Sandra Wilfling, BSc.

Versatile DSP Architecture for a Radio Receiver

Master Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's degree program:
Information and Computer Engineering

submitted to

Graz University of Technology

Supervisors Ass.Prof. Dipl.-Ing. Dr.techn. Peter Söser
Dipl.-Ing. Dr.techn. Ulrich Mühlmann

Institute of Electronics
Head: Univ.-Prof. Dipl.-Ing. Dr.techn. Bernd Deutschmann

Graz, September 2020

Affidavit

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 indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Abstract

Radio Frequency Identification Technology (RFID) is widely used in our daily lives, such as in electronic payment systems or product identification. With the growing field of applications for RFID technology, the development of RFID receivers is becoming more and more complex.

A classical RFID receiver Application-specific Integrated Circuit (ASIC) design has the downside of very long development cycles. In addition, the non-versatile architecture does not allow fixes and architectural updates in a very late product development stage. By the idea of using a versatile Digital Signal Processor (DSP) architecture, these problems can be solved. Firstly, only one single architecture needs to be taped-out and secondly, the flexible software-oriented structure allows to do modifications even in the design-in phase or can be subject to later firmware updates on the customer side.

In order to propose a DSP architecture for an existing RFID receiver, the digital core of the receiver system is analyzed. Firstly, the critical parts of the receiver are identified. For these parts, optimizations are proposed and evaluated. From this point, the most promising optimizations are implemented on a DSP hardware and analyzed in terms of performance. Finally, the results of the performance analysis are presented.

Keywords: RFID Technology, RF-Signal Processing, Matched Filter, Matlab/Simulink, DSP Development, Texas Instruments (TI) DSK6713

Kurzfassung

Radio Frequency Identification Technology (RFID) ist eine weit verbreitete Technologie, die ihre Anwendungen unter anderem in elektronischen Zahlungssystemen und in der Produktidentifikation findet. Mit dem stetig wachsenden Anwendungsbereich für RFID-Produkte steigern sich auch die Anforderungen in der Entwicklung von RFID-Empfängern.

Die langen Entwicklungszeiten von Application-specific Integrated Circuits (ASICs) schränken die Weiterentwicklung wesentlich ein. Zusätzlich erlaubt die Architektur der ASICs keine Reparaturen und Modifikationen zu späteren Zeitpunkten. Durch eine Digital Signal Processor (DSP)-Architektur können diese Probleme gelöst werden. Eine flexible softwareorientierte Struktur erlaubt Modifikationen und Firmware-Updates auf der Kundenseite. Zusätzlich können verschiedene Anwendungsgebiete über Softwareänderungen abgedeckt werden.

Diese Arbeit behandelt den Entwurf einer DSP-Architektur für einen bereits existierenden RFID-Empfänger. Darin wird der Digitalteil des Empfängers analysiert. Im ersten Schritt werden die kritischen Teile des Systems identifiziert. Für diese Komponenten werden verschiedene Optimierungen vorgeschlagen und evaluiert. Die vielversprechendsten Optimierungen werden auf einer DSP-Hardware implementiert und auf ihre Performance getestet. Im Abschluss werden die Ergebnisse der Performanceanalyse präsentiert.

Stichwörter: RFID Technologie, RF Signalverarbeitung, Matched Filtering, Matlab/Simulink, DSP Entwicklung, TI DSK6713

Contents

List of Figures

1 Introduction

Wireless communication technology has become more and more popular during the last few years. Nowadays, wireless systems are widely used in our daily lives, for instance, in smartphones, computer networks or smart cards.

The various applications of wireless communication have brought up different types of technologies, which are each suited for a certain purpose and range. In long range communication, the state-of-the-art consists of 3G and 4G, with the new 5G technology coming. Middle or short range communication, which is mainly indoors, relies on technologies such as Wi-Fi and Bluetooth. In close-range communication, which involves ranges of centimeters, the standard is RFID/Near-Field Communication (NFC) technology.

The focus of this work is set on RFID systems and the architecture of RFID receivers. In general, the RFID technology is designed for short range wireless communication. Data transfer is established through a magnetic field, which has a range of centimeters. Therefore, RFID is widely used for applications with close contact between the communication partners, for instance, in the scanning of smart cards or in contactless payment.

In most cases, RFID communication systems are implemented by Integrated Circuits (ICs). The main producers of RFID systems are microelectronic companies, such as Infineon Technologies, NXP Semiconductors or ST Microelectronics. Examples for RFID readers are the PN512 [13] from NXP and the ST Microelectronics ST25R3912 [16]. These chips both implement RFID frontends for proximity coupling at 13.56 MHz.

1.1 Motivation

In the development of microchips, crucial factors are the chip area and the power consumption. These two factors directly influence the production costs of the chip. Most systems aim to minimize them by using various optimizations. In NFC technology, most transmitters and receivers are implemented as mixed-signal ICs, which consist of an analog and a digital part. While the analog part covers the basic processing steps, the digital core implements signal processing algorithms, control logic and data transfer protocols. An overview of a digital RFID receiver core is depicted in Figure 1.1.

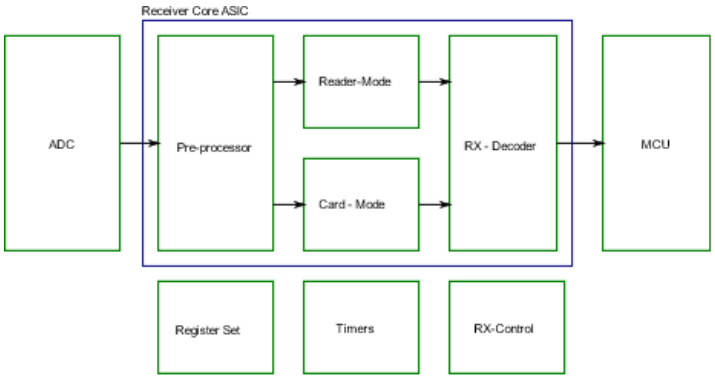


Figure 1.1: Overview of an RFID receiver digital core

In the design of digital cores, several steps can be taken to optimize in terms of performance. Optimization is crucial especially in RFID receivers, which implement complex signal processing algorithms. An approach is to add a microcontroller core to the system, which can implement some of the most power-consuming algorithms of the system. For this purpose, a processor which is designed for signal processing applications, a so-called DSP, can be used.

The aim of this work is to evaluate a digital RFID receiver core and to optimize the receiver in terms of computational performance. In addition, part of the signal processing algorithms should be implemented on a DSP. Finally, the performance of the DSP implementation should be analyzed.

2 Related Work

This chapter covers the background of the thesis, including an introduction to RFID technology and the signal processing of RFID systems. These sections relate to the books 'RFID Handbook' [6] and 'Communication Systems Engineering' [14]. In addition, a special emphasis is put into the design of digital filters and also on the platforms that are used for implementing digital signal processing algorithms.

2.1 RFID Technology

RFID is a technology that makes it possible to create a contactless communication within a close range. In the most basic version, RFID systems consist of two elements, a reader and a transponder. An overview of an RFID system is depicted in Figure 2.1.

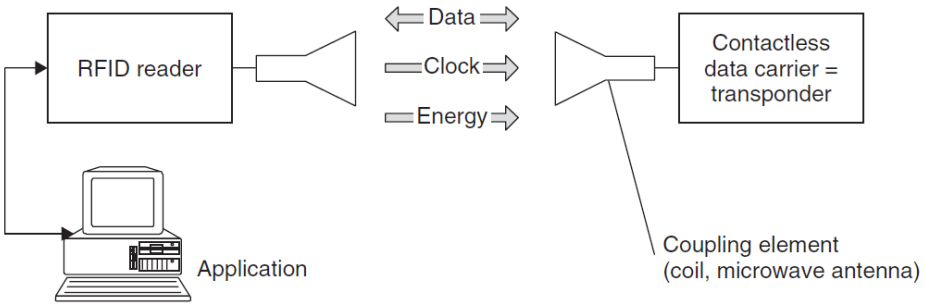


Figure 2.1: Overview of an RFID system, cf. 'RFID Handbook' Figure 1.6 [6]

The active element in the communication is the RFID reader. The reader transmits requests for identification to the transponder. Then, the transponder answers by transmitting data.

In the first RFID systems, transponders were mainly used to transmit a Unique Identifier (UID) in order to identify a product. Nowadays, however, RFID transponders can implement more advanced communication methods and protocols.

2.1.1 Characteristics of RFID Systems

The RFID technology offers many options for implementing a contactless communication system. Various manufacturers, such as ST Microelectronics, NXP Semiconductors, or Infineon Technologies have been creating RFID systems with different features. In order to characterize these systems, several parameters can be used.

Carrier Frequency and Coupling Type

The main characteristic of RFID systems is the carrier or operating frequency. The frequency bands can be classified into Low Frequency (LF) at 30 to 300 kHz, High Frequency (HF) which ranges from 3 to 30 MHz, and Ultra High Frequency (UHF) at 300 MHz to 3 GHz and microwave systems above 3 GHz. The carrier frequency is also related to the type of coupling that is used for communication. While HF and LF systems use the magnetic field of the antenna (inductive coupling), UHF and microwave systems use backscattering from electromagnetic waves.
The operating frequency bands for an RFID system must be licensed in the region of use. Therefore, most systems operate at a few standard frequency bands which are available in many countries. For instance, in the EU, the mainly used bands are in the range of 100-135 kHz for LF bands, in the HF bands the carrier frequency of 13.56 MHz and in UHF range at 865 to 868 MHz.
The system that is used in the thesis is specified for the 13.56 MHz HF band, so a special focus is set on the technologies that use this frequency band.

Coupling Range

Another important characteristic of RF systems is the coupling range. The coupling range defines the maximum distance between two communication partners where a communication is still possible. The range is influenced by many factors, for instance by the type of coupling, which can be inductive or backscattering.
The main classes that define the coupling range are close-coupling ($<$ 1cm), proximity coupling (1cm-1.5m) or vicinity coupling.

Operating Mode

The operating mode describes the power supply of RFID tags in inductive coupling systems. Since inductive coupling systems use a magnetic field to create a communication channel, the energy from the magnetic field can be used to act as the power supply for the tag.

2.1.2 ISO Standards and NFC Definitions

RFID technology is standardized by various standards, which define the characteristics of data transfer from physical layer upwards. The standards define physical coupling characteristics, data encodings and transfer protocols of RF systems. In the HF band there are various standards defined by the International Standardisation Organisation/International Electrotechnical Commission (ISO), such as ISO 10563 for close coupling systems, ISO 14443 [9] for proximity coupling and ISO 15693 for vicinity coupling. Another widely used set of standards is the Japanese Industrial Standards (JIS), from which for instance JIS X 6319-4 (FeliCa) is widely used.

While these standards cover the low level protocols (physical layer, air interface, anticollision and basic data transfer), the ISO 18092-NFCIP-1 standard defines NFC communication. The NFC standard relies on the RFID standards for the definition of the physical interface.

In addition to the ISO standards the NFC forum has defined five Tag Types. These Tag Types also include a specification for higher level data transfer

protocols, tag memory size, etc. The Tag Types specify certain data rates from the ISO standards for use, as depicted in Figure 2.2.

NFC Forum Classification	NFC-A		NFC-B	NFC-F	NFC-V
Tag Type	Type 1	Type 2	Type 3	Type 4	Type 5
ISO Standard	ISO 14443-A		ISO 14443-B	JIS 6319-4	ISO 15693
Data Rate	106 kBit/s		212, 424 kBit/s	106,212,424 kBit/s	26.48 kBit/s
Memory	< 2kB		< 2kB	32 kB	64 kB

Figure 2.2: Overview of Tag Types specifications and ISO standards, according to [20].

2.2 RF Signal Processing

In digital communications, the step of signal conditioning has become more and more important. New application fields, such as mobile radio systems, require NFC devices to operate in the vicinity of various other communication systems with different technologies. Hence, the accurate processing of transmitted and received signals has become a critical factor in digital communication systems. In recent technology, RFID and NFC strongly rely on digital signal processing techniques in their transmitters and receivers. This section is going to give an overview of the techniques applied in state-of-the-art NFC.

2.2.1 Modulation/Demodulation

The main step of RF transmission is modulation, which transfers the signal from the baseband into the required frequency band. The modulation is done by multiplying the baseband signal with a high-frequency carrier signal, which is usually implemented as a mixer. The multiplication in the time domain is represented by a convolution in the frequency domain, therefore shifting the spectrum of the baseband signal to the carrier frequency.

The main modulation techniques are amplitude modulation, frequency modulation and phase modulation. This section references [4] and [6].

In NFC systems, some modulation techniques that are widely used are Amplitude Shift Keying (ASK), On-Off Keying (OOK) and Binary Phase Shift Keying (BPSK). In most of the NFC modulation techniques, the carrier wave is not modulated directly, instead one or two subcarriers are used. The subcarriers operate at a fraction of the carrier frequency and are modulated by ASK or BPSK modulation. The modulated subcarrier is then multiplied with the carrier wave, as depicted in Figure 2.3.

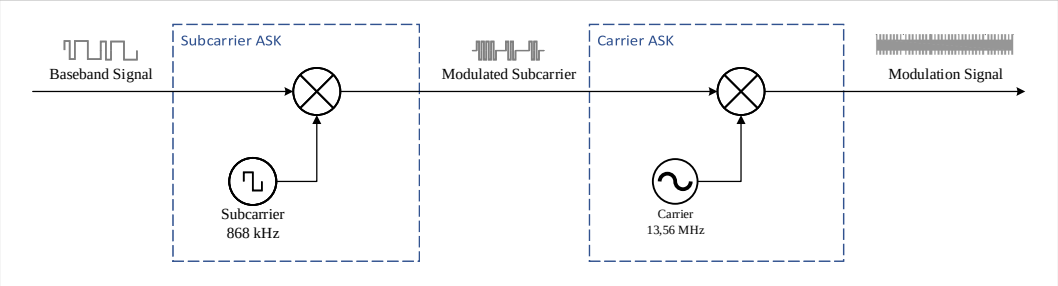


Figure 2.3: Schematic overview of an ASK modulation with subcarrier, cf. Finkenzeller [6], Fig. 6.12

On the receiver side, the signal must be transferred back into baseband domain by demodulation. This step, however, is way more complex than the modulation. While the basic transfer to baseband frequency can be done by a multiplication, it must be considered that a received signal can be affected by distortions, interference and noise during the transfer through the communication channel. In order to recover the information from a distorted signal, more complex techniques must be employed. This is particularly important of RFID readers, which may receive weak signals from the corresponding transponder. In addition, some applications, e.g. passive transponders, require the recovery of the carrier as clock for the digital part of their system.

The modulation systems used in NFC technology require the demodulation of ASK or BPSK signals. For ASK, the simplest demodulation technique would be an envelope detector, which can be realised by a rectifier and a

lowpass filter. However, this detector only recovers the signal amplitude. For signals affected by phase distortion through the communication channel, the detected amplitude may decrease and even cross zero, meaning that information about the signal is lost. If the signal is affected by phase distortions, a coherent demodulation such as the In-Phase-Quadrature (I-Q) demodulation system is necessary.

I-Q Demodulation

The I-Q demodulation is based on the multiplication of the input signal with two different carrier signals, one that is supposed to be in phase, and the other one with a phase difference of 90 degrees. The resulting signals can then be interpreted as the real (in-phase) and the imaginary value (quadrature phase) of the input signal. An overview of this system is depicted in Figure 2.4.

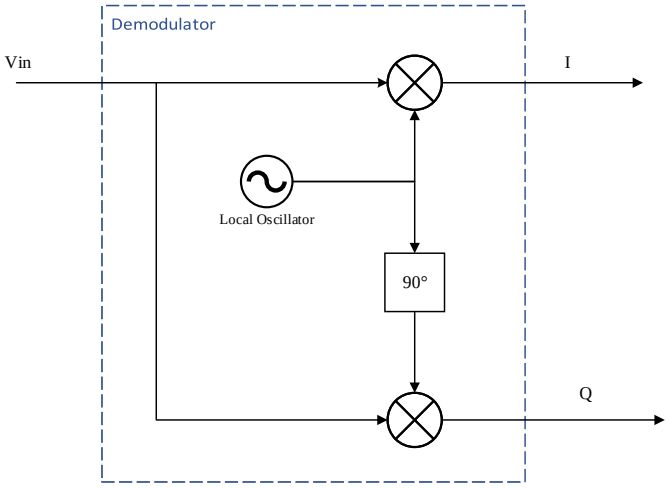
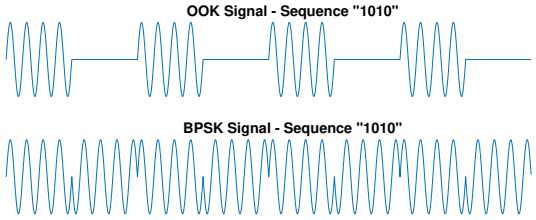


Figure 2.4: Schematic overview of an I-Q demodulator

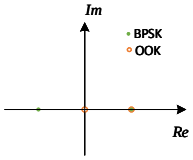
After the mixer stage, the I and Q signals can be processed by lowpass filters, Direct Current (DC) removal, gain compensation etc. The next stages

in the Radio Frequency (RF) processing chain are symbol detection and synchronization.

The I-Q demodulation retains the phase information of the input signal. Signals that have been affected by phase distortions can therefore be recovered without losing most of the information. The I and Q values can then be depicted in a constellation diagram, which shows the different symbols used in the modulation and their amplitude and phase values. Figure 2.5 depicts the modulation signals and the corresponding constellation diagram for BPSK and OOK signals.



(a) Modulated signals of BPSK and OOK



(b) Constellation Diagram: OOK and BPSK

Figure 2.5: Signals and constellation diagram of the modulation types OOK and BPSK

Modern RF systems use mixed-signal modulation/demodulation systems. The implementation of an I-Q demodulator requires an internal oscillator to generate the carrier frequency input for the mixer. For creating these local oscillators, there are multiple possibilities. Some systems use Phase-Locked Loops (PLLs), which are usually implemented in analog design with digital control logic. Another approach is to create the local oscillator from direct digital synthesis.

2.2.2 Symbol Detection - SNR Maximization

In the RF signal processing chain, the next step after the demodulation/mixer stage is the symbol detection. After the carrier wave has been mixed, only the baseband signal should remain. However, noise and distortions are added to

the signal while passing through the communication channel. These factors affect the signal so that a simple detection based on the original decision boundaries may not be sufficient.

AWGN

One of the main factors that influence the signal through the communication channel is zero-mean Additive White Gaussian Noise (AWGN). The AWGN is defined as:

$$y[n] = x[n] + v[n] \tag{2.1}$$
$$v[n] \sim \mathcal{N}(0, \sigma_v^2)$$

The effect of AWGN on a modulation signal is shown in the constellation diagram in Figure 2.6. In this constellation diagram, the symbols 0 and 1 are affected by noise, causing a distortion in amplitude and phase. The symbols 0 and 1 can still be distinguished from each other, but the distance between the symbols has decreased. This may lead to decision errors when the signal is more strongly corrupted by the noise.

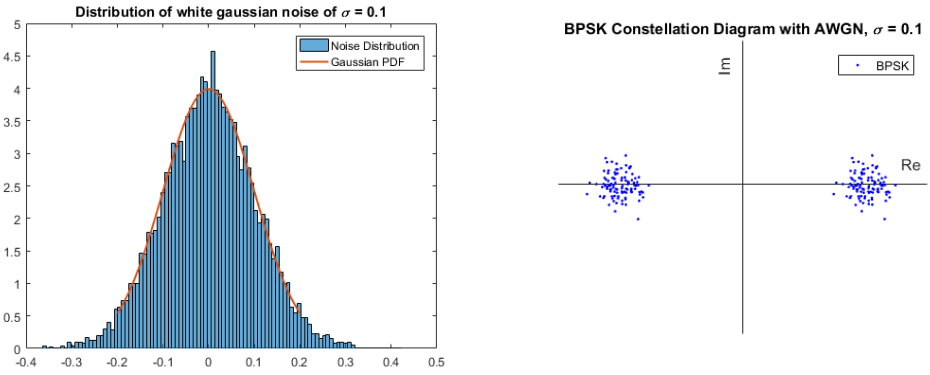


Figure 2.6: Distribution of AWGN (left) and Constellation Diagram of a BPSK signal that is affected by AWGN (right)

Symbol Detection

In order to recover the information from a noisy signal, there are multiple techniques, according to [14]. The idea of these techniques is to minimize the probability of errors in the detection of symbols.

This method is based on the optimum detector [14], which is implemented by the cross-correlation function. By using this function, it is possible to maximize the Signal-to-Noise Ratio (SNR) of a noisy signal when the expected input signal is known. The cross-correlation is defined as:

$$r_{xy}[n] = \sum_{k=-\infty}^{\infty} x[k]y[n+k] \tag{2.2}$$

A correlation-type demodulator calculates the cross-correlation of the input signal with each of the possible expected signals. The input signal $x[n]$ is multiplied with the expected signal $y[n]$ that is shifted by an offset k, and then integrated, yielding correlation signals $r_{xy}[n]$. There are several methods that implement the correlation demodulator, such as the correlator, the matched filter and the integrate-and-dump receiver.

Correlator

The first technique that implements the cross-correlation is the correlator, which is depicted in Figure 2.7. For each expected symbol, the correlation value is calculated by a multiplication and an integration. These correlation values can then be compared, yielding the most likely symbol. While the depicted structure is defined for the time-continuous domain, the same structure can be applied in time-discrete domain. The correlator is the most straight-forward implementation of a correlation-type demodulator, but not the most performance-efficient method.

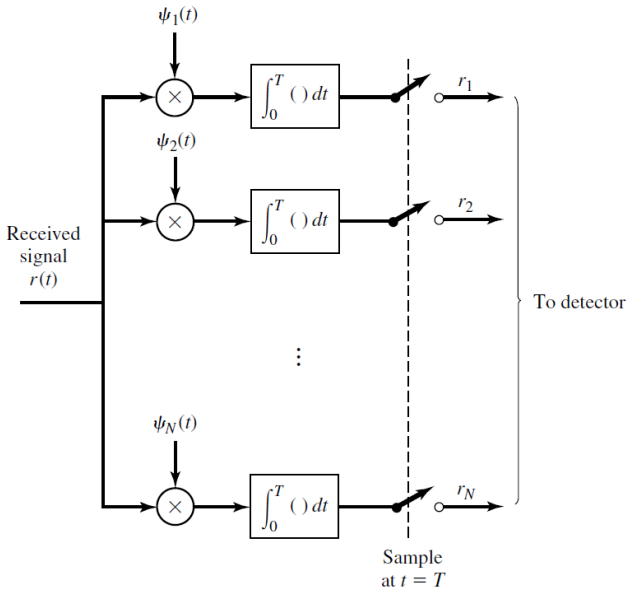


Figure 2.7: Overview of a correlation demodulator, cf. 'Correlation-type demodulator', [14], p.371

Matched Filter

A more efficient method is the matched filter. The matched filter also uses the auto-correlation function to minimize the error probability, this time, however, it is implemented by the method of the convolution. The auto-correlation can be rewritten as a convolution of two signals:

$$r_{xy}[n] = \sum_{k=-\infty}^{\infty} x[k]y[n+k] =$$

$$\sum_{k=-\infty}^{\infty} x[k]y[-(-n-k)] =$$

$$\sum_{k=-\infty}^{\infty} x[k]y[(-n-k)] = x[n] * y[-n]$$

This convolution can be implemented by a filter with an impulse response that is equal to the time-reversed signal $y[n]$. The detection of multiple symbols can be implemented by a structure as depicted in Figure 2.8. The filter functions $\psi_1, \psi_2, ...\psi_N$ are correspondent to the time-reversed waveform of each symbol. Similar to the structure of the correlator, this structure can also be applied in time-discrete domain.

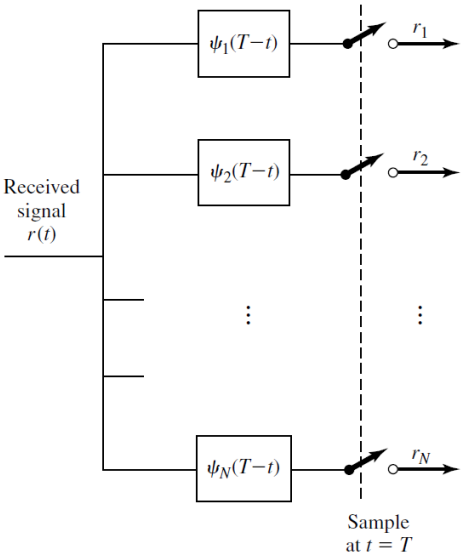


Figure 2.8: Overview of a matched filter topology, cf. 'Matched filter-type demodulator', [14], p.377

This structure can be generated by Finite Impulse Response (FIR) filters, which directly use the signal values as the impulse response. While FIR filters require a large complexity for long signals, other topologies like the Cascaded Integrator Comb (CIC) that can be used. In general, the main requirement for synthesizing a matched filter is the knowledge of the expected symbol.

Integrate-and Dump Receiver

A method that provides a simpler implementation than the matched filter and the correlator is the Integrate-and-Dump receiver. This receiver is applied when the expected signal is in a rectangular shape. As the correlator and the matched filter, the Integrate-and-Dump Receiver is based on the mathematical definition of the cross-correlation. When correlating a real-valued signal with a rectangular signal, the correlation operation can be described by an integration:

$$y[n] = \begin{cases} 1 & 0 <= n < \frac{N}{2} \\ 0 & else \end{cases}$$

$$r_{xy}[n] = \sum_{k=-\infty}^{\infty} x[k]y[n+k] = \sum_{k=0}^{\frac{N}{2}} x[k]$$

This integration can be implemented by an integrate-and dump circuit as described in [17]. The input signal is integrated for $\frac{N}{2}$ samples, then the integrator is reset. At the sampling instant $\frac{N}{2}$, the output of the integrate-and-dump receiver is equal to the output of a matched filter with the impulse response $y[n]$. The receiver is more simple and more computationally efficient than the matched filter, since it only requires an integrator and a sampler instead of a filter operation.

2.2.3 Symbol Synchronization

In digital communication systems, the synchronization between transmitter and receiver is crucial to guarantee a functional data transfer. While in wired communication systems the synchronization can be achieved by transmitting a clock signal simultaneously to the data transmission, the synchronization cannot be achieved so easily in wireless systems. In many wireless systems the clock must be recovered from the transmitted data signal.

Coherent receivers such as the I-Q demodulator require the exact symbol timings for a correct functionality. The output of the detectors, such as a correlator or matched filter, must be sampled at the right sampling point. The output of a correlator is in a triangular shape, such as depicted in Figure 2.10 due to the integrating property of the correlation function. This output should then be sampled at its maximum. In a signal affected by noise, the maximum may be shifted in time, therefore a periodic sampling method would lead to errors. The sampling times have to be adjusted. For solving this problem, there are various algorithms, such as peak detection, early-late gating or maximum-likelihood detection.

Peak Detection

In order to sample the matched filter output at its maximum, peak detection methods can be employed. For detecting the peak, a common method is to analyze the first derivative of the signal. In a local maximum of a function, the first derivative has a value of zero and changes its sign. This property can be exploited by using a differentiator and a zero-crossing detector on the signal to find the peaks. Such a topology is depicted in Figure 2.9.

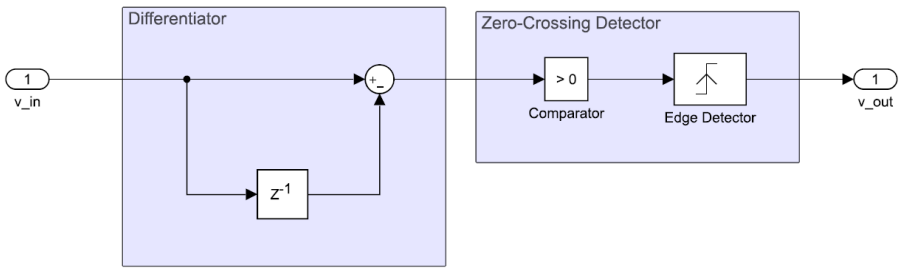


Figure 2.9: Overview of a simple peak detector that can be used for synchronization. The differentiated input signal is sent through a zero-crossing detector to determine the sampling instants.

Due to noise and distortions in the signal, there may be multiple local maxima in the signal. In order to filter unwanted peaks, the input signal can be smoothed by a lowpass filter. For this application a moving average filter would be a reasonable choice.

Early-Late Gating

A more advanced method is the Early-Late-Gating algorithm, which was described in [14] and [15]. [14] explains this concept by a closed control loop, which controls a sampler based on the values of the sampled signal at certain points of time. In every step of the algorithm, the input signal is sampled at an arbitrary time T. Next to this sample, two more samples are taken: one sample at an earlier point $T_E = T - \delta$ and one sample at a later point $T_L = T + \delta$. The sampling values of T_E and T_L are compared to find out if the sampling instant is too early or too late. Figure 2.10 depicts the timing of the sampler for three different cases.

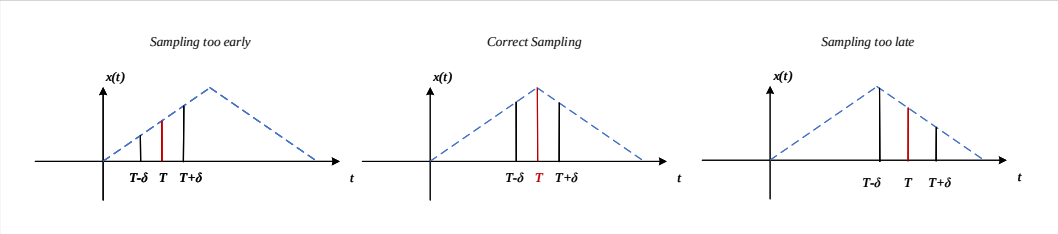


Figure 2.10: Early and late sampling of correlator output

If the sampling instant is too early, the value of $x(t)$ at T_E is higher than at T_L, and vice versa. The difference between the signals at T_E and T_L can be evaluated to determine the correct sampling instant.

A topology that implements the Early-Late-Gate method was described in the dissertation of Stevens [17], and is depicted in Figure 2.11.

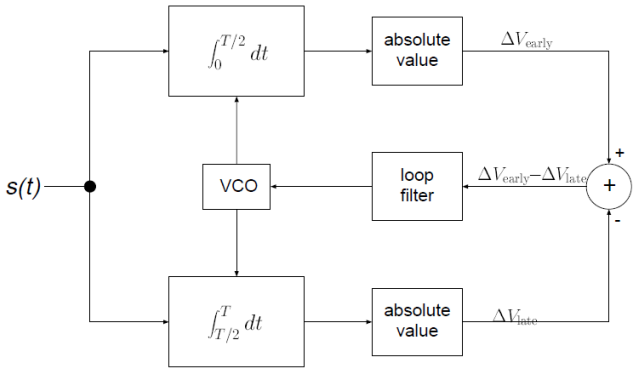


Figure 2.11: Implementation of an Early-Late Gating synchronizer, cf. [17],p.27, 'Early-late-gate data synchronizer'.

In this topology, there are two integrators, which integrate over the first half (early gate) and the second half (late gate) of the signal. The difference between the integrated samples is used to adjust the integration time of the integrators. When the output values of both integrators are equal, the sampling instant is at the correct timing.

2.3 Digital Filter Design

Digital filters are the main building block in signal processing. The most basic functionality of a digital filter is to modify the spectral characteristics of signals. While they can be used in simple applications, such as lowpass, highpass or bandpass filtering, they also find use in more complex algorithms. Since these filters are so widely used, various different topologies, such as FIR, Infinite Impulse Response (IIR) or CIC have been developed throughout the years. The topologies and design methods introduced in this section are related to textbooks by Lyons [10] and Oppenheim [12].

2.3.1 FIR Filters

The simplest filter topology is the FIR filter. This is due to the benefits of the finite impulse response, which include linear phase, stability, and a simple filter structure. The input-output relation of an FIR filter is defined as following for a filter length (taps) of N:

$$y[n] = \sum_{k=0}^{N-1} a_k x[n-k] \tag{2.3}$$

The order of an FIR filter is correspondent to the number of coefficients in the filter (including zero-valued coefficients). The transfer function of this filter is defined as:

$$H(z) = \sum_{k=0}^{N-1} a_k z^{-k} \tag{2.4}$$

As the transfer function contains of zeros but no poles, the filter is stable and the phase of the filter is linear. These filters can be created with a certain specified impulse response or magnitude/phase by making use of optimization algorithms. Widely used techniques in FIR filter design are the least-squares method and the Parks-McClellan algorithm.

FIR filters are one of the simplest topologies for digital filters. These filters consist of a forward path, in which various coefficients are multiplied with the delayed input signal.

For synthesizing filters with sharp edges in the magnitude response, high filter orders and therefore a high number of coefficients are necessary. This leads to higher computational efforts compared to other topologies.

2.3.2 IIR Filters

Digital filters that do not have a finite impulse response are called IIR filters. A basic definition of an IIR filter is:

$$y[n] = \sum_{i=0}^{N} b_i x[n-i] + \sum_{k=1}^{M} a_k y[n-k] \tag{2.5}$$

The delayed samples of the output signal $y[n-k]$ create a feedback inside the filter structure. The transfer function of such an IIR filter is:

$$H(z) = \frac{\sum_{i=0}^{N} b_i z^{-i}}{1 - \sum_{k=1}^{M} a_k z^{-k}} \tag{2.6}$$

The order of the filter is defined by the factor M.

IIR Filter Design

There are various methods for the design of IIR filters. Some of these methods are based on the design of an analog filter, which is then transformed into the digital domain. For instance, one of these methods is the impulse invariance method, which is based on evaluating the impulse response of a digital filter. This method is defined by the following steps, according to Lyons [10]:

1. Design analog filter impulse response $H(s)$.
2. Apply the bilinear transformation to the continuous impulse response to receive $H(z)$.
3. Substitute the sampling time t_S into the transfer function $H(z)$.

The z-transformed impulse response can be used as coefficient set for designing the filter.

2.3.3 Filter Implementation

There are multiple ways in which digital filters can be implemented. Since the difference equations that define a digital filter consist of multiplications, additions and delays, the implementation of a filter can be realized by a combination of these elements. The implementations have different benefits and drawbacks in terms of execution speed and numerical stability.

Direct Forms

The most basic filter implementation is the Direct Form 1, which can implement both FIR and IIR filters. This implementation contains a forward path and a feedback path. Each path consists of cascaded delay elements, the output of the delays are multiplied with the corresponding coefficients and accumulated. An implementation of Direct Form 1 is depicted in Figure 2.12.

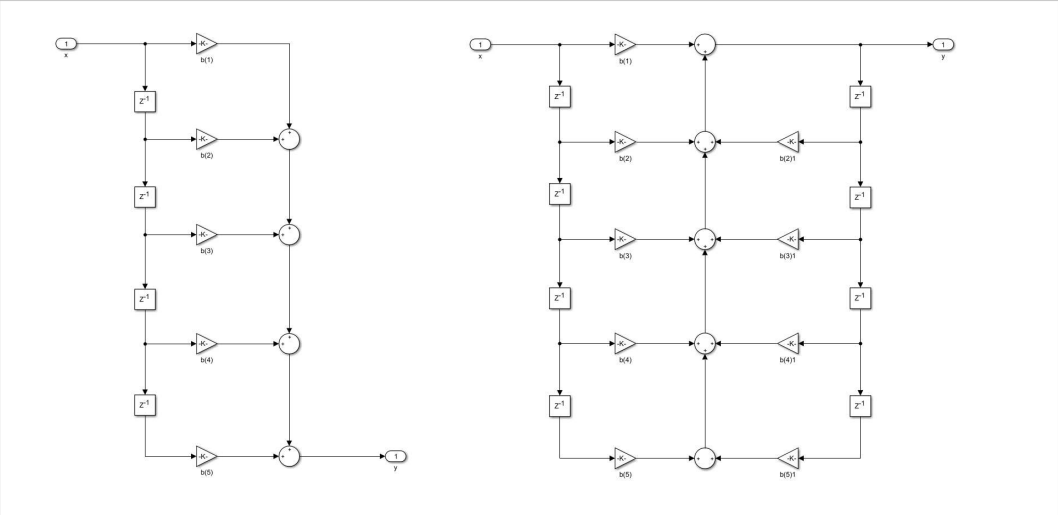


Figure 2.12: Comparison of an FIR filter (left) to an IIR implementation in Direct Form 1 (right)

Since the Direct Form 1 is not a very performance-efficient solution, other implementations of digital filters have been created, such as the Direct Form 2 other the Transposed Direct Form 2. These implementations reduce the number of delays in the system by adjusting the filter structure.

High-order IIR Filters

While FIR filters of higher orders can be implemented in a Direct Form structure, IIR filters of high orders must be analyzed carefully. Since IIR filters are not guaranteed to be stable, filters at high orders may experience

instabilities due to quantization errors. In order to minimize such risks, IIR filters of high orders are mostly created from a cascade of second-order filters, which are also called biquads.

2.3.4 Filter Performance

The performance of a digital filter depends on the number of operations that are necessary to compute one single filtering step. The operations in a digital filter implementation can be divided into arithmetic and memory operations.

The arithmetic operations in digital filters consist of multiplication and addition operations. The input signal is multiplied with different coefficients and stored in delay elements, the results are added. The operation of multiplication and addition can be combined into a Multiply-Accumulate (MAC) operation. The memory operations are defined by the delay elements in the system. The process of storing a value in a delay element requires time and resources, which must be considered when analyzing the performance.

In FIR filter design the performance is directly proportional to the number of taps in the filter. Each tap is represented by a delay, a multiplication and an addition.

2.3.5 Recursive FIR Filters

The design of FIR filters is strongly based on optimization algorithms. These algorithms take a desired magnitude response as specification, and iteratively calculate filter coefficients to fit to the magnitude response. In order to fit the designed filter to the specifications, the algorithms often use filters with high orders. Such FIR filters are often implemented with orders of 100 or higher, causing performance issues.

The performance of these filters can be improved by adopting a recursive strategy to design the filter. This approach is based on separating a large filter structure into smaller structures. In the implementation of an FIR filter, the filter structure is created directly from the impulse response. This means

that each sample of the impulse response requires a MAC operation. In order to increase the efficiency of the filter, the filter structure is split up into a cascade of smaller filters. The technique of creating recursive filters is described by Hassan et al. [7] and also by Vainio et al. [22].

The recursive filter design is based on finding an analytical expression of the filter transfer function in the z-domain. The analytic expression should consist mainly of multiplications of different terms, which can then be implemented as a cascade of filters. For this purpose, basic arithmetic rules such as polynomial division, binomial formulas, geometric sums or similar are used. For instance, an FIR filter defined by a cubic function

$$H(z) = z^{-3} + 3z^{-2} + 3z^{-1} + 1$$

can be described by a cascade of three filters:

$$H(z) = (1 + z^{-1}) \cdot (1 + z^{-1}) \cdot (1 + z^{-1})$$

The decomposition yields three smaller filter structures. The structures are then cascaded with each other. The cascaded implementation decreases the computational costs since the structures are simpler. In this example, two multiplications have been removed from the equation, the filter can be implemented by additions. For filters of higher orders the computational costs may decrease significantly (see [7]).

The recursive implementation may introduce pole-zero cancellations into the system, therefore the stability of the filter is no longer granted. Inaccuracies in the implementation, such as fixed-point inaccuracy or rounding errors, can create instabilities in the system. Therefore, the stability of recursive filters must be analyzed carefully.

2.3.6 CIC Filters

Similar to the recursive FIR filter, a performance-efficient topology of filters is the CIC structure. This structure was first mentioned by Eugene Hogenauer in [8]. These filters consist of multiple stages, which can be implemented by addition and delay elements. There are no multiplications

required, which means that these filters require low computational effort. While CIC filters are mainly used in sampling-rate conversions such as interpolation and decimation, they can also be used for implementing FIR filters such as the moving average.

The CIC principle is based on combining an integrator filter with a comb filter to create a uniform impulse response.

The discrete integrator is defined as:

$$H_I(z) = \frac{1}{1 - z^{-1}}$$

A comb filter consists of an impulse response of two samples: one positive and one negative. The samples are separated by the delay M, creating a transfer function of:

$$H_C(z) = 1 - z^{-M}$$

When these two filters are combined, the following transfer function is created:

$$H(z) = H_C(z) \cdot H_I(z) = \frac{1 - z^{-M}}{1 - z^{-1}}$$

The impulse response of this filter is equivalent to a moving average filter with M samples and gain M. The filter can be implemented by the following topology (Figure 2.13):

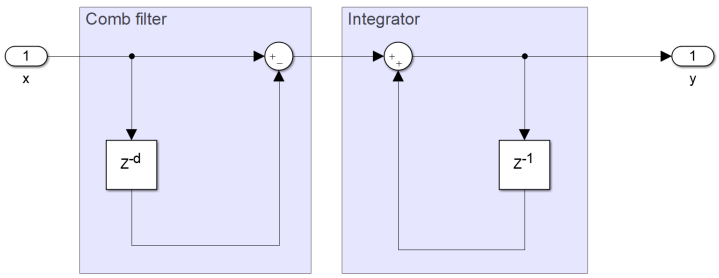


Figure 2.13: Schematic overview of a CIC filter

CIC Filter Applications

CIC filters are widely used in digital signal processing. This is due to their simple and performance-efficient implementation, which consists purely of additions and delay elements. CIC filters are mainly used in multi-rate systems, as decimation or interpolation filters for up- and downsampling. In interpolation, the rate conversion is done between the comb filtering and integrating step, while in decimation the comb stage is placed before the rate conversion.

2.4 System Architectures in Signal Processing

Signal processing algorithms can be implemented on many computing platforms. Since the algorithms consist of mathematical structures, they can be translated into various computational languages or implemented in digital design or as analog circuit. Each of these methods offers benefits and drawbacks in terms of long-term stability, implementation cost and performance.

In order to optimize the performance of signal processing algorithms, various architectures have been invented. In the first designs of processors, mainly general-purpose microprocessors were implemented. However, in state-of-the art technology, the standard architectures are specifically designed for signal processing, so-called DSPs architectures. Alternatively, filter algorithms are implemented in Hardware Description Language (HDL) on Field Programmable Gate Arrays (FPGAs) or ASICs.

This section introduces and compares different architectures for DSP and FPGAs, relating to the book Software-Defined Radio for Engineers [3].

2.4.1 FPGAs

In the field of circuit design FPGAs have become a popular implementation platform for signal processing algorithms. While the FPGA has its origin in simple logic cells and programmable logic arrays, state-of-the art FPGAs

offer a wide range of functionality. The main benefits of FPGA in comparison with processors are the variable clock frequency and the parallel execution of logic. In comparison with ASICs, FPGAs are a cheaper solution, even though they are more power consuming and require more chip area [5].

An FPGA consists of various logic cells, which are connected by programmable switches. Depending on the programming, different structures of both combinational and sequential logic can be constructed. The main building blocks of FPGA logic are memory cells, flip-flops, look-up tables (combinational logic) and clock/routing elements.

In addition to these basic cells, state-of-the-art FPGAs contain additional elements. For development, FPGAs are often embedded into System-on-Chip (SoC). The combination with a microcontroller, RAM or flash memory and bus interfaces makes FPGAs a versatile platform for signal processing.

An example for a widely used FPGA SoC is the Xilinx Zynq series, which contains a Xilinx Artix/Kintex/Virtex FPGA, an ARM processor core and additional coprocessors. An overview of the Zynq-7000 is depicted in Figure 2.14.

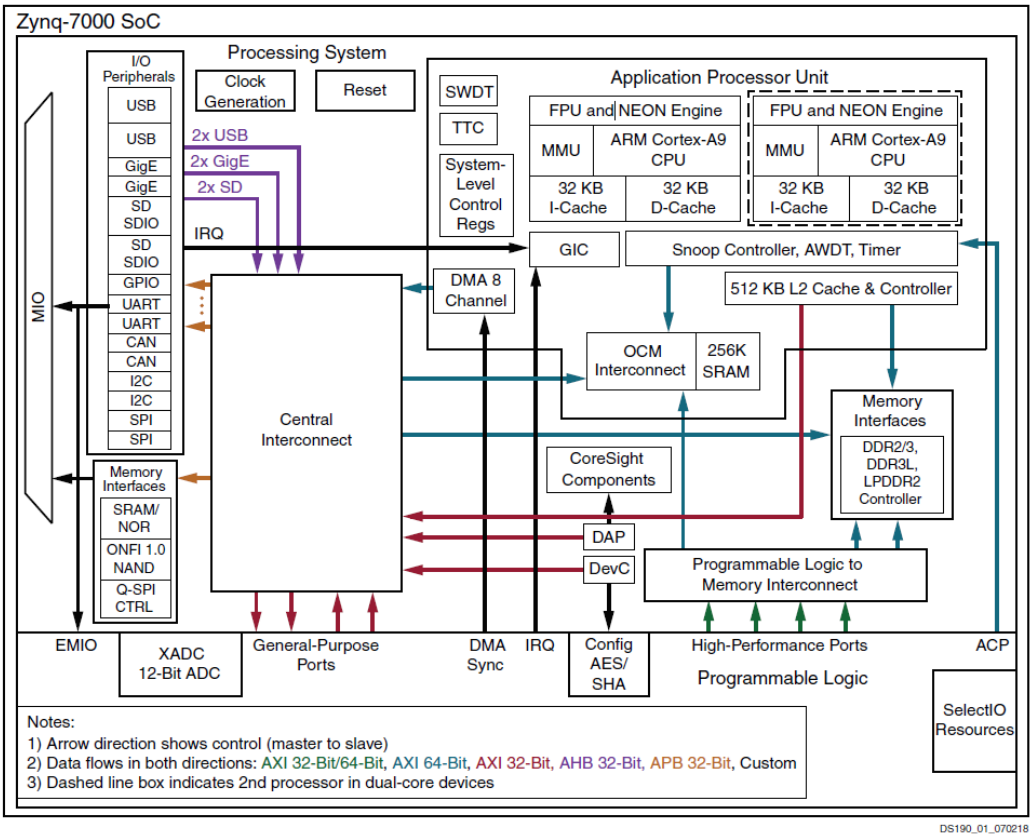


Figure 2.14: Overview of the Xilinx Zynq-ZC7045 [24]

FPGAs - DSP Slices

Many FPGAs contain so-called DSP slices. DSP slices are logic cells designed for DSP operations, mostly they implement a multiply-accumulate operation. In Verilog code, the slices can be instantiated directly, or assigned through multiplication and addition operators. The basic MAC operation would be implemented as:

$$P = (A \cdot B) + C$$

Therefore, a DSP slice must contain at least a multiplier and an accumulator. Depending on the implementation of the slice, various additional operations can be used in combination with the MAC.

DSP slices are embedded in many state-of-the art FPGAs, for instance, in the Xilinx Zynq as the DSP48E1 slice [1]. This slice is described in Figure 2.15.

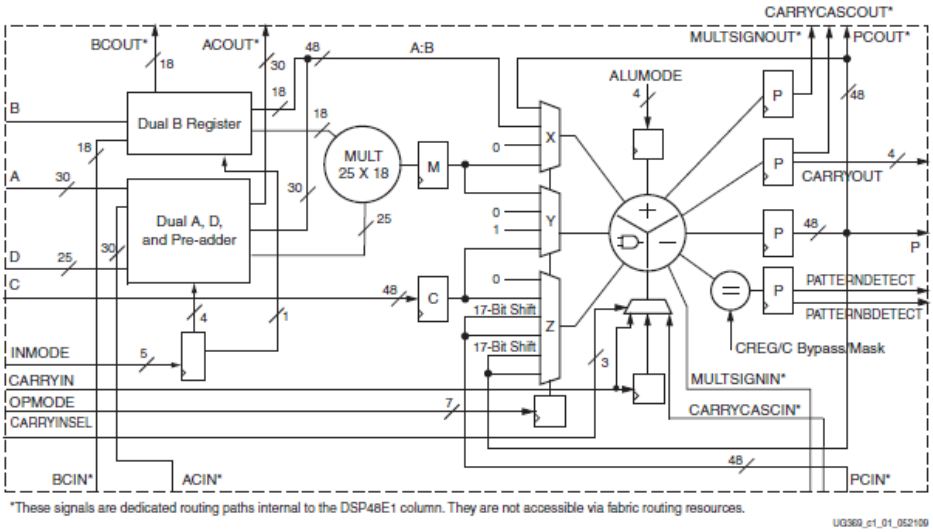


Figure 2-1: **7 Series FPGA DSP48E1 Slice**

Figure 2.15: Architecture of the Zynq DSP Slice, [1]

The main part of the slice is the hardware multiplier, which supports inputs of 25 x 18 bit. Afterwards, a 42 bit arithmetic/logic operation (addition/ subtraction, and, or, xor etc.) can be executed in the slice. In addition to this basic functionality, the DSP48E1 also includes a pre-adder before the multiplication and rounding and saturation logic. The most advanced mathematical operation that can be implemented by the DSP48E1 is:

$$P = (A \pm D) \cdot B \pm C$$

The slice can be configured to detect overflows and underflows, and implements a saturation logic. While the DSP48E1 can be used without any clock input, it also offers the option to use a pipelining register between the multiplier and arithmetic/logic and registers at the main ouptuts.

2.4.2 DSP Architectures

DSPs are processors that are designed specifically for digital signal processing purposes. In contrast to general purpose processors, DSP architectures are optimized for high data throughput and parallel processing of instructions. Since the instructions that are used in signal processing are mainly filtering operations, DSPs are designed to execute arithmetic operations efficiently.

Many DSPs support special arithmetic operations that are common in signal processing, such as complex number arithmetic, Fast Fourier Transformation (FFT) algorithms or Viterbi decoding.

Functional Units

The architecture of a DSP is defined by several features. The main part of the architecture are the functional units, which include the Arithmetic-Logic Unit (ALU), the address generation units and the datapath. While in a general purpose Central Processing Unit (CPU), the functional units are usually implemented only once per unit type, a DSP features multiple instances of the functional units.

The ALU is often distributed into multiple parallel units. Since in DSP applications many multiplication and addition operations are required, most DSPs contain a dedicated unit for the MAC operation. The MAC unit can be implemented multiple times in order to allow parallel processing, such as in the Analog Devices Blackfin core, which supports two parallel 16-bit and one 32-bit MAC operations. The architecture of the Blackfin core is depicted in Figure 2.16.

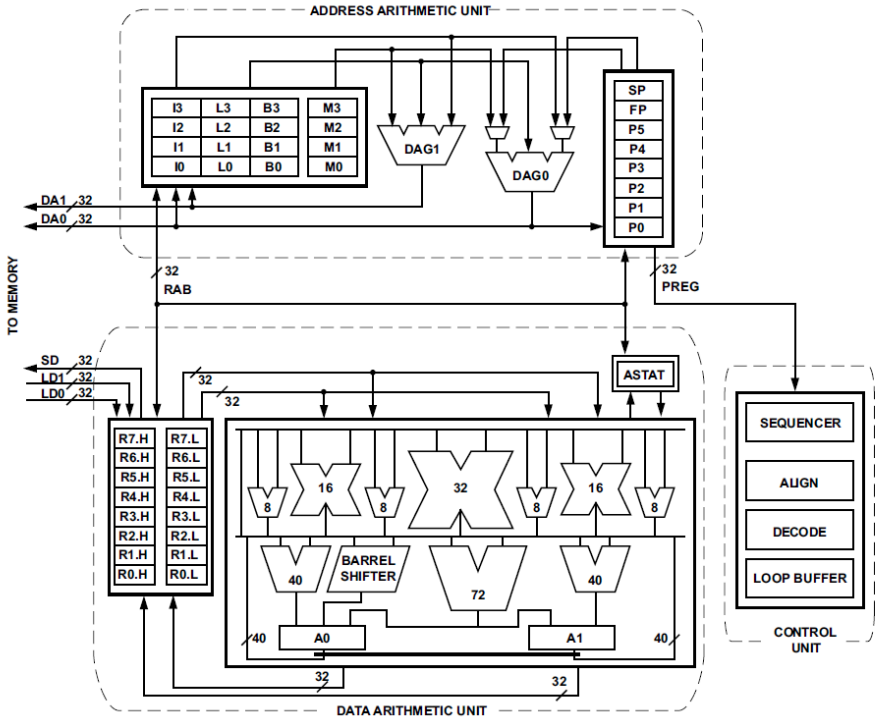


Figure 2.16: Overview of the Analog Devices Blackfin core

The MAC unit is often extended by a rounding or saturation unit. The other arithmetic and logic operations are mostly divided into separate ALUs for different bit widths, for instance into one unit each for 16-bit, 32-bit and 64-bit instructions.

The number of functional units in a DSP can be used as a guideline to estimate its possible performance. The more functional units are available, the more operations can be executed in parallel. However, it is important to notice that the functional units do not operate at full capacity all the time. When using the functional units, factors such as the memory latency or the data dependencies inside the code play an important role.

In order to fully use the parallel arithmetic and MAC units, most DSPs implement multiple datapath structures and load/store units. This means

that there are multiple address generation units that are then connected to the datapath and the ALU.

Execution Times and Pipelining

DSPs implement various arithmetic/logic and memory instructions, which may have different execution times. For instance, in the case of memory access operations like Load/Store, the execution time depends on the datapath architecture of the CPU and the internal delay of the memory. Especially when the memory is clocked at lower frequency than the CPU, memory access may take multiple cycles. In contrast to this, a simple integer logic operation can be implemented as a single-cycle instruction.

In order to execute the instructions in an optimal execution time, state-of-the-art DSPs implement pipelining mechanisms. Pipelining distributes the instructions into stages, at the most basic version into Fetch, Decode and Execute. These stages can then be executed in parallel, leading to a higher throughput.

More advanced DSPs implement multi-stage pipelines, such as the Analog Devices Blackfin with a 10-stage pipeline and the TI TMS320C6713 with a 16-stage pipeline (depicted in Figure 2.17).

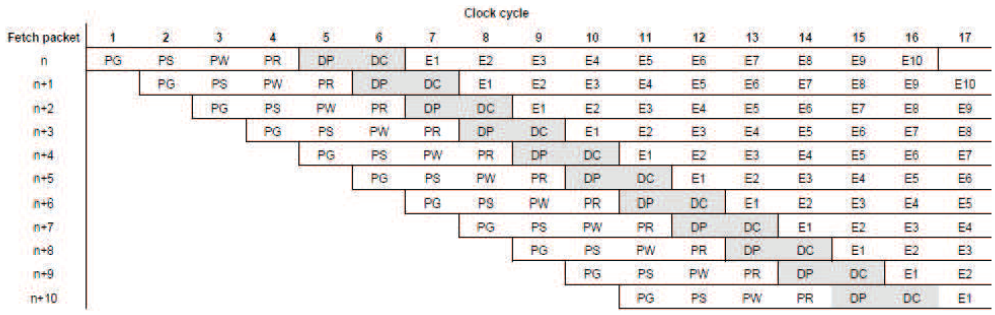


Figure 2.17: Overview of the 16-stage C6713 pipeline, from [23]

In this pipeline, there are four sub-stages for the Fetch stage, two sub-stages for the Decode stage and up to 10 sub-stages for the Execute stage, depending on the type of instruction.

Floating/Fixed Point Arithmetic

While general-purpose processors implement floating-point arithmetic as a standard feature, DSPs often omit floating-point calculations. This is due to the large complexity of a floating-point arithmetic pipeline, which takes multiple shift and arithmetic operations. The drawbacks are the longer execution time (multiple cycles required) and the required chip area, which is a critical factor in DSP cores for ASICs.

Therefore, many smaller DSP cores only use fixed-point/integer arithmetic. While this arithmetic offers a significantly smaller computation range than floating-point arithmetic, the complexity of the architecture is drastically reduced. The computation range limits can be partially compensated by supporting larger bit widths in computation.

An example for a fixed-point DSP is the NXP Coolflux, which is optimized for ASIC development. The Coolflux is available in a 16-bit and 32-bit architecture. An overview of this core is depicted in Figure 2.18. Optionally, the Coolflux core can be combined with additional coprocessors for Viterbi decoding or complex number operations.

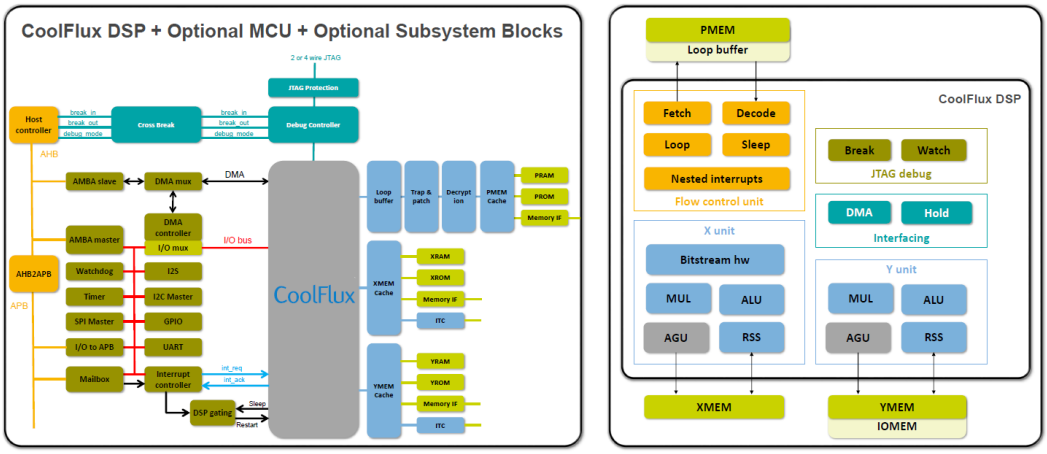


Figure 2.18: Overview of the NXP Coolflux DSP core, cf.[21]

3 Design

The main focus of this work is to analyze an existing RFID receiver system and to optimize the signal processing algorithms inside it. Therefore, the system had to be analyzed in terms of performance. Then, an optimization of the system was proposed and evaluated in Simulink. This chapter gives an introduction into the existing system and the proposed optimizations.

3.1 Existing System Overview

The system on which this work is based on is a mixed-signal RF transceiver IC which is developed by NXP Semiconductors. This IC can be operated both as a reader and a transponder, in either Reader Mode (RM) and Card Mode (CM).

In RM the IC supports the ISO standards ISO 14443-A and B for data rates up to 848 kBd and ISO 18092 FeliCa for up to 424 kBd. The reader mode also supports ISO 15693 for single sub-carrier modes. In CM the IC supports the ISO standards ISO 14443-A, ISO-14443-B and ISO 18092 for selected data rates. Table 3.1 shows an overview of the standards supported in reader or card mode.

Mode	Standard	Supported Data Rates
RM	ISO 14443-A	106, 212, 424, 848 kBd
RM	ISO 14443-B	106, 212, 424, 848 kBd
RM	ISO 15693 Single Sub-Carrier	6.6, 26.5 kBd
RM	ISO 15693 Single Sub-Carrier	53, 106, 212 kBd
RM	ISO 18092 FeliCa	212, 424 kBd
CM	ISO 18092	106, 212, 424 kBd
CM	ISO 14443-A	106, 212, 424, 848 kBd
CM	ISO 14443-B	106, 212, 424, 848 kBd

Table 3.1: Supported ISO standards by the existing hardware implementation

In this work the focus is set on analyzing and optimizing the reader mode of the existing system. Therefore, the general functionality of the reader mode is introduced.

3.1.1 Existing System - Reader Mode

The reader mode signal processing chain consists of multiple steps. The incoming data consists of an I and a Q signal, which are then processed separately. Firstly, the data is preprocessed, then a matched filter and an averaging filter are implemented. Then, the I and Q channels are combined. The final stage of the signal processing chain is a symbol synchronizer.

SNR Maximization - Matched Filtering

One of the main components of the signal processing chain is a matched filter, which is used for signal detection. Since the RF modulation techniques use a rectangular subcarrier signal, the matched filter detector must replicate rectangular signals. For instance, for BPSK modulation, a pure square signal and a square signal with phase inversion must be detected. A system of matched filters for BPSK would have the following impulse responses (Figure 3.1):

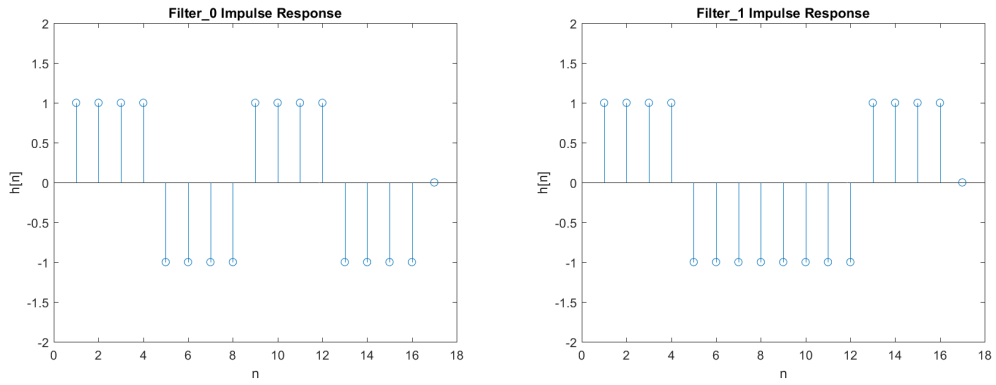


Figure 3.1: Matched filters for BPSK signals: Filter to detect a logic 0 (left) and a logic 1 (right)

The impulse response is defined as following (Filter_0):

$$\begin{aligned} h[n] = \delta[n] + \delta[n-1] + ... + \delta[n-3] - \delta[n-4] - ... \\ -\delta[n-7] + \delta[n-8] + ... + \delta[n-11] - \delta[n-12] - ... - \delta[n-15] \end{aligned} \tag{3.1}$$

For Filter_1, the impulse response would be defined as:

$$\begin{aligned} h[n] = \delta[n] + \delta[n-1] + ... + \delta[n-3] - \delta[n-4] - ... \\ -\delta[n-7] - ... - \delta[n-11] + \delta[n-12] - ... + \delta[n-15] \end{aligned} \tag{3.2}$$

Absolute Value Calculation + Averaging

In order to detect a BPSK signal, the maximum value of the matched filter must be analyzed. Before obtaining the maximum, the signal is smoothed. This is implemented by an absolute value calculation and a moving average filter.

The moving average is implemented as a recursive filter:

$$x_C[n] = \frac{1}{M}\left(x[n] - x[n-M]\right) \tag{3.3}$$

$$y[n] = y[n-1] + x_C[n] \tag{3.4}$$

Channel Combination

The I and Q signal must be combined to calculate the magnitude of the complex input signal. This is defined as following:

$$mag[n] = \sqrt{x_i^2[n] + x_q^2[n]} \tag{3.5}$$

Since this computation is expensive due to the square and square root operations, a more efficient algorithm was used.

The alpha-min-beta-max algorithm is used to approximate the magnitude of a complex signal. This algorithm is defined as:

$$mag[n] = \alpha \cdot min(|x_i[n]|, |x_q[n]|) + \beta \cdot max(|x_i[n]|, |x_q[n]|) \tag{3.6}$$

The parameters α and β are defined by empirical measurements. In some cases, when the values of α and β fulfill certain constraints, the multiplication can be implemented by an integer operation or a shift operation, making the computation more efficient. In addition, the absolute value calculation can be omitted since the input values of the channel combiner are already positive.

Symbol Synchronization

After the channel combination, the signal must be checked against timing inaccuracies, e.g. jtters. In order to make sure that the sampling of the matched filter output happens at the point where the signal is at a maximum, the signal processing chain contains a zero-crossing detection synchronizer.

This system uses a two-stage filter cascaded with a zero-crossing detector for symbol synchronization. The output of this system is a one-bit signal which changes its level at the optimum sampling points. The synchronization filter

is defined as following:

$$x_C[n] = \frac{1}{M}(x[n] - 2x[n-M] + x[n-2M]) \tag{3.7}$$

$$z\,[n] = z\,[n-1] + x_C\,[n] \tag{3.8}$$

$$y\,[n] = y\,[n-1] + z[n] \tag{3.9}$$

The zero crossing detection is implemented as:

$$sync\,[n] = sign\,(y\,[n] - y\,[n-1]) \tag{3.10}$$

After the symbol synchronization the signal can be decoded.

3.2 Proposed Optimizations - Matched Filtering

There are different ways to optimize the performance of a digital filter system in terms of execution time. In the existing system, the main bottleneck was the matched filter signal detector. Therefore, a main focus was set on the optimization of the matched filters.

3.2.1 Comb Filtering

In the existing filter system, the signal detector for subcarrier RF signals was implemented by multiple FIR filters with up to 64 taps each. The runtime of the FIR filtering operation is directly proportional to the length of the filter impulse response, every tap of the filter requires a delay, a multiplication and an addition. In filters with higher orders (e.g. $N > 100$), this leads to a drastic increase in computation time due to long delay lines and a large number of multiplications.

Since these filters caused the main computational expense, the first approach in optimization was to simplify the matched filter. In this process the recursion technique was applied.

The original implementation contains two matched filters for each BPSK data rate with impulse responses as depicted in Figure 3.1. Since these filters were computationally expensive, the existing system already implemented a first optimization by using the CIC principle. The FIR filter was split into a comb filter $x_C[n]$ and a weighted integrator. The splitting operation was done by calculating the first derivative of the impulse response.

For the impulse response $h[n]$ of Filter_0 (Equation 3.1), the first derivative of the impulse response resulted in the following comb function:

$$x_C[n] = x[n] - 2x[n-4] + 2x[n-8] - 2x[n-12] + x[n-16] \tag{3.11}$$

The comb was then followed by an integrator:

$$y[n] = y[n-1] + \frac{1}{M} x_C[n] \tag{3.12}$$

In addition, the weight factor M was defined to normalize the filter gain for different filter lengths in samples, and could therefore be set to a different value for each filter.

The resulting impulse response of the comb filter and the full filter are depicted in Figure 3.2.

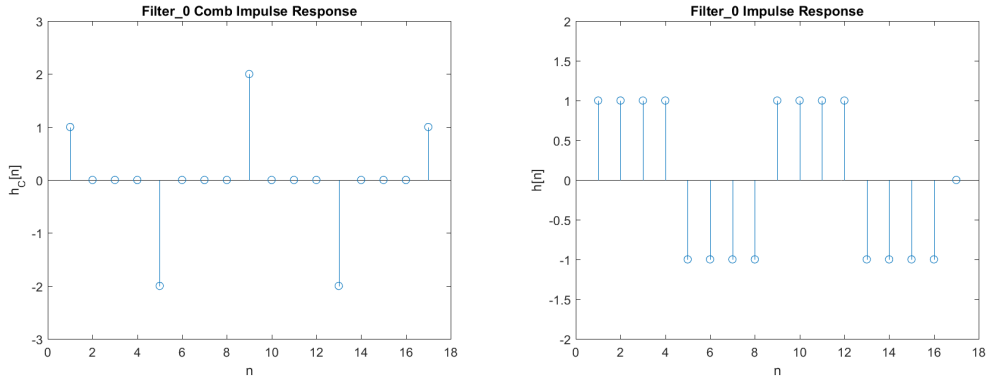


Figure 3.2: Impulse response of the comb filter $x_C[n]$ (left) and full impulse response $h[n]$ for Filter_0. The generated impulse response is identical to the original FIR impulse response (referenced in Figure 3.1).

For Filter_1, the differentiation results in:

$$x_C[n] = x[n] - 2x[n-4] + 2x[n-12] - x[n-16] \tag{3.13}$$

$$y[n] = y[n-1] + \frac{1}{M}x_C[n]$$

The impulse response is then generated as (Figure 3.3):

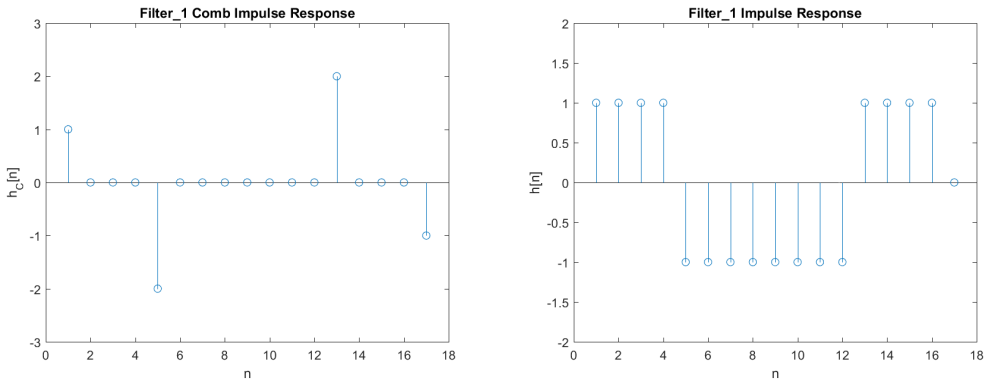


Figure 3.3: Impulse response of the comb filter $x_C[n]$ (left) and full impulse response $h[n]$ for Filter_1. As for Filter_0, the generated impulse response is identical to the original (referenced in Figure 3.1).

3.2.2 Recursive Filter Implementation

The comb filtering topology is already more computationally efficient than the original FIR filter. However, in order to further increase the performance of the matched filtering, another optimization was introduced. The optimized filter should use a minimum number of calculations.

In order to reach this goal, an approach that was used in [7] was applied. The aim of this approach is to decrease the number of taps in the filter by using recursion. This makes use of the fact that transfer functions of filters can be expressed as the result of a rational function in the z-domain. In the z-domain, an FIR transfer function can be described by a polynomial of order N. This polynomial can then be simplified by means of polynomial division or other arithmetic operations, leading to a simplified transfer function. Since the simplification may introduce additional poles into the system, the stability of the new filter must be analyzed carefully.

The application of this principle was then applied for Filter_0 and Filter_1. Since the derivations are similar for both filters, only the derivation for Filter_0 is described fully. The derivation is based on the comb filter equations 3.11 and 3.12.

The first step of the optimization is a generalization of the matched filter

equations. For this purpose, the parameters N and k are defined. N is defined as the number of taps of the whole filter, while k is the number of taps per half subcarrier period in the matched filter impulse response. The number of taps in the comb filter corresponds to the value $\frac{N}{k} + 1$. Therefore, equation 3.11 can be rewritten as:

$$x_C[n] = x[n] - 2x[n-k] + 2x[n-2k] - 2x[n-3k] + x[n-N] \tag{3.14}$$

The next step of the optimization is the Z-Transformation to gain the transfer function of the comb $H_C(z)$:

$$X_C(z) = X(z) - 2X(z)z^{-k} + 2X(z)z^{-2k} - 2X(z)z^{-3k} + X(z)z^{-N}$$

$$H_C(z) = \frac{X_C(z)}{X(z)} = 1 - 2z^{-k} + 2z^{-2k} - 2z^{-3k} + z^{-N}$$

In order to simplify the transfer function a correction function was introduced:

$$H_{corr}(z) = 1 - z^{-N}$$
$$H_{C,corr}(z) = H_C(z) + H_{corr}(z)$$
$$H_{C,corr}(z) = 2 - 2z^{-k} + 2z^{-2k} - 2z^{-3k} = 2\left(z^{-k} + z^{-2k} - z^{-3k}\right)$$

$$H_{C,corr}(z) = 2\left(1 + z^{-k} + z^{-2k} - z^{-3k}\right) = 2\left(\sum_{i=0}^{1} z^{-2ki} - \sum_{i=0}^{1} z^{-k(2i+1)}\right)$$

The sum range is substituted by $\frac{N}{2k} - 1$, resulting in:

$$H_{C,corr}(z) = 2\left(\sum_{i=0}^{\frac{N}{2k}-1} z^{-2ki} - \sum_{i=0}^{\frac{N}{2k}-1} z^{-2ki} \cdot z^{-k}\right) = 2 \cdot (1 - z^{-k}) \sum_{i=0}^{\frac{N}{2k}-1} z^{-2ki}$$

Using the geometric sum formula $\sum_{i=0}^{N} q^i = \frac{1-q^{N+1}}{1-q}$, the transfer function is

further simplified:

$$H_{C,corr}(z) = z^{-2ki} = 2 \cdot (1 - z^{-k}) \cdot \frac{1 - z^{-2k(\frac{N}{2k})}}{1 - z^{-2k}}$$

$$H_{C,corr}(z) = 2 \cdot (1 - z^{-k}) \frac{1 - z^{-N}}{1 - z^{-2k}} = 2 \cdot (1 - z^{-N}) \frac{1 - z^{-k}}{1 - z^{-2k}} = 2 \cdot \frac{1 - z^{-N}}{1 + z^{-k}}$$

The last step in the z-domain is removing the correction $H_{corr}(z)$ and combining the comb filter with an integrator.

$$H_C(z) = (1 - z^{-N}) \cdot \frac{1 - z^{-k}}{1 + z^{-k}} = H_{C,1}(z) \cdot H_{C,2}(z)$$

$$H_{C,1}(z) = (1 - z^{-N})$$

$$H_{C,2}(z) = \frac{1 - z^{-k}}{1 + z^{-k}}$$

$$H(z) = H_C(z) \cdot \frac{1}{1 - z^{-1}}$$

In the time domain, the filter is expressed by three difference equations:

$$h_{C,1} : x_{C1}[n] = x[n] - x[n - N] \tag{3.15}$$

$$h_{C,2} : x_C[n] = x_{C1}[n] - y[n - k] - x_{C1}[n - k] \tag{3.16}$$

$$h : y[n] = y[n - 1] + \frac{1}{M} x_C[n] \tag{3.17}$$

The same transformation was applied to the matched filters for symbol 1, resulting in the following difference equations:

$$h_{C,1} : x_{C1}[n] = x[n] - x[n - N] \tag{3.18}$$

$$h_{C,2} : x_C[n] = x_{C1}[n] - y[n - k] - x_{C1}[n - k] \tag{3.19}$$

$$h : y[n] = y[n - 1] + \frac{1}{M} x_C[n] \tag{3.20}$$

For both filters, a recursive system consisting of three stages could be generated. This system then had to be analyzed and compared to the original model.

Impulse response

The first part of the analysis was the generation of the impulse responses for Filter_0 and Filter_1 to check whether the resulting impulse response was still the same. Therefore, the impulse responses $h_{C,1}$ and $h_{C,2}$ of the generated Filter_0 are depicted in Figure 3.4.

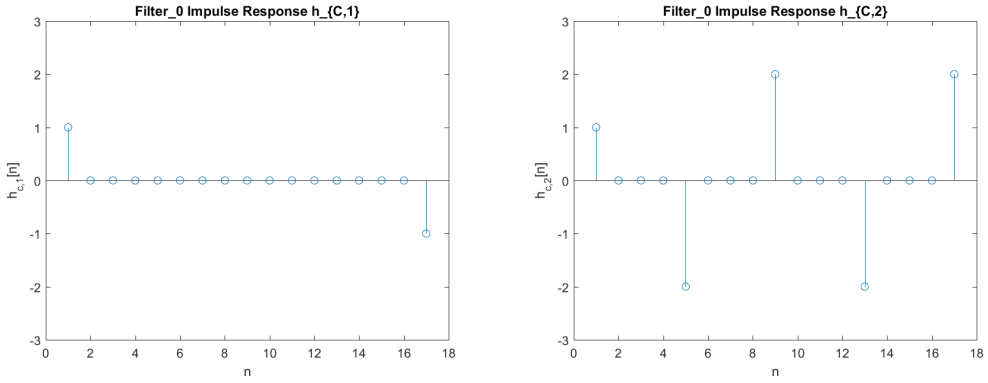


Figure 3.4: Impulse response of the generated filters $h_{C,1}$ (left) and $h_{C,2}$ (right) for Filter_0.

Combining $h_{C,1}$ and $h_{C,2}$ lead to the comb filter function h_C (depicted in Figure 3.5):

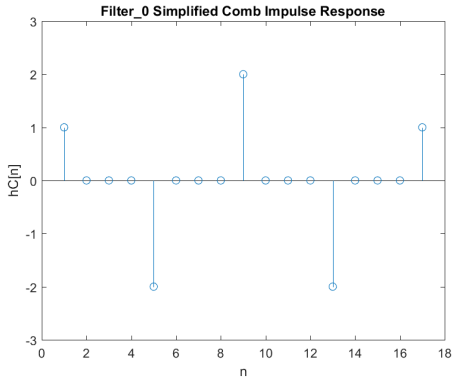


Figure 3.5: Impulse response h_C for generated Filter_0.

Comparing with Figure 3.2, it is visible that the optimized filter has the same impulse response as the original comb filter (Equation 3.11).

The same procedure was applied for Filter_1 (depicted in Figure 3.6):

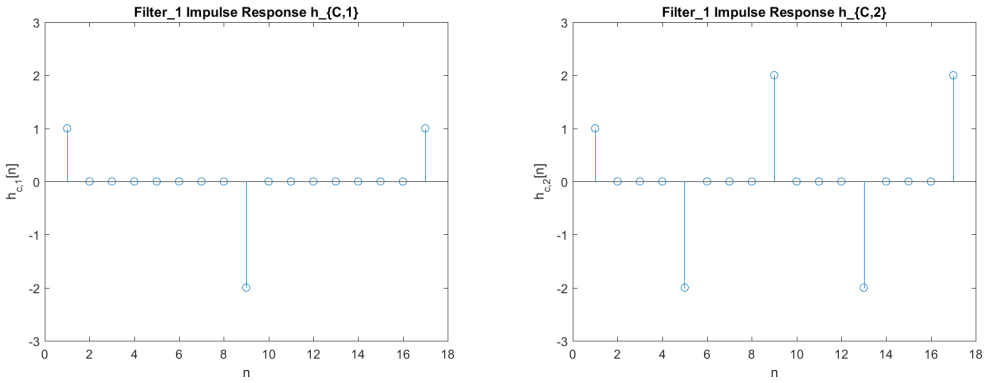


Figure 3.6: Impulse response of the generated filters $h_{C,1}$ (left top) and $h_{C,2}$ (right) for Filter_1

Combining $h_{C,1}$ and $h_{C,2}$ leads to the generated comb filter function for Filter_1 (depicted in Figure 3.7):

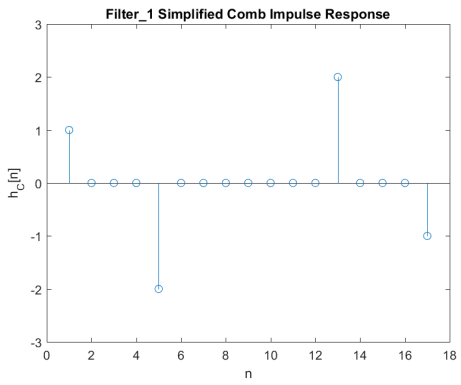


Figure 3.7: Impulse response h_C for generated Filter_1

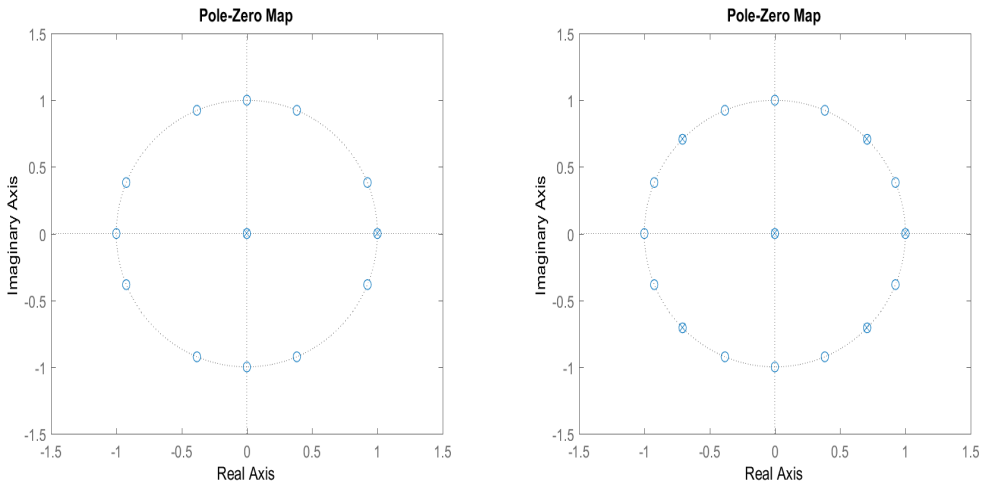
As for Filter_0, the impulse response h_C of the generated filter for Filter_1 is equal to the original comb filter (comparing with Figure 3.3).

This means that the impulse response of the matched filter system is not affected by the recursive optimization. However, this check alone is not enough to prove the feasibility of the optimization. Therefore, further analysis had to be performed. The first step was a pole-zero analysis.

Pole-zero analysis

A pole-zero analysis is essential to check the stability of the optimized filter. While the optimization has not changed the impulse response of the filter, the change of topology may introduce new poles and zeros in the z-domain into the system. If there is a pole outside the unit circle, the system is unstable and not usable. Therefore, the pole-zero maps of the original and optimized filter must be analyzed. This was done for Filter_0 and Filter_1.

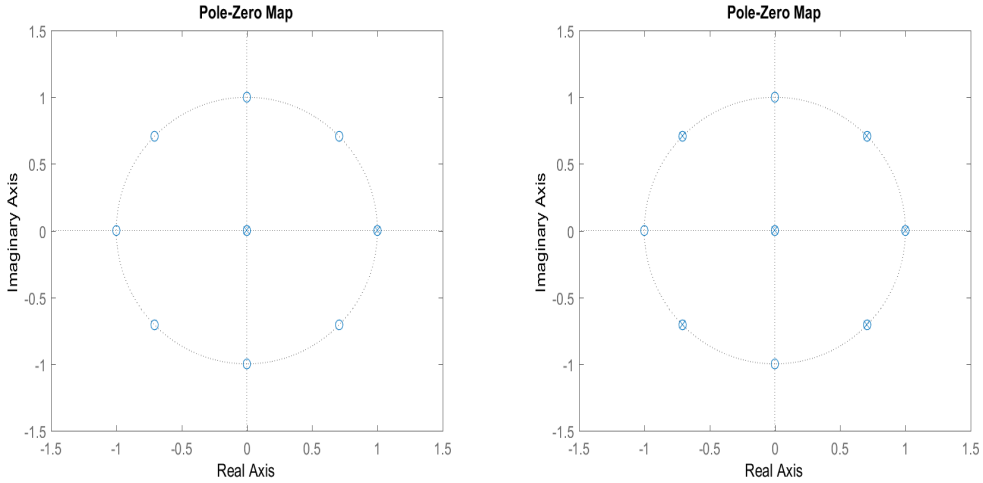
In Figure 3.8 and Figure 3.9, the pole-zero maps for Filter_0 and Filter_1 are depicted. The recursion adds canceling poles and zeros to the system. The optimization introduces $\frac{N}{2k}$ additional pole-zero cancellations. Since these poles are located on the unit circle, the filter is at the stability limit, which must be considered in the implementation.



(a) Pole-zero map of original Filter_0.

(b) Pole-zero map of optimized Filter_0.

Figure 3.8: Comparison of pole-zero maps of original and recursive solution for Filter_0



(a) Pole-zero map of original Filter_1.

(b) Pole-zero map of optimized Filter_1.

Figure 3.9: Comparison of pole-zero maps of original and recursive solution for Filter_1

Accumulator width estimation

In the design of filters at the stability limit there are some critical points to consider. In order to prevent instabilities that would render the filter unusable, the implementation must exactly match the calculated transfer function. This is not always possible, since the implementation of a filter may introduce quantization effects such as rounding inaccuracies, limit cycles or overflows into the system. Since any inaccuracies could move the poles on the unit circle outside of the unit circle, there must be a special focus on preventing rounding operations and overflows. In order to prevent these, the filter was implemented purely using integer arithmetic, avoiding any gain operations that would need rounding inside the filter system. In addition, the width of the accumulators inside the filter system had to be estimated in order to prevent overflows.

The estimation of the accumulator width was done by expanding the input range by the maximum gain of each filter stage. For estimating the maximum gain, a widely used method is the calculation of the DC gain. In the matched filter, however, this method could not be used since the filter contains a zero

at zero frequency, leading to a DC gain of zero. Instead, the 1-norm was used as an estimator for each filter stage. The 1-norm is defined in Equation 3.21:

$$\| h \|_1 = \sum |h[n]| \tag{3.21}$$

Using this norm, the accumulator width can be calculated as:

$$w_{out} = log_2(\| h \|_1) + w_{in} \tag{3.22}$$

For the matched filter set, the maximum DC gain was calculated as 10 bits. Therefore, for a 10-bit input data width, the required accumulator width becomes 20 bits. In order to use 16-bit data in the matched filter, the input data width would have to be reduced to 6 bits.

Implementation

In order to create a computationally efficient filter, a cascaded form consisting of three stages in Transposed Direct Form 2 was implemented in Simulink. Each stage represents one difference equation from the equation sets 3.17 and 3.20. The implementation of the system is shown in Figure 3.10 and 3.11.

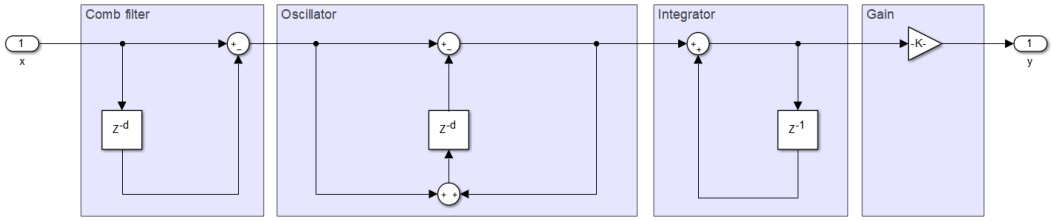


Figure 3.10: Simulink implementation of *Filter_0*.

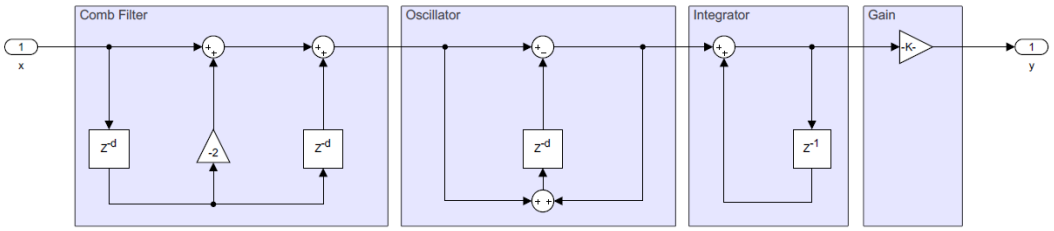


Figure 3.11: Simulink implementation of *Filter_1*.

Gain Factor

The gain of the matched filter plays an important role in the implementation of the filter system. In the original system, the DC gain of the filters was expected to be 4. Therefore, the gain factor M was defined as:

$$M = \frac{N}{4} \tag{3.23}$$

In the differential equations, the gain is implemented as a division. Since division operations are computationally expensive, the gain was implemented as a more efficient bit shift operation. From the gain factor, a bit shift factor m was defined as:

$$m = log_2\left(\frac{1}{M}\right) \tag{3.24}$$

The bit shift operation introduces a rounding into the system. These rounding errors had to be considered in the implementation, which was crucial since the filter system was designed to be operating at the stability limit.

During testing, these errors led to unstable behavior of the output. An example of this behavior is depicted in Figure 3.12. In order to avoid this, the gain element $\frac{1}{M}$ was moved to the output of the system. While this did not eradicate the rounding errors, the stability problems could be removed and the system was deemed to be stable.

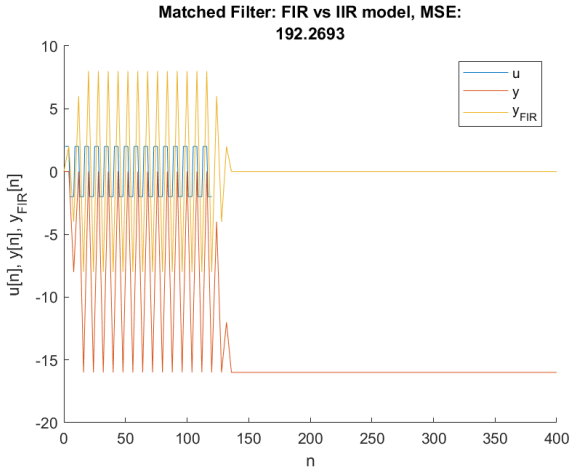


Figure 3.12: System errors due to rounding inside filter structure.

Test Setup

In order to prove the operation of the optimized filter, a test setup was developed. The implementation was compared to a reference model of the original FIR filter. This model was implemented as a Simulink block (depicted in Figure 3.13).

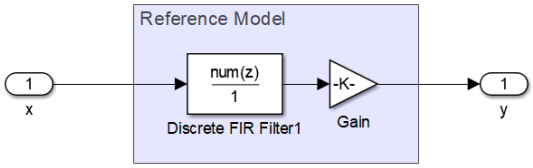


Figure 3.13: Reference model of a matched filter

In addition, a testbench was created to compare the reference model against the implementation (depicted in Figure 3.14). In the testbench, the intermediate signals $x_{C,1}[n]$ and $x_C[n]$ (referenced in the equation sets 3.17 and 3.20) were also passed to the Matlab workspace for further analysis.

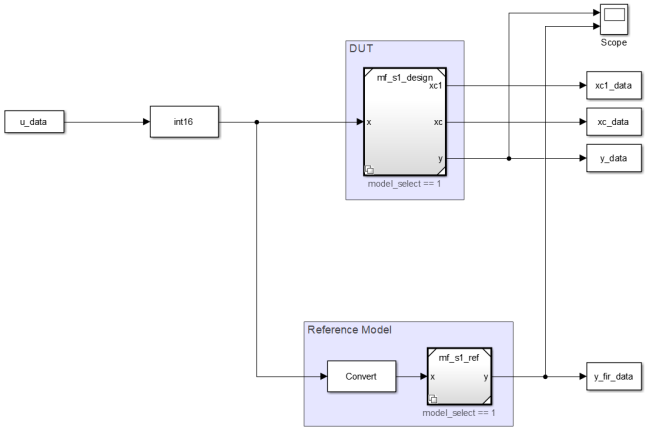


Figure 3.14: Simulink test setup for the matched filters

The inputs for the testbench were created in Matlab. In order to test various conditions, different input signals were applied to the filters. First, a square signal and a BPSK signal were generated as input to test the function of the matched filter. For the matched filters the input data width had to be 12 bit signed, the output data width had to be set to 19 bits. The outputs of the two models were compared and the error between the outputs was calculated. For the error calculation the Mean Squared Error (MSE) was used as metric. The MSE of the implementation is calculated as:

$$MSE_{Filter_x} = \frac{1}{N}\sum_{n=1}^{N}(y - y_{FIR})^2 \tag{3.25}$$

To pass a test the MSE had to be zero over the whole signal.

In addition, the filter was tested against input signals corresponding to the additional pole-zero cancellations. The same tests were applied to the matched filter for symbol 1. These tests passed without errors.

Finally, the filter was tested against overflows by applying a ramp, a noise signal, an impulse and a step function as input signals.

3.2.3 Dual Matched Filter Set

The matched filters Filter_0 and Filter_1 are used in combination to detect both a logic 0 and logic 1 of the input signal. In a BPSK detection chain the filters are executed in parallel, creating two output signals for each symbol (as depicted in Figure 2.8). Since both filters partially show the same characteristics, it is possible to share some of the arithmetic operations for both filters. For instance, the FIR impulse response of Filter_0 and Filter_1 share the first 16 taps of the filter. In a pure FIR implementation these 16 taps could be shared by both filters. Due to the linearity of the filters, the common part of the impulse response can be extracted:

$$h_0[n] = h_{Common}[n] + h_{Filter0}[n]$$
$$h_1[n] = h_{Common}[n] + h_{Filter1}[n]$$

The common part of the filter structure can then be implemented in a separate structure, as depicted in Figure 3.15.

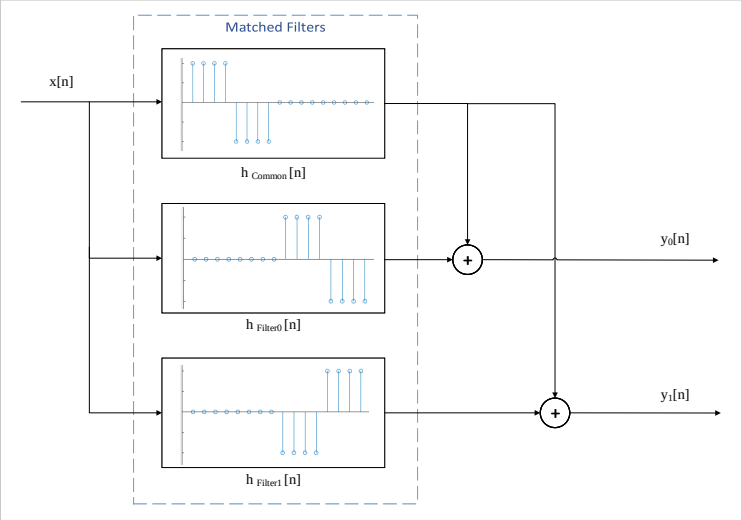


Figure 3.15: Combination of a matched filter system for Filter_0 and Filter_1

The advantage of such a dual structure is the reduced computation cost due to the sharing of the first half of the impulse response by both filters.

The zero samples in the impulse responses h_{Common}, $h_{Filter0}$ and $h_{Filter1}$ do not have to be computed. Therefore, the number of taps in the whole filter system is reduced by 25 %.

The dual structure of the matched filters can be implemented by a pure FIR filter set and also by a recursive filter set. A Simulink implementation of these filter sets is depicted in Figure 3.16 and Figure 3.17.

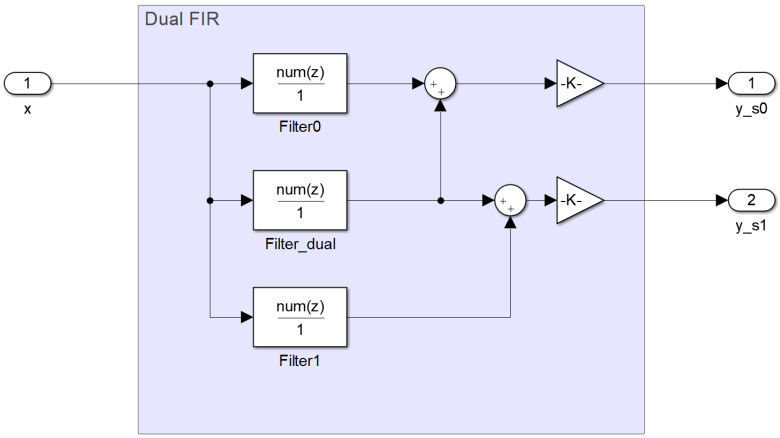


Figure 3.16: Simulink implementation of a dual system for a pure FIR filter set

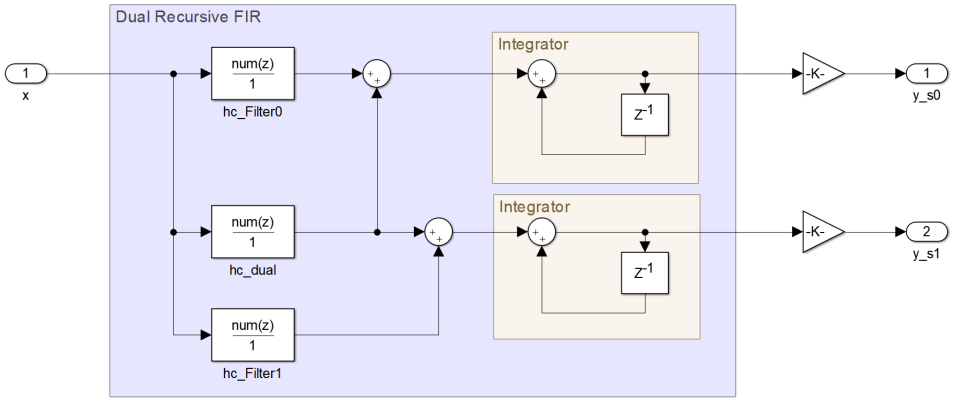


Figure 3.17: Simulink implementation of a dual system for a single recursion filter set

4 Implementation

This chapter covers the implementation of the receiver algorithm on a DSP development kit. The implementation was proposed to be done on the TI kit. This development kit was selected due to its C6713 processor, which is one of state-of-the art DSPs and its compatibility with Matlab/Simulink.

The implementation was based on the Simulink models that were created during the design and optimization of the matched filter algorithm. These models were generated as C code and executed on the development kit. In addition, in order to measure the performance of the algorithm, a test setup was created in Simulink.

4.1 DSK6713 Development Kit

The platform for the implementation was selected to be the DSP Starter Kit (DSK) 6713 development kit. This kit was developed by TI in cooperation with Spectrum Digital and can be used for evaluating the C6000 processor family [18]. The main component of the DSK6713 is the TMS320C6713 floating-point DSP, which can be programmed and debugged via a Joint Test Action Group (JTAG) interface. The memory of the DSP is extended by 16 MB of RAM and 512 kB of flash memory. In addition, there is an AIC23 audio codec embedded on the board, which can be controlled by the DSP. These components are depicted in the block diagram Figure 4.1. The kit can be connected to a PC per USB (on-board JTAG emulator), and the power supply is 5V.

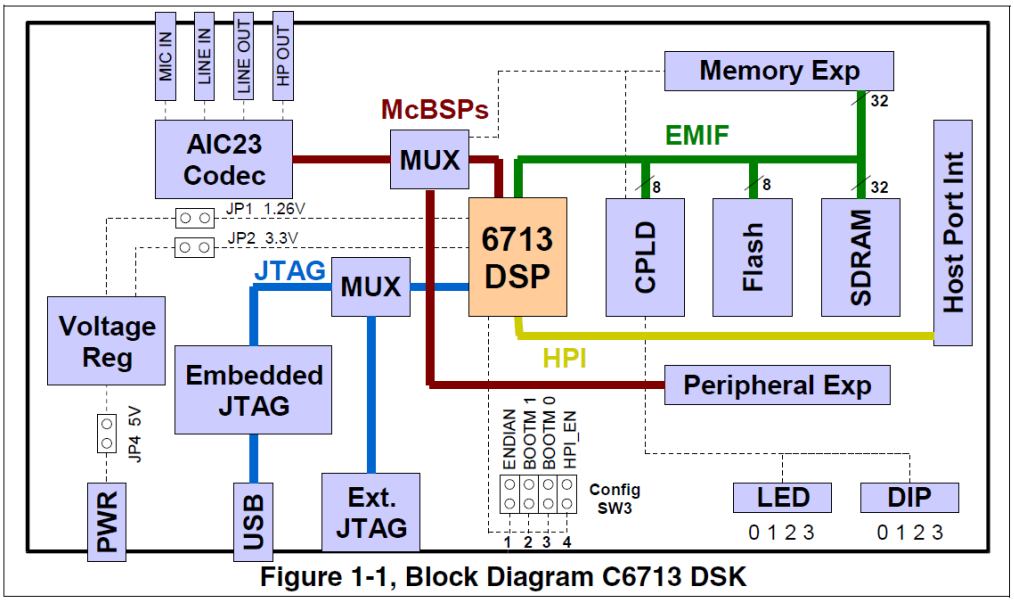


Figure 4.1: Schematic overview of the DSK6713 development kit, cf. 'Block Diagram C6713 DSK', DSK6713 Technical Reference [18]

For software development on the DSK6713, TI provides the Code Composer Studio software. This software includes a full toolchain for the kit, including a compiler, optimization tools, and a debug suite. In addition, the DSK6713 kit is supported by the MathWorks Embedded Coder Support Package for Matlab/Simulink. This package enables C code generation from Matlab or Simulink for the DSK6713. In addition, the support package offers the IDELink function, which allows to include the kit into the simulation flow.

4.1.1 C6713 DSP

The main component of the DSK6713 is the C6713 DSP core, which contains a Very Long Instruction Word (VLIW) processor architecture. The core is powered by a supply of 1.2V and operates at a maximum clock frequency of 225 MHz. An overview of the C6713 is depicted in Figure 4.2.

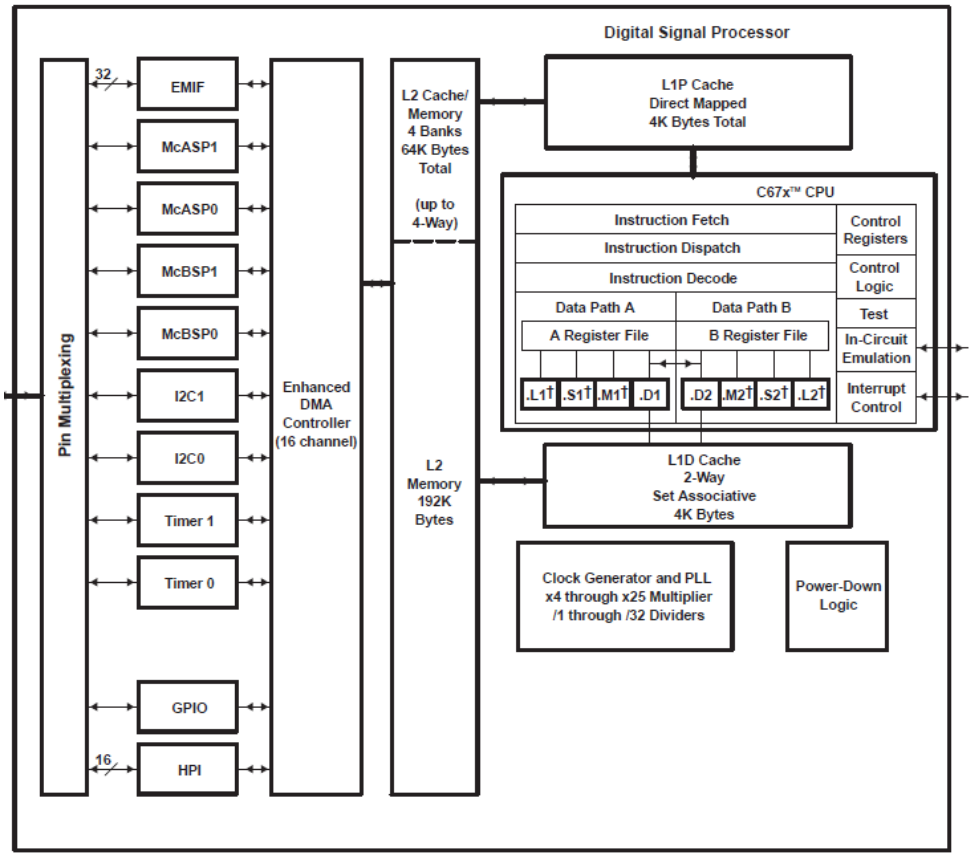


Figure 4.2: C6713 Block Diagram

The main component of the C6713 is the CPU, which contains eight functional units (shown in Figure 4.3). The architecture is separated into two datapaths, which each contain one MAC (.Mx), Load/Store (.Dx), and two ALUs (.Sx) and (.Lx). The .Sx ALU supports 16-bit and 32-bit instructions, the .Lx ALU is mainly used for 32-bit instructions. The .Mx, .Lx, and .Sx ALUs support floating-point and fixed-point instructions. Corresponding to each datapath there is a set of 32 CPU registers each, named Ax and Bx. When these units are used in parallel, each unit can access its own register set for input and output. In addition, there are two crosspaths available, which can be used to access registers from the opposite set.

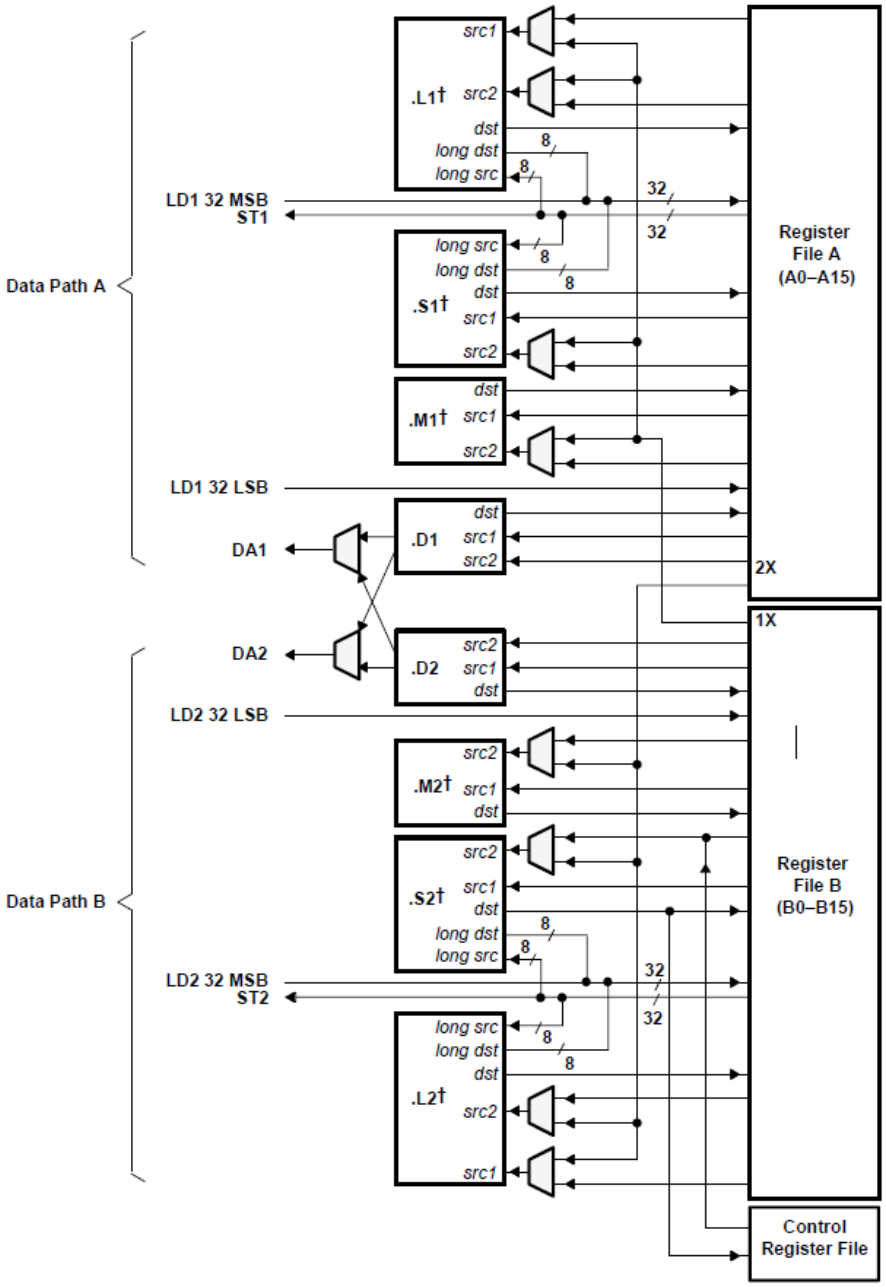


Figure 4.3: C6713 Functional Units

The C6713 is programmed in the languages C and Assembly. In addition, the two languages can be combined by using linear assembly, which can be embedded into C code.

During programming, when writing C code or linear assembly code, the toolchain distributes the instructions among the functional units. In contrast, when pure assembly language is used, each of the instructions can be assigned to a dedicated functional unit manually. Then the programmer is responsible for distribution of instructions and optimization or parallelization.

Memory

The memory of the DSK6713 is split up into internal memory, external RAM, and external flash memory. The C6713 contains 256k of internal memory, which consists of L1 cache and L2 cache.

Each of the memory modules can be accessed as a certain address region. The address layout of the C6713 processor is depicted in Figure 4.4.

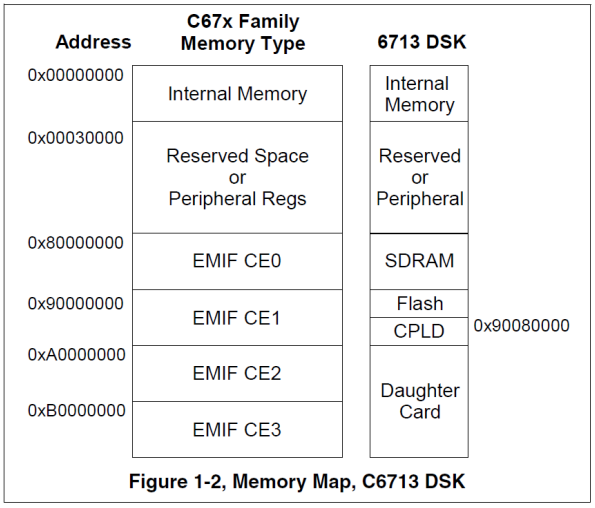


Figure 4.4: DSK6713 Memory Map

4.1.2 Code Composer Studio

Code Composer Studio (CCS) is a collection of development tools created by TI which is specifically designed to support embedded system development in C/C++ and Assembly. The suite contains an Integrated Development Environment (IDE), which is based on the Eclipse C/C++ Development Tooling (CDT), and a toolchain, which is compatible with several microcontroller and DSP products of TI. The toolchain includes code generation tools, on-chip debugging tools and performance measurement tools. The CCS versions 3 to 5 support the DSK6713 development kit, the latest version of this software is CCSv9. [11]

4.1.3 TI Toolchain - Code Generation Tools

The compiler, linker and assembler for the DSK6713 are included in a package called the TI Code Generation Tools (CGT), which are designed for the processors of the C6000 series. This section refers to the overview of George Mock in the TI Training series [11].

Compiler and Linker

The C6000 compiler offers various optimization methods for improving the execution efficiency of the code on assembly instruction level. There are four levels of optimization, which can each be set by a compiler flag. The lowest level of optimization, -o0, only covers statement-wide optimization. The levels -o1 and -o2 cover block- and function-wise optimization and level -o3 optimizes the code over the range of the whole file. The standard level of optimization is set to -o2. For the optimization levels of -o2 and -o3 advanced features such as software pipelining are enabled in the compiler. These optimizations may greatly reduce the number of instructions required for execution, however they may negatively affect the debugging of the code. Therefore, it is recommended by TI to disable the optimizations while debugging, and then re-compile the code with optimizations.

In addition to the compiler, the linker also offers settings that can be configured, most importantly the memory layout. For defining this, the linker accepts a command file. It is important to define the entry points for stack and heap, since these are the sections that are accessed most often. If possible, these sections should be set into the fastest accessible memory regions.

The compiler and linker generate object files, which are specified by the Common Object File Format (COFF). A COFF File consists of file headers, code sections, relocation information, a symbol table and a string table. The COFF format is no longer supported in newer versions of the CGT, from CGT 7.2 upwards the EABI file format is used.

Chip Support Library

The TI Chip Support Library (CSL) for the C6000 processors provides an application programming interface for configuring and controlling the DSP on-chip peripherals. The C6000 CSL includes access functions to the serial ports, the memory interfaces, the general-purpose I/O and many others. The library supports the processors C6201-C6713.

Board Support Library

In addition to the CSL, TI and Spectrum Digital provide a Board Support Library (BSL) for the DSK6713 development kit. This library contains functions that are designed to access the modules of the DSK6713 that are not covered by the CSL. The BSL provides access to the switches and LEDs of the development kit, the flash memory and the AIC23 audio codec.

4.1.4 Configuration and Connection

In order to connect the Matlab and CCS software to the DSK6713 board, several steps must be taken. Firstly, CCS must be configured to use the correct toolchain settings for the DSK6713. The main part of the setup is the

configuration of the debug emulator. When CCS is connected to the development kit, diagnosis functions are provided to ensure the functionality of the kit. Afterwards, the link to Matlab can be established.

Debug Emulator

According to Spectrum Digital [18], the connection to the DSK6713 is established through the Spectrum Digital DSK-EVM-eZdsp onboard USB emulator. This emulator enables communication through an Universal Serial Bus (USB) connection by emulating the JTAG protocol.

GEL files

The CCS debugger is configured by scripts in General Extension Language (GEL), a C-like language. The main purpose of GEL files is to act as an initialization script for the hardware. This includes configuring peripherals of the hardware, such as interrupt settings and cache initialization. Another important function at startup is to create a memory map for the debugger to define the accessible memory regions. In addition, the setup can include a self-test of the peripherals. Further features include runtime debugging options, such as access to variables at runtime. [2]

4.1.5 Simulink Interface

Matlab provides an interface between the CCS IDE and the Simulink simulator, which is called IDELink. The IDELink makes it possible to connect the Simulink software to embedded system development IDEs, for instance, to the TI CCS and the Analog Devices VisualDSP++. The Simulink simulation is linked to the IDE, which is then connected to the target hardware. IDE Link supports debugging on target hardware and for certain targets processor-in-loop simulation. For TI products, the CCS versions 3.3 to 5 are supported.

4.1.6 Processor-in-Loop Simulation

In the design of signal processing algorithms, the algorithms are first modeled in a simulation tool, and then implemented on a targeted hardware. This process can be improved by Processor-in-Loop (PIL) simulation, which includes the targeted hardware in the simulation flow.

In this method the simulation model is first converted into code, then executed on the target hardware. The hardware provides data to the simulation tool through an interface. PIL is widely used in control theory to simulate models of different controllers on a real-time platform.

4.2 Test setup

In the implementation, a PIL setup was used to evaluate the performance of the receiver algorithm on the DSK6713. The purpose of the implementation was to compare different filter structures in sense of execution time and also to find possible optimizations. In addition, different settings for compiler optimization and parallelization of the DSP were evaluated.

4.2.1 PIL Setup

In order to set up the PIL simulation, the algorithms had to be modeled in Simulink, then translated into C code and embedded into the simulation as a PIL block. This workflow was first executed for the matched filter, and then adapted for the whole signal processing chain.

During the setup, the toolchain and the software requirements for the system had to be analyzed. The DSK6713 offers different methods of integration into a Simulink environment in combination with Code Composer Studio.

The DSK6713 is supported by the Code Composer Studio versions 3 to 5. While version 3 offers the full integration of the DSK6713 into Simulink, versions 4 and 5 only offer reduced compatibility. The generation of C code and executables is still supported by using makefiles, the PIL setup

is no longer possible. An overview of the different methods and their requirements is depicted in Figure 4.5.

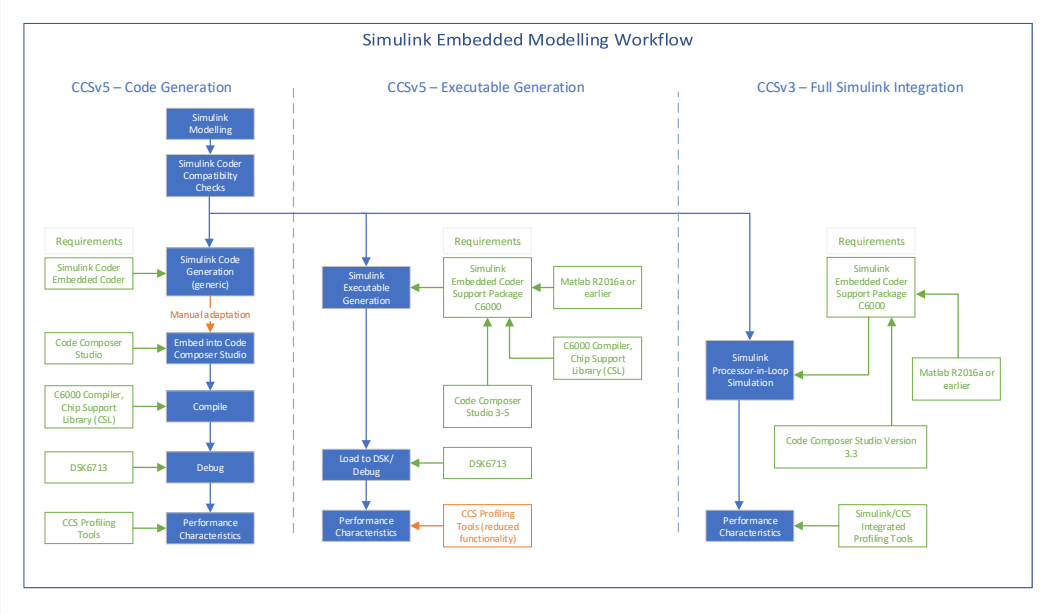


Figure 4.5: Overview of different Simulink integration methods and their software requirements

The first method that was evaluated during the setup was the pure C code generation in Simulink and the compilation and execution in Code Composer Studio 5. However, this method had the drawback that a manual adaptation of the C code was necessary to compile the code in CCSv5. Therefore, the next step was to move the executable generation to Simulink and only execute the generated binary from CCSv5. This method created strong limitations in the profiling of the generated code.

Finally, a full PIL setup was implemented in the CCSv3 software. For this, an Embedded Coder Support package for C6000 processors was required. In order to use the Embedded Coder Support package with CCSv3, the following tools (depicted in Figure 4.6) had to be installed:

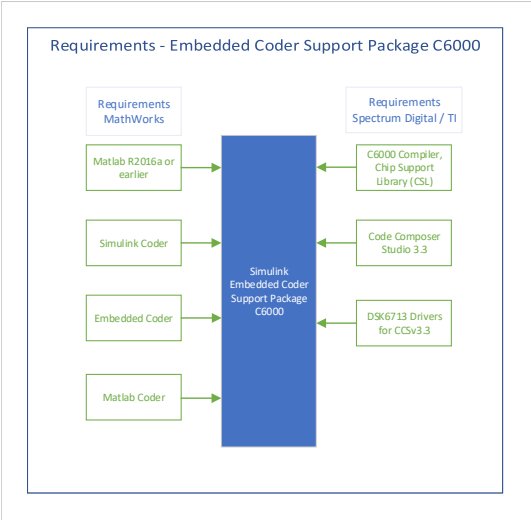


Figure 4.6: Simulink Embedded Coder requirements

After the setup of the DSK6713 in combination with the Embedded Coder Support Package, the signal processing algorithms of the NFC receiver could be implemented on the DSK6713.

4.2.2 Matched Filter

In the original model of the NFC receiver, the main bottleneck in terms of performance was the matched filter. Therefore, the implementation was first focused on improving the matched filter algorithm and evaluating its performance. For this purpose, different topologies of the matched filter were modeled in Simulink and evaluated in a PIL simulation. In addition, different compilation methods from the DSK6713 code generation framework were evaluated in order to optimize the performance.

In Simulink, the PIL simulation is implemented by firstly generating code, and then embedding the generated code into a testbench in form of a PIL block. An instance of a generated PIL block is depicted in Figure 4.7.

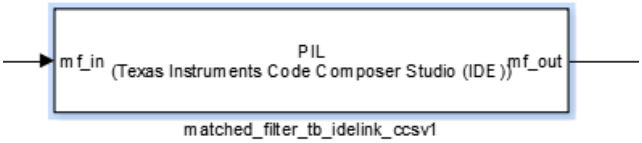


Figure 4.7: Matched Filter as generated PIL block

In order to check whether the PIL block still produced the expected output, a test setup was implemented in Simulink. An overview of the test setup is depicted in Figure 4.8.

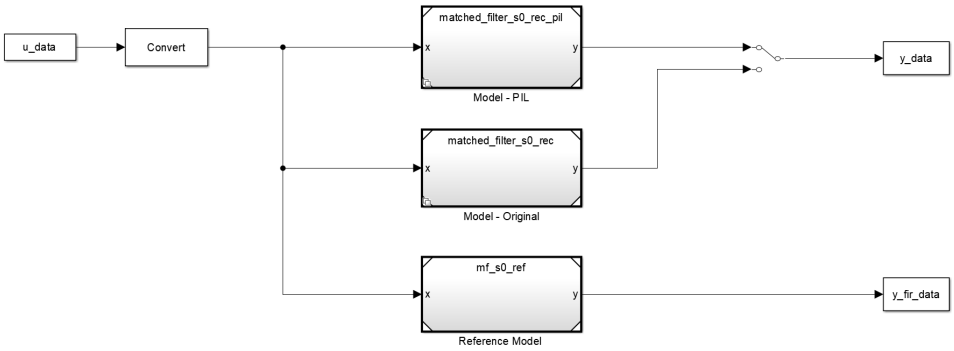


Figure 4.8: PIL Simulation Testbench for a single matched filter

In the test setup, different PIL blocks can be selected as a referenced model. The module under test can be compared to an FIR filter reference implementation. Additionally, a Simulink implementation of the generated PIL block was added to the testbench for debug purposes.

For the evaluation of a full matched filter set, a second testbench was created. This setup is depicted in Figure 4.9.

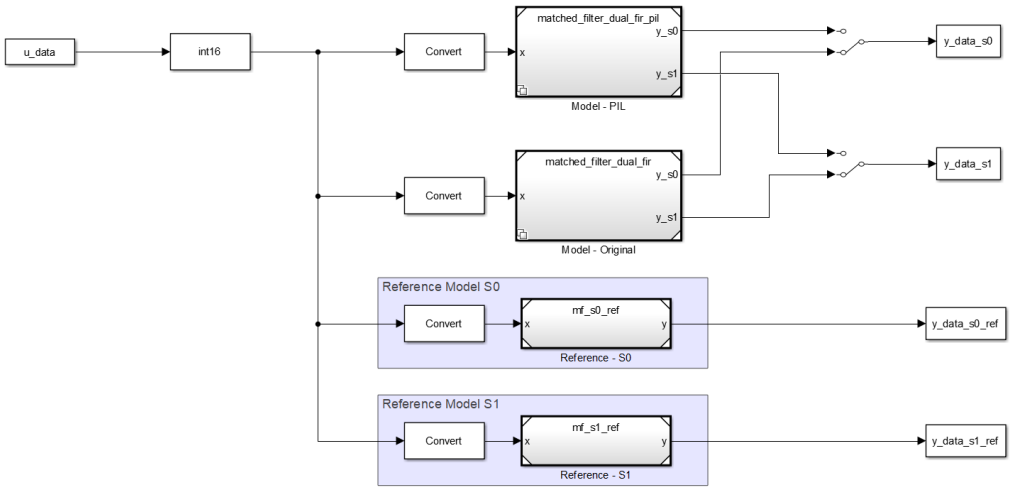


Figure 4.9: PIL Simulation Testbench for a dual matched filter set

In this setup, the performance of different combinations of the two matched filters should be analyzed. Similar to the testbench for a single matched filter, the testbench contains the PIL blocks for different implementations as referenced model variants. Each matched filter is compared to a reference model.

5 Results and Conclusion

The implementation of the matched filter was tested for various topologies and implementations. After testing the functionality and stability of the different topologies, the performance of each filter was analyzed. The performance was first estimated from the design of the filters, and then measured form the implementation. In this chapter, the results of the performance analysis are presented. Additionally, an outlook into further work is given.

5.1 Results

The matched filter was optimized and then implemented on the DSP platform. The implementation was first tested to grant its functionality and then analyzed in terms of performance. For this purpose, a first estimation of the performance was developed during the design phase. Afterwards, this estimation was compared to the real measurement results. The tests were done for a set of filters with the following parameters (Table 5.1 and 5.2):

Filter_x	N	$\frac{1}{M}$	k
0	16	4	4
0	32	8	8
0	64	16	8
0	64	16	4
0	64	16	2
0	32	8	4
0	32	8	16

Table 5.1: Parameters of Filter_0 used in the estimation of computational effort.

Filter_x	N	$\frac{1}{M}$	k
1	16	4	4
1	32	8	8
1	64	16	8
1	64	16	4
1	64	16	2

Table 5.2: Parameters of Filter_1 used in the estimation of computational effort.

5.1.1 Performance Estimation

The first results that are presented in this work are the results of a performance estimation. This estimation was done mainly to achive a first overview of the performance of the different topologies and to check whether an optimization would increase the filter performance at all.

Estimation - DSK6713 Modelling

In order to estimate the performance, a model of the DSK6713 development kit and its architecture was created. The development kit implements the matched filter as C code and executes this code on the DSP core. The performance of this execution can be defined by the clock cycles that are necessary to execute the filtering operations. For this, the number of instructions and the number of cycles per instruction are important factors.

For the execution of a single instruction on the DSP, the cycle times vary depending on the instruction type. The cycles per instruction for the operation used in the matched filters are listed below (referencing the TI TMS320C6713 Assembly Instruction Set [19]):

$$\tau\,[cycles] = \left\{ \begin{array}{cc} 1 & addition/subtraction \\ 1 & gain/shift \\ 2 & multiplication \\ 5 & load/store \end{array} \right\}$$

First Estimation - Single Matched Filter

With the help of the DSK6713 model, a first estimation was calculated. In the first estimation each filter was evaluated separately. This was done due to the differences in the topologies of Filter_0 and Filter_1. Figure 5.1 and 5.2 depict the results of this estimation for Filter_0 and Filter_1.

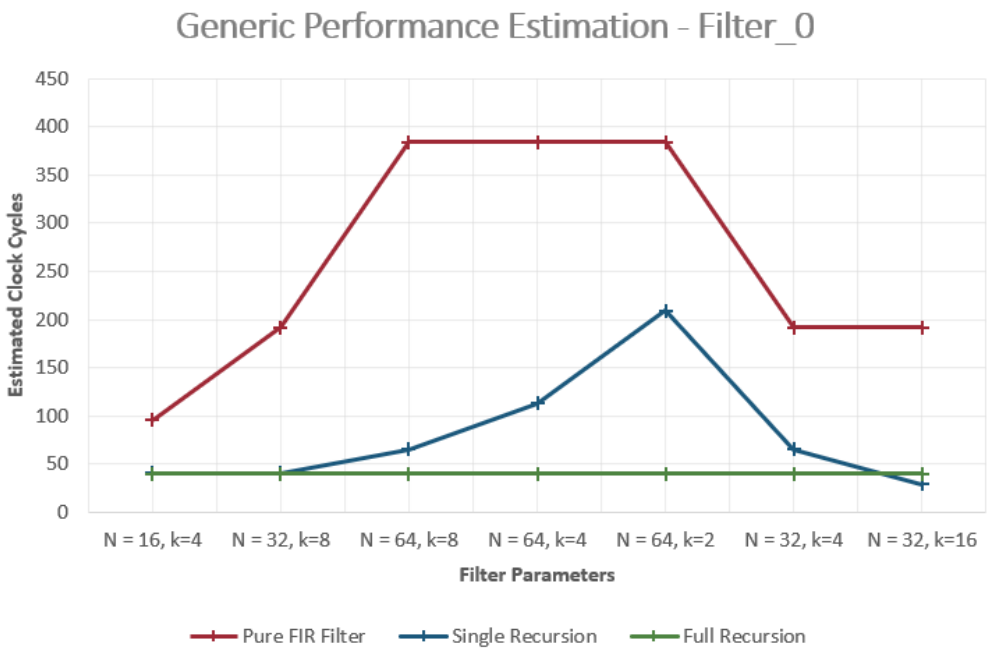


Figure 5.1: Performance estimation for Filter_0 on a generic platform

The estimation shows that the performance of the pure FIR implementation mainly depends on the number of taps N for both filters. The performance of the single recursion filter is better than the pure FIR filter in any case. The fully recursive filter has a constant performance for each parameter set.

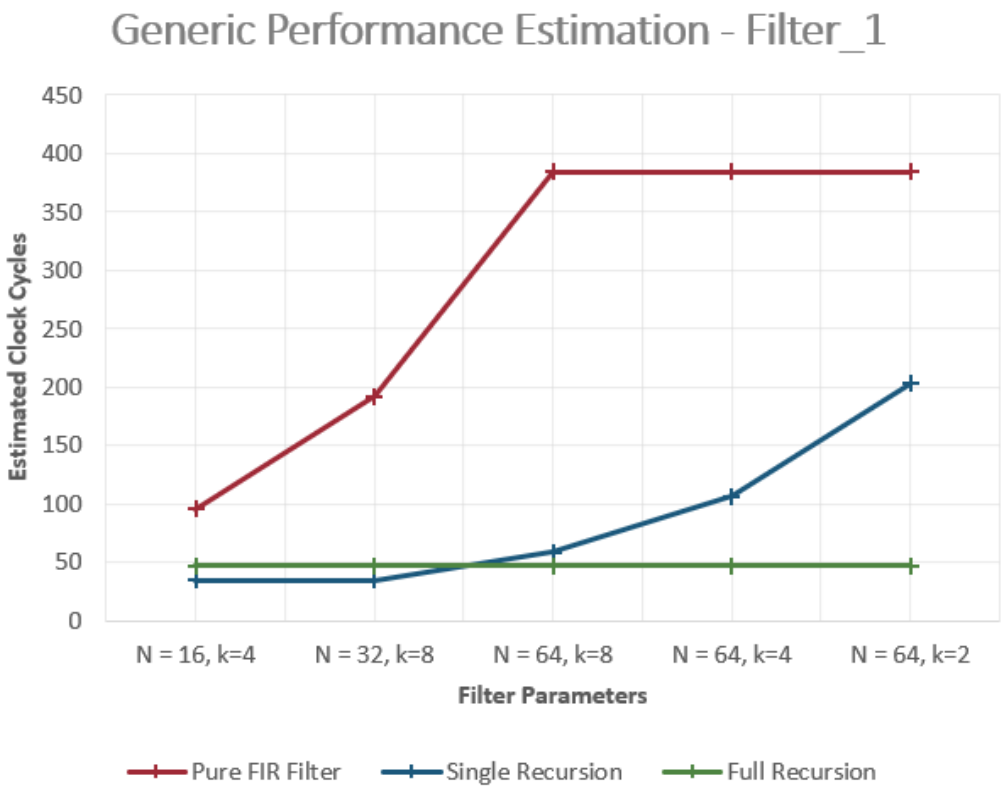


Figure 5.2: Performance estimation for Filter_1 on a generic platform

When comparing the results for Filter_0 and Filter_1 for equal parameter sets, it can be seen that there are no significant differences in performance between the two filter topologies.

Estimation - Full Matched Filter Set

Afterwards, the performance of a full filter set was estimated. In addition to the standard implementation, a dual filter set was evaluated. The result of this evaluation is depicted in Figure 5.3):

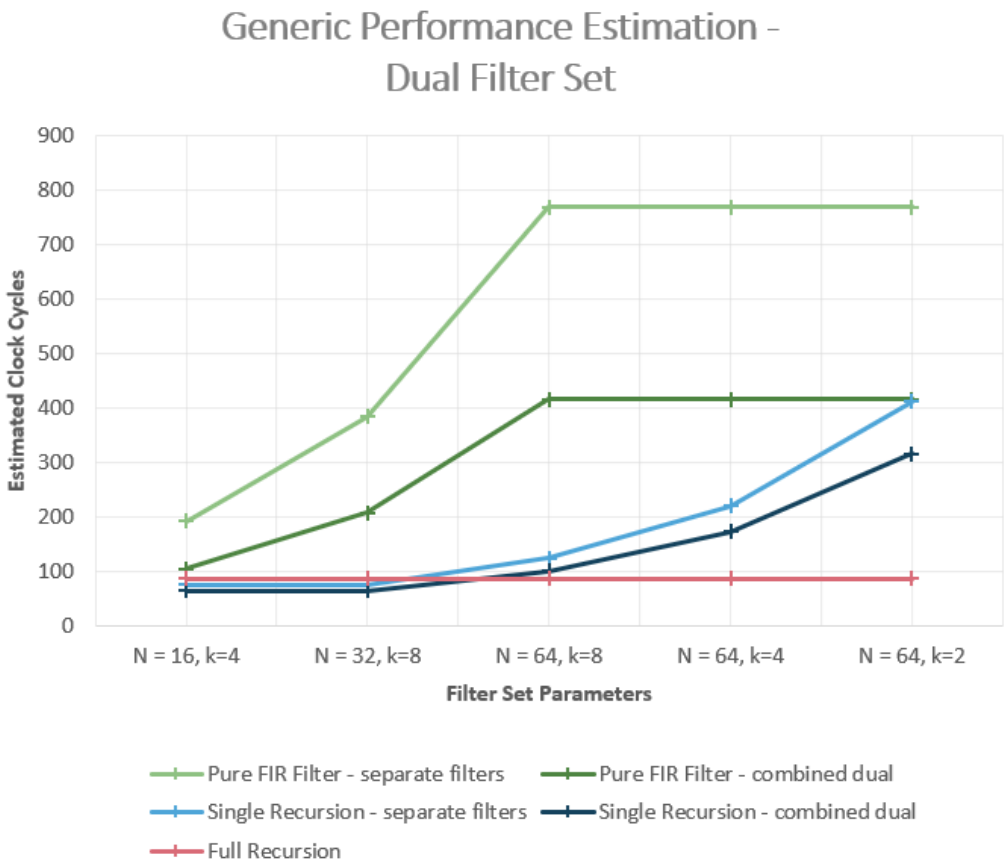


Figure 5.3: Performance estimation for a full filter set on a generic platform

The combination of two filter sets into a dual filter system shows a significant reduction for the pure FIR filter, for the single recursion there is still an improvement. The fully recursive system was not combined due to the nature of the topology.

Estimation Conclusions

The estimation states that the pure FIR filter implementation is significantly more computationally expensive than the other implementations. The single

recursive filter is more efficient for filters with up to $N = 16$ taps. For filters with a larger number of taps the fully recursive filter is more efficient. Another benefit of the fully recursive filter is the constant execution time regardless of the parameters N and k. In all topologies, the combined versions of the filter set show a drastically better performance than their counterparts without combination.

5.1.2 Measurement Results

The performance estimation was used as a basic starting point for the implementation and the performance measurement. In the implementation, different ways of optimization were evaluated to improve the performance.

Single Matched Filter - Evaluation of different compilation options

Firstly, a single matched filter was evaluated. In the Simulink workflow, there were options to improve the performance by compiler optimization. For this, the three optimization stages $-o1, -o2$ and $-o3$ were evaluated. In addition, an evaluation of the performance between a floating-point implementation and a fixed-point/integer implementation was done. The results of the performance evaluation are depicted in Figure 5.4.

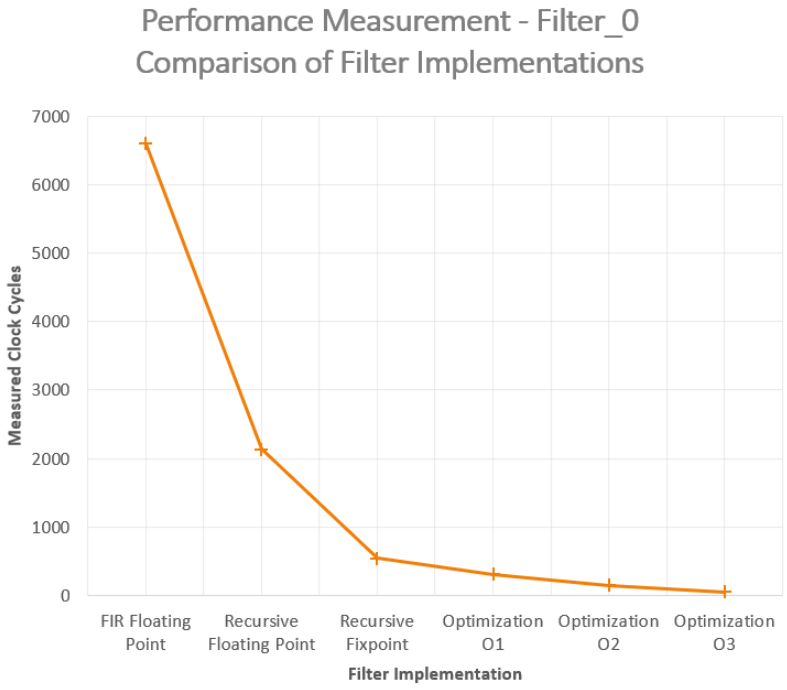


Figure 5.4: Performance measurement results for Filter 0 on the DSK6713

The starting point of the performance measurement was the reference FIR filter model (Figure 3.13). The optimization of the filter topology brings a significant reduction of computation time. A further increase is shown by replacing floating-point operations by fixed-point operations. The use of compiler optimization gives a further increase in performance, showing that the performance reaches the estimated value of clock cycles.

Matched Filter - Full Set

In addition to the measurement for a single filter, a full filter set was analyzed. In this analysis, a reference model of both filters was used as a starting point. Then, different topologies and combination methods were evaluated. The filters were implemented as fixed-point version, compiler

optimization was omitted. The results of the evaluation are depicted in Figure 5.5.

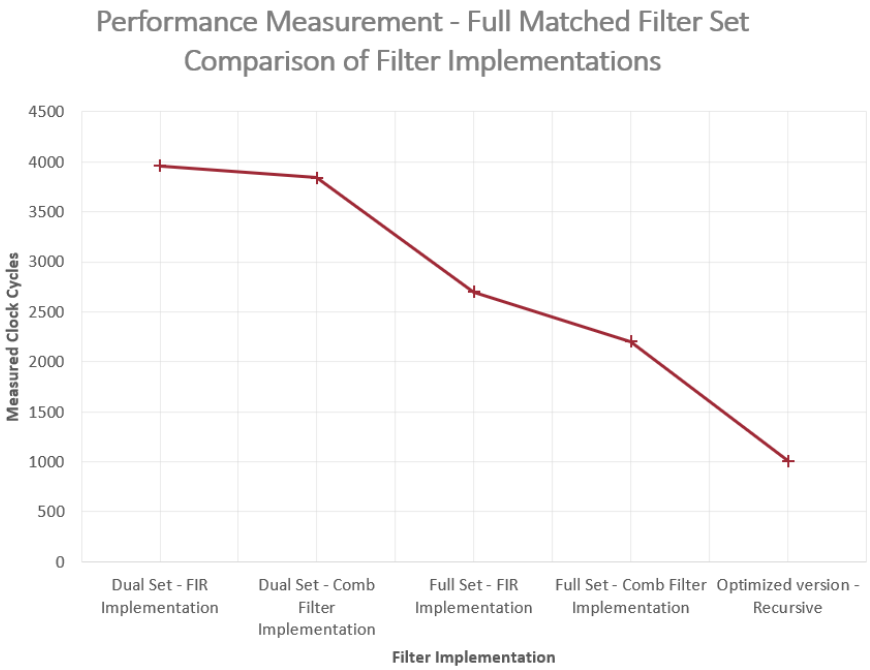


Figure 5.5: Performance measurement results for a full filter set on the DSK6713

The results show that the recursive FIR version of the full set performs better than the original FIR filters. However, the combination of the filters into a dual set, does not produce the expected results of a performance improvement. This may be due to the implementation of the full filter set in Simulink, which can be improved further. In a full set, the optimized version of filters performs significantly better than all other implementations. Therefore, the recommendation of this work is to use the optimized matched filter topology.

5.2 Outlook

The outcomes of this work offer several options for further work and improvement. The design of this work resulted in an optimized topology for the matched filter and a design test setup for the DSK6713 development kit.

The matched filter was optimized and implemented as a recursive filter topology. The structure of the filter still offers possibilities for improvements, such as a poly-phase implementation of the topology, an integrate-and-dump implementation of the matched filter or a reduction of the sampling rate to improve the performance. Further improvements can be evaluated and compared against the existing solution.

While in this work the main purpose was to improve the matched filter topology and to evaluate the performance of the matched filter, the methods of this work can be used for more components of the existing signal processing chain, such as the channel combiner, the symbol synchronizer or the averaging filter. These filters can be modeled in Simulink and executed on the DSP kit, giving information about the performance of different filter topologies.

In conclusion, the implementation of the filter on the development kit showed an efficient way for generating C code from Simulink models. Therefore it is possible to generate C code for different platforms. A smaller DSP core such as the NXP Coolflux could be embedded into an ASIC alongside the existing Verilog implementation. This core could then take over the more complex signal processing operations of the filtering chain, reducing chip area and power consumption of the ASIC.

Appendix

Bibliography

[1] *7 Series DSP48E1 Slice User Guide*. Tech. rep. Xilinx Inc., 2018 (cit. on p. 27).

[2] Alan Campbell et al. *Creating Device Initialization GEL Files*. Tech. rep. Texas Instruments, 2004 (cit. on p. 59).

[3] Travis F. Collins and Robin Getz. *Software-Defined Radio for Engineers*. Artech House, 2018. ISBN: 978-1-63081-457-1. (Cit. on p. 24).

[4] Edmund Ehrlich. "Digital Demodulator Architecture of a Contactless Reader System for HF RFID Applications Supporting Data Rates up to 13.56 Mbit/sec." PhD thesis. Graz University of Technology, 2011 (cit. on p. 7).

[5] U. Farooq, Z. Marrakchi, and H. Mehrez. *Tree-based Heterogeneous FPGA Architectures - Application Specific Exploration and Optimization*. Springer, 2012 (cit. on p. 25).

[6] Klaus Finkenzeller. *RFID-Handbuch: Grundlagen und praktische Anwendungen von Transpondern, kontaktlosen Chipkarten und NFC*. 7th ed. München: Hanser, 2015. ISBN: 978-3-446-43943-6 (cit. on pp. 3, 7).

[7] F. Hassan and S. Khorbotly. "Recursive implementation of exponential linear phase FIR filters." In: *2011 18th IEEE International Conference on Electronics, Circuits, and Systems*. Dec. 2011, pp. 559–562 (cit. on pp. 22, 39).

[8] E. Hogenauer. "An economical class of digital filters for decimation and interpolation." In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 29.2 (Apr. 1981), pp. 155–162. ISSN: 0096-3518 (cit. on p. 22).

[9] *International Standard ISO/IEC 14443*. ISO/IEC, 2016 (cit. on p. 5).

[10] Richard G. Lyons. *Understanding Digital Signal Processing*. Pearson, 2011 (cit. on pp. 17, 19).

[11] George Mock. *C Compiler Tips and Tricks. TI Training*. Texas Instruments Inc. Mar. 2015. URL: `training.ti.com/c-compiler-tips-tricks` (cit. on p. 57).

[12] Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck. *Discrete-time Signal Processing (2Nd Ed.)* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1999. ISBN: 0-13-754920-2 (cit. on p. 17).

[13] *PN512 Full NFC Forum-compliant frontend*. Tech. rep. NXP Semiconductors, June 2016 (cit. on p. 1).

[14] John G. Proakis and Masoud Salehi. *Communication Systems Engineering*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994. ISBN: 0-13-158932-6 (cit. on pp. 3, 11–13, 16).

[15] P.N. Ravichandran et al. "Design and Implementation of Early-Late Gate Bit Synchronizer for Satellite Communication." In: *National Conference on Communications*. Jan. 2009, pp. 15–18 (cit. on p. 16).

[16] *ST25R3912 ST25R3913 High performance HF reader / NFC initiator for payment applications with 1 W output power*. STMicroelectronics. Aug. 2018 (cit. on p. 1).

[17] Andrew E. Stevens. "An Integrate-and-Dump Receiver for Fiber Optic Networks." PhD thesis. Columbia University, 1995 (cit. on pp. 14, 16, 17).

[18] *TMS320C6713 DSK Technical Reference*. Tech. rep. Spectrum Digital Inc., 2003 (cit. on pp. 52, 53, 59).

[19] *TMS320C67x/C67x+ DSP CPU and Instruction Set Referencre Guide*. Tech. rep. Texas Instruments Inc., 2006 (cit. on p. 66).

[20] *TN1216 Technical Note, ST25 NFC guide*. Tech. rep. STMicroelectronics, Oct. 2016 (cit. on p. 6).

[21] *Ultra Low Power CoolFlux DSP*. Tech. rep. NXP Semiconductors Inc., 2015-2018 (cit. on p. 31).

[22] O. Vainio, M. Renfors, and T. Saramaki. "Recursive implementation of FIR differentiators with optimum noise attenuation." In: *Quality Measurement: The Indispensable Bridge between Theory and Reality (No Measurements? No Science! Joint Conference - 1996: IEEE Instrumentation and Measurement Technology Conference and IMEKO Tec*. Vol. 1. June 1996, 344–349 vol.1 (cit. on p. 22).

[23] Paul Yin. *Introduction to TMS320C6000 DSP Optimization*. Tech. rep. Texas Instruments Inc., 2011 (cit. on p. 30).

[24] *Zynq-7000 SoC Data Sheet: Overview*. Tech. rep. Xilinx Inc., 2018 (cit. on p. 26).

List of Acronyms

RF	Radio Frequency
RFID	Radio Frequency Identification Technology
LF	Low Frequency
HF	High Frequency
UHF	Ultra High Frequency
ISO	International Standardisation Organisation/International Electrotechnical Commission
JIS	Japanese Industrial Standards
EPC	Electronic Product Code
NFC	Near-Field Communication
UID	Unique Identifier
I-Q	In-Phase-Quadrature
ASK	Amplitude Shift Keying
BPSK	Binary Phase Shift Keying
OOK	On-Off Keying
AWGN	Additive White Gaussian Noise
SNR	Signal-to-Noise Ratio
RM	Reader Mode
CM	Card Mode

IC	Integrated Circuit
DC	Direct Current
AC	Alternating Current
PLL	Phase-Locked Loop
MAC	Multiply-Accumulate
FFT	Fast Fourier Transformation
IIR	Infinite Impulse Response
FIR	Finite Impulse Response
MSE	Mean Squared Error
CIC	Cascaded Integrator Comb
CPU	Central Processing Unit
USB	Universal Serial Bus
ALU	Arithmetic-Logic Unit
DSP	Digital Signal Processor
ASIC	Application-specific Integrated Circuit
FPGA	Field Programmable Gate Array
SoC	System-on-Chip
HDL	Hardware Description Language
VLIW	Very Long Instruction Word
TI	Texas Instruments
CCS	Code Composer Studio
IDE	Integrated Development Environment
DSK	DSP Starter Kit
CDT	C/C++ Development Tooling
GEL	General Extension Language

CSL Chip Support Library

BSL Board Support Library

COFF Common Object File Format

CGT Code Generation Tools

JTAG Joint Test Action Group

PIL Processor-in-Loop