Hardware Acceleration of Singular Spectral Analysis: A Case Study on NQR Spectroscopy

K. Thulasiram Varma, Rangababu Peesapati, and Ch. V. Rama Rao

***Abstract*—Singular Spectral Analysis (SSA) is a computationally intensive approach to denoise and detect a time-series signal. It requires the evaluation of eigenvalues and eigenvectors of a covariance matrix, which is the computationally intensive step in the SSA algorithm. The current work presents a feasible approach to implement the algorithm in embedded hardware using a PYNQ-Z2 Field Programmable Gate Array (FPGA) board. We implemented the algorithm using both the Processing System (PS) and the Programmable Logic (PL) of the PYNQ-Z2 System on Chip (SoC) with the help of a High-Level Synthesis (HLS) tool. A case study is carried out on a Nuclear Quadruple Resonance (NQR) signal. The implementation result demonstrates a hardware acceleration of 15.48x with respect to the equivalent software implementation of the algorithm on the ARM Cortex-A9 processor.**

***Index Terms*—Singular Spectral Analysis(SSA), System on Chip (SoC), High-Level Synthesis (HLS), Pynq-Z2 Field Programmable Gate Array (FPGA).**

Original Research Paper
DOI: 10.53314/ELS2529022V

I. Introduction

In the present-day scenario, data play a vital role in all applications. Many decomposition-based algorithms exist to extract meaningful information from a large amount of data. Singular Value Decomposition (SVD) is a technique that is popularly used in many domains, such as finance, medicine, image processing, computer vision, machine learning, and weather forecasting. Based on the principals of SVD, other feature selection algorithms, such as Principal Component Analysis (PCA), Independent Component Analysis (ICA), and Singular Spectral Analysis (SSA), were developed.

PCA transforms and projects the data into a new coordinate system. The set of new variables is called the principal components. The first principal component consists of maximum variance in the data, and the second one consists of reminding maximum variance orthogonal to the first principal component. Similarly, other principal components are derived. ICA is used in Blind Source Separation (BSS) problems, which helps to identify the origin of the signal sound in a mixed environment. The SSA algorithm specializes in decomposing the time series signal into trends, oscillations, and noise. All of the aforementioned algorithms are computationally intensive due to their reliance on matrix multiplication, eigenvalue computation, and inverse projection operation [1]–[3].

The primary motivation of this work is to reduce the computational complexity of the SSA hardware accelerator, which helps in trend extraction, noise reduction, and periodicity identification in large time series data. Applications range from financial forecasting to environmental monitoring and biomedical signal processing. The novelty of this work is twofold: implementing the SSA algorithm with the HLS approach using a covariance calculation of the Hankel matrix with a vectorial matrix multiplication with only upper diagonal matrix elements and further utilizing the power method to find the selective dominant eigenvalues and vectors for the reconstruction of the signal.

High-level synthesis (HLS) tools help developers to design hardware by describing the desired functionality with high-level languages such as C, C++, or SystemC rather than traditional hardware description languages (HDLs) like VHDL or Verilog. However, these HLS tools generate Verilog hardware. Furthermore, HLS provides advantages of improved productivity and seamless integration with the RTL and SoC framework, which find their applications ranging from cryptography and Internet of Things (IoT) to machine learning tasks and many more [4], [5]. The work explored the possibility of implementing the whole algorithm with a hardware-software co-design approach where signal reconstruction is performed on the Processing System (PS) side and matrix multiplication and eigenvalues and vectors computation was carried out on the Programmable Logic (PL) side. Intellectual property (IP)s were generated through the Vivado High-Level Synthesis (HLS) tools for the operations above. Further, these IPs were integrated into SoC's PL side. Hardware-software co-design on FPGA improves the overall system's throughput by exploiting parallel computation,

In addition, this paper presents a case study of the SSA algorithm and its hardware implementation on Nuclear Quadrupole Resonance (NQR) for denoising, trend, and periodicity identification. NQR signal processing is a non-invasive technique for identifying chemical substances [6]. This property has applications in the identification of narcotics and explosive materials. Acquiring a signal in an open environment filled with external noise and interference is challenging. To clean/denoise the signal and estimate the signal parameters, signal processing techniques and their hardware are essential.

We propose an SSA-based signal denoising approach in

Manuscript received on November 9th, 2024. Received in revised form on January 31st, 2025. Accepted for publication on March 10th, 2025.

K. Thulasiram Varma is with the Department of Electronics and Communication Engineering, National Institute of Technology Meghalaya, Shillong 793003, India.(e-mail: kthulasivarma@nitm.ac.in; phone: +917888967498).

Rangababu Peesapati is with the Department of Electronics and Communication Engineering, Indian Institute of Information Technology, Design and Manufacturing, Kurnool, 518008, India.

Ch. V. Rama Rao is with the Department of Electronics and Communication Engineering, National Institute of Technology Warangal, 506004, Telangana, India.

previous work [7] and observe a 32.4 dB improvement in signal quality in terms of SNR. In this paper, we carry forward the work and implement the SSA accelerator on FPGA hardware. The SSA hardware provides an acceleration factor of 15.48X, demonstrated compared to the software.

The related work on SSA implementations on FPGA is presented in Section II. The mathematical preliminaries of the SSA algorithm are explained in Section III. Section IV contains the proposed hardware architecture to implement the SSA algorithm on FPGA. Section V of this paper presents the results and discussion, followed by a conclusion and references.

II. Related works

Although the applications of the SSA algorithm and its variants are explored in the literature, very few works have been reported that implement the SSA algorithm on hardware such as FPGA. Huang et al., [8] implemented the SSA algorithm on the Hankel tensor using Tucker decomposition on various computational platforms such as Central Processing Unit (CPU), Graphics Processing Unit (GPU), and FPGA. Byun et al., [9], and his team extracted Auditory Evoked Potential (AEP) signals from the brain by eliminating multiple noises using SSA and ensemble averaging. Cyclone IV FPGA implemented the SSA algorithm with a 1 MHz operating frequency for signal extraction. 61.2% reduction in stimulus repetitions was observed, and the signal matching of 83.2% to the original was observed. The SVD portion of the SSA algorithm contains dominant signal singular values extracted using a one-sided Jacobi algorithm, which requires Coordinate Rotation Digital Computer (CORDIC) functions for trigonometric operations.

For eigenvalues and vector extraction of a matrix, mathematical approaches like Jacobi rotation, Lanczos-QR decomposition, and household transformation were studied and successfully implemented on different Field Programmable Gate Array (FPGA) to compute eigenvalues and eigenvectors of a matrix. Gleb et al., [10] developed two approaches to quickly multiplying the Hankel matrix with a vector. The Fast Fourier Transform (FFT) is applied to the auxiliary Hankel matrix and vector in the first approach. The drawbacks of this approach are loss of data due to cancellation errors. The second approach uses Karatsuba multiplication and observed lesser multiplication operations without data loss. Mohammad et al., [11] worked on the acceleration of PCA using High-Level Synthesis (HLS) on both Virtex 7 FPGA and Zed boards. Reported an improvement in the execution time and reduction in resource utility and power consumption compared to GPU, CPU and other works, and observed 0.67 ms execution time for 30x16 matrix with respect to Virtex 7 FPGA. Using the same approach of accelerating PCA on a Zed board, they observed an execution time of 0.44 sec with an operating frequency of 90 MHz on an image size of 640x480x12. Aysel et al., [12] implemented PCA on FPGA. In implementing PCA normalization, covariance matrix calculations and eigenvalue decomposition steps were performed using the Jacobi cycle-by-row method for the matrix size of 100×100. It uses the CORDIC algorithm to accelerate trigonometric and square root operations while implementing the Jacobi method on the FPGA. Based on the reported work [12], the authors implemented SSA using a Singular Value Decomposition (SVD) based approach, which performs principal components extraction that involves inversion and multiplication of the whole matrix. Further performing inversion operations for an entire matrix leads to increased resources and time-consumption of operation. The reported hardware SVD implementation consumes more than 121% of BRAM resources to implement a 100×100 matrix, which is not feasible. Although the literature suggests multiple works to implement Singular Spectral Analysis (SSA) using Singular Value Decomposition (SVD) and other approaches, there is a need to develop simple and computationally inexpensive methods to implement the SSA algorithm on FPGA. To overcome this problem, we proposed effective matrix multiplication, eigenvalue, and vector calculation approaches to extract the principal components of the 240×240 matrix.

III. Singular Spectral Analysis

SSA is a data-driven approach that decomposes time series signals into trends, oscillations, and noise using principal components [13]. The SSA algorithm has been used to remove artifacts and noise from biological signal processing [14]. It is also used in weather forecasting to find trends and seasonality in huge and dynamically changing weather data [15].

Fig. 1 shows that the SSA algorithm is divided into decomposition and reconstruction. The first stage of the algorithm internally consists of two steps: embedding the signal onto a Hankel matrix and identifying principal components. In the second stage of the algorithm, grouping and anti-diagonal averaging are performed to retrieve the desired signal in its original dimension [16].

The first step of the SSA algorithm is to embed the one-dimensional input signal onto a Hankel matrix, as shown in (1), where N is the length of the signal and window length $M = N/2$. Since the embed matrix $X^{'}$ is Hankel in nature, all anti-diagonal elements are the same.

$$X^{'} = \begin{bmatrix} X(t_1) & X(t_2) & ... & X(t_{N-M+1}) \\ X(t_2) & X(t_3) & ... & X(t_{N-M+2}) \\ ... & ... & ... & ... \\ X(t_{N-M+1}) & X(t_{N-M+2}) & ... & X(t_N) \end{bmatrix} \quad (1)$$

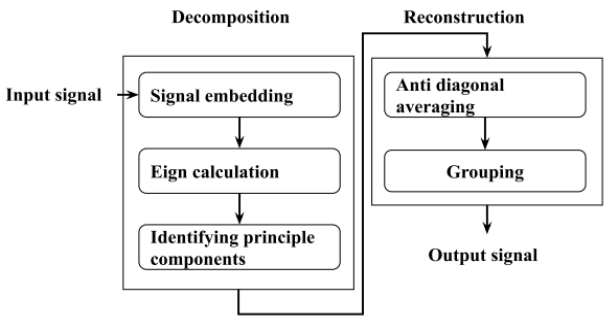

Fig. 1. SSA Flowchart.

Once the Hankel matrix is generated, to find the principal components (PC), the covariance of the Hankel matrix is calculated using (2), and from the covariance matrix, the eigenvalues (λ) and the eigenvectors (ρ) are computed.

$$C = (X^{'})^{T} * X^{'}/(N - M + 1) \tag{2}$$

$$PC = (X^{'}) * (\rho) \tag{3}$$

To obtain principal components, the eigenvalues and corresponding eigenvectors are sorted in descending order, and the eigenvectors are multiplied by the transpose of the embedded matrix as shown in (3).

The grouping step involves separating and summing similar principal components on the basis of the eigenvalues. Finally, the resultant matrix is inverted upside down after multiplying the PC matrix with ρ^{T}. In the last step of the algorithm, the sub-diagonal elements starting from $(N - M + 1) + n$ where $n = 1....N$ averaged, which results in an output signal of length N [13], [17].

IV. Hardware Implementation

The SSA hardware involves multiple levels of implementation and refinements. The algorithm is first computed at the software level using MATLAB and then translated to Python scripting language to run the algorithm on the Processing System (PS) of the FPGA. At this stage, profiling has been performed to find out the bottlenecks of computation and observed that out of three major parts of the algorithm, i.e. covariance calculation, eigenvalue and eigenvector extraction, and reconstruction. The matrix multiplication computation involved in the calculation of the covariance matrix consumes 91% of the total time on PS of the FPGA compared to the eigenvalue and vector extraction in other parts of the algorithm. Based on this, the implementation is split into two parts: lightweight and fast operations on PS and computationally intensive operations, which are converted into HLS and designed and integrated as hardware accelerators on the PL side of the SoC. In this approach, the covariance calculation, eigenvalue computation, and vector calculations are part of the algorithm on the accelerator on the PL and the reconstruction part of the algorithm on the PS. The entire design flow of the implementation is depicted in Fig. 2.

With the help of the Vivado High-Level Syntheses (HLS) tool, the code can be written in high-level C code to convert synthesized RTL sequentially and in a pipeline fashion with the help of pragmas. A custom SSA accelerator IP-core with computationally intensive operations is built to implement in the PL using (2)–(7).

As we can see from (2) for the calculation of covariance, the embedded matrix ($X^{'}$) is multiplied by its transpose and divided by the window length (M). To implement this equation and to fetch the vectors through DMA seamlessly, an optimal solution is proposed for computing matrix multiplication where only a one-dimensional vector is used, unlike the traditional matrix multiplication approach, where each row of matrix A is multiplied with every column of matrix B. Since the input Hankel matrix is symmetric in nature,

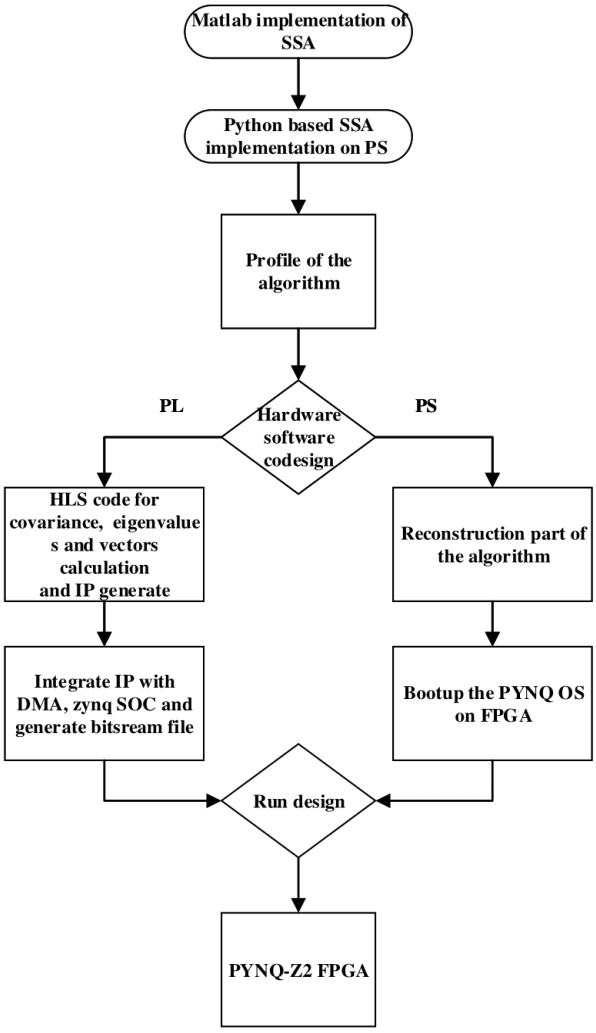

Fig. 2. Design implementation flow.

i.e., the upper triangle of the matrix is equal to the lower triangle, instead of calculating covariance for the whole matrix, only the upper triangle elements of the matrix are computed. The corresponding covariance is calculated. In this way, the computation is further reduced by the number of iterations and accumulation operations shown in Algorithm 1.

Once the covariance matrix is calculated, the power method is implemented as shown in Algorithms 2 and 3. The power method is an iterative approach to finding dominant eigenvalues and corresponding eigenvectors. This approach best suits the diagonal matrix obtained by performing the covariance calculation in the previous step. The power method begins with an initial guess of the dominant eigenvector, usually normalized all one vector as shown in (4) [18].

$$V = V_0/\left\|V_0\right\|_2 \quad V_0 = \begin{bmatrix}1 & . & . & . & 1\end{bmatrix}^{T} \tag{4}$$

Compute (5) to obtain new eigenvectors until the converging criteria is met. The eigenvalue for the vector is calculated using (6).

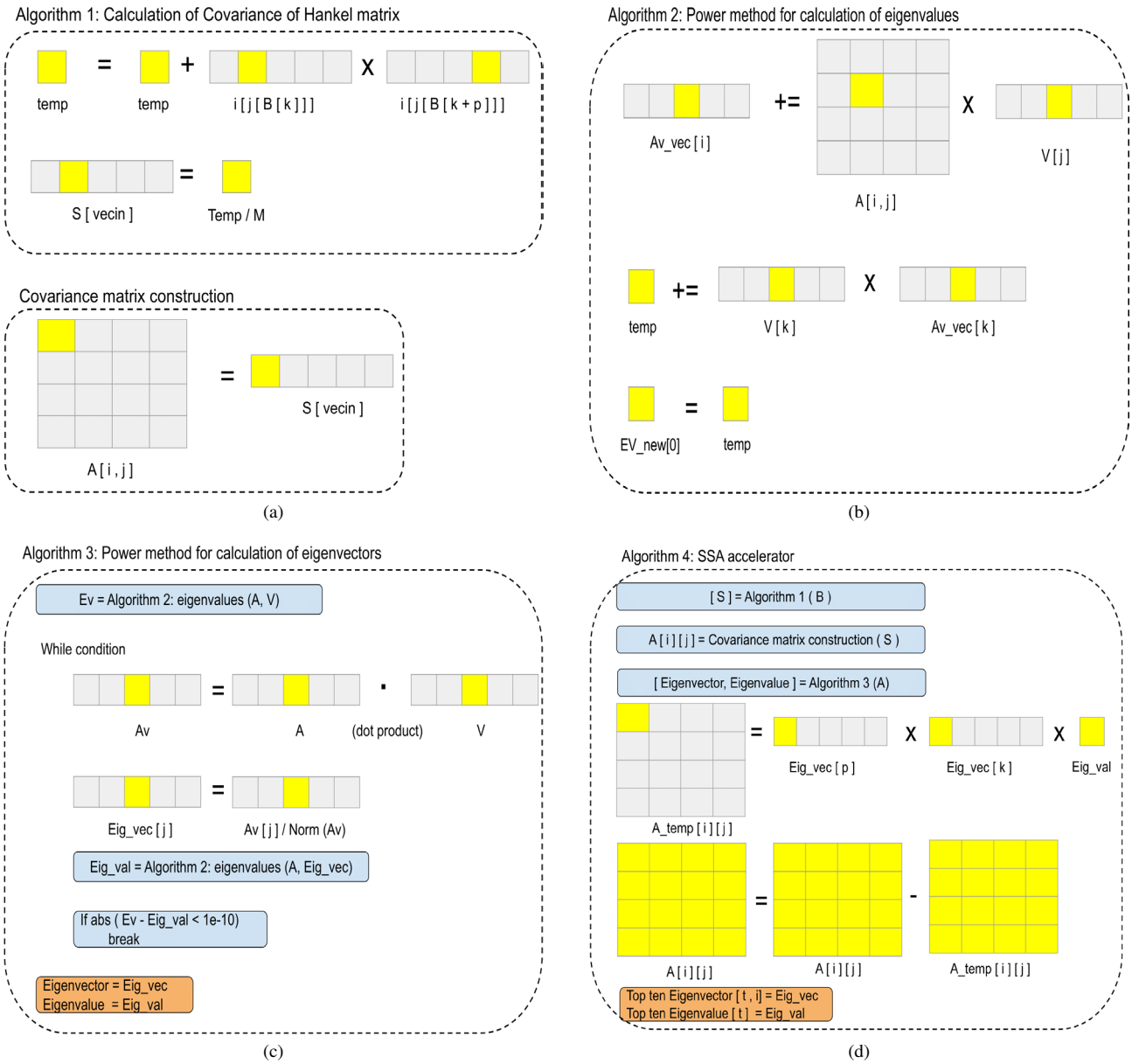

Fig. 3. (a) Algorithm 1: Covariance calculation and matrix construction, (b) Algorithm 2: Power method for calculation of eigenvalues,(c) Algorithm 3: Power method for calculation of eigenvectors,(d) Algorithm 4: SSA accelerator.

$$V_i = AV_{i-1} / \|AV_{i-1}\|_2 \tag{5}$$

$$EV = AV_i * V_i^T \tag{6}$$

The stopping criterion for the iteration is achieved when the difference between the new eigenvalue and the previous eigenvalue is less than 1e-10, i.e. $error = abs(EV - EV\_new)$.

$$A_{i+1} = A_i - EV * V * V^T \tag{7}$$

This eigenvalue computation involves the calculation of the second highest eigenvalue and eigenvectors, the product of the first highest eigenvectors and eigenvalue is subtracted from the main covariance matrix as shown in (7); likewise, the top ten eigenvalues and eigenvectors are calculated for this application, the same is depicted in Algorithm 4 [19], [20].

Fig. 3 illustrates the algorithmic implementation of a custom SSA IP core starting from the calculation of the multiplication of the covariance matrix and the construction of the resultant matrix in Fig. 3(a), the functional operation of the calculation of the eigenvalue and the eigenvector using the power method is shown in Figs. 3(b) and 3(c), and the complete top-level wrapper to calculate the top 10 eigenvalues and vectors is shown in Fig. 3(d).

V. Results and Discussion

The SSA algorithm is implemented on the PYNQ-Z2 FPGA board, which has a 650MHz ARM Cortex-A9 core processor with 512MB DDR3 RAM. The board includes 630 KB of fast

Algorithm 1 Calculation of Covariance of Hankel matrix

Input: B [L]
Output: S (Upper-triangle elements of the covariance matrix)

Ensure: $temp, i, io, p, vecin = 0; j, k = io$
  for all $i < M$ **do**
    for all $j < M$ **do**
      for all $k < M + io$ **do**
        $Temp = temp + B[k] * B[k + p]$
      end for
      $p = p + 1$
      $S[vecin] = temp/M$
      $vecin = vecin + 1$
      $temp = 0$
    end for
    $p = 0$
    $io = io + 1$
  end for

Algorithm 2 Power method for calculation of eigenvalues

Input : A[M][M], V[M]
Output : EV_new[vec]

Ensure: $temp, i, j, k = 0; vec = 1$
  for all $i < M$ **do**
    for all $j < M$ **do**
      $Av\_vec[i] += A[i][j] * V[j]$
    end for
  end for
  for all $k < M$ **do**
    $Temp += V[k] * Av\_vec[k]$
  end for
  $EV\_new[0] = temp$

Algorithm 3 Power method for calculation of eigenvectors

Input : A[M][M]
Output : Eig_vec,Eig_val

Ensure: $temp, norm\_temp, i, j, k = 0; vec = 1$
  for all $i < M$ **do**
    $V[i] = 1/sqrt(n)$
  end for
  $Algorithm\_2(A, V, EV)$
  while true **do**
    Av=A*V
    for all $i < M$ **do**
      $Temp += pow(Av[i], 2)$
    end for
    $norm\_temp = sqrt(temp)$
    for all $j < M$ **do**
      $Eig\_vec[j] = Av[j]/norm\_temp$
    end for
    $Algorithm\_2(A, Eig\_vec, Eig\_val)$
    if (abs(EV-Eig_val)<1e-10) **then**
      $Break$
    end if
    for all $k < M$ **do**
      $V[k] = Eig\_vec[k]$
      $Av[k] = 0$
    end for
    $EV = Eig\_val$
    $EV\_new\_out[0] = Eig\_val$
  end while

block RAM and 220 Digital Signal Processing (DSP) slices on the PL side. The experimental setup is shown in Fig. 4. The results are discussed with the help of a case study in the following subsection where SSA IP was used. The SSA IP is implemented sequentially and in a pipeline fashion. Further, floating and fixed point versions of the SSA IP on PL were explored.

A. Case study on NQR signal

Various spectroscopic methods, such as NMR, Raman, and Nuclear quadrupole resonance (NQR), are used to determine solid and liquid samples. They are used to evaluate the quality of agricultural and pharmaceutical products and to identify narcotics and explosive material [6]. Developing an efficient denoise algorithm and hardware implementation is essential to provide rapid time analysis. The case study implements the SSA algorithm to denoise the signal in less acquisition time than conventional averaging methods. The software-level implementation of the SSA algorithm is presented in the previous work [7]. In this current work, we implemented the same algorithm on the FPGA.

As shown in Fig. 5(a), the NQR signal of length 480 samples with different Signal to Noise Ratio (SNR), i.e. -5 dB,

Algorithm 4 SSA accelerator

Input : B[L]
Output : Eig_out[l]

Ensure: $ite, p, s, out\_l = 0; eig\_N = M + 1$
  $Algorithm\_1(B, S)$
  $A[M][M] = convertS[lo]$
  for all $i < ite$ **do**
    $Algorithm\_3(A, Eig\_vec, Eig\_Val)$
    for all $j < M$ **do**
      for all $k < M$ **do**
        $A\_temp[j][k] = Eig\_vec[p] * Eig\_vec[k] * Eig\_val[0]$
        $A = A - A\_temp$
      end for
      $p += 1$
    end for
  end for
  for all $l < M$ **do**
    $Eig\_out[l] = Eig\_vec[l]$
    $Eig\_out[l - 1] = Eig\_val[0]$
  end for

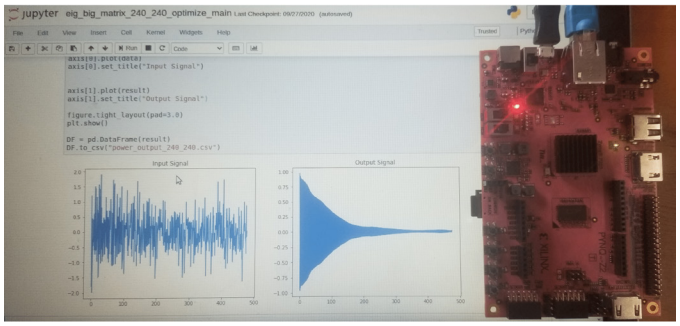

Fig. 4. Experimental setup.

0 dB and 5 dB, respectively, is generated in the PS with the help of (8),

$$S = A.e^{-t/T}cos(2\pi * Fc * t) \tag{8}$$

where Amplitude (A) = 1V, resonance frequency (Fc) = 10.25e4 Hz, sampling frequency (Fs) = 20.5e4 Hz and decay constants (T) = 0.5e-3 sec. The generated NQR signal is passed to the SSA IP core in PL with the help of Direct Memory access (DMA), where a covariance calculation module and power method module are built to extract eigenvalues and eigenvectors. The same is depicted in Fig. 6.

Figs. 5(c) and 5(d) corresponds to the dominant two eigenvalues and their eigenvectors of the covariance matrix; from this plot, it is evident that the first eigenvalue is the most dominant value for this signal; further, it is used for reconstruction of the original signal. The resulting eigenvalues and eigenvectors are appended into a one-dimensional array of 2410 samples of length and sent back to PS through DMA. The timing diagram in Fig. 7 provides a clear picture of data transfer from PS to PL via the master Advanced eXtensible Interface (AXI) stream bus. When the TREADY signal is high, TVALID is active for the duration of the data transfer (TDATA). TLAST specifies the last transaction of the data in the AXI stream protocol. The output eigenvalue was determined using covariance, eigenvalue, and vector calculation at the (IP) accelerator using Algorithms 1, 2, and 3. Further, the data were retrieved back to the PS using a slave AXI stream interface to reconstruct the denoised signal.

A comparison study is made with respect to resource utilisation and execution time between the sequential and pipeline approaches of implementing the algorithms on different signal lengths, and results are shown in Table I. From Table I, it is evident that the pipeline approach shows less amount of execution time compared to the sequential approach and further pipeline hardware consumes (BRAM-1%, DSP slices-12%, FF-43%, LUT-67%) higher resources than compared to sequential hardware while implementing the submodules of covariance calculation and power method on the PL side to extract eigenvalues and corresponding eigenvectors. From resources metrics, we can observe that for 512 sample signals, the BRAM utilisation is exceeded, so we considered a length of 480 sample signals. Apart from implementing the SSA algorithm in single-precision floating-point representation, fixed-point <29, 12> (29 total bits out of which 12 are integers, and the remaining 17 are fractional bits) were used for covariance calculation part of the algorithm and observed the reduction of resources utility in Flip Flops (FF) and Look-Up Tables (LUTs) in the order of 31.3% to 28% respectively as shown in Fig. 8.

Comparison is made between NQR signals with different SNR in Table II. From this table, we can observe that if the input signal power is more significant than noise, the time consumed by the system to execute the algorithm is less; this is due to the convergence rate of error while calculating the eigenvalues and vectors in PL. By referring to the input signal with 0 dB SNR in floating point (pipeline) representation, the total time to implement the whole SSA algorithm using PL modules (Matrix multiplication, eigenvalue, and vector computation) and signal reconstruction modules at the PS side is 14.36 sec. Further, an investigation was carried out by implementing covariance matrix calculation and power method on the PS. The total time on PS alone to execute the software version of the SSA algorithm is 222.3 sec. These results show that with the help of PL, an acceleration factor of 15.48x is achieved. Root Mean Square Error (RMS) of 1.10e-5 is observed between the output signal obtained from PL-PS codesign and PS alone implementation. The results show that the proposed hardware architecture for covariance calculation and eigenvalues and eigenvectors extraction performs better than the software version.

To check the feasibility and performance impact of the proposed approach on different FPGAs, we replicated the entire approach on the larger FPGA, i.e., ZCU104; the results indicate the reduction of hardware resources with improved performance. The hardware implementation of the algorithm on ZCU104 consumes 33% of BRAM, 6% of DSP, 11% of FF and 19% of LUT, and the total execution time is reported as 15.26 sec with both PL and PS in place. On PS alone, the algorithm requires 84.1 sec compared to 222.3 sec in PYNQ-Z2. However, the PS power consumption of the ZCU104 board is higher compared to PYNQ-Z2. ZCU104 consumes 2.63 watts compared to 1.25 watts on PYNQ-Z2. The compactness and power consumption criteria for the proposed algorithm of PYNQ-Z2 are best fit for portable and remote applications with limited power resources. However, for a larger signal reconstruction, ZCU104 is helpful because of PL RAM.

$$FOM = [(LUT + FF) * 8] + [(BRAM + DSP) * 16] \tag{9}$$

Based on the BRAM, DSP slice, FF, and LUTs counts. The Figure of Merit (FoM) values are calculated for all the proposed approaches using (9), and corresponding values are shown in Table III. As the metric of resource utility, the FOM values of pipeline-based SSA on PYNQ-Z2 are higher than that of fixed-point implementation of the algorithm.

Table IV depicts the comparison between existing and proposed principal component extraction methods. The proposed approach is implemented on 480 samples of synthesized NQR signals. Principal component extraction is an intermediate step used in SSA to identify dominant principal components in a noisy signal.

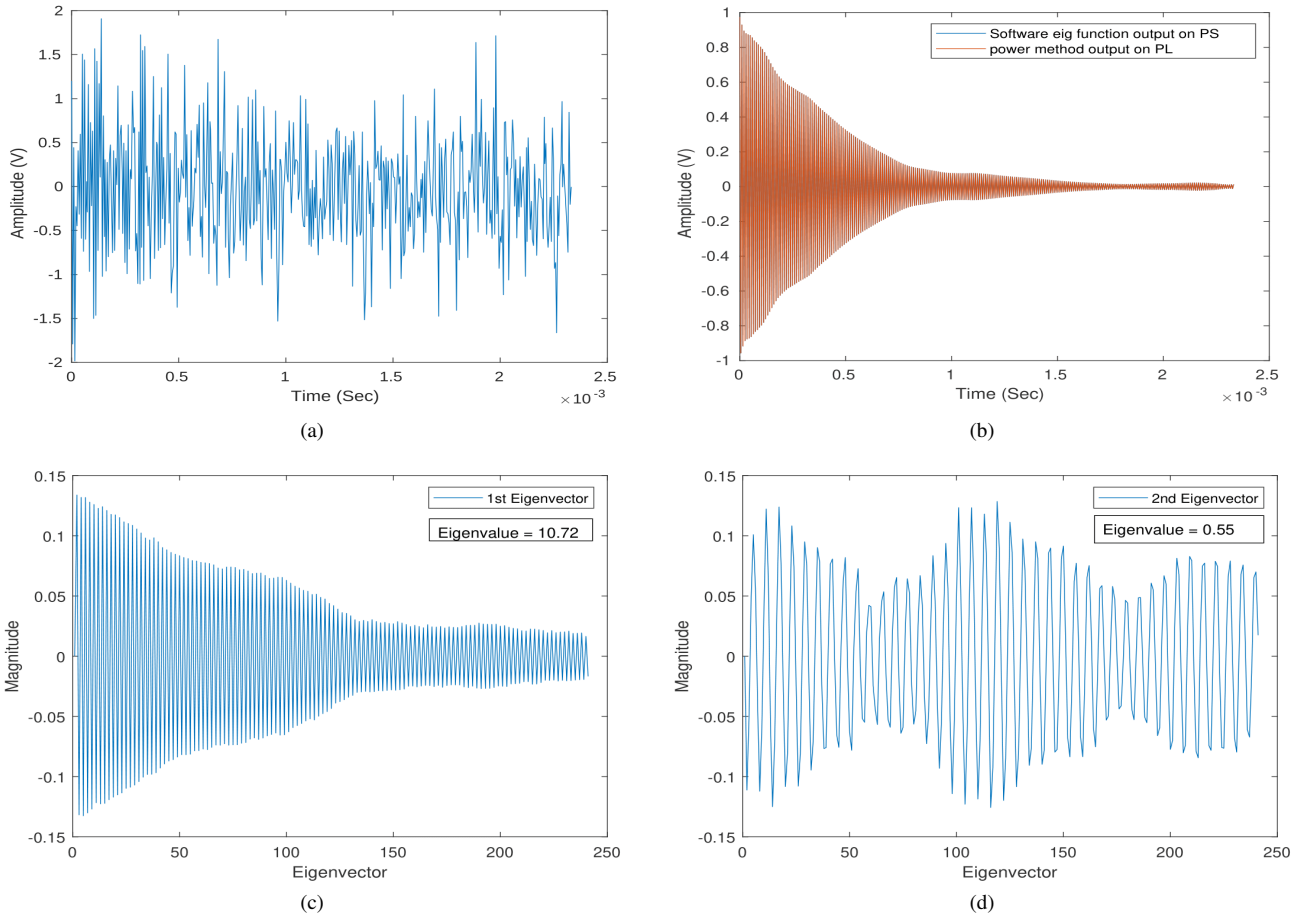

Fig. 5. (a) Noisy NQR signal (SNR=0 dB), (b) Denoised output signal (SNR=24.46 dB),(c) 1st Dominant eigenvalue and corresponding eigenvector,(d) 2nd Dominant eigenvalue and corresponding eigenvector.

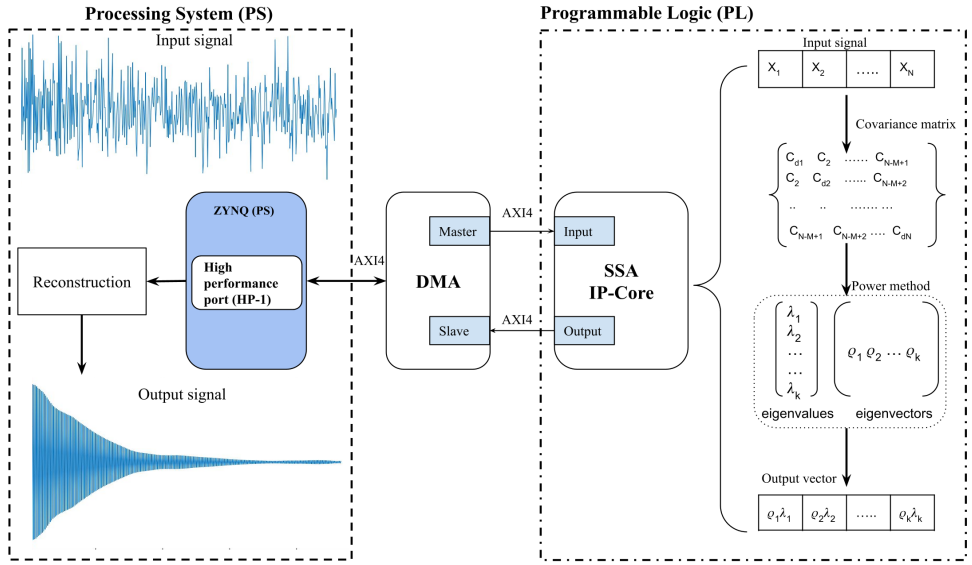

Fig. 6. Hardware design.

The SVD-based principal component extraction methods are faster than the proposed method. However, the proposed work contains overhead modules, such as matrix construction, which affects the overall speed. Furthermore, FoM of literature works [22], [23] shows that they consumed 2.1X and 3.4X higher resources than the proposed work.

In the future, we will work on a hybrid solution combining the proposed work and the SVD-based approach to extract principal components more effectively.

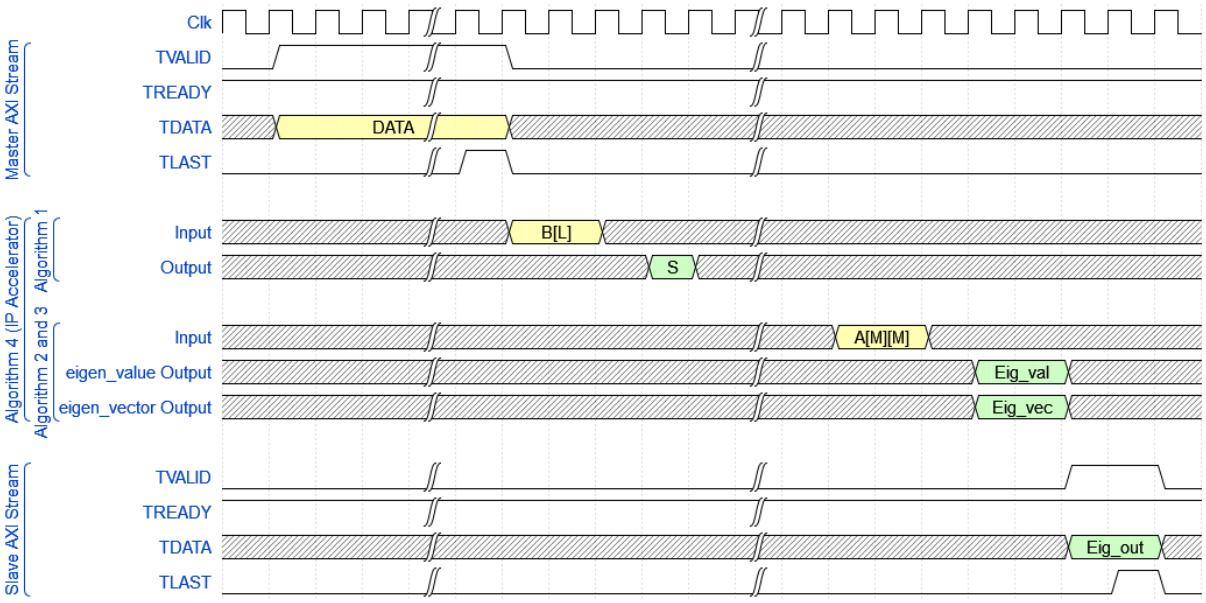

Fig. 7. Implementation timing diagram

TABLE I
RESOURCES UTILIZED IN COMPUTING SSA USING SEQUENTIAL AND PIPELINE APPROACHES

Resources	**Sequential Approach**	**Pipeline Approach**						
Signal Sample Length	**128**	**256**	**480**	**512**	**128**	**256**	**480**	**512**
BRAM_18K (%)	32	50	97	120*	33	51	98	121*
DSP48E (%)	42	42	42	42	48	48	54	48
FF (%)	23	23	24	23	36	46	67	68
LUT (%)	33	32	33	34	52	66	100	95
Execution Time (Sec)	5.16	12.82	25.93	-	3.83	8.44	14.36	-
Total Power (Watts)	1.78	1.82	1.98	-	1.94	1.98	2.18	-

* BRAM resources exhausted, not feasible to implement on the FPGA.

TABLE II
COMPARISON BETWEEN DIFFERENT INPUT SIGNALS IN TERMS OF SNR AND EXECUTION TIME

Input signal SNR (dB)	**Output signal SNR (dB)**	**Floating point sequential approach execution time (sec)**	**Floating point pipeline approach execution time (sec)**	**Fixed point pipeline approach execution time(sec)**	**Floating point PS implementation of the algorithm (sec)**	**Acceleration factor (PS implementation /Floating point pipeline implementation)**
-5	20.90	33.79	17.72	18.10	226.56	12.78x
0	24.46	25.93	14.36	14.99	222.31	15.48x
5	29.51	24.60	13.90	14.69	221.70	15.94x

TABLE III
IMPLEMENTATION OF SSA ON PL WITH DIFFERENT DATA TYPES

Approach	**RMS error w.r.t PS output**	**FOM***	**Static Power (Watts)**	**Dynamic Power (Watts)**
Floating point on PL (Sequential)	$1.10e^{-5}$	357328	0.16	1.82
Floating point on PL (Pipeline)	$1.10e^{-5}$	1006688	0.17	2.01
Fixed point on PL	$1.18e^{-5}$	706272	0.16	1.95

*Figure Of Merit (FOM) refer to (9).

TABLE IV
COMPARISON BETWEEN EXISTING AND PROPOSED PRINCIPAL COMPONENT EXTRACTION METHODS

Work	Platform	Matrix size	Execution time (sec)	FOM
[21] SVD on Reconfigurable System	Spartan-3E XC3S500E (50 MHZ)	40x40	0.48	73976
[22] Scalable FPGA Engine for Singular Value Decomposition	ZC706 (150 MHz)	200x200	0.01	2127120
[23] BCV Jacobi Algorithm SVD Solver	XC7V690T-3FFG1761 (200 MHz)	256x256	0.01	3425944
[Proposed work] SSA on NQR signal	PYNQ-Z2 (100 MHz)	240x240	14.36	1006688

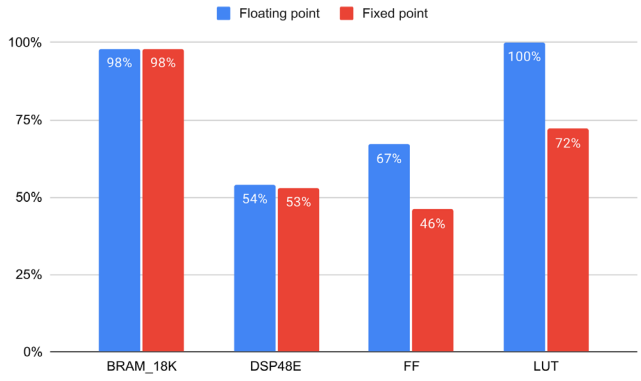

Fig. 8. Floating point and fixed point resource utility chart.

VI. Conclusion

Computationally effective Hankel matrix multiplication and desired eigenvalues and eigenvectors extraction are proposed for denoising the NQR signal. Overall, an optimal solution for implementing the SSA algorithm on the PYNQ-Z2 FPGA board is presented in this paper. An acceleration factor of 15.48x is observed while implementing the SSA algorithm on PL for covariance matrix calculation and eigenvalues and eigenvectors extraction, and PS for performing the signal reconstruction portion of the algorithm. Further, fixed-point implementation results suggest a reduction in the resource utility of the algorithm.

Acknowledgement

This work is part of the Board of Research in Nuclear Sciences (BRNS) funded project 34/14/16/2018-BRNS. The authors thank the Bhabha Atomic Research Centre (BARC) for supporting this work.

References

[1] K. Furuta and J. Ishikawa, "Anti-personnel landmine detection for humanitarian demining," *Springer*, 2009, DOI:10.1007/978-1-84882-346-4.

[2] V. S. K. Kokkiligadda *et al.*, "FPGA-based hardware accelerator for matrix inversion," *SN Comput. Sci.*, vol. 4, no. 2, p. 147, 2023, DOI:10.1007/s42979-022-01542-x.

[3] V. S. K. Kumar and S. L. Sabat, "Hardware-efficient implementation of principal component analysis using high-level synthesis," in *Proc. IEEE Int. Conf. Electron. Comput. Commun. Technol. (CONECCT)*, pp. 1–5, 2024, DOI:10.1109/CONECCT62155.2024.10677202.

[4] W. Qian, Z. Zhu, C. Zhu, W. Luo, and Y. Zhu, "Efficient deployment of single shot multibox detector network on FPGAs," *Integration*, vol. 99, p. 102255, 2024, DOI:10.1016/j.vlsi.2024.102255.

[5] X. Wang, J. Cheng, F. Chang, L. Zhu, H. Chang, and K. Mei, "A bandwidth enhancement method of VTA based on paralleled memory access design," *Integration*, vol. 94, p. 102102, 2024, DOI:10.2139/ssrn.4521941.

[6] C. Chen, F. Zhang, S. Bhunia, and S. Mandal, "Broadband quantitative NQR for authentication of vitamins and dietary supplements," *J. Magn. Reson.*, vol. 278, pp. 67–79, 2017, DOI:10.1016/j.jmr.2017.03.011.

[7] K.T. Varma, R. Peesapati, C.V.R. Rao, A. K. Rajarajan, M. Dixit, and G. Joshi, "A Novel SSA-NLLSF Approach for Denoising NQR Signals," *Appl. Magn. Reson.*, vol. 52, no. 11, pp. 1601–1613, Aug. 2021, DOI:10.1007/s00723-021-01415-1.

[8] W.-P. Huang, B. P. Y. Kwan, W. Ding, B. Min, R. C. C. Cheung, L. Qi, and H. Yan, "High performance hardware architecture for singular spectrum analysis of Hankel tensors," *Microprocess. Microsyst.*, vol. 64, pp. 120–127, 2019, DOI:10.1016/j.micpro.2018.10.004.

[9] W. Byun, Y. Ku, J.-H. Kim, and H.C. Kim, "Fast auditory evoked potential extraction with real-time singular spectrum analysis," *Electron. Lett.*, vol. 53, no. 16, pp. 1094–1096, 2017, DOI: 10.1049/el.2017.1425.

[10] G. Beliakov, "On fast matrix-vector multiplication with a Hankel matrix in multiprecision arithmetics," *arXiv:1402.5287*, 2014, DOI:10.48550/arXiv.1402.5287.

[11] M. A. Mansoori and M. R. Casu, "High-level design of a flexible PCA hardware accelerator using a new block-streaming method," *Electronics*, vol. 9, no. 3, p. 449, 2020, DOI:10.3390/electronics9030449.

[12] A. Karimova, "FPGA Implementation of PCA Dimensionality Reduction Technique," Master's thesis, NTNU, 2018.

[13] D. Claessen and A. Groth, "A beginner's guide to SSA," CERES, Ecole Normale Supérieure, Paris, France, 2002.

[14] A. K. Maddirala and R. A. Shaik, "Separation of sources from single-channel EEG signals using independent component analysis," *IEEE Trans. Instrum. Meas.*, vol. 67, no. 2, pp. 382–393, 2017, DOI:10.1109/TIM.2017.2775358.

[15] J. R. K. K. Dabbakuti, R. Peesapati, M. Yarrakula, K. K. Anumandla, and S. V. Madduri, "Implementation of storm-time ionospheric forecasting algorithm using SSA-ANN model," *IET Radar Sonar Navig.*, vol. 14, no. 8, pp. 1249–1255, 2020, DOI:10.1049/iet-rsn.2019.0551.

[16] H. Hassani, "Singular spectrum analysis: methodology and comparison," Cardiff University and Central Bank of the Islamic Republic of Iran, 2007, DOI:10.6339/JDS.2007.05(2).396.

[17] J. B. Elsner and A. A. Tsonis, *Singular spectrum analysis: A new tool in time series analysis*. Springer, 2013, DOI:10.1007/978-3-642-34913-3.

[18] S. Andrilli and D. Hecker, "Chapter 9 - Numerical methods," in *Elementary Linear Algebra*, 4th ed. Boston, MA, USA: Academic Press, pp. 587–643, 2010, DOI: 9780123747518, 0123747511.

[19] A. Blum, J. Hopcroft, and R. Kannan, "Foundations of data science," *Vorabversion eines Lehrbuchs*, vol. 5, p. 5, 2016, DOI:10.1017/9781108755528.

[20] W. Ford, "Chapter 18 - The algebraic eigenvalue problem," in *Numer. Linear Algebra Appl.*, Academic Press, pp. 379–438, 2015, DOI:10.1016/B978-0-12-394435-1.00018-1.

[21] R. Mohanty *et al.*, "Design and performance analysis of fixed-point Jacobi SVD algorithm on reconfigurable system," *IERI Procedia*, vol. 7, pp. 21–27, 2014, DOI:10.1016/j.ieri.2014.08.005.

[22] Y. Wang, J.-J. Lee, Y. Ding, and P. Li, "A scalable FPGA engine for parallel acceleration of singular value decomposition," in *Proc. Int. Symp. Qual. Electron. Des. (ISQED)*, pp. 370–376, 2020, DOI: 10.1109/ISQED48828.2020.9137055.

[23] T. Hu *et al.*, "A novel fully hardware-implemented SVD solver based on ultra-parallel BCV Jacobi algorithm," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 12, pp. 5114–5118, 2022, DOI: 10.1109/TCSII.2022.3200750.