



**SPATIAL METHODS FOR DIRECTION OF ARRIVAL ESTIMATION AND
HARDWARE IMPLEMENTATION**

GAMZE TAŞYÜREK

MAY 2020

SPATIAL METHODS FOR DIRECTION OF ARRIVAL ESTIMATION
AND
HARDWARE IMPLEMENTATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES OF
ÇANKAYA UNIVERSITY

BY
GAMZE TAŞYÜREK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF
MASTER OF SCIENCE
IN
ELECTRONIC AND COMMUNICATION ENGINEERING
DEPARTMENT


MAY 2020

STATEMENT OF NON-PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: Gamze TAŞYÜREK

Signature

: 

Date

: 10.07.2020

ABSTRACT

SPATIAL METHODS FOR DIRECTION OF ARRIVAL ESTIMATION AND HARDWARE IMPLEMENTATION

TAŞYÜREK, Gamze

M.Sc., Department of Electronic and Communication Engineering

Supervisor: Assoc. Prof. Dr. Orhan GAZİ

May 2020, 102 pages

In this thesis, spatial methods for direction of arrival estimation and its hardware implementation are discussed. Estimation of the direction of arrival (DOA) of signals are widely used in different fields such as radar, sonar, acoustics, astronomy, and communications technologies. Spatial spectrum focuses on researching the spatial characteristics of the signal and the direction of the source. It displays signal propagation across all directions to the receiver. Therefore, if the spatial range of the signal is detected, then DOA can be found. This technology is very essential in signal processing, and it has expanded rapidly in recent years in particular finding the DOA of multiple signal sources.

In this context, many algorithms have been used and have made great accomplishments over the last few years. In this thesis, firstly, the Capon beamforming and conventional beamforming based on ULA arrangement are simulated in MATLAB. The effects of the number of sensors, the distance between sensors, the number of samples, and the effect of SNR value were examined. The high number of sensors, high SNR values and the high number of samples increased the resolution and accuracy, while the distance between the sensors was chosen more

than half of the wavelength, which led to the prediction of false angles. So it is observed that it would be best to keep the spacing at half of the wavelength. After MATLAB simulations, conventional beamforming is implemented in VHDL which is a hardware description language. For the VHDL implementation, signed fixed point numbers are used. The DOA estimation is implemented in VHDL for ULA. According to the simulation results, the VHDL algorithm achieved the angle values with a small margin of error.

Keywords: DOA Estimation, Spatial Spectrum Methods, Beamforming, Array Signal Processing, VHDL, FPGA, Fixed Point Numbers, Hardware Implementation



ÖZ

SİNYALLERİN GELİŞ AÇILARININ MEKASAL YÖNTEMLERLE TAHMİN EDİLMESİ VE DONANIM UYGULAMALARININ YAPILMASI

TAŞYÜREK, Gamze

Yüksek Lisans, Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Tez Yöneticisi: Doç. Dr. Orhan GAZİ

Mayıs 2020, 102 sayfa

Bu tezde varış tahmin yönü için mekânsal yöntemler ve donanım uygulamaları çalışılmıştır. Sinyallerin varış yönünün tahmini, radar, sonar, akustik, astronomi ve iletişim teknolojileri gibi farklı alanlarda yaygın olarak kullanılmaktadır. Mekânsal spektrum, sinyalin mekânsal özelliklerini ve sinyal kaynağının yönünü araştırmaya odaklanır. Alıcıya tüm yönlerden gelen sinyal yayılımını gösterir. Bu nedenle, sinyalin mekânsal aralığı tespit edilirse, varış tahmin yönü bulunabilir. Bu teknoloji sinyal işlemede çok önemlidir ve son yıllarda özellikle çok sayıda sinyal kaynağının varış tahmin yönünü bulmak için hızla genişlemiştir.

Bu bağlamda, birçok algoritma kullanılmış ve son birkaç yılda büyük başarılar elde etmiştir. Bu çalışmada, ilk olarak, düzgün doğrusal dizilerin yerleşimi kullanılarak Capon hüzmeye şekillendirme ve geleneksel hüzmeye şekillendirme teknikleri MATLAB ortamında benzetilmiştir. Sensör sayısının, sensörler arasındaki mesafenin, örnekleme sayısının ve SNR değerinin etkisi incelenmiştir. Yüksek sensör sayısı, yüksek SNR değerleri ve yüksek örnekleme sayısı çözünürlüğü ve doğruluğu arttırırken, sensörler arasındaki mesafe dalga boyunun yarısından fazlası

seçildiğinde, yanlış açıların tahminine yol açtı. Bu nedenle, sensörler arasındaki mesafeyi dalga boyunun yarısında tutmanın en iyi olduğu gözlemlenmiştir. MATLAB benzetimlerinden sonra bir donanım tanımlama dili olan VHDL'de geleneksel hüzme şekillendirme yöntemi gerçekleştirilmiştir. Gerçekleşim esnasında sayıların gösterimi için 'signed fixed-point' gösterim formatı kullanılmıştır. Gerçekleştirim için VIVADO platformu kullanılmıştır. VIVADO benzetim sonuçlarına göre, VHDL ile yazılan donanım gerçekleştirimi programları küçük bir hata payı ile açı geliş değerlerini hesaplamıştır.

Anahtar Kelimeler: Varış Tahmin Yönü Kestirimi, Uzamsal Spektrum Yöntemleri, Hüzme Şekillendirme, Dizilim Sinyal İşleme, VHDL programlama, FPGA, Fixed Point Sayılar, DonanımUygulamaları

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis advisor, Assoc. Prof. Dr. Orhan GAZI, for his help, valuable supervision and useful advice for my research, without him, which much of this work would not have been possible.

I wish to thank the examining committee for their kindness during the presentation of this thesis.

I would like to thank TÜBİTAK İLTAREN, which offers the opportunity to benefit from all kinds of opportunities by supporting my thesis project.

I would like to present my gratitude to my colleagues.

Finally, I would like to express my deep gratitude to my family for their endless and continuous encourage and support throughout these years. They always stood beside me all the way. All acquirements, which I have achieved until now, have been possible under favor of their reliance, guidance, limitless and endless support.

TABLE OF CONTENTS

STATEMENT OF NON-PLAGIARISM	iii
ABSTRACT	iv
ÖZ.....	vi
ACKNOWLEDGEMENTS	viii
TABLE OF CONTENTS	ix
LIST OF TABLES	xi
LIST OF FIGURES	xii
LIST OF ABBREVIATIONS	xvi
INTRODUCTION	1
1.1. Background and significance of the study	1
1.2. Problem Statement	2
1.3. Objectives.....	2
1.4. Scope of the Work.....	2
1.5. Organization of Thesis	3
THEORETICAL BACKGROUND ON DOA ESTIMATION	5
2.1. Introduction	5
2.2. Historical development of antenna arrays.....	5
2.3. Structural analysis of spatial spectrum estimation system.....	6
2.4. Overview of the common methods for development of DOA estimation	8
2.5. Parameters affecting DOA estimation	10
2.5.1. Number of array elements.....	10
2.5.2. Number of snapshots.....	12
2.5.3. SNR.....	12
2.5.4. Element spacing	13
METHODOLOGY	16
3.1. Introduction to System Model.....	16
3.2. Assumptions.....	16

3.2.1. Point Source Assumption	16
3.2.2. Far Field Assumption	17
3.2.3. Narrowband Signal Assumption	17
3.2.4. Correlation of Sources.....	17
3.3. Signal Model	18
3.4. Algorithms	25
3.4.1. Non-Subspace Algorithms	26
SIMULATION RESULTS AND DISCUSSION	32
4.1. Introduction	32
4.2. Experiment 1: Basic simulations of conventional beamforming and Capon beamforming	32
4.3. Experiment 2: Relationship between DOA and the number of sensors.....	34
4.4. Experiment 3: Relationship between DOA and the number of snapshots	36
4.5. Experiment 4: Relationship between DOA and the element spacing	38
4.6. Experiment 5: Relationship between DOA and SNR	40
VHDL IMPLEMENTATION of CONVENTIONAL BEAMFORMING	42
5.1 Introduction to VHDL Implementation	42
5.2. VHDL Implementation of Uniform Linear Array Algorithm.....	46
5.2.1. VHDL Implementation of Sinusoidal Wave.....	47
5.2.2. VHDL Generation of Direction Matrix A	49
5.2.3. VHDL Generation of Output Matrix Y	50
5.3. VHDL Implementation of Beamforming Algorithm	54
5.3.1. VHDL Generation of Complex Conjugate Transpose of Matrix Y	55
5.3.2. VHDL Generation of Sample Covariance Matrix R	59
5.3.3. VHDL Implementation of Finding Arrival Angle	69
CONCLUSION	73
REFERENCES.....	75
APPENDICES	81
APPENDIX A: MATLAB codes for Conventional Beamforming.....	82
APPENDIX B: VHDL codes for Conventinal Beamforming.....	85

LIST OF TABLES

Table 1 Comparison Between Beamforming Techniques For Doa Estimation[1]....	25
Table 2 Complex Number Representation	44
Table 3 One Period Of Sine Wave With 64 Samples.....	48
Table 4 Complex A Matrix Values	49
Table 5 Complex Matrix Y Values Through Column1 To Column4	50
Table 6 Complex Matrix Y Values, From Column-5 To Column-8.....	51
Table 7 Complex Matrix Y Values, From Column-9 To Column-12.....	51
Table 8 Complex Y Matrix Values, From Column-13 To Column-16.....	52
Table 9 Complex Y Transpose Matrix Values Column1 To Column4.....	56
Table 10 Complex Y Transpose Matrix Values Column5 To Column8.....	57
Table 11 Covariance Matrix R Values, From Column-1 To Column-4	61
Table 12 Covariance Matrix R Values, From Column-5 To Column-8	61
Table 13 Sample Covariance Matrix R Values Column1 To Column4.....	65
Table 14 Sample Covariance Matrix R Values, From Column-5 To Column-8	66
Table 15 Diagonal Of Phi Matrix	70

LIST OF FIGURES

Figure 1 The System Structure For Doa Spatial Spectrum Estimation.....	7
Figure 2 The Array Factor For 4 And 8 Element. Spacing Between Elements Is 0.5λ [51]	11
Figure 3 The Array Factor For 16 And 32 Element. Spacing Between Elements Is 0.5λ [51]	12
Figure 4 The Array Factors For Arrays With Element Spacing Of 0.25λ And 0.5λ [51]	14
Figure 5 The Array Factors For Arrays With Element Spacing Of 1λ And 2λ [51].	15
Figure 6 A Plane Wave Incident On A Uniform Linear Array Of M Uniformly Spaced Sensors	23
Figure 7 Tapped Delay Line Structure Spatial Filter	27
Figure 8 Conventional Beamformer.....	28
Figure 9 Adaptive Beamformer.....	30
Figure 10 Conventional Beamforming With Three Signal Sources	33
Figure 11 Capon Beamforming With Three Signal Sources.....	33
Figure 12 Conventional Beamforming With Different Number Of Sensors	35
Figure 13 Capon Beamforming With Different Number Of Sensors	35
Figure 14 Conventional Beamforming With Different Number Of Snapshots	37
Figure 15 Capon Beamforming With Different Number Of Snapshots	37
Figure 16 Conventional Beamforming With Different Sensor Element Spacing	39
Figure 17 Capon Beamforming With Different Sensor Element Spacing	39
Figure 18 Conventional Beamforming With Different SNR Values.	41
Figure 19 Capon Beamforming With Different SNR Values	41
Figure 20 Vhdl Design Flow For Simulation Level.....	42
Figure 21 Binary Notation Of A Positive Number	43
Figure 22 Binary Notation Of A Negative Number	43
Figure 23 Fixed Point Notation.....	43

Figure 24	Vhdl Syntax Of A Basic Definition Of Complex Number.....	44
Figure 25	Syntax Rule For Vhdl Array Definition	45
Figure 26	Ula Algorithm In Matlab	46
Figure 27	Sine Look Up Table (Lut) Generation With Matlab.....	47
Figure 28	Sine Look Up Table (Lut) Samples	47
Figure 29	Vivado Simulation Of Sine Wave	48
Figure 30	Vivado Simulation Of Complex A Matrix	49
Figure 31	Vivado Generation Of 0 th Row Of Y Matrix	52
Figure 32	Vivado Generation Of 1 th Row Of Y Matrix	53
Figure 33	Vivado Generation Of 7 th Row Of Y Matrix	54
Figure 34	Beamforming Algorithm In Matlab.....	55
Figure 35	Transpose Function In Vhdl	56
Figure 36	Vivado Generation Of 0 th And 1 th Row Of Y' Matrix.....	58
Figure 37	Vivado Generation Of 14 th And 15 th Row Of Y' Matrix.....	59
Figure 38	Vivado Generation Of 0 th And 1 th Row Of R Matrix	62
Figure 39	Vivado Generation Of 2 th And 3 th Row Of R Matrix	63
Figure 40	Vivado Generation Of 4 th And 5 th Row Of R Matrix	64
Figure 41	Vivado Generation Of 6 th And 7 th Row Of R Matrix	65
Figure 42	Vivado Generation Of 0 th And 1 th Row Of R Matrix	66
Figure 43	Vivado Generation Of 2 th And 3 th Row Of R Matrix	67
Figure 44	Vivado Generation Of 4 th And 5 th Row Of R Matrix	68
Figure 45	Vivado Generation Of 6 th And 7 th Row Of R Matrix	69
Figure 46	Vivado Generated Diagonal Matrix, From [0,0] To [15,15]	71
Figure 47	Vivado Generated Diagonal Matrix, From [15,15] To [31,31]	72

LIST OF SYMBOLS

$[\cdot]^*$	Complex conjugate transpose
$[\cdot]^{-1}$	Matrix inversion
$[\cdot]^T$	Transpose
π	Mathematical constant, ratio of a circle's circumference to its diameter
\in	Set membership operator
$e^{[\cdot]}$	Exponential function
Σ	Summation operator
$ \cdot $	Absolute value
$E\{\cdot\}$	Expectation operator
$a(\theta)$	Steering vector as a function of incident angle
\mathbf{A}	Steering matrix
\mathbf{AF}	Array Factor
β_s	Bandwidth of the signal
c	Speed of light
d	Element spacing
f	Frequency
f_s	Sampling frequency
f_c	Carrier frequency
k	Wave number
K	Number of signal sources
M	Number of sensor arrays
$ek(t)$	Noise at k th element
N	Number of elements in an array
r	Radial distance from origin
R	Correlation matrix
R_n	Noise correlation matrix
$sk(t)$	k th signal

$s(t)$	Baseband signal
t	Time variable
$x(t)$	All the signals induced on all elements
$\bar{h}_k(t)$	Impulse response of sensor
$\bar{y}_k(t)$	Output of sensor
σ^2	Noise variance
ω_s	Angular spatial frequency
ω_c	Angular carrier frequency
λ	Wavelength
τ_k	Propagation delay to kth antenna with respect to first



LIST OF ABBREVIATIONS

ADC	Analog to Digital Converter
FPGA	Field Programmable Gate Array
DFT	Discrete Fourier Transform
DOA	Direction of Arrival
DOAE	Direction of Arrival Estimation
DTFT	Discrete Time Fourier Transform
ESPRIT	Estimation of Signal Parameters via Rotational Invariance Technique
FFT	Fast Fourier Transform
LUT	Look Up Table
MUSIC	Multiple Signal Classification
MSB	Most Significant Bit
MVDR	Minimum Variance Distortionless Response
NLA	Non-Uniform Linear Array
SNR	Signal to Noise Ratio
ULA	Uniform Linear Array
VHDL	Very High –Speed Integrated Circuit Hardware Description Language

CHAPTER 1

INTRODUCTION

1.1. Background and significance of the study

Direction of arrival estimation is used to estimate the angle at which an acoustic or electromagnetic wave arrives to an array of sensor or antennas [1].

Over the past decades, DOA estimation has been an important research topic in signal processing. Practically, it has been applied in the field of sonar [2], radar [3], navigation, radio astronomy, wireless communications [4, 5], tracking of different objects internet, broadcasting, wireless sensor networks [6] and imaging [7]. Because of its flexible and convenient features, DOAE has contributed a lot in these fields. Array processing and DOA are not very different subjects from each other, they should be considered together, and DOA should be supported using array antennas instead of single antenna in order to achieve high resolution and better performance. This array structure over a single antenna allows spatial samplings of received signal. There are many different super resolution algorithms for finding arrival angle. Spectral estimation, beamforming, Capon, Bartlett, Min-norm, ESPRIT, MUSIC, Root-MUSIC, Maximum Likelihood are some examples of DOA algorithms [8, 9].

Every method has some advantages and weaknesses. The most suitable methods should be selected for the desired purpose. For example, Capon algorithm need inverse matrix calculation [10]. MUSIC algorithm needs a large number of calculations while searching the spectral angle [11]. Conventional beamforming is inefficient when multiple signal sources are used [12]. Geometry of the array structure also affects the resolution and applicability of the methods. Based on last updates and experiments, non-uniform linear array has some advantages over linear uniform array [13, 14]. In addition, circular or rectangular array forms increases dimensionality of DOA.

Although more than one-dimensional array structures increase the resolution, they need large computations [15].

Having meaningful theoretical results does not mean the suitability of algorithm for practical implementations.

DOA is an important research topic that keeps its popularity for the up to date studies in signal processing field.

1.2. Problem Statement

The problem is considered as estimating the angular location of K signal sources using a sensor array consisting of M sensors that are equally spaced with distance d , which is assumed a half of the wavelength. The source signals are assumed uncorrelated and narrowband. In addition, white Gaussian noise located in the far field region with regard to physical size of the antennas is also uncorrelated that can be explained by statistical independency between each array noise. [16] The direction of arrivals of more than one source located at $\theta_1, \theta_2, \dots, \theta_k$ can be found by determining the peaks of spatial spectrum.

1.3. Objectives

The main objective of this research is the implementation of beamforming and direction of arrival (DOA) estimation measurement using uniform linear array (ULA). It is shown in [17] that the evaluations of techniques for implementation are based on the factors that affect the resolution and accuracy of the DOA estimation and these factors depend on the number of signal sources, number of array elements, number of snapshots, element spacing and signal to noise ratio [18].

Determining the desired signal location can be used in order to set up measurement platforms, which contributes high robustness to the system [19].

1.4. Scope of the Work

The scope of this research work is involves the formulation of the mathematical model of the direction of arrival estimation and simulation of uniform linear array, conventional beamforming, and Capon estimation in MATLAB platform. After

MATLAB modelling and having obtained the simulation results, the conventional beamforming is implemented in VHDL using Vivado Design Suite. VHDL is used to describe the structural and behavioral characteristics of digital logic circuits, it allows the system to be modelled and simulated before translating the design into real hardware by the gates, and wires [20]. In addition, VHDL allows the use of not only sequential statements but also concurrent statements that can be explained by programming blocks simultaneously that makes the design faster. The testbenches, which are commonly called collection of simulation models, is written in order to get information about verification of functionality of the design and simulated by Vivado Simulator [21].

1.5. Organization of Thesis

The structure of the thesis is as follows:

- **Chapter-2** presents the theoretical background of antenna arrays, performance of uniform linear array antenna structure with some spatial spectrum estimation system. This chapter also includes basic principles of beamforming and factors affecting DOA estimation.
- **Chapter-3** includes the methods used in this thesis study. The mathematical model of DOA estimation for ULA, and description of beamforming concept are explained.
- **Chapter-4** describes simulation and discussion of the results, obtained using MATLAB, the beamforming and direction arrival estimation by varying input parameters such as number of signal sources, number of snapshots, signal to noise ratio and number of array elements. All simulations are performed in Matlab2017b platform.
- **Chapter-5** describes VHDL implementation of conventional beamforming and the simulation results. All simulations are performed in Vivado Simulator on Windows operating system. Resolution and accuracy performance of the design is discussed.

- **Chapter-6** summarizes the overall study in details and conclusion, recommendation and discussions are highlighted.



CHAPTER 2

THEORETICAL BACKGROUND ON DOA ESTIMATION

2.1. Introduction

Array processing is an important research area, which deals with the processing of the signals received from an antenna array and extracting information about signal parameters [22].

The goal of array signal processing is not only to predict the values of parameters by using available temporal and spatial information but also to alleviate the effects of noise and interference. A sensor array is considered as a group of sensors conveyed in a certain geometry pattern. In contrast to traditional single sensor, an array provides some advantages such that it increases the antenna gain in the direction of the signal sources and it suppresses the gain in the directions of interferences and noise [22]. The ability of dealing with interference and strengthening signal gain results in high signal-to-noise ratio (SNR). Detection of direction and distance of impinging signal sources is another advantage that explains why array signal processing theory can be considered as an important research field. DOA estimation can be considered as one of the most important research topics of the array processing. In this section, antenna arrays, spatial spectrum, some common methods for DOA and the factor affecting DOA are discussed.

2.2. Historical development of antenna arrays

An antenna array is a geometric configuration of a series of two or more antenna components used to transmit or receive electromagnetic waves [23]. These antennas are regularly spaced to get unique amplitude and phase values. The advances in electronics, antenna theory, signal processing and information theory improved the ability of wireless communication systems [24].

In 1940, for military applications, the concept of an antenna array was introduced [25].

This technology was significant in wireless communications because it improved the antenna reception and transmission characteristics used in these structures. The array also helped the antenna system to be electronically guided to transmit or receive data mainly from a specific direction without moving the structure mechanically.

As the signal processing area evolves, arrays can be used to receive energy from a desired direction while rejecting information or destroying energy in undesirable directions.

As a result, the arrays could be used to minimize unintended interference such as jamming. Radiation from other sources not intended for the system is the main source of the unintended interference [26].

In addition, the developments in signal processing led to the idea of adaptive antenna arrays.

Depending on the operating environment; these arrays adjust their radiation or reception pattern [27]. Although there has been a great deal of development and work on signal processing issues, physical geometry has not received sufficient attention. This situation can be explained due to the mathematical complexity of optimizing location of the elements for different situations. Optimization of array geometry is expected to support the continuous development performance of desired systems [28].

2.3. Structural analysis of spatial spectrum estimation system

Spatial spectrum estimation is a specific estimation technology to obtain space parameters of a signal by using space arrays [29]. The spatial spectrum can be expressed in terms of three dimensions, which refer the expansion of the temporal spectrum. The entire spectrum should include three parts, which are incident signal space, spatial array receiver and estimation of parameters. The fundamental assumptions for propagation model and medium channel depend on the variations in each of these three parts [30]. The space can be splitted into three suitable spaces which can be listed as target stage, observation stage, and estimation stage, respectively.

In target stage, parameters of multiple signal sources and complex environment are available. In this stage, it is observed that some of the spatial filtering techniques are

used to estimate the indefinite parameters of the signals which occurs due to the complex environment.

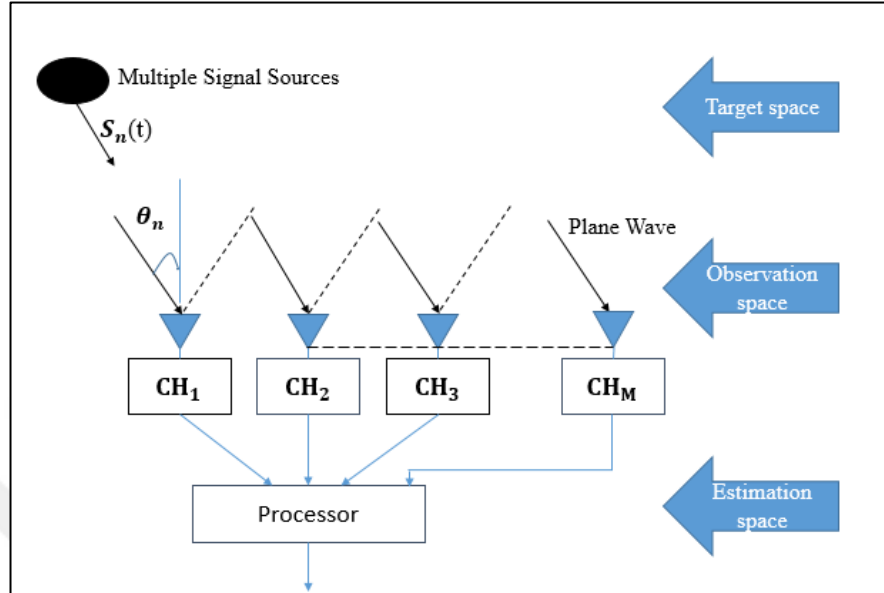


Figure 1 The system structure for DOA spatial spectrum estimation

Observation stage, which follows the target stage, receives the signal radiations from target space. Due to the existence of complex environment in target stage, the received data may include not only some signal features but also some complex space environment features. Elevation angle, azimuth angle, distance and polarization can be examples of signal features while miscellaneous waves, noise and interference can be examples of complex space environment features. Furthermore, the collected data may include some distortion due to some space array element characteristics such as mutual coupling effects and inconsistent frequency bands.

The observation stage, which follows the target stage, receives signal radiations from the target space. According to the complexity of the environment of the target stage, the signal received contains some characteristics such as the distance, the azimuth angle, the elevation angle, the polarization, etc. Complex space environment characteristics include the miscellaneous waves, the interference, noise, etc. This stage can be considered as multidimensional, since it is supposed that the system receives data from more than one channel. However, only one channel is usually used in time domain processing.

Final stage is the estimation stage where the estimation of spatial spectrum by applying array correction and spatial filtering techniques in order to extract signal parameters from complex environment is performed. There is no doubt that estimation stage can be considered as a reconstruction of the target stage. Mutual coupling of arrays, complexity of environment, different channels can affect the precision of reconstruction.

According to all information discussed so far, the energy distribution of signals from all spatial directions can be expressed as spatial spectrum. If the spatial spectrum of signals is obtained, the arrival path of the signals can also be identified. Therefore, DOA estimation is also known as spatial spectrum estimation [31].

2.4. Overview of the common methods for development of DOA estimation

Direction of arrival estimation is an important research field in signal processing. At the beginning, linear spectrum estimation, which mainly involves periodogram method based on Fourier Transform, is used to find arrival angle. However, due to the effect of the Rayleigh fading [32], it is not possible to obtain high-resolution performance, so it can not achieve satisfactory performance. Then, the statistical analysis of estimation of the maximum likelihood spectrum with a good accuracy and robust character took attention of researchers. However, this estimation method needs high-dimensional parameter space that means large amounts of computation to be done. Therefore, it is not practical [33, 35].

Estimation of maximum entropy method was considered by Burg in 1967. This method consists of some models, which are AR (Autoregressive model, MA (Moving Average Model), ARMA (Auto-Regressive and Moving Average Model). The common property of these methods is having high resolution. Nevertheless, a great amount of calculation is needed [34-35].

Depending on the eigenvalue decomposition, some spectrum estimations, which are Estimation Signal Parameters via Rotational Invariance Technique (ESPRIT) and Multiple Signal Classification (MUSIC) algorithm, were introduced in 1980's [36-37]. MUSIC was derived from the maximum entropy method based on one-dimensional implementation of it. In addition, it resembles the maximum likelihood method in terms of some features [38-39].

In spite of the fact that, MUSIC has better results and preferred by many researchers, it involves heavy computations.

ESPRIT and its improved versions VIA-ESPRIT, TLS-ESPRIT and GEESE were considered in [40]. The most powerful features of these algorithms are not only having high resolution but also avoiding heavy computations while searching the spectrum. Hence, DOA estimation can be faster in terms of speed. However, as a weakness, ESPRIT algorithm and its derivations need some special array structures in order to find DOA angle, so its applications are rarely seen [40].

Time-domain characteristics of the signals should be considered while processing in spatial domain. Signals can be sampled simultaneously in time domain and spatial domain when sufficient information about signals is available. In order to compensate the insufficient information about spatial domain, two-dimensional array structures should be used to avoid the limitations of array structures and enhance the arithmetic capability while handling the noise.

In real time applications, the existence of colored should be taken into account as well as thermal noise. Higher order cumulant techniques are used in signal processing for DOA estimation in white noise, Gaussian or non-Gaussian spatial colored noise [41]. In array signal processing, when the multiple signals are received by an antenna array, the signal sources may not be fully known, and the transmission channel may be unknown and may vary in time.

For the transmission channel, this unknown feature causes a limitation for high resolution of DOA. Therefore, researchers introduced the concept of blind DOA prediction, which can estimate the properties of the channel under unknown conditions and has wide application possibilities [42].

Discrimination of adaptive blind signaling was studied by Herault–Jutten in 1991 [43]. Since then researchers have recommended many different algorithms. All these algorithms can be applied for DOA estimation. Non-stationary signals, whose characteristics are restricted by variation of time and duration, include many natural and synthetic signals, such as biomedical signals, sound, sonar and radar signals. Taking into account of the non-stationary and nonlinear properties of the actual system, DOA using artificial neural networks is a research field of recent years [44].

In miscellaneous DOA estimation methods based on spatial spectrum estimation, the MUSIC method has a higher stability, better resolution, an acceptable amount of calculation, and array implementations structure. Therefore, MUSIC method is preferred in experimental studies.

2.5. Parameters affecting DOA estimation

Accuracy and resolution of DOA estimation are affected from a number of factors, which depend not only on arriving signal source but also on the actual application environment [45]. Some influential factors are mentioned theoretically in this section and different simulation experiments are considered to observe how accuracy of DOA is affected.

2.5.1. Number of array elements

As mentioned before, array antenna includes two or more sensors used for sending and/or capturing electromagnetic waves. In comparison to the traditional signal orientation antenna, the signals from antennas are combined together and processed to get better performance that can be explained by making an increment of total gain, improvement of the spatial resolution, reducing the interference from a specific direction, and steering the system at the direction of interest [46].

At this point, the total radiation pattern which is the multiplication of the pattern of a single element and array factor should be considered. Array factor is a complex valued function of far-field radiation pattern, obtained for an array of N isotropic radiators located at coordinates \vec{r}_n , and it is calculated as [23]

$$AF(\hat{r}) = \sum_{n=1}^N a_n e^{jk\hat{r} \cdot \vec{r}_n}$$

where

a_n are the complex-valued excitation coefficients, \hat{r} and is the direction unit vector. The AF written in the transmitting mode, $e^{j\omega t}$, whereas in receiving mode, corresponding expression for AF includes a negative sign appearing in the exponential factors, $e^{-j\omega t}$ [47] [48] where it is shown that array factor depends on position, phase and amplitude of every individual element in the array structure while physical

dimensions and electromagnetic characteristics of the radiating element the depend on element pattern [49]. As the number of array elements increase, the number of side lobes increase as well, however, side lobes smaller peak levels which mean that directionality is improved, and interference is reduced. Besides, beams are narrower and gain of the antenna array is better due to the spatial diversity [49].

As a conclusion, if the remaining array parameters are kept same, the greater number of array elements means the better estimation performance [50].

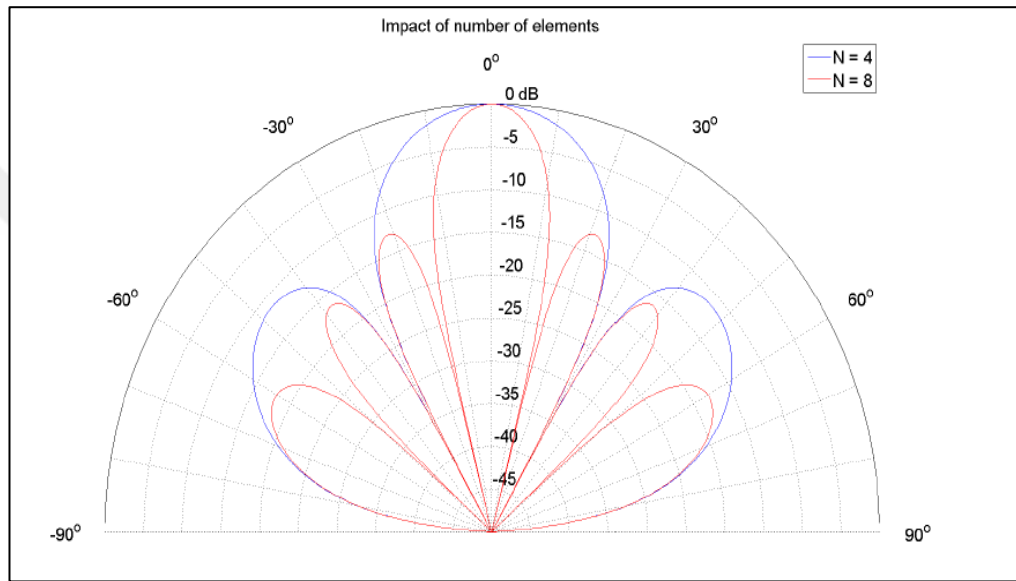


Figure 2 The array factor for 4 and 8 element. Spacing between elements is 0.5λ [51]

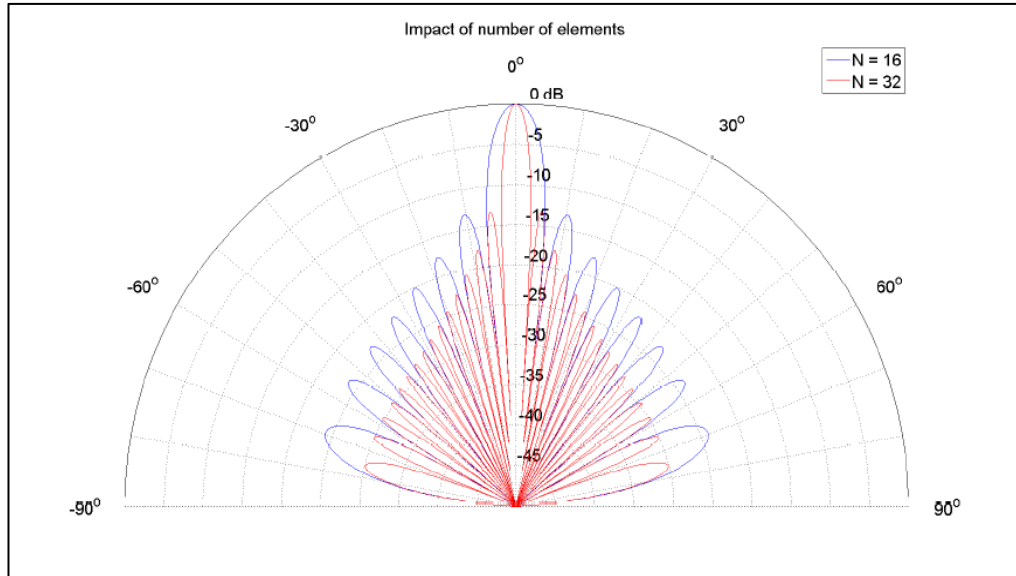


Figure 3 The array factor for 16 and 32 element. Spacing between elements is 0.5λ [51]

Figure 2 and Figure 3 illustrate the impact of number of elements on array factor [51]. In Figure 3, it is seen clearly that array factor gets more side-lobes by increasing the number of elements. However, the side-lobe levels are lower compared to Figure 2. Furthermore, number of nulls is increased, and beams are narrower that they can handle with the interference more easily and this improves directionality.

2.5.2. Number of snapshots

Number of snapshots should be considered in time and frequency domains. Number of samples refers to the number of snapshots in time domain whereas the behavior of Discrete Fourier Transform (DFT) of time frames contains information about the number of snapshots in frequency domain [52]. Frequency-domain analysis shows how the signal energy is distributed over a range of frequencies. If frequency domain includes many components, frequency ranges can be analyzed. As a result, accuracy of DOA can be increased when remaining parameters are kept the same [53].

2.5.3. SNR

SNR is defined as the ratio of signal power to noise power. It is often expressed in decibels and it measures the difference between desired signal power and noise power.

Any signal processing system is affected by noise; whose strength directly influences the system performance. If the noise power increases gradually while signal power remains the same, low SNR is achieved, algorithms will fail to have angular resolution due to the lack of performance [54].

2.5.4. Element spacing

Element spacing, along with the number of array elements, is another factor that affects accuracy of DOA. In order to achieve high spatial discrimination, the aperture of the antenna is arranged to be large in terms of wavelength. As a result, the gain is increased. The relationship between antenna main lobe beamwidth and its physical shape is inversely proportional [46].

If the aperture is decreased, the beamwidth of the main lobe becomes sufficiently wide. Consequently, it becomes too difficult to separate the signals if multiple signals sources fall in the wide main lobe. When spacing is increased, gain of sidelobes will be close to the gain of main lobe. In this case, they are called grating lobes that repeat themselves periodically [48]. There will be an unknown case at the receiver side, since the coming signals may belong to the direction of the grating lobe or direction of the main beam lobe. Also, they may have the same response. In transmission side, the transmitted power can be wasted for the sake of grating lobes since the receiver is on the direction of main beam. Furthermore, this wasted transmitted power can affect other users in terms of strong interference. Therefore, grating lobes should be avoided by reducing element spacing to prevent this ambiguity. However, making elements closer to each other increases their mutual coupling effects. In order to occupy the same aperture, to keep its properties, more number of elements should be used to compensate reduced element spacing [55].

Since using more elements may lead to a higher cost, there should be a tradeoff to choose the element spacing [56-57].

If grating lobes exist, it can be explained in terms of spatial aliasing, which resembles a temporal aliasing. Sampling is used to convert continuous time signal into a discrete time signal using an analog to digital converter (ADC). If the signal is sampled at a rate lower than NYQUIST sampling rate, aliasing exists. As a result, two signals having different frequency intervals could refer to the same discrete signal. Likewise,

the signals received by antennas are sampled spatially in phased arrays. The sampling operation will not be densely enough when the antennas are not far enough from each other. The signals coming from different directions will share the identical steering vector, and it causes an ambiguity in terms of DOAs of the sources and this is called as spatial aliasing [58]. When the spacing is chosen less than half of the wavelength, the performance of beamforming with regard to resolution does not promote. Consequently, the element spacing is generally chosen to be half of the wavelength in order to keep grating lobes away from the main beam lobe and to maximize the aperture. The element spacing is measured considering the wavelength, which depends on frequency. In this thesis, narrowband systems are considered.

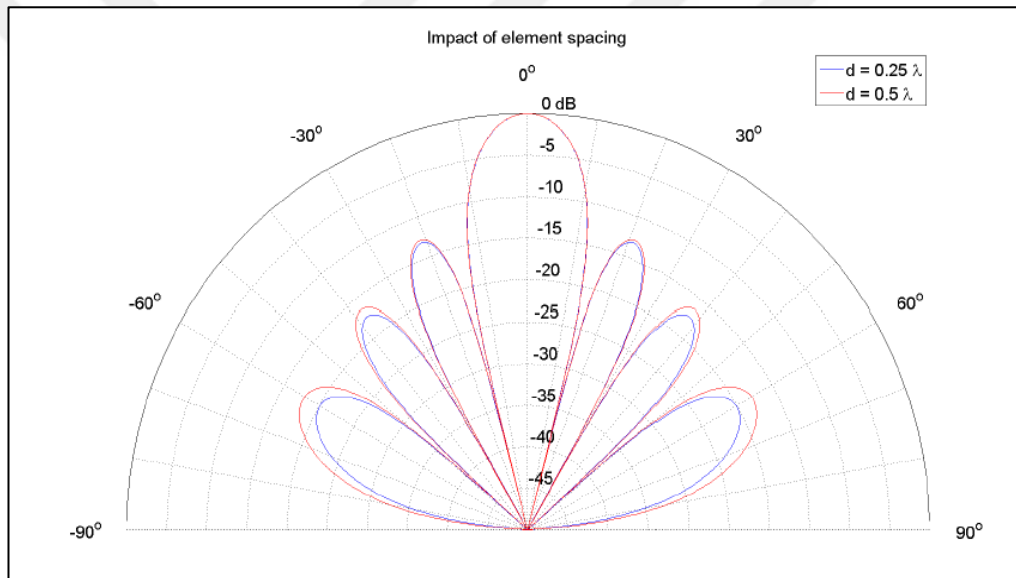


Figure 4 The array factors for arrays with element spacing of 0.25λ and 0.5λ [51]

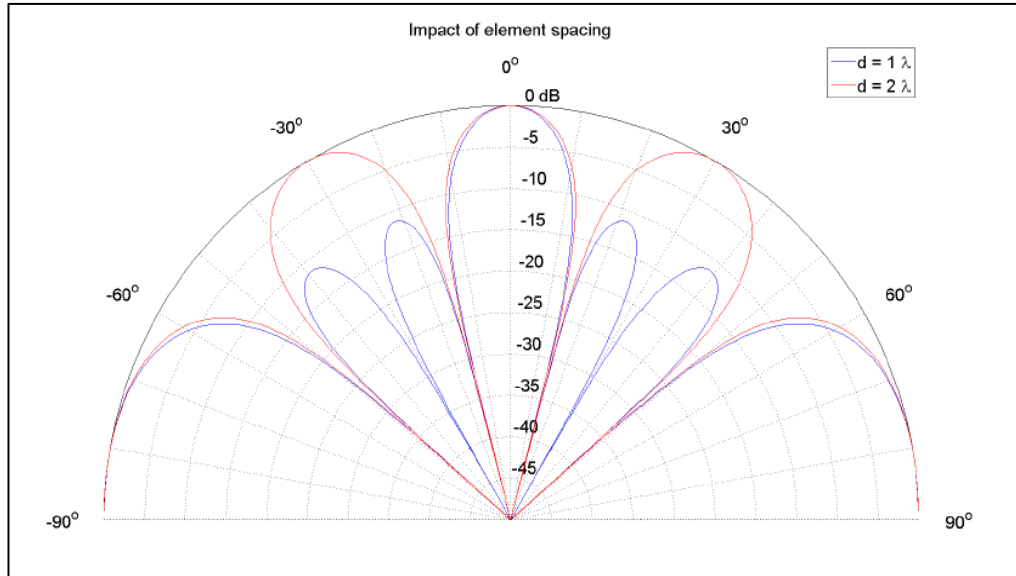


Figure 5 The array factors for arrays with element spacing of 1λ and 2λ [51]

The Figures 4 and 5 are plotted using the formulas given in [51].

Figures 4 and 5 show the effect of the element spacing and the existence of the grating lobes. In both figures, the arrays have the same aperture, which is equal to 4λ , but they have different element spacing [51].

In Figure 3, element spacing is takes as 0.25λ or 0.5λ , and there are no grating lobes, while in Figure 4 element spacing is takes as 1λ or 2λ , and there are 2 grating lobes for 1λ spacing, and 4 grating lobes for 2λ spacing.

CHAPTER 3

METHODOLOGY

3.1. Introduction to System Model

In this chapter, mathematical modeling of the communication system is performed before introducing any type of DOA estimation algorithms. The model is applicable throughout the discussion, and the same notations are used to describe all algorithms.

3.2. Assumptions

There are some important assumptions to be considered for the model. Firstly, a point source is supposed to generate the emitting signal. Second assumption is related to far field region. The signal sources are assumed to be far away from the sensor array so that waves on the array behave like planar waves. Third, a narrowband signal hypothesis, which depends on signal bandwidth, is assumed. The fourth assumption is that sources are not correlated with each other. Lastly, noise existing in the sensor is temporally and spatially uncorrelated.

3.2.1. Point Source Assumption

Estimating the DOA of a wave starting from a point source is the most analyzed DOA estimation scenario [59]. This assumption refers that the size of the source is small compared to the distance between source and the antennas receiving the signal. The opening angle is zero. Consequently; the signal source corresponding with the direction of the array is identified uniquely.

3.2.2. Far Field Assumption

According to previous assumption, a point source produces waves that propagate spherically around itself, however, when the antenna array is located at a large distance away from the source, the waves impinging upon sensors happens to be the plane waves [60].

Plane waves produce only constant phase difference between sequential sensor elements. It is assumed that every element that belongs to the same lattice has an omnidirectional and equal response. In addition, locations of the array elements are known perfectly in order to make appropriate delay calculations. There is no magnitude variation between the received signals, since spacing between array elements is small enough. Furthermore, the medium channel is lossless, phase is steady, and the propagation speed is uniform. All the receiving signals can be considered as individual plane waves, since the array has finite number of signals [16]. If sources are located in near field, our assumption are not applicable. Depending on the location of the sources, sensors have extra phases. The performance of the algorithms suffers from these unknown phases. It is too difficult to model the system using near field assumption. Hence, it is logical to choose far field assumption [16].

3.2.3. Narrowband Signal Assumption

The narrowband signal hypothesis, used in array signal processing, assumes that the time-bandwidth product is “small,” i.e., [61]

$$\beta_s \tau \ll 1$$

where β_s and τ refer to the bandwidth of the transmitted signal, and the propagation delay of wavefront respectively. According to above expression, bandwidth of the signal is less than the reciprocal of the wave propagation delay. T narrowband assumption assures that all array elements can capture a signal concurrently.

3.2.4. Correlation of Sources

The existence of coherent sources is another major problem for DOA estimation algorithms. Coherent sources appear due to multipath effects, cluttering and enemy jamming process [62]. In almost all real-time applications these effects are seen. In

order to handle the coherency of sources, different spatial smoothing techniques [63, 64, 65] are used. Throughout this thesis, correlation case, multipath effects and jamming are not considered. Thus, all sources are assumed to be uncorrelated.

3.2.5. Noise Assumption

Information observed in sensor elements come into existence with signal and noise components. Generally, signal component consists of both the source of interest and other unwanted sources like reflections. In addition, noise can be formed as environmental noise and thermal noise in all electronic components. Noise is assumed to be spatially and temporally stable and there is no correlation between signal components and noise components [16]. Noise has a zero-mean value with a variance σ^2 , i.e., white Gaussian noise. In addition, noise samples are statistically independent [66], i.e.,

$$E\{e_k(t)e_i(t)\} = \delta_{k,i}(t)$$

where $e_k(t)$ is the noise sample at k^{th} sensor at time t [16].

3.3. Signal Model

All notations used to describe the system model is summarized in this chapter with respect to the purpose of discussion and these notations are developed in accordance with the work of Stoica in order to avoid the literature conflicts in DOA estimation algorithms [16].

First, establishing an array model for a single source case is considered. Generic array model of multiple sources is achieved by applying superposition principle.

Let us consider a single waveform $x(t)$ acting on an array. The reference point should be chosen from either a sensor in the array or any other location that is far enough to the array in order satisfy the planar wave assumption.

All signals captured by the array are real time-continuous signals, hence, t is a continuous variable.

τ_k stands for the time needed for the plane wave to travel from reference point to the k^{th} sensor where k is ranges from 1 to number of sensors m .

Impulse response of a sensor is denoted by $\bar{h}_k(t)$, and the additive noise which can be either thermal noise or environmental noise is expressed by $\bar{e}_k(t)$, the output signal of the k^{th} sensor can be written as,

$$\bar{y}_k(t) = \bar{h}_k(t) * x(t - \tau_k) + \bar{e}_k(t) \quad (3.1)$$

where '*' refers to the convolution operation. The impulse response $\bar{h}_k(t)$ and array geometry are known quantities whereas $x(t)$ and τ_k are unknown parameters. When the value of τ_k is estimated, angle of arrival of the sources can be predicted as well. Equation (3.1) can be made simpler and more useful by applying the narrowband assumption. It makes more sense to apply this assumption in frequency domain.

Fourier transforms of $\bar{y}_k(t)$, $\bar{h}_k(t)$, $x(t)$ and $\bar{e}_k(t)$ are denoted by $\bar{Y}_k(\omega)$, $\bar{H}_k(\omega)$, $X(\omega)$ and $\bar{E}_k(\omega)$, respectively. According to these notations and properties of Fourier transform, Equation (3.1) can be expressed in frequency domain as,

$$\bar{Y}_k(\omega) = \bar{H}_k(\omega)X(\omega)e^{-j\omega\tau_k} + \bar{E}_k(\omega) \quad (3.2)$$

The spectrum of a signal whose characteristic is bandpass with a carrier frequency ω_c can be fully defined by the spectrum of the corresponding lowpass signal.

Let $s(t)$ indicate the baseband signal related to $x(t)$, and $S(\omega)$ represents the FT of $s(t)$. The process of obtaining $x(t)$ from $s(t)$ is called as modulation while the reverse is called demodulation. In the following equations, some relevant information about modulation and demodulation process will be given in order to get better understanding, although it is not strictly related to the source-location problem.

The physical signal $x(t)$ has real values. Therefore, the spectrum function is even, and it is symmetric about $(\omega = 0)$. On the other hand, the demodulated signal $s(t)$ may have complex values. Since the transmitted signal is obtained by modulating the real valued signals, baseband spectrum will be symmetric about $\omega = \omega_c$. Since the propagation medium does not often provide symmetry property, the received signal with a corresponding baseband spectrum is not even. This means that demodulated signal includes complex values.

When $s(t)$ is multiplied by $e^{j\omega_c t}$, considering the shifting property of Fourier transform, $S(\omega)$ moves to the right, as long as the value of ω_c is greater than zero as illustrated in

$$S(\omega) = \int_{-\infty}^{+\infty} s(t)e^{-j\omega t} dt \quad (3.3)$$

$$\int_{-\infty}^{+\infty} s(t)e^{j\omega_c t}e^{-j\omega t} dt = \int_{-\infty}^{+\infty} s(t)e^{-j(\omega-\omega_c)t} dt = S(\omega - \omega_c) \quad (3.4)$$

The formula given in (3.4) is called complex modulation. As described previously, the modulated signal has real values which means an even spectrum. Therefore, the shift of $S(\omega)$ to the right as described in (3.4) must accompany to a left shift by ω_c .

Thus, we can write that

$$X(\omega) = S(\omega - \omega_c) + S^*(-(\omega + \omega_c)) \quad (3.5)$$

Demodulation of (3.5) can be performed multiplying $x(t)$ with $e^{-j\omega_c t}$, which yields

$$S(\omega) + S^*(-\omega - 2\omega_c) \quad (3.6)$$

and passing the resulting signal through a low-pass filter.

It can be clearly seen that, $X(\omega)$ includes not only the baseband component of $S(\omega)$ but also bandpass component of $S^*(-\omega - \omega_c)$. When $X(\omega)$ given by (3.5) is inserted into (3.2), the output of the sensor is obtained in frequency domain as

$$\bar{Y}_k(\omega) = \bar{H}_k(\omega)[S(\omega - \omega_c) + S^*(-\omega - \omega_c)]e^{-j\omega\tau_k} + \bar{E}_k(\omega) \quad (3.7)$$

Demodulation of \bar{y}_k is performed using

$$\tilde{y}_k(t) = \bar{y}_k(t)e^{-j\omega_c t} \quad (3.8)$$

Fourier Transform of $\tilde{y}_k(t)$ can be expressed as

$$\tilde{Y}_k(\omega) = \bar{H}_k(\omega + \omega_c)[S(\omega) + S^*(-\omega - 2\omega_c)]e^{-j(\omega+\omega_c)\tau_k} + \bar{E}_k(\omega + \omega_c) \quad (3.9)$$

$\tilde{Y}_k(\omega)$ is low-pass filtered to exclude higher frequency term $S^*(-\omega - 2\omega_c)$.

Result of the filtering process is given in (3.10)

$$Y_k(\omega) = H_k(\omega + \omega_c)S(\omega)e^{-j(\omega+\omega_c)\tau_k} + E_k(\omega + \omega_c) \quad (3.10)$$

where $E_k(\omega + \omega_c)$ and $H_k(\omega + \omega_c)$ are low-pass components of $\bar{E}_k(\omega + \omega_c)$ and $\bar{H}_k(\omega + \omega_c)$ respectively.

As previously discussed in Section 3.2.3, as $|\omega|$ increases, $|S(\omega)|$ decreases quickly. This assumption is valid if we have [67]

$$BW \times \Delta T_{max} \ll 1 \quad (3.11)$$

$$\Delta T_{max} \triangleq \max_{n,m=0,\dots,N-1} \{\Delta T_{nm}\} \quad (3.12)$$

where BW refers to the bandwidth of $s(t)$ and ΔT_{nm} denotes the time duration for the incoming signal to travel between from n^{th} to m^{th} sensor.

Equation (3.10) can be written as

$$Y_k(\omega) = H_k(\omega_c)S(\omega)e^{-j\omega_c\tau_k} + E_k(\omega+\omega_c) \quad (3.13)$$

assuming the flatness of frequency response of sensor array around ω_c , i.e.,

$$H_k(\omega_c) = H_k(\omega + \omega_c) \quad (3.14)$$

The time domain expression for (3.13) is found by taking Inverse Fourier Transform and the result is written as,

$$y_k(t) = H_k(\omega_c)e^{-j\omega_c\tau_k} + s(t) e_k(t) \quad (3.15)$$

We define the array transfer vector or direction vector, \mathbf{a} , as in

$$a(\theta) = [H_1(\omega_c)e^{-j\omega_c\tau_1} \quad \dots \quad H_m(\omega_c)e^{-j\omega_c\tau_m}]^T \quad (3.16)$$

Since array structures and transfer characteristic of the sensors are assumed to be known, τ_1, \dots, τ_m which are the functions of θ carry information about the angle of arrival which is the major parameter of interest in this thesis. Furthermore, due to the fact that all sensors have omnidirectional characteristic, $H_k(\omega_c)$ is not affected from θ .

By making use of equation (3.16), equation (3.15) can be written as,

$$y(\theta) = a(\theta)s(t) + e(t) \quad (3.17)$$

where

$$y(\theta) = [y_1(t) \quad \dots \quad y_m(t)]^T \quad (3.18)$$

$$e(\theta) = [e_1(t) \quad \dots \quad e_m(t)]^T \quad (3.19)$$

refers to the output vector and noise vector, respectively.

Choosing the first sensor as a reference point, equation (3.16) can be written as

$$a(\theta) = [1 \quad e^{-j\omega_c\tau_1} \quad \dots \quad e^{-j\omega_c\tau_{m-1}}]^T \quad (3.20)$$

Equation (3.20) and (3.18) can be modified for multiple sources by applying superposition principle since all sensors are uniform. The steering matrix \mathbf{A} for multiple sources is defined as

$$\mathbf{A} = [a(\theta_1), \dots, a(\theta_n)] \quad (3.21)$$

And the received signal can be written using the steering matrix as

$$y(t) = [a(\theta_1), \dots, a(\theta_n)] \begin{bmatrix} s_1(t) \\ \vdots \\ s_n(t) \end{bmatrix} + e(t) \triangleq \mathbf{A}s(t) + e(t) \quad (3.22)$$

where θ_k represents DOA angle of k^{th} source, and $S_k(t)$ represents signal of k^{th} source.

Geometry of array, circular, triangular, NLA, ULA, etc., form of the incoming wave, planar, spherical, circular, etc., and angle of DOA of sources affect τ_k .

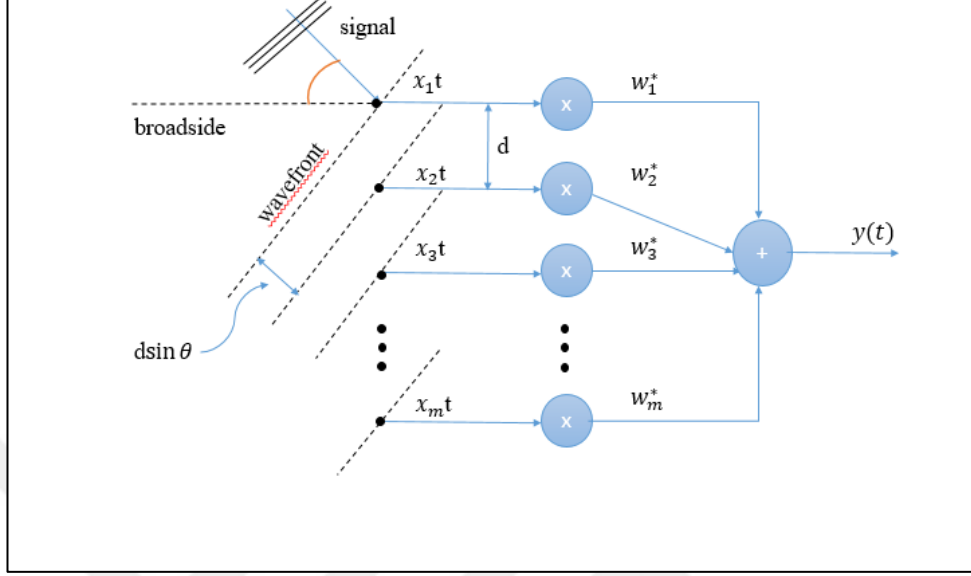


Figure 6 A plane wave incident on a uniform linear array of M uniformly spaced sensors

Figure 6 illustrates a plane wave received by a uniform linear array of M uniformly spaced sensors. From Figure 6, under the plane wave assumption and the choosing first sensor as the reference point, τ_k can be calculated as,

$$\tau_k = (k - 1) \frac{d \sin(\theta)}{c}, \theta \in [-90, 90] \quad (3.23)$$

where c refers to the propagation velocity, for example the speed of light, and d refers to the distance between sensors.

Inserting (3.23) into (3.20), we obtain

$$\mathbf{a}(\theta) = [1 \quad e^{-j\omega_c d \sin\theta/c} \quad \dots \quad e^{-j(m-1)\omega_c d \sin\theta/c}]^T \quad (3.24)$$

Wavelength of the signal, denoted by λ , is the distance traveled by the wave in one period and it is calculated as

$$\lambda = c/f_c \quad f_c = \omega_c/2\pi \quad (3.25)$$

The spatial frequency ω_s can be written as,

$$f_s = f_c \frac{d \sin(\theta)}{c} = \frac{d \sin(\theta)}{\lambda} \quad (3.26)$$

$$\omega_s = 2\pi f_s = \omega_c \frac{d \sin(\theta)}{c} \quad (3.27)$$

The direction vector can be redefined as,

$$\mathbf{a}(\theta) = [1 \quad e^{-j\omega_s} \quad \dots \quad e^{-j(m-1)\omega_s}]^T \quad (3.28)$$

In order to prevent overlapping in spectral domain, the sampling frequency should be chosen to satisfy

$$|\omega_s| \ll \pi \quad (3.29)$$

If equation (3.29) is satisfied, then there is no spatial aliasing, and based on this expression the following equivalent equations can be derived

$$\left| \omega_c \frac{d \sin(\theta)}{c} \right| \leq \pi \quad (3.30)$$

$$\left| 2\pi f \frac{d \sin(\theta)}{c} \right| \leq \pi \quad (3.31)$$

$$\left| \frac{d \sin(\theta)}{\lambda} \right| \leq \frac{1}{2} \quad (3.32)$$

$$d |\sin(\theta)| \leq \frac{\lambda}{2} \quad (3.33)$$

$$d \leq \frac{\lambda}{2}, \theta \in [-90, 90] \quad (3.34)$$

Expression (3.34) explains that half of the signal wavelength should be greater than d , which can be called spatial sampling period. The expression in (3.34) can be considered as spatial NYQUIST sampling theorem.

3.4. Algorithms

In this section, beamforming algorithms are introduced. Different DOA algorithms are compared in Table 1 where advantages and disadvantages of the techniques, mentioned in survey of DOA [1] are illustrated.

Table 1 Comparison between beamforming techniques for DOA estimation[1].

Technique	Merits	Demerits
Diagonal Loading Beamformer	Robust against finite sample errors	No reliable way to choose the diagonal loading factor, which directly affects its performance.
Eigen-Space Based Performance	Excellent robustness against arbitrary steering vector errors	Degrade severely if the subspace dimension L is uncertain or known imprecisely.
LCMV Linear Constraint Minimum Variance	Improved robustness	Strong degradation of the output SINR
MVDR Minimum Variance Distortionless Response	Gives distortionless performance in the Direction of Interest	Unable to distinguish between two closely spaced plane waves
Root MVDR	Better Performance	Lesser threshold compared to MVDR
MUSIC Multiple Signal Classification Algorithm	High level of orthogonality between signals. Higher resolution & accuracy	Gives the pseudo spectrum only.
Root MUSIC	Less computational time, higher resolution	Limited to linear antennas, equispaced.
ESPRIT Estimation of Signal Parameters via Rotational Invariance Technique	No need of searching the maxima in pseudo spectrum. Less sensitive to noise	More prone to errors.

There are different types of algorithms improved throughout the years. However, not all the algorithms are primary subject of this thesis. Table 1 gives an insight of common types of algorithms.

Algorithms have superiority to each other in some aspects while they have deficiencies to each other in some other aspects such as robustness, accuracy, computational complexity and noise sensitivity.

3.4.1. Non-Subspace Algorithms

The vector of arrays $a(\theta)$ is assumed to be known, and this model describes the array as a spatial sampling device. Array structure can be expressed using tapped delay line implementation of a spatial FIR filter, which is defined as

$$y_F(t) = \sum_{k=0}^{m-1} h_k u(t-k) \triangleq h^* y(t) \quad (3.35)$$

where h_k are filter weights, and $u(t)$ is the input of the filter. Filter coefficients can be written in vector form as

$$h = [h_0 \ h_1 \ \dots \ h_{m-1}]^* \quad (3.36)$$

and the received signal can be expressed in vector form as

$$y(t) = [u(t) \ u(t-1) \ \dots \ u(t-m+1)]^T \quad (3.37)$$

and using $\{y_k(t)\}_{k=1}^m$ spatial samples, spatial filtering is performed using

$$y_F(t) = h^* y(t) \quad (3.38)$$

Using the notations described previously, $y(t)$ can be rewritten as

$$y(t) = a(\theta)s(t) \quad (3.39)$$

where using (3.38), we obtain

$$y_F(t) = [h^* a(\theta)]s(t) \quad (3.40)$$

which implies that the tap values, $\{h\}_{k=1}^m$, of the spatial filter can be adjusted to improve or attenuate the signals received from an angle θ . The equation (3.40) is basic for DOA methods described in this section. In Figure 7, the tapped delay line implementation of FIR filter is described [16].

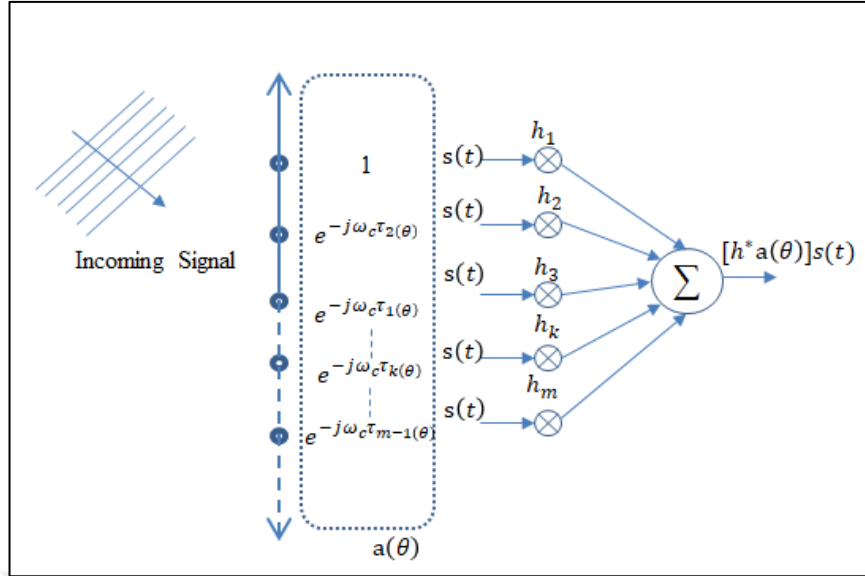


Figure 7 Tapped Delay Line Structure Spatial Filter

The power of the output signal in equation (3.38) can be calculated using

$$E\{|y_F(t)|^2\} = \mathbf{h}^* \mathbf{R} \mathbf{h} \quad (3.41)$$

where

$$\mathbf{R} = E\{y(t) y^*(t)\} \quad (3.42)$$

According to power equation (3.41), $\mathbf{h}^* \mathbf{R} \mathbf{h}$ should have peak. Non-adaptive conventional beamforming known as Bartlett beamforming, and adaptive Capon beamforming which tries to calculate the peak value of (3.41) will be introduced in the next subsections.

3.4.1.1. Beamforming Techniques

Beamforming is used in conjunction with a series of antennas / sensors to receive / transmit signals from a specific spatial direction in the existence of noise and interference. Therefore, it can be considered as a spatial filtering [1, 67]. It is a classic but regularly evolving field with tremendous practical applications. Over the last decade, interest in beam shaping running by applications in wireless communications increased, and numerous techniques are proposed.

A receiver beamformer is commonly used to estimate the signal from a particular direction in the presence of noise and blocking signals. In a receiver beamformer,

output of the sensors are combined using spatial filter coefficients, i.e., using weight vector. Hence, signals from a desired direction are combined, whereas signals coming from other directions are attenuated.

The most common beamforming techniques include conventional and adaptive beamformers.

For conventional beamformers, the weight vector for a given DOA can be pre-calculated independently of the received data. Therefore, they are called data independent beamformers, and they provide a fixed response for all signal and interference scenarios.

Adaptive beamformers are not data independent, and weight vectors are a function of incoming data [68]. Much better resolution and better interference resistance is observed for adaptive beamformers than independent beamformers.

Conventional (Bartlett) Beamforming

A conventional non-adaptive beamformers consist of delay, sum units employing weighting vectors. Because of the geometry of sensors, the time delay between each sensor is retrieved and the output of the sensors are summed.

In Figure 8, the structure of conventional beamformer is shown.

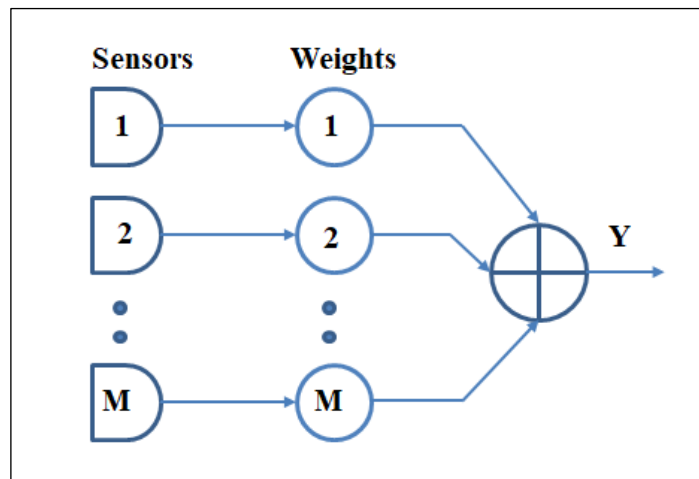


Figure 8 Conventional Beamformer

This method is also known as Bartlett beamforming [12]. The power spectrum based on this method is computed as follows. According to previous subsection, filter \mathbf{h} must

satisfy two conditions. First condition is that it should pass the signals with a given θ , and second condition is that it should attenuate the remaining signals arriving at angles different than θ .

In the view of (3.40), the first condition for the filter can be written as

$$\mathbf{h} * a(\theta) = 1 \quad (3.43)$$

By using the identical sensors, normalized transfer vector $a(\theta)$ can be written as

$$\mathbf{a}^*(\theta)\mathbf{a}(\theta) = m \quad (3.44)$$

Considering second condition, accepting $y(t)$ in (3.41) is spatially white, we can write

$$E\{\mathbf{y}(t) \mathbf{y}^*(t)\} = \mathbf{R} = \mathbf{I} \quad (3.45)$$

The power of the filtered signal calculated as

$$E\{|y_F(t)|^2\} = \mathbf{h}^* \mathbf{R} \mathbf{h} = \mathbf{h}^* \mathbf{h} \quad (3.46)$$

All these expressions illustrate that the spatially white signal in sensor output has the same power for all directions of θ . It is also required that \mathbf{h} minimizes the power in (3.46), which leads to the expression

$$\min_{\mathbf{h}} \mathbf{h}^* \mathbf{h} \text{ subject to } \mathbf{h} * a(\theta) = 1 \quad (3.47)$$

The optimum filter coefficients can be calculated as

$$\mathbf{h} = \frac{\mathbf{a}(\theta)}{\mathbf{a}^*(\theta)\mathbf{a}(\theta)} \quad (3.48)$$

When (3.48) is inserted into (3.46), we obtain

$$E\{|y_F(t)|^2\} = \frac{\mathbf{a}^*(\theta)\mathbf{R}\mathbf{a}(\theta)}{m^2} \quad (3.49)$$

However, theoretical covariance matrix \mathbf{R} can be only estimated using the finite data as

$$\hat{\mathbf{R}} = \frac{1}{N} \sum_{n=1}^N \mathbf{y}(n) \mathbf{y}^*(n) \quad (3.50)$$

where N represents the number of snapshots. The beamformer DOA estimates are the positions of the highest peaks of

$$\mathbf{a}^*(\theta) \mathbf{R} \mathbf{a}(\theta) \quad (3.51)$$

The main weakness of conventional beamforming can be explained that all freedom degrees are used in the array in order to form a beam in required direction. If multiple sources occur, this method has a limitation to keep an acceptable level of main beamwidth lobe and side lobe. As a result, a low resolution is observed.

Adaptive Beamformer

Adaptive beamformers are data dependent which makes adaptive beamformers to optimize a series of weights vectors. In Figure 9, the structure of adaptive beamformer is shown.

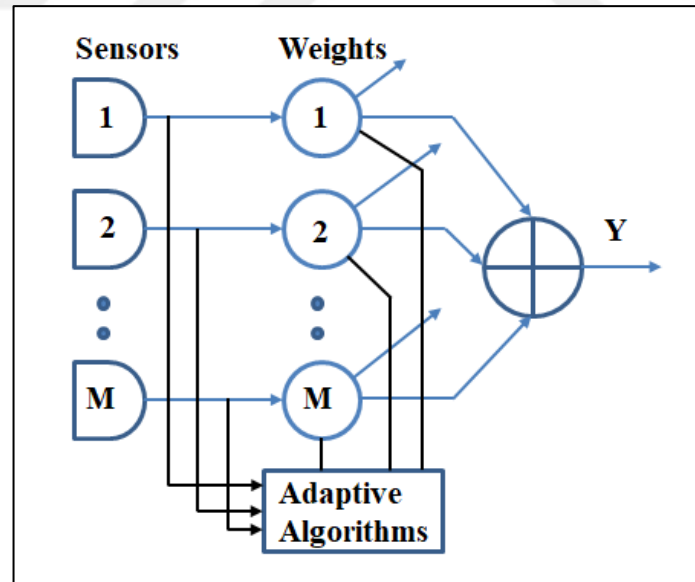


Figure 9 Adaptive Beamformer

Capon beamforming is one of the adaptive beamforming techniques. The brief mathematical explanation will be given in next subsection.

3.4.1.2. Capon Beamformer

Capon method is an adaptive beamforming technique that is developed by Capon [10] in order to reduce the negative aspects of conventional beamforming. As previously discussed, conventional beamforming has deficiency if multiple signal sources are used. Spatial spectrum estimation contains not only signal source power from desired direction but also power coming from other directions. Capon method overcomes the low-resolution problem associated with the Bartlett method.

Nevertheless, Capon method also has a weakness that occurs when the direction of other signals is close to the direction of incident signals. In this situation, Capon method will make some errors.

The constraint optimization of Capon beamforming method resembles the conventional beamforming. However, a data dependent model is found in Capon in terms of \mathbf{h} while conventional beamforming is data independent. The partial filter design of Capon is performed using

$$\min_{\mathbf{h}} \mathbf{h}^* \mathbf{R} \mathbf{h} \text{ subject to } \mathbf{h}^* \mathbf{a}(\theta) = 1 \quad (3.52)$$

Thus, incoming signal coming directly from θ is passed by Capon filter, while other signals from different directions are attenuated. However, conventional beamforming only attenuates any other directions even though there is no signal source available.

The solution of (3.52) is found as

$$\mathbf{h} = \frac{\mathbf{R}^{-1} \mathbf{a}(\theta)}{\mathbf{a}^*(\theta) \mathbf{R}^{-1} \mathbf{a}(\theta)} \quad (3.53)$$

When it is inserted into the power of the filtered signal, the output power can be calculated as

$$E\{|y_F(n)|^2\} = \frac{1}{\mathbf{a}^*(\theta) \mathbf{R}^{-1} \mathbf{a}(\theta)} \quad (3.54)$$

Using finite sample data theorem, Capon DOA estimation is achieved using the peaks of

$$\frac{1}{\mathbf{a}^*(\theta) \mathbf{R}^{-1} \mathbf{a}(\theta)} \quad (3.55)$$

CHAPTER 4

SIMULATION RESULTS AND DISCUSSION

4.1. Introduction

Numerical simulations are given in this chapter to investigate uniform linear array performance analysis. The source signals are all narrowband signals. In addition, the angle of arrival belongs to the interval $[-\frac{\pi}{2}, +\frac{\pi}{2}]$. The signals are also uncorrelated with each other and background noise which is assumed to be white Gaussian complex noise with zero mean. Employing different values of variables, the number of snapshots (N), number of array elements (M), signal to noise ratio (SNR), and spacing between sensor elements (d), in MATLAB simulations, it is possible to demonstrate the differences between conventional and Capon beamforming techniques with ULA. These simulations are performed considering

- i. The effects of various number of sensors on DOA estimation.
- ii. The effects of various number of snapshots on DOA estimation.
- iii. The effects of elements spacing between sensors on DOA estimation.
- iv. The effects of various level of SNR on DOA estimation.

4.2. Experiment 1: Basic simulations of conventional beamforming and Capon beamforming

The first simulation illustrates how the angles of three signals are detected by the conventional beamforming and Capon beamforming algorithms.

The signal sources are selected as three uncorrelated narrowband signals. The incident angles are 15° , 40° and 65° respectively. The background noise has the characteristics

of white Gaussian noise. The SNR value is 10dB. The array element number, M , is 20, carrier frequency, f_c , is equal to 1Mhz. The speed of light, c , is 3.00×10^8 m/s. In addition, depending on the c and f_c , the wavelength, λ , of the signal can be calculated as 300 meters. The distance between two sensors is equal to half of the wavelength. Number of snapshots is 200. According to given numeric parameters the simulation results are shown in Figure 10 and Figure 11.

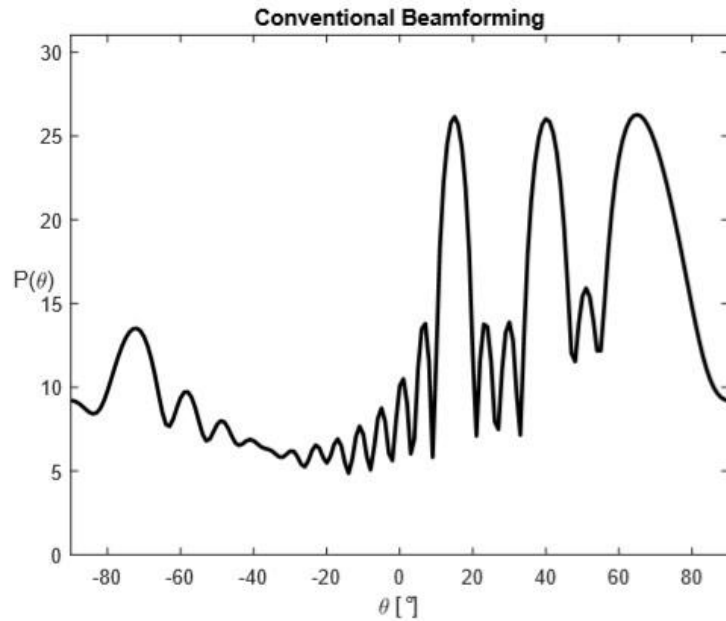


Figure 10 Conventional beamforming with three signal sources

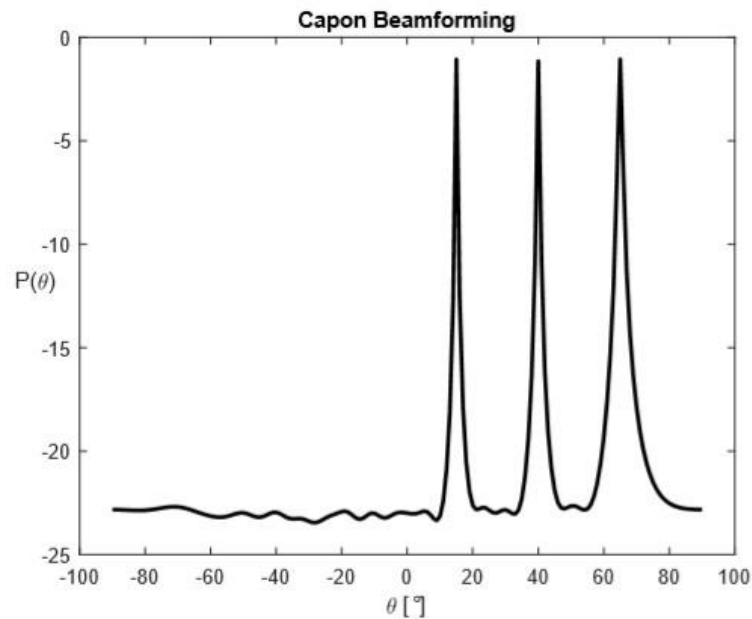


Figure 11 Capon beamforming with three signal sources

4.3. Experiment 2: Relationship between DOA and the number of sensors

In this experiment, the effect of element number M is measured via simulations. All other parameters are kept same except the number of sensors. Two uncorrelated narrowband signals are used as signal sources. The incident angles are 0° and 40° respectively. The background noise has the characteristics of white Gaussian noise. The SNR value is 10dB. The sensor numbers represented with M_1, M_2, M_3 and they are chosen as 5, 16, and 32 respectively. Carrier frequency, f_c is equal to 1MHz. The speed of light, c , is 3.00×10^8 m/s. The wavelength, λ of the signal is equal to 300 meters. The distance between two sensors is equal to half of the wavelength. Number of snapshots is 200. According to given numeric parameters, the simulation results are illustrated in Figure 12 and Figure 13.

In Figure 12 and Figure 13, the black dashed line corresponds to the sensor number 5, the blue dash-dotted line corresponds to the sensor number 16, and the red solid line corresponds to the sensor number 32. When other parameters are kept same, increasing the number of sensors make the beamwidth narrower, and as a result, improvement in the directivity of the antenna, and improvement in the discrimination of spatial signals is observed. If it is desired to obtain more accurate estimations, more sensors should be used. However, using more sensors causes more data to be processed and needs a large amount of computation. In practice, the number of sensors should be selected properly as needed, in order to avoid wasting of resources and low operating speed.

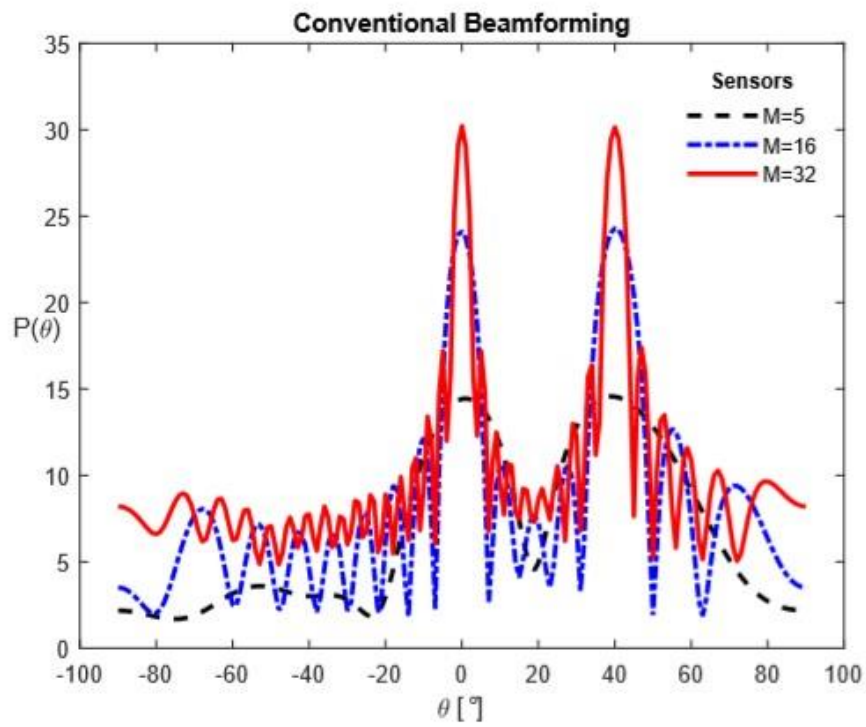


Figure 12 Conventional beamforming with different number of sensors

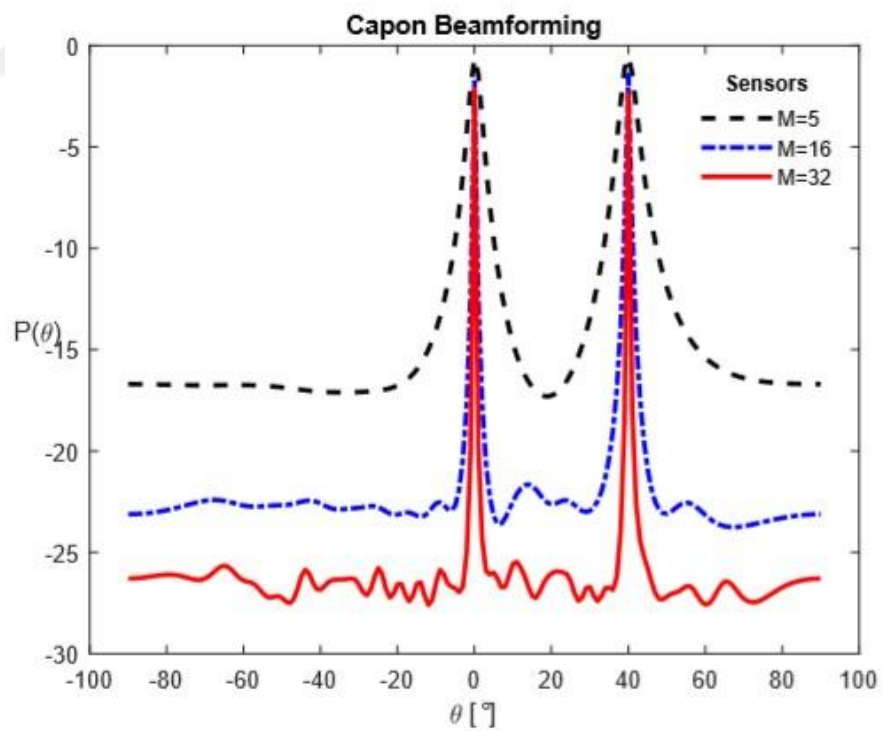


Figure 13 Capon beamforming with different number of sensors

4.4. Experiment 3: Relationship between DOA and the number of snapshots

In this experiment, the effect of snapshots N is measured via simulations. All other parameters are kept the same except for the number of snapshots. Two uncorrelated narrowband signals are used as signal sources. The incident angles are 0° and 40° respectively. The background noise has the characteristics of white Gaussian noise. The SNR value is 10dB. The number of snapshots is indicated by N_1 , N_2 , N_3 , and they are chosen as 25, 50, and 250, respectively. Carrier frequency, f_c is equal to 1 MHz. The speed of light, c is 3.00×10^8 m/s. In addition, the wavelength, λ of the signal is equal to 300 meters. Element number is 20. The distance between two sensors is equal to half of the wavelength. According to the given numeric parameters, the simulation results are depicted in Figures 14 and 15.

In from Figure 14 and Figure 15, the black dashed line is the simulation result when the number of snapshots is 25, the blue dash-dotted line is the simulation result when the number of snapshots is 50, and the red solid line is the simulation result when number of snapshots is 250.

When other parameters are kept same, it is seen that increasing the number of snapshots make the beamwidth narrower. As a result, improvement in the directivity of the antenna, and improvements in the discrimination of spatial signals are observed. Accuracy of Capon beamforming is increased; however, it does not affect the accuracy of conventional beamforming so much. If it is desired to obtain more accurate estimations, number of samples should be increased. However, using more samples causes more data to be processed, and processing latency increased. In practice, the

number of samples should be selected properly According to NYQUIST criterion in order to save resources and to obtain high operating speed.

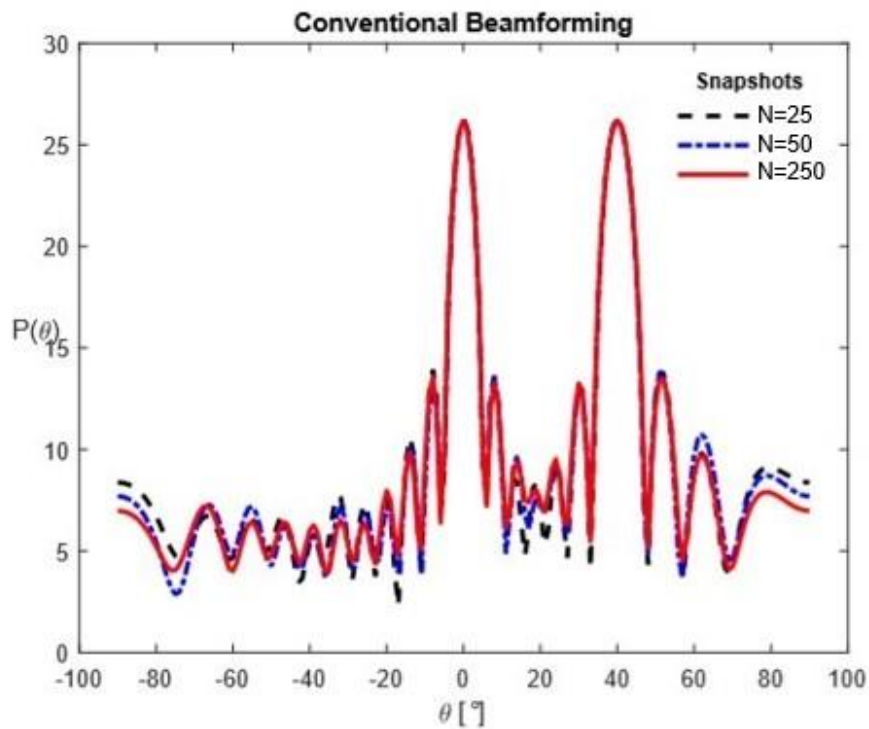


Figure 14 Conventional beamforming with different number of snapshots

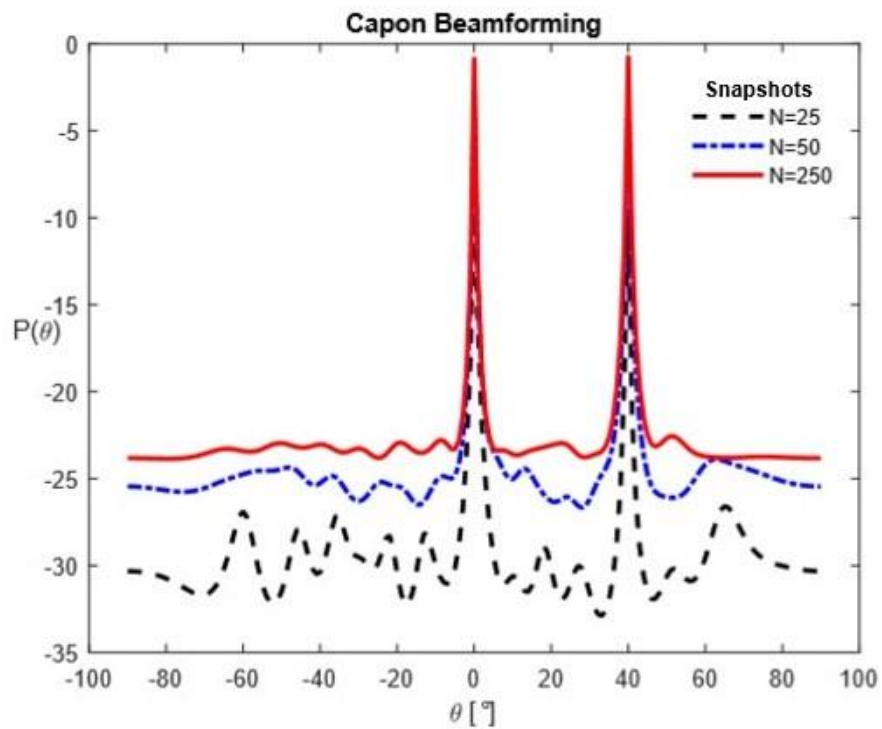


Figure 15 Capon beamforming with different number of snapshots

4.5. Experiment 4: Relationship between DOA and the element spacing

In this experiment, the effect of element spacing is measured. All other parameters are kept same except for the element spacing. Two uncorrelated narrowband signals are used as signal sources. The incident angles are 0° and 40° respectively. The background noise has the characteristics of white Gaussian noise. The SNR value is 10dB. The spacing between array sensors are represented by d_1 , d_2 , d_3 , and they are chosen as 0.1λ , 0.5λ , and 1λ respectively. Carrier frequency, f_c is equal to 1MHz. The speed of light, c is 3.00×10^8 m/s. In addition, the wavelength, λ of the signal is equal to 300 meters. Number of elements is 20. Number of snapshots is 250. According to given numeric parameters, the simulation results are shown in Figure 16 and 17.

In Figures 16 and 17, the black dashed line corresponds to the element spacing 0.1λ , the blue dash-dotted line corresponds to the element spacing 0.5λ , and the red solid line corresponds to the element spacing 1λ .

When other parameters are kept the same, and when the array element spacing is not greater than half of the wavelength, it is seen that the beamwidth becomes narrower. As a result, improvement in the directivity of the antenna, and improvements in the discrimination of spatial signals are observed. When the spacing is chosen larger than half of the wavelength, the estimated spectrum has false peaks. The estimation accuracy is lost.

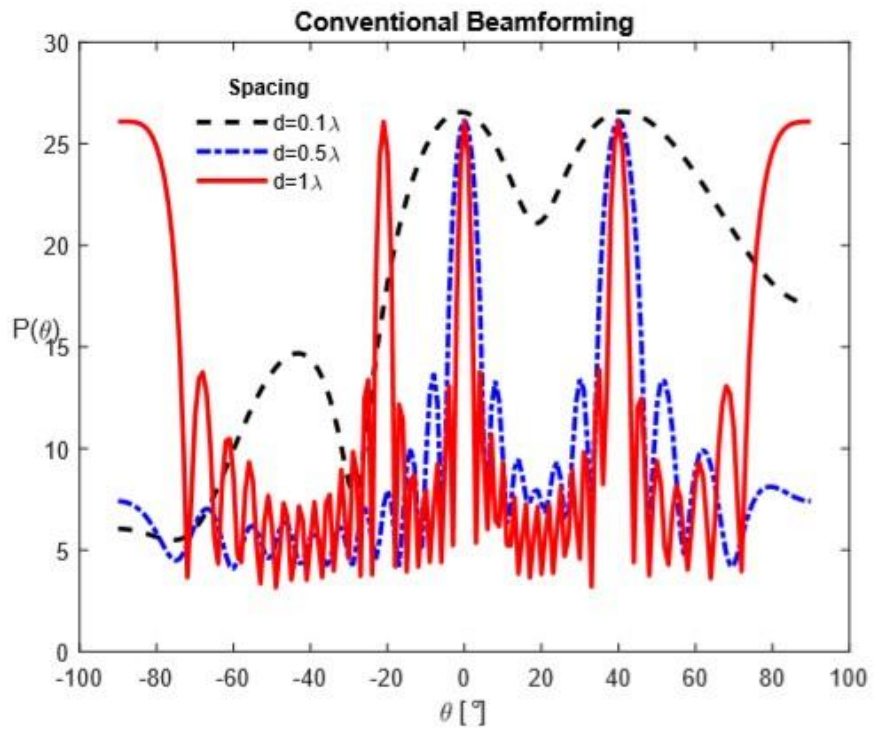


Figure 16 Conventional beamforming with different sensor element spacing

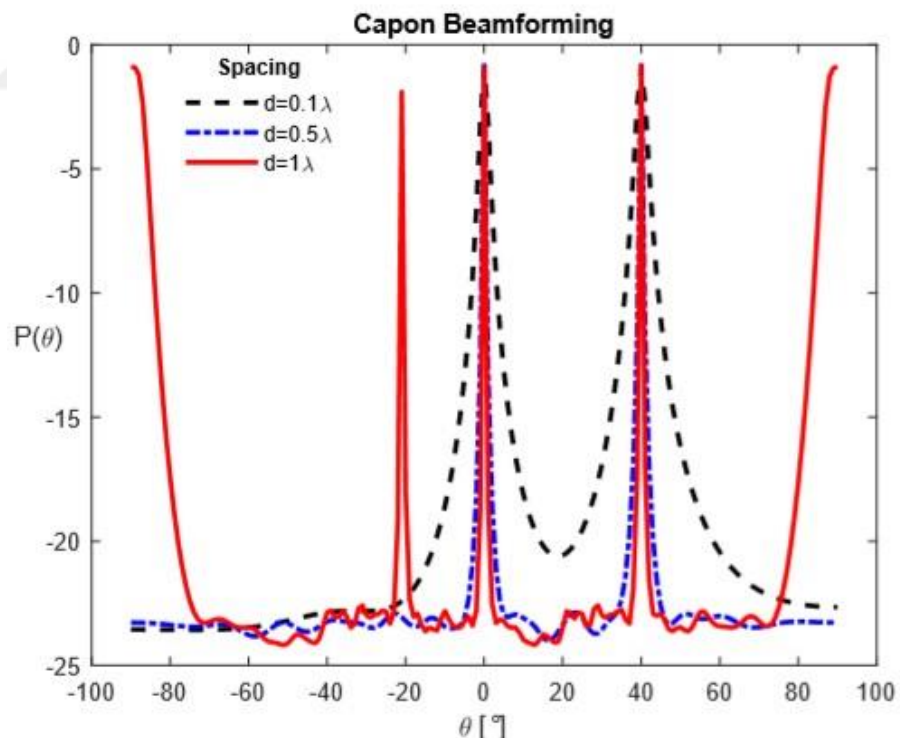


Figure 17 Capon beamforming with different sensor element spacing

4.6. Experiment 5: Relationship between DOA and SNR

In this experiment, the effect of SNR is measured. All other parameters are kept the same except for SNR values. Two uncorrelated narrowband signals are used as signal sources. The incident angles are 0° and 40° respectively. The background noise has the characteristics of white Gaussian noise. SNR values are represented by SNR_1 , SNR_2 , SNR_3 and they are chosen as -10dB, 0dB, and 10dB, respectively. Carrier frequency, f_c is equal to 1MHz. The speed of light, c is 3.00×10^8 m/s. The wavelength λ of the signal is equal to 300 meters. Number of elements is 20. The distance between two sensors is equal to half of the wavelength. Number of snapshots is 250. According to given numeric parameters, the simulation results are shown in Figures 18 and 19.

In Figures 18 and 19, the black dashed line corresponds to the SNR -10dB, the blue dash-dotted line corresponds to the SNR 0dB, and the red solid line corresponds to the SNR 10dB.

When other parameters are kept the same, it is seen that increasing SNR level results in narrower beamwidth. As a result, improvement in the directivity of the antenna, and improvements in the discrimination of spatial signals are observed. The resolutions of Capon and Conventional beamforming algorithms improve. The performance of Capon and Conventional beamforming algorithm drops sharply at low SNR values.

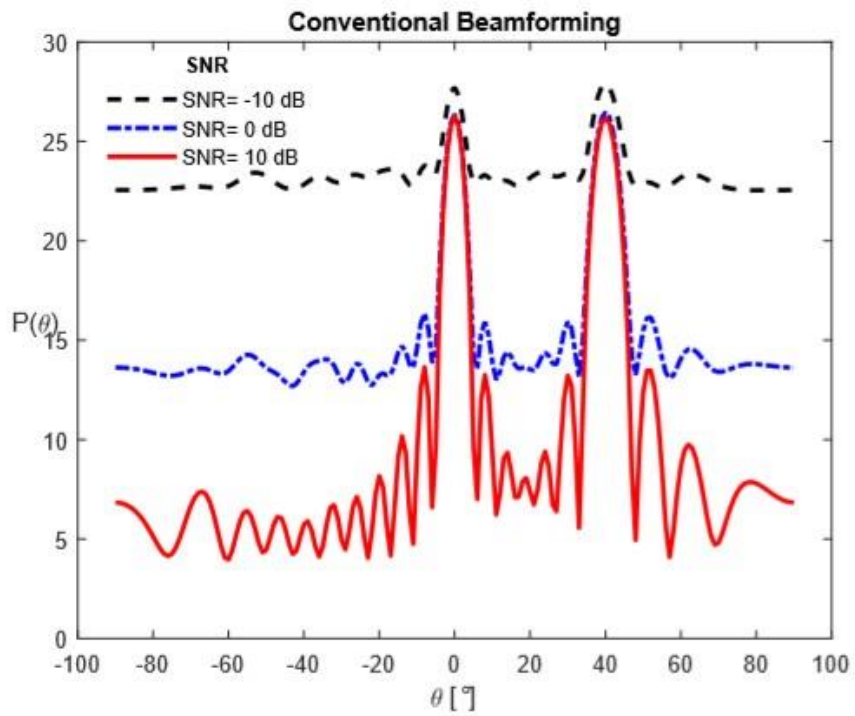


Figure 18 Conventional beamforming with different SNR values.

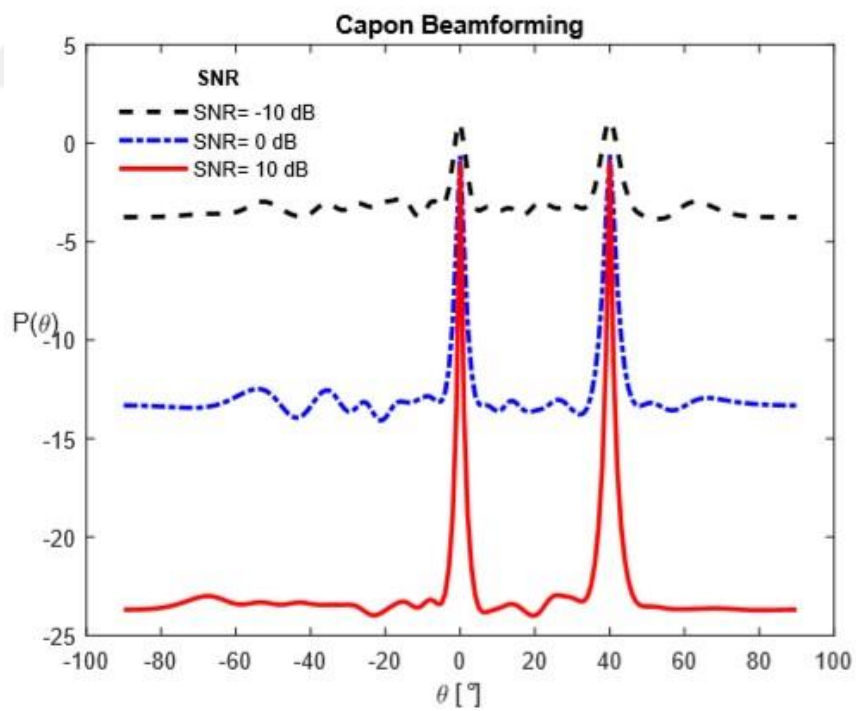


Figure 19 Capon beamforming with different SNR values

CHAPTER 5

VHDL IMPLEMENTATION of CONVENTIONAL BEAMFORMING

5.1 Introduction to VHDL Implementation

In this chapter, VHDL implementation of conventional beamforming is performed. First, theoretical formulas are implemented in MATLAB. Afterwards; MATLAB codes are converted to VHDL. In Figure 20, the design flow of VHDL programming on simulation level is shown. First, the requirements are analyzed which means that the inputs and outputs should be determined before writing any algorithm. According to minimum and maximum values of the required parameters, the intervals should be selected carefully in order to cover all values. After writing VHDL code, necessary libraries that include signed and unsigned arithmetic functions for vectors, logical operators and conversion functions should be compiled [70-71].

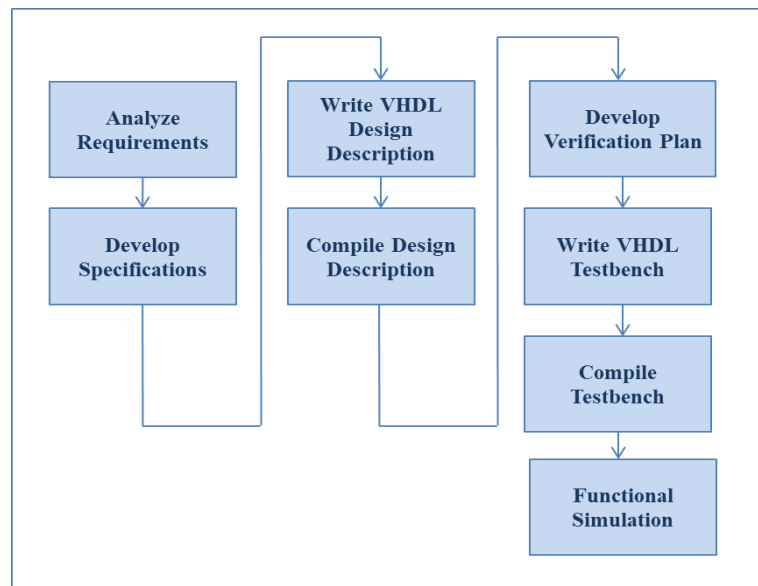


Figure 20 VHDL Design Flow for Simulation Level

In the next step, accuracy of the hardware model should be verified.

A test bench is used for functional simulation [70]. A test bench is an another VHDL file used to supply inputs for the design and display the outputs. In this thesis, several test benches are used to test the codes step by step. Furthermore, all calculations and mathematical operations in VHDL codes are written using fixed point numbers which are included in fixed point package. In VHDL, in order to represent decimal numbers, fixed-point representation is used. Fixed-point numbers have decimal points. The integer 23 can be represented using 8 bits as in Figure 21.

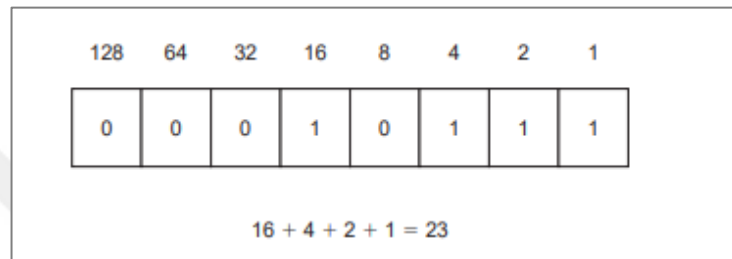


Figure 21 Binary Notation of a Positive Number

If the binary representation of a negative number is required, signed approach is used by taking two's complement form of the number where Most Significant Bit (MSB) defines the sign bit [71]. Figure 22 shows the binary representation of -23.

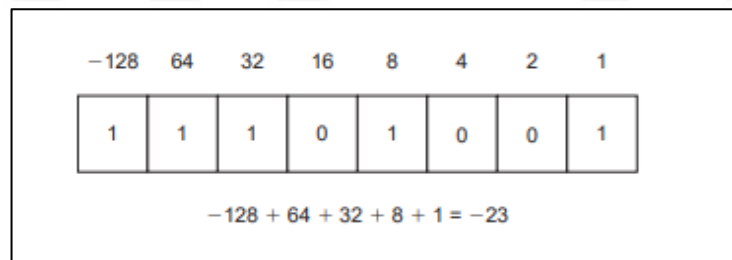


Figure 22 Binary Notation of a Negative Number

In 'fixed-point' notation, the position of decimal point is indicated, the rest is the same as in Figure 21 and 22. The fixed-point representation of -2.875 is shown in Figure 23.

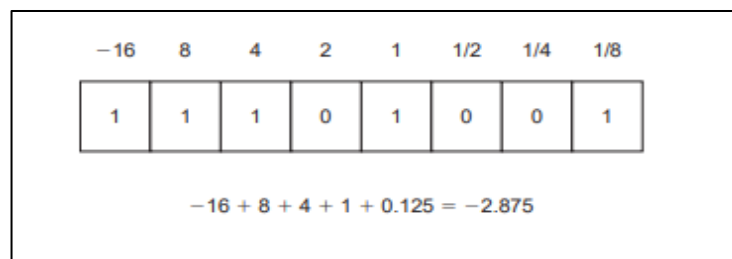


Figure 23 Fixed Point Notation

In Chapter 3 we use complex numbers which are combinations of real number and imaginary numbers.

The imaginary number which is known as “i” or “j” is the square root of -1 .

The algebraic representation of complex numbers is shown in Table 2. If real part has no component, it is called purely imaginary, if imaginary part has no component it is called purely real.

Table 2 Complex Number Representation

Complex Number	Real Part	Imaginary Part	
$3 + 2i$	3	2	
5	5	0	Purely Real
$-6i$	0	-6	Purely Imaginary

However, in VHDL imaginary number “i” is not known directly. A complex number should be defined by using type record definition

```

constant nl : integer := -8;
constant nh : integer := 4;

type complex is record
  re: sfixed(nh downto nl);
  im : sfixed(nh downto nl);
end record complex;

```

Figure 24 VHDL syntax of a basic definition of complex number

In Figure 24, the definition of complex numbers using fixed-point notation is given. In Figure 24, the constants **nh** and **nl** specify the bit number of integer part and fractional part. Defining bit numbers in this way enables the writing of generic codes.

Besides, “re” and “im” indicate the real and imaginary parts. The syntax for signed fixed-point type is defined by “sfixed”. Complex numbers are defined such that they include signed fixed **re** and **im** parts.

For matrices or tables multidimensional arrays can be used. An array consists of a series of values which are all of the same type. Every item's location in an array is specified by a scalar value, named its index [71]. In order to construct an array, we

need to define array data type first. The syntax to define an array is shown in Figure 25.

```
Syntax for VHDL array definition:
type type_name is array (range_start to range_end) of element_type;

Syntax for VHDL Two-Dimensional(2D) array definition:
type type_name is array (range1_start to range1_end, range2_start to range2_end) of
element_type;

Example: 2D Array Declaration

constant num_cols: integer := 16;

constant num_rows: integer := 8;

type re_in_matrix_8x16 is array (0 to num_rows - 1, 0 to num_cols -1) of complex;
```

Figure 25 Syntax Rule for VHDL Array Definition

Only one range is used for single-dimensional arrays. The range is labeled by “**range_start**” and “**range_end**” data words. “**type_name**” can be selected any logical word for data type name. “**element_type**” shows the type of each element in the array.

In order to declare a matrix, multidimensional arrays can be. In this thesis, two dimensional arrays are used. A 2D array is defined in Figure 25. According to figure, there are two ranges which are labeled by “**range1_start**”, “**range1_end**” and “**range2_start**”, “**range2_end**”.

Every element of defined arrays is a complex number which is defined in Figure 22. The difference “**range1_start-range1_end**” defines the row size of the matrix. The row elements are indexed from 0 up to 7. The difference “**range2_start-range2_end**” defines the column size of the matrix. The column elements are indexed from 0 up to 15. VHDL codes are written in this thesis using signed fixed-point numbers, and complex numbers and complex matrices with different sizes are introduced in user defined package. VHDL definitions for matrix data types are available in Appendix B.

5.2. VHDL Implementation of Uniform Linear Array Algorithm

We first explain the implementation of uniform linear array in VHDL. According to expression (3.22) given in Chapter 3 we have

$$y(t) = [a(\theta_1), \dots, a(\theta_n)] \begin{bmatrix} s_1(t) \\ \vdots \\ s_n(t) \end{bmatrix} + e(t) \triangleq \mathbf{A}s(t) + e(t) \quad (3.22)$$

$y(t)$ is generated in Matlab.

The code box given in Figure 26 shows the generation of $y(t)$ in MATLAB.

Algorithm : How to write Uniform Linear Array algorithm in Matlab	
Result: Generates N snapshots of ULA sensor data	
Inputs:	
theta	→ arrival angles of the m sources in degrees
P	→ The covariance matrix of the source signals
N	→ number of snapshots to generate
sig2	→ noise variance
m	→ number of sensors
d	→ sensor spacing in wavelengths
Y	→ m x N data matrix $Y = [y(1), \dots, y(N)]$
<hr/>	
j =sqrt(-1)	→ <i>complex number</i>
A =exp(-2*pi*j*d*[0:m-1].'*sin([theta(:)']*pi/180))	→ <i>generate the A matrix</i>
n =max(size(P))	→ <i>size of P</i>
s =(sqrtm(P)')*randn(n,N)	→ <i>generate the source component</i>
e =sqrt(sig2/2)*(randn(m,N)+j*randn(m,N))	→ <i>generate the noise component</i>
Y =A*s+e	→ <i>generate the ULA data</i>

Figure 26 ULA algorithm in MATLAB

The code takes the inputs which are θ , P , N , noise variance ($sig2$), d and produces data matrix Y with a size of $m \times N$.

θ represents the arrival angles of the sources in degrees. P is an identity matrix which is $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ with size of number of sources. N represents the number of samples while m shows the number of sensors, and d represents the spacing between these sensors. While implementing the VHDL code, number of sources is chosen as 2, noise variance equals to one, and number of sensors equals to 8, sensor spacing between two sensors is 0.5λ . Arrival angles of these two sources θ_1 and θ_2 are equal to 0° and 25° respectively.

5.2.1. VHDL Implementation of Sinusoidal Wave

In Figure 27, the time vector t shows the sampling instants. N_s , represents the number of samples taken from sine wave per-second. For sampling period 1, the time vector can be represented as $[0, \frac{1}{N_s}, \frac{2}{N_s}, \dots, \frac{N_s-1}{N_s}]$ where N_s equals to 64. It is sufficient to generate one period of the sine for the range $[0, 2\pi]$, since it is a periodic function.

Algorithm 1: How to write Sampling of Sine Wave in Matlab
Result: Generates Sine Samples for VHDL
Inputs:
N_s → number of samples : 64
t → time : $[0: 1/N_s: 1-1/N_s]$
f → frequency: 1Hz
Sine-Values = $\sin(2*\pi*f*t)$ → <i>Sinusoidal wave</i>
plot(Sine-Values)) → <i>Illustration</i>

Figure 27 Sine Look Up Table (LUT) generation with MATLAB

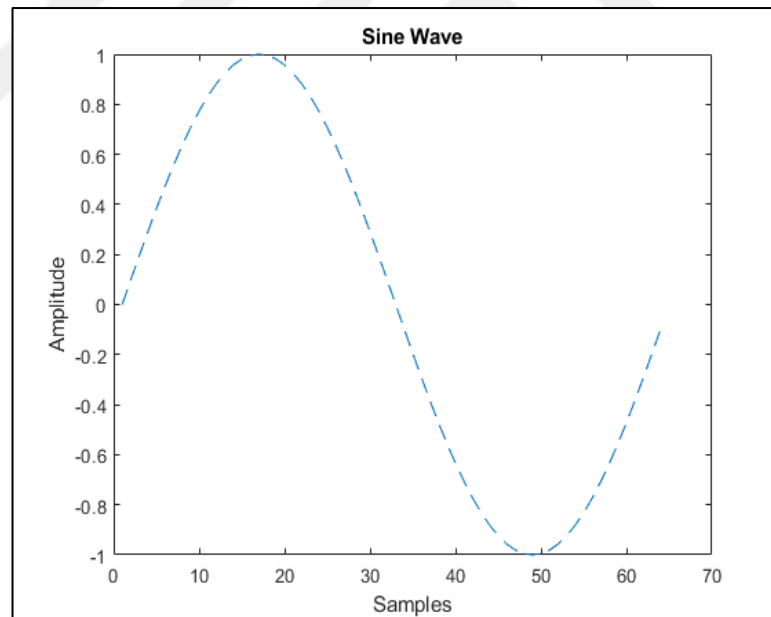


Figure 28 Sine Look Up Table (LUT) Samples

The amplitude of the sine wave is not quantized that means the amplitude samples have a range from -1 to 1. The real signed values are used. A simple MATLAB code for the generation of sinusoidal wave with 64 samples is shown in Figure 27. In Figure 28, the X axis is used for the samples and Y axis is used for the amplitudes. MATLAB generated sine values are shown in Table-3

Table 3 One period of Sine wave with 64 samples

Table : Sine Values from Matlab $[0, 2\pi]$							
0	0.0980	0.1951	0.2903	0.3827	0.4714	0.5556	0.6344
0.7071	0.7730	0.8315	0.8819	0.9239	0.9569	0.9808	0.9952
1.0000	0.9952	0.9808	0.9569	0.9239	0.8819	0.8315	0.7730
0.7071	0.6344	0.5556	0.4714	0.3827	0.2903	0.1951	0.0980
0.0000	-0.0980	-0.1951	-0.2903	-0.3827	-0.4714	-0.5556	-0.6344
-0.7071	-0.7730	-0.8315	-0.8819	-0.9239	-0.9569	-0.9808	-0.9952
-1.0000	-0.9952	-0.9808	-0.9569	-0.9239	-0.8819	-0.8315	-0.7730
-0.7071	-0.6344	-0.5556	-0.4714	-0.3827	-0.2903	-0.1951	-0.0980

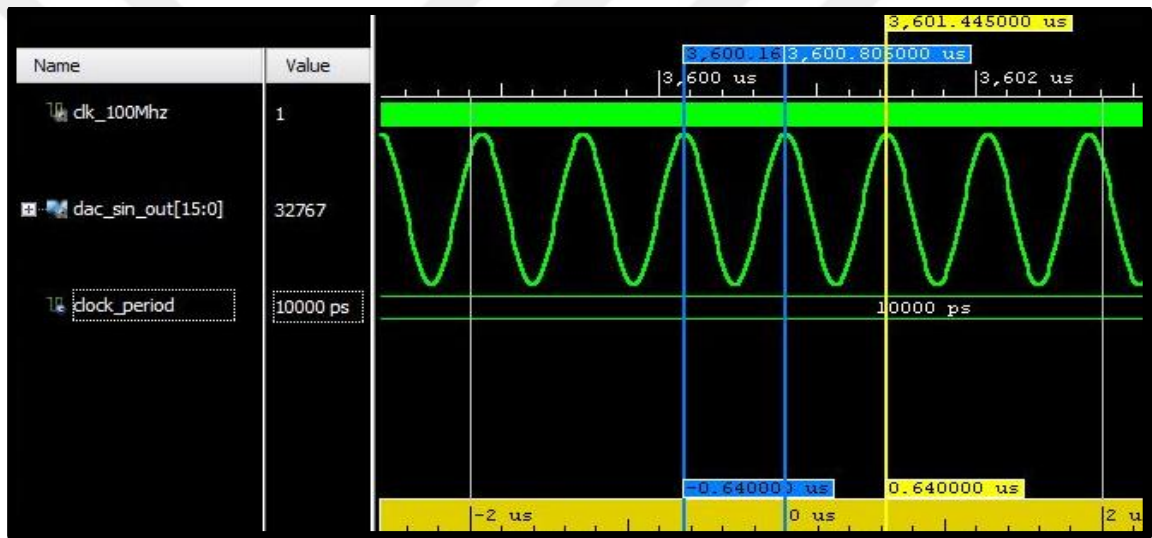


Figure 29 Vivado Simulation of Sine wave

The values that is given in Table 2 is used while writing the VHDL code for the generation of sine wave. VIVADO generated sine wave is depicted in Figure 29. An artificial clock is used to for plotting the samples of sine wave in Vivado simulator. The clock period is chosen as 10ns which means 100Mhz operating frequency. In Figure 29, the time difference between blue cursors is to $0.64\mu s$. This time value indicates that output samples are read for every clock cycle, and for 64 samples, 64 clock periods are needed.

5.2.2. VHDL Generation of Direction Matrix A

Table 4 shows the values of matrix A which denotes steering or direction matrix for multiple sources as declared in expression (3.21). The dimension of complex matrix A is 8×2 where row size depends on the number of sensors and column size depends on number of sources.

Table 4 Complex A Matrix Values

Table : Complex A Matrix Values			
real	imag	real	imag
1.000	0.000	1.000	0.000
1.000	0.000	0.241	-0.971
1.000	0.000	-0.8841	-0.4673
1.000	0.000	-0.666	0.746
1.000	0.000	0.5633	+0.8262
1.000	0.000	0.938	-0.348
1.000	0.000	-0.1119	-0.9937
1.000	0.000	-0.991	-0.131

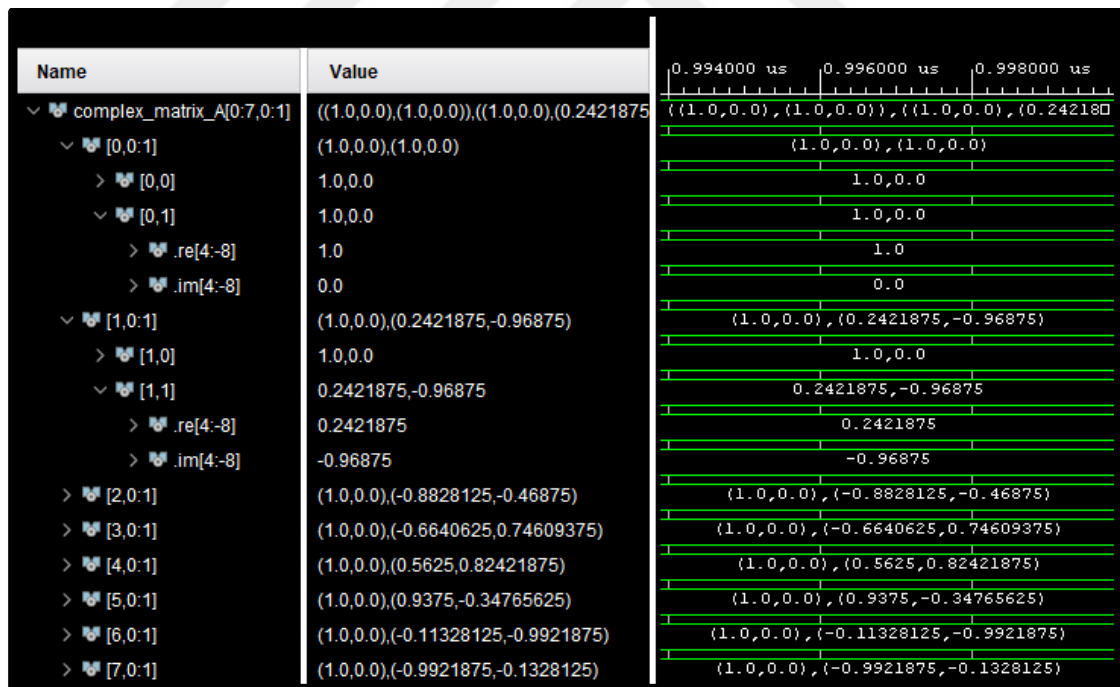


Figure 30 Vivado Simulation of Complex A matrix

The Vivado generation of A matrix is shown in Figure 30 where name of the signals, the values and digital waveform style are depicted.

The same artificial clock and the same clock period that is already mentioned in sine wave demonstration is used, and green horizontal lines show the values.

Every row and column of matrix has real (.re) and imaginary (.im) parts. The **re** and **im** parts of 0th row and 1st column, and 1st row and 1st column in the Figure 30 are shown in more details for better illustration. The remaining rows and columns are expressed in this way as well. It should be observed that the expected values and simulation results are mostly consistent. Integer parts are represented with 4 bits. However, some values may have small variations due to fractional part that is represented by 12 bits. If bit number is increased, precision will be better. However, using more bits makes mathematical operations complicated in hardware implementation.

5.2.3. VHDL Generation of Output Matrix Y

The next step is generation of Y matrix which contains the output values of the ULA sensors with 16 samples as depicted in Figure 26.

Matrix Y has random values because it includes the values from the noise source and the signal source. The source samples are generated by normally distributed random variable, and it is packed as 2-by-8 complex matrix. In addition, the noise source samples are produced by normally distributed random variable, and it is packed as 8-by-16 complex matrix. As a result, the size of complex matrix is 8x16 where row size indicates the number of sensors, and column size shows the number of samples.

Table 5 Complex Matrix Y Values Through Column1 to Column4

Table: Complex Y Matrix Values Columns 1 through 4							
real	imag	real	imag	real	imag	real	imag
-1.1374	- 0.3006	-3.4809	- 0.4429	3.4221	- 1.0755	-0.2627	+ 0.5649
0.4710	+ 1.9812	0.3163	+ 2.0680	2.0329	- 1.3481	1.0987	+ 0.1538
2.7140	+ 0.7088	1.2537	+ 0.2085	-1.5069	- 0.8221	-0.2763	+ 0.4370
2.0598	- 2.6316	0.4229	- 0.7637	0.7584	+ 0.9263	0.2800	+ 0.2391
0.4976	- 2.0262	-1.3018	- 2.0633	0.5173	+ 1.3782	0.4313	- 0.7564
-0.0363	+ 0.9938	-2.8618	+ 1.9570	1.6480	- 0.0873	-0.0050	+ 0.5617
0.7150	+ 1.5629	0.1790	+ 1.1730	1.0403	- 1.1201	0.6320	- 0.3943
2.6651	- 0.5806	1.5113	+ 0.3953	0.6949	+ 0.4641	0.2999	- 0.4178

Because of the large dimension of Y, the values shown in four different tables. Table 5 shows the columns 1 to 4, Table 6 displays the columns 5 to 8, Table 7 displays the columns 9 to 12, and Table 8 displays the columns 13 to 16, respectively.

Table 6 Complex Matrix Y Values, from Column-5 to Column-8

Table: Complex Y Matrix Values Columns 5 through 8							
real	imag	real	imag	real	imag	real	imag
-2.4596	+ 0.1251	-4.8424	+ 0.1623	-1.0695	-0.5212	0.8846	+ 0.0799
-3.6230	- 0.7815	-2.8390	+ 2.5622	-0.2097	- 0.3168	0.8220	- 0.3460
-1.6859	- 0.3648	+0.9039	+ 0.6475	-0.1205	+ 1.0870	-1.4852	- 0.2444
-3.3010	+ 0.8543	-1.2557	- 1.5350	0.8478	- 0.0941	0.6929	+ 2.4757
-2.0858	+ 1.2204	-2.9441	- 1.4912	-1.0358	- 0.8402	2.5072	- 0.2656
-1.5432	- 0.3421	-4.1486	+ 0.8721	-0.8759	+ 0.9653	1.7246	- 1.5467
-2.4710	- 0.3458	-2.0982	+ 3.5279	0.4239	+ 1.0724	0.1124	- 1.4793
-3.7022	- 0.2444	-0.6404	- 0.1275	0.3806	+ 0.3294	0.0434	- 0.8615

Table 7 Complex Matrix Y Values, from Column-9 to Column-12

Table:Complex Y Matrix Values Columns 9 through 12							
real	imag	real	imag	real	imag	real	imag
-0.5660	- 0.3066	0.1470	- 0.9074	1.5529	- 0.3587	1.6580	- 0.7299
2.0192	+ 0.5517	-0.2517	- 1.7445	1.3046	- 0.3536	1.2023	+ 1.0694
2.1340	+ 0.1684	-0.2822	+ 0.5907	-0.6967	- 0.0767	1.8448	- 0.2346
1.8750	- 0.1326	0.2339	- 1.2230	-0.2526	+ 0.4493	1.0756	- 0.1976
-0.1509	- 0.2958	-0.6952	+ 0.1304	0.8120	+ 0.0127	0.3591	+ 0.5273
0.4360	+ 0.7716	0.9734	- 0.0099	1.4063	- 1.0510	-0.2733	- 0.1663
0.5403	+ 1.9442	0.3548	+ 1.4748	1.5642	- 0.3090	1.0226	+ 0.8590
1.3228	+ 0.9551	0.8440	- 0.0621	0.0895	- 1.0195	1.6075	- 0.2268

While constructing integer and fractional numbers of Y matrix in VHDL, the syntax of complex number in Figure 24 is used.

The same artificial clock, i.e., the same clock period, is used and green horizontal lines show indicate the values. Since the illustration of simulation results needs a large-size figure, not every value is shown. The values of 0th row, 1th row and 7th row are given in Figures 31, 32, and 33, respectively.

Every row and column of matrix has real (**.re**) and imaginer (**.im**) parts. The **re** and **im** parts of the first 3 columns of the 0th row are shown in more detail in Figure 31. The

remaining rows and columns have the same parts as well. Integer parts are represented with 4 bits while fractional parts are represented by 12 bits.

Table 8 Complex Y Matrix Values, from Column-13 to Column-16

Table:Complex Y Matrix Values Columns 13 through 16							
real	imag	real	imag	real	imag	real	imag
0.6537	+ 0.7162	-2.1960	+ 0.6955	-1.2288	- 0.2927	2.0488	+ 1.0012
-0.2870	- 0.5331	-2.1678	+ 0.2792	-0.9427	+ 0.0614	1.2332	- 1.9360
-1.2564	- 0.6215	-0.3948	+ 1.0445	-2.4781	+ 1.5954	-0.8250	+ 0.3417
-1.5941	+ 0.1393	-1.5668	- 0.7522	-1.9540	+ 0.3872	0.3795	+ 1.6466
0.0719	+ 0.0951	-1.1319	+ 0.2708	-2.5051	- 0.6216	0.5822	+ 0.7158
0.6698	- 0.5415	-3.1081	- 0.1938	-1.3451	+ 0.5920	2.4950	+ 0.9476
-0.2276	- 1.3113	-1.6948	+ 0.6263	-0.1882	+ 0.1256	0.5332	- 0.7107
-1.1915	- 0.4205	-1.8074	+ 0.7523	-1.1224	+ 0.5491	0.9868	- 0.9826

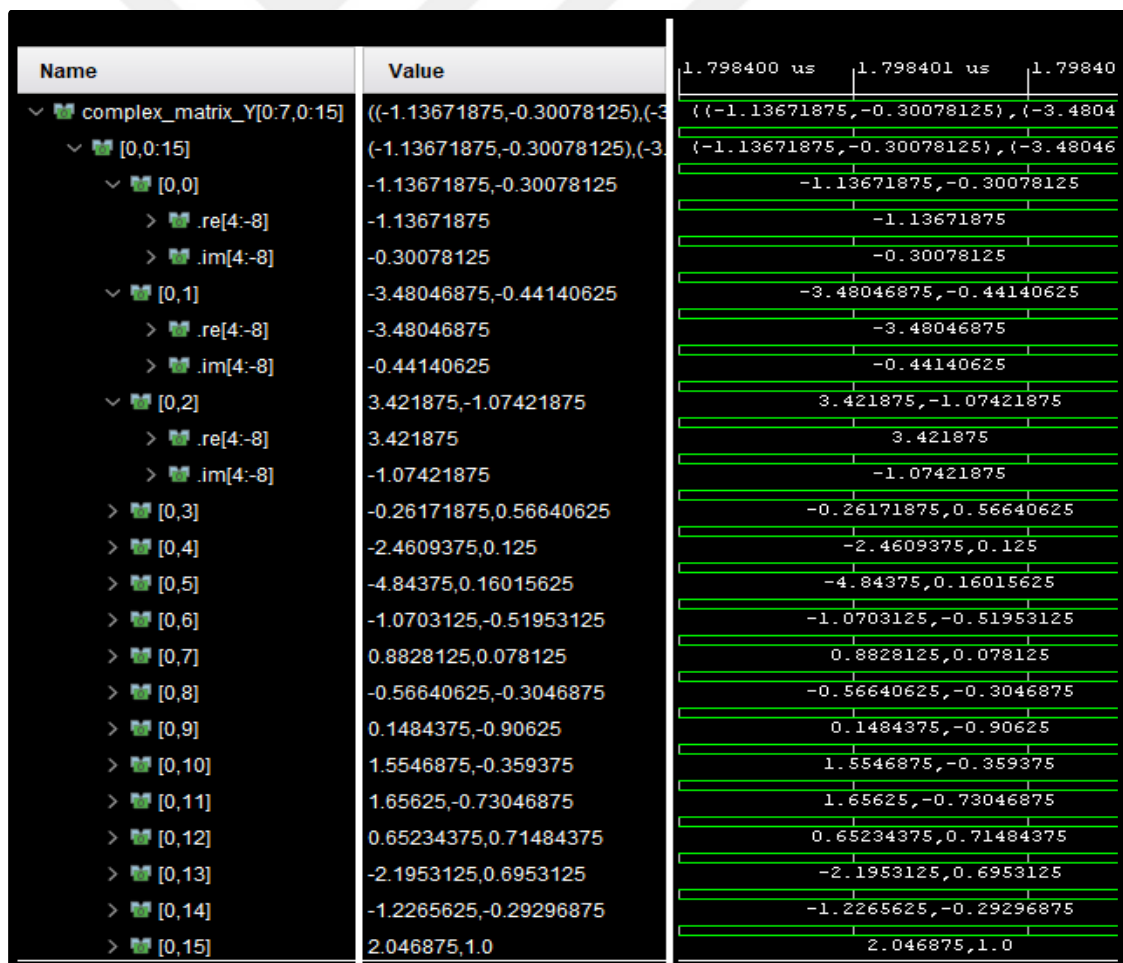


Figure 31 Vivado Generation of 0th row of Y matrix

Name	Value	1.798403 us	1.798404 us	1.798405 us
complex_matrix_Y[0:7,0:15]	((-1.13671875,-0.30078125),(-3.48046875,-0.30078125),(-1.13671875,-0.30078125),(-3.48046875,-0.30078125),(-1.13671875,-0.30078125),(-3.48046875,-0.30078125),(-1.13671875,-0.30078125),(-3.48046875,-0.30078125),(-1.13671875,-0.30078125),(-3.48046875,-0.30078125),(-1.13671875,-0.30078125),(-3.48046875,-0.30078125),(-1.13671875,-0.30078125),(-3.48046875,-0.30078125),(-1.13671875,-0.30078125),(-3.48046875,-0.30078125))			
> [0,0:15]	(-1.13671875,-0.30078125),(-3.48046875,-0.30078125),(-1.13671875,-0.30078125),(-3.48046875,-0.30078125),(-1.13671875,-0.30078125),(-3.48046875,-0.30078125),(-1.13671875,-0.30078125),(-3.48046875,-0.30078125),(-1.13671875,-0.30078125),(-3.48046875,-0.30078125),(-1.13671875,-0.30078125),(-3.48046875,-0.30078125),(-1.13671875,-0.30078125),(-3.48046875,-0.30078125),(-1.13671875,-0.30078125),(-3.48046875,-0.30078125)			
> [1,0:15]	(0.46875,1.98046875),(0.31640625,2.06640625),(-0.2109375,-0.31640625),(-0.2109375,-0.31640625),(-0.2109375,-0.31640625),(-0.2109375,-0.31640625),(-0.2109375,-0.31640625),(-0.2109375,-0.31640625),(-0.2109375,-0.31640625),(-0.2109375,-0.31640625),(-0.2109375,-0.31640625),(-0.2109375,-0.31640625),(-0.2109375,-0.31640625),(-0.2109375,-0.31640625),(-0.2109375,-0.31640625),(-0.2109375,-0.31640625)			
> [1,0]	0.46875,1.98046875		0.46875,1.98046875	
> [1,0].re[4:-8]	0.46875		0.46875	
> [1,0].im[4:-8]	1.98046875		1.98046875	
> [1,1]	0.31640625,2.06640625		0.31640625,2.06640625	
> [1,1].re[4:-8]	0.31640625		0.31640625	
> [1,1].im[4:-8]	2.06640625		2.06640625	
> [1,2]	2.03125,-1.34765625		2.03125,-1.34765625	
> [1,3]	1.09765625,0.15234375		1.09765625,0.15234375	
> [1,4]	-3.62109375,-0.78125		-3.62109375,-0.78125	
> [1,5]	-2.83984375,2.5625		-2.83984375,2.5625	
> [1,6]	-0.2109375,-0.31640625		-0.2109375,-0.31640625	
> [1,7]	0.8203125,-0.34375		0.8203125,-0.34375	
> [1,8]	2.01953125,0.55078125		2.01953125,0.55078125	
> [1,9]	-0.25,-1.7421875		-0.25,-1.7421875	
> [1,10]	1.3046875,-0.3515625		1.3046875,-0.3515625	
> [1,11]	1.203125,1.0703125		1.203125,1.0703125	
> [1,12]	-0.28515625,-0.53125		-0.28515625,-0.53125	
> [1,13]	-2.16796875,0.27734375		-2.16796875,0.27734375	
> [1,14]	-0.94140625,0.0625		-0.94140625,0.0625	
> [1,15]	1.234375,-1.9375		1.234375,-1.9375	

Figure 32 Vivado Generation of 1th row of Y matrix

Name	Value	1.798400 us	1.798401 us	1.79840
> complex_matrix_Y[0:7,0:15]	((-1.13671875,-0.30078125),(-3.4804	((-1.13671875,-0.30078125),(-3.4804		
v [7,0:15]	(2.6640625,-0.58203125),(1.51171875	(2.6640625,-0.58203125),(1.51171875		
v [7,0]	2.6640625,-0.58203125	2.6640625,-0.58203125		
> .re[4:-8]	2.6640625	2.6640625		
> .im[4:-8]	-0.58203125	-0.58203125		
v [7,1]	1.51171875,0.39453125	1.51171875,0.39453125		
> .re[4:-8]	1.51171875	1.51171875		
> .im[4:-8]	0.39453125	0.39453125		
> [7,2]	0.6953125,0.46484375	0.6953125,0.46484375		
> [7,3]	0.30078125,-0.41796875	0.30078125,-0.41796875		
> [7,4]	-3.703125,-0.2421875	-3.703125,-0.2421875		
> [7,5]	-0.640625,-0.12890625	-0.640625,-0.12890625		
> [7,6]	0.3828125,0.328125	0.3828125,0.328125		
> [7,7]	0.04296875,-0.859375	0.04296875,-0.859375		
> [7,8]	1.32421875,0.953125	1.32421875,0.953125		
> [7,9]	0.84375,-0.0625	0.84375,-0.0625		
> [7,10]	0.08984375,-1.01953125	0.08984375,-1.01953125		
> [7,11]	1.609375,-0.2265625	1.609375,-0.2265625		
> [7,12]	-1.19140625,-0.421875	-1.19140625,-0.421875		
> [7,13]	-1.8046875,0.75	-1.8046875,0.75		
> [7,14]	-1.12109375,0.546875	-1.12109375,0.546875		
> [7,15]	0.98828125,-0.984375	0.98828125,-0.984375		

Figure 33 Vivado Generation of 7th row of Y matrix

5.3. VHDL Implementation of Beamforming Algorithm

In this section, VHDL implementation of conventional beamforming, known as Bartlett beamforming, is explained. According to expression (3.50) and (3.51) that is given in Chapter 3, we have

$$\hat{\mathbf{R}} = \frac{1}{N} \sum_{n=1}^N \mathbf{y}(n) \mathbf{y}^*(n) \quad (3.50)$$

$$\mathbf{a}^*(\theta) \mathbf{R} \mathbf{a}(\theta) \quad (3.51)$$

In Figure 34, MATLAB implementation of beamforming algorithm is given.

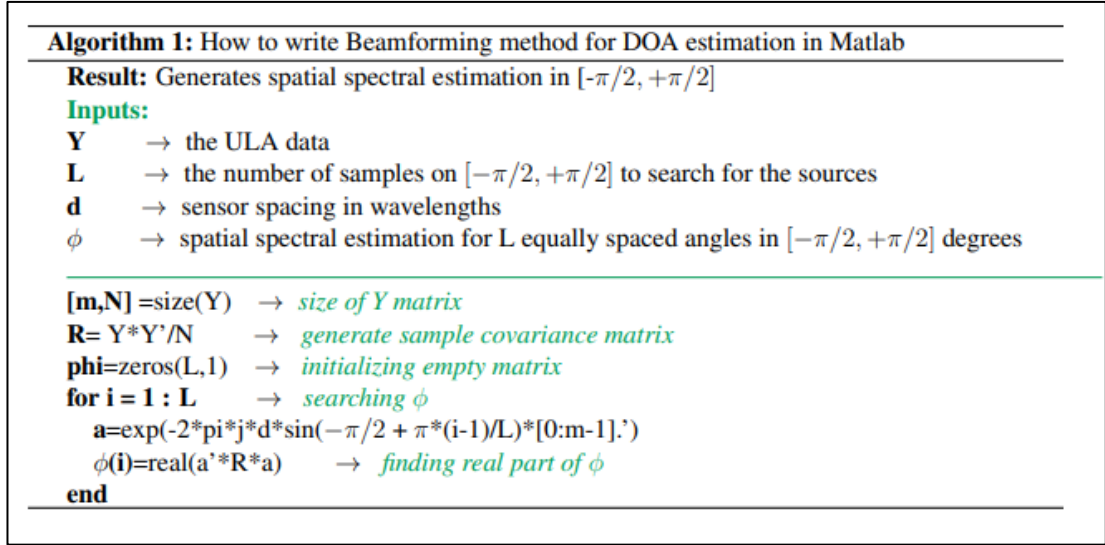


Figure 34 Beamforming algorithm in MATLAB

It is seen from Figure 34 that in order to calculate sample covariance matrix, first we need to calculate the complex conjugate transpose of Y matrix.

5.3.1. VHDL Generation of Complex Conjugate Transpose of Matrix Y

The complex conjugate transpose of a matrix is obtained by interchanging the row and column index for each element. The operation also negates the imaginary part of any complex numbers [72].

The complex Y matrix is generated in previous subsection. By using these values complex conjugate transpose of Y matrix will be calculated. Figure 35 shows the transpose calculation in VHDL.

In transpose function implementation, rows and columns of $\mathbf{Y}_{8 \times 16}$ matrix are interchanged and the resulting matrix $\mathbf{Y}'_{16 \times 8}$ is returned. The matrices contain complex numbers. The function takes $\mathbf{Y}_{8 \times 16}$ matrix as input and returns $\mathbf{Y}'_{16 \times 8}$ as output.

```

function transpose8x16( cplx8_l6 : re_im_matrix_8x16) return re_im_matrix_16x8 is
variable i,j : integer :=0;
variable ret: re_im_matrix_16x8;
begin
  for i in 0 to num_rows -1 loop
    for j in 0 to num_cols -1 loop
      ret (j,i).re := cplx8_l6(i,j).re;
      ret (j,i).im := to_sfixed(-to_real(cplx8_l6(i,j).im) , nh,na);
    end loop;
  end loop;
  return ret;
end function;

```

Figure 35 Transpose function in VHDL

Due to the large dimensions of complex matrix $Y'_{16 \times 8}$, the values are shown in two different tables. Table 9 shows from column 1 to column 4, Table 10 shows column 5 to column 8, respectively.

Table 9 Complex Y Transpose Matrix Values Column1 to Column4

Table:Complex Y Transpose Matrix Values Columns 1 through 4							
real	imag	real	imag	real	imag	real	imag
-1.1374	+ 0.3006	0.4710	- 1.9812	2.7140	- 0.7088	2.0598	+ 2.6316
-3.4809	+ 0.4429	0.3163	- 2.0680	1.2537	- 0.2085	0.4229	+ 0.7637
3.4221	+ 1.0755	2.0329	+ 1.3481	-1.5069	+ 0.8221	0.7584	- 0.9263
-0.2627	- 0.5649	1.0987	- 0.1538	-0.2763	- 0.4370	0.2800	- 0.2391
-2.4596	- 0.1251	-3.6230	+ 0.7815	-1.6859	+ 0.3648	-3.3010	- 0.8543
-4.8424	- 0.1623	-2.8390	- 2.5622	0.9039	- 0.6475	-1.2557	+ 1.5350
-1.0695	+ 0.5212	-0.2097	+ 0.3168	-0.1205	- 1.0870	0.8478	+ 0.0941
0.8846	- 0.0799	0.8220	+ 0.3460	-1.4852	+ 0.2444	0.6929	- 2.4757
-0.5660	+ 0.3066	2.0192	- 0.5517	2.1340	- 0.1684	1.8750	+ 0.1326
0.1470	+ 0.9074	-0.2517	+ 1.7445	-0.2822	- 0.5907	0.2339	+ 1.2230
1.5529	+ 0.3587	1.3046	+ 0.3536	-0.6967	+ 0.0767	-0.2526	- 0.4493
1.6580	+ 0.7299	1.2023	- 1.0694	1.8448	+ 0.2346	1.0756	+ 0.1976
0.6537	- 0.7162	-0.2870	+ 0.5331	-1.2564	+ 0.6215	-1.5941	- 0.1393
-2.1960	- 0.6955	-2.1678	- 0.2792	-0.3948	- 1.0445	-1.5668	+ 0.7522
-1.2288	+ 0.2927	-0.9427	- 0.0614	-2.4781	- 1.5954	-1.9540	- 0.3872
2.0488	- 1.0012	1.2332	+ 1.9360	-0.8250	- 0.3417	0.3795	- 1.6466

Table 10 Complex Y Transpose Matrix Values Column5 to Column8

Table:Complex Y Transpose Matrix Values Columns 5 through 8							
real	imag	real	imag	real	imag	real	imag
0.4976	+ 2.0262	-0.0363	- 0.9938	0.7150	- 1.5629	2.6651	+ 0.5806
-1.3018	+ 2.0633	-2.8618	- 1.9570	0.1790	- 1.1730	1.5113	- 0.
0.5173	- 1.3782	1.6480	+ 0.0873	1.0403	+ 1.1201	0.6949	- 0.4641
0.4313	+ 0.7564	-0.0050	- 0.5617	0.6320	+ 0.3943	0.2999	+ 0.4178
-2.0858	- 1.2204	-1.5432	+ 0.3421	-2.4710	+ 0.3458	-3.7022	+ 0.2444
-2.9441	+ 1.4912	-4.1486	- 0.8721	-2.0982	- 3.5279	-0.6404	+ 0.1275
-1.0358	+ 0.8402	-0.8759	- 0.9653	0.4239	- 1.0724	0.3806	- 0.3294
2.5072	+ 0.2656	1.7246	+ 1.5467	0.1124	+ 1.4793	0.0434	+ 0.8615
-0.1509	+ 0.2958	0.4360	- 0.7716	0.5403	- 1.9442	1.3228	- 0.9551
-0.6952	- 0.1304	0.9734	+ 0.0099	0.3548	- 1.4748	0.8440	+ 0.0621
0.8120	- 0.0127	1.4063	+ 1.0510	1.5642	+ 0.3090	0.0895	+ 1.0195
0.3591	- 0.5273	-0.2733	+ 0.1663	1.0226	- 0.8590	1.6075	+ 0.2268
0.0719	- 0.0951	0.6698	+ 0.5415	-0.2276	+ 1.3113	-1.1915	+ 0.4205
-1.1319	- 0.2708	-3.1081	+ 0.1938	-1.6948	- 0.6263	-1.8074	- 0.7523
-2.5051	+ 0.6216	-1.3451	- 0.5920	-0.1882	- 0.1256	-1.1224	- 0.5491
0.5822	- 0.7158	2.4950	- 0.9476	0.5332	+ 0.7107	0.9868	+ 0.9826

Name	Value	
clk	1	3.499965 us
complexMatrix_Y_tr[0:15,0:7]	((1edd,004d),(0078,1e05),(02b7,1f4a),	{(1edd,004d),(0078,1e05),(02b7,1f4a),
[0,0:7]	(-1.13671875,0.30078125),(0.46875,-	(-1.13671875,0.30078125),(0.46875,-
[0,0]	-1.13671875,0.30078125	-1.13671875,0.30078125
.re[4:-8]	-1.13671875	-1.13671875
.im[4:-8]	0.30078125	0.30078125
[0,1]	0.46875,-1.98046875	0.46875,-1.98046875
[0,2]	2.71484375,-0.7109375	2.71484375,-0.7109375
[0,3]	2.05859375,2.6328125	2.05859375,2.6328125
[0,4]	0.49609375,2.02734375	0.49609375,2.02734375
[0,5]	-0.03515625,-0.9921875	-0.03515625,-0.9921875
[0,6]	0.71484375,-1.5625	0.71484375,-1.5625
[0,7]	2.6640625,0.58203125	2.6640625,0.58203125
[1,0:7]	(-3.48046875,0.44140625),(0.3164062	(-3.48046875,0.44140625),(0.3164062
[1,0]	-3.48046875,0.44140625	-3.48046875,0.44140625
.re[4:-8]	-3.48046875	-3.48046875
.im[4:-8]	0.44140625	0.44140625
[1,1]	0.31640625,-2.06640625	0.31640625,-2.06640625
[1,2]	1.25390625,-0.20703125	1.25390625,-0.20703125
[1,3]	0.421875,0.765625	0.421875,0.765625
[1,4]	-1.30078125,2.0625	-1.30078125,2.0625
[1,5]	-2.86328125,-1.95703125	-2.86328125,-1.95703125
[1,6]	0.1796875,-1.171875	0.1796875,-1.171875
[1,7]	1.51171875,-0.39453125	1.51171875,-0.39453125

Figure 36 Vivado Generation of 0th and 1th row of Y' matrix

In VHDL coding, while constructing integer and fractional bit numbers of $Y'_{16 \times 8}$ matrix, the syntax of complex number in Figure 24 is used. Integer parts are represented with 4 bits, fractional part that is represented by 12 bits.

The same artificial clock, i.e., the same clock period is used. The values of first two rows which are 0th row and 1th row are depicted in Figure 36. The last two rows which are 14th row and 15th row are illustrated in Figure 37.

Every row and column of the matrix has real (.re) and imaginary (.im) parts. The **re** and **im** parts of some values are shown in more details for demonstration in Figure 36. The remaining rows and columns have also **re** and **im** parts.

Name	Value	
> [13,0:7]	(-2.1933125,-0.0933125),(-2.10790625,-0.0933125),(-2.0125,-0.0933125),(-1.9170875,-0.0933125),(-1.821675,-0.0933125),(-1.72625,-0.0933125),(-1.630825,-0.0933125),(-1.535375,-0.0933125)	4.500000 us 5.000000 us
▼ [14,0:7]	(-1.2265625,0.29296875),(-0.94140625,0.29296875),(-0.65625,0.29296875),(-0.37109375,0.29296875),(-0.0859375,0.29296875),(-0.18046875,0.29296875),(-0.465625,0.29296875),(-0.7503125,0.29296875)	(-1.2265625,0.29296875),(-0.94140625,0.29296875),(-0.65625,0.29296875),(-0.37109375,0.29296875),(-0.0859375,0.29296875),(-0.18046875,0.29296875),(-0.465625,0.29296875),(-0.7503125,0.29296875)
> [14,0]	-1.2265625,0.29296875	-1.2265625,0.29296875
> [14,1]	-0.94140625,-0.0625	-0.94140625,-0.0625
> [14,2]	-2.4765625,-1.59375	-2.4765625,-1.59375
> [14,3]	-1.953125,-0.38671875	-1.953125,-0.38671875
> [14,4]	-2.50390625,0.62109375	-2.50390625,0.62109375
> [14,5]	-1.34375,-0.59375	-1.34375,-0.59375
> [14,6]	-0.1875,-0.125	-0.1875,-0.125
> [14,7]	-1.12109375,-0.546875	-1.12109375,-0.546875
▼ [15,0:7]	(2.046875,-1.0),(1.234375,1.9375),(-0.82421875,-0.34375),0.37890625,-1.6484375,0.58203125,-0.71484375,2.49609375,-0.94921875,0.53125,0.7109375,0.98828125,0.984375	(2.046875,-1.0),(1.234375,1.9375),(-0.82421875,-0.34375),0.37890625,-1.6484375,0.58203125,-0.71484375,2.49609375,-0.94921875,0.53125,0.7109375,0.98828125,0.984375
> [15,0]	2.046875,-1.0	2.046875,-1.0
> [15,1]	1.234375,1.9375	1.234375,1.9375
> [15,2]	-0.82421875,-0.34375	-0.82421875,-0.34375
> [15,3]	0.37890625,-1.6484375	0.37890625,-1.6484375
> [15,4]	0.58203125,-0.71484375	0.58203125,-0.71484375
> [15,5]	2.49609375,-0.94921875	2.49609375,-0.94921875
> [15,6]	0.53125,0.7109375	0.53125,0.7109375
> [15,7]	0.98828125,0.984375	0.98828125,0.984375

Figure 37 Vivado Generation of 14th and 15th row of Y' matrix

5.3.2. VHDL Generation of Sample Covariance Matrix $\hat{\mathbf{R}}$

After calculating the Y matrix and its complex conjugate transpose, we multiply these two matrices to get a sample covariance matrix in (3.50)

$$\hat{\mathbf{R}} = \frac{1}{N} \sum_{n=1}^N \mathbf{y}(n) \mathbf{y}^*(n) \quad (3.50)$$

Multiplication of matrices can be achieved using row-to-column wise multiplication and addition. It means that the row elements of the first matrix are multiplied by the column elements of the second matrix and added. The important point is that the column size of the first matrix and the row size of the second matrix must be equal to each other. In this case, the product matrix has the same number of rows as the first matrix and the same number of columns as the second matrix.

If it is known that the dimension of Y matrix is 8x16, and dimension of its complex conjugate transpose is 16x8, the dimension of sample covariance matrix will be 8x8.

In VHDL, first, a complex matrix multiplication function is written, and next covariance matrix is calculated. Subsequently, sample covariance matrix is obtained by dividing the covariance matrix by the number of samples, which is set to 16.

The elements of product matrix is represented with signed fixed-point complex numbers.

After the multiplication and addition, the bit numbers of the product matrix should be rearranged again.

The maximum value of multiplication should be determined, and the number of bits should be determined again.

The values resulting from the product matrix should be prevented from overflow, losing the sign bit or expressing them with an insufficient number of bits may result in inaccurate calculations.

The following expressions show the complex matrix multiplication operation

$$\sum_{i=1}^N x_i y_i \quad (5.1)$$

$$P_{real} = x_{real} y_{real} - x_{imag} y_{imag} \quad (5.2)$$

$$P_{imag} = x_{real} y_{imag} + x_{imag} y_{real} \quad (5.3)$$

$$S_{real} = x_{real} + y_{real} \quad (5.4)$$

$$S_{imag} = x_{imag} + y_{imag} \quad (5.5)$$

where x_i and y_i represent the complex numbers that consists real and imaginary parts. N shows the length of sequences. Their product is represented by P_{real} and P_{imag} and their summation is represented with S_{real} and S_{imag} [71]

Table 11 Covariance Matrix **R** Values, from Column-1 to Column-4

Table : Covariance Matrix "R" Values Columns 1 through 4							
real	imag	real	imag	real	imag	real	imag
78.8210	+ 0.0000	38.8365	+24.3809	-11.7558	+ 9.7583	21.1717	-26.5157
38.8365	-24.3809	68.2160	+ 0.0000	12.4721	+25.1336	14.5335	- 3.3731
-11.7558	- 9.7583	12.4721	-25.1336	42.0263	+ 0.0000	16.3260	+17.6441
21.1717	+26.5157	14.5335	+ 3.3731	16.3260	-17.6441	55.3453	+ 0.0000
37.6569	+21.2437	10.5129	+10.2073	-4.0787	- 3.1971	30.2247	-17.6382
56.8879	-14.3499	42.4491	+26.3327	-3.7473	+19.0841	10.2151	-14.0169
22.7522	-27.5953	45.1999	- 5.2859	13.7916	+22.4217	-0.6278	+ 2.4498
12.9612	- 0.0557	31.8641	- 9.8194	24.7407	+ 1.0615	28.6861	+ 8.3528

Because of the large dimensions of covariance matrix **R**, the values are shown in two different tables. Table 11 shows the values from column-1 to column-4, Table 12 shows the values from column-5 to column-8, respectively.

Table 12 Covariance Matrix **R** Values, from Column-5 to Column-8

Table : Covariance Matrix "R" Values Columns 5 through 8							
real	imag	real	imag	real	imag	real	imag
37.6569	-21.2437	56.8879	+14.3499	22.7522	+27.5953	12.9612	+ 0.0557
10.5129	-10.2073	42.4491	-26.3327	45.1999	+ 5.2859	31.8641	+ 9.8194
-4.0787	+ 3.1971	-3.7473	-19.0841	13.7916	-22.4217	24.7407	- 1.0615
30.2247	+17.6382	10.2151	+14.0169	-0.6278	- 2.4498	28.6861	- 8.3528
48.6558	+ 0.0000	25.0724	+24.9113	2.6013	+18.1047	14.2361	- 4.1142
25.0724	-24.9113	68.2233	+ 0.0000	34.6135	+26.0242	17.5765	+16.6763
2.6013	-18.1047	34.6135	-26.0242	50.4332	+ 0.0000	24.7548	+11.5147
14.2361	+ 4.1142	17.5765	-16.6763	24.7548	-11.5147	42.0234	+ 0.0000

Since the result is an 8x8 matrix, it was not possible to show its values with a single figure. The Vivado generated covariance matrix is shown by 4 figures, each containing the values of 2 rows.

Figure 38 shows 0th and 1th row, Figure 39 shows 2th and 3th row, Figure 40 shows 4th and 5th row, Figure 41 shows 6th and 7th row, respectively.

Name	Value	
covariance_matrix[0:7,0:7]	((4.92520141601563,0.0),(2.4268798	5.000000 us ((4.92520141601563,0.0) ,(2.4268
[0,0:7]	(78.80322265625,0.0),(38.830078125	(78.80322265625,0.0) ,(38.830078
> [0,0]	78.80322265625,0.0	78.80322265625,0.0
> [0,1]	38.830078125,24.379150390625	38.830078125,24.379150390625
> [0,2]	-11.74609375,9.74560546875	-11.74609375,9.74560546875
> [0,3]	21.17724609375,-26.505126953125	21.17724609375,-26.505126953125
> [0,4]	37.647705078125,-21.25	37.647705078125,-21.25
> [0,5]	56.894287109375,14.328857421875	56.894287109375,14.328857421875
> [0,6]	22.74462890625,27.595458984375	22.74462890625,27.595458984375
> [0,7]	12.968017578125,0.0458984375	12.968017578125,0.0458984375
[1,0:7]	(38.830078125,-24.379150390625),((38.830078125,-24.379150390625)
> [1,0]	38.830078125,-24.379150390625	38.830078125,-24.379150390625
> [1,1]	68.178466796875,0.0	68.178466796875,0.0
> [1,2]	12.462158203125,25.11328125	12.462158203125,25.11328125
> [1,3]	14.51025390625,-3.39404296875	14.51025390625,-3.39404296875
> [1,4]	10.5068359375,-10.22021484375	10.5068359375,-10.22021484375
> [1,5]	42.435791015625,-26.342529296875	42.435791015625,-26.342529296875
> [1,6]	45.165771484375,5.298583984375	45.165771484375,5.298583984375
> [1,7]	31.843505859375,9.813720703125	31.843505859375,9.813720703125

Figure 38 Vivado Generation of 0th and 1th row of *R* matrix

Name	Value	
covariance_matrix[0:7,0:7]	((4.92520141601563,0.0),(2.4268798	5.000000 us {(4.92520141601563,0.0),(2.4268798
> [0,0:7]	(78.80322265625,0.0),(38.830078125	{(78.80322265625,0.0),(38.830078125
> [1,0:7]	(38.830078125,-24.379150390625),({(38.830078125,-24.379150390625),(
> [2,0:7]	(-11.74609375,-9.74560546875),(12.4	{(-11.74609375,-9.74560546875),0
> [2,0]	-11.74609375,-9.74560546875	{-11.74609375,-9.74560546875
> [2,1]	12.462158203125,-25.11328125	{12.462158203125,-25.11328125
> [2,2]	42.006103515625,0.0	{42.006103515625,0.0
> [2,3]	16.33056640625,17.65673828125	{16.33056640625,17.65673828125
> [2,4]	-4.076171875,3.21240234375	{-4.076171875,3.21240234375
> [2,5]	-3.74560546875,-19.065673828125	{-3.74560546875,-19.065673828125
> [2,6]	13.783203125,-22.41796875	{13.783203125,-22.41796875
> [2,7]	24.727294921875,-1.041259765625	{24.727294921875,-1.041259765625
> [3,0:7]	(21.17724609375,26.505126953125)	{(21.17724609375,26.505126953125)
> [3,0]	21.17724609375,26.505126953125	{21.17724609375,26.505126953125
> [3,1]	14.51025390625,3.39404296875	{14.51025390625,3.39404296875
> [3,2]	16.33056640625,-17.65673828125	{16.33056640625,-17.65673828125
> [3,3]	55.3427734375,0.0	{55.3427734375,0.0
> [3,4]	30.228271484375,17.640869140625	{30.228271484375,17.640869140625
> [3,5]	10.222900390625,14.024169921875	{10.222900390625,14.024169921875
> [3,6]	-0.641357421875,-2.4423828125	{-0.641357421875,-2.4423828125
> [3,7]	28.690673828125,-8.3544921875	{28.690673828125,-8.3544921875

Figure 39 Vivado Generation of 2th and 3th row of R matrix

Name	Value	
covariance_matrix[0:7,0:7]	((4.92520141601563,0.0),(2.426879882	(4.92520141601563,0.0),(2.426879882
> [0,0:7]	(78.80322265625,0.0),(38.830078125,2	(78.80322265625,0.0),(38.830078125,2
> [1,0:7]	(38.830078125,-24.379150390625),(68.	(38.830078125,-24.379150390625),(68.
> [2,0:7]	(-11.74609375,-9.74560546875),(12.462	(-11.74609375,-9.74560546875),(12.462
> [3,0:7]	(21.17724609375,26.505126953125),(1	(21.17724609375,26.505126953125),(1
> [4,0:7]	(37.647705078125,21.25),(10.50683593	(37.647705078125,21.25),(10.50683593
> [4,0]	37.647705078125,21.25	37.647705078125,21.25
> [4,1]	10.5068359375,10.22021484375	10.5068359375,10.22021484375
> [4,2]	-4.076171875,-3.21240234375	-4.076171875,-3.21240234375
> [4,3]	30.228271484375,-17.640869140625	30.228271484375,-17.640869140625
> [4,4]	48.656494140625,0.0	48.656494140625,0.0
> [4,5]	25.082275390625,24.909423828125	25.082275390625,24.909423828125
> [4,6]	2.596435546875,18.10400390625	2.596435546875,18.10400390625
> [4,7]	14.242431640625,-4.123291015625	14.242431640625,-4.123291015625
> [5,0:7]	(56.894287109375,-14.328857421875),	(56.894287109375,-14.328857421875),
> [5,0]	56.894287109375,-14.328857421875	56.894287109375,-14.328857421875
> [5,1]	42.435791015625,26.342529296875	42.435791015625,26.342529296875
> [5,2]	-3.74560546875,19.065673828125	-3.74560546875,19.065673828125
> [5,3]	10.222900390625,-14.024169921875	10.222900390625,-14.024169921875
> [5,4]	25.082275390625,-24.909423828125	25.082275390625,-24.909423828125
> [5,5]	68.254150390625,0.0	68.254150390625,0.0
> [5,6]	34.596435546875,26.02685546875	34.596435546875,26.02685546875
> [5,7]	17.5595703125,16.686767578125	17.5595703125,16.686767578125

Figure 40 Vivado Generation of 4th and 5th row of R matrix

Name	Value	1.000000 us	2.000000 us
> [4,0:7]	(37.647705078125,21.25),(10.5068359375,10.5068359375)	(37.647705078125,21.25)	(10.5068359375,10.5068359375)
> [5,0:7]	(56.894287109375,-14.328857421875),(10.5068359375,10.5068359375)	(56.894287109375,-14.328857421875)	(10.5068359375,10.5068359375)
√ [6,0:7]	(22.74462890625,-27.595458984375),(42.1875,42.1875)	(22.74462890625,-27.595458984375)	(42.1875,42.1875)
> [6,0]	22.74462890625,-27.595458984375	22.74462890625,-27.595458984375	
> [6,1]	45.165771484375,-5.298583984375	45.165771484375,-5.298583984375	
> [6,2]	13.783203125,22.41796875	13.783203125,22.41796875	
> [6,3]	-0.641357421875,2.4423828125	-0.641357421875,2.4423828125	
> [6,4]	2.596435546875,-18.10400390625	2.596435546875,-18.10400390625	
> [6,5]	34.596435546875,-26.02685546875	34.596435546875,-26.02685546875	
> [6,6]	50.403564453125,0.0	50.403564453125,0.0	
> [6,7]	24.724853515625,11.517822265625	24.724853515625,11.517822265625	
√ [7,0:7]	(12.968017578125,-0.0458984375),(31.843505859375,-9.813720703125)	(12.968017578125,-0.0458984375)	(31.843505859375,-9.813720703125)
> [7,0]	12.968017578125,-0.0458984375	12.968017578125,-0.0458984375	
> [7,1]	31.843505859375,-9.813720703125	31.843505859375,-9.813720703125	
> [7,2]	24.727294921875,1.041259765625	24.727294921875,1.041259765625	
> [7,3]	28.690673828125,8.3544921875	28.690673828125,8.3544921875	
> [7,4]	14.242431640625,4.123291015625	14.242431640625,4.123291015625	
> [7,5]	17.5595703125,-16.686767578125	17.5595703125,-16.686767578125	
> [7,6]	24.724853515625,-11.517822265625	24.724853515625,-11.517822265625	
> [7,7]	42.020263671875,0.0	42.020263671875,0.0	

Figure 41 Vivado Generation of 6th and 7th row of R matrix

At the remaining step, the sample covariance matrix \hat{R} is obtained by dividing the covariance matrix by the number of samples. Since the number of samples is 16, and it can be represented as 2^4 . The elements of the covariance matrix are shifted to the right 4 times; such a shifting result in division by 16. Afterwards, by using **resize** function, integer and fractional parts can be re-adjusted by using a sufficient number of bits.

Table 13 Sample Covariance Matrix \hat{R} Values Column1 to Column4

real	imag	real	imag	real	imag	real	imag
4.9263	+ 0.0000	2.4273	+ 1.5238	-0.7347	+ 0.6099	1.3232	- 1.6572
2.4273	- 1.5238	4.2635	+ 0.0000	0.7795	+ 1.5709	0.9083	- 0.2108
-0.7347	- 0.6099	0.7795	- 1.5709	2.6266	+ 0.0000	1.0204	+ 1.1028
1.3232	+ 1.6572	0.9083	+ 0.2108	1.0204	- 1.1028	3.4591	+ 0.0000
2.3536	+ 1.3277	0.6571	+ 0.6380	-0.2549	- 0.1998	1.8890	- 1.1024
3.5555	- 0.8969	2.6531	+ 1.6458	-0.2342	+ 1.1928	0.6384	- 0.8761
1.4220	- 1.7247	2.8250	- 0.3304	0.8620	+ 1.4014	-0.0392	+ 0.1531
0.8101	- 0.0035	1.9915	- 0.6137	1.5463	+ 0.0663	1.7929	+ 0.5221

Since the result is an 8x8 matrix again, it was not possible to show its values in a single figure. The Vivado generated $\hat{\mathbf{R}}$ matrix is shown in 4 figures, each containing the values of 2 rows.

Figure 42 shows 0th and 1th row, Figure 43 shows 2th and 3th row, Figure 44 shows 4th and 5th row, Figure 45 shows 6th and 7th row, respectively.

When the values in Table 11 and Table 12 are divided by 16, the expected values are obtained in Table 13 and Table 14. Vivado generations satisfy desired accuracy.

For example, the real value of 0th row and 0th column in Figure 38 is 78.833 and the real value of 0th row and 0th column in Figure 42 is 4.9252.

Likewise, the real value of 1th row and 0th column in Figure 38 is 38.830078, and the real value of 1th row and 0th column in Figure 42 is 2.426879.

The generated values with the ideal values are compared in this way.

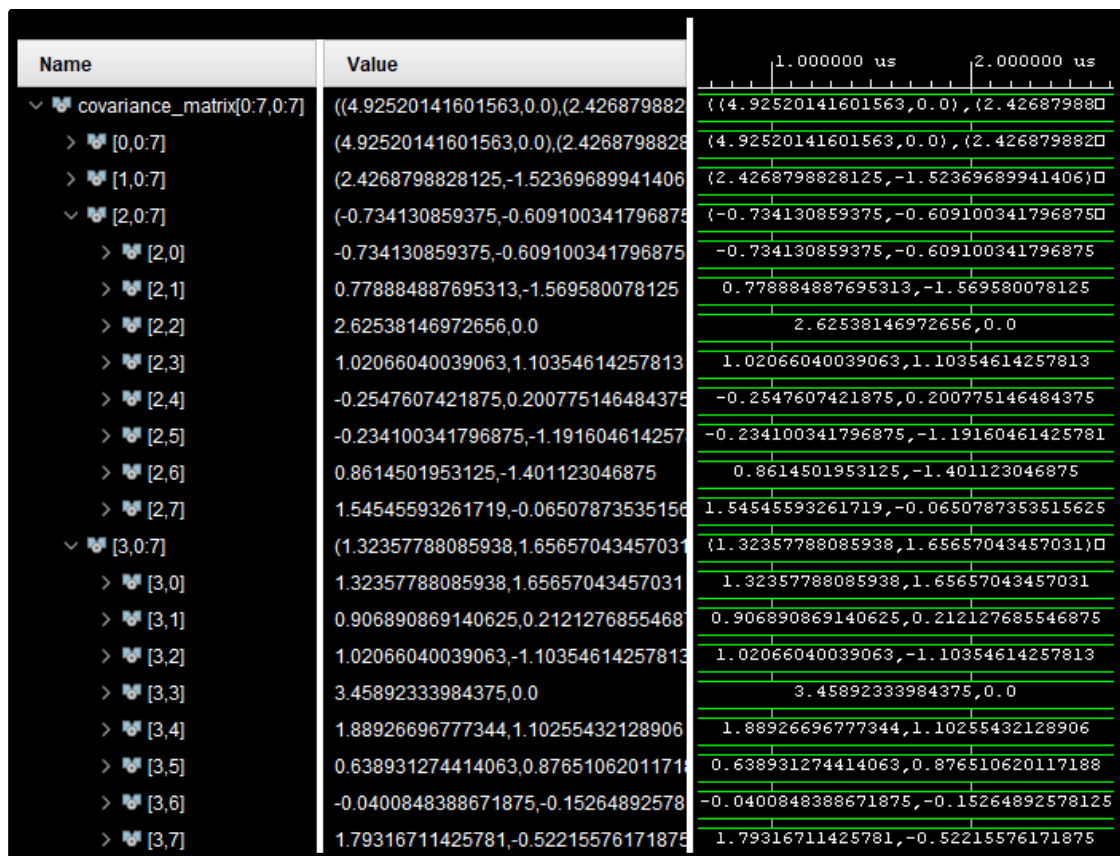


Figure 43 Vivado Generation of 2th and 3th row of $\hat{\mathbf{R}}$ matrix

Name	Value	
▼ covariance_matrix[0:7,0:7]	((4.92520141601563,0.0),(2.4268798828125,-1.52369689941406100633594,-0.8955535888671875,3.55589294433594,-0.8955535888671875,2.65223693847656,1.64640808105469,-0.234100341796875,1.19160461425781,0.638931274414063,-0.8765106201171875,1.56764221191406,-1.55683898925781,4.26588439941406,0.0,2.16227722167969,1.62667846679688,1.09747314453125,1.04292297363281))	1.000000 us 2.000000 us
> [0,0:7]	(4.92520141601563,0.0),(2.4268798828125,-1.52369689941406100633594,-0.8955535888671875,3.55589294433594,-0.8955535888671875,2.65223693847656,1.64640808105469,-0.234100341796875,1.19160461425781,0.638931274414063,-0.8765106201171875,1.56764221191406,-1.55683898925781,4.26588439941406,0.0,2.16227722167969,1.62667846679688,1.09747314453125,1.04292297363281)	(4.92520141601563,0.0),(2.4268798828125,-1.52369689941406100633594,-0.8955535888671875,3.55589294433594,-0.8955535888671875,2.65223693847656,1.64640808105469,-0.234100341796875,1.19160461425781,0.638931274414063,-0.8765106201171875,1.56764221191406,-1.55683898925781,4.26588439941406,0.0,2.16227722167969,1.62667846679688,1.09747314453125,1.04292297363281)
> [1,0:7]	(2.4268798828125,-1.52369689941406100633594,-0.8955535888671875,3.55589294433594,-0.8955535888671875,2.65223693847656,1.64640808105469,-0.234100341796875,1.19160461425781,0.638931274414063,-0.8765106201171875,1.56764221191406,-1.55683898925781,4.26588439941406,0.0,2.16227722167969,1.62667846679688,1.09747314453125,1.04292297363281)	(2.4268798828125,-1.52369689941406100633594,-0.8955535888671875,3.55589294433594,-0.8955535888671875,2.65223693847656,1.64640808105469,-0.234100341796875,1.19160461425781,0.638931274414063,-0.8765106201171875,1.56764221191406,-1.55683898925781,4.26588439941406,0.0,2.16227722167969,1.62667846679688,1.09747314453125,1.04292297363281)
> [2,0:7]	(-0.734130859375,-0.609100341796875,1.19160461425781,0.638931274414063,-0.8765106201171875,1.56764221191406,-1.55683898925781,4.26588439941406,0.0,2.16227722167969,1.62667846679688,1.09747314453125,1.04292297363281,1.09747314453125,1.04292297363281,1.09747314453125,1.04292297363281,1.09747314453125,1.04292297363281,1.09747314453125,1.04292297363281)	(-0.734130859375,-0.609100341796875,1.19160461425781,0.638931274414063,-0.8765106201171875,1.56764221191406,-1.55683898925781,4.26588439941406,0.0,2.16227722167969,1.62667846679688,1.09747314453125,1.04292297363281,1.09747314453125,1.04292297363281,1.09747314453125,1.04292297363281,1.09747314453125,1.04292297363281,1.09747314453125,1.04292297363281)
> [3,0:7]	(1.32357788085938,1.65657043457031,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375)	(1.32357788085938,1.65657043457031,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375)
▼ [4,0:7]	(2.35298156738281,1.328125,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375)	(2.35298156738281,1.328125,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375,0.65667724609375,0.638763427734375)
> [4,0]	2.35298156738281,1.328125	2.35298156738281,1.328125
> [4,1]	0.65667724609375,0.638763427734375	0.65667724609375,0.638763427734375
> [4,2]	-0.2547607421875,-0.200775146484375	-0.2547607421875,-0.200775146484375
> [4,3]	1.88926696777344,-1.10255432128906	1.88926696777344,-1.10255432128906
> [4,4]	3.04103088378906,0.0	3.04103088378906,0.0
> [4,5]	1.56764221191406,1.55683898925781	1.56764221191406,1.55683898925781
> [4,6]	0.162277221679688,1.13150024414063	0.162277221679688,1.13150024414063
> [4,7]	0.890151977539063,-0.257705688476563	0.890151977539063,-0.257705688476563
▼ [5,0:7]	(3.55589294433594,-0.8955535888671875,2.65223693847656,1.64640808105469,-0.234100341796875,1.19160461425781,0.638931274414063,-0.8765106201171875,1.56764221191406,-1.55683898925781,4.26588439941406,0.0,2.16227722167969,1.62667846679688,1.09747314453125,1.04292297363281,1.09747314453125,1.04292297363281,1.09747314453125,1.04292297363281)	(3.55589294433594,-0.8955535888671875,2.65223693847656,1.64640808105469,-0.234100341796875,1.19160461425781,0.638931274414063,-0.8765106201171875,1.56764221191406,-1.55683898925781,4.26588439941406,0.0,2.16227722167969,1.62667846679688,1.09747314453125,1.04292297363281,1.09747314453125,1.04292297363281,1.09747314453125,1.04292297363281)
> [5,0]	3.55589294433594,-0.8955535888671875	3.55589294433594,-0.8955535888671875
> [5,1]	2.65223693847656,1.64640808105469	2.65223693847656,1.64640808105469
> [5,2]	-0.234100341796875,1.19160461425781	-0.234100341796875,1.19160461425781
> [5,3]	0.638931274414063,-0.8765106201171875	0.638931274414063,-0.8765106201171875
> [5,4]	1.56764221191406,-1.55683898925781	1.56764221191406,-1.55683898925781
> [5,5]	4.26588439941406,0.0	4.26588439941406,0.0
> [5,6]	2.16227722167969,1.62667846679688	2.16227722167969,1.62667846679688
> [5,7]	1.09747314453125,1.04292297363281	1.09747314453125,1.04292297363281

Figure 44 Vivado Generation of 4th and 5th row of \hat{R} matrix

Name	Value	1.000000 us	2.000000 us
> [2,0:7]	(-0.734130859375,-0.609100341796875	(-0.734130859375,-0.609100341796875	
> [3,0:7]	(1.32357788085938,1.65657043457031	(1.32357788085938,1.65657043457031	
> [4,0:7]	(2.35298156738281,1.328125),(0.65667	(2.35298156738281,1.328125),(0.65667	
> [5,0:7]	(3.55589294433594,-0.8955535888671	(3.55589294433594,-0.8955535888671	
✓ [6,0:7]	(1.42153930664063,-1.7247161865234	(1.42153930664063,-1.7247161865234	
> [6,0]	1.42153930664063,-1.72471618652344	1.42153930664063,-1.72471618652344	
> [6,1]	2.82286071777344,-0.33116149902343	2.82286071777344,-0.33116149902343	
> [6,2]	0.8614501953125,1.401123046875	0.8614501953125,1.401123046875	
> [6,3]	-0.0400848388671875,0.152648925781	-0.0400848388671875,0.152648925781	
> [6,4]	0.162277221679688,-1.1315002441406	0.162277221679688,-1.1315002441406	
> [6,5]	2.16227722167969,-1.62667846679688	2.16227722167969,-1.62667846679688	
> [6,6]	3.15022277832031,0.0	3.15022277832031,0.0	
> [6,7]	1.54530334472656,0.71986389160156	1.54530334472656,0.71986389160156	
✓ [7,0:7]	(0.810501098632813,-0.002868652343	(0.810501098632813,-0.002868652343	
> [7,0]	0.810501098632813,-0.0028686523437	0.810501098632813,-0.0028686523437	
> [7,1]	1.99021911621094,-0.61335754394531	1.99021911621094,-0.61335754394531	
> [7,2]	1.54545593261719,0.06507873535156	1.54545593261719,0.06507873535156	
> [7,3]	1.79316711425781,0.52215576171875	1.79316711425781,0.52215576171875	
> [7,4]	0.890151977539063,0.25770568847656	0.890151977539063,0.25770568847656	
> [7,5]	1.09747314453125,-1.04292297363281	1.09747314453125,-1.04292297363281	
> [7,6]	1.54530334472656,-0.71986389160156	1.54530334472656,-0.71986389160156	
> [7,7]	2.62626647949219,0.0	2.62626647949219,0.0	

Figure 45 Vivado Generation of 6th and 7th row of $\hat{\mathbf{R}}$ matrix

5.3.3. VHDL Implementation of Finding Arrival Angle

To find the arrival angle, we need to implement the expression

$$\mathbf{a}^*(\theta)\mathbf{R}\mathbf{a}(\theta) \quad (3.51)$$

The θ values fall into the interval $[\frac{-\pi}{2}, \frac{+\pi}{2}]$. According to Figure 34, L expresses the number of samples on $[\frac{-\pi}{2}, \frac{+\pi}{2}]$ to search for the angle of the sources. \mathbf{a} indicates the normalized transfer or direction vector, \mathbf{a}^* is the complex conjugate transpose of \mathbf{a} . σ is the spatial spectral estimation at L **equally** spaced angles, briefly, it expresses the DOA estimation.

The dimension of \mathbf{a} matrix depends on L and the number of sensors m . According to given numerical parameters, \mathbf{a} matrix has 32 rows and 8 columns. As a result, dimension of \mathbf{a}^* matrix will be 32x8. The dimension of $\hat{\mathbf{R}}$ matrix is 8x8.

First, $\mathbf{Ra}(\theta)$ matrix is calculated, and next $\mathbf{a}^*\mathbf{Ra}(\theta)$ is evaluated according to the matrix multiplication rules.

The resulting matrix σ is obtained by taking the real part of the $\mathbf{a}^*\mathbf{Ra}(\theta)$ matrix and the dimension of σ matrix will be 32x32. The diagonal of this matrix shows the angle values. In other words, the angles between $[-90^\circ]$ and $[+90^\circ]$ are mapped equally into a matrix with a dimension of 32x32. For instance, $[-90^\circ]$ is mapped into the first row and first column of diagonal matrix and $[+90^\circ]$ is mapped into the last row and last column of diagonal matrix. Moreover, 0° will be located in the midpoint of the diagonal matrix.

In order to find incident angles, the maximum of two indices of the diagonal matrix should be determined since there are two signal sources.

The index values found are used in the expressions (5.6) and (5.7)

$$\left(\frac{180^\circ}{L}\right) * \text{maximum}_{index_1} - 90^\circ \quad (5.6)$$

$$\left(\frac{180^\circ}{L}\right) * \text{maximum}_{index_2} - 90^\circ \quad (5.7)$$

Table 15 Diagonal of Phi Matrix

Diagonal of Phi Matrix			
Index	Value	Index	Value
0	8.6911	16	99.5999
1	8.6471	17	60.1665
2	8.5576	18	14.1836
3	8.4707	19	44.7324
4	8.2379	20	87.6518
5	7.4780	21	74.9434
6	6.3298	22	34.8041
7	6.5671	23	12.1458
8	9.7548	24	10.9700
9	11.9795	25	15.2155
10	7.7584	26	15.8611
11	4.7779	27	13.5346
12	10.7480	28	11.0632
13	9.5240	29	9.6008
14	11.2083	30	8.9679
15	57.8416	31	8.7441

According to the Table 15, the first maximum index is 16 and second maximum index is 20. In addition, Vivado generated diagonal matrix with expected index values is shown in Figure 46 and Figure 47.

Using these indexes in (5.6) and (5.7), we obtain the results

$$\left(\frac{180^\circ}{32}\right) * 16 - 90^\circ = 0^\circ$$

$$\left(\frac{180^\circ}{32}\right) * 20 - 90^\circ = 22.50^\circ$$

Thus, DOA estimation of the two sources is achieved. The incident angles of the sources were 0° and 25.0° . There is a 2.50° difference for second source. If the L value is increased, more accurate angle degrees can be obtained. However, using a larger L value, increases the computations and complex matrix multiplications in VHDL.

Name	Value	
o_phi_vector[0:31][10:-12]	8.687744140625,8.644775390625	8.6877441□
> o_phi_vector[0][10:-12]	8.687744140625	8.6877441□
> o_phi_vector[1][10:-12]	8.644775390625	8.6447753□
> o_phi_vector[2][10:-12]	8.527099609375	8.5270996□
> o_phi_vector[3][10:-12]	8.4638671875	8.4638671□
> o_phi_vector[4][10:-12]	8.232666015625	8.2326660□
> o_phi_vector[5][10:-12]	7.487060546875	7.4870605□
> o_phi_vector[6][10:-12]	6.30517578125	6.3051757□
> o_phi_vector[7][10:-12]	6.553466796875	6.5534667□
> o_phi_vector[8][10:-12]	9.72705078125	9.7270507□
> o_phi_vector[9][10:-12]	11.962646484375	11.962646□
> o_phi_vector[10][10:-12]	7.777099609375	7.7770996□
> o_phi_vector[11][10:-12]	4.773193359375	4.7731933□
> o_phi_vector[12][10:-12]	10.691162109375	10.691162□
> o_phi_vector[13][10:-12]	9.52734375	9.52734375
> o_phi_vector[14][10:-12]	11.22998046875	11.229980□
> o_phi_vector[15][10:-12]	57.848388671875	57.848388□

Figure 46 Vivado Generated Diagonal Matrix, from [0,0] to [15,15]

> [15][10:-12]	57.848388671875	57.84838880
> [16][10:-12]	99.574462890625	99.57446200
> [17][10:-12]	60.173583984375	60.17358300
> [18][10:-12]	14.18017578125	14.18017500
> [19][10:-12]	44.745849609375	44.74584900
> [20][10:-12]	87.474609375	87.47460900
> [21][10:-12]	74.961181640625	74.96118100
> [22][10:-12]	34.794921875	34.79492100
> [23][10:-12]	12.099609375	12.09960900
> [24][10:-12]	10.978271484375	10.97827100
> [25][10:-12]	15.1982421875	15.19824200
> [26][10:-12]	15.82275390625	15.82275300
> [27][10:-12]	13.54638671875	13.54638600
> [28][10:-12]	11.06396484375	11.06396400
> [29][10:-12]	9.609130859375	9.60913080
> [30][10:-12]	8.94287109375	8.94287100
> [31][10:-12]	8.718505859375	8.71850580

Figure 47 Vivado Generated Diagonal Matrix, from [15,15] to [31,31]

CHAPTER 6

CONCLUSION

Direction of arrival (DOA) estimation is used to predict the angle at which an acoustic or electromagnetic wave arriving to sensor arrays. DOA estimation is used in different engineering fields such as sonar, radar, navigation, radio astronomy, wireless communications, tracking of different objects. The important issue in DOA is to use an antenna array, which is used to receive signals in order to extract information about signal parameters. In this thesis, the theoretical study of conventional beamforming and Capon beamforming is researched and VHDL implementation of uniform linear array for DOA estimation is performed.

In this thesis, first, the background of DOA estimation, historical developments of antenna arrays, structural analysis of spatial spectrum estimation, mathematical model of DOA and the parameters affecting accuracy of DOA estimation were inspected. Afterwards, conventional beamforming and Capon beamforming was implemented in MATLAB. From the simulation results it is seen that beamforming methods gets higher resolutions when the number of array elements or the number of snapshots and SNR level increase. However, increasing the element spacing between the sensors beyond half of the wavelength causes false peaks in the spectrum. The false angle values from other directions were observed. In addition, Capon beamforming has better accuracy and narrower beamwidth in all circumstances when compared to the conventional beamforming.

Using the numerical values calculated using MATLAB, conventional beamforming algorithm is implemented in VHD which is a hardware language used to describe the structural and behavioral characteristics of digital logic circuits. In addition, it allows the system to be modelled and simulated by the gates, and wires. It makes possible to use concurrent statements in order to programming blocks simultaneously.

Subsequently, it makes the design faster, which is one of the reasons why we used it in this thesis. In order to make simulations of VHDL codes, different testbenches are used to get information about functionality of the design by comparing output responses with the expected output values. Parameters used in VHDL implementation are as follows. θ represents the arrival angles of the sources in degrees. N represents the number of samples while m shows the number of sensors and d represents the spacing between these sensors. L shows the number of samples. The used parameter values are as follows. Number of sources is equal to 2, number of sensors is equal to 8, sensor spacing between two sensors is 0.5λ . Arrival angles of two sources θ_1 and θ_2 are equal to 0° and 25° respectively. L is equal to 32.

Conventional beamforming is implemented in VHDL using these values and expressions available Chapter 3.

In our implementations, first, sine wave that includes the angle values is generated. Secondly, direction matrix A is implemented. Then Y , which is the output of uniform linear array sensors, was obtained. In order to obtain sample covariance matrix \hat{R} , complex conjugate transpose of Y matrix was obtained. Matrix multiplication of two matrices was performed and normalized according to the number of samples. Afterwards; normalized direction matrix a and a^* was implemented. Finally, matrix containing theta values was created and two angle values were obtained. The exact value of the first angle was obtained, however, the second value was obtained with 2.5 degree error. In order to find more accurate results, L value can be increased to search angles using more steps.

REFERENCES

- [1] Krishnaveni, V., Kesavamurthy, T. and Aparna, B. (2013). "*Beamforming for Direction of Arrival (DOA) Estimation A Survey*", International Journal of Computer Applications. vol.61, pp.4-11.
- [2] Blabut, R. E., Millerm, W. and Wilcox, C. H. (1991). "*Radar and Sonar, Part I*", SpringerVerlag, New York.
- [3] Richards, M. A.(2005). "*Fundamentals of Radar Signal Processing*", McGraw-Hill.
- [4] Saunders, S.R. (2007). "*Antennas and Propagation for Wireless Communication Systems*" (2nd ed.), John Wiley Sons.
- [5] Boccuzzi, J. (2007). "*Signal Processing for Wireless Communications*", McGraw-Hill.
- [6] Raghavendra, C.S., Sivalingam, K. M. and Znati, T.(2004). "*Wireless Sensor Networks*", Springer Science.
- [7] Naidu, P. S.(2009). "*Sensor Array Signal Processing*"(2nd ed.), CRC Press.
- [8] Wang, H.Y. (1990). "*Modern Spectrum Estimation*", Dongnan University Press.
- [9] Mondal, D. (2013). "*Studies of Different Direction of Arrival (DOA) Estimation Algorithm for Smart Antenna in Wireless Communication*", The International Journal of Electronics & Communication Technology (IJEET).vol.4, pp. 47-51.
- [10] Capon, J.(1987). "*High-resolution frequency-wavenumber spectrum analysis*", Pmc. IEEE. vol. 57, pp.1408-1418.
- [11] Dhering, N. A. and Bansode, B. N. (2013). "*Performance evaluation of direction of arrival estimation using MUSIC and ESPRIT algorithms for mobile communication system*", International Journal of Advanced Research in Computer Science and Software Engineering. vol.3, pp.1453-1462.
- [12] Bartlett, M. S.(1950). "*Periodogram analysis and continuous spectra*", Biometrika. vol.37, pp.1-16.

- [13] Zhang, X. and Zheng, B. (2000). “*Communication Signal Processing*”, National Defence Industry Press.
- [14] Zhang, X.(2000). “*Modern Signal Processing*”, Tsinghua Press.
- [15] Ziskind, I.(1988). “*Maximum Likelihood Localization of Multiple Sources by Alternating Projection*”, IEEE Trans on ASSP. vol.36 pp.1558-1560.
- [16] Stoica, P. and Moses, R. L. (2005). “*Spectral analysis of signals*”, Upper Saddle River, N.J: Pearson, Prentice Hall.
- [17] Chen, Z., Gokeda, G. and Yu,Y.(2010). “*Introduction to Direction-of-Arrival Estimation*”, Boston: Artech House.
- [18] Tang, H. (2014). “DOA estimation based on MUSIC algorithm”,(Bachelor thesis Linneuniversitetet Kalmar, Vaxjo, Sweden).
- [19] Khmou, Y., Safi, S. and Frikel, M. (2014). “*Comparative study between several direction of arrival estimation methods*”, J. of Telecommun. and Inform. Technol. vol.1, pp.41–48.
- [20] Gazi, O. (2019). “A tutorial introduction to VHDL programming”, Springer.
- [21] “Vivado Simulator”, Xilinx Company, [Online]. Available: <https://www.xilinx.com/products/design-tools/vivado/simulator.html#waveform>. [Accessed 02 March 2020].
- [22] Haykin, S., Reilly, J. P., and Vertaschitsch, E. (1992). “ *Some Aspects of Array Signal Processing*”, IEEE Proc. vol.139, pp.1-26.
- [23] Balanis, C. A. (2005). “*Antenna Theory: Analysis and Design*” (3rd ed.), New York: Wiley.
- [24] Bondyopadhyay, P. K. (2000). “*The first application of array antenna*”, IEEE International Conference on Phased Array Systems and Technology. pp.29–32.
- [25] Alvarez, L. W. (1987). “*Alvarez: Adventures of a Physicist*”, Basic Books :New York.
- [26] Yuanxun, W., Yongxi, Q. and Tatsuo, I. (2002). “*A novel smart antennas system implementation for broadband wireless communications*”, IEEE Transactions on Antennas and Propagation. vol.50, pp.600-606.
- [27] Hafeth, H.(2004). “*An overview of adaptive antenna systems*”, Postgraduate Course in Radio Communications, Finland: Helsinki University of Technology.

- [28] Keerthi, K. A. (2015). “*Adaptive beamforming smart antenna for wireless communication system*”, International Research Journal of Engineering and Technology (IRJET). vol.2, pp.2038-2043.
- [29] Stoica, P. and Randolph, M.(1997). “*Basic definitions and the spectral estimation problem, lecture notes*” , Columbus, Ohio: Prentice Hall.
- [30] Nikolova, N. (2014). “*Linear array theory, lecture notes*”, McMaster University, Hamiton, Ontario, Canada
- [31] Banuprakash, R., Ganapathy, H., Sowmya, M. and Swetha, M. (2016). “*Evaluation of MUSIC algorithm for DOA estimation in smart antenna*”, International Advanced Research Journal in Science, Engineering and Technology (IARJSET). vol.3, pp.185 – 188.
- [32] Naidu, P. (2009). “*Sensor Array Signal Processing*”, Boca Raton: CRC Press
- [33] Stoica, P. (1990). “*Maximum Likelihood Method for Direction of Arrival Estimation*”, IEEE Trans on ASSP. vol.38, pp.1132-1143.
- [34] Miller, M. I., and Fuhrmann, D. R. (1990). “*Maximum-likelihood narrow-band direction finding and EM algorithms*”, IEEE Transactions on Acoustics, Speech, and Signal Processing. vol.38, pp.1560–1577.
- [35] Ziskind, I. and Wax M., (1988). “*Maximum Likelihood Localization of Multiple Sources by Alternating Projection*”, IEEE Trans on ASSP. vol.36, pp.1553-1560.
- [36] DeGroat, R.D. (1993). “*The Constrained MUSIC Problem*”, IEEE Trans on SP. vol.41, pp.1145-1149.
- [37] Richard, F. (1990). “*Analysis of Min-norm and MUSIC with Arbitrary Array Geometry*”, IEEE Trans on AES. vol.26, pp.976-985.
- [38] Stoica P. and Nehorai A., (1989). “*MUSIC, maximum likelihood, and Cramer-Rao bound*”, IEEE Transactions on Acoustics, Speech, and Signal Processing. vol.37, pp. 720-741.
- [39] Lee, H. B. and Zoltowski, M. D. (1990). “*Resolution Threshold Beamspace MUSIC for Two Closely Spaced Emitters*”, IEEE Trans on ASSP. vol.38 pp.723-738.
- [40] Gavish, M. and Weiss, A.J., (1993). “*Performance Analysis of the VIA ESPRIT Algorithm*”, IEE-Proc-F. vol.140, pp.123-128.

- [41] Shan, T. J. and Wax M., (1985). “*Adaptive beamforming for Coherent Signals and Inference*”, IEEE Trans on ASSP. vol. 33, pp.527-536.
- [42] Zhang, X.F., Chen, C., Li J.F., and Xu D.Z. (2014). “*Blind DOA and Polarization Estimation for Polarization-sensitive Array using Dimension Reduction MUSIC*”, Multidimensional Systems and Signal Processing. vol.25, pp.67-82.
- [43] Comon, P. et. al.(1991).“*Blind separation of sources, Part II: Problem statement*”, Signal Process. vol.24, pp.11–20.
- [44] Kim, Y. and Ling, H., (2011). “*Direction of Arrival Estimation of Humans with a Small Sensor Array using an Artificial Neural Network*”, EMW Publishing, Progress in Electromagnetics Research B. vol. 27. pp. 127-49.
- [45] Yang, L.N. (2007). “*Study of Factors Affecting Accuracy of DOA*”, Modern Radar. vol. 29, pp.70-73.
- [46] Zooghyby, A.E.(2005). “*Smart antenna engineering*”, Artech House, Norwood, MA, first ed.
- [47] IEEE Standard for definitions of terms for antennas". *IEEE Std.* 2014.
- [48] Antenna-theory [Online]. Available: <http://www.antenna-theory.com/> [Accessed 24 December 2019].
- [49] Liberti,J. C., and Rappaport, T.S. (1999).“*Smart antennas for wireless communications: IS-95 and third generation CDMA applications*”, Prentice Hall, Upper Saddle River, NJ, first ed.
- [50] Ouargui, I. E., Safi, S. and Frikel, M. (2018). “*Minimum Array Elements for Resolution of Several Direction of Arrival Estimation Methods in Various Noise-Level Environments*”, Journal of Telecommunications and Information Technology. pp.87-94.
- [51] Thelabbookpages: microphonearraybeamforming: [Online]. Available: <http://www.labbookpages.co.uk/audio/beamforming.html>[Accessed 24 December 2019].
- [52] Proakis, J.G. and Manolakis, D. G.(1996). “*Digital Signal Processing. Principles, Algorithms, and Applications*”, Prentice Hall.
- [53] Peter, G., Angeliki, X. and Christoph, M. (2015). “*Multiple and single snapshots compressive beamforming*”, The Journal of Acoustical Society of America. pp.1-12.

- [54] Yu, C.H. and Li, J.L. (2012). “*A White Noise Filtering Method for DOA Estimation of Coherent Signals in Low SNR*”, *Signal Processing*. vol.28, pp.957-962.
- [55] Drabowitch, S. (et al.) (1998). “*Modern antennas*”(1st ed.), Chapman and Hall, Boundary Row, London.
- [56] Fourikis, N.(2000). “*Advanced array systems, applications and RF Technologies*” (1st ed.). London: Academic Press.
- [57] Manolakis, D. G., Ingle, V.K. and Kogon, S.M. (2005). “*Statistical and adaptive signal processing: spectral estimation, signal modeling, adaptive filtering and array processing*” (1st ed.), Norwood, MA: Artech House.
- [58] Liu, W. and Weiss, S. (2010). “*Wideband beamforming concepts and techniques*”(1st ed.). Hoboken, NJ: John Wiley and Sons.
- [59] Godara, L. C. (2001). “*Handbook of Antennas in Wireless Communications*”, Boca Raton, FL: CRC Press.
- [60] Kumaresan, R. and Tufts, D. W. (1983). “*Estimating the angles of arrival of multiple plane waves*”, *IEEE Trans. Aerosp. Electron. Syst.* vol.19, pp.134-139.
- [61] Theodoridis, S. and Chellappa, R. (2013). “*Academic Press Library in Signal Processing, Volume 3: Array and Statistical Signal Processing*” (1st ed.), Orlando, FL: Academic Press Inc.
- [62] Shan, T. J., Wax, M. and Kailath, T. (1985). “*On spatial smoothing for direction-of-arrival estimation of coherent signals*”, *Acoustics, Speech and Signal Processing IEEE Transactions*. vol.33, pp.806-811.
- [63] Li, J. (1992). “*Improved angular resolution for spatial smoothing techniques*”, *Signal Processing, IEEE Transactions*. vol.40, pp.3078-3081.
- [64] Choi, Y.H. (2002). “*Subspace-based coherent source localization with forward/backward covariance matrices*”, *Radar, Sonar and Navigation, IEE Proceedings*, vol.149, pp.145-151.
- [65] Piper, J. E. (2011). “*Beamforming narrowband and broadband signals*”, Naval Surface Warfare Center (NSWC).
- [66] Okkonen. J. (2013). “*Uniform linear adaptive antenna array beamforming implementation with a wireless open-access research platform*”, (Bachelor thesis University of Oulu, Department of Computer Science and Engineering).
- [67] Van Trees, H. (2002). “*Optimum Array Processing*”. New York: Wiley.

- [68] Van Veen, B. and Buckley, K. (1988). “*Beamforming: A Versatile Approach to Spatial Filtering*”, IEEE ASSP magazine. vol.5, pp.4-24.
- [69] Featherstone, W., Strangeways, H.J., Zatman, M.A., and Mewes, H. (1997). “*A Novel Method to Improvement the Performance of Capon’s minimum Variance Estimator*”, London UK. IEE. vol.1, pp.322-325.
- [70] Wilson, P. (2015). “*Design Recipes for FPGAs: Using Verilog and VHDL*”. Newnes.
- [71] Bishop, D. (2020). “*Floating point package user's guide*”.
- [72] Mathworks [Online] Available:
<https://www.mathworks.com/help/matlab/ref/ctranspose.html> [Accessed 24 April 2020].



APPENDIX A: MATLAB codes for Conventional Beamforming

A.1 Matlab code for Main Program

```
% Specifying required data:
% Signals arriving at 0 and 25 degree →DOA estimation
% P: Covariance matrix of two signal sources.
% theta : the value of incident angles,
%N: number of samples ,
% m: number of sensors,
% d : sensor spacing ,
% sig2: noise variance
% L: the number of samples on [-pi/2,pi/2] to search for the sources.
P = eye(2);
theta = 25; N=16; m=8; d=0.5; sig2=1 ;L=32;
% Preparing the empty arrays
y = zeros(m, N, 50);
phi1 = zeros(50, L);
%% Data Processing
% Because of the noise component in generation of the ULA data
% take fifty iterations of ULA data generation and average the final results.
for i=1:50,
    y(:,:,i) = uladata([0, theta] ,P,N,sig2,m,d);
    phi1(i,:) = beamform(y(:,:,i), L,d);
end
% calculate the average spectrum corresponding to each set of ULA data and plot it.
phi1avg(1:L) = mean(phi1(:,1:L));
figure(1)
plot(linspace(-90,90,L),20*log10(phi1avg))
ylabel('dB')
title('Averaged Spatial Spectrum: Beamforming method');
xlabel('n')
```

A2. Matlab code for Conventional Beamforming

```
function phi=beamform(Y,L,d)
% The Beamforming method for direction of arrival estimation%
% phi=beamform(Y,L,d);
% Y <- the ULA data
% L <- the number of samples on [-pi/2,pi/2] to search for the sources
% d <- sensor spacing in wavelengths
% phi -> the spatial spectral estimate at L equally spaced angles
%      in [-90,90] degrees
[m,N]=size(Y);
% compute the sample covariance matrix
R=Y*Y'/N;
phi=zeros(L,1);
for i = 1 : L,
    a=exp(-2*pi*j*d*sin(-pi/2 + pi*(i-1)/L)*[0:m-1].');
    phi(i)=real(a'*R*a);
end
```

A3. Matlab code for Uniform Linear Array generetaion

```
function Y=uladata(theta,P,N,sig2,m,d)
%Generates N snapshots of ULA sensor data
% theta: arrival angles of the m sources in degrees
% P: the covariance matrix of the source signals
% N: number of snapshots to generate
% sig2: noise variance
% m: number of sensors
% d: sensor spacing in wavelengths
% A: direction matrix
% Y : m x N data matrix Y = [y(1)...,y(N)]
% generate the A matrix
j=sqrt(-1);
A=exp(-2*pi*j*d*[0:m-1].'*sin([theta(:).']*pi/180));
% generate the source signals
n=max(size(P));
s=(sqrtm(P))*randn(n,N);
% generate the noise component
e=sqrt(sig2/2)*(randn(m,N)+j*randn(m,N));
% generate the ULA data
Y=A*s+e;
```

APPENDIX B: VHDL codes for Conventional Beamforming

B.1: VHDL Implementation of Sine Wave

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

library ieee_proposed;
use ieee_proposed.fixed_pkg.all;

entity sineGeneration is
port(
  clk_100Mhz : in std_logic;
  dac_sin_out : out std_logic_vector(15 downto 0)
);
end sineGeneration;

architecture Behavioral of sineGeneration is

  --Integer part
  constant na: integer := -15;
  --Fraction part
  constant nh: integer := 0;
  --Fixed point vector for 64 samples
  type s_fixed_vector is array (0 to 63) of sfixed(nh downto na);

  signal sine_vector: s_fixed_vector;
  --Counter for 64 samples
  signal sin_count: natural range 0 to 63 := 0;
begin

  sine_vector<=(to_sfixed(0,nh,na),to_sfixed(0.0980,nh,na),to_sfixed(0.1951,nh,na),to
_sfixed(0.2903,nh,na),to_sfixed(0.3827,nh,na),to_sfixed(0.4714,nh,na),
to_sfixed(0.5556,nh,na),to_sfixed(0.6344,nh,na),to_sfixed(0.7071,nh,na),to_sfixed(0
.7730,nh,na),to_sfixed(0.8315,nh,na),to_sfixed(0.8819,nh,na),to_sfixed(0.9239,nh,na
),to_sfixed(0.9569 ,nh,na),to_sfixed(0.9808 ,nh,na),to_sfixed(0.9952 ,nh,na),
to_sfixed(1.0000,nh,na),to_sfixed(0.9952,nh,na),to_sfixed(0.9808,nh,na),to_sfixed(0
.9569 ,nh,na),to_sfixed( 0.9239 ,nh,na),to_sfixed(0.8819,nh,na), to_sfixed( 0.8315
,nh,na),to_sfixed(0.7730,nh,na),to_sfixed(0.7071,nh,na),to_sfixed(0.6344
,nh,na),to_sfixed(0.5556 ,nh,na),to_sfixed(0.4714,nh,na), to_sfixed(0.3827,nh,na),
to_sfixed(0.2903,nh,na),to_sfixed(0.1951,nh,na),to_sfixed(0.0980,nh,na),to_sfixed(0
.0000 ,nh,na),to_sfixed(-0.0980,nh,na), to_sfixed( -0.1951 ,nh,na),to_sfixed(-0.2903
,nh,na),to_sfixed(-0.3827,nh,na),to_sfixed(-0.4714 ,nh,na),to_sfixed( -0.5556
,nh,na),to_sfixed(-0.6344 ,nh,na),to_sfixed(-0.7071 ,nh,na),
```

```

to_sfixed(0.7730,nh,na), to_sfixed(0.8315,nh,na),to_sfixed(0.8819,nh,na),to_sfixed(-
0.9239,nh,na),to_sfixed(-0.9569 ,nh,na),to_sfixed(-0.9808 ,nh,na), to_sfixed(-0.9952
,nh,na),to_sfixed(-1.0000 ,nh,na),to_sfixed(-0.9952 ,nh,na),to_sfixed( -0.9808,nh,na),
to_sfixed(-0.9569 ,nh,na), to_sfixed(-0.9239,nh,na), to_sfixed( -
0.8819,nh,na),to_sfixed(-0.8315 ,nh,na),to_sfixed(-0.7730 ,nh,na),to_sfixed(-
0.7071,nh,na), to_sfixed(-0.6344 ,nh,na),to_sfixed(-0.5556 ,nh,na), to_sfixed(-
0.4714,nh,na),to_sfixed(-0.3827 ,nh,na),to_sfixed(-0.2903 ,nh,na),to_sfixed( -0.1951
,nh,na), to_sfixed(-0.0980 ,nh,na));

```

--This process is used in order to generate one period of wave by taking one value at each clock.

```

sin_out_process: process (clk_100Mhz)
begin
if (clk_100Mhz'event and clk_100Mhz ='1') then
dac_sin_out <= std_logic_vector(sine_vector(sin_count));
if (sin_count=63) then
sin_count <= 0;
else
sin_count <= sin_count +1;
end if;
end if;
end process;
end Behavioral;

```

B.2: VHDL Implementation of Direction Matrix A

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

library ieee_proposed;
use ieee_proposed.fixed_pkg.all;
use work.matrix_package.all;

-- This VHDL code generates direction matrix A[8x2]
-- 8: number of sensors
-- 2: number of signals

entity direction_matrix_A is
Port (
clk: in std_logic;
complex_matrix_A: out re_im_matrix_8x2);
end direction_matrix_A;

architecture Behavioral of direction_matrix_A is

signal complex_matrix_A_temp: re_im_matrix_8x2;

type MatrixData is array (0 to 2*num_rows_A*num_cols_A-1) of real;

signal A_elements: MatrixData := ( 1.000, 0.000, 1.000, 0.000, 1.000, 0.000, 1.000,
0.000, 1.000, 0.000, 1.000, 0.000, 1.000, 0.000, 1.000, 0.000, 1.000, 0.000, 0.241 , -
0.971 , -0.884 , -0.467 , -0.666, 0.746, 0.563, 0.826, 0.938 , -0.348 , -0.112 , -0.994 ,
-0.991 , -0.131 );

signal i,j: integer :=0;

begin
--FOR LOOP for real and imaginair part generation of A matrix

FY1:for i in 0 to num_rows_A-1 generate
begin
FY2:for j in 0 to num_cols_A-1 generate
begin
complex_matrix_A_temp(i,j).re <= to_sfixed(A_elements(2*(i+num_rows_A*j))
,nh, na);
complex_matrix_A_temp(i,j).im<=to_sfixed(A_elements(2*(i+num_rows_A*j)+1) ,
nh, na);
end generate;
end generate;
complex_matrix_A <= complex_matrix_A_temp;
end Behavioral;
```

B.3: VHDL Implementation of Matrix Y and Complex Conjugate Transpose Matrix of Y

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;

library ieee_proposed;
use ieee_proposed.fixed_pkg.all;
use work.matrix_package.all;

-- This VHDL code generates output matrix Y[8x16] and Y.'[16x8]
-- 8: number of sensors
-- 16: number of samples

entity matrixGeneration is
  Port (
    clk: in std_logic;
    complex_matrix_Y: out re_im_matrix_8x16;
    complex_matrix_Y_tr: out re_im_matrix_16x8
  );
end matrixGeneration;

architecture Behavioral of matrixGeneration is

  signal complex_matrix_Y_temp: re_im_matrix_8x16;

  type MatrixData is array (0 to 2*num_rows*num_cols-1) of real;

  signal Y_elements: MatrixData := ( -1.137 , -0.301, 0.471, 1.981, 2.714, 0.709, 2.060
  , -2.632, 0.498 , -2.026 , -0.036, 0.994, 0.715, 1.563, 2.665 , -0.581 , -3.481 , -0.443,
  0.316, 2.068, 1.254, 0.208, 0.423 , -0.764 , -1.302 , -2.063 , -2.862, 1.957, 0.179, 1.173,
  1.511, 0.395, 3.422 , -1.076, 2.033 , -1.348 , -1.507 , -0.822, 0.758, 0.926, 0.517, 1.378,
  1.648 , -0.087, 1.040 , -1.120, 0.695, 0.464 , -0.263, 0.565, 1.099, 0.154 , -0.276, 0.437,
  0.280, 0.239, 0.431 , -0.756 , -0.005, 0.562, 0.632 , -0.394, 0.300 , -0.418 , -2.460,
  0.125 , -3.623 , -0.782 , -1.686 , -0.365 , -3.301, 0.854 , -2.086, 1.220 , -1.543 , -0.342
  , -2.471 , -0.346 , -3.702 , -0.244 , -4.842, 0.162 , -2.839, 2.562, 0.904, 0.648 , -1.256
  , -1.535 , -2.944 , -1.491 , -4.149, 0.872 , -2.098, 3.528 , -0.640 , -0.128 , -1.069 , -
  0.521 , -0.210 , -0.317 , -0.121, 1.087, 0.848 , -0.094 , -1.036 , -0.840 , -0.876, 0.965,
  0.424, 1.072, 0.381, 0.329, 0.885, 0.080, 0.822 , -0.346 , -1.485 , -0.244, 0.693, 2.476,
  2.507 , -0.266, 1.725 , -1.547, 0.112 , -1.479, 0.043 , -0.861 , -0.566 , -0.307, 2.019,
  0.552, 2.134, 0.168, 1.875 , -0.133 , -0.151 , -0.296, 0.436, 0.772, 0.540, 1.944, 1.323,
  0.955, 0.147 , -0.907 , -0.252 , -1.744 , -0.282, 0.591, 0.234 , -1.223 , -0.695, 0.130,
  0.973 , -0.010, 0.355, 1.475, 0.844 , -0.062, 1.553 , -0.359, 1.305 , -0.354 , -0.697 , -
  0.077 , -0.253, 0.449, 0.812, 0.013, 1.406 , -1.051, 1.564 , -0.309, 0.089 , -1.020, 1.658
  , -0.730, 1.202, 1.069, 1.845 , -0.235, 1.076 , -0.198, 0.359, 0.527 , -0.273 , -0.166,
  1.023, 0.859, 1.608 , -0.227, 0.654, 0.716 , -0.287 , -0.533 , -1.256 , -0.622 , -1.594,
```



```
0.139, 0.072, 0.095, 0.670 , -0.542 , -0.228 , -1.311 , -1.191 , -0.421 , -2.196, 0.695 , -
2.168, 0.279 , -0.395, 1.044 , -1.567 , -0.752 , -1.132, 0.271 , -3.108 , -0.194 , -1.695,
0.626 , -1.807, 0.752 , -1.229 , -0.293 , -0.943, 0.061 , -2.478, 1.595 , -1.954, 0.387 ,
-2.505 , -0.622 , -1.345, 0.592 , -0.188, 0.126 , -1.122, 0.549, 2.049, 1.001, 1.233 , -
1.936 , -0.825, 0.342, 0.380, 1.647, 0.582, 0.716, 2.495, 0.948, 0.533 , -0.711, 0.987 ,
-0.983 );
```

```
signal i,j: integer :=0;
begin
```

```

    FY1:for i in 0 to num_rows-1 generate
    begin
        FY2:for j in 0 to num_cols-1 generate
        begin
            complex_matrix_Y_temp(i,j).re <= to_sfixed(Y_elements(2*(i+num_rows*j)) ,
nh, na);
            complex_matrix_Y_temp(i,j).im <= to_sfixed(Y_elements(2*(i+num_rows*j)+1)
, nh, na);
        end generate;
    end generate;

    complex_matrix_Y    <= complex_matrix_Y_temp;
    complex_matrix_Y_tr <= transpose8x16(complex_matrix_Y_temp);

end Behavioral;
```

B.4: VHDL Implementation of Sample Covariance Matrix R

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
library ieee_proposed;
use ieee_proposed.fixed_pkg.all;
use work.matrix_package.all;

-- This VHDL code generates sample covariance matrix R[8x8]
-- Multiplication of Y[8x16] and Y.'[16x8] returns as [8x8] square matrix
-- 8: number of sensors
-- 16: number of samples

entity CovmatrixGeneration is
  Port (
    clk: in std_logic;
    covariance_matrix: out re_im_matrix_8x8
  );
end CovmatrixGeneration;

architecture Behavioral of CovmatrixGeneration is

  signal complexMatrix_Y: re_im_matrix_8x16;
  signal complexMatrix_Y_tr: re_im_matrix_16x8;

  component matrixGeneration
    port (
      clk: in std_logic;
      complex_matrix_Y: out re_im_matrix_8x16;
      complex_matrix_Y_tr: out re_im_matrix_16x8);
  end component;
begin

  Matrix_PM: matrixGeneration
  PORT MAP
    ( clk=>clk,
      complex_matrix_Y => complexMatrix_Y,
      complex_matrix_Y_tr => complexMatrix_Y_tr);

  mult_process: process (clk) is
  begin
    if(clk'event and clk='1') then
      covariance_matrix <= matrix_mult_8x8(complexMatrix_Y,complexMatrix_Y_tr);
    end if;
  end process mult_process;

end Behavioral;
```

B.5: VHDL Implementation of Product Matrix of R.a

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
library ieee_proposed;
use ieee_proposed.fixed_pkg.all;
use work.matrix_package.all;

-- This VHDL code generates product matrix of sample covariance matrix R[8x8] --
-- and a[8x32]
-- Multiplication of R[8x8] and a[8x32] and returns as [8x32] matrix
-- 8: number of sensors
-- 32: L: the number of samples on [-pi/2,pi/2] to search for the sources
entity R_a_matrix is
  Port (
    clk: in std_logic;
    matrix_R_a: out re_im_matrix_8x32
  );
end R_a_matrix;

architecture Behavioral of R_a_matrix is

  signal covarianceMatrix: re_im_matrix_8x8;

  signal matrix_a_tr: re_im_matrix_32x8;

  signal matrix_a: re_im_matrix_8x32;

  component CovmatrixGeneration
    port (
      clk: in std_logic;
      covariance_matrix: out re_im_matrix_8x8);
  end component;

  component a_matrix is
    Port (
      clk: in std_logic;
      o_matrix_a: out re_im_matrix_8x32;
      o_matrix_a_tr: out re_im_matrix_32x8
    );
  end component;
begin

  covMatrix_PM: CovmatrixGeneration
  PORT MAP ( clk =>clk,
  covariance_matrix =>covarianceMatrix);
```

```
a_matrix_PM: a_matrix
PORT MAP ( clk=>clk,
          o_matrix_a =>matrix_a,
          o_matrix_a_tr => matrix_a_tr);

mult_process: process (clk) is
begin
  if(clk'event and clk='1') then
    matrix_R_a <= matrix_mult_8x32(covarianceMatrix, matrix_a);
  end if;
end process mult_process;

end Behavioral;
```



B.6 : VHDL Implementation of Phi (a'.R.a) Matrix

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;

library ieee_proposed;
use ieee_proposed.fixed_pkg.all;
use work.matrix_package.all;

-- This VHDL code generates phi matrix by multiplying a.' matrix and product
--matrix R.a
-- Multiplication of a.'[32x8] and R.a[8x32] returns as [32x32] square matrix
-- 8: number of sensors
-- 32: L: the number of samples on [-pi/2,pi/2] to search for the sources
--after multiplication it finds main diagonal of the square matrix phi

entity phi_matrix is
  Port (
    clk: in std_logic;
    o_phi_matrix: out re_im_matrix_32x32
  );
end phi_matrix;

architecture Behavioral of phi_matrix is

  signal matrix_Ra: re_im_matrix_8x32_final;

  signal matrix_a_tr: re_im_matrix_32x8;

  signal matrix_a: re_im_matrix_8x32;

  signal o_phi_vector: vect_32;

  component a_matrix is
    Port (
      clk: in std_logic;
      o_matrix_a: out re_im_matrix_8x32;
      o_matrix_a_tr: out re_im_matrix_32x8);
  end component;

  component R_a_matrix is
    Port (
      clk: in std_logic;
      matrix_R_a: out re_im_matrix_8x32_final );
  end component;
```

```

begin
  R_a_matrix_PM: R_a_matrix
  PORT MAP ( clk      =>clk,
            matrix_R_a =>matrix_Ra );

  a_matrix_PM: a_matrix
  PORT MAP ( clk      =>clk,
            o_matrix_a =>matrix_a,
            o_matrix_a_tr => matrix_a_tr);

  mult_process: process (clk) is
  begin
    if(clk'event and clk='1') then
      o_phi_matrix <= matrix_mult_32x32(matrix_a_tr,matrix_Ra);
      o_phi_vector <= diag(o_phi_matrix);
    end if;
  end process mult_process;

end Behavioral;

```

B.7 VHDL Implementation of Matrix Package

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
library ieee_proposed;
use ieee_proposed.fixed_pkg.all;
package matrix_package is

----CONSTANTS----
constant na: integer := -8;      -- fraction part Y and Y.'
constant nh: integer := 4;      --integer part Y and Y.'
constant na_m: integer := -12;  -- fraction part R
constant nh_m: integer := 10;   -- integer part R
constant na_s: integer := -16;  -- fraction part Y and Y' matrix summation
constant nh_sum: integer := 25;
constant na_a2: integer := -20;
constant nh_a2: integer := 16;
constant nh_a3: integer := 23;
constant nh_phi2: integer := 16;
constant num_cols: integer := 16;
constant num_rows: integer := 8;
constant num_cols_a: integer := 32;
constant num_rows_a: integer := 8;

----TYPE DECLERATIONS----
type complex is record
re: sfixed(nh downto na);
im: sfixed(nh downto na);
end record complex;

type complex_m is record
re: sfixed(nh_m downto na_m);
im: sfixed(nh_m downto na_m);
end record complex_m;

type complex_s is record
re: sfixed(nh_s downto na_s);
im: sfixed(nh_s downto na_s);
end record complex_s;

type complex_sum is record
re: sfixed(nh_sum downto na_sum);
im: sfixed(nh_sum downto na_sum);
end record complex_sum;
```

```
type complex_a2 is record
re: sfixed(nh_a2 downto na_a2);
im: sfixed(nh_a2 downto na_a2);
end record complex_a2;
```

```
type complex_a3 is record
re: sfixed(nh_a3 downto na_a2);
im: sfixed(nh_a3 downto na_a2);
end record complex_a3;
```

```
type complex_phi2 is record
re: sfixed(nh_phi2 downto na_a2);
im: sfixed(nh_phi2 downto na_a2);
end record complex_phi2;
```

```
type complex_phi3 is record
re: sfixed(nh_phi3 downto na_a2);
im: sfixed(nh_phi3 downto na_a2);
end record complex_phi3;
```

```
type re_im_matrix_8x16 is array (0 to num_rows - 1, 0 to num_cols -1) of complex;
type re_im_matrix_16x8 is array (0 to num_cols -1, 0 to num_rows -1) of complex;
type re_im_matrix_8x8 is array (0 to num_cols_mult -1, 0 to num_rows_mult -1) of
complex_m;
type re_im_matrix_8x8_temp is array (0 to num_cols_mult -1, 0 to num_rows_mult
-1) of complex_sum;
type re_im_matrix_8x128 is array (0 to num_cols_mult -1, 0 to 127) of complex_s;
type re_im_matrix_8x32 is array (0 to num_rows_a - 1, 0 to num_cols_a -1) of
complex;
type re_im_matrix_32x8 is array (0 to num_cols_a -1, 0 to num_rows_a -1) of
complex;
type re_im_matrix_8x64 is array (0 to num_rows_a -1, 0 to 63 ) of complex_a2;
type re_im_matrix_8x256 is array (0 to num_rows_a -1, 0 to 255 ) of complex_a2;
type re_im_matrix_8x32_temp is array (0 to num_rows_a -1, 0 to num_cols_a -1) of
complex_a3;
type re_im_matrix_8x32_final is array (0 to num_rows_a -1, 0 to num_cols_a -1) of
complex_m;
type re_im_matrix_32x32 is array (0 to num_cols_phi -1, 0 to num_rows_phi -1) of
complex_m;
type re_im_matrix_32x256 is array (0 to num_cols_a -1, 0 to 255 ) of complex_phi2;
type re_im_matrix_32x32_temp is array (0 to num_cols_phi -1, 0 to num_rows_phi -
1) of complex_phi3;
```

--diagonal matrix

```
type vect_32 is array (0 to num_cols_phi - 1) of sfixed(nh_m downto na_m);
```



```

---FUNCTIONS--
function transpose8x32( cplx8_32: re_im_matrix_8x32) return re_im_matrix_32x8;

function transpose8x16( cplx8_16: re_im_matrix_8x16) return re_im_matrix_16x8;

function matrix_mult_8x8(cplx8_16: re_im_matrix_8x16; cplx16_8:
re_im_matrix_16x8 ) return re_im_matrix_8x8;

function matrix_mult_8x32(cplx8_8: re_im_matrix_8x8; cplx8_32:
re_im_matrix_8x32 ) return re_im_matrix_8x32_final;

function matrix_mult_32x32(cplx32_8: re_im_matrix_32x8; cplx8_32:
re_im_matrix_8x32_final ) return re_im_matrix_32x32;

function diag(cplx32_32: re_im_matrix_32x32) return vect_32;

end matrix_package;

---PACKAGE BODY

package body matrix_package is

--TRANSPOSE 8x16

function transpose8x16( cplx8_16: re_im_matrix_8x16) return re_im_matrix_16x8 is
variable i,j: integer :=0;
variable ret: re_im_matrix_16x8;

begin

for i in 0 to num_rows -1 loop
for j in 0 to num_cols -1 loop

ret (j,i).re := cplx8_16(i,j).re;
ret (j,i).im := to_sfixed(-to_real(cplx8_16(i,j).im) , nh, na);

end loop;
end loop;
return ret;
end function;

----TRANSPOSE 8x32----

function transpose8x32( cplx8_32: re_im_matrix_8x32) return re_im_matrix_32x8 is

variable i,j: integer :=0;
variable ret_32: re_im_matrix_32x8;

```

```

begin
for i in 0 to num_rows_a -1 loop
for j in 0 to num_cols_a -1 loop
ret_32(j,i).re := cplx8_32(i,j).re;
ret_32(j,i).im := to_sfixed(-to_real(cplx8_32(i,j).im) , nh, na);
end loop;
end loop;
return ret_32;
end function;
---8x8 MATRIX MULTIPLICATION---

```

```

function matrix_mult_8x8( cplx8_16: re_im_matrix_8x16; cplx16_8:
re_im_matrix_16x8) return re_im_matrix_8x8 is

```

```

variable i,j,k: integer :=0;
variable mult: re_im_matrix_8x8;
variable mult_temp: re_im_matrix_8x128;
variable mult_temp_1: re_im_matrix_8x8_temp;

```

```

begin
for i in 0 to 7 loop
for j in 0 to 7 loop
for k in 0 to 15 loop

```

```

mult_temp (i, j * 16 +k).re := cplx8_16(i,k).re * cplx16_8(k,j).re - cplx8_16(i,k).im
* cplx16_8(k,j).im;
mult_temp (i, j * 16 +k).im := cplx8_16(i,k).re * cplx16_8(k,j).im +
cplx8_16(i,k).im * cplx16_8(k,j).re;

```

```

end loop;
end loop;
end loop;

```

```

for i in 0 to 7 loop
for j in 0 to 7 loop

```

```

mult_temp_1(i,j).re := mult_temp(i, 16 * j + 0).re +mult_temp(i, 16 * j + 1).re +
mult_temp(i, 16 * j + 2).re +mult_temp(i, 16 * j + 3).re +
mult_temp(i, 16 * j + 4).re +mult_temp(i, 16 * j + 5).re + mult_temp(i, 16 * j + 6).re
+mult_temp(i, 16 * j + 7).re +
mult_temp(i, 16 * j + 8).re +mult_temp(i, 16 * j + 9).re + mult_temp(i, 16 * j +
10).re +mult_temp(i, 16 * j + 11).re +
mult_temp(i, 16 * j + 12).re +mult_temp(i, 16 * j + 13).re + mult_temp(i, 16 * j +
14).re +mult_temp(i, 16 * j + 15).re;
mult_temp_1(i,j).im := mult_temp(i, 16 * j + 0).im +mult_temp(i, 16 * j + 1).im +
mult_temp(i, 16 * j + 2).im +mult_temp(i, 16 * j + 3).im +
mult_temp(i, 16 * j + 4).im +mult_temp(i, 16 * j + 5).im + mult_temp(i, 16 * j +
6).im +mult_temp(i, 16 * j + 7).im +

```

```

mult_temp(i, 16 * j + 8).im +mult_temp(i, 16 * j + 9).im + mult_temp(i, 16 * j +
10).im +mult_temp(i, 16 * j + 11).im +
mult_temp(i, 16 * j + 12).im +mult_temp(i, 16 * j + 13).im + mult_temp(i, 16 * j +
14).im +mult_temp(i, 16 * j + 15).im;

```

```

end loop;
end loop;

```

```
--DIVIDE BY 16
```

```

for i in 0 to 7 loop --num_rows1 -1
for j in 0 to 7 loop --num_col2 -1

```

```

mult(i,j).re := resize((mult_temp_1(i,j).re sra 4) , nh_m,na_m);
mult(i,j).im := resize((mult_temp_1(i,j).im sra 4) , nh_m,na_m);
end loop;
end loop;
return mult;
end function;

```

```
--8x32 MATRIX MULTIPLICATION
```

```

function matrix_mult_8x32( cplx8_8: re_im_matrix_8x8; cplx8_32:
re_im_matrix_8x32 ) return re_im_matrix_8x32_final is

```

```

variable i,j,k: integer :=0;
variable mult: re_im_matrix_8x32_final;
variable mult_temp: re_im_matrix_8x256;
variable mult_temp_1: re_im_matrix_8x32_temp;

```

```

begin
for i in 0 to 7 loop
for j in 0 to 31 loop
for k in 0 to 7 loop

```

```

mult_temp (i, j * 8 +k).re := cplx8_8(i,k).re * cplx8_32(k,j).re - cplx8_8(i,k).im *
cplx8_32(k,j).im;
mult_temp (i, j * 8 +k).im := cplx8_8(i,k).re * cplx8_32(k,j).im + cplx8_8(i,k).im *
cplx8_32(k,j).re;
end loop;
end loop;
end loop;

```

```

for i in 0 to 7 loop
for j in 0 to 31 loop

```

```

mult_temp_1(i,j).re := mult_temp(i, 8 * j + 0).re +mult_temp(i, 8 * j + 1).re +
mult_temp(i, 8 * j + 2).re +mult_temp(i, 8 * j + 3).re +

```

```

mult_temp(i, 8 * j + 4).re +mult_temp(i, 8 * j + 5).re + mult_temp(i, 8 * j + 6).re
+mult_temp(i, 8 * j + 7).re;
mult_temp_1(i,j).im := mult_temp(i, 8 * j + 0).im +mult_temp(i, 8 * j + 1).im +
mult_temp(i, 8 * j + 2).im +mult_temp(i, 8 * j + 3).im +
mult_temp(i, 8 * j + 4).im +mult_temp(i, 8 * j + 5).im + mult_temp(i, 8 * j + 6).im
+mult_temp(i, 8 * j + 7).im;
end loop;
end loop;

```

--RESIZING

```

for i in 0 to 7 loop --num_rows1 -1
for j in 0 to 31 loop --num_col2 -1
mult(i,j).re := resize(mult_temp_1(i,j).re, nh_m,na_m);
mult(i,j).im := resize(mult_temp_1(i,j).im, nh_m,na_m);
end loop;
end loop;
return mult;
end function;

```

--32x32 MATRIX MULTIPLICATION

```

function matrix_mult_32x32( cplx32_8: re_im_matrix_32x8; cplx8_32:
re_im_matrix_8x32_final ) return re_im_matrix_32x32 is

```

```

variable i,j,k: integer :=0;
variable mult: re_im_matrix_32x32;
variable mult_temp: re_im_matrix_32x256;
variable mult_temp_1: re_im_matrix_32x32_temp;

```

begin

```

for i in 0 to 31 loop
for j in 0 to 31 loop
for k in 0 to 7 loop

```

```

mult_temp (i, j * 8 +k).re := cplx32_8(i,k).re * cplx8_32(k,j).re - cplx32_8(i,k).im
* cplx8_32(k,j).im;
end loop;
end loop;
end loop;

```

```

for i in 0 to 31 loop
for j in 0 to 31 loop

```

```

mult_temp_1(i,j).re := mult_temp(i, 8 * j + 0).re +mult_temp(i, 8 * j + 1).re +
mult_temp(i, 8 * j + 2).re +mult_temp(i, 8 * j + 3).re +
mult_temp(i, 8 * j + 4).re +mult_temp(i, 8 * j + 5).re + mult_temp(i, 8 * j + 6).re
+mult_temp(i, 8 * j + 7).re;
end loop;

```

```

end loop;

-- RESIZING
for i in 0 to 31 loop
for j in 0 to 31 loop

mult(i,j).re := resize(mult_temp_1(i,j).re, nh_m,na_m);
end loop;
end loop;
return mult;

end function;

--DIAGONAL MATRIX---
function diag(cplx32_32: re_im_matrix_32x32) return vect_32 is

variable i,j: integer :=0;
variable diag_vector: vect_32;

begin
for i in 0 to 31 loop
diag_vector(i) := cplx32_32(i,i).re;
end loop;

return diag_vector;
end function;

end matrix_package;

```

B.8: VHDL Testbench of Phi (a'.R.a) Matrix

--This is the final TESTBENCH is written to see DOA estimation results.
--There is a artificial clock which period is 10 ns.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

library ieee_proposed;
use ieee_proposed.fixed_pkg.all;
use work.matrix_package.all;

entity phi_matrix_tb is end phi_matrix_tb;

architecture TestBench of phi_matrix_tb is

    component phi_matrix is
    Port (
        clk: in std_logic;
        o_phi_matrix: out re_im_matrix_32x3 );
    end component;

    --This matrix includes angle values.
    signal matrix_phi_o: re_im_matrix_32x32;
    signal clk: STD_LOGIC;
    constant clock_period: time := 10 ns;

begin
    uut: phi_matrix port map (clk, matrix_phi_o);

    --This process shows the duration of simulation
    stimulus: process
    begin
        wait for 500 ns;
        report "sim end" severity failure;
    end process;

    --This process is used to create artificial clock pulse
    clk_process: process
    begin
        clk <= '0';
        wait for clock_period/2;
        clk <= '1';
        wait for clock_period/2;
    end process;
end TestBench;
```