VLSI Design of Approximate Message Passing for Signal Restoration and Compressive Sensing

Patrick Maechler, *Student Member, IEEE,* Christoph Studer, *Member, IEEE,* David Bellasi, *Student Member, IEEE,* Arian Maleki, *Member, IEEE,* Andreas Burg, *Member, IEEE,* Norbert Felber, Hubert Kaeslin, *Senior Member, IEEE,* and Richard G. Baraniuk, *Fellow, IEEE*

***Abstract*—Sparse signal recovery finds use in a variety of practical applications, such as signal and image restoration and the recovery of signals acquired by compressive sensing. In this paper, we present two generic VLSI architectures that implement the approximate message passing (AMP) algorithm for sparse signal recovery. The first architecture, referred to as AMP-M, employs parallel multiply-accumulate units and is suitable for recovery problems based on unstructured (e.g., random) matrices. The second architecture, referred to as AMP-T, takes advantage of fast linear transforms, which arise in many real-world applications. To demonstrate the effectiveness of both architectures, we present corresponding VLSI and FPGA implementation results for an audio restoration application. We show that AMP-T is superior to AMP-M with respect to silicon area, throughput, and power consumption, whereas AMP-M offers more flexibility.**

***Index Terms*—Approximate message passing (AMP), compressive sensing, fast discrete cosine transform, field-programmable gate array (FPGA), ℓ_1-norm minimization, signal restoration, sparse signal recovery, very-large scale integration (VLSI).**

I. Introduction

SPARSE signal recovery has established itself as a powerful tool in various fields by enabling the recovery of a sparse vector from an underdetermined system of linear equations using sophisticated (non-linear) recovery algorithms [1]. Since many natural or man-made signals exhibit a sparse representation in certain bases (e.g., speech signals are approximately sparse in the Fourier domain) and many real-world problems can be formulated in terms of a system of linear equations, sparse signal recovery finds use in a large number of practical applications. Prominent examples are the restoration of audio signals or images from saturation, impulse noise, or narrowband interference [2]–[4], signal separation [5], de-noising [6], de-blurring [7], super-resolution [6], and in-painting [8], as well as compressive sensing (CS) [9], [10]. CS has recently gained significant attention in the research community by enabling the sampling of sparse signals using fewer measurements than the Nyquist rate suggests. In particular, CS has the potential of lowering the costs of sampling (compared to conventional analog-to-digital converters) and is used in a large number of practical applications, such as magnetic resonance imaging (MRI) [11], electroencephalography [12], imaging devices [13], radar [14], or wireless communication [15], [16].

Unfortunately, high-performance sparse signal recovery algorithms typically require a significant computational effort for the problem sizes occurring in most practical applications. While the computational complexity is not a major issue for applications where off-line processing on CPUs or graphics processing units (GPUs) can be afforded (e.g., in MRI), it becomes extremely challenging for applications requiring real-time processing at high throughput or for implementations on battery-powered (e.g., mobile) devices. Hence, to meet the stringent throughput, latency, and power-consumption constraints of real-time audio, image, and video restoration, CS-based imaging devices, radar, or wireless systems, developing *dedicated* hardware implementations, such as application specific integrated circuits (ASICs) or field-programmable gate arrays (FPGAs), is of paramount importance.

While significant research effort has been devoted to the design of high-performance and low-complexity sparse signal recovery algorithms, e.g., [17]–[25], much less is known about their economical implementation in dedicated hardware. Notable exceptions are the ASIC designs reported in [15], where the authors compared several implementations of greedy pursuit (GP) algorithms for sparse channel estimation in wireless communication systems. A similar recovery algorithm specifically designed for signals acquired by the modulated wideband converter was implemented on an FPGA in [16]. Another FPGA implementation for generic CS problems of dimension 32×128 was developed in [26]. All these implementations rely on GP algorithms, which are well-suited for the recovery of very sparse signals in hardware. However, for applications featuring less sparse (or approximately sparse) signals, such as audio signals, images, or videos, these al-

Manuscript received February 29, 2012; revised July 3, 2012; accepted August 2, 2012.

P. Maechler, D. Bellasi, N. Felber and H. Kaeslin are with the Dept. of Electrical Engineering and Information Technology, ETH Zürich, 8092 Zürich, Switzerland (e-mail: {maechler, felber, kaeslin}@iis.ee.ethz.ch). Website: http://www.iis.ee.ethz.ch

A. Burg is with the Inst. of Electrical Engineering, EPFL, 1015 Lausanne, Switzerland (e-mail: andreas.burg@epfl.ch). Website: http://tcl.epfl.ch

C. Studer, A. Maleki, and R. G. Baraniuk are with the Dept. of Electrical and Computer Engineering, Rice University, Houston, 77004 TX, USA (e-mail: {studer, arian.maleki, richb}@rice.edu). Website: http://dsp.rice.edu

The authors would like to acknowledge the support of M. Liechti, B. Muheim, and F. K. Gürkaynak during the ASIC design. We furthermore thank C. Aubel, H. Bölcskei, A. Bracher, P. Kuppinger, and G. Pope for inspiring discussions on signal restoration and separation. We also thank the anonymous reviewers for their valuable comments, which helped us to improve the exposition of our results.

The work of C. Studer was supported by the Swiss National Science Foundation (SNSF) under Grant PA00P2-134155. The work of A. Maleki and R. G. Baraniuk was supported in part by the Grants NSF CCF-0431150, CCF-0728867, CCF-0926127, DARPA/ONR N66001-08-1-2065, N66001-11-1-4090, N66001-11-C-4092, ONR N00014-08-1-1112, N00014-10-1-0989, AFOSR FA9550-09-1-0432, ARO MURIs W911NF-07-1-0185 and W911NF-09-1-0383, and by the Texas Instruments Leadership University Program.

gorithms quickly become inefficient in terms of throughput, silicon area, and power consumption, as their complexity scales roughly linearly in the signal's sparsity level (see [15] for a corresponding discussion).

A. Contributions

In this paper, we present—to the best of our knowledge—the first VLSI designs of a basis pursuit denoising (BPDN) solver for signal restoration and signal recovery from CS measurements. We compare possible candidate algorithms and identify the approximate message passing (AMP) algorithm [25] to be well suited for the recovery of approximately sparse signals, such as audio signals, images, or videos, in hardware. To demonstrate the suitability of AMP for VLSI implementations, we develop two generic architectures:

- The first architecture, referred to as AMP-M, is a general-purpose solution employing multiply-accumulate (MAC) units, which is suitable for sparse signal recovery problems relying on arbitrary (e.g., unstructured or random) linear measurements.
- The second architecture, referred to as AMP-T, is specifically designed for recovery problems for which the measurement matrices (i.e., the aggregation of the linear measurement operators) have fast transform algorithms.

In order to demonstrate the efficacy of both sparse-signal recovery architectures, we present corresponding VLSI designs for a real-time audio restoration example. For the AMP-T architecture, we employ a fast implementation of the discrete cosine transform (DCT), which substantially improves upon the MAC-based AMP-M solution in terms of silicon area, throughput, and power consumption. For both architectures, we present reference VLSI and FPGA implementation results to highlight the effectiveness of AMP-T and AMP-M for sparse signal recovery and CS in practical systems.

B. Outline of the Paper

The remainder of the paper is organized as follows. Section II briefly introduces CS and evaluates prominent sparse signal recovery algorithms, including AMP. Section III discusses the application of AMP for signal restoration. The VLSI architectures for AMP-M and AMP-T are detailed in Section IV; corresponding reference VLSI and FPGA implementation results and a comparison to existing solutions are given in Section V. We conclude in Section VI.

C. Notation

Lowercase and uppercase boldface letters stand for column vectors and matrices, respectively. The ith entry of a vector $\mathbf{a}$ is a_i; the kth column and the ℓth entry on the kth column of a matrix $\mathbf{A}$ are denoted by $\mathbf{a}_k$ and $[\mathbf{A}]_{k,\ell}$, respectively. $\mathbf{I}_M$ and $\mathbf{0}_{M\times N}$ stand for the $M \times M$ identity and the $M \times N$ all-zeros matrix, respectively; the transpose of a matrix $\mathbf{A}$ is designated by $\mathbf{A}^T$. The Euclidean (or ℓ_2) norm of a vector $\mathbf{x}$ is denoted by $\|\mathbf{x}\|_2$. The ℓ_1-norm of $\mathbf{x}$ is designated by $\|\mathbf{x}\|_1$, and $\|\mathbf{x}\|_0$, often referred to as the ℓ_0-norm, corresponds to the number of non-zero entries in $\mathbf{x}$. The complex conjugate, real part, and imaginary part of $a \in \mathbb{C}$ is denoted by a^*, $\Re\{a\}$ and $\Im\{a\}$, respectively. For $x \in \mathbb{R}$, we define $[x]_+ = \max\{x, 0\}$.

II. Sparse Signal Recovery and Compressive Sensing (CS)

We next review the basics of sparse signal recovery and CS and evaluate potential candidate recovery algorithms, with a focus on their suitability for VLSI. We then summarize AMP [25], which is considered in the remainder of the paper.

A. Compressive Sensing in a Nutshell

Compressive sensing (CS), as put forward in [9], [10], aims to sample a signal vector $\mathbf{y} \in \mathbb{R}^N$ using fewer measurements than the Nyquist rate suggests. Specifically, CS considers the acquisition of $\mathbf{y}$ through M linear (and non-adaptive) measurements as follows:

$$\mathbf{z} = \mathbf{\Phi}\mathbf{y} + \mathbf{n} \tag{1}$$

Here, $\mathbf{\Phi} \in \mathbb{R}^{M\times N}$ is a sensing matrix satisfying $M < N$, and $\mathbf{n} \in \mathbb{R}^M$ represents additive measurement noise. Since the recovery of $\mathbf{y}$ from the noiseless measurements $\mathbf{z} = \mathbf{\Phi}\mathbf{y}$ corresponds to solving an under-determined set of linear equations, the estimation of $\mathbf{y}$ from the noisy measurements (1) is, in general, an ill-posed problem. Nevertheless, many natural or man-made signals have a sparse representation $\mathbf{x}$ in a given orthonormal basis $\mathbf{\Psi}$, i.e., $\mathbf{y} = \mathbf{\Psi}\mathbf{x}$, where only a few entries $K \ll N$ of $\mathbf{x}$ are non-zero. Sparsity enables us to estimate the signal $\mathbf{y}$ if the effective matrix $\mathbf{D} = \mathbf{\Phi}\mathbf{\Psi}$ satisfies the so-called restricted isometry property (RIP) [10]. For instance, if the entries of $\mathbf{\Phi}$ are i.i.d. zero-mean Gaussian, then $\mathbf{D}$ is known to satisfy the RIP with overwhelming probability provided that [27]

$$M \sim K \log(N/K). \tag{2}$$

In this case, a fundamental result of CS states that a stable estimate for $\mathbf{x}$ can be obtained with the aid of sparse signal recovery algorithms. More specifically, recovery from (1) is commonly achieved by a convex optimization-based method known as basis pursuit de-noising [28]

$$\text{(BPDN)} \quad \underset{\tilde{\mathbf{x}}}{\text{minimize}}\ \|\tilde{\mathbf{x}}\|_1 \quad \text{subject to } \|\mathbf{z} - \mathbf{D}\tilde{\mathbf{x}}\|_2 \le \varepsilon,$$

with $\|\mathbf{n}\|_2 \le \varepsilon$. Finally, an estimate for the desired signal vector $\mathbf{y}$ can be obtained by computing $\mathbf{\Psi}\hat{\mathbf{x}}$, where $\hat{\mathbf{x}}$ denotes the solution to BPDN.

B. Evaluation of Sparse Signal Recovery Algorithms

In order to compute the solution to BPDN, a variety of optimal and sub-optimal sparse signal recovery algorithms have been proposed in the literature [17]–[19], [22]–[25]. We next evaluate several potential candidate algorithms with respect to their suitability for VLSI implementation.

1) Interior-point methods: Convex optimization problems, such as BPDN, can be solved accurately using interior point methods [17]. Such methods are known to exhibit high computational complexity for moderate-to-large problem sizes and typically require high numerical precision; both drawbacks render an efficient implementation in VLSI challenging.

2) First-order methods: To alleviate the complexity and precision requirements of interior-point methods, a variety of first-order methods (i.e., algorithms involving matrix-vector multiplications with $\mathbf{D}$ and $\mathbf{D}^T$ only) for solving the Lagrangian BPDN problem

$$(\text{BPDN}^*) \quad \underset{\tilde{\mathbf{x}}}{\text{minimize}}\ \|\mathbf{z}-\mathbf{D}\tilde{\mathbf{x}}\|_2^2 + 2\lambda\|\tilde{\mathbf{x}}\|_1$$

have been proposed in the literature [18], [23]. Here, the regularization parameter $\lambda > 0$ controls a trade-off between fidelity to the measurements and ℓ_1-norm of the solution $\hat{\mathbf{x}}$.

Iterative soft-thresholding (IST) [18], for example, is a simple first-order method that computes

$$\mathbf{x}^{i+1} = \eta_{\theta^i}\big(\mathbf{x}^i + \mathbf{D}^T\mathbf{r}^{i-1}\big) \quad \text{with} \quad \mathbf{r}^{i-1} = \mathbf{z} - \mathbf{D}\mathbf{x}^i$$

for each iteration $i = 1, \ldots, I_{\max}$. Here, $I_{\max}$ denotes the maximum number of iterations, $\mathbf{x}^i$ the current estimate for $\mathbf{x}$, $\mathbf{r}^{i-1}$ represents the residual error, θ^i is an iteration-dependent threshold, and $\eta_{\theta^i}(\cdot)$ implements an entry-wise soft-thresholding policy as follows:

$$\eta_\theta(x) = \text{sign}(x)\,[|x|-\theta]_+ \,. \tag{3}$$

The IST algorithm is able to deliver the result to BPDN* given that $\mathbf{D}$ and the thresholds θ^i satisfy certain properties [29]. Unfortunately, IST exhibits slow convergence, which eventually leads to high computational complexity. To this end, first-order algorithms achieving faster convergence than IST have been developed in the literature, e.g., [20], [21], [23]. The associated computational complexity is, however, still too high for most real-time applications in dedicated hardware.

3) Greedy pursuit (GP): Rather than solving BPDN or BPDN* altogether, a variety of GP-based algorithms that deliver approximations to these convex problems have been proposed in the literature, including:

- Matching pursuit (MP) [19] iteratively identifies the column $\mathbf{d}_i$ of $\mathbf{D}$ that is most correlated to a current signal estimate, followed by a simple update that computes an improved signal estimate. While each iteration of MP requires very low computational effort, the number of iterations heavily depends on the sparsity level K and hence, MP is only suitable for extremely sparse signals.
- Orthogonal matching pursuit (OMP) [22] and compressive sampling matching pursuit (CoSaMP) [24] are more sophisticated GP-based algorithms that incorporate a least squares (LS) step to compute a signal estimate. The LS step significantly reduces the number of required iterations compared to MP, but it induces a high computational complexity per iteration and requires considerable numerical precision (see, e.g., [15]).

While GPs are well-suited for recovering very sparse signals in VLSI, as in certain applications in wireless communication or radar [15], [16], [26], for example, they turn out to be rather inefficient for large-dimensional problems and/or (approximately sparse) signals exhibiting moderate-to-high sparsity levels, such as audio signals, images, or videos.

Algorithm 1 Approximate Message Passing (AMP) [25]

```
1: initialize $\mathbf{r}^0 = \mathbf{z}$ and $\mathbf{x}^0 = \mathbf{0}_{N\times 1}$
2: for $i = 1, \ldots, I_{\max}$ do
3:     $\theta \leftarrow \lambda \frac{1}{\sqrt{M}}\|\mathbf{r}^{i-1}\|_2$
4:     $\mathbf{x}^i \leftarrow \eta_\theta\big(\mathbf{x}^{i-1} + \mathbf{D}^T\mathbf{r}^{i-1}\big)$
5:     $b \leftarrow \frac{1}{M}\|\mathbf{x}^i\|_0$
6:     $\mathbf{r}^i \leftarrow \mathbf{z} - \mathbf{D}\mathbf{x}^i + b\mathbf{r}^{i-1}$
7: end for
8: return $\mathbf{x}^{I_{\max}}$
```

C. Approximate Message Passing (AMP)

AMP [25] is a recently developed sparse signal recovery algorithm that delivers excellent recovery performance, exhibits fast convergence at low computational complexity per iteration, while requiring low arithmetic precision. All these properties render AMP advantageous for the implementation in VLSI compared to the algorithms evaluated above.

1) Algorithm: The pseudo-code for AMP is given in Algorithm 1. One can immediately see that AMP has a similar structure as IST (cf. Section II-B2), with the key difference that the residual $\mathbf{r}^i$ is computed not only based upon the current estimate $\mathbf{x}^i$ but also using the residual obtained in the previous iteration $\mathbf{r}^{i-1}$ (cf. line 6). In order to identify a suitable thresholding policy, we decided to set θ proportionally to the regularization parameter λ and the root mean square error (RMSE) of the residual $\mathit{RMSE} = \frac{1}{\sqrt{M}}\|\mathbf{r}^{i-1}\|_2$ (see line 3 of Algorithm 1) as proposed in [30]. All these differences to IST render AMP vastly superior in terms of convergence rate, without noticeably penalizing the complexity per iteration. Moreover, AMP provably delivers the solution to BPDN* for matrices $\mathbf{D}$ having zero-mean i.i.d. Gaussian distributed entries[1] of order $1/\sqrt{M}$ [25]. For arbitrary (e.g., deterministic) matrices, AMP does not necessarily converge to the BPDN* solution. We emphasize, however, that AMP has been shown (empirically) to deliver excellent recovery performance for a large class of deterministic and highly structured matrices, such as sub-sampled Fourier or DCT matrices [31].

2) Computational complexity: The main computational burden of AMP corresponds to the matrix-vector multiplications required in each iteration (cf. lines 4 and 6 in Algorithm 1). If the multiplication with the matrix $\mathbf{D}$ and $\mathbf{D}^T$ can be replaced by a fast transform, e.g., using a fast Fourier transform (FFT), then each iteration can be performed with very low computational complexity (see Section IV-C for a corresponding example). The maximum number of iterations $I_{\max}$ required by the algorithm to converge usually ranges between 10-to-100 (depending on the target accuracy, the signal sparsity, and the dimensionality of $\mathbf{D}$). The square-root computation in the RMSE (cf. line 3) requires a specialized hardware unit, but can be implemented at low cost (see Section IV-A1 for the details). All remaining computations can be carried out efficiently using standard circuitry, which renders AMP well-suitable for the implementation in VLSI.

[1]Proper normalization of the columns in $\mathbf{D}$ is *crucial* for AMP to converge to the solution to BPDN*; see [25] for the details.

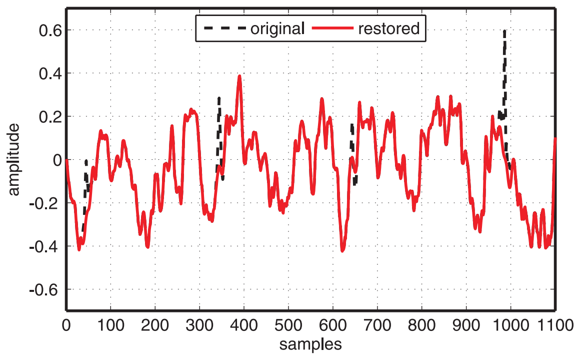
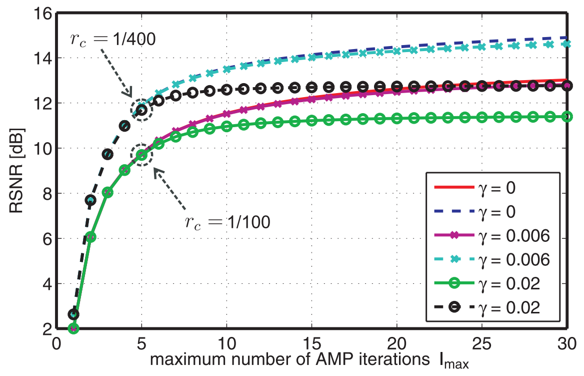
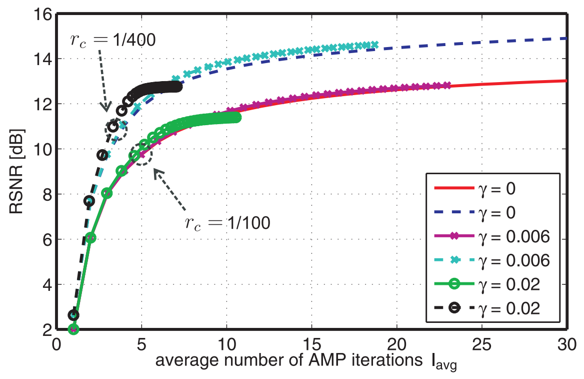


Fig. 1. Audio recovery example for the old phonograph recording "Mussorgsky" from [32]; a snapshot of the original (corrupted) signal and the signal recovered by AMP using the DCT–identity pair are shown.



(a) Convergence behavior for different click rates and ET thresholds.



(b) Impact of ET to the RSNR.

Fig. 2. Convergence behavior and impact of early termination (ET) to the RSNR performance of AMP for sparsity-based audio restoration.

3) Early termination: The complexity of AMP can be reduced further by means of *early termination* (ET). In particular, the iterations of AMP can be terminated as soon as the RMSE is small enough (depending on the target application). To this end, we define an ET threshold $\gamma \geq 0$ and stop the iterative procedure (cf. lines 2–7) as soon as $RMSE \leq \gamma$. Determining when ET occurs comes at virtually no additional hardware costs, since computation of the RMSE is anyway required for the soft-thresholding parameter θ (cf. line 3). Furthermore, the ET threshold γ in combination with the maximum number of iterations I_{max} allows us to trade performance for complexity; this trade-off is analyzed in Section III-B for an audio restoration application.

III. Signal Restoration

In addition to CS, sparse signal recovery has been employed in the restoration of signals corrupted by impulse noise and/or saturation [2]–[4]. We next show how signal restoration can be formulated as a sparse signal recovery problem and demonstrate the suitability of AMP for this particular application.

A. Signal Restoration as a Sparse Signal Recovery Problem

As shown in [2]–[4], signals corrupted by impulse noise and saturation can be modeled as

$$\mathbf{z} = \mathbf{A}\mathbf{a} + \mathbf{B}\mathbf{b} + \mathbf{n} = \mathbf{D}\mathbf{x} + \mathbf{n}, \tag{4}$$

with $\mathbf{D} = [\,\mathbf{A}\ \mathbf{B}\,]$, $\mathbf{x} = [\,\mathbf{a}^T\ \mathbf{b}^T\,]^T$, and the corrupted observation $\mathbf{z} \in \mathbb{R}^M$. Here, the matrix $\mathbf{A} \in \mathbb{R}^{M \times N_a}$ sparsifies the signal $\mathbf{s} = \mathbf{A}\mathbf{a}$ to be restored and the matrix $\mathbf{B} \in \mathbb{R}^{M \times N_b}$ sparsifies the corruptions on the signal $\mathbf{s}$. Signal restoration now amounts to the recovery of the sparse vector $\mathbf{x}$ from (4) using, BPDN or BPDN*, for example, followed by computing $\mathbf{s} = \mathbf{A}\mathbf{a}$. In certain cases, one can identify the locations of the corrupted entries in $\mathbf{z}$ prior to recovery, which typically results in improved restoration performance [2]–[4].

For the restoration to succeed, the matrix $\mathbf{A}$ must not only sparsify the (uncorrupted) signal $\mathbf{s}$, but also be incoherent to $\mathbf{B}$; i.e., the mutual coherence [3]

$$\mu_m = \max_{k,\ell} \frac{|\mathbf{a}_k^H \mathbf{b}_\ell|}{\|\mathbf{a}_k\|_2 \|\mathbf{b}_\ell\|_2}$$

should be small (see [3], [4] for a theoretical analysis). This important observation allows one to select a matrix pair $\mathbf{A}, \mathbf{B}$ that is suitable for the given signal-restoration application.

For the restoration of audio signals from clicks/pops and saturation, as considered in the remainder of the paper, setting $\mathbf{A}$ to the $M \times M$ (unitary) DCT matrix defined as

$$[\mathbf{A}]_{k,\ell} = \sqrt{\frac{c_k}{M}} \cos\left(\frac{(2\ell-1)(k-1)\pi}{2M}\right)$$

with $c_k = 1$ for $k = 1$ and $c_k = 2$ otherwise, enables one to sparsify audio signals; setting $\mathbf{B} = \mathbf{I}_M$ sparsifies clicks/pops and saturation artifacts. Since $\mathbf{A}$ is incoherent to $\mathbf{B}$, excellent performance for audio restoration can be achieved by using this pair of matrices. In particular, as shown in [3], [4], if the number of corrupted entries of $\mathbf{z}$ is small compared to its dimension M, then the DCT–identity pair is guaranteed to enable stable recovery of $\mathbf{x} = [\,\mathbf{a}^T\ \mathbf{b}^T\,]^T$ and, hence, of the desired (uncorrupted) signal $\mathbf{s} = \mathbf{A}\mathbf{a}$.

B. Numerical Results for AMP

We next evaluate the performance of AMP for audio restoration. Figure 1 shows a snapshot of the old phonograph recording "Mussorgsky" from [32] and the restored signal via AMP. Restoration is performed in blocks of length $M = 512$ using the DCT–identity pair; 16 samples between each pair of adjacent blocks are added and overlapped using raised-cosine windows to avoid artifacts at the block boundaries. Each block is recovered using AMP with $I_{\text{max}} = 20$ iterations, $\lambda = 2$, and no ET (i.e., $\gamma = 0$). Figure 1 illustrates the fact that AMP using the DCT–identity pair efficiently removes clicks/pops from old phonograph recordings, without knowing the locations of the sparse corruptions (also see [4] for similar results).

The performance and complexity of AMP for audio restoration is studied in Figure 2. We artificially corrupt a 16 bit 44.1 kHz speech signal from [33]; the clicks/pops are modeled by adding Gaussian pulses consisting of five samples whose peak value is uniformly distributed in $[-1, 1]$. We define the click rate r_c as the number of clicks per sample. The restoration performance is measured using the recovery signal-to-noise-ratio (RSNR), defined as $RSNR = \|\mathbf{s}\|_2^2/\|\mathbf{s} - \hat{\mathbf{s}}\|_2^2$, where $\mathbf{s}$ corresponds to the original (uncorrupted) signal and $\hat{\mathbf{s}}$ to the signal restored by AMP. Figure 2(a) shows the RSNR for different numbers of maximum iterations I_{max}; the regularization parameter λ 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 considered click rates.

The impact of ET on the performance and complexity is studied in Figure 2(b). We see that, for $\gamma = 0.02$, the number of average iterations I_{avg} is reduced while slightly degrading the RSNR. Lowering the ET threshold leads to a smaller reduction of average iterations I_{avg} but results in higher RSNR. Thus, carefully selecting γ enables one to reduce the complexity of AMP at *no loss* in terms of RSNR. For example, Figure 2(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, e.g., by silencing the entire circuit during idle clock cycles.

IV. VLSI Architectures of the AMP Algorithm

In this section, we present two novel VLSI architectures for the AMP algorithm. The first architecture, referred to as AMP-M, is a generic multiply-accumulate (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 $\mathbf{D}$ and $\mathbf{D}^T$ can be carried out by a fast transform. While the computational complexity of AMP-M is dominated by matrix-vector multiplications scaling with NM, a fast transform computes the same operation with lower asymptotical complexity. We start by describing the architectural principles for AMP-M and AMP-T suitable for arbitrary sparse signal recovery applications and then derive corresponding optimized architectures for audio restoration.

A. AMP-M: MAC-Based AMP Architecture

The first architecture implements the matrix-vector multiplications on lines 4 and 6 of Algorithm 1 using a pre-defined number of parallel MAC units. This MAC-based architecture has the advantage of being suitable for *arbitrary* matrices $\mathbf{D}$, including unstructured (e.g., random) matrices or matrices obtained through dictionary learning [34], as used, e.g., in many signal restoration or de-noising problems. Moreover, if $\mathbf{D}$ is explicitly stored in a memory, the matrices used in AMP-M can be reconfigured at run-time, without the need to re-design and re-implement the circuit.

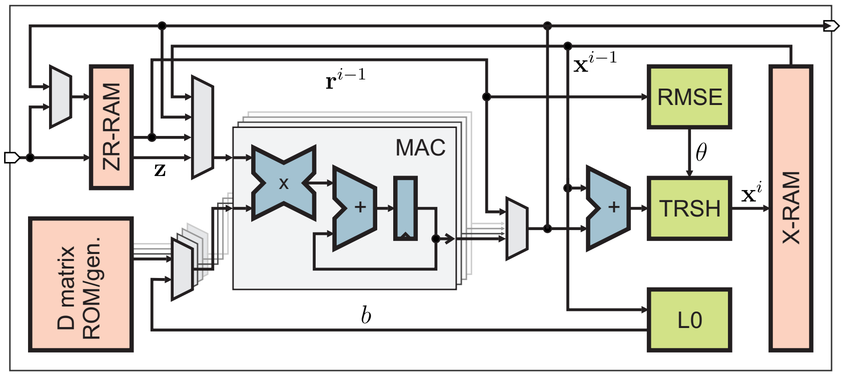
1) Architecture: Figure 3(a) shows the high-level block diagram of the AMP-M architecture. The AMP algorithm requires memories to store the input signal $\mathbf{z}$, the residual $\mathbf{r}^i$, and the signal estimate $\mathbf{x}^i$ obtained after applying the thresholding function. The input vector $\mathbf{z}$ and the residual $\mathbf{r}^i$ can be stored in the same memory (referred to as ZR-RAM in Figure 3(a)), i.e., each coefficient of $\mathbf{z}$ and $\mathbf{r}^i$ can be stored at the *same* address, which allows for memory instances having suitable address-to-word-length ratios leading to small S-RAM macro cells. The signal estimate $\mathbf{x}^i$ is stored in a separate memory, referred to as X-RAM. Depending on the application, the entries of the matrix $\mathbf{D}$ are either stored in a RAM or ROM, or can 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 Section V-B). 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 $\mathbf{D}$ with $\mathbf{x}^i$ or $\mathbf{D}^T$ with $\mathbf{r}^{i-1}$. Hence, each MAC unit requires access to a different entry of $\mathbf{D}$ in each clock cycle, while the same vector entry is shared among all units (see Figure 3(a)).

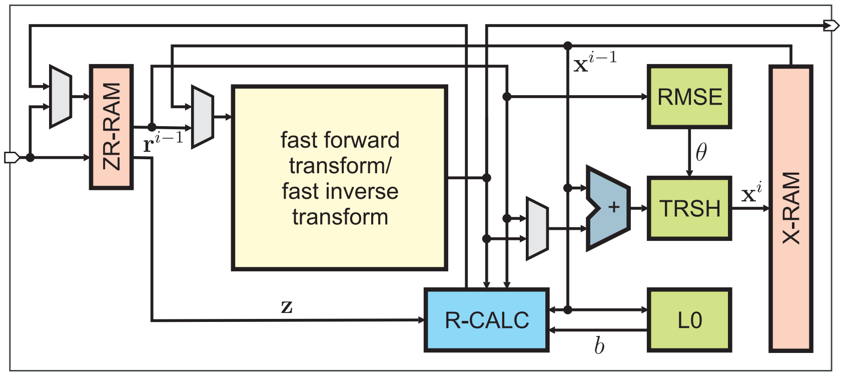
The RMSE is computed using a separate unit that is specialized for computing sums of squares. The subsequent square root computation is implemented using the approximation developed in [35], which requires neither multipliers nor look-up-tables (LUTs). The RMSE is computed in parallel to the matrix-vector multiplication $\mathbf{D}^T\mathbf{r}^{i-1}$ (line 4 of Algorithm 1). Note that the rather limited numerical accuracy of the deployed square-root approximation was found to be sufficient for our purposes (see the discussion in Section V-A).

To implement the thresholding function (3), we instantiated a subtract-compare-select unit that applies thresholding in a serial and element-wise manner (performed in the TRSH unit). The ℓ_0-norm on line 5 of Algorithm 1 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 $\mathbf{r}^i$ (line 6 of Algorithm 1), are performed using the available MAC units in a time-shared fashion.

2) Optimization for audio restoration: The main bottleneck of the AMP-M architecture is the memory bandwidth required to deliver the entries of $\mathbf{D}$ to the parallel MAC units. For unstructured (e.g., random) matrices, (multi-port) LUTs, ROMs, or on-chip S-RAMs can be used for small dimensions (in the order of a few hundred kbit). For large-dimensional problems, external memories (e.g., off-chip D-RAMs) become necessary, which shifts the memory bottleneck to the bandwidth of the external memory interface. Fortunately, in many real-world applications, the matrix $\mathbf{D}$ is highly structured; hence, it is often possible to generate its coefficients on the fly at very high throughput. Specifically, for the DCT–identity pair



(a) High-level block diagram of the MAC-based AMP-M architecture.



(b) High-level block diagram of the transform-based AMP-T architecture.

Fig. 3. VLSI architectures of the approximate message passing (AMP) algorithm for signal restoration and compressive sensing (CS).

often used in audio restoration, we can avoid the explicit storage of all entries of the DCT matrix (which would result in prohibitively large on-chip memory). Instead, we exploit the regular structure of the DCT matrix and make use of symmetries to generate the $M\times 2M$ matrix $\mathbf{D}$ at high throughput by using a small cosine LUT having only M entries. The LUT address is calculated on the basis of the row and column 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 $\mathbf{D}$, only M values needed to be stored; this results in a $1024\times$ 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.

B. AMP-T: Transformation-Based AMP Architecture

While the AMP-M architecture is well-suited for unstructured matrices, small-scale problems, or applications for which the matrix $\mathbf{D}$ 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 $\mathbf{D}$) render its application difficult for throughput intensive and/or large-scale problems. Fortunately, in many practical applications the matrix $\mathbf{D}$ has 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.

1) Architecture: Figure 3(b) 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:

- No storage for the matrix $\mathbf{D}$ or logic to generate its entries on the fly is required.
- The parallel MAC units have been replaced by a specialized fast transform unit, which must support both the fast forward transform and its inverse.
- 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 ℓ_0-norm is computed simultaneously to the fast inverse transform.

The architecture for carrying out the fast forward and its inverse heavily depends on the used transform and algorithm. 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 Section V, AMP-T substantially improves upon AMP-M in terms of throughput, silicon area, and power consumption.

2) Optimization for audio restoration: For the audio restoration application considered in this paper, we use an architecture implementing a fast DCT (FCT) and its inverse (IFCT). The corresponding algorithm and the resulting VLSI architec-

TABLE I
HIGH-LEVEL COMPARISON OF FCT/IFCT ALGORITHMS

Algorithm	Regularity	Memory	Complexity
Matrix-vector multiplication	++	−−	−−
Direct approach [37]	−−	−	++
Recursive method [38]	−	++	++
Via $2M$-point FFT [39]	+	−	−
Via M-point FFT [40]	+	+	+
Via $M/2$-point FFT [40]	+	++	++

ture are detailed in Section IV-C. The additions 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 thresholding.

C. VLSI Implementation of the FCT/IFCT

Existing VLSI implementations of a fast DCT/IDCT have mainly been designed for MPEG-2 video compression, which relies on problems of size 8×8 (see, e.g., [36]). For the targeted 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 next evaluate potential algorithms for efficiently computing a large-dimensional FCT/IFCT and then, we detail the architecture used in the final AMP-T implementation.

1) Algorithm evaluation: A variety of algorithms to compute the FCT/IFCT have been proposed in the literature [37]–[40]. A high-level comparison of some of the most prominent algorithms is provided in Table I. We consider the algorithm's regularity, memory requirements, and computational complexity. 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.

The algorithm proposed in [37] directly performs divide-and-conquer on the DCT matrix to achieve very low computational complexity. This algorithm, however, exhibits an irregular data flow and corresponding architectures cannot easily be parametrized to support different problem sizes. Another direct approach is the recursive method proposed in [38], which is more efficient than [37] (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 fast Fourier transform (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 [39]. An improved algorithm relying on an M-dimensional FFT only, was proposed in [40]. This approach exhibits lower complexity 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 [40]. This approach reduces the computational complexity and memory requirements compared to the M-FFT approach, while maintaining high regularity. Hence, we

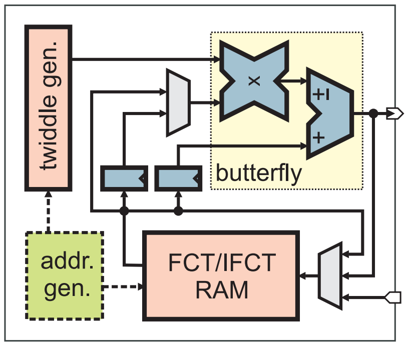


Fig. 4. High-level block diagram of FCT/IFCT unit. The highlighted block corresponds to the time-shared radix-2 FFT butterfly.

decided to implement this $M/2$-FFT-based algorithm in the remainder of the paper.

2) $M/2$-FFT-based FCT/IFCT algorithm: The $M/2$-FFT approach described in [40] is summarized in Table II and performs the FCT and IFCT in multiple steps. For the FCT, the entries of the real-valued input vector $\mathbf{x}$ are first reordered and stored to a vector $\mathbf{c}'$. Then, the reordered vector is converted into a complex-valued vector $\mathbf{c}$ of half the length, i.e., $M/2$. The main task of the FCT algorithm is to compute a $M/2$-length FFT of the vector $\mathbf{c}$. The result $\mathbf{f}$ is expanded into a conjugate-symmetric vector $\mathbf{f}'$, which corresponds to a M-point FFT of the real-valued vector $\mathbf{c}'$. To obtain the result $\mathbf{a}$ of the FCT, the entries of $\mathbf{f}'$ are rotated by certain twiddle factors (as used in the FFT), defined as

$$t_k^M = \exp\left(2\pi j \frac{k-1}{M}\right), \tag{5}$$

followed by extracting the real value of the rotated entries of $\mathbf{f}'$. The procedure for the IFCT is analogous to that of the FCT; see Table II for the details.

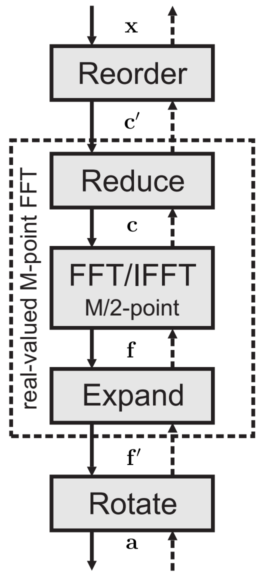
3) Architecture: We now detail the FCT/IFCT architecture used in AMP-T for audio restoration. To process a stereo 192 kHz audio signal with a block size of $M = 512$ samples and 16 samples overlap, restoration of one block must be completed within 1.29 ms. Since $I_{\text{max}} = 20$, an FCT/IFCT operation must be computed in no more than 32.3 µs.

Figure 4 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 [41] and also controls the operations carried out during the other phases of the algorithm.

We perform a complex-valued $M/2$ in-place FFT/IFFT 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

TABLE II
NECESSARY COMPUTATIONS FOR THE FCT (TOP) AND IFCT (BOTTOM) ALGORITHM AS IN [40]

$\mathbf{a} = \text{FCT}(\mathbf{x})$		$\mathbf{x}, \mathbf{c}', \mathbf{a} \in \mathbb{R}^M,\ \mathbf{f}' \in \mathbb{C}^M,\ \mathbf{f}, \mathbf{c} \in \mathbb{C}^{M/2}$
Reorder	$c'_k = \begin{cases} x_{2k-1} \\ x_{2(M-k)+2} \end{cases}$	$k = 1, \dots, M/2$ $k = M/2+1, \dots, M$
Reduce	$c_k = c'_{2k-1} + jc'_{2k}$	$k = 1, \dots, M/2$
FFT	$\mathbf{f} = \text{FFT}(\mathbf{c})$	
Expand	$f'_k = \begin{cases} f_k + f^*_{M/2-k+2} - j(t_k^M)^{-1}(f_k - f^*_{M/2-k+2}) \\ f^*_{k-M/2} \end{cases}$	$k = 1, \dots, M/2$ $k = M/2+1, \dots, M$
Rotate	$a_k = \begin{cases} \frac{1}{\sqrt{M}} \Re\{f'_1\} \\ \sqrt{\frac{2}{M}} \Re\{(t_k^{4M})^{-1} f'_k\} \end{cases}$	$k = 1$ $k = 2, \dots, M$
$\mathbf{x} = \text{IFCT}(\mathbf{a})$		$\mathbf{x}, \mathbf{c}', \mathbf{a} \in \mathbb{R}^M,\ \mathbf{f}' \in \mathbb{C}^M,\ \mathbf{f}, \mathbf{c} \in \mathbb{C}^{M/2}$
Inv. rotate	$f'_k = \begin{cases} \sqrt{M} a_1 \\ \sqrt{\frac{M}{2}} t_k^{4M} (a_k - j a_{M-k+2}) \end{cases}$	$k = 1$ $k = 2, \dots, M$
Inv. expand	$f_k = \frac{1}{2}\left(f_k + f^*_{M/2-k+2} + jt_k^M(f_k - f^*_{M/2-k+2})\right)$	$k = 1, \dots, M/2$
IFFT	$\mathbf{c} = \text{IFFT}(\mathbf{f})$	
Inv. reduce	$c'_k = \begin{cases} \Re\{c_k\} \\ \Im\{c_k\} \end{cases}$	$k = 1, 3, \dots, M-1$ $k = 2, 4, \dots M$
Inv. reorder	$x_k = \begin{cases} c'_k \\ c'_{M+M/2-k+1} \end{cases}$	$k = 1, \dots, M/2$ $k = M/2+1, \dots, M$



the FCT/IFCT steps in Table II. 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 is able to compute 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 designs described next.

V. IMPLEMENTATION RESULTS

In this section, we provide reference VLSI and FPGA implementation results of AMP-M and AMP-T for the audio restoration application described in Section III-B. We emphasize that the corresponding implementation results do also reflect the performance, implementation complexity, and power consumption of the proposed VLSI designs for signal recovery from CS measurements.

A. Fixed-Point Parameters for Audio Restoration

In order to optimize the hardware efficiency (in terms of area per throughput) and the power dissipation, fixed-point arithmetic is employed in our AMP architectures. For the targeted audio restoration application, the most critical word length of the AMP-T architecture resides in the FCT/IFCT unit. To achieve the performance of a floating-point implementation with a 16 bit quantized audio input/output, 26 bit are sufficient for the real and imaginary part in the $M/2$-point FFT/IFFT block. Another important word length parameter is the accuracy of the RMSE and the resulting thresholding parameter θ. Simulation results have shown that only 11 bit are sufficient to represent θ to achieve the full RSNR performance with respect to the original audio signal. Therefore, we employ the fast and low-area square-root approximation in [35] to calculate θ. The Z-RAM uses 16 bit, the XU-, XL-, and R-RAM use 26 bit word-width. The AMP-M-architecture requires the same memory word lengths as AMP-T; the precision required in the accumulator of the MAC units in AMP-M, however, corresponds to 30 bit to achieve the performance of AMP-T. The implementation loss of both AMP architectures is less than 0.013 dB RSNR compared to their floating-point models.

In the final designs, the regularization parameter λ and the ET threshold γ are both configurable at run-time: λ is tunable to the values $\{0.125, 0.25, 0.5, 1, 2, 4\}$, whereas γ can be set to any positive number representable by 13 fraction bits.

B. Comparison of AMP-M and AMP-T

In order to compare the hardware complexity of AMP-M and AMP-T, we synthesized both architectures in a 65 nm CMOS technology. The target throughputs for both designs were 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. 5 shows the standard cell and memory area (in mm^2) after synthesis of AMP-M and AMP-T. The number of required

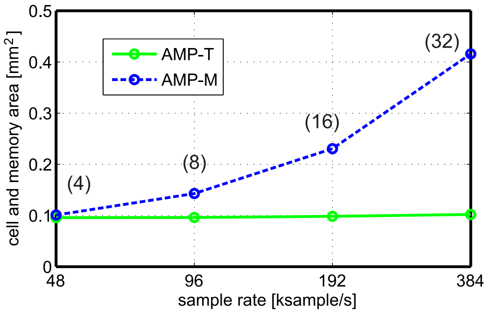


Fig. 5. Synthesis results of AMP-M and AMP-T for real-time audio restoration in 65 nm 1P8M CMOS technology. The numbers given in parentheses correspond to the number of MAC units in the associated AMP-M design.

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. The reason for this behavior is that implementing the FCT/IFCT rather than using MAC units to perform matrix-vector multiplications requires substantially fewer operations and, therefore, fewer hardware resources. This behavior is reflected in the number of clock cycles required by AMP-M and AMP-T, which can be approximated as follows:

$$C_{\text{AMP-M}} \approx 2I_{\text{max}}(4M + M^2/P) + M$$
$$C_{\text{AMP-T}} \approx 2I_{\text{max}}(7M + M\log_2(M) + 2) + M.$$

Here, P is the number of parallel MAC units. Since, for large M, AMP-M scales approximately with $I_{\text{max}}M^2/P$ and AMP-T with $I_{\text{max}}M\log_2(M)$, we conclude that the transform-based architecture is both faster and more efficient (in terms of area and power consumption) for large-scale problems.

C. ASIC Implementation

To demonstrate the efficacy of AMP-M and AMP-T, we designed a reference ASIC including both designs for real-time audio restoration in 1P8M 65 nm CMOS technology. The target is to process 192 ksample/s stereo audio signals with 16 samples overlap between adjacent blocks; this requires an AMP-throughput of 396 ksample/s. Figure 6 shows the corresponding chip layout, where we highlighted both designs and their main processing blocks. The corresponding post-layout results are listed in Table III. A detailed area and power breakdown of both ASIC designs is provided in Table IV.

From Table III, we 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. We furthermore observe that AMP-T is roughly five times smaller than AMP-M. Note that AMP-M requires less memory compared to AMP-T, which is due to the facts that i) we do not store the DCT matrix in AMP-M but compute its entries on the fly, and ii) AMP-T requires an additional memory (compared to AMP-M) within the FCT/IFCT unit.

The power figures shown in Table III and Table IV are extracted from post-layout simulations using node activities obtained from simulations with actual audio data at maximum

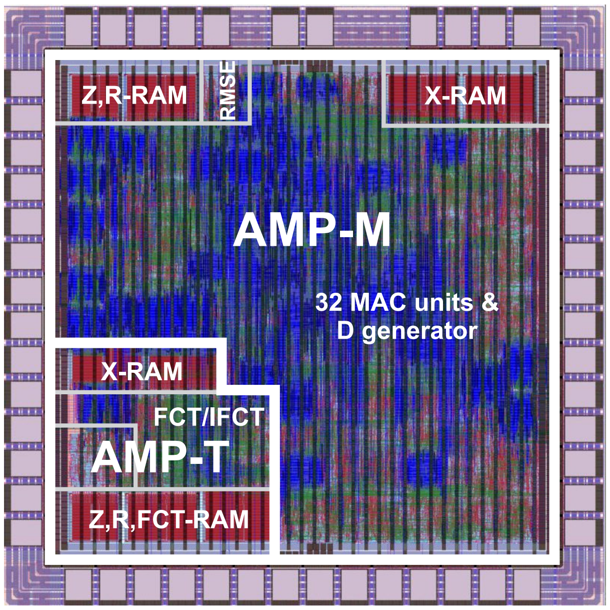


Fig. 6. Layout of an ASIC containing the AMP-M and AMP-T designs for real-time audio restoration in 1P8M 65 nm CMOS technology.

TABLE III
POST-LAYOUT IMPLEMENTATION RESULTS IN 1P8M 65 nm CMOS

	AMP-M	AMP-T
Max. clock freq. [MHz]	333	256
Throughput [ksample/s]	397	399
Cell area[a] [kGE]	302.6	33.9
Memories [kB]	5.76	7.25
Core area [mm^2]	0.629	0.136
Power consumption [mW]	177.5	24.4
Energy efficiency [µJ/sample]	0.447	0.061

[a]Standard cells only, excluding SRAM macro cells; 1 GE equals 1.44 µm^2.

clock frequency, 1.2 V core voltage, and at 298 K. AMP-T turns out to be roughly 7× more energy efficient than AMP-M in terms of μJ per sample, which highlights the effectiveness of the AMP-T design.

From Table IV 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 on-the-fly.

D. FPGA Implementation

In addition to the ASIC design shown above, we mapped both AMP architectures to Xilinx Spartan-6 FPGAs, which are fabricated in a 45 nm low-power CMOS technology. To improve the throughput of both architectures (compared to a straightforward implementation), we performed optimizations for the underlying FPGA structure. 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

TABLE IV
CELL AREA AND POWER BREAKDOWN OF THE INDIVIDUAL DESIGNS

	AMP-M		AMP-T	
	kGE[a] (%)	mW (%)	kGE[a] (%)	mW (%)
32 MAC units	197 (65)	76.5 (43)	–	–
$\mathbf{D}$ matrix gen.	93.6 (31)	88.1 (50)	–	–
FCT/IFCT unit	–	–	24.5 (72)	16.5 (68)
– butterfly	–	–	16.6	9
– twiddle gen.	–	–	2.6	0.7
RMSE	3 (1)	1.2 (1)	2.9 (9)	0.5 (2)
RAMs (X,R,Z)	–	7.9 (4)	–	6.2 (25)
Miscellaneous	9 (3)	3.8 (2)	6.5 (19)	1.2 (5)
Total	302.6 (100)	177.5 (100)	33.9 (100)	24.4 (100)

[a]Standard cells only.

TABLE V
IMPLEMENTATION RESULTS FOR XILINX SPARTAN-6 FPGAS (SG −3)

Architecture	AMP-M	AMP-T
FPGA type	XC6SLX75	XC6SLX9
Clock freq. [MHz]	41.2	35.4
Throughput [ksample/s]	47.9	92.8
Occupied slices	3525	1200
Block RAMs	24	11
DSP blocks	132	15
Power consumption[a] [mW]	516	96
Energy efficiency [μJ/sample]	10.76	1.05

[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.

the ASIC design. We also included an AC'97 audio interface to process audio signals from analog audio sources in real time. The interface consists mainly of control circuitry and in-/output buffers to implement windowing and overlapping.

1) AMP-T optimization: For the AMP-T architecture, we replaced the single-port S-RAMs with dual-port memories, since dual-port block RAMs are readily available in the FPGA used. This modification enables us to compute each butterfly operation in two clock cycles (compared to four cycles required by the architecture used in the ASIC) resulting in a $2\times$ speed-up of the FCT/IFCT. The number of clock cycles required by this modified AMP-T architecture is approximately

$$C_{\text{AMP-T2}} \approx 2I_{\max}(5M + M/2\log_2(M) + 2) + M,$$

which leads to an overall throughput increase of 40% compared to the architecture used in the AMP-T ASIC at almost no increase in FPGA logic complexity.

2) AMP-M optimizations: In the AMP-M architecture, the synthesized LUT in the $\mathbf{D}$ 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%).

3) Comparison: The FPGA implementation results of the two optimized designs are shown in Table V. Note that AMP-T can be mapped to 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 a single audio channel in real time. For stereo processing, the number of MAC units must be doubled, which would require an FPGA of twice the logic capacity. Note that the audio interface requires only 140 slices, 2 RAM blocks, and a single DSP slice.

We additionally conducted power measurements using the integrated power monitor of a Digilent Atlys prototype board. Stereo audio data is fed into the line-in port, sampled at 44.1 ksample/s, restored using AMP, and then fed to a digital-to-analog converter. The resulting power consumption and energy efficiency is reported in Table V, 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.

We emphasize 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. An even more pronounced difference can be observed in terms of power efficiency. Specifically, both ASIC designs outperform the FPGA implementations by a factor of 17 and 24 for AMP-T and AMP-M, respectively.

E. Comparison with Existing Sparse Signal Recovery Circuits

We finally compare both AMP designs to the ASIC implementations of MP and OMP presented in [15] for channel estimation in 3GPP-LTE.[2] A direct comparison is difficult, because the ASICs in [15] perform sparse signal recovery in 0.5 ms of signals with roughly 12 to 18 significant entries and problems of dimension 200×256; moreover, both applications have different precision requirements.

Nevertheless, by scaling[3] the required operations per time unit of the MP and OMP implementation using results of [15, Table I] to the throughput required by the single-channel audio restoration problem considered here, the estimated circuit area of OMP is more than $4\times$ larger than AMP-M, which is mainly caused by the complexity required by LS estimations for the high sparsity levels typically arising in audio signals or images. For signals having very low sparsity levels (as it is the case in sparse channel estimation, for example), however, OMP is likely to be more efficient than AMP.

The scaled circuit area of MP requires only half the area of AMP-T, but delivers inferior performance when used for signal restoration or CS applications with strong undersampling. Nevertheless, MP remains a valid low-complexity alternative to AMP in applications where sub-optimal sparse signal recovery performance can be tolerated.

[2]We do not provide a comparison with the FPGA implementations in [16], [26], as details about the performance and complexity scaling for larger problem sizes are missing; furthermore, the recovery performance for approximately sparse signals is unknown.

[3]We assume that the circuit area of the implementations in [15] scales linearly with the number of operations per time unit. Furthermore, MP and OMP are assumed to require 200 and 100 iterations, respectively.

VI. Conclusions

Among the two generic VLSI architectures of the approximate message passing (AMP) algorithm for sparse signal recovery, the first one, referred to as AMP-M, was shown to be suitable for the recovery of signals acquired by compressive sensing (CS) or signal restoration problems relying on unstructured (e.g., random or learned) matrices. The second architecture, referred to as AMP-T, is able to exploit fast transforms, which significantly reduces circuit area and power dissipation compared to AMP-M. To demonstrate the suitability of AMP for real-time sparse signal recovery in dedicated hardware, we have implemented both architectures in 65 nm CMOS technology for a high-rate audio restoration application. Moreover, we demonstrated the real-time restoration capabilities of both architectures using an FPGA prototype implementation.

There are many avenues for future work. A theoretical performance analysis of AMP in the presence of fixed-point arithmetic and early termination is a challenging open research topic. On the VLSI implementation side, developing an AMP-T architecture suitable for real-time recovery of images or videos from CS measurements is part of ongoing work.

References

[1] M. Elad, *Sparse and Redundant Representations – From Theory to Applications in Signal and Image Processing*. New York, NY, USA: Springer, 2010.

[2] A. Adler, V. Emiya, M. G. Jafari, M. Elad, R. Gribonval, and M. D. Plumbley, "A constrained matching pursuit approach to audio declipping," in *Proc. IEEE ICASSP*, Prague, Czech Republic, May 2011, pp. 329–332.

[3] C. Studer, P. Kuppinger, G. Pope, and H. Bölcskei, "Recovery of sparsely corrupted signals," *IEEE Trans. Inf. Theory*, vol. 58, no. 5, pp. 3115–3130, May 2012.

[4] C. Studer and R. G. Baraniuk, "Recovery guarantees for restoration and separation of approximately sparse signals," in *Proc. 49th Ann. Allerton Conf. on Comm., Control, and Computing*, Sept. 2011, pp. 736–743.

[5] G. Kutyniok, "Data separation by sparse representations," in *Compressed Sensing: Theory and Applications*, Y. C. Eldar and G. Kutyniok, Eds. New York, NY, USA: Cambridge University Press, 2012.

[6] S. G. Mallat, *A Wavelet Tour of Signal Processing: The Sparse Way*. Burlington, MA, USA: Academic Press, 2009.

[7] J.-F. Cai, S. Osher, and Z. Shen, "Split Bregman methods and frame based image restoration," *Multiscale Modeling & Simulation*, vol. 8, no. 2, pp. 337–369, Jan. 2010.

[8] M. Elad, J.-L. Starck, P. Querre, and D. L. Donoho, "Simultaneous cartoon and texture image inpainting using morphological component analysis (MCA)," *Appl. Comput. Harmon. Anal.*, vol. 19, no. 3, pp. 340–358, Jan. 2005.

[9] D. Donoho, "Compressed sensing," *IEEE Trans. Inf. Theory*, vol. 52, no. 4, pp. 1289–1306, Apr. 2006.

[10] E. Candés, J. Romberg, and T. Tao, "Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information," *IEEE Trans. Inf. Theory*, vol. 52, no. 2, pp. 489–509, Feb. 2006.

[11] M. Lustig, D. L. Donoho, and J. M. Pauly, "Sparse MRI: The application of compressed sensing for rapid MR imaging," *Mag. Reson. in Med.*, vol. 58, no. 6, pp. 1182–1195, Dec. 2007.

[12] J. Haboba, M. Mangia, R. Rovatti, and G. Setti, "An architecture for 1-bit localized compressive sensing with applications to EEG," in *Proc. IEEE BioCas*, Sand Diego, CA, USA, Nov. 2011, pp. 137–140.

[13] M. Duarte, M. Davenport, D. Takhar, J. Laska, T. Sun, K. Kelly, and R. Baraniuk, "Single-pixel imaging via compressive sampling," *IEEE Sig. Proc. Mag.*, vol. 25, no. 2, pp. 83–91, Mar. 2008.

[14] R. Baraniuk and P. Steeghs, "Compressive radar imaging," in *Proc. IEEE Radar Conf.*, Boston, MA, USA, Apr. 2007, pp. 128–133.

[15] P. Maechler, P. Greisen, B. Sporrer, S. Steiner, N. Felber, and A. Burg, "Implementation of greedy algorithms for LTE sparse channel estimation," in *Proc. 44th Asilomar Conf. Signals, Systems and Computers*, Pacific Grove, CA, USA, Nov. 2010, pp. 400–405.

[16] M. Mishali, R. Hilgendorf, E. Shoshan, I. Rivkin, and Y. C. Eldar, "Generic sensing hardware and real-time reconstruction for structured analog signals," in *Proc. ISCAS*, Rio de Janeiro, Brazil, May 2011, pp. 1748–1751.

[17] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004.

[18] I. Daubechies, M. Defrise, and C. de Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Comm. Pure Appl. Math.*, vol. 57, no. 11, pp. 1413–1457, Aug. 2004.

[19] S. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Trans. Signal Process.*, vol. 41, no. 12, pp. 3397–3415, Dec. 1993.

[20] P. L. Combettes and V. R. Wajs, "Signal recovery by proximal forward-backward splitting," *SIAM J. Multiscale Model. Simul.*, vol. 4, no. 4, pp. 1168–1200, 2005.

[21] T. Hale, W. Yin, and Y. Zhang, "A fixed-point continuation method for ℓ_1-regularized minimization with applications to compressed sensing," Dept. Computat. Appl. Math., Rice Univ., Houston, TX, Tech. Rep. TR07-07, 2007.

[22] Y. Pati, R. Rezaiifar, and P. Krishnaprasad, "Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition," in *Proc. 27th Asilomar Conf. Signals, Systems and Computers*, Pacific Grove, CA, USA, Nov. 1993, pp. 40–44.

[23] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Imag. Sci.*, vol. 2, no. 1, pp. 183–202, Jan. 2009.

[24] D. Needell and J. Tropp, "CoSaMP: Iterative signal recovery from incomplete and inaccurate samples," *Appl. Comput. Harmon. Anal.*, vol. 26, no. 3, pp. 301–321, May 2009.

[25] D. Donoho, A. Maleki, and A. Montanari, "Message-passing algorithms for compressed sensing," *Proc. of the National Academy of Sciences*, vol. 106, no. 45, pp. 18 914–18 919, Sept. 2009.

[26] A. Septimus and R. Steinberg, "Compressive sampling hardware reconstruction," in *Proc. ISCAS*, Rio de Janeiro, Brazil, May 2010, pp. 3316–3319.

[27] R. G. Baraniuk, M. A. Davenport, R. A. DeVore, and M. B. Wakin, "A simple proof of the restricted isometry property for random matrices," *Constr. Approx.*, vol. 28, no. 3, pp. 253–263, Dec. 2008.

[28] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 33–61, Aug. 1998.

[29] A. Maleki, "Coherence analysis of iterative thresholding algorithms," in *Proc. 47th Ann. Allerton Conf. on Comm., Control, and Computing*, Monticello, IL, USA, Sep. 2009, pp. 236–243.

[30] A. Montanari, "Graphical models concepts in compressed sensing," *arXiv:1011.4328v3*, Mar. 2011.

[31] A. Maleki, "Approximate message passing algorithms for compressed sensing," Ph.D. dissertation, Stanford University, Stanford, CA, USA, Jan. 2011.

[32] S. J. Godsill and P. J. W. Rayner, *Digital audio restoration*. Springer-Verlag, London, 1998.

[33] R. G. Baraniuk, "Goodbye, textbooks; hello, open-source learning," [online accessed on July 1, 2011] URL `http://www.ted.com/talks`, Feb. 2006.

[34] M. Aharon, M. Elad, and A. M. Bruckstein, "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Trans. Sig. Proc.*, vol. 54, no. 11, pp. 4311–4322, Nov. 2006.

[35] I. Park and T. Kim, "Multiplier-less and table-less linear approximation for square and square-root," in *Proc. IEEE Int. Conf. Computer Design*, Lake Tahoe, CA, USA, Oct. 2009, pp. 378–383.

[36] M. Vetterli and A. Ligtenberg, "A discrete fourier-cosine transform chip," *IEEE J. Sel. Areas Commun.*, vol. 4, no. 1, pp. 49–61, Jan. 1986.

[37] W. Chen, C. Smith, and S. Fralick, "A fast computational algorithm for the discrete cosine transform," *IEEE Trans. Commun.*, vol. 25, no. 9, pp. 1004–1009, Sept. 1977.

[38] M. Vetterli and H. Nussbaumer, "Simple FFT and DCT algorithms with reduced number of operations," *Signal Processing*, vol. 6, no. 4, pp. 267–278, Aug. 1984.

[39] N. Ahmed, T. Natarajan, and K. Rao, "Discrete cosine transform," *IEEE Trans. Comput.*, vol. 23, no. 1, pp. 90–93, Jan. 1974.

[40] J. Makhoul, "A fast cosine transform in one and two dimensions," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 28, no. 1, pp. 27–34, Feb. 1980.

[41] Y. Ma, "An effective memory addressing scheme for FFT processors," *IEEE Trans. Signal Process.*, vol. 47, no. 3, pp. 907–911, Mar. 1999.