

**A STUDY IN THE DESIGN OF DIGITAL BEAMFORMERS**

by

Sherk Chung

Submitted to the

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

in partial fulfillment of the requirements

FOR THE DEGREES OF

BACHELOR OF SCIENCE

and

MASTER OF SCIENCE

at the

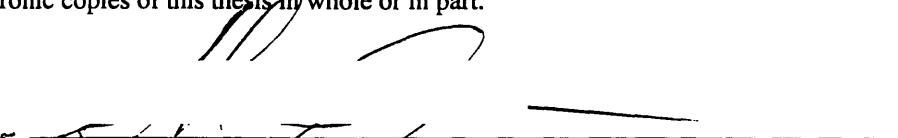
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June, 1994

© Sherk Chung, 1994. All Rights Reserved.

The author hereby grants to MIT the permission to reproduce and to distribute publicly paper and electronic copies of this thesis in whole or in part.


Signature of Author

  
Department of Electrical Engineering and Computer Science,  
June 1994.

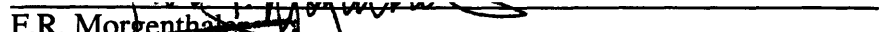
Certified by

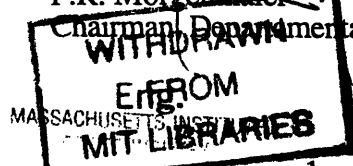
  
Dr. Dan E. Dudgeon  
Academic Thesis Supervisor, Lincoln Laboratory

Certified by

  
Dr. Amir I. Zaghloul  
Company Thesis Supervisor, COMSAT Laboratories

Accepted by

  
F.R. Morgenthaler  
Chairman, Departmental Graduate Committee



JUL 13 1994

# A Study in the Design of Digital Beamformers

by

Sherk Chung

## Abstract

Standard beam forming methods for communication satellites make use of microwave circuits to shape and direct the beams. Due to recent advances in digital technology, independent beam forming networks (BFN) can be implemented digitally and designed to fit on a series of chips. Certain digital designs will overcome many of the obstacles commonly encountered in analog BFNs. This thesis discusses the issues affecting the design of multibeam digital beamformers, and the trade-offs associated with implementing the beamformer as a digital system. A design of a beam forming system, which is intended for use onboard communication satellites, is also included. The proposed beamformer will control and shape multiple beams simultaneously in a large phased array antenna.

**Academic Thesis Supervisor:** Dr. Dan E. Dudgeon  
Senior Staff Member  
Lincoln Laboratory

**Company Thesis Supervisor:** Dr. Amir I. Zaghoul  
Manager, Satellite Antennas Dept.  
COMSAT Laboratories

## Table of Contents

Abstract.....	2
Table of Contents.....	3
List of Figures.....	4
Acknowledgments.....	5
1. OVERVIEW.....	6
1.1 Introduction.....	6
1.2 Theoretical Overview.....	7
2. REQUIREMENTS FOR MULTIPLE BEAM SATELLITES.....	10
2.1 On Board Circuit Restrictions.....	10
2.2 Cost and Reliability.....	10
2.3 General System Features.....	10
2.4 Frequency Reuse.....	11
3. CHANNELIZATION.....	11
3.1 Usage of Channelization.....	11
3.2 The Polyphase Filter.....	12
4. TRADE-OFFS AFFECTING THE DBF.....	13
4.1 Advantages in Digital Beam Forming.....	14
4.2 Limitations to Digital Beam Forming.....	14
5. SENSITIVITY ANALYSIS OF THE DBF.....	15
5.1 Quantization Error of the ADC.....	15
5.2 Error Tolerance for Weighting Coefficients.....	16
5.2.1 Error Tolerance Requirements.....	17
5.2.2 Analysis Using ARRAY.....	18
5.2.3 Simulation Results.....	22
6. THE SATELLITE ANTENNA.....	23
6.1 Antenna Type.....	23
6.2 Antenna Directivity and Gain.....	25
6.3 Grating Lobe Elimination.....	27
7. DESIGNS FOR A DBF.....	28
7.1 Narrow-Band vs. Wide-Band DBF.....	28
7.2 Conventional DBF Matrix.....	29
7.3 Improved Design for a WB DBF.....	30
7.4 Design for a Distributed NB DBF.....	31
7.4.1 NB Receive DBF Algorithm.....	32
7.4.2 NB Transmit DBF Algorithm.....	33
7.4.3 Design Appraisal.....	34
7.5 DBF Components.....	36
7.5.1 DBF Module.....	36
7.5.2 Reorder Buffer.....	36
7.5.3 DEMUX/REMUX.....	37
8. SUMMARY AND CONCLUSIONS.....	38
8.1 Conclusions on Error Analysis.....	38
8.2 Design Summary.....	39
8.2.1 Conclusions on Satellite Antenna Study.....	39
8.2.2 Usage of NB/WB DBF.....	39
8.3 Reliability of the DBF.....	40
8.4 Suggestions for Improvements on DBF Design.....	41
References.....	42
Appendix 1.....	43
Appendix 2.....	44
Appendix 3.....	47

## List of Figures

Figure 1.1a: Satellite block diagram.....	7
Figure 1.2a: Conventional beamformer.....	8
Figure 1.2b: Beam forming matrix.....	9
Figure 3.2a: Polyphase filters.....	13
Figure 5.2a: Phase errors due to quantization.....	17
Figure 5.2b: Signal weighted with unquantized coefficients.....	19
Figure 5.2c: 4-bit Quantization of weights.....	19
Figure 5.2d: 6-bit Quantization of weights.....	20
Figure 5.2e: 7-bit Quantization of weights.....	20
Figure 5.2f: 8-bit Quantization of weights.....	21
Figure 5.2g: 9-bit Quantization of weights.....	21
Figure 5.2h: Quantization effects on areas of frequency reuse.....	22
Figure 6.1a: Focused reflector-fed array.....	24
Figure 6.2a: Directivity of a phased array antenna element.....	26
Figure 6.3a: Grating lobes.....	26
Figure 6.3b: Grating lobe movement.....	27
Figure 7.2a: Conventional WB DBF design.....	29
Figure 7.3a: A design for a distributed WB DBF.....	30
Figure 7.4a: A design for a distributed NB DBF.....	31
Figure 7.4b: The Receive DBF process.....	33
Figure 7.4c: The Transmit DBF process.....	33
Figure 7.5a: The DBF module.....	35
Figure 7.5b: The Reorder module.....	37
Figure 7.5c: The DEMUX module.....	38
Figure 8.2a: Distributed NB DBF vs. Distributed WB DBF.....	40

# Acknowledgments

I would like to thank all the staff members of the Satellite Antennas Department for their valuable help in the research performed at COMSAT Labs. In particular, I would like to thank:

Dr. Amir I. Zaghoul, manager of the Satellite Antennas Department at COMSAT Labs., for his invaluable guidance in the theory and requirements of beam forming systems,

Dr. Soheil Sayegh, Principal Scientist at COMSAT Labs., for his insight in digital systems and extensive help in the design of digital beamformers,

Dr. Dan E. Dudgeon, Senior Staff Member at Lincoln Lab., for his generous support and help in signal processing theory,

Eric Kohls, member of the Satellite Antennas Department at COMSAT Labs., for his assistance on satellite systems and antennas,

and finally my parents, Dr. and Mrs. K. S. Chung, who have always supported my academic goals and have constantly provided encouragement.

# 1. OVERVIEW

## 1.1 Introduction

The use and demand for communication satellite links increase as the markets for mobile communications continue to grow. Also growing are the demands in coverage requirements imposed upon the satellites. The satellite links must be able to support multiple beams with frequency reuse and power sharing between the beams. Ever changing traffic patterns of mobile communication units also require the system to steer and re-configure the beam. These needs are best furnished with onboard satellite beamformers. To date most satellites use analog (microwave) beam forming systems to handle the multi-beam communication process. As cellular applications expand, satellites will be required to handle more beams, and have the ability to adaptively null interference from other signals. The solution for multiple beam handling on analog systems is to route the signals with microwave switch matrices (MSM) and beam forming matrices (BFM), which are functionally separate from the onboard digital processor and antenna apertures. Although these BFMs give analog beamformers the capability to simultaneously direct multiple beams, their size and power losses grow as a function of both the number of radiating antenna elements and the number of beams generated by the system.

With a digital version of the beam forming system, all the routing is performed at baseband, as an extension to the onboard processor (see **Figure 1.1a**). The beam forming circuitry can also be distributed across the antenna, and in the case of phased array antennas, may be integrated into the active

radiating elements. The cost and implementation advantages resulting from the compact size and modularity of integrated Digital Beamformer (DBF) chips provide a strong incentive to use DBFs in lieu of equivalent analog systems.

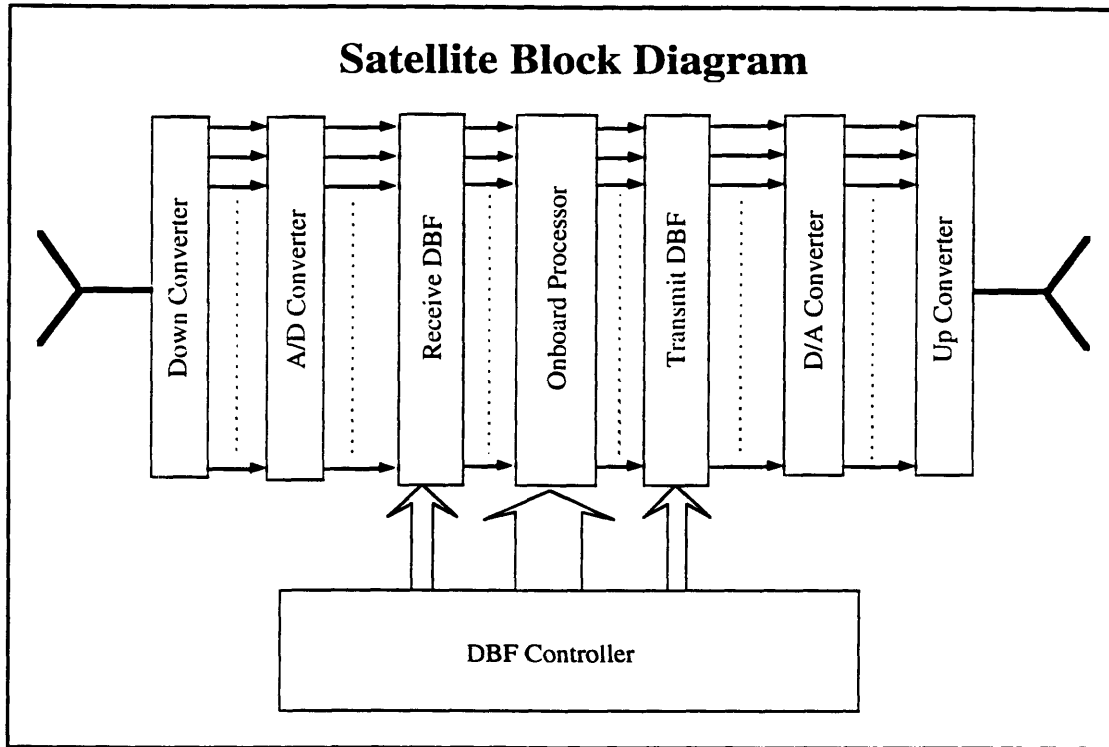
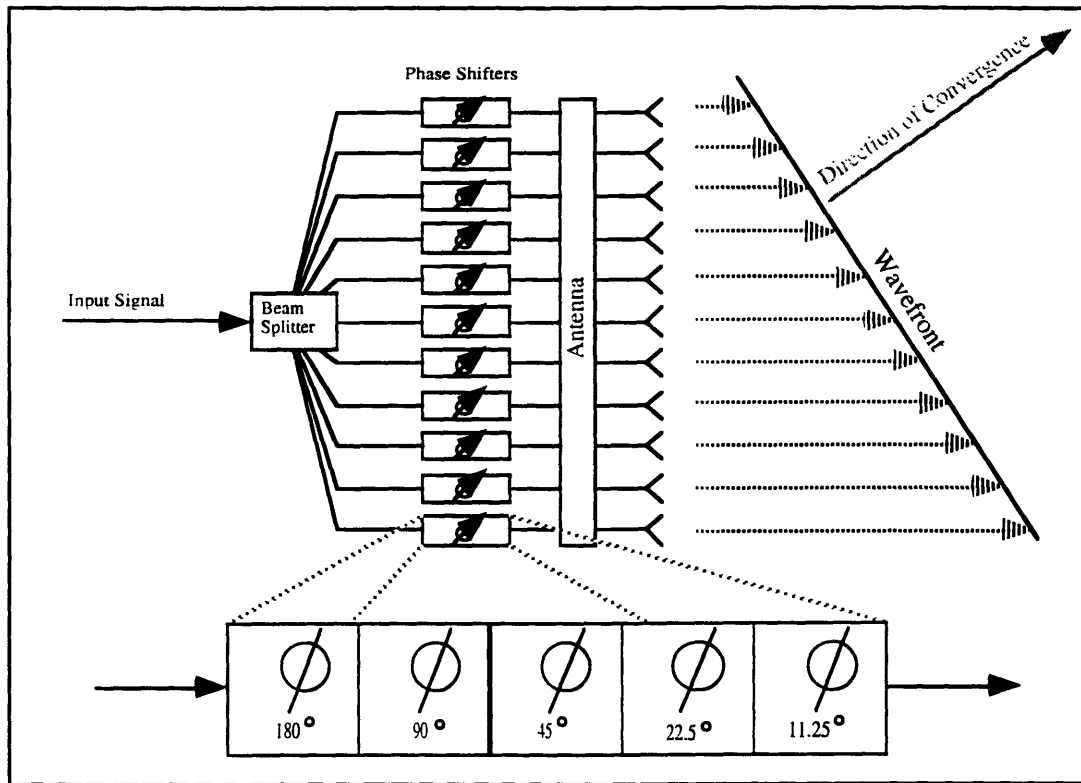


Figure 1.1a: Satellite block diagram.

## 1.2 Theoretical Overview

To explain the concept of beam forming, the antenna is modeled as an array of radiating elements. To transmit the beam in a certain direction, the beam is split evenly into the array elements and incremental delays are added across the elements, thus affecting the phase of the signals radiating from the antenna (see Figure 1.2a). The phase changes will cause the beam to converge in a certain direction, and in other directions the beam will be out of phase and may appear as sidelobes. Simply stated, given a look direction  $\theta$  and an

antenna array of  $N$  elements, the phase of the split signals would change by  $2\pi k \sin(\theta) / \lambda$  for  $k = 1$  to  $N$  (where  $\lambda$  is the wavelength of the signal). In some applications, it is also desirable to change the shape of the outgoing beam. This is accomplished by scaling the amplitude of the signals.

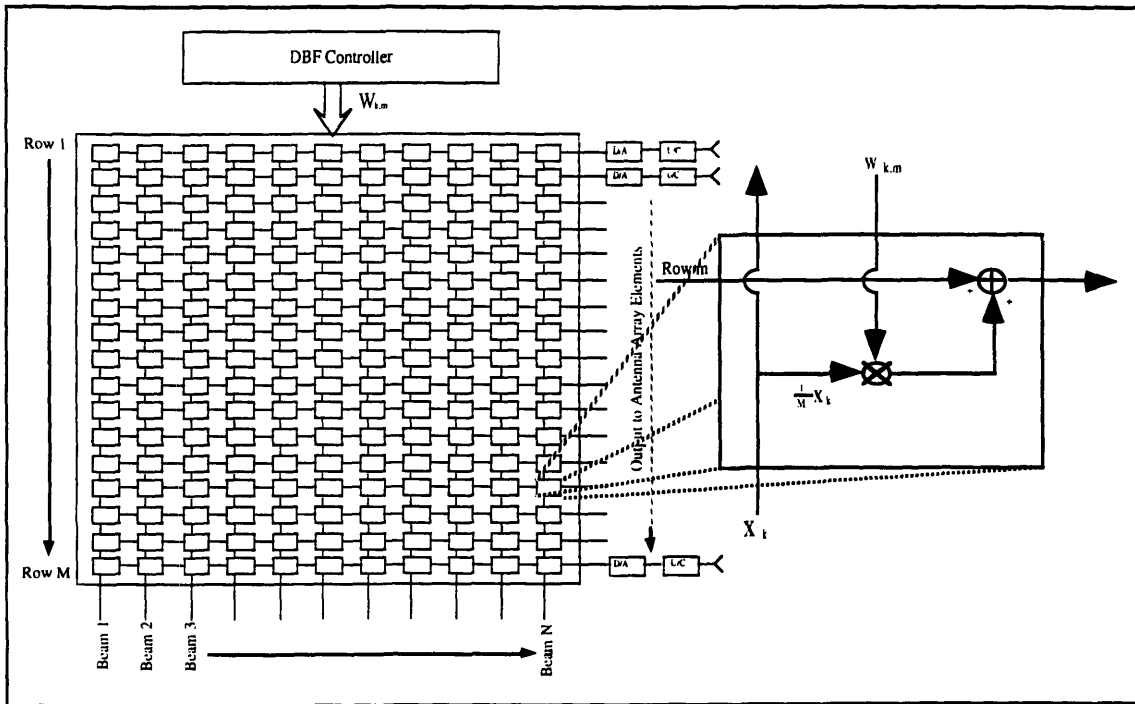


**Figure 1.2a:** Conventional beamformer.

Incremental delays are added across the antenna elements to change the phase of the radiating elements.

Conventional microwave RF beam forming methods implement the required delays using microwave phase shifters. The phase shifters may consist of a number of cascaded modules, with the first causing a  $\pi$  phase shift, the next a  $\pi/2$  phase shift, the next a  $\pi/4$  phase shift, etc. If only 5 modules are cascaded to form a delay, it could have up to a  $\pi/32$  error margin for the phase. Amplitude scaling is performed using RF attenuators.





**Figure 1.2b:** Beam forming matrix.  
Each element radiates a sum of the weighted signals.

Whereas in analog systems the signals must be passed through attenuators and phase shifters, in a digital system the entire beam forming process for multiple beams can be reduced to complex multiplications and additions. To control the direction and shape of a beam, each individual signal radiating from the antenna elements must be scaled in magnitude and in phase, which is equivalent to multiplying the signal by a complex number (commonly referred to as a *weight*). To transmit multiple beams, the beam forming process described above is repeated in parallel for each beam (see **Figure 1.2b**). Each antenna element radiates a sum of weighted signals. The weight is varied according to the position of the element and the direction of the beam. The beams are weighted independently and simultaneously, so the look directions of the beams need not be the same.

## **2. REQUIREMENTS FOR MULTIPLE BEAM SATELLITES**

### **2.1 On Board Circuit Restrictions**

The BFN must adhere to some basic restrictions. Among other limitations, size, weight, and power usage are common to all satellite systems. In a system that uses an onboard digital processor, it may be possible to implement the DBF as a small extension to the processor, and would thus contribute minimally to the size and weight of the payload. If the DBF acts as an independent processor, the layout must be carefully designed as to not exceed any of the power and mass limitations.

### **2.2 Cost and Reliability**

The DBF must be designed to keep the implementation of the system at a relatively low cost. If the entire system consists of identical modules without complicated connectivity, the modules can be mass produced and assembled at a lower cost. Since digital circuits tend to be lighter and smaller than their analog counterparts, hardware modularity becomes a necessity. Redundancy is needed for high reliability. Failing to meet these requisites may render the satellite a financial burden rather than an asset.

### **2.3 General System Features**

The DBF must also support flexibility and reconfigurability. Changing traffic patterns make it essential that the satellite be able to dynamically redirect beams to different locations. Conflicts in channel usage may require the satellite to reconfigure carrier frequencies on the fly. Sharing the power between beams allows for such changes. The DBF must be flexible enough to meet all these demands.

## **2.4 Frequency Reuse**

Another important factor is the requirement of frequency reuse. If two beams that are being transmitted from the same antenna are directed far enough away from each other, both beams may be able to use the same frequency without interfering with each other as long as the sidelobe levels of the beams are kept within certain bounds. When two beams containing signals that use the same frequency are steered too close together, the BFN must compensate by moving the signals of one of the beams to a different frequency channel. Providing this feature will affect both the size and computational burden of the DBF.

## **3. CHANNELIZATION**

### **3.1 Usage of Channelization**

The means for frequency reuse fall primarily on the channelization processor. Channelization is an operation commonly associated with beam forming, but is not in itself a requirement in steering beams. It is required when signals need to be broken down to individual channels for re-assignment. Although it can be functionally independent of the BFN, if the beam forming is performed across individual channels instead of the full bandwidth of the signals, the beam forming and channelization operations can be combined. For the purposes of this text, beam forming and channelization are both important to the satellite communication process, and the channelizer will be considered as a component of the beamformer.

### 3.2 The Polyphase Filter

The means for breaking the beam down into individual channels is through a polyphase filter. A common polyphase filter consists of a bank of filters that have the same amplitude response but are shifted in frequency. Extracting channels from a frequency spectrum makes good use of a polyphase network. The filter that was used to extract the first channel is the same one used to filter out the second channel with a shift in frequency (which corresponds to a phase scaling in the time domain). A shift in frequency by  $f_0$  will lead to a scaling of  $e^{-j2\pi f_0 n}$  in time. Hence given a lowpass filter  $g[n]$  each polyphase filter  $h_p[n]$  will be:

$$h_p[n] = g[n]e^{-j2\pi f_0 \cdot pn} \quad \text{for } p=1 \text{ to } N$$

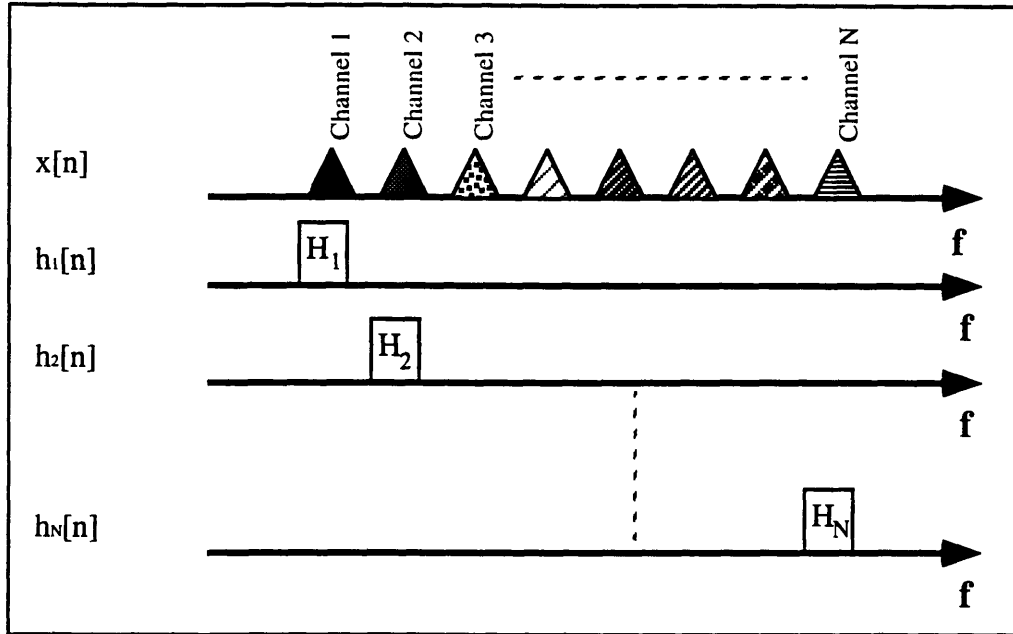
where  $N$  is the number of channels (see **Figure 3.2a**). Convolution of the input signal with each  $h_p[n]$  we get

$$\begin{aligned} y[n] &= \sum h_p[k] \cdot x[n-k] \\ &= \sum x[n-k] \cdot g[n]e^{-j2\pi f_0 (p \oplus N)} \end{aligned}$$

which can be reduced to an altered DFT algorithm, thus allowing the use of an FFT module. The  $\oplus$  sign refers to the modulo operator.

After each channel is extracted, the output can be decimated, since the resulting bandwidth of the channel is necessarily smaller than the composite wave form composed of all the channels. If there are  $N$  channels aligned side-by-side, then each channel can be decimated by a factor of  $N$ . But since

there are  $N$  of these outputs, the total number of output samples equals the total number of input samples, and the bit rate remains constant throughout.



**Figure 3.2a:** Polyphase filters.

The filters used to extract channels from the beam have the same magnitude and differ only in phase. They are shifted along the frequency axis to filter out the different channels.

#### 4. TRADE-OFFS AFFECTING THE DBF

Employing a digital circuit as a beamformer has both advantages and downfalls. The analog BFN can only be implemented as a hardware system, while the DBF may simply be a program run by an onboard computer. This gives the DBF the flexibility to incorporate additional features, such as error correction and data compression, without additional hardware. However, the processing must be done in real-time, which may not be so practical.

#### 4.1 Advantages in Digital Beam Forming

From an *implementation* point of view, there are several advantages and some limitations that the DBF has when compared to the analog BFN. A DBF can generate multiple independently steerable beams with *no loss in signal-to-noise ratio* during the computation. It can also adaptively cancel interference from other beams, adding corrections for mutual coupling effects and incorporating corrections for gain and phase mismatch errors in the receiver channels. As the number of beams and antenna elements extends into the hundreds, the interconnectivity of the analog BFN becomes a serious design problem. The circuitry for the DBF can be implemented as a series of integrated circuits, giving the beamformer the desired modularity and eliminating the RF cabling associated with analog BFNs.

#### 4.2 Limitations to Digital Beam Forming

The limitations of the DBF are those shared by all real-time digital systems. Namely, the quantization effects and sampling speeds are key issues. The signals must be sampled quickly enough to satisfy the Nyquist rate. In systems such as INTELSAT where total signal bandwidths can range up to 500 MHz, sampling at the Nyquist rate becomes a serious obstacle. And when hundreds of beams and antenna elements are involved, the computational burden of the DBF may be overwhelmingly large. In addition, the burden is increased by the down/up conversions required to do the beam forming at baseband. It may then become necessary to break down the traffic into smaller bands and process each band in parallel. This can be achieved by using a hybrid analog/digital system, where microwave components are used to filter out the separate bands that are then passed onto the DBF for digital

processing. Even then, microwave filters can be bulky, and having several filters per antenna element may present a size problem.

Digital beam forming is attractive, however, if other functions of the onboard processor are done digitally. Then the down/up conversions and analog to digital conversions are done regardless of the BFN type.

## **5. SENSITIVITY ANALYSIS OF THE DBF**

The error induced by quantization of the analog to digital converters (ADC) will have a direct impact on the performance of the satellite antenna. In a DBF system, the number of bits used to represent the signal and corresponding weights must be determined based on the required performance. Using a large number of bits to represent a sample will yield high accuracy, but will greatly increase size, weight, power consumption, and cost of the system, as well as complicate interconnectivity between the chips and add to the computational load. On the other hand, using an inadequate number of bits will degrade the quality of the signal. The optimal solution is to find a size that is small enough to keep the circuit size acceptable, and will also maintain the accuracy needed in the DBF.

### **5.1 Quantization Error of the ADC**

A common measure of signal quality is the signal-to-noise ratio (SNR), defined as the ratio of signal variance (or power) to noise variance. The number of bits used to represent the signal will have a direct effect on the SNR. A  $(B+1)$  bit quantizer will yield a noise variance:

$$\sigma_e^2 = \frac{2^{-2B} X_m^2}{12}$$

where  $X_m$  is the full-scale value of the quantizer (in this case the ADC). Expressed in decibels, the SNR becomes:

$$SNR = 10 \log_{10} \left( \frac{\sigma_x^2}{\sigma_e^2} \right) = 10 \log_{10} \left( \frac{12 \cdot 2^{2B} \sigma_x^2}{X_m^2} \right) = 6.02B + 10.8 - 20 \log_{10} \left( \frac{X_m}{\sigma_x} \right)$$

We can see that the SNR increases roughly 6 dB for each additional bit.  $X_m$  is a parameter of the quantizer and is usually fixed.  $\sigma_x$  is the rms value of the amplitude of the signal, and therefore must be less than the peak amplitude of the signal. If  $\sigma_x$  is too large, then the peak amplitude of the signal will exceed the full-scale value of the quantizer, resulting in a distortion of the original signal. When this happens the formula given above does not apply. However, if  $\sigma_x$  is too small, we can see from the term

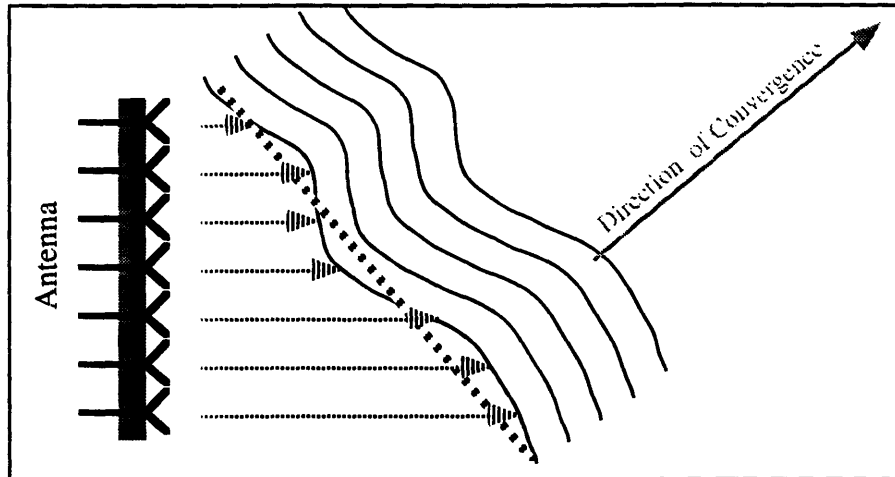
$$-20 \log_{10} \left( \frac{X_m}{\sigma_x} \right)$$

that the SNR is decreased. It is therefore important that the peak amplitude of the signal be matched carefully to the full-scale value of the quantizer.

## 5.2 Error Tolerance for Weighting Coefficients

The weighting coefficients are not signals as such, and therefore another standard must be used to determine the word length that will yield the accuracy needed for the DBF.





**Figure 5.2a:** Phase errors due to quantization.

Quantization of the weights will cause an error in the phase alignment. On average, the errors cancel each other out and only the gain of the beam is affected.

### 5.2.1 Error Tolerance Requirements

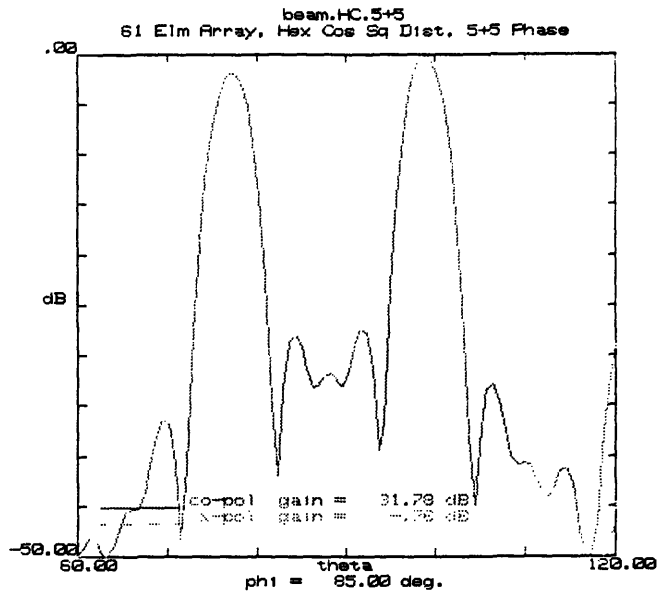
Although issues such as peak gain and steering accuracy are of some concern, the most important requirement is the suppression of sidelobe levels in the areas of frequency reuse. As discussed earlier, the beam is formed by weighting the input signal with complex coefficients. Ordinarily, the phase component of the weight is the sole input needed to direct the beam. The magnitude of the weight is used primarily to taper off sidelobe levels. Quantizing the phase of the weight will yield an error in the phase of the signal being radiated by an antenna element. Since some signals have a positive error and some negative, these phase errors average each other out, and the radiating signals converge in the predetermined direction (see **Figure 5.2a**). The only important side effect to phase mismatch errors is a slight reduction in gain. Quantizing the magnitudes of the weights also has a minor effect on the gain of the beam. These changes in gain, however, are only noticeable when very few bits are used, and even then the error margin

is very small. One major factor that is altered by quantization is the suppression of sidelobes. Using too few bits to represent the magnitude could mean a rise in sidelobes in the areas of frequency reuse.

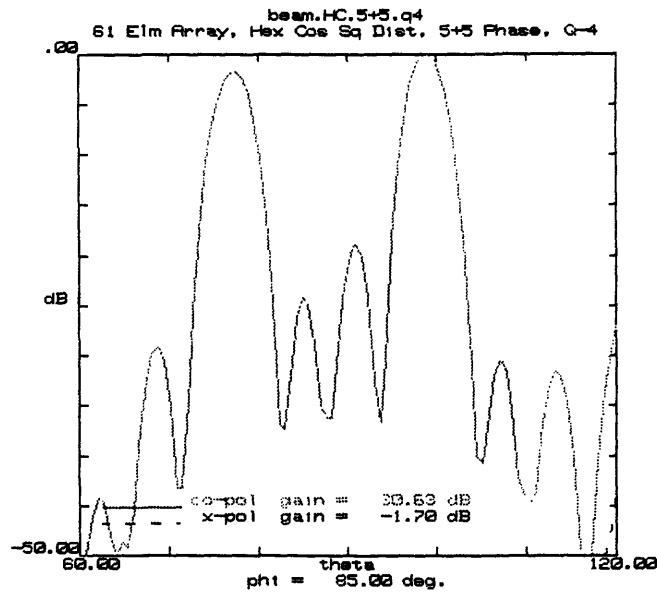
### 5.2.2 Analysis Using ARRAY

Computer simulations were performed to analyze the performance of a beam former with different weight quantization sizes. The ARRAY program, developed at COMSAT Laboratories as a tool in designing antenna arrays, was used to generate a set of complex coefficients in polar form (i.e., in the form  $Ae^{j\phi}$ ) to shape a beam. Different antenna sizes, amplitude distributions, and beam angles were used. The coefficients were then quantized from 4 to 12 bits successively, and used to shape a beam. The original unquantized complex coefficients were generated to keep the sidelobe levels at -25 dB relative to the beam peak in the isolated areas of frequency re-use, when the beam is steered within  $10^\circ$  from boresight. The shaped beams were evaluated on peak gain and sidelobe levels in the areas of frequency reuse. The successive improvement with each additional bit can be seen in **Figures 5.2b-g**.

Complex numbers in digital circuits are separated into I and Q channels (in rectangular form, i.e.,  $A+jB$ ). It was necessary to verify that complex weights would yield the same results in I/Q format as they did in the ARRAY program. The coefficients from the ARRAY program were converted to rectangular form, quantized (again, from 4 to 12 bits), converted back to polar form, and used to shape a beam. There was an increase in accuracy and sidelobe suppression up to 6 bits. After 8 bits, there was no significant improvement.



**Figure 5.2b:** Signal weighted with unquantized coefficients. The figure above represents a cross-section of a farfield plot. The angles theta and phi are the horizon and azimuth spans (in degrees) of the corresponding contour plot.



**Figure 5.2c:** 4-bit Quantization of weights.

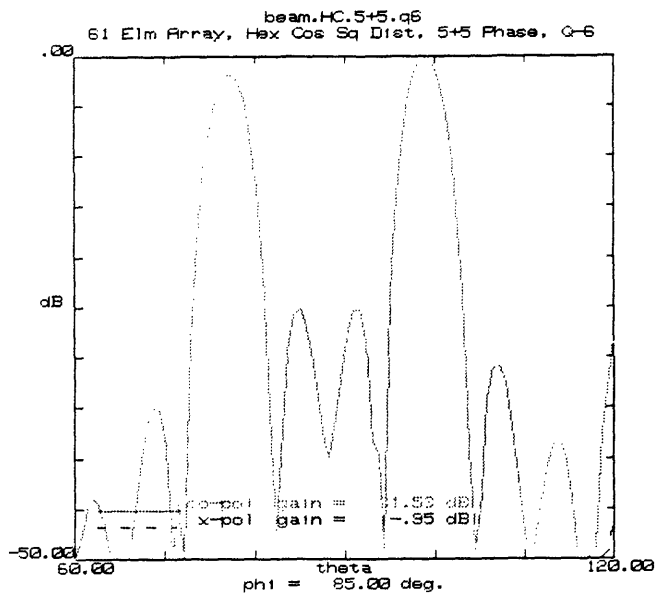


Figure 5.2d: 6-bit Quantization of weights.

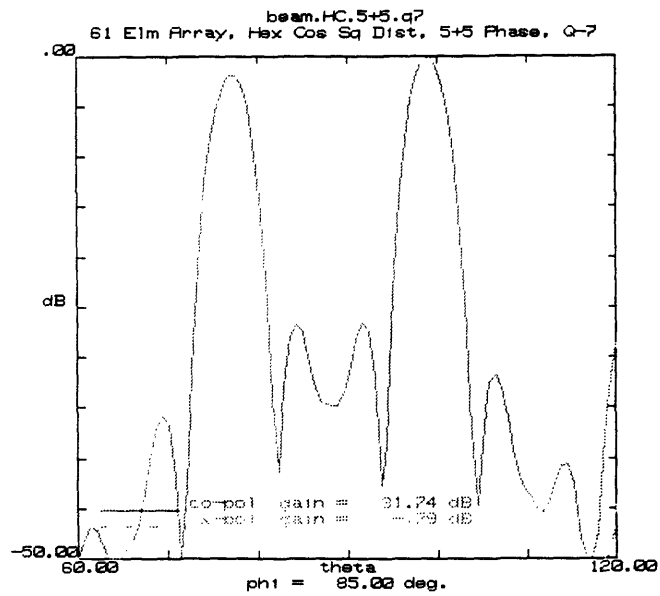


Figure 5.2e: 7-bit Quantization of weights.

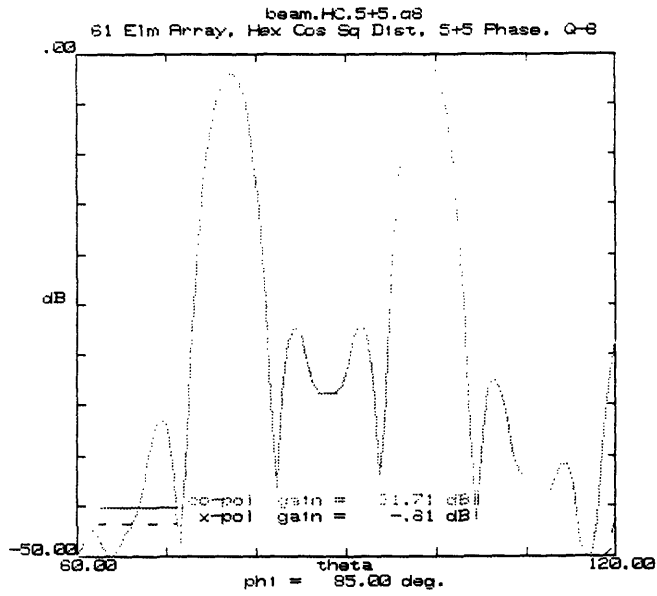


Figure 5.2f: 8-bit Quantization of weights.

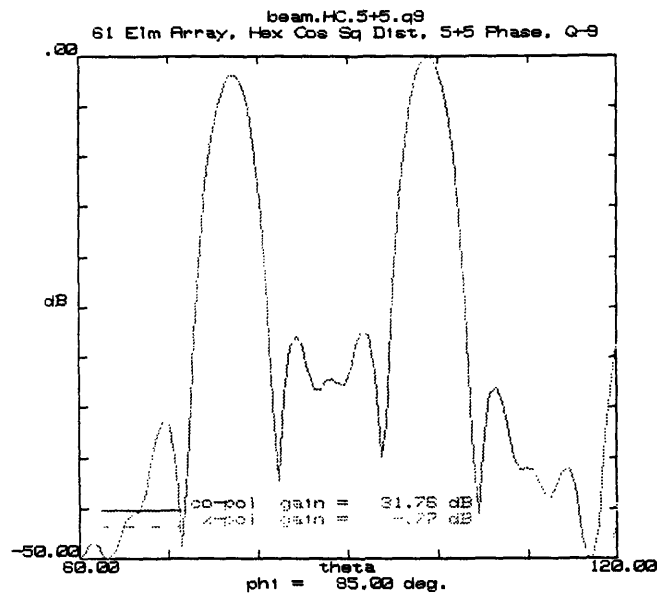
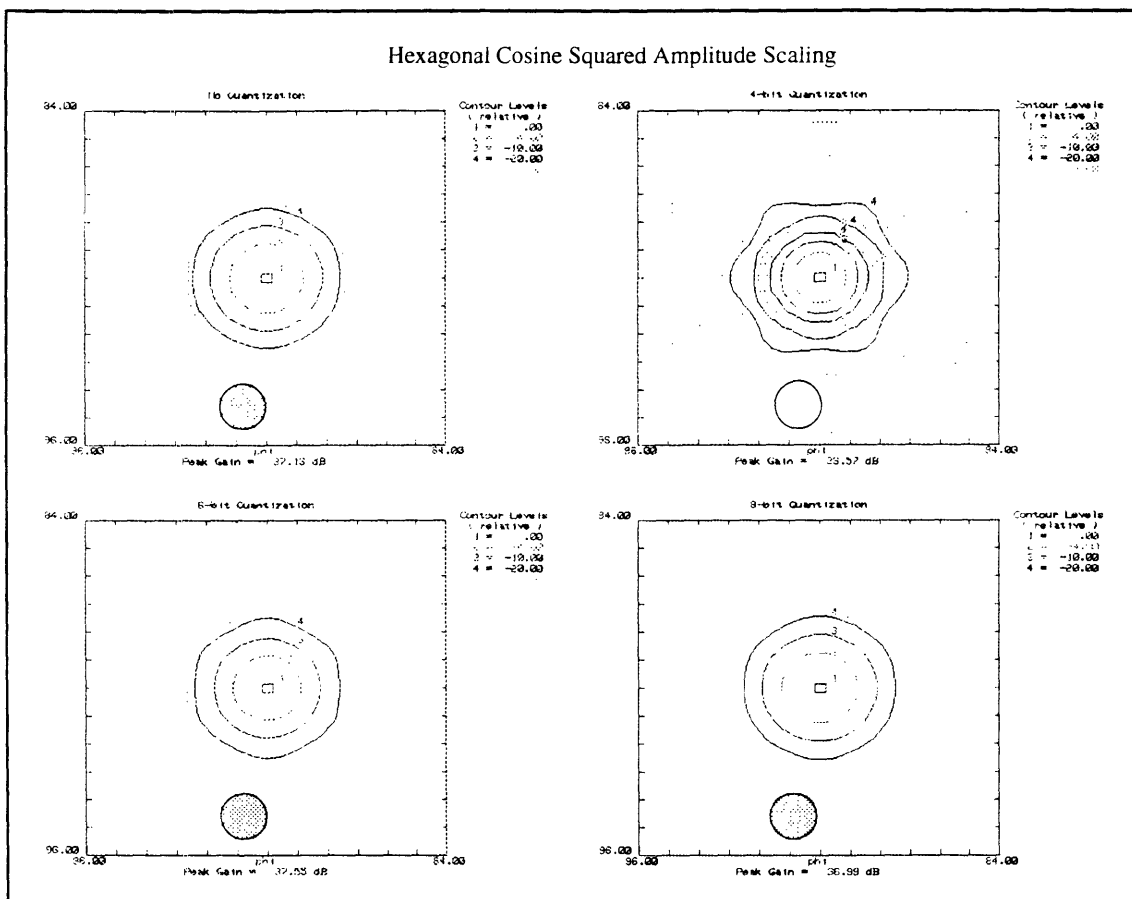


Figure 5.2g: 9-bit Quantization of weights.

### 5.2.3 Simulation Results

The beams shaped with the 4-bit weights produced a pattern which barely resembled the nominal pattern, with very high sidelobe error margins. A vast improvement resulted from increasing the number of bits representing the coefficients from 4 to 6. At 6 bits, the shaped beam closely resembled the original, the peak gains were nearly equal, and the sidelobe levels were down to an acceptable range (25 dB down). After six bits, the increase in accuracy was negligible, as was the peak gain, and there was no betterment in sidelobe level. After 8 bits there was no apparent improvement whatsoever (as shown in Figures 5.2b-g).



**Figure 5.2h:** Quantization effects on areas of frequency reuse. Shaded circles represent areas of frequency reuse. Notice that 4-bit quantization is not enough to suppress the sidelobes in the designated area.

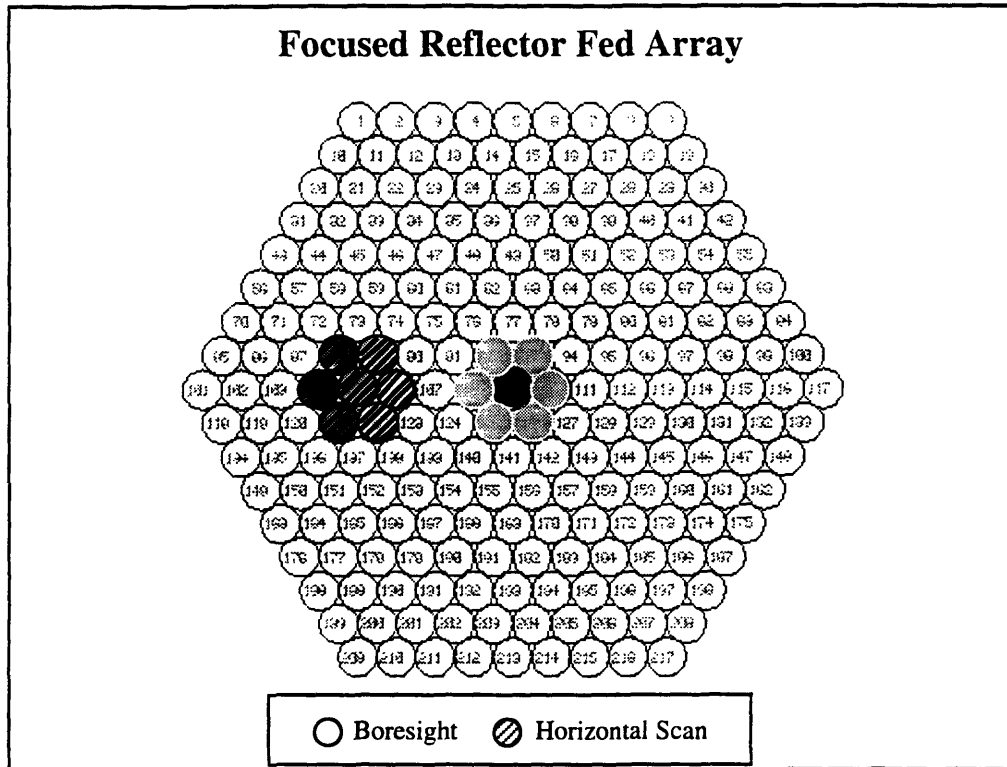
Farfield contour plots of a beam are shown in **Figure 5.2h**. The same beam was formed with weights quantized to 4, 6, and 8 bits successively. In this example we can see that although a 4-bit representation of the weights is insufficient, going to 6-bit word lengths gives us the desired sidelobe level tapering for frequency reuse, and nothing vital is gained by increasing the quantization level to 8 bits.

## 6. THE SATELLITE ANTENNA

An important part of the designing process of a beamformer is to choose the size and type of the satellite antenna.

### 6.1 Antenna Type

Two of the main categories of satellite antennas are focused reflector systems and direct-radiating phased arrays. The functional difference between the two is that the former uses only a small number of elements to transmit a beam, while the latter uses all the array elements to form a beam. Phased array antennas radiate directly from the elements in all directions. Reflector antenna elements reflect the signal off a dish to increase directivity. For phased arrays the standard beam forming procedures of complex weighting is used. For reflector arrays, however, the direction of the beam can be altered by using a different cluster of elements on the antenna and weighting them differently in magnitude and/or phase (see **Figure 6.1a**).



**Figure 6.1a:** Focused reflector-fed array.

Reflector arrays use only a small number of the available elements to transmit a beam. In the figure above, darker elements represent heavier magnitude weighting. The striped cluster will scan the beam horizontally due to the uneven magnitude scaling, while the center cluster will keep the beam at boresight.

The design of the antenna will have a direct effect on the design of the BFN. Usage of only a fraction of available elements will require switching networks to route the outgoing beams to the correct elements. Although fewer signals are amplified, the complexity of the switching networks will grow as a function of elements and beams. Currently most satellites use the reflector system to transmit a relatively small number of beams. In the future, satellites must be able to support non-uniform and time varying traffic distributions, and may require a very large number of simultaneous beams. The gain requirements and antenna size restrictions for such



applications may favor the usage of phased array antennas. The remainder of this thesis will deal mainly with systems using phased arrays. Unless otherwise specified, all future mentions of antennas may be assumed to be phased array antennas.

## 6.2 Antenna Directivity and Gain

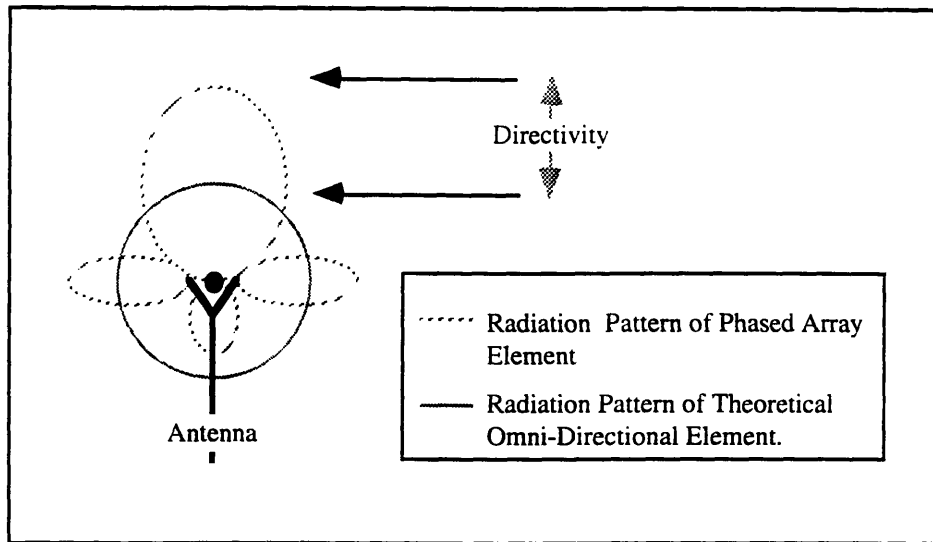
The size of a phased array antenna aperture will affect the directivity of the beam. The directivity of the beam in a certain direction is denoted by

$$D = \frac{4\pi A}{\lambda^2} \eta$$

where  $A$  is the total surface area of the antenna aperture,  $\lambda$  is the wavelength of the signal, and  $\eta$  is the aperture efficiency. Directivity is the ratio between an isotropic pattern and the array pattern (see **Figure 6.2a**). More directivity means that the signal is stronger in a certain direction. The term *gain* is used to describe the directivity minus some efficiency loss due to associated circuitry. Thus,

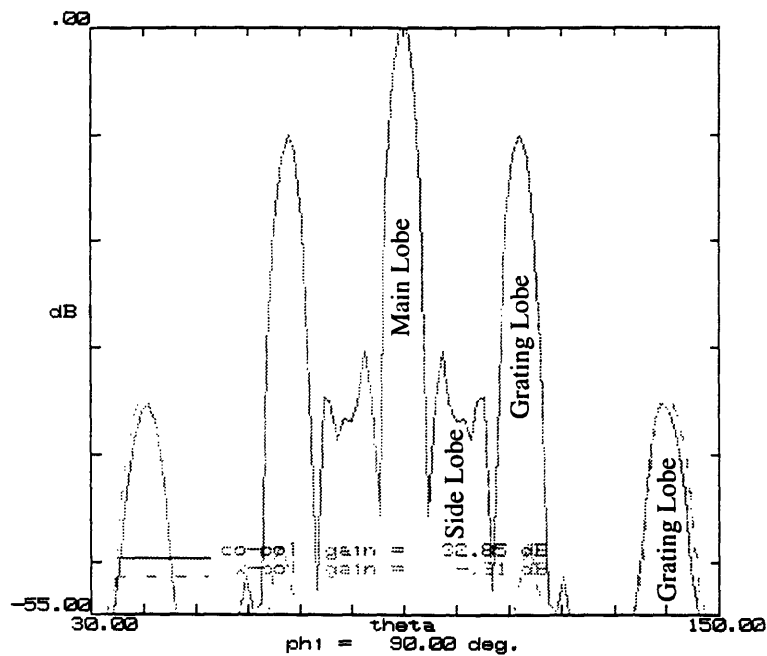
$$G = \gamma D = \gamma \frac{4\pi A}{\lambda^2}$$

where  $\gamma$  includes the aperture efficiency and the efficiency associated with conduction, dielectric, and mismatch losses. It is clear that the larger the antenna, the higher the directivity it can attain.



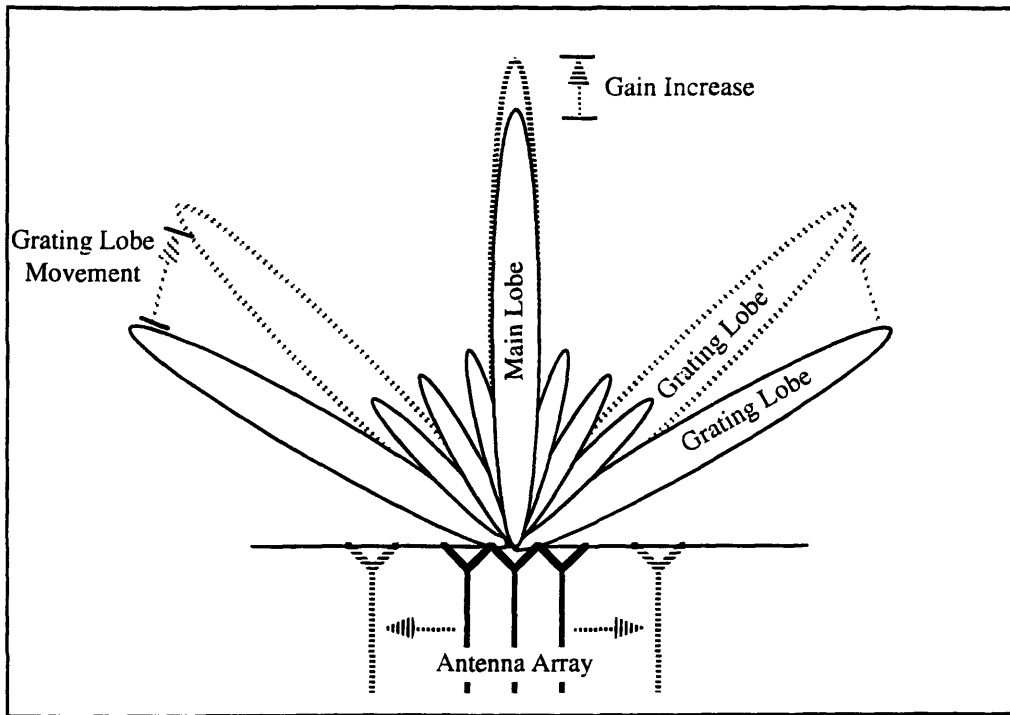
**Figure 6.2a:** Directivity of a phased array antenna element. Directivity is the difference between the actual and theoretical radiation patterns of an isotropic element. It is a measure of signal strength in a given direction.

**Farfield Cross-Cut Pattern of a 61 Element Hexagonal Array With 1.5l Spacing.**



**Figure 6.3a:** Grating lobes.

The graph above is a cross-section of a farfield contour plot such as the ones shown in Figure 5.2h, where theta and phi are the horizon and azimuth of the farfield contour plot. Grating lobes are replicas of the main lobe that appear at intervals determined by the antenna element spacing and the beam wavelength.



**Figure 6.3b:** Grating lobe movement.

As the antenna elements are spread farther apart, the gain increases, but the grating lobes move closer to the main lobe.

### 6.3 Grating Lobe Elimination

Grating lobes are replicas of the main lobe that appear in the farfield at  $2\pi$  intervals (i.e., when the path length differences are a full wavelength). The angle between the grating lobes to the main lobe is inversely proportional to the distance between array elements (see **Figure 6.3a**). Specifically, the angles of the grating lobes are:

$$\theta = \cos^{-1}\left(\frac{\lambda}{d}\right)$$

where  $\theta$  is the direction of the main lobe and  $d$  is the distance in between array elements (see **Figure 6.3b**). By spacing the elements closely together, the

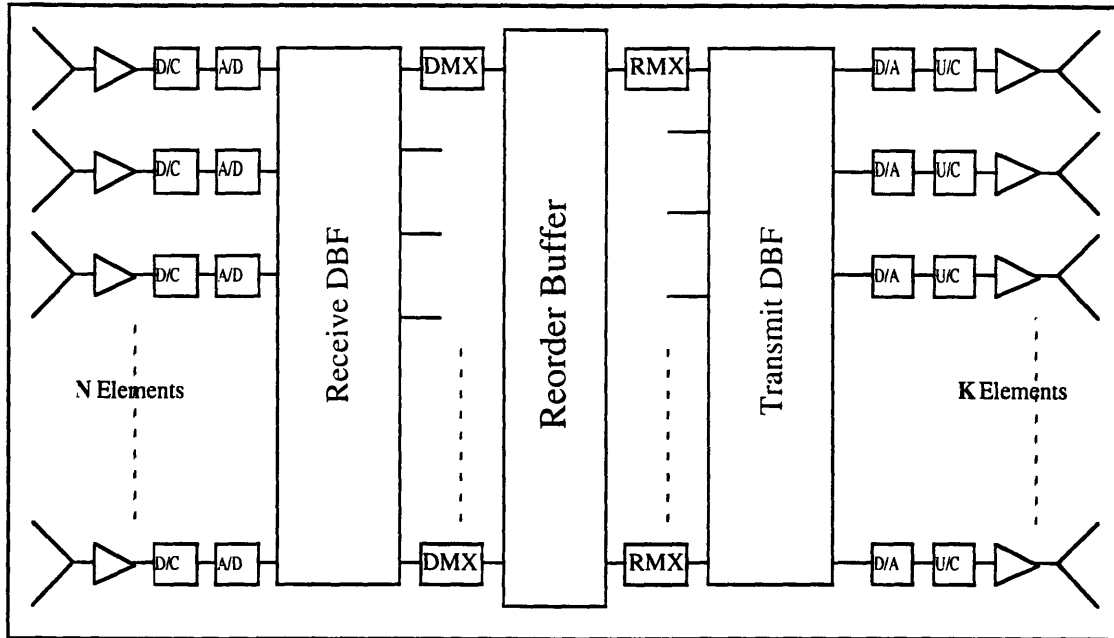
grating lobes can be made to appear off the scope of the coverage area. From a typical geosynchronous satellite orbit, the earth spans about  $9^\circ$  from the center. Thus if the grating lobes are more than  $18^\circ$  from the main lobe, they will not interfere with satellite communications. However, by moving the elements closer together, more array elements are needed to fill the total aperture of the antenna. Keeping this in mind, it would be preferable to have a DBF whose complexity does not grow inordinately large as the number of elements increases.

## **7. DESIGNS FOR A DBF**

In this section DBF architectures with different antenna types will be compared, and designs will be proposed for a DBF with a phased array antenna.

### **7.1 Narrow-Band vs. Wide-Band DBF**

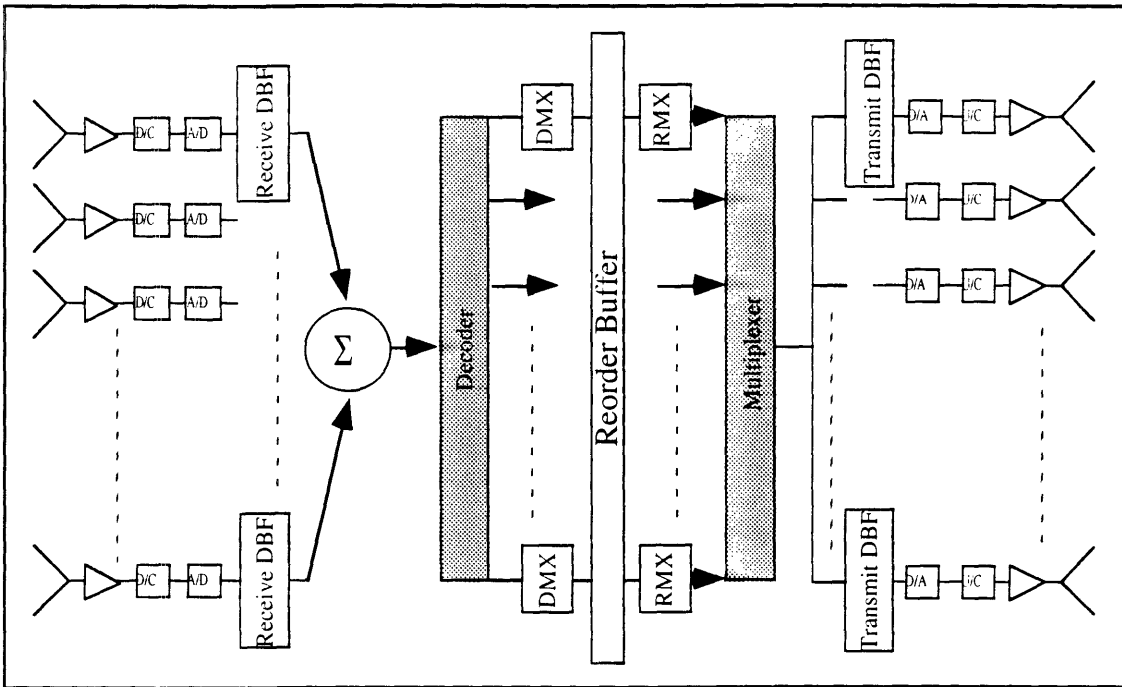
A narrow-band (NB) DBF is not to be confused with a system that handles signals of small aggregate bandwidths. Instead, an NB DBF is a system that performs the needed computation on a channel by channel basis (hence narrow-band), whereas a wide-band (WB) DBF does the computation across the entire frequency spectrum of the incoming signals. To illustrate this point let us consider a case where the entire traffic (perhaps of 10 MHz) transmitted through the satellite consists of one signal that occupies three channels of 32 KHz each. The NB DBF will isolate each 32 KHz channel and process it separately, while the WB DBF will process the entire 10 MHz to form the beam.



**Figure 7.2a:** Conventional WB DBF design.  
 The DBF circuit takes input from all the array elements and outputs all the beams simultaneously.

## 7.2 Conventional DBF Matrix

A common way to implement the DBF is shown in **Figure 7.2a**, with the DBF modules implemented in the form of a weighted matrix, as shown in **Figure 1.2b**. This (WB) design closely resembles a conventional analog BFN matrix, and carries some of the drawbacks accompanying it. Being like the analog BFN, the circuit requires complicated layering of component paths. Notice that the DBF circuit increases as both the number of antenna elements and the number of beams are increased. Also note that the number of beams that this system can transmit is fixed. The complexity of the interconnections makes the DBF expensive and less robust. A switching network is also required to re-route channels to different beams (shown as the Reorder Buffer). These problems make it necessary to break the DBF matrix into modular components to reduce cost and increase reliability.



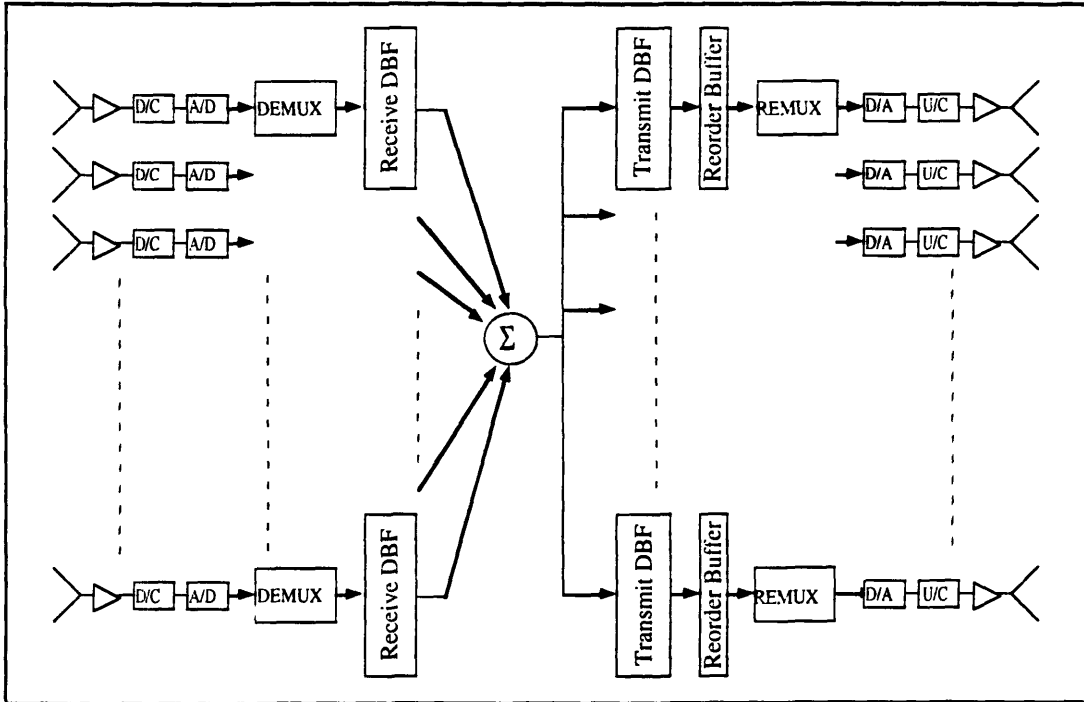
**Figure 7.3a:** A design for a distributed WB DBF.

This design distributes the beam forming circuitry across the antenna elements, eliminating the complicated cabling seen in the matrix BFN.

### 7.3 Improved Design for a WB DBF

A design that distributes the DBF across the array elements and thus eliminates the wiring problem is shown in **Figure 7.3a**. As opposed to the matrix design shown in **Figure 7.2b**, all the beam forming is done sequentially. The receive DBF must replicate each input sample as many times as there are beams and multiply each replica by the correct weight. The output will be a multiplexed sequence of beams (i.e., the first sample from beam 1, the first sample from beam 2, . . . , the first sample from beam M, and then the second sample from beam 1, etc.) The frequency demultiplexer (DEMUX), however, cannot work on a time multiplexed sequence. Thus the decoder is required to separate the sequence into separate beams, whose signals are then re-mapped to the desired channels by way of the

DEMUX/Reorder/REMUX circuits. The re-mapped beams are then time multiplexed, weighted separately at each array element, and transmitted.



**Figure 7.4a:** A design for a distributed NB DBF.

The system is composed of a series of modules that can be implemented as an extension to the antenna. Unlike the WB DBF, no switching matrix is required to re-route channels to different beams.

#### 7.4 Design for a Distributed NB DBF

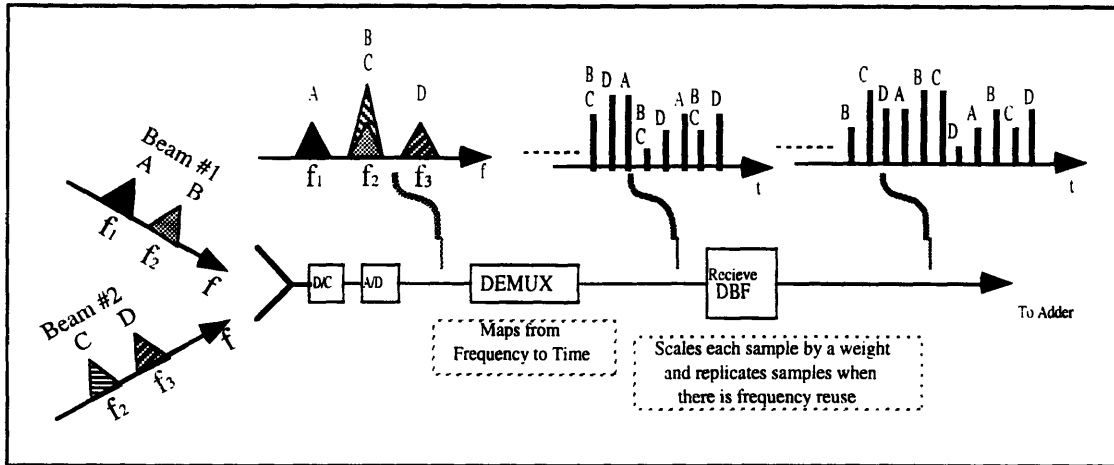
Some problems still remain. A switching network across all beams is still needed to re-route channels, and the size of the system still grows with both the number of array elements and beams. The number of beams also remains fixed, and the power usage of a WB DBF with a phased array antenna can be extremely large. In a case such as telephone transmissions, where phones are used heavily during business hours and barely at all during pre-dawn hours, a system using this design will be required to support the maximum bandwidth used during business hours. During pre-dawn hours,

much of the DBF computations in a WB DBF are "wasted," since all the channels are processed while only a very few are actually used. During these hours a substantially smaller system could be used. These drawbacks can be resolved by forming the beam at narrow band. Keeping in mind the advantages to a distributed modular system, a design for NB DBF was drawn as shown in **Figure 7.4a**.

#### **7.4.1 NB Receive DBF Algorithm**

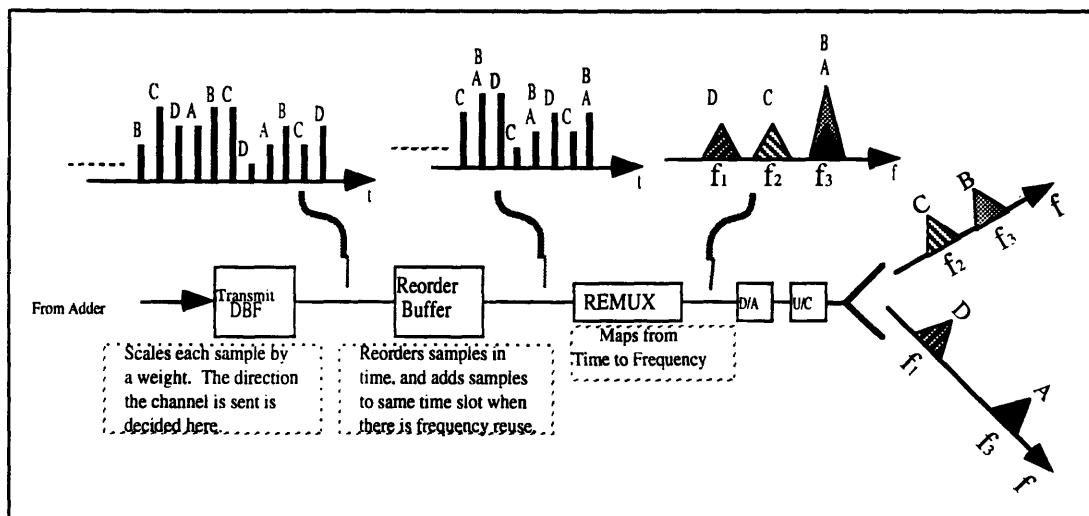
The incoming signal is down converted from RF to baseband, digitized, and then frequency de-multiplexed. The DEMUX module performs the frequency de-multiplexing that will yield a mapping from frequency to time (see **Figure 7.5c**). Given a signal composed of  $N$  channels as input, the output of the DEMUX module would yield the first sample from channel 1, the first sample from channel 2, . . . , the first sample from channel  $N$ , and then the second sample from channel 1, the second sample from channel 2, etc. Each sample is multiplied by its corresponding weight in the Receive DBF module, and then added to the samples from the rest of the antenna elements to complete the beam forming process. The output from the adder is a TDM sequence of the channels as described above, i.e., the channels interleaved in time. If one channel contains information from two signals (as is the case of frequency reuse) the Receive DBF module will generate two output samples from each input sample of that channel, with each output sample weighted according to the direction each signal is coming from (see **Figure 7.4b**).





**Figure 7.4b:** The Receive DBF process.

All incoming signals are mapped to a TDM by channel format via the DEMUX. The Receive DBF module replicates samples that contain information from more than one signal, and multiplies each sample by its correct weight.



**Figure 7.4c:** The Transmit DBF process.

Each sample is weighted by the Transmit DBF module to place the channel in the correct beam. The samples are then reordered in time as a preprocess to the REMUX, which maps the samples from time to its corresponding frequency.

#### 7.4.2 NB Transmit DBF Algorithm

On the transmit side, each DBF chip receives the same outgoing sample (in parallel) and multiplies it by the weight containing the correct phase delay

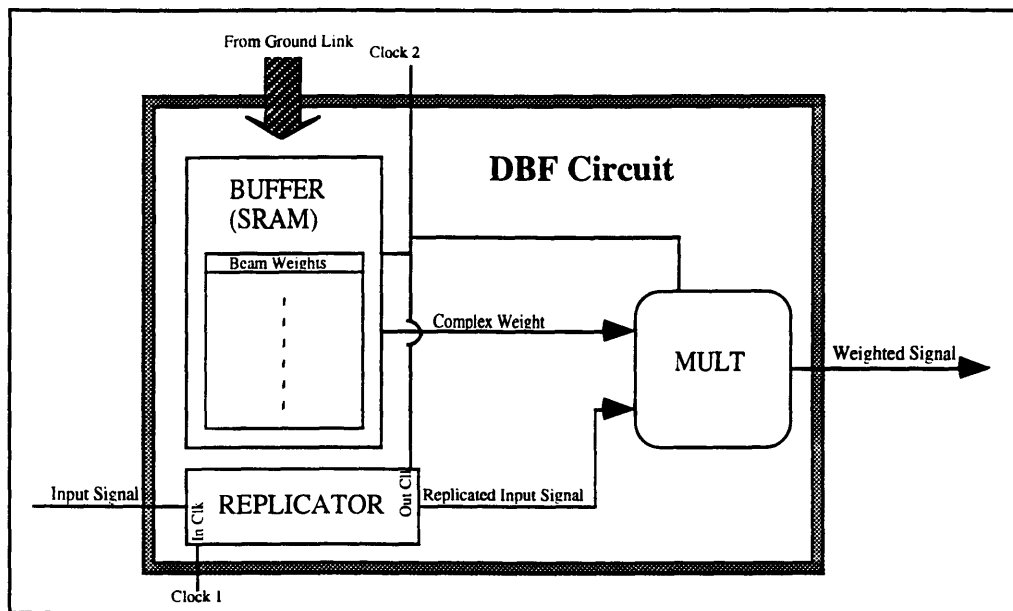
that will make the beam converge in a given direction. Hence, the beam (or direction) the channel is sent to is decided at the DBF chip. Once the samples are weighted, they are then reordered on the time axis (by the Reorder Buffer) as a setup for the REMUX module. The REMUX module maps the samples from time to frequency (the inverse of the DEMUX module). Thus, a reordering of the samples in time will allow the REMUX to map the samples to different frequency channels. Using this the DBF can re-configure the individual channel usage of the beams. Furthermore, by adding samples to the same time slot, the sum is mapped to the same frequency, giving the DBF the ability to support frequency reuse (see **Figure 7.4c**).

### **7.4.3 Design Appraisal**

The modularity of the system allows the DBF to be implemented as an extension to the antenna array. The signal for each element is processed separately, so there are no complicated interconnectivity problems. No switching network is required to re-route channels, since that operation is now done on a per-element basis. Since the processing is done per element, the size of the system is a function of the number of elements, but does not change with an increase in the number of beams. In situations where the number of beams transmitted varies drastically over a period of time, this is an ideal system to use. Having such a layout increases the reliability of the DBF, since no one path is critical to the forming of the beams. If one of the array elements fails, the resulting beam can still be formed from the remaining elements, with only a slight loss in signal strength. The one critical path following the adder can be backed up with redundant routes to separate adders. Since all the elements contain identical DBF chips, the uniformity of the system is preserved. Finally, using identical modules will

allow the chips to be mass produced, thus reducing the overall cost of the DBF.

There are some disadvantages to forming the beam at narrow band. The beams are formed in a time-division multiplexed (TDM) format. If the onboard processor includes functions such as signal compression and error correction, a TDM format may not be compatible with the standard compression and error correction algorithms. However, if a simple receive-transmit operation is desired, the NB DBF design is an excellent choice.



**Figure 7.5a:** The DBF module.

The DBF module contains a replicator to account for samples which contain information from more than one signal (as in the case of frequency reuse). Samples are then multiplied by complex weights corresponding to the shape and direction of the output beam.

## 7.5 DBF Components

The components of the NB DBF are quite simple. The DEMUX/REMUX modules are polyphase synthesizers that can be bought and

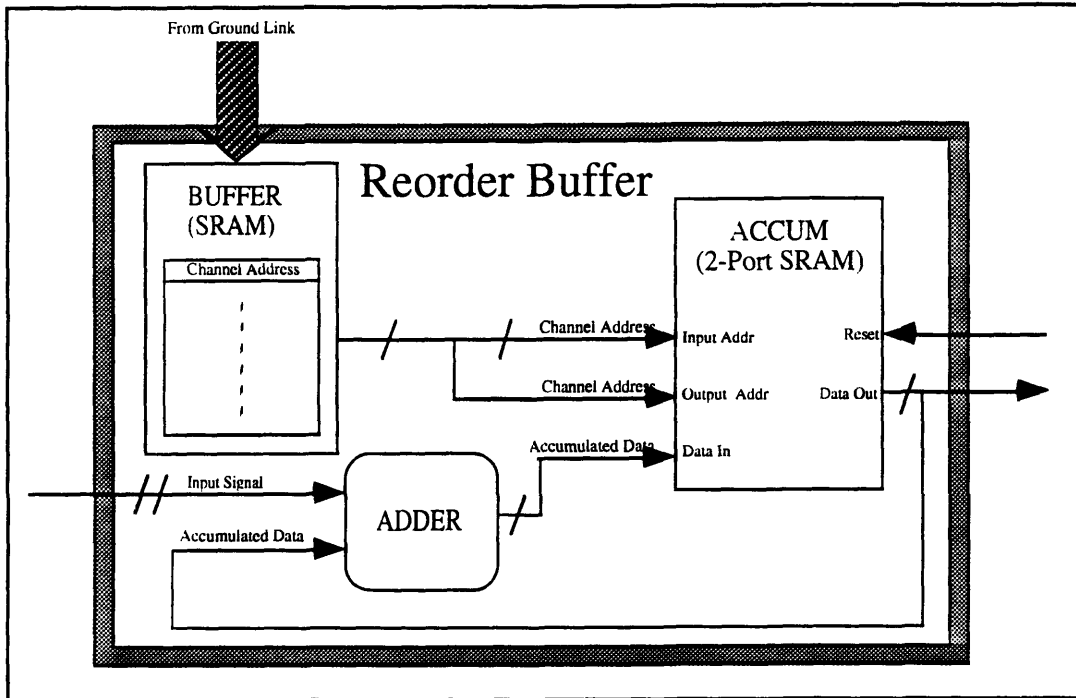
programmed to perform the desired FFT function. The layouts of the DBF and Reorder Buffer circuits are quite simple, and can be produced at low cost.

### 7.5.1 DBF Module

A possible layout of the DBF circuit is shown in **Figure 7.5a**. The Replicator is modeled as a FIFO queue where the input may be clocked at a different rate than the output (in the cases of frequency reuse). The complex weights are held in a random access memory buffer and used to scale incoming samples. There are a finite number of beams that will cover the entire scope of the earth (usually about 200, depending on the application). Thus all the weights can be stored in a small memory module and need not be computed onboard the satellite. Instead, this information can be uploaded from a ground satellite station.

### 7.5.2 Reorder Buffer

Since the DEMUX/REMUX modules provide the correct time-frequency mappings, the reorder buffer only needs to re-arrange the time sequence of the samples. This is done by writing the samples into a memory buffer in the desired order. An accumulator is required to support frequency re-use, since more than two samples may occupy the same time slot (and therefore share the same frequency after the REMUX). This is done by reading the target address of the buffer and adding the incoming sample to the already existing sum. Thus many samples can be summed into the same time slot to support frequency reuse (see **Figure 7.5b**).

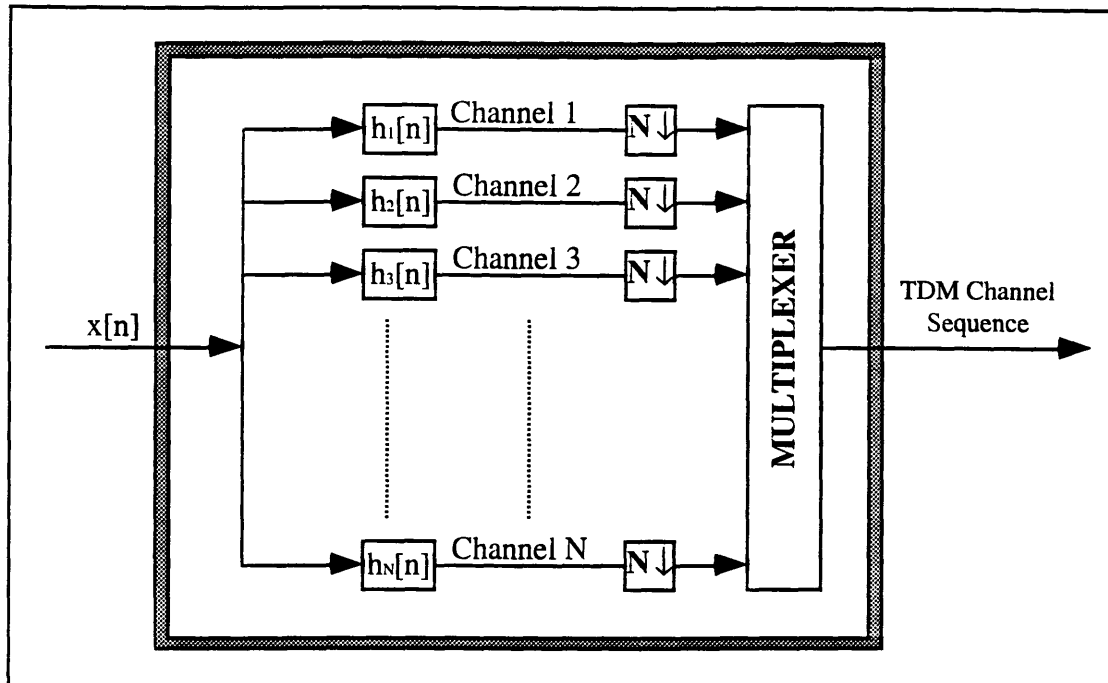


**Figure 7.5b:** The Reorder module.

The Reorder Buffer places samples in the desired order in its memory buffer, adding samples that are placed in the same time location. Samples that are added to the same time slot will share the same frequency.

### 7.5.3 DEMUX/REMUX

A *functional* block diagram of the DEMUX module is shown in **Figure 7.5c**. In reality, since all filters are equal in magnitude and differ only in phase, a polyphase synthesizer will perform the required filtering without having to branch off into  $N$  different parallel filtering paths. As described earlier, each extracted channel can be decimated by a factor of  $N$ , since the signal was sampled for  $N$  times the channel bandwidth. The REMUX does the opposite. It first interpolates and replicates (in parallel) the channels across the frequency spectrum, and then filters out the replicas that are in the desired frequency area, using the same polyphase filters.



**Figure 7.5c:** The DEMUX module.

The DEMUX filters out each channel and then interleaves the channels in time. The filtered signal can be decimated by  $N$  since its bandwidth is  $1/N$  times the bandwidth of  $x[n]$ .

## 8. SUMMARY AND CONCLUSIONS

### 8.1 Conclusions on Error Analysis

From the simulations performed using the ARRAY program, it is evident that 6 bits is a good choice for the complex coefficients given sidelobe level requirements of -25 dB. This yields an adequate bus size that will not make connectivity an unconquerable problem, but will maintain the accuracy needed to satisfy peak gain and frequency reuse requirements.

The number of bits used to quantize the incoming signal will depend largely on the application of the satellite. For instance, video signals require a

higher SNR than audio, since video signals require a sharp image, while the human ear is very good at filtering out background noise.

## **8.2 Design Summary**

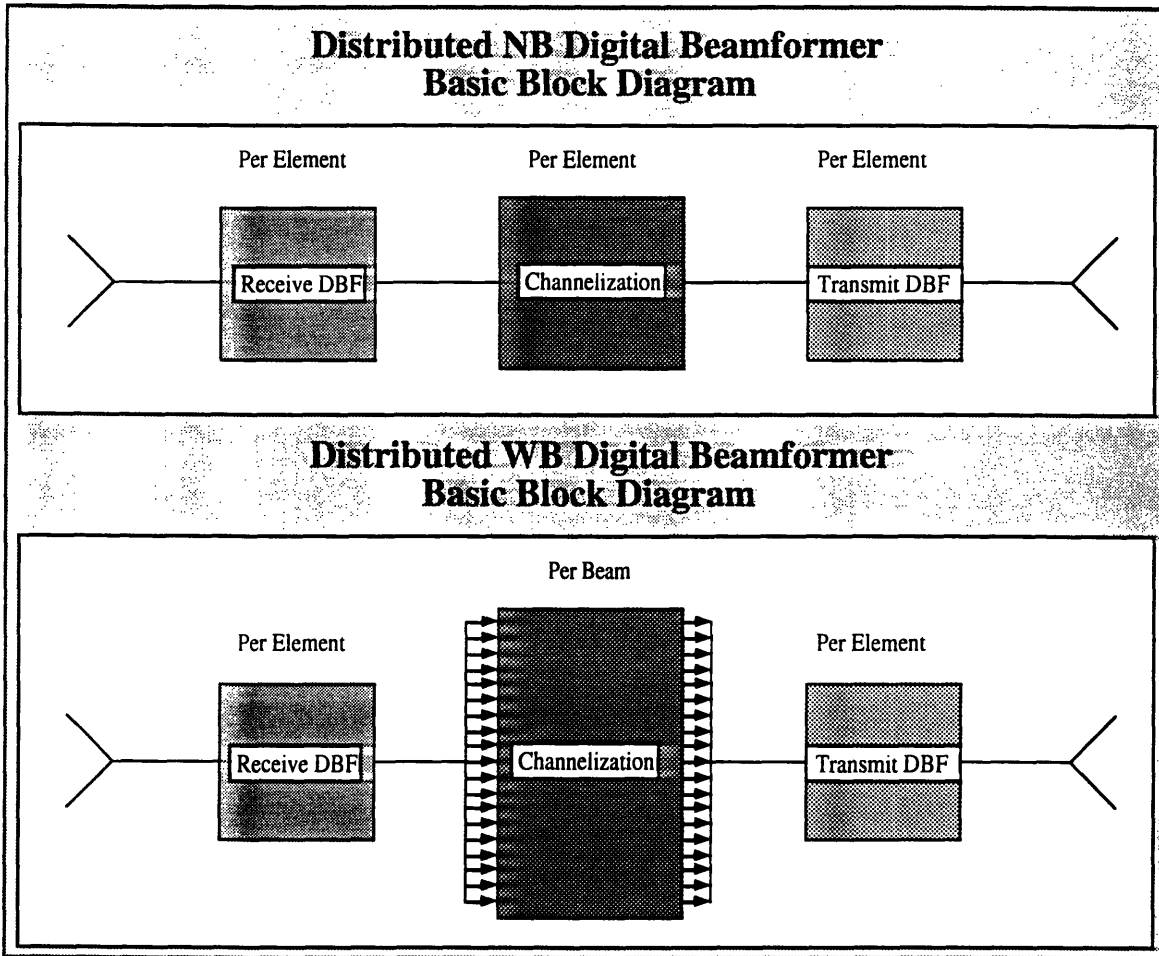
The following conclusions were drawn about the DBF designs.

### **8.2.1 Conclusions on Satellite Antenna Study**

The antenna type plays a major role in the design of the DBF. A WB DBF with a phased array antenna uses too much power, since it processes the *total* bandwidth of the beam over *all* the elements. Its circuitry also increases as a function of beams and elements regardless of the antenna type. The power consumption of the WB DBF with a focused antenna is about the same for the NB DBF with both phased arrays and reflector arrays. However, using a phased array antenna design with a NB DBF yields an inherent robustness not achieved with the alternate options.

### **8.2.2 Usage of NB/WB DBF**

The WB DBF may be preferable to the NB DBF when the satellite transmits a fixed number of beams with an outstandingly large number of antenna elements. The computational load increases less than the NB DBF as the number of elements grows, since some of the processing is done per beam instead of per element (see **Figure 8.2a**). Note that since all the array elements are used, these designs are relevant only to direct radiating phased array antennas, and are inadequate for a reflector antenna design that uses only a small number of the available elements to form a beam.



**Figure 8.2a:** Distributed NB DBF vs. Distributed WB DBF.  
 The NB DBF does all the processing per element, while the WB DBF does the beam forming per element and the channelization per beam.

If the number of beams transmitted varies over time, however, the NB DBF would be the appropriate choice. The NB DBF is modular, surmounts the interconnectivity issue, and eliminates the need for a switching network. Furthermore, the processing is done per element, which allows a variable number of beams to be processed without affecting the layout of the DBF.

### 8.3 Reliability of the DBF

The NB DBF has inherent redundancies that make it more robust than its WB counterpart. Since all the processing is done separately per element, a



failure in one array element does not affect the results from the other elements. In the WB DBF matrix shown in **Figure 1.2b** the signal coming from one element affects the entire system. There are no critical paths in the NB DBF save the adder, which can be backed up with other adders. Even then, adders are simple and robust circuits, and are less likely to fail than more complex modules such as the polyphase synthesizer.

#### **8.4 Suggestions for Improvements on DBF Design**

There are many alterations and additions that could improve the performance of the DBF, such as the inclusion of adaptive error correction modules. The most outstanding setback to DBFs is the technology needed to create processors that can multiply millions of complex numbers per second. Current CMOS technology, which is used in most instances, is usually limited to the order of tens of MHz, thus limiting the bandwidth of the beams transmitted by the DBF. Future implementations might include the usage of GaAs chips, which are able to perform multiply-accumulate operations on the order of hundreds of MHz. This would be ideal for INTELSAT applications, which require the support of bandwidths of up to 500 MHz.

## References

Acampora, A. S. "Digital Error Rate Performance of Active Phased Array Satellite Systems," IEEE Transactions on Antennas and Propagation. Vol AP-26, No. 6. Nov. 1987.

Barrett, M. & Coromina, F. "Development and Implementation of an Adaptive Digital Beamforming Network for Satellite Communication Systems," ERA Technology Ltd.

Beach, M. A., Swales, S. C., Bateman, A., Edwards, D. J., & McGeehan, J. P. "An Adaptive Antenna Array for Future Land Mobile Satellite Terminals," Univ. of Bristol, UK.

Devred, H. & Roger, J. "Experimental Digital Beam Forming Antenna," Thompson CSF-DRS.

Humbert, W. R. & Brandow, W. IV. "Error Analysis in a Practical Digital Beamformer," Rome Labs.

Pridham, Roger G. & Mucci, Ronald A. "A Novel Approach to Digital Beamforming," Ratheon Co., Submarine Division. 1977.

Robertson, W. & Pincock, D. "A New Architecture for Digital Time Domain Beamforming," IEEE Transactions on Acoustics, Speech, and Signal Processing. Vol ASSP-35, Dec. 1987.

Oppenheim, A.V. & Schafer, R.W. Discrete-Time Signal Processing. Prentice Hall, 1989.

Swartzlander, Earl E. Jr. "Signal Processor Design for Digital Beam Forming," TRW.

Zaghloul, A.I., Sorbello, R.M., & Assal, F.T. "Development of Active Phased Array for Reconfigurable Satellite Antennas," European Space Agency's COST 213/KUL Phased Array Workshop, Leuven, Belgium, October 26-27, 1988.

## **Appendix 1:** Glossary of Terms

A/D	Analog-to-Digital
ADC	Analog-to-Digital Converter
BFN	Beam Forming Network
CMOS	Complementary Metal-Oxide Semiconductor
DAC	Digital-to-Analog Converter
dB	Decibel
DBF	Digital Beamformer
D/C	Down Converter
FFT	Fast Fourier Transform
FDM	Frequency Division Multiplex
GaAs	Gallium Arsenide
GEO	Geostationary Earth Orbit
Inmarsat	International Maritime Satellite Organization
Intelsat	International Telecommunications Satellite Organization
I/Q	In-Phase/In-Quadrature
MAC	Multiply-Accumulate
NB	Narrow Band
RF	Radio Frequency
SAW	Surface Acoustic Wave
SNR	Signal-to-Noise Ratio
TDM	Time Division Multiplex
WB	Wide Band

## **Appendix 2: The ARRAY Program**

### **Usage of the ARRAY Program**

The ARRAY program, designed and implemented at COMSAT Laboratories, is a menu driven program intended for use as a tool to facilitate satellite antenna design. It takes as input a file that contains descriptive information of the modeled antenna (such as number of elements, geometry, complex weighting of the individual elements, etc). The input file can be created interactively from the ARRAY program, which prompts the user for all necessary values. The ARRAY program can then form a beam using the antenna element weights, and plot the resulting waveform on a pre-selected graphical interface window.

### **Sample ARRAY Input File**

```
1 90 90 60.00 120.00 60.00 120.00
5
1.500 .140 .000 1.000 .000 1.000 90.000
61
10.39200 6.00000 .00000
10.39200 3.00000 .00000
10.39200 .00000 .00000
10.39200 -3.00000 .00000
10.39200 -6.00000 .00000
7.79400 7.50000 .00000
7.79400 +.50000 .00000
7.79400 1.50000 .00000
7.79400 -1.50000 .00000
7.79400 -4.50000 .00000
7.79400 -7.50000 .00000
5.19600 9.00000 .00000
5.19600 6.00000 .00000
5.19600 3.00000 .00000
5.19600 .00000 .00000
5.19600 -3.00000 .00000
5.19600 -6.00000 .00000
5.19600 -9.00000 .00000
2.59800 10.50000 .00000
2.59800 7.50000 .00000
2.59800 4.50000 .00000
2.59800 1.50000 .00000
2.59800 -1.50000 .00000
2.59800 -4.50000 .00000
2.59800 -7.50000 .00000
2.59800 -10.50000 .00000
.00000 12.00000 .00000
```

.00000	9.00000	.00000
.00000	6.00000	.00000
.00000	3.00000	.00000
.00000	.00000	.00000
.00000	-3.00000	.00000
.00000	-6.00000	.00000
.00000	-9.00000	.00000
.00000	-12.00000	.00000
-2.59800	10.50000	.00000
-2.59800	7.50000	.00000
-2.59800	4.50000	.00000
-2.59800	1.50000	.00000
-2.59800	-1.50000	.00000
-2.59800	-4.50000	.00000
-2.59800	-7.50000	.00000
-2.59800	-10.50000	.00000
-5.19600	9.00000	.00000
-5.19600	6.00000	.00000
-5.19600	3.00000	.00000
-5.19600	.00000	.00000
-5.19600	-3.00000	.00000
-5.19600	-6.00000	.00000
-5.19600	-9.00000	.00000
-7.79400	7.50000	.00000
-7.79400	4.50000	.00000
-7.79400	1.50000	.00000
-7.79400	-1.50000	.00000
-7.79400	-4.50000	.00000
-7.79400	-7.50000	.00000
-10.39200	6.00000	.00000
-10.39200	3.00000	.00000
-10.39200	.00000	.00000
-10.39200	-3.00000	.00000
-10.39200	-6.00000	.00000
.02607	45.58	
.02607	139.35	
.02607	-126.88	
.02607	-33.11	
.02607	60.66	
.02607	-149.59	
.09432	-55.82	
.09432	37.95	
.09432	131.72	
.09432	-134.51	
.02607	-40.74	
.02607	15.25	
.09432	109.02	
.17869	-157.21	
.17869	-63.44	
.17869	30.33	
.09432	124.10	
.02607	-142.13	
.02607	-179.92	
.09432	-86.15	
.17869	7.62	
.24694	101.39	
.24694	-164.84	
.17869	-71.07	

.09432	22.70
.02607	116.47
.02607	-15.08
.09432	78.69
.17869	172.46
.24694	-93.77
.27301	.00
.24694	93.77
.17869	-172.46
.09432	-78.69
.02607	15.08
.02607	-116.47
.09432	-22.70
.17869	71.07
.24694	164.84
.24694	-101.39
.17869	-7.62
.09432	86.15
.02607	179.92
.02607	142.13
.09432	-124.10
.17869	-30.33
.17869	63.44
.17869	157.21
.09432	-109.02
.02607	-15.25
.02607	40.74
.09432	134.51
.09432	-131.72
.09432	-37.95
.09432	55.82
.02607	149.59
.02607	-60.66
.02607	33.11
.02607	126.88
.02607	-139.35
.02607	-45.58

### Appendix 3: DBF Toolkit Source Code

```

/*****
** FILE: antenna.c
** AUTHOR: Sherk Chung
** DATE: 7/23/93
** LOCATION: COMSAT Labs, Clarksburg, MD
**
** LAST MODIFIED:
** MODIFICATIONS:
**
** COMMENTS: functions used to calculate
**           beamformer weights
**
*****/

#include "headers.h"

Number Wavelength (float freq)
{
    return ((Number) (10e-3*C/freq)); /* Mult by 10e-3 since freq is in MHz */
    /* Wavelength in Um           */
}

BeamArrayPtr CalculateWeights (int beamnum, float **beamangles,
                               float *beamfreqs)
{
    BeamArrayPtr cap;
    int k, i, j;
    Number lambda, theta_x, theta_y, phase;
    ComplexPtr weight = CN_NewP(0,0);

    fprintf (stderr, "Calculating Weights .");
    cap = BA_New (beamnum, ANTENNA_ELMS);

    for (k = 0; k < beamnum; k++) {
        lambda = Wavelength (beamfreqs[k]);
        theta_x = beamangles[k][0];
        theta_y = beamangles[k][1];

        /* Set Phases horizontally (x-direction)
        -----*/
        for (j = 0; j < ANTENNA_WIDTH; j++) {
            phase = (Number) (2*pi*j*sin(theta_x)/lambda);
            for (i = 0; i < ANTENNA_HEIGHT; i++) {
                BA_SetNum (cap, k, i*ANTENNA_WIDTH+j, 1, phase);
            }
        }
    }
}

```

```

fprintf (stderr, ".");

/* Set Phases vertically (y-direction)
-----*/
for (i = 0; i < ANTENNA_HEIGHT; i++) {
    phase = (Number) (2*pi*i*sin(theta_x)/lambda);
    for (j = 0; j < ANTENNA_WIDTH; j++) {
        CN_SetNum(weight,1,phase);
        CN_Mult2 (BA_Num(cap,k,i*ANTENNA_WIDTH+j), weight);
    }
}
}
}
fprintf (stderr, "DONE\n\n");
return (cap);
}

```



```

/*****
** FILE: array.c
** AUTHOR: Sherk Chung
** DATE: 6/29/93
** LOCATION: COMSAT Labs, Clarksburg, MD
**
** LAST MODIFIED:
** MODIFICATIONS:
*
** COMMENTS: support for array structure
**
*****/

#include "headers.h"

/* -----( Complex Arrays )-----*/

ComplexArrayPtr CA_New (int size)
{
    int i;
    ComplexArrayPtr cap = malloc (sizeof (ComplexArray));
    ComplexPtr *ca = calloc (size, sizeof (ComplexPtr));
    for (i = 0; i < size; i++) ca[i] = CN_NewP (0, 0);
    ELMS(cap) = size;
    CURR(cap) = 0;
    DATA(cap) = ca;
    return (cap);
}

void CA_Free (ComplexArrayPtr cap)
{
    int i;
    for (i = 0; i < ELMS(cap); i++) CN_Free (DATA(cap)[i]);
    free (cap);
}

```

```

/*****
** FILE: beam.c
** AUTHOR: Sherk Chung
** DATE: 6/29/93
** LOCATION: COMSAT Labs, Clarksburg, MD
**
** LAST MODIFIED:
** MODIFICATIONS:
**
** COMMENTS: support for beam array structure
**
**
*****/

#include "headers.h"

/* -----( Beam Arrays )-----*/

BeamArrayPtr BA_New(int length, int signal_length)
{
    int i;
    BeamArrayPtr ba = malloc (sizeof (BeamArray));
    BeamPtr *bp = calloc (length, sizeof (BeamPtr));
    for (i = 0; i < length; i++) bp[i] = Beam_New(signal_length);
    ELMS(ba) = length;
    CURR(ba) = 0;
    DATA(ba) = bp;
    return (ba);
}

void BA_Free (BeamArrayPtr bap)
{
    int i;
    for (i = 0; i < ELMS(bap); i++) Beam_Free (DATA(bap)[i]);
    free (bap);
}

void BA_Scale (BeamArrayPtr bap, float num)
{
    int i, j, nr;
    BeamPtr bp;

    for (i = 0; i < ELMS(bap); i++) {
        bp = DATA(bap)[i];
        for (j = 0; j < ELMS(bp); j++) {
            nr = (int) ((float) MAG(DATA(bp)[j]) * num);
            Beam_SetNum (bp, j, nr, PHASE(DATA(bp)[j]));
        }
    }
}

```

```

}
}

```

```

BeamArrayPtr BA_Transpose (BeamArrayPtr bap)
{
    int i, j;
    ComplexPtr tmp;
    Number rl, im;
    BeamArrayPtr newbap = BA_New (BA_WIDTH(bap), BA_HEIGHT(bap));

    for (i = 0; i < BA_HEIGHT(bap); i++) {
        fprintf (stderr, ".");
        for (j = 0; j < BA_WIDTH(bap); j++) {
            rl = MAG(DATA(DATA(bap)[i])[j]);
            im = PHASE(DATA(DATA(bap)[i])[j]);
            BA_SetNum (newbap, j, i, rl, im);
        }
    }
    return (newbap);
}

```

```

BeamArrayPtr BA_Mult (BeamArrayPtr bap1, BeamArrayPtr bap2)
{
    int i,j,k,m, width, height;
    ComplexPtr num = CN_NewP (0,0);
    BeamArrayPtr prod = BA_New (BA_HEIGHT(bap1), BA_WIDTH(bap2));

    if ((width = BA_WIDTH(bap1)) != (height = BA_HEIGHT(bap2)))
        ERROR("Incompatible BeamArray Sizes","BeamArrayPtr");

    for (i = 0; i < BA_HEIGHT(bap1); i++) {
        fprintf (stderr, ".");
        for (k = 0; k < BA_WIDTH(bap2); k++) {
            CN_SetNum (num, 0,0);
            for (j = 0; j < BA_WIDTH(bap1); j++) {
                CN_Add2 (num, CN_Mult (DATA(DATA(bap1)[i])[j],
                    DATA(DATA(bap2)[j])[k]));
            }
            if (MAG(num) >= 255) ERROR ("Overflow", "BA_Mult");
        }
    }
    return (prod);
}

```

```

/*****
** FILE: beamformer.c
** AUTHOR: Sherk Chung
** DATE: 6/29/93
** LOCATION: COMSAT Labs, Clarksburg, MD
**
** LAST MODIFIED:
** MODIFICATIONS:
**
** COMMENTS: main program used to simulate a
**           digital beamformer.
**
*****/

#include "headers.h"

void ERROR (char *mesg, char* loc)
{
    fprintf (stderr, " *****< E R R O R >***** \n %s in function %s\n\007",
            mesg, loc);
    exit (1);
}

BeamArrayPtr BeamForm (BeamArrayPtr beams, BeamArrayPtr weights)
{
    int i,j;
    BeamArrayPtr outbeams, beamatrix;

    fprintf (stderr, "Forming BEAM .");
    beamatrix = BA_Transpose (beams);
    outbeams = BA_Mult (beamatrix, weights);
    fprintf (stderr, ".DONE\n\n");
    return (outbeams);
}

void main(int argc, char *argv[])
{
    BeamArrayPtr beams;      /* Input Beams          */
    BeamArrayPtr outbeams;   /* Radiating Element signals */
    BeamArrayPtr weights;    /* Weights to direct antenna */
    float **beamangles, *beamfreqs;
    int i, j, beamnum;
    char *outfile, *beamdir;

    if (argc != 4) {
        fprintf (stderr, " USAGE: beamformer <beams directory> <#beams> <outfile>\n");
        exit (0);
    }

    /* Parse Arguments

```

```

-----*/
beamdir = strdup (argv[1]);
beamnum = atoi (argv[2]);
outfile = strdup (argv[3]);

beamfreqs = calloc (beamnum, sizeof (float));
beamangles = calloc (beamnum, sizeof (float *));
for (i = 0; i < beamnum; i++) beamangles[i] = calloc (2, sizeof (float));

/* Read Input Files
-----*/
beams = ReadInputBeams (beamdir, beamnum, beamangles, beamfreqs);

/* Do BeamForming
-----*/
weights = CalculateWeights (beamnum, beamangles, beamfreqs);
outbeams = BeamForm (beams, weights);

/* Write Output File
-----*/
WriteOutputSignal (outbeams, outfile);
}

```

```

/*****
** FILE: complex.c
** AUTHOR: Sherk Chung
** DATE: 6/29/93
** LOCATION: COMSAT Labs, Clarksburg, MD
**
** LAST MODIFIED:
** MODIFICATIONS:
**
** COMMENTS: Routines to support struct complex numbers
**           defined in complex.h
**
*****/

```

```
#include "headers.h"
```

```
ComplexPtr CN_NewR (Number mag, Number phase)
```

```
{
  ComplexPtr num3 = malloc(sizeof (Complex));
  REAL(num3) = mag;
  IMAG(num3) = phase;
  num3->rect = TRUE;
  num3->polar = FALSE;
  return (num3);
}
```

```
ComplexPtr CN_NewP (Number mag, Number phase)
```

```
{
  ComplexPtr num3 = malloc(sizeof (Complex));
  MAG(num3) = mag;
  PHASE(num3) = phase;
  num3->polar = TRUE;
  num3->rect = FALSE;
  return (num3);
}
```

```
ComplexPtr CN_Copy (ComplexPtr cp)
```

```
{
  if (cp->polar) return (CN_NewP (MAG(cp), PHASE(cp)));
  else if (cp->rect) return (CN_NewR (REAL(cp), IMAG(cp)));
  else ERROR ("Unknown Type", "CN_Copy");
}
```

```
ComplexPtr CN_Polar2Rect (ComplexPtr n)
```

```
{
  return (CN_NewR(MAG(n)*cos(PHASE(n)), MAG(n)*sin(PHASE(n))));
}
```

```

ComplexPtr CN_Rect2Polar (ComplexPtr n)
{
    return (CN_NewP(sqrt(REAL(n) * REAL(n) + IMAG(n) * IMAG (n)),
                    atan(IMAG(n) / REAL(n))));
}

```

```

ComplexPtr CN_Mult (ComplexPtr num1, ComplexPtr num2)
{
    ComplexPtr num3;

    if (num1->rect && num2->rect) {
        num3 = CN_NewR ((REAL(num1) * REAL(num2)) - (IMAG(num1) * IMAG(num2)),
                        (REAL(num1) * IMAG(num2)) + (IMAG(num1) * REAL(num2)));
    } else if (num1->polar && num2->polar) {
        num3 = CN_NewP (MAG(num1) * MAG(num2), PHASE(num1) + PHASE(num2));
    } else {
        ERROR ("Incompatible types", "CN_Mult");
    }
    return (num3);
}

```

```

ComplexPtr CN_Mult2 (ComplexPtr num1, ComplexPtr num2)
{
    if (num1->rect && num2->rect) {
        CN_SetNum(num1, (REAL(num1) * REAL(num2)) - (IMAG(num1) * IMAG(num2)),
                    (REAL(num1) * IMAG(num2)) + (IMAG(num1) * REAL(num2)));
    } else if (num1->polar && num2->polar) {
        CN_SetNum(num1, MAG(num1) * MAG(num2), PHASE(num1) + PHASE(num2));
    } else {
        ERROR ("Incompatible types", "CN_Mult2");
    }
    return (num1);
}

```

```

ComplexPtr CN_Add (ComplexPtr num1, ComplexPtr num2)
{
    ComplexPtr num3;
    ComplexPtr num4, num5;

    if (num1->rect && num2->rect) {
        num3 = CN_NewR (REAL(num1) + REAL(num2), IMAG(num1) + IMAG(num2));
    } else if (num1->polar && num2->polar) {
        num4 = CN_Polar2Rect (num1);
        num5 = CN_Polar2Rect (num2);
        num3 = CN_Rect2Polar (CN_Add (num4, num5));
    } else {
        ERROR ("Adding Diff Types", "CN_Add");
    }
    return (num3);
}

```

```

ComplexPtr CN_Add2 (ComplexPtr num1, ComplexPtr num2)
{
    ComplexPtr num3;
    ComplexPtr num4, num5;

    if (num1->rect && num2->rect) {
        CN_SetNum (num1, REAL(num1) + REAL(num2), IMAG(num1) + IMAG(num2));
    } else if (num1->polar && num2->polar) {
        num4 = CN_Polar2Rect (num1);
        num5 = CN_Polar2Rect (num2);
        num3 = CN_Rect2Polar (CN_Add (num4, num5));
        CN_Free(num3);
        CN_Free(num4);
        CN_Free(num5);
        CN_SetNum(num1, REAL(num3), IMAG(num3));
    } else {
        ERROR ("Adding Diff Types", "CN_Add2");
    }
    return (num1);
}

```

```

ComplexPtr CN_Expt (ComplexPtr num1, Number k)
{
    ComplexPtr num, num2;

    if (num1->rect) {
        num2 = CN_Rect2Polar (num1);
        PHASE(num2) *= k;
        num = CN_Polar2Rect (num2);
    } else if (num1->polar) {
        num = CN_Copy (num1);
        PHASE(num) *= k;
    }
    return (num);
}

```

```

void CN_Free (ComplexPtr cp)
{
    free (cp);
}

```



```
/* File: control.c
   Used to generate signals
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define C 1.0
#define C1 5.0

float f(int i)
{
    float x = (float) i;
    return (1.0 - (exp(-x/C1))*(sin(x/C) + cos(x/C)));
}

void main ()
{
    int i;

    for (i = 0; i < 100; i++)
        printf ("%f\n", f(i));
}
```

```

/*****
** FILE: dft.c
** AUTHOR: Sherk Chung
** DATE: 6/29/93
** LOCATION: COMSAT Labs, Clarksburg, MD
**
** LAST MODIFIED:
** MODIFICATIONS:
**
** COMMENTS: fft function
**
*****/

#include <stdlib.h>
#include <math.h>
#include "beamformer.h"
#include "array.h"

BeamPtr dft (BeamPtr bp)
{
    int k, n;
    ComplexPtr *x, *X;
    BeamPtr newcp = (BeamPtr) Beam_New(ELMS(bp));
    ComplexPtr W = (Complex *) CN_NewP (1, -2*pi/ELMS(bp));

    X = DATA(newcp); x = DATA(bp);
    for (k = 0; k < ELMS(bp); k++) {
        for (n = 0; n < ELMS(bp); n++) {
            X[k] += CN_Mult (x[n], CN_Expt (W, k*n));
        }
    }
}

/*
** FUNCTION: Massive Dft
** USAGE:   NewCmplxArray = MassiveDft (ComplexArrayPtr ComplexArray)
** PURPOSE: Perform DFT of an array of beams,
**          each beam of length ELMS(cp)
*/
BeamArrayPtr MassiveDft (BeamArrayPtr cp)
{
    int k, n, i, j;
    ComplexPtr *X;
    BeamArrayPtr newcp = (BeamArrayPtr) BA_New (ELMS(cp), 0);

    for (i = 0; i < ELMS(cp); i++) {
        DATA(newcp)[i] = dft (DATA(cp)[i]);
    }
    return (newcp);
}

```

```

/*****
*****
** FILE: error.c
** AUTHOR: Sherk Chung
** DATE: 8/25/93
** LOCATION: COMSAT Labs, Clarksburg, MD
**
** LAST MODIFIED:
** MODIFICATIONS:
**
** COMMENTS: Yields angular error beamformer due to phase quantization
**           for given quant level, signal wavelength, distance between
**           antenna array elements, and scan angle of beam.
**
** USAGE: error <quant bits> <lambda> <dist> <scan angle>
** WHERE: <quant bits> = number of bits to use for phase quantization
**        <lambda>    = wavelength of signal
**        <dist>     = distance between antenna array elements
**        <scan angle> = beam direction relative to boresight
**
*****
*****/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define pi 3.14159265
#define R2D(n) n*180.0/pi
#define DEBUG 1

float quantize (int qllevel, float num)
/*
** FUNCTION: quantize
** ARGUMENTS: qllevel = number of bits to quantize to
**            num      = number to quantize
**
** PURPOSE: To quantize a number to a certain number of bits
*/
{
    float maxout, minout, n2, n1, n;

    /* Find Range of Qlevel Bits
    -----*/
    minout = 1.0;
    maxout = pow(2.0, (float) qllevel);

    /* Linear Interpolate from 0-360 to minout-maxout, and truncate
    to integer value. Then interpolate back to 0-360 scale.
    -----*/
    n = ((num*(maxout-minout))/(360.0)+minout); /* Interpolate */
    n1 = (int) n; /* Round Off to */
}

```

```

if ((n - (float) n1) > 0.5) n1++;          /* Nearest Int */
n2 = (((float) n1-minout)*360.0)/(maxout-minout); /* Interpolate Back */

return (n2);
}

/*------( Main Program )-----*/

void main (int argc, char *argv[])
{
int qllevel;
float error, m, denom, dist, lambda, theta, qtheta, qphase, phase;

if (argc != 5) {
    fprintf(stderr, "USAGE: error <quant bits> <lambda> <dist> <scan angle>\n");
    exit (0);
}
/* Parse Input
-----*/
qllevel = atoi (argv[1]); /* Quantization Level */
lambda = atof (argv[2]); /* Wavelength of signal */
dist = atof (argv[3]); /* Dist between Antenna Elms */
theta = atof (argv[4]); /* Scan angle */

/* Convert to radians
-----*/
theta = (theta*pi)/180.0;

/* Find Correct Phase
-----*/
phase = (dist*2.0*pi*sin(theta))/lambda;

/* Find quantized phase
-----*/
qphase = quantize (qllevel, phase);
if (DEBUG) fprintf (stderr, "Phase1: %f\nPhase2: %f\n", phase, qphase);

/* Find Angle Corresponding to Quantized Phase
-----*/
qtheta = asin ((qphase*lambda) / (dist*2.0*pi));
if (DEBUG) fprintf (stderr, "Theta1: %f\nTheta2: %f\n", theta, qtheta);

/* Find and Print Difference Between Real Angle and Angle Obtained
from Quantized Phase.
-----*/
error = R2D((theta - qtheta));
printf ("\nWith %d bits, %f signal wavelength, and %f Element spacing, \n the error is:
%f degrees \n", qllevel, lambda, dist, error);
}

```

```

/*****
** FILE: io.c
** AUTHOR: Sherk Chung
** DATE: 6/29/93
** LOCATION: COMSAT Labs, Clarksburg, MD
**
** LAST MODIFIED:
** MODIFICATIONS:
**
** COMMENTS: I/O functions used in beamformer.c
**
*****/

#include "headers.h"

FILE *OpenBeamFile (char *beamdir, char *ext)
{
    char name[40], beamname[80];
    FILE *beamfile;

    strcpy (name, "/beam.");
    strcpy (beamname, beamdir);
    strcat (beamname, name);
    strcat (beamname, ext);
    beamfile = fopen (beamname, "r");
    if (beamfile == NULL) {
        strcat (beamname, ": FILE not Found");
        ERROR (beamname, "ReadInputBeams");
    }
    return (beamfile);
}

BeamArrayPtr ReadInputBeams (char *beamdir, int beamnum, float **beamangles,
                             float *beamfreqs)
{
    int i, j, num;
    FILE *beamfile;
    char bnum[4];
    BeamArrayPtr bap;

    fprintf (stderr, "Reading Input Beams . . .\n"); /**/

    bap = BA_New (beamnum, SIGNAL_LENGTH);

    for (i = 0; i < beamnum; i++) {
        sprintf (bnum, "%d", i+1);
        fprintf (stderr, " Reading Beam #%s\r", bnum);

        /* Open File
        -----*/

```

```

beamfile = OpenBeamFile (beamdir, bnum);

/* Read Frequency
-----*/
fscanf (beamfile, "%f", &(beamfreqs[i]));

/* Read Direction
-----*/
fscanf (beamfile, "%f", &(beamangles[i][0]));
fscanf (beamfile, "%f", &(beamangles[i][1]));

/* Read data from File
-----*/
for (j = 0; j < SIGNAL_LENGTH; j++) {
    fscanf (beamfile, "%d", &num);
    BA_SetNum (bap, i, j, num, 0);
}
fclose (beamfile);
}
fprintf (stderr, " DONE Reading Beams  \n\n");

return (bap);
}

float *ReadInputAngles (char *beamdir, int beamnum)
{
    int i;
    float angle, *alist;
    char *ext = strdup ("angles");
    FILE *anglefile;

    alist = calloc (beamnum, sizeof (float));
    anglefile = OpenBeamFile (beamdir, ext);

    for (i = 0; i < beamnum; i++) {
        fscanf (anglefile, "%f", &(alist[i]));
    }
    return (alist);
}

void WriteOutputSignal (BeamArrayPtr outbeams, char *outfile)
{
    int i, j;
    FILE *fp;

    fprintf (stderr, "Writing Output to %s .", outfile);
    fp = fopen (outfile, "w");

    for (i = 0; i < ELMS(outbeams); i++)
        fprintf (fp, "Beam #%d%d", 0, i);

    for (j = 0; j < SIGNAL_LENGTH; j++) {

```

```
for (i = 0; i = ELMS(outbeams): i++) {
  if (j == 0) fprintf (stderr, ".");
  fprintf (fp, "%d %d ",
          MAG(BA_Num(outbeams.i,j)),
          PHASE(BA_Num(outbeams.i,j)));
}
}
fprintf (stderr, ".DONE\n\n");
}
/**/
```

```
/* File: makeant.c
   Makes list of floats. Used for testing.
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void main()
{
    FILE *fp;
    int i, j;

    fp = fopen("ant", "w");

    for (i = 0; i < 8; i++) {
        for (j = 0; j < 8; j++) {
            fprintf (fp, "%f. ", sqrt((double) i*i + j*j));
        }
        fprintf (fp, "\n");
    }
    fclose (fp);
}
```



```

/*****
*****
** FILE: maxerror.c
** AUTHOR: Sherk Chung
** DATE: 8/23/93
** LOCATION: COMSAT Labs, Clarksburg, MD
**
** LAST MODIFIED:
** MODIFICATIONS:
**
** COMMENTS: Yields maximum angular error beamformer due to phase quantization
**           for given quant level, signal wavelength, and distance between
**           antenna array elements.
**
** USAGE: maxerror <quant bits> <lambda> <dist>
** WHERE: <quant bits> = number of bits to use for phase quantization
**        <lambda>     = wavelength of signal
**        <dist>      = distance between antenna array elements
**
*****
*****/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define pi 3.14159265
#define R2D(n) n*180.0/pi
#define DEBUG 0

void main (int argc, char *argv[])
{
    int qllevel;
    float error, m, denom, dist, lambda, theta, qtheta, qphase, phase;

    if (argc != 4) {
        fprintf (stderr, "USAGE: maxerror <quant bits> <lambda> <dist>\n");
        exit (0);
    }
    /* Parse Input
    -----*/
    qllevel = atoi (argv[1]); /* Quantization Level */
    lambda = atof (argv[2]); /* Wavelength of signal */
    dist = atof (argv[3]); /* Dist between Antenna Elms */
    theta = 0;

    /* Set phase = 0 and qphase to be next step size to find
    max error level */

    m = 360.0 / pow(2.0, (float) qllevel); /* Step size */
    phase = 0;
    qphase = m/2.0; /* Max error occurs midway between step sizes */

```

```
if (DEBUG) fprintf (stderr, "Phase1: %f\nPhase2: %f\n", phase. qphase);

qtheta = asin ((qphase*lambda) / (dist*2.0*pi));
if (DEBUG) fprintf (stderr, "Theta1: %f\nTheta2: %f\n", theta. qtheta);

error = R2D((qtheta - theta));
printf ("\nWith %d bits, %f signal wavelength, and %f Element spacing, \n the
MAXIMUM error is: %f degrees \n",qlevel, lambda, dist, error);
}
```

```

/* File: makesignal.c
   Generates a discrete signal into a file. Used for testing.
*/

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "beamformer.h"

main (int argc, char *argv[])
{
    int i;
    float num, x, j, deg, hsl;
    FILE *fp;

    if (argc != 3) {
        fprintf (stderr, " USAGE: makesignal <num> <filename>\n");
        exit (0);
    }

    num = atof (argv[1]);
    fp = fopen(argv[2], "w");
    fprintf (fp, "%f ", (float) num);    /* Frequency */
    fprintf (fp, "%f ", (pi/num)*0.5);  /* X direction */
    fprintf (fp, "%f ", (pi/num)*0.2);  /* Y direction */

    hsl = SIGNAL_LENGTH/5.0;
    for (i = 0; i < SIGNAL_LENGTH; i++) {
        j = ((float)i)/hsl;
        deg = j*j*j;
        while (deg > 360.0) deg -= 360.0;
        x = sin((float)num*j);
        x = x*127.5 + 127.5; /**/
        fprintf (fp, "%f %f\n", x, deg);
    }
    fclose (fp);
}

```

```

/*****
****
** FILE: quant.c
** AUTHOR: Sherk Chung
** DATE: 8/19/93
** LOCATION: COMSAT Labs, Clarksburg, MD
**
** LAST MODIFIED:
** MODIFICATIONS:
**
** COMMENTS: Quantizes Complex Coefficients of ARRAY Input Files
**
*****/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAX_NORM 1.0 /* Output Normalized from MIM_NORM to MAX_NORM */
#define MIN_NORM 0.0
#ifndef NUM
#define NUM 1000
#endif

void main (int argc, char *argv[])
{
    int i,j, retval, count, verbose = 0;
    int qllevel, amp, ph1;
    char *infile, *outfile, line[160];
    float amps[NUM], phase1[NUM], maxamp, minamp, maxout, minout;
    FILE *infp, *outfp;

    if (argc < 4 || argc > 5) {
        fprintf (stderr, "USAGE: quant <qllevel> <infile> <outfile>\n");
        exit (0);
    }
    /* Parse Input
    -----*/
    qllevel = atoi (argv[1]); /* Quantization Level */
    infile = argv[2]; /* Input File */
    outfile = argv[3]; /* Output File */
    if (argc == 5) /* Set VERBOSE to ON */
        if (!strcmp(argv[4], "-v")) verbose = 1;

    /* Read Input File
    -----*/
    infp = fopen (infile, "r");
    if (infp == NULL) {
        fprintf (stderr, "FILE: %s NOT FOUND\n", infile);
    }
    outfp = fopen (outfile, "w");
    if (outfp == NULL) {

```

```

    fprintf (stderr, "FILE: %s CANNOT BE OPENED\n", outfile);
}

retval = 1;
maxamp = 0.0;
minamp = 1000000.0;

if (verbose) printf ("Reading Input File .");

/* Copy 1st 4 lines (antenna description)
-----*/
fputs (fgets (line, 160, infp), outfp);
fputs (fgets (line, 160, infp), outfp);
fputs (fgets (line, 160, infp), outfp);
fputs (fgets (line, 160, infp), outfp); /* This is # of Array Elms */
count = atoi (line);

/* Copy next 'count' lines */
for (i = 0; i < count; i++)
    fputs (fgets (line, 160, infp), outfp);

for (i = 0; i < count; i++) {
    retval = fscanf (infp, "%f %f", &(amps[i]),
                    &(phase1[i]));
    if (retval == EOF)
        fprintf (stderr, "Unexpected EOF encountered in %d\n", infile);
    if (phase1[i] < 0.0) phase1[i] += 360.0;
    if (verbose) printf (".");
}

maxout = pow(2.0, (float) qllevel);
minout = 0.0;
maxamp = 1.0;
minamp = 0.0;

if (verbose) {
    printf ("DONE.\n\nQuant RANGE: %f ---> %f\n", minout, maxout);
    printf ("Amplitude RANGE: %f ---> %f\n", minamp, maxamp);
    printf ("\nQuantizing Values .");
}
for (i = 0; i < count; i++) {
    /* Normalize amps and phases to range from 1 to 2^qllevel */
    if (minamp == maxamp) {
        while (minamp > 1.0) minamp /= 10.0;
        maxamp = minamp;
    } else {
        amp = (int)(((amps[i]-minamp)*(maxout-minout))/(maxamp-minamp)+minout);
    }
    ph1 = (int) (((phase1[i]-0.0)*(maxout-minout))/(360.0)+minout);

    /* Quantize to Integer values, then cast them to float */
    amps[i] = (float) amp;
    phase1[i] = (float) ph1;
}

```

```
/* Normalize amps to range from 0.0 to 1.0 */
if (maxamp != minamp) amps[i] = (amps[i] - minout)/(maxout-minout);

/* Normalize phases to range from 0.0 to 360.0 */
phase1[i] = ((phase1[i]-minout)*(360.0))/(maxout-minout);

/* Write output to file */
fprintf (outfp, "    %f    %f\n", amps[i], phase1[i]); /**/

if (verbose) printf (".");
}
if (verbose) printf ("DONE.\n\n");
fclose (infp);
fclose (outfp);
}
```

```

/*****
** FILE: rquant.c
** AUTHOR: Sherk Chung
** DATE: 9/7/93
** LOCATION: COMSAT Labs, Clarksburg, MD
**
** LAST MODIFIED:
** MODIFICATIONS:
**
** COMMENTS: Quantizes Rectangular Complex Coefficients of ARRAY Input Files
**
*****/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAX_NORM 1.0 /* Output Normalized from MIM_NORM to MAX_NORM */
#define MIN_NORM 0.0
#ifndef NUM
#define NUM 1000
#endif

#define pi 3.14159265
#define d2r(x) ((x*pi)/180.0)
#define r2d(x) ((x*180.0)/pi)
#define REAL(m,p) (float) m*cos(d2r(p))
#define IMAG(m,p) (float) ((double)m*sin((double)d2r(p)))
#define MAG(r,i) (float) hypot ((double)r,(double)i)
#define PHASE(r,i) (float) r2d(atan ((double)(i/r)))

void main (int argc, char *argv[])
{
    int i,j, retval, count, verbose = 0;
    int qllevel, amp, ph1;
    char *infile, *outfile, line[160];
    float amps[NUM], phase1[NUM], maxamp, minamp, maxout, minout, tmp_amp;
    float tmp_ph1;
    FILE *infp, *outfp;

    if (argc < 4 || argc > 5) {
        fprintf (stderr, "USAGE: quant <Qlevel> <infile> <outfile>\n");
        exit (0);
    }
    /* Parse Input
    -----*/
    qllevel = atoi (argv[1]); /* Quantization Level */
    infile = argv[2]; /* Input File */
    outfile = argv[3]; /* Output File */
    if (argc == 5) /* Set VERBOSE to ON */
        if (!strcmp(argv[4], "-v")) verbose = 1;

    /* Read Input File

```

```

-----*/
infp = fopen (infile, "r");
if (infp == NULL) {
    fprintf (stderr, "FILE: %s NOT FOUND\n", infile);
}
outfp = fopen (outfile, "w");
if (outfp == NULL) {
    fprintf (stderr, "FILE: %s CANNOT BE OPENED\n", outfile);
}

retval = 1;
maxamp = 0.0;
minamp = 1000000.0;

if (verbose) printf ("Reading Input File .");

/* Copy 1st 4 lines (antenna description)
-----*/
fputs (fgets (line, 160, infp), outfp);
fputs (fgets (line, 160, infp), outfp);
fputs (fgets (line, 160, infp), outfp);
fputs (fgets (line, 160, infp), outfp); /* This is # of Array Elms */
count = atoi (line);

/* Copy next 'count' lines */
for (i = 0; i < count; i++)
    fputs (fgets (line, 160, infp), outfp);

for (i = 0; i < count; i++) {
    retval = fscanf (infp, "%f %f", &(amps[i]),
                    &(phase1[i]));
    if (phase1[i] < 0.0) phase1[i] += 360.0;
    if (retval == EOF)
        fprintf (stderr, "Unexpected EOF encountered in %d\n", i);

    /* Convert from Polar to Rectangular Coords */
    tmp_amp = REAL(amps[i], phase1[i]);
    tmp_ph1 = IMAG(amps[i], phase1[i]);

    /*fprintf (stderr, "%f %f %f %f\n", tmp_amp, tmp_ph1, amps[i], phase1[i]); */

    phase1[i] = tmp_ph1;
    amps[i] = tmp_amp;
    if (verbose) printf (".");
}

maxout = pow(2.0, (float) qllevel);
minout = 0.0;
minamp = -1.0;
maxamp = 1.0;

if (verbose) {
    printf ("DONE.\n\nQuant RANGE: %f ---> %f\n", minout, maxout);
    printf ("Amplitude RANGE: %f ---> %f\n", maxamp, minamp);
}

```



```

printf ("\nQuantizing Values .");
}
for (i = 0; i < count; i++) {
/* Normalize amps and phases to range from 1 to 2^qllevel */
amp = (int)(((amps[i]-minamp)*(maxout-minout))/(maxamp-minamp)+minout);
ph1 = (int)(((phase1[i]-minamp)*(maxout-minout))/(maxamp-minamp)+minout);

/* Quantize to Integer values, then cast them to float */
amps[i] = (float) amp;
phase1[i] = (float) ph1; /**/

/* Normalize real and imag to range from -1.0 to 1.0 */
amps[i] = ((amps[i] - minout)*(maxamp-minamp))/(maxout-minout)+minamp;
phase1[i] = ((phase1[i] - minout)*(maxamp-minamp))/(maxout-minout)+minamp;

/*fprintf (stderr, "%d %d %f %f\n", amp, ph1, amps[i], phase1[i]);**/

/* Convert back to Polar Coords */
tmp_amp = MAG(amps[i],phase1[i]);
if (tmp_amp == 0.0) {
if (phase1[i] == 0.0) phase1[i] = 0.0;
else if (phase1[i] > 0) phase1[i] = 90.0;
else if (phase1[i] < 0) phase1[i] = 270.0;
} else phase1[i] = PHASE(tmp_amp,phase1[i]);
amps[i] = tmp_amp;

/* Write output to file */
fprintf (outfp, " %f %f\n", amps[i], phase1[i]); /**/

if (verbose) printf (".");
}
if (verbose) printf ("DONE.\n\n");
fclose (infp);
fclose (outfp);
}

```

```

/*****
****
** FILE: snr.c
** AUTHOR: Sherk Chung
** DATE: 8/19/93
** LOCATION: COMSAT Labs, Clarksburg, MD
**
** LAST MODIFIED:
** MODIFICATIONS:
**
** COMMENTS: Finds SNR of a quantizer for a certain number of bits
**
*****/

```

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

```

```

void main (int argc, char *argv[])
{
    int bits;
    float B, snr, msa, mqa, Xm, VARx;

    if (argc < 2 || argc > 4) {
        fprintf (stderr, "USAGE: snr <bits> [<max sig ampl> <max quant ampl>]\n");
        exit (0);
    }
    /* Parse Input
    -----*/
    bits = atoi (argv[1]); /* Quantization bits */
    if (argc > 2) {
        msa = atof (argv[2]); /* Max Signal Ampl. */
    } else {
        msa = 1.0; /* Default msa */
    }
    if (argc > 3) {
        mqa = atof (argv[3]); /* Max Quantizer Ampl. */
    } else {
        mqa = 1.0; /* Default mqa */
    }
    B = (float) bits - 1.0;
    VARx = msa/sqrt(2.0);
    Xm = mqa;

    snr = 6.02*B+10.8 - 20*log(Xm/VARx);
    printf ("%f\n", snr);
}

```