Doctoral Thesis

VLSI architectures for compressive sensing and sparse signal recovery

Author(s):
Mächler, Patrick

Publication Date:
2012

Permanent Link:
https://doi.org/10.3929/ethz-a-007619661

Rights / License:
In Copyright - Non-Commercial Use Permitted
VLSI Architectures for Compressive Sensing and Sparse Signal Recovery

A dissertation submitted to

ETH ZURICH

for the degree of

Doctor of Sciences

presented by

PATRICK MÄCHLER
Dipl. El.-Ing. ETH
born September 11th, 1984
citizen of Erlenbach ZH, Switzerland

accepted on the recommendation of

Prof. Dr. Hubert Kaeslin, examiner
Prof. Dr. Helmut Bölcskei, co-examiner
Prof. Dr. Andreas Burg, co-examiner

2012
Acknowledgments

First, I would like to thank Prof. Hubert Kaeslin for readily stepping in as my supervisor, supporting my research, and guiding me through my thesis. I am also very grateful for the advice and the many interesting discussions with Prof. Andreas Burg. Without his visions and insights, this thesis would not have been possible. Further, I would like to thank Prof. Helmut Bölskei for acting as a co-examiner of my thesis. A special thank goes to Christoph Studer for his inputs and the fruitful collaboration. I very much appreciated the help of Norbert Felber as my group leader and his dedication to the work with students.

Only the exceptional support of the microelectronics design center by Frank Gürkaynak and Beat Muheim made the design of our ASICs possible. I also very much appreciated the collaboration with the communication theory group of ETH, especially with Graeme Pope and Michael Lerjen, which led to many interesting projects.

Thanks to all the colleagues at the digital design group, we always had a pleasant atmosphere in our lab. For their support in our common student projects I would like to thank Pit Greisen, Christoph Keller, Michael Mühlberghuber, Christian Senning, and Markus Wenk. I am also grateful to the many students which contributed to this thesis, which includes David Bellasi, Lin Bai, Fabian Huber, Joscha Märkle-Huss, Benjamin Sporrer, and Sebastian Steiner. I would also like to thank all the IIS staff for making our life at the institute much easier, especially Christine Haller for her administrative support, Hansjörg Gisler for fixing every problem, and Christoph Wicki for his IT competence.
At last but not at least, I would like to thank my parents and family for supporting me during all the years of working on my thesis and my girlfriend for giving me new motivation every day.
Abstract

The introduction of compressive sensing (CS) led to a new paradigm in signal processing. Traditionally, signals are sampled above their Nyquist rate. Using CS, the same information is acquired with much fewer measurements, provided a sparse representation of the signal exists. This makes CS a very promising technology with a large number of potential applications.

While the acquisition of measurements is simplified, the reconstruction of the original signal becomes more involved. Sparse signal recovery algorithms solve the corresponding systems of underdetermined linear equations and have proven very efficient for various applications. Examples include de-noising, the restoration of corrupted signals, signal separation, super-resolution, and in-painting. All applications are based on the observation that many natural and man-made signals have sparse representations in some suitable bases.

In the last few years, impressive progress has been made in the development and characterization of fast recovery algorithms. However, the computational effort for successful signal recovery remains high, even for problems of moderate size. Reconstruction becomes especially challenging for real-time applications with stringent power constraints, e.g., on mobile devices. Such applications require efficient hardware implementations of sparse signal recovery algorithms, which we develop in this thesis. We present different architectures of greedy algorithms for a number of selected applications.

The first example is the estimation of sparse channels in broadband wireless communication. The use of sparse recovery algorithms efficiently reduces noise and, thus, increases estimation quality. Architectures for three algorithms are developed and their realizations
in application-specific integrated circuits (ASICs) are compared. We show that approximative algorithms deliver good results at low hardware complexity.

The second application is the recovery of signals corrupted by structured noise. Using the example of audio restoration from corruptions by clicks and pops, fast realizations of the approximate message passing algorithm are designed. Two fundamentally different architectures —one relying on fast transforms, the other relying on parallel processing power— are developed and compared. Large gains in terms of throughput and circuit complexity are realized by applying fast transforms in the context of CS recovery. The choice of the most attractive algorithms and architectures depends on the sparsity, the number of measurements, and the basis in which the samples are taken. In general, approximate message passing is found to be very well suited for hardware recovery of moderately sparse signals while serial greedy pursuits are better suited for very sparse signals.

Further, a new application of CS in localization is explored. We show how sparse recovery increases the detection accuracy in passive radar systems based on WiFi signals.

Finally, also a new sensing device, acquiring measurements with very low hardware complexity, is introduced. This modified analog-to-digital converter samples at non-uniformly distributed points, which allows the reconstruction of Fourier-sparse signals from very few measurements. All the presented examples and hardware implementations bring CS one step closer to practical applications.
Zusammenfassung


Als erstes Beispiel wurde die Kanalschätzung in drahtlosen Kommunikationssystemen gewählt. Wir haben drei ASICs mit unterschiedlichen Algorithmen fabriziert, welche zu einer Verringerung des Rauschanteils in der Kanalschätzung führten. Wir zeigen, dass Algorithmen, welche die Rekonstruktion näherungsweise lösen, in diesem Fall sehr gute Resultate bei kleiner Hardware-Komplexität erreichen.


Des Weiteren wird eine neue Anwendung im Bereich der Lokalisation diskutiert. Wir zeigen, wie CS Rekonstruktions-Algorithmen die Genauigkeit von passivem Radar, basierend auf WiFi-Signalen, verbessern können.

Zuletzt wird auch noch ein neuer Analog-digital-Wandler mit sehr tiefen Anforderungen an die Hardware vorgestellt. Diese Schaltung tastet ein Signal zu unregelmäßigen Zeitpunkten ab und erlaubt somit eine effiziente Schätzung von aktiven Frequenzen mittels CS. Mit all den präsentierten Beispielen und Hardware-Architekturen soll ein Beitrag geleistet werden, um die Kluft zwischen Theorie und praktischen Anwendungen von CS zu verkleinern.
Contents

Abstract vii

Zusammenfassung ix

1 Introduction 1
   1.1 A New Paradigm .......................... 3
   1.2 Sparsity .................................. 5
   1.3 Compressive Measurements ................. 7
   1.4 Recovery Conditions ....................... 8
      1.4.1 Null Space .......................... 10
      1.4.2 Restricted Isometry .................. 11
      1.4.3 Incoherence ......................... 12
   1.5 Recovery Algorithms ....................... 13
      1.5.1 Convex Relaxation .................... 13
      1.5.2 Greedy Algorithms ................... 14
      1.5.3 Performance Analysis ................ 22
   1.6 CS Summary ................................ 24
   1.7 Previous Art ................................ 25
      1.7.1 Compressive Sample Acquisition ....... 25
      1.7.2 Implementations of Recovery Algorithms 26
   1.8 Contributions ............................. 27
   1.9 Outline .................................. 29

2 Channel Estimation 31
   2.1 System Model ............................. 33
      2.1.1 3GPP LTE Physical Layer ............... 33

xii
CONTENTS

4.3 Acoustic Demonstrator ........................................... 94
  4.3.1 System Architecture ........................................ 95
  4.3.2 Detection Performance ..................................... 97
4.4 Summary .............................................................. 99

5 Non-Uniform Sampling .............................................. 101
  5.1 Sparse Spectrum Sensing ....................................... 102
    5.1.1 Ideal Random Sampling ................................... 103
  5.2 Random Slope ADC .............................................. 103
    5.2.1 Traditional Slope ADC .................................... 103
    5.2.2 Random Sampling Slope (RSS-)ADC ......................... 106
  5.3 Reconstruction .................................................. 108
  5.4 Evaluation ....................................................... 109
  5.5 Non-Ideal Hardware ............................................. 113
  5.6 Off-Grid Frequencies .......................................... 116
  5.7 Summary .......................................................... 117

6 Comparison of Recovery Algorithms ............................. 119
  6.1 Numerical Performance Evaluation ......................... 120
    6.1.1 AMP for Fourier-Sparse Signals .......................... 120
    6.1.2 Simulations ................................................. 121
  6.2 Implemented Designs ........................................... 121
  6.3 Generic FPGA Implementations ............................... 124
    6.3.1 Highly Parallel OMP ...................................... 125
    6.3.2 Highly Parallel AMP ...................................... 128
    6.3.3 Hardware Evaluation ...................................... 129
  6.4 Summary .......................................................... 132

7 Conclusions .......................................................... 133
  7.1 VLSI Architectures for Sparse Signal Recovery .............. 133
  7.2 Applications ..................................................... 135
  7.3 Concluding Remarks ............................................ 137

A Notation and Acronyms ........................................... 139
  Symbols .............................................................. 139
  Operators ............................................................ 141
  Acronyms ............................................................ 142
Chapter 1

Introduction

Sparse signal recovery is a new method in signal processing that enables a large variety of surprising applications. By providing tools to solve under-determined systems of linear equations, sparse solutions to otherwise intractable problems can be found. Since many natural or man-made signals are sparse in an appropriate basis, a wide range of possible applications emerged. They include as diverse fields as medical imaging, wireless communication, localization, or image and audio processing. All applications exploit the inherent sparsity of signals and rely on sparse recovery algorithms. These recovery algorithms are, thus, an important aspect of every application and their complexity determines whether economic implementations are possible.

Unfortunately, the non-linear sparse recovery algorithms require significant computational effort, even for problems of moderate size. Many software implementations of such recovery algorithms exist, which are mostly suitable for applications that allow for off-line processing. However, implementations become extremely challenging for applications requiring real-time processing or with tight power constraints on battery-powered (mobile) devices. Hence, to meet the stringent throughput, latency, and power-consumption constraints of real-time applications, the development of dedicated hardware architectures for application-specific integrated circuits (ASICs) or field-programmable gate arrays (FPGAs) becomes necessary. Promising
applications of real-time sparse signal recovery algorithms are wireless
communication, radar, and audio restoration, which will all be dis-
cussed in this thesis. As realizations in dedicated hardware have dif-
ferent requirements than software implementations, algorithms must
specifically be evaluated and selected according to their suitability
for efficient hardware architectures. The main criteria are a regular
control flow, low memory requirements, and suitability for fixed-point
arithmetics.

Sparse signal recovery algorithms enable various applications such
as the restoration of corrupted signals, de-noising, super-resolution,
in-painting, as well as compressive sensing (CS). Rather than sampling
a signal at the possibly very high Nyquist rate, CS [1, 2] allows the
acquisition of a sparse or compressible signal by far fewer measure-
ments, depending on the actual information content of the signal. In-
painting amounts to the restoration of missing entries in a structured
signal, such as images [3]. Restoration of corrupted signals is achieved
through the identification of the corruption’s sparse support within
a signal that is sparse itself. This allows to suppress corruptions
and to recover the original signal [4]. Super-resolution increases the
signal resolution using a method that goes beyond simple interpolation
and estimates additional samples based on a sparse representation.
De-noising improves signal quality by suppressing noise outside of the
subspace spanned by the desired signal [5].

The most prominent application is certainly CS, which changes the
way signals are sampled. The Nyquist-Shannon sampling theorem
is one of the most fundamental theorems in signal processing. It
states that a band-limited, continuous-time signal can be perfectly
reconstructed from its samples when the samples are taken uniformly
at a rate at least twice the highest frequency of the signal (the so-called
Nyquist rate) [6, 7]. This enabled the still ongoing digital revolution in
signal processing: audio, image, video, telecommunication, and many
other signals can be processed at lower cost, more accurately and
more power-efficiently in the digital than in the analog domain. Also,
digital signal processing enabled much more sophisticated signal pro-
cessing algorithms, which increased the performance of many systems
compared to purely analog filtering.

While the performance of digital signal processors continues to
grow according to Moore’s law, the efficiency of sensors is not always
1.1. A NEW PARADIGM

Improving at the same pace. The acquisition of samples may be very time consuming as, e.g., in magnetic resonance imaging (MRI), or the sensors may be very costly as, e.g., in hyper-spectral imaging. Taking digital photography as an example, one realizes that the large amount of data captured by an image sensor with many millions of pixels is reduced by compression algorithms to typically around 10% of its initial amount after the image is captured. This is possible because the image signal is compressible or approximately sparse in a known basis (in this case the wavelet basis). More generally, data compression is possible whenever an (approximately) sparse representation of a signal exists, which means that a signal can be represented by a small number of components in a given basis, while all other components are zero or almost so. Compression by a transformation into an appropriate basis is known as transform coding [8].

However, this raises the question if such a large amount of data really needs to be acquired in the first place when most of it is discarded afterward anyway. Finding an answer to this question leads directly to the fundamental idea behind CS: instead of sampling a signal at Nyquist rate and compress it afterward, can the signal not be acquired directly in a compressed form? CS introduces a corresponding sampling scheme based on non-adaptive linear projections on an incoherent basis, which has the potential to considerably reduce the number of samples needed.

In this first chapter, an introduction to CS is given in Sec. 1.1. CS requires sparse representations, which are discussed in Sec. 1.2. Sensing schemes acquiring compressive measurements are introduced in Sec. 1.3 and are analyzed and discussed in Sec. 1.4. A survey of the most prominent reconstruction algorithms is presented in Sec. 1.5. As this thesis focuses on VLSI architectures for sparse signal recovery and their applications, the corresponding literature is reviewed in Sec. 1.7 and our contributions are summarized in Sec. 1.8. A complete list of symbols and operators used in this thesis can be found in Appendix A.

1.1 A New Paradigm

Compressive sensing is a framework that enables the reconstruction of sparse signals from much fewer measurements than the Nyquist rate
CHAPTER 1. INTRODUCTION

suggested. CS was introduced by Donoho, Candès, Romberg, and Tao in 2006 [1, 2]. This non-adaptive method to acquire measurements directly in a compressed form established a new paradigm in signal processing. Rather than sampling a signal at its Nyquist rate, CS allows to sample at a lower rate, closer to the actual information rate of the signal. This idea has gained significant attention in the research community over the past years. The global research effort in CS is still increasing, not only in information theory, but also in as diverse research areas such as mathematics, computer science, electrical engineering, communications, biology, and even economics. Good introductions to CS can be found in [9], [10], [11], and [12].

A system acquiring CS measurements can be divided into two parts. First, compressive measurements are acquired by linear projection onto an incoherent basis. In an ideal case, such measurements are readily available. Otherwise, projections onto an appropriate basis must be performed explicitly.

Second, reconstruction recovers the original signal from the compressive measurements. Sparse signal recovery algorithms solve the corresponding set or under-determined linear equations. Reconstruction is only possible if certain conditions on the measurements and the sparsity of the original signals are met. These conditions and other essential contributions of CS are reviewed in the remainder of this chapter.

Although CS is a very young field, similar ideas have been around for a long time. Possibly the first approach to reduce samples of sparse signals is Prony’s method introduced in 1795. This method enables the estimation of non-zero amplitudes in series of complex exponentials and found many applications [13]. In 1967, Landau defined the necessary number of samples for Fourier sparse signals with known frequency support, the so-called Landau rate [14]. Later, Feng and Bresler proposed an universal sampling pattern for multi-band signals with unknown support [15]. An early paper by Donoho and Stark, published in 1989 [16], discusses uncertainty relations of Fourier transforms and shows how signals with unoccupied frequencies can be reconstructed from fewer measurements. Vetterli et al. described an approach similar to CS in 2002 to sample signals with limited „rate of innovation” [17, 18], where the number of measurements was reduced using special sampling kernels.
1.2 Sparsity

As the Shannon-Nyquist theorem is still valid in the general case of a band-limited signal, reducing the sampling rate is only possible if a signal does not make use of all the degrees of freedom offered by the occupied bandwidth. This is the case for sparse signals, which exhibit a certain structure and can therefore be represented by a small number of significant coefficients in an appropriate basis. A signal is called $K$-sparse if it can be represented by $K$ non-zero coefficients. If $x \in \mathbb{C}^N$ is the sparse representation of this signal, only $K$ out of its $N$ coefficients $x_i$ have non-zero values. The sparsity of a signal is defined as $\rho = K/N$. Typically, a signal is called sparse if $K \ll N$ and $\rho \ll 1$. Even if a finite-length discrete signal $a[n]$ does not consist of mostly zeros itself, it might still be sparse after a transformation into another basis where a sparse representation is possible. This transformation to a sparsifying basis is known as transform coding [8] and is heavily used in compression algorithms. Assume that $a$ shows this sparsity property in the orthonormal basis $\{\psi_1 \psi_2 \ldots \psi_N\}$. Then, vector $a$ is sparsified when transformed into this new basis

$$a = \sum_{i=1}^{N} x_i \psi_i \quad (1.1)$$

resulting in a sparsely occupied vector $x = \{x_i\}_{i=1}^{N}$. The above equation is more conveniently written in matrix notation with the sparsifying transform matrix $\Psi = [\psi_1 \psi_2 \ldots \psi_N]$:

$$a = \Psi x \quad (1.2)$$

Sparsity is often measured by the $\ell_0$-“norm” $\| \cdot \|_0$, which is defined as the cardinality of the support of $x$: $\text{supp}(x) = \{i | x_i \neq 0\}$, i.e. the number of non-zero elements.

$$K = \|x\|_0 = |\text{supp}(x)| \quad (1.3)$$

Note that $\| \cdot \|_0$ does not fulfill the properties of a norm. However, this abuse of notation can be justified by the fact that $\| \cdot \|_0$ is the limit of $\ell_p$-norms.

$$\lim_{p \to 0} (\|x\|_p)^p = \|x\|_0 \quad (1.4)$$
with the $\ell_p$-norm ($p \in [1, \infty)$) defined as:

$$\|x\|_p = \left( \sum_{n=1}^{N} |x_n|^p \right)^{1/p}$$

(1.5)

Natural signals are often not perfectly but only approximately sparse. These signals are also called *compressible* and the amplitudes of their coefficients decay rapidly when arranged in descending order. This means that these signals can be well represented by a small number of coefficients while all other coefficients are nearly zero. The error introduced by neglecting these almost zero coefficients is usually negligible.

**Example: Images** Natural images can often be characterized by only a few sharp edges and larger smooth or textured areas. This allows for an approximately sparse representation in a wavelet basis (as used in JPEG2000 compression). The dictionary elements are shifted and scaled versions of wavelets. An example of a Haar-wavelet decomposition is shown in Fig. 1.1. Only light coefficients are significant, dark ones are nearly zero.
1.3 Compressive Measurements

This section explains how measurements with a reduced sampling rate must be taken such that a sparse signal can still be reconstructed perfectly. Each measurement should contain certain information about all non-zero components. A compressed signal representation is, therefore, not possible in a basis similar to the sparsifying basis and requires a more elaborate measurement scheme.

In CS, $M < N$ measurements $y = \{y_i\}_{i=1}^M$ are acquired by projections onto vectors $\{\omega_i\}_{i=1}^M$. This corresponds to computing the inner product $y_i = \langle a, \omega_i \rangle$. Setting $\Omega = [\omega_1^T \omega_2^T \ldots \omega_M^T]^T$, this can be written in matrix notation.

$$y = \Omega a$$  \hspace{1cm} (1.6)

Substituting $a$ by its sparse representation (cf. Sec. 1.2) and setting $\Omega \Psi = \Phi$ leads to

$$y = \Omega a = \Omega \Psi x = \Phi x .$$  \hspace{1cm} (1.7)

We will refer to $\Phi$ as the measurement matrix. The measurement matrix links each sparse coefficient to a certain pattern in the measurements. A basis vector $\phi_k$ of $\Phi = [\phi_1 \phi_2 \ldots \phi_N]$ is called a dictionary element or atom. The illustration in Fig. 1.2 compares a measurement matrix for Nyquist sampling (which is an identity matrix) to a typical CS measurement matrix, which is a fully occupied fat matrix with random coefficients. In the CS framework, measurements are non-adaptive, i.e., the sensing scheme is independent of the present signal. Since $\Psi$ is an orthogonal transformation, reconstructing $x$ is equivalent to the reconstruction of $a$. This now raises the question how a measurement matrix needs to be designed, such that an original signal $a$ can be reconstructed from its measurements $y$. The criteria for suitable measurement matrices $\Phi$, which guarantee successful reconstruction of $x$, will be discussed in the following section.

Example: Single pixel camera An example of a device implementing a suitable measurement strategy is the single pixel camera [19]. An image is acquired by projection onto a digital micromirror device followed by a measurement of the superposition of all reflections with a single pixel. Multiple (pseudo-random) on/off-patterns on
the micromirror array need to be measured, such that an image can be successfully reconstructed. In this way, fewer measurements are required compared to individually sampling every pixel.

1.4 Recovery Conditions

Reconstructing $x$ from the noiseless measurements $y = \Phi x$ corresponds to solving an under-determined set of linear equations, which means that, in general, the estimation of $x$ is an ill-posed problem. If, however, the signal $x$ is known to be sparse, there exists – under certain conditions – a unique solution. The goal of sparse recovery is to find the sparsest possible solution which satisfies (1.7). This optimization can be formulated as

$$\hat{x} = \arg\min_x \|x\|_0 \text{ subject to } y = \Phi x.$$  

Unfortunately, $\ell_0$-minimization requires the exhaustive enumeration of all $\binom{N}{K}$ possible sparse patterns, which results in an NP-hard problem [20] and becomes computationally intractable – even for small problem sizes. Therefore, devising efficient recovery algorithms is a crucial point to make CS practically useful.

A key contribution of CS was the proof that $\ell_0$-minimization can be replaced by a convex $\ell_1$-minimization returning exactly the same
1.4. RECOVERY CONDITIONS

Figure 1.3: Illustration of $\ell_2$- (left), $\ell_1$- (middle), and $\ell_0$-minimization (right) for the 2-dimensional case.

A sparse result with high probability and under some conditions on the measurement matrix. This formulation as an $\ell_1$-optimization problem is known as basis pursuit (BP).

$$\hat{x} = \arg\min_x \|x\|_1 \text{ subject to } y = \Phi x$$  \hspace{0.5cm} (1.9)

The case of a two dimensional ($N = 2$) reconstruction problem is illustrated in Fig. 1.3. The original signal $x$ is assumed to be $K = 1$ sparse, which means that the signal is confined to one axis while the other component is zero. Minimizing the $\ell_2$-norm means blowing up the $\ell_2$-ball until the constraint line is touched. However, the solution is typically not sparse since the $\ell_2$-ball does not usually touch the line on an axis. If we use an $\ell_1$-ball (a rhombus) instead, the corners will reach the line first, which corresponds to a 1-sparse solution. This gives in most cases the same result as actually computing the $\ell_0$-“norm”, which restricts the solution to one axis in the $K = 1$ case. Going from the NP-hard $\ell_0$-minimization (1.8) to the convex $\ell_1$-minimization (1.9) greatly reduces the required computational effort.

**Modeling of Noise** Any physical sensing device will corrupt measurements by a certain level of noise $e \in \mathbb{C}^M$.

$$y = \Phi x + e$$  \hspace{0.5cm} (1.10)

Reconstruction from noisy measurements can be stated in a relaxed form as a basis pursuit de-noising (BPDN) [21] problem, which allows some measurement mismatch of $\epsilon > 0$.

$$\hat{x} = \arg\min_x \|x\|_1 \text{ subject to } \|\Phi x - y\|_2 \leq \epsilon$$  \hspace{0.5cm} (1.11)
It is clear that noisy measurements do not allow for perfect reconstruction of a signal. Yet, one can derive good bounds on the reconstruction error.

1.4.1 Null Space

The null space or kernel of $\Phi$ allows to formulate some conditions on the measurement matrix that need to be fulfilled for a successful reconstruction. The null space of $\Phi$ contains all vectors $x$, which are mapped to $0$.

$$\mathcal{N}(\Phi) = \{x : \Phi x = 0\}$$

(1.12)

An obvious condition is that two different $K$-sparse signals $x \neq x'$ do not result in the same measurement vector, i.e., their difference is not part of the null space of the measurement matrix.

$$\Phi(x - x') \neq 0 \iff x - x' \notin \mathcal{N}(\Phi)$$

(1.13)

The difference between two $K$-sparse vectors is at most $2K$-sparse. Therefore, a $K$-sparse $x$ is uniquely defined if $\mathcal{N}(\Phi)$ contains no $2K$-sparse vectors. This corresponds to the condition that any $2K$ columns of $\Phi$ are linearly independent, which leads to a lower bound on the number of measurements, a necessary condition that reconstruction is possible:

$$M \geq 2K$$

(1.14)

Except for some special cases, this condition is not sufficient and it does not allow for a stable recovery of approximately sparse signals. The null space property (NSP) is a measure of how concentrated the null space is. The NSP is satisfied if there is a constant $\gamma \in (0, 1)$ such that

$$\|n_S\|_2 \leq \gamma \|n_{S^C}\|_1, \text{ for all } n \in \mathcal{N}(\Phi)$$

(1.15)

for all sets $S \subset \{1, \ldots, N\}$ with cardinality $K$ and their complements $S^C$, defined as a set of all elements which are not part of the set $S$; $S^C = \{1, \ldots, N\} \setminus S$. If the NSP is satisfied, an exactly $K$-sparse signal is perfectly reconstructed by $\ell_1$-minimization. For approximately $K$-sparse signals, one can establish an upper bound on the error of the $\ell_1$-minimization $\hat{x}$ [11]:

$$\|x - \hat{x}\|_1 \leq 2 \frac{1 + \gamma}{1 - \gamma} \sigma_K(x)_1$$

(1.16)
where $\sigma_K(x)_p$ is the minimal error introduced by the best possible $K$-sparse approximation.

$$
\sigma_K(x)_p = \min_{\|x\|_0=K} \|x - \hat{x}\|_p
$$

(1.17)

### 1.4.2 Restricted Isometry

We also need means to quantify the robustness of CS under noisy and only approximately sparse conditions. The most prominent criterion is the restricted isometry property (RIP) [22] (also called uniform uncertainty principle). A matrix $\Phi$ fulfills the RIP with restricted isometry constant $\delta_k$ if

$$
1 - \delta_k \leq \frac{\|\Phi x\|_2}{\|x\|_2} \leq 1 + \delta_k
$$

(1.18)

for every $k$-sparse vector $x \in \mathbb{C}^N$. The RIP measures how well distances are preserved in a linear transformation. From the above discussion of the null space, it is clear that, if the RIP is fulfilled for $2K$ with $\delta_{2K} < 1$, there are no two $K$-sparse vectors $x$ that result in the same measurement vector $y$.

Candés et al. [23] introduced an error bound for the solution of (1.11) valid under the condition that $\delta_{3K} + 3\delta_{4K} < 2$ and that the error $\|e\|_2 < \epsilon$ is bounded.

$$
\|x - \hat{x}\|_2 \leq C_1 \epsilon + C_2 \sigma_K(x)_1/\sqrt{K}
$$

(1.19)

The constants $C_1, C_2$ are small for a reasonable $\delta_{4K}$. A better solution was found later in [24] requiring only $\delta_{2K} < \sqrt{2} - 1$ for a bounded error $\|e\|_2 < \epsilon$ to achieve the bound in (1.19). Even slightly tighter bounds are reported in newer literature (e.g. [25]).

A bound as stated in (1.19) is the best we can hope for. There is only one term depending on the measurement noise $\epsilon$ and one stemming from not-perfectly-sparse signals $\sigma(x)_1$, both of which cannot be avoided. Note that this guarantees perfect reconstruction in the noiseless and perfectly sparse case whenever $\delta_{2K} < \sqrt{2} - 1$. Unfortunately, calculating the restricted isometry constant of a given matrix is itself a problem of combinatorial complexity.
1.4.3 Incoherence

A criterion related to the RIP is the incoherence of the projection basis $\Omega$ and the sparsifying basis $\Psi$. The mutual coherence is defined as

$$\mu(\Omega, \Psi) = \max_{i \in [1,M], j \in [1,N]} |<\omega_i, \psi_j>|. \quad (1.20)$$

Also the coherence of the combined measurement matrix $\Phi$ is a valid criterion.

$$\mu(\Phi) = \max_{i, j \in [1,N], i \neq j} |<\phi_i, \phi_j>| \quad (1.21)$$

A relation to the restricted isometry constant can be established by the following bound [26]:

$$\delta_K \leq (K - 1)\mu \quad (1.22)$$

The less coherent the two bases $\Omega$ and $\Psi$ are, the better the reconstruction works. As a consequence, a signal with a sparse representation in one basis cannot have a sparse representation in the other basis. An example of such a pair of bases is the Fourier and time base pair. As known from the uncertainty principle, it is not possible to have a sparse representation in both bases. Finding good measurement matrices, thus, amounts to finding incoherent bases.

Surprisingly, randomly chosen measurement matrices proved to perform very well with high probability [27]. Good examples include the following random matrices:

**Random Gaussian** Each entry of $\Phi$ is independently drawn from a normal distribution with zero mean and variance $1/M$. It has been shown that the $\ell_1$-minimization (1.9) returns the correct $K$-sparse solution with overwhelming probability when

$$M \geq CK \log(N/K) \quad (1.23)$$

for a small constant $C > 0$. More generally, the same performance is achieved for any basis $\Psi$ when $\Omega$ is a random Gaussian matrix, i.e., $\Phi$ is sub-Gaussian.

**Random Binary** Each entry of $\Phi$ is selected from a Bernoulli distribution and can assume the values $\pm 1/\sqrt{M}$ with the same probability. The conditions for perfect recovery are the same as for Gaussian matrices.
Random Selection of Fourier Samples

$M$ rows from a $N$-point discrete Fourier transform matrix are randomly selected. This corresponds to measuring a random selection of frequencies of a signal sparse in time or vice versa. The requirement for a correct reconstruction by $\ell_1$-minimization is

$$M \geq CK(\log N)^{4}$$

for a small constant $C$.

1.5 Recovery Algorithms

A plethora of algorithms has been introduced recently in order to reconstruct the original signal $x$ from its compressed measurements $y$ by solving (1.8) or an approximation thereof. While directly solving the $\ell_0$-minimization problem (1.8) is theoretically possible, the complexity quickly becomes intractable, even for small problems. We thus restrict ourselves to algorithms solving the convex $\ell_1$-minimization (1.9) and greedy approximations.

In the remainder of this section, a selection of algorithms is introduced. The main focus lies on their structure and their suitability for very large scale integration (VLSI) implementation. The main criteria for efficient VLSI integration are a simple control flow, high regularity, well separable tasks, reasonably low memory requirements, and suitability for fixed-point arithmetics [28]. Eventually, their performance is compared by means of numerical simulations in Sec. 1.5.3.

1.5.1 Convex Relaxation

Convex relaxation algorithms are able to accurately solve the convex BP problem (1.9) and are well studied [29]. The earliest algorithms were based on the simplex method, which was later replaced by the interior-point method. Both methods solve the convex relaxation problem by iteratively approaching the optimal solution. In the case of noisy data, the problem can be re-written as BPDN problem (1.11). Many algorithms solve BPDN in its Lagrangian form:

$$\hat{x} = \arg \min_{x} \frac{1}{2}\|y - \Phi x\|_{2}^{2} + \lambda\|x\|_{1}$$

(1.25)
The regularization parameter $\lambda$ balances sparsity with fidelity to measurements. For a certain choice of $\lambda$, which is, however, a priori unknown, (1.25) is equivalent to (1.11). Popular choices for algorithms solving (1.25) are primal-dual interior-point methods [29] or the newer fixed-point continuation [30], which is based on soft-thresholding. A slightly different formulation of BPDN is solved by the least absolute shrinkage and selection operator (LASSO) [31].

$$\hat{x} = \arg \min_x \|y - \Phi x\|_2^2 \text{ subject to } \|x\|_1 \leq t$$  \hspace{1cm} (1.26)

These convex optimization methods are able to solve problems of moderate to large size, but are known to demand high computing power, have often complicated structures, and typically require high numerical precision. While these are no large obstacles in software implementations, these drawbacks make convex optimization unsuitable for VLSI implementation [28]: the complicated structure leads to an irregular control flow and the precision requirements make fixed-point computations inefficient. Also, often large memories are required which are expensive on dedicated hardware.

### Linear Program

A linear program (LP) is a standard problem in the following form:

$$\min c^T x \text{ subject to } Ax = b, \ x \geq 0$$ \hspace{1cm} (1.27)

As many algorithms solving this problem exist, it can be useful to formulate the $\ell_1$-minimization as LP. [21] gives the following translation.

$$A = [\Phi - \Phi], \ b = y, \ c = [1 \ 1]^T, \ x = [u^T \ v^T]^T, \ u - v = y$$ \hspace{1cm} (1.28)

In the case of noisy measurements, a similar transformation results in a second-order cone program.

### 1.5.2 Greedy Algorithms

For VLSI implementations with real-time requirements, simpler and faster algorithms than convex relaxations or linear programs are usually needed. Greedy algorithms are an interesting option to reduce
1.5. RECOVERY ALGORITHMS

complexity. They typically produce sub-optimal solutions, but the required computation time is greatly reduced while the optimal solution is approximated well. Some of the more sophisticated variations even allow for similar conversion guarantees as $\ell_1$-optimizers. For these reasons, greedy algorithms recently gained significant attraction and many different versions were developed. Here, we review the following selection of prominent greedy algorithms:

1. Matching pursuit (MP)
2. Orthogonal matching pursuit (OMP)
3. Gradient pursuit (GP)
4. Compressive sampling matching pursuit (CoSaMP)
5. Subspace pursuit (SP)
6. Iterative hard thresholding (IHT)
7. Iterative soft thresholding (IST)
8. Approximate message passing (AMP)

Greedy algorithms operate in an iterative manner. A first class of algorithms, which we refer to as serial greedy pursuits, sequentially add one dictionary element to the estimate in each iteration. Algorithms 1–3 belong to this class. They are computed by the pseudo-code listed in Alg. 1. A data dependency graph showing the iterative nature is given in Fig. 1.4. Algorithms 1–3 differ only in the estimation function updateX($S^n$), where $S^n$ denotes the current state and includes the complete set of variables $S^n = \{x^{n-1}, g^n, i^n, \Phi, \Gamma^n, r^{n-1}, y\}$. In each iteration, the dictionary element with the strongest correlation to the current residual $r$ is added to the set of non-zero coefficients $\Gamma^n$. All the chosen elements span a subspace in which the estimation of $x$ is performed. The columns of the measurement matrix $\Phi$ are assumed to be normalized to 1 ($\|\phi_i\|_2 = 1$).

Parallel greedy pursuits (4–5) are a second class of greedy algorithms, which evaluate a full set of candidates ($\geq K$) in each iteration. This allows for a re-evaluation of the selected set of dictionary elements in each iteration, which means that new elements can be added and obsolete ones discarded.

The very similar third class, the thresholding algorithms (6–8), can also add and discard multiple elements per iteration. In each iteration, the previous estimate is adapted by a certain step-size. A non-linear
Algorithm 1 Serial greedy pursuits (MP, OMP, GP)

1: \( r^0 \leftarrow y; \ x^0 \leftarrow 0_M; \ n \leftarrow 1; \ \Gamma^0 \leftarrow \emptyset \)
2: \textbf{while} stopping criterion not met \textbf{do}
3: \hspace{1em} \( g^n \leftarrow \Phi^H r^{n-1}; \)
4: \hspace{1em} \( i^n \leftarrow \arg\max_i |g^n_i|; \)
5: \hspace{1em} \( \Gamma^n \leftarrow \Gamma^{n-1} \cup i^n; \)
6: \hspace{1em} \( x^n \leftarrow \text{updateX}(S^n); \)
7: \hspace{1em} \( r^n \leftarrow y - \Phi x^n; \)
8: \hspace{1em} \( n \leftarrow n + 1; \)
9: \hspace{1em} \textbf{end while}
10: \textbf{Output} \( x^n; \)

\[
\text{updateX}(S^n) = x^{n-1} + e_{i^n} g^n_{i^n}, \quad (1.29)
\]

Figure 1.4: Iterative processing in serial greedy pursuits.

The thresholding function is used to select the elements with the largest contribution.

Matching Pursuit (MP)

MP is the simplest greedy algorithm and was first introduced in [32]. After choosing the most suitable dictionary element (the one having the highest correlation with the measurements), the contribution of the selected element is directly added to the sparse estimate. Since the elements are not orthogonal in general, the same element might be updated multiple times. The MP update function changes only one element in the current estimate \( x^n \).

\[
\text{updateX}(S^n) = x^{n-1} + e_{i^n} g^n_{i^n}, \quad (1.29)
\]
where $\mathbf{e}_i$ is a unit vector with all elements zero, except a '1' at position $i$. The computationally most expensive operation of the MP algorithm is the matrix-vector product on line 3 of Alg. 1. However, it is well-known \cite{33} that after the first iteration, this operation can be replaced by a less complex update operation

$$g^n = \mathbf{\Phi}^H r = g^{n-1} - g_{i \in n} \mathbf{\Phi}^H \mathbf{\Phi}_{i \in n-1}$$  \hspace{1cm} (1.30)$$

using pre-computed correlation coefficients $\mathbf{\Phi}^H \mathbf{\Phi}$ by combining lines 3 and 7 of Alg. 1. This update strategy requires storing $\mathbf{\Phi}^H \mathbf{\Phi} g$ for $g = 1 \ldots N$ but reduces the number of multiplications per iteration from $NM$ to $N$.

Its very easy operation principle makes MP suitable for VLSI implementations. However, MP may converge very slowly, depending on the properties of the measurement matrix. There is not even a guarantee that it converges at all. Although the computational complexity is very low for each iteration, the possibly large number of required iterations might decrease MP's efficiency.

**Orthogonal Matching Pursuit (OMP)**

OMP is an extension of the MP algorithm and was introduced in \cite{33}. The extension consists in a more elaborate update function that includes all previously chosen dictionary elements. The update is calculated by finding the least squares (LS) optimum in the subspace spanned by the already chosen dictionary elements $\mathbf{\Phi}_{\Gamma_n}$.

$$\text{updateX} (S_n) = \arg \min_z ||\mathbf{\Phi}_{\Gamma_n} z - y||_2$$  \hspace{1cm} (1.31)$$

As a result, the new estimate will be orthogonal to the residual after each iteration, which implies that the same element will not be selected again. The number of required iterations is therefore limited by the sparsity $K$.

The additional LS optimization significantly increases the complexity per iteration compared to MP. The required effort to compute (1.31) can, however, be reduced by avoiding the computation of the full LS optimization in each iteration. When the LS step is computed by a matrix decomposition such as the Cholesky or the QR decomposition (QRD), the decomposed matrices can be stored and
provide a starting point for the decomposition in the next iteration. As efficient methods to calculate a matrix decomposition exists, OMP is nevertheless suitable for VHDL implementation, mostly due to the regular structure of the LS optimization. Further complexity reduction is possible, as it will be discussed in the hardware implementation Sec. 2.3.3.

**Gradient Pursuit (GP)**

The gradient pursuit algorithm was introduced in [34] as an approximation of OMP while the complexity is kept near that of MP. The GP algorithm updates all coefficients of the preliminary estimate $x^n$ by moving into a certain update direction within the subspace spanned by all selected coefficients $x_{\Gamma^n}$. The most obvious choice for the update direction is the gradient $g^n_{\Gamma^n}$ already computed during the correlation $g^n = \Phi^H r^{n-1}$ in line 3 of Alg. 1. By setting $c^n = \Phi_{\Gamma^n} g^n_{\Gamma^n}$, the update function can be defined as

$$\text{updateX}(S^n) = x^{n-1} + \frac{\langle r^{n-1}, c^n \rangle}{\|c^n\|_2^2} g^n_{\Gamma^n}. \quad (1.32)$$

Correlation can be simplified in the same way as in MP. However, all coefficients of $x^{n-1}_{\Gamma^n}$ might have changed in GP. Therefore, in each iteration multiple updates must be performed on $g$.

**Compressive Sampling Matching Pursuit (CoSaMP)**

The CoSaMP algorithm [35] is shown in Alg. 2. In each iteration, the $2K$ best fitting atoms are added to the $K$ atoms of the last iteration. This results in no more than $3K$ atoms, which are reduced to the best $K$ atoms after a LS step. This procedure is more complex than all the greedy algorithms introduced above but it allows to remove unfitting elements, which is not possible in serial greedy pursuits.

The functions which need to be implemented are very similar to OMP, which make CoSaMP basically suitable for VLSI implementation. However, since the set of selected atoms can change in each iteration, CoSaMP is much less regular than OMP and many of the complexity reduction methods from OMP cannot be applied.
Algorithm 2: CoSaMP

1: $r^0 \leftarrow y, \Gamma \leftarrow \emptyset, n \leftarrow 1$
2: while stopping criterion not met do
3: \hspace{1em} $c^n \leftarrow \Phi^H r^{n-1}$
4: \hspace{1em} $\Gamma \leftarrow \Gamma \cup \text{findLargestIndices}(|c^n|^2, 2K)$
5: \hspace{1em} $x_{\text{new}} \leftarrow \arg\min_x \|\Phi_{\Gamma}x - y\|_2^2$
6: \hspace{1em} $r \leftarrow r - \Phi_{\Gamma}x_{\text{new}}$
7: \hspace{1em} $n \leftarrow n + 1$
8: end while
9: $x \leftarrow 0$
10: Output $x_{\Gamma} \leftarrow x_{\text{new}, \Gamma}$

This significantly increases the implementation and computational complexity of CoSaMP compared to OMP.

Subspace Pursuit (SP)

SP is very similar to CoSaMP and was developed at the same time by [36]. The only difference is that instead of $2K$ only $K$ new elements are selected in each iteration. This makes the algorithms less complex and changes the analysis of the performance guarantees. While better analytical bounds were derived for CoSaMP than for SP, in many applications SP outperforms CoSaMP.

Iterative Hard-Thresholding (IHT)

IHT algorithms [37] have a very simple structure. In each iteration the following update is performed:

$$x \leftarrow \eta_K \left( x + \mu \Phi^T (y - \Phi x) \right)$$

with step-size $\mu$ and a non-linear thresholding operator $\eta_K(\cdot)$, which keeps only the $K$ largest components and sets all others to 0. Unfortunately, setting the step size $\mu$ to a fixed value can not provide a stable and guaranteed convergence. Therefore, an adaptive step-size as described in [38] needs to be implemented, making the algorithm
more complicated and more difficult to implement in hardware due to the irregular step-size calculation.

**Iterative Soft-Thresholding (IST)**

IST [39] is an adaption of IHT that replaces hard-thresholding by the following element-wise soft-thresholding operation:

\[
\eta_\theta(x) = \text{sign}(x) \left[ |x| - \theta \right]_+
\]

The threshold \( \theta \) needs to be adapted in each iteration. IST is able to deliver the result to (1.25) given that \( \Phi \) and the thresholds \( \theta \) satisfy certain properties [40]. Unfortunately, IST exhibits slow convergence, which leads to high computational complexity. Although IST is performing only simple operations, which can efficiently be implemented in dedicated hardware, the slow convergence often discourages real-time applications.

**Approximate Message Passing (AMP)**

The recently introduced AMP algorithm [41] shows a similar structure to IHT and IST. However, derived from message passing algorithms, two important differences are present:

- The residual \( r^i \) is computed not only based on the current estimate \( x^i \) but also on the residual obtained in the previous iteration \( r^{i-1} \).

- The threshold \( \theta \) is updated in each iteration based on the regularization parameter \( \lambda \) and the last residual \( r^{i-1} \). No explicit rule to calculate \( \theta \) is given in the original AMP publications, but two alternative are proposed in [42].

Both differences improve the convergence rate of AMP compared to IST. In particular, AMP provably delivers the solution to (1.25) for matrices \( \Phi \) having zero-mean i.i.d. (sub-)Gaussian distributed entries of order \( 1/\sqrt{M} \) [41] at very low computational complexity. While for arbitrary matrices, AMP does not necessarily converge to the solution of (1.25), [43] has shown that AMP performs excellently for many deterministic and highly structured matrices, such as sub-sampled
1.5. RECOVERY ALGORITHMS

Algorithm 3 Approximate message passing (AMP)

1: initialize $r^0 \leftarrow y$ and $x^0 \leftarrow 0_N$
2: for $i = 1, \ldots, I_{\text{max}}$ do
3: \[ \theta \leftarrow \lambda \frac{1}{\sqrt{M}} \| r^{i-1} \|_2 \]
4: \[ x^i \leftarrow \eta_\theta (x^{i-1} + \Phi^T r^{i-1}) \]
5: \[ b \leftarrow \frac{1}{M} \| x^i \|_0 \]
6: \[ r^i \leftarrow y - \Phi x^i + b r^{i-1} \]
7: end for
8: return $x^{I_{\text{max}}}$

Fourier matrices. The AMP implementations in this work follow the pseudo-code given in Alg. 3. The threshold $\theta$ is chosen proportionally to $\lambda$ and the root mean squared error (RMSE) of the residual error $\text{RMSE} = \frac{1}{\sqrt{M}} \| r^{i-1} \|_2$ [42].

Due to its regular structure, the simple arithmetic operations, and the fast convergence, AMP is very well suited for VLSI implementations. A drawback compared to greedy pursuits is that two matrix-vector multiplications must be performed per iteration instead of only one.

The original message passing algorithms are Bayesian algorithms that solve the CS reconstruction problem by belief propagation [44]. These full Bayesian algorithms might be attractive if the measurement matrix $\Phi$ is sparse itself, otherwise they are much too complex.

Stopping Criterion

Since all greedy algorithms operate iteratively, stopping criteria have to be established. The simplest possibility is to fix the number of iterations to a value that is known to deliver good convergence for a certain class of problems. A more dynamic criterion is to evaluate after each iteration how well the algorithm has converged. For example, an algorithm can be stopped as soon as the energy of the residual $r$ falls below a certain threshold, which is set relatively to the noise. Another method to evaluate convergence is to measure the difference to the solution of the last iteration. When the norm of the residual is not further decreasing, iterations can be stopped. Which criterion fits
best must be evaluated for each application and algorithm. For every real-time implementation it is, however, important to set an upper limit on the number of iterations such that the results are not delayed beyond the timing constraints of the given application.

1.5.3 Performance Analysis

Numerical simulations were performed using Matlab to compare the algorithms introduced above\(^1\). For our simulations, the dimension of the sparse signal is set to \(N = 512\), \(M = 256\) measurements are taken, the support of its sparse components is randomly chosen, and the amplitudes are \(\pm 1/\sqrt{K}\) with equal probability. The measurement matrices \(\Phi\) are generated with i.i.d. random Gaussian entries and its columns are normalized to \(\|\phi\|_2^2 = 1\). In our simulations, the sparsity \(\rho = K/N\) is swept.

The results are shown in Fig. 1.5 where the support recovery rate is used as a quality metric. We define the support recovery rate as the ratio of successful reconstructions to the total number of reconstructions in a Monte Carlo simulation. A reconstruction is called successful when all non-zero coefficients are identified correctly. The results show that \(\ell_1\)-minimization achieves the best performance, followed by AMP, SP and CoSaMP. The required computing time per reconstruction is shown in Fig. 1.6 on a logarithmic time scale as an approximative measure for computational complexity. The computing time not only depends on the algorithmic complexity but also on the software implementation style and other concurrent processes. However, the large measured time differences allow to draw some sensible conclusions. \(\ell_1\)-minimization and IST are by far the most computationally demanding algorithms, which makes them unattractive for real-time applications. All the other greedy algorithms have lower run times. Among them, IHT is generally the most complex and AMP the least complex for less sparse signals. In this scenario, AMP offers an attractive performance-computing time trade-off.

\(^1\)For the \(\ell_1\)-minimization, the "l1-magic" packet by J. Romberg was used. Available online at: http://users.ece.gatech.edu/~justin/l1magic/. IST was calculated with an optimally tuned version [45], available at http://sparselab.stanford.edu/\textit{OptimalTuning/main.htm}. Own implementations were used for the other algorithms.
1.5. RECOVERY ALGORITHMS

Figure 1.5: Support recovery rate of various reconstruction algorithms with $M/N = 1/2$.

Figure 1.6: Average computing time of various reconstruction algorithms.
CHAPTER 1. INTRODUCTION

Note that with a different ratio of $M/N$, other noise levels, or different measurement matrices, the result may look very different. Thus, each problem must be individually examined to find the best algorithms. A more thorough numerical analysis of optimally tuned greedy algorithms using Donoho-Tanner phase transition diagrams is performed in [45]. Also, a more detailed performance evaluation at different sparsity levels and undersampling rates is presented in Chapter 6.

1.6 CS Summary

Compressive sensing is a new paradigm in signal processing that allows to sample signals at their ‘information rate’ rather than at the Nyquist rate. If a signal is sparse (i.e. most of the components are zero) in any given basis, the number of measurements can be largely reduced. Measurements need to be taken by linear projections onto vectors. Those vectors must form a basis that is incoherent to the basis in which the signal can be sparsely represented.

Many algorithms were proposed to reconstruct sparse signals from the acquired measurements. $\ell_0$-minimization gives the optimal performance but is an NP-hard problem. Fortunately, $\ell_0$-minimization can be replaced by convex $\ell_1$-minimization, which will return exactly the same result under certain conditions on the measurement matrix. The most common condition on the measurement matrix is formulated using the RIP given in (1.18). If the RIP is fulfilled with restricted isometry constant $\delta_{2K} < \sqrt{2} - 1$, a $K$-sparse signal can be perfectly reconstructed from noiseless measurements. A more intuitive criterion is the incoherence of the measurement matrix columns. A good example of such matrices are random (sub-)Gaussian matrices, which allow reconstruction with very high probability if the following condition on the number of measurements is met: $M \geq CK/\log(N/K)$ with a constant $C > 0$. The required computing effort of an $\ell_1$-minimization can be further reduced by using greedy algorithms instead, which, however, may reduce the reconstruction performance. Depending on the measurement matrix, the sparsity, and the noise level, different algorithms perform best.
1.7 Previous Art

While significant research effort has been devoted to theoretical aspects of CS, such as high-performance and low-complexity sparse signal recovery algorithms and corresponding convergence criteria, much less is known about efficient implementations of CS sensors and signal reconstruction in dedicated hardware. We first review CS signal acquisition systems and second describe the few available hardware implementations of reconstruction algorithms.

1.7.1 Compressive Sample Acquisition

In some applications, signal acquisition in an incoherent domain is already inherent to the functional principle, such as in MRI [46] where signals are acquired in a 2-dimensional Fourier space. Other applications require the development of specialized sensing hardware. A prominent example of such a specialized CS sensing device is the single pixel camera [19], introduced in Sec. 1.3. This device allows to reduce the number of pixels to one by using a digital micromirror device. While the extremely cheap pixel sensors for visible light make a single-pixel camera uneconomical for classical photography, it might find applications in more complex spectral imagers, such as in multi-spectral imaging. Another imaging application is described in [47], where the number of readouts of a complementary metal-oxide semiconductor (CMOS) imaging chip is reduced by adding up the measurements of multiple pixels in a randomized way.

The acquisition of wide-band signals is another promising field of application for CS. The goal is to reduce the number of samples necessary to acquire a spectrally sparse signal. The random demodulator [48] achieves this goal by mixing the incoming signal with a pseudo-random ±1 sequence followed by integration, which results in a reduced number of analog-to-digital conversions. However, many parallel branches of mixers and integrators become necessary for better performance. An ASIC implementation of such a system for data compression in wireless sensor nodes is presented in [49]. The authors argue that the projection onto pseudo noise (PN) sequences is more energy-efficient in the digital than in the analog domain. On the other hand, [50] present a new architecture for analog-to-information
conversion that is claimed to be more energy-efficient than projections in the digital domain.

For sparse multiband signals, [51] proposes a sub-Nyquist sampling scheme called modulated wide-band converter. Similar to the random demodulator, multiple branches of mixers with pseudo-random sequences are used. Instead of being summed up in integrators, the signals are passed through filters before they are sampled at a low rate. The multiband signal is then recovered by algorithms operating on a continuous spectrum.

An alternative sampling scheme for spectrally sparse signals are non-uniform samplers. Two possible implementation strategies are described in [52]: one uses a bank of low-rate analog-to-digital converters (ADCs) with shifted starting points, the other one utilizes capacitors to store analog values until they get sampled by a low-rate ADC. A very recent hardware implementation that uses an edge-triggered ADC for power reduction acquires 25 MHz bandwidth signals with 10 times fewer samples [53]. An even higher bandwidth of 2 GHz is covered by hardware presented in [54], which is based on a custom sample-and-hold chip with non-uniform trigger signals followed by an off-the-shelf ADC.

Further CS applications were proposed for radar [55, 56], electrocardiography (ECG) [57] and electroencephalography (EEG) [58] together with corresponding sensing schemes. Radar applications will be reviewed later in this thesis in Sec. 4.1.

1.7.2 Implementations of Recovery Algorithms

Applications such as radar or spectrum sensing will strongly benefit from real-time signal reconstruction. While a large effort has been made to reduce the computational complexity of recovery algorithms, very little is known about efficient ways to perform this recovery in dedicated hardware. Since even the fastest algorithms exhibit significant computational complexity for the problem sizes occurring in most practical applications, implementations that increase the processing speed are in high demand.

Two early FPGA implementations of MP perform sparse channel estimation for wireless communication [59] or under-water acoustic communication [60].
A recovery algorithm specifically designed for signals acquired by the modulated wide-band converter was implemented on an FPGA in [61], which executes a variation of OMP for the reconstruction of band-sparse signals. A Gram-Schmidt procedure is applied for LS minimization and the architecture is based on 144 parallel complex-valued multipliers for fast reconstruction speed.

Another FPGA implementation of OMP for generic CS problems of dimension $32 \times 128$ was developed in [62]. An alternative Cholesky decomposition is used to solve the LS problem. Processing speed was increased by using 32 parallel multipliers. However, only signals with a sparsity up to 5 can be processed by this implementation.

1.8 Contributions

In this thesis, a number of promising applications of sparse signal recovery with real-time, low-power, and low-complexity requirements are selected and corresponding VLSI architectures are designed. All previous hardware implementations rely on many parallel multipliers available in large FPGAs. In contrast to these generic designs, our designs are optimized to specific applications. In the first example, only the generation of matrix coefficients is application-specific. In our second example, all matrix-vector multiplications are replaced by efficient application-specific arithmetic operations.

Channel estimation  The sparse multi-path characteristics of many broadband wireless communication channels allow for effective denoising of the channel estimate by sparse recovery algorithms. Since many wireless devices are battery-operated, dedicated hardware best meets the low-power constraints. To this end, we developed VLSI architectures for the three greedy algorithms, MP, GP, and OMP and measured and compared various figures of merit of their ASIC implementations. The GP algorithm was reformulated and fast update procedures were applied in OMP. Also, efficient methods to generate matrix coefficients at a high rate were proposed. The corresponding implementations are – to the best of our knowledge – the first ASIC implementations of GP and OMP. The resulting gain in a mobile
communication system was evaluated by numerical simulations and was shown to considerably increase estimation quality.

Signal restoration When restoring signals in real time, potentially high-throughput requirements must be met. We show that high-definition audio streams can be restored from corruptions by clicks and pops in real-time using fast ASIC implementations with specialized architectures. While serial greedy pursuits are well-suited for very sparse signals, they scale poorly for not very sparse problems, such as for audio signals. AMP was identified as a promising alternative in such cases since it converges quickly, even for less sparse signals. Two architectures are proposed for AMP. The first architecture employs parallel multiply-accumulate units, which are also suitable for recovery problems based on unstructured matrices. The second architecture takes advantage of fast linear transforms, which arise in audio restoration and many other real-world applications. The presented hardware designs of both architectures are not only the first AMP ASIC designs but also the first hardware designs of a BPDN solver. An FPGA implementation of AMP resulted in the first real-time demonstrator for sparse signal recovery.

Localization We also evaluate a new application of sparse signal recovery in localization. We show that CS can improve the detection performance in a passive bistatic radar, which detects the signals emitted from WiFi base stations. We also present an acoustic demonstrator, which sends and receives audio pulses incoherent to point targets. The sparse target scene is then reconstructed on-line on an FPGA.

Non-uniform sampling Beside reconstruction, signal acquisition is a second important part of every CS system. Here, we propose a new sampling device for Fourier-sparse signals, which we refer to as random sampling slope ADC (RSS-ADC). This non-uniform sampler is based on a slope ADC, which is known for its simplicity and very low complexity. We show how its simple hardware structure can be exploited to obtain non-uniformly spaced samples, which are suitable to recover Fourier-sparse signals. Practical issues, such as propagation
delays of components or the influence of signal-dependent sampling times are discussed. Also, suitable reconstruction algorithms are presented, which are adapted for best performance with Fourier-sparse signals and enable the detection of off-grid frequencies.

**Evaluation** Another important contribution is the evaluation and comparison of all the implemented algorithms. Depending on the factor of under-sampling and the signal sparsity, different algorithms perform best. We show that serial greedy pursuits are well-suited for highly sparse signals while AMP better fits moderately sparse signals. Generic implementations of OMP and AMP are presented to compare their hardware complexity and their suitability for VLSI implementation.

Parts of this thesis were previously presented at international conferences and published in journals [63,64,65,66,67,68,69,70].

## 1.9 Outline

After the introduction to CS in this first chapter, sparse channel estimation for LTE mobile communication systems is discussed in Chapter 2, where three ASIC implementations of MP, GP, and OMP are presented. In Chapter 3, signal separation and its application to audio signal restoration is introduced together with corresponding VLSI architectures for AMP. Applications of sparse recovery to localization are discussed in Chapter 4, which includes a WiFi-based passive radar system and an acoustic localization demonstrator. In Chapter 5, the non-uniform sampler RSS-ADC is presented and its properties are analyzed. A comparison of all implemented algorithms is conducted in Chapter 6. Also, two generic FPGA implementations of OMP and AMP are presented and compared. Final conclusions are drawn in Chapter 7.
Chapter 2

Channel Estimation

Broadband wireless systems often operate under channel conditions that are characterized by a sparse channel impulse response (CIR). A sparse CIR appears in the time domain, when only a limited number of reflections is received. In a time-varying channel, each reflector usually moves with a certain velocity, which leads to additional sparsity in the Doppler domain.

CS can then be applied to the estimation of time varying and frequency selective wireless channels. In a first scenario, where pilot signals can be arbitrarily chosen, the number of measurements can be reduced and, thus, the spectral efficiency is increased [71]. As a second possible scenario, where the amount of training is fixed, CS-based channel estimation can improve the quality of the channel estimate through sparsity-aided denoising.

Orthogonal frequency-division multiplexing (OFDM) modulation is often the technology of choice for modern broadband wireless systems, such as in the downlink of the 3rd generation partnership project (3GPP) long term evolution (LTE) standard [72]. The error-rate performance of such a coherent OFDM communication systems depends heavily on the quality of the available channel state information (CSI). While CSI acquisition of broadband channels requires the estimation of many parameters, channel measurements have shown that such channels can often be described by only a small number of propagation paths. Hence, the degrees of freedom of the channel are limited.
Algorithms exploiting this sparsity have recently received significant attention in the context of CS, but have already been considered for channel estimation prior to the CS-era. The benefits of using MP for the estimation of communication channels with a sparse CIR were first demonstrated in [73]. Later, sparsity in the delay-Doppler function of shallow water communication channels was exploited in [74], where MP was used to track sparse reflections in a fast time-varying environment. In [71], CS is applied to the estimation of time-varying and frequency selective channels in multi-carrier systems such as OFDM. The authors show that an approximately sparse representation can be found in the delay-Doppler domain and that, with randomly distributed pilot tones and CS-based estimation, BP optimization achieves better channel estimates with only half the training tones compared to LS estimation. In [75], it was shown that CS-based algorithms such as BP and OMP outperform subspace-based sparse reconstruction algorithms for the estimation of underwater acoustic channels.

Initial hardware implementations of sparse channel estimators in [59] were designed for direct sequence code division multiple access (DS-CDMA) systems. The authors proposed a highly parallel architecture of the MP algorithm on an FPGA to achieve high throughput. Not surprisingly, the authors of [60] state that, for a shallow-water acoustic communication system, a highly parallel FPGA implementation of MP channel estimation outperforms corresponding DSP and microprocessor (XILINX MicroBlaze) implementations in terms of both power consumption and processing time.

In this chapter, ASIC implementations of sparse channel estimators for LTE are presented. First, LTE and a sparse channel model are introduced in Sec. 2.1. The application of sparse recovery algorithms to the estimation of such sparse channels is then described in Sec. 2.2. In the following Sec. 2.3, VLSI architectures for the three greedy algorithms MP, GP, and OMP for channel estimation in a 3GPP LTE system are presented and their hardware complexity and denoising performance is compared. The complexity/performance trade-offs are analyzed using parametrized designs with varying configurations in Sec. 2.4. One configuration of each algorithm was fabricated in a 180 nm process and the corresponding measurements are reported and discussed at the end of this chapter.
2.1. System Model

2.1.1 3GPP LTE Physical Layer

3GPP LTE is an upcoming standard for cellular mobile communication, which enables a high throughput of up to 300 Mbit/s in the downlink from base station (BS) to user and up to 170 Mbit/s in the uplink. Evolved universal terrestrial radio access (E-UTRAN) is the air interface to LTE (OSI layers 1 and 2), which supports bandwidths between 1.4 MHz and 20 MHz [72]. OFDM with up to 2048 sub-channels is employed in the downlink and single carrier-frequency division multiple access is used for the uplink, which enables higher spectral efficiency than previous standards. LTE also supports multiple-input multiple-output (MIMO) transmissions with up to four receive and four transmit antennas. In this work we focus on the single-input single-output downlink, but the same algorithms can easily be extended to the MIMO case.

The subcarriers and the OFDM symbols of the downlink channel are divided into physical resource blocks, which are dynamically allocated to multiple users. One resource block (with a duration of 0.5 ms) consists of 6 to 7 OFDM symbols and 12 subcarriers. Depending on the channel delay spread, short or long cyclic prefixes are chosen.

LTE employs pilot-assisted channel estimation. The training in a resource block of 0.5 ms duration is distributed across frequency and time according to the pattern shown in Fig. 2.1. In one block, four subcarriers are trained, which generates measurements for every third subcarrier. Traditionally, the channel estimates are averaged and interpolated over time and frequency, for example by 2D-Wiener filters.

LTE can operate under a multitude of different channel conditions, from long delay spreads to high speed trains. Here, we are primarily interested in multi-path fading channels with long delay spreads. Three delay profiles are defined in the standard [76], where the extended typical urban model has the highest delay spread of 5 µs with nine delay taps.
2.1.2 Channel Model

Let $L$ be the number of dominant paths with complex-valued gains $a_i$ and delays $\tau_i$. The corresponding frequency-selective CIR can be written as

$$h(\tau) = \sum_{i=1}^{L} a_i \delta(\tau - \tau_i) ,$$  \hspace{1cm} (2.1)$$

where we have neglected the time-dependency of the impulse response. The ideal peaks of the CIR $h(\tau)$ are broadened by low-pass filters in
the transmitter and receiver, respectively (cf. Fig. 2.2), which represent the limited bandwidths. Thus, the effective CIR is given by

\[ \tilde{h}(\tau) = g^r(\tau) * h(\tau) * g^t(\tau) = \sum_{i=1}^{P} a_i g(\tau - \tau_i) \] (2.2)

where \( g^t(\tau) \) and \( g^r(\tau) \) denote transmit and receive filters and \( g(\tau) = (g^r * g^t)(\tau) \). The received signal is given by a convolution of the transmitted signal \( s(t) \) and the channel \( \tilde{h} \) with added complex-valued white Gaussian noise \( n(t) \) with variance \( \sigma_n^2 \).

\[ q(t) = (\tilde{h} * s)(t) + n(t) \] (2.3)

### 2.1.3 OFDM Communication

OFDM divides the available bandwidth \( B = 1/T \) into multiple orthogonal subcarriers, which can be treated as narrow-band channels. One OFDM symbol is the inverse discrete Fourier transform (DFT) of \( D \) subcarriers (tones) with modulated symbols \( c_k \).

\[ s[z] = \frac{1}{\sqrt{D}} \sum_{k=0}^{D-1} c_k e^{j 2\pi \frac{zk}{D}} \] (2.4)

Before transmitting, a cyclic prefix (CP) is added, which prevents inter-symbol interference and allows for easy cyclic deconvolution in frequency domain. To maintain these properties, the cyclic prefix is dimensioned to be longer than the maximal delay spread of the channel.

The OFDM receiver depicted in Fig. 2.3 samples the base-band signal \( q(t) \) with period \( T \). After the cyclic prefix is removed, a discrete Fourier transform converts the samples into the frequency domain:

\[ r_k = \frac{1}{\sqrt{D}} \sum_{n=0}^{D-1} q(nT) e^{-j 2\pi \frac{nk}{D}} \] (2.5)

where \( k \in [0, D - 1] \) is the subcarrier index. For the transmitted symbols \( c_k \), one receives

\[ r_k = \text{FFT}(\tilde{h})_k c_k + \tilde{n}_k . \] (2.6)
CSI is acquired by measuring training symbols. On certain pilot tones with indices $P = [p_1 p_2 \ldots p_M]$ the known symbols $c_P$ are transmitted. The signals received on these pilot tones are fed to the channel estimator. The channel coefficient of the measured tones are then estimated by $y_i = r_{p_i} \cdot c_{p_i}^*/\|c_{p_i}\|^2$. The remaining untrained tones are estimated using, e.g., interpolation. The decoder then recovers the transmitted signal using the output of the channel estimator.

### 2.2 Sparse Recovery in LTE

In order to apply sparse de-noising to LTE channel estimation, we need to formalize the channel estimation as a CS recovery problem. Therefore, one first has to establish the linear relationship between the measurements taken in frequency domain and the time domain basis which allows for a sparse representation of the channel. The measurement matrix $\Phi$ defines which dictionary elements can be selected by sparse recovery algorithms and how they are related to the measurements. An obvious choice for the dictionary elements is the DFT of a single sample of the CIR $\delta[t-kT]$, which implies $x_k = \tilde{h}(kT)$. Note that with this choice, $x$ appears less sparse than $L$ after transmit and receive filtering since each reflection is broadened across a few samples. The columns of the measurement matrix are assumed to be normalized to $\|\Phi_g\|_2 = 1$. This leads to a measurement matrix

$$\Phi_{m,n} = \frac{1}{\sqrt{M}} \exp \left( -j2\pi \frac{p_m(n-1)}{D} \right)$$  \hspace{1cm} (2.7)
with \( m \in [1, M] \), \( n \in [1, N] \) for \( M \) measured pilot tones, and a maximum channel length \( N \). Note that \( \Phi \) is constructed from a \( D \times D \) DFT matrix by selecting only the first \( N \) columns and the \( M \) rows corresponding to the pilot tones.

LTE supports an extended cyclic prefix of 512 samples for 20 MHz bandwidth or 256 samples for 10 MHz bandwidth. Assuming that the maximum CIR length is limited by the length of the cyclic prefix (thus no inter-symbol interference occurs), we end up with maximally \( N = 512 \) dictionary elements. \( M = 400 \) pilot tones are measured in a 2048 sub-channel 20 MHz OFDM system and \( M = 200 \) in a 1024 sub-channel 10 MHz system, which results in a slightly under-determined linear equation. A channel estimate can then be obtained by applying CS recovery algorithms and transforming the estimated channel \( \hat{x} \) back to the Fourier domain \( \hat{f} = \text{FFT}(\hat{x}) \).

### 2.2.1 Recovery Algorithm Selection

The goals of our VLSI implementations are to perform real-time channel estimation with low hardware complexity, i.e., small chip area and low energy dissipation. Therefore, an algorithm needs to converge quickly and must be suitable for hardware implementation (see Sec. 1.5). An additional algorithmic condition for a de-noising application is, of course, robustness to noise.

Convex optimization algorithms require complex control flows and a high computational effort, which makes it very hard to meet the real-time requirement. Thus, the simpler greedy algorithms are preferred for implementation. The goal of this work is to increase the estimation quality of truly sparse channels under noisy condition. Therefore, serial greedy pursuits are the preferred candidates (cf. Chapter 6). The three selected algorithms are MP, GP, and OMP, which were introduced in Sec. 1.5.2.

Further, a suitable and easily implementable stopping criterion must be selected. For our de-noising application, the greedy iterations are stopped as soon as the residual falls below a certain threshold, which is set proportional to the noise level. In this way, only channel taps that are above noise level are taken into consideration. Since the calculation of the \( \ell_2 \)-norm of the residual \( \|r\|_2 \) requires additional hardware, we propose to compare the selected maximum correlation
value to a threshold. This simple criterion approximates the full calculation well and needs no additional computations. Thus, the iteration is stopped as soon as the currently largest element drops below the noise level. An additional constraint on the maximum number of iterations is further used to limit the worst-case processing time.

2.2.2 Simulations

In order to evaluate the performance of the three selected greedy algorithms, numerical Monte-Carlo simulations of the sparse channel estimation were performed. The channel model employed for the simulations is the extended typical urban model with nine propagation paths. All paths were assumed to be subject to Rayleigh fading and white Gaussian noise was added to the received signal. The impulse response \( \tilde{h}(\tau) \) includes root raised-cosine filters \( g^t \) and \( g^r \) with roll-off factor \( \rho = 0.25 \).

The performance of the three algorithms introduced above is compared in Fig. 2.4(a) by plotting the mean squared error (MSE) of the channel estimation against the signal-to-noise ratio (SNR) at the receiver. One can observe that CS channel estimation provides a gain of more than 9 dB in the low SNR regime. At high SNR, the gain is getting smaller since more taps are above noise level. Thus, the estimation problem effectively becomes less sparse. One can observe that GP achieves almost OMP performance while the performance of MP degrades more in the high SNR regime. As a theoretical limit, we also plot a genie-aided LS estimator which operates only on the significant taps, which are assumed to be known to the receiver.

Fig. 2.4(b) plots the average number of iterations performed before the stopping criterion is met. The number of iterations of MP is rapidly increasing for higher SNR, while GP and OMP require significantly fewer iterations.
2.3 VLSI Architectures for Serial Greedy Pursuits

There are two fundamentally different ways how the correlation with all columns of $\Phi$ can be computed in greedy pursuits (cf. line 3 of Alg. 1). First, the straightforward approach is to perform the matrix-vector multiplications in multiply-accumulator (MAC) units. A wide accumulation register allows for high precision. This operation can easily be performed with any degree of parallelism ($\leq N$). Second,
the structure of the measurement matrix allows to use a fast Fourier transform (FFT). On one hand, using an FFT results in a lower number of multiplications. On the other hand, memory requirements are increased since an FFT of size $D$ has to be performed, even if only one quarter of the results are used and an even lower fraction of input values are non-zero. A thorough investigation revealed that no significant saving can be achieved by pruning of butterfly operations in such a scenario [77]. Only with an in/output occupancy of less than 10%, considerable savings in the operation count can be realized by pruning for practically relevant VLSI architectures. To maintain a higher flexibility and to keep the memory requirements low, the direct computation was thus chosen for our implementations. Also, note that an FFT is only used in the first iteration of serial greedy pursuits and that the subsequent updates anyway require MAC units.

First, the hardware architecture for MP is described, which serves as a basis for all further implementations. GP then builds on MP and OMP builds on GP. Thus, the complexity is increasing in each step.

2.3.1 Matching Pursuit

The computationally most expensive operation of the MP algorithm is the first matrix-vector product $\Phi^H r^0$, which correlates the measurements with the dictionary elements. In order to increase the throughput, the correlation is computed on $P$ parallel complex-valued multiply-accumulators (CMACs). This requires the coefficients of the measurement matrix as parallel inputs on all CMACs. Parallel memory access is a bottleneck of this circuit since it cannot be provided by a standard random access memory (RAM). Saving the entire $M \times N$ measurement matrix in a $P$-port RAM would result in a prohibitively large memory. Fortunately, the measurement matrix for the proposed LTE channel estimator has a DFT-like structure (2.7) with a limited number of possible coefficients. Therefore, only the $D$ different DFT factors need to be stored, which number is further reduced by a factor of 4 by exploiting symmetries on the complex-valued plane. Thus, a look-up table (LUT) of size $D/4$ is sufficient to store all required values. The coefficients of $\Phi$ are then generated by first computing the position on the unit circle, where all the $D$ possible DFT coefficients are located, and then accessing the LUT.
Figure 2.5: High-level block diagram of MP and GP (dashed).
After the first iteration, the expensive correlation is replaced by the less complex update function using pre-computed correlation coefficients as described in Sec. 1.5.2. However, this method requires to precompute and store the correlation matrix $\Phi^H\Phi$. Fortunately, this matrix is also highly structured: it is an Hermitian Toeplitz matrix. This allows to reduce the number of stored coefficients to the first row of $\Phi^H\Phi$ – a single vector of length $N$ instead of an $N \times N$ matrix. To perform the updates in parallel, $P$ coefficients have to be generated per clock cycle. To avoid large multi-port RAMs, both the measurement matrix and the update LUT were implemented using random logic.

The implemented architecture is depicted in Fig. 2.5. Most of the computations are performed on $P$ parallel multiply and storage units (MSUs). Each MSU contains a CMAC and a local RAM to store its results. In this way, each MSU works on its designated part of the estimate $g$ without the need for much interaction with other units. The number $P$ of parallel units is configurable at compile time. Thus, by a simple change of a constant, the degree of parallelization can be modified, which allows for an easy exploration of the area-speed trade-off.

Only a few additional hardware units are needed: two blocks to extract the correct values from the two LUTs and one block to determine the maximum value and its index. As the LUTs contain only a subset of all values and exploit symmetries, some inputs arrive inversed and/or complex conjugated at the MSUs. Instead of instantiating units that change signs, which would increase the critical path and area, this information is passed into the MSUs, which reconfigure their complex-valued multipliers and adders on the fly to compensate the inverse signs. Three phases, controlled by a finite state machine (FSM), are required to compute the MP channel estimation. The full implemented algorithm including all described modifications is listed in Alg. 4:

1. **Correlation** In the first iteration, the measurements are correlated with all columns of $\Phi$, which is done in parallel in the MSUs and the results $g^1$ are stored in their Correlation RAMs. The CMACs also compute the squared absolute values, from which the maximum is detected in a specialized unit.
2. Updates The vector $g^n$ is updated using the pre-computed correlation values from the Update LUT. After squaring the entries of $g^n$, the largest component is selected. The newly selected element is added to the Sparse RAM, which is kept small by storing only the coefficients and their indices instead of the full vector $x^n$. Updates are performed until the stopping criterion is met or the maximum supported sparsity is reached.

3. Transformation into frequency domain After the stopping criterion is met and the sparse CIR has been determined, $x^n$ must be transformed back into frequency domain. To this end, the stored sparse elements are multiplied with the corresponding rows of $\Phi$. Due to the sparsity of the coefficients, this operation is calculated more efficiently in our MSUs than by an FFT.

2.3.2 Gradient Pursuit

The same simplification of the correlation as in MP can be applied to GP in every iteration except the first one. However, all coefficients of $x^{n-1}_\Gamma$ might have changed in GP and not only the one added last, as in MP. Therefore, multiple updates must be performed on $g$, which increases the number of computations in each iteration. This fast update procedure implies that the residual $r$ is no longer explicitly computed. The residual is, however, required to determine the amplitude of the gradient. To this end, we reformulated the function $\text{updateX}(S^n)$ in (1.32), which resulted in an expression independent from $r$ and simpler to compute:

$$
\langle r, c \rangle = c^H r = (\Phi_{\Gamma^n} g^n_{\Gamma^n})^H r = (g^n_{\Gamma^n})^H \Phi^H_{\Gamma^n} r
$$

$$
= (g^n_{\Gamma^n})^H g^n_{\Gamma^n} = ||g^n_{\Gamma^n}||^2_2
$$

$$(2.8)$$

$$
\text{updateX}(S^n) = x^{n-1} + \frac{||g^n_{\Gamma^n}||^2_2}{||c^n||^2_2} g^n_{\Gamma^n}
$$

$$(2.9)$$

When implemented with fix-point precision, this algorithm showed numerical instability at the squaring and divide operations. To solve this problem, $g^n_{\Gamma^n}$ and $c^n$ were prescaled by a power of two, determined by the norms of those two numbers in the last iteration. Using this pseudo-floating point method, a stable implementation was possible.
without having to extend the wordwidth. The implemented version of the algorithm is given in Alg. 5.

The MP architecture above can be extended to perform GP reconstruction. The only additional hardware blocks required are a divider and additional memory to store $x_{\text{update}} = x_{\Gamma^n} - x_{\Gamma^{n-1}}$ (added to the Sparse RAM). All other extra operations can be performed with the existing MP hardware. Thus, mainly the FSM has to be adapted. Due to the higher complexity of GP, the number of parallel MSUs must be increased to complete a sufficient number of iterations in the same time as MP.

### 2.3.3 Orthogonal Matching Pursuit

The third and most complex of the presented channel estimators is based on OMP. There is a multitude of implementation variants for the computationally challenging LS estimation (for a survey, see [78]).

- The naive approach is to compute the Moore-Penrose pseudo inverse in each iteration $x^n = \Phi^\dagger \Gamma y = (\Phi^H \Phi)^{-1} \Phi^H \Gamma y$, which includes an expensive matrix inversion operation.
2.3. VLSI ARCHITECTURES

Algorithm 4 Implemented MP
1: $x_0 \leftarrow 0; n \leftarrow 1; G \leftarrow \Phi H \Phi$
2: $g^1 \leftarrow \Phi^H y$
3: $i^1 \leftarrow \arg \max_i |g_i^1|$
4: while $|g_i^n| > \gamma$ and $n \leq I_{\text{max}}$ do
5: $x_i^n \leftarrow x_i^{n-1} + g_i^n$
6: $g^{n+1} \leftarrow g^n - g_i^n G_i^n$
7: $n \leftarrow n + 1$
8: $i^n \leftarrow \arg \max_i |g_i^n|$
end while
10: Output $\hat{f} = \Phi x^{n-1}$

Algorithm 5 Implemented GP
1: $x_0 \leftarrow 0; n \leftarrow 1; \Gamma^0 \leftarrow \emptyset; G \leftarrow \Phi H \Phi$
2: $g^1 \leftarrow \Phi^H y$
3: $i^1 \leftarrow \arg \max_i |g_i^1|$
4: while $|g_i^n| > \gamma$ and $n \leq I_{\text{max}}$ do
5: $\Gamma^n \leftarrow \Gamma^{n-1} \cup i^n$
6: $x_{\text{update}}^n = \frac{|g_{\Gamma^n}|^2}{||e^n||^2_2} g_{\Gamma^n}$
7: $x_{\Gamma^n}^n \leftarrow x_{\Gamma^n}^{n-1} + x_{\text{update}}^n$
8: for $z = 1 \ldots |\Gamma^n|$ do
9: $g \leftarrow g - [x_{\text{update}}^n]_z G_{\Gamma^n}^z$
end for
10: $n \leftarrow n + 1$
12: $i^n \leftarrow \arg \max_i |g_i^n|$
end while
14: Output $\hat{f} = \Phi x^{n-1}$
• Using the matrix inversion lemma, the calculation of the matrix inverse in every iteration can be replaced by a simpler update procedure [33].

• A Cholesky decomposition can be performed on the Gramian matrix $\Phi^H \Phi$, which is Hermitian and positive semidefinite. Therefore, a Cholesky decomposition into the product of a lower triangular matrix $L$ and its Hermitian transposed always exists. The decomposition is amenable to iterative updating, which significantly lowers the complexity. This algorithm can either be implemented to update the residual in each iteration or to directly update the initial correlations.

• QRD decomposes the matrix $\Phi^H \Phi$ into a unitary matrix $Q$ and an upper-triangular matrix $R$. This decomposition can also be efficiently updated in each iteration. Noting that all but the last column of $Q$ are orthogonal to the last residual allows to skip the calculation of the sparse coefficients. As an alternative, the sparse coefficients can be calculated by back-substitution in each iteration combined with a fast update on the correlation coefficients.

The naive approach is very inefficient except for very small problems. Using decompositions is almost always more efficient than the matrix inversion lemma approach [78]. Whether Cholesky or QRD is faster, depends on the exact problem at hand. As QRD scales better with increasing problem size, it was chosen for our implementation.

As in our GP implementation, the fast update of the correlation coefficients should be applied, which is the most efficient approach for very sparse problems. Therefore, back-substitution had to be performed in each iteration. This implementation also allows to build on the previous GP design, which is used to perform correlation, update, and maximum detection. The GP architecture then had to be extended by a QRD unit, which computes the LS optimization in the following way:

$$\Phi^H \Phi = Q \begin{bmatrix} R \\ 0 \end{bmatrix}$$

$$\hat{x} = \Phi^\dagger y = (\Phi^H \Phi)^{-1} \Phi^H y = (R^H Q^H QR)^{-1} R^h Q^H y$$
Algorithm 6 Incremental QRD by Modified Gram-Schmidt

1: \( \mathbf{R}^n \leftarrow \begin{bmatrix} \mathbf{R}^{n-1} & 0 \\ 0 & 0 \end{bmatrix}, \mathbf{\xi}^n \leftarrow \Phi_i \)
2: \textbf{for} \( j = 1, \ldots, n - 1 \) \textbf{do}
3: \( \mathbf{R}^n_{jn} \leftarrow (\mathbf{Q}^{n-1})^H_j \mathbf{\xi}^n \)
4: \( \mathbf{\xi}^n \leftarrow \mathbf{\xi}^n - \mathbf{R}^n_{jn} \mathbf{Q}^{n-1}_j \)
5: \textbf{end for}
6: \( \mathbf{R}^{nn}_n = \sqrt{\|\mathbf{\xi}^n\|^2_2} \)
7: \( \mathbf{Q}^n = [\mathbf{Q}^{n-1} \ ; \ \mathbf{\xi}^n / \mathbf{R}^{nn}_n] \)

\[
\begin{align*}
\mathbf{R}^n \hat{\mathbf{x}} &= \mathbf{Q}^H \mathbf{y} \\
= \mathbf{R}^{-1} \mathbf{Q}^H \mathbf{y}
\end{align*}
\] (2.10)

which translates into a linear system of equations that is easily solved by back-substitution.

\[
\mathbf{R} \hat{\mathbf{x}} = \mathbf{Q}^H \mathbf{y}
\] (2.11)

The QRD is carried out by a modified Gram-Schmidt orthogonalization [79] that is detailed in Alg. 6. Note that the QRD of the last iteration is reused, i.e., only the newly selected dictionary element is added to the decomposition. This saves a considerable number of operations but requires the storage of \( \mathbf{Q} \) and \( \mathbf{R} \). Additional RAMs are required to store \( \mathbf{x} \) and the iteratively rotated \( \mathbf{y} \).

The corresponding block diagram is shown in Fig. 2.6. Two arithmetic units containing CMACs and a coordinate rotation digital computer (CORDIC) serve as basic building blocks of the QRD unit. For the calculation of the norm of a vector, as required by the modified Gram-Schmidt process, and for the division used during back-substitution, a pipelined CORDIC architecture is used. This implementation of the norm calculation avoids the numerical problems (such as an increased dynamic range) associated with squaring followed by the computation of a square root.

2.4 Comparison

Complexity The scaling of the computational complexity and memory requirements of the implementations is analyzed in Tbl. 2.1.

\( K \)
CHAPTER 2. CHANNEL ESTIMATION

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Operations (CMAC operations or division)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP</td>
<td>$MN + N + 2(I - 1)N + KQ$</td>
</tr>
<tr>
<td>GP</td>
<td>$MN + N + \left( \sum_{k=2}^{I} Nk + Mk + k + M + 1 + k \right) + KQ$</td>
</tr>
<tr>
<td>OMP</td>
<td>$MN + N + \left( \sum_{k=2}^{K} Mk + 2kM + 1 + k \right) + KQ$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Memory (words)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP</td>
<td>$N + M + K$</td>
</tr>
<tr>
<td>GP</td>
<td>$N + M + 2K$</td>
</tr>
<tr>
<td>OMP</td>
<td>$N + M + 2K + KM + K^2 + 2M$</td>
</tr>
</tbody>
</table>

Table 2.1: Complexity of serial greedy pursuits.

and $I$ denote the number of added dictionary elements and the number of iterations performed, respectively (with $I \geq K$). $Q$ indicates the number of sub-carriers which finally need to be estimated. While MP shows a linear complexity increase with the number of iterations, the complexity of GP and OMP is growing quadratically. The memory requirements of MP and GP are similarly small compared to the larger additional memory for LS optimization in OMP.

**Fixed-point simulation** All algorithms were implemented with fixed-point arithmetics to obtain good area and power efficiency. Bit-true simulations of the fixed-point algorithms were conducted to evaluate the performance in terms of MSE.

An input wordwidth of 9 bit was used in all designs. All internal wordwidths were optimized to provide near floating-point performance at minimal area. The MP CMACs use 14 bit inputs for each, the real and the imaginary part, 18 bit in the accumulator, and 11 bit for the LUTs. GP requires one more bit in the CMACs. OMP builds on the GP design and uses 17 bit in the additional LS arithmetic units.

In addition to the word-width constraints, the maximum number of iterations $I$ must be constrained in a real-time system to deliver the results within the specified time. Fig. 2.7 compares the fixed-point performance of all the greedy algorithms with different limits on the
maximum number of iterations. As a reference, the MSE of a floating-
point LS estimator is included. The first observation is that a high
number of greedy iterations is only needed in the high SNR regime
since in this regime more taps are significant, i.e., above noise level.
Moreover, MP needs clearly the most iterations and OMP reaches
the lowest MSE. Further increasing the number of iterations will not
result in better performance for the given architectures.

**Synthesis results** In order to obtain comparable implementations,
the same constraint on the processing time is applied to all three
architectures. Here, it is assumed that the reconstruction must have
finished before a new set of measurements is ready, which is after
0.5 ms, the duration of one resource block.

A parametrized VHDL design allowed to synthesize all implemen-
tations with multiple degrees of parallelism. For each configuration
further area/performance trade-offs are achieved by synthesizing the
design for different target clock frequencies. These two parameters,
the number of parallel units and the clock frequency, determine how
many iterations can be performed in the available time, which then
determines the performance of the estimator.

The performance of the synthesized circuits is evaluated by com-
paring the SNR at which the MSE crosses the MSE of a LS esti-
mator, which can be considered as the maximum supported SNR of
the design. Again, the extended typical urban channel model was
employed. The area required for a given performance in a 180nm
CMOS technology is shown in Fig. 2.8. All designs were synthesized
for a channel bandwidth of 10 MHz. The results in Fig. 2.8 suggest
that MP has by far the lowest area requirements. However, to reach
the highest possible performance, GP or OMP must be chosen. They
both show similar performance while OMP requires more than twice
the area of GP. The designs with the best performance utilize 2 MSUs
for MP and 8 MSUs for GP and OMP, respectively.

Table 2.2 compares synthesis results for a fixed performance goal.
We set the SNR at which the MSE lines cross the LS estimate to 20 dB
and determine the necessary number of iteration and the resulting gate
area.
CHAPTER 2. CHANNEL ESTIMATION

Figure 2.7: Comparison of fixed-point greedy algorithms with a maximum of $I$ iterations for 10 MHz bandwidth.

Figure 2.8: Area of the implementations vs. performance (number of parallel MSUs in brackets) for 10 MHz bandwidth. LS crossing is increased by supporting more iterations (annotated at each node).
2.4. COMPARISON

<table>
<thead>
<tr>
<th>Required iterations</th>
<th>MP</th>
<th>GP</th>
<th>OMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area [mm$^2$]</td>
<td>0.374</td>
<td>0.735</td>
<td>1.580</td>
</tr>
</tbody>
</table>

Table 2.2: Comparison of synthesis results of all three implementations with an LS crossing at 20 dB.

<table>
<thead>
<tr>
<th>Complexity [kGE]</th>
<th>MP</th>
<th>GP</th>
<th>OMP (LS only)</th>
<th>OMP (est.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core area [mm$^2$]</td>
<td>0.73</td>
<td>1.21</td>
<td>1.21</td>
<td>2.42</td>
</tr>
<tr>
<td>Max. frequency [MHz]</td>
<td>140</td>
<td>128</td>
<td>166</td>
<td>128</td>
</tr>
<tr>
<td>Power [mW]</td>
<td>88</td>
<td>209</td>
<td>196</td>
<td>200</td>
</tr>
<tr>
<td>Energy/ch.est. [$\mu$J]</td>
<td>44</td>
<td>104</td>
<td>98</td>
<td>202</td>
</tr>
</tbody>
</table>

Table 2.3: Chip figures for 180 nm 1P6M CMOS and estimation of full OMP.

2.4.1 ASIC Implementation Results

For all three algorithms, one configuration was chosen to be fabricated in a 180 nm CMOS technology. Due to chip size constraints, the 10 MHz mode (half of the maximum LTE bandwidth) was chosen. Doubling the bandwidth to the full 20 MHz means doubling both $M$ and $N$ while the sparsity $K$ remains approximately constant. The complexity then scales according to Tbl. 2.1. All implementations contain full scan chains to make sure circuits are testable. The most important figures of these implementations are compiled in Tbl. 2.3. The OMP chip contains only the LS part of the algorithm. Combining this part with a chip of GP’s size results in a full OMP implementation. Estimations of the corresponding figures are shown in the last column of Tbl. 2.3. The achievable speed and implemented memory sizes allow up to 50, 18, and 10 iterations for MP, GP, and OMP, respectively. This results in LS intersections at 24, 25, and 18 dB. Thus, OMP is limited by the number of possible iterations. Compared to a full LTE
baseband receiver, such as the network-on-chip-based implementation in 65 nm CMOS with an area of 29.6 mm$^2$ [80] (roughly 227 mm$^2$ in 180 nm CMOS technology), the occupied area of the proposed channel estimators is very low.

2.5 Summary

In this chapter, sparsity-based channel estimation using greedy pursuits is discussed. We demonstrate the de-noising capabilities of this approach on LTE channel estimation by means of simulations. The algorithms were compared using the LS crossing point as a figure of merit. This quantity states up to which noise level a CS-based implementation is better than a LS estimation. Three serial greedy pursuits are then implemented and synthesized with different degrees of parallelism and sparsity support.

Efficient VLSI architectures for the sparse recovery algorithms MP, GP, and OMP are proposed. They all rely on parallel CMAC units, on-the-fly generated measurement matrix coefficients, and fast update procedures. Generating the measurement matrix instead of storing it strongly reduces storage requirements. While MP required the lowest chip area, GP achieved a better performance using a three times higher area. The performance of the OMP implementation was similar to GP in this applications even though OMP uses a much larger area.

For all three implementations it was shown that especially in the low SNR regime, significant gains of up to 7 dB in the MSE of channel estimates could be obtained compared to LS estimation. The small MP is sufficient if the system is operated in the low-SNR regime. For better performance at high SNR, GP is recommended. The occupied silicon area of all implementations is small compared to full baseband processors.
Figure 2.9: Micrographs of the fabricated 180 nm chips performing serial greedy pursuits.
Chapter 3

Signal Restoration

In this chapter, VLSI architectures for the restoration of corrupted signals are presented. In contrast to the previous chapter, we assume here that not only the signal itself is sparse, but that also the corruption thereof is structured and has a sparse representation. In this way, it is possible to separate the desired part of a corrupted signal from the corruption and to restore the original signal [4]. Signal separation has been employed to restore images, audio, and communication signals from corruptions, such as impulse noise or saturation [81, 4, 82]. Since this type of noise has a concentrated local support, separation from the original signal with smoother characteristics becomes possible.

The corresponding restoration problem can be formulated as a sparse signal recovery problem, which is shown in Sec. 3.1. The restoration of audio signals corrupted by clicks and pops is used as an example application throughout this chapter and is introduced in Sec. 3.1.1. VLSI architectures suitable for real-time restoration of corrupted audio signals are presented in Sec. 3.2. While the last chapter focused on serial greedy pursuits, here AMP is chosen for implementation because it promises better performance for applications featuring less sparse signals, such as audio signals, images, or videos. Two generic AMP architectures were designed, which are then optimized for the audio restoration application. The first architecture, referred to as AMP-M, is a general-purpose solution employing MAC units, which is suitable for sparse signal recovery problems relying on
arbitrary (e.g., unstructured or random) measurement matrices. The
second architecture, referred to as AMP-T, is specifically designed for
recovery problems for which the multiplications with a measurement
matrix can be replaced by fast transform algorithms. Both architec-
tures were implemented on an ASIC as well as on an FPGA (Sec. 3.3).
The ASIC designs deliver real-time restoration of 192 ksample/s stereo
signals while the FPGA implementations supports real-time restora-
tion of 44.1 ksample/s stereo signals.

3.1 Sparse Signal Restoration

As shown in [81, 4, 82], signals corrupted by structured noise can be
modeled as

\[ y = Aa + Bb + n = \Phi x + n. \]  

(3.1)

Here, \( y \in \mathbb{R}^M \) denotes the corrupted observations, the matrix \( A \in \mathbb{R}^{M \times N_a} \) sparsifies the signal \( s = Aa \) to be restored and the matrix \( B \in \mathbb{R}^{M \times N_b} \) sparsifies the corruptions on the signal \( s \). Unstructured additive (measurement) noise, i.e. minor contributions which are
not sparsified in either basis, is denoted by \( n \in \mathbb{R}^M \). The signal
restoration problem is formulated as an ordinary CS recovery problem
by setting \( \Phi = [A \ B] \), \( x = [a^T \ b^T]^T \). Signal restoration now
amounts to the recovery of the sparse vector \( x \) from (3.1) using one of
the algorithms introduced in Sec. 1.5, followed by computing \( s = Aa \).

In certain cases, one can identify the locations of the corrupted entries
in \( z \) prior to recovery, which typically results in improved restoration
performance [4].

For the restoration to succeed, the matrix \( A \) must not only sparsify
the uncorrupted signal \( s \), but must also be incoherent to \( B \), which
corresponds to a small mutual coherence [4].

\[ \mu_m = \max_{k,\ell} \frac{|A_k^h B_{\ell}|}{\|A_k\|_2 \|B_{\ell}\|_2} \]  

(3.2)

This condition allows one to select a matrix pair \( A, B \) that is suitable
for the given signal restoration application. Thus, signal restoration
corresponds to the separation of a signal into two components, the
desired part and the corruption, based on incoherent sparsifying bases.
3.1. SPARSE SIGNAL RESTORATION

3.1.1 Audio Declicking

Audio signals from speech and music can often be sparsely represented in a discrete cosine basis. If these signals are corrupted by clicks or pops, which is typical for old phonograph recordings, restoration of the original signal by signal separation becomes possible.

For the restoration of audio signals from clicks/pops and saturation, setting $A$ to the $M \times M$ (unitary) discrete cosine transform (DCT) matrix defined as

$$A_{k,\ell} = \sqrt{\frac{c_k}{M}} \cos\left(\frac{(2\ell - 1)(k - 1)\pi}{2M}\right)$$

with $c_k = 1$ for $k = 1$ and $c_k = 2$ otherwise, enables one to sparsify audio signals; setting $B = I_M$ sparsifies clicks/pops and saturation errors. Since $A$ is incoherent to $B$, excellent performance for audio restoration can be achieved by using this pair of matrices. In particular, as shown in [4,82], if the number of corrupted entries of $z$ is small compared to its dimension $M$, the DCT–identity pair is guaranteed to enable stable recovery of $x = [at^T \ b^T]^T$ and, hence, to restore the original signal $s = Aa$.

3.1.2 Evaluation of AMP and Early Termination

Since the goal of our application is to restore corrupted audio signals in real-time, an algorithm with very fast convergence must be found that works even with comparatively small and only approximate sparsity. AMP proved to perform well under these conditions, which we demonstrate in this section.

The performance and complexity of AMP for audio restoration is studied in Fig. 3.1, where the restoration of an artificially corrupted 16 bit 44.1 ksample/s speech signal from [83] is evaluated. Restoration is performed by using the DCT–identity pair in blocks of length $M = 512$, which showed good reconstruction performance with reasonable hardware (memory) demands. 16 samples between each pair of adjacent blocks are overlapped using a tapered cosine (i.e., Tukey) windows to avoid artifacts at the block boundaries. Each block is recovered using AMP with $I_{\text{max}} = 20$ iterations and $\lambda = 2$. The clicks/pops are generated as Gaussian pulses consisting of five samples.
(a) Convergence behavior for different click rates and ET thresholds

(b) Impact of ET to the RSNR

Figure 3.1: Convergence behavior and impact of early termination (ET) to the RSNR performance of AMP for sparsity-based audio restoration.
whose peak value is uniformly distributed in $[-1,1]$. We define the click rate $0 \leq r_c \ll 1$ as the average number of clicks per sample. The restoration performance is measured using the recovery signal-to-noise-ratio (RSNR), defined as

$$\text{RSNR} = \frac{\|s\|^2}{\|s - \hat{s}\|^2},$$

(3.4)

where $s$ corresponds to the original (uncorrupted) signal and $\hat{s}$ to the signal restored by AMP. Figure 3.1(a) shows the RSNR for different numbers of maximum iterations $I_{\text{max}}$ (the regularization parameter $\lambda$ has been optimized for each click rate $r_c$). One can immediately see that AMP converges quickly; i.e., setting $I_{\text{max}} = 20$ turns out to be sufficient for near-optimal performance for the click rates considered. The RSNR of the corrupted signals are 3.2 dB and 7.2 dB for $r_c = 1/100$ and $r_c = 1/400$, respectively. Thus, gains of up to 9 dB are possible with the given example and $I_{\text{max}} = 20$.

**Early Termination** One way to reduce computational complexity is to apply early termination (ET), which allows to stop iterations before the maximum allowed number $I_{\text{max}}$ is achieved. To this end, the RMSE of the residual is compared to a threshold $\gamma$. When the RMSE falls below $\gamma$, the algorithm is considered to have converged. Since the RMSE is calculated anyway for the threshold $\theta$, ET comes at very little additional hardware complexity. The impact of ET on the performance and complexity is studied in Fig. 3.1. We see that, for a threshold $\gamma = 0.02$, the number of average iterations $I_{\text{avg}}$ is strongly reduced while only slightly degrading the RSNR. Lowering the ET threshold leads to a smaller reduction of average iterations $I_{\text{avg}}$ but results in higher RSNR. Thus, carefully selecting $\gamma$ enables one to reduce the complexity of AMP at no loss in terms of RSNR. For example, Fig. 3.1(b) shows that for $r_c = 1/400$, $\gamma = 0.006$, and $I_{\text{max}} = 25$, only 16.2 iterations are necessary to achieve the same performance of $I_{\text{max}} = 20$ without ET. Hence, in practice, ET can either be used to increase the average restoration throughput or for power reduction by silencing the entire circuit during idle clock cycles.
3.2 VLSI Architectures for AMP

In this section, two VLSI architectures for the AMP algorithm are presented. The first architecture, referred to as AMP-M, is a generic MAC-based solution that is applicable to arbitrary sparsity-based signal restoration and CS problems. The second architecture, referred to as AMP-T, is a generic solution for situations where multiplications of a vector with $\Phi$ and $\Phi^T$ can be carried out by a fast transform. First, the architectural principles of both approaches are introduced, which apply to any application, especially CS recovery. In a second step, the architecture is optimized for the audio declicking application. A diagram of the AMP algorithm is given in Fig. 3.2, which shows the order of execution and possible parallel execution in AMP.

3.2.1 MAC-Based Architecture

The first architecture implements the matrix-vector multiplications on lines 4 and 6 of Alg. 3 using a pre-defined number of parallel MAC units. This MAC-based architecture has the advantage of being suitable for arbitrary matrices $\Phi$, including unstructured matrices or matrices obtained through dictionary learning [84], as used in many signal restoration or de-noising problems. Moreover, if $\Phi$ is explicitly stored in a memory, the matrices used in AMP-M can be reconfigured.
3.2. VLSI ARCHITECTURES FOR AMP

Figure 3.3 shows the high-level block diagram of the AMP-M architecture. At run-time, without the need to re-design and re-implement the circuit.

Figure 3.3 shows the high-level block diagram of the AMP-M architecture. The AMP algorithm requires memories to store the input signal $y$, the residual $r^i$, and the signal estimate $x^i$ obtained after applying the thresholding function. The input vector $y$ and the residual $r^i$ can be stored in the same memory (referred to as YR-RAM in Fig. 3.3), i.e., each coefficient of $z$ and $r^i$ can be stored at the same address. The signal estimate $x^i$ is stored in a separate memory, referred to as X-RAM. Depending on the application, the entries of the matrix $\Phi$ are either stored in a RAM, a LUT, or a read-only memory (ROM), or may be generated on the fly.

All matrix-vector multiplications are carried out in $P$ parallel MAC instances; the number of MAC units is configurable during compile-time of the architecture and determines the maximum achievable throughput (see Sec. 3.3.2). Pipeline registers are added at the multiplier inputs to increase the maximum achievable clock frequency. Each MAC unit is used to sequentially compute an inner product of a row of the matrix $\Phi$ with $x^i$ or $\Phi^T$ with $r^{i-1}$. Hence, each MAC unit requires access to a different entry of $\Phi$ in each clock cycle, while the same vector entry is shared among all units (see Fig. 3.3).
CHAPTER 3. SIGNAL RESTORATION

The RMSE is computed using a separate unit that is specialized for computing sums of squares. The subsequent square root computation is implemented using the approximation developed in [85], which requires neither multipliers nor LUTs. The RMSE is computed in parallel to the matrix-vector multiplication \( \Phi^T r_{i-1} \) (line 4 of Alg. 3). Note that the rather limited numerical accuracy of the square-root approximation deployed was found to be sufficient for our purposes (cf. Sec. 3.3.1).

To implement the thresholding function (1.34), we instantiated a subtract-compare-select unit that applies thresholding in a serial and element-wise manner (performed in the TRSH unit). The \( \ell_0\)-norm on line 5 of Alg. 3 is computed in the L0-unit, which counts the non-zero entries of \( \mathbf{x}^i \) in a serial manner and concurrently to the matrix-vector multiplications. To avoid additional hardware resources, all remaining arithmetic operations, e.g., computation of the residual \( r_i \) (line 6 of Alg. 3), are performed using the available MAC units.

**Optimization for audio restoration**

The main bottleneck of the AMP-M architecture is the memory bandwidth required to deliver the entries of \( \Phi \) to the parallel MAC units. For unstructured matrices, multi-port LUTs, ROMs, or on-chip static random access memory (S-RAM) can be used for small dimensions. For large-dimensional problems, external memories become necessary, which allows for higher capacities. However, implementing an external memory interface with a high enough bandwidth for all parallel multipliers becomes very challenging. Fortunately, in many real-world applications, the matrix \( \Phi \) is highly structured, which often enables on-the-fly generation of its coefficients at very high rate.

Specifically, for the DCT–identity pair used for audio declicking, one can avoid the explicit storage of all entries of the DCT matrix (which would result in prohibitively large on-chip memory). Instead, the regular structure of the DCT matrix is exploited and symmetries are used to generate the \( M \times 2M \) matrix \( \Phi \) at high rate by a small cosine LUT having only \( M \) entries. The LUT address is calculated on the basis of the row and column indices of the required DCT entry. The parallel LUT outputs (or their negative values) are directly fed to the MAC units. Thus, instead of a multi-port \( M \times 2M \) memory for
explicitly storing $\Phi$, only $M$ values needed to be stored; this results in a 1024× memory-size reduction for a block size of $M = 512$ samples. The multiplications with the identity basis obviously do not require any memory and are implemented by simple control logic.

3.2.2 Transform-Based Architecture

While the AMP-M architecture is well-suited for unstructured matrices, small-scale problems, or applications for which the matrix $\Phi$ must be re-configurable at run time, the complexity scaling, storage requirements, and memory bandwidth requirements (which are roughly proportional to the number of entries $MN$ of the matrix $\Phi$) render its application difficult for throughput-intensive or large-scale problems. Fortunately, in many practical applications the multiplication with matrix $\Phi$ can be replaced by a fast transform, e.g., the fast Fourier, DCT, Hadamard, or wavelet transform (or combinations thereof), which allows for the design of more efficient VLSI implementations. The AMP-T architecture described next exploits these advantages.

Figure 3.4 shows the high-level block diagram of the AMP-T architecture. The structure of AMP-T is similar to that of the AMP-M architecture, apart from the following key differences:
• The parallel MAC units have been replaced by a specialized fast transform unit, which must support one of the above fast forward transform and its inverse.

• Neither memory for the matrix $\Phi$ nor logic to generate its entries on the fly are required.

• The residual, which was calculated in the MAC units in the AMP-M architecture, is now computed in a dedicated unit (referred to as R-CALC); this unit only consists of a small multiplier and a few adders.

• The RMSE is calculated simultaneously to the fast forward transform, whereas the "$\ell_0$"-norm is computed simultaneously to the fast inverse transform.

The architecture for carrying out the fast forward transform and its inverse heavily depends on the transform and algorithm used. Hence, AMP-T is less flexible compared to AMP-M, as the transform unit must be re-designed for each target application. However, as shown in Sec. 3.3, AMP-T has the potential to substantially decreases the complexity of AMP-M.

Optimization for audio restoration

For the audio restoration application considered here, we use an architecture implementing a fast DCT (FCT) and its inverse (IFCT). The corresponding algorithm and the resulting VLSI architecture are detailed in Sec. 3.2.3. The simple operations required to implement the identity basis are carried out in the R-CALC unit. The X-RAM has been divided into two memories to support parallel access, which enables fast interleaved thresholding of the two parts of $x = [a^T \ b^T]^T$.

3.2.3 VLSI Design of the FCT/IFCT

In order to complete the AMP-T design for our audio declicker, an efficient implementation of the FCT/IFCT must be found. Most of the existing VLSI implementations of a fast DCT/inverse discrete cosine transform (IDCT) were designed for MPEG-2 video compression, which evaluates problems of size $8 \times 8$ (see, e.g., [86]). For the targeted
3.2. VLSI ARCHITECTURES FOR AMP

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Regularity</th>
<th>Memory</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix-vector multiplication</td>
<td>++</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Direct approach [87]</td>
<td>--</td>
<td>--</td>
<td>++</td>
</tr>
<tr>
<td>Recursive method [88]</td>
<td>--</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Via 2M-point FFT [89]</td>
<td>+</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Via M-point FFT [90]</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Via M/2-point FFT [90]</td>
<td>+</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

Table 3.1: High-level comparison of FCT/IFCT algorithms, where positive signs indicate better suitability for hardware implementation.

audio restoration application, however, the problem size is \( M = 512 \) for which—to the best of our knowledge—no VLSI architecture has been described in the open literature. To this end, we first evaluate potential algorithms for efficiently computing a large-dimensional FCT/IFCT. The algorithm chosen for our implementation is then summarized and a corresponding VLSI architecture is presented.

Algorithm evaluation

A variety of algorithms to compute the FCT/IFCT have been proposed in the literature [87,88,89,90]. A high-level comparison of some of the most prominent algorithms is provided in Tbl. 3.1. We consider the algorithm’s regularity, memory requirements, and computational complexity (processing effort). While the computational complexity is a key metric for most implementations, regularity and low memory requirements are of similar importance when designing dedicated VLSI circuits. As a reference, we compare all candidate algorithms to a straightforward matrix-vector multiplication-based DCT/IDCT approach, as it is used in AMP-M.

The algorithm proposed in [87] follows a divide-and-conquer approach, which is directly applied to the DCT matrix, and which achieves a very low computational complexity. This algorithm, however, exhibits a very irregular data flow and corresponding architectures cannot easily be parametrized to support different problem sizes.
Another direct approach is the recursive method proposed in [88], which is more efficient than [87] (in terms of operation count and memory), but still lacks a regular data flow. Another line of fast DCT algorithms relies on the well-established FFT. A straightforward approach is based on a \(2M\)-point FFT, which exhibits high regularity and requires almost no overhead for the DCT-to-FFT conversion [89].

An improved algorithm relying on an \(M\)-dimensional FFT only, was proposed in [90]. This approach results in a lower computational effort while causing only a small conversion overhead. An even faster method replaces the real-valued FFT by a complex-valued \(M/2\)-dimensional FFT followed by a few additional computations [90]. This approach reduces the computational complexity and memory requirements compared to the \(M\)-FFT approach, while maintaining high regularity. Hence, we decided to use this \(M/2\)-FFT-based algorithm in the AMP-T realization.
\[ a = \text{FCT}(x) \quad x, c', a \in \mathbb{R}^M, \quad f' \in \mathbb{C}^M, \quad f, c \in \mathbb{C}^{M/2} \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reorder</td>
<td>[ c'<em>k = \begin{cases} x</em>{2k-1} &amp; k = 1, \ldots, M/2 \ x_{2(M-k)+2} &amp; k = M/2 + 1, \ldots, M \end{cases} ]</td>
</tr>
<tr>
<td>Reduce</td>
<td>[ c_k = c'<em>{2k-1} + jc'</em>{2k} \quad k = 1, \ldots, M/2 ]</td>
</tr>
<tr>
<td>FFT</td>
<td>[ f = \text{FFT}(c) ]</td>
</tr>
<tr>
<td>Expand</td>
<td>[ f'<em>k = \begin{cases} f_k + f^*</em>{M/2-k+2} - j(t^M_k)^{-1}(f_k - f^<em>_{M/2-k+2}) &amp; k = 1, \ldots, M/2 \ f^</em>_{k-M/2} &amp; k = M/2 + 1, \ldots, M \end{cases} ]</td>
</tr>
<tr>
<td>Rotate</td>
<td>[ a_k = \begin{cases} \frac{1}{\sqrt{M}} \mathcal{R}{f'_1} &amp; k = 1 \ \frac{2}{M} \mathcal{R}{(t^A_M)^{-1}f'_k} &amp; k = 2, \ldots, M \end{cases} ]</td>
</tr>
</tbody>
</table>

\[ x = \text{IFCT}(a) \quad x, c', a \in \mathbb{R}^M, \quad f' \in \mathbb{C}^M, \quad f, c \in \mathbb{C}^{M/2} \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inv. rotate</td>
<td>[ f'<em>k = \begin{cases} \sqrt{M}a_1 &amp; k = 1 \ \frac{M}{2}t^A_k (a_k - ja</em>{M-k+2}) &amp; k = 2, \ldots, M \end{cases} ]</td>
</tr>
<tr>
<td>Inv. expand</td>
<td>[ f_k = \frac{1}{2} \left( f_k + f^<em>_{M/2-k+2} + j\frac{M}{2}t^A_k (f_k - f^</em>_{M/2-k+2}) \right) \quad k = 1, \ldots, M/2 ]</td>
</tr>
<tr>
<td>IFFT</td>
<td>[ c = \text{IFFT}(f) ]</td>
</tr>
<tr>
<td>Inv. reduce</td>
<td>[ c'_k = \begin{cases} \mathcal{R}{c_k} &amp; k = 1, 3, \ldots, M - 1 \ \mathcal{I}{c_k} &amp; k = 2, 4, \ldots, M \end{cases} ]</td>
</tr>
<tr>
<td>Inv. reorder</td>
<td>[ x_k = \begin{cases} c'<em>k &amp; k = 1, \ldots, M/2 \ c'</em>{M+M/2-k+1} &amp; k = M/2 + 1, \ldots, M \end{cases} ]</td>
</tr>
</tbody>
</table>

Table 3.2: All steps with formulas to compute the FCT/IFCT according to [90].
CHAPTER 3. SIGNAL RESTORATION

\textit{M/2-FFT-based FCT/IFCT algorithm}

The implemented \(M/2\)-FFT approach is summarized in Fig. 3.5 and detailed in Tbl. 3.2 with all steps required to perform the FCT and IFCT. In order to compute an FCT, the entries of the real-valued input vector \(x\) are first reordered in a new vector \(c'\). Then, the reordered vector is converted into a complex-valued vector \(c\) of half the length, i.e., \(M/2\). The main task of the FCT algorithm is to compute a \(M/2\)-length complex-valued FFT of the vector \(c\). The result \(f\) is then expanded into a conjugate-symmetric vector \(f'\), which corresponds to a \(M\)-point FFT of the real-valued vector \(c'\). To obtain the result \(a\) of the FCT, the entries of \(f'\) are rotated by certain twiddle factors (as used in the FFT), defined as

\[
\hat{t}_k^M = \exp\left(2\pi j \frac{k - 1}{M}\right),
\]

followed by extracting the real value of the rotated entries of \(f'\). The procedure for the IFCT is analogous to that of the FCT; see Tbl. 3.2 for the details. Redundant operations, such as the computation and storage of conjugate-symmetric vectors are skipped in the hardware implementation.

\textbf{Architecture}

In the following, we present a VLSI architecture performing the \(M/2\)-FFT-based FCT/IFCT algorithm. To process a 192 ksample/s stereo audio signal with a block size of \(M = 512\) samples and 16 samples overlap, restoration of one block must be completed within 1.29 ms. With \(I_{\text{max}} = 20\), an FCT/IFCT operation must be computed in no more than 32.3 \(\mu\)s.

Figure 3.6 shows the high-level block diagram of an FCT/IFCT architecture achieving the specified throughput in 65 nm CMOS technology. The input vectors and intermediate results are stored in a single-port S-RAM with \(M/2 = 256\) complex-valued entries. The address generator computes the FFT addressing scheme proposed in [91] and also controls the operations carried out during the other stages of the FCT/IFCT algorithm.
Figure 3.6: High-level block diagram of FCT/IFCT unit. The highlighted block corresponds to the time-shared radix-2 FFT butterfly.

The complex-valued $M/2$ in-place FFT/inverse fast Fourier transform (IFFT) is performed using a single radix-2 butterfly in a time-shared fashion. With a single memory access per clock cycle, each butterfly operation requires four clock cycles. The complex-valued multiplier in the butterfly is implemented using four real-valued multipliers and two adders. All the additional operations (carried out in the Reorder, Reduce, Expand, and Rotate phases) are also calculated on the same butterfly unit by re-using the existing arithmetic circuitry. A dedicated twiddle generator unit is used to provide the necessary factors for the FFT as well as for the FCT/IFCT steps in Tbl. 3.2. This unit contains a real-valued LUT with 512 entries; the real and imaginary parts of the twiddle factors are assembled from two consecutive table look-ups.

The resulting architecture computes an $M = 512$ FCT/IFCT in 8,200 clock cycles, which is sufficiently fast for the targeted audio restoration assuming a clock frequency of at least 255 MHz. We note that for applications requiring substantially higher throughput, such as for sparse signal recovery of high-resolution images or videos, significantly faster FFT/IFFT architectures become necessary; this can be achieved by parallel and higher-order butterfly units, as well as by
using parallel multi-port S-RAM macro cells. The FCT/IFCT architecture developed here enables us to achieve the specified throughput at minimum silicon area and, hence, was chosen for the VLSI design of AMP-T.

3.3 Hardware Implementation Results

In this section, reference VLSI and FPGA implementation results of AMP-M and AMP-T for the audio restoration application described in Sec. 3.1.1 are provided. The achieved throughput, area, and power numbers can also be applied to the more generic case of a recovery from compressive measurements.

3.3.1 Fixed-Point Parameters for Audio Restoration

In order to optimize hardware and power efficiency, fixed-point arithmetic is employed in both AMP architectures. The targeted audio restoration application uses input and output word widths of 16 bit. In AMP-M, the word width of the accumulator in the MAC units is critical. Setting it to 30 bit allowed to closely match floating-point performance at the output. The most critical word length of the AMP-T architecture resides in the DCT/IDCT unit. The simulation results in Fig. 3.7 show that using 26 bit for the real and imaginary part in the butterfly equals floating-point precision. Most of the remaining word lengths are identical in both architectures. The Y-RAM uses 16 bit, the XU-, XL-, and R-RAM use 26 bit word width.

An interesting case is the word-length optimization of the threshold $\theta$, which is also plotted in Fig. 3.7. One can see that 11 bit are sufficient to achieve near-floating point performance. These moderate accuracy requirements allow us to employ the fast and low-area square-root approximation described in [85]. In the example plotted, the implementation loss of the final implementation in terms of RSNR is less than 0.013 dB compared to the floating-point model. Note that Fig. 3.7 shows the reconstruction of a music signal with a gain of 17 dB, which is higher than the gain obtained with the speech signal evaluated in Fig. 3.1.
3.3. HARDWARE IMPLEMENTATION RESULTS

In the final designs, the regularization parameter \( \lambda \) and the ET threshold \( \gamma \) are both configurable at run-time: \( \lambda \) is settable to the values \( \{0.125, 0.25, 0.5, 1, 2, 4\} \), whereas \( \gamma \) can be adjusted to any positive number representable by 13 fractional bits.

3.3.2 Comparison of AMP-M and AMP-T

In order to compare the hardware complexity of AMP-M and AMP-T, both architectures were synthesized in a 65 nm CMOS technology. The target throughput for both designs was set such that real-time restoration of audio signals can be performed with different sampling rates up to 384 ksample/s, which corresponds to a high-quality 192 ksample/s stereo signal. For AMP-M, the throughput can be increased by instantiating more parallel MAC units. In order to process a single audio channel with 48 ksample/s in real-time, four parallel MAC units are required; processing 384 ksample/s in real-time necessitates 32 parallel MAC units. The throughput of AMP-T can be adjusted (up to a certain speed) by reducing the critical path of the AMP-T architecture during synthesis.

Fig. 3.8 shows the standard cell and memory area after synthesis of AMP-M and AMP-T. The number of required MAC units in AMP-M is annotated in parentheses. When targeting a high throughput, the silicon area of AMP-T is substantially smaller than that of AMP-M. This is mainly due to the reduced number of operations required in the DCT/IDCT compared to matrix-vector multiplications. This behavior is also reflected in the number of clock cycles required by AMP-M and AMP-T, which can be approximated as follows:

\[
C_{\text{AMP-M}} \approx 2I_{\text{max}}(4M + M^2/P) + M
\]

(3.6)

\[
C_{\text{AMP-T}} \approx 2I_{\text{max}}(7M + M \log_2(M) + 2) + M.
\]

(3.7)

Here, \( P \) is the number of parallel MAC units. For large \( M \), AMP-M scales with \( I_{\text{max}}M^2/P \) and AMP-T with \( I_{\text{max}}M \log_2(M) \). Therefore, the transform-based architecture requires less operation for a large \( M \), which allows for smaller and more energy-efficient implementations.
Figure 3.7: Word-length optimization for DCT and threshold $\theta$.

Figure 3.8: Synthesis results of AMP-M and AMP-T for real-time audio restoration in 65 nm 1P8M CMOS technology. The numbers given in parentheses correspond to the number of MAC units.
### 3.3. HARDWARE IMPLEMENTATION RESULTS

<table>
<thead>
<tr>
<th></th>
<th>AMP-M</th>
<th>AMP-T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. clock freq. [MHz]</td>
<td>333</td>
<td>256</td>
</tr>
<tr>
<td>Throughput [ksample/s]</td>
<td>397</td>
<td>399</td>
</tr>
<tr>
<td>Cell area(^a) [kGE]</td>
<td>303</td>
<td>33.9</td>
</tr>
<tr>
<td>Memories [kB]</td>
<td>5.76</td>
<td>7.25</td>
</tr>
<tr>
<td>Core area [mm(^2)]</td>
<td>0.629</td>
<td>0.136</td>
</tr>
<tr>
<td>Power consumption [mW]</td>
<td>177.5</td>
<td>24.4</td>
</tr>
<tr>
<td>Energy efficiency [(\mu)J/sample]</td>
<td>0.447</td>
<td>0.061</td>
</tr>
</tbody>
</table>

\(^a\)Standard cells only, excluding S-RAM macro cells. 1 GE equals 1.44 \(\mu\)m\(^2\).

Table 3.3: Post-layout implementation results in 1P8M 65 nm CMOS.

### 3.3.3 ASIC Implementation

A reference ASIC including both designs for real-time audio restoration was developed in 65 nm CMOS technology with 1 polysilicon and 8 metal layers (1P8M). The target is to process 192 ksample/s stereo audio signals with 16 samples overlap between adjacent blocks, which requires an AMP-throughput of 396 ksample/s. During back-end design, both architectures were integrated onto the same silicon chip. Fig. 3.9 shows the corresponding chip layout, where the main processing blocks and all RAMs are highlighted. The corresponding post-layout results are listed in Tbl. 3.3. A detailed area and power breakdown of both ASIC designs is provided in Tbl. 3.4.

From Tbl. 3.3, one can see that both designs achieve the specified target throughput of 396 ksample/s. AMP-M runs at a higher clock frequency of 333 MHz compared to 256 MHz for AMP-T, since more pipelining stages are used in AMP-M. The most important difference is, however, that the chip area of AMP-T is roughly five times smaller than that of AMP-M. Note that AMP-M requires less memory compared to AMP-T, which is due to the facts that we do not store the DCT matrix in AMP-M but compute its entries on the fly from synthesized LUTs, and AMP-T requires an additional memory within the FCT/IFCT unit.
Figure 3.9: Layout of the audio declicking ASIC containing the AMP-M and AMP-T designs in 1P8M 65 nm CMOS technology.
### 3.3. HARDWARE IMPLEMENTATION RESULTS

<table>
<thead>
<tr>
<th></th>
<th>AMP-M</th>
<th></th>
<th>AMP-T</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>kGE(^a) (%)</td>
<td>mW (%)</td>
<td>kGE(^a) (%)</td>
<td>mW (%)</td>
</tr>
<tr>
<td>32 MAC units</td>
<td>197 (65)</td>
<td>77 (43)</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Φ matrix gen.</td>
<td>94 (31)</td>
<td>88.1 (50)</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>FCT/IFCT unit</td>
<td>–</td>
<td>–</td>
<td>24.5 (72)</td>
<td>16.5 (68)</td>
</tr>
<tr>
<td>– butterfly</td>
<td>–</td>
<td>–</td>
<td>16.6</td>
<td>9</td>
</tr>
<tr>
<td>– twiddle gen.</td>
<td>–</td>
<td>–</td>
<td>2.6</td>
<td>0.7</td>
</tr>
<tr>
<td>RMSE</td>
<td>3 (1)</td>
<td>1.2 (1)</td>
<td>2.9 (9)</td>
<td>0.5 (2)</td>
</tr>
<tr>
<td>RAMs (X,R,Z)</td>
<td>–</td>
<td>7.9 (4)</td>
<td>–</td>
<td>6.2 (25)</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>9 (3)</td>
<td>3.8 (2)</td>
<td>6.5 (19)</td>
<td>1.2 (5)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>303 (100)</td>
<td>178 (100)</td>
<td><strong>33.9 (100)</strong></td>
<td><strong>24.4 (100)</strong></td>
</tr>
</tbody>
</table>

\(^a\)Standard cells only, excluding S-RAM macro cells.

Table 3.4: Cell area and power breakdown of the individual designs.

The power figures shown in Tbl. 3.3 and Tbl. 3.4 are extracted from post-layout simulations using node activities obtained from simulations with actual audio data at maximum clock frequency, 1.2 V core voltage, and at 25\(^\circ\) C. AMP-T turns out to be roughly 7\(×\) more energy efficient than AMP-M in terms of \(\mu\)J per sample, which highlights the advantages of the fast-transform based approach used in the AMP-T design.

From Tbl. 3.4 we see that the FCT/IFCT unit of AMP-T occupies almost 3/4 of the overall circuit area. The remaining blocks, i.e., RMSE calculation and thresholding, make up for around 1/4 of the design. In the AMP-M ASIC, almost 2/3 of the circuit area is occupied by the 32 parallel MAC units and almost 1/3 is required to generate the entries of the DCT matrix.

### 3.3.4 FPGA Implementation

Both AMP architectures were ported to a Xilinx Spartan-6 FPGA, which is fabricated in a 45 nm low-power CMOS technology. We slightly adapted both designs to better fit the FPGA structure in order
to improve throughput, which will be explained later in this section. The basic parameters, such as the number of MAC units in AMP-M or the FCT architecture in AMP-T are, however, equivalent to those of the ASIC design, which makes the implementations comparable.

A large benefit of FPGA implementations is the ability to quickly test applications on prototyping boards. Our target platform is a Digilent Atlys board featuring a Xilinx Spartan-6 (XC6SLX45) FPGA and an AC’97 audio chip (LM4550 by Texas Instruments) containing multiple ADCs and digital-to-analog converters (DACs). We thus included an AC’97 audio interface, which allowed us to process audio signals from analog audio sources in real time. Stereo audio data is fed into the line-in port, sampled at 44.1 ksample/s, restored using AMP, and then fed to a DAC. A block diagram of the audio interface is shown in Fig. 3.10. The interface receives bit-serial data, which are converted into parallel busses and buffered in RAMs configured as first-in first-out (FIFO) chains which are required since AMP operates block-wise. Additional small FIFOs and arithmetic units are needed to process and recombine overlapping blocks. Instead of tapered cosine windows, windows with linearly fading edges over 8 samples are applied due to their more economical hardware implementation. The left and right channel of the stereo audio stream are processed in an interleaved manner on the same time-shared AMP unit. An FSM controls the writing and reading to and from the FIFOs as well as the execution of the AMP block.

AMP-T optimization

In the FCT/IFCT block, we replaced the single-port S-RAMs with dual-port memories as dual-port block RAMs are readily available in the FPGA used. This modification enables a speed-up of the FCT/IFCT by a factor of 2, since each butterfly operation is now computed in two instead of four clock cycles. The number of clock cycles required by this modified AMP-T architecture is approximately

\[
C_{\text{AMP-T2}} \approx 2I_{\text{max}} \left( 5M + \frac{M}{2} \log_2(M) + 2 \right) + M, \quad (3.8)
\]
3.3. HARDWARE IMPLEMENTATION RESULTS

Figure 3.10: Block diagram of the audio interface for the AMP declicker prototype.

which is 40% fewer compared to the architecture used in the AMP-T ASIC. Except for an additional adder in the butterfly, FPGA logic usage is almost not increased.

**AMP-M optimizations**

In the AMP-M architecture, the synthesized 32-port LUT in the \( \Phi \) matrix generator is replaced by 16 dual-port ROMs. Moreover, additional pipeline registers are introduced after the multipliers, which increases the maximum clock frequency by 20% while slightly increasing the processing latency (i.e., less than 1%).

**Comparison**

The FPGA implementation results of the two optimized designs when mapped to the smallest possible Xilinx Spartan-6 FPGA are shown in Tbl. 3.5. AMP-T fits on a very small XC6SLX9 FPGA, whereas AMP-M requires the much larger XC6SLX75 FPGA. The optimized AMP-T architecture is able to process stereo signals with the standard sampling rate of 44.1 ksample/s. Despite of the larger FPGA, AMP-M achieves only half the throughput, which, however, still allows us to process one audio channel in real time. For stereo processing, the
Table 3.5: Implementation results for Xilinx Spartan-6 FPGAs.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>AMP-M</th>
<th>AMP-T</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPGA type</td>
<td>XC6SLX75</td>
<td>XC6SLX9</td>
</tr>
<tr>
<td>Clock freq. [MHz]</td>
<td>41.2</td>
<td>35.4</td>
</tr>
<tr>
<td>Throughput [ksample/s]</td>
<td>47.9</td>
<td>92.8</td>
</tr>
<tr>
<td>Occupied slices</td>
<td>3525</td>
<td>1200</td>
</tr>
<tr>
<td>Block RAMs</td>
<td>24</td>
<td>11</td>
</tr>
<tr>
<td>DSP blocks</td>
<td>132</td>
<td>15</td>
</tr>
<tr>
<td>Power consumption[^a] [mW]</td>
<td>516</td>
<td>96</td>
</tr>
<tr>
<td>Energy efficiency [µJ/sample]</td>
<td>10.8</td>
<td>1.05</td>
</tr>
</tbody>
</table>

\[^a\]Power consumption is measured on a Digilent Atlys platform featuring an XC6SLX45 FPGA (a down-sized version of AMP-M with 4 MAC units was measured; the power figures were scaled to 32 MAC units). Since other devices are connected to the same power supply, only the difference between an active and inactive AMP core is reported.

number of MAC units must be doubled, which would require an FPGA of twice the logic capacity. The extra audio interface requires only 140 logic slices, 2 RAM blocks, and a single DSP slice. A picture of the Digilent Atlys prototype board setup performing real-time audio restoration is given in Fig. 3.11. Only AMP-T is small enough to fit onto the provided FPGA.

Power measurements were conducted using the integrated power monitor of the prototype board. The resulting power consumption and energy efficiency is reported in Tbl. 3.5, which demonstrates that AMP-T is roughly ten times more energy efficient than AMP-M. Hence, if the flexibility advantage of AMP-M is not required, then the AMP-T architecture is the preferred solution for FPGA implementations with respect to complexity, throughput, and power consumption.

Note that both FPGA designs only achieve 1/4 and 1/8 of the throughput of AMP-T and AMP-M compared to the ASIC designs, even though the FPGAs are fabricated in a more advanced 45 nm technology compared to our 65 nm ASIC implementation. An even more
3.4 Summary

We have presented two architectures, AMP-M and AMP-T, which are both able to recover sparse signals using the AMP algorithm. AMP-T includes a fast cosine transform of size 512, for which no VLSI designs have been presented in the open literature yet. We, therefore, conducted an evaluation of different DCT algorithms to select the most suitable for our design. AMP-M relies on the processing power of many parallel multipliers. AMP is well suitable for parallelization as long as the multiplication coefficients are provided in parallel and at high rate. To this end, a specialized coefficient generator was developed, based on small LUTs.

Figure 3.11: Setup for real-time audio declicking with FPGA prototype board and stereo headphones.

A pronounced difference can be observed in terms of energy efficiency. Specifically, both ASIC designs outperform the FPGA implementations by a factor of 17 and 24 for AMP-T and AMP-M, respectively.
Figure 3.12: Audio recovery example for the old phonograph recording “Mussorgsky” from [92]; a snapshot of the original (naturally corrupted) signal and the signal restored by AMP using the DCT–identity pair are shown.

To demonstrate the capability of AMP for signal recovery, the designs were targeted at the recovery of audio signals corrupted by clicks and pops. Fig. 3.12 shows a snapshot of the signal waveform of the old phonograph recording “Mussorgsky” from [92] and the signal restored by AMP. This illustrates the fact that AMP using the DCT–identity pair efficiently removes clicks/pops from old, naturally corrupted phonograph recordings without any knowledge of the locations of the sparse corruptions. Our FPGA prototype performs this operation in real time, which makes it the first real-time realization of a sparse signal restoration algorithm.
Chapter 4

Localization

Localization is a promising application of sparse signal processing since the number of interesting targets to be localized is typically very small. Thus, there are only a few parameters to be estimated while many measurements are required. The resulting problem is closely linked to channel estimation discussed in Chapter 2. The principle of both applications is that signals are reflected by a few objects and superpositions of these reflections are measured. In both applications CS methods enable higher precision and a reduced number of measurements.

Real-time detection of targets from CS measurements requires fast VLSI implementations. Since most relevant VLSI aspects are already discussed in Chapter 2, the focus of this chapter is on new localization applications which benefit from sparse recovery.

Applications of CS to classical radar were already extensively discussed in literature [55, 93, 56, 94, 95]. A short review thereof is given in Sec. 4.1. Based on this discussion, new applications to WiFi-based passive radars are introduced in Sec. 4.2. We also developed an acoustic demonstrator for sparsity-based localization, which is presented in Sec. 4.3.
CHAPTER 4. LOCALIZATION

4.1 Radar

Ideas for exploiting sparsity and taking incoherent measurements have been around for a long time in the radar literature (some of them are reviewed in [93]). With the introduction of CS, those ideas gained momentum. After a brief comment on sparsity, the most prominent applications of CS to radar are reviewed.

4.1.1 Sparsity

In order to translate the radar detection problem into a CS problem, a discrete and finite-dimensional representation of the radar scene must be found, that results in a sparse vector $x$. In many radar applications, this representation is established by a discrete grid defined in the range-Doppler plane. The range is obtained from the round-trip time of the radar signal and the Doppler shift is measured by the frequency offset of the returned signal. Thus, an object is located on the delay-Doppler plane using the tuple $(\Delta t, \Delta f)$. Assuming sparsity on the range-Doppler grid means that each object can be modeled as a point target located on a grid point and having a certain discrete velocity. This is a reasonable assumption for many small targets and fine enough grids. In order to obtain truly sparse representations, the grid must usually be chosen finer than what the bandwidth or ambiguity function (Sec. 4.2.2) suggest. A four times overcomplete measurement matrix (i.e., four times more atoms are used than the number needed to span the whole solution space) has been shown to work well in many cases (see, e.g., [56]). Especially in indoor environments, many reflections will be received from walls, ceilings, etc., that cannot be sparsely represented. Therefore, all clutter must be removed before sparse recovery algorithms can be applied. Clutter removal is achieved using background subtraction methods or by simply ignoring objects without Doppler shift (i.e., $\Delta f = 0$).

Defining a sparse representation in a synthetic aperture radar image is less straightforward. While many scenes are sparse in a wavelet domain, noisy reflections from objects with a rough surface, known as speckle, make sparse representations impossible. This noise is generated by reflections on many small objects within one pixel of resolution which can add up destructively or constructively, resulting
in an exponentially distributed amplitude. Fortunately, most objects of interest are man-made with smooth surfaces. They reflect phase-coherent waves and allow for an approximately sparse representation in the wavelet domain [96].

4.1.2 Sensing Schemes

Three acquisition schemes for compressive measurements are shown in Fig. 4.1 and discussed in the following paragraphs. All three schemes aim at reducing the number of measurements.

**Incoherent pulses** The first paper on CS radar imaging [55] proposed a system which transmits a PN or chirp sequence. Such sequences are incoherent to the impulse response of point targets on a range-Doppler grid. The proposed receiver can then be implemented
as a low-rate ADC\footnote{While the sampling rate of the ADC can be low, the supported input bandwidth must still cover the full band. This leads to intentional aliasing.} sampling below the Nyquist frequency. However, sub-sampling at the receiver leads to a loss in the receiver SNR via noise folding \cite{97}. \cite{95} proposed transmitting incoherent Alltop sequences and receiving reflections with a Nyquist-rate ADC. CS-based reconstruction then allows for higher detection performance.

**Linear projection**  Compressive measurements can reduce the sample acquisition rate by projecting the high-bandwidth signal at the receiver onto a PN sequence, integrating the result, and sampling it at a lower frequency. This approach is known as random demodulator \cite{48}. In order to gather enough information, the signals usually needs to be projected onto multiple sequences in parallel. Using the example of a ground penetrating radar, \cite{94} increased the resolution with this sampling scheme. While only very simple short pulses need to be transmitted, the receivers need additional elements for the linear projection, such as high-bandwidth mixers.

**Pulse compression**  Step-frequency radars transmit pulses at discrete frequencies. CS-based schemes proposed, e.g. in \cite{56}, reduce the number of transmitted frequencies by selecting and transmitting only a random subset thereof. The different frequencies are either transmitted sequentially or at the same time. Thus, either measurement time or receiver complexity can be reduced. A similar pulse compression scheme was applied to through-wall imaging \cite{98}. In a stepped-frequency system, the measured frequencies as well as the measured positions on an array were reduced. From a transmit energy point of view, this system is more efficient than the incoherent pulses or linear projection approaches; the whole bandwidth of the transmitted signal is captured and no SNR loss occurs at the receiver.

### 4.2 WiFi-Based Passive Bistatic Radar

Passive radars offer a low-cost alternative to the commonly used active systems. The signals of existing radio transmitters (such as TV
stations or mobile communication BS) and their reflections on targets are analyzed by a passive radar receiver to determine the location of the targets. While passive radars are easy to set up and their presence is difficult to detect, they have to deal with non-cooperative transmitters, which puts higher demands on signal processing.

Since WiFi [99] BSs are becoming ubiquitous nowadays, exploiting their signals might be interesting for local area surveillance. The first experiments to detect humans using WiFi signals were conducted in [100]. The regularly broadcasted beacons, which identify the BS to potential users, were utilized for localization. By correlating the signal received with the transmitted one, the detection of one person was possible in a wide open field without much clutter. Experiments in high clutter indoor environments with passive bistatic radars using OFDM-modulated WiFi signals were conducted in [101], where it was possible to detected one moving person even through a wall. Two persons moving in opposite directions were, however, already difficult to detect. Experimental results of a WiFi-based passive radar system detecting range and speed of a target in a parking lot were presented in [102]. It was demonstrated that a moving vehicle can be detected, but a person standing next to it is severely masked by the strong reflection of the car. Only the joint application of disturbance removal techniques and ambiguity function control filters allowed for a detection of the moving vehicle and the person.

Thus, the goal of our work is to enable the detection of closely spaced targets, even when a strong reflector masks weaker reflectors in its vicinity. This goal should be achieved using CS and the fact that target scenes are often sparse. To this end, the suitability of WiFi signals for incoherent measurements must be analyzed and corresponding measurement schemes must be chosen.

A similar passive radar system for OFDM signals using CS was proposed in [103]. Digital video/audio broadcasting (DVB/DAB) stations are used as illuminators to identify flying targets. The number of symbols that must be known at the receiver from trainings or after decoding using outer codes was reduced while maintaining good detection accuracy.

The WiFi setup considered here is shown in Fig. 4.2. We measure the differential bistatic delay \( \tau = (r_{tx} + r_{rx} - r_d)/c \) (the delay between
the reference signal and the reflection), which translates into the target’s position, and the Doppler frequency to obtain its velocity.

### 4.2.1 WiFi Standards

The IEEE 802.11 WiFi standards [99] use direct sequence spread spectrum (DSSS) modulation in 802.11b with 11 MHz bandwidth and OFDM with 20 MHz bandwidth in 802.11a/g/n (802.11n additionally supports a 40 MHz mode, which we will not further consider here).

**OFDM** was introduced in Sec. 2.1.3 for the LTE downlink. In 802.11a/g/n, the available bandwidth is divided into $N_S = 64$ sub-carriers, of which $N_T = 52$ are occupied. A cyclic prefix of 0.8 $\mu$s is added to a symbol of 3.2 $\mu$s duration. At the beginning of every 802.11 frame, known training symbols are transmitted that allow the receiver to obtain a channel estimate.

**DSSS** spreads a symbol by multiplying it with a chip sequence of higher frequency. In 802.11b, an 11-chip Barker code is used:

$$b = [+1, -1, +1, +1, -1, +1, +1, -1, -1, -1]$$

Transmitting a symbol, thus, takes 1 $\mu$s. Data is modulated using differential binary/quadratic phase shift keying. Training symbols are transmitted in an up to 144 bit long preamble, containing the synchronization (SYNC) and start of frame delimiter (SFD) fields.
4.2. WIFI-BASED PASSIVE BISTATIC RADAR

4.2.2 Ambiguity Function

The *ambiguity function* of a radar signal characterizes how well radar targets with a certain delay and Doppler difference can be distinguished. An ambiguity function analysis of WiFi signals was conducted in [104]. The range resolution obtained in measurements was 25.4 m (27.3 m theoretical) for DSSS beacons and 18.8 m (18.1 m theoretical) for OFDM frames. The theoretical values are determined by the inverse of the actually occupied bandwidth $\Delta r = c_0/B$. The Doppler resolution is determined by the observation time; the observation of many subsequent frames is required to obtain a good velocity resolution. In both the range and Doppler dimensions, relatively large sidelobes were identified, which explains the masking of closely spaced targets observed in [101,102].

4.2.3 High Accuracy WiFi Radar

In this section, WiFi signals are analyzed for their suitability for compressive measurements with the objective of increasing target detection accuracy. Since we aim at a high-resolution WiFi-radar, OFDM-based standards are preferred due to their higher bandwidth and, thus, better resolution. In an OFDM training symbol, there are $N_T = 52$ out of $N_S = 64$ OFDM subcarriers trained. The untrained subcarriers are not used for data transmission and serve as guard band against adjacent channels or to skip the direct conversion subcarrier. Transforming the resulting estimate into the time domain will reveal all reflections and CS reconstruction can identify the ones corresponding to radar targets. With sparsity in a discrete time basis and measurements performed in a subset of the discrete Fourier basis, incoherence of the measurement matrix is given (cf. Sec. 1.4.3). Thus, measuring trained subcarriers meets CS preconditions for range estimations. The resulting system is similar to pulse compression in a step-frequency radar with slight undersampling.

Previous WiFi-based passive radars [102] required a reference signal obtained from an antenna placed right next to the BS. This is, however, unnecessary in the described setup since only training tones are used, which are known to the receiver.
Since we are relying on training symbols, the maximum supported delay spread (and thus the maximum supported range) is defined by the length of the cyclic prefix. In the long training phase, a double length cyclic prefix of duration $T_t = 1.6 \mu s$ is used [99]. This results in a maximum supported range of $r_{\text{max}} = r_{\text{tx}} + r_{\text{rx}} - r_d = T_t c_0 = 480$ m.

Formulation as CS problem

In order to obtain a CS formulation, range as well as Doppler frequencies must be described on a discrete grid. The sparse vector $\mathbf{x}$ is composed of all the delay profiles $\mathbf{x}_v \in \mathbb{R}^R$ (with $R$ data points in the range dimension) at all considered Doppler frequencies $\omega_v, \ v \in [1,V]$ (with $V$ data points in the Doppler dimension) stacked on top of each other (see (4.2)). This results in a vector $\mathbf{x}$ of size $N = RV$ with a sparsity $K$ equal to the number of targets. The measurements vector $\mathbf{y}$ contains all the measured subcarriers from $S$ subsequent frames, recorded at times $t_s, \ s \in [1,S]$. Thus, the measurement vector has a dimension of $M = SN_T$.

Next, the measurement matrix is constructed, which includes multiple discrete Fourier transforms. The Doppler shift within one training sequence is assumed to be negligible (as in [103])\footnote{Assuming a maximum velocity of 100 km/h, the maximal phase shift during the 8 $\mu$s training is $\Delta \omega = 2\pi v/c_0 T f_c = 0.011$, which is negligible.}. We define a DFT matrix $\mathbf{F}$, which contains only a subset $\mathcal{S} = \{s_1 s_2 \ldots s_{N_T}\}$ of trained subcarriers and that limits the length of the delay profile to $R$.

$$F_{m,n} = \frac{1}{\sqrt{N_T}} \exp \left( -j2\pi \frac{s_m \cdot (n-1)}{N_S} \right) \quad (4.1)$$

with $m \in [1,N_T], \ n \in [1,R]$ for $N_T$ measured pilot tones and a delay profile length $R \leq N_S$. $\mathbf{F}$ is constructed from a $N_S \times N_S$ DFT matrix by selecting only the first $P$ columns and the $N_T$ rows corresponding
4.2. WIFI-BASED PASSIVE BISTATIC RADAR

to the pilot tones. $\Phi$ establishes a linear relation between the measurements $y_s$ of a training symbol at time $t_s$, $s \in [1, S]$ (assuming $t_1 = 0$) and the range profile $x_v$ at different Doppler shifts $\omega_v$, $v \in [1, V]$.

$$
\begin{bmatrix}
  y_1 \\
  y_2 \\
  \vdots \\
  y_S
\end{bmatrix}
= 
\begin{bmatrix}
  F \\
  e^{j\omega_1 t_2} F \\
  \vdots \\
  e^{j\omega_1 t_S} F
\end{bmatrix}
\begin{bmatrix}
  \vdots \\
  \vdots \\
  \vdots \\
  \vdots
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_V
\end{bmatrix}
$$

Target detection is, thus, performed by identifying the support of $x$ using CS recovery algorithms.

**Simulations**

A simulation environment for an OFDM WiFi passive bistatic radar was set up in order to test and evaluate CS-based target localization algorithms. The targets were distributed on arbitrary positions on the range-Doppler plane. Since they are not necessarily positioned on grid points, a fourfold overcomplete measurement matrix is applied (cf. Sec. 4.1.1). With a 20 MHz sampling rate, the delay profile is represented by 32 samples, which increases to $R = 128$ for fourfold overcompletion. A maximum velocity of 100 km/h and a resolution of 0.5 m/s in $r_{rx}$ direction result in $V = 61$ grid points in the Doppler domain. Vector $x$, thus, has a dimension of $N = RV = 7807$. Assuming that 20 frames are recorded at random times within a 50 ms period, an underdetermined problem with only $M = SN_T = 20 \cdot 52 = 1040$ measurements results. The sparse delay profile $x$ is then recovered using (4.2) and MP [32]. Suppression of highly correlated neighbors of selected targets is applied (as in [105]) for better reconstruction results and FFTs are used instead of matrix-vector multiplications for fast processing speed.

In Fig. 4.3, CS reconstruction of a sparse scene with six targets with different reflectivity (indicated by their color intensity) is compared to the image produced by correlating the received and transmitted signals. One can clearly see that sidelobes in the ambiguity function mask closely spaced targets in the correlation image, while
CS is still able to identify all six targets. The effect of noise together with the performance of different CS reconstruction algorithms is studied in Fig. 4.4, where the percentage of correctly detected targets (arranged as in Fig. 4.3) is evaluated. In this case, OMP shows the best performance, closely followed by MP.

4.2.4 Multi-Channel WiFi Radar

CS-aided post processing can simplify the identification of targets as shown above but it cannot go much beyond the resolution limits set by the ambiguity function. To overcome the rather coarse resolution, increasing the measured bandwidth seems to be the only viable solution.

In urban environments multiple BS, transmitting in different channels, are usually within range. Recording the signals on all available channels increases the observed bandwidth and, thus, increases the possible resolution. Training symbols are transmitted on the various channels at various times in an uncoordinated manner. This leads to measurements of certain subcarriers while others are left unobserved, which corresponds to the principle of pulse compression in a step-frequency radar. CS reconstruction algorithms can ‘fill in’ the holes and reproduce a sparse scene with the accuracy offered by the entire available bandwidth instead of only one channel.

We consider the 2.4 GHz industrial, scientific and medical (ISM) band. The trained subcarriers of channels 1 through 13 (permitted in Europe) span a total bandwidth of 76.6 MHz. This leads to a theoretical range resolution of \( \Delta r = c_0 / 76.6 \text{ MHz} = 3.92 \text{ m} \). The measurement matrix defined in (4.2) must now be extended to include measurements performed in different channels, which is achieved by only a few changes on how the subset of trained subcarriers is defined in (4.1). First, the sampling rate is increased from 20 MHz to 80 MHz. (Note that this is only the sampling rate at which \( x_v \) is represented; a physical sampling rate of 20 MHz is still possible if the center frequency is tunable.) Those 80 MHz are subdivided into \( N_S = 4 \cdot 64 = 256 \) subcarriers. For each received frame, a subset \( S \subset [1, N_S] \) of \( N_T = 52 \) subcarriers is measured. The partial DFT
4.2. WIFI-BASED PASSIVE BISTATIC RADAR

Figure 4.3: OFDM passive radar simulation: correlation-based vs. CS-based detection ($S = 20$ frames, noise-free).

Figure 4.4: OFDM passive radar simulation: SNR sweep with various greedy reconstruction algorithms.
matrix $\mathbf{F}_S$ contains only the columns corresponding to the measured subcarriers in the set $\mathcal{S}$.

\[
\begin{bmatrix}
y_1 \\
y_2 \\
\vdots \\
y_L
\end{bmatrix} =
\begin{bmatrix}
\mathbf{F}_{S_1} & \cdots & \mathbf{F}_{S_1} \\
e^{j\omega_1 t_2} \mathbf{F}_{S_2} & \cdots & e^{j\omega_1 t_2} \mathbf{F}_{S_2} \\
e^{j\omega_1 t_3} \mathbf{F}_{S_3} & \cdots & e^{j\omega_1 t_3} \mathbf{F}_{S_3} \\
\vdots & \cdots & \vdots \\
e^{j\omega_V t_S} \mathbf{F}_{S} & \cdots & e^{j\omega_V t_S} \mathbf{F}_{S}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_V
\end{bmatrix}
\] (4.3)

Simulations with a four times higher resolution compared to the single channel case were performed. The results after measuring 20 frames randomly distributed on channels 1, 5, 9, and 13 with 0 dB SNR are presented in Fig. 4.5. CS-based detection resulted in an exact recovery of five of the six targets and a good approximation of the sixth target.

### 4.2.5 Reduced Sampling Rate

When we are not aiming at an increased accuracy but at a reduced number of measurements, a different sampling scheme needs to be applied. Under normal circumstances, achieving the necessary sampling
4.2. **WIFI-BASED PASSIVE BISTATIC RADAR**

---

![Diagram of Nyquist sampling and subsampling]

**Figure 4.6**: Subsampling of a Baker DSSS signal.

---

![Graph showing Ground truth, Correlation, and CS]

**Figure 4.7**: DSSS passive radar simulation: correlation and CS-based detection with SNR 10 dB and $S = 10$ frames.

---

Rates for WiFi-signals is no problem. However, one could think of energy-starved sensor nodes, where reducing the number of acquired samples might be helpful to save power and memory.

Simply reducing the sampling rate is not possible for OFDM-based training symbols. It is, however, possible for DSSS training frames, as illustrated in Fig. 4.6. As the known training is spread over a large number of chips, only a fraction thereof must be captured. Transmitting DSSS-modulated frames corresponds to transmitting an incoherent PN sequence (as in the ‘incoherent pulse’ scheme). The measurement matrix $\Phi$ then contains shifted and subsampled versions of the transmitted Barker PN code at various Doppler shifts.
The simulated system records the SYNC and SFD fields of an 802.11b frame [99]. It is assumed that 10 such frames are recorded within 50 ms. The example shown in Fig. 4.7 contains 4 targets and receives signals with an SNR of 10 dB. Even with an undersampling factor of 128, CS reconstruction allows for a perfect detection of all targets. Note that higher undersampling lowers the received SNR. Thus, this scheme is only possible in environments with low noise.

4.3 Acoustic Demonstrator

In order to show the capabilities of the CS-based approach to radar detection, an acoustic demonstrator was developed. This allowed for faster and easier prototyping compared to a conventional electromagnetic radar. The acoustic demonstrator consists of a loudspeaker and a linear array of 16 microphones, which enables target detection on a two-dimensional plane. The setup with the custom printed circuit board (PCB) is depicted in Fig. 4.8. Each of the omni-directional microphones is attached to a pre-amplifier on a removable module. Two 8-channel ADCs convert the signals from the microphones and a DAC provides the signal to the loudspeaker. The converters are connected to a Xilinx Spartan-3 500E FPGA. In a first version, this
4.3. **ACOUSTIC DEMONSTRATOR**

FPGA contained only an interface from the DAC and ADCs to a PC via a universal serial bus (USB). In a second version, the collected data is processed online on the FPGA and the result is presented on an attached display.

### 4.3.1 System Architecture

The localization system is able to detect objects on a plane in front of the microphone array. The targets are modeled as sparse point reflectors on a 2-dimensional grid. The most suitable sensing scheme in the given setup is the transmission of PN sequences, which are incoherent to point targets. Maximum length ±1 sequences proved to be a good choice due to their good auto-correlation properties and efficient generation in hardware.

Conventional object detection amounts to correlation with the expected PN sequences. Using a CS approach, the sparsity of the scene is exploited to achieve better detection performance. In the assumed highly sparse environment, simulations showed that MP delivers very good results. The main computational bottle-neck is thus the multiplication with the large measurement matrix. The measurement matrix resulting from the described setup consists of shifted versions of the transmitted PN sequence. Multiplication with the measurement matrix amounts to a correlation of the recordings at all microphones with shifted versions of the same PN sequence. Note that the best performance is only achieved by including the filter effects of loudspeaker and microphone in the reference sequence.

The naive but computationally expensive approach is to iterate over each grid point on the plane and perform correlations with the echo that would be expected from the given point. A faster approach is to calculate the cross-correlation with the reference sequence once for each microphone and then iteratively add up the results at correct shifts for every grid point. Even faster implementations can be achieved when the correlation is performed in the Fourier domain via FFTs as a cyclic convolution. Using this FFT-based approach, it is possible to perform the measurements with a PN sequence of length 255 and 4 MP iterations within 47.5 ms on the small Spartan-3 FPGA.
The corresponding VLSI architecture [106] is depicted in Fig. 4.9. The speaker interface controls the DAC and transmits the PN sequence to the loudspeaker. The echoes are then received over serial interfaces from the two ADCs. The received signals are stored in block RAMs after a room echo subtraction was performed. A dedicated FFT unit is used to perform correlation via frequency domain multiplications. Two real-valued sequences are transformed at the same time by packing them into a complex-valued signal. The expected response is already stored in frequency domain representation (in the PN Seq. RAM). After transforming the result back to time domain, the correlations are stored in the correlation RAM. The target detection unit searches for the location with the strongest reflection by iterating through all grid points and assembling the correlations from all microphones with a certain offset. Once a target is detected, its contribution is subtracted from the original recordings. This requires the storage of the expected sequence in time domain (in the PN seq. RAM). The location of the detected targets is finally presented on a screen attached via VGA port.
4.3. ACOUSTIC DEMONSTRATOR

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock freq. [MHz]</td>
<td>33</td>
</tr>
<tr>
<td>Occupied slices</td>
<td>3492 (75%)</td>
</tr>
<tr>
<td>Block RAMs</td>
<td>20 (100%)</td>
</tr>
<tr>
<td>DSP blocks</td>
<td>12 (60%)</td>
</tr>
</tbody>
</table>

Table 4.1: Implementation results for Xilinx Spartan-3 XC3S500E FPGA.

Multiple iterations of this procedure are performed in order to detect more objects. In each iteration one target is detected and the contribution to the measurements is sequentially removed. This procedure increases the visibility of weaker reflections which would be masked by strong reflectors otherwise.

The resulting usage of FPGA resources is listed in Tbl. 4.1. All block RAM is used, in the largest part by the memories for recordings and correlation values from all microphones. Multiple clock domains are required for the audio and video interfaces. The signal processing blocks run at the listed frequency of 33 MHz. This allows for an update rate of 21 frames per second when using four MP iterations.

4.3.2 Detection Performance

The system is configured to detect objects in an area of $2\text{m} \times 2\text{m}$ using a $40 \times 40$ grid. The microphones are arranged on a uniformly spaced linear array of length 27 cm. While a larger array leads to better resolution, randomized placements of the microphones did not show any positive effects; the PN sequences seem to produce enough incoherence for CS recovery. An example for the detection of two targets in shown in Fig. 4.10. Two cardboard boxes were placed in front of the microphone array at a distance of 1–2 m. While the correlation image in Fig. 4.10(b) does not clearly reveal the target’s localization, the image in Fig. 4.10(c) after CS-based target detection shows the position of both reflectors in the room.
Figure 4.10: Images produced by the acoustic demonstrator for the localization of two objects.
4.4 Summary

In this chapter, we evaluated how sparse representations can enhance the performance of localization systems. We started by reviewing the CS radar literature and identified three schemes: incoherent pulses, linear projections, and pulse compression. As a first application, WiFi-based passive radar was evaluated. It has been shown that the OFDM trainings can be considered as incoherent pulses and that they allow to increase the detection accuracy when CS recovery is applied. Especially closely spaced targets were separated better than using correlation techniques. A further increase in precision was obtained by the inclusion of multiple WiFi channels from the ISM band.

The second application is the acoustic demonstrator. In contrast to WiFi, full control over the transmitted sequence was available. The chosen sequence, incoherent to point targets, was a maximum length sequence, which enabled the detection of multiple reflectors in front of a microphone array.

In both applications the sparsity of the target scene is exploited in order to obtain a better picture of the scene. Sparse signal recovery algorithms were shown to be well-suited for localization, especially since sidelobes of reflecting objects are efficiently removed.
Chapter 5

Non-Uniform Sampling

Up to now, we were focusing on reconstruction algorithms. The measurements were assumed to be available in a suitable basis (such as in channel estimation or OFDM-based radar). In this chapter, the focus is shifted to sample acquisition by introducing a specialized compressive sampling device. The target application of our new sampling device is spectrum sensing. Our goal is to acquire spectrum measurements at minimal hardware costs. To this end, a new concept for a non-uniform sampler is presented. No hardware is fabricated but various hardware aspects are analyzed and discussed.

Scanning a large bandwidth to identify occupied bands is required in a variety of applications. One of them is cognitive radio (CR) [107]. The idea behind CR is to allow secondary users to transmit in the same spectrum as licensed primary users, whenever unoccupied bands in the spectrum of the primary users are detected. Thus, CR can significantly increase spectrum utilization. Unfortunately, quickly identifying unused frequency bands by scanning large bandwidths is a challenging task requiring complex and power-hungry hardware.

An opportunity to simplify spectrum sensing opens up if the spectrum is only sparsely occupied. Indeed, measurements show that licensed spectrum is often heavily underutilized. Spectrum occupancies of around 10% for TV bands (470–854 MHz) or even less than 0.5% for the lower L bands (1–1.5 GHz) are reported in [108] when using a threshold of -100 dBm per 10 kHz channel. Having such low
CHAPTER 5. NON-UNIFORM SAMPLING

utilization and, thus, a very sparse spectrum, allows to utilize CS techniques for spectrum sensing. This leads to a reduced number of required measurements. However, specialized sampling devices must be employed to benefit from this promising reduction of samples.

Several acquisition schemes for sparse signals were reviewed in Sec. 1.7. Examples for acquisition systems suitable for a large class of sparse signals are the random demodulator [48] or the modulated wideband converter [51]. For signals sparse in the Fourier-basis, a much simpler sampling scheme becomes an interesting alternative. Since time and frequency bases are incoherent, a sufficient number of non-uniformly spaced samples in the time domain permits the reconstruction of Fourier-sparse signals. Traditional ADC architectures are not suitable for non-uniform sampling without introducing artificial delays and therefore wasting hardware resources. Recent examples of specialized non-uniform samplers are the edge-triggered ADC presented in [53] or the sample-and-hold architecture of [54]. Both are, however, based on traditional ADCs and need to insert certain idle periods.

In this chapter, we present a new non-uniform sampler, referred to as random sampling slope ADC (RSS-ADC). The goal of this new architecture is to obtain non-uniformly distributed samples at minimal hardware cost. To this end, the principles of sparse spectrum sensing are introduced in Sec. 5.1. The RSS-ADC architecture is then presented in Sec. 5.2. Restoration algorithms and their necessary adaptations are discussed in Sec. 5.3. An evaluation of the reconstruction quality is performed in Sec. 5.4. Hardware delays and the effect of off-grid frequency components are studied in Sec. 5.5 and Sec. 5.6, respectively.

5.1 Sparse Spectrum Sensing

In the spectrum sensing application considered here, the continuous time signal $y(t)$ is assumed to be band-limited to a frequency $f_B$ with a sufficiently sparse discrete frequency domain representation $\mathbf{x}$. We want to recover the Fourier-sparse signal with a reduced number of measurements using CS recovery algorithms.
5.2. RANDOM SLOPE ADC

5.1.1 Ideal Random Sampling

An ideal random sampler takes discrete measurements \( y \) randomly distributed in the time domain with a time-resolution \( T_0 \) < \( 1/f_N \) with \( f_N = 2f_B \) being the Nyquist frequency. We consider a signal of finite length \( NT_0 \) which allows to distinguish \( N \) discrete frequencies. In order to use CS recovery algorithms, a corresponding measurement matrix must be provided. For the conventional periodic sampling, the measurement matrix \( \Phi \) corresponds to a full DFT matrix with \( N = M \). By reducing the number of measurements, i.e., \( M < N \), the signal \( y(t) \) becomes undersampled with an effective sampling rate of \( f_{\text{eff}} = M/(NT_0) \).

\[
y_m = y(t)|_{t = k_mT_0}, \quad k_m \in \{1, N\}, \quad k_m > k_{m-1} \quad (5.1)
\]

\[
\Phi_{m,n} = \frac{1}{\sqrt{M}} \exp \left( -j2\pi \frac{(k_m - 1)(n - 1)}{N} \right) \quad (5.2)
\]

If the sampling instances \( k_m \) are selected randomly from a grid with resolution \( T_0 \), the measurement matrix created according to (5.2) meets the criteria for reliable signal recovery with high probability provided that a sufficiently large number \( M \) of time domain samples is collected [2].

5.2 Random Slope ADC

The proposed RSS-ADC performs a pseudo-random sampling of an input signal. It is based on the very simple slope ADC which is introduced in the following section.

5.2.1 Traditional Slope ADC

A slope ADC generates a linear reference slope and detects its intersection point with the input signal. The reference slope of a traditional slope ADC is periodically reset and restarted (Fig. 5.1(a)). Thus, the exact acquisition time is signal-dependent but the periodic reset of the
Figure 5.1: Reference slopes (solid) generated for given input signal (dashed).

reference slope allows no more than one sample per period. The major advantage of a slope ADC is its simplicity in hardware. Disadvantages are its slow speed, especially for high resolutions, and —in traditional systems— the unevenly spaced sampling instances. Either the slope ADC must be preceded by a zero-order hold circuit to obtain an even spacing of the samples or the irregularity must be compensated during post-processing. Nevertheless, this ADC architecture is getting particularly attractive for nanometer-scale CMOS converters due to a higher energy efficiency compared to flash ADCs. For example, the very recent design in [109] achieves 1 Msample/s with 9 bit resolution and 14 µW power dissipation using a 90 nm CMOS technology.

**Hardware** A major advantage of the slope ADC is its simple hardware architecture, of which an example is depicted in Fig. 5.2. In the following, we describe all components and their physical limitations.
The first stage is the analog-to-time conversion. This includes the ramp generator and the comparator. The most important limitation is the propagation delay $T_{pd}$ of the comparator, which determines the time needed for the comparator output to switch after the differential of the two input signals changed its sign. Also, the generated ramp is not perfectly linear if an analog ramp-generator is used with non-ideal current sources or reset transistors. Especially the discharging of the capacitor is not immediate but will cause an additional delay. As an alternative, a DAC can be used to generate a digital ramp. The sum of all delays from the input signal crossing until a new ramp can be started is denoted $T_d$. This includes the propagation delay of the comparator $T_{pd}$, after which the ramp reset starts, and the time needed to switch the comparator output back (Fig. 5.7).

The second stage is the time-to-digital conversion, which measures the time at which the comparator output switches. It can be characterized by the time resolution $T_0$ at which the comparator output is sampled. $T_0$ is, in one part, defined by the frequency that clocks the counter in the time-to-digital conversion. The resolution $T_0$ can then be further reduced using, e.g., delay lines. Given the resolution in time and the slope steepness $a$, the quantization of the amplitude is fixed.

**Sampling** When the reference slope and the input signal intersect, the next integer multiple of $T_0$ is recorded. In order to achieve a
sampling rate of $f_S \geq f_N$ with a resolution of $B$ bits, the clock frequency $f_0 = 1/T_0$ must be set to

$$f_0 = (2^B - 1)f_S \gg f_N \quad (5.3)$$

The requirements on the clock frequency and the bandwidth of the comparator thus increase exponentially with the resolution $B$, which requires the comparator to run significantly above the intended sampling frequency $f_S$. If no zero-order hold element is used, the effects of unevenly spaced samples need to be compensated by digital post-processing. For example by using iterative low-pass filtering [110], the sampling instances can be fit into a periodic grid again. However, this approach requires $f_S$ to be significantly above the Nyquist rate.

5.2.2 Random Sampling Slope (RSS-)ADC

The samples acquired by a slope ADC are already non-uniformly spaced. We now introduce the new RSS-ADC, which further increases the non-uniformity of the sampling times. The key feature of the RSS-ADC is the fast reset of the reference slope right after the reference slope has reached the input signal (Fig. 5.1(b)). The RSS-ADC contains a ramp generator, a comparator, and a time-to-digital converter. The only difference to the traditional slope ADC is the feedback-path to the reset of the ramp generator, which is drawn as a dashed line in Fig. 5.2.

This modification allows to obtain a more uneven spacing since the periodic reset is removed, i.e, more than one sample can be acquired within a period of $1/f_S$. This permits collecting a larger number of samples without complicating the circuit. Note that while a given hardware implementation now generates more samples, fewer samples are actually needed for signal reconstruction. This leads to a twofold advantage of the RSS-ADC compared to a traditional slope ADC:

1. Having non-uniformly distributed samples allows to apply CS techniques, which reduce the number of required samples. Thus, hardware requirements are relaxed.
2. The above introduced RSS-ADC principle acquires more samples than the traditional slope ADC with the same comparator. Since additional samples are not needed, the required hardware speed can be further reduced.
5.2. RANDOM SLOPE ADC

A similar random sampler was proposed concurrently and independently to the RSS-ADC [111]. This sampler is also based on a slope ADC and sparse recovery. In contrast to our implementation, wait periods of random length are introduced before a new slope is started and no fast reset of the slopes is done. This reduces the hardware efficiency compared to the RSS-ADC since all components stay idle during the wait periods.

**Sampling on a discrete grid** With ideal hardware \((T_{pd} = 0)\), the lowest possible sampling interval is given by the time-to-digital resolution \(T_0\). The sampling instances \(k_m\) are determined by the time \(c_m\) at which the cross-overs of the reference slopes with the input signal are detected. The series of sampling times in integer multiples of \(T_0\) is then given by

\[
k_m = \sum_{m'=1}^{m} c_{m'}.
\]  

(5.4)

New measurements are acquired until the end of the sampling window is reached at \(t = NT_0\). The measured values \(y_m\) are determined according to \(y_m = c_m a + a_0\) (cf. Fig. 5.3). The increment \(a\) within time \(T_0\) of a reference slope raising from \(-A\) to \(A\) with a resolution of \(B\) bits is given by \(a = 2A/(2^B - 2)\) and the initial offset is \(a_0 = -A - a/2\).
5.3 Reconstruction

The reconstruction of the frequency domain signal $x$ is most conveniently done by a greedy algorithm. A small adaptation must, however, be made. Since we consider a real-valued time domain signal, the corresponding spectrum $x \in \mathbb{C}^N$ is conjugate symmetric. For optimal signal reconstruction, this knowledge must be incorporated into the reconstruction algorithm. To this end, real and imaginary parts of $x$ are first separated. Since $x_1$ and $x_{N/2+1}$ are real-valued, we end up with $N/2 + 1$ real values and $N/2 - 1$ imaginary values. The total number of degrees of freedom is thus $N$. Since the two real-valued components $x_1$ and $x_{N/2+1}$ have no practical significance, we skip them for the sake of simplifying the notation. The new real-valued vector $\tilde{x} \in \mathbb{R}^{N-2}$ is constructed as

$$\tilde{x} = \begin{bmatrix} \text{Re}\{x_2\} & \text{Re}\{x_3\} & \ldots & \text{Re}\{x_{N/2}\} \\ \text{Im}\{x_2\} & \text{Im}\{x_3\} & \ldots & \text{Im}\{x_{N/2}\} \end{bmatrix}^T. \quad (5.5)$$

The corresponding measurement matrix, for which $y = \tilde{\Phi}\tilde{x}$ holds, enforces conjugate symmetry in the frequency domain and only allows real-valued measurements.

$$\Phi_{m,n} = \begin{cases} \frac{2}{\sqrt{M}} \cos \left( \frac{2\pi (k_m-1)n}{N} \right) & \text{if } 1 \leq n < \frac{N}{2} \\ \frac{2}{\sqrt{M}} \sin \left( \frac{2\pi (k_m-1)(3N-n)}{N} \right) & \text{if } \frac{N}{2} \leq n \leq N - 2 \end{cases} \quad (5.6)$$

In order to reconstruct $\tilde{x}$ from $y$, we apply the CoSaMP algorithm [35] introduced in Sec. 1.5.2. Due to the new formulation of the measurement matrix, the LS optimization will always produce conjugate symmetric estimates, which was not the case with the original complex-valued measurement matrix. One must account for the dependency of the two elements $\tilde{x}_m$, $\tilde{x}_{N/2-1+m}$, representing the real and imaginary parts of the same discrete frequency component. Thus, we modify the original algorithm to always select the two corresponding components together. To this end, we perform the component selection using the magnitude of the complex coefficient. As a result, the findLargestIndices() function of Alg. 2 selects the most important contributions by calculating the sum of the two squared components

$$\sqrt{\tilde{x}_{h}^2 + \tilde{x}_{h+N/2-1}^2} = \sqrt{\Re\{x_{h+1}\}^2 + \Im\{x_{h+1}\}^2}. \quad (5.7)$$
5.4. EVALUATION

CoSaMP selects $2K$ component-pairs in the first step (line 4 of Alg. 2) and then reduces the set to $K$ component-pairs in the second selection step (line 6).

5.4 Evaluation

In order to evaluate the new ADC architecture, various numerical simulations were performed. The support recovery rate is used as a quality metric. Fig. 5.4 shows the support recovery rate of different slope ADCs with varying ramp resolution ($B$) in bits and varying sparsity ($K/N$) while $f_0$ and the comparator bandwidth are fixed. Thus, all the systems simulated for this plot require the same hardware complexity. The counter clock frequency is set to only three times the Nyquist frequency ($f_0 = 3f_N$), the hardware is assumed ideal ($T_{pd} = 0$), and the number of distinguishable discrete frequencies is set to $N = 512$. The signal to be acquired is a discrete multi-tone signal where all possible frequencies lie on a grid and each occupied frequency corresponds to a single entry in $x$. The K-sparse support set is randomly chosen. The amplitudes are normally distributed with a mean of 10 and a standard deviation of 1. The phase is uniformly distributed.

By sweeping the ramp resolution (number of bits $B$), not only the quantization granularity is changed but also the effective sampling rate. The slope $n$ gets less steep as the number of bits $B$ increases and, thus, it takes longer to acquire a sample. This leads to fewer but more precise measurements.

The performance of the traditional slope ADC with the filtering method described in [110] is shown in Fig. 5.4(a). The lighter the color, the higher is the percentage of successful scans. One can clearly see that as soon as the number of acquired samples gets too small (i.e. falls below the Nyquist rate), detection of active frequencies is not possible anymore. Fig. 5.4(b) depicts the performance of the new RSS-ADC. Compared to the traditional slope ADC, detection is still possible with fewer samples.

For comparison, Fig. 5.4(c) shows an ideal random sampler that acquires the same number of samples as the RSS-ADC. One can thus
Figure 5.4: Support recovery rate of the RSS-ADC compared to the traditional slope ADC and random sampling for varying sparsity and ramp resolution.
5.4. EVALUATION

Figure 5.5: Effective sampling rate of the new RSS-ADC and the traditional slope ADC.

observe the small loss in recovery performance stemming from the not ideally random but signal-dependent sample distribution.

The simulations show that with a resolution of $B = 3$ bits, the best support recovery rate is achieved. While stronger undersampling of a signal is possible with higher ramp resolution, this is not the main goal of this architecture; the efficient utilization of the hardware resources is far more important. Since the hardware costs remain constant and the performance is getting worse, higher resolution with stronger undersampling is not desired in the RSS-ADC.

Fig. 5.5 shows the average number of samples acquired by the conventional and the new RSS-ADC with $f_0 = 3f_N$. At $B = 3$ bits, the proposed ADC shows an average sampling rate of more than 20% below the Nyquist frequency. Using the conventional slope ADC with $B = 3$, an effective sampling rate of only $f_0/(2^B - 1) = 0.43f_N$ is obtained.

**Comparison to Conventional Slope ADC**

A support recovery performance comparison of a traditional slope ADC to a CS-aided slope ADC and an RSS-ADC is shown in Fig. 5.6(a). In this comparison, the RSS-ADC is compared to the following two setups:

**Slope ADC** In the traditional converter, the iterative low-pass filtering method described in [110] is applied to the non-uniform
(a) Support recovery rate

(b) Effective sampling rate

Figure 5.6: Comparison of traditional to CS-aided slope ADC and RSS-ADC with fixed sparsity.
samples in order to recover the band-limited signal. After transformation to the discrete Fourier domain, the spectral components with the $K$ highest magnitudes are selected.

**CS slope ADC** As an intermediate step, the performance of a CS-aided slope ADC is plotted. It uses the same hardware as the traditional slope ADC with full slopes from $-A$ to $A$. However, a CS reconstruction algorithm instead of low-pass filters is used to estimate the spectrum.

The same constraint on $f_0$ is set for all ADCs, which makes the hardware requirements comparable. In the following simulations, a quantization accuracy of $B = 3$ bits and $N = 512$ is used. The comparison is done at two different sparsity levels of $K = 25$ (≈ 5% of $N$) and $K = 100$ (≈ 20% of $N$).

Figure 5.6 now allows to attribute the gains of the RSS-ADC to two different factors. The difference between the slope ADC and the CS slope ADC curves show what can be gained by purely applying CS recovery to a traditional slope ADC without changes in the analog frontend. The additional gains obtained with an RSS-ADC result from the increased number of samples and from the additional randomness. The increase of the effective sampling rate is illustrated in Fig. 5.6(b).

In conclusion, Fig. 5.6 shows that the RSS-ADC achieves a 100% detection rate with a much smaller $f_0$ and, thus, much simpler and cheaper hardware. More precisely, the speed requirements are reduced by a factor of 2.3 and 2.5 for $K = 25$ and $K = 100$, respectively.

### 5.5 Non-Ideal Hardware

Until now all the RSS-ADC components were considered ideal with no significant delays. In order to evaluate how well the RSS-ADC is working with less ideal hardware components, hardware propagation delays need to be considered. An illustration of the RSS-ADC reference signal with included hardware delays is depicted in Fig. 5.7. A first order low-pass filter is applied to the ideal ramp in order to model an exponential discharging during reset. The most interesting delay parameter for our purpose is the time $T_d$ since it determines the speed requirements of the comparator and thus the hardware
complexity. A higher \( T_d \) simplifies the hardware but reduces the number of measurements. \( T_d \) indicates the smallest possible spacing in between two samples but it does not limit the time resolution \( T_0 \), which might still be much higher and allows for a supported bandwidth higher than \( 1/T_d \). A practical example of a very fast 250 Msample/s slope ADC [112] shows a \( T_{pd}/T_0 \) ratio of 4.

In order to quantify the effect of this delay on the system performance, different values for \( T_d \) were simulated with 600 random sparse patterns (Fig. 5.8). The series of sampling points now include an additional slope reset delay of \( r = T_d/T_0 \) compared to the ideal case (5.4).

\[
k_m = m r + \sum_{m' = 1}^{m} c_{m'}
\]

We fix \( T_0 = 1/(3 f_N) \), the resolution to \( B = 3 \) bit and the number of non-zero elements to \( K = 16 \). In Fig. 5.8, the RSS-ADC is compared to a perfect random sampler and a random sampler which also requires a minimum spacing of \( T_d \) in between two samples but is not signal-dependent. Both random samplers use exactly the same number of samples as the RSS-ADC, which results in the same effective sampling rate for all three samplers. As the simulations show, delays of up to \( T_d = 7T_0 \) do not decrease the support recovery rate for a sparsity of \( K = 16 \) but significantly reduce speed requirements and reduce the effective sampling rate to around 40% of the Nyquist frequency. The comparison with a perfect random sampler shows what could be achieved with no constraints on the sampling instances. In order
5.5. NON-IDEAL HARDWARE

(a) Support recovery rate

(b) Effective sampling rate

Figure 5.8: Effect of delay constraint: Comparison of RSS-ADC to random sampler (RS) with and without any constraint on minimum sample spacing.
to identify which part of the performance difference is due to the delay constraint and which part is due to the signal-dependence of the RSS-ADC, the intermediate curve of a random sampler with delay constraint needs to be examined. It turns out that the delay constraint has a smaller influence than the signal dependence of the sampling points.

5.6 Off-Grid Frequencies

For the initial investigations so far, all active tones were placed on a grid, where each entry in the vector $\mathbf{x}$ corresponds to one frequency bin. In real-world applications however, the sparse frequency components are continuously distributed over arbitrary frequencies. When continuously distributed frequencies need to be reconstructed using a dictionary with $N$ discrete elements, the result is often not sparse anymore due to the heavy sidelobes introduced by the Dirichlet sampling kernel.

Possible remedies are described in [105]. One approach is to increase the resolution in the frequency domain by adding dictionary elements to $\Phi$ for intermediate frequencies. This corresponds to computing a DFT of a zero-padded signal and results in a redundant
5.7. **Summary**

In this chapter we have introduced the principles of a new non-uniform sampling device, the RSS-ADC. Using CS recovery algorithms, a sparsely occupied spectrum is efficiently estimated. The number of required samples and the sampling precision were significantly reduced compared to the traditional slope ADC. This allowed speed reductions of the analog frontend by a factor of $2^{3} \sim 2^{5}$

The RSS-ADC is, however, no ideal random sampler due to the signal-dependence of the sampling time. This leads to a certain loss, which was investigated by numerical simulations. Other non-ideal factors stem from delays in hardware components. It was shown that considerable delay can be tolerated, which further reduces the requirements on the analog components.
Chapter 6

Comparison of Recovery Algorithms

The goal of this chapter is to compare the two classes of greedy algorithms, for which VLSI implementations have been developed in this thesis: serial greedy pursuits and thresholding algorithms. Depending on the structure of the problem at hand, different algorithms become the most suitable choice. Here, we evaluate and compare our implementations in terms of performance and hardware complexity. The comparison is performed in two steps. First, we identify which algorithm performs best under which circumstances by numerical simulations. The corresponding results are presented in Sec. 6.1. This evaluation is performed using the practically relevant reconstruction problem of Fourier-sparse signals from non-uniformly acquired samples.

Second, we compare the hardware complexity of the corresponding implementations. On one hand, the implementations of Chapters 2 and 3 are compared in Sec. 6.2. On the other hand, new FPGA implementations of OMP and AMP were designed with similar architectures and the same speed target. This allows for an accurate comparison of hardware complexity. Both architectures are generic and not limited to certain bases. They are discussed in Sec. 6.3.
6.1 Numerical Performance Evaluation

In order to compare different greedy algorithms, we continue the evaluation of the rather generic reconstruction problems occurring in non-uniform samplers; a Fourier-sparse signal is assumed to be sampled at random points in the time domain. Similar problem structures were also encountered in sparse channel estimation and in audio restoration. Thus, having a form of sub-sampled DFT-matrix as a measurement matrix has many more practical applications than the Gaussian matrices considered in Sec. 1.5.3.

The required modifications of the random sampler recovery algorithms, which take the conjugate symmetry in the frequency domain into account, were already explained for CoSaMP in Sec. 5.3. The extension to MP, GP, and OMP is straightforward; again, the measurement matrix needs to be changed according to (5.6) and the findLargestIndices() function of Alg. 1 needs to be adapted such that the frequency component with the largest amplitude is selected. Note that the purpose of the measurement matrix adaptation is mainly to maintain conjugate symmetry during LS-optimization. Algorithms, such as MP, without an LS step, also work using the original complex-valued measurement matrix.

6.1.1 AMP for Fourier-Sparse Signals

AMP requires further adaptations to perform well with complex-valued signals. The general complex-valued case is studied in [113]. The soft thresholding function is implemented as proximity operator of the complex $\ell_1$-norm. For a threshold $\theta$ and a complex number $ae^{j\omega}$ in its polar representation, the following component-wise thresholding operator results:

$$\eta_\theta(ae^{j\omega}) = [a - \theta]^+e^{j\omega}$$

(6.1)

For the Fourier-sparse case with real-valued measurements, [113, Proposition 2.1] can be simplified to the following residual update using a complex-valued $\ell_0$-“norm”.

$$r^i = y - \Phi x^i + br^{i-1} \text{ with } b = \frac{1}{M}\|x^i\|_0$$

(6.2)
The performance of this adaptation is evaluated in the following section.

6.1.2 Simulations

Simulations of the random sampling reconstruction problem are performed with $N = 1024$ possible discrete frequencies. In each of the 1000 simulation runs, the occupied frequencies are randomly selected. Their amplitudes are normally distributed with a mean of 10 and a standard deviation of 1 and the phases are uniformly distributed. The support recovery rates achieved for OMP, GP, MP, and AMP at various sparsity levels with $M = 256$, $N = 1024$ are plotted in Fig. 6.1. AMP is shown in its unmodified real-valued and its adapted complex-valued version. The adaptation clearly improves the performance and makes AMP the best choice for this scenario.

In the next scenario, the number of measurements is swept. We examine two cases, one with a highly sparse ($K = 10$) and another with a moderately sparse ($K = 50$) signal. Fig. 6.2 shows the performance of the same greedy algorithms as above. For AMP, the regularization parameter $\lambda$ was optimized for each simulation point individually. While AMP clearly shows the best performance in the $K = 50$ case, OMP is the best choice in the very sparse situation. Additionally, OMP is more than $10 \times$ faster than AMP for $K = 10$. Thus, OMP is a very attractive choice in highly sparse situations and AMP solves less sparse problems better and faster. The alternatives MP and GP might not always achieve the highest performance but are often very good low-complexity alternatives to OMP. An overview of our findings is given in Tbl. 6.1.

6.2 Implemented Designs

We start our evaluation of hardware complexity by a rough comparison of the designs presented so far in this thesis. The channel estimation ASICs presented in Chapter 2 perform sparse signal recovery of signals with 12 to 18 significant entries and problems of dimension $200 \times 256$ in 0.5 ms. They are implemented in 180 nm CMOS technology. The AMP designs of Chapter 3 solve a signal restoration
Figure 6.1: Recovery of Fourier-sparse signals with various sparsity levels, $M = 256$, $N = 1024$.

Figure 6.2: Measurement number sweep for the recovery of Fourier-sparse signals in highly sparse ($K = 10$) and less sparse ($K = 50$) case with $N = 1024$. 
Table 6.1: Qualitative comparison of greedy reconstruction algorithms. Positive signs indicate lower complexity and higher performance.

<table>
<thead>
<tr>
<th></th>
<th>Complexity</th>
<th>AMP</th>
<th>OMP</th>
<th>MP</th>
<th>GP</th>
</tr>
</thead>
<tbody>
<tr>
<td>highly sparse</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>((K = 10))</td>
<td>Complexity</td>
<td>++</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
</tr>
<tr>
<td></td>
<td>Performance</td>
<td>+</td>
<td>++</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>moderately sparse</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>((K = 50))</td>
<td>Complexity</td>
<td>+</td>
<td>-</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Performance</td>
<td>++</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

problem of size 512 \(\times\) 1024 in 1.29 ms using a 65 nm technology. The restored audio signals tend to be less sparse compared to the channel estimation case.

In order to enable meaningful comparisons, the AMP designs were synthesized for the same 180 nm technology as the serial greedy pursuits. The resulting figures are listed in Tbl. 6.2. The AMP designs were then scaled to the problem size and speed requirements for channel estimation. We assume that the circuit area scales linearly with the number of operations per time and use the scaling rule (3.6). The number of AMP iterations is kept constant at \(I_{\text{max}} = 20\). The resulting area estimate is 15.2 mm\(^2\) and 1.8 mm\(^2\) for AMP-M and AMP-T, respectively. This makes both AMP implementations clearly larger than the implementations of MP and GP in Tbl. 2.2. AMP-T has a similar size like OMP, which, however, does not use any fast transforms.

For a comparison from the opposite viewpoint, the channel estimators were scaled up such that they solve the audio restoration problem. To this end, MP, GP and OMP are assumed to iterate 200, 110 and 100 times, respectively. The three implementations of serial greedy pursuits are scaled to the audio restoration problem size using the formulas in Tbl. 2.1. As a result, the estimated circuit area of OMP is more than 2\(\times\) larger than AMP-M, which is mainly caused by the complexity required by LS estimations for the high \(K\)-levels typically arising in audio signals. The scaled area of GP is slightly smaller than AMP-M and the scaled MP requires only half the area
## 6.3 Generic FPGA Implementations

The recovery algorithms MP and GP are often good low-complexity approximations and can be used whenever their recovery quality meets the specifications. If higher performance is needed, OMP or AMP should be chosen for hardware implementation. While the selection criteria for OMP and AMP were evaluated from a performance point of view in Sec. 6.1, the comparison of hardware complexity needs further investigation. In order to obtain data on the required hardware resources, two generic designs of OMP and AMP were developed for high-performance FPGAs. Setting the same targets on speed and precision allows for a direct hardware comparison. The designs support measurement matrices of arbitrary structure, i.e., neither fast transforms nor efficient matrix generators were used; instead, the full measurement matrix is stored on the numerous on-chip RAM blocks available on today’s large FPGAs. To achieve a fast execution speed,
a large number of parallel MAC units are employed, arranged as a vector multiplication unit (VMU). Whenever possible, the VMU is fully utilized. Whether an efficient implementation of an algorithm is possible in such a hardware configuration is mostly determined by how amenable the algorithm is to parallelization, how few additional, specialized units are required, and how easily the data-flow is controlled.

6.3.1 Highly Parallel OMP

The generic OMP implementation follows the design in Sec. 2.3.3 for the most part. Again, the LS optimization problem is solved by QRD, which is implemented using the modified Gram-Schmidt orthogonalization [79] listed in Alg. 6. This allows to iteratively update the decomposed matrices $Q$ and $R$.

In contrast to the previous implementation, we do, however, not apply the fast update procedure on vector $g$. This would require storing the correlation coefficients $\Phi^H \Phi$. When no structure is assumed on $\Phi$, this results in a very large memory which we want to avoid. Without fast updates on the correlation coefficients, another efficient update method operating on the residual $r$ becomes possible. Having performed a QRD, the residual update can be considerably simplified (see [62]) by replacing line 7 of Alg. 1 with the following:

$$r^i \leftarrow r^{i-1} - Q_i (Q_i)^T r^{i-1}$$ (6.3)

This fast operation replaces the matrix-vector multiplication on line 7 of Alg. 1 and avoids explicit calculation of $x$. The resulting algorithm is given in Alg. 7. This simplification is possible since the residual is orthogonal to all but the latest added column of $\Phi$. Back-substitution to calculate $x$ then becomes unnecessary. Only to calculate the final solution in the last iteration, back substitution is required to obtain $x^K$:

$$Rx^K = Q^T y$$ (6.4)

Architecture

The execution of OMP includes two computationally expensive steps, which are the matrix-vector multiplication $\Phi^T r$ on line 3 of Alg. 1 and
Figure 6.3: Architectures for highly parallel restoration on FPGA.
6.3. GENERIC FPGA IMPLEMENTATIONS

Algorithm 7 FPGA implementation of OMP

1: $r^0 \leftarrow y$; $x^0 \leftarrow 0_M$; $\Gamma^0 \leftarrow \emptyset$
2: for $n = 1, \ldots, K$ do
3:     $g^n \leftarrow \Phi^H r^{n-1}$;
4:     $i^n \leftarrow \text{argmax}_i |g^n_i|$;
5:     $\Gamma^n \leftarrow \Gamma^{n-1} \cup i^n$;
6:     $[Q^n R^n] \leftarrow \text{QRD}(\Phi_{\Gamma^n})$;
7:     $r^i \leftarrow r^{i-1} - Q^n_i (Q^n_i)^T r^{i-1}$;
8: end for
9: $x^K = (R^n)^{-1} (Q^n)^T y$
10: Output $x^K$

the LS optimization. To speed up these operations, a large number of parallel multipliers $P$ is instantiated and configured as a VMU, which is the central computation unit of the architecture, as shown in Fig. 6.3(a). Thus, the main challenge of this design is to supply the VMU with enough parallel data at high speed. To do so, the memory storing $\Phi$ is split into at least $P$ block RAMs that can be accessed in parallel. Each block RAM holds one row of $\Phi$. Other memories, such as the ones for the measured vector $y$ and the residual $r$ are implemented as register banks, where each element is accessible in parallel.

While the VMU is shared for most of the computations, a few additional hardware units perform specialized tasks. The MAX unit finds the maximum index from the correlations with columns of $\Phi$. In the BackS unit, the back substitution in the LS algorithm is computed using one multiplier and one adder. An operation which also needs specialized hardware is the square root reciprocal (SRR) computing $1/\sqrt{\cdot}$ in the QRD, which combines parts of lines 6 and 7 in Alg. 6 and makes explicit division unnecessary. The OMP VMU has multiple operation modes that allow to compute all multiplication operations in OMP:

- In the matrix-vector mode, the unit multiplies one column of a matrix with a vector in parallel and adds up all coefficients in a subsequent pipe-lined adder tree. This mode is used for the calculation of $\Phi^T r$ and the QRD.
• In the subtraction mode, a vector is subtracted from a vector-scalar product. This is used during QRD and the residual update.

• In the scalar mode, the VMU multiplies a vector with a scalar and outputs the results in parallel.

The square root reciprocal is implemented according to [114] using a look-up table to obtain a first estimation, followed by a modified Newton-Raphson step. In- and output signals are scaled to cover a higher dynamic range. Using only one Newton-Raphson iteration proved to be accurate enough for our application.

6.3.2 Highly Parallel AMP

The generic AMP design requires the same specialized hardware units as the AMP-M architecture discussed in Sec. 3.2.1, which are the RMSE, the L0, and the thresholding (TRSH) unit. In contrast to OMP, not only multiplications with $\Phi^T$ but also with $\Phi$ need to be performed. This requires a different VMU configuration since the measurement matrix is stored in the RAMs in such a way that one row can be accessed in parallel, but not one column. Therefore, the VMU needs to support two modes: a parallel mode that calculates one entry of the result vector at the same time using an adder tree, and a serial mode where one entry is calculated over multiple cycles in the same MAC. This dual mode operation was avoided in Sec. 3.2.2 by designing a measurement matrix generator that was able to produce the entries of multiple rows or columns at the parallel outputs. In a RAM-based implementation of unstructured matrices, this is hard to achieve. Thus, the combinational complexity of the multiplication units in this architecture is slightly higher than the one presented in Sec. 3.2.2. However, storing the full measurement matrix in RAMs adds the benefit of run-time reconfigurability.

Architecture

The AMP architecture uses a slightly modified VMU as central processing block. This allows to efficiently compute the matrix-vector multiplications on lines 4 and 6 of Alg. 3. A block diagram is provided
in Fig. 6.3(b). A number of small additional blocks are necessary to compute the remaining operations. The L0 block calculates $b$ by counting the number of non-zero elements. The RMSE unit is responsible for computing the threshold $\theta$, which is then fed into the TRSH block that performs soft thresholding.

Similar to the OMP implementation of the VMU, different operation modes need to be supported. Here, a parallel and a serial mode are implemented to support multiplications with $\Phi^T$ and $\Phi$.

- **Parallel mode:** In order to perform a multiplication with $\Phi^T$, each entry of the resulting vector is computed by a parallel multiplication of one row of $\Phi^T$ followed by an addition of all multiplication results in an adder tree. As this mode outputs the resulting vector serially, the subsequent thresholding operation can be computed in a simple serial unit.

- **Serial mode:** In order to perform a multiplication with $\Phi$, each entry of the resulting vector is computed serially on a single multiplier, configured as a MAC unit. In this mode, the resulting vector is presented in parallel at the output of the MAC array.

In order to calculate the RMSE, a dedicated MAC is instantiated to serially square and accumulate the entries of the residual. The square root is then approximated by the method proposed in [85].

### 6.3.3 Hardware Evaluation

#### Fixed-Point Operation

To allow for fast and energy-efficient computations, fixed-point operations are used throughout the two implementations. The word-widths (annotated in Fig. 6.3) were optimized to simultaneously obtain near-floating-point performance and to match well with the signal processing structures available on the FPGA. The input and output word-width of both algorithms is set to 18 bit. The most important parameters are the word-widths of the VMU, which are set to 18 bit and 25 bit on the inputs and 42 bit on the accumulator. The measurement matrix $\Phi$ is stored with an accuracy of 25 bit. The resulting implementation loss with respect to a floating-point Matlab implementation is 0.6 dB and 0.4 dB for OMP and AMP, respectively.
FPGA Results

Both designs were mapped to a Xilinx Virtex-6 LX240T FPGA, where recovery problems of size $256 \times 1024$ are solved on $P = 256$ parallel multipliers. The implementation results with corresponding resource usages are shown in Tbl. 6.3. Note that AMP requires less than half the number of slices compared to OMP, which is mainly due to the additional register banks for $y$ and $r$. OMP also requires additional memories for the $Q$ and $R$ matrices. Since the memory storing $\Phi$ dominates block RAM usage, RAMs for $Q$ and $R$ only lead to a small increase in RAM utilization. Further, AMP achieves a higher maximum clock frequency compared to OMP, mainly because the VMU block is used in more configurations than in AMP, which results in a more complex wiring with large parallel buses.

Testing

In order to test the circuitry, a universal asynchronous receiver/ transmitter (UART) interface to a PC has been implemented, which allows to download measurement vectors to the FPGA and to retrieve the reconstructed sparse estimates. An evaluation of the hardware was performed using four times sub-sampled images in blocks of size $32 \times 32$ pixels. The images are assumed to be sparse in the wavelet basis. Thus, $\Psi$ implements a 2-dimensional wavelet transform and $\Omega$ is a Gaussian matrix. For the processed example image (‘Lena’ with size $256 \times 256$ in 64 blocks), OMP achieves a reconstructed signal-to-noise ratio of 23.5 dB, while AMP achieves 21.4 dB.

Comparisons

Compared to a reconstruction by Matlab on a PC with a 2.6 GHz dual-core central processing unit (CPU), the FPGA implementation is $\sim 4000$ times faster for OMP and $\sim 5000$ times faster for AMP. The only comparable OMP implementation is given in [62]. When scaling our OMP implementation down to the smaller problem size of $32 \times 128$ used in [62] and mapping it to a Virtex-5 FPGA, our implementation shows a $1.5 \times$ faster speed than [62].

The OMP and AMP implementations use the same number of parallel multipliers in the VMU unit. AMP converges well after 40
Table 6.3: FPGA implementation results with number of occupied hardware units and device utilization in parentheses on a Xilinx XC6VLX240T (speed grade -2).

<table>
<thead>
<tr>
<th></th>
<th>AMP</th>
<th>OMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency [MHz]</td>
<td>165</td>
<td>100</td>
</tr>
<tr>
<td>Proc. time [µs]</td>
<td>$15.8 \cdot I_{max}$</td>
<td>$11 \cdot K + 0.6 + \sum_{l=1}^{K} 0.34 \cdot l$</td>
</tr>
<tr>
<td>Slices</td>
<td>12113 (32%)</td>
<td>32010 (84%)</td>
</tr>
<tr>
<td>Block RAMs</td>
<td>256 (61%)</td>
<td>258 (62%)</td>
</tr>
<tr>
<td>DSP slices</td>
<td>258 (33%)</td>
<td>261 (33%)</td>
</tr>
</tbody>
</table>

iterations for an image recovery problem of size $256 \times 1024$. In the same time AMP performs 40 iterations, OMP can complete 36 iterations. Thus, in the given configuration, OMP is faster for problems with less than 36 non-zero coefficients. For all less sparse problems, AMP is faster. In general, AMP better fits the hardware structure than OMP; less memory and less complex control flows are required, resulting in higher speed and lower resource usage.

This OMP implementation uses a different fast update procedure than the one presented in Sec. 2.3.3. Here, the residual is updated using the orthogonality provided by the QRD; in Sec. 2.3.3, the sparse estimation is updated using correlation coefficients. The update of correlation coefficients is computationally more efficient, when the problem is very sparse. The fast residual update is more efficient for less sparse signals and does not require the storage of correlation coefficients. The cross-over point where the latter method becomes more efficient than the former is approximated by $K(K+N) \approx NM$ in [78]. Thus, the method chosen for our channel estimators in Chapter 2 is optimal for these very sparse problems and the method used in the generic design presented here works best in a moderately sparse scenario.
6.4 Summary

This chapter compares the different classes of greedy algorithms implemented in this thesis and discusses their most suitable application cases. More precisely, it is shown that serial greedy pursuits are well suited for very sparse problems. Under very sparse conditions, OMP has the highest recovery rate. AMP is better suited for less sparse recovery problems. A graphical representation of these results is given in Fig. 6.4.

We present two fast and highly parallel FPGA implementations of the OMP and AMP algorithms to further compare their complexity. Both implementations solve generic CS problems of size $256 \times 1024$ with reconfigurable measurement matrices. While AMP has lower hardware requirements and is more suitable for less sparse problems, OMP performs faster for recovery problems with, in our case, less than 36 non-zero coefficients.
Chapter 7

Conclusions

In this thesis VLSI architectures and applications for sparse signal processing were explored. Dedicated hardware was shown to enable real-time processing in a number of practical applications. The corresponding VLSI designs are all based on greedy algorithms, which deliver good reconstruction performance at low hardware complexity. All the implemented algorithms were selected based on an evaluation of their suitability for VLSI implementations.

7.1 VLSI Architectures for Sparse Signal Recovery

Serial Greedy Pursuits

Three architectures for the serial greedy pursuits MP, GP, and OMP were designed for the example application of sparse channel estimation. The proposed architectures use parallel complex-valued multipliers to attain the required throughput on a small area. Implementing fast parallel generation of the measurement matrix coefficients from LUTs with strongly reduced size made our designs much more economical than ROM-based architectures. Also, exploiting similar structures in the correlation update matrix allows to efficiently implement a coefficient generator with high throughput.
MP is able to exploit the sparsity assumption of the channel using the smallest silicon area. The GP implementation requires about three times the area of MP, but also improves the estimation by a few dB. GP can be implemented very efficiently using our newly introduced formulation of the algorithm. The OMP algorithm turned out to be overly complex for this real-time system. Even with a much larger area than GP, the possible number of iterations was not sufficient to capture all relevant taps in the high-SNR regime. Thus, GP is the most attractive choice for high-SNR sparse channel estimation.

The presented ASIC implementations are the most energy-efficient greedy reconstruction algorithms known so far. Also, the silicon area of these sparsity-based LTE channel estimator circuits was shown to be low compared to a typical full LTE baseband receiver.

A second MP architecture was designed for an acoustic localization application based on periodically transmitted PN sequences. The corresponding highly structured measurement matrix allowed for a low-complexity FPGA architecture. By calculating correlations in the discrete frequency domain, the number of arithmetic operations was strongly reduced.

### Approximate Message Passing

Our analysis of AMP shows that this recovery algorithm is very well suited for hardware implementation. Two fundamentally different architectures were proposed for AMP; the transform-based AMP-T and AMP-M using parallel multipliers. The implementation of both architectures resulted in the first VLSI designs of AMP, which makes them also the first ASIC designs of a BPDN solver.

The evaluation of AMP implementations on ASICs and FPGAs leads to the conclusion that the transform-based AMP-T design is much more efficient in terms of area and power consumption. However, AMP-M must be chosen when no such fast transform exist (e.g., for unstructured or random measurement matrices) or when online reconfigurability of the measurement matrix is required.

While an unstructured dictionary obtained by training might recover a signal better compared to a generic basis, this hardware evaluation shows that recovery with unstructured trained dictionaries comes at a significant overhead in hardware cost compared to a generic basis
with a fast transform. Therefore, implementing fast transforms is a very attractive solutions in many cases, even when arbitrary matrices promise slightly better reconstruction quality. Fast transforms are especially attractive since a large choice is available, such as FFT, DCT, wavelet transform, Hadamard transform, and all combinations thereof.

### Comparison

Implementations of OMP and AMP were compared more closely using parameterizable architectures for FPGAs. In general, OMP was shown to be faster for very sparse signals. Since the processing time of OMP increases quadratically with the number of non-zero elements \( K \) and AMP’s complexity is not strongly dependent on \( K \), AMP becomes quickly more efficient as \( K \) becomes larger. For our specific implementation and problem size, the corresponding cross-over point was determined. In general, AMP better fits the criteria for efficient VLSI implementation than OMP due to lower memory requirements and a less complex control flow.

The two low-complexity algorithms MP and GP provide often very good approximations of the results obtained by OMP with significantly reduced hardware requirements. It is, therefore, always advisable to check whether successful reconstruction is possible with these approximations, as it was demonstrated in our channel estimation example. In general, problems with low under-sampling and a measurement matrix with almost orthogonal columns are more amenable to low-complexity approximations.

### 7.2 Applications

All our VLSI architectures were developed for specific applications to, on one hand, have realistic constraints and, on the other hand, show the practical relevance of the proposed architectures and implementations. Possible gains of sparsity-aware processing were shown for the following applications:
Chapter 7. Conclusions

Channel estimation  Greedy recovery algorithms were shown to improve the channel estimation quality in LTE receivers provided that the number of reflections is small. By including only taps above noise level, the estimation is efficiently de-noised. In the low SNR regime, gains of up to 7 dB were obtained for a channel model with 10 MHz bandwidth and long delay spreads. The additional circuitry for sparse channel estimation only marginally increases the size of an LTE receiver. Thus, sparse channel estimation is an efficient way of improving the quality of the channel estimate and eventually increasing the receiver performance.

Audio restoration  Separating signal components according to their sparsity property in different bases allows to restore corrupted signals. This principle was successfully applied to the recovery of audio signals corrupted by clicks and pops. Our ASIC designs have shown that high-rate 192 ksample/s stereo audio signals can be restored in real time. The presented FPGA realization led to the first real-time demonstration of sparsity-based signal restoration. The designs can be extended to support other types of corruptions when a suitable sparsifying basis can be found.

Passive WiFi-based radar  Using WiFi signals for passive radar applications makes the detection of closely spaced target difficult. It was shown that, compared to the correlation image, sparse recovery achieves a much better picture of a sparse scene illuminated by a BS transmitting OFDM frames. An even better resolution was achieved by the inclusion of multiple BSs transmitting at random times in random channels.

Non-uniform sampling  The RSS-ADC is introduced as a new sampling device that acquires unevenly spaced samples. A careful evaluation of its parameters and hardware constraints revealed that this device is suitable for the recovery of Fourier-sparse signals from sub-Nyquist sampling. A penalty for not sampling at completely random times must, however, be accepted. Due to its simple hardware and low power consumption, this ADC is suitable for, e.g., sensor nodes with strong power constraints. However, one has to keep in
mind that additional power is needed for reconstruction. Therefore, the most suitable scenario is that samples are acquired by the RSS-ADC on a low-power sensor device and the spectrum is reconstructed on a device without power limitations.

7.3 Concluding Remarks

A list of contributions from this thesis is provided in Sec. 1.8. The evaluation of our various implementations has shown that each application requires a careful selection of the most suitable algorithm and architecture. There is no algorithm that performs best in every case. The first selection criterion should be the attainable recovery rate, which is evaluated by Monte-Carlo simulations. Once an algorithm is selected, it must be tuned for efficient implementation. Usually different update procedures or fast transforms are available and must be evaluated. We have shown that selecting the most suitable algorithm and applying optimizations has the potential to significantly lower hardware costs. Also, identifying operations with reduced precision requirements, like the AMP threshold, can make implementations faster and smaller.

Many efficient implementations of recovery algorithms exist for CPUs or graphics processing units (GPUs), which allow more complex control flows than VLSI implementations. Such software-based implementations are better suitable when no stringent time or power constraints apply.

Applications such as de-noised channel estimation or signal restoration are not strictly CS but are all based on sparse recovery algorithms, which found many applications beside CS. Pure CS schemes are suitable when measurements are expensive and (signal) noise is not critical. Counterexamples are certain applications in ultra-wideband communication, which we investigated, but had to conclude that noise folding eliminates most of the benefits obtainable through CS.

However, there are many more applications that might benefit from VLSI architectures for sparse signal recovery. One example is the reconstruction of images or live video. To this end, more efficient ways of including additional model constraints in the recovery process need to be explored.
Appendix A

Notation and Acronyms

Symbols

\( \mathbf{X}, \mathbf{x} \)  Boldface upper- and lower-case letters represent matrices and column vectors, respectively
\( M \)  number of measurements
\( N \)  dimension of sparse signal
\( K \)  number of non-zero coefficients in \( \mathbf{x} \)
\( \mathbf{x} \)  sparse representation of signal (vector of dimension \( N \))
\( \mathbf{y} \)  measurements (vector of dimension \( M \))
\( \rho \)  sparsity (=\( K/N \))
\( \Psi \)  sparsifying transformation matrix
\( \Omega \)  projection matrix
\( \Phi \)  measurement matrix \( \Phi = \Omega \Psi \)
\( \mathbf{A} \)  sparsifying basis for desired signal parts
\( \mathbf{B} \)  sparsifying basis for corruptions
\( \tilde{\Phi} \)  modified (real-valued) measurement matrix
\( \tilde{x} \)  modified (real-valued) sparse representation
\( \mathbf{I}_n \)  \( n \)-dimensional identity matrix
\( \mathbf{0}_n \)  all-zero vector of length \( n \)
\( \emptyset \)  empty set
Symbols

$\delta_k$ restricted isometry constant
$\lambda$ regularization parameter
$\theta$ threshold
$c_0$ speed of light
$P$ number of parallel processing units
$L$ number of propagation paths in wireless channel
$D$ number of subcarriers in an OFDM system
$R$ range profile length for localization
$V$ Doppler profile length for localization
$S$ number of recorded training frames
$S$ set of trained subcarriers
$\mathcal{F}$ discrete Fourier matrix
$T$ symbol duration

$f_N$ Nyquist frequency
$f_{\text{eff}}$ effective sampling rate
$B$ ADC resolution in bits
$T_0$ smallest resolvable time in time-to-digital converter
$T_d$ time delay in analog-to-time converter
$T_{pd}$ propagation delay of comparator
### Operators

- $(\cdot)^H$: Hermitian transposition
- $(\cdot)^T$: matrix transposition
- $(\cdot)^*$: complex conjugate
- $(\cdot)^\dagger$: Moore-Penrose pseudo-inverse
- $(\cdot)^{-1}$: inverse function, inverse of square matrix
- $\Re\{\cdot\}$: real part of complex-valued argument
- $\Im\{\cdot\}$: imaginary part of complex-valued argument
- $|\cdot|$: absolute value of a scalar / cardinality of a set
- $[\cdot]^+$: nearest non-negative value $[\cdot]^+ = \max\{\cdot, 0\}$
- $(\cdot)^{(i)}$: symbol in $i$-th iteration
- $s_i$: $i$-th entry of vector $s$
- $h_i$: $i$-th column of matrix $H$
- $H_i$: $i$-th row of matrix $H$
- $H_{i,j}$: entry of the $i$-th row and $j$-th column of matrix $H$
- $H_{\Gamma}$: columns of matrix $H$ contained in set $\Gamma$
- $\|\cdot\|_0$: $\ell_0$-“norm”, defined as the number of non-zero elements
- $\|\cdot\|_1$: $\ell_1$-norm, i.e., $\sum_{i=1}^n |\Re\{x_i\}| + |\Im\{x_i\}|$ for $x \in \mathbb{C}^n$
- $\|\cdot\|_2$: $\ell_2$-norm or Euclidean norm, i.e., $\sqrt{\sum_{i=1}^n |x_i|^2}$ for $x \in \mathbb{C}^n$
- $\log(\cdot)$: logarithm
- $(\cdot) \mod m$: modulo-m operator
- $\text{sgn}(\cdot)$: signum operator
- $\mathcal{N}(\cdot)$: null space
- $\sigma_K(\cdot)_p$: best $K$-sparse approximation in $\ell_p$
- $\mu(\cdot)$: coherence between rows of a matrix
- * convolution
## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3GPP</td>
<td>3rd generation partnership project</td>
</tr>
<tr>
<td>ADC</td>
<td>analog-to-digital converter</td>
</tr>
<tr>
<td>AMP</td>
<td>approximate message passing</td>
</tr>
<tr>
<td>AMP-M</td>
<td>multiplication-based AMP architecture</td>
</tr>
<tr>
<td>AMP-T</td>
<td>transform-based AMP architecture</td>
</tr>
<tr>
<td>ASIC</td>
<td>application-specific integrated circuit</td>
</tr>
<tr>
<td>BP</td>
<td>basis pursuit</td>
</tr>
<tr>
<td>BPDN</td>
<td>basis pursuit de-noising</td>
</tr>
<tr>
<td>BS</td>
<td>base station</td>
</tr>
<tr>
<td>CIR</td>
<td>channel impulse response</td>
</tr>
<tr>
<td>CMAC</td>
<td>complex-valued multiply-accumulator</td>
</tr>
<tr>
<td>CMOS</td>
<td>complementary metal-oxide semiconductor</td>
</tr>
<tr>
<td>CORDIC</td>
<td>coordinate rotation digital computer</td>
</tr>
<tr>
<td>CoSaMP</td>
<td>compressive sampling matching pursuit</td>
</tr>
<tr>
<td>CP</td>
<td>cyclic prefix</td>
</tr>
<tr>
<td>CPU</td>
<td>central processing unit</td>
</tr>
<tr>
<td>CR</td>
<td>cognitive radio</td>
</tr>
<tr>
<td>CS</td>
<td>compressive sensing</td>
</tr>
<tr>
<td>CSI</td>
<td>channel state information</td>
</tr>
<tr>
<td>DAC</td>
<td>digital-to-analog converter</td>
</tr>
<tr>
<td>DCT</td>
<td>discrete cosine transform</td>
</tr>
<tr>
<td>DFT</td>
<td>discrete Fourier transform</td>
</tr>
<tr>
<td>DSSS</td>
<td>direct sequence spread spectrum</td>
</tr>
<tr>
<td>ECG</td>
<td>electrocardiography</td>
</tr>
<tr>
<td>ET</td>
<td>early termination</td>
</tr>
<tr>
<td>FCT</td>
<td>fast discrete cosine transform</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>FFT</td>
<td>fast Fourier transform</td>
</tr>
<tr>
<td>FIFO</td>
<td>first-in first-out</td>
</tr>
<tr>
<td>FPGA</td>
<td>field-programmable gate array</td>
</tr>
<tr>
<td>FSM</td>
<td>finite state machine</td>
</tr>
<tr>
<td>GE</td>
<td>gate equivalent</td>
</tr>
<tr>
<td>GP</td>
<td>gradient pursuit</td>
</tr>
<tr>
<td>GPU</td>
<td>graphics processing unit</td>
</tr>
<tr>
<td>IDCT</td>
<td>inverse discrete cosine transform</td>
</tr>
<tr>
<td>IFCT</td>
<td>inverse fast discrete cosine transform</td>
</tr>
<tr>
<td>IFFT</td>
<td>inverse fast Fourier transform</td>
</tr>
<tr>
<td>IHT</td>
<td>iterative hard thresholding</td>
</tr>
<tr>
<td>i.i.d.</td>
<td>independent and identically distributed</td>
</tr>
<tr>
<td>ISM</td>
<td>industrial, scientific and medical</td>
</tr>
<tr>
<td>IST</td>
<td>iterative soft thresholding</td>
</tr>
<tr>
<td>LS</td>
<td>least squares</td>
</tr>
<tr>
<td>LTE</td>
<td>long term evolution</td>
</tr>
<tr>
<td>LUT</td>
<td>look-up table</td>
</tr>
<tr>
<td>MAC</td>
<td>multiply-accumulator</td>
</tr>
<tr>
<td>MIMO</td>
<td>multiple-input multiple-output</td>
</tr>
<tr>
<td>MP</td>
<td>matching pursuit</td>
</tr>
<tr>
<td>MRI</td>
<td>magnetic resonance imaging</td>
</tr>
<tr>
<td>MSE</td>
<td>mean squared error</td>
</tr>
<tr>
<td>MSU</td>
<td>multiply and storage unit</td>
</tr>
<tr>
<td>NSP</td>
<td>null space property</td>
</tr>
<tr>
<td>OFDM</td>
<td>orthogonal frequency-division multiplexing</td>
</tr>
<tr>
<td>OMP</td>
<td>orthogonal matching pursuit</td>
</tr>
<tr>
<td>PCB</td>
<td>printed circuit board</td>
</tr>
</tbody>
</table>
PN  pseudo noise
QRD  QR decomposition
RAM  random access memory
RIP  restricted isometry property
RMSE root mean squared error
ROM  read-only memory
RS   random sampler
RSNR recovery signal-to-noise-ratio
RSS-ADC random sampling slope analog-to-digital converter
SFD  start of frame delimiter
SNR  signal-to-noise ratio
SP   subspace pursuit
S-RAM static random access memory
SYNC synchronization
TRSH thresholding
VHDL very high speed integrated circuit hardware description language
VLSI very large scale integration
VMU  vector multiplication unit
Bibliography


[72] E-UTRAN: Physical channels and modulation, 3GPP Std. TS 36.211, Mar. 2009.


Curriculum Vitae

Patrick Mächler was born in Switzerland in 1984. He received his MSc degree in information technology and electrical engineering from ETH Zurich, Switzerland in 2008. In 2008, he was a visiting researcher at Berkeley Wireless Research Center (BWRC), UC Berkeley, CA, USA. In the same year, he joined the Integrated Systems Laboratory of ETH Zurich as a research assistant. His research interests include digital signal processing, VLSI architectures, compressive sensing, and wireless communication.