A Scalable, Multi-Mode SVD Precoding ASIC Based on the Cyclic Jacobi Method

D. Guenther, R. Leupers, G. Ascheid
{guenther, leupers, ascheid}@ice.rwth-aachen.de

***Abstract*— Modern wireless communication standards define new high throughput use cases like 8x8 multiple-input, multiple-output (MIMO) antenna setups and a 256-QAM constellation alphabet in the case of IEEE 802.11ac. Baseband precoding at the transmitter is a key technique to achieve the corresponding data rates at a reasonable signal-to-noise ratio (SNR). Multi-mode capability, (i.e., the ability to support multiple MIMO setups) is crucial for legacy compatibility or for the adaptation to the individual configurations of mobile stations. This paper presents an application-specific integrated circuit (ASIC) template for singular value decomposition (SVD) based linear precoding supporting multi-mode MIMO. A two-sided cyclic Jacobi algorithm is applied to decompose the SVD computation exclusively into 2x2 vector arithmetic units. A fixed computation pattern is executed iteratively on the input data. Iteration control allows a graceful trading of communication performance for a reduction of computational complexity. As a proof-of-concept, the architecture template is configured to support 2x2, 4x4, 6x6, and 8x8 MIMO and is layouted for 90 nm CMOS with a core area of 1.34 mm^2 and a clock frequency of 752 MHz. The achieved throughput is 188, 15.7, 6.27, and 1.68 million SVDs per second, respectively.**

***Index Terms*—Precoding, MIMO, SVD, multi-mode, ASIC, IEEE 802.11ac**

I. Introduction

Modern wireless communication standards introduce new use cases (e.g., spatial multiplexing with more antennas, higher code rates, denser constellation alphabets) to achieve higher data rates than established standards. IEEE 802.11n [1] wireless LAN, for example, employs spatial multiplexing with up to $M_\text{T} = M_\text{R} = 4$ transmit and receive antennas and a 64-QAM constellation. The recent IEEE 802.11ac standard [2] supports spatial multiplexing for setups up to $M_\text{T} = M_\text{R} = 8$ and a 256-QAM constellation alphabet. To achieve the high data rates promised by IEEE 802.11ac at a reasonable signal-to-noise ratio (SNR), additional processing is required. The computational complexity of multiple input, multiple output (MIMO) detection algorithms like sphere detection rises exponentially with the number of transmit streams and the number of bits per constellation label [3]. Therefore, it is infeasible to burden the potentially battery-powered receiver with higher computational complexity. Instead, part of the computational load should be moved to the transmitter. Linear precoding based on singular value decomposition (SVD) is a viable solution to this problem.

The remainder of Section I introduces the basics of SVD-based linear precoding, gives an overview of existing linear precoding algorithms and architectures, and motivates the need for the more versatile architecture presented in this paper. This work follows a divide-and-conquer approach, both, on an algorithmic and architectural level. The underlying principle is to compute the SVD of bigger size channel matrices based on 2×2 vector arithmetic only, instead of designing a circuit that is specific to one particular MIMO setup. Therefore, Section II starts by introducing a suitable algorithm and an architecture for 2×2 SVD which are then extended to $N\times N$ in Section III. Numerical precision requirements and achievable communication performance are discussed in Section IV. Section V presents a prototype layout and efficiency benchmarks of the architecture when configured to support up to 8×8 MIMO. Section VI concludes this paper.

A. Problem Formulation

A MIMO transmission over a frequency-flat, wireless channel can generally be modeled as

$$\mathbf{y} = \mathbf{H}\mathbf{s} + \mathbf{n} \tag{1}$$

with transmit vector $\mathbf{s} \in \mathbb{C}^{M_\text{T}\times1}$, receive vector $\mathbf{y} \in \mathbb{C}^{M_\text{R}\times1}$, channel matrix $\mathbf{H} \in \mathbb{C}^{M_\text{R}\times M_\text{T}}$ containing the fading coefficients between each transmit and receive antenna pair, and additive noise vector $\mathbf{n} \in \mathbb{C}^{M_\text{R}\times1}$. SVD-based precoding decomposes the channel matrix so that (1) can be rewritten as

$$\mathbf{y} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^H\mathbf{s} + \mathbf{n}. \tag{2}$$

The diagonal matrix $\mathbf{\Lambda} \in \mathbb{R}^{M_\text{R}\times M_\text{T}}$ contains the singular values of $\mathbf{H}$. Matrices $\mathbf{U} \in \mathbb{C}^{M_\text{R}\times M_\text{R}}$ and $\mathbf{V} \in \mathbb{C}^{M_\text{T}\times M_\text{T}}$ are both unitary. Left multiplying (2) by $\mathbf{U}^H$ and substituting $\tilde{\mathbf{s}} = \mathbf{V}^H\mathbf{s}$ and $\tilde{\mathbf{y}} = \mathbf{U}^H\mathbf{y}$ leads to

$$\tilde{\mathbf{y}} = \mathbf{\Lambda}\tilde{\mathbf{s}} + \tilde{\mathbf{n}} \tag{3}$$

with $\tilde{\mathbf{n}} = \mathbf{U}^H\mathbf{n}$. Since $\mathbf{\Lambda}$ in (3) is diagonal, the MIMO transmission from (1) is transformed into $\operatorname{rank}(\mathbf{H})$ parallel single input, single output (SISO) transmissions. The accumulated data rate of these transmissions is maximized by dividing the available transmit power among the SISO transmissions according to the waterfilling algorithm [4]. Therefore, SVD-based precoding first assigns power to $M_\text{S} \leq \operatorname{rank}(\mathbf{H})$ SISO streams comprised by vector $\mathbf{x} \in \mathbb{C}^{M_\text{S}\times1}$ and then multiplies the result by unitary matrix $\mathbf{V}$ so that

$$\mathbf{s} = \mathbf{V}\mathbf{P}\mathbf{x}, \tag{4}$$

where the diagonal power allocation matrix $\mathbf{P} \in \mathbb{R}^{M_\text{T}\times M_\text{S}}$ contains the square roots of the assigned power levels.

B. Related Work

Hardware implementations for SVD have been a subject to very large scale integration (VLSI) research for several decades. Most implementations can be divided into one of two groups, depending on the underlying decomposition algorithm.

1) Jacobi-Based Implementations: In 1960, Forsythe and Henrici described the *cyclic Jacobi method* [5] that computes the SVD of a complex-valued $m \times n$ matrix ($m, n \in \mathbb{N}^{+}$) based on left and right multiplications (therefore called two-sided) of the input matrix by a series of unitary transformation matrices (UTMs). These UTMs are identity matrices except for four scalar elements at positions (p,p), (p,q), (q,p), and (q,q) with $p, q \in \mathbb{N}$. The algorithm iterates over the input matrix in a series of *sweeps*, where in each sweep, all viable p and q are selected in a cyclic fashion. In 1982, Brent and Luk [6] proposed the implementation of the SVD of a real-valued $m \times n$ matrix ($m \geq n$) on a linear multiprocessor array as well as on a two-dimensional array. The underlying algorithm was based on one-sided plane rotations. A particular contribution of [6] was to exchange the cyclic processing scheme of [5] for a new scheme called *parallel ordering*. This scheme partitions the SVD process into $\lceil n/2 \rceil$ independent sub-processes which operate on disjunct sections of the input matrix. The two-dimensional processing architecture mentioned in [6] was developed further in [7], using a two-sided Jacobi algorithm to calculate the SVD of an $n \times n$ matrix. The potential for computing $m \times n$ SVDs by first calculating the QR factorization and then decomposing the $\mathbf{R}$-matrix was also briefly discussed. In 1992, Hemkumar [8] used the parallel ordering scheme for the computation of SVDs of complex-valued, square matrices. As in [7], the associated algorithm is based on the two-sided Jacobi algorithm and it was implemented on a two-dimensional systolic array.

2) Golub and Kahan Based Implementations: In 1965, Golub and Kahan [9] suggested a numerically stabilized algorithm to compute SVDs in a two-step approach. First, the input matrix is transformed to a bidiagonal form (e.g., by a series of Householder transformations [10]). Then, the intermediate matrix is diagonalized, delivering the singular values on the diagonal of the resulting matrix. The architecture proposed in [11] calculates SVDs according to [9] for 4×4 complex-valued matrices based on Givens rotations [12]. Since the QR factorization of a complex-valued matrix can also be calculated using Givens rotations and QR factorization is required for other baseband processing tasks (e.g., preprocessing for sphere detection [3]), the design in [11] can be reconfigured to calculate QR factorizations instead. A version of [11] tailored to SVD only is presented in [13]. Designs [11] and [13] have a relatively small hardware complexity of around 40 kilo gate equivalents (kGE). Throughput requirements of standards like IEEE 802.11n are supposed to be achieved by entity duplication. Authors distance themselves from systolic architectures like [8], [14], claiming that while these types of architectures achieve a high throughput, the penalty in hardware complexity is too high, leading to poor hardware efficiency.

C. Contribution

With the advent of new communication standards, the number of use cases that have to be supported by the precoding hardware increases steadily. While the high computational complexity of SVD calls for an ASIC solution, the approach of designing a decomposition architecture for only one antenna configuration (e.g., 4×4 in [11], [13]) is not viable anymore for a progressive architecture. Instead, this work presents a design that is highly tailored to SVD but at the same time versatile in terms of supported use cases. A related aspect to the support of multiple use cases is *numerically aware processing* [15]. It is intuitive that numerical precision requirements vary depending on the use case. Therefore, our versatile architecture is not only flexible with respect to use cases but also regarding the employed numerical precision.

Authors of [11], [13] discard systolic architectures due to their high hardware complexity. However, the Jacobi-based algorithms in [5], [6], [8] should not be neglected, particularly due to their seamless scalability to different matrix sizes. Therefore, the concepts of [5], [6], [8] are retargeted in this work to a different kind of target architecture. The complex-valued, two-step, two-sided 2×2 SVD is computed by a fully pipelined accelerator. This accelerator is combined with two multiplication engines and two register files to realize the full-size two-sided unitary transformations and the computation of the precoding matrix. Due to its support for numerically aware processing, the architecture is dubbed *napSVD*.

II. 2x2 Algorithm and Architecture

This section discusses the 2×2 SVD algorithm and presents the resulting SVD architecture. Section II-A introduces the CORDIC algorithm, whereafter Section II-B describes how the 2×2 SVD can be composed of CORDIC and vector arithmetic operations as suggested by [8]. The hardware architecture implementing that algorithm is shown in Section II-C.

A. CORDIC Algorithm

The coordinate rotation digital computer (CORDIC) algorithm [16] computes trigonometric functions with limited hardware resources. A two-dimensional vector $\mathbf{v}_0 = [x_0, y_0]^T$ is rotated by an angle $\varPhi$, resulting in a vector $\mathbf{v}_L = [x_L, y_L]^T$. The rotation by angle $\varPhi$ can be expressed as a series of L micro-rotations by angles $\alpha_i, i \in \{0, \ldots, L-1\}$ so that

$$\varPhi = \sum_{i=0}^{L-1} \sigma_i \alpha_i, \quad \sigma_i \in \{+1, -1\}. \tag{5}$$

The bipolar variable σ_i controls the direction of the i-th micro-rotation. The values α_i are chosen so that

$$\begin{aligned} \tan \alpha_i &= 2^{-i} \Leftrightarrow \alpha_i = \arctan\left(2^{-i}\right) && (6) \\ \begin{pmatrix} x_L \\ y_L \end{pmatrix} &= \kappa \prod_{i=0}^{L-1} \begin{pmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} && (7) \\ \kappa &= \prod_{i=0}^{L-1} \cos \alpha_i. && (8) \end{aligned}$$

The correction factor κ has to be applied once after the last iteration. The iteration for each scalar element is given by

$$\begin{aligned} x_{i+1} &= x_i - \sigma_i\, 2^{-i} y_i \\ y_{i+1} &= y_i + \sigma_i\, 2^{-i} x_i \\ z_{i+1} &= z_i - \sigma_i \arctan 2^{-i}, \end{aligned} \tag{9}$$

where z_i corresponds to the rotation angle including an initial offset z_0. The choice of z_0 and the computation scheme for σ_i stipulate the CORDIC mode. The two modes relevant for this work are *vectoring* and *rotation*.

1) Vectoring: The input vector is rotated so that the y-component converges to zero. Due to the limited range of the CORDIC rotation, a preprocessing step that rotates input vector $\tilde{\mathbf{v}}_0 = [\tilde{x}_0, \tilde{y}_0]^T$ and phase $\tilde{z}_0$ into the first quadrant of the Cartesian coordinate system is required. The output of preprocessing, $\mathbf{v}_0$ and z_0, then serves as input to (9).

$$(x_0,\, y_0,\, z_0) = \begin{cases} (+\tilde{x}_0,\, +\tilde{y}_0,\, 0) & \tilde{x}_0 \geq 0,\, \tilde{y}_0 \geq 0 \\ (+\tilde{y}_0,\, -\tilde{x}_0,\, \pi/2) & \tilde{x}_0 < 0,\, \tilde{y}_0 \geq 0 \\ (-\tilde{x}_0,\, -\tilde{y}_0,\, \pi) & \tilde{x}_0 < 0,\, \tilde{y}_0 < 0 \\ (-\tilde{y}_0,\, +\tilde{x}_0,\, 3\pi/2) & \tilde{x}_0 \geq 0,\, \tilde{y}_0 < 0 \end{cases} \tag{10}$$

The micro-rotation direction is chosen according to

$$\sigma_i = -\operatorname{sign}(y_i). \tag{11}$$

2) Rotation: The input vector is rotated by a specific angle Φ. If the requested rotation angle exceeds the maximum angular range of the CORDIC algorithm, an additional preprocessing step is required to obtain a modified starting vector in the first quadrant of the Cartesian coordinate system. Two-dimensional vector $\mathbf{v}$ and phase z_0 as required by (9) are obtained from the arbitrary inputs $\tilde{\mathbf{v}}$ and $\tilde{z}_0 = \Phi$ according to

$$(x_0,\, y_0,\, z_0) = \begin{cases} (+\tilde{x}_0,\, +\tilde{y}_0,\, \tilde{z}_0) & 0 \leq \tilde{z}_0 < \pi/2 \\ (+\tilde{y}_0,\, -\tilde{x}_0,\, \tilde{z}_0 - \pi/2) & \pi/2 \leq \tilde{z}_0 < \pi \\ (-\tilde{x}_0,\, -\tilde{y}_0,\, \tilde{z}_0 - \pi) & \pi \leq \tilde{z}_0 < 3\pi/2 \\ (-\tilde{y}_0,\, +\tilde{x}_0,\, \tilde{z}_0 - 3\pi/2) & 3\pi/2 \leq \tilde{z}_0 < 2\pi \end{cases} \tag{12}$$

for micro-rotation directions

$$\sigma_i = \operatorname{sign}(z_i). \tag{13}$$

B. SVD Algorithm

A two-sided unitary transformation of a matrix $\mathbf{M} \in \mathbb{C}^{2\times 2}$ can generally be expressed as

$$\mathbf{V}_\mathrm{l}(\Phi, \theta_\alpha, \theta_\beta)\, \mathbf{M}\, \mathbf{V}_\mathrm{r}(\Psi, \theta_\gamma, \theta_\delta) = \begin{pmatrix} c_\Phi \mathrm{e}^{i\theta_\alpha} & -s_\Phi \mathrm{e}^{i\theta_\beta} \\ s_\Phi \mathrm{e}^{i\theta_\alpha} & c_\Phi \mathrm{e}^{i\theta_\beta} \end{pmatrix} \mathbf{M} \begin{pmatrix} c_\Psi \mathrm{e}^{i\theta_\gamma} & s_\Psi \mathrm{e}^{i\theta_\gamma} \\ -s_\Psi \mathrm{e}^{i\theta_\delta} & c_\Psi \mathrm{e}^{i\theta_\delta} \end{pmatrix} \tag{14}$$

with unitary matrices $\mathbf{V}_\mathrm{l}, \mathbf{V}_\mathrm{r} \in \mathbb{C}^{2\times 2}$. Variables c_Φ and s_Φ denote the cosine and sine of angle Φ, and c_Ψ and s_Ψ denote the cosine and sine of angle Ψ. The remaining transformation parameters are given by $\theta_\alpha, \theta_\beta, \theta_\gamma, \theta_\delta \in [0 \ldots 2\pi[$. The algorithm in [8] calculates the SVD of a 2×2 matrix by two unitary transformations based on (14). The first transformation generates an upper triangular matrix

$$\tilde{\mathbf{M}} = \mathbf{V}_{\mathrm{l}_1} \mathbf{M} \mathbf{V}_{\mathrm{r}_1} = \mathbf{V}_\mathrm{l}(\Phi_1, \theta_{\alpha_1}, \theta_{\beta_1})\, \mathbf{M}\, \mathbf{V}_\mathrm{r}(\Psi_1, \theta_{\gamma_1}, \theta_{\delta_1}) \tag{15}$$

with Φ_1, θ_{α_1}, θ_{β_1}, Ψ_1, θ_{γ_1}, θ_{δ_1} given by [8]

$$\begin{aligned} \Phi_1 &= 0 \\ \theta_{\alpha_1} = \theta_{\beta_1} &= -\frac{\theta_{m_{22}} + \theta_{m_{21}}}{2} \\ \Psi_1 &= \tan^{-1}\left(\frac{|M_{21}|}{|M_{22}|}\right) \\ \theta_{\gamma_1} = -\theta_{\delta_1} &= \frac{\theta_{m_{22}} - \theta_{m_{21}}}{2}. \end{aligned} \tag{16}$$

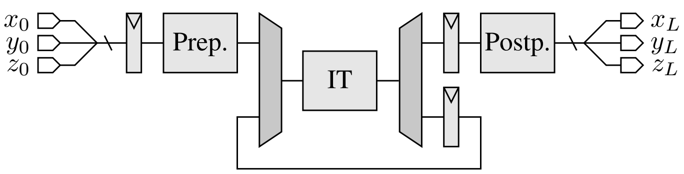


Fig. 1: Schematic of CORDIC architecture template.

Variable $\theta_{m_{ij}}$ denotes the azimuth of scalar element M_{ij} of matrix $\mathbf{M}$. The second transformation delivers a real-valued diagonal matrix

$$\bar{\mathbf{M}} = \mathbf{V}_{\mathrm{l}_2} \tilde{\mathbf{M}} \mathbf{V}_{\mathrm{r}_2} = \mathbf{V}_\mathrm{l}(\Phi_2, \theta_{\alpha_2}, \theta_{\beta_2})\, \tilde{\mathbf{M}}\, \mathbf{V}_\mathrm{r}(\Psi_2, \theta_{\gamma_2}, \theta_{\delta_2}) \tag{17}$$

that contains the singular values of $\mathbf{M}$. The corresponding transformation parameters Φ_2, θ_{α_2}, θ_{β_2}, Ψ_2, θ_{γ_2}, θ_{δ_2} based on $\tilde{\mathbf{M}}$ are given by [8]

$$\begin{aligned} \theta_{\alpha_2} &= -\frac{\theta_{\tilde{m}_{12}} + \theta_{\tilde{m}_{11}}}{2} \\ \theta_{\beta_2} = \theta_{\gamma_2} = -\theta_{\delta_2} &= \frac{\theta_{\tilde{m}_{12}} - \theta_{\tilde{m}_{11}}}{2} \\ \tan(\Phi_2 \pm \Psi_2) &= -\left(\frac{|\tilde{M}_{12}|}{|\tilde{M}_{22}| \mp |\tilde{M}_{11}|}\right). \end{aligned} \tag{18}$$

Note that the complex-valued phase terms as well as the cosine and sine in (14) can be generated by the CORDIC algorithm in rotation mode. The polar representation of $\mathbf{M}$ and $\tilde{\mathbf{M}}$ as well as the trigonometric functions in (16) and (18) can be derived using a CORDIC in vectoring mode. The diagonal matrix containing the singular values is given by

$$\mathbf{\Lambda} = \mathbf{V}_{\mathrm{l}_2} \mathbf{V}_{\mathrm{l}_1}\, \mathbf{M}\, \mathbf{V}_{\mathrm{r}_1} \mathbf{V}_{\mathrm{r}_2}. \tag{19}$$

C. Architecture

Section II-C1 describes the architecture of the CORDIC unit which is used to realize a 2×2 UTM generator (Section II-C2). Four of these generators are required to implement the two-step, two-sided unitary transformation that generates a 2×2 SVD (Section II-C4).

1) CORDIC: The CORDIC architecture template is depicted in Figure 1. It is designed to perform one CORDIC operation (i.e., vectoring or rotation) in multiple cycles, reusing the same hardware. Therefore, the elements within the template are controlled by a finite state machine (FSM) and adapted to the current processing cycle. The preprocessing block (Prep.) implements the rotation of the input vector to the first quadrant of the Cartesian coordinate system (see (10), (12)). The preprocessed input is then forwarded within the same clock cycle to the iterator block (IT). The iterator block consists of a chain of CORDIC micro-rotation units as in (9). The output of the chain can be fed back to the input multiple times (e.g., depending on numerical precision requirements). After finishing the iteration phase, the result is routed to the postprocessing block (Postp.) in the subsequent clock cycle. The postprocessing block multiplies the scalar elements of the post-iteration vector by correction factor κ according to (8).

The architecture template supports runtime-adaptable parameters to tune the numerical precision of the CORDIC operation to the numerical precision requirements of the

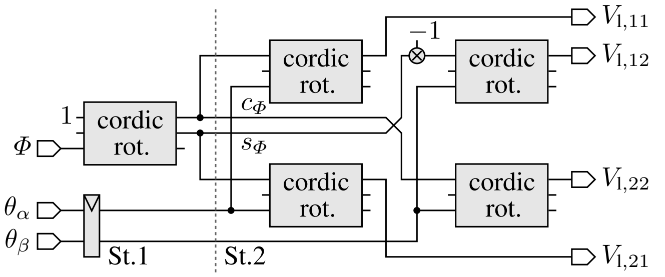


Fig. 2: Schematic of left-hand side 2×2 unitary transformation matrix (UTM) generator.

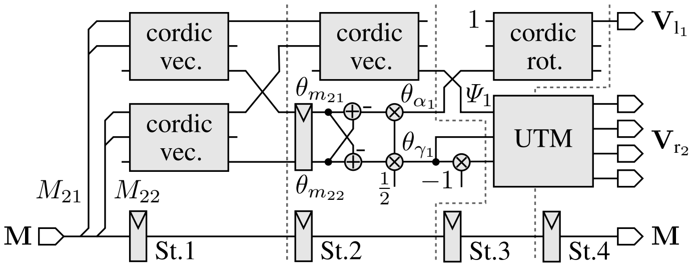


Fig. 3: Schematic of Q1 transformation unit.

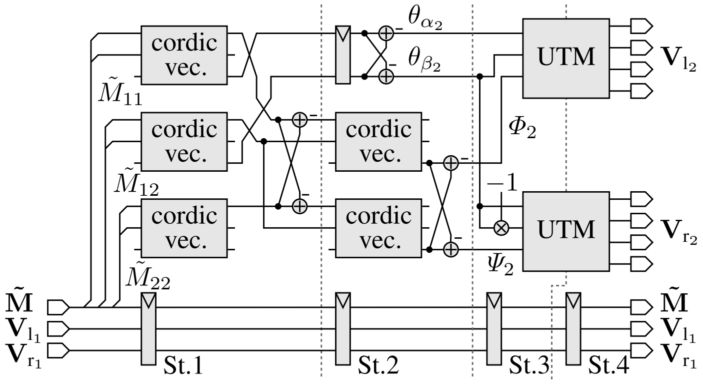


Fig. 4: Schematic of Q2 transformation unit.

surrounding application. By doing so, switching activity in the circuit is minimized, which reduces the energy consumption per CORDIC operation.

- **Adapting CORDIC iteration cycles** configures the number of micro-rotations on the granularity level of iterators in the IT block.
- **Adapting the iterator chain length** by bypassing a configurable number of iterators adapts the number of micro-rotations on the granularity level of a single rotation.
- **A configurable bitmask**, applied to a certain number of least significant bits (LSBs) prior to each micro-rotation and the postprocessing unit, implements an adaptive number format.

2) 2x2 UTM Generation: The schematic of the unitary transformation matrix generator for left-hand side matrix $\mathbf{V}_\mathrm{l}(\Phi,\theta_\alpha,\theta_\beta)$ is shown in Figure 2. Inputs and outputs of CORDIC units are ordered according to Figure 1. For all following schematics, unlabeled inputs are supposed to be zeroed. The UTM generation circuit is divided into two coarse-grained *computational stages* (St.). Each computational stage performs its respective task in a maximum of C_S clock cycles before passing the result to the next state. The time corresponding to C_S clock cycles is referred to as a *computational cycle*. The computation of the left-hand and right-hand unitary matrices $\mathbf{V}_\mathrm{l}(\Phi,\theta_\alpha,\theta_\beta)$ and $\mathbf{V}_\mathrm{r}(\Psi,\theta_\gamma,\theta_\delta)$ in (14) requires the sines and cosines of Φ and Ψ. These trigonometric functions are generated in the first computational stage by a CORDIC unit in rotation (rot.) mode, passing 1 and 0 to x_0 and y_0, respectively, and Φ or Ψ to z_0. At the output of the CORDIC unit, the cosine and sine are available at ports x_L and y_L. These values are then passed to the second computational stage and assigned to the real-valued components of the complex-valued inputs of four CORDIC units in rotation mode, while the imaginary components are set to zero. Forwarding θ_α and θ_β (for $\mathbf{V}_\mathrm{l}(\Phi,\theta_\alpha,\theta_\beta)$) or θ_γ and θ_δ (for $\mathbf{V}_\mathrm{r}(\Psi,\theta_\gamma,\theta_\delta)$) to the corresponding z_0 ports generates the desired transformation matrices.

3) Transformations Q1 and Q2: The 2×2 SVD of input matrix $\mathbf{M}$ is realized by two transformations: Q1 and Q2, whereof each requires two UTM generators.

The schematic of Q1 is illustrated in Figure 3. The circuit is spread across four computational stages. The first two calculate the transformation parameters from (16) and the last two generate the UTMs. The first stage calculates amplitudes and phases of scalar components M_{21} and M_{22} of input matrix $\mathbf{M}$, utilizing two CORDIC units in vectoring mode (vec.). From these results, the second stage computes Ψ_1 based on the amplitudes of M_{21} and M_{22}. Phases θ_{α_1} and θ_{γ_1} are calculated using $\theta_{m_{21}}$ and $\theta_{m_{22}}$ according to (16). In the next two stages, the 2×2 transformation matrices of Q1 are computed. Due to $\Phi_1 = 0$, the non-diagonal parts of $\mathbf{V}_\mathrm{l}(\Phi_1,\theta_{\alpha_1},\theta_{\beta_1})$ are zero as well. Furthermore, $\theta_{\alpha_1} = \theta_{\beta_1}$ means that the two diagonal elements are equal. Thus, the computation of $\mathbf{V}_\mathrm{l}(\Phi_1,\theta_{\alpha_1},\theta_{\beta_1})$ can be reduced to a single CORDIC operation in rotation mode. As a result, the left transformation matrix is already available in the third computational stage, while the computation of the right matrix is finished in stage four. In addition, the input matrix is stored in a first-in, first-out (FIFO) buffer that samples its input every C_S-th clock cycle. The delayed value of $\mathbf{M}$ is used to calculate $\tilde{\mathbf{M}}$ as in (15).

The structure of the Q2 block is shown in Figure 4. The parameters for UTM generation according to (18) are calculated in the first two computational stages. The last two stages generate the transformation matrices themselves. The first stage computes amplitudes and phases of the scalar elements $\tilde{M}_{11}$, $\tilde{M}_{12}$, and $\tilde{M}_{22}$ of matrix $\tilde{\mathbf{M}}$. Since the operand to the tangent in (18) is a fraction of amplitude terms, there is no need for CORDIC postprocessing in the preceding CORDIC unit, because the correction factor κ cancels out. As a result, sums $|\tilde{M}_{22}| \mp |\tilde{M}_{11}|$ can be calculated in the last cycle of the first computational stage. The second stage computes θ_{α_2}, θ_{β_2}, θ_{γ_2}, and θ_{δ_2}, as well as the phases Φ_2 and Ψ_2. For the latter two, the design exploits the fact that the phase output of the CORDIC operation does not require postprocessing, so Φ_2 and Ψ_2 are computed in the last cycle of the second computational stage. Stages three and four take the previously calculated parameters as inputs and generate the left and right-hand UTMs $\mathbf{V}_\mathrm{l}(\Phi_2,\theta_{\alpha_2},\theta_{\beta_2})$ and $\mathbf{V}_\mathrm{r}(\Psi_2,\theta_{\gamma_2},\theta_{\delta_2})$. Q2 also buffers the transformation matrices generated by Q1 in a FIFO, so they can be combined with the results of Q2 to derive the

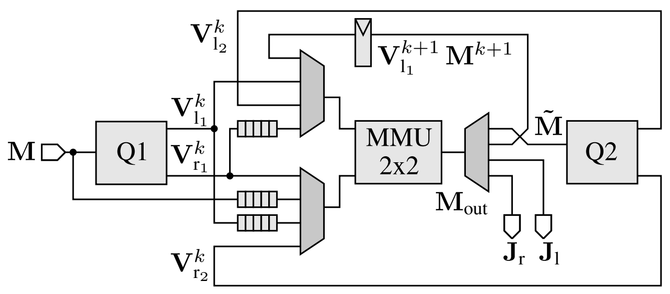

$\mathbf{V}^k_{\mathrm{l}_2}$
$\mathbf{V}^{k+1}_{\mathrm{l}_1}\,\mathbf{M}^{k+1}$
$\mathbf{V}^k_{\mathrm{l}_1}$
$\mathbf{V}^k_{\mathrm{r}_1}$
$\mathbf{M}$
Q1
MMU 2x2
$\tilde{\mathbf{M}}$
Q2
$\mathbf{M}_{\text{out}}$
$\mathbf{J}_\mathrm{r}$
$\mathbf{J}_\mathrm{l}$
$\mathbf{V}^k_{\mathrm{r}_2}$

Fig. 5: Schematic of 2×2 SVD generator.

overall transformation matrices.

4) 2x2 SVD: The architecture of the 2×2 SVD block is illustrated in Figure 5. The 2×2 SVD generator contains one instance of Q1 and Q2, and a 2×2 matrix multiplication unit (MMU) to compute $\tilde{\mathbf{M}}$ from (15) and the final left and right-hand unitary matrices

$$\begin{aligned}\mathbf{J}_\mathrm{l}(\mathbf{M}) &= \mathbf{V}_{\mathrm{l}_2}\mathbf{V}_{\mathrm{l}_1} \\ \mathbf{J}_\mathrm{r}(\mathbf{M}) &= \mathbf{V}_{\mathrm{r}_1}\mathbf{V}_{\mathrm{r}_2},\end{aligned} \tag{20}$$

meaning the multiplication unit has to perform a total of four matrix multiplications per 2×2 SVD. Allocating one cycle per matrix multiplication motivates the choice of $C_\mathrm{S} = 4$, so that the 2×2 SVD generator computes one SVD every four clock cycles. To achieve a competitive clock frequency, the matrix multiplier has an additional pipeline register between the real-valued scalar multipliers and the subsequent adders. Whilst increasing the clock frequency, this pipelining introduces a potential data hazard. The MMU has a budget of $C_\mathrm{S} = 4$ clock cycles to compute matrix $\tilde{\mathbf{M}}$ which is required as input to Q2, and for output matrices $\mathbf{J}_\mathrm{l}$ and $\mathbf{J}_\mathrm{r}$ (see (20)). Since the computation of one matrix multiplication has a latency of two clock cycles, the calculation of $\tilde{\mathbf{M}}$ alone introduces a latency of four cycles. However, $\mathbf{V}_{\mathrm{l}_2}$ and $\mathbf{V}_{\mathrm{r}_2}$ depend on $\tilde{\mathbf{M}}$, so all calculations in the MMU that use these two matrices have to be scheduled after the calculation of $\tilde{\mathbf{M}}$ but within the same computational cycle. With a latency of four cycles for the computation of $\tilde{\mathbf{M}}$, this constraint cannot be fulfilled. To overcome this issue, the design exploits the fact that $\mathbf{V}_{\mathrm{l}_1}$ is available C_S clock cycles prior to $\mathbf{V}_{\mathrm{r}_1}$. Consequently, the intermediate result $\mathbf{V}_{\mathrm{l}_1}\mathbf{M}$ is computed one computational cycle prior to the computation of $\tilde{\mathbf{M}}$ and stored in a clock-gated register until it is used in the next computational cycle. The assignments of multiplications to clock cycles i within computational cycle k is given by

$$\mathbf{M}_{\text{out}} = \begin{cases} \left(\mathbf{V}^k_{\mathrm{l}_1}\mathbf{M}^k\right)\mathbf{V}^k_{\mathrm{r}_1} & i = 1 \\ \mathbf{V}^{k+1}_{\mathrm{l}_1}\mathbf{M}^{k+1} & i = 2 \\ \mathbf{V}^k_{\mathrm{l}_2}\,\mathbf{V}^k_{\mathrm{l}_1} & i = 3 \\ \mathbf{V}^k_{\mathrm{r}_1}\,\mathbf{V}^k_{\mathrm{r}_2} & i = 4\,. \end{cases} \tag{21}$$

III. $N \times N$ SVD Algorithm and Architecture

The Jacobi method allows the computation of the SVD of an $N \times N$ matrix based on two-sided unitary transformations. Section III-A introduces the underlying algorithm from [5] which is then transferred into a representation entirely based on 2×2 arithmetic. Section III-B describes how the previously introduced 2×2 SVD architecture can be embedded into a bigger circuit where it is used to implement $N \times N$ SVDs. Control flow and address generation of the $N \times N$ architecture are designed to guarantee full utilization of the 2×2 SVD unit.

Algorithm 1: Cyclic Jacobi algorithm for SVD of $N \times N$ matrix [5].

```
1  $\mathbf{\Lambda} \leftarrow \mathbf{M}$
2  $\mathbf{V} \leftarrow \mathbf{I}_N$
3  for $s \leftarrow 1$ to $N_{SW}$ do
4      for $p \leftarrow 1$ to $N-1$ step $2$ do
5          for $q \leftarrow p+1$ to $N$ do
6              $\mathbf{\Lambda} \leftarrow \mathbf{J}^\mathrm{a}_\mathrm{l}(\mathbf{\Lambda}^{(p,q)})\,\mathbf{\Lambda}\,\mathbf{J}^\mathrm{a}_\mathrm{r}(\mathbf{\Lambda}^{(p,q)})$
7              $\mathbf{V} \leftarrow \mathbf{V}\,\mathbf{J}^\mathrm{a}_\mathrm{r}(\mathbf{\Lambda}^{(p,q)})$
8          end
9      end
10 end
```

A. Algorithm

The cyclic Jacobi method [5] for SVD of matrix $\mathbf{M} \in \mathbb{C}^{N \times N}$ in its original form is shown in Algorithm 1, where $\mathbf{I}_N$ denotes an $N \times N$ identity matrix. The singular values of $\mathbf{M}$ are computed in an iterative fashion based on 2×2 SVDs and $N \times N$ matrix-matrix multiplications. To that end, the input matrix is multiplied by a series of augmented left and right-hand transformation matrices $\mathbf{J}^\mathrm{a}_\mathrm{l}$ and $\mathbf{J}^\mathrm{a}_\mathrm{r}$. These matrices are identity except at four positions: $(p,p),(p,q),(q,p),(q,q)$ with $p < q \leq N$ and $p \in \{2n+1, n \in \mathbb{N}\}$. At these four positions, $\mathbf{J}^\mathrm{a}_\mathrm{l}$ and $\mathbf{J}^\mathrm{a}_\mathrm{r}$ are defined by $\mathbf{J}_\mathrm{l}(\mathbf{\Lambda}^{(p,q)})$ and $\mathbf{J}_\mathrm{r}(\mathbf{\Lambda}^{(p,q)})$. Here, $\mathbf{\Lambda}^{(p,q)} \in \mathbb{C}^{2\times 2}$ denotes a matrix composed of four elements of $\mathbf{\Lambda}$ at the aforementioned positions. The scalar elements for row i and column j of $\mathbf{J}^\mathrm{a}_\mathrm{l}$ and $\mathbf{J}^\mathrm{a}_\mathrm{r}$ are given by

$$J^\mathrm{a}_{\mathrm{l},ij}\left(\mathbf{\Lambda}^{(p,q)}\right) = \begin{cases} 1 & i{=}j \wedge i{\neq}p \wedge j{\neq}q \\ J_{\mathrm{l},11}(\mathbf{\Lambda}^{(p,q)}) & i{=}p \wedge j{=}p \\ J_{\mathrm{l},12}(\mathbf{\Lambda}^{(p,q)}) & i{=}p \wedge j{=}q \\ J_{\mathrm{l},21}(\mathbf{\Lambda}^{(p,q)}) & i{=}q \wedge j{=}p \\ J_{\mathrm{l},22}(\mathbf{\Lambda}^{(p,q)}) & i{=}q \wedge j{=}q \\ 0 & \text{else} \end{cases} \tag{22}$$

$$J^\mathrm{a}_{\mathrm{r},ij}\left(\mathbf{\Lambda}^{(p,q)}\right) = \begin{cases} 1 & i{=}j \wedge i{\neq}p \wedge j{\neq}q \\ J_{\mathrm{r},11}(\mathbf{\Lambda}^{(p,q)}) & i{=}p \wedge j{=}p \\ J_{\mathrm{r},12}(\mathbf{\Lambda}^{(p,q)}) & i{=}p \wedge j{=}q \\ J_{\mathrm{r},21}(\mathbf{\Lambda}^{(p,q)}) & i{=}q \wedge j{=}p \\ J_{\mathrm{r},22}(\mathbf{\Lambda}^{(p,q)}) & i{=}q \wedge j{=}q \\ 0 & \text{else} \end{cases} \tag{23}$$

For each iteration s, which is also referred to as a *sweep*, Algorithm 1 iterates over all pairs (p,q) to update matrices $\mathbf{\Lambda}$ and $\mathbf{V}$. After a sufficient number of sweeps N_SW, matrix $\mathbf{\Lambda}$ contains the singular values of $\mathbf{M}$ on its diagonal and $\mathbf{V}$ converges to the precoding matrix from (4).

Algorithm 1 can be optimized by exploiting the fact that a sweep over all pairs (p,q) contains $N-1$ groups of $N/2$ pairs that can be processed in parallel without interfering with each other [6]. The corresponding rearrangement of pairs (p,q)

is called *parallel ordering*. The pairs can be generated in a hardware-friendly fashion by a register bank and a fixed permutation mesh between the inputs and outputs of the bank. The register contents are initialized with increasing integers from 1 to N. For each following clock cycle, the circuit generates a new set of independent pairs. Figure 6 shows the generation of all pairs for an 8×8 SVD. The preliminary values $(\tilde{p}, \tilde{q})$ from the register bank have to be postprocessed according to

$$(p, q) = \begin{cases} (\tilde{p}, \tilde{q}) & \tilde{p} < \tilde{q} \\ (\tilde{q}, \tilde{p}) & \tilde{q} < \tilde{p}. \end{cases} \tag{24}$$

Since the parallel ordering pairs are disjunct, Algorithm 1 can be rewritten so the left and right-hand multiplication matrices from Line 6 are not based on just one pair from the current parallel ordering permutation but on all $N/2$ of them. This step facilitates a high throughput hardware implementation by enabling the computation of more transformations on the input matrix without having to wait for the results of previous transformations which first have to pass through the entire processing pipeline. From the disjunct pairs $(p_v, q_v), v \in \{1, \dots, N/2\}$ of the current parallel ordering permutation, the combined left-hand transformation matrix $\mathbf{J}_\mathrm{l}^\mathrm{c} \in \mathbb{C}^{N \times N}$ is given by

$$J^\mathrm{c}_{\mathrm{l},ij} = \begin{cases} J_{\mathrm{l},11}(\mathbf{\Lambda}^{(p_v,q_v)}) & i{=}p_v \wedge j{=}p_v \\ J_{\mathrm{l},12}(\mathbf{\Lambda}^{(p_v,q_v)}) & i{=}p_v \wedge j{=}q_v \\ J_{\mathrm{l},21}(\mathbf{\Lambda}^{(p_v,q_v)}) & i{=}q_v \wedge j{=}p_v \\ J_{\mathrm{l},22}(\mathbf{\Lambda}^{(p_v,q_v)}) & i{=}q_v \wedge j{=}q_v \\ 0 & \text{else} \end{cases} \tag{25}$$

for all v. Similarly, based on all disjunct pairs $(p_u, q_u), u \in \{1, \dots, N/2\}$, the combined right-hand matrix $\mathbf{J}_\mathrm{r}^\mathrm{c} \in \mathbb{C}^{N \times N}$ is

$$J^\mathrm{c}_{\mathrm{r},ij} = \begin{cases} J_{\mathrm{r},11}(\mathbf{\Lambda}^{(p_u,q_u)}) & i{=}p_u \wedge j{=}p_u \\ J_{\mathrm{r},12}(\mathbf{\Lambda}^{(p_u,q_u)}) & i{=}p_u \wedge j{=}q_u \\ J_{\mathrm{r},21}(\mathbf{\Lambda}^{(p_u,q_u)}) & i{=}q_u \wedge j{=}p_u \\ J_{\mathrm{r},22}(\mathbf{\Lambda}^{(p_u,q_u)}) & i{=}q_u \wedge j{=}q_u \\ 0 & \text{else} \end{cases} \tag{26}$$

for all u. The resulting matrices have exactly two non-zero entries in each row. Also, pairs of two rows in each matrix have their non-zero entries in the same columns. This means that the right-multiplication of matrix $\mathbf{\Lambda}$ by $\mathbf{J}_\mathrm{r}^\mathrm{c}$ corresponds to $N/2$ independent multiplications of submatrices of $\mathbf{\Lambda}$ by the 2×2 matrices $\mathbf{J}_\mathrm{r}(\mathbf{\Lambda}^{(p_u,q_u)})$ for all u. The submatrices of $\mathbf{\Lambda}$ are constructed from two columns indicated by (p_u, q_u). Similarly, the left multiplication of $\mathbf{\Lambda}$ by $\mathbf{J}_\mathrm{l}^\mathrm{c}$ corresponds to $N/2$ independent multiplications of the 2×2 matrices $\mathbf{J}_\mathrm{l}(\mathbf{\Lambda}^{(p_v,q_v)})$ by two rows of $\mathbf{\Lambda}$ indicated by (p_v, q_v) for all v. Therefore, the multiplication in Line 6 of Algorithm 1 can be rewritten based on 2×2 arithmetic. This is particularly attractive when designing a flexible circuit for $N \times N$ SVD, since the core arithmetic functionality remains the same and only the surrounding control flow changes. The resulting processing scheme based on parallel ordering is illustrated in Algorithm 2, where n iterates over all parallel ordering permutations, and v and u iterate over all pairs of the current permutation for the left and right-hand side factors $\mathbf{J}_\mathrm{l}$ and $\mathbf{J}_\mathrm{r}$. Access to matrix $\mathbf{V}$ is described by $\mathbf{V}^{(v,p,q)}$ which denotes a 2×2 submatrix from consecutive rows $2v - 1$ and $2v$ at colums p and q of $\mathbf{V}$.

Algorithm 2: Jacobi algorithm with parallel ordering for SVD of $N \times N$ matrix.

```
1  $\mathbf{\Lambda} \leftarrow \mathbf{M}$
2  $\mathbf{V} \leftarrow \mathbf{I}_N$
3  for $s \leftarrow 1$ to $N_{SW}$ do
4      for $n \leftarrow 1$ to $N-1$ do
5          Compute next set of pairs $(p_1, q_1), .., (p_{N/2}, q_{N/2})$
6          for $u \leftarrow 1$ to $N/2$ do
7              for $v \leftarrow 1$ to $N/2$ do
8                  $\mathbf{\Lambda}^{(p_v,q_u)} \leftarrow \mathbf{J}_\mathrm{l}(\mathbf{\Lambda}^{(p_v,q_v)})\mathbf{\Lambda}^{(p_v,q_u)}\mathbf{J}_\mathrm{r}(\mathbf{\Lambda}^{(p_u,q_u)})$
9                  $\mathbf{V}^{(v,p_u,q_u)} \leftarrow \mathbf{V}^{(v,p_u,q_u)}\mathbf{J}_\mathrm{r}(\mathbf{\Lambda}^{(p_u,q_u)})$
10             end
11         end
12     end
13 end
```

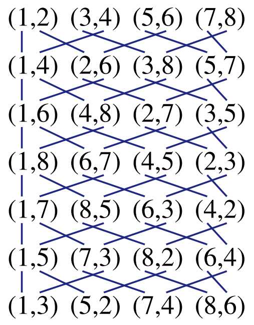


Fig. 6: Generation of parallel ordering pairs $(\tilde{p}, \tilde{q})$ for 8×8 SVD (see (24)).

B. Architecture

The structure of the $N \times N$ SVD architecture implementing Algorithm 2 is shown in Figure 7. In the following, the main components are introduced. The IO register file (IO reg) acts as a local cache. It is accessible from the outside to write the input matrices $\mathbf{M}$ and read out the result, once $\mathbf{M}$ has been iteratively transformed to $\mathbf{\Lambda}$. To the inside, it provides 2×2 submatrices of the current version of $\mathbf{\Lambda}$ as input to the 2×2 SVD block to compute $\mathbf{J}_\mathrm{l}(\mathbf{\Lambda}^{(p_v,q_v)})$ and $\mathbf{J}_\mathrm{r}(\mathbf{\Lambda}^{(p_u,q_u)})$. It also provides $\mathbf{\Lambda}^{(p_v,q_u)}$ to the $\mathbf{\Lambda}$ multiplication engine ($\mathbf{\Lambda}$ mul). The result produced by the multiplication engine (i.e., the left side of Algorithm 2, Line 8) is fed back to the IO register file to update the $\mathbf{\Lambda}$-matrix. After all sweeps have been processed, the register file contains the final matrix $\mathbf{\Lambda}$. To match the access pattern of Algorithm 2, the register file accepts four scalar inputs and delivers four scalar outputs at each clock cycle. The storage elements are organized in a two-dimensional grid, whereof two rows can be addressed simultaneously via two row indices. Out of these two rows, two columns are selected by two column indices. The resulting multiplexing and demultiplexing complexity is significant enough to prolong the critical path. Therefore, there is an additional pipeline register in front of the demultiplexing network from the input of the register file to the actual storage cells. Also, all computational elements connected to the output

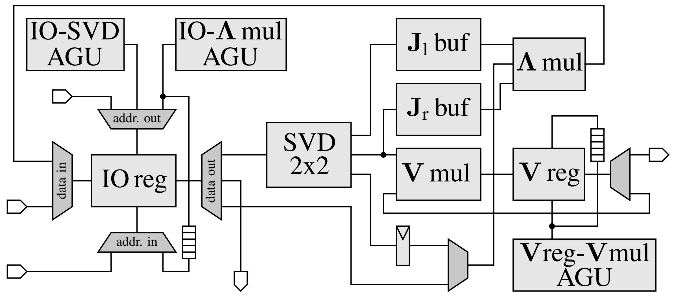


Fig. 7: Schematic of $N \times N$ SVD generator.

of the IO register file have their inputs directly connected to a pipeline register so there is a sufficient time budget for the multiplexing network at the output of the IO register file. Additional pipeline registers are inserted at the same positions in the register file holding matrix $\mathbf{V}$.

The access pattern described by the two for-loops in Line 6 and 7 of Algorithm 2 can only be realized if all $N/2$ matrices $\mathbf{J}_\mathrm{l}$ and $\mathbf{J}_\mathrm{r}$ of the current parallel ordering permutation have been computed. However, the 2×2 SVD block computes these matrices successively. Therefore, two buffers ($\mathbf{J}_\mathrm{l}$ buf, $\mathbf{J}_\mathrm{r}$ buf) are placed in front of the $\mathbf{\Lambda}$ multiplication engine. They delay the computation of Line 8 until the 2×2 SVD block has delivered enough matrices $\mathbf{J}_\mathrm{l}$ and $\mathbf{J}_\mathrm{r}$ of the current parallel ordering permutation, so that once the buffers are filled, the $\mathbf{\Lambda}$ multiplication engine can run at the highest possible utilization. The third input to the multiplication engine is matrix $\mathbf{\Lambda}^{(p_v,q_u)}$, the middle-factor from Algorithm 2, Line 8. In case of $v = u$, this input comes from the 2×2 SVD engine through which it passed via a FIFO. For all remaining clock cycles of the current computational cycle, the IO register file can supply data to the $\mathbf{\Lambda}$ multiplication engine directly.

The $\mathbf{V}$ multiplication engine ($\mathbf{V}$ mul) generates the $N \times N$ precoding matrix $\mathbf{V}$ from the 2×2 output matrices $\mathbf{J}_\mathrm{r}$ of the 2×2 SVD. The computation follows Line 9 of Algorithm 2. The intermediate version of the precoding matrix is stored in the $\mathbf{V}$ register file ($\mathbf{V}$ reg). Since Algorithm 2 operates on consecutive rows of $\mathbf{V}$, the register file can be simplified as opposed to the $\mathbf{\Lambda}$ register file. While still providing random access to two columns, the row-access is now controlled by a single address so that the content of one row in the register file matches two consecutive rows of matrix $\mathbf{V}$.

1) Control Flow & Address Generation: The control flow of the $N \times N$ SVD block is mainly defined by the way the design steps through the loops in Algorithm 2 and the corresponding accesses patterns to the IO register file. In addition to the flow presented in Algorithm 2, the hardware implementation also interleaves the processing of a certain number of matrices to enable full utilization of the 2×2 SVD pipeline. The overall access pattern and the corresponding address generation is composed of a number of nested control loops that are introduced in the following from the inside to the outside of the loop hierarchy of Algorithm 2.

- **Serial access control** refers to the serialization of parallel ordering permutations so data can be transferred from and to the $\mathbf{\Lambda}$ and $\mathbf{V}$ register files whose interfaces are designed for a single 2×2 matrix per access.
- **Matrix access control** decides which of the input matrices in the IO register file is processed. Every time one serial access cycle is completed, the control flow switches to the next matrix where the same serial access pattern with the same parallel ordering permutation is used.
- **Parallel access control** manages the iteration through the parallel ordering permutations. Once all serial accesses of the current parallel ordering permutation have been used on all matrices, the next parallel ordering permutation is generated.
- **Sweep control** keeps track of what sweep the architecture is in to know when the SVD computation is finished. After completion, the design switches into an IO state, so results can be read out and the matrices for the next set of SVDs can be written in.

The address generation units (AGUs) in Figure 7 follow the above control flow. Runtime configuration of the input matrix sizes is implemented by excluding a configurable number of registers from the parallel ordering pair generation in Figure 6. To reduce control overhead, the AGU for the data flow from the IO register file to the $\mathbf{\Lambda}$ multiplication engine (IO-$\mathbf{\Lambda}$ mul AGU) and the AGU between the $\mathbf{V}$ register file and the $\mathbf{V}$ multiplication engine ($\mathbf{V}$ reg-$\mathbf{V}$ mul AGU) buffer the generated read addresses for the inputs to the multiplication engines in two FIFOs to reuse the same addresses for writeback.

2) Latency Analysis and Circuit Dimensioning: To efficiently utilize the $N \times N$ SVD architecture, all computational stages should be kept busy at all times. Since Algorithm 2 performs SVD in an iterative fashion, the result of one iteration has to be computed by the 2×2 SVD unit and the multiplication engines, and then be written back to the IO register file before the next iteration can start. Parallel ordering allows the independent processing of $N/2$ SVDs of size 2×2 without updating the input matrix. However, the architecture in Figure 7 consists of more than $N/2$ computational stages, which means that the design cannot be fully utilized when operating on one input matrix only. This problem can be mitigated by processing M_I input matrices in an interleaved scheme, as indicated in Section III-B1.

Since handling a bigger number of matrices results in increased storage requirements for both register files, it is desirable to reduce interleaving to the smallest degree that enables maximum utilization of all functional units. To that end, the previously introduced architecture and its data flow have to be analyzed with respect to latency (i.e., the total number of computational stages). The 2×2 SVD block itself causes a latency of nine computational cycles; four from Q1, another four from Q2, and one from the 2×2 MMU. Buffering of $\mathbf{J}_\mathrm{l}$ and $\mathbf{J}_\mathrm{r}$ causes an additional latency of two computational cycles. Finally, the two cascaded matrix multipliers in the $\mathbf{\Lambda}$ multiplication engine cause another delay corresponding to one computational cycle. Therefore, the overall latency of the processing pipeline equals $L_\mathrm{T} = 12$ computational cycles, meaning the processing pipeline has to be fed with at least $L_\mathrm{T} + 1$ inputs for 2×2 SVD calculation before operating again on the first input matrix. Since each parallel ordering permutation contains $N/2$ independent pairs, the number M_I of interleaved matrices has to satisfy

$$M_\mathrm{I} \geq \left\lceil \frac{2(L_\mathrm{T}+1)}{N} \right\rceil = \left\lceil \frac{26}{N} \right\rceil . \quad (27)$$

The associated storage requirement in terms of complex-valued scalars based on (27) is

$$S_{\mathrm{I}} = M_{\mathrm{I}} N^2 = \left\lceil \frac{2(L_{\mathrm{T}}+1)}{N} \right\rceil N^2 = \left\lceil \frac{26}{N} \right\rceil N^2. \tag{28}$$

In line with the IEEE 802.11ac standard, 8×8 MIMO is defined as the maximum antenna setup and the $N \times N$ SVD template is configured accordingly. Based on (27) and (28), we see that $N = 8$ has a storage requirement of $M_{\mathrm{I}} = 4$, $S_{\mathrm{I}} = 256$, so the IO register file is designed to hold four 8×8 matrices arranged in 32 rows and 8 columns.

Next, the multiplication engines for $\mathbf{\Lambda}$ and $\mathbf{V}$ are dimensioned so they can handle the throughput generated by the 2×2 SVD block which always operates at full utilization. Based on Algorithm 2, there are

$$M_{\mathrm{mul}}^{\mathrm{pp}} = N^2/4 \tag{29}$$

three-factor matrix multiplications required per parallel ordering permutation. The matrix factors stem from $N/2$ SVDs of size 2×2, each taking $C_{\mathrm{S}} = 4$ clock cycles to compute, totaling in an SVD execution cycle count of

$$C_{\mathrm{svd}}^{\mathrm{pp}} = 2N \tag{30}$$

for one parallel ordering permutation. For the multiplication engine to match the throughput of the SVD block, it has a cycle budget of

$$C_{\mathrm{mul}}^{\mathrm{pp}} = \frac{C_{\mathrm{svd}}^{\mathrm{pp}}}{M_{\mathrm{mul}}^{\mathrm{pp}}} = \frac{8}{N} \tag{31}$$

per two-sided matrix multiplication which is one for the maximum use case of $N = 8$. This means the multiplication engine has to be designed to deliver a throughput of one three-factor matrix multiplication per clock cycle. Therefore, the $\mathbf{\Lambda}$ multiplication engine is composed of two cascaded matrix multiplication units whereof the second takes the output of the first as its left-hand input. The same argumentation applies to the dimensioning of the $\mathbf{V}$ multiplication engine with respect to one-sided matrix multiplications, so the corresponding block contains a single, pipelined 2×2 matrix multiplier.

IV. Numerical Precision Analysis

This section discusses the numerical precision requirements of $N \times N$ SVD precoding using the napSVD architecture. The results are summarized in Table I. The maximum requirements are considered achieved when the communication performance (i.e., FER) is indistinguishable from the performance delivered by a reference implementation using double precision floating-point arithmetic. The napSVD has three configuration parameters for numerical precision: number of sweeps, scalar wordwidth, and number of CORDIC iterations. All three influence energy consumption and FER. Reducing CORDIC iterations and scalar wordwidth below the configuration for floating-point equivalence causes significant FER-impairments while offering comparably small reductions of energy consumption. Therefore, both parameters are fixed to the use case requirements in Table I. In contrast, the number of sweeps is directly proportional to energy consumption, so the trade-off of energy consumption and FER is considered in the following.

Figure 8a to 8c show the FERs for a 4×4, 6×6 and 8×8 antenna setup for a varying number of sweeps N_{SW}. The analysis is based on extensive Monte Carlo simulations of an i.i.d. Rayleigh slow fading channel with additive white Gaussian noise (AWGN) according to the TGn-C model [17]. The frame structure conforms with the IEEE 802.11ac standard with an up-front block preamble. Each frame contains 2,304 byte of uncoded information, which corresponds to the maximum size of a MAC service data unit (MSDU) in IEEE 802.11ac (excluding the directional multi-gigabit (DMG) beamforming mode). All setups use LDPC channel coding according to IEEE 802.11 for the highest codeword length of 1944 bit, the highest code rate of $5/6$ and the densest symbol constellation alphabet: 256-QAM. MIMO detection is performed by a low-complexity non-iterative linear MMSE algorithm [18].

Antenna setup	2×2	4×4	6×6	8×8
CORDIC iterations	4	5	5	6
Scalar wordwidth	10	12	12	13
Number of sweeps	1	2	2-3	3-4

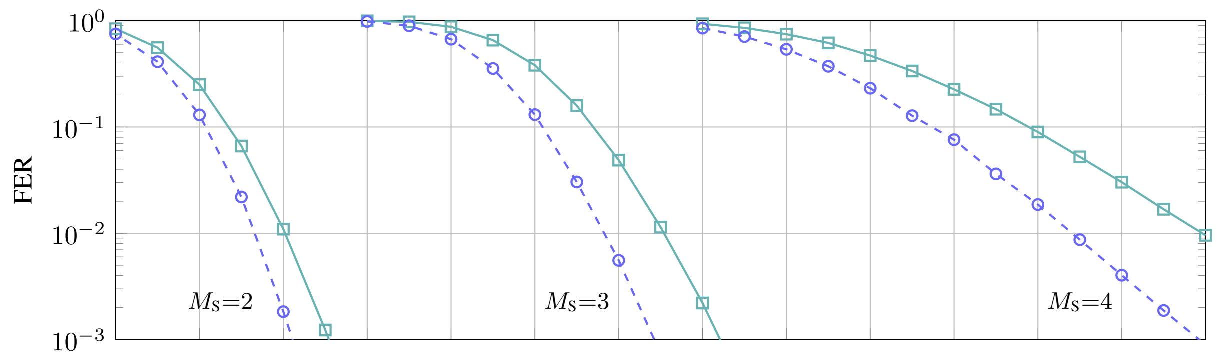
TABLE I: Precision requirements for SVD precoding.

The analysis shows that the precision requirement in terms of sweeps changes between antenna configurations but also within a single configuration when using a different number of eigenmodes/streams M_{S}. For a 4×4 transmission, two sweeps deliver double floating-point equivalent communication performance. For a target FER of $1\,\%$, the offset when using one sweep is around $3\,\mathrm{dB}$ when using all four eigenmodes and circa $1.4\,\mathrm{dB}$ and $0.7\,\mathrm{dB}$ for three or two eigenmodes, respectively. A 6×6 transmission requires three sweeps for floating-point equivalent communication performance for $M_{\mathrm{S}} \in \{5, 6\}$, and two sweeps otherwise. For $M_{\mathrm{S}} = 5$, switching from three sweeps to two causes a degradation of $0.7\,\mathrm{dB}$. For $M_{\mathrm{S}} = 6$, the gap for switching from three sweeps to two is significantly larger: around $3\,\mathrm{dB}$. Applying one sweep only is not even feasible, since the target FER is not achieved for realistic SNR values. The 8×8 setup needs four sweeps for floating-point equivalent communication performance at $M_{\mathrm{S}} = 8$ and three sweeps when using less eigenmodes. The gap in communication performance when using two sweeps instead of three varies from an almost negligible degradation for small M_{S} up to a $1.3\,\mathrm{dB}$ offset for seven modes. When reducing further to one sweep, the gap widens to slightly above $1\,\mathrm{dB}$ for small M_{S} via $4\,\mathrm{dB}$ for seven eigenmodes up to an impractically high SNR for eight modes.

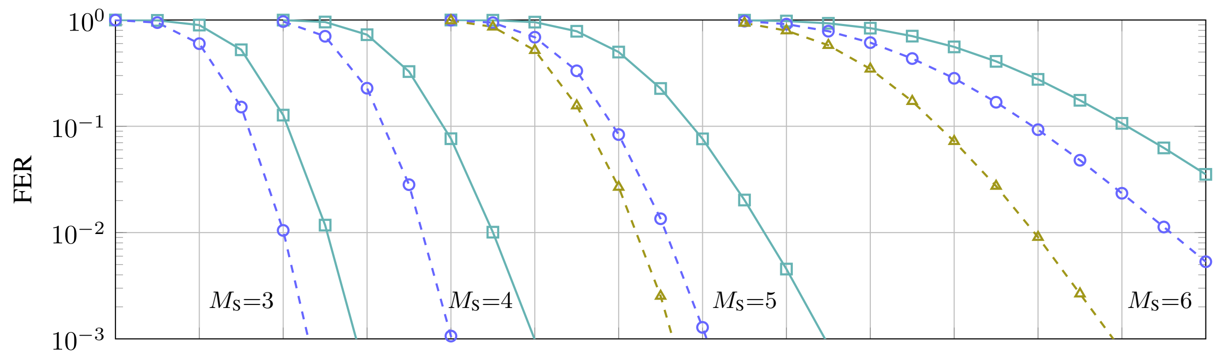
The execution time of the SVD algorithm scales linearly with the number of sweeps, and energy consumption scales linearly with execution time. For this reason, the number of sweeps is considered the primary parameter to trade communication performance for reduced computational complexity and energy consumption. Depending on the use case and SNR, this trade-off can be very attractive. Consider a 4×4 MIMO transmission with $M_{\mathrm{S}} = 2$ as in Figure 8a, for example. At $20\,\mathrm{dB}$, a transmission at around $99.8\,\%$ of the theoretically achievable maximum data rate can be exchanged for a transmission at $99\,\%$ whilst reducing the computational complexity and energy consumption of precoding by $50\,\%$.

V. Implementation Results

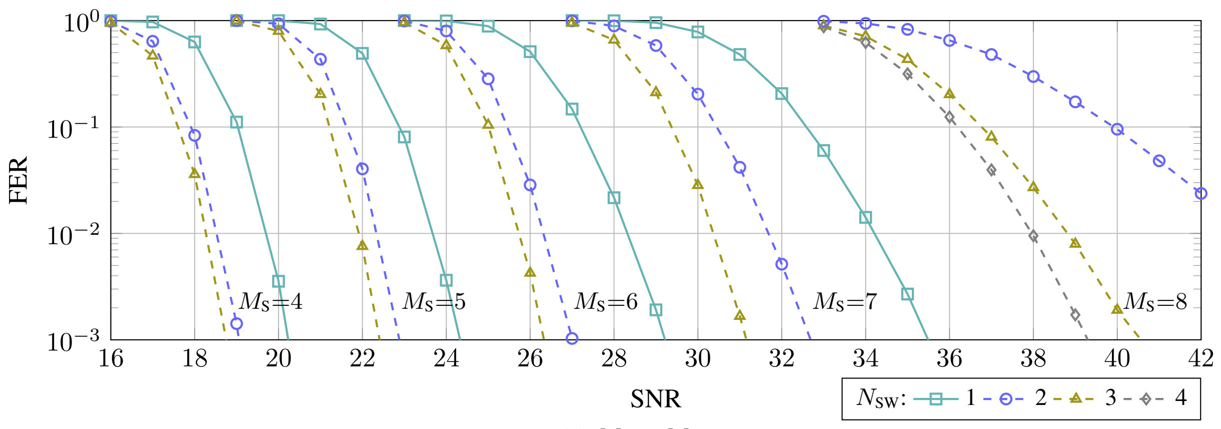
The $N \times N$ napSVD architecture template is design-time configurable with respect to the maximum supported N by



(a) $M_T = M_R = 4$



(b) $M_T = M_R = 6$



(c) $M_T = M_R = 8$

Fig. 8: Impact of number of sweeps on FER. Code rate: $5/6$. Constellation alphabet: 256-QAM.

setting the number of registers in the parallel ordering pair generation (Figure 6) of the AGUs, the size of the register files holding matrices $\mathbf{\Lambda}$ and $\mathbf{V}$ (see (28)), and the maximum throughput of both multiplication engines (see (31)). The architecture was layouted for $N = 8$ for a $90\,\mathrm{nm}$ CMOS technology with $1\,\mathrm{V}$ supply voltage. The wordwidth was chosen according to the maximum precision requirements for the SVD of an 8×8 matrix, meaning 13 bit per real-valued scalar inside the IO register file (wordwidth varies within the ASIC) and 12 bit per angular value. Figure 9 shows the resulting layout. The post-layout design achieves a clock frequency of $f_{\mathrm{clk}} = 752\,\mathrm{MHz}$. The design footprint is $1219\,\mathrm{\mu m}$ by $1214\,\mathrm{\mu m}$ including–and $1159\,\mathrm{\mu m}$ by $1154\,\mathrm{\mu m}$ excluding–power rings, which results in a die area of $1.48\,\mathrm{mm}^2$ and a standard cell area of $1.34\,\mathrm{mm}^2$.

Within the coherence time $T_{\mathrm{co}} = c/(8 f_{\mathrm{c}} v_{\mathrm{trx}}^{\max})$ [4] of an IEEE 802.11ac channel at carrier frequency $f_{\mathrm{c}} = 5\,\mathrm{GHz}$, the napSVD can compute the 8×8 precoding matrices for

$$N_{\mathrm{cl}}^{8\times 8} = \left\lfloor \frac{c\, f_{\mathrm{clk}}}{8 f_{\mathrm{c}}\, v_{\mathrm{trx}}^{\max} M_{\mathrm{F,u}}\, C_{\mathrm{svd}}^{8\times 8}} \right\rfloor = 5 \qquad (32)$$

wireless clients for the maximum $160\,\mathrm{MHz}$ channel with $M_{\mathrm{F,u}} = 484$ used subcarriers, maximum relative transmitter-receiver velocity $v_{\mathrm{trx}}^{\max} = 5\,\mathrm{m\,s^{-1}}$, speed of light c, and SVD computation cycle count $C_{\mathrm{svd}}^{8\times 8} = 448$. In conclusion, our architecture is IEEE 802.11ac real-time capable for multi-user scenarios.

Table II provides a breakdown of power consumption and hardware complexity of the napSVD architecture when performing 8×8 SVD. The 2×2 SVD accelerator is the most significant source of power consumption (49.7 %), followed by the two 2×2 matrix multiplication engines for $\mathbf{\Lambda}$ and $\mathbf{V}$ matrices (26.6 and 8.3 %). Despite a significant share of 19.6 and 12.3 % in hardware complexity, the IO and $\mathbf{V}$-matrix register files only consume 7.2 and 2.8 % of the overall power.

Functional unit	kGE	%	mW	%
SVD 2x2	133.0	37.1	383.0	49.7
$\mathbf{\Lambda}$ mul	70.5	19.7	205.0	26.6
IO reg	70.2	19.6	55.4	7.2
$\mathbf{V}$ mul	26.1	7.3	63.7	8.3
$\mathbf{V}$ reg	44.0	12.3	21.7	2.8
Misc	14.9	4.2	41.2	5.4
Total	358.7		770.0	

TABLE II: Hardware complexity and power consumption of 8×8 napSVD architecture (f_{clk} = 752 MHz, 90 nm CMOS).

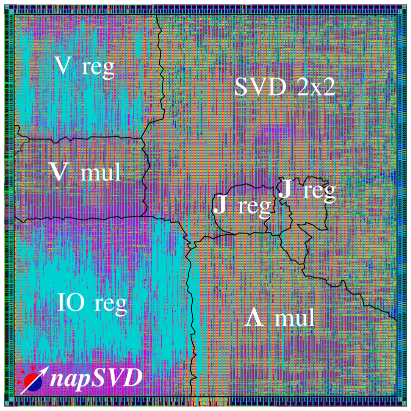


Fig. 9: Layout of 8×8 napSVD architecture in 90 nm CMOS.

Even though the two register files have the exact same storage capacity, the IO variant has a higher hardware complexity than the $\mathbf{V}$-matrix register file. The underlying cause is the more complex addressing scheme of the IO register file that allows reading and writing of a 2×2 submatrix specified by two row-indices and two column-indices. This requires a more complex multiplexing and demultiplexing network than for the $\mathbf{V}$-matrix register file, which provides random column access but always delivers data from two adjacent rows.

A. Use Case Energy Benchmark

This section assesses the power and energy consumption of the napSVD architecture when executing different use cases. The analysis is based on the precision requirements from Table I for 2×2, 4×4, 6×6 and 8×8 SVD precoding. Table III presents power consumptions $P^{\text{svd}}_{2\times2}$, $P^{\text{mul}}_{\mathbf{\Lambda}}$, $P^{\text{mul}}_{\mathbf{V}}$ and P_{Σ} of the 2×2 SVD block, the multiplication engines to generate matrices $\mathbf{\Lambda}$ and $\mathbf{V}$, and the overall napSVD design. Furthermore, the energy E_{sw} per sweep and the energy E_{clk} per clock cycle are listed. The energy is derived from a time-based power analysis in Synopsys PrimeTime, based on a simulation of the post-layout netlist with realistic stimuli from a channel simulator. There is a significant drop of 175 mW in power consumption when switching from 8×8 down to the 4×4 antenna configuration. Several factors contribute to this behavior. Smaller antenna configurations can operate numerically stable with less CORDIC iterations and a narrower wordwidth than the 8×8 setup (see Table I). The strong reduction in power consumption for the multiplication units (i.e., 49 % for the calculation of $\mathbf{\Lambda}$ comparing $N = 8$ and $N = 4$) also results from the multiplicative calculation scheme

Antenna setup	$P^{\text{svd}}_{2\times2}$ [mW]	$P^{\text{mul}}_{\mathbf{\Lambda}}$ [mW]	$P^{\text{mul}}_{\mathbf{V}}$ [mW]	P_{Σ} [mW]	E_{sw} [nJ]	E_{clk} [nJ]
2×2	318	-	-	402	2.14	0.534
4×4	363	104	36.1	595	19.0	0.791
6×6	367	155	48.1	673	53.7	0.895
8×8	383	205	63.7	770	115	1.027

TABLE III: Power and energy benchmark of SVD on post-layout napSVD model with f_{clk} = 752 MHz.

for $\mathbf{\Lambda}$ and $\mathbf{V}$ shown in Algorithm 2. While the 2×2 SVD accelerator is always operating under full load, the utilization of the multiplication engines depends on the size of the input matrix. In addition, Table III compares the energy consumption per SVD sweep among different matrix sizes. Due to the difference in cycle counts per sweep and in base power consumption, one sweep for an 8×8 matrix consumes around six times more energy than for 4×4. Considering that a 4×4 SVD requires up to two sweeps, and an 8×8 SVD demands up to four sweeps, the total energy per SVD is up to 12 times higher for the 8×8 use case than for 4×4.

B. Comparison with State-of-the-art

This section puts the napSVD design into perspective by comparing it with other SVD architectures from the literature. For a fair comparison, all designs are scaled to 90 nm CMOS technology and a core voltage of 1 V. Results are summarized in Table IV. Note that energy consumption is difficult to compare among different works due to different implementation levels and simulation approaches. For this reason, we only list energy for works based on tape-outs or with a detailed description of how the results were obtained[1]. The matrix decomposition units MDU1 and MDU2 from [11] compute the SVD or QR factorization of 4×4 matrices. SVD is performed based on the Golub-Kahan (GK) algorithm. Both, bidiagonalization and diagonalization are expressed entirely based on Givens rotations. The slim core arithmetic unit consists of one two-dimensional CORDIC unit and one multiply-accumulate unit only. The main difference between MDU1 and MDU2 is that the latter can adapt the number of CORDIC micro-rotations at runtime. MDU1 and MDU2 have a small size of 42.3 and 38.1 kGE at a clock frequency of 133 and 272 MHz, respectively. The small hardware complexity per MDU is penalized by a high cycle count of 1,539 and 4,306 per SVD. For 4×4 MIMO, the napSVD architecture has a significant advantage in hardware efficiency of a factor of 10.7 and 13.2 over MDU1 and MDU2. In the light of this significant gap, it should be noted that the napSVD architecture does not support QR factorization. The architecture in [13] is based on [11] but tailored to 4×4 SVD only. Still, the napSVD has a hardware efficiency advantage factor of 3.1

[1]The variance of simulated energy consumption is significant among different simulation approaches. To demonstrate this effect, the energy consumption of the napSVD design was evaluated using a simple but common approach based on fixed toggle rates at the input ports. These toggle rates are then propagated through the design and switching activity is derived accordingly. This approach delivered an energy consumption 20 times lower than the results obtained from the setup described above. Even for taped-out designs like the MIMO detector from [19], a change of stimuli alone caused a sixfold difference in energy consumption.

over [13], in addition to supporting multiple matrix dimensions efficiently. One contribution to the efficiency advantage of the napSVD over [11] and [13] comes from its beneficial numerical properties. Even for 8×8 MIMO, the two-sided Jacobi scheme from Algorithm 2 operates numerically stable on a 13-bit fixed-point data format using CORDIC units with a maximum of six micro-rotations. The GK-based designs in [11] and [13], on the other hand, contain memory with 16-bit per real-valued scalar, and [13] mentions that the internal wordwidth widens up to 19 bit. Also, the design in [13] requires nine CORDIC micro-rotations for numerically stable operation. Authors of [14] use a systolic array for *generalized triangular decomposition* with SVD as a special case for up to $8{\times}8$ MIMO. Inefficiencies associated with systolic arrays [11], [13], particularly hardware underutilization, give the napSVD architecture a 25-fold advantage in terms of area efficiency for 8×8 channel matrices. Since [14] supports smaller size matrices by disabling processing elements in the array, the napSVD, that always fully utilizes its core 2×2 SVD unit, has a 100-fold area efficiency advantage for 4×4 MIMO.

Authors of [20] compute SVDs up to 4×4. The underlying algorithm called SL-SVD (superlinear convergence) relies on the ordered singular values $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \sigma_4$ being different enough so that $\sigma_1^n \gg \sigma_2^n \gg \sigma_3^n \gg \sigma_4^n$ is fulfilled for a sufficiently high $n \in \mathbb{N}$. The authors performed simulations for one channel scenario and claim that $n = 8$ is sufficient. All further efficiency assessments are based on that claim. Note that the choice of n is highly dependent on the channel characteristics and will be higher if the singular values lie closer together. This becomes more and more problematic for bigger size channel matrices. In the corner case for two or more singular values being equal, the algorithm never converges. A channel with equal singular values is the optimum case that maximizes channel capacity [4], though. The aforementioned assumption buys the design in [20] an advantage of 25 % over the napSVD in terms of area efficiency for 4×4 SVD.

ASIP design [21] provides an interesting comparison with a programmable solution. The architecture of [21] is rather generic with a data path composed of four floating-point units for multiplication and addition, and four accelerators for square root calculation. Similar to the napSVD architecture, the ASIP can also compute SVDs of different size matrices. However, in the case of [21], this flexibility is penalized by a high execution time. Therefore, the napSVD has a 188-fold advantage in area efficiency over [21] for 8×8 SVD.

VI. Conclusion

This paper presented the napSVD ASIC template for SVD-based linear MIMO precoding for high throughput applications (e.g., IEEE 802.11ac). The algorithmic and architectural focus of the design is on versatility to support multi-mode (i.e., multiple antenna setups) MIMO precoding. To that end, the entire SVD is based on 2×2 matrix arithmetic operations that can be combined to form SVDs of bigger size matrices by application of the two-sided Jacobi method [5]. Therefore, the main computational units are a CORDIC based 2×2 SVD accelerator and two 2×2 matrix multiplication engines. The 2×2 data path is mainly controlled by address generation units that employ parallel ordering [6]. The data access scheme is repeated on the input data for multiple iterations/sweeps until the result converges to the actual singular value decomposition. To adapt to the numerical precision requirements of different matrix sizes, the ASIC is configurable in terms of used fixed-point wordwidth, number of CORDIC iterations in the 2×2 SVD accelerator, and number of conducted sweeps.

As a proof-of-concept, the template was design-time configured to support up to 8×8 MIMO. The resulting architecture has a hardware complexity of 359 kGE and achieves a clock frequency of $752\,\mathrm{MHz}$ using $90\,\mathrm{nm}$ CMOS technology with $1\,\mathrm{V}$ core voltage. A layout was conducted, occupying $1.34\,\mathrm{mm}^2$ of standard cell area. Numerical precision requirements for 2×2, 4×4, 6×6 and 8×8 MIMO in terms of wordwidth, CORDIC iterations and sweeps were evaluated by extensive Monte-Carlo simulations. Additionally, it was explored how the number of sweeps can be used to trade communication performance in terms of low frame error rate against energy efficiency. This trade-off is particularly attractive, since the energy consumption per SVD is directly proportional to the number of sweeps. When operating at full precision, the resulting area efficiencies of the napSVD are 140, 11.7, 4.68 and 1.25 Mmat/s/mm^2. The post-layout energy efficiency is 468, 26.3, 9.31 and 2.91 mat/µJ.

Comparable ASICs from the open literature typically only support up to 4×4 MIMO and are often limited to that configuration exclusively, which is a significant loss of versatility compared to the napSVD. Aided by favorable numerical properties, the napSVD has a hardware efficiency advantage of a factor around three to 13 over the well-known designs in [11], [13] and achieves similar performance to [20] despite being more flexible, both in terms of supported antenna setups and wireless channel types. The ASIC-ASIP efficiency gap to [21] spans two orders of magnitude.

The main contribution of this work is to demonstrate that an ASIC implementation for a computationally demanding application like MIMO precoding can still be versatile in terms of supported use cases by following a systematic divide-and-conquer approach in algorithm and architecture design. Achieving superior or similar efficiencies compared to other designs from the literature, our design also proves that versatility and efficiency do not need to be contradicting.

References

[1] "IEEE Standard for Information technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Specific requirements Part 11, Amendment 5," *IEEE Std 802.11n-2009*, pp. 1–502, Oct. 2009.

[2] "IEEE Standard for Information technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Specific requirements Part 11, Amendment 4," *IEEE Std 802.11ac-2013*, pp. 1–425, Dec. 2013.

[3] E. Witte, F. Borlenghi, G. Ascheid, R. Leupers, and H. Meyr, "A Scalable VLSI Architecture for Soft-Input Soft-Output Single Tree-Search Sphere Decoding," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 57, no. 9, pp. 706–710, Sep. 2010.

[4] D. Tse and P. Viswanath, *Fundamentals of Wireless Communications*. Cambridge University Press, 2004.

[5] G. Forsythe and P. Henrici, "The cyclic Jacobi method for computing the principal values of a complex matrix," *Transactions of the American Mathematical Society*, vol. 94, pp. 1–23, 1960.

[6] R. Brent and F. Luk, "A systolic architecture for singular value decomposition," Department of Computer Science, Cornell University, Ithaca, NY, USA, Tech. Rep., 1982. [Online]. Available: http://hdl.handle.net/1813/6361

[7] R. Brent, F. Luk, and C. Van Loan, "Computation of the Singular Value Decomposition Using Mesh-Connected Processors," Department

	napSVD	MDU1 [11]	MDU2 [11]	Senning [13]	Yang [14]	Zhan [20]	Kaji [21]
Process [nm]	90	180	180	180	90	90	90
Implementation	layout	tape-out	tape-out	synthesis	layout	layout	synthesis
Architecture type	ASIC	ASIC	ASIC	ASIC	ASIC	ASIC	ASIP
SVD algorithm	2-Sided Jacobi	GK	GK	GK	2-Sided Jacobi	SL-SVD	GK
Matrix dimensions	2x2 / 4x4 / 6x6 / 8x8	4x4	4x4	4x4	2x2 / 4x4 / 6x6 / 8x8	1x1-4x4	4x4 / 8x8 / 16x16
Computation cycles	4 / 48 / 120 / 448	1,539	4,306	492	- / 490 / - / 1,138	26	28,816 / 295,323 / -
Core voltage [V]	1	1.8	1.8	-	1	1	-
Clock frequency [MHz]	752	133	272	149	112	182	400
Core area [mm^2]	1.34	0.41	0.37	-	1.96	0.48	-
HW complexity [kGE]	359	42.3	38.1	42.3	378	120	54.5
Scaled clock freq. [MHz]	752	266	544	298	112	182	400
Scaled core power [mW]	402 / 595 / 673 / 770	49	33	-	-	-	-
Scaled core area [mm^2]	1.34	0.10	0.093	-	1.96	0.48	-
HW eff. [mat/s/GE]	524 / 43.7 / 17.5 / 4.68	4.09	3.32	14.3	- / 0.607 / - / 0.261	58.3	0.255 / 0.025 / -
Area eff. [Mmat/s/mm^2]	140 / 11.7 / 4.68 / 1.25	1.69	1.37	-	- / 0.117 / - / 0.050	14.6	-
Energy eff. [mat/μJ]	468 / 26.3 / 9.31 / 2.91	1.75	1.93	-	-	-	-

TABLE IV: Comparison of napSVD architecture with state-of-the-art. All efficiencies scaled to 90 nm CMOS.

of Computer Science, Cornell University, Ithaca, NY, USA, Tech. Rep., 1983. [Online]. Available: http://hdl.handle.net/1813/6367

[8] N. Hemkumar and J. Cavallaro, "A systolic VLSI architecture for complex SVD," in *Circuits and Systems, 1992. ISCAS '92. Proceedings., 1992 IEEE International Symposium on*, vol. 3, May 1992, pp. 1061–1064.

[9] G. Golub and W. Kahan, "Calculating the Singular Values and Pseudo-Inverse of a Matrix," *Journal of the Society for Industrial and Applied Mathematics Series B Numerical Analysis*, vol. 2, no. 2, pp. 205–224, 1965. [Online]. Available: http://dx.doi.org/10.1137/0702016

[10] A. S. Householder, "Unitary Triangularization of a Nonsymmetric Matrix," *J. ACM*, vol. 5, no. 4, pp. 339–342, Oct. 1958. [Online]. Available: http://doi.acm.org/10.1145/320941.320947

[11] C. Studer, P. Blosch, P. Friedli, and A. Burg, "Matrix Decomposition Architecture for MIMO Systems: Design and Implementation Trade-offs," in *Signals, Systems and Computers, ACSSC 2007. Conference Record of the Forty-First Asilomar Conference on*, Nov. 2007, pp. 1986–1990.

[12] G. H. Golub and C. F. Van Loan, *Matrix computations (3rd ed.)*. Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[13] C. Senning, C. Studer, P. Luethi, and W. Fichtner, "Hardware-efficient steering matrix computation architecture for MIMO communication systems," in *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, May 2008, pp. 304–307.

[14] C. H. Yang, C. W. Chou, C. S. Hsu, and C. E. Chen, "A Systolic Array Based GTD Processor With a Parallel Algorithm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 4, pp. 1099–1108, Apr. 2015.

[15] D. Guenther, R. Leupers, and G. Ascheid, "Efficiency Enablers of Lightweight SDR for MIMO Baseband Processing," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 24, no. 2, pp. 567–577, Feb. 2016.

[16] J. Volder, "The CORDIC Trigonometric Computing Technique," *Electronic Computers, IRE Transactions on*, vol. EC-8, no. 3, pp. 330–334, Sep. 1959.

[17] E. Perahia and R. Stacey, *Next Generation Wireless LANs: Throughput, Robustness, and Reliability in 802.11n*. Cambridge, UK: Cambridge University Press, 2010.

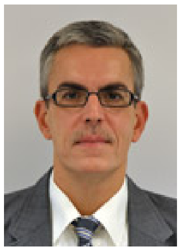
[18] D. Seethaler, G. Matz, and F. Hlawatsch, "An efficient MMSE-based demodulator for MIMO bit-interleaved coded modulation," in *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, vol. 4, Nov. 2004, pp. 2455–2459.

[19] C. Studer, S. Fateh, and D. Seethaler, "ASIC Implementation of Soft-Input Soft-Output MIMO Detection Using MMSE Parallel Interference Cancellation," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 7, pp. 1754–1765, Jul. 2011.

[20] C.-Z. Zhan, Y.-L. Chen, and A.-Y. Wu, "Iterative Superlinear-Convergence SVD Beamforming Algorithm and VLSI Architecture for MIMO-OFDM Systems," *Signal Processing, IEEE Transactions on*, vol. 60, no. 6, pp. 3264–3277, Jun. 2012.

[21] T. Kaji, S. Yoshizawa, and Y. Miyanaga, "Development of an ASIP-based singular value decomposition processor in SVD-MIMO systems," in *Intelligent Signal Processing and Communications Systems (ISPACS), 2011 International Symposium on*, Dec. 2011, pp. 1–5.

Daniel Guenther received the Dipl.-Ing. degree in Electrical Engineering (Communications Eng.) from RWTH Aachen University in 2011. He is a Ph.D. student at the Institute for Integrated Signal Processing at RWTH Aachen University. His research focuses on software defined radio solutions and the design of flexible hardware for mobile, wireless communications.



Rainer Leupers received the M.Sc. and Ph.D. degrees in Computer Science with honors from the University of Dortmund, Germany, in 1992 and 1997. From 1997-2001 he was the chief engineer at the Embedded Systems chair at the University of Dortmund. During 1999-2001 he was also a team leader at ICD, where he headed industrial service projects. In 2002, Dr. Leupers joined RWTH Aachen University as a professor for Software for Systems on Silicon. His research and teaching activities comprise software development tools, processor architectures, and electronic design automation for embedded systems, with emphasis on compilers, ASIPs, and MPSoC design tools. He has been a co-founder of LISATek, an EDA tool provider for embedded processor design, acquired by CoWare Inc. He has served as consultant for various companies, as an expert for the European Commission in FP7, and in the management boards of large scale projects like UMIC, HiPEAC, ARTIST, SHAPES and Euretile.

Gerd Ascheid received his Diploma and Ph.D. (Dr.-Ing.) degrees in Electrical Engineering (Communication Eng.) from RWTH Aachen University. In 1988 he started as a co-founder and managing director CADIS GmbH which successfully brought the system simulation tool COSSAP to the market. In 1994 CADIS was acquired by Synopsys Inc., a California-based EDA market leader. From 1994-2003 Gerd Ascheid worked as Director/Senior Director with Synopsys. In 2003 Gerd Ascheid joined RWTH Aachen University as a full professor in the Institute for Communication Technologies and Embedded Systems. His research interest is in wireless communication algorithms, application specific integrated platforms, in particular, for mobile terminals and cyber physical devices. Gerd Ascheid has co-authored three books, published numerous papers in the domain of digital communication algorithms and ASIC implementation and is founder of several successful start-up companies.