SPARSE BAYESIAN LEARNING FOR DIRECTIONS OF ARRIVAL ON AN FPGA

Herbert Groll[1]*, Christoph Mecklenbräuker*[1] *and Peter Gerstoft*[2]

[1]TU Wien, Vienna, Austria
[2]University of California San Diego, La Jolla, CA, USA

ABSTRACT

A direction of arrival (DOA) estimator based on sparse Bayesian learning (SBL) is implemented as a fixed-point arithmetic prototype for an FPGA platform. The prototype is developed from a known algorithm mainly using high-level synthesis with C++ based model specifications. The specialized equations of the algorithm are reduced to arithmetic operations considering the signal flow within the iterative structure. Cholesky factorization is used to solve the matrix inverse problem. Scheduling of each module is done as soon as possible to make use of the parallel FPGA architecture. Different fixed-point word length assumptions are explained and implementation results are shown in terms of resources and latency. Finally, a representative DOA source scenario is simulated and tested with the implemented prototype hardware in the loop. The comparison with a floating-point reference implementation is found to have good agreement with the fixed-point implementation.

Index Terms— Array processing, directions of arrival (DOA) estimation, FPGA, high-level synthesis

1. INTRODUCTION

Direction of arrival (DOA) estimation is used in radar, communication, sonar and seismology applications for localizing sources by sensing wavefields. A sparse representation of the DOA problem allows estimation with high spatial or temporal resolution [1]. Sparse Bayesian learning (SBL) [2], solves this undetermined linear inverse problem for which numerous algorithms have been presented [3–5].

A field-programmable gate array (FPGA) implementation for SBL is of great interest due to possible real-time dimension reduction of online measurements, e.g. in directional evaluation of fading and two-ray model measurements [6–8]. Moreover, FPGAs are becoming more tightly integrated together with high-speed analog-digital converters (ADCs) which enable new applications. The use of arbitrary precision fixed-point arithmetic in FPGAs offers fast processing without loosing too much accuracy.

In this paper we present an FPGA implementation of the SBL DOA estimator in [4] using fixed-point arithmetic. The SBL algorithm performs better than MUSIC [4] and ultimately estimates one signal power parameter per source rather than the complex source amplitudes individually per snapshot. This amounts to a significant reduction of the degrees of freedom in the estimation problem resulting in low variance of the DOA estimates. FPGA-based DOA estimation implementations using MUSIC were presented in [9, 10] which have been realized by hardware description on the register-transfer level. Highly developed FPGAs enable rapid prototyping of algorithms using high-level synthesis (HLS) for FPGA as used in this work. A prototyping setup similar to [11] ensures fast verification of the prototype.

2. SIGNAL MODEL

A K-sparse vector $\boldsymbol{x}_l \in \mathbb{C}^M$ is observed as signal $\boldsymbol{y}_l \in \mathbb{C}^N$ on a sensor array with N sensors, with $N \ll M$, through linear transformation with a transfer matrix $\boldsymbol{A} = [\boldsymbol{a}_1, \ldots, \boldsymbol{a}_M] \in \mathbb{C}^{N\times M}$ and noise $\boldsymbol{n}_l$. A grid with M points define the possible DOA of K impinging waves with far-field and narrowband assumption at a fixed frequency ω and velocity of propagation c. The indices of the K nonzero sources of $\boldsymbol{x}_l$, where $K \ll M$, define the active set $\mathcal{M}_l = \{m \in \mathbb{N} | x_{ml} \neq 0\} = \{m_1, m_2, \ldots, m_K\}$. A total of L snapshots are taken with stationary active set $\mathcal{M}_l = \mathcal{M}$ to form the multi-snapshot, or multiple measurement vector (MMV), $\boldsymbol{Y} = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_L] \in \mathbb{C}^{N\times L}$ as

$$\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{X} + \boldsymbol{N} \tag{1}$$

with $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L] \in \mathbb{C}^{M\times L}$ and additive noise $\boldsymbol{N} = [\boldsymbol{n}_1, \ldots, \boldsymbol{n}_L] \in \mathbb{C}^{N\times L}$ which is assumed to be i.i.d complex Gaussian, $\mathcal{CN}(0, \sigma^2)$, across sensors and snapshots.

Each array steering vector $\boldsymbol{a}_m$, as m-th column of $\boldsymbol{A}$, describes the time delays $\boldsymbol{\tau}_m = [\tau_{m1}, \ldots, \tau_{mN}]^T$ with respect to the sensors for a wave front with direction θ_m. For a uniform linear array (ULA) with element spacing d, the nm-th element of $\boldsymbol{A}$ is $e^{-j\omega\tau_{nm}} = e^{-j(n-1)\frac{\omega d}{c}\sin\theta_m}$.

2.1. Bayesian Formulation

The SBL framework [4] based on [3] defines a Gaussian likelihood function of $\boldsymbol{Y}|\boldsymbol{X}$ [4, Eq. (3)] as

$$p(\boldsymbol{Y}|\boldsymbol{X}; \sigma^2) = \frac{\exp(-\frac{1}{\sigma^2}\|\boldsymbol{Y} - \boldsymbol{A}\boldsymbol{X}\|_{\mathcal{F}}^2)}{(\pi\sigma^2)^{NL}}, \tag{2}$$

Authorized licensed use limited to: INSTITUTE OF ACOUSTICS CAS. Downloaded on December 30,2023 at 12:37:06 UTC from IEEE Xplore. Restrictions apply.

additive complex Gaussian noise with variable variance σ^2. The prior distribution of each complex source amplitude x_{ml} is modeled with hyperparameters $\gamma_m \in \boldsymbol{\gamma} = [\gamma_1, \dots, \gamma_M]^T$ stationary across different snapshots l as

$$p_m(x_{ml};\gamma_m) = \begin{cases} \delta(x_{ml}), & \text{for } \gamma_m = 0 \\ \frac{1}{\pi\gamma_m}\mathrm{e}^{-|x_{ml}|^2/\gamma_m}, & \text{for } \gamma_m > 0 \end{cases} \tag{3}$$

$$p(\boldsymbol{X};\boldsymbol{\gamma}) = \prod_{l=1}^{L} \mathcal{CN}(\boldsymbol{0},\boldsymbol{\Gamma}) \tag{4}$$

with $\boldsymbol{\Gamma} = \mathrm{diag}(\boldsymbol{\gamma}) = \mathsf{E}[\boldsymbol{x}_l\boldsymbol{x}_l^H;\boldsymbol{\gamma}]$, the covariance matrix of the uncorrelated complex source amplitudes. The SBL algorithm of [4] estimates the source powers by estimating the hyperparameters $\boldsymbol{\Gamma}$.

3. HARDWARE ALGORITHM

Table 1. SBL FPGA Algorithm

```
1: Input: $\boldsymbol{Y}$
2: Initialize: $\boldsymbol{L_\Sigma} = \text{cholesky}\left(0.1 \cdot \boldsymbol{I}_N + \boldsymbol{A}\boldsymbol{A}^H\right)$, $j_\text{max} = 500$
3: $\boldsymbol{S}_y = \frac{1}{L}\sum_{l=1}^{L} \boldsymbol{y}_l\boldsymbol{y}_l^H$
4: $\text{tr}(\boldsymbol{S}_y)$
5: while $\Delta\gamma_M > \epsilon_\text{min} \cdot \|\boldsymbol{\gamma}^\text{old}\|_1$ and $j < j_\text{max}$ do
6:     $j = j + 1$, $\boldsymbol{\gamma}^\text{old} = \boldsymbol{\gamma}^\text{new}$
7:     for $m = 1$ to $M$ do
8:         $\gamma_m^\text{new} = \gamma_m^\text{old} \cdot \sqrt{\frac{(\boldsymbol{L_\Sigma}^{-1}\boldsymbol{a}_m)^{-H}\boldsymbol{L_\Sigma}^{-1}\boldsymbol{S}_y\boldsymbol{L_\Sigma}^{-H}(\boldsymbol{L_\Sigma}^{-1}\boldsymbol{a}_m)}{\|\boldsymbol{L_\Sigma}^{-1}\boldsymbol{a}_m\|_2^2}}$
9:         $\boldsymbol{\Sigma}_{y,m} = \boldsymbol{\Sigma}_{y,m-1} + \boldsymbol{a}_m\boldsymbol{a}_m^H\gamma_m^\text{new}$
10:        $\Delta\gamma_m = \Delta\gamma_{m-1} + |\gamma_m^\text{new} - \gamma_m^\text{old}|$
11:        $g_m = \sum_{i=m-1}^{m+1}\gamma_i^\text{new}$
12:        $p_m = \begin{cases} g_m & (g_{m-1} < g_m \le g_{m+1}) \wedge (1 < m < M) \\ 0 & \text{else} \end{cases}$
13:        $\mathcal{M}_m = \{i \in (\mathcal{M}_{m-1} \cup m) \mid K \text{ largest peaks in } p_i\}$
14:    end for
15:    $\boldsymbol{A}_\mathcal{M} = [\boldsymbol{a}_{m_1}, \dots, \boldsymbol{a}_{m_K}]$
16:    $\boldsymbol{R}_\mathcal{M} = \text{cholesky}\left(\boldsymbol{A}_\mathcal{M}^H\boldsymbol{A}_\mathcal{M}\right)$
17:    $\boldsymbol{Q} = \boldsymbol{A}_\mathcal{M}\boldsymbol{R}_\mathcal{M}^{-1}$
18:    $(\sigma^2)^\text{new} = \frac{1}{N-K}\left(\text{tr}(\boldsymbol{S}_y) - \text{tr}(\boldsymbol{Q}^H\boldsymbol{S}_y\boldsymbol{Q})\right)$
19:    $\boldsymbol{\Sigma}_y = \boldsymbol{\Sigma}_{y,M} + (\sigma^2)^\text{new}\boldsymbol{I}_N$
20:    $\boldsymbol{L_\Sigma} = \text{cholesky}(\boldsymbol{\Sigma}_y)$
21: end while
22: Output: $\mathcal{M}$, $\boldsymbol{\gamma}^\text{new}$, $(\sigma^2)^\text{new}$
```

A SBL algorithm, based on [4, Algorithm 1] and adopted for FPGA hardware implementation, is summarized in Table 1. At the input, each multi-snapshot $\boldsymbol{Y}$ is used only as data sample covariance matrix $\boldsymbol{S}_y$, defined as

$$\boldsymbol{S}_y = \frac{1}{L}\sum_{l=1}^{L}\boldsymbol{y}_l\boldsymbol{y}_l^H \tag{5}$$

which leads to an averaging over L snapshots. Only the lower triangle of Hermitian $\boldsymbol{y}_l\boldsymbol{y}_l^H$ is calculated during snapshot acquisition. Double-buffering ensures gap-free operation between multi-snapshots. By allowing only $L = 2^{L_b}$ with $L_b \in \mathbb{N}$, $\boldsymbol{S}_y$ is derived through reinterpretation of the fixed-point data type, i.e. shifting the binary point. The hyperparameters γ_m^new in [4, Eq. (SBL1)] are updated iteratively as

$$\gamma_m^\text{new} = \frac{\gamma_m^\text{old}}{\sqrt{L}}\left\|\boldsymbol{Y}^H\boldsymbol{\Sigma}_y^{-1}\boldsymbol{a}_m\right\|_2 / \sqrt{\boldsymbol{a}_m^H\boldsymbol{\Sigma}_y^{-1}\boldsymbol{a}_m} \tag{6}$$

with the inverse data model covariance matrix $\boldsymbol{\Sigma}_y^{-1}$. [4, Eq. (SBL)] and [4, Eq. (M-SBL)] are not considered in this work. $\boldsymbol{\Sigma}_y$ is a Hermitian positive definite matrix with its lower triangular Cholesky factor

$$\boldsymbol{L_\Sigma} = \text{cholesky}\left(\boldsymbol{\Sigma}_y\right), \tag{7}$$

where $\boldsymbol{\Sigma}_y = \boldsymbol{L_\Sigma}\boldsymbol{L_\Sigma}^H$ is the Cholesky factorization. $\boldsymbol{L_\Sigma}$ is used to solve

$$\boldsymbol{\Sigma}_y^{-1}\boldsymbol{a}_m = \boldsymbol{L_\Sigma}^{-H}\boldsymbol{L_\Sigma}^{-1}\boldsymbol{a}_m \tag{8}$$

by forward- and back-substitution. Using Eq. (8) and $\boldsymbol{S}_y$, updating hyperparameters γ_m^new changes to

$$\gamma_m^\text{new} = \gamma_m^\text{old} \cdot \sqrt{\frac{(\boldsymbol{L_\Sigma}^{-1}\boldsymbol{a}_m)^{-H}\boldsymbol{L_\Sigma}^{-1}\boldsymbol{S}_y\boldsymbol{L_\Sigma}^{-H}(\boldsymbol{L_\Sigma}^{-1}\boldsymbol{a}_m)}{\|\boldsymbol{L_\Sigma}^{-1}\boldsymbol{a}_m\|_2^2}} \tag{SBL1}$$

where $\boldsymbol{L_\Sigma}^{-1}\boldsymbol{a}_m$ is reused in the denominator as $\boldsymbol{a}_m^H\boldsymbol{\Sigma}_y^{-1}\boldsymbol{a}_m = \|\boldsymbol{L_\Sigma}^{-1}\boldsymbol{a}_m\|_2^2$. We reuse intermediate results during Cholesky factorization for forward- or back-substitution to reduce costly $^1/_x$-operations.

The data model covariance matrix $\boldsymbol{\Sigma}_y$ is calculated in $M + 1$ passes. First, all contributions from each γ_m are summed

$$\boldsymbol{\Sigma}_{y,m} = \boldsymbol{\Sigma}_{y,m-1} + \boldsymbol{a}_m\boldsymbol{a}_m^H\gamma_m^\text{new}, \quad \boldsymbol{\Sigma}_{y,0} = \boldsymbol{0} \tag{9}$$

by only evaluating the lower triangle of $\boldsymbol{a}_m\boldsymbol{a}_m^H$. Finally, the noise estimate is added to the real diagonal as

$$\boldsymbol{\Sigma}_y = \boldsymbol{\Sigma}_{y,M} + (\sigma^2)^\text{new}\boldsymbol{I}_N. \tag{10}$$

For finding the active set $\mathcal{M}$, we use a 3-stage processing pipeline. First, to reduce false peaks due to limited precision we filter the $\boldsymbol{\gamma}^\text{new} = [\gamma_1^\text{new}, \dots, \gamma_M^\text{new}]^T$ with a moving sum of size 3

$$g_m = \sum_{i=m-1}^{m+1}\gamma_i^\text{new}. \tag{11}$$

No scaling by $^1/_3$ is applied as only indices are relevant. Second, we use a trivial peak detection algorithm on g_m

$$p_m = \begin{cases} g_m & (g_{m-1} < g_m \le g_{m+1}) \wedge (1 < m < M) \\ 0 & \text{else.} \end{cases} \tag{12}$$

Authorized licensed use limited to: INSTITUTE OF ACOUSTICS CAS. Downloaded on December 30,2023 at 12:37:06 UTC from IEEE Xplore. Restrictions apply.

Third, the K largest peaks are selected in M passes out of $K+1$ peak candidates

$$\mathcal{M}_m = \{i \in (\mathcal{M}_{m-1} \cup m) \mid K \text{ largest peaks in } p_i\} \quad (13)$$

with $\mathcal{M} = \mathcal{M}_M$ and $\mathcal{M}_0 = \{\}$. The transfer matrix defined by the active set is $\boldsymbol{A}_{\mathcal{M}} \in \mathbb{C}^{N\times K}$. Using the upper triangular Cholesky factor

$$\boldsymbol{R}_{\mathcal{M}} = \text{cholesky}\left(\boldsymbol{A}_{\mathcal{M}}^H \boldsymbol{A}_{\mathcal{M}}\right) \quad (14)$$

with $\boldsymbol{A}_{\mathcal{M}}^H \boldsymbol{A}_{\mathcal{M}} = \boldsymbol{R}_{\mathcal{M}}^H \boldsymbol{R}_{\mathcal{M}}$, the projection matrix $\boldsymbol{P}$ based on the active set is

$$\boldsymbol{P} = \boldsymbol{A}_{\mathcal{M}}(\boldsymbol{R}_{\mathcal{M}}^H \boldsymbol{R}_{\mathcal{M}})^{-1} \boldsymbol{A}_{\mathcal{M}}^H = \boldsymbol{Q}\boldsymbol{Q}^H \quad (15)$$

with $\boldsymbol{Q} \in \mathbb{C}^{N\times K}$ as

$$\boldsymbol{Q} = \boldsymbol{A}_{\mathcal{M}} \boldsymbol{R}_{\mathcal{M}}^{-1}, \quad (16)$$

calculated with back-substitution. Alternatively, a matrix $\boldsymbol{Q}$ can be directly obtained by QR factorization. However, Vivado HLS only includes a floating-point implementation of an QR factorization but a fixed-point Cholesky factorization.

The noise variance estimation [4, Eq. (27)] is rewritten using Eq. (15), Eq. (16), and $\text{tr}(\boldsymbol{Q}\boldsymbol{Q}^H\boldsymbol{S}_y) = \text{tr}(\boldsymbol{Q}^H\boldsymbol{S}_y\boldsymbol{Q})$ as

$$(\sigma^2)^{\text{new}} = \frac{1}{(N-K)}\left(\text{tr}(\boldsymbol{S}_y) - \text{tr}(\boldsymbol{Q}^H\boldsymbol{S}_y\boldsymbol{Q})\right) \quad (17)$$

with constant $(N-K)^{-1}$, a signal part $\text{tr}(\boldsymbol{S}_y)$ which is calculated only once per multi-snapshot, and an active signal part $\text{tr}(\boldsymbol{Q}^H\boldsymbol{S}_y\boldsymbol{Q})$ which only needs the upper triangle of $\boldsymbol{S}_y$. Hyperparameters ($\boldsymbol{\gamma}^{\text{new}}$, $(\sigma^2)^{\text{new}}$) are updated iteratively up to $j_{\max}$ iterations or as long as source power changes sufficiently,

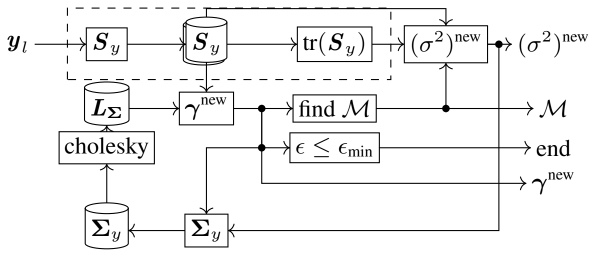
$$\Delta\gamma_m = \Delta\gamma_{m-1} + |\gamma_m^{\text{new}} - \gamma_m^{\text{old}}|, \quad \Delta\gamma_0 = 0 \quad (18)$$

$$\Delta\gamma_M > \epsilon_{\min} \cdot \|\boldsymbol{\gamma}^{\text{old}}\|_1, \quad (19)$$

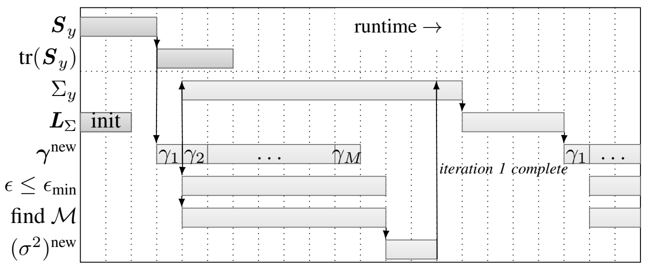
reformulated from the convergence rate ϵ [4, Eq. 25] to avoid a division and optimize for serial input.

4. HARDWARE IMPLEMENTATION

The algorithm [4, Algorithm 1] is split up into modules according to Fig. 1 (a) with well defined interfaces. Either streaming or dual-port memory is used between the modules. *Xilinx Vivado HLS 2017.2* is used for transforming the hardware specification of individual modules on an algorithmic level based on C++ into a specification in Verilog or VHDL on the register-transfer level. We use several modules, opposed to a monolithic HLS description of the algorithm, which reduces complexity for the HLS tool and lead to more predictable results. Data dependencies between modules translate to a coarse grain scheduling scheme as shown in Fig. 1 (b). The development of each module focuses on the algorithmic level using a subset of C++, allowed by *Vivado HLS* to produce synthesizable results. HLS linear algebra library and custom operations are both using arbitrary precision fixed-point signed and unsigned data types `ap_fixed<W,WI>` and `ap_ufixed<W,WI>` each with total bit-width `W`, integer width `WI`, and fraction width `WF=W-WI`.


$\boldsymbol{y}_l$
$\boldsymbol{S}_y$
$\boldsymbol{S}_y$
tr($\boldsymbol{S}_y$)
$(\sigma^2)^{\text{new}}$
$(\sigma^2)^{\text{new}}$
$\boldsymbol{L}_{\boldsymbol{\Sigma}}$
$\boldsymbol{\gamma}^{\text{new}}$
find $\mathcal{M}$
$\mathcal{M}$
cholesky
$\epsilon \leq \epsilon_{\min}$
end
$\boldsymbol{\gamma}^{\text{new}}$
$\boldsymbol{\Sigma}_y$
$\boldsymbol{\Sigma}_y$

(a) data path



(b) Scheduling

Fig. 1. (a) data path and (b) scheduling of modules for the implemented SBL algorithm. Structure for iterative fitting of $\boldsymbol{\Sigma}_y$. $\boldsymbol{S}_y$ and $\text{tr}(\boldsymbol{S}_y)$ need to be calculated only once per multi-snapshot. Data dependencies limit possible parallelism of operations.

The input $\boldsymbol{y}_l$ is a 32 bit wide sensor data stream. Each sensor sample consists of concatenated two's complement signed real and imaginary part with `W`=16 bit each and `WI=2` to allow values symmetric around zero. This enables, e.g., feeding the system with samples from an ADC, optionally prepended with automatic gain control (ACG). The outputs of the system are $(\sigma^2)^{\text{new}}$ with `W`=32 bit, $\boldsymbol{\gamma}^{\text{new}}$ with `W`=48 bit, active set $\mathcal{M}$ as unsigned integers, and exit criteria signaling. The transfer $\boldsymbol{A}$ is pre-calculated for a given sensor array with `W`=16 bit, implemented as a look-up table and instantiated at several HLS modules. Normalization of $\boldsymbol{A}$ by $1/\sqrt{M}$ is beneficial for required `WI` when calculating $\boldsymbol{L}_{\boldsymbol{\Sigma}}$.

4.1. Hardware Results

The resources and latencies of the synthesized system are based on the specific parameters M, N, K, and clock frequency f_{clk}. For $M = 512$, $N = 16$, $K = 3$ and $f_{\text{clk}} = 150$ MHz, a summary of resources for each module is in Fig. 2 together with introduced latency by scheduling a single mod-

Authorized licensed use limited to: INSTITUTE OF ACOUSTICS CAS. Downloaded on December 30,2023 at 12:37:06 UTC from IEEE Xplore. Restrictions apply.

ule. The synthesized HLS C++ implementation can processes $43\,\text{iterations/s}$. This is faster than the MATLAB implementation (floating-point reference) of $24\,\text{iterations/s}$ on a *Intel Xeon CPU E5-2690 v3* at $2.60\,\text{GHz}$.

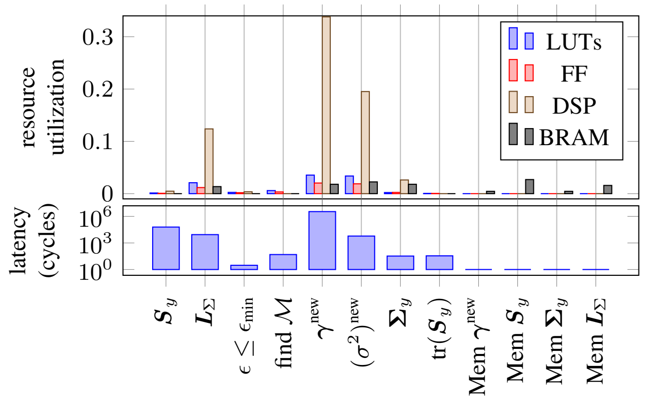


Fig. 2. Resource utilization and introduced latency for each module. Based on used Kintex-7 XC7K325T FPGA with Slice look-up-tables (LUTs), Slice registers (FF), special digital-signal-processing (DSP) slices, block RAM (BRAM) of $18\,\text{kbit}$. Total LUT=203800, FF=407600, DSP=840, BRAM=890.

Clearly, calculating γ_m^{new} utilizes most resources and contributes severely to the overall latency, e.g. due to $1/\sqrt{x}$, which could be improved by fast approximation techniques [12] or stronger parallelism.

5. SIMULATION RESULTS

The fixed-point prototype, developed with *Vivado HLS*, is running on an Kintex 7 FPGA and connected through a TCP/IP prototyping setup with MATLAB. It is compared with a floating-point reference in MATLAB. An example scenario similar to [4] places $K = 3$ independent sources on an angular grid in the interval $\theta \in [-90, 90)^\circ$ with $M = 512$ different angles. The sources are located at -3.3, 1.9, and 74.9° with magnitudes 12, 22, and $20\,\text{dB}$. An ULA with $N = 16$ sensors is used. Additive i.i.d. complex Gaussian noise is added according to a given array SNR $= 20\log_{10}(\|\boldsymbol{A}\boldsymbol{x}_l\|_2/n_{\text{rel}})$. The MMV $\boldsymbol{Y}$ has $L = 64$ snapshots, is mapped to $[-1,1] + [-1,1]j$ for both, and quantized for the fixed-point prototype. A Monte Carlo simulation with $J = 100$ realizations are carried out. A widely used quality measure for DOA estimation is

$$\text{DOA RMSE} = \sqrt{\frac{1}{K}\frac{1}{J}\sum_k^K\sum_j^J |\theta_{m_k} - \hat{\theta}_{m_k,j}|^2} \qquad (20)$$

with k-th source direction θ_{m_k} and k-th estimated direction $\hat{\theta}_{m_k,j}$, for which the fixed-point FPGA implementation

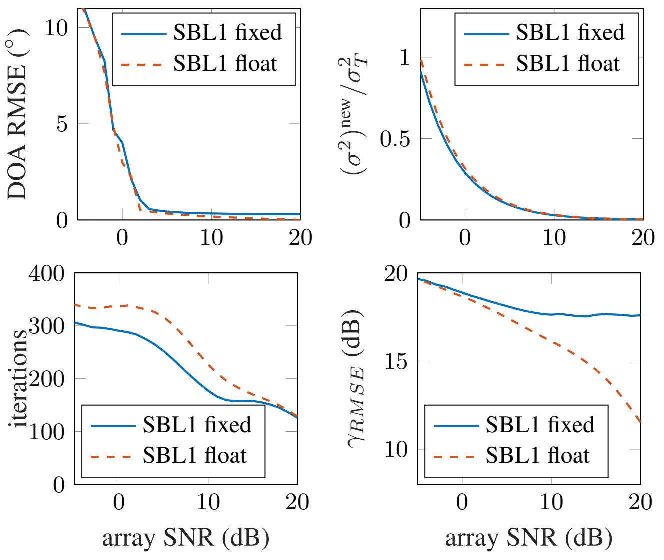


Fig. 3. Comparison between the presented fixed-point implementation and floating-point reference for $J = 100$ Monte Carlo simulations. (a) DOA estimation error, (b) noise power estimation, (c) iterations for $\epsilon \leq \epsilon_{\min}$, (d) source power estimation error.

is compared with the floating-point reference in Fig. 3 (a). Noise power estimation is shown in Fig. 3 (b) with respect to $\sigma_T = 10^{-\text{SNR}/10}\mathsf{E}[\|\boldsymbol{A}\boldsymbol{X}\|_{\mathcal{F}}^2]/L/N$. The mean number of iterations necessary for the stop criteria Eq. (19) is shown in Fig. 3 (c). The error in source power estimation is shown in Fig. 3 (d) as

$$\gamma_{\text{RMSE}} = \sqrt{\frac{1}{K}\frac{1}{J}\sum_k^K\sum_j^J |\mathsf{E}[|x_{m_k,l}|^2]_j - \gamma_{m_k,j}^{\text{new}}|^2} \qquad (21)$$

with k-th source power $\mathsf{E}[|x_{m_k,l}|^2]_j$ and k-th estimated source power $\gamma_{m_k,j}^{\text{new}}$. For high array SNR, estimated source power of the fixed-point implementation is less accurate than the floating-point reference due to limited dynamic range in calculating a single iteration.

6. CONCLUSION

The presented implementation of a sparse Bayesian learning algorithm for directions of arrival estimation based on [4] is suitable for FPGA implementation using on fixed-point arithmetic. It offers higher performance with continuous operation and has a good agreement with the floating-point reference in terms of DOA root mean squared error. Several signal processing steps of the SBL algorithm benefit from the parallelism inherent to the FPGA architecture.

Authorized licensed use limited to: INSTITUTE OF ACOUSTICS CAS. Downloaded on December 30,2023 at 12:37:06 UTC from IEEE Xplore. Restrictions apply.

7. REFERENCES

[1] D. Malioutov, M. Cetin, and A. S. Willsky, "A sparse signal reconstruction perspective for source localization with sensor arrays," *IEEE Transactions on Signal Processing*, vol. 53, no. 8, pp. 3010–3022, Aug 2005.

[2] M. E. Tipping, "Sparse Bayesian learning and the relevance vector machine," *Journal of machine learning research*, vol. 1, pp. 211–244, Jun 2001.

[3] D. P. Wipf and B. D. Rao, "An empirical Bayesian strategy for solving the simultaneous sparse approximation problem," *IEEE Transactions on Signal Processing*, vol. 55, no. 7, pp. 3704–3716, Jul 2007.

[4] P. Gerstoft, C. F. Mecklenbräuker, A. Xenaki, and S. Nannuru, "Multisnapshot sparse Bayesian learning for DOA," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1469–1473, 2016.

[5] Z. Zhang and B. D. Rao, "Sparse signal recovery with temporally correlated source vectors using sparse Bayesian learning," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 5, pp. 912–926, 2011.

[6] E. Zöchmann, M. Lerch, S. Caban, R. Langwieser, C.F. Mecklenbräuker, and M. Rupp, "Directional evaluation of receive power, Rician K-factor and RMS delay spread obtained from power measurements of 60 GHz indoor channels," in *Proc. of IEEE-APS Topical Conference on Antennas and Propagation in Wireless Communications (APWC)*. IEEE, 2016, pp. 246–249.

[7] E. Zöchmann, M. Lerch, S. Pratschner, R. Nissel, S. Caban, and M. Rupp, "Associating spatial information to directional millimeter wave channel measurements," in *Proc. of IEEE 86th Vehicular Technology Conference (VTC-Fall)*. IEEE, 2017, pp. 1–5.

[8] E. Zöchmann, K. Guan, and M. Rupp, "Two-ray models in mmWave communications," in *Proc. of IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, July 2017, pp. 1–5.

[9] D. Xu, Z. Liu, X. Qi, Y. Xu, and Y. Zeng, "A FPGA-based implementation of MUSIC for centrosymmetric circular array," in *Proc. of 9th International Conference on Signal Processing (ICSP)*, Oct 2008, pp. 490–493.

[10] K. Huang, J. Sha, W. Shi, and Z. Wang, "An efficient FPGA implementation for 2-D MUSIC algorithm," *Circuits, Systems, and Signal Processing*, vol. 35, no. 5, pp. 1795–1805, May 2016.

[11] A. Suardi, E. C. Kerrigan, and G. A. Constantinides, "Fast FPGA prototyping toolbox for embedded optimization," in *Proc. of Eur. Control Conference (ECC)*. IEEE, 2015, pp. 2589–2594.

[12] M. Iştoan and B. Pasca, "Flexible fixed-point function generation for FPGAs," in *Proc. of IEEE 24th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2017, pp. 123–130.

Authorized licensed use limited to: INSTITUTE OF ACOUSTICS CAS. Downloaded on December 30,2023 at 12:37:06 UTC from IEEE Xplore. Restrictions apply.