data source and the receiver. Because the signal has a 125kHz carrier, a conventional antenna pair can not be used due to the fact that $\frac{\lambda}{4}$-antennas must have a length of about 600m for the 125kHz band so that an electromagnetic wave can propagate. Therefore a short-range magnetic coupling is used. The coupling is done via mutual inductances of a pair of coils, where each is part of an oscillating circuit (fig. 2.2).

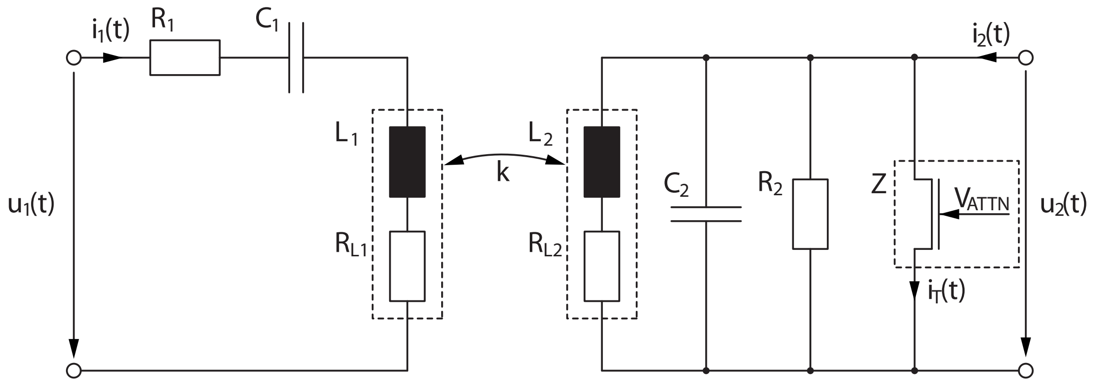


Figure 2.2.: The *air interface* exists of two oscillation circuits which are coupled via mutual inductances.

2.3.2. Functionality

The transmitter feeds the output signal ($u_1(t)$) into an oscillating circuit (app. A.3). The coil L_1 of the oscillating circuit produces a magnetic field. A larger current $i_1(t)$ through the coil generates a stronger magnetic field. Therefore the oscillating circuit has to be a serial RLC circuit. Because the circuit has a resonant frequency at 125kHz and a bandwidth of 16kHz, the signal for transmission is less damped than any other frequencies. This leads to a signal shaping as shown in fig. 2.3.

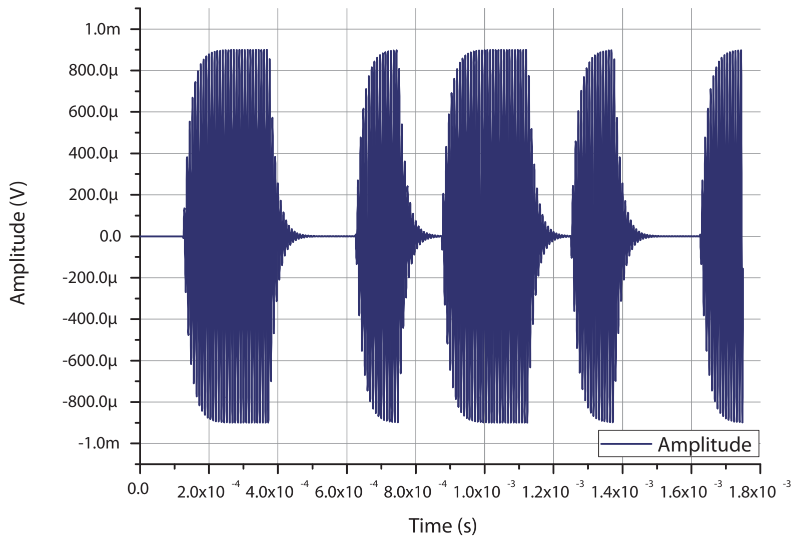


Figure 2.3.: Signal with an amplitude of 0.9mV after passing the *air interface*. The shaping is done by the bandpass behavior.

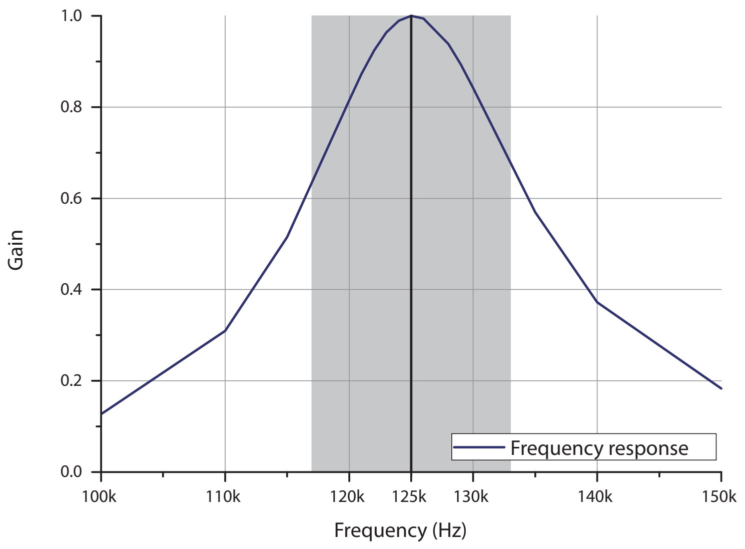


Figure 2.4.: Frequency response of the *air interface*; the gray rectangle marks the passband bandwidth of 16kHz.

A second coil (L_2) is used as an antenna for the receiver. The magnetic field of coil L_1 produces a voltage in any coil within its field. This is done via mutual inductance (app. A.2). Thus the antenna of the receiver also consists of a coil L_2 in an oscillation circuit. Consequently the voltage over coil L_2 represents the received signal and the oscillating RLC circuit has to have a parallel structure. The resonant frequency is 125kHz and the bandwidth is 16kHz, similar to the transmitter. The frequency response of the whole *air interface* system shows a bandpass behavior (fig. 2.4) with a passband of 16kHz at a center frequency of 125kHz.

The so called coupling factor k is a quantity of the magnetic coupling of two coils. A bigger distance between two coils leads to a smaller coupling factor. The mutual inductance is defined as $M = k\sqrt{L_1 L_2}$. So the coupling factor has a big influence to the gain of the *air interface* which can be shown in fig. 2.5. Typically the coupling factor k is very small and therefore, the gain of the circuit is very small.

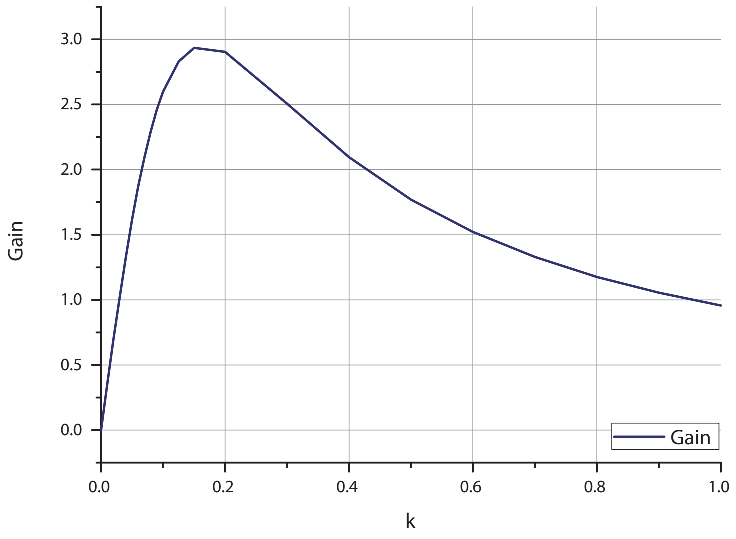


Figure 2.5.: Gain of the *air interface* against the coupling factor k.

The output of the *air interface* is the voltage $u_2(t)$, whose amplitude needs to be in a certain region. To attenuate the output voltage $u_2(t)$, the control voltage V_{ATTN} from the *active gain control* (sec. 2.4.4) sets the resistance of the input impedance Z. The input impedance Z of the receiver is not a linear device, but a nonlinear MOS transistor. To model a wide region of the output characteristic of the transistor the so called EKV model is used (fig. 2.6 and app. B).

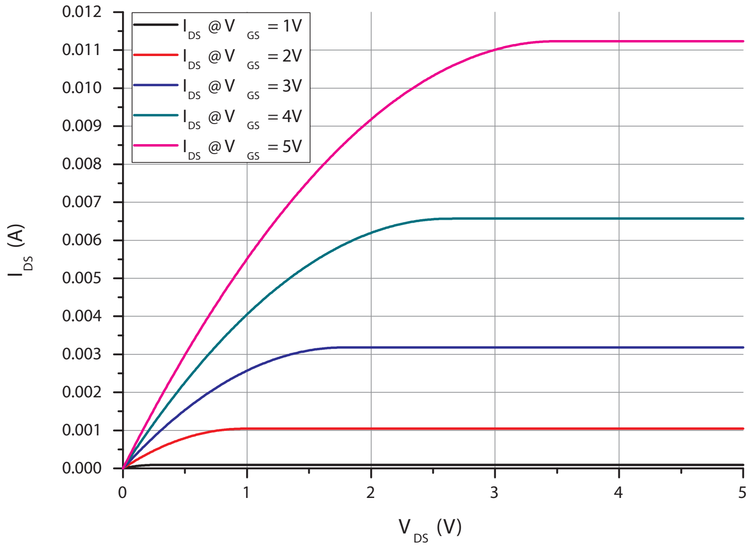


Figure 2.6.: Output characteristic of the EKV MOS transistor model with different gate-source voltages.

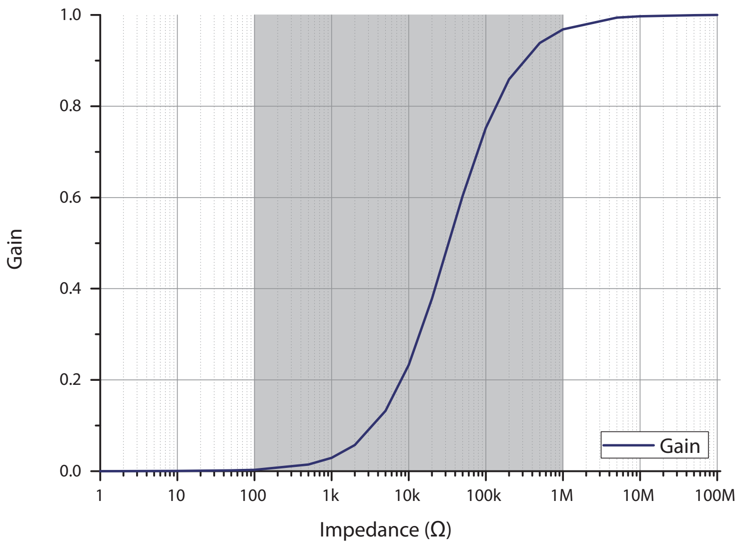


Figure 2.7.: Gain of the *air interface* in dependence of the receiver input impedance; the range of the input impedance is marked as gray rectangle.

The EKV model [2] was designed by Christian C. Enz, Francois Krummenacher and Eric A. Vittoz, all from Swiss Federal Institute of Technology of Lausanne, and first published in 1995. In difference to the standard Shichman-Hodges model [3] it covers the whole region, from weak inversion to saturation, in only one equation and uses much less parameters than the standard BSIM models which are widely used in standard circuit simulation programs like PSPICE [4, 5]. The level-one EKV model uses 11 parameters which can be found in Table B.1 [6] and three general constants which can be found in Table B.2 [7]. The behavior of the attenuation with a MOS transistor is shown in fig. 2.7.

The antenna coils have a nonlinear current behavior, because a current through a coil changes its inductance (fig. 2.8). The higher the current the lower the inductance. This is called saturation effect. If the current through the coil is increasing, then more and more Weiss domains of the ferro magnetic core of the coil are arranged parallel to the magnetic flux lines. This leads to a decreasing inductance of the coil [8]. The term *saturation current* refers to the current through the coil where the inductance has 80% of its maximum value. It should also be mentioned that a real coil always has a nonzero serial resistance. So the impedance Z_L of a real coil can be written as $Z_L = R_L + j\omega L$.

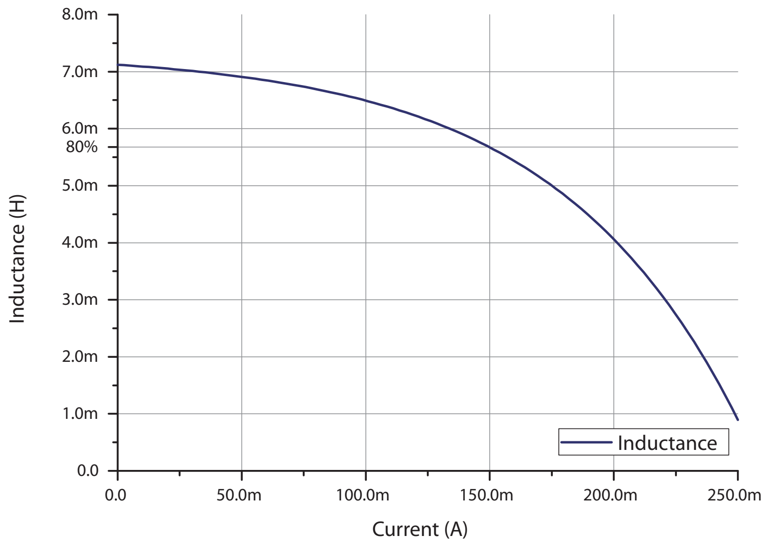


Figure 2.8.: Saturation effect of a 7.1mH coil; the saturation current is at 150mA.

2.3.3. Modeling

The saturation effect caused by the current through a coil can be modeled (curve fitted to a known saturation current curve of an EPCOS 3.9μH inductance [9]) as

$$L(i) = a \cdot \left(1 - e^{\frac{i}{b} - c}\right). \tag{2.1}$$

The voltage produced by the self inductance can be estimated as $u_L(t) = \frac{d(L \cdot i)}{dt}$. That means that a change of inductance generates a voltage. We can approximate this effect by

$$u_L(i) = \tilde{L} \cdot \frac{di}{dt} \tag{2.2}$$

with

$$\tilde{L}(i) = L' \cdot i + L \tag{2.3}$$
$$= a \cdot \left[1 - \left(1 + \frac{i}{b}\right) \cdot e^{\frac{i}{b} - c}\right], \tag{2.4}$$

where a, b, and c are the fitting parameters.

The physical *air interface* consists of a serial RLC circuit at the transmitter side which is coupled via a mutual inductance with a parallel RLC circuit at the receiver side. Important for this system is to know how the output $u_2(t)$ behaves when the input $u_1(t)$ changes. This leads to a system of two differential equations of the second order:

$$\begin{aligned} \ddot{u}_{C1} = &\frac{R_1 L_2}{M^2 - L_1 L_2} \cdot \dot{u}_{C1} + \frac{L_2}{C_1 (M^2 - L_1 L_2)} \cdot u_{C1} \\ &+ \frac{M}{C_1 (M^2 - L_1 L_2)} \cdot u_2 - \frac{L_2}{C_1 (M^2 - L_1 L_2)} \cdot u_1 \end{aligned} \tag{2.5}$$

$$\begin{aligned} \ddot{u}_2 = &\frac{M C_1 R_1}{C_2 (M^2 - L_1 L_2)} \cdot \dot{u}_{C1} + \frac{M}{C_2 (M^2 - L_1 L_2)} \cdot u_{C1} \\ &- \frac{1}{C_2}\left(\frac{1}{Z} + \frac{1}{R_2}\right) \cdot \dot{u}_2 + \frac{L_1}{C_2 (M^2 - L_1 L_2)} \cdot u_2 \\ &- \frac{M}{C_2 (M^2 - L_1 L_2)} \cdot u_1, \end{aligned} \tag{2.6}$$

which also can be represented as transfer function in Laplace domain as

$$H(s) = \frac{U_2(s)}{U_1(s)} = \frac{\alpha \cdot s^2}{\beta \cdot s^4 + \gamma \cdot s^3 + \delta \cdot s^2 + \epsilon \cdot s + 1} \tag{2.7}$$

with

$$\begin{aligned} \alpha &= M C_1 \\ \beta &= C_1 L_1 C_2 L_2 - M^2 C_1 C_2 \\ \gamma &= C_1 L_2 \left[R_1 C_2 + L_1 \left(\frac{1}{R_2} - \frac{1}{Z}\right)\right] + M^2 \left(\frac{1}{Z} - \frac{C_1}{R_2}\right) \\ \delta &= L_2 C_2 + C_1 R_1 L_2 \frac{1}{R_2} + C_1 L_1 - L_1 R_1 L2 \frac{1}{Z} \\ \epsilon &= L_2 \left(\frac{1}{R_2} + \frac{1}{Z}\right) + C_1 R_1. \end{aligned}$$

These equations represent the linear case. However, physical systems are rarely linear, but nonlinear. The nonlinearities which were discussed earlier (sec. 2.3.2) are the nonlinear behaviors of the coils L_1 and L_2 (e.g. saturation current) and the nonlinearity of the input impedance Z. The input impedance Z is modeled as current source $i_T(t)$. Including all these nonlinear behaviors we get the expanded system description

$$U_2(s) = \frac{A}{C} \cdot U_1(s) - \frac{B}{C} \cdot I_T(s) \tag{2.8}$$

with

$$
\begin{aligned}
A &= s^2 \cdot (C_1 M) \\
B &= s^3 \cdot C_1 \left(L_1 L_2 - M^2 \right) + s^2 \left(C_1 L_2 \left(R_1 + R_{L1} \right) + C_1 L_1 R_{L2} \right) + \\
&\quad s \left(L_2 + C_1 R_{L2} \left(R_1 + R_{L1} \right) \right) + R_{L2} \\
C &= \alpha \cdot s^4 + \beta \cdot s^3 + \gamma \cdot s^2 + \delta \cdot s + \epsilon \\
\alpha &= C_1 C_2 \left(L_1 L_2 - M^2 \right) \\
\beta &= C_1 \frac{L_1 L_2 - M^2}{R_2} + C_1 C_2 L_1 R_{L2} + C_1 C_2 L_2 \left(R_1 + R_{L1} \right) \\
\gamma &= \frac{C_1 L_1}{R_2} \left(R_1 + R_2 + R_{L1} + R_{L2} \right) + C_1 C_2 \left(R_1 R_{L2} + R_{L1} R_{L2} \right) + C_2 L_2 \\
\delta &= C_1 R_1 \left(\frac{R_{L2}}{R_2} + 1 \right) + C_1 R_{L1} \left(\frac{R_{L2}}{R_2} + 1 \right) + \frac{L_2}{R_2} + R_{L2} C_2 \\
\epsilon &= \frac{R_{L2}}{R_2} + 1.
\end{aligned}
$$

$L_1 \left(i_1 \right)$ and $L_2 \left(i_2 \right)$ are the nonlinear self inductances, which fulfill the inductance function equ. 2.3. $M = k \cdot \sqrt{L_1 \left(i_i \right) L_2 \left(i_2 \right)}$ is the mutual inductances. The nonlinear input impedance is represented as current source I_T (equ. B.1) with $V_{bs} = 0$, $V_{gs} = u_2$ and V_{ds} is the attenuation control voltage V_{ATTN}.

2.4. Analog Front End

2.4.1. Overview

The *analog front end* (*AFE*) performs the analog signal processing. It exists of a *limiter* structure with a *received signal strength indicator* (*RSSI*) generator which accomplishes the demodulation, a data filter with data comparator for hard detection of the chips, and an *active gain control* (*AGC*) to adjust the input signal strength, i.e., the level of the *air interface* voltage $u_2(t)$.

2.4.2. Limiter and Received Signal Strength Indicator

Functionality

The *limiter* is a sequence of eight amplifiers (fig. 2.9) with a gain of 2, a low pass bandwidth of 250kHz, and a saturation of ± 0.1V (the gain behavior of a single stage is shown in fig. 2.10).

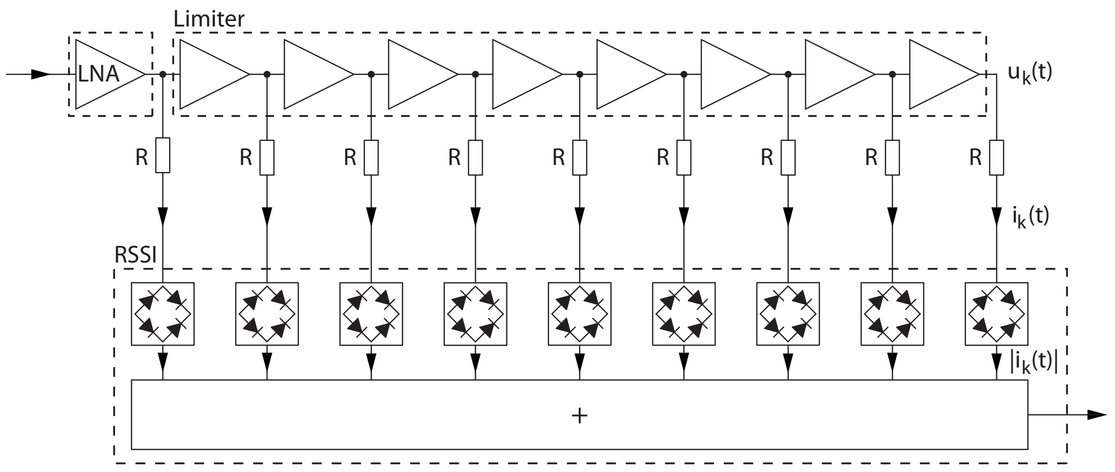


Figure 2.9.: .
Architecture of the *limiter* and the *RSSI* generator with a preamplification LNA.

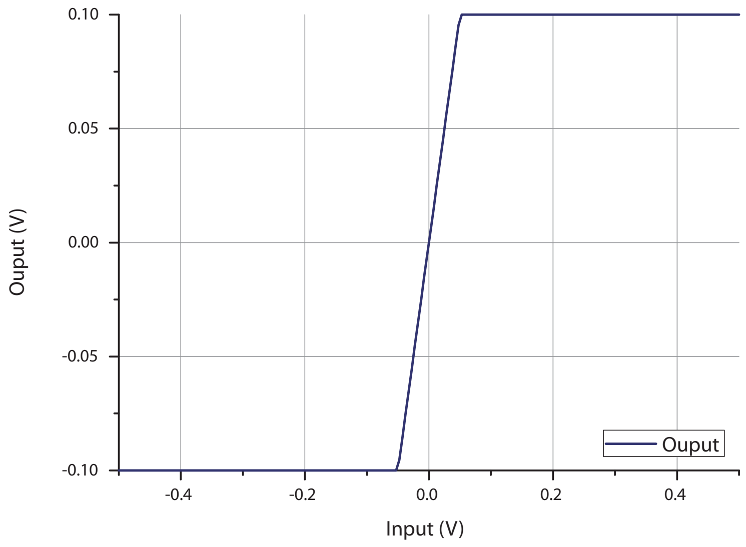
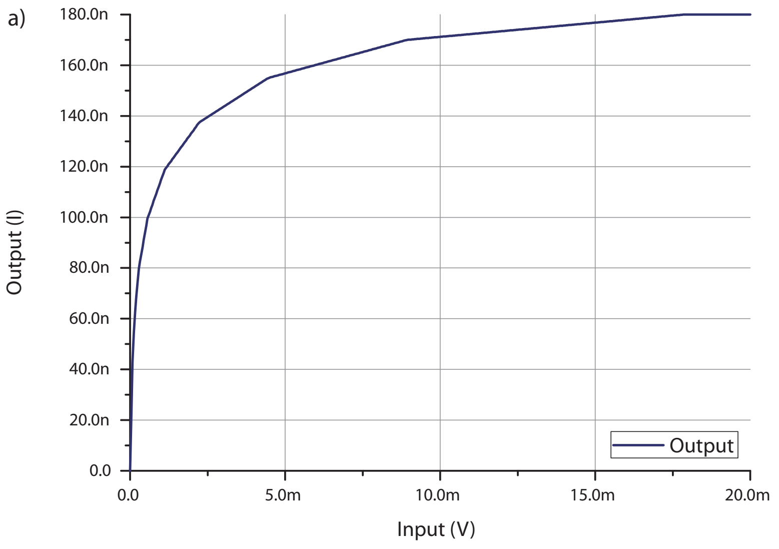


Figure 2.10.: Gain behavior of a *limiter* stage with an amplification of 2 and a saturation of ±0.1V.

A *low noise amplifier* (*LNA*) with the same specifications as a *limiter* stage, but a gain of 5.6, preamplifies the receiver input signal. The output voltages of the amplifiers are converted into currents (e.g. with a resistor $I = \frac{U}{R}$, with $R = 5M\Omega$) , rectified and added up. The generated signal is called *received signal strength indicator* (*RSSI*). The *RSSI* shows a logarithmic behavior (fig. 2.11) due to the cascading of amplifiers of the *limiter*.



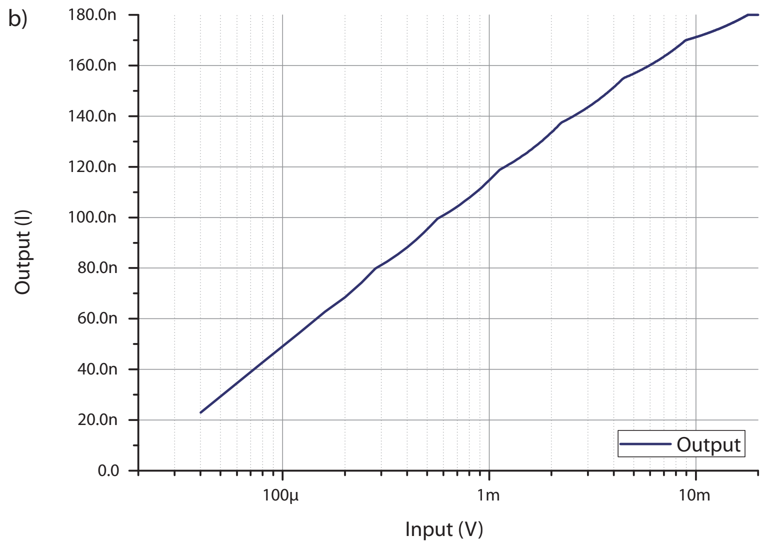


Figure 2.11.: Figure a) shows the RSSI DC characteristic of input voltages from 0V to 20mV; figure b) shows the same characteristic with logarithmic scale for the input signal.

To account for the noise in the system the signal from the *air interface* is impaired by *White Gaussian Noise* with a root mean square voltage of $V_{Noise,rms} = 200\mu V$ (fig. 2.12) in the model. The output of the *RSSI* current signal for an example, is shown in fig 2.13.

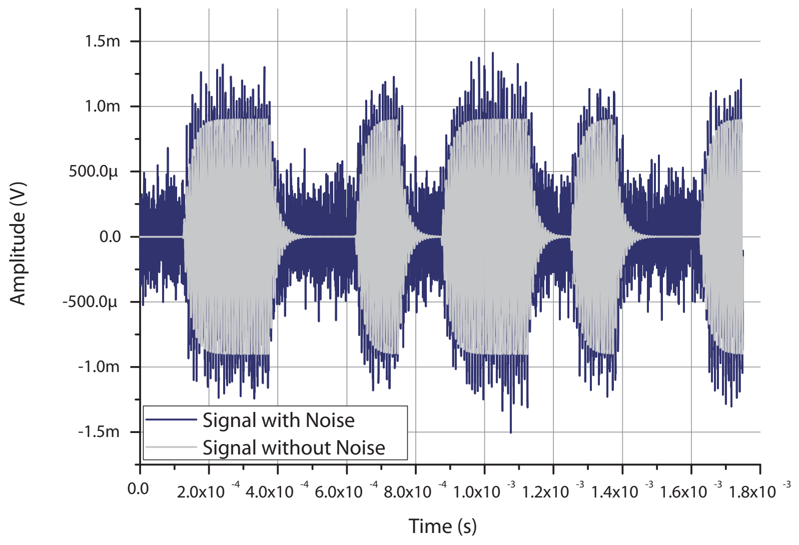


Figure 2.12.: Signal from the *air interface* added with *White Gaussian Noise* with a $V_{Noise,rms}$ of $200\mu V$.

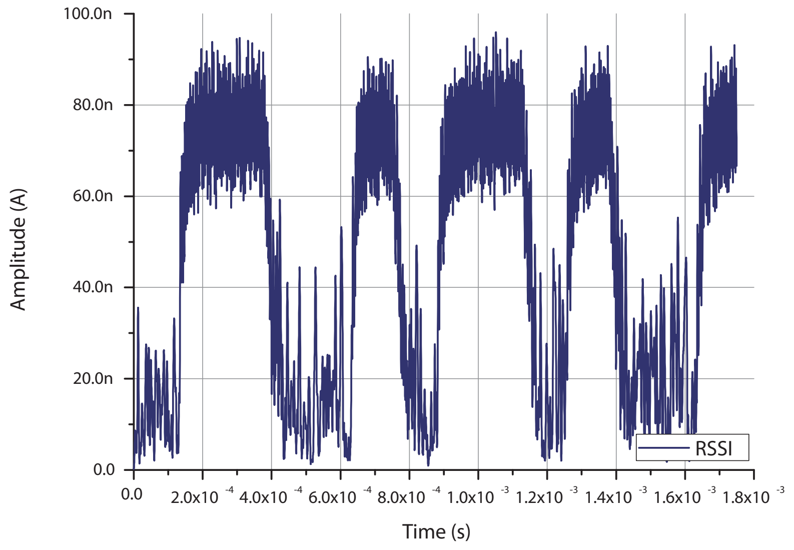


Figure 2.13.: *RSSI* of the example signal.

Modeling

The LNA and the eight stages of the *limiter* are modeled as first order lowpass filters with a transfer function of

$$H(s) = \frac{K}{1 + s\frac{1}{2\pi 250000}}, \tag{2.9}$$

where K is a gain factor, which differs between the *LNA* ($K = 5.6$) and the *limiter* stages ($K = 2$).

Because the output voltages of the eight *limiter* stages have a saturation behavior, they are limited to ±0.1mV and then fed into the next *limiter* stage. For the *RSSI* generation these saturated output voltages are then transformed into currents with the linear function, i.e.,

$$i_k(t) = 0.2 \cdot 10^{-6} \cdot u_k(t). \tag{2.10}$$

Then the absolute value is taken and the sum from all these currents is made. The resulting current is called the *RSSI* signal:

$$RSSI = \sum_{k=0}^{8} |i_k(t)|. \tag{2.11}$$

2.4.3. Data Filter and Data Comparator

Functionality

The *data filter* and *data comparator* (fig. 2.14) are used for hard detecting the signal amplitude, they quantize the amplitude. Because we only have a 1bit signal, the output of the quantization process is either *Zero* or *One*. The hard detection is done with a comparator, which compares the input signal to a threshold. Is the signal greater than the threshold the digital output *LFRAW* is *One*, and *Zero* otherwise.

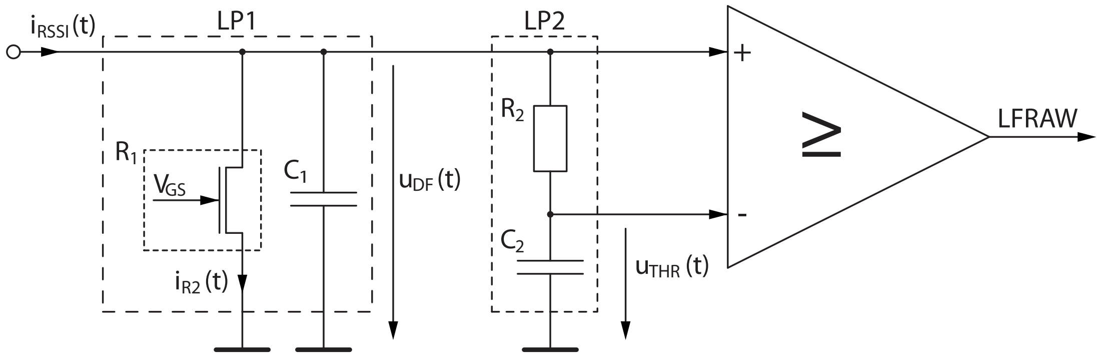


Figure 2.14.: A comparator compares the 11kHz lowpass filtered RSSI signal to a threshold. The threshold is the mean of the signal, which is generated with a low pass filter with a cut-off frequency of 500Hz.

To obtain the *binary* signal *LFRAW*, two lowpass filters $LP1$ and $LP2$ are used. The first lowpass filter has a cut-off frequency f_{c1} of 11kHz. It also converts the RSSI current $i_{RSSI}(t)$ back to a voltage signal. The resistor R_2 is nonlinear, due to the fact that it is a MOS transistor. The aim of this filter is to attenuate all signals with a higher frequency than the data signal so that these frequency cannot disturb the amplitude quantization process. The second lowpass filter with a cut-off frequency f_{c2} of 500Hz generates the mean within a certain time window of the input signal. This mean, denoted as $u_{THR}(t)$, is used as threshold for the comparator, which compares the lowpass filtered input signal $u_{DF}(t)$ to this threshold. Is the input voltage $u_{DF}(t)$ greater or equal to the threshold signal $u_{THR}(t)$ the output of the comparator is *One*, and *Zero* otherwise (fig. 2.15 and fig. 2.16).

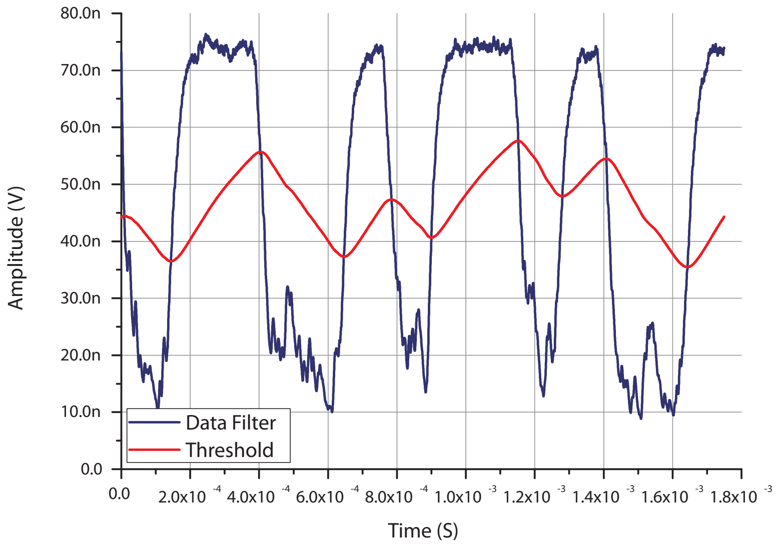


Figure 2.15.: Signal of the *data filter* (LP1) and the *threshold generator* (LP2). Where the *data filter* signal is greater or equal to the threshold, the output of the *data comparator* is an *One*, otherwise a *Zero*.

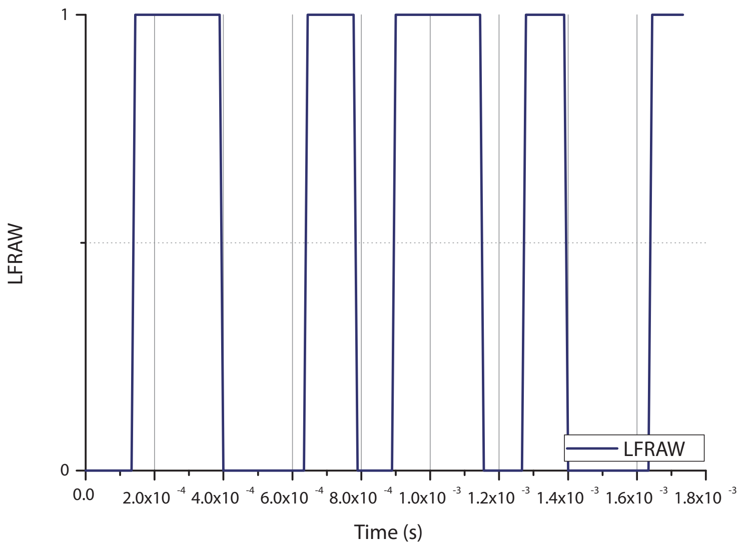


Figure 2.16.: Output of the *data comparator* (*LFRAW*).

Modeling

This system is modeled with transfer functions and a decision maker. The output voltages of the lowpass filters $LP1$ and $LP2$ are given by

$$U_1(s) = \frac{sR_2C_2 + 1}{s^2R_2C_1C_2 + s(C_1 + C_2)} \cdot (I_{RSSI}(s) - I_{R1}(s)) \tag{2.12}$$

$$U_2(s) = \frac{1}{s^2R_2C_1C_2 + s(C_1 + C_2)} \cdot (I_{RSSI}(s) - I_{R1}(s)) \,. \tag{2.13}$$

There are several constraints, which have to be considered: the product R_1C_1 sets the cut-off frequency of $LP1$, i.e., $R_1C_1 = \frac{1}{2\pi f_{C1}}$, the product R_2C_2 sets the cut-off frequency of $LP2$, i.e., $R_2C_2 = \frac{1}{2\pi f_{C2}}$, R_2 should be very high so that LP2 does not influence LP1 too much, and $I_{R1}(s)$ is modeled in the time-domain with the EKV MOS transistor model (app. B) with V_{GS} set for a specific resistance.

2.4.4. Active Gain Control

Functionality

The *active gain control* (fig. 2.17) consists of a lowpass filter LP with a cut-off frequency of 16.5kHz and an operational transconductance amplifier (OTA). The lowpass filter LP converts the current of the *RSSI* generator $i_{RSSI}(t)$ into a voltage $u_{LP}(t)$. The *OTA* compares the output voltage $u_{LP}(t)$ of the filter to a threshold voltage V_{AGC}. The output current of the OTA $i_{AGC}(t)$ is subtracted by the constant current I_0 of 1.25nA. This is done to be able to get a negative current too. Integration of this current with the capacitance C_2 at 24pF results in the control voltage V_{ATTN}, which is the gate-source voltage of the input impedance Z of the *air interface*.

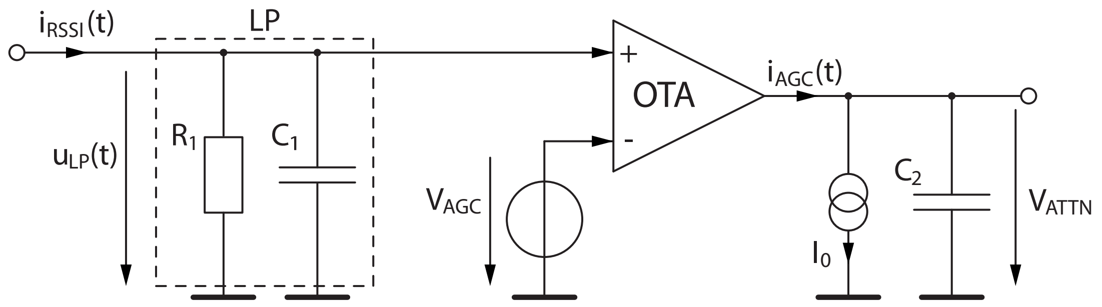
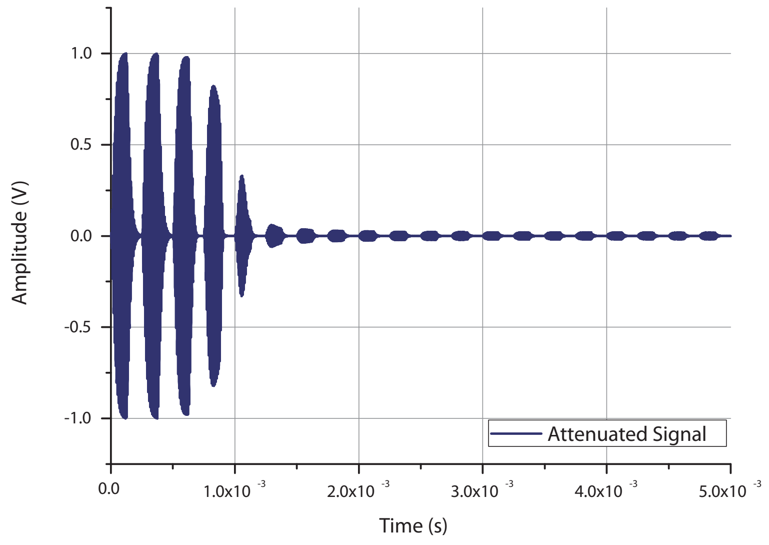


Figure 2.17.: Architecture of the *active gain control.* A comparison of the input signal strength to a threshold leads to the control voltage V_{ATTN} to set the resistance of the MOS transistor Z of the *air interface.*

The *active gain control* controls the amplitude of the input signal (fig. 2.18). If too high, the input voltage is attenuated to a required range of the amplitude. This is done do provide a constant signal strength.



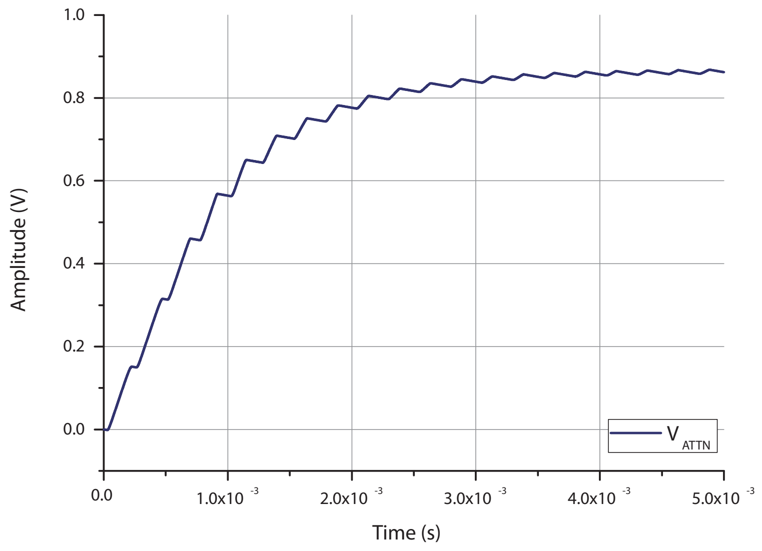


Figure 2.18.: The *active gain control* generates a control signal to set the resistance of the input impedance Z to attenuate the input signal. The figure above shows the damped Signal. The control voltage V_{ATTN} is shown in the second figure.

Modeling

Because the lowpass filter LP has a cut-off frequency f_c at 16.5kHz and it has to convert the input current $i_{RSSI}(t)$ into the voltage u_{LP}, which is going to be compared to the threshold voltage V_{AGC}, its transfer function is

$$U_{LP}(s) = \frac{R_1}{sR_1C_1 + 1} \cdot I_{RSSI}(s), \tag{2.14}$$

with $R_1C_1 = \frac{1}{2\pi 16500}$. The maximum value of the voltage $u_{LP}(t)$ depends therefore on the value of the resistance.

The OTA amplifies a voltage difference $u_{diff}(t) = u_{LP}(t) - V_{AGC}$ from 0 to 100mV linearly into a current $i_{AGC}(t)$ from 0 to 40nA. A voltage difference $u_{diff}(t)$ of more than 100mV is saturated to a current of 40nA (fig. 2.19).

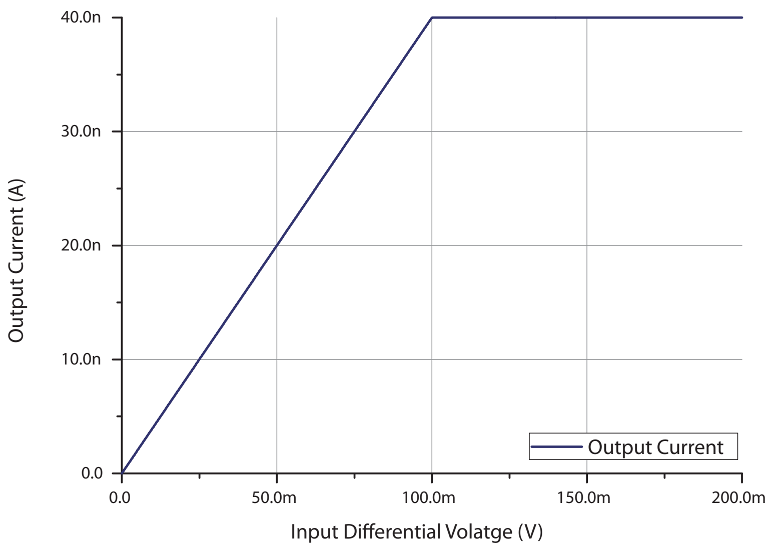


Figure 2.19.: Output characteristic of the OTA.

2.5. Digital Baseband Processor

2.5.1. Overview

The *digital baseband processor* (fig. 2.20) performs the digital signal processing. The first occurring flip-flop of the *DBP* does the sampling of the binary continuous-time *LFRAW* signal, which is produced by the *analog front end*. Because the *DBP* is clocked with a frequency of 90kHz, the processing is done with oversampling. The chip rate is the double of the data rate with 4kbaud, therefore one chip is about eleven samples long. The tasks of the *DBP* are to reconstruct the chips from the binary continuous-time *LFRAW* signal. To detect the start of a transmission, a so called *SyncPattern* is used. A *SyncPattern* is a unique sequence of chips. After the *SyncPattern* is found the Manchester decoder can start to decode the data from the incoming chip sequence. A finite state machine controls the different functional units.

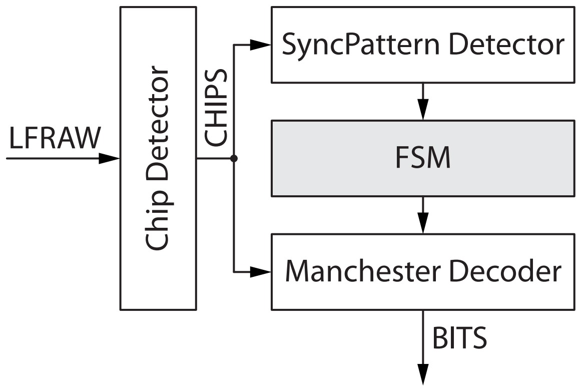


Figure 2.20.: Block architecture of the *digital baseband processor*.

2.5.2. Chip Detector

The detection of the individual chips is done in a rather simple way. A transition from an *One* to a *Zero* chip, or vice versa, is recognized by a simple *XOR* operation of two successive following samples (fig. 2.21). After the detection of this transition a counter starts to count down from its preset value, which is five when we assume the chip length is about eleven samples. When it reaches zero we assume to be in the middle of the chip and the current sample is taken for further processing. The counter is then set to eleven, the number of samples of a chip. The counter starts to count down again. If there is no transition found during this count, a new chip sample is taken when the counter reaches the zero again. If there is a transition found during the count, the counter is set back to the preset value five and the counting process starts again.

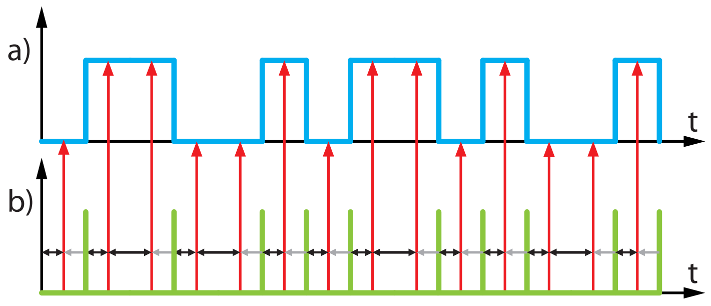


Figure 2.21.: This figure shows the procedure of chip detection: a) is the incoming data signal *LFRAW* from the *analog front end*, b) shows the detected transitions and the times when a chip sample is taken. The small black arrows are the counting events from a detected transition to the middle of the chip. The long black arrows are the longer counting events from the middle of the chip to the middle of the following chip and the short gray arrows are the long counting events, which are interrupted by a transition.

2.5.3. Finite State Machine and SyncPattern

The *Sync Pattern* indicates the start of transmission; it is a unique chip sequence (sec. 1.4.3 and fig. 2.22). The *finite state machine* (*FSM*), which is the control logic of the the *digital baseband processor*, has to detect the *SyncPattern* and to initiate the Manchester decoding afterwards.

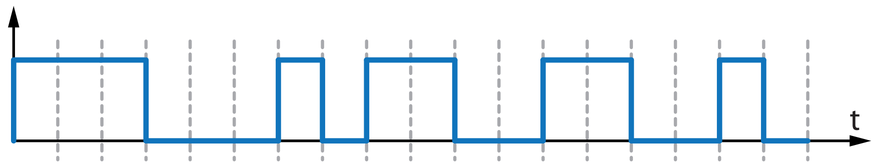


Figure 2.22.: *SyncPattern* to secure the receiver recognizes a data transmission.

2.5.4. Manchester Decoder

The *Manchester decoder* decodes the data message from the chip signal. For every bit two chips are used. Therefore a new bit starts every second chip. If the two chips which belong to a bit are a *Zero* followed by an *One*, the bit is decoded as an *One*. If the two chips are vice versa, the bit is decoded as a *Zero*. Other chip combinations are not allowed and the *Manchester decoder* recognizes an error at the transmission.

2.6. Simulation Configuration

In a first approach, the behavior of the digital receiver system was modeled in a co-simulation environment. The *air interface* and the *analog front end* were modeled in *The MathWorks Simulink*. *Simulink* has the benefit that it is possible to generate complex systems with simple sub blocks, e.g. transfer functions, in a graphical user interface (GUI). Because a common design method for digital circuits is to describe them in a hardware description language (HDL), the *digital baseband processor* was modeled in *MentorGraphics ModelSim* as a *VHDL* model. The co-simulation was done with the HDL co-simulation 'Simulink Link for ModelSim' block. This block establishes an HDL daemon server which connects to an instance of a VHDL design in *ModelSim*, which has been loaded for the co-simulation with *Simulink*. The bidirectional data transfer was done with shared memory.

In a next step, to speed up the simulation, a baseband setting of the *air interface* and the *analog front end* was used. Also the *active gain control* was removed. Therefore a baseband equivalent of the bandpass signal was used. This means for our case a 125kHz down shift in frequency for the *air interface* and the *limiter* stage filters. So the low pass bandwidth of the system was reduced from 250kHz to 125kHz, which speeds up the simulation time by a factor of 2. Still the simulations of *bit error rates* and *message success rates* took a lot of time due to slow signal handling of the co-simulation daemon.

For a final design step to speed up the simulation, the system is now modeled in *The MathWorks Matlab* (fig. 2.23). All analog filters of the *analog front end* are replaced by bilinear transformed baseband equivalents. The *digital baseband processor* is modeled in *Matlab* too. The evolution of the receiver model is shown in fig. 2.24. A reduction of details of the model leads to a faster model, but also to less accuracy.

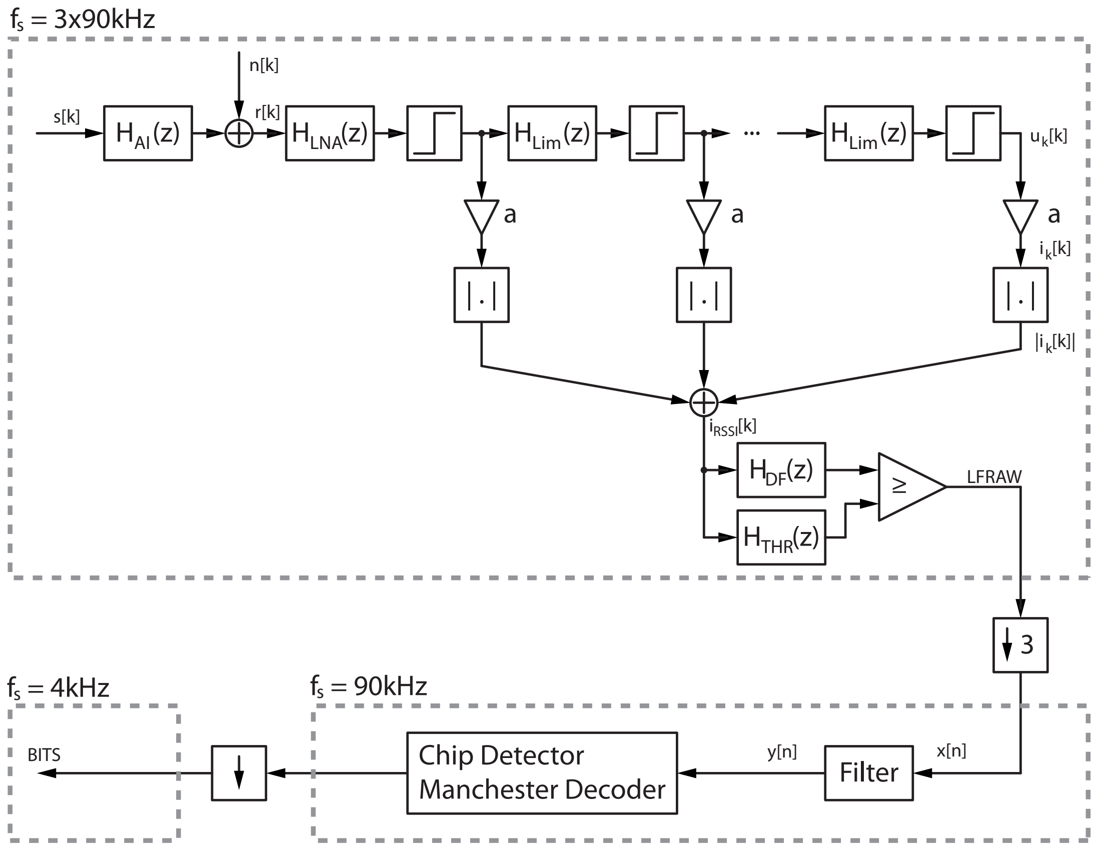


Figure 2.23.: Baseband equivalent *Matlab* model of the receiver system. $H_{AI}(z)$, $H_{LNA}(z)$, $H_{Lim}(z)$, $H_{DF}(z)$, and $H_{THR}(z)$ are lowpass filters. $H_{AI}(z)$ has a f_c of 8Khz. $H_{LNA}(z)$ and $H_{Lim}(z)$ have a f_c of 125kHz. $H_{DF}(z)$ has an f_c of 11kHz and $H_{THR}(z)$ has an f_c of 500Hz. a is a gain factor, with $a = 0.2 \cdot 10^{-6}$. The model is split into several sampling regions. The entire analog part of the receiver is sampled with a f_s of $3 \cdot 90kHz$ due to the Shannon sampling theorem [10]. The digital part is clocked with 90kHz, therefore the sampling frequency is 90kHz.

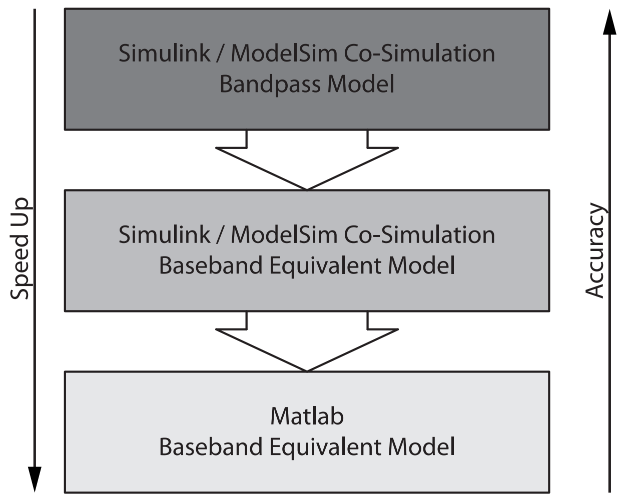

Simulink / ModelSim Co-Simulation
Bandpass Model
Simulink / ModelSim Co-Simulation
Baseband Equivalent Model
Matlab
Baseband Equivalent Model
Speed Up
Accuracy

Figure 2.24.: Evolution of the development of the simulation model.

2.7. System Performance Evaluation

2.7.1. Overview

To characterize the system performance several methods are used. These methods are the estimation of the *bit error rate* and the estimation of the *message success rate* for various signal strengths.

The signal strength is indicated as the bit energy (E_b) over the noise power density (N_0). The bit energy for a Manchester coded signal on 1Ω is

$$E_b = \frac{A^2}{2D}, \tag{2.15}$$

where A is the amplitude of the bit symbol and D as the data rate of the signal. The noise of the system is a *White Gaussian Noise* with a root mean square voltage of $V_{Noise,rms} = 200\mu V$ (sec. 2.4.2). The noise power density on 1Ω is therefore

$$N_0 = 2\frac{\sigma_0^2}{B}, \tag{2.16}$$

where σ_0 is the standard deviation of the noise, which is equivalent to $V_{Noise,rms}$, and B is the bandwidth of 250kHz [11]. Therefore $\frac{E_b}{N_0}$ is given by

$$\frac{E_b}{N_0} = \frac{A^2 B}{4D\sigma_0^2}. \tag{2.17}$$

In the investigated receiver architecture the noise level is roughly constant, whereas the signal amplidude A is variable. The amplitude in dependency of $\frac{E_b}{N_0}$ is then

$$A = \sqrt{\frac{4D\sigma_0^2}{B}\frac{E_b}{N_0}}. \tag{2.18}$$

2.7.2. Bit Error Rate

The *bit error rate* (*BER*) is the probability that a bit error occurs at a specific $\frac{E_b}{N_0}$ [1]. The receiver implementation is not able to resolve a bit error. So that the entire transmission is discarded in case of a bit error. To evaluate the *bit error rate*, we restart the simulation whenever an error occurs. This is done n times so that at the end n errors are observed. The quantities N_i of all bits until the ith error occurred are summed up to $N = \sum_1^n N_i$, with $i = 1...n$. The *BER* for a single $\frac{E_b}{N_0}$ value is given by

$$BER = \frac{n}{N}, \tag{2.19}$$

where we assume ergodic behavior. In the simulation setup, we have to take care that all filters are properly tuned, when we restart the simulation.

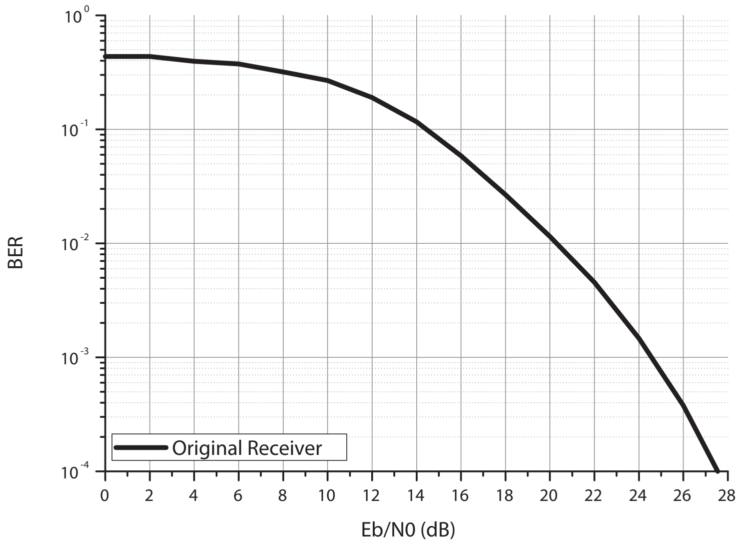


Figure 2.25.: *BER* over $\frac{E_b}{N_0}$ for the original receiver system.

The *BER* performance of the receiver is shown in fig. 2.25. A bit *bit error rate* of 1% requires an $\frac{E_b}{N_0}$ of about 20dB.

2.7.3. Message Success Rate

The *message success rate* (*MSR*) is the probability of correct received messages depending on a specific $\frac{E_b}{N_0}$. When for a specific $\frac{E_b}{N_0}$ M messages were sent and m were received correctly, the *message success rate* is given as

$$MSR = \frac{m}{M}. \tag{2.20}$$

A data message has eight bits. The *MSR* behavior over $\frac{E_b}{N_0}$ is shown in fig. 2.26.

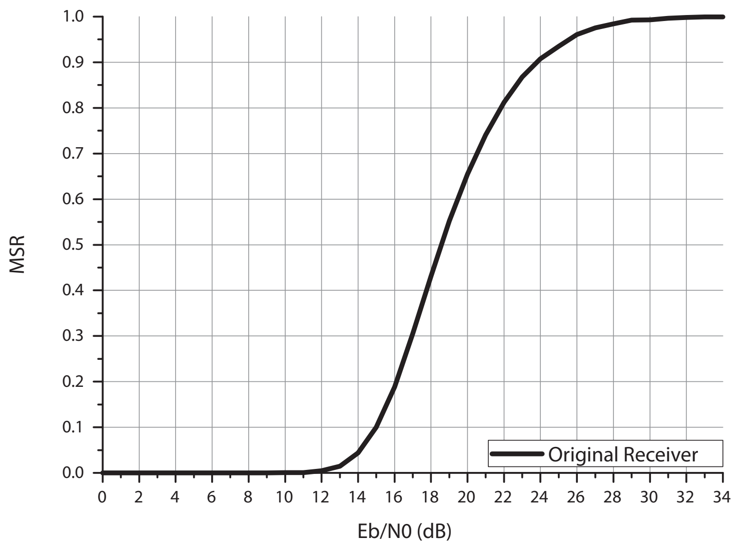


Figure 2.26.: *MSR* over $\frac{E_b}{N_0}$ for the original receiver system.

It can be seen that at an $\frac{E_b}{N_0}$ of 15dB 10% of the sent messages are received correctly. For 90% correctly received messages the $\frac{E_b}{N_0}$ has to be 24dB.

3. Enhanced Detection Schemes

3.1. Overview

After the hard detection of the *analog front end* there is still remaining noise in the data stream. Due to the oversampling of the chips it is possible to remove some noise in order to enhance the capability of the receiver to detect the correct data. To get rid of the noise we have to know how it affects the data stream. Since we are now in the discrete domain with a sample rate of 90kHz, one chip consists of approximately eleven samples. Due to noise several samples are inverted and therefore wrong. The aim is to design 1bit filters which are able to reconstruct the originally sent chips or at least reduce the number of impaired samples. *VHDL* models of all described filters and decoder can be found in app. D.

In the following we will use the notation $x[n]$ for the actual input samples and $y[n]$ for the actual output samples. The input $x[n]$ is the sampled binary continuous-time output signal *LFRAW* of the *analog front end*. The sample rate f_s is 90kHz. Therefore $x[n]$ denotes to $x(nT_s)$, where $x(t)$ is the signal *LFRAW* and T_s is $\frac{1}{f_s}$.

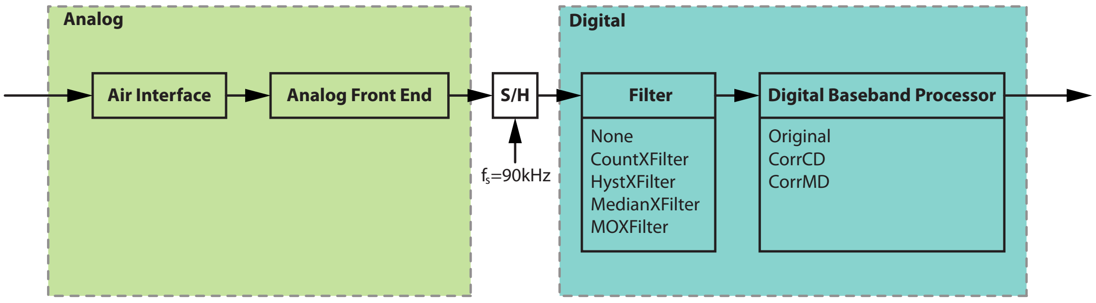


Figure 3.1.: Architecture of the enhanced receiver.

3.2. Counting Filter

3.2.1. Theory

The *CountXFilter* is the simplest 1bit filter which is going to be presented in this thesis. The main idea is to use a free running digital counter which increases if the input sample $x[n]$ is *One* and decreases if the input sample $x[n]$ is *Zero* (equ. 3.1). To be certain that there is no underflow or overflow in sense of a modulo effect of the counter, it is limited. So the counter can only have values between zero and a maximum value. This maximum value, the *maxcount*, is a degree of freedom and chosen to get the best performance. To decide whether the output sample $y[n]$ is *One* or *Zero*, the actual value of the counter $c[n]$ is compared to a threshold (equ. 3.2). This threshold is the half of *maxcount*, because we assume a digital *One* has equal probability as a digital *Zero*. If the actual value $c[n]$ of the counter is higher than the threshold, the output sample $y[n]$ is *One*, otherwise it is *Zero*. An example of the noise filtering of a *Count5Filter* with a *maxcount* of five is shown in fig. 3.2.

$$c[n+1] = \begin{cases} c[n]+1 & \text{for } x[n]=1 \wedge c[n] < maxcount, \\ c[n]-1 & \text{for } x[n]=0 \wedge c[n] > 0, \\ c[n] & \text{otherwise.} \end{cases} \tag{3.1}$$

$$y[n] = \begin{cases} 1 & \text{for } c[n] \geq \frac{maxcount}{2}, \\ 0 & \text{for } c[n] < \frac{maxcount}{2}. \end{cases} \tag{3.2}$$

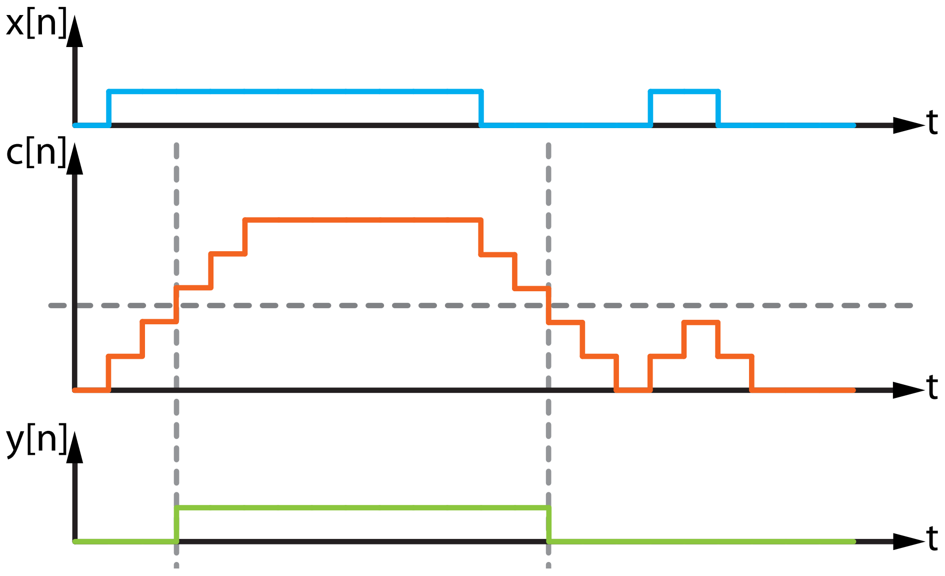


Figure 3.2.: Behavior of the *Count5Filter*. $x[n]$ is the input sequence, $c[n]$ is the counter value, and $y[n]$ is the output sequence.

All noise peaks which occur when the value of the counter $c[n]$ is near its limits are filtered out. Disturbed samples, however, which occur when the value of the counter $c[n]$ is near the threshold may not be eliminated. The output has a delay of $\lceil \frac{maxvalue}{2} \rceil$ samples.

3.2.2. Design

As mentioned previously, the *CountXFilter* consists of a digital up/down counter with a range between 0 and *maxcount*. It is clocked with the system clock of 90kHz and controlled via the input sample $x[n]$. The actual value of the counter is then evaluated with a comparator. This voter compares the counter value to the predefined threshold $\frac{maxcount}{2}$. The architecture of the *CountXFilter* is shown in fig 3.3.

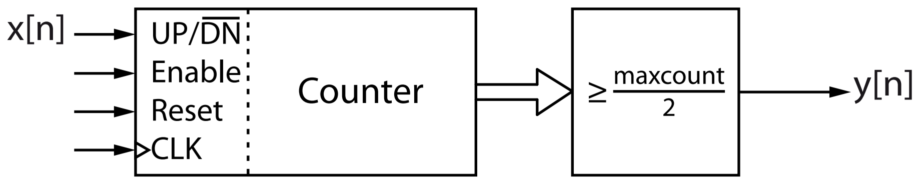


Figure 3.3.: Architecture of the CountXFilter.

3.2.3. Evaluation

The *bit error rate* estimations for several *CountXFilter* with *maxcount* values of three, five, seven, nine, and eleven are shown in fig. 3.4. The less $\frac{E_b}{N_0}$ a filter needs to achieve a certain *BER* value, the better it is. The best performance is achieved with a *Count5Filter*, which has a *maxcount* value of five. This filter needs about 3dB less $\frac{E_b}{N_0}$ than the original receiver for the same *BER* performance. With an $\frac{E_b}{N_0}$ of 17dB the *BER* is less than 1%.

The *message success rates* for several *CountXFilter* with *maxcount* values of three, five, seven, nine, and eleven are shown in fig. 3.5. Again the less $\frac{E_b}{N_0}$ a filter needs for a certain *MSR* value, the better it is. The *Count5Filter* shows again the best performance. It needs about 4dB less $\frac{E_b}{N_0}$ than the original receiver. It can be seen that at an $\frac{E_b}{N_0}$ of 11dB, 10% of the sent messages are received correctly. For 90% correctly received messages the $\frac{E_b}{N_0}$ has to be above 20dB.

The *Count9Filter* and the *Count11Filter* show worse performances than the original filter due to too long filters.

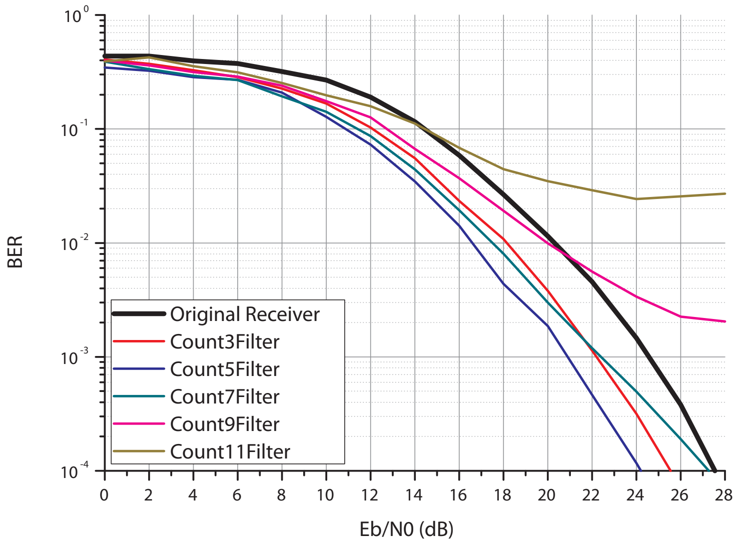


Figure 3.4.: This figure shows the *BER* of the *CountXFilter* with *maxcount* values from three to eleven.

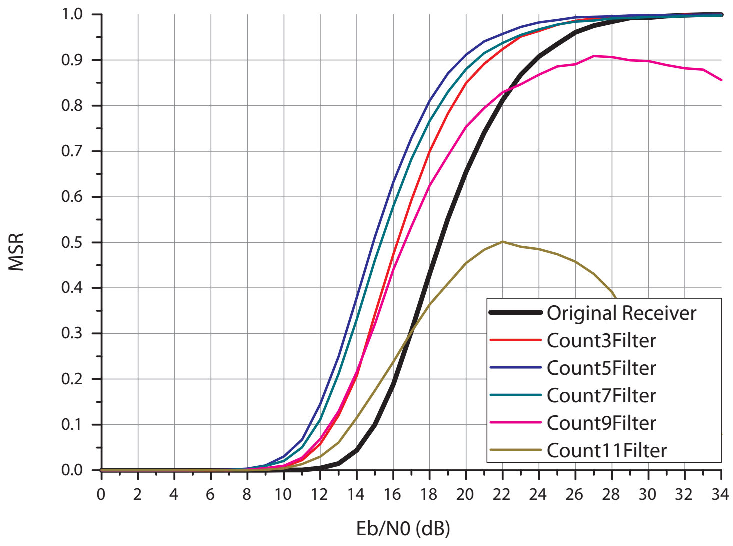


Figure 3.5.: This figure shows the *MSR* of the *CountXFilter* with *maxcount* values from three to eleven.

3.3. Counting Hysteresis Filter

3.3.1. Theory

The *HystXFilter* is an improvement of the *CountXFilter* (sec. 3.2). This filter is designed to eliminate the disadvantage of the *CountXFilter*, which appears when the counter value is close to the threshold of the comparator.

The principle of the free running digital counter is almost the same (equ. 3.3). The difference is that the decision is done with the help of a hysteresis. To do this, the voter separates the counter values into three equally sized areas. Each area has *statesize* counter values. A transition from an output *Zero* to an output *One* is done, when the counter value $c[n]$ is greater or equal to twice the *statesize*. A transition from an output *One* to an output *Zero* is done, when the counter value $c[n]$ is less or equal $\textit{statesize} - 1$. This is shown in equ. 3.4. So the voter is a state machine with the two states *high* and *low* (fig. 3.6).

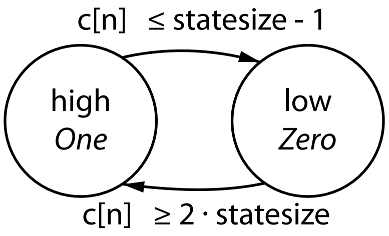


Figure 3.6.: States of the *HystXFilter* State Machine.

So the actual output sample $y[n]$ depends only on the actual state $s[n]$. The output sample $y[n]$ is *Zero*, when the state $s[n]$ is at the *low* state and is *One*, when the state $s[n]$ is at the *high* state. An example of the noise filtering of a *Hyst2Filter* with a *statesize* of two is shown in fig. 3.7.

$$c[n+1] = \begin{cases} c[n]+1 & \text{for } x[n] = 1 \wedge c[n] < 3 \cdot \textit{statesize} - 1, \\ c[n]-1 & \text{for } x[n] = 0 \wedge c[n] > 0, \\ c[n] & \text{otherwise.} \end{cases} \tag{3.3}$$

$$s[n+1] = \begin{cases} \textit{high} & \text{for } c[n] \geq 2 \cdot \textit{statesize}, \\ \textit{low} & \text{for } c[n] \leq \textit{statesize} - 1, \\ s[n] & \text{otherwise.} \end{cases} \tag{3.4}$$

$$y[n] = \begin{cases} 1 & \text{for } s[n] = \textit{high}, \\ 0 & \text{for } s[n] = \textit{low}. \end{cases} \tag{3.5}$$

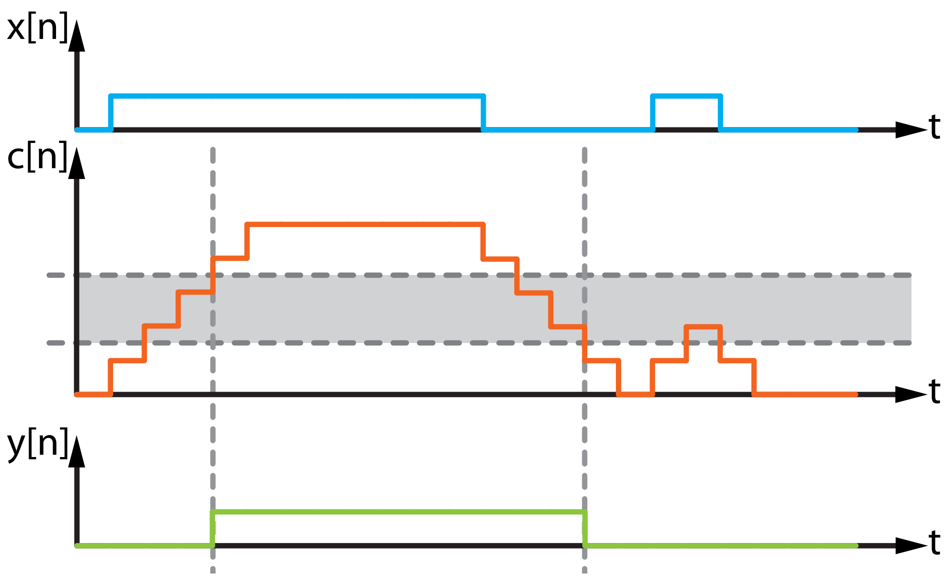


Figure 3.7.: Behavior of the *Hyst2Filter*. $x[n]$ is the input sequence, $c[n]$ is the counter value, and $y[n]$ is the output sequence.

3.3.2. Design

The *HystXFilter* consists of a digital up/down counter with a value range from zero to $3 \cdot \textit{statesize} - 1$. The counter is controlled via the input sample $x[n]$. The evaluation is done within a voter. This voter is a *finite state machine* (*FSM*) with the two states *high* and *low*. Since the output depends on the states only, it is a Moore machine [12]. The transition from *high* to low is done when the counter value $c[n]$ is less or equal $\textit{statesize} - 1$, else the state stays the same. The transition from *low* to *high* is done when the counter value $c[n]$ is greater or equal to $2 \cdot \textit{statesize}$. The actual state defines the output. If the state is *high* the output is *One*. If the state is *low* the output is *Zero*. An architecture of a *HystXfilter* is shown in fig 3.8.

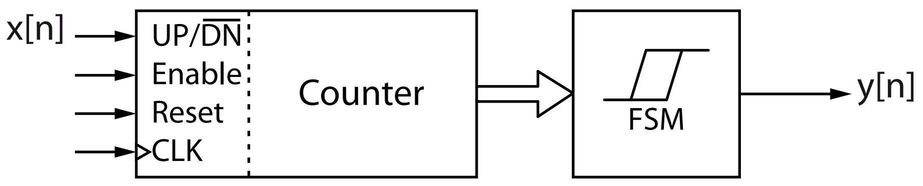


Figure 3.8.: Architecture of the *HystXFilter*.

3.3.3. Evaluation

The *bit error rate* estimations for several *HystXFilters* with *statesize* values of one, two, three, four, and five are shown in fig. 3.9. The *Hyst2Filter*, which has a *statesize* value of two, shows the best performance. This filter needs about 4dB less $\frac{E_b}{N_0}$ than the original receiver for the same *BER* performance. With an $\frac{E_b}{N_0}$ of 16dB the *BER* is less than 1%.

The *message success rates* for several *HystXFilter* with *statesize* values of one, two, three, four, and five are shown in fig. 3.10. The *Hyst2Filter* shows again the best performance. It needs about 4dB to 5dB less $\frac{E_b}{N_0}$ for the same *MSR* performance than the original receiver. It can be seen that at an $\frac{E_b}{N_0}$ of 11dB, 10% of the sent messages are received correctly. For 90% correctly received messages the $\frac{E_b}{N_0}$ has to be above than 19dB.

The *Hyst4Filter* and the *Hyst5Filter* have worse performances then the original receiver due to too long filters.

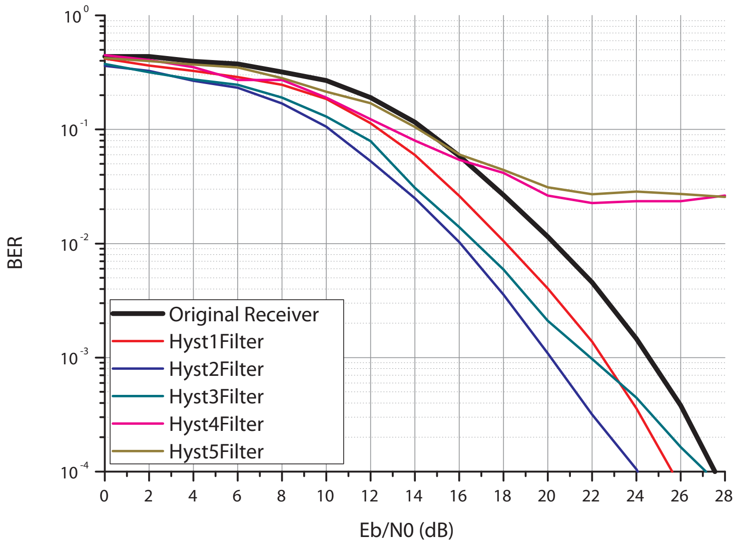


Figure 3.9.: This figure shown the *BER* of *HystXFilter* with *statesize* values from one to five.

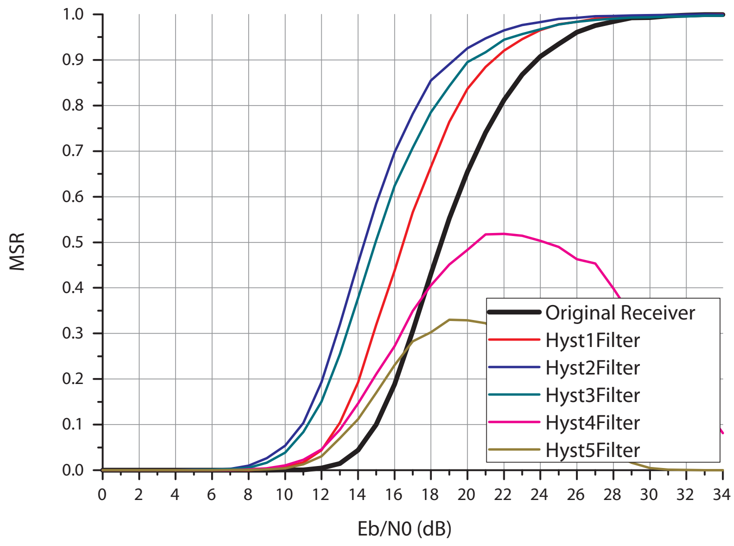


Figure 3.10.: This figure shown the *MSR* of *HystXFilter* with *statesize* values from one to five.

3.4. Median Filter

3.4.1. Theory

The *MedianXFilter* is an ordinary median filter. To get the median of a set of *filtersize* elements we need to sort the set from the smallest to the largest value, numerate the set from 0 to $\mathit{filtersize} - 1$, and pick the element in the middle [13]. *filtersize* has to be odd to be sure that there exists a single element in the middle, the $(\frac{\mathit{filtersize}-1}{2} + 1)$th element. In other words: $\frac{\mathit{filtersize}-1}{2} + 1$ has to be an integer.

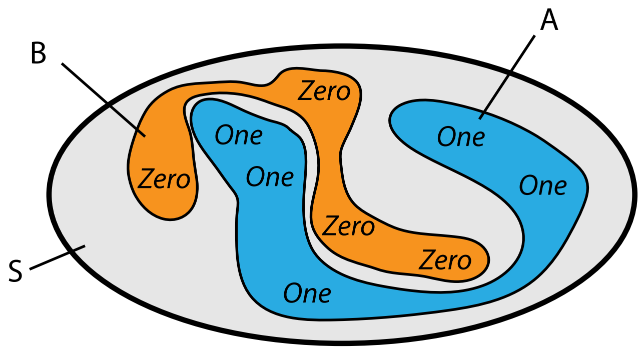


Figure 3.11.: Set Space of the Median Function.

Because we use a 1bit signal we just have to deal with the symbols *One* and *Zero*. We separate the set of all elements into two subsets with elements of the same symbol (fig. 3.11). So the median is always part of the subset with the most elements. If our set exists of *filtersize* elements, k times an *One* and $\mathit{filtersize} - k$ times a *Zero*. The output of the median filter is *One* when $k > \mathit{filtersize} - k$ and *Zero* when $k < \mathit{filtersize} - k$. An example of the noise filtering of a *Median5Filter* with a *filtersize* of five is shown in fig. 3.12.

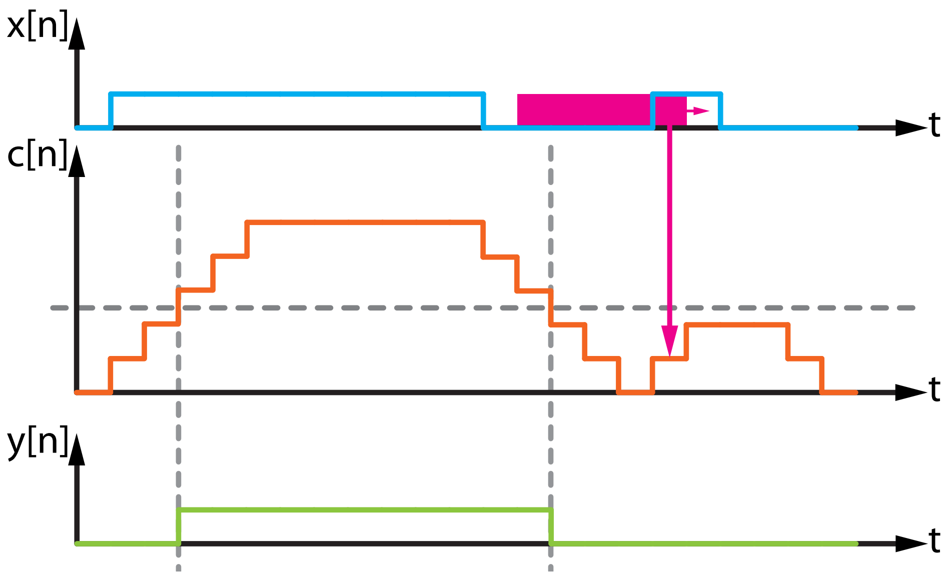


Figure 3.12.: Behavior of the *Median5Filter*. $x[n]$ is the input sequence, $c[n]$ is the counter value, and $y[n]$ is the output sequence

3.4.2. Design

The design of the *MedianXFilter* consists of a digital up/down counter, gates to control the counter, a delay line, and a voter (fig. 3.13). The delay line has a length of *filtersize*+1. The up/down counter is controlled via logical interconnections of the first and the last element of the delay line. The counter increases if the first delay element is *One*. The counter decreases in case the last element of the delay line is *One*. If the last and the first element of the delay line are *One*, then the counter value remains the same. This is also in case for two *Zeros*. Consequently the value of the counter represents the number of *Ones* within the filter with the filter length of *filtersize*. A comparator decides depending on the counter value if the output is *One* or *Zero*. If the counter value $c[n]$ is lower than $\frac{filtersize}{2}$, then the output is *Zero*, and *One* otherwise.

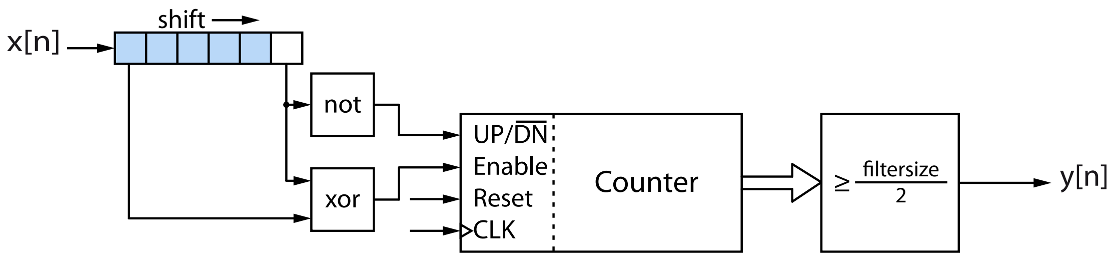


Figure 3.13.: Architecture of the *MedianXFilter*.

3.4.3. Evaluation

The *bit error rate* estimations for several *MedianXFilters* with *filtersize* values of three, five, seven, nine, and eleven are shown in fig. 3.14. The *Median7Filter*, which has a *filtersize* value of seven, shows the best performance. This filter needs about 4dB less $\frac{E_b}{N_0}$ than the original receiver for the same *BER* performance. With an $\frac{E_b}{N_0}$ of 16dB the *BER* is less than 1%.

The *message success rates* for several *MedianXFilters* with *filtersize* values of three, five, seven, nine, and eleven are shown in fig. 3.15. The *Median7Filter* shows again the best performance. It needs about 4dB to 5dB less $\frac{E_b}{N_0}$ for the same *MSR* performance than the original receiver. It can be seen that at an $\frac{E_b}{N_0}$ of 11dB, 10% of the sent messages are received correctly. For 90% correctly received messages the $\frac{E_b}{N_0}$ has to be above 19dB.

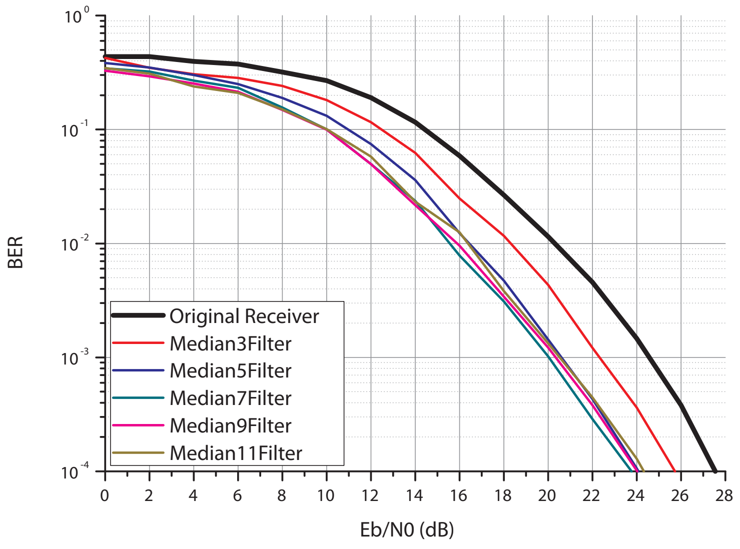


Figure 3.14.: This figure shows the *BER* of several *MedianXFilter* with *filtersize* values from three to eleven.

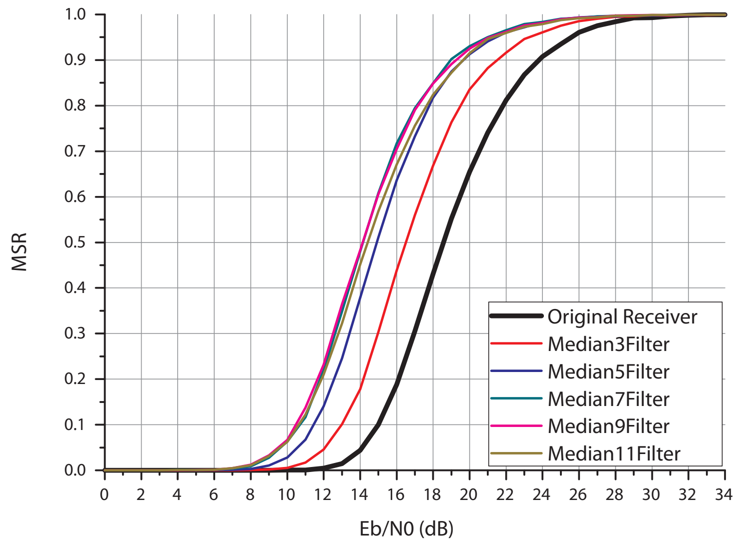


Figure 3.15.: This figure shows the *MSR* of several *MedianXFilter* with *filtersize* values from three to eleven.

3.5. Mathematical Morphology Filter

3.5.1. Theory

The *MOXFilter* is based upon the theory of *mathematical morphology* [14]. *mathematical morphology* was developed by Jean Serra and Georges Matheron at École nationale supérieure des mines de Paris (Paris School of Mines) at Fontainebleau in the year 1964. The primary application was to estimate the profitability of mines with the use of image analysis of rock sections. J. Serra and G. Matheron [14] were aware that there exist no uniform objective scientific tool to evaluate the shapes of inclusions and grains. The same problem had other sciences. An other example is cell biology, whose scientists want to evaluate the characteristic shapes of various cells. The *mathematical morphology* provides a set of tools to handle these problems. Evolved from just black/white images the *mathematical morphology* is nowadays able to handle gray scale images and higher dimensions, e.g. to analyze lattices.

The *mathematical morphology* is defined in Euclidean space $\mathbb{R}^n$ and in digital space $\mathbb{Z}^n$. A morphologic operation is always done by transformations. If we define E as a Euclidean Space $\mathbb{R}^n$ and $A \in E$ as subset of E then $B \in E$ is a so called *structuring element* with a defined center.

The fundamental transformations are the *erosion* and the *dilation*. *Erosion* [14] is defined as

$$C = A \ominus B = \{z \in E | B_z \subseteq A\} \tag{3.6}$$

with

$$B_z = \{b + z | b \subseteq B\}, \tag{3.7}$$

or in short

$$C = A \ominus B = \bigcap_{b \in B} A_{-b}. \tag{3.8}$$

If we have a two dimensional binary image in mind, then this transformation has as output the subset C of E of all moved center points of B, where the intersection of B with A is B. In short: The output is the set of all center points of a moved B, where B is congruent with A (fig 3.16b and fig. 3.17c).

Dilation [14] is defined as

$$C = A \oplus B = \{z \in E | \left(B^s\right)_z \cap A \neq \emptyset\} \tag{3.9}$$

with

$$B^s = \{x \in E | -x \in B\}, \tag{3.10}$$

or in short

$$C = A \oplus B = \bigcup_{b \in B} A_b. \tag{3.11}$$

If we have again a two dimensional binary image in mind, this transformation has as output the subset C of E of all unions of all moved B with A, where the center points of B are element of A (fig 3.16c and fig. 3.17b). Again in short: All points of A are expanded by the shape of B.

Based on these fundamental transformations exist the *opening* and the *closing* (fig. 3.17) transformations. *Opening* is defined as

$$C = A \circ B = (A \ominus B) \oplus B, \tag{3.12}$$

which is the application of an *erosion*, followed by an application of a *dilation*. *Closing* is defined as

$$C = A \bullet B = (A \oplus B) \ominus B, \tag{3.13}$$

which is the application of a *dilation*, followed by an application of an *erosion*. [14]

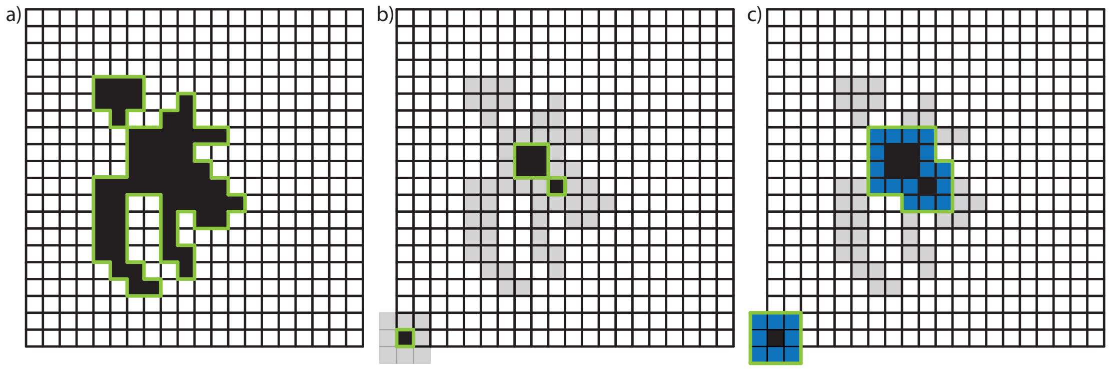


Figure 3.16.: *Opening* exists of an *erosion* transformation (b) followed by a *dilation* transformation (c). The *structuring element* for the *erosion* and the *dilation* is shown in their left lower corner.

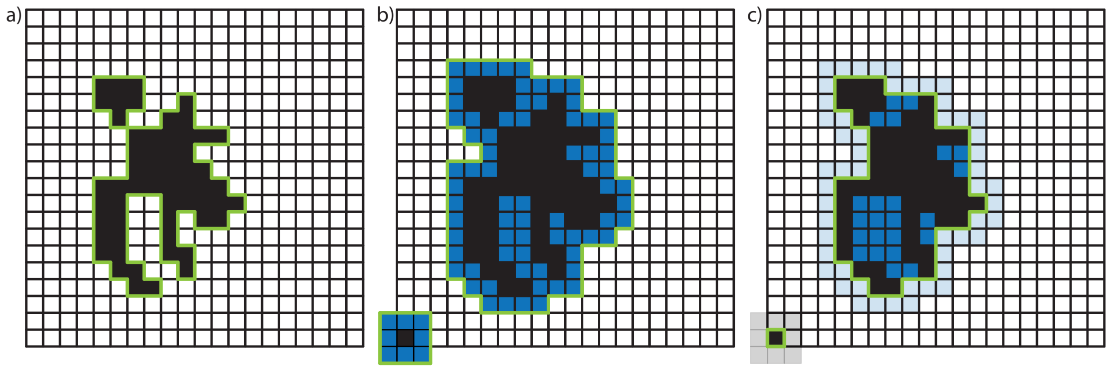


Figure 3.17.: *Closing* exists of a *dilation* transformation (b) followed by an *erosion* transformation (c). The *structuring element* for the *dilation* and the *erosion* is shown in their left lower corner.

In fig. 3.16 the *opening* transformation is shown. It can be seen, that all *Ones* (the black areas) vanish, if their cohesive areas can't be filled with the *structural element*. For noise removal, the *structural element* defines the limit for the filter to remove noise. Noisy fragments which are equal or greater than the *structural element* cannot be removed. In fig. 3.17 the *closing* transformation is shown. This transformation 'fills' the holes of a cohesive area. The *structural element* is again a limit. Here, a noisy area cannot be filled when the empty space is equal or greater than the *structural element*. A successive execution of an *opening* and a *closing* is able to remove noisy (filled, or unfilled) areas.

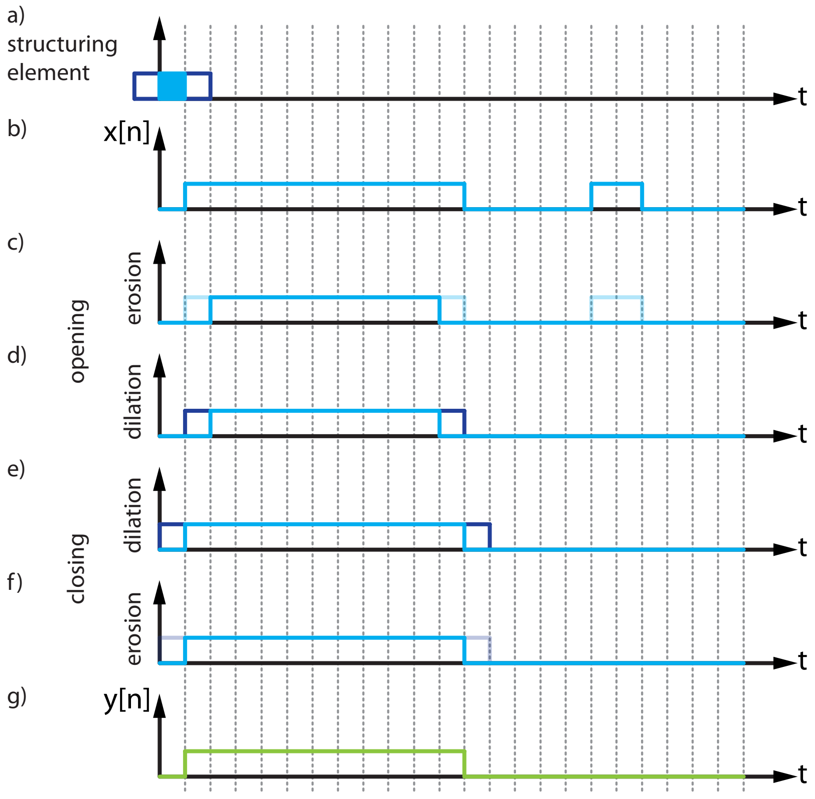


Figure 3.18.: For 1bit noise filtering one dimensional morphologic transformations can be used. The *structuring element* for *erosion* and *dilation* is shown in a). The *opening* of the input sequence b) is performed with an *erosion* c) and a *dilation* d). Thereafter *closing* is performed with a *dilation* e) and an *erosion* f). The output sequence is shown in g).

In the use as 1bit binary noise filter, we just use *opening* and *closing* in an one dimensional digital space $\mathbb{Z}$. The *structural element* B is therefore an one-dimensional odd array of *Ones*. An example of the noise filtering of a *MO3Filter* with an array length of the *structural element* of three is shown in fig. 3.18.

3.5.2. Design

We know that the *dilation* transformation can be represented as an operation, which expands the area for every *One* by the size of the *structuring element.* With center symmetric *structuring elements*, an *erosion* transformation can also be done by a *dilation* of the inverted digital space, followed by an inversion of the result. That means that an *erosion* can also be represented by expanding the area of every *Zero* by the size of the *structuring element.*

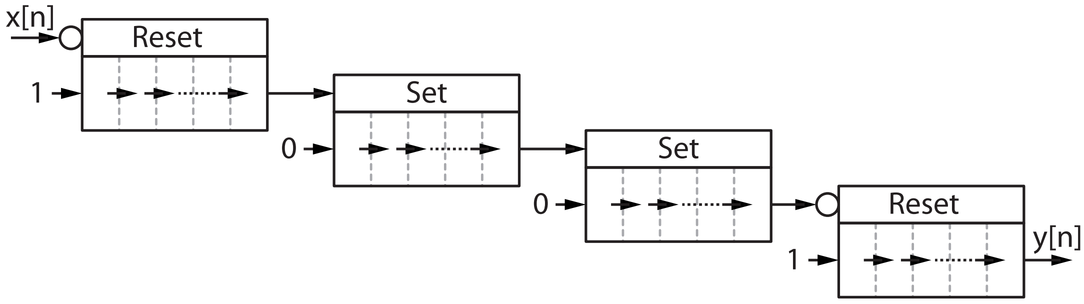


Figure 3.19.: Architecture of a *MOXFilter* built with shift registers. The first two shift registers perform the *opening* transformation. The last two shift registers perform the *closing* transformation.

An one-dimensional digital filter which uses *mathematic morphology* can be implemented as a sequence of four shift registers (fig. 3.19). The one dimensional *dilation* stages can be represented as a shift register which shifts all its content at every clock cycle. A *Zero* always refills the empty side of the register after a shift is performed. The input of such a *dilation* stage sets the whole shift register to *Ones*, when the input is *One.* This setting operation is intrinsically a *dilation.* The output of the *dilation* stage is the output of the shift register. An one-dimensional *erosion* stage can be represented as a shift register, similar to the *dilation* stages. The difference is, that an *One* refills the empty side of the register after every shift operation. A second difference is, that the register is reset to *Zeros*, when the input of this stage is *Zero.* This is intrinsically an *erosion.* The output of the *erosion* stage is the output of the shift register.

These stages are cascadable. So an *opening* transformation is done by a cascade of an *erosion* followed by a *dilation.* A *closing* transformation is performed by a cascade of a *dilation* followed by an *erosion.*

3.5.3. Evaluation

The *bit error rate* estimations for several *MOXFilters* with *structuring elements* of the size three, five, seven, nine, and eleven are shown in fig. 3.20. The *MO5Filter*, which has a *structuring element* with the size of five, shows the best performance. This filter needs about 3dB to 5dB less $\frac{E_b}{N_0}$ than the original receiver for the same *BER* performance.

With an $\frac{E_b}{N_0}$ of 15dB the *BER* is less than 1%.

The *message success rates* for several *MOXFilters* with *structuring elements* of the size three, five, seven, nine, and eleven are shown in fig. 3.21. The *MO5Filter* shows again the best performance. It needs about 4dB to 6dB less $\frac{E_b}{N_0}$ for the same *MSR* performance than the original receiver. It can be seen that at an $\frac{E_b}{N_0}$ of 11dB, 10% of the sent messages are received correctly. For 90% correctly received messages the $\frac{E_b}{N_0}$ has to be above 18dB.

The *MO9Filter* and the *MO11Filter* have worse performances than the original receiver due to too long filters.

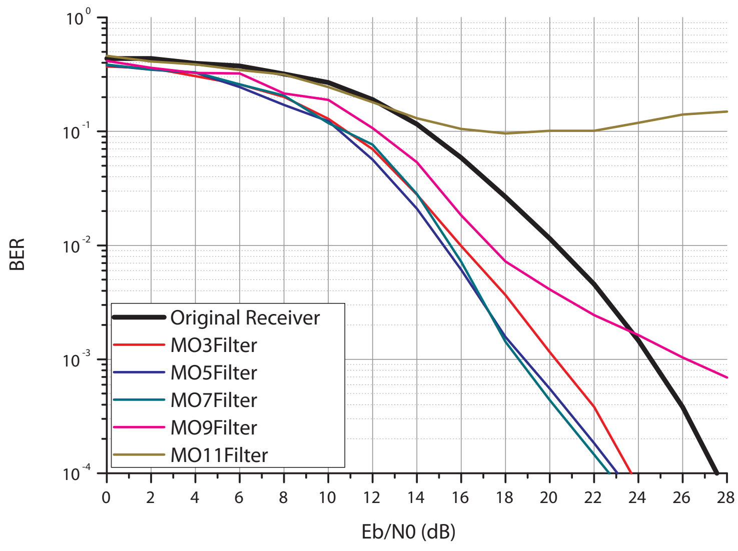


Figure 3.20.: This figure shows the *BER* of several *MOXFilter* with a *structural element* with the size of three to eleven.

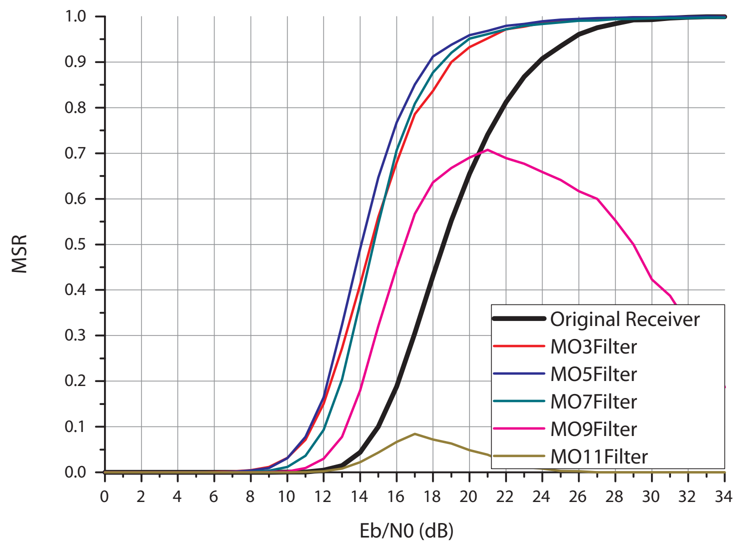


Figure 3.21.: This figure shows the *MSR* of several *MOXFilter* with a *structural element* with the size of three to eleven.

3.6. Correlation Chip Detector

3.6.1. Theory

The best method to estimate the information of a noisy signal is the use of a *matched filter*. If we filter a received signal $r(t) = s(t) + n(t)$, where $s(t)$ is the sent signal and $n(t)$ is the channel noise, the mathematical description for the output signal $z(t)$ is given by the convolution of $r(t)$ with the impulse response of the matched filter $h(t)$, i.e.,

$$z(t) = r(t) * h(t) = \int_0^t r(\tau)h(t-\tau)d\tau. \tag{3.14}$$

For a *matched filter* the impulse response $h(t)$ is denoted as

$$h(t) = \begin{cases} ks(T-t) & 0 \leq t \leq T \\ 0 & \text{otherwise.} \end{cases} \tag{3.15}$$

This leads us, for the time $t = T$ and $k = 1$, to the following equation

$$z(T) = \int_0^T r(\tau)s(\tau)d\tau \tag{3.16}$$

which is also known as a *correlation* [1].

In a digital system we always have more than one symbol, therefore we have to perform the correlation function with each symbol and the filter impulse responses. To decide which symbol was transmitted, the value with the largest correlation value is chosen. In our specific example we consider two digital symbols, a *One* and a *Zero* symbol. So we estimate the correct symbol by a simple subtraction of the correlator outputs, i.e.,

$$z[N] = \sum_{n=0}^{N} \overline{r[n] \otimes h_1[n]} - \sum_{n=0}^{N} \overline{r[n] \otimes h_2[n]}, \tag{3.17}$$

where $\otimes$ represents the *binary XOR* operator and $\overline{x}$ represents the *binary NOT* operation on x. Is $z[N]$ larger than zero, symbol h_1 is taken and symbol h_2 otherwise.

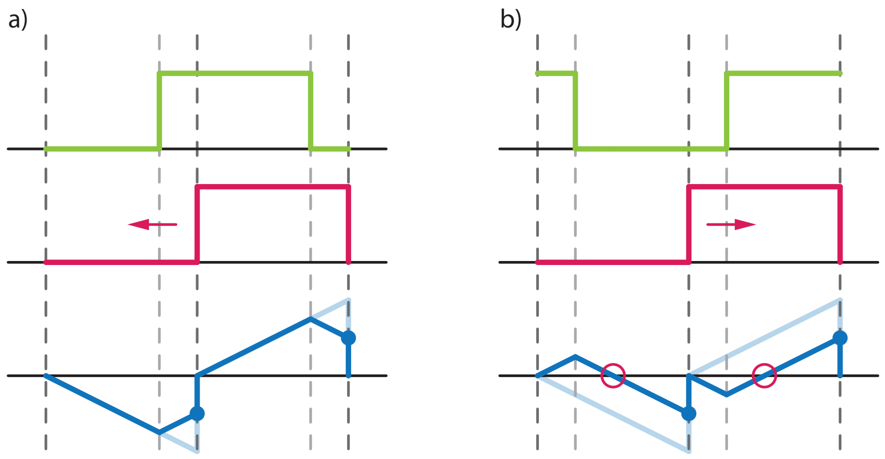


Figure 3.22.: The *Correlation Chip Detector* tries to get in phase with the input sequence. The green plots in a) and b) show the input signals for two different phase shift cases. The red plots are from the *CorrCD* generated data clock cycles. The blue plots show the behavior of the correlation process. To know if the clock cycle should be shifted to the left or to the right, zero crossings of the correlator are detected. If a zero crossing is detected (b) the clock cycle has to be shifted right and to the left otherwise (a).

We have seen that the best estimation of a chip corresponds to a correlation (equ. 3.16). But the output of this function depends on the starting time $t = 0$. If the correlation function has a phase offset to the input signal, the correlation is not performed correctly. This means that the *Correlation Chip Detector* (*CorrCD*) needs the ability to synchronize the correlation function with the phase of the incoming signal. To do this, the correlator detects whether the maximum or minimum possible correlation values are reached and whether there was a zero crossing during the summation of the correlation. In case the maximal or minimum possible correlation values are not reached for an ideal input signal means, that there exists a phase offset between the signal and the correlation cock. This is shown in fig. 3.22. So the start of the correlation has to be shifted in time. This is done by shortening the next correlation period or on extending the actual one by one sample. To know whether it needs to shortened or extended the zero crossing is important. If there was a zero crossing, then the actual period is going to be extended, otherwise the next period is going to be shortened. After several chips, the *Correlation Chip Detector* is in phase with the input signal. Noise will activate drift of the *Correlation Chip Detector*. Under the assumption that the noise is zero mean, the clock of the chip detector will jitter around its ideal starting point.

3.6.2. Design

Because our chip symbols are orthogonal, a *One* symbol ($h[n]$) consists of a certain amount of *One* samples and the *Zero* symbol ($\bar{h}[n]$) of the same amount of *Zero* samples, we can merge the sums to

$$z[N] = \sum_{n=0}^{N} \left(\overline{r[n] \otimes h[n]} - \overline{r[n] \otimes \bar{h}[n]} \right). \tag{3.18}$$

This sum can also be represented as

$$z[N] = \sum_{n=0}^{N} (-1)^{r[n] \otimes h[n]}, \tag{3.19}$$

where only one chip symbol ($h[n]$) is needed. This sum represents a counter, which counts up if the signal $r[n]$ matches the chip symbol $h[n]$ and counts down, otherwise.

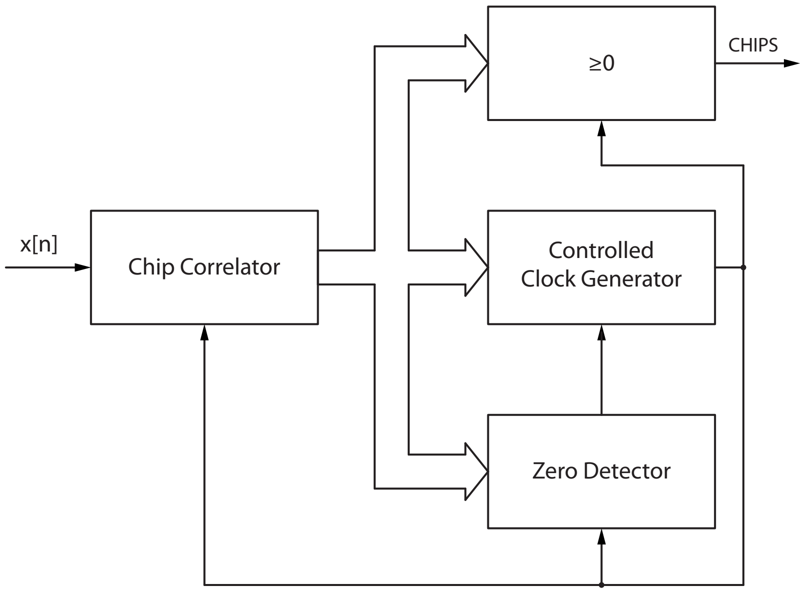


Figure 3.23.: Block architecture of the *Correlation Chip Detector*.

The block architecture of the *CorrCD* is shown in fig 3.23. The output of the *chip correlator* is the actual correlation value. A *zero detector* detects zero crossings of the *correlator* output. A *controlled clock generator* produces an internal data clock and tries to get in phase with the input sequence. To decide whether the clock should be shifted left or right, the *controlled clock generator* uses informations of the *zero detector* and the output of the *chip correlator*. After every chip correlation cycle a voter decides if the output value of the *chip correlator* is greater or equal zero or not. If it is greater or equal zero the output *CHIPS* is *One*, and *Zero* otherwise.

3.6.3. Evaluation

The *bit error rate* estimation is shown in fig. 3.24. The *CorrCD* needs about 4dB to 8dB less $\frac{E_b}{N_0}$ than the original receiver for the same *BER* performance. With an $\frac{E_b}{N_0}$ of 14dB the *BER* is less than 1%.

The *message success rate* is shown in fig. 3.25. The *CorrCD* needs about 4dB to 6dB less $\frac{E_b}{N_0}$ for the same *MSR* performance than the original receiver. It can be seen that at an $\frac{E_b}{N_0}$ of 11dB, 10% of the sent messages are received correctly. For 90% correctly received messages the $\frac{E_b}{N_0}$ has to be 18dB.

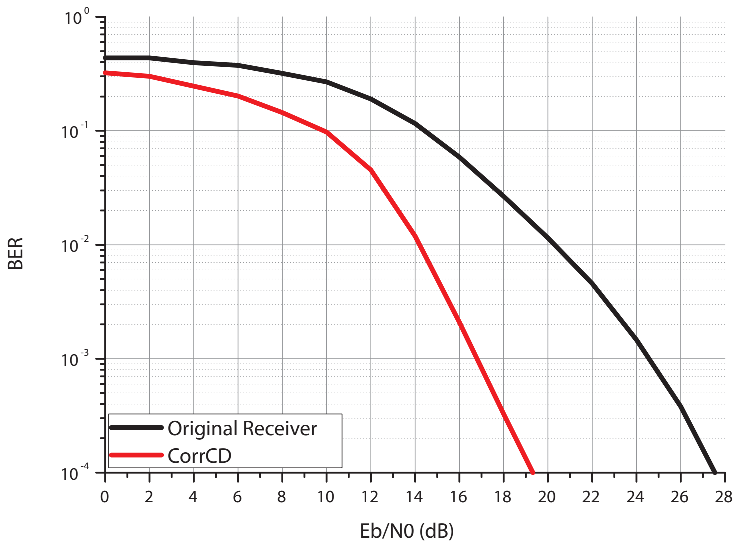


Figure 3.24.: This figure shows the *BER* of the *Correlation Chip Detector*.

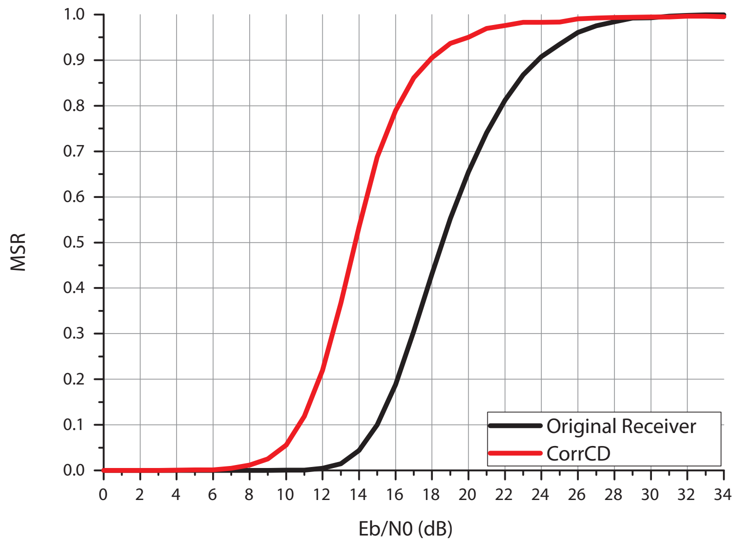


Figure 3.25.: This figure shows the *MSR* of the *Correlation Chip Detector*.

3.7. Correlation Manchester Decoder

3.7.1. Theory

The *Correlation Manchester Decoder* (*CorrMD*) is an upgrade to the *Correlation Chip Detector*. The *CorrCD* is used to synchronize the internal data clock to the input sequence. Additionally the *Correlation Manchester Decoder* has a bit correlator, which correlates over whole bit symbols. The correlation function stays more or less the same. In contrast to the chip correlator, however, the *Correlation Manchester Decoder* operates on bit level instead of the chip level. A bit symbols consists of two different chip symbols, due to Manchester coding. So the counter for the correlation needs to have twice the range of a chip symbol correlator.

3.7.2. Design

The expanded block architecture of the *CorrMD* is shown in fig. 3.26. The *bit correlator* produces a correlation output which is going to be evaluated after every bit symbol. If the correlation value is higher or equal to zero, the output *BITS* is *One*, *Zero* otherwise. Note that the *CHIP* signal is still required to detect the *SyncPattern*.

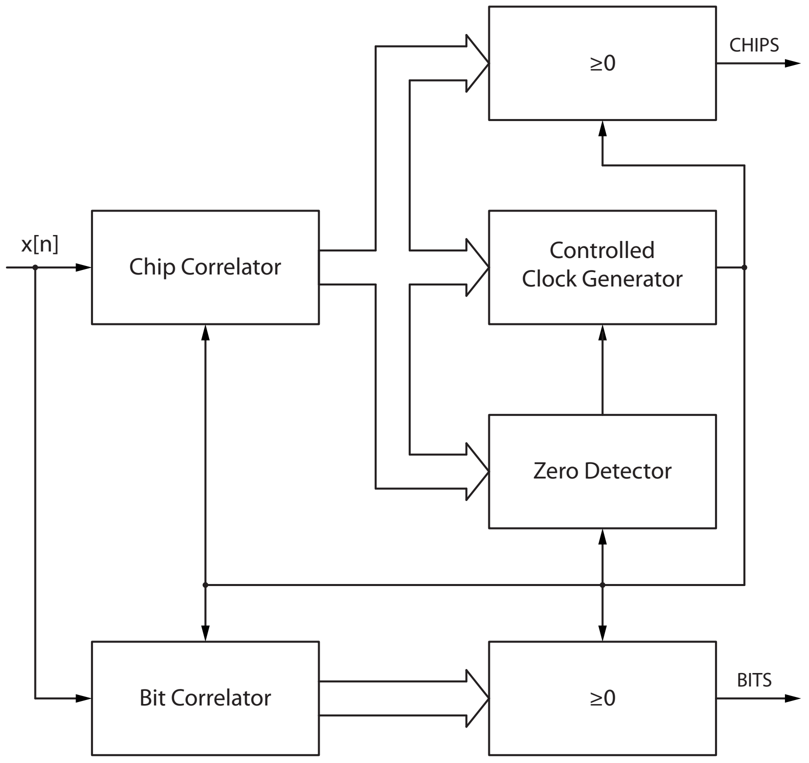


Figure 3.26.: Block architecture of the *Correlation Manchester Decoder*.

3.7.3. Evaluation

The *bit error rate* estimation is shown in fig. 3.27. The *CorrMD* needs about 6dB to 11dB less $\frac{E_b}{N_0}$ than the original receiver for the same *BER* performance. With an $\frac{E_b}{N_0}$ of 13dB the *BER* is less than 1%.

The *message success rate* is shown in fig. 3.28. The *CorrMD* needs about 5dB to 6dB less $\frac{E_b}{N_0}$ for the same *MSR* performance than the original receiver. It can be seen that at an $\frac{E_b}{N_0}$ of 10dB, 10% of the sent messages are received correctly. For 90% correctly received messages the $\frac{E_b}{N_0}$ has to be 18dB.

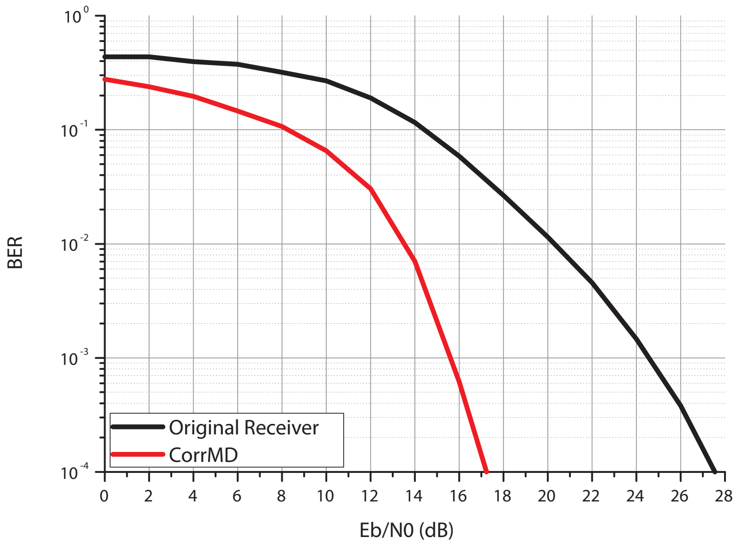


Figure 3.27.: This figure shows the *BER* of the *Correlation Manchester Decoder*.

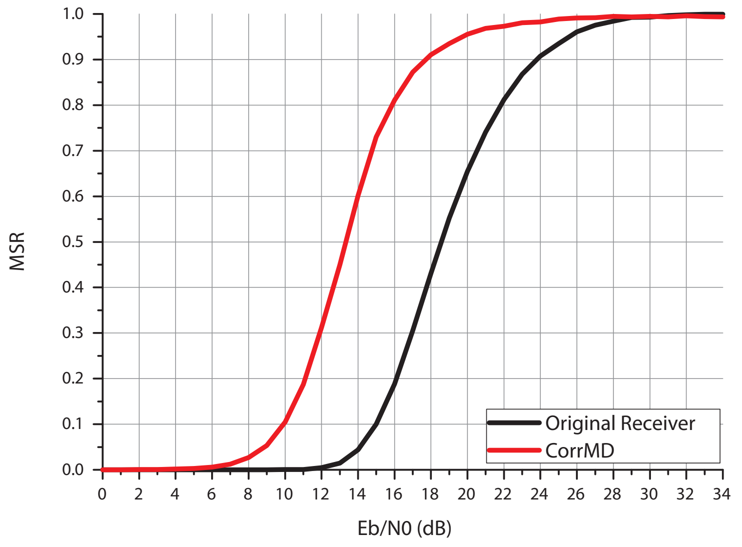


Figure 3.28.: This figure shows the *MSR* of the *Correlation Manchester Decoder*.

4. Summary of Results and Comparison

4.1. Overview

In chapter 3 we have introduced several filters that enhance the ability to receive correct data. For each method we have shown the *bit error rates* and *message success rates*. In this chapter these filters are compared for different characteristics. This is also done in combination with the *Correlation Chip Detector* and the *Correlation Manchester Decoder*.

4.2. Bit Error Rate

All filters which have been discussed in chapter 3 have the ability to improve the *bit error rate*. The best filters of each design are summarized in tab. 4.1 and fig. 4.1. In this table, the $\frac{E_b}{N_0}$ of the best filters is evaluated for several *BER* values. It can be seen that for a *BER* of 10^{-1} the best filter is the *Median7Filter*. For a *BER* of 10^{-2} the best filter is the *MO5Filter*. For lower *BER* the best filter is again the *MO7Filter*. So there is none filter which can be declared as the 'best' filter for the entire *BER* range. The *MO5Filter* gives good performance over the whole *BER* range. Basically a filter can reduce the $\frac{E_b}{N_0}$ requirements for similar *BER* performance, by up to 6dB.

In tab. 4.1 the *BER* of the original receiver is also compared to the *Correlation Chip Detector* and the *Correlation Manchester Decoder* without filters. It can be seen that the enhanced decoders *CorrCD* and *CorrMD* show a much better *BER* performance than the original receiver with its best performing filters. The *CorrCD* shows a better *BER* behavior with up to 8dB reduced $\frac{E_b}{N_0}$ requirements. The best *BER* performance without a filter has the *CorrMD* with up to 10dB reduced $\frac{E_b}{N_0}$ requirements.

The performance of the *CorrCD* can also be improved by filters (fig. 4.2). For low $\frac{E_b}{N_0}$ values the best filter is the *Median7Filter*. For higher $\frac{E_b}{N_0}$ values, the *MO5Filter* shows the best performance. Mainly the performance for low *BER* can be improved. For the *CorrMD* the *Median7Filter* shows the best performance (fig. 4.3). It should be noted, that all filters show a better performance with the *CorrMD* than with the *CorrCD*.

Filter	**BER**			
	10^{-1}	10^{-2}	10^{-3}	10^{-4}
		Original		
without filter	14.5dB	20.0dB	24.5dB	27.5dB
Count5Filter	11.0dB	16.5dB	21.0dB	24.0dB
Hyst2Filter	10.0dB	16.0dB	20.0dB	24.0dB
Median7Filter	**10.0dB**	15.5dB	20.0dB	23.5dB
MO5Filter	10.5dB	**15.0dB**	19.0dB	23.0dB
MO7Filter	11.0dB	15.5dB	**18.5dB**	**22.5dB**
		CorrCD		
without filter	10.0dB	14.5dB	17.0dB	19.5dB
Count3Filter	9.0dB	13.5dB	17.0dB	20.0dB
Hyst1Filter	9.0dB	14.0dB	17.0dB	20.0dB
Median7Filter	**8.0dB**	**13.0dB**	**16.5dB**	19.0dB
MO3Filter	9.0dB	13.5dB	16.5dB	19.5dB
MO5Filter	9.5dB	14.0dB	16.5dB	**19.0dB**
		CorrMD		
without filter	8.0dB	13.5dB	15.5dB	17.0dB
Count3Filter	7.5dB	12.5dB	14.5dB	16.5dB
Hyst1Filter	7.0dB	12.5dB	14.5dB	16.5dB
Median7Filter	**6.5dB**	**11.5dB**	**14.5dB**	**16.0dB**
MO3Filter	7.0dB	12.0dB	14.5dB	16.0dB

Table 4.1.: $\frac{E_b}{N_0}$ values for several *BER* for the best filters.

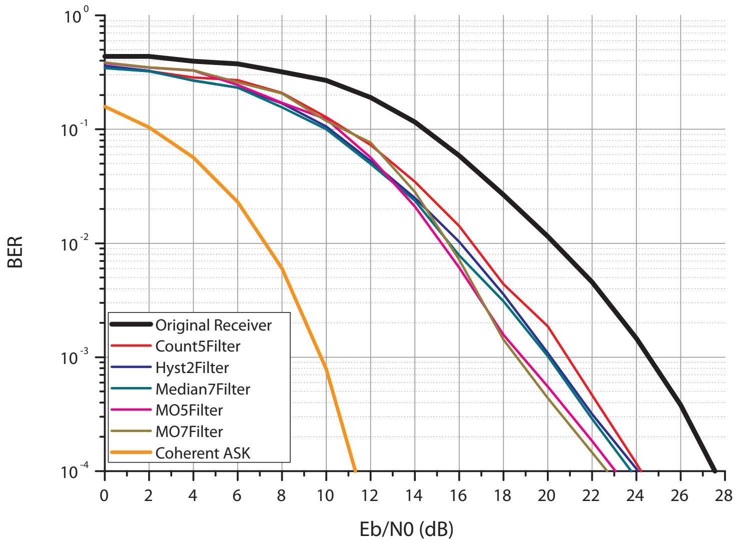


Figure 4.1.: *BER* of the original receiver with various filter and of an optimum coherent ASK receiver.

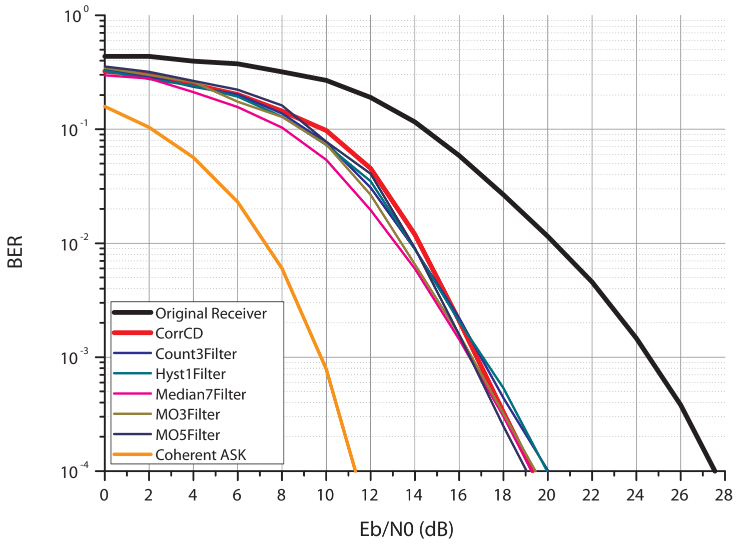


Figure 4.2.: *BER* of the *Correlation Chip Detector* with various filter and of an optimum coherent ASK receiver.

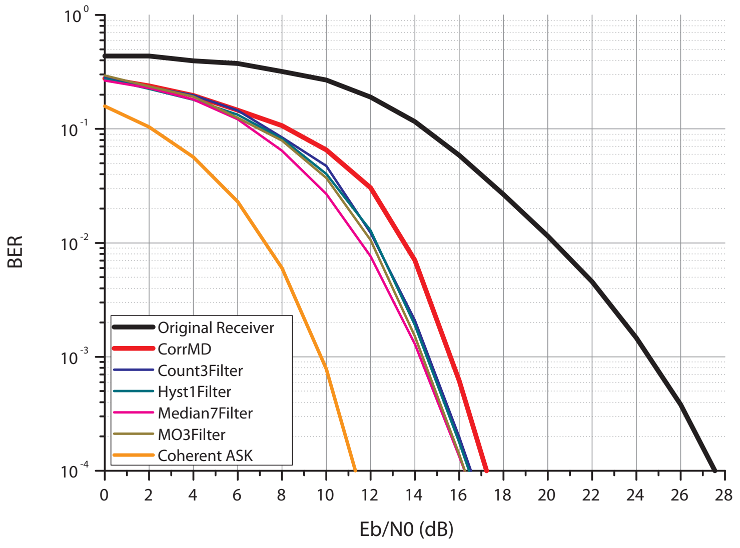


Figure 4.3.: *BER* of the *Correlation Manchester Decoder* with various filter and of an optimum coherent ASK receiver.

4.3. Message Success Rate

The filters introduced in chapter 3 can improve the *MSR* performance of the original receiver by 6dB (fig. 4.4). The best performance for low *MSR* shows the *Median7Filter*. For higher *MSR* the best filter is the *MO5Filter*. The $\frac{E_b}{N_0}$ for several *MSR* values are summarized in tab. 4.2.

The *CorrCD* (fig. 4.5) and the *CorrMD* (fig. 4.6) show a better *MSR* performance than the original receiver with its best filters. The *CorrCD* and the *CorrMD* need up to 6dB less $\frac{E_b}{N_0}$ than the original receiver.

The *MSR* performance of the *CorrCD* and the *CorrMD* can also be enhanced with filters. They show their best performance at a low *MSR* with the *Median7Filter*. For an *MSR* of about 0.5 the best filter is the *MO3Filter*. For high *MSR* values the best filter is the *MO5Filter*. A compromise for a good *MSR* performance in all regions is the *MO3Filter*. The *CorrMD* with its best filters shows an *MSR* performance improvement of up to 1dB than the *CorrCD* with its best filters.

Filter	**MSR**		
	0.1	0.5	0.9
	Original		
without filter	15.0dB	18.5dB	24.0dB
Count5Filter	11.0dB	15.0dB	20.0dB
Hyst2Filter	11.0dB	14.5dB	19.0dB
Median7Filter	**10.5dB**	14.0dB	19.0dB
MO5Filter	11.0dB	**14.0dB**	**18.0dB**
	CorrCD		
without filter	10.5dB	14.0dB	18.0dB
Count3Filter	10.0dB	13.0dB	18.0dB
Hyst1Filter	9.5dB	13.0dB	18.0dB
Median7Filter	**9.0dB**	12.5dB	17.5dB
MO3Filter	9.0dB	**12.5dB**	17.0dB
MO5Filter	10.0dB	13.0dB	**16.5dB**
	CorrMD		
without filter	10.0dB	13.0dB	17.5dB
Count3Filter	9.0dB	12.5dB	18.0dB
Hyst1Filter	9.0dB	12.5dB	17.5dB
Median7Filter	**8.5dB**	12.0dB	17.5dB
MO3Filter	8.5dB	**11.0dB**	17.0dB
MO5Filter	10.0dB	12.5dB	**16.0dB**

Table 4.2.: $\frac{E_b}{N_0}$ values for several *MSR* for the best filters.

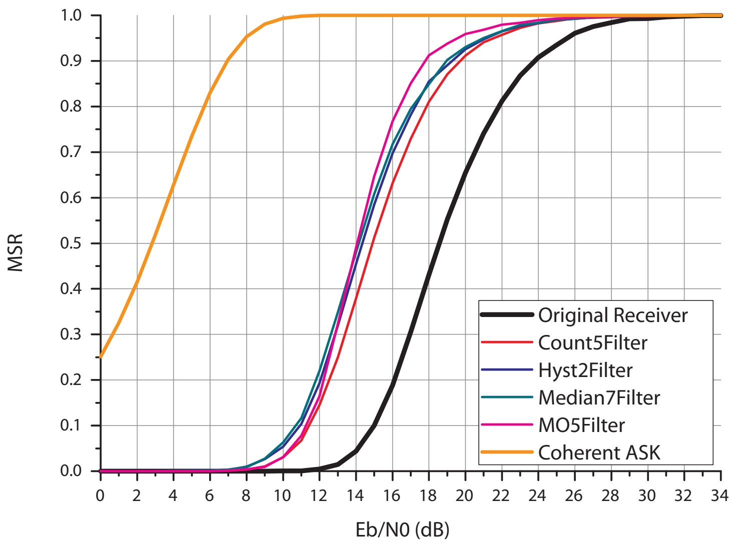


Figure 4.4.: *MSR* of the original receiver with various filters and of an optimum coherent ASK receiver.

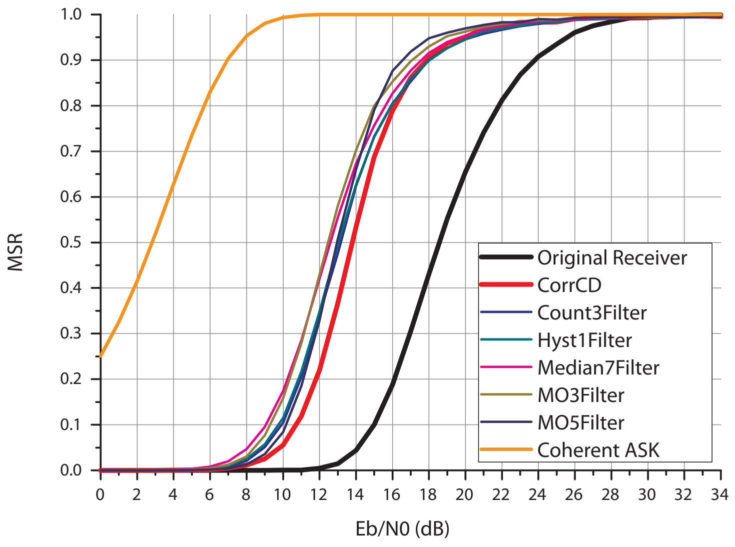


Figure 4.5.: *MSR* of the *Correlation Chip Detector* with various filters and of an optimum coherent ASK receiver.

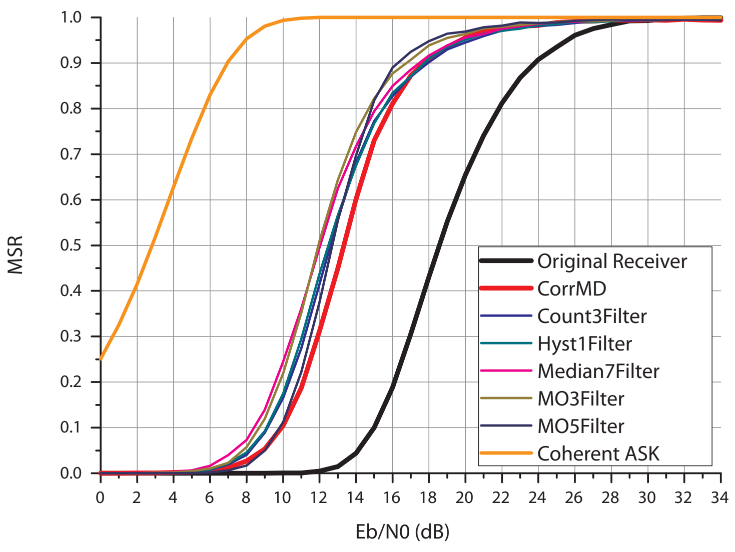


Figure 4.6.: *MSR* of the *Correlation Manchester Decoder* with various filters and of an optimum coherent ASK receiver.

4.4. Complexity

The complexity is measured in the number of flip-flops which are used for the filter or the detector. Because both are clocked with 90kHz, every flip-flop increases the energy consumption of the receiver. Also the area on the chip increases with every single flip-flop. For less power consumption and less chip area, a filter with a small number of flip-flops is therefore better. The number of flip-flops N_{FF} for the different filters in the dependence of their filter length constants are listed in the following equations:

$$N_{FF,CountXFilter} = \lceil \log_2 (maxcount + 1) \rceil + 1 \tag{4.1}$$
$$N_{FF,HystXFilter} = \lceil \log_2 (3 \cdot statesize) \rceil + 1 \tag{4.2}$$
$$N_{FF,MedianXFilter} = filtersize + \lceil \log_2 (filtersize + 1) \rceil + 2 \tag{4.3}$$
$$N_{FF,MOXFilter} = 4 \cdot masksize. \tag{4.4}$$

It can be seen, that the number of flip-flops of the *CountXFilter* and the *HystXFilter* are increasing less than for the other filters. The fastest increasing number of flip-flops has the *MOXFilter*. The number of flip-flops N_{FF} for the best filters for the original receiver are listed in tab. 4.3. The smallest number of flip-flops have the *Count5Filter* and the *Hyst2Filter*. The highest number of flip-flops have the *MO5Filter* and the *MO7Filter*.

The number of flip-flops of the different decoders (*Original*, *CorrCD*, and *CorrMD*) is shown in tab. 4.3. It can be seen, that the original receiver has half of the flip-flops than the two more advanced decoders. But when using the original receiver with a simple filter (e.g. *CountXFilter*) the number of flip-flops is almost the same like the number of flip-flops of the *CorrCD*. With a *Median7Filter* or one of the *MOXFilters*, the original receiver needs more flip-flops than the *Correlation Manchester Decoder*. It has to be mentioned that the *Correlation Chip Detector* and the *Correlation Manchester Decoder* have better *BER* performances than the original receiver with any filter.

Filter/Decoder	**Number of Flip-Flops**
Count5Filter	+4
Hyst2Filter	+4
Median7Filter	+12
MO5Filter	+20
MO7Filter	+28
Original	6 (+0)
CorrCD	11 (+5)
CorrMD	15 (+9)

Table 4.3.: Number of flip-flops for the specific filter or decoder.

4.5. Summary

In this chapter all enhanced detection schemes were evaluated for their *bit error rate*, *message success rate*, and their complexity. It can bee seen, that the best compromise for the original receiver is the *MO5Filter* (tab. 4.1, fig. 4.1), which gives a good performance over the entire *BER* and *MSR* ranges. Because of the high complexity of this filter (tab. 4.3) a change to an other type of receiver is recommend. A possibility is to use the *Correlation Chip Detector*. This chip detector has a better *BER* and *MSR* performance than the original receiver with any filter but less additional complexity. The number of flip-flops of the *CorrCD* is almost equal to the number of flip-flops of the original receiver with the simple *Count5Filter*. The original receiver with the *MO5Filter* needs more than twice the number of flip-flops than the *CorrCD* itself.

To improve the performance of the *Correlation Chip Detector* it is recommend to use the *Correlation Manchester Decoder* instead. The *CorrMD* gives a better *BER* and a better *MSR* performance than the the *CorrCD* with any filter. Also the number of flip-flops of the *CorrMD* is less or equal than the number of flip-flops of the *CorrCD* with any filter.

The best performance is obtained from a combination of the *Correlation Manchester Decoder* with the *Median7Filter*. It needs 16dB to have a *BER* of 10^{-4}. This is 11.5dB less $\frac{E_b}{N_0}$ than the original receiver needs for the same performance. This increase of performance has the cost of 21 additional flip-flops which increases the power consumption and the needed chip area.

These improvements are possible due to the oversampling of the datastream. A single wrong sample could disturb the chip detection process of the original receiver. Because these different introduced filters use always several samples, wrong samples could be filtered out.

5. Conclusion and Outlook

This thesis has shown that it is possible to enhance the low complexity limiter based LF receiver with digital 1bit filters. These filters were designed for low power consumption and low complexity.

The data transition is done with Manchester coded data with an OOK modulation with carrier frequency of 125kHz. The receiver consists of a *limiter* structure with a *RSSI* generator. This *RSSI* signal is lowpass filtered and compared to a threshold to produce a 1bit amplitude quantized data stream. At the end of the *analog front end* this 1bit quantized continuous time signal is sampled and input to the *digital baseband processor*. The *digital baseband processor* detects the chips and decodes the data bits.

Between the *AFE* and the *DBP*, digital 1bit filters are implemented. Four different filter designs have been introduced in this thesis. The simplest filter was the *CountX-Filter*, which is a free running counter with a threshold comparator. The second filter, which has been introduced, is the *HystXFilter*. This filter is similar to the *CountXFilter*, but the decision is done with a hysteresis. The third filter was the *MedianXFilter*, which is an ordinary median filter. The most complex filter is the *MOXFilter*. This filter uses the theory of *mathematical morphology*. All these filters enhance the capability of the receiver to receive correct data. The best performance with respect to the *bit error rate* (*BER*) and *message success rate* (*MSR*) have been the *Median7Filter*, the *MO5Filter*, and the *MO7Filter*. The *Median7Filter* yields the best performance for low $\frac{E_b}{N_0}$ and the *MO7Filter* yields the best performance for high $\frac{E_b}{N_0}$. The *MO5Filter* is a good compromise for the entire $\frac{E_b}{N_0}$ range.

Additionally two new decoders have been introduced. The first decoder was the *Correlation Chip Detector* (*CorrCD*). This decoder tries to get in phase with the data clock and correlates the input stream with a chip symbol. The more complex decoder is the *Correlation Manchester Decoder* (*CorrMD*). It uses the *CorrCD* as basement. Additional it has a correlator, which correlates over a whole bit symbol. Therefore it has a slightly better performance than the *CorrCD*. These advanced decoders have a better performance than the original receiver with any of the filters. Certainly it is possible to use 1bit filters to enhance the performance of these advanced decoders. For the *CorrCD* the *Median7Filter* enhances the performance at a low $\frac{E_b}{N_0}$ area. For the *CorrMD* the *Median7Filter* enhances the performance over the entire $\frac{E_b}{N_0}$ range.

The next step for further development and research is to implement these filters and decoders in silicon (e.g ASIC or FPGA). Based on the chip realization real time measurement could be done and the simulations could be verified. Also combinations of sequential different filters could be in interest. It is also possible to adopt these filters for other receiver architectures.

A. Magnetic Field and Oscillating Circuits

A.1. Magnetic Field and the Biot-Savart Law

The ancient Greek reported about strange behaviors of stones they found near the city of Magnesia. Some stones seem to adduct to each other, other stones seem to reject. Nowadays we name this behavior after the city it first occurred: magnetism.

Experiments have shown that there exist no magnetic sources or sinks but so called poles, the north and south pole, where the magnetic lines of force leave and enter the magnetic material. That is the reason why only opposite poles attract. Mathematically magnetism can be represented as a magnetic vector field.

In the year 1802 the Italian jurist, economist and physicist Gian Domenico Romagnosi was the first who found a link between magnetism and electricity. The Danish physicist and chemist Hans Christian Ørsted published 1820 a paper about the first systematic experiments of electromagnetic occurrences of a current-driven wire and a compass needle.

The fundamental equations for the electromagnetic physics are the Maxwell's equations which state for the general case

$$\nabla \cdot \mathbf{D} = \rho \qquad \text{...Gauss's law} \tag{A.1}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \qquad \text{...Faraday's law of induction} \tag{A.2}$$

$$\nabla \cdot \mathbf{B} = 0 \qquad \text{...Gauss's law of magnetism} \tag{A.3}$$

$$\nabla \times \mathbf{H} = \mathbf{J} + \frac{\partial \mathbf{D}}{\partial t} \qquad \text{...Ampere's law} \tag{A.4}$$

with the constitutive relations

$$\mathbf{B} = \mu \mathbf{H} \tag{A.5}$$

$$\mathbf{D} = \epsilon \mathbf{E} \tag{A.6}$$

$$\mathbf{J} = \sigma \mathbf{E} \qquad \text{...Ohm's law,} \tag{A.7}$$

where $\mathbf{D}$ is the electric flux density, $\mathbf{E}$ is the electric field, $\mathbf{B}$ is the magnetic flux density, $\mathbf{H}$ is the magnetic field, $\mathbf{J}$ is the vectorial current density, ρ is the charge density, ϵ is the electric constant, μ is the magnetic constant and σ is the electrical conductivity.

For a static case the Ampere's law (equ. A.4) reduces to

$$\nabla \times \mathbf{H} = \mathbf{J}, \tag{A.8}$$

which reflects the results of the Ørsted-experiments. These equations are still unhandy. To get out of this dilemma the vector potential $\mathbf{A}$ is introduced. This vector potential is defined with the characteristic of

$$\nabla \cdot \nabla \times \mathbf{A} = 0. \tag{A.9}$$

So it is possible to represent the magnetic field $\mathbf{H}$ as

$$\mathbf{H} = \frac{1}{\mu} \nabla \times \mathbf{A}. \tag{A.10}$$

Now we substitute the magnetic field $\mathbf{H}$ in the static Ampere's law (equ. A.8). This leads to the relation

$$\nabla \times \nabla \times \mathbf{A} = \mu \mathbf{J} \tag{A.11}$$

which can be represented as [13]

$$\nabla \nabla \cdot \mathbf{A} - \Delta \mathbf{A} = \mu \mathbf{J}. \tag{A.12}$$

Because of the Helmholz-theorem [15] $\nabla\nabla \cdot \mathbf{A}$ can be set freely to 0, the previous equation can be simplified to

$$\Delta \mathbf{A} = -\mu \mathbf{J}. \tag{A.13}$$

The evaluation of this equation leads to the following triple integral:

$$\mathbf{A}(\mathbf{r}) = \frac{\mu}{4\pi} \iiint \frac{\mathbf{J}(\tilde{\mathbf{r}})}{\|\mathbf{r} - \tilde{\mathbf{r}}\|} d\tilde{V}. \tag{A.14}$$

Now we have all we need to calculate the magnetic field $\mathbf{H}$. With the equations A.14 and A.10 we get the Biot-Savart law:

$$\mathbf{H}(\mathbf{r}) = \frac{1}{4\pi} \nabla \times \iiint \frac{\mathbf{J}(\tilde{\mathbf{r}})}{\|\mathbf{r} - \tilde{\mathbf{r}}\|} d\tilde{V}. \tag{A.15}$$

In words this equation means, that the magnetic field $\mathbf{H}$ at a certain position $\mathbf{r}$ depends on all existing vector current densities and their distance to the position $\mathbf{r}$. [15, 16]

A.2. Self Inductance and Mutual Inductance

Faraday's law of induction (equ. A.16) states that a changing magnetic flux $\Phi(t)$ generates a voltage $u_i(t)$ in a closed circuit. The voltages $u_i(t)$ is called inductive voltage.

$$u_i(t) = \frac{d\Phi(t)}{dt} \qquad \text{...Faraday's law of induction} \tag{A.16}$$

$$\Phi(t) = \iint \mathbf{B}(\mathbf{r}, t)\, d\mathbf{\Gamma} \qquad \text{...magnetic flux} \tag{A.17}$$

It doesn't matter if the magnetic flux $\Phi(t)$ is changing due to a changing $\mathbf{B}$ over time or a changing area $\mathbf{\Gamma}$.

Because every current produces a magnetic field in its proximity a change of this current leads to a generation of an inductive voltage in its own circuit. This is called self-induction. If there are just magnetic neutral materials or materials with constant permeability in the proximity of the closed circuit, the magnetic flux density is proportional to the current at every time because of the Biot-Savart law (equ. A.15). So the magnetic flux can be written as

$$\Phi = Li, \tag{A.18}$$

where, i is the current and L is the coefficient of induction, also called as self inductance. In this case the law of induction is a very well known:

$$u_i\left(t\right) = L\frac{di\left(t\right)}{dt}. \tag{A.19}$$

The self inductance of a cylindric coil can be determined with the following equations: With the Biot-Savart law (equ. A.15) the magnetic field $\mathbf{H}$ in the inner of a coil with the length l and with n windings per meter can be estimated as

$$H = ni, \tag{A.20}$$

because the magnetic field in a long thin coil is homogeneous and for a coil with N windings following equations are valid:

$$\oint \mathbf{H}d\mathbf{s} \approx \int_B^A Hds = HL = Ni \tag{A.21}$$

$$H = \frac{N}{L}i. \tag{A.22}$$

The magnetic flux density is then $B = \mu ni$. The magnetic flux for the cross section of the cylindric coil is

$$\Phi = B\Gamma = \mu n\Gamma i. \tag{A.23}$$

With a variation of the current in the coil the change of the flux is

$$\frac{d\Phi}{dt} = \mu n\Gamma\frac{di}{dt}. \tag{A.24}$$

Therefore the inductive voltage between the two ends of the coil with $N = nl$ windings can be estimated as

$$u_i = N\Phi = \mu n^2 l\Gamma\frac{di}{dt} = L\frac{di}{dt}. \tag{A.25}$$

The self inductance L is then

$$L = \mu n^2 l\Gamma. \tag{A.26}$$

We know from the derivation of the Biot-Savart law that a current, in our case in a closed circuit, induces at a position $\mathbf{r}_2$ an magnetic field $\mathbf{H}$ with the vector potential $\mathbf{A}\left(\mathbf{r}_2\right)$ (equ. A.14). If there exists a second closed circuit near the first one the magnetic field causes a magnetic flux

$$\Phi_2 = \iint \mathbf{B}d\mathbf{\Gamma}_2 = \iint \nabla \times \mathbf{A}d\mathbf{\Gamma}_2 = \int \mathbf{A}d\mathbf{s}_2 \tag{A.27}$$

through the area $\mathbf{\Gamma}_2$. For the case of very thin wires, the magnetic vector potential can be simplified to

$$\mathbf{A}(\mathbf{r}_2) = \frac{\mu i_1}{4\pi} \int \frac{d\mathbf{s}_1}{\|\mathbf{r}_{12}\|} \tag{A.28}$$

and the magnetic flux becomes

$$\Phi_2 = \frac{\mu i_1}{4\pi} \iint \frac{d\mathbf{s}_1 d\mathbf{s}_2}{\|\mathbf{r}_{12}\|}. \tag{A.29}$$

Due to A.18 the inductance is

$$L_{12} = L_{21} = M = \frac{\mu}{4\pi} \iint \frac{d\mathbf{s}_1 d\mathbf{s}_2}{\|\mathbf{r}_{12}\|}, \tag{A.30}$$

which is called mutual inductance M. Therefore the mutual inductance is the coefficient of induction when a current in the first closed circuit induces a inductive voltage into the second open circuit, so that the inductive voltage is proportional to M [8,15]:

$$u_{i2}(t) = -M\frac{di_1(t)}{dt}. \tag{A.31}$$

A.3. Oscillating Circuits

The ideal oscillating circuit consists of an interconnection of a capacitor and a inductance. Because of Kirchhoff's voltage law the sum of all voltages is

$$\sum_i u_i(t) = u_L(t) + u_C(t) = 0. \tag{A.32}$$

With the typical characteristics of capacitances and inductances

$$u_L(t) = L\frac{di(t)}{dt} \tag{A.33}$$

$$i(t) = C\frac{du_C(t)}{dt} \tag{A.34}$$

we get the ideal oscillator equations for the capacitor voltage or the inductance current

$$\ddot{u}_C(t) + \frac{1}{LC}u_C(t) = 0 \tag{A.35}$$

$$\ddot{i}_L(t) + \frac{1}{LC}i_L(t) = 0 \tag{A.36}$$

with the solutions

$$u_C(t) = \hat{u}_C \cos(\omega_0 t + \varphi_0) \tag{A.37}$$

$$i_L(t) = \hat{i}_L \cos(\omega_0 t + \varphi_0) \tag{A.38}$$

$$\omega_0 = \sqrt{\frac{1}{LC}}, \tag{A.39}$$

where ω_0 is the angular frequency, φ_0 is the starting offset phase, $\hat{u}_C$ is the amplitude of the initial capacitance voltage and $\hat{i}_L$ is the amplitude of the initial inductance current. [16]

In a real implementation of an oscillating circuit we always have some Ohmic resistance. The resistance can lead to serial or parallel RLC circuits.

A.3.1. Serial RLC Circuits

In a serial circuit the current is the same for all components. The mathematical description is an ordinary differential equation for a damped harmonic oscillator

$$\ddot{i}(t) + \frac{R}{L}\dot{i}(t) + \frac{1}{LC}i(t) = 0 \tag{A.40}$$

with the general solution

$$i(t) = A_1 e^{-(\alpha-\beta)t} + A_2 e^{-(\alpha+\beta)t} \tag{A.41}$$

$$\alpha = \frac{R}{2L} \tag{A.42}$$

$$\beta = \sqrt{\frac{R^2}{4L^2} - \frac{1}{LC}}, \tag{A.43}$$

where A_1 and A_2 are constants. There exist three different cases: over damping, critical damping and under damping.

Over Damping

In this case is β a real value. Therefore A_1 and A_2 are real, too. With the initial condition of $i(0) = I_0$ and $\dot{i}(t) = \dot{I}_0$ we get the special solution

$$i(t) = I_0 e^{-\alpha t}\left[\cosh(\beta t) + \frac{\alpha}{\beta}\sinh(\beta t)\right]. \tag{A.44}$$

Critical Damping

In this case is β zero. With the initial condition of $i(0) = I_0$ and $\dot{i}(t) = \dot{I}_0$ the special solution is

$$i(t) = e^{-\alpha t}\left[I_0 + \left(\dot{I}_0 + \alpha I_0\right)t\right]. \tag{A.45}$$

Under Damping

In this case is β a complex value. The solution is

$$i(t) = I_0 e^{-\alpha t}\cos(\omega t + \varphi_0) \tag{A.46}$$

$$\omega = \sqrt{\frac{1}{LC} - \frac{R^2}{4L^2}}, \tag{A.47}$$

where I_0 is the amplitude and φ_0 is the starting offset phase.

Damped Driven Harmonic Oscillator

This is the case we are interested in. In this case the damped harmonic oscillator is driven by an input voltage. The equation of the damped driven harmonic oscillator is therefore

$$\ddot{u}_C(t) + \frac{R}{L}\dot{u}_C(t) + \frac{1}{LC}u_C(t) = \frac{1}{LC}u(t). \tag{A.48}$$

In the majority of cases the input voltage is a single sinusoidal

$$u(t) = U_0 \cos(\omega t), \tag{A.49}$$

with the angular frequency ω and amplitude U_0. Because of the forced input voltage $u(t) = U_0 \cos(\omega t)$ with a constant amplitude the amplitude of the circuit current $i(t) = I_0 \cos(\omega t - \varphi)$ is also constant, but with a phase offset φ.

The dissipated power of the resistor R is

$$P = i^2(t) R = \frac{u^2(t)}{Z^2} R = \frac{(U_0 \cos(\omega t))^2}{Z^2}, \tag{A.50}$$

with the complex electrical impedance Z

$$Z = R + j\left(\omega L - \frac{1}{\omega C}\right) \tag{A.51}$$

of the serial oscillating circuit. So the mean dissipated power of the resistor is

$$\overline{P} = \frac{1}{2}\frac{U_0^2 R}{R^2 + \left(\omega L - \frac{1}{\omega C}\right)^2} \tag{A.52}$$

which leads to a maximum dissipated power at $\omega = \omega_0$ of

$$\overline{P}_{max} = \frac{1}{2}\frac{U_0^2}{R}. \tag{A.53}$$

A.3.2. Parallel RLC Circuits

Parallel RLC circuits are very similar to the serial RLC cases. But instead of an oscillating circuit current $i(t)$ we have an oscillating capacity voltage $u_C(t)$

$$\ddot{u}_C(t) + \frac{1}{RC}\dot{u}_C(t) + \frac{1}{LC}u_C(t) = 0 \tag{A.54}$$

with the general solution

$$u_C(t) = A_1 e^{-(\alpha-\beta)t} + A_2 e^{-(\alpha+\beta)t} \tag{A.55}$$

$$\alpha = \frac{1}{2RC} \tag{A.56}$$

$$\beta = \sqrt{\frac{1}{4R^2C^2} - \frac{1}{LC}}. \tag{A.57}$$

So we can estimate that the general solution is more or less the same as the solution of the serial RLC oscillator. Therefore the cases of damped harmonic oscillators for a parallel RLC circuit are equivalent to the damped harmonic oscillator with a serial RLC circuit.

A.4. Coupled Resonant Circuits

As discussed before (sec. A.2) a current in a coil L_1 introduces a voltage in another coil L_2 which is in the magnetic field of the first one. On the other hand a current in the coil L_2 introduces a voltage in the first coil L_1. This is called magnetic coupling. That means we can couple two different circuits via the mutual inductance M. A setting of two coupled serial RLC circuits leads to this set of differential equations:

$$L_1 \frac{d^2 i_1}{dt^2} + R_1 \frac{di_1}{dt} + \frac{i_1}{C_1} = -M \frac{d^2 i_2}{dt^2} \tag{A.58}$$

$$L_2 \frac{d^2 i_2}{dt^2} + R_2 \frac{di_2}{dt} + \frac{i_2}{C_2} = -M \frac{d^2 i_1}{dt^2}. \tag{A.59}$$

With the approaches $i_1 = I_1 e^{j\omega t}$ and $i_2 = I_2 e^{j\omega t}$ for the solutions of the currents, the equations for the coupled system can be written as

$$\left(-L_1\omega^2 + j\omega R_1 + \frac{1}{C_1}\right) I_1 - \omega^2 M I_2 = 0 \tag{A.60}$$

$$-\omega^2 M I_1 + \left(-L_2\omega^2 + j\omega R_2 + \frac{1}{C_2}\right) I_2 = 0 \tag{A.61}$$

which only leads to a non trivial solution when the determinant is non-zero. For the simplifications $R_1 = R_2 = 0$, $L_1 = L_2 = L$, $C_1 = C_2 = C$ and $M = k\sqrt{L_1 L_2}$ we get two equations for the resonant frequency ω:

$$\omega_1 = \sqrt{\frac{1}{(L-M)\,C}} = \frac{\omega_0}{\sqrt{1-k}} \tag{A.62}$$

$$\omega_2 = \sqrt{\frac{1}{(L+M)\,C}} = \frac{\omega_0}{\sqrt{1+k}} \tag{A.63}$$

with

$$\omega_0 = \sqrt{\frac{1-\frac{M}{L}}{(L-M)\,C}}. \tag{A.64}$$

It can be seen that the coupling splits the frequency ω_0 of the uncoupled circuit into two frequencies $\omega_{1,2}$. $\Delta\omega = \omega_1 - \omega_2$ is for a weak coupling ($M << L$) $\Delta\omega = \omega_0 k$. [8]

B. EKV MOS Transistor Model

In ordinary MOS transistor models like in the Shichman-Hodges model [3], all voltages are referenced to the source potential. The EKV model differs in this because all voltages are referenced to the bulk potential. To change between these different referencing methods the following equations can be used:

$$\begin{aligned} V_G &= V_{gs} - V_{bs} \\ V_D &= V_{ds} - V_{bs} \\ V_S &= -V_{bs}. \end{aligned}$$

V_G is therefore the gate-bulk voltage, V_D is the drain-bulk voltage and V_S is the source-bulk voltage.

In tab. B.1 all parameters which are needed for the EKV model are listed . In tab. B.2 all used general constants are listed.

Name	**Description**	**Default Values**	**Unit**
COX	Gate oxide capacitance	-	F/m^2
VTO	Nominal threshold voltage	0.5	V
GAMMA	Body effect factor	1	$V^{1/2}$
PHI	Bulk Fermi potential	0.7	V
KP	Transconductance parameter	50E-6	A/V^2
THETA	Mobility reduction coefficient	0	$1/V$
DL	Channel length correction	0	μm
DW	Channel width correction	0	μm
LAMBDA	Depletion length coefficient	0.5	-
LETA	Short channel effect coefficient	0.1	-
WETA	Narrow width effect coefficient	0.25	-

Table B.1.: Parameters of the EKV Transistor model

Name	**Description**	**Values**	**Unit**
T	Model simulation temperature	300	K
k	Bolzmann constant	1.3807E-23	JK^{-1}
q	Magnitude of electron charge	1.602E-19	C

Table B.2.: General constants of the EK MOS Transistor model

With all these definitions we are able to calculate the drain-source current I_{DS} in one *simple* formula (equ. B.1) with several additional calculations [2, 6, 7].

$$\begin{aligned}
V_T &= \frac{kT}{q} \\
L_{eff} &= L + \mathrm{DL} \\
W_{eff} &= W + \mathrm{DW} \\
V_{GS} &= V_G - \mathrm{VTO} - \mathrm{PHI} + \mathrm{GAMMA} \cdot \sqrt{\mathrm{PHI}} \\
\gamma_s &= \mathrm{GAMMA} - \\
&\quad \frac{\varepsilon_{ox}\varepsilon_{si}}{\mathrm{COX}} \cdot \left(\frac{\mathrm{LETA}}{L_{eff}} \cdot \sqrt{\mathrm{PHI} + V_D} + \left(\frac{\mathrm{LETA}}{L_{eff}} - 3\frac{\mathrm{WETA}}{W_{eff}} \right) \cdot \sqrt{\mathrm{PHI} + V_S} \right) \\
V_P &= V_{GS} - \mathrm{PHI} - \gamma_s \cdot \left(\sqrt{V_{GS} + \left(\frac{\gamma_s}{2} \right)^2} - \frac{\gamma_s}{2} \right) \\
n &= 1 + \frac{\mathrm{GAMMA}}{2\sqrt{V_P + \mathrm{PHI} + 4V_T}} \\
\beta &= \mathrm{KP} \cdot \frac{W_{eff}}{L_{eff}} \cdot \frac{1}{1 + \mathrm{THETA} \cdot V_P} \\
I_S &= 2n\beta V_T^2 \\
I_F &= I_S \cdot \left[\log \left(1 + e^{\frac{V_G - \mathrm{VTO} - n \cdot V_S}{2nV_T}} \right) \right]^2 \\
I_R &= I_S \cdot \left[\log \left(1 + e^{\frac{V_G - \mathrm{VTO} - n \cdot V_D}{2nV_T}} \right) \right]^2 \\
I_{DS} &= (I_F - I_R) \cdot (1 + \mathrm{LAMBDA} \cdot V_{DS}) \,. \qquad \text{(B.1)}
\end{aligned}$$

C. BER and MSR Results

This appendix shows further *bit error rate* and *message success rate* results. Its main focus is the *Correlation Chip Detector* and the *Correlation Manchester Decoder* with all examined filter combinations.

C.1. Original Receiver

The *bit error rate* and the *message success rate* results of the original receiver are shown in fig. 2.25 and fig. 2.26. *BER* and *MSR* results with different filters can be seen in chapter 3.

C.2. Correlation Chip Detector

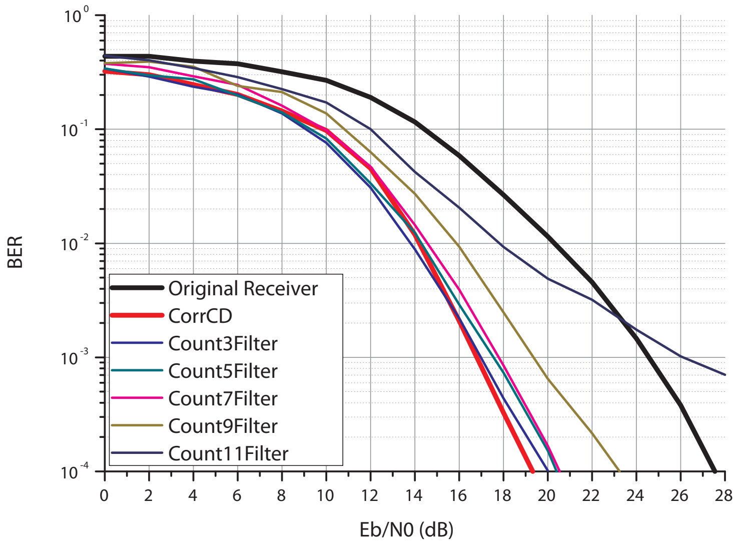


Figure C.1.: This figure shows the *BER* of the *Correlation Chip Detector* with *CountXFilter*.

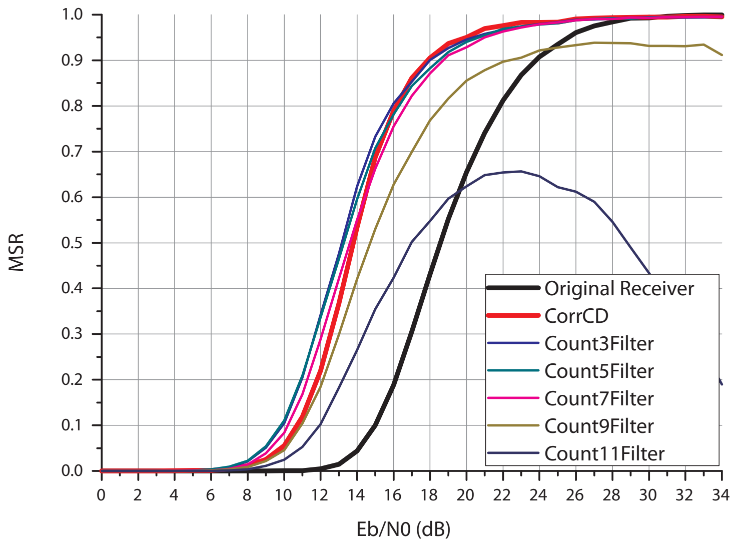


Figure C.2.: This figure shows the *MSR* of the *Correlation Chip Detector* with *CountXFilter*.

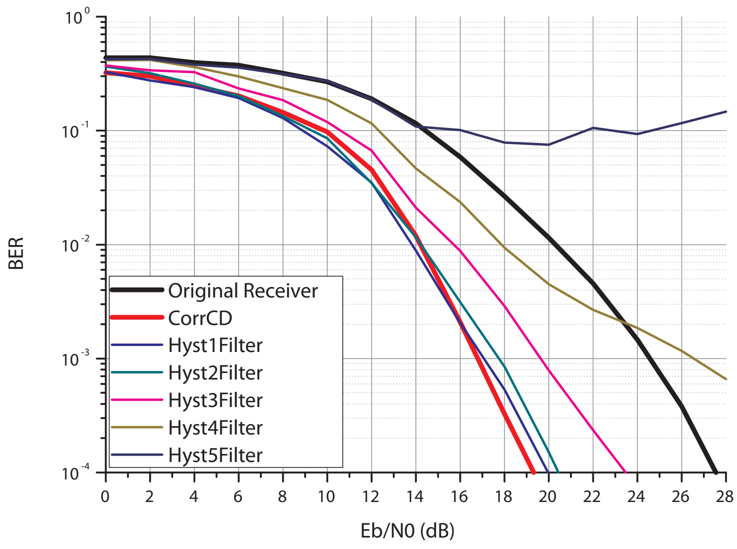


Figure C.3.: This figure shows the *BER* of the *Correlation Chip Detector* with *HystXFilter*.

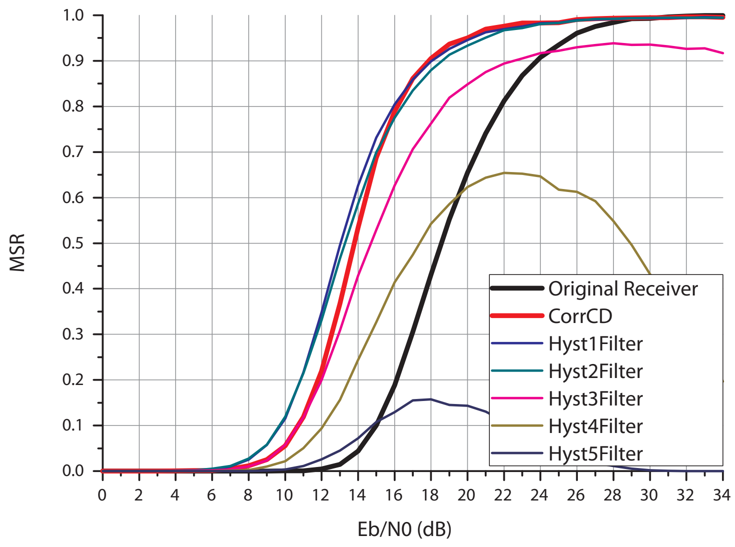


Figure C.4.: This figure shows the *MSR* of the *Correlation Chip Detector* with *HystXFilter*.

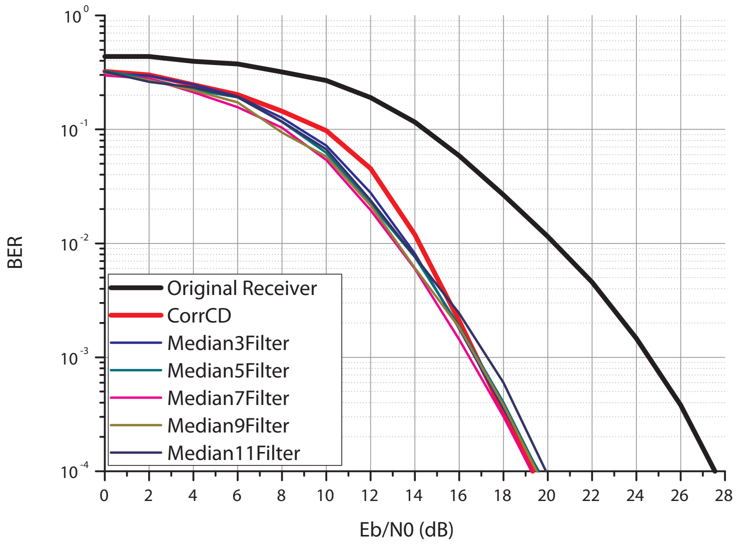


Figure C.5.: This figure shows the *BER* of the *Correlation Chip Detector* with *MedianXFilter*.

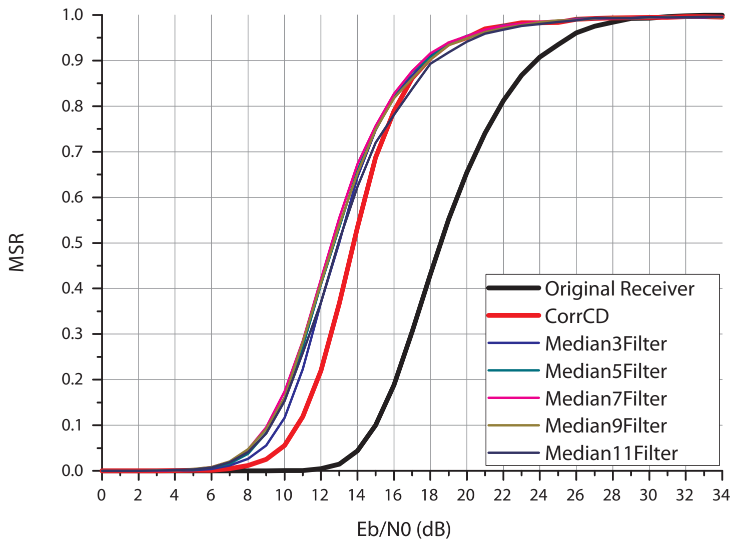


Figure C.6.: This figure shows the *MSR* of the *Correlation Chip Detector* with *MedianXFilter*.

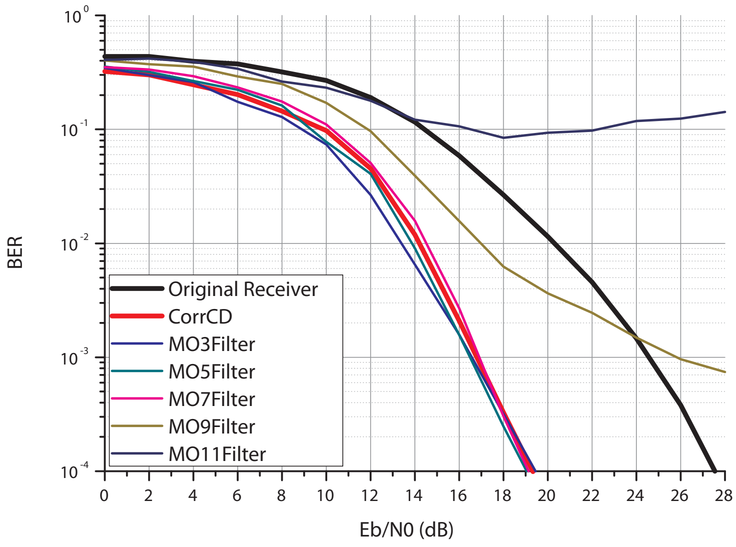


Figure C.7.: This figure shows the *BER* of the *Correlation Chip Detector* with *MOXFilter*.

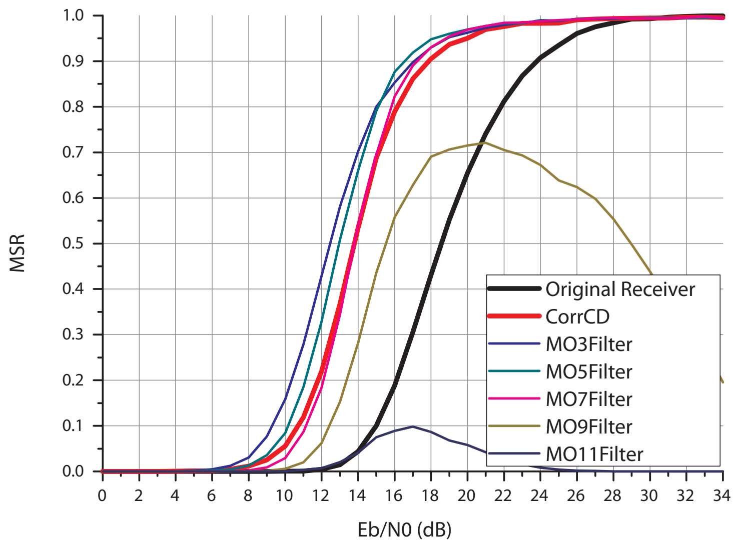


Figure C.8.: This figure shows the *MSR* of the *Correlation Chip Detector* with *MOXFilter*.

C.3. Correlation Manchester Decoder

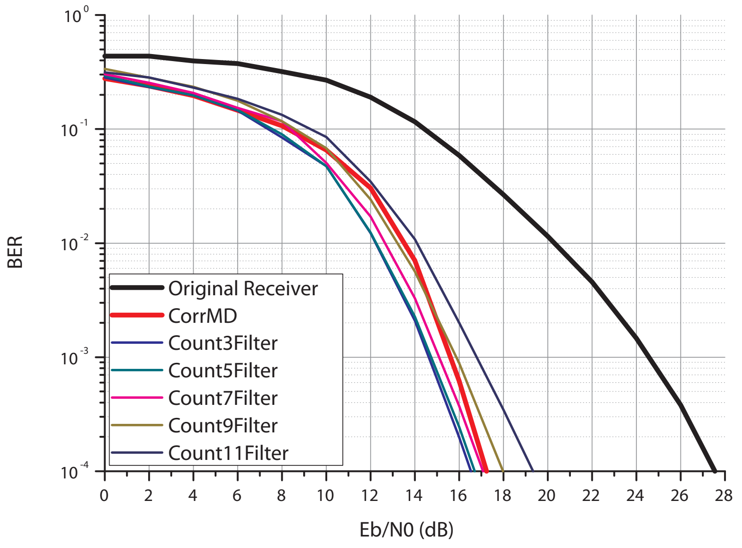


Figure C.9.: This figure shows the *BER* of the *Correlation Manchester Decoder* with *CountXFilter*.

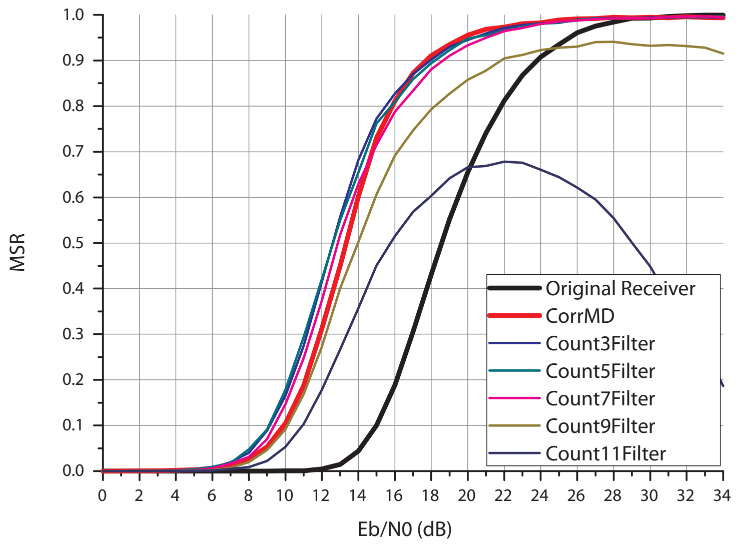


Figure C.10.: This figure shows the *MSR* of the *Correlation Manchester Decoder* with *CountXFilter*.

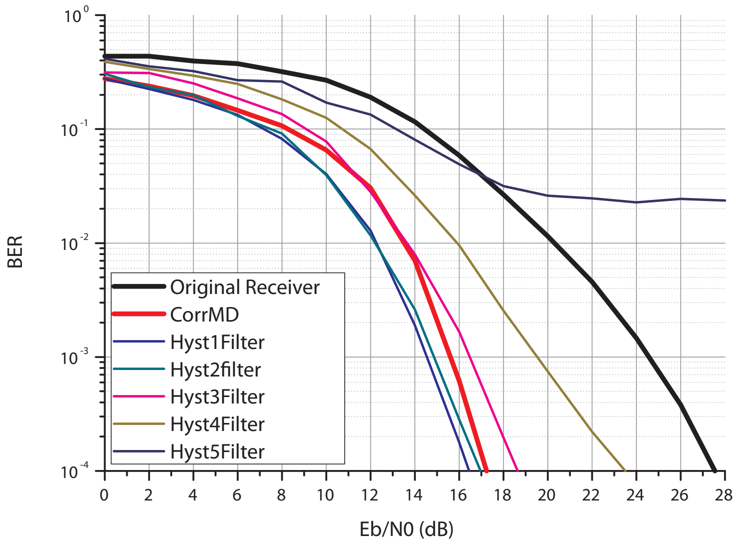


Figure C.11.: This figure shows the *BER* of the *Correlation Manchester Decoder* with *HystXFilter*.

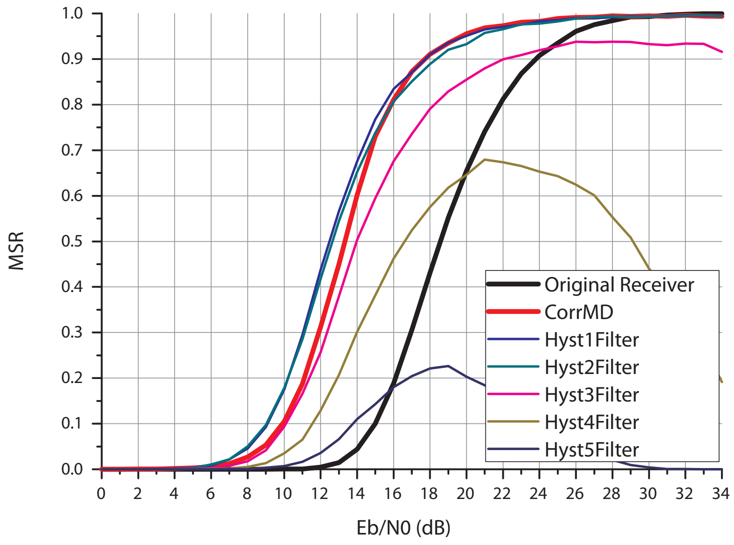


Figure C.12.: This figure shows the *MSR* of the *Correlation Manchester Decoder* with *HystXFilter*.

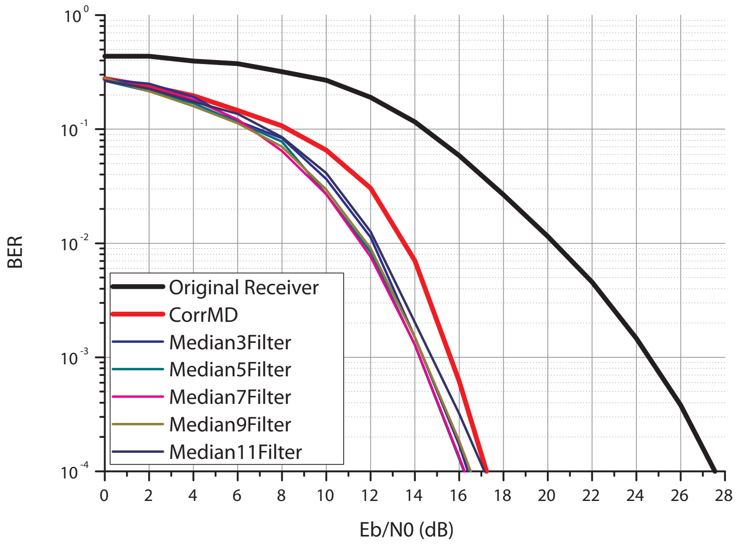


Figure C.13.: This figure shows the *BER* of the *Correlation Manchester Decoder* with *MedianXFilter*.

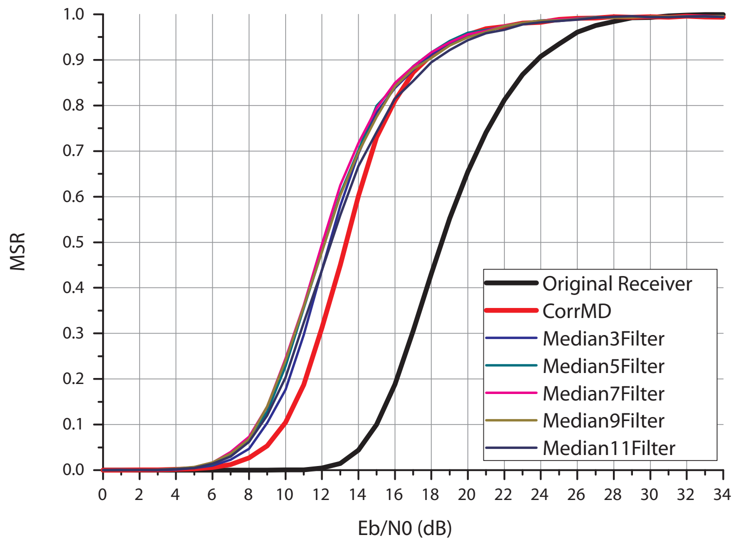


Figure C.14.: This figure shows the *MSR* of the *Correlation Manchester Decoder* with *MedianXFilter*.

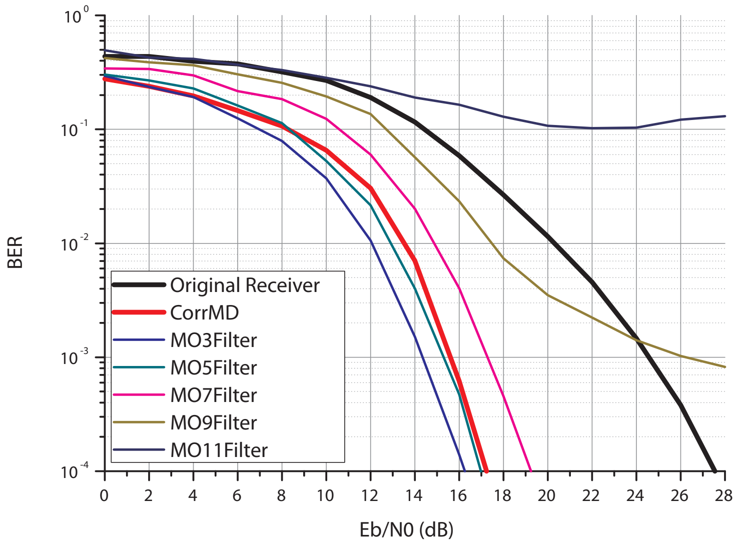


Figure C.15.: This figure shows the *BER* of the *Correlation Manchester Decoder* with *MOXFilter*.

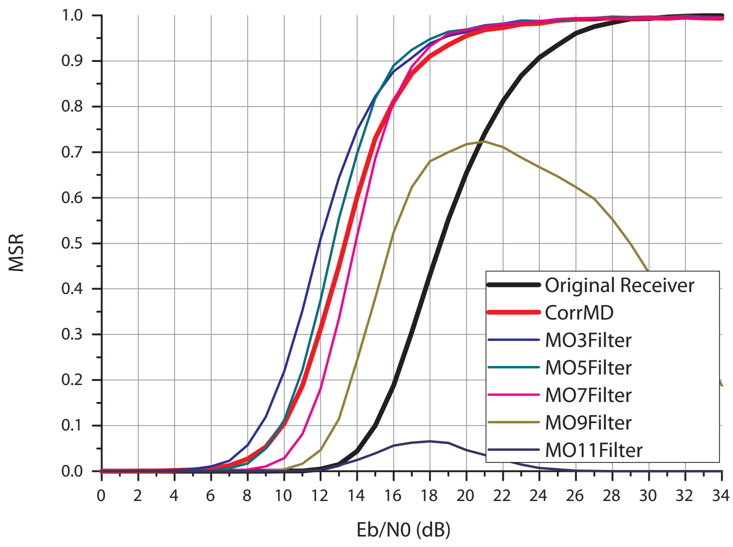


Figure C.16.: This figure shows the *MSR* of the *Correlation Manchester Decoder* with *MOXFilter*.

D. VHDL Models

In this appendix the *VHDL* models of the noise filters and the decoders are shown. The theoretical descriptions of the function and their design are given in chapter 3.

D.1. CountXFilter

Listing D.1: Listing of the *CountXFilter*

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.std_logic_unsigned.all;

entity CountXFilter is
    generic(maxcount: positive := 3);
    port(CLK: in bit;
         RESET: in bit;
         IND:   in bit;
         OUTD: out bit);
end entity CountXFilter;

architecture BEHAVE of CountXFilter is
    signal counter: natural;
    signal Q: bit;
begin

    CNT: process (CLK, RESET)
    begin
        if RESET = '1' then
            counter <= maxcount*1/2;
        elsif CLK = '1' and CLK'event then
            if IND = '1' then
                if counter < maxcount then
                    counter <= counter + 1;
                end if;
            else
                if counter > 0 then
                    counter <= counter - 1;
                end if;
            end if;
        end if;
    end process CNT;

    ST: process (CLK, RESET)
    begin
        if RESET = '1' then
            Q <= '0';
        elsif CLK = '1' and CLK'event then
            if counter > maxcount/2 then
                Q <= '1';
            else
                Q <= '0';
            end if;
        end if;
    end process ST;

    OUTD <= Q;

end architecture BEHAVE;
```

D.2. HystXFilter

Listing D.2: Listing of the *HystXFilter*

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.std_logic_unsigned.all;

entity HystXFilter is
    generic(statesize: positive := 2);
    port(CLK: in bit;
         RESET: in bit;
         IND:   in bit;
         OUTD: out bit);
end entity HystXFilter;

architecture BEHAVE of HystXFilter is
    type STATE_TYPE is (LOW, HIGH);
    signal state: STATE_TYPE;
    signal counter: natural;
    signal Q: bit;
begin

    STATECNT: process (CLK, RESET)
    begin
        if RESET = '1' then
            counter <= statesize*3/2;
        elsif CLK = '1' and CLK'event then

            if IND = '1' then
                if counter < (3*statesize) then
```

```
                    counter <= counter + 1;
                end if;
            else
                if counter > 1 then
                    counter <= counter − 1;
                end if;
            end if;
        end if;
    end process STATECNT;

    ST: process (CLK, RESET)
    begin
        if RESET = '1' then
            state <= LOW;
            Q <= '0';
        elsif CLK = '1' and CLK'event then
            if ((state = LOW) and (counter >= (2*statesize + 1))) then
                state <= HIGH;
                Q <= '1';
            elsif ((state = HIGH) and (counter <= statesize))then
                state <= LOW;
                Q <= '0';
            end if;
        end if;
    end process ST;

    OUTD <= Q;

end architecture BEHAVE;
```

D.3. MedianXFilter

Listing D.3: Listing of the *MedianXFilter*

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.std_logic_unsigned.all;

entity MedianXFilter is
    generic( filtersize : positive := 3);
    port(CLK: in bit;
         RESET: in bit;
         IND:   in bit;
         OUTD: out bit);
end entity MedianXFilter;

architecture BEHAVE of MedianXFilter is
    signal counter: natural;
    signal shiftr : bit_vector(0 to filtersize );
    signal inr: bit;
    signal outr: bit;
    signal Q: bit;
begin

    inr <= shiftr(0);
    outr <= shiftr( filtersize );

    SFR: process (CLK, RESET)
    begin
        if RESET = '1' then
            shiftr <= (others => '0');
        elsif CLK = '1' and CLK'event then
            shiftr <= IND & shiftr(0 to filtersize-1);
        end if;
    end process SFR;

    CNT: process (CLK, RESET)
    begin
        if RESET = '1' then
            counter <= 0;
        elsif CLK = '1' and CLK'event then
            if (inr = '1' and outr = '0') then
                if counter < filtersize then
                    counter <= counter + 1;
                end if;
            elsif (inr = '0' and outr = '1') then
                if counter > 0 then
                    counter <= counter - 1;
                end if;
            end if;
        end if;
    end process CNT;

    ST: process (CLK, RESET)
    begin
        if RESET = '1' then
            Q <= '0';
        elsif CLK = '1' and CLK'event then
            if counter <= filtersize/2 then
                Q <= '0';
            else
                Q <= '1';
            end if;
        end if;
    end process ST;

    OUTD <= Q;
```

```
end architecture BEHAVE;
```

D.4. MOXFilter

Listing D.4: Listing of the *MOXFilter*

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.std_logic_unsigned.all;

entity MOXFilter is
    generic(masksize: positive := 3);
    port(CLK: in bit;
         RESET: in bit;
         IND:   in bit;
         OUTD: out bit);
end entity MOXFilter;

architecture BEHAVE of MOXFilter is
    signal In2First: bit;
    signal First2Second: bit;
    signal Second2Third: bit;
    signal Third2Fourth: bit;
    signal Fourth2Out: bit;
    signal First: bit_vector(0 to masksize-1);
    signal Second: bit_vector(0 to masksize-1);
    signal Third: bit_vector(0 to masksize-1);
    signal Fourth: bit_vector(0 to masksize-1);
begin

    In2First <= IND;
    First2Second <= First(masksize-1);
    Second2Third <= Second(masksize-1);
    Third2Fourth <= Third(masksize-1);
    Fourth2OUT <= Fourth(masksize-1);

    P1: process(CLK,RESET)
    begin
        if RESET = '1' then
           First <= (others => '0');
           Second <= (others => '0');
           Third <= (others => '0');
           Fourth <= (others => '0');
        elsif CLK'event and CLK = '1' then
           if In2First = '0' then
               First <= (others => '0');
           else
```

```
                First  <= '1' & First(0 to masksize−2);
            end if;
            if First2Second = '1' then
                Second <= (others => '1');
            else
                Second <= '0' & Second(0 to masksize−2);
            end if;
            if Second2Third = '1' then
                Third <= (others => '1');
            else
                Third <= '0' & Third(0 to masksize−2);
            end if;
            if Third2Fourth = '0' then
                Fourth <= (others => '0');
            else
                Fourth <= '1' & Fourth(0 to masksize−2);
            end if;
        end if;
    end process P1;

    OUTD <= Fourth2Out;

end architecture BEHAVE;
```

D.5. CorrCD

Listing D.5: Listing of the *CorrCD*

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.std_logic_unsigned.all;

entity CorrCD is
    port(CLK:   in bit;
         RESET: in bit;
         IND:   in bit;
         CHIP:  out bit;
         STROBE: out bit);
end entity CorrCD;

architecture BEHAVE of CorrCD is

    signal sample_counter: std_logic_vector(3 downto 0);
    signal reset_sample_counter: boolean;
    signal set_sample_counter: boolean;
    signal down_sample_counter: boolean;
    signal chip_counter: std_logic_vector(4 downto 0);
    signal reset_chip_counter: boolean;
    signal flag_longer: boolean;
    signal reset_flag_longer: boolean;
    signal flag_shorter: boolean;
    signal flag_shorter_en: boolean;
    signal isEleven: boolean;
    signal isTen: boolean;
    signal IND_D0: bit;
    signal IND_D1: bit;
    signal CLK_INT: bit;
    signal flag_CLK: bit;
    signal CHIP_INT: bit;
    signal STROBE_INT: bit;
    signal STSHR: bit_vector(4 downto 0);

begin

    SYNC: process(CLK, RESET)
    begin
        if RESET = '1' then
            IND_D0 <= '0';
            IND_D1 <= '0';
            CHIP <= '0';
            CLK_INT <= '0';
        elsif CLK = '1' and CLK'event then
            IND_D0 <= IND;
            IND_D1 <= IND_D0;
            CHIP <= CHIP_INT;
            CLK_INT <= flag_CLK;
        end if;
    end process SYNC;

    Sample_CNT: process(CLK, RESET)
    begin
        if RESET = '1' then
            sample_counter <= "0000";
        elsif CLK = '1' and CLK'event then
            if reset_sample_counter = true then
                sample_counter <= "0001";
            elsif set_sample_counter = true then
                sample_counter <= "0010";
            elsif down_sample_counter = true then
                sample_counter <= sample_counter;
            else
                sample_counter <= sample_counter + 1;
            end if;
        end if;
    end process Sample_CNT;

    CHIP_CORRELATOR: process(CLK, RESET)
```

```
begin
    if RESET = '1' then
        chip_counter <= "10000";
    elsif CLK = '1' and CLK'event then
        if reset_chip_counter then
            if IND_D0 = '1' then
                chip_counter <= "10001";
            else
                chip_counter <= "01111";
            end if;
        else
            if IND_D0 = '1' then
                if chip_counter < "11011" then
                    chip_counter <= chip_counter + 1;
                end if;
            else
                if chip_counter > "00101" then
                    chip_counter <= chip_counter - 1;
                end if;
            end if;
        end if;
    end if;
end process CHIP_CORRELATOR;

Zero_Detector: process(CLK, RESET)
begin
    if RESET = '1' then
        flag_longer <= true;
    elsif CLK = '1' and CLK'event then
        if reset_flag_longer = true then
            flag_longer <= false;
        elsif chip_counter = "10000" then
            flag_longer <= not(flag_longer);
        end if;
    end if;
end process Zero_Detector;

STB_SHIFT: process(CLK, RESET)
begin
    if RESET = '1' then
        STSHR <= (others => '0');
    elsif CLK = '1' and CLK'event then
        STSHR <= STSHR(3 downto 0) & STROBE_INT;
        STROBE <= STSHR(4);
    end if;
end process STB_SHIFT;

isEleven <= (chip_counter = "11011") or (chip_counter = "00101");
isTen <= (chip_counter = "11010") or (chip_counter = "00110");
```

```
CONTROL: process(RESET,CLK)
begin
    if RESET = '1' then
        flag_shorter_en <= false;
        flag_shorter <= false;
        down_sample_counter <= false;
        reset_chip_counter <= false;
        reset_sample_counter <= false;
        set_sample_counter <= false;
        reset_flag_longer <= false;
        flag_CLK <= '0';
        STROBE_INT <= '0';
        CHIP_INT <= '0';
    elsif CLK = '0' and CLK'event then
        down_sample_counter <= false;
        reset_chip_counter <= false;
        reset_sample_counter <= false;
        reset_flag_longer <= false;
        set_sample_counter <= false;
        STROBE_INT <= '0';
        if sample_counter = "1011" then
            reset_flag_longer <= true;
            if ((flag_longer = true) and not(reset_flag_longer)) then
                down_sample_counter <= true;
                flag_shorter_en <= false;
            elsif (not(flag_longer) and (flag_shorter_en) and not(isEleven)) then
                if (flag_shorter and isTen) then
                    reset_sample_counter <= true;
                    flag_shorter_en <= true;
                    flag_shorter <= false;
                    flag_CLK <= not flag_CLK;
                    if chip_counter >= "10000" then
                        CHIP_INT <= '1';
                    else
                        CHIP_INT <= '0';
                    end if;
                    reset_chip_counter <= true;
                    STROBE_INT <= '1';
                else
                    set_sample_counter <= true;
                    flag_shorter_en <= true;
                    flag_shorter <= true;
                    flag_CLK <= not flag_CLK;
                    if chip_counter >= "10000" then
                        CHIP_INT <= '1';
                    else
                        CHIP_INT <= '0';
                    end if;
                    reset_chip_counter <= true;
                    STROBE_INT <= '1';
                end if;
            else
                reset_sample_counter <= true;
                flag_shorter_en <= true;
                flag_CLK <= not flag_CLK;
                if chip_counter >= "10000" then
                    CHIP_INT <= '1';
                else
                    CHIP_INT <= '0';
                end if;
                reset_chip_counter <= true;
                STROBE_INT <= '1';
            end if;
        end if;
    end if;
end process CONTROL;

end architecture BEHAVE;
```

D.6. CorrMD

Listing D.6: Listing of the *CorrMD*

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.std_logic_unsigned.all;

entity CorrMD is
    port(CLK:   in bit;
         RESET: in bit;
         IND:   in bit;
         SYNC:  in bit;
         CHIPS:  out bit;
         STROBE: out bit;
         BITS:   out bit);
end entity CorrMD;

architecture BEHAVE of CorrMD is

    signal sample_counter: std_logic_vector(3 downto 0);
    signal reset_sample_counter: boolean;
    signal set_sample_counter: boolean;
    signal down_sample_counter: boolean;
    signal chip_counter: std_logic_vector(4 downto 0);
    signal reset_chip_counter: boolean;
    signal flag_longer: boolean;
    signal reset_flag_longer: boolean;
    signal flag_shorter: boolean;
    signal flag_shorter_en: boolean;
    signal isEleven: boolean;
    signal isTen: boolean;
    signal IND_D0: bit;
    signal IND_D1: bit;
    signal CLK_INT: bit;
    signal flag_CLK: bit;
    signal CHIPS_INT: bit;
    signal STROBE_INT: bit;
    signal STSHR: bit_vector(4 downto 0);
    signal bit_counter: std_logic_vector(5 downto 0);
    signal reset_bit_counter: boolean;
    signal start_bit_counter: bit;
    signal lohichip: bit;
    signal toggle_bit_xor: boolean;
    signal BITS_INT: bit;
    signal lohichip_start: bit;
    signal decode_start: bit;
    signal run_bit_counter: bit;

begin

    SYNC_P: process(CLK, RESET)
    begin
        if RESET = '1' then
            IND_D0 <= '0';
            IND_D1 <= '0';
            CHIPS <= '0';
            CLK_INT <= '0';
        elsif CLK = '1' and CLK'event then
            IND_D0 <= IND;
            IND_D1 <= IND_D0;
            CHIPS <= CHIPS_INT;
            BITS <= (BITS_INT xor lohichip_start);
            CLK_INT <= flag_CLK;
        end if;
    end process SYNC_P;

    Sample_CNT: process(CLK, RESET)
    begin
        if RESET = '1' then
            sample_counter <= "0000";
        elsif CLK = '1' and CLK'event then
            if reset_sample_counter = true then
                sample_counter <= "0001";
            elsif set_sample_counter = true then
                sample_counter <= "0010";
            elsif down_sample_counter = true then
                sample_counter <= sample_counter;
            else
                sample_counter <= sample_counter + 1;
            end if;
        end if;
    end process Sample_CNT;

    Chip_Correlator: process(CLK, RESET)
    begin
        if RESET = '1' then
            chip_counter <= "10000";
        elsif CLK = '1' and CLK'event then
            if reset_chip_counter then
                if IND_D0 = '1' then
                    chip_counter <= "10001";
                else
                    chip_counter <= "01111";
                end if;
            else
                if IND_D0 = '1' then
                    if chip_counter < "11011" then
                        chip_counter <= chip_counter + 1;
                    end if;
                else
                    if chip_counter > "00101" then
                        chip_counter <= chip_counter - 1;
                    end if;
                end if;
            end if;
        end if;
    end process Chip_Correlator;


    Bit_Corelator: process(CLK, RESET)
    begin
        if RESET = '1' then
            bit_counter <= "100000";
        elsif CLK = '1' and CLK'event then
            if (reset_bit_counter or (start_bit_counter = '1')) then
                if ((not (IND_D0 xor lohichip)) = '0') then
                    bit_counter <= "100001";
                else
                    bit_counter <= "011111";
                end if;
            else
                if ((not (IND_D0 xor lohichip)) = '1') then
                    if bit_counter < "110110" then
                        bit_counter <= bit_counter + 1;
                    end if;
                else
                    if bit_counter > "001010" then
                        bit_counter <= bit_counter - 1;
                    end if;
                end if;
            end if;

        end if;
    end process Bit_Corelator;

    Bit_XOR: process (CLK, RESET)
    begin
        if RESET = '1' then
            lohichip <= '0';
        elsif CLK = '1'and CLK'event then
            if toggle_bit_xor = true then
                lohichip <= not lohichip;
            end if;
        end if;
    end process Bit_XOR;

    decode_start <= SYNC;

    SaveLoHi: process (CLK, RESET)
    begin
        if RESET = '1' then
            lohichip_start <= '0';
            run_bit_counter <= '0';
        elsif CLK = '1' and CLK'event then
            if decode_start = '1' then
                lohichip_start <= lohichip;
                run_bit_counter <= '1';
            end if;
        end if;
    end process SaveLoHi;

    Zero_Detector: process(CLK, RESET)
    begin
        if RESET = '1' then
            flag_longer <= true;
        elsif CLK = '1' and CLK'event then

            if reset_flag_longer = true then
                flag_longer <= false;
            elsif chip_counter = "10000" then
                flag_longer <= not(flag_longer);
            end if;
        end if;
```

```
end process Zero_Detector;

STB_SHIFT: process(CLK, RESET)
begin
    if RESET = '1' then
        STSHR <= (others => '0');
    elsif CLK = '1' and CLK'event then
        STSHR <= STSHR(3 downto 0) & STROBE_INT;
        STROBE <= STSHR(4);
    end if;
end process STB_SHIFT;

isEleven <= (chip_counter = "11011") or (chip_counter = "00101");
isTen <= (chip_counter = "11010") or (chip_counter = "00110");

CONTROL: process(RESET,CLK)
begin
    if RESET = '1' then
        flag_shorter_en <= false;
        flag_shorter <= false;
        down_sample_counter <= false;
        reset_chip_counter <= false;
        reset_sample_counter <= false;
        set_sample_counter <= false;
        reset_flag_longer <= false;
        flag_CLK <= '0';
        STROBE_INT <= '0';
        CHIPS_INT <= '0';
        BITS_INT <= '0';
        toggle_bit_xor <= false;
        reset_bit_counter <= false;
    elsif CLK = '0' and CLK'event then
        down_sample_counter <= false;
        reset_chip_counter <= false;
        reset_sample_counter <= false;
        reset_flag_longer <= false;
        set_sample_counter <= false;
        STROBE_INT <= '0';
        reset_bit_counter <= false;
        toggle_bit_xor <= false;
        if sample_counter = "1011" then
            reset_flag_longer <= true;
            if ((flag_longer = true) and not(reset_flag_longer)) then
                down_sample_counter <= true;
                flag_shorter_en <= false;
            elsif (not(flag_longer) and (flag_shorter_en) and not(isEleven)) then
                if (flag_shorter and isTen) then
                    reset_sample_counter <= true;
                    flag_shorter_en <= true;
                    flag_shorter <= false;
                    flag_CLK <= not flag_CLK;
                    if chip_counter >= "10000" then
                        CHIPS_INT <= '1';
                    else
                        CHIPS_INT <= '0';
                    end if;
                    if ((lohichip = (not lohichip_start))
                    and (run_bit_counter = '1')) then
                        if bit_counter >= "100000" then
                            BITS_INT <= '1';
                        else
                            BITS_INT <= '0';
                        end if;
                        reset_bit_counter <= true;
                    end if;
                    if run_bit_counter = '0' then
                        reset_bit_counter <= true;
                    end if;
                    reset_chip_counter <= true;
                    STROBE_INT <= '1';
                    toggle_bit_xor <= true;
                else
                    set_sample_counter <= true;
                    flag_shorter_en <= true;
                    flag_shorter <= true;
                    flag_CLK <= not flag_CLK;
                    if chip_counter >= "10000" then
                        CHIPS_INT <= '1';
                    else
                        CHIPS_INT <= '0';
                    end if;
                    if ((lohichip = (not lohichip_start))
                    and (run_bit_counter = '1')) then
                        if bit_counter >= "100000" then
                            BITS_INT <= '1';
                        else
                            BITS_INT <= '0';
                        end if;
                        reset_bit_counter <= true;
                    end if;
                    if run_bit_counter = '0' then
                        reset_bit_counter <= true;
                    end if;
                    reset_chip_counter <= true;
                    STROBE_INT <= '1';
                    toggle_bit_xor <= true;
                end if;
            else
```

```
                    reset_sample_counter <= true;
                    flag_shorter_en <= true;
                    flag_CLK <= not flag_CLK;
                    if chip_counter >= "10000" then
                        CHIPS_INT <= '1';
                    else
                        CHIPS_INT <= '0';
                    end if;
                    if ((lohichip = (not lohichip_start))
                    and (run_bit_counter = '1')) then
                        if bit_counter >= "100000" then
                            BITS_INT <= '1';
                        else
                            BITS_INT <= '0';
                        end if;
                        reset_bit_counter <= true;
                    end if;
                    if run_bit_counter = '0' then
                        reset_bit_counter <= true;
                    end if;
                    reset_chip_counter <= true;
                    STROBE_INT <= '1';
                    toggle_bit_xor <= true;
                end if;
            end if;
        end if;
    end process CONTROL;

end architecture BEHAVE;
```

Nomenclature

$\nabla\times$	Curl Operator
Δ	Laplace Operator
$\nabla\cdot$	Divergence Operator
∇	Gradient Operator
λ	Wave Length
Ω	Ohm
ω_0	Carrier Frequency
ϕ	Offset Phase
π	Circular Constant
σ_0	Standard Deviation of the Noise
A	Amplitude
B	Band Width
C	Capacitance
$c[n]$	Counter Value
D	Data Rate
f	Frequency
$H(s)$	Transfer Function
i	Current
$i_T(t)$	Current of a Transistor Current Source
$I_T\left(s\right)$	Current of Transistor Current Source in Laplace Domain
j	Imaginary Unit
K	Gain Factor
k	Coupling Factor

L	Inductance
M	Mutual Inductance
N_{FF}	Number of Flip-Flops
R	Ohmic Resistance
$s[n]$	Actual State
u	Voltage
$U(s)$	Voltage in Laplace Domain
V_{ATTN}	Attenuation Control Voltage
V_{bs}	Bulk-Source Voltage
V_{ds}	Drain-Source Voltage
V_{gs}	Gate-Source Voltage
$V_{Noise,rms}$	Root Mean Square Voltage of Noise
$x[n]$	Actual Input Sample
$y[n]$	Actual Output Sample
Z	Impedance
AFE	Analog Front End
AGC	Active Gain Control
ASK	Amplitude Shift Keying
BER	Bit Error Rate
DBP	Digital Baseband Processor
FSM	Finite State Machine
LNA	Low Noise Amplifier
MSR	Message Success Rate
$RSSI$	Received Signal Strength Indicator
$VHDL$	VHSIC Hardware Description Language
XOR	Binary XOR Operation
ABS	Anti-Lock Breaking System

ASK	Amplitude Shift Keying
baud	Baud
CO_2	Carbon Dioxide
FSK	Frequency Shift Keying
H	Henry
HDL	Hardware Description Language
Hz	Hertz
LF	Low Frequency
m	Meter
MOS	Metal Oxide Semiconductor
OOK	On-Off-Keying
OTA	Operational Transconductance Amplifier
T	Time
TPMS	Tire Pressure Monitoring System
V	Volt

List of Figures

List of Tables

Listings

Bibliography

[1] B. Sklar, *Digital Communications*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, Inc., 2001.

[2] C. C. Enz, F. Krummenacher, and E. A. Vittoz, "An analytical transistor model valid in all regions of operation and dedicated to low-voltage and low-current applications," *Analog Integrated Circuits and Signal Processing*, vol. 8, pp. 83–114, 1995.

[3] H. Shichman and D. A. Hodges, "Modeling and simulation of insulated-gate field-effect transistor switching circuits," *IEEE Journal of Solid State Circuits*, 1968.

[4] B. Razavi, *Design of Analog CMOS Integrated Circuits.* McGraw-Hill, 2001.

[5] K. Hoffmann, *Systemintegration.* München,: Oldenburg Wissenschaftsverlag GmbH, 2003.

[6] M. Bucher, C. Lallement, C. Enz, and F. Krummenacher, "Accurate mos modelling for analog circuit simulation using the ekv model," *Circuits and Systems*, vol. 4, pp. 703–706, 1996.

[7] M. Bucher, C. Lallement, C. Enz, F. Theodoloz, and F. Krummenacher, "The epfl-ekv mosfet model equations for simulation," Electronic Laboratories, Swiss Federal Institute of Technology (EPFL), Tech. Rep., March 1999.

[8] W. Demtröder, *Experimentalphysik 2*, 3rd ed. Berlin, Heidelberg: Springer-Verlag, 2004.

[9] "Compact power packs," *EPCOS Components*, vol. 1, 2008.

[10] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, Inc., 1999.

[11] P. Z. Peebles, *Digital Communication Systems.* Prentice Hall, Inc., 1987.

[12] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits*, 2nd ed. Pearson Education International, 2003.

[13] I. N. Bronstein and K. A. Semendjaew, *Taschenbuch der Mathematik*, 6th ed. Frankfurt am Main: Wissenschaftlicher Verlag Harri Deutsch GmbH, 2005.

[14] J. Serra, *Image Analysis and Mathematical Morphology.* Orlando: Academic Press, Inc., 1982.

[15] K. Küpfmüller, W. Mathis, and A. Reibiger, *Theoretische Elektrotechnik*, 16th ed. Berlin, Heidelberg: Springer-Verlag, 2005.

[16] E. Hering, R. Martin, and M. Stohrer, *Physik für Ingenieure*, 8th ed. Berlin, Heidelberg: Springer-Verlag, 2002.