Design and Implementation of a Fixed-Point Systolic Architecture for an MVDR Wide-Band Beamformer in FPGA

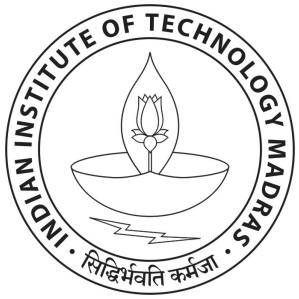
A THESIS

submitted by

GIRISH M

for the award of the degree of

MASTER OF TECHNOLOGY



DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.

JUNE 2014

THESIS CERTIFICATE

This is to certify that the thesis titled **Design and Implementation of a Fixed-Point Systolic Architecture for an MVDR Wide-Band Beamformer in FPGA**, submitted by **Girish M**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology** , is a bona fide record of the work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Nitin Chandrachoodan

Advisor,

Associate Professor,

Dept. of Electrical Engineering,

IIT-Madras, 600 036

Place: Chennai

Date: 13th June 2014

ACKNOWLEDGEMENTS

I would like to thank my Guide Dr.Nitin Chandrachoodan for giving me an opportunity to work under his guidance and for supporting me with valuable information and suggestions throughout my project work. I would also like to express my sincere gratitude to him for giving me enough freedom to explore a new tool and showing the right directions to proceed. I also take this moment as an opportunity to thank Sh. Ananthanarayanan, Director, NPOL, my parent organization, Sh. S.K.Shenoi, Group Head and Smt. Subhadra Bhai, Division Head for supporting me to pursue my M.Tech program.

I would like to thank Dr.Srikrishna Bashyam for the motivating Adaptive Signal Processing classes and Dr. Dileep Nair for being my faculty adviser and supporting me throughout my course here. Great thanks to Ajmal, Pavan, Jobin, Karthikeyan in particular and other members of ESL group for giving valid inputs through their presentations and mode of work. Thanks are due to Ramprasad who inspired me with his skills in Latex and prompted me to use the same. Also, I would cherish the time that I had spent with friends like Vijay, Syed, Avinash and Mahesh in the DSDL lab. Special thanks to Janaki Madam for giving an access to the DSDL lab.

I would like to thank all my batchmates especially Senthil and Selwin for being my good friends for these two years and helping me out in domestic issues. I cannot miss to acknowledge my family here especially my kids Gayatri and Giridhar who were not so troublesome to their Mother during these two years. I have no words to express my gratitude to my wife, Shiny, for taking care of my kids and certain

familial and domestic issues in my absence.

This thesis is dedicated to my family and to all my loved ones.

ABSTRACT

This thesis presents the design and implementation of a Fixed point Systolic Architecture for an MVDR Wide-Band Beamformer. The design is accomplished using a systolic array approach minimizing the latency and the resources consumed without trading off much on speed and the signal processing constraints, which basically include the SNR requirements and stationarity assumptions. Besides the design itself is scalable to meet larger array and wider band requirements. Moreover the proceesing elements in the array are pipelined and regular in nature, which makes it more tractable.

The goal is to simulate the design and realize it on a Xilinx FPGA platform and evaluate the results for various fixed-point precisions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

CBF	Conventional Beamformer
MVDR	Minimum Variance Distortion-less Response
CSD	Cross Spectral Density
FPGA	Field Programmable Gate Array
ASIC	Application Specific Integrated Chip
DSP	Digital Signal Processor
GPU	Graphics Processing Unit
SNR	Signal to Noise Ratio
FFT	Fast Fourier Transform
CORDIC	COordinate Rotation DIgital Computer
PE	Processing Element
MRA	Maximum Response Axis

CHAPTER 1

Introduction

1.1 Motivation

Beamformer is an essential component in sensor arrays used in any sonar or radar application irrespective of the deployment, be it Commercial or Military, especially because of the SNR improvement that it provides. Adaptive Beamformer differs from a Conventional Beamformer in its beam pattern. The latter has a fixed Beam pattern depending on the geometry of the array, while the Beam pattern of the former adapts to the environment depending on the amount of interferences. A conventional Delay-Sum Beamformer and its beam pattern are shown in Figure.1.1.

MVDR Beamformer [12] tries to minimize the variance or power of the combined output from a sensor array in a least square sense subject to a constraint that the signal from a desired direction is passed in without distortion. The Power Plots of the Conventional and MVDR Beamformer are shown in Figure.1.2.

The solution to this problem amounts to computation of a set of adaptive weights which in turn involves the computation of inverse of the Cross-Correlation/Cross-Spectral-Density Matrix. The Inverse computation is of the order of N^3 operations, where N is the size of the input data. There are techniques available in literature to compute the weights without performing the inverse computation. One such technique is explored and applied to the problem in this project and a new architecture to realize the same in fixed-point precision has been proposed.

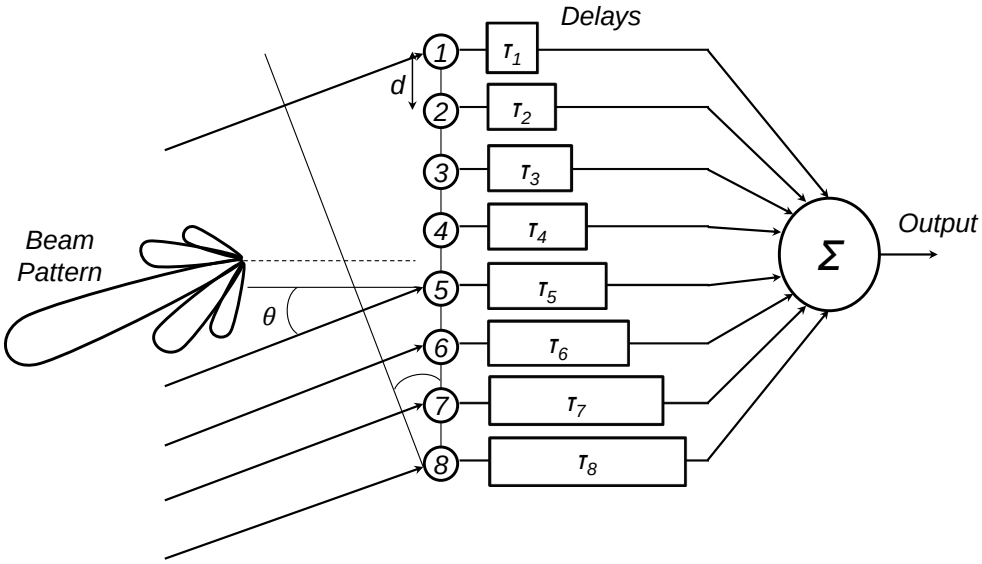


Figure 1.1: Beam Pattern of a CBF.

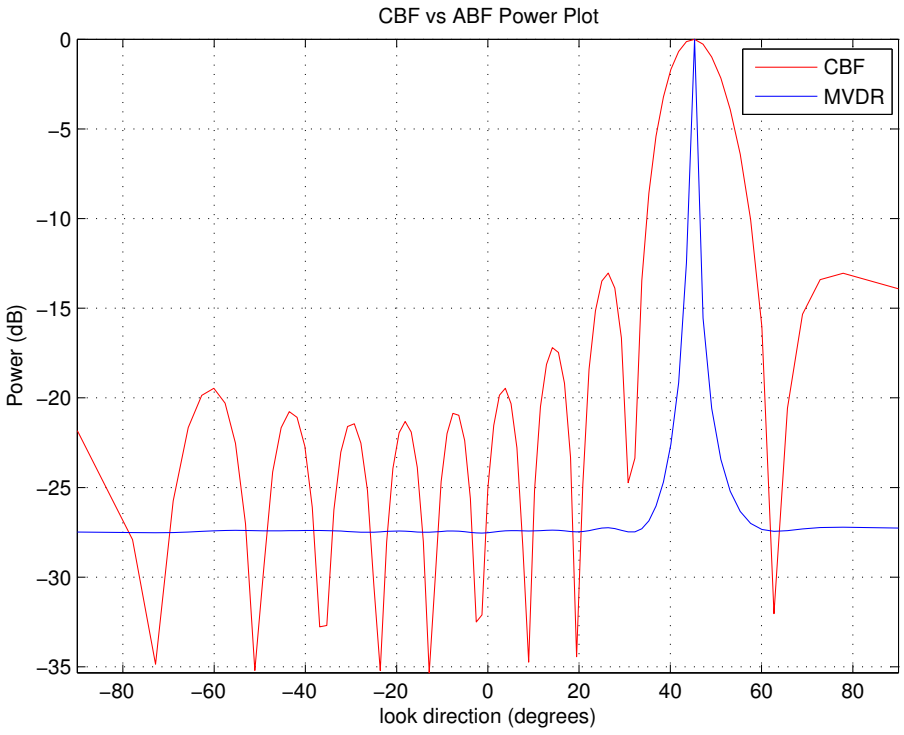


Figure 1.2: Energy Plots of a CBF & MVDR BF.

1.2 Objective

The main objective of this work is to design and implement an MVDR wide-band Beamformer using fixed-point precision in FPGA with an intent to minimize the latency and resources consumed without trading off much on speed and signal processing contraints like SNR and stationarity assumptions. The scope of the work confines to sensor arrays which are either linear or circular in nature which can cater for an azimuthal resolution in the range of 1 to 2 degrees. The band of operation of the array is assumed to be in lower range typically from 100 Hz to 15 KHz with a maximum band width of 5 KHz. This band is further split into narrow bands with a resolution of the order of 100 to 200 Hz. The array size, number of sub apertures and the number of look Angles per aperture are decided by the application and whether the architecture can meet these specifications depends on the operational clock frequency of the Systolic Array.

1.3 Thesis organisation

The rest of the thesis is organized as described below.

Chapter 2 discusses about the MVDR Beamformer and its adaptation to cater for a wider band and the parameters of significance. A typical array of interest is considered and the memory and computational requirements are discussed. This chapter also deals with the technique adopted to realize the inverse free adaptive MVDR beamformer and explores in brief the other existing platforms that are used to realize the same application and their pros and cons.

Chapter 3 explains in detail the proposed final architecture along with architectures which were used to arrive at the final one. This chapter also discusses about the implementation approach, tools and the IP cores used to realize the architecture.

Chapter 4 summarizes the performance analysis with simulation results.

CHAPTER 2

MVDR Wide-Band Beamformer

2.1 Adaptation

The MVDR Beamformer proposed is a wide-band beamformer for an M sensor array. The array could be of any arbitrary geometry and the path delays that the signal encounters while impinging the array matter. The signal is assumed to be narrow band or of single frequency. In order to cater for a wider band, the desired band is split into multiple narrow bands and an MVDR beamformer is constructed for each narrow band. Hence the computations are done in the frequency domain rather than in the time domain.

The Sensor array data is received as a block of time series.This time series is normalized to have zero mean and unit variance prior to the beamforming. The preprocessed data is used to compute the FFT and the frequency bins in the band of interest are selected for the beamforming.

Supposing there are K bins of interest, the number of cross-spectral density matrices that have to be computed are K each of dimension $M \times M$. The number of steering vectors required for N is of the order of $N \times K$ each of dimension $M \times 1$. The steering vectors can be grouped to form a 3d matrix of size $N \times K \times M$. The adaptive weights are computed using the following formula

$$w_j(\theta) = \frac{R_j^{-1} A_j(\theta)}{A_j^H(\theta) R_j^{-1} A_j(\theta)} \tag{2.1}$$

where

θ is the look angle or Beam direction,

j is a bin of interest ranging from 1 to K,

$A_j(\theta)$ is the steering vector corresponding to the j^{th} bin in the look angle θ and is computed as $e^{j\omega\tau_i}$, τ_i are the respective path delays to each of the sensors in the array. In equation (2.1), R_j is the Cross-Spectral Density Matrix corresponding to the j^{th} bin and is obtained by

$$R_j = E\left[X_j X_j^H\right] \tag{2.2}$$

where X_j in equation (2.2), is the array snapshot vector or bin vector comprising of the FFT coefficients w.r.t j^{th} bin from the M sensors. The power associated with each bin in the given look angle is given as

$$P_j(\theta) = \frac{1}{A_j^H(\theta) R_j^{-1} A_j(\theta)} \tag{2.3}$$

and the net power in the given look angle is just the sum of all the individual powers of the bins

$$P(\theta) = \sum_{j=1}^{K} P_j(\theta) \tag{2.4}$$

2.2 Computational Analysis

2.2.1 Flow Diagram

The flow diagram for the MVDR Wide-Band Beamformer is as depicted below in Figure.2.1.

The digital time series data from the sensors is subjected to FFT and the bins which fall in the band of interest are extracted to form bin vectors. In practice equation (2.2) cannot be estimated by time averaging because the bin vector or the observations in the frequency domain is never truly stationary and/or ergodic. As a result the available averaging time is limited. One approach of time varying

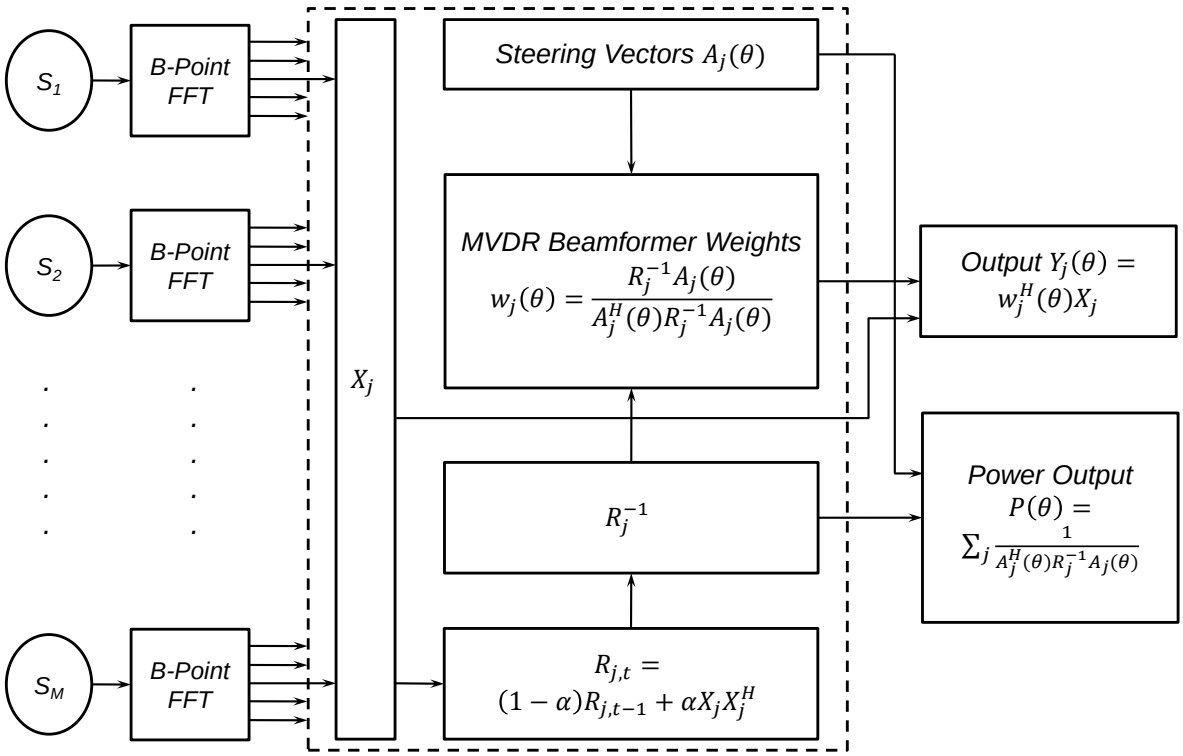


Figure 2.1: Flow Diagram

adaptive estimation of R_j proposed by Owsley [2] is using an exponential averager which is given by

$$R_{j,t} = (1-\alpha)\, R_{j,t-1} + \alpha X_{j,t} X_{j,t}^H \tag{2.5}$$

where α is a smoothing factor. In dynamic situations where the direction of arrival of a particular interference is varying with time, the effective averaging time for estimating the inter-element CSD Matrix R_j is limited by a temporal stationarity assumption. Thus the variance of the elements which contribute to R_j, which is actually inversely proportional to the averaging time, has a lower bound determined by the finite averaging time. Specifically if N_I is the number of statistically independent vectors of X_j, which are exponentially averaged to obtain a stable and invertible estimate of R_j, then the variance on the beam power estimator statistic as found in (2.3) is inversely proportional to $N_I - M + 1$, and $M < N_I$. Usually N_I is of the order of 3 to 4 times of M by empirical observations

and theoretical suggestions.

1	2	3	...	N_I
B Samples	B Samples	B Samples	...	B Samples

For every snapshot of B Samples, the following tasks have to be carried out.

1. B-point FFT for M Sensors.
2. Form bin vectors X_j and find $X_{j,t}X_{j,t}^H$ for all the K bins in the band of interest.
3. Update R_j as in equation (2.5)

After N_I snapshots

1. Compute R_j^{-1} for all the K bins.
2. Find optimal weights as in equation (2.1).
3. Find $\|w_j^H(\theta)\, X_j\|^2$ for all the K bins and sum them to obtain the total power in a particular look direction θ.
4. Repeat steps 2 & 3 to get the power output in all the look directions, N.
5. Alternatively steps 2 & 3 can be skipped to compute the power directly by using the equations (2.3) and (2.4).
6. Repeat step 5 for all the look directions, N.

2.2.2 Computational Requirements

The computations involved in the foreground process of updating the R_j Matrices are

1. For FFT say N_F operations.
2. To find $X_{j,t}X_{j,t}^H$, $M^2K\,(4Mult + 2Add)\,/2 + MK\,(4Mult + 2Add)\,/2$ operations.
3. To update R_j, $M^2K\,(4Mult + 2Add)\,/2 + MK\,(4Mult + 2Add)\,/2$ operations.

Therefore the total number of operations for N_I snapshots is $N_I\left(N_F + \left(M^2K + MK\right)\left(4Mult + 2Add\right)\right)$ and they have to be performed in a span of $N_I BT_S$ units of time.

The number of operations involved in computing the power estimate for various look directions are as listed below,

1. To find R_j^{-1} for all the K bins, KM^3 operations.
2. To find power in all the N look directions, $NK\left(\left(M^2 + M\right)\left(4Mult + 2Add\right) + \left(MAdd\right)\right)$ operations.

Therefore the total number of operations involved in the above process of computing the power for various look angles is of the order of $KM^3 + NK\left(\left(M^2 + M\right)\left(4Mult + 2Add\right) + \left(MAdd\right)\right)$ and they have to be completed in a span of $N_I BT_S$ units of time. Assuming a lower bound for $N_I \simeq 4M$, the timing requirement that restricts the pipelining of the above two processes can be obtained as $4MBT_S$ units of time.

2.2.3 Case Study

A typical case of interest is considered here and the specifications are as follows

Main Array	Circular Array of 32 Sensors
Aperture size	12 Sensors
Resolution	1.25^0
Number of Look Angles/Aperture	9
Band of Interest	1KHz to 3.5KHz
Sampling Frequency	24KHz
FFT size	256
Number of Bins of Interest	28

Table 2.1: Specifications

For the case considered above, where $M = 12$, $B = 256$ and $T_S = 41.67\mu$s the timing parameter which governs the pipelining of the above two processes turns out to be 0.512 secs. This timing parameter is what is referred to as the **Averaging Period**.

2.3 Robust Inverse Free Square Root Algorithm

The MVDR weights in equation (2.1) requires the computation of the inverse of the cross spectral density matrix R_j^{-1}. For a broadband beamformer, this has to be done for all the frequency bins in the band of interest. Since the Inverse computation is intensive and requires operations of the order of M^3, a better way to compute the weights was proposed by Owsley.

2.3.1 Inverse Free Method

Relooking at the Optimal weights equation (2.1), It is clear that to get rid of the Inverse term, some algebraic manipulations are required. In that point of view, equations (2.1) and (2.3) can be re-written as,

$$w_j(\theta) = \frac{p_j(\theta)}{A_j^H(\theta)\, p_j(\theta)} \tag{2.6}$$

and

$$P_j(\theta) = \frac{1}{A_j^H(\theta)\, p_j(\theta)} \tag{2.7}$$

respectively. Where,

$$p_j(\theta) = R_j^{-1} A_j(\theta) \tag{2.8}$$

and

$$A_j(\theta) = R_j p_j(\theta) \tag{2.9}$$

From (2.9) and (2.6), It is evident that the optimal weights could be computed in two steps.

Step1 involves the solution of expression given in equation (2.9) for vector $p_j(\theta)$.

Step2 involves substituting the solution obtained in Step 1 for $p_j(\theta)$ in equation (2.6).

2.3.2 Cholesky Factorization of CSD Matrix

The cross spectral density matrix, is usually computed using the equation,

$$R_j = X_j X_j^H \tag{2.10}$$

where the Expectation in equation (2.2) has been replaced by stochastic approximation.

Applying Cholesky Fcatorization on the CSD Matrix, equation (2.9) can be rewritten as

$$A_j(\theta) = L_j L_j^H p_j(\theta) \tag{2.11}$$

where L_j is the Cholesky factor or the Square root of R_j and is a Lower triangular matrix. L_j^H is the complex transpose of L_j. Substituting for $L_j^H p_j(\theta)$ as $u_j(\theta)$ results in

$$A_j(\theta) = L_j u_j(\theta) \tag{2.12}$$

and

$$u_j(\theta) = L_j^H p_j(\theta) \tag{2.13}$$

Further Algebraic manipulation leads to

$$A_j^H(\theta)\, p_j(\theta) = \| u_j(\theta) \|^2 \tag{2.14}$$

thereby reducing the optimal weights in equation (2.6) and power in equation (2.7) as follows

$$w_j(\theta) = \frac{p_j(\theta)}{\|u_j(\theta)\|^2} \tag{2.15}$$

and

$$P_j(\theta) = \frac{1}{\|u_j(\theta)\|^2} \tag{2.16}$$

The problem of solving for weights and the power estimate has narrowed down to solving two quantities $u_j(\theta)$ and $p_j(\theta)$ Wherein, $u_j(\theta)$ could be computed from equation (2.12) by forward substitution as L_j is a lower triangular matrix. Having computed $u_j(\theta)$, $p_j(\theta)$ could be obtained from equation (2.13) by a back substitution.

2.3.3 Adding Robustness

The MVDR weights are optimized by minimizing the total power at beamformer output with a unity gain constraint for a signal from a desired Look Angle. This constraint makes it a highly selective spatial filter. If the true target DOA is slightly off the desired Look Angle, the signal will be treated as interference. Therefore the power output in the beamformer for targets which lie in between the Look Angles will be reduced. To mitigate this effect it is desirable to slightly broaden the spatial response of the filter so as to make it robust. One simple method to achieve this is diagonal loading, in which a small value is added to the diagonal elements of the CSD matrix R_j initially at the beginning of every averaging period,i.e.

$$R_j = R_j + \varepsilon I \tag{2.17}$$

where $\varepsilon = 0.01$

The statistics of the interference will generally be non-stationary as discussed earlier and hence the CSD Matrices are estimated using an exponential averager with a forgetting/smoothing factor as in equation (2.5). The smoothing factor α can be obtained as

$$\alpha = \frac{1}{1 + \frac{T_c}{T_a}} \tag{2.18}$$

where T_c is the Averaging Period or the Time constant of integration and T_a is the time required to collect one snapshot of array data from M sensors. For the better estimate of the CSD matrix, the criteria on time constant has already been discussed. The ratio of $\frac{T_c}{T_a}$ should be chosen accordingly.

2.3.4 Cholesky Factor Update

The Cholesky factor of R_j can be updated directly from the Bin vectors without explicitly computing the cross spectral density matrix R_j. The procedure for updating the Cholesky factor L_j is similar to the way the CSD Matrices are updated as in equation (2.5). The update can be rewritten as

$$L_{j,t} L_{j,t}^H = (1 - \alpha)\, L_{j,t-1} L_{j,t-1}^H + \alpha X_{j,t} X_{j,t}^H \tag{2.19}$$

The new update of the Cholesky factor $L_{j,t}$ can be obtained recursively by means of a unitary transformation. If Q is the unitary transformation, then equation (2.19) can be written as

$$L_{j,t} L_{j,t}^H = \left([\beta L_{j,t-1} \mid \gamma X_{j,t}]\, Q\right) \left([\beta L_{j,t-1} \mid \gamma X_{j,t}]\, Q\right)^H \tag{2.20}$$

Where $\beta = \sqrt{1-\alpha}$ and $\gamma = \sqrt{\alpha}$

The transformation forces the elements in $X_{j,t}$ to zero as illustrated below,

$$[\beta L_{j,t-1} \mid \gamma X_{j,t}]\, Q \rightarrow [\beta L_{j,t} \mid 0] \tag{2.21}$$

The above unitary transformation is accomplished using two Givens rotations [6], the first involving the bin vector X_j alone

$$\begin{vmatrix} re\,(x_1) & im\,(x_1) \\ re\,(x_2) & im\,(x_2) \\ . & . \\ re\,(x_M) & im\,(x_M) \end{vmatrix} Q \rightarrow \begin{vmatrix} re\,(\hat{x}_1) & 0 \\ re\,(\hat{x}_2) & im\,(\hat{x}_2) \\ . & . \\ re\,(\hat{x}_M) & im\,(\hat{x}_M) \end{vmatrix}$$

and the second rotation involving the updated bin vector $\hat{X}_j$ and the Cholesky factor L_j

$$\begin{vmatrix} l_{11} & re\,(\hat{x}_1) \\ re\,(l_{12}) + j{*}im\,(l_{12}) & re\,(\hat{x}_2) + j{*}im\,(\hat{x}_2) \\ . & . \\ re\,(l_{1M}) + j{*}im\,(l_{1M}) & re\,(\hat{x}_M) + j{*}im\,(\hat{x}_M) \end{vmatrix} Q \rightarrow \begin{vmatrix} \bar{l}_{11} & 0 \\ re\left(\bar{l}_{12}\right) + j{*}im\left(\bar{l}_{12}\right) & re\,(\bar{x}_2) + j{*}im\,(\bar{x}_2) \\ . & . \\ re\left(\bar{l}_{1M}\right) + j{*}im\left(\bar{l}_{1M}\right) & re\,(\bar{x}_M) + j{*}im\,(\bar{x}_M) \end{vmatrix}$$

The above two Rotations continue until all the elements in X_j are nullified updating all the columns of the Cholesky factor L_j. The Update process continues until the Averaging period and the Weights and the Power are computed by solving for u_j and p_j, substituting the updated Cholesky factor in equations (2.12) and (2.13) respectively.

2.4 Beamformer on various Platforms

The platforms used to implement the Beamformer or in general a Signal Processing application range from ASICs to General Purpose DSPs to GPUs and FPGAs these days. Choosing a platform and precision to realize most of the Signal Processing applications depend on the dynamic range of the data to be processed. There are certain applications which could very well be implemented using fixed-point precision rather than floating point precision depending on the dynamic range of the data to be processed. Though ASICs were designed to perform a specific application, they were not scalable and the demanding requirements to cater for larger arrays and highly computational intensive Algorithms paved way for various other platforms like DSPs, GPUs and FPGAs.

2.4.1 DSP based approach

General Purpose DSPs are limited by the computing power they possess. Though they are easily programmable, they are bound by the incoming data rates and the sequence of tasks which need to be performed on the incoming data. They have a fixed architecture and generally do not support pipelining of the data. Consider the case of an Analog Devices Tiger SHARC processor which operates at 500 MHz bus speed and 200 MHz core clock. Though the core operates at high clock speed, it is restricted by the resources to perform a highly computation intensive algorithm which acts on multiple sensors at the same time, on its own. As in the case of a wide-band Beamformer application, the incoming data time series is subjected to a frequency domain transformation using an FFT processor. A single FFT function call or subroutine may have to cater for all the sensors in the DSP, whereas each sensor could be handled by an individual FFT processor in the FPGA. This can drastically bring down the FPGA clock speed and thereby the power dissipated considering the power per unit area metric. Moreover the

tasks to be performed on the data, have to be categorized into foreground and background processes in the case of DSPs. The number of having such processes is limited and scheduling of these processes is again critical and needs to be done with proper care to avoid stalls and conflicts. The job of partitioning the tasks into sub tasks so as fit into DSPs becomes crucial here and hence the DSPs need to be chosen depending on the application. For instance in the case of a wide-band MVDR beamformer, one approach would be to divide the process of updating the CSD Matrices for the various bins of interest among various cores within a DSP and have a cluster of such DSPs to handle a set of apertures. This would be a foreground process along with the computation of instantaneous beamformer outputs and FFT on the incoming data. The computation of adaptive weights followed by the beamformer output power which is a slower process will be run as a background process in parallel. Thus the area over which an application is realized increases enormously by having a cluster of DSPs, though decreasing the power dissipation but increasing the space requirements which are critical in certain space and defence applications.

2.4.2 GPU based approach

GPUs have potential computational power and basically behave as a parallel co-processor to the CPU capable of outperforming a general CPU in terms of FLOPS and Bandwidth. The stream processor or the cores within the GPU behave like processing elements which enable data parallelism. The architecture also supports pipelining. Though they have high computational power, they have their own restrictions in the form of limited memory, be it local memory or shared memory and the pattern in which the data are accessed. Moreover the interface between the CPU and the GPU also is critical in terms of bandwidth. It has also been observed that GPUs are efficient as long as they operate on large chunks of data in one shot which tries to utilize all the cores rather than performing operations on

small chunks of data. Hence the CPU has to cater for a high speed bus interface with sufficiently large memory in order to utilize the computing power of GPU to its fullest extent. Besides they are not always reliable for hard real-time applications because of the scheduling overheads and the above mentioned limitations. The same rationale with regard to task partitioning holds for GPUs as well. Moreover the overheads involved in running the application in GPUs and DSPs are more as the applications are realized in the form of software which is either interrupt driven or controlled by a handshaking mechanism which can hamper the hard real-time behaviour of the applications which has sampling rates of the order of GHz. Though the modern DSPs have addressed some of the above mentioned issues with evolving architectures, the software to realize the application needs to be written taking into account all the restrictions imposed by the processor and the compiler together.

2.5 why FPGA based approach

FPGAs on the other hand with huge resources of logic cells, DSP blocks and memories and higher operational speeds are a better choice to realize hard real-time applications as the designs are realized as hardware which are data driven in most cases and not restricted by any software overheads which include scheduling and function calls of interruptive nature. The designs are often realized using multiple instances running in parallel to meet the demanding requirements put forth by multi-sensor applications. The designs are scalable in nature by utilizing the unused resources within the FPGA with the in-system programmability feature. The other significant advantage of FPGA is that the designs could be realized using fast parallel pipelined architectures which relax the clock speed and memory requirements resulting in periodic and regular hardware blocks. In short, FPGAs become an obvious choice because of the following reasons

1. The designs are realized as hardware and are scalable

2. They provide in-system programmability

3. They are free from software overheads

4. Fast Parallel Pipelined Architectures which often result in periodic and regular structures relaxing the constraints can be realized

With the advent of time-area optimized Intellectual Property (IP) Cores, it is no more difficult to realize any signal processing algorithm in FPGAs. The critical component in the realization is however meeting all the timing requirements with respect to the design realization.

CHAPTER 3

Design Philosophy

3.1 Systolic Array based Design

The proposed architecture is parallel pipelined systolic array architecture. The term Systolic comes from human cardiac system to describe the circulation of blood to and from heart through blood vessels to various parts of the body. From the VLSI point of view, it is analogous to the data being pumped into a Processing Element (PE) and pumped out of it after getting processed. Ideally the data should be continuous so that PEs can keep processing them and pump the processed data out of the system. In other words the PEs shouldnt starve for data.

3.2 Existing Architecture

Gentleman & Kung [1] were the first to propose a Systolic array based approach to compute the weights using the Recursive Least Squares (RLS) method. The approach had two steps basically a QR update from the observation vectors followed by a Back-Solve process. This work was followed by Hudson & Shepherd [4] who proposed a kalman closed loop feedback structure, which computes the weights based on the error residuals in a recursive manner. Many authors like Schreiber, Mcwhirter, Owsley, Liu, Rader etc [2], [3], [5], [7], [8] and more recently Dick and Harris [11] followed up with really good work proposing and publishing literatures on the proposed architectures either individually or teaming up with others which can be considered to fall in one of the two following categories. The first category comprises of architectures where the weights are computed recursively in every iteration, whereas the second comprises of architectures where the

weights are computed over a period of time and the iterations are meant only to update the intermediate data which is used subsequently, after a prescribed number of iterations, in weights computation. All the architectures proposed use, in one-way or the other, orthogonalization techniques which make use of unitary transformations like *Givens*, *Modified Givens*, *Gram-Schmidt* and *House-Holder* transformations etc on the observation vectors.Authors like Frantzeskakis, Liu [9] and Gotze, Schwiegelshohn [10] have also proposed architecture to avoid divisions in the weights computation.

3.2.1 Features

Most of the authors have emphasized and tried to achieve architectures with the following features

- Single and pipelined in nature
- Consume least memory resources
- Provide maximum parallelism
- Compute Weights in every iteration

3.2.2 Desired Features

All the above architectures can very well be adopted for the problem under consideration which is a wide-band MVDR Beamformer. A wide-band MVDR beamformer as explained in previous chapter comprises of a number of narrow band beamformers depending on how finer the wider band is split into. From the Adaptation proposed in the previous chapter, the incoming time series is subjected to FFT to get the narrow bands or the centre frequencies of narrow bands in the wider-band of interest. Adopting the above proposed architectures would result

in a 2-D Systolic array comprising of K 1-D Systolic Arrays catering for each of the K narrow bands considered. The architecture further blows up to a 3-D Systolic Array by adding another dimension which is the number of Apertures considered within the Sensor Array. Though this architecture is memory-less and provides maximum parallelism, the amount of resources consumed is enormous. Moreover the latency of most of the proposed architectures are of the order of the size of Observation vectors as these vectors are pumped in one element at a time to the Systolic Array.

So the objective here is to propose a fast parallel pipelined architecture that would minimize the resources along with the latency in the pipeline without trading off much on the timing constraints put forth by the algorithm. The architecture proposed as part of the thesis is two-step process and basically involves the Cholesky factor update as the first and two Back-Solve processes in cascade as the second step. The first of the Back-Solve processes in cascade aids in computing the power in the beam while the second uses the output from the first to compute the weights for each of the narrow band beamformers considered. Each step is realized as an independent systolic array with adequate memory to interface between the arrays and accommodate all the narrow bands and apertures enabling in diminishing the 3-D Systolic Array to a 2-D Systolic Array. The proposed architecture is a square-root free architecture and makes use of CORDIC algorithm to realize the Givens Rotations used in the Cholesky factor update step.

In adaptive filters, it is of utmost importance how quicker the weights are computed and applied to the new set of observation vectors. The minimum number of iterations required to converge to the optimal weights is fixed for a given algorithm and for a given set of parameters. Given these constraints how faster the hardware can accomplish this goal is the question. The answer lies in Latency. Reducing the Latency or the time which the data spends in the pipeline helps in accomplishing this goal.

For the MVDR wide-band Beamformer under consideration, the first step involv-

ing the updating of Cholesky factors of the CSD matrices with the incoming bin vectors and the second step involving the computation of the adaptive weights and the beamformer output power from the updated Cholesky factor are executed in tandem except that the latter phase operates on the updated Cholesky factor obtained from the previous averaging cycle. This can be better explained using a timing diagram as shown in Figure.3.1.

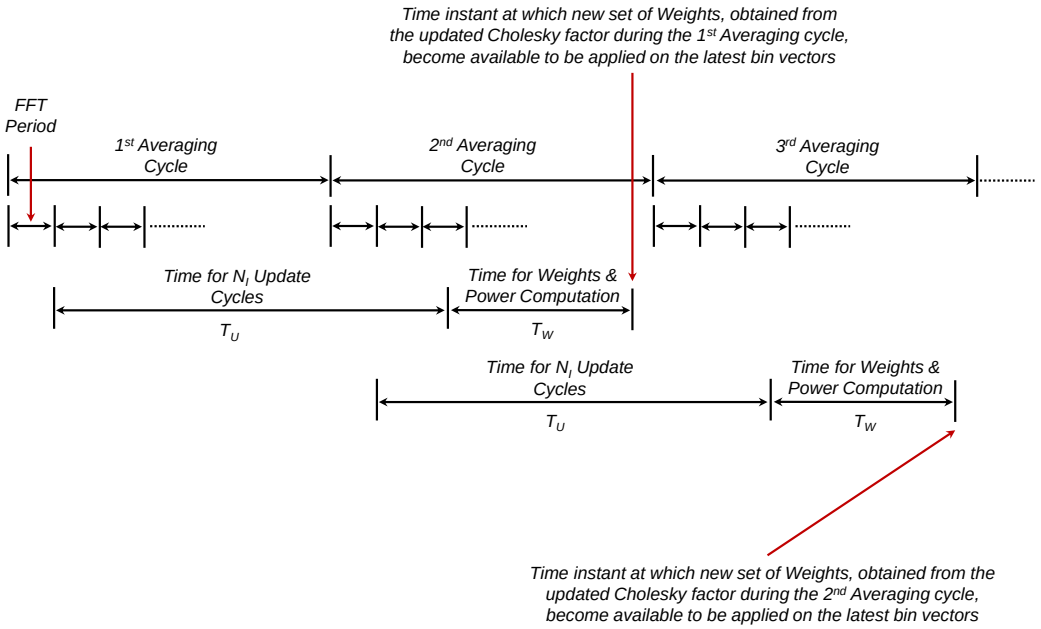


Figure 3.1: Timing Diagram.

From the philosophy of pipelining, the update period T_U could very well be extended till the start of the third averaging cycle and there is no restriction on weights computation period T_W as such. From the algorithm point of view, the quicker the weights are computed and applied on the new bin vectors, the better the estimate at the beamformer output and thereby better reconstruction of the dynamically varying scenarios. Hence the total time required to compute the weights i.e. $(T_U + T_W)$ must be minimized. The lower limit for this quantity

is however the time taken for one averaging cycle or the time taken to update the Cholesky factor of the CSD matrices for N_I iterations or snapshots. But this could end up in an architecture which consumes more resources and the hardware realizing the weights and power computation module remaining idle for most of the time. The latency mentioned here has two components, one associated with the pipeline architecture for Update module and the other associated with the architecture for weights and power computation module. One way of bringing down the latency is to apply the observation vector as a whole to the systolic array rather than one element at a time. Another option is to avoid the reverse paths in the Back-Solve process as seen in most of the existing architectures. Though the options could consume additional resources, they bring down the latency significantly and have been incorporated in the proposed architectures.

The basic block diagram of the MVDR wide-band Beamformer is as shown in Figure.3.2.

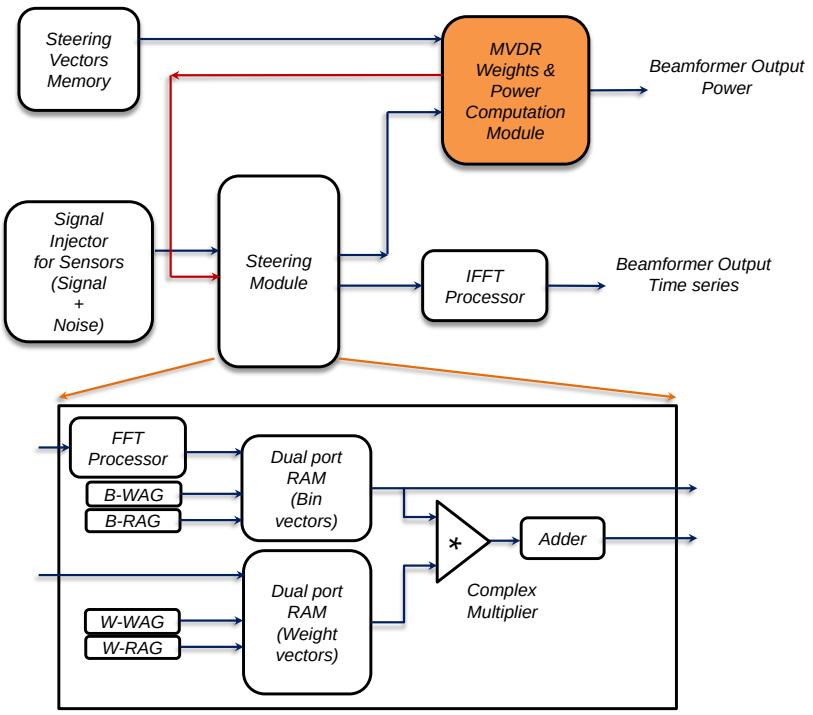


Figure 3.2: Block Diagram.

The Steering Module in the block diagram tries to compute the dot product between the Weight vectors and the appropriate Bin vectors $w_j^H(\theta) X_j$ as discussed in the computational analysis in Chapter 2. The output from the Steering module is fed to an Inverse FFT module to get the beamformer output time series. The Steering Module comprises of an FFT module along with the Dual port RAMs and their respective address generators to hold the Bin vectors and Weight vectors. If the weight vectors Dual port RAM is initially loaded with the Steering vectors, then the Beamformer would behave like a conventional Beamformer in the absence of MVDR Weights and Power Computation module. So the major block or the block of importance is the MVDR Weights an Power Computation module. Three Architectures have been proposed for Cholesky factor update each evolved out of the former with an intent to reduce the resources and latency. The architecture for the Back-Solve process is developed as an independent Systolic array with intermediate buffering of the updated Cholesky factors to derive the weights and power.

3.3 Architecture for Cholesky factor update

3.3.1 Architecture 1

3.3.1.1 Concept

Figure.3.3 shows the schematic of the architecture. Here the observation vectors which are basically the Bin vectors are applied as a whole to the first stage. The first real rotation as discussed in the adaptation part of Chapter 2, is accomplished using two vectoring mode operations of CORDIC algorithm. The first complex element in the observation vector is subjected to a translation (T *in* $Rotation1$) resulting in an angular output and a real component which is basically the magnitude of the complex input. The angular output is fed as an input to the second

vectoring mode operation ($R\ in\ Rotation1$) along with the other complex elements in the observation vector. Hence they have to be delayed till the angular output from the first vectoring mode operation of the first rotation is obtained. Meanwhile the magnitude output obtained from the translation operation of the first element of the observation vector forms the imaginary input to the first translation operation in the second rotation step ($T\ in\ Rotation2$) to get the angular output required for the second complex rotation. The real input is the first element in the first column of the Cholesky factor, as the diagonal elements in the Cholesky factor are always real. The angular output is used to generate the elements of the rotation matrix required for the second complex rotation which are basically the sine and cosine of the angular input. This is done using a rotation mode operation ($S\ in\ Rotation2$) of the CORDIC algorithm. The remaining complex elements in the first column of the Cholesky factor and the complex vector obtained from the first rotation form two columns of the matrix to be multiplied by the rotation matrix. This Matrix Multiplication is realized using a set of Complex Multipliers in parallel accomplishing the second rotation and thereby updating the first column of the Cholesky factor. The reduced observation vector from the first stage is passed onto the second stage to update the next column of the Cholesky factor.

- $T \Rightarrow (x, y, 0) \rightarrow (r, 0, \theta)$
- $R \Rightarrow (x_1, y_1, \theta_1) \rightarrow (x_2, y_2, \theta_2)$
- $S \Rightarrow (1, 0, \theta) \rightarrow (cos\theta, sin\theta, 0)$

The second and the subsequent stages excluding the last stage are a replica of the first stage except for a reduction in number of vectoring mode operations and in the size of the Matrix Multiplier at each stage. This reduction is obvious because

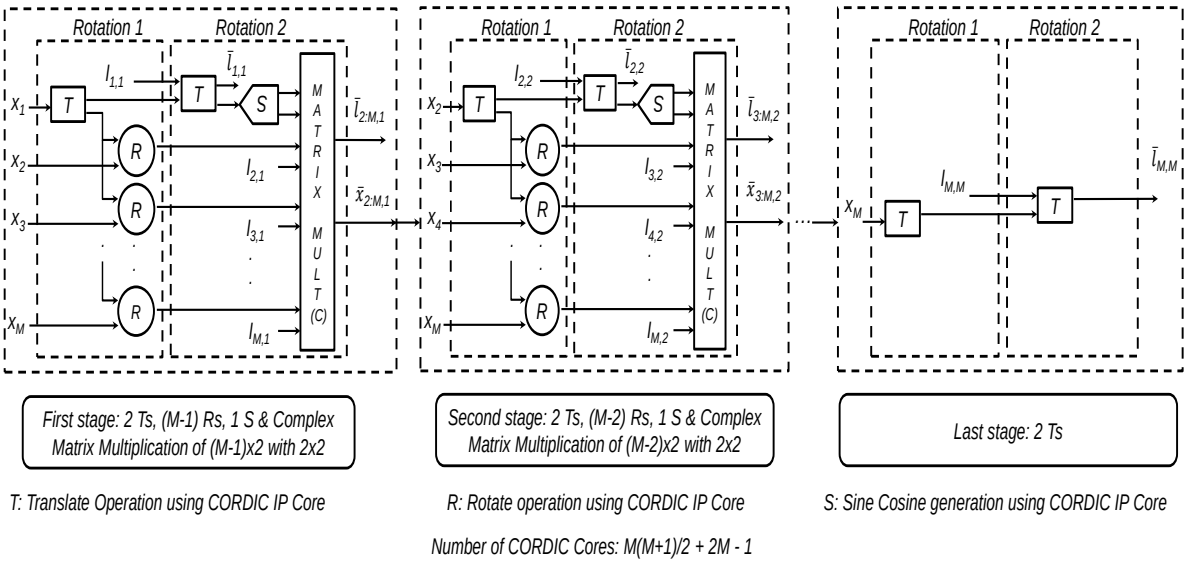


Figure 3.3: Architecture 1.

the size of the observation vector gets reduced by a factor of 1 at each stage.

The last stage basically comprises of two rotations involving real elements and this is accomplished using two translate or vectoring mode operations of CORDIC algorithm. At the end of the last stage, the bin vector is nullified and the appropriate Cholesky factor gets updated. In the next cycle the Cholesky factor of the next bin gets updated and so on.

3.3.1.2 Analysis

The bin vectors can be pumped in continuously one after the other as the hardware for the first rotation is free to handle the subsequent bin vectors while the hardware for the second rotation deals with the output of the first rotation of the previous bin vector. This helps in bringing down the latency by an order of the size of the observation vector compared to the architectures discussed in the previous section.

From the figure, it is clear that a total of $M + 2$ CORDIC IP cores are required to reduce the first element in the bin vector to zero. The next element in the bin vector can be reduced with $M + 1$ CORDIC IP cores ($M - 1$ for $Rotation1$) &

(2 $for\ Rotation2$). Similarly the third will require M CORDIC IP cores $(M-2)$ & (2) for Rotation1 and Rotation2 respectively and so on. Therefore this architecture requires a total of $M\left(M+1\right)/2+2M-1$ CORDIC IP cores.

3.3.2 Architecture 2

3.3.2.1 Concept

In the previous architecture, the first rotation in the every stage was accomplished using a set of vectoring mode operations in parallel designated as R. But as pointed out in the analysis of the previous section, these operations are realized with individual CORDIC IP cores in real hardware, which consume a great amount of resources. In fact the first rotation can be accomplished the same way as the second rotation by having a single Rotation mode operation to generate the elements of the rotation matrix followed by a Matrix Multiplier realized as a set of Real Multipliers in parallel rather than Complex Multipliers. The resulting architecture will be as shown in Figure.3.4.

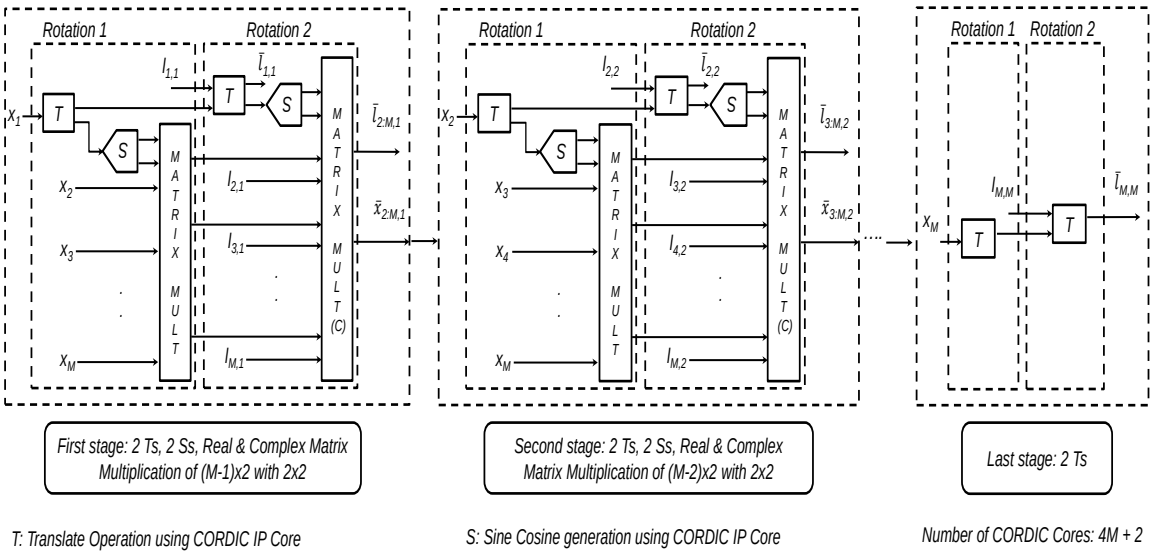


Figure 3.4: Architecture 2.

3.3.2.2 Analysis

From the figure, it is evident that the structure for the first rotation looks similar to the second rotation except for the Matrix Multiplier which is real. This architecture results in additional delay, though not as large as the pipeline delay, due to the addition of the Multipliers in the first stage and there is no significant reduction in the hardware as the CORDIC IP cores in the first rotation of the previous architecture have been replaced by Multipliers. However the number of CORDIC IP cores required for the update has come down to $4M + 2$.

3.3.3 Architecture 3

3.3.3.1 Concept

The similarity in the structures for the first and second rotation in architecture 2 can be exploited to bring down the resources further. Also the Complex Multiplier in the second rotation step can be reused with appropriate manipulation to perform the real Matrix Multiplication in the first rotation step. However, this imposes a constraint on the way the bin vectors and the Cholesky factor columns are accessed. They have to be accessed in a interleaved manner enabling the merging of the structures to accomplish both the rotations resulting in an architecture as shown in Figure. 3.5.

The architecture illustrates a single stage in detail wherein Multiplexers are inserted before the first translate step and the Matrix Multiplier to enable interleaving of the bin vectors and Cholesky factors at the input and the outputs of the first and second rotation respectively with appropriate shimming delays. A correction module is inserted in the architecture which comprises of an adder & a subtracter. This correction module aids in reusing the Complex Multipliers to perform the real Matrix Multiplication in the first rotation. A Systolic array comprising of

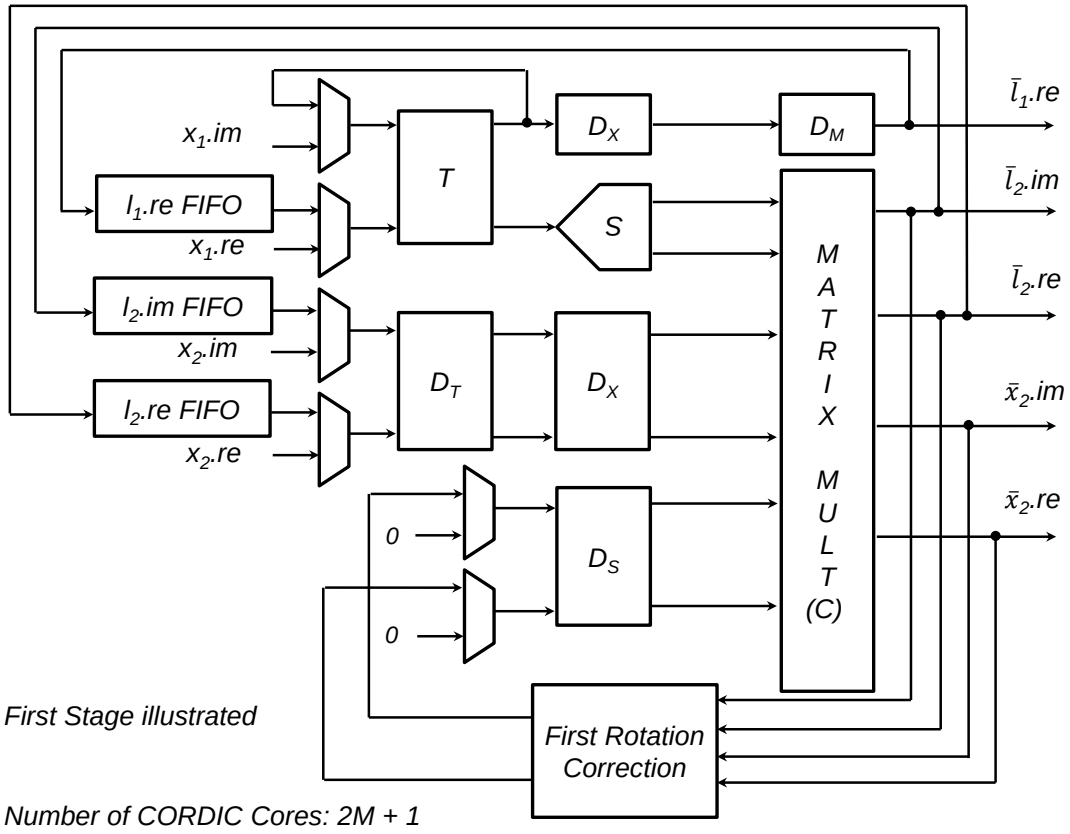


Figure 3.5: Architecture 3.

various processing elements combining the operations that suits each element is shown in Figure. 3.6.

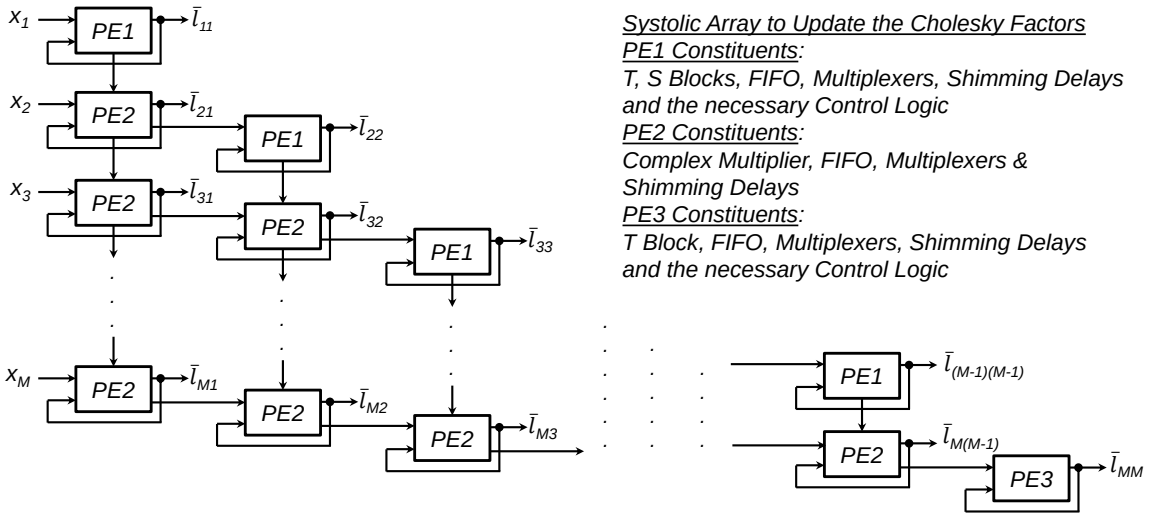


Figure 3.6: Systolic Array for Cholesky Factor Update.

Each Column in the array represents a stage. PE1 or the Processing Element 1 basically comprises of the T and S operations as discussed in Architecture 1 along with the control logic and FIFOs to hold the diagonal elements of the Cholesky factor. The shimming delays are also incorporated into the element. It generates the sine and cosine component required for the rotation matrix and passes it to PE2. All the PE2 receive the outputs from PE1 at the same instant though it appears to be sequential in the schematic. PE2 comprises of Complex Multipliers, Correction module, Multiplexers, FIFOs to hold the lower triangular elements of the Cholesky factor and the shimming delays. The outputs from PE2 in one column proceed to the next column. PE3 is for the last stage and has a single T operation along with FIFO and Multiplexers.

3.3.3.2 Analysis

The hardware corresponding to the first rotation in every stage has been completely removed. Though Multiplexers and a Correction module have been added to the architecture, they don't consume much resources as that of the Multipliers and CORDIC IP cores. Moreover the total number of CORDIC IP cores required has come down to $2M + 1$. Thus a significant reduction in hardware has been achieved. However the latency has gone up by a factor of 2 because of the presence of feedback paths in order to enable the same structure to perform both the rotations. The same latency as that of the previous architecture could be retained by operating the IP cores at double the clock rate. This could have implications on the upper bounds of various parameters of significance like number of bins, number of look angles, number of apertures and so on when compared to the previous architectures. Nevertheless the architecture will fit into an FPGA which may not be sufficient to hold the former architectures for a given set of parameters specified within the limits.

3.4 Architecture for Back-Solve Process

3.4.1 Concept

The architecture for Back-solve process is developed to facilitate the computation of power as soon as the updated Cholesky factors become available after the averaging period. The computation of adaptive weights can be avoided if power alone is required as the output as it would bring down the hardware requirements. In most cases power output is sufficient and requires a single Back-Solve step. The computation of adaptive weights requires another Back-Solve step in cascade with appropriate buffering of Cholesky factors. However a copy of the same architecture can cater for both the Back-Solve steps. The idea is to propagate the solution

obtained for each equation to the next to get its solution as early as possible. The solving begins with the division of the first element of the steering vector with the first diagonal element of the updated Cholesky factor. The solution is propagated down to all the rest of the equations as shown in Figure. 3.7.

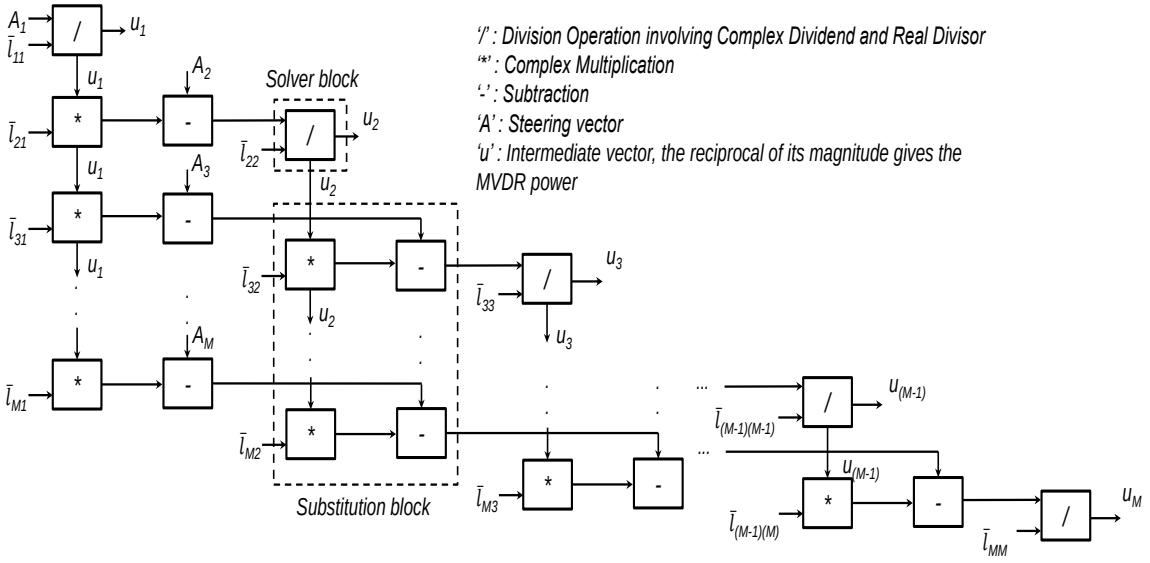


Figure 3.7: Systolic Array for Back-Solve process.

Similarly the solution of the next equation is propagated to the rest of the equations so that they get nearer to the solution faster.

3.4.2 Analysis

The architecture requires $2M$ Division operations and $M\left(M-1\right)/2$ Complex Multiplication operations to cater for the complex nature of Steering vectors and achieve parallelism. Though there is a significant increase in the resources compared to the existing architectures, there is a significant reduction in latency because there are no reverse paths involved. Moreover the Back-Solve process is a critical step in the computation of weights and can be done faster compared to the Cholesky factor update process because the latter has a lower time bound of one

FFT snap-shot time period. This can be considered as the minimum time interval to complete an update cycle. This is not a hard bound because the hardware could be made to run faster. But it would lead to a situation where the hardware starves for the next set of bin vectors. In the case of a Back-Solve process, there is no such bound. Sooner the weights get computed, sooner they become available to the bin vectors giving rise to better estimates.

CHAPTER 4

Implementation and Simulation Results

4.1 Tools used

Most of the control logic required for the architecture have been realized in verilog and verified using the Xilinx ISE version 14.4. The control logic are then converted to blocks using the Xilinx System Generator tool which in turn invokes the Matlab Simulink graphical environment. CORDIC algorithm, which is used to accomplish the Givens rotations, FFT algorithm and other major operations like Division have been implemented using the Xilinx IP cores which are available from the System Generator environment in the form of blocks. FIFOs and Dual port RAMs used in the architectures are also available as system generator blocks. wherever possible ready made Xilinx blocks have been made use of so as to exploit the resources and features available on the Xilinx FPGA platforms. These blocks are integrated in the Simulink environment to realize the final architecture. Matlab equivalent of the Hardware is also generated in tandem with the help of Matlab functions and Simulink blocks.

4.2 Implementation

The implementation is divided into two parts. The first part is the bin vector generation from the sensor samples and the second part is the MVDR Beamformer power computation module which takes bin vectors as its input. Most of the blocks are realized using the standard blocks and templates available in the System Generator environment.

4.2.1 Bin Vector Generation

The bin vector generation comprises of the following components.

4.2.1.1 Signal Generator

This is a matlab component and is not realized in hardware. It generates the signal as received at the input of the sensors with uncorrelated noise added to it. This component is essential in testing the design and tries to emulate the actual scenario with various SNR conditions and directional signals. Figure. 4.1 represents the Signal Generator block in the System Generator environment. It

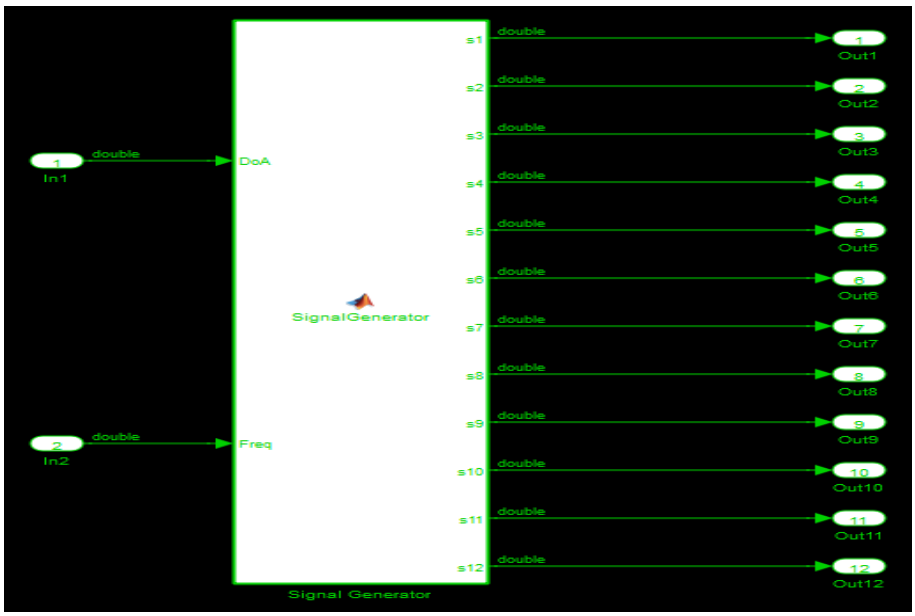


Figure 4.1: Signal Generator module.

has two inputs namely the DoA which can select one of the nine look angles and the tonal Frequency which lies in the band of interest. The SNR conditions are set with in the function.

4.2.1.2 Input Buffer

The role of this component is in rate conversion and is primarily implemented using the Dual Port RAMs. As an attempt to bring down the resources, a single FFT IP core is used to get the FFT output for all the sensors. Hence there is a need to multiplex the sensor time series data to the input of the FFT core. The write operations to the RAM happen at the sampling rate, while the read operations happen much faster and the rate depends on the number of look angles and bins in the band of interest. The read & writes happen in a ping pong fashion. The address generators associated with the RAMs are also included with this component. This component is composed of two blocks namely IN_WAG and IN_RAG as depicted in Figure. 4.2.

IN_WAG comprises of a Write Address Generator for the write page of the

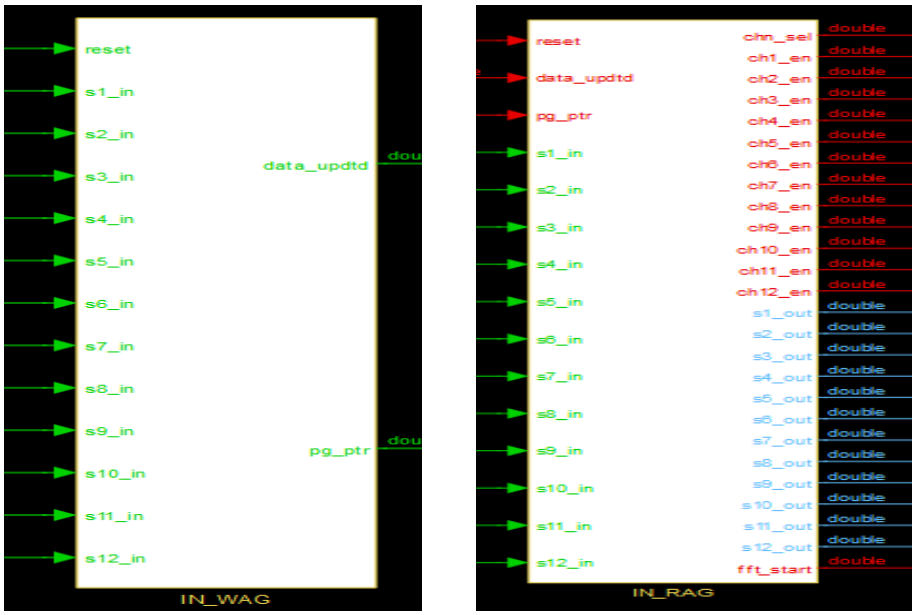


Figure 4.2: Input Buffer module.

Dual Port RAM. It also generates a page pointer (*pg_ptr*) and a data updated (*data_updtd*) signal. These signals are used by IN_RAG and assist in accessing the RAM in a ping-pong fashion. IN_RAG comprises of a Read Address Generator for the read page of the Dual Port RAM. It also generates the channel enable signals to multiplex the sensors along with a fft_start signal to start the FFT

operation on the sensor inputs. Each sensor is provided with a read and write page in IN_RAG and IN_WAG respectively. The Address Generators along with control signals were realized using the System Genrator Black Box template. The Dual Port RAMs are realized using the Xilinx System Generator standard DPRAM blocks.

4.2.1.3 FFT Core

The FFT Core component comprises of a Multiplexer to time multiplex the sensor inputs, an FFT IP core and additional logic to generate the control signals for the subsequent components. The FFT IP core used is a Xilinx System Generator IP core version 7.1 [13]. It is configured to perform a 256 point FFT in pipelined streaming IO mode. The twiddle factor width is chosen as 24. The input precision is in Fix_16_15 format. In order to retain the same precision at the output of the core, a suitable scaling factor is provided to the scaling input of the IP core. Truncation is used as the rounding mode. The done signal from the FFT core is used to generate the control signals like *done4wag*, *done4rag* and *bme* (*bin memory enable*) for the Bin vector memory. Figure. 4.3 shows the FFT IP core block.

Control signals like *done4wag* and *bme* are used to generate the write address to store the bins obtained from the FFT core component in the respective sensor Bin memories.

4.2.1.4 Bin Storage

The Bin Storage component comprises of two blocks namely the BV_WAG and BV_RAG as depicted in Figure. 4.4.

BV_WAG block houses the write pages of the Dual Port RAMs used to store the FFT bins from all the sensors and a common Write Address Generator. The *bme* from the fft core component enables in choosing the appropriate Bin Memory for

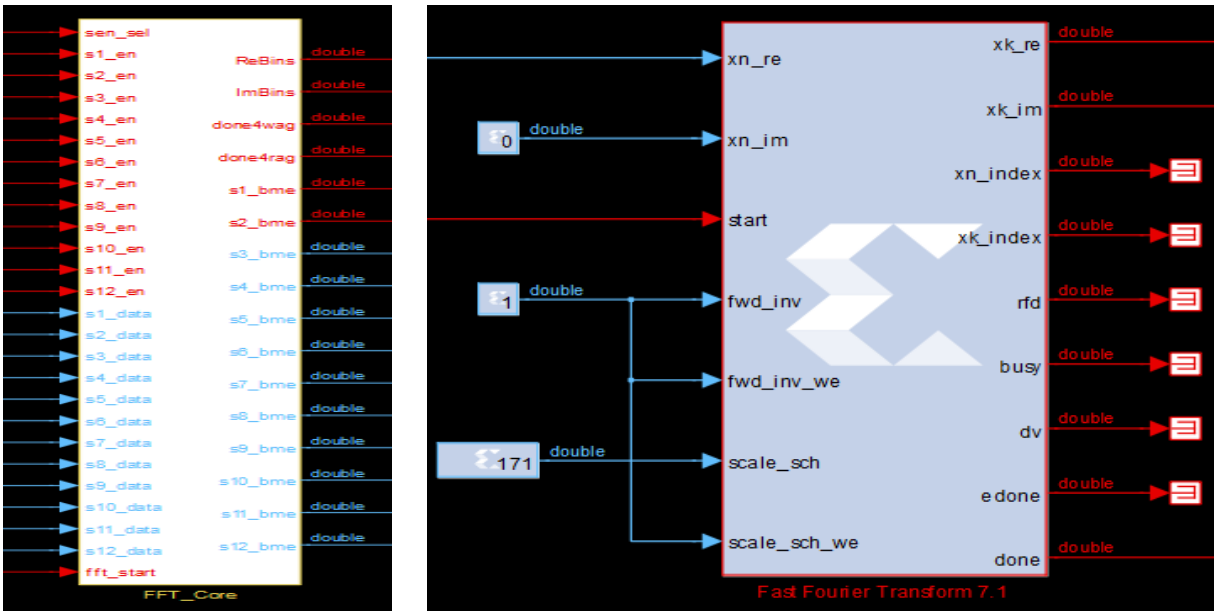


Figure 4.3: FFT Core module with IP Core.

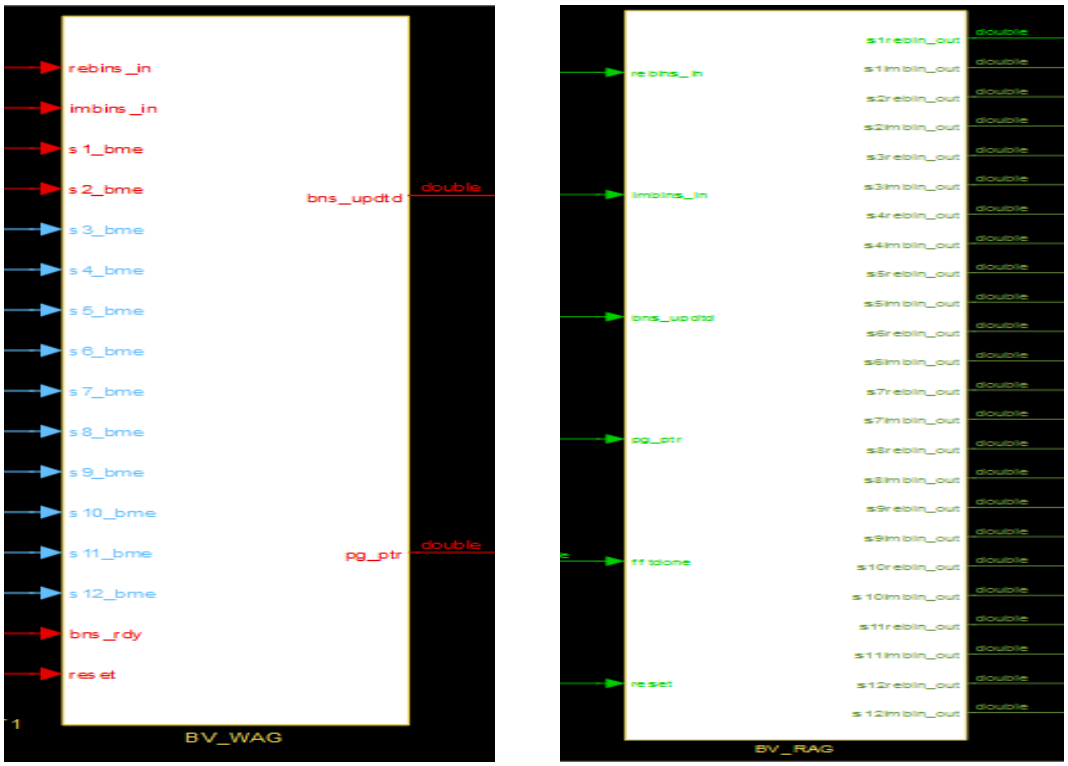


Figure 4.4: Bin Storage.

each sensor. It also generates two control signals for BV_RAG namely the page pointer (*pg_ptr*) and bins updated status (*data_updtd*), which assist in ping-pong access of the Dual Port RAMs. The BV_RAG block houses the read pages of the Dual Port RAMs and a common Read Address Generator to read out the bin vectors from the appropriate Bin Memory to Cholesky Factor Update block in the MVDR Beamformer power computation module. The Address Generators along with control signals were realized using the System Generator Black Box template. The Dual Port RAMs are realized using the Xilinx System Generator blocks.

4.2.2 MVDR Beamformer Power Computation module

MVDR Beamformer Power Computation module comprises of two major components as shown in Figure. 4.5. The Systolic Arrays for Cholesky Factor Update and Back-Solve Process. The Figure 4.5 depicts the actual implementation in the Xilinx System Generator environment.

4.2.2.1 Systolic Array for Cholesky Factor Update

This array has 12 stages to cater for the case of a 12 sensor sub-aperture and is expandable by adding hardware columns comprising of a $PE1$ and appropriate number of $PE2$s to meet the needs of a larger sub aperture. $PE1$ as dicussed in Chapter 3, has three control logic blocks which govern the flow of data and the operations performed on it. They are the $lfifologic$, *lxlogic* and the *philogic* as shown in Figure.4.6. The control logic blocks are realized using the System Generator Black Box template. $lfifologic$ provides the reset to the other control logic blocks and generates a write enable ($fifo\_we$) and select (*ival_sel*) to enable the writes to Cholelsky factor FIFO and switch between its initial value and updated values respectively. The *lxlogic* in the first stage doesn't have (*ps_ready*) signal, which is essential from the second stage onwards to know the status of the previous stage. The ($fifo\_rd$) signal from *lxlogic* enables in reading out the Cholesky factor

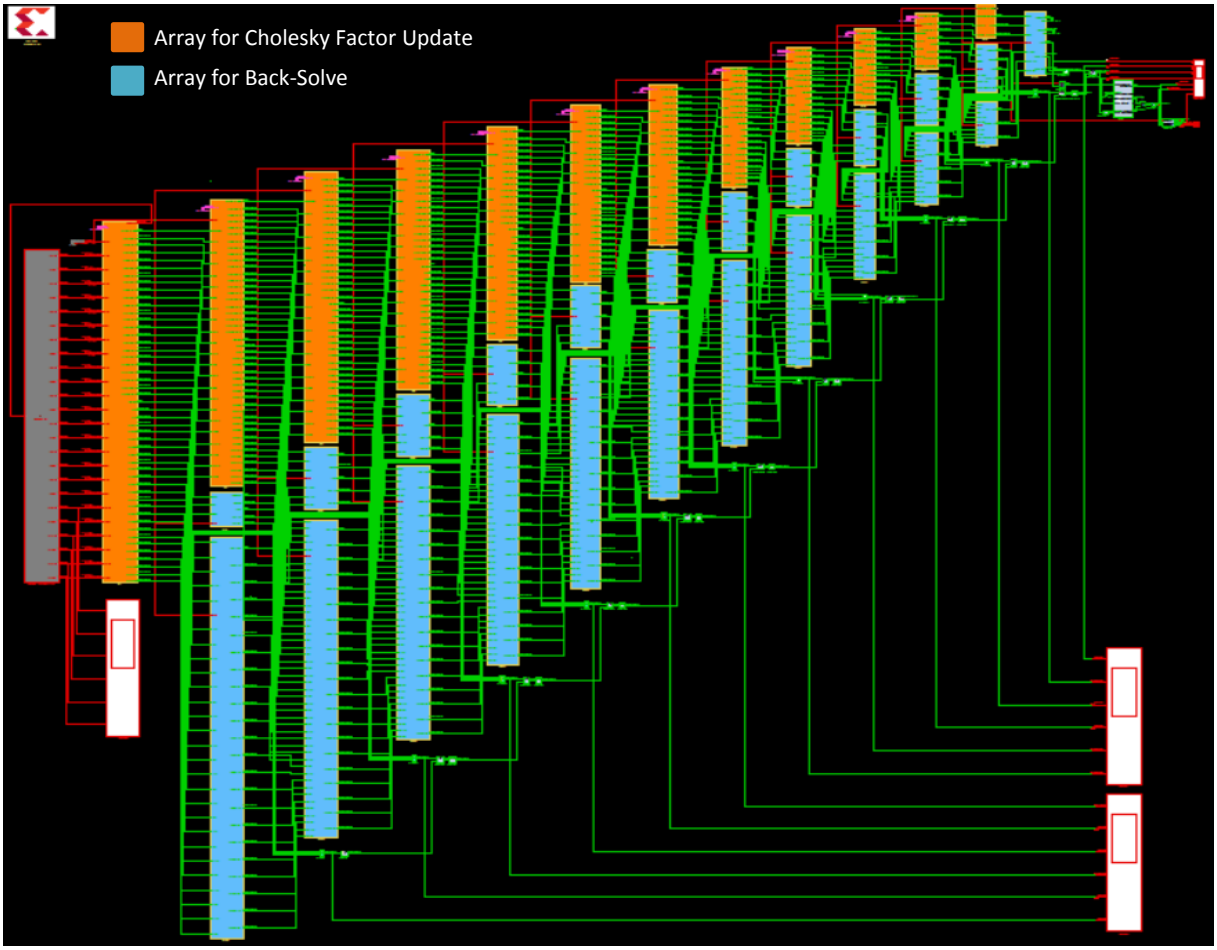


Figure 4.5: MVDR Beamformer Power Computation Module.

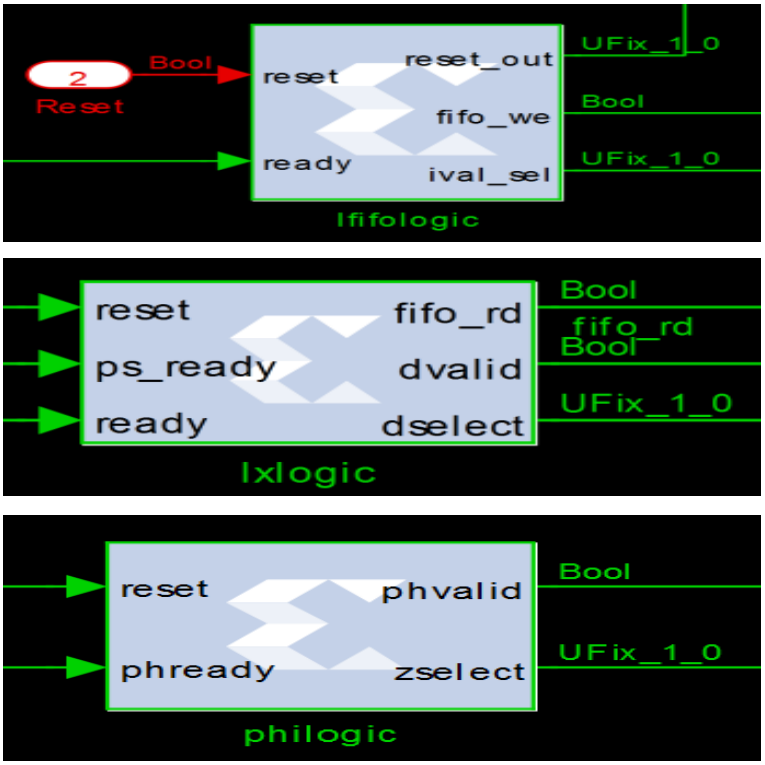


Figure 4.6: Control logic for PE1.

FIFO contents. The (*dvalid*) signal goes as the data valid input to the CORDIC Translate IP core and (*dselect*) enables in interleaving the Cholesky factors and the bin vectors. *philogic* block generates the phase valid signal (*phvalid*) to the Sine Cosine CORDIC IP core and a select signal (*zselect*) to switch between zero data and first rotation outputs to the Complex Multipliers in PE2.

The two major blocks in PE1 are the Translate and the Sine Cosine CORDIC IP cores [14] which together perform the first and second rotations on the diagonal elements of the Cholesky factors. The Translate IP core has complex inputs and a data valid signal to validate the inputs, while the Sine Cosine IP core has a phase input and a phase valid signal to validate the phase input as shown in Figure. 4.7. Both the cores are configured to operate in the optimal pipeline mode and word serial fashion to compromise between speed and resources. The IO width is 16 bit and rounding mode is chosen as truncation, though 8 bit and 12 bit precisions were tested and evaluated. The phase output from the Translate core goes directly to the Sine Cosine core while the output valid signal feeds the *philogic* block. The

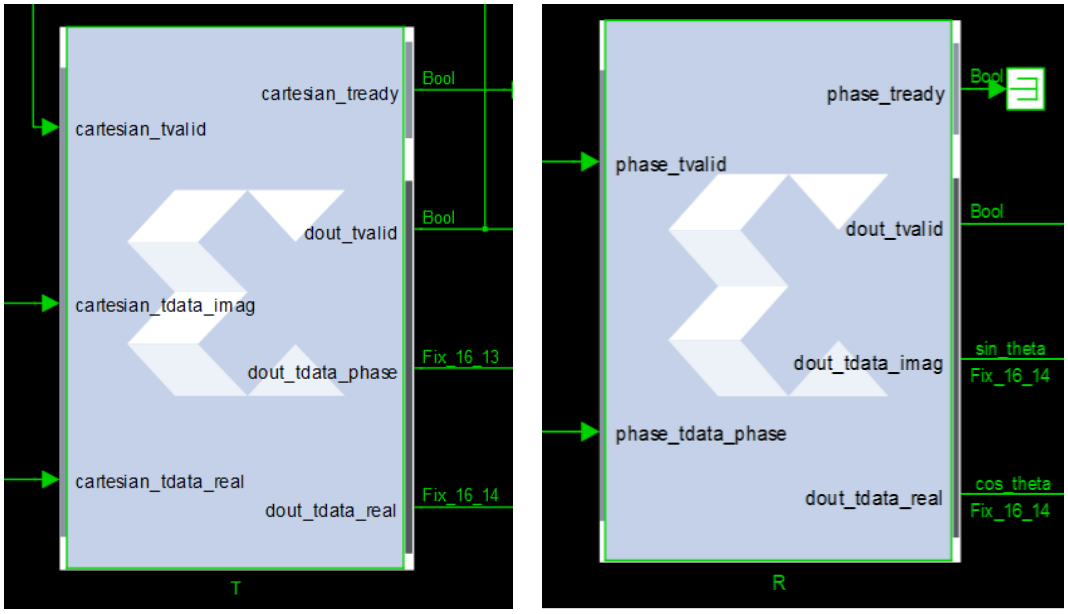


Figure 4.7: CORDIC IP Core in Translate (T) and Sine Cosine Generation (R) mode.

real output from the Translate core is subjected to a sign correction logic as the core produces only the absolute value at its output. The corrected output is fed back to the core again through the Multiplexer. The modules involved in correction logic at the input and the output of the core are depicted in Figure. 4.8.

SignCorr block gets the quadrant information of the input element by computing its sign and feeds the *OutCorr* block to apply the quadrant correction. The *xlmcmp* and *xlsigncorrect* blocks in *SignCorr* and *OutCorr* components are realized as System Generator MCode blocks.

PE1 PE2 and PE3 include the following common blocks namely the $lfifo$ and *lxsel* as shown in Figure. 4.9. $lfifo$ block houses the FIFOs to hold the Cholesky factors and the Multiplexers to switch between the initial values and updated values as seen expanded in Figure. 4.10. Whereas the *lxsel* houses the data interleaving logic comprising of Multiplexers and Multipliers realizing the appropriate regularization of Cholesky factors and the Bin vectors as shown in 4.11. The regularization is accomplished by a scaling operation with a Multiplier. The regularization of Bin vectors is absent in the second and the subsequent stages. PE2 has its own blocks like shimming delays, First Rotation Correction and the Matrix

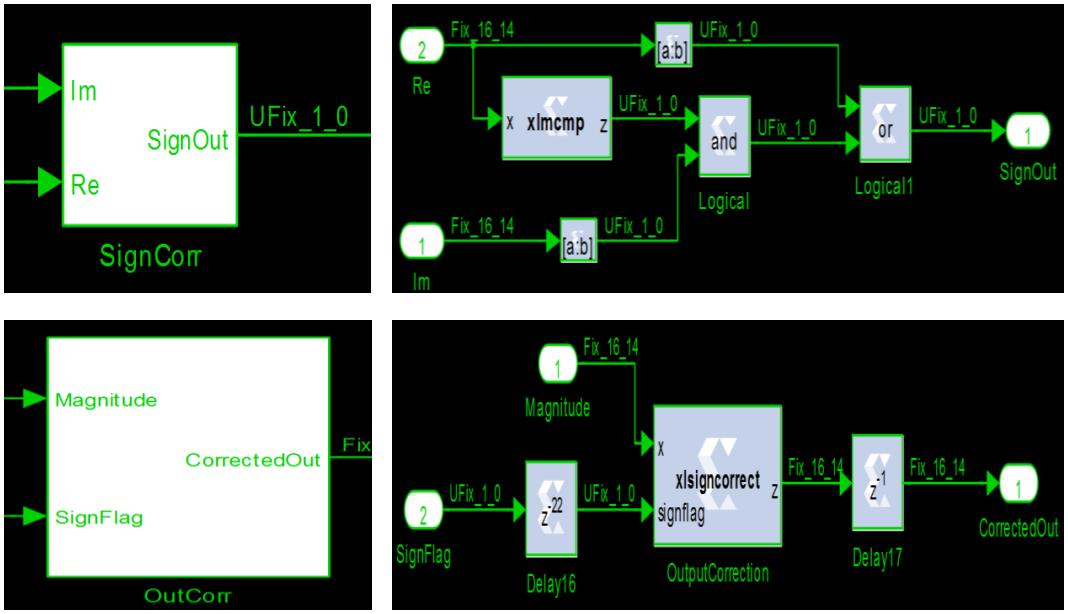


Figure 4.8: Correction logic for the Translate core.

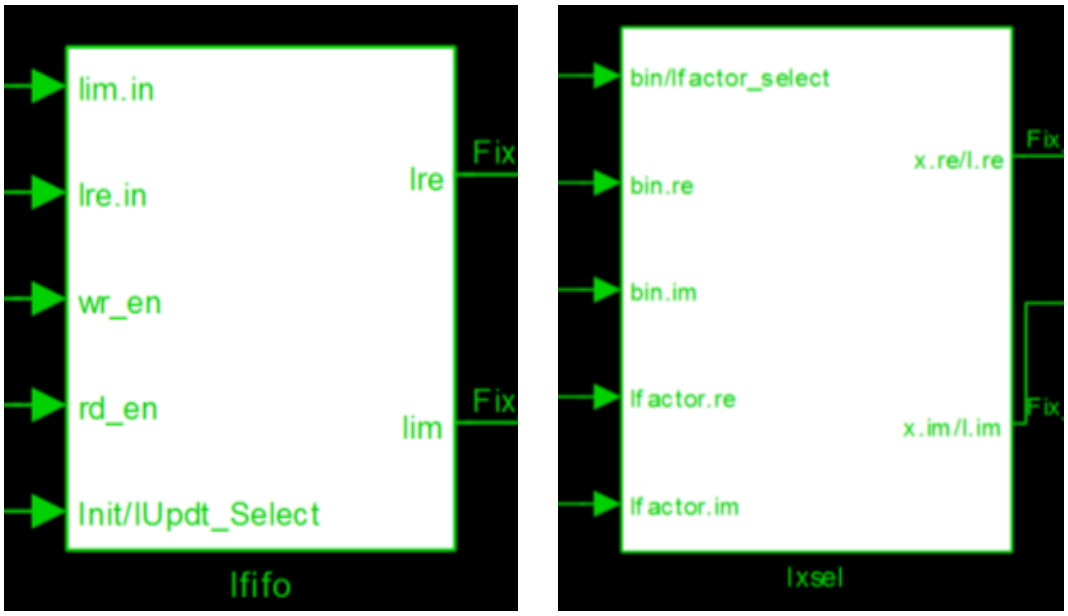


Figure 4.9: Common blocks of PE1 PE2 and PE3.

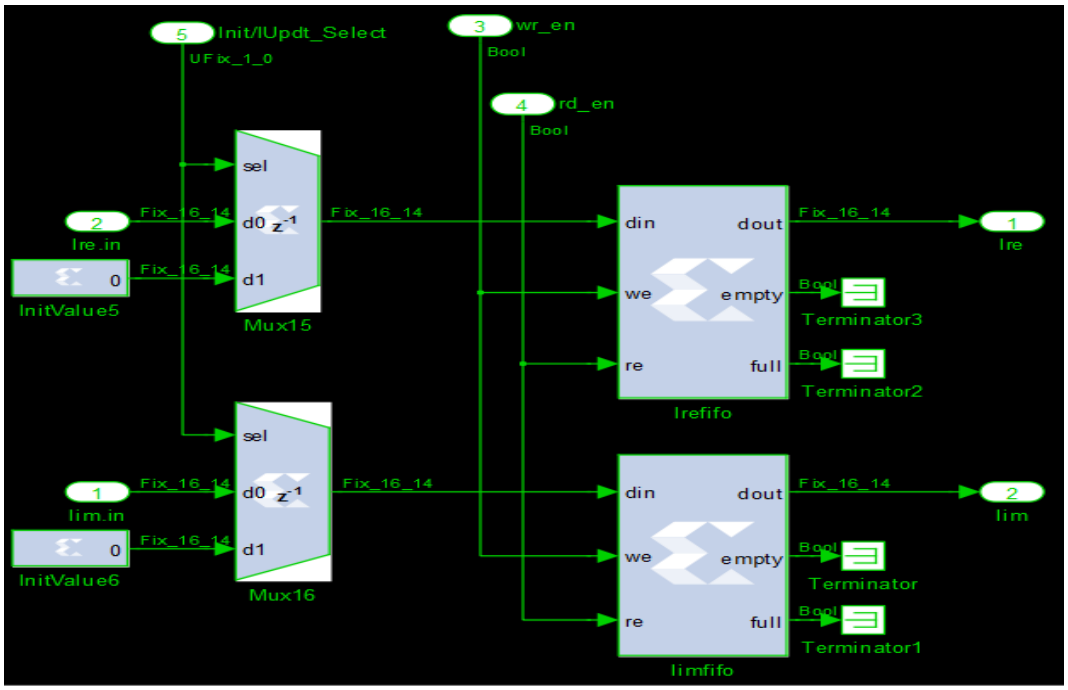


Figure 4.10: lfifo block expanded.

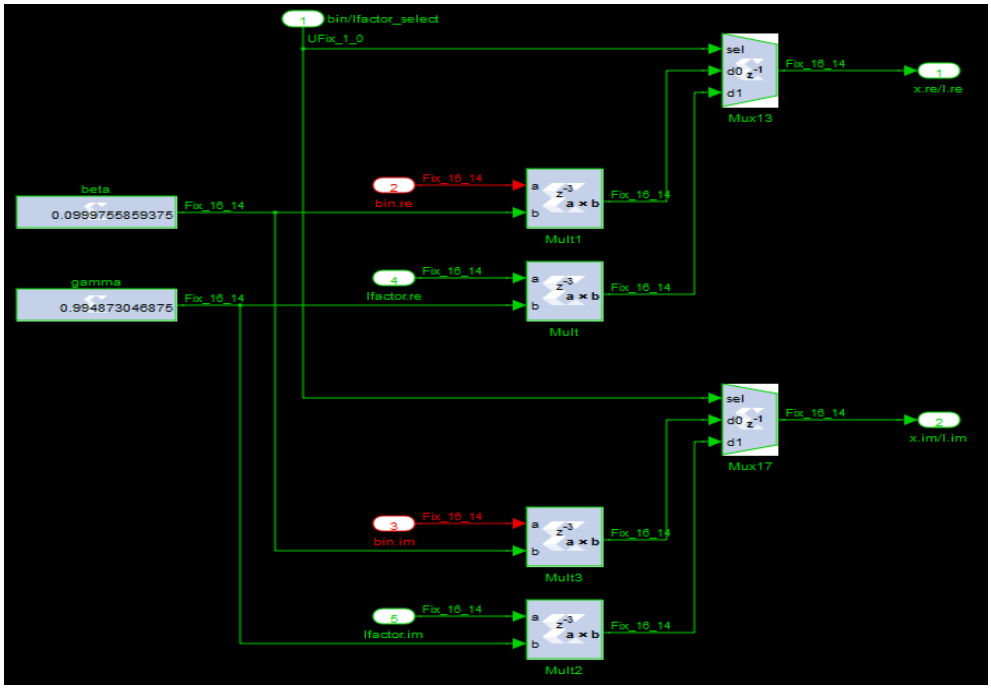


Figure 4.11: lxsel block expanded.

Multiplier blocks. Shimming Delays namely TD and RD as shown in Figure. 4.12 are to compensate for the pipeline delays encountered in the Translate and Sine Cosine CORDIC IP cores respectively. First Rotation Correction block gathers

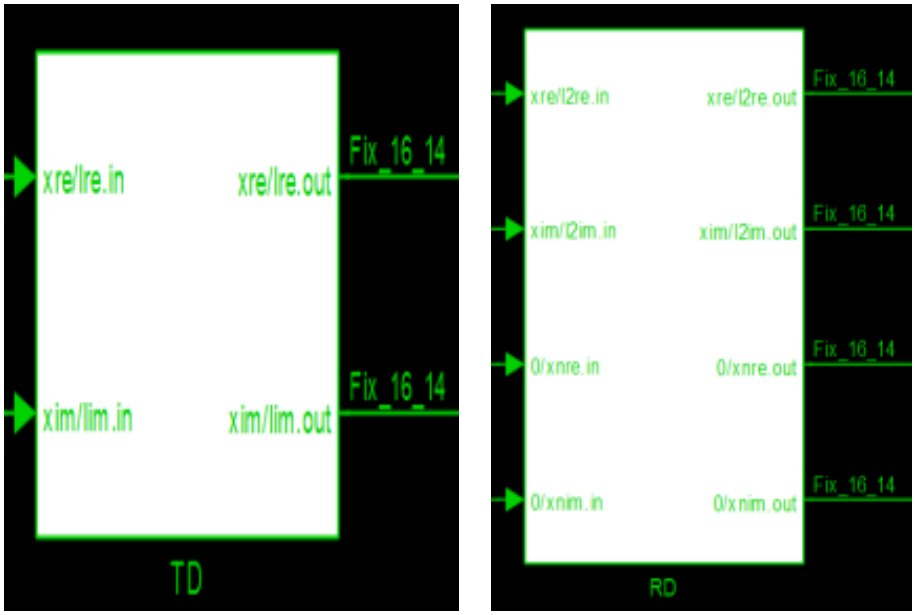


Figure 4.12: Shimming Delays.

the outputs from the Matrix Multiplier after the first rotation and computes the actual output with the help of an adder and subtracter as shown in Figure. 4.13. It also has Multiplexers which decide the inputs to the Matrix Multiplier during the first and second rotation. The Matrix Multiplier block has 8 real Multipliers in parallel followed by 4 real Adder/Subtracter circuits in parallel to perform a 2×2 Complex Matrix Multiplication. The Sine Cosine CORDIC core from PE1 provides the *cos_theta sin_theta* data along with the enable signal (*enab*) to perform the Matrix Multiplication. The outputs from the Matrix Multiplication block go to First Rotation Correction block ($uzsel\_frcorr$) as well as the Cholesky factor FIFO block ($lfifo$). Figure. 4.14 shows the Matrix Multiplier block as realized in the System Generator environment. The Processing Element PE3 has all the blocks of PE1 except for the control logic (*philogic*) and Sine Cosine CORDIC IP core (R) blocks.

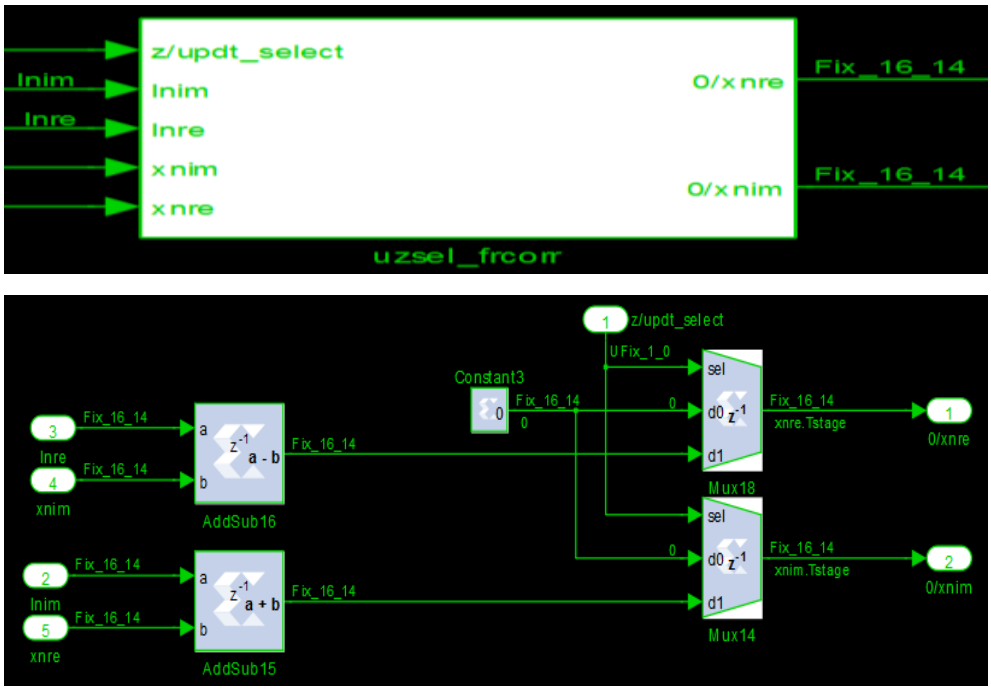


Figure 4.13: First Rotation Correction.

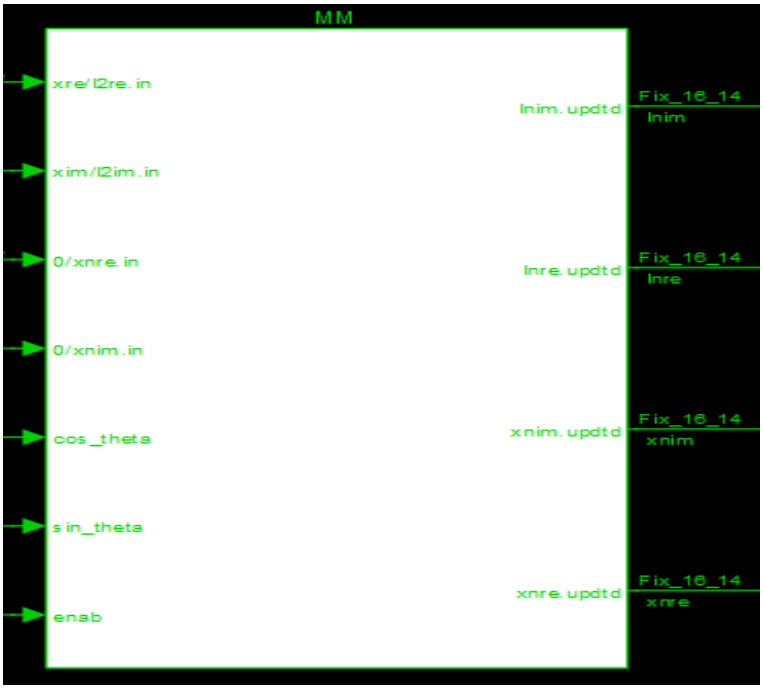


Figure 4.14: Matrix Multiplication.

4.2.2.2 Systolic Array for Back-Solve Process

The Systolic Array for Back-Solve Process has 12 stages with each stage having two major blocks except the last stage. The major blocks present in all the eleven stages are a *Solver* block and a *Substitution* block. The last stage has only the *Solver* block. This block computes the elements of the intermediate vector one after the other at each stage as depicted in the Figure. 3.7 in Chapter 3. The elements obtained from each *Solver* block are used in the *Substitution* block of the same stage to propagate the solution to the subsequent stages.

The *Solver* block comprises of two Division functions, an intermediate FIFO to hold the diagonal real elements of the updated Cholesky factor and the necessary control logic to govern the operations and flow of data. Two Division blocks are required to handle the division of complex steering vector in parallel. The Division function is realized using the Xilinx Divider Generator block version 4.0 [15] with additional logic to combine the integer and fractional quotient as shown in Figure. 4.15. The Divider is configured to use the radix-2 algorithm with a fractional width of 8 which decides the latency automatically. The integer width is chosen to be 16 to cater for a wider range of the inputs. The FIFO is similar to the one in the Cholesky factor update array. The *Solver* and *Substitution* blocks in the first stage also houses the Steering vector ROMs in addition to the above resources. The control signals used in the *Solver* block are derived from some of the output signals from the Cholesky factor update array like the data output valid from the Sine Cosine CORDIC IP core ($dovalid$), write enable ($wren$) and the data select ($ivsel$) signals generated from the $lfifologic$. The control logic block generates the read and write enable signals for the intermediate FIFO ($rdfifo$ & $owren$) as well as a select signal ($oivsel$) to update the intermediate FIFO with either the new Cholesky factor update or the old factor. In addition to these signals the control logic block in the first stage also generates the address for the steering vector ROMs as shown in the Figure. 4.16. The *Substitution* block comprises of a

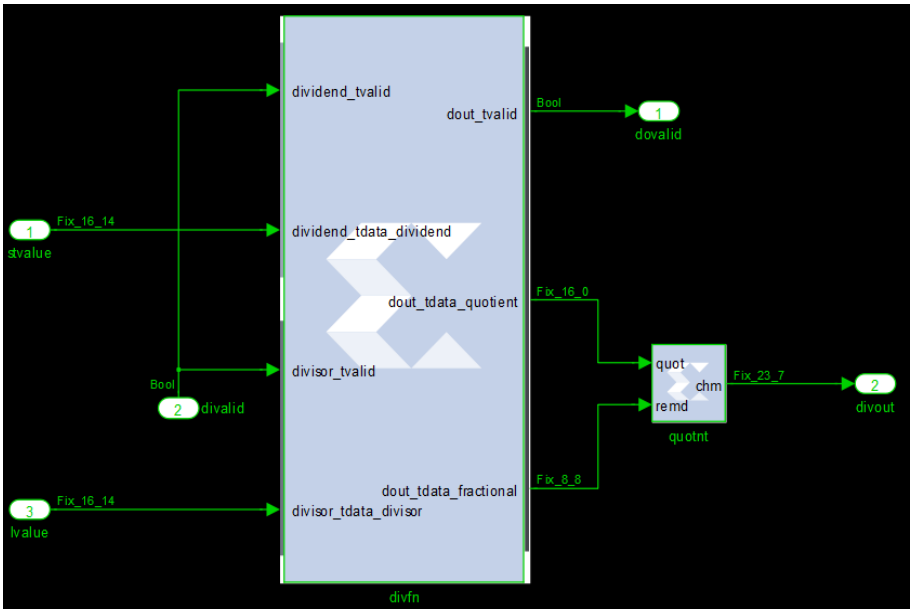


Figure 4.15: Division Function.

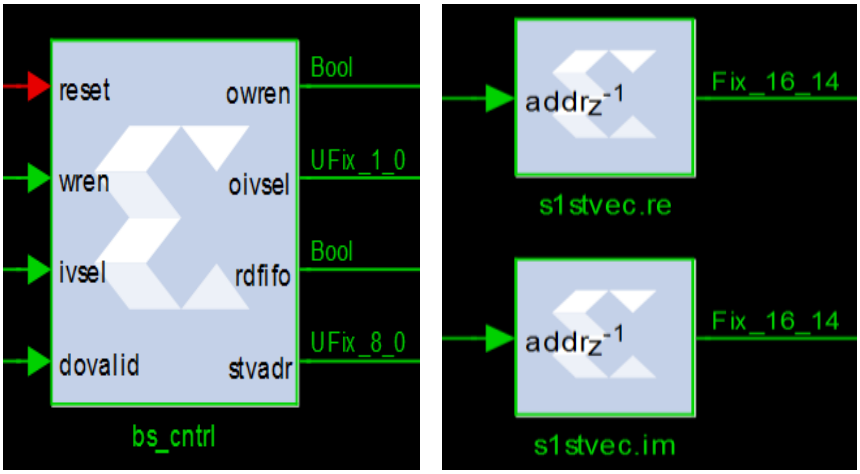


Figure 4.16: Control Logic for Solver Block & Steering vector ROMs.

set of complex Multipliers and Subtracters with additional control logic besides the intermediate FIFOs to hold the updated lower triangular off-diagonal elements of the Cholesky factors. The number of Multipliers and Subtracters come down by 1 at each stage and hence the last stage doesn't have any. Besides the intermediate vector elements from the *Solver* block, control signals like the read and write enable signals for the intermediate FIFO ($rdfifo$ & $owren$) as well as the select signal ($oivsel$) are routed to this block to handle read and write operations of FIFOs. The intermediate vector element is routed to all the Complex Multipliers in parallel where it gets multiplied by the Cholesky factor elements of that stage and passed on to the set of Subtracters in parallel where it gets subtracted from the outputs obtained from a similar operation in the previous stage. In the first stage, the Steering vectors form the previous stage output. The Multipliers are realized using the System Generator Complex Multiplier block as shown in Figure. 4.17. The control logic generates the address for the Steering vector ROMs ($stvadr$)and enable signal for Subtracters ($suben$) housed in this block. It also generates a ready signal for the next stage ($nxtrdy$) to begin its process. The control logic in the *Substitution* block of subsequent stages generate only the ready signal for the next stage. Other signals are absent. The output data valid signal from the Complex Multipliers drives the enable signal of the Subtracters in the subsequent stages. The control logic for both the blocks have been realized using the System Generator Black Box template.

4.2.2.3 Power computation

As and when the elements of the intermediate vector becomes ready they are fed to a magnitude generation block at each stage and gets added up to the output of magnitude generation block in the previous stage resulting in the magnitude of the intermediate vector as a whole at the last stage. The magnitude function blocks are realized using Xilinx standard Multiplier and Adder blocks with appropriate

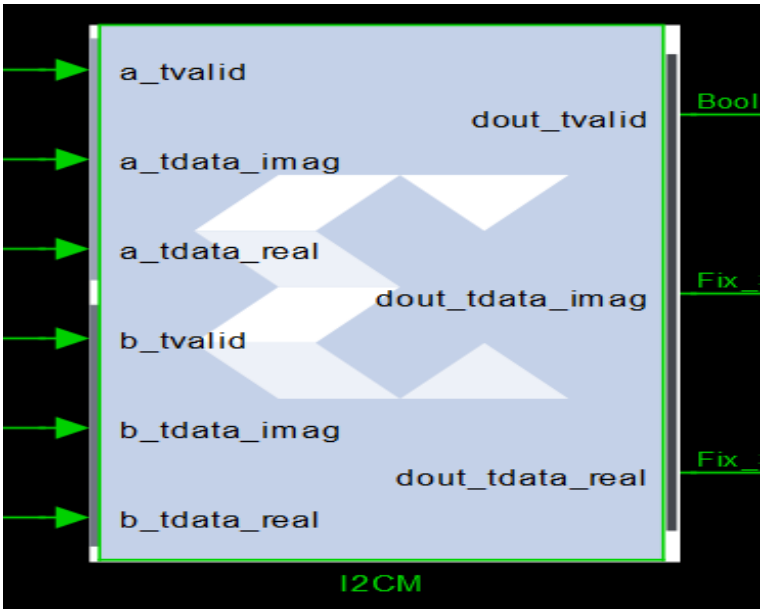


Figure 4.17: Complex Multiplier.

scaling as shown in Figure. 4.18. The magnitude of the intermediate vector is fed

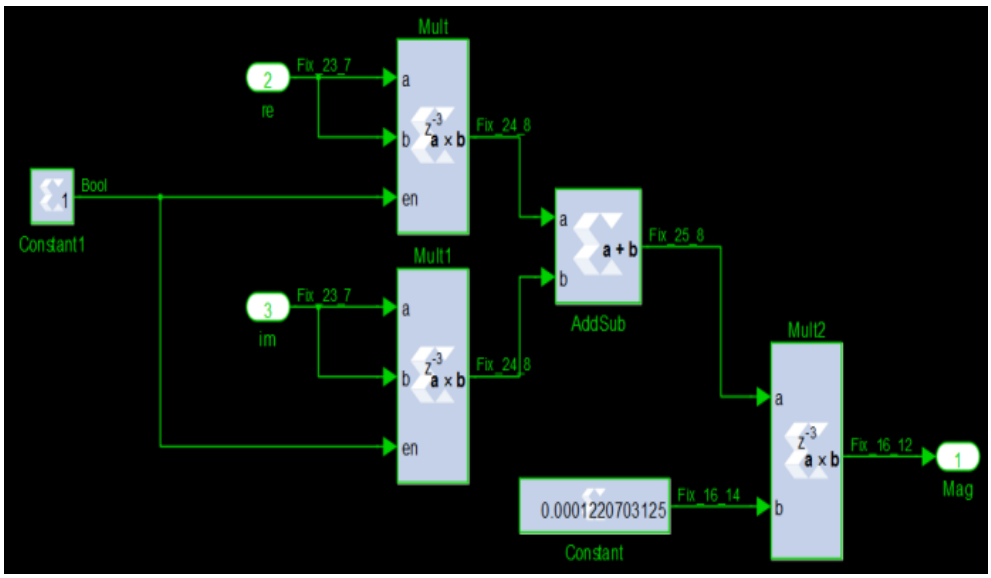


Figure 4.18: Magnitude Function.

to a Division block similar to the one discussed in the Systolic Array for Back-Solve process subsection. The Division block generates the reciprocal of the magnitude which is the estimate of power associated with a bin. It is fed to an accumulator which is realized as a system generator black box as shown in Figure. 4.19 to get the power estimate in a beam. The hardware for the weights computation has

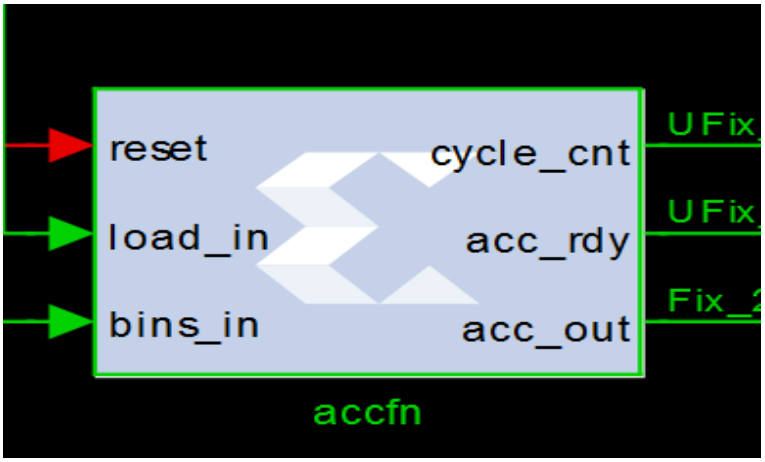


Figure 4.19: Accumulator Function.

not been dealt here because it requires only an additional Back-Solve Array in cascade with the existing array with additional routing of the Cholesky factors to the appropriate nodes in the array.

4.3 Simulation Results

The critical component of the MVDR wide-band beamformer namely Power Computation module was synthesized on a Xilinx Virtex6 FPGA platform. The target device was chosen as xc6vlx240t-3ff784, which has about 300K logic cells, Configurable logic blocks in the from of 37680 slices and approximately 4000Kbits of distributed RAM and DSP48E1 slices about 768 in number. Moreover the device has enough memory in the form of block RAM blocks with 416 36Kbit blocks. The device can handle user IOs upto 720 in number.

4.3.1 Resource Utilization Summary

The operational modes of CORDIC IP core used to realize the system namely the Translate and Sine Cosine Generation modes consume the least amount of resources among other modes when configured in word serial architecture and

optimal pipeline mode.

The system has about 312 FIFOs (*FIFOs in both the Systolic Arrays*) of depth 32 words and 24 ROMs (*Steering vector ROMS*) of depth 256 words which are of size 16 bits leading to a total memory size of about 258 Kbits. The number of IOs in the system add up to 600 including outputs which have been drawn out of the system for debugging. Idealy the Device chosen has more resources than what is required in some fronts. The actual estimate regarding the resource utilization by other hardware blocks in the system after synthesis also shows the same as is evident from the Device Utilization summary in Table. 4.1.

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	47668	301440	15%
Number of Slice LUTs	44260	150720	29%
Number of fully used LUT-FF pairs	38457	53471	71%
Number of Block RAM/FIFO	123	416	29%

Table 4.1: Device Utilization Summary

4.3.2 Timing Summary

The timing summary after synthesis of the design is as shown in the Table. 4.2.

Minimum period	3.156ns ($MaximumFrequency : 316.827MHz$)
Minimum input arrival time before clock	1.418ns
Maximum output required time after clock	0.378ns
Maximum combinational path delay	No path found

Table 4.2: Timing Summary

The latency associated with the first and second rotation in the Cholesky factor update array is 28 cycles each for 16 bit precision. So the rate at which an update becomes available at the output for bin after bin is 84 cycles. From the timing summary table, going by the maximum frequency possible, each update would take $\approx$ 252ns($84 \times 3.15ns$). Therefore the time taken to update 28 bins would be $\approx 7\mu$s. Whereas the FFT snapshot interval as per the specification in the case study is 10.67ms which is too high compared to the time taken to update all the

bins. The architecture is so designed that it can handle multiple apertures just by increasing the FIFO size retaining the other modules. Moreover it is quite evident from the Resource Utilization summary that there is enough memory resources left in the form of block RAMs/FIFOs on the device after the design was synthesized for a single Aperture. So the requirement of handling 32 apertures can be easily met. An increase in Aperture size will increase the size of the array, which in turn could have implications on the maximum frequency due to allocation and placement of additional resources. This in turn can bring down the number of Apertures that can be handled. Figures. 4.20 and 4.21 provide an idea of the rate at which updates are performed.

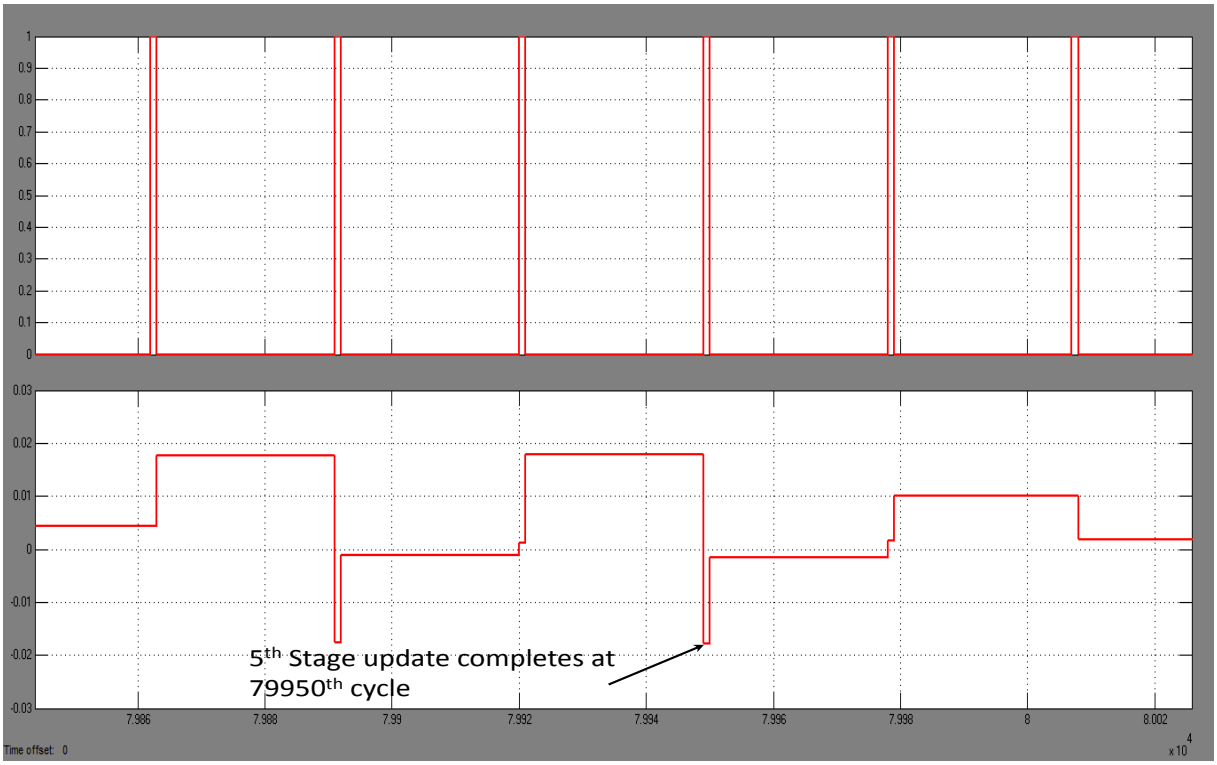


Figure 4.20: Stage 5 Output of the Cholesky factor update array.

In the Back-Solve array, the Division function is pipelined and the complex Multiplication and Subtraction that follow together contribute to a latency of 40 cycles. Each element of the intermediate vector used in computing the power associated with a bin is generated at this rate. As per the architecture, as and when the el-

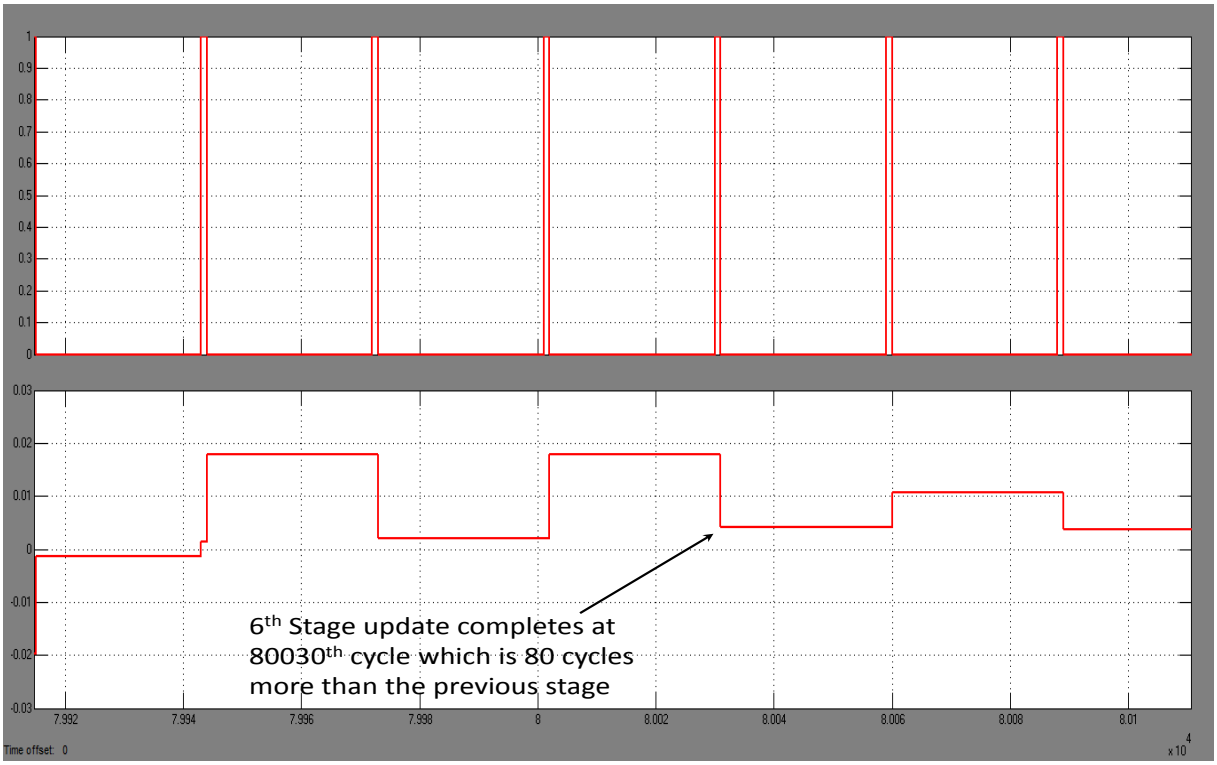


Figure 4.21: Stage 6 Output of the Cholesky factor update array.

ements of the intermediate vector become available their magnitude is calculated and passed to the next stage. Therefore in every 40 cycles the magnitude of the intermediate vectors associated with each bin becomes available to the reciprocal circuit. The reciprocal circuit which is again a Divider also has a latency of 40 cycles. So the power associated with each bin in a given look angle will be available at every 40 cycles. The time taken to compute the total power will be then $\approx 3.5\mu$s$(28 \times 40 \times 3.15ns)$. Though there is an initial delay of twelve stages, the output power of every look angle gets computed at the above mentioned rate. The total time taken to compute the power associated with all look angles in a given aperture will be $\approx 32\mu$s(9×3.5) which is very small compared to the averaging period of 512ms. Figure. 4.22 gives an idea of the number of cycles consumed to compute the power. An equal amount of time will be sufficient to calculate the weights as well thus achieving the target of reducing the time taken to compute the adaptive weights.

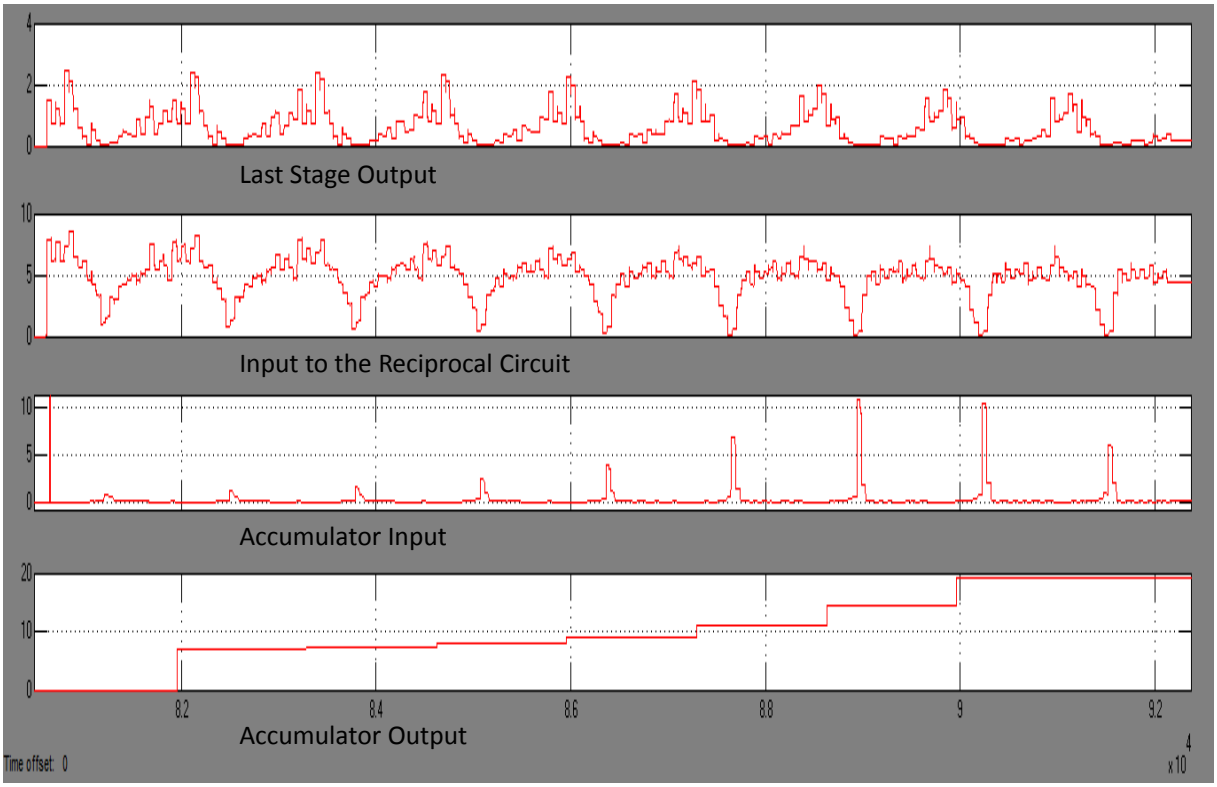


Figure 4.22: Accumulator Output.

4.3.3 Evaluation for various Precision

The Choleky factor update was evaluated for various precisions before synthesis and the mean error and mean square error are tabulated as in Table. 4.3

Precision	Format	Mean Error	Mean Square Error
8 bit	8_6	0.007	8.2e-05
12 bit	12_10	5e-04	3.4e-07
16 bit	16_14	3e-05	1.4e-09

Table 4.3: Performance of Design with various precisions

4.3.4 Simulation Waveforms

The system was simulated with two cases. The first case was a 2KHz tonal injected along the 4th Look Angle or -1.667 degrees away from the MRA of the Aperture at an SNR of -8db. The output as obtained from the Accumulator block of the system was plotted in Matlab as is shown in Figure. 4.23. The second case was

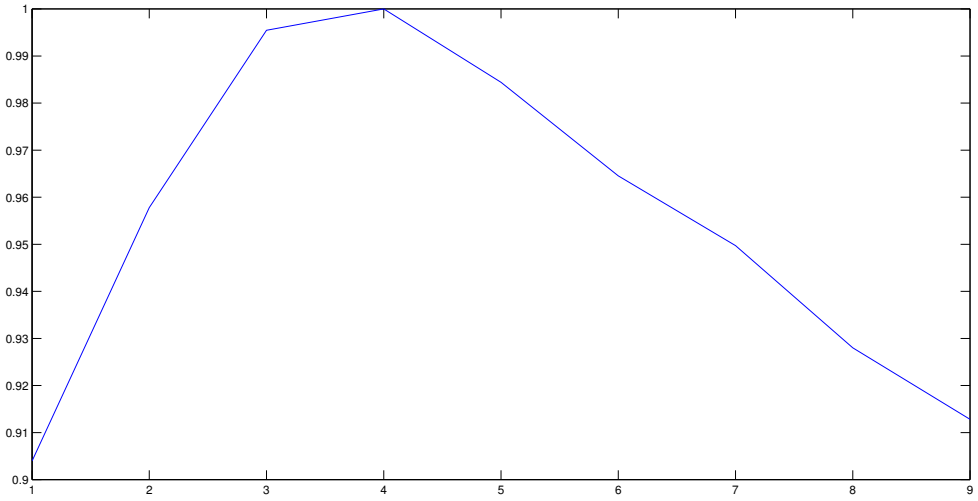


Figure 4.23: System Output: Case 1

a 1KHz tonal injected along the MRA or the 5th Look Angle at an SNR of -8db. The Output waveform is shown in Figure. 4.24.

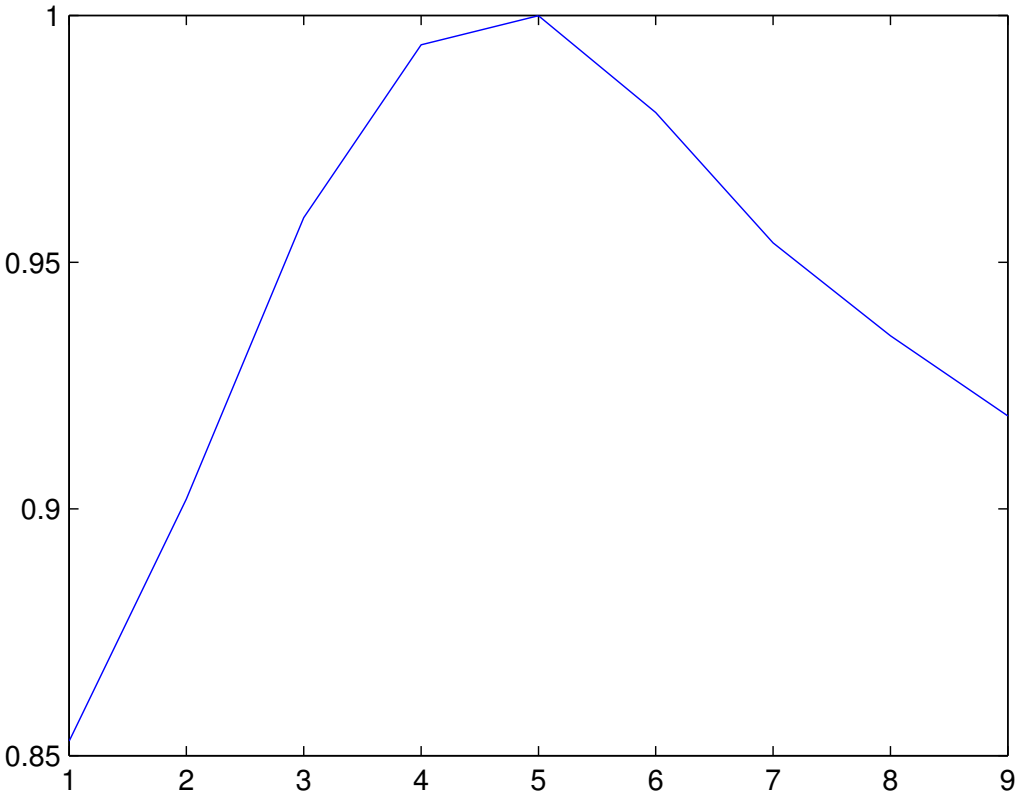


Figure 4.24: System Output: Case 2

4.3.5 Summary & Scope for Future

A Systolic Array based approach with reduced latency and not compromising much on the resources, was adopted to realize the MVDR wide-band Beamformer and was implemented using the Xilinx System Generator tool. Though the design has been tested for -8db SNR, it needs to be tested for adverse SNR conditions specified for a given application. Similarly an analysis on the optimum word length in Fixed precision, meeting these adverse SNR requirements also needs to be carried out.

CHAPTER 5

REFERENCES

[1] W.M.Gentleman and H.T.Kung, "Matrix Triangularization by Systolic Array," Proc. SPIE, Real-Time Signal Processing IV, 1981.

[2] N.L.Owsley, "Systolic Array Adaptive Beamforming," NUSC Technical Report, 1987

[3] J.G.McWhirter and T.J.Shepherd, "Systolic Array Processor for MVDR Beamforming," IEE Proceedings, Vol 136, April 1989.

[4] J.E Hudson and T.J.Shepherd, "Parallel Weight Extraction by a Systolic Least Squares Algorithm," Proc. SPIE, Advanced Algorithms and Architectures for Signal Processing IV, 1989.

[5] T.J.Shepherd, J.G.McWhirter and J.E Hudson, "Parallel Weight extraction from a Systolic Adaptive Beamforming," Mathematics in Signal Processing II, 1990.

[6] W.Givens, "Computation of Plane Unitary Rotations Transforming a General Matrix to Triangular Form," J. Soc. Ind. App. Math., 1958.

[7] C.M.Rader, "VLSI Systolic Arrays for Adaptive Nulling," IEEE Signal Processing Magazine, 1996.

[8] C.F.T.Tang, K.J.R.Liu and S.A.Tretter, "A VLSI Algorithm and Architecture of CRLS Adaptive Beamforming," Proc. of the Conf. on Information Sciences and Systems, March, 1991.

[9] E. N. Frantzeskakis and K. J. R. Liu, "A Class of Square Root and Division Free Algorithms and Architectures for QRD-Based Adaptive Signal Processing," IEEE Transactions on Signal Processing, Vol. 42, No. 9, September, 1994.

[10] J. Gotze and U. Schwiegelshohn, "A square root and division free Givens rotation for solving least squares problems on systolic arrays," SIAMJ. Sci., Statist. Comput., Vol. 12, No. 4, July 1991.

[11] Chris Dick, Fred Harris, Miroslav Pajic and Dragan Vuletic, "Real-Time QRD-based Beamforming on an FPGA Platform," 2006.

[12] A.H.Sayed,"Adaptive Filters", 2008.

[13] Xilinx,"LogicCORE IP Fast Fourier Transform v7.1 Datasheet, DS260," 2011.

[14] Xilinx,"LogicCORE IP CORDIC v5.0 Datasheet, DS858," 2011.

[15] Xilinx,"LogicCORE IP Divider Generator v4.0 Datasheet, DS819," 2011.