Kronecker Structured Multirate Sensor Array Signal Processing Systems

By

William David Sánchez Rojas

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
In
ELECTRICAL ENGINEERING
(Digital Signal Processing)

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS
December, 2004

Approved by:

_____ _____
Manuel Jiménez Cedeño, Ph.D. Date
Member, Graduate Committee

_____ _____
Efraín O'Neill Carrillo, Ph.D. Date
Member, Graduate Committee

_____ _____
Domingo Rodríguez, Ph. D. Date
Chairperson, Graduated Committee

_____ _____
Isidoro Couvertier, Ph. D. Date
Director, Electrical and Computer
Engineer Department

_____ _____
Félix Fernández, Ph. D. Date
Representative of Graduate Studies

Abstract

This document presents digital signal processing formulations and computing methods using the DSP processor TMS320C6711, of Texas Instruments, to implement multirate systems with discrete and finite length signals. In many digital signal processing applications sampling rates need to be changed for high computational efficiency losing the desired information carried by the signal. These multirate systems play an important role in many engineering and communications applications such as sensor arrays, beamformnig, FIR filters, filter banks, time frequency representations and systems with associated diverse sampling rates. Special emphasis is given to the concepts of modularity and scalability during the hardware implementation.

The main goal of this work consists in reducing the sampling rate to control the lose of desired information in a communication signal. In order to obtain only the desired information stored in the original communication signal with associated lower computational effort. The implementation of these concepts was made on the DSP processor TMS320C6711 of Texas Instruments. Finally a multirate beamforming with 32 sensors were implemented on the DSP processor using simulated data from Matlab®.

Resumen

Este documento presenta formulaciones de procesamiento digital de señales y métodos computacionales usando el procesador TMS320C6711 DSP de Texas Instruments para implementar sistemas multi-frecuencia de muestreo con señales digitales y de finita duración. En muchas aplicaciones de procesamiento digital de señales la frecuencia de muestreo necesita ser alterada para una mayor eficiencia computacional perdiendo solo la información que se desea de la señal portadora . Estos sistemas de multi-frecuencia juegan un papel muy importante en muchas aplicaciones de comunicaciones e ingeniería tales como: arreglos de sensores, “beamforming”, filtros FIR, bancos de filtros, representaciones tiempo frecuencia y sistemas con diversas frecuencias de muestreo. Un énfasis muy especial es dado a los conceptos de modularidad y escalabilidad durante la implementación del hardware.

El objetivo más importante de este trabajo consiste en reducir la frecuencia de muestreo de una señal teniendo el control de la información que se pierde en una señal de comunicaciones. Para entonces obtener solo la información deseada con una asociada disminución en el esfuerzo computacional. La implementación de estos conceptos fueron hechos utilizando el procesador TMS320C6711 de Texas Instruments. Finalmente un sistema de multi-frequencia de muestreo y “beamforming” con 32 sensores fue implementado en el procesador DSP usando datos simulados desde Matlab®.

To my Lord Jesus, Yari, my family and all my friends for their important support.

Acknowledgements

I am pleased to say thanks to my God for this opportunity on this prestigious University of Puerto Rico; Dr. Domingo Rodríguez my advisor and teacher for his friendship and for his constant support to my work; Dr. Manuel Jiménez and Dr. Efrain O'Neill for their mentoring and corrections that contributed to the enhancement of this thesis. I am grateful to the Electrical and Computer Engineer Department and PRECISE project, for the economical support during my graduate studies. Finally I thank all people that contributed with this new achievement in my life, specially my family.

Table of Contents

List of Figures

List of Tables

Appendices

Chapter 1

Introduction

This work deals with the formulation of computing methods for the action of multirate systems on discrete and finite length signals where the sampling rates need to be changed for high computational efficiency. These multirate systems play an important role in many engineering and communications applications such as sensor arrays, beamformnig, FIR filters, filter banks, time frequency representations, and systems with associated diverse sampling rates.

Kronecker Structured Multirate (KSM) offers the mathematical framework of this Signal Processing System. It explains, with matrix formulations, different representations, operations, and transformations of communications signals represented by vectors.

Sensor Arrays consist of a set of sensors that spatiotemporally measure a wavefield. Several sensors, sampling a common wave field, may be merging to produce more refined information about the communication signals.

The emphasis of this work is about modular and scalable computing methods, for signal processing applications using Digital Signal Processors (DSP). The modular and scalable approach implies that the functions and structures of the algorithmic treatment should adapt to changes in the scales of an associated system, and the size or

dimensionality of the signals to be processed. The fundamental information obtained by our algorithms are important variables involved in communications signal processing systems to quantify, represent, transform, encode, decode, qualify information-carried and obtained by sensor arrays.

Algorithm is defined as a procedure to solve a problem in a finite number of steps. A problem is anything which requires a solution. Multirate systems are defined as systems that can increase or decrease the sampling spacing (and thus the sampling rate) of individual signals before, or while, processing them. Communications signal processing is described here as an area dealing with the analysis, design, and implementation of circuits, signals, and systems for the transmission and reception of communications signals. A communications signal is defined as a information coded by a signal, appearing in any of the stages of an arbitrary communication system. This work concentrates on Multirate systems for digital communications and the way to implement this concept using sensor arrays technology.

1.1 Previous Work

The interest of this work is in applications in which signal enhancement can be achieved by processing the waveform received by a single sensor, but often it is advantageous to use an array of sensors using multirate techniques. The treatment of a desired signal is mostly done through algorithmic techniques implemented on physical DSP units. The technology used to make the processing of digital signals varies from PC

workstations, matrix software development tools, until digital signal processors (DSP) units. Some of the most relevant publications are associated with the development of algorithm formulations based on Kronecker products and multirate techniques related to sensor arrays and their implementation as described below.

J. Jonson, R. W. Jonson, D. Rodríguez, and R. Tolimieri proposed a methodology for designing, modifying, and implementing Fourier transform algorithms on various architectures [1]. In 1990, they presented all the descriptions and properties of tensor products (Kronecker products) that will play a major role in the design and implementation of Fourier transform algorithms. The formalism of tensor product notation can be used to keep track of the complex index calculation needed in Fourier transform algorithms.

D. B. Ward, Z. Ding, and R. A. Kennedy proposed a broadband DOA estimation using frequency-invariant beam-space processing [2]. In 1995, they presented a new method of beam-space direction of arrival (DOA) estimation for multiple far-field broadband signals. A novel multirate beamforming structure having a frequency invariant property is applied to the array outputs.

D. B. Ward, R.A. Kennedy and R. C. Williamson proposed a theory and design of broadband sensor arrays with frequency invariant far-field beam patterns [3]. In 1998, they presented the frequency invariant beam pattern property defined in terms of a continuously distributed sensor, and the problem of designing a practical sensor array

was treated as an approximation to this continuous sensor using a discrete set of filtered broadband omni directional array elements. The design methodology is suitable for one-, two-, and three-dimensional array elements based in multirate techniques.

M. Ghavami and R. Kohomo, proposed a rectangular arrays for uniform wideband beamforming with adjustable structure [4]. In 2000, they presented the increasing of the demand for different broadband services and applications that was a key problem of the future mobile communications system. Because the limitations of the available spectrum for providing high data rate communications for new cellular scribers, it is predicted that the application of smart antennas can increase the system capacity and performance.

A. Quichanegua and D. Rodríguez, proposed a Kronecker DFT multi- beamforming implementation approach [5]. In 2003, they present a new methodology for the hardware implementation of multi-beamforming algorithms based on Kronecker products decomposition. Kronecker products algebra was used in this work as a tool language to identify integrated and coherent manner similarities and differences between fast Fourier transform (FFT) algorithm formulation in order to achieve efficient hardware core implementation.

J.C Chen, L.Yip, H. Wang, D. Maniezzo, R.E. Hudson, J. Elson, K. Yao and D. Estrin proposed a DSP implementation of a distributed acoustical beamforming on a wireless sensor platform [6]. In 2003 they proposed to perform beamforming based on coherent processing of acoustical waveforms collected from the sensor nodes for

detection, localization, tracking, identification, and signal to noise ratio (SNR) enhancement of acoustical sources counting the number of such sources and estimating the impulse responses of the acoustical channels.

1.2 Justification

The function of a Multirate system is to alter the sampling rate (up-sampling/down-sampling) of discrete-time signals to give a new sampling rate for other signal processing system. This new sampling rate offers the system, the possibility to perform their operations spending lower computational effort. The lower computational effort is obtained because the new length of the communication signal to be processed decreases with respect to the original sampling rate, this is associated directly with the number of points or samples of the original communication signal.

The main goal of this work consists in reducing the sampling rates but controlling the lose information of the original communication signal, in order to extract the relevant information stored on the original communication signal with the associated lower computational effort. The implementation of this concept was made using the DSP processor TMS320C6711 of Texas Instruments using an array of microphones and A/D converters. For complete characterization of sound phenomenon, time-frequency algorithms were implemented in order to obtain the truth capability of this floating point hipper performance processor. The theoretical framework is based on Kronecker products and the physical structures are based on sensors array.

1.3 Thesis Objectives

- Understand the concepts of multirate signal processing, sensor arrays and Kronecker Array Signal (KAS) algebra as a language for computational signal processing systems.
- Learn about Matlab tools, DSP architectures, PC-stations, in the field of multirate and sensor arrays.
- Develop algorithms for multirate sensor arrays, based on the characteristics of modularity and scalability.
- Map different algorithms into DSP units during the implementation process.

1.4 Research Methodology

In order to achieve the proposed objectives of this thesis, we fallow method bellow:

- Review and research of the literature and fundamental principles involved in digital signal processing, multirate signal processing, sensor arrays, filters banks and Kronecker mathematical formulations, in order to observe, quantify, represent, transform, qualify and render information-carrying signals in our sensor arrays reality. This step involves analysis and synthesis of the theoretical information and identification of specific hardware and software tools used for digital signal processing applications.

- Define the mathematical formulation based on Kronecker products to develop algorithms as operations matrix-vector, using MATLAB (witch stands for MATrix LABoratory).
- Selection and learning of software and hardware tools for development and implementation of the algorithms to achieve multirate sensor arrays. In this step we will define the environment of development and implementation of the algorithms for DSP units. This environment will be used throughout the PC Workstation platform, MATLAB tools, and digital signal processing (DSP) microprocessor units.
- Mapping algorithms developed to DSP computing units using a defined modular, scalable methodology. Coding the algorithms using C language.

1.5 Original Contributions

This work examines the implementation of multirate concepts on real DSP unit such as DSP320C6711, that is the last floating point DSP processor developed by Texas Instruments (T.I.). It determines the capability of these units to perform signal processing operations based on double precision variables (64-bits), in order to evaluate execution time and memory capacity of this processor.

In addition the work developed physical hardware implementation of sensors array using an array of six-microphones as unit dimensional array (ULA), developing hardware for a signal conditioner interface (*AIP-0404-1)* and obtaining real data from

microphones way 16-bits six- channels A/D converter (ADS8364 of T.I.). For a complete characterization of particular sound phenomenon.

In order to evaluate the DSP320C6711's capability of processing floating point variables, different signal processing algorithms for time-frequency representations were implemented. Some of these algorithms are Short Time Fourier Transform (STFT), Cyclic Correlation and Ambiguity Function (AF) using real data from A/Ds or simulated with Matlab®. All these algorithms and the data used are stored on a CD as a library resource for students working at the university DSP laboratories. Because there are not tools like this provided before.

Finally, as an important application a Multirate Beamforming system was developed using real and simulated data. This shows that the sensor array structures based on Kronecker products are an important tool for modularity and scalability approach, which are used in Radar and Sonar applications.

Chapter 2
Multirate Systems

This chapter presents basic concepts on multirate systems [7]. Discrete time systems with unequal sampling rates, in various parts of it are called multirate systems. Where sampling rate needs to be converted into an equivalent signal with different sampling rate. To achieve this, is important to understand the concepts of down-sampling and up-sampling and their input and output relations in the time and frequency domain. The cascade equivalences for up and down sampling are then explained too. For cascade up and down sampling rate alterations there has to be some details given, of the use of lowpass digital filters. The frequency response specifications of these filters are developed next. A computational sampling rate implementation is then illustrated by a specific design problem. The DSP320C6711 processor of Texas Instruments and its development kit was used to perform the real implementation.

2.1 The Basic Sample Rate Alteration Concepts

The two basic components in sampling rate alteration are the up-sampler and down-sampler. Figure 2.1 shows the block diagram representation for this two components. The block diagram representation of the up-sampler, also called “sampling rate expander” and the block diagram of the down-sampler “sampling rate compressor”.

The L positive integer factor represents the up-samples introduced between each sample of the original signal $x[n]$ to produce the output signal $y[n]$, and the M positive integer

factor represents the down-samples taken from the original signal *x[n]* to produce the output signal *y[n]*.

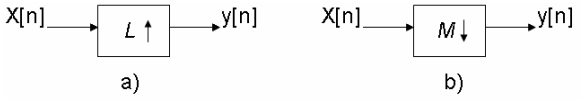


Figure 2.1 Block diagram representation for a) Up-sampling, b) Down-sampling

2.1.1 Time Domain Characterization

An up-sampler with an up-sampling factor L, where L is a positive integer, develops an output sequence *y[n]* with a sampling rate that is L times larger than the input sequence *x[n]*. This operation is implemented by inserting $L - 1$ equidistant zero-valued samples between two consecutive samples of the input sequence *x[n]* according to the relation

$$y[n] = \begin{cases} x[n / L];\ n = 0, \pm L, \pm 2L, ... \\ 0;\ otherwise \end{cases} . \qquad (2.1)$$

The up-sampling operation is illustrated in Figure 2.2 using Matlab®.

In a real application, the zero-valued samples inserted by the up-sampler are replaced with appropriated values interpolated using filtering process. This makes the new higher-rate sequence useful. This process is called interpolation, and will be discussed later in this chapter.

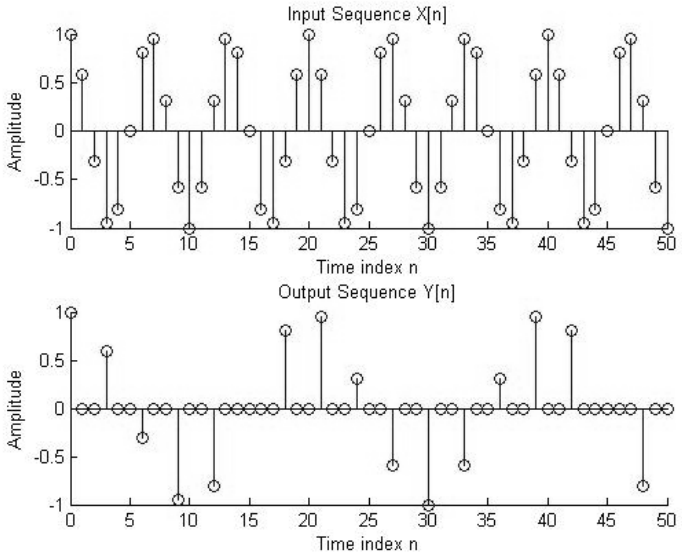


Figure 2.2 Illustration of the L↑=3, up-sampling process.

Furthermore, the down-sampler with a down-sampling factor *M*, where *M* is a positive integer, obtains an output sequence *y[n]* with a sampling rate that is (1/*M*)th of the input sequence *x[n]*. The down-sampling operation is implemented by keeping every *M*th sample of the input sequence and removing *M*-1 in-between samples, to generate the output sequence according to the relation

$$y[n] = x[nM] . \tag{2.2}$$

As a result, all input samples with indices equal to an integer multiple of *M*, are retained at the output and all others are discarded, as shown in Figure 2.3 .

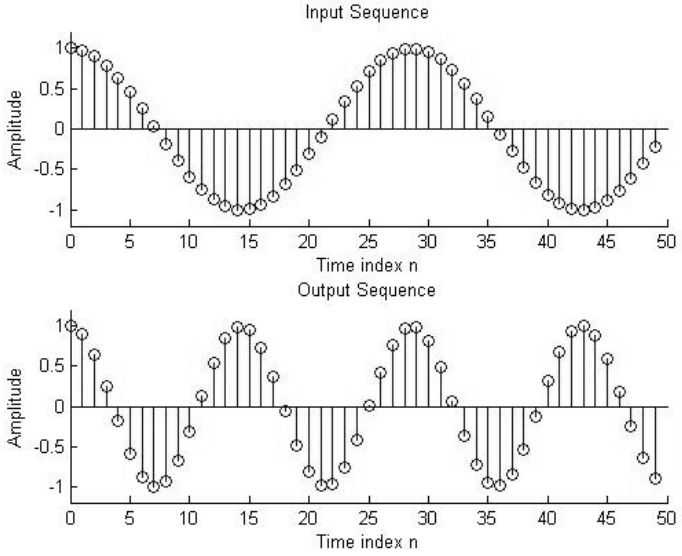


Figure 2.3 Illustration of the M↓=2, down-sampling process.

The up-sampler and the down-sampler are linear but time-varying discrete system. Down-sampling time-varying property is demonstrated for

$$\begin{aligned} y_1[n] &= x[Mn - n_0] \\ y[n - n_0] &= x[M(n - n_0)] \\ x[Mn - Mn_0] &\neq y_1[n]. \end{aligned} \tag{2.3}$$

Up-sampling time-varying property is demonstrated for

$$\begin{aligned} y_1[n] &= x[n/L - n_0] \\ y[n - n_0] &= x[1/L(n - n_0)] \\ x[n/L - n_0/L] &\neq y_1[n]. \end{aligned} \tag{2.4}$$

The linearity property of down-sampling is demonstrated using superposition

$$\begin{aligned} y[n] &= x[Mn] \\ x_3[n] &= \alpha x_1[Mn] + \beta x_2[Mn] \\ y_3[n] = \alpha y_1[Mn] + \beta y_2[Mn] &= \alpha x_1[Mn] + \beta x_2[Mn] = x_3[n]. \end{aligned} \tag{2.5}$$

The linearity property of up-sampling is demonstrated using superposition

$$y\ [n] = x\ [n / L]$$
$$x_3[n] = \alpha x_1[n / L] + \beta x_2[n / L]$$
$$y_3[n] = \alpha y_1[n / L] + \beta y_2[n / L] = \alpha x_1[n / L] + \beta x_2[n / L] = x_3[n]. \quad (2.6)$$

2.1.2 Frequency Domain Characterization

For better understanding we first derive the relations between the spectrums of the input and the output for a factor of L=*2* up-sampler.

If

$$y[n] = \begin{cases} x[n / 2]; & n = 0, \pm L, \pm 2L, ... \\ 0; & otherwise \end{cases} . \quad (2.7)$$

In terms of the z-transform, the input-output relation is then given by

$$Y(z) = \sum_{n=-\infty}^{\infty} y[n] z^{-n} = \sum_{\substack{n=-\infty \\ n-even}}^{\infty} x[n / 2] z^{-n} , \quad (2.8)$$

replacing m=*n/2*

$$\sum_{m=-\infty}^{\infty} x[m] z^{-2m} = X(z^2) . \quad (2.9)$$

In general it can be said that for a factor of *L* up-sampler the output *z* transform of the output with respect to the input is given by

$$Y(z) = X(z^L), \quad (2.10)$$

for $z = e^{j\omega}$ the above equation becomes

$$Y(e^{j\omega}) = X(e^{j\omega L}), \quad (2.11)$$

indicating that the Fourier transform is compressed by a factor of L, in this case $L\uparrow=2$. This process is called “imaging” because we get L-1 additional image of the input spectrum in the base band. Figure 2.4 shows the normal spectrum of the input signal *x[n]*, then this signal is up-sampling by $L\uparrow=2$, Figure 2.5 shows the output signal *y[n]* and the correspond spectrum with the imaging of the original spectrum.

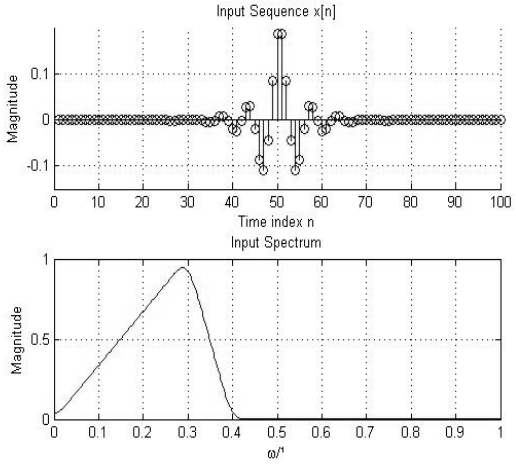


Figure 2.4 Input sequence and input spectrum for *x[n]*.

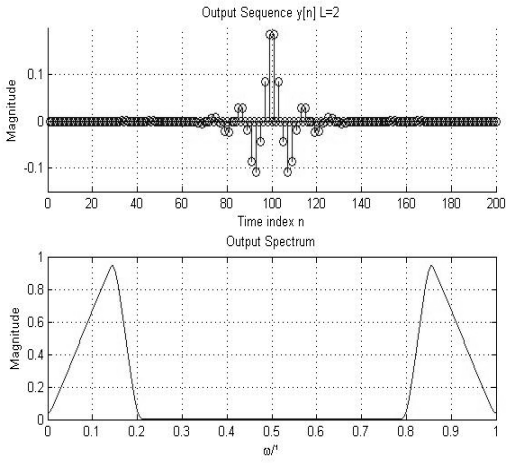


Figure 2.5 Output sequence and output spectrum for *y[n]*.

For $L\uparrow=3$ Figure 2.6 shows the original input spectrum and correspond L-1 spectral imaging of the output.

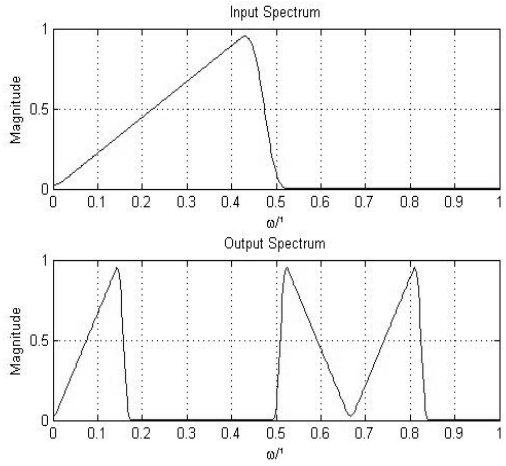


Figure 2.6 Input and output spectrum, Up-sampling by $L\uparrow=3$.

Now we derive the relations of the input and output spectrums relations of a down-sampler, applying the z-transform.

If

$$y[n] = x[Mn]$$

$$Y(z) = \sum_{n=-\infty}^{\infty} x[Mn]\, z^{-n} \quad . \tag{2.12}$$

The right hand equation cannot be directly expressed in terms of $X(z)$. An intermediate sequence $x_{int}[n]$ is used

$$x_{\text{int}}[n] = \begin{cases} x[n], & n = 0, \pm M, \pm 2M \\ 0\,; \textit{otherwise} & . \end{cases} \tag{2.13}$$

Then

$$Y(z) = \sum_{n=-\infty}^{\infty} x[Mn]\, z^{-n} = \sum_{n=-\infty}^{\infty} x_{\text{int}}[Mn]\, z^{-n}$$

$$k = Mn$$

$$Y(z) = \sum_{n=-\infty}^{\infty} x_{\text{int}}[k]\, z^{-k/M} = X_{\text{int}}[z^{1/M}]\,. \tag{2.14}$$

Now if $x_{int}[n]$ can be related to $x[n]$ through $x_{int}[n] = c[n]x[n]$, where c$[n]$ is defined by

$$c[n] = \begin{cases} 1;\ n = 0, \pm M, \pm 2M \\ 0;\ otherwise \end{cases}. \tag{2.15}$$

A convenient representation of $c[n]$ is given by

$$c[n] = \frac{1}{M} \sum_{k=0}^{M-1} W_M^{kn}\,. \tag{2.16}$$

where $W_M = e^{-j2\pi/M}$. Substituting in $x_{int}[n] = c[n]x[n]$ and making use of z-transform of $x_{int}[n]$, we obtain

$$X_{\text{int}}(n) = \sum_{n=-\infty}^{\infty} c[n]\,x[n]\,z^{-n} = \frac{1}{M} \sum_{n=-\infty}^{\infty} \left(\sum_{k=0}^{M-1} W_M^{kn} \right) x[n]\, z^{-n}$$

$$X_{\text{int}}(n) = \frac{1}{M} \sum_{k=0}^{M-1} \left(\sum_{k=-\infty}^{\infty} x[n] W_M^{kn} z^{-n} \right) = \frac{1}{M} \sum_{k=0}^{M-1} X(zW_M^{-k})\,. \tag{2.17}$$

From $Y(z) = X_{\text{int}}[z^{1/M}]$, we get

$$Y(z) = \frac{1}{M} \sum_{k=0}^{M-1} X[z^{1/M} W_M^{-k}]\,. \tag{2.18}$$

To understand the implication relation of the above relation, we can consider the case of down-sampler $M{\downarrow}=2$, by replacing $z = e^{j\omega}$ then

$$Y(z) = \frac{1}{2} \sum_{k=0}^{2-1} X[z^{1/2} W_2^{-k}]\,, \tag{2.19}$$

expanding and replacing we get

$$Y\left(e^{j\omega}\right) = \frac{1}{2}\left\{X\left(e^{j\omega/2}\right) + X\left(e^{j(\omega+2\pi)/2}\right)\right\}. \qquad (2.20)$$

The output spectrum expresses the original spectrum of *x[n]* expanded by 2, and the same expanded spectrum shifted by 2π. Figure 2.7 shows the input sequence and spectrum of *x[n]*, Figure 2.8 shows the output sequence and spectrum of down-sampling by $M\downarrow=2$.

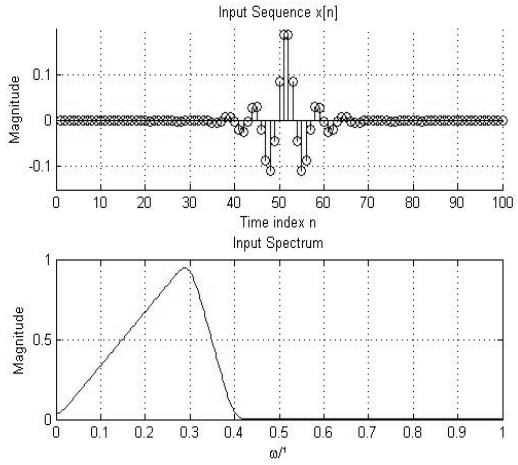


Figure 2.7 Input sequence and spectrum of *x[n]*.

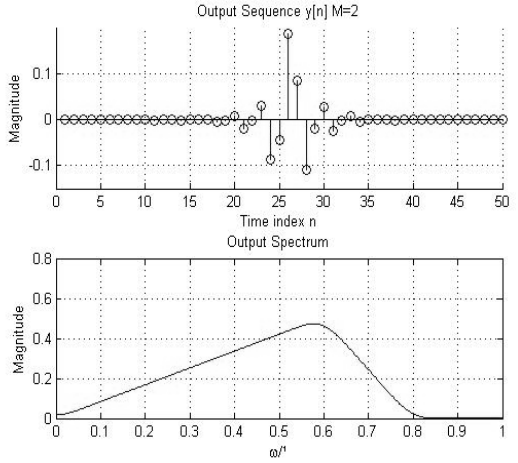


Figure 2.8 Output sequence and spectrum of *y[n]*, down-sampling $M\downarrow=2$.

We need to consider that if the original spectrum $X(e^{j\omega})$ of the input sequence *x[n]* is non zero for $|\omega| \geq \pi/2$, this causes an overlap and the output spectrum experiments "aliasing", that takes place due to undersamplig. Figure 2.9 shows an input sequence x*[n]* with spectrum $X(e^{j\omega})$ and $|\omega| \geq \pi/2$, the output experiments "aliasing".

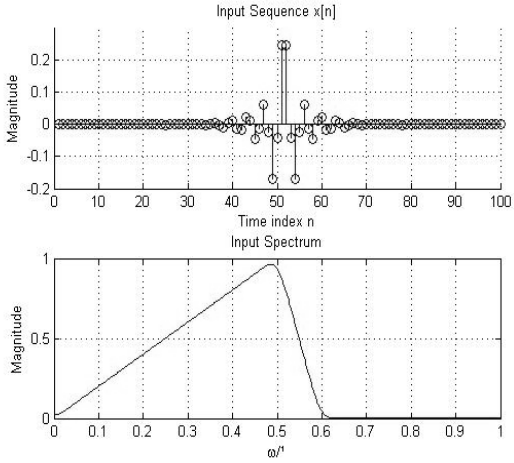


Figure 2.9 Input sequence and spectrum of x[n], with non zero for $|\omega| \geq \pi/2$.

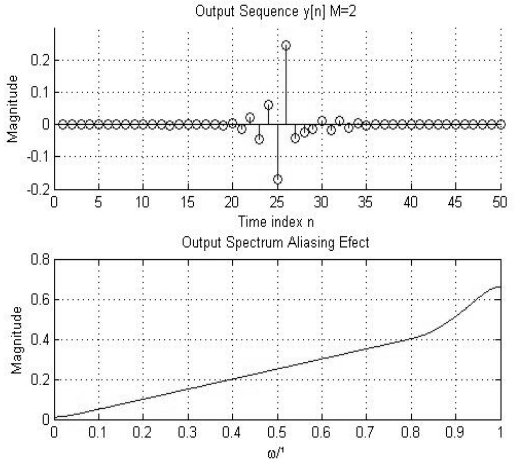


Figure 2.10 Output sequence and spectrum of *y[n]*, with aliasing effect.

2.2 Cascade Connections

The complex multirate system is formed by an interconnection of basic sample rate alteration devices and the components of an LTI digital filter. In many applications these devices appear in cascade form, because not only integer up or down sampling rate is needed Furthermore some applications use up/down or down/up fractional rates as shown in Figure 2.11.

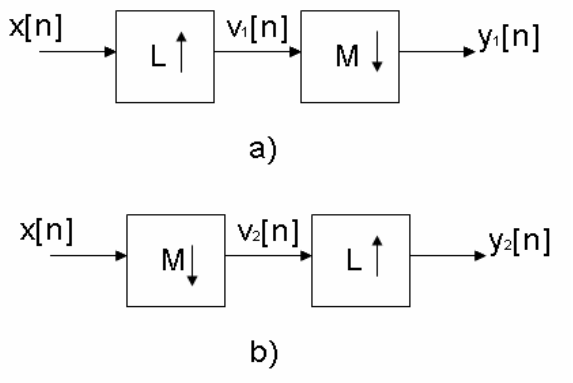


Figure 2.11 Cascade arrangements a) up/down sampler b) down/up sampler.

2.3 Filters in Sampling Rate Converters Systems

From the sampling theorem, it is known that the critical sampling rate of a discrete time signal with spectrum occupying the full Nyquist range, cannot be reduced any further. This is because such reduction will introduce aliasing. For that reason, the bandwidth of a critically sample signal must be reduced by lowpass filtering, before its sampling rate is reduced by a down sampler. Likewise, the zero-valued samples

introduced by an up-sampler must be interpolated to more appropriate values for an effective sampling rates increase. Also, this interpolation can be simply achieved by digital lowpass filtering.

2.3.1 Filter Specifications

Since up sampling causes periodic repetition of the basic spectrum as shown in Figure 2.5, the unwanted images in the spectra of the up-sampled signal *y[n]* must be removed by using a lowpass filter *Hi(z)*, called the "interpolation filter" Figure 2.12(a). On the other hand, as indicated in Figure 2.12(b), prior to down sampling, the signal $v_1[n]$ should be bandlimited to $|\omega| \geq \pi / M$ by means of a lowpass filter $H_d(z)$, called the "decimation filter", to avoid aliasing caused by down sampling process.

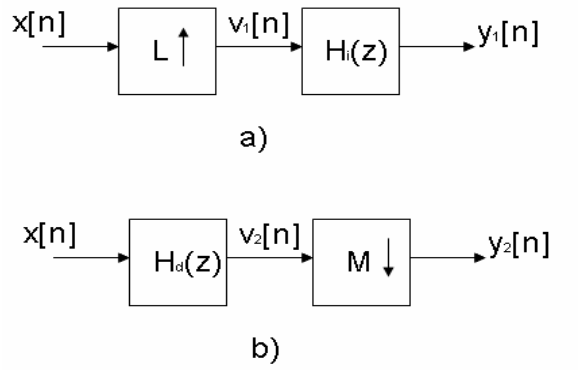


Figure 2.12 Filters in sampling rate alteration a) interpolator and b) decimator.

The specifications of the interpolator filter *Hi(z)* are based on the bandwidth of the spectrums of $x[n] \rightarrow X(e^{j\omega})$ Figure 2.13(a), $v_1[n] \rightarrow X(e^{j\omega L})$, $H_i(e^{j\omega})$ Figure2.13(b)

and $Y_1(e^{j\omega})$ Figure 2.13(c). In practice, a transition band is provided to ensure the realizability and stability of the lowpass interpolation filter $H_i(e^{j\omega})$ with a cut frequency at π / L and gain *L*, the output of the filter will be precisely *y[n]*. Hence, the desired lowpass filter should have a stopband edge at $\omega_s = \pi / L$ and a passband edge ω_p close to the stopband ω_s to reduce the distortion of the spectrum of the signal *x[n]* . If ω_c denotes the highest frequency that needs to be preserved in the signal to be interpolated, the passband edge ω_p of the lowpass filter should be at $\omega_p = \omega_c / L$. The specifications for the lowpass interpolation filter are thus given by

$$H_i(e^{j\omega}) = \begin{cases} L\,; n = |\omega| \le \omega_c / L \\ 0\,; \pi / L \le |\omega| \le \pi \end{cases} . \qquad (2.21)$$

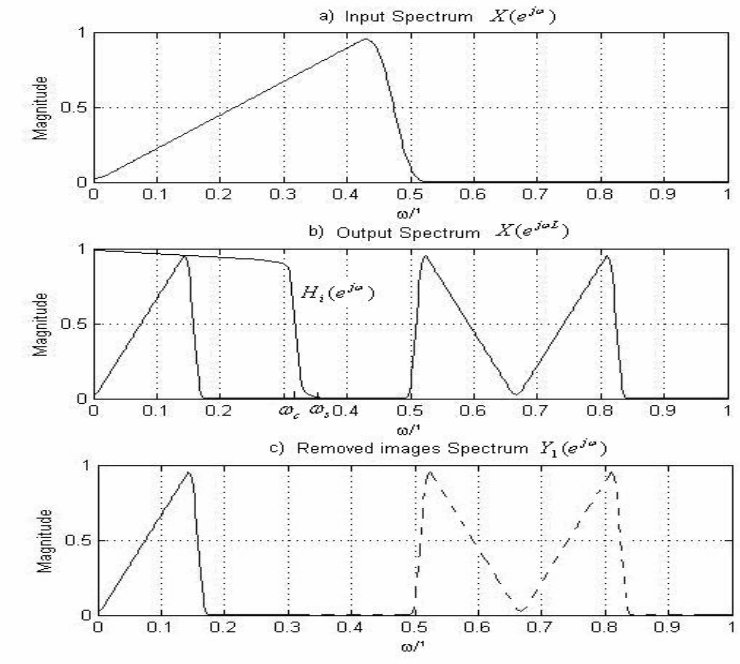


Figure 2.13 Spectrum of a) the input *x[n]*, b) the output *v1[n]* and interpolator filter for L↑=3, and c) the output *y[n]*.

In a similar manner, the developed specifications of the lowpass decimation filter are given by

$$H_d(e^{j\omega}) = \begin{cases} 1;\ n = |\omega| \leq \omega_c / M \\ 0;\ \pi / M \leq |\omega| \leq \pi . \end{cases} \quad (2.22)$$

The two digital lowpass filters studied before and their specifications, guarantee the complete information reproduction of the input signal with the associated frequency sampling *Fs1*, and the output signal with the associated frequency sampling *Fs2*, for up sampling *Fs2* > *Fs1* and for down sampling *Fs2* < *Fs1*.

2.3.2 Filter for Fractional Sampling Rate Converters

A fractional change in the sampling rate can be achieved by cascading a factor of M↓ decimator with a factor of *L*↑ interpolator, where *M* and *L* are positive integers. Such cascade is equivalent to a decimator with a decimation factor of *M/L*. There are three possible cascade connections, as shown in Figure 2.14. Of these two, the 2.14b) is more efficient since only one of the filters, *Hi(z)* or $H_d(z)$, is adequate to serve as the interpolation filter and the decimation filter, depending on which one of the two stopband frequencies, π/L or π/M is a minimum. It should be noted in Figure 2.14a) that in general, preserve less of the signal's frequency content than the one on Figure 2.14b), because the multirate system starts with a lowpass filter to reduce the spectral content of the input signal in order to avoid aliasing. Hence, the desired configuration for the fractional sampling rate alteration is as indicated in Figure 2.14c), where the lowpass filter *Hi(z)* has a normalized stopband cutoff frequency at

$$\omega_s = \min\left(\frac{\pi}{L}, \frac{\pi}{M}\right). \tag{2.23}$$

which suppressed the imaging caused by the interpolator while, at the same time ensures the absence of aliasing that would be caused by the decimator.

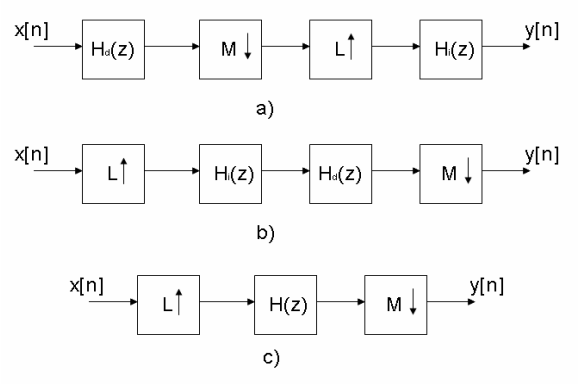


Figure 2.14 General schemes for increasing the sampling rate by *L/M*

2.4 A Real Computational Implementation on DSP processor TMS320C6711

This implementation consists of a fractional rate change by *M/L=2/3* or *M/L=3/2* from the input signal. Based on the concepts of up-sampling, down-sampling and FIR (Finite Impulse Response) lowpass filters, these show the base of a Multi-rate systems, and proof different sample rates to know the DSP real ranges.

2.4.1 System Flow Chart Implementation

Figure 2.15 shows the multirate system flow chart implemented on the DSK320C6711 (Development Starter Kid) of Texas Instruments. The steps for the real implementation are the following:

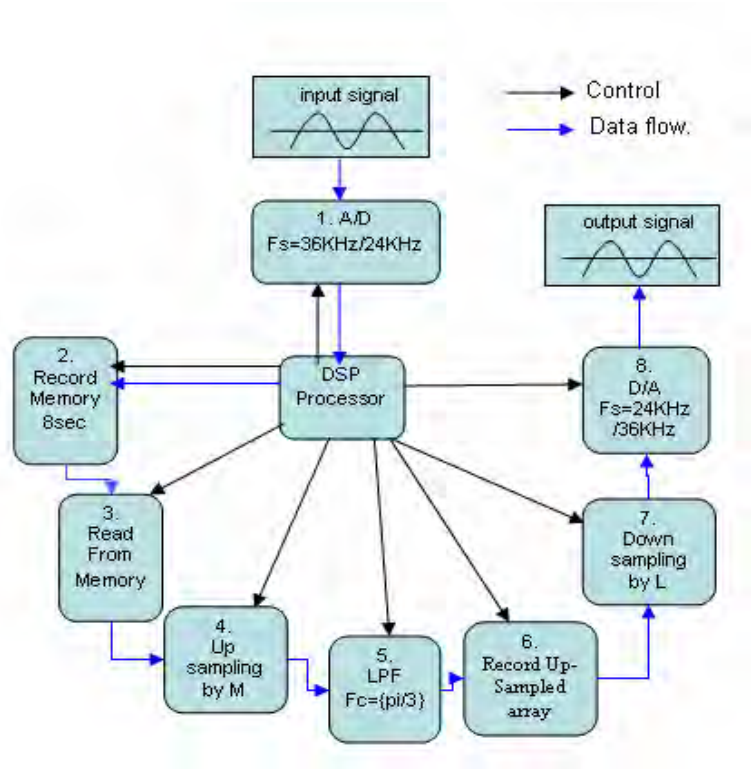


Figure 2.15 System flow chart implemented on DSK320C6711 of T.I.

Input signal: Signal generator, microphone or music with maximum bandwidth of 4KHz.

1. A/D converter: Sampler input signal at a rate of 36KHz or 24KHz.
2. Record Memory: It stores the sampled signal into a buffer array of size 36Ksps*8sec. or 24Ksps * 8sec.

3. Read from Memory: Array to recover the samples stored in the buffer memory.
4. Up-sampling by *M*: Introduce (*M-1*) zeros between each sample read from the buffer-memory.
5. LPF: It removes spectral images, interpolates and limits frequency for the down-sampling stage.
6. Record up-sampled array: This array stores the output of the up-sampling routine.
7. Down-sampling by *L*: Reads every *L* sample from the output memory.
8. D/A converter: It converts from digital to analog with a sampling rate of 24 KHz or 36 KHz.

The output signal is reproduced with the corresponding frequency sampling (*Fsout*), the output D/A converter operates at frequency 24 KHz or 36KHz. This is obtained from the following relation

$$Fsout = \frac{F_{\sin} * M}{L}$$

$$Fs_1 out = \frac{36\ KHz * 2}{3} = 24\ KHz$$

$$Fs_2 out = \frac{24\ KHz * 3}{2} = 36\ KHz \quad . \qquad (2.24)$$

2.4.2 Routines Flow Chart

The system algorithm is described on Figure 2.16.It defines the different stages and routines implemented on the DSK320C6711 using the sound daughter card PCM3003 of T.I. On the Appendix A the complete specifications and hardware settings are described.

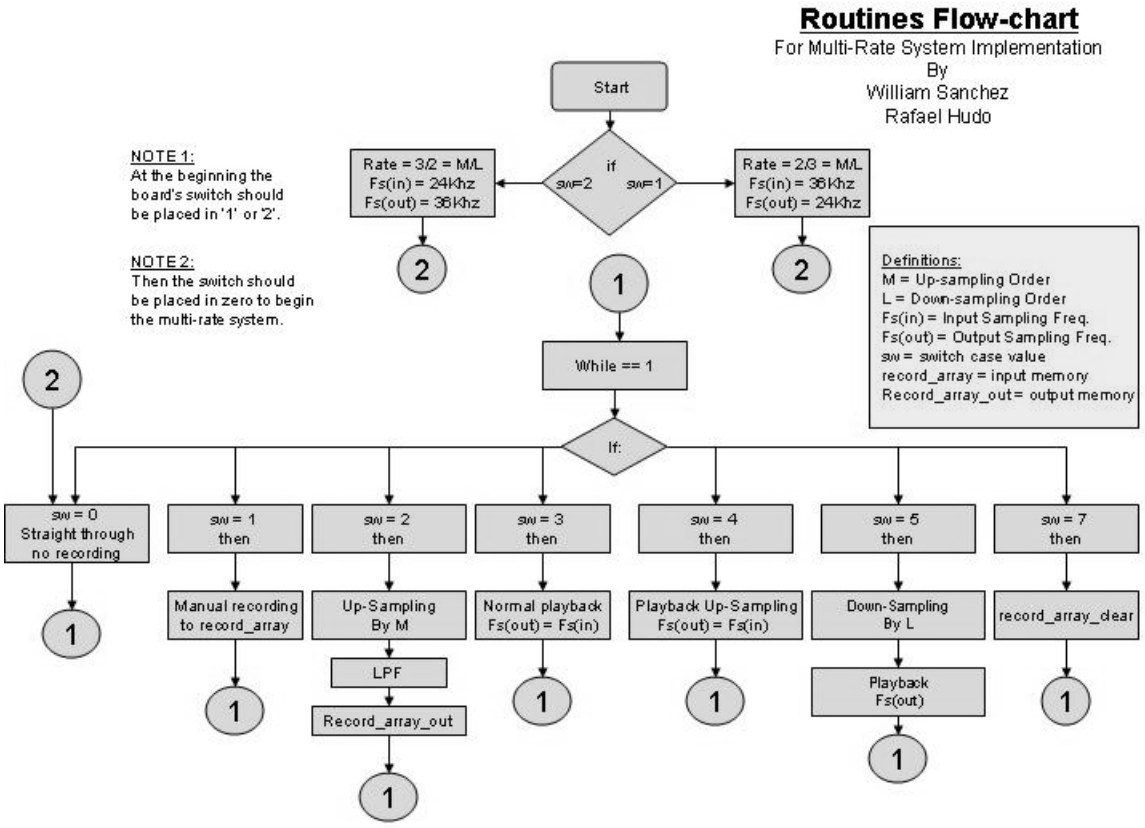


Figure 2.16 System Routines flow chart implemented on DSK320C6711 of T.I.

2.4.3 Multirate Results

The following steps explain the different functions of the Multirate project implemented.

Step # 1.

Channel 1: Original Input Signal (Vpp = 500mv and 1 KHz).

Channel 2: Up-Sampled Signal by 2

Fs(in) = 36 KHz.

Fs(out) = 36 KHz.

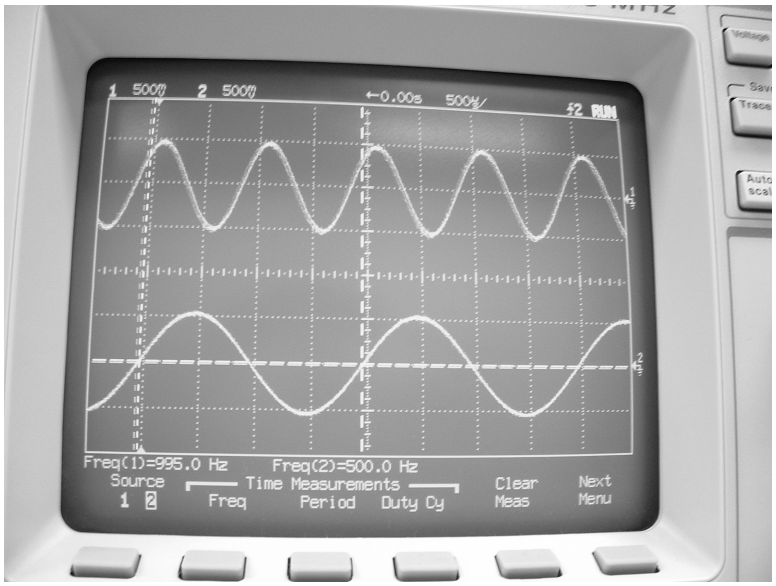


Figure 2.17 Up-sampling by two.

Figure 2.17. shows in channel 2, up-sampled by 2, it has half frequency of the original input signal, presented in channel 1. The up-sampling process increases the size of the original input signal by *M=2*. To show that the up sampling process works, we selected Fs (out) with equal value as Fs (in) so that the output signal has two times the samples than the original signal, and comes out with half the frequency of the original signal. On the other hand is important to know that the output signal has been filtered by LPF with cut frequency of $\pi/3$, to interpolate the zero value samples added with the original samples and to eliminate the spectral images generated by the up-sampler.

Step# 2

Channel 1: Original Input Signal (Vpp = 500mv and 1 KHz).

Channel 2: Signal after Up/Down sampling processes, rate *M/L= 2/3.*

Fs (in) = 36 KHz.

Fs (out) = 24 KHz.

Figure 2.18 shows that the signal in channel 2, after the up/down sampling processes, has the same frequency than the original recorded signal. The reason for this, is that the input signal's sampling frequency, Fs(in) = 36 KHz, is multiplied by the Up/Down sampling rate *M/L= 2/3*, resulting in an output sampling frequency, Fs (out) = 24 KHz. To show that the Up/Down sampling processes works, we selected Fs (out) = 24 KHz in order to reproduce correctly the input signal.

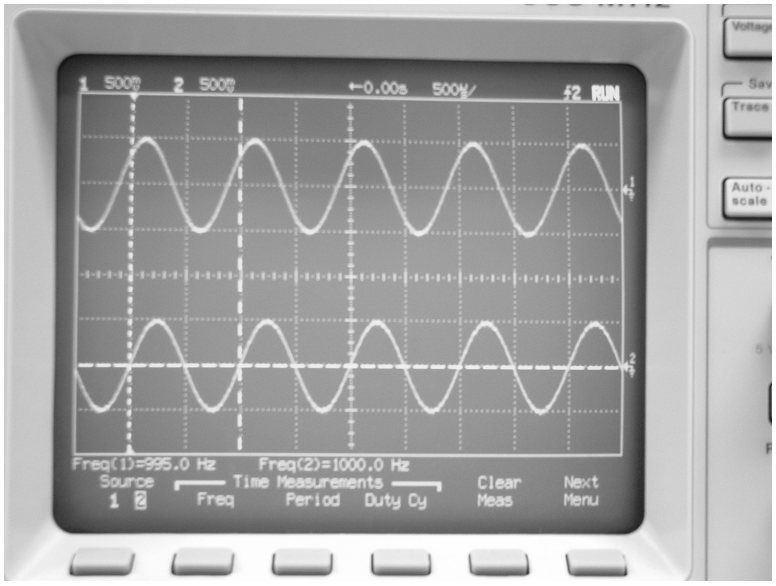


Figure 2.18 Up/Down sampling rate *M/L=2/3.*

The signal has been filtered to eliminate the spectral images and possible aliasing created by the up-sampler and down-sampler respectively. The time delay for the processing of the complete system was shown by the shift in phase between signals of channel 1 and 2.

Step# 3

Channel 1: Original Input Signal (Vpp = 500mv and 1 KHz).

Channel 2: Up-Sampled Signal by 3

Fs (in) = 24 KHz.

Fs (out) = 24 KHz.

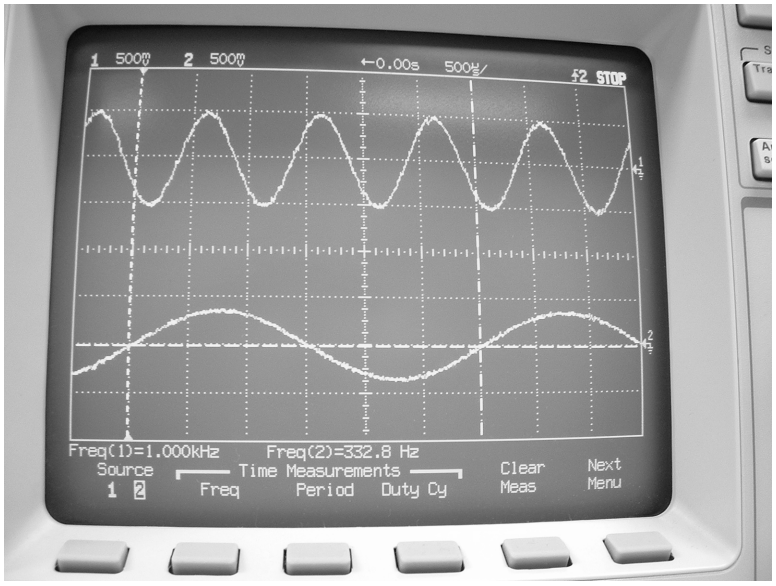


Figure 2.19 Up-Sampling by *M=3*.

Figure 2.19 shows that the signal in channel 2, up-sampled by 3, has 1/3 the frequency of the original input signal, presented in channel 1, because the up-sampling process increases the size of the original input signal by *M=3*. To show that the up-sampling process works, we selected Fs (out) with equal value as Fs (in) so that the output signal, with three times the samples as the original signal, comes out with 1/3 of the frequency of the original signal. The output signal has been filtered by LPF with cut frequency of $\pi/3$

to interpolate the zero value samples added, and to eliminate the spectral images generated by the up-sampler.

Step# 4

Channel 1: Original Input Signal (Vpp = 500mv and 1 KHz).

Channel 2: Signal after Up/Down sampling processes, rate *M/L= 3/2.*

Fs (in) = 24 KHz.

Fs (out) = 36 KHz.

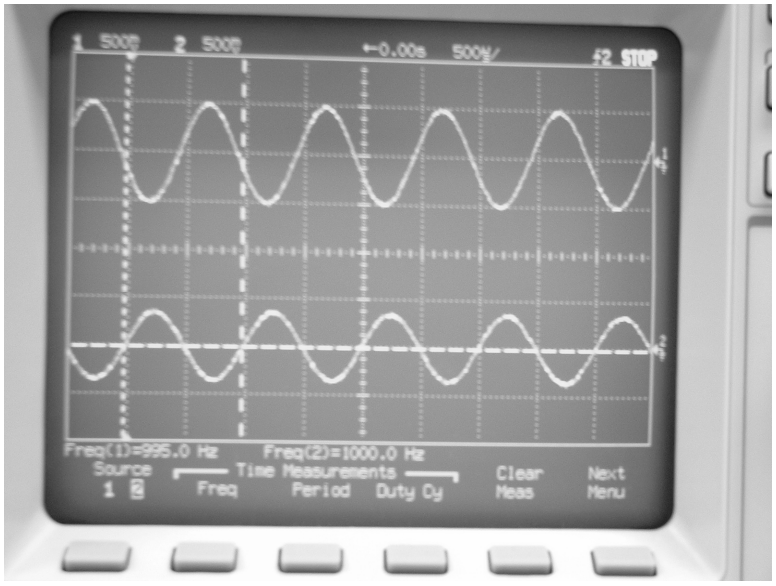


Figure 2.20 Up/Down sampling rate *M/L=2/3.*

Figure 2.20 shows that the signal in channel 2, after the up/down sampling processes, has the same frequency as the original recorded signal. This is due to the input signal's sampling frequency, Fs(in) = 24 KHz, which is multiplied by the up/down sampling rate *M/L= 3/2*, resulting in a output sampling frequency, Fs (out) = 36 KHz. To demonstrate that the up/down sampling processes works, we selected Fs (out) = 36 KHz in order to

reproduce correctly the input signal. This signal has been filtered to eliminate the spectral images and possible aliasing created by the up-sampler and down-sampler respectively.

2.4.4 Lowpass filter design with MatLab

We used Matlab® as a tool for the coefficients filter design. This LPF lowpass digital filter is used for interpolation, spectral images rejection and anti-aliasing. We designed a finite impulse response FIR filter, with order of 13 and a cut frequency of л/3. We used the FIR1 command to generate the coefficients used, by the implementation, that are located in the file *COEF_CLPF.h.* (see appendix A for complete implementation).Figure 2.21 shows Frequency and Phase response for the designed filter.

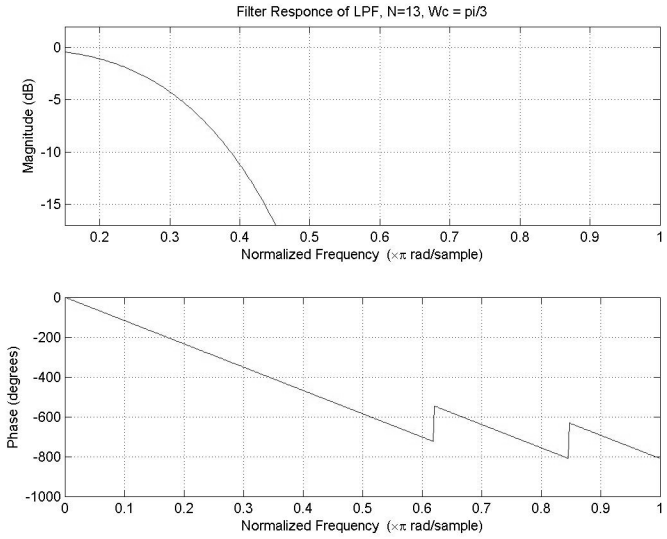


Figure 2.21. Frequency and Phase Response for LPF.

Chapter 3

Kronecker Products Algebra

For the implementation of signal processing we designed computational structures. To achieve this, Kronecker products algebra, a branch in finite multilinear algebra which serves as an organizational language, was used. The mapping between Kronecker products notation and code generation for core implementations is made by using the mathematical properties of Kronecker products [8]. Below you will find some of the basic properties of Kronecker products that are key elements on the development and implementations of digital signal processing algorithms, such as filter banks algorithms and digital beamforming.

3.1 Properties of Kronecker Products

Given two matrices, $A(m \times n)$ and $B(k \times l)$, the Kronecker product is defined as the $(mk \times nl)$ matrix

$$A \otimes B = [Ab_{[r,s]}] = \begin{bmatrix} Ab_{0,0} & Ab_{0,1} & \cdots & Ab_{0,l-1} \\ Ab_{1,0} & Ab_{1,1} & \cdots & Ab_{1,l-1} \\ \vdots & \vdots & & \vdots \\ Ab_{k-1,0} & Ab_{k-1,1} & \cdots & Ab_{k1,l-1} \end{bmatrix} . \qquad (3.1)$$

Where $b_{r,s}$ is the (*r*,*s*)-th element of *B*. this definition is in fact a left Kronecker product [8]. Notice that the Kronecker product of the equation (3.1) is not commutative, i.e., $A \otimes B \neq B \otimes A$. This work concentrates mostly on square matrices, that is, $m = k$ and $n = l$, but the definition generally applies to matrices of any dimensions.

We now state the key properties of the Kronecker products that are useful to understand matrix decompositions and further Kronecker formulations [9].

- **Scalar Multiplication:** If α is a scalar, then

$$A \otimes (\alpha B) = \alpha (A \otimes B). \tag{3.2}$$

- **Distributive Law:** The Kronecker product is distributive with respect to addition

$$(A + B) \otimes C = A \otimes C + B \otimes C \tag{3.3}$$

$$A \otimes (B + C) = A \otimes B + (A \otimes C). \tag{3.4}$$

- **Associative Law**: The Kronecker product is associative

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C. \tag{3.5}$$

- **Identity Product**: Given I_{rc}, the $r \times c$ identity matrix,

$$I_{rc} = I_r \otimes I_c. \tag{3.6}$$

- **Transpose**: The transpose for both matrix and tensor operations is useful for manipulating symmetric matrices (e.g. the Fourier matrix), where the original matrix and the transpose are equal.

$$(AB)^T = B^T A^T$$

$$(A \otimes B)^T = A^T \otimes B^T \quad . \tag{3.7}$$

- **Mixed Product Rule**: Let A and C be $M \times M$ and B and D be $N \times N$ matrices. Thus,

$$(A \otimes B)(C \otimes D) = AC \otimes BD \ . \tag{3.8}$$

One useful identity which follows (3.8), given $D(l \times s)$, then

$$A \otimes D = (AI_N) \otimes (I_l D) = (A \otimes I_l)(I_N \otimes D) \tag{3.9}$$

- Let U be a $M \times 1$ vector denoted by $U_M = [u_0, u_1, u_2, \cdots, u_{M-1}]^T$, where $u_0 = u_1 = u_2 = \cdots = u_{M-1}$. Then, $(I_N \otimes U_M^T) = [I_{N,0,} I_{N,1,} \cdots, I_{N,M-1}]$.

The action of the matrix $C = A \otimes B$ on an arbitrary *MN* dimensional vector can be performed efficiently with the aid of the following decomposition

$$A \otimes B = (A \otimes I_N)(I_M \otimes B) = (I_M \otimes B)(A \otimes I_N), \tag{3.10}$$

where I_M and I_N are *N* and *M* dimensional identity matrices.

3.1.1 Stride Permutation Matrices

We are able to use a class of matrices called stride permutations [9], which commute a Kronecker product; given that Kronecker products are not commutative. These permutation matrices let Kronecker product formulations to be mapped to parallel and vector architectures, converting parallel operations $(B \otimes I_M)$ to vector operations $(I_N \otimes A)$. Essentially, a stride permutation matrix, denoted by $P_{N,S}$ is a square matrix which its effect over a $N \times 1$ vector, to move the components of the vector to new

locations using the parameter *s*, called the stride. The stride is related with the size of a vector, being $N = R \cdot S$, where R and S are integer numbers. Consider a $R \cdot S \times R \cdot S$ matrix $P_{RS,RS}$ represented as follows

$$P_{RS,S} = \begin{bmatrix} p_{(0,0)} & p_{(0,1)} & \cdots & p_{(0,RS-1)} \\ p_{(1,0)\_} & p_{(1,1)} & \cdots & p_{(1,RS-1)} \\ \vdots & \vdots & \ddots & \vdots \\ p_{(RS-1,0)} & p_{(RS-1,1)} & & p_{(RS-1,RS-1)} \end{bmatrix}. \tag{3.11}$$

where the elements $p_{(i,j)}$ of the permutation matrix are: $p_{(i,j)} = 1$ for $i = j = (RS-1)$, and for $j = iS \bmod (RS-1)$, $0 \le i < (RS-1)$; $p_{(i,j)} = 0$ for the other *i,j* elements.

For instance, let N=4 and S=2 be the order of permutation matrix and the stride respectively. Then $P_{4,2} = P_{2\cdot 2,2}$ is generated as follows

$p_{(i,j)} = 1$ for (*i,j*)=(0,0), (1,2), (2,1), (3,3); $p_{(i,j)} = 0$ for the other *i,j* elements.

The stride permutation matrix becomes

$$P_{4,2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.13}$$

The action of $P_{4,2}$ over a 4×1 vector *x* is represented in the expression below:

$$P_{4,2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_0 \\ x_2 \\ x_1 \\ x_3 \end{bmatrix}. \tag{3.14}$$

In general, $P_{N,S}$ reorders the coordinates at stride S into S consecutives segments of *M* elements, the *i*-th segment beginning with x_{i-1}. The most important property of the stride permutations is that they commute the factors in the Kronecker products of matrices. We now state some of the properties of the stride permutations.

If *N*=*RS*, then the stride permutation matrix $P_{N,S}^{-1} = P_{N,R}$.

If *N*=*RS*, then the stride permutation matrix $P_{N,S}^{T} = P_{N,R}$.

If *N*=*RS*, then the product between $P_{N,R}$ and $P_{N,S}$ becomes

$$P_{N,R} \cdot P_{N,S} = I_n . \tag{3.15}$$

Now, we state a theorem for the commutation of Kronecker factors using the previous stride permutations properties.

Theorem 3.1 If A is a $R \times R$ matrix and B is an $S \times S$ matrix then

$$P_{N,S}(A \otimes B)P_{N,S}^{-1} = (B \otimes A). \tag{3.16}$$

Details of the stride permutations and proof of theorems and properties are described in [9]. In this chapter, we have presented basic concepts on Kronecker products algebra, which are used in the development of computing algorithms for digital signal processing.

3.2 A Generalized Kronecker product

In this section we will introduce a generalized matrix product [8], which inherits some useful algebraic properties from the standard Kronecker products matrices.

Definition 3.1 Given a set of N $(m\times r)$ matrices A_i, i = 0, 1,…,*N-1*, denoted by $\{A\}_N$, and an $(N\times l)$ matrix B, we define the $(mN\times rl)$ matrix $(\{A\}_N \otimes B)$ as

$$\{A\}_N \otimes B = \begin{Bmatrix} A_0 \otimes b_0 \\ A_1 \otimes b_1 \\ \vdots \\ A_{N-1} \otimes b_{N-1} \end{Bmatrix} \quad , \tag{3.17}$$

where b_i denotes the ith row vector of B. If each matrix A_i is identical, then reduces to the usual Kronecker product of matrices.

Example 3.1

Let

$$\{A\}_2 = \begin{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\ \begin{bmatrix} 1 & -j \\ 1 & j \end{bmatrix} \end{bmatrix} \quad B = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

The definition of 3.1 yields

$$\{A\}_2 \otimes B = \begin{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & -j \\ 1 & j \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \\ 1 & j & -1 & -j \end{bmatrix},$$

which is recognized as a (4×4) DFT matrix with the rows arranged in bit-reversed order.

Definition 3.2 Let $\{A\}_N$ be a sequence of N, $(m\times n)$ matrices and E be a single $(n\times r)$ matrix.

Then

$$\{A\}_N \otimes E \equiv \begin{Bmatrix} A_0 E \\ A_1 E \\ \vdots \\ A_{N-1} E \end{Bmatrix}, \tag{3.18}$$

where each matrix in the sequence is $(m \times r)$. From property (3.10) we know

$$(\{A\}_N E) \otimes (\{B\}_N F) = (\{A\}_N \otimes \{B\}_N)(E \otimes F). \tag{3.19}$$

The next two identities are useful in developing sparse matrix factorizations.

Identity 3.1 This identity yields a block-diagonal matrix containing the matrices A_i.

$$\{A\}_N \otimes I_N = \bigoplus_{i=0}^{N-1} A_i = A_0 \oplus A_1 \oplus \cdots \oplus A_{N-1} \tag{3.20}$$

Identity 3.2 Now if we can express a *R* matrix as *p* sets of matrices denoted by $\{A^{(k)}\}_{N_K}$, *k = 0, 1,… p-1,* where each matrix is $(m \times n)$, and the *k*th set has $N_k = m^k$ matrices. Consider the matrix *R* formed as

$$R = \{A^{(p-1)}\}_{m^{p-1}} \otimes \{A^{(p-2)}\}_{m^{p-2}} \otimes \cdots \otimes \{A^{(1)}\}_m \otimes A^{(0)}. \tag{3.21}$$

The matrix *R* admits the sparse matrix factorization

$$R = \prod_{k=0}^{p-1} \left[\bigoplus_{i=0}^{m^{p-k-1}-1} (I_{n^k} \otimes A_i^{(p-k-1)} \right]. \tag{3.22}$$

Example 3.2

For convenience, we show the identity *m=n=2, p=3,* as the identity can be established for other dimensions by following the same pattern of steps. Let $R = \{A^{(2)}\}_4 \otimes \{A^{(1)}\}_2 \otimes A^{(0)}$ be written explicitly as

$$R=\begin{bmatrix}A_0^2\\A_1^2\\A_2^2\\A_3^2\end{bmatrix}\otimes\begin{bmatrix}A_0^1\\A_1^1\end{bmatrix}\otimes A^{(0)}=\begin{bmatrix}\begin{bmatrix}A_0^2\\A_1^2\end{bmatrix}\otimes A_0^1\\\begin{bmatrix}A_2^2\\A_3^2\end{bmatrix}\otimes A_1^1\end{bmatrix}\otimes A^{(0)}=[D]\otimes A^{(0)}, \qquad (3.23)$$

where for convenience we have set $D=\{A^{(2)}\}_4\otimes\{A^{(1)}\}_2$. Using equation (3.9) we have

$$[D]\otimes A^{(0)}=([D]\otimes I_2)(I_4\otimes A^{(0)}). \qquad (3.24)$$

Now, to solve for $[D]\otimes I_2$, note from equation (3.20) that

$$\begin{aligned}[D]\otimes I_2&=\begin{bmatrix}A_0^{(2)}\\A_1^{(2)}\end{bmatrix}\otimes A_0^{(1)}\oplus\begin{bmatrix}A_0^{(2)}\\A_1^{(2)}\end{bmatrix}\otimes A_1^{(1)}\\[D]\otimes I_2&=\left(\begin{bmatrix}A_0^{(2)}\\A_1^{(2)}\end{bmatrix}\otimes I_2\right)\left(I_2\otimes A_0^{(1)}\right)\oplus\left(\begin{bmatrix}A_2^{(2)}\\A_3^{(2)}\end{bmatrix}\otimes I_2\right)\left(I_2\otimes A_1^{(1)}\right)\\[D]\otimes I_2&=\left(\begin{bmatrix}A_0^{(2)}\\A_1^{(2)}\end{bmatrix}\otimes I_2\oplus\begin{bmatrix}A_2^{(2)}\\A_3^{(2)}\end{bmatrix}\otimes I_2\right)\left(I_2\otimes A_0^{(1)}\oplus I_2\otimes A_1^{(1)}\right)\end{aligned} \qquad (3.25)$$

we obtain the sparse matrix factorization

$$R=\begin{bmatrix}A_0^{(2)}&&&\\&A_1^{(2)}&&\\&&A_2^{(2)}&\\&&&A_3^{(2)}\end{bmatrix}\begin{bmatrix}I_2A_0^{(1)}&\\&I_2A_0^{(1)}\end{bmatrix}\left[I_4\otimes A^{(0)}\right], \qquad (3.26)$$

which agrees with (3.22) for *m=n=2, p=3.*

Identity 3.3 DFT matrices may also be expressed as

$$R_N = \{B\}_{N/2} \otimes R_{N/2}, \qquad R_1 = 1, \tag{3.27}$$

and

$$B_i = \begin{bmatrix} 1 & W_N^{\langle i \rangle} \\ 1 & -W_N^{\langle i \rangle} \end{bmatrix} \qquad i = 0,1,\cdots,\frac{N}{2}-1, \tag{3.28}$$

where $W_N = e^{-j2\pi/N}$.

3.3 Filter Bank Structure

First a simple DFT filter bank [8].

$$h(z) = \frac{1}{N} R_N a(z), \tag{3.29}$$

where R_N is an $(N \times N)$ DFT matrix, and $a(z)$ is a "delay" vector obtained as

$$a(z) = [1 \quad z^{-1} \quad z^{-2} \quad \cdots \quad z^{-(N-1)}]^t. \tag{3.30}$$

The word "delay" is used for $a(z)$, as the term z^{-1} denotes the backward delay operator.

The utility of this filter bank in signal processing stems from a modest signal decorrelation property obtained at the filter bank outputs.

The choice of R_N as a DFT matrix in (3.29) is for handiness, since this choice also allows output samples obtained from the filter bank, to be interpreted as a Short Time Fourier Transform of a sliding window of the input samples. The elements $H_k(z), \quad k = 0,1,\cdots,N-1$ of $h(z)$ are shown in this case to satisfy

$$H_k(z) = H_0(zW_N^{-k}), \qquad k = 1,2,\cdots,N-1. \tag{3.31}$$

Thus, the system can be understood as a bank of bandpass filters, where the frequency response of each band is frequency shifted of that of the adjacent band.

Let us choose $N=8$ and derive two different realizations. Fist express from (3.27)

$$\begin{aligned} R_8 &= \{B\}_4 \otimes \{R_4\} \\ R_8 &= \{B\}_4 \otimes \{B\}_2 \otimes \{R_2\} \end{aligned} \tag{3.32}$$

$$R_8 = \begin{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & & & \\ & \begin{bmatrix} 1 & -j \\ 1 & j \end{bmatrix} & & \\ & & \begin{bmatrix} 1 & e^{-j\pi/4} \\ 1 & -e^{-j\pi/4} \end{bmatrix} & \\ & & & \begin{bmatrix} 1 & e^{-j3\pi/4} \\ 1 & -e^{-j3\pi/4} \end{bmatrix} \end{bmatrix} \tag{3.33}$$

$$\bullet \begin{bmatrix} I_2 \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & \\ & I_2 \otimes \begin{bmatrix} 1 & -j \\ 1 & j \end{bmatrix} \end{bmatrix} \bullet \begin{bmatrix} I_4 \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \end{bmatrix}$$

A flowgraph representation of the filter bank appears in Figure 3.1.

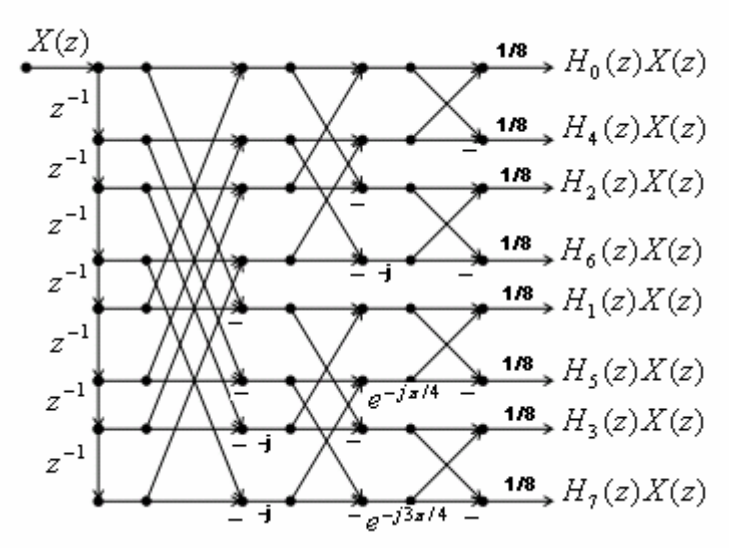


Figure 3.1 Simple DFT filter bank

To obtain an alternative realization, observe that the delay vector can be factored as

$$a(z) = [1 \quad z^{-1} \quad z^{-2} \quad z^{-3}\, z^{-4} \quad z^{-5} \quad z^{-6} \quad z^{-7}]^t = \begin{bmatrix} 1 \\ z^{-1} \end{bmatrix} \otimes \begin{bmatrix} 1 \\ z^{-2} \end{bmatrix} \otimes \begin{bmatrix} 1 \\ z^{-4} \end{bmatrix}, \quad (3.34)$$

replacing on (3.32) and using the definition 3.2

$$h(z) = \frac{1}{8}\left(\{B\}_4 \otimes \{B_2\} \otimes \{R_2\}\right)\left(\begin{bmatrix} 1 \\ z^{-1} \end{bmatrix} \otimes \begin{bmatrix} 1 \\ z^{-2} \end{bmatrix} \otimes \begin{bmatrix} 1 \\ z^{-4} \end{bmatrix}.\right)$$

$$h(z) == \frac{1}{8}\left(\{B\}_4 \begin{bmatrix} 1 \\ z^{-1} \end{bmatrix} \otimes \{B_2\} \begin{bmatrix} 1 \\ z^{-2} \end{bmatrix} \otimes \{R_2\} \begin{bmatrix} 1 \\ z^{-4} \end{bmatrix}\right). \quad (3.35)$$

we obtain

$$h(z)=\frac{1}{8}\begin{bmatrix}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1 & \\ & z^{-1}\end{bmatrix} & & & \\ & \begin{bmatrix}1 & -j\\ 1 & j\end{bmatrix}\begin{bmatrix}1 & \\ & z^{-1}\end{bmatrix} & & \\ & & \begin{bmatrix}1 & e^{-j\pi/4}\\ 1 & -e^{-j\pi/4}\end{bmatrix}\begin{bmatrix}1 & \\ & z^{-1}\end{bmatrix} & \\ & & & \begin{bmatrix}1 & e^{-j3\pi/4}\\ 1 & -e^{-j3\pi/4}\end{bmatrix}\begin{bmatrix}1 & \\ & z^{-1}\end{bmatrix}\end{bmatrix} \bullet \begin{bmatrix}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1 & \\ & z^{-2}\end{bmatrix} & \\ & \begin{bmatrix}1 & -j\\ 1 & j\end{bmatrix}\begin{bmatrix}1 & \\ & z^{-2}\end{bmatrix}\end{bmatrix} \bullet \begin{bmatrix}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1 & \\ & z^{-4}\end{bmatrix}\end{bmatrix} \tag{3.36}$$

The correspond signal flowgraph appears on Figure 3.2.

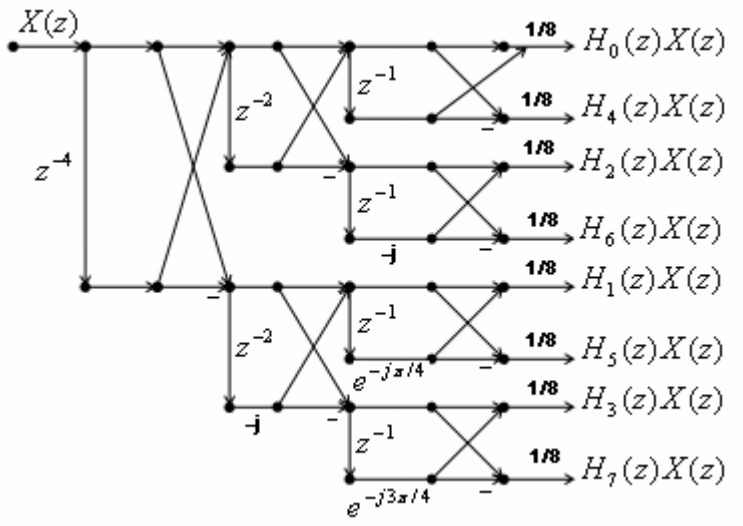


Figure 3.2 Equivalent DFT filter bank

Figure 3.2 has more delay elements than Figure 3.1, but the computational requirements have been reduced. For transforms of larger dimension, this can represent a

substantial saving in computational complexity. For example, with a larger N (taken to be a power of two), a direct implementation from equation (3.33) as Figure 3.1 would require (N-*1*) number of delays, and $(N/2)\log_2(N)$ number of butterflies. Whereas the equivalent realization from equation (3.36), as in Figure 3.2 would require $(N/2)\log_2(N)$ number of delays and (N-*1*) number of butterflies.

Chapter 4

Sensor Array Structures

In this chapter we will present the problem of detecting signals using information from multiple sensors. The objective is to be able to understand the advantages of using a sensor array over a single element. Once we achieve this, we will learn about complex representation of the signals and DFT (Discrete Fourier Transform) for DOA (Detection of Arriving Angle), near and far wave fields, and signal to noise ratio (SNR) enhancement. Finally, a radar signal processing Toolbox for Matlab® (DBT 2.1®) shows an example of sensor arrays beamforming.

In active sensing situations (e.g., radar and sonar), a known wave form of finite duration is generated, which in turn propagates through a medium, and is reflected by some target back to the point of origin. The transmitted signal is usually modified, both in amplitude and phase by the target characteristics, which by themselves might be changing with time and its position in space. For that reason, important digital signal processing theory, time-frequency representations and algorithms are implemented to obtain the information carried by the signal.

4.1 Basic Concepts of Signal Complex Representation

This chapter concentrates on the digital processing of plane sound waves arriving

at a passive sensor array. The structures of the receptor sensors are called unit dimensional array (ULA), and for our case, the study emphasizes in the linear array structures. The sensor and structure can be extended in quantity and dimension through the use of Kronecker array products.

Changes in amplitude, direction and frequency can be modeled for signals far from the source, as a sinusoidal plane wave carrying an amount of energy, and propagating with a constant velocity away from the source [10]. A plane wave has some attributes such as amplitude, wavelength λ, temporal frequency f_c, spatial frequency *k*, and propagation speed ν. A propagating plane wave can be sensed and modeled at a specific time and spatial point along a propagating direction, say the *x*-direction, using the following expression

$$S(t, x_0) = V_0 e^{j2\pi(f_c t - k x_0)}. \tag{4.1}$$

4.1.2 Spatial Sampling of a Plane Wave

A plane wave can be spatially sampled, using an array of omnidirectional sensors, in order to extract information about its propagation direction and frequency content.

Features of a sensor array, such as number of sensors and distance between sensors, are related with the wavelength of the incident wave [11]. In our study case, a linear sensor array is utilized, and the sampling along the array can be represented as a

finite, discrete signal to be processed by the DSP processor. This signal along with linear sensor array axis at time *t* can be expressed as follows

$$S(t, x_0) = V_0 e^{j2\pi\left(f_c t - \frac{x_0}{\lambda}\sin\theta_0\right)}. \quad (4.2)$$

Figure 4.1 shows an incident plane wave at angle θ_0 over a linear array of equally spaced sensor with separation distances *d.*

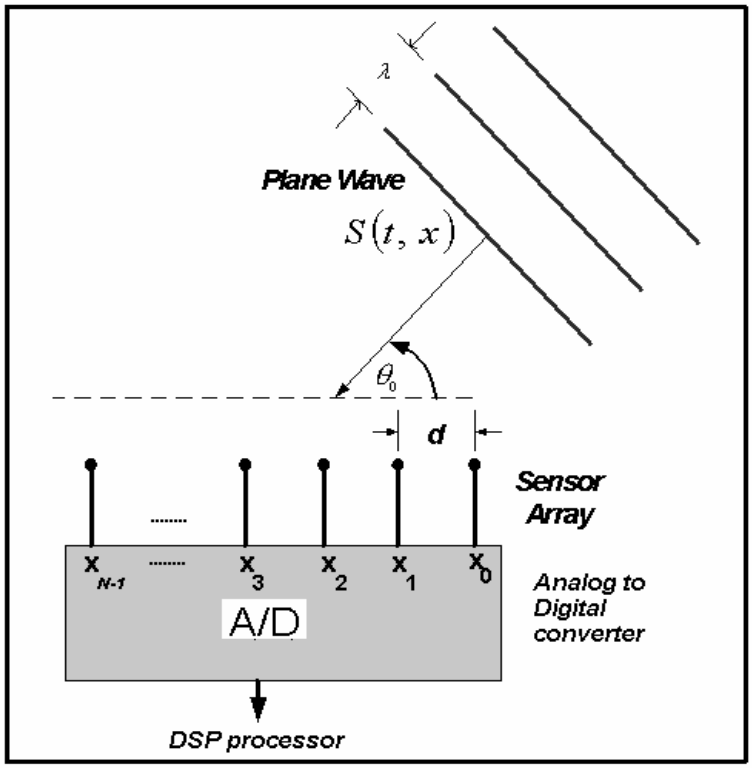


Figure 4.1 Sensor Array Model for DSP implementation

Then, considering the position of the sensor on the axis *x*, the signal can be expressed in terms of the position of the sensors changing *x* by *–k.d* for convenience as follows

$$\phi_k = V_0 e^{j2\pi\left(f_c t + \frac{k.d}{\lambda}\sin\theta_0\right)} . \quad (4.3)$$

In general, the signal ϕ_k at the *k-th* sensor can be

$$\phi_k = \phi_0 e^{j2\pi\left(\frac{k.d}{\lambda}\beta_0\right)} , \quad (4.4)$$

where $\beta_0 = \sin\theta_0$. Spatial samples from all of linear sensor arrays can be expressed as an input vector $\Phi(\beta_0) = [\phi_0, \phi_1 ..... \phi_{N-1}]^T$, where $\Phi(\beta_0)$ represents a monochromatic plane wave coming from β_0 incidence direction.

4.1.3 DFT for Direction of Arriving (DOA) Signal

In the time domain this operation is performed using the time delay in order to obtain the coherent sum over the *N* sensors. The direction of the arriving signal is treated from the point of view of a linear transformation over a finite and discrete input signal. The DOA, is denoted as a row vector such that $B(\beta_0)\Phi(\beta_0) = N\phi_0$, where

$$B(\beta_0) = \left[1, e^{-j2\pi\left(\frac{d}{\lambda}\beta_0\right)}, e^{-j2\pi\left(\frac{2d}{\lambda}\beta_0\right)}, ..., e^{-j2\pi\left(\frac{(N-1)d}{\lambda}\beta_0\right)}\right] . \quad (4.5)$$

In general, the DOA, or commonly defined as beamforming vector, is used to steer an input vector $\Phi(\beta_0)$, towards $B(\beta_0)$ direction obtaining as a result a beam pattern of a linear array steered to a specific direction of arriving. The product $B(\beta_0)\Phi(\beta_0)$ can be represented as

$$B(\beta_0)\Phi(\beta) = \sum_{k=0}^{N-1} \phi_k e^{-j2\pi kv} , \quad (4.6)$$

where $v = (d/\lambda)\beta_0$ is called the spatial spectrum variable. The previous description can be extended to the formulation of multiple directions, in order to cover a discrete set of *M* angles along the entire set defined by $(-\pi < \theta < \pi)$ or $(-1 < \beta < 1)$. Multi-beamforming or multi angles detection can be formulated in terms of a matrix $\underline{\boldsymbol{B}}$ in which rows represent linear transformations $\underline{B}(\beta_k)$, for *k = 0,1,…M-1*, acting over a sampled data vector $\Phi(\beta)$ of length *N* as follows

$$\underline{B} = \begin{bmatrix} B(\beta_0) \\ B(\beta_1) \\ \vdots \\ B(\beta_{M-1}) \end{bmatrix} = \begin{bmatrix} 1 & e^{\left(-j2\pi\frac{d}{\lambda}\beta_0\right)} & \cdots & e^{\left(-j2\pi\frac{(N-1)d}{\lambda}\beta_0\right)} \\ 1 & e^{\left(-j2\pi\frac{d}{\lambda}\beta_1\right)} & \cdots & e^{\left(-j2\pi\frac{(N-1)d}{\lambda}\beta_1\right)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & e^{\left(-j2\pi\frac{d}{\lambda}\beta_{M-1}\right)} & \cdots & e^{\left(-j2\pi\frac{(N-1)d}{\lambda}\beta_{M-1}\right)} \end{bmatrix}. \quad (4.7)$$

Considering the case of *M* steering directions and *N* sensors, being *M=N*, a single linear transformation $B(\beta_k)$ can be written as

$$B(\beta_k) = \left[1, e^{-j2\pi(\frac{k}{N})}, e^{-j2\pi(\frac{2k}{N})}, ..., e^{-j2\pi(\frac{(N-1)k}{N})}\right]. \quad (4.8)$$

In (4.7), the steering angles are specifically chosen as $\beta_k = k \cdot \Delta\beta$, $\Delta\beta = \lambda / Nd$ and k={0,1,2,….,N-1}; $k \in Z_N$. That permits the redefinition of the equation (4.9) for multiple direction angles as a $N \times N$ matrix denoted by

$$\underline{B} = \begin{bmatrix} B(0) \\ B\left(\frac{1}{N}\right) \\ \vdots \\ B\left(\frac{N-1}{N}\right) \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & W_N & \cdots & W_N^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{N-1} & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix}. \quad (4.9)$$

where $W_N^k = e^{-j2\pi\left(\frac{k}{N}\right)}$. Finally the multi-beamforming matrix $\boldsymbol{B}$ becomes the Discrete Fourier Transform (DFT) matrix.

4.2 Signal to Noise ration advantage using an Array

The possibility of modifying the array outputs to enhance the desired signal reception, and simultaneously suppress the undesired ones, can be illustrated by considering a single source situation as in Figure 4.1, in presence of *N* identical sensors [10]. Let $d_1, d_2 ..... d_N$ represent the normalized distances of this sensors with respect to a reference point and $\phi_0(t)$ the complex envelope of the signal at that point. On the other hand let $n_1(t), n_2(t), ..., n_M(t)$ represent the respective noise components that are assumed in practice independent and identical process (in case of Gaussian distribution). This is evident with $\phi_k(t)$, representing the complex envelope of the total received signal at the *k*-th sensor, and using (4.4)

$$\phi_k(t) = \phi_0(t) e^{j2\pi\left(\frac{k.d}{\lambda}\beta_0\right)} + n_k(t) . \tag{4.10}$$

and the input signal-to-noise ratio (SNR) is

$$(SNR)_k = \frac{E\left|\phi_0(t)\right|^2}{E\left|n_k(t)\right|^2} = \frac{P_0}{\sigma^2}, \tag{4.11}$$

where P_0 represents the signal power received at sensor *k=0*. From (4.5) the signal components can be coherently combined, if the array output is phase shift by $e^{-j2\pi\frac{kd}{\lambda}\beta_0}$;

$k = 0,1,....,N-1$ and the resulting signals are added up. This gives the output signal *y(t)* to be

$$y(t) = \sum_{k=0}^{N-1} \phi_k(t) e^{-j2\pi(\frac{k.d}{\lambda}\beta_0)} = N\phi_0(t) + \sum_{k=0}^{N-1} n_k(t) e^{-j2\pi(\frac{k.d}{\lambda}\beta_0)} = N\phi_0(t) + n_0(t) + n_1(t) + ... + n_{N-1}(t)$$

$$(SNR)_0 = \frac{E\left|N\,\phi_0(t)\right|^2}{E\left|n_0(t) + n_1(t) + \ldots + n_{N-1}(t)\right|^2}$$

$$(SNR)_0 = \frac{N^2 P_0}{\left|\sum_i \sum_j E\left[n_i(t)\, n_j^*(t)\right]\right|}$$

$$(SNR)_0 = \frac{N^2 P_0}{N\sigma^2} = N\,(SNR)_k \,. \qquad (4.12)$$

Thus a simple phase shifting and adding operation among the sensor outputs results in an improvement in the signal-to-noise ratio by a factor equal to the number of sensors *N*.

4.3 Near and Far Waves Field

Based on the distance away from the face of the source, where the radiated wave is measured, two important regions are identified. In the near field region the electromagnetic waves emitted from the source have spherical waveforms (equi-phase fronts). In the far field region, the wavefronts can be represented by plane waves [12]. We are interested in modeling the electromagnetic waves in the far field, taking the following criterion. Considering Figure 4.2 where a radiating source at point *O* emits spherical waves. A receiving array of sensors of length *d* is at distance *r* away of the source.

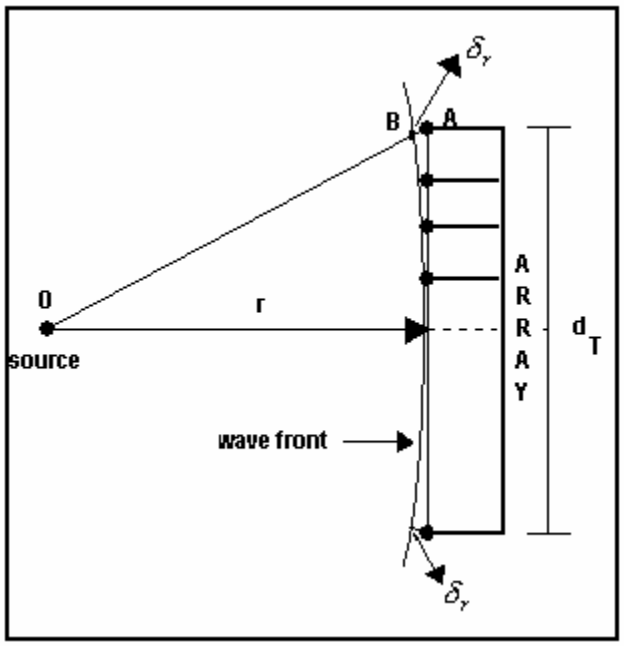


Figure 4.2 Spherical front Wave and Plane front Wave difference.

The phase difference between a spherical wave and a locally plane wave at the receiving array of sensors can be expressed in terms of the distance δ_r. The distance δ_r is given by the difference equation (4.13)

$$\delta_r = \overline{AO} - \overline{OB} = \sqrt{r^2 + \left(\frac{d_T}{2}\right)^2} - r_k \quad , \tag{4.13}$$

and since in the far field $d_T << r$, the equation (4.5) is approximated via binomial expansion by

$$\delta_r = r\left(\sqrt{1 + \left(\frac{d_T}{2r}\right)^2} - 1\right) \approx \frac{d_T^2}{8r} . \tag{4.14}$$

It is customary to assume far field when the distance δ_r corresponds to less than 1/16 of the wavelength. More precisely, if

$$\delta_r = \frac{d_T^2}{8r} \leq \frac{\lambda}{16}, \tag{4.15}$$

then a useful expression for far field is

$$r \geq \frac{2d_T^2}{\lambda}. \tag{4.16}$$

Note that a far sound wave is a function of both the antenna size and the operating wavelength.

4.4 Toolbox for Array Sensor Evaluation

The main objective of this section, is the use of application DOA (detection of arriving angle) in radar and, the correspond computational implementation using a signal processing Matlab® Toolbox for radar DBT [13]. The Toolbox is especially suited for processing in the spatial dimension using signals from an antenna array. Both simulated and measured signals can be used. The objectives of DBT are to help us in the research on array processing; to perform these on measured radar signals that support cooperation in development of software tools, and to serve as a software demonstrator. DBT was under development at FOA (Defense Research Establishment of Sweden).

4.4.1 Definition of the computational application using the DBT Toolbox

We used this Toolbox to determine the number of sensors required when two signals arrive from two different positions. We want to learn about how many sensors we

need if the differences between arrive angles are far or close. This Toolbox help us to determine and simulate a possible model of sensor array characterized by the number of sensors, and the distance between sensors for determine signal to be received.

4.4.2 How to use the Toolbox

DBT is an extension of Matlab® programming language with data types and functions for signal processing in radar, especially antenna array processing. This constitutes a language on a higher level than standard Matlab® [13].

4.4.2.1 Sequence of Commands

A typical main program using DBT has the following sequence of commands, some of which can be omitted.

- Definition of receiver antenna.
- Acquire a signal, simulated or measured.
- Calibration compensation of signal or set the compensation method to be used.
- Conventional processing.
- Select data for model based processing.
- Estimate and modify a spatial correlation matrix.
- Model based detection and estimation.
- Present the result.

An example program with a (unit dimensional array) ULA antenna with distance

d = 0.45*λ, where λ represent the wave length, and two signals at $\theta_1 = 32^\circ$ and $\theta_2 = 40^\circ$. The output plot is based on the number of sensors to be used for correct discrimination between this two (direction of arrive) DOA. A basic step for running this example is shown on Figure 4.3.

```
% Definition of constants. Create elements and
calculate element positions:
...
% Create the array:
  ant2 = defant('array',elemPos,[],elem2);
% Generate simulated received antenna signals:
  sig = compsim4(ant2, lambda, 100, 'rndnw', ...
   [theta, .phi, SNR, alpha, dalpha, dist, ...
   eye(size(theta,1))], 'rndnw');
% DOA-spectrum with beamform:
  spect1 = sdoaspc('cbf',sig, smplPoints);
Plot DOA spectra:
  splot2(spect1)
```

Figure 4.3 Programming steps for Beamforming using DBT Toolbox [13].

Figure 4.4 shows the output beam for an array of six sensors. We only observe a main lobule with the maximum at *34°* , but the two original signals has *32°* and *40°* degree of angle incidence. Figure 4.5 shows the output beam for an array of twelve sensors, where we only see a main lobule with the maximum at *33°* , but the two original signals has *32°* and *40°* degree of angle incidence. Figure 4.6 shows the output beam for an array of twenty-four sensors, where we detect two lobules with the maximum at *32°*and *40°*, but the resolution is not enough, they are mixed.

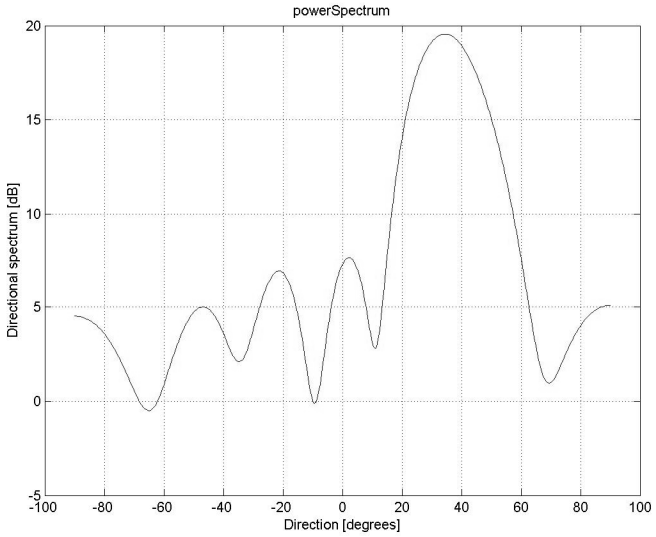


Figure 4.4 Output beam using a six sensor array.

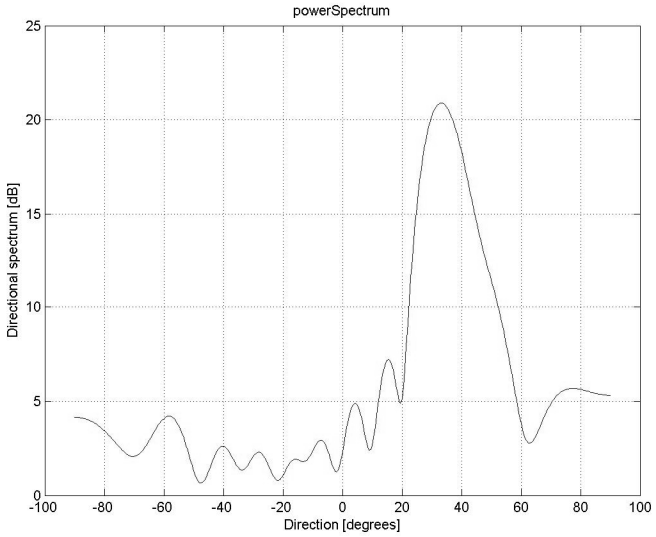


Figure 4.5 Output beam using a twelve sensor array.

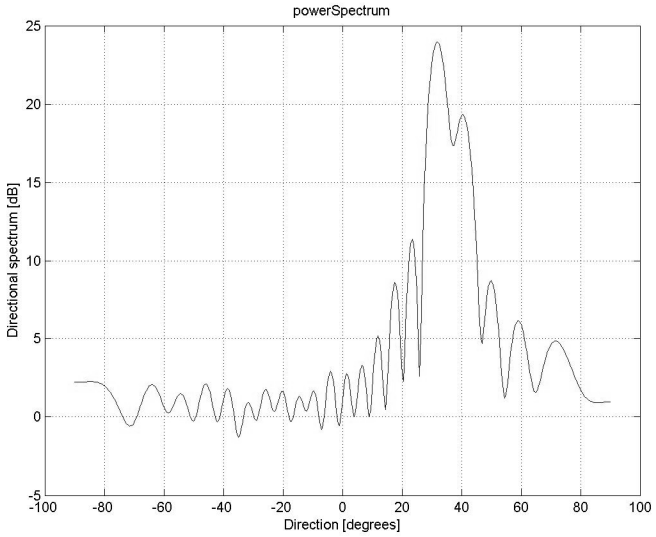


Figure 4.6 Output beam using a twenty-four sensor array.

Figure 4.7 shows the output beam for an array of forty-eight sensors; where we can observe two lobules with the maximum at *32°*and *40°* with good resolution.

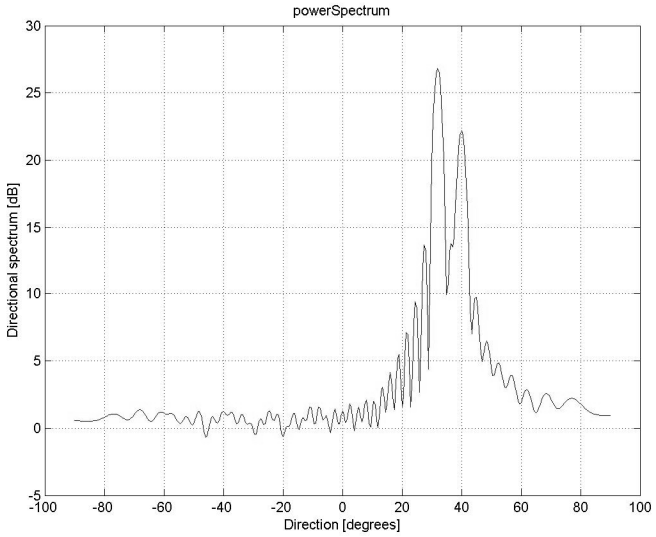


Figure 4.7 Output beam using a forty-eight sensor array.

Finally on Appendix B all Matlab® code to run this example is shown.

Chapter 5

Time-Frequency (TF) Representations

In the real world it is not common to find stationary signals. For that reason a joint representation TF, of this kind of signals, are more efficient than time or frequency domain representations alone. The disadvantage of joint this representation TF is computationally intensive, because the algorithms developed are more complex to map from time domain into the joint TF representation [14]. Nowadays with the development of new powerful digital signal processors the real time could be reached. The following real implementations based on short time Fourier Transform and Ambiguity function will be explained on this chapter.

5.1 Short Time Fourier Transform (STFT)

The theoretical key that describes computing spectra over finite time intervals is the STFT. Calculating this quantity means that we apply to a signal at time *n* a window of duration *M* (window-points), then evaluate the Fourier Transform of the product:

$$X[n]_k = \sum_{m=-\infty}^{\infty} h(n-m)\,x(m)\,W_M^{-mk} \quad . \tag{5.1}$$

Here *h(n-m)* denotes the finite window defined over *[0,M]*. The window duration defines the frequency resolution of the short-time Fourier analysis, because $X[n]_k$ equals to

Fourier Transform of the product between the sensor signal $x(m)$ and the original window at *n*, the signal's spectrum is smoothed by the window's spectrum [15].

From (5.1) we can define STFT as follows

$$X[n]_k = \sum_{m=-\infty}^{\infty} h(n-m)[x(m)W_M^{-mk}]$$

$$X[n]_k = [x(m)W_M^{-mk}] * h(m) \tag{5.2}$$

where W_M^{-mk} represent $e^{-\frac{j2\pi km}{M}}$, $*$ represent convolution and the product $[x(m)W_M^{-mk}]$ represent the modulation of $x(m)$ with $\omega_k = \frac{2\pi k}{M}$. The last equation (5.2) defines the filter method to compute the STFT [16] as shown in Figure 5.1

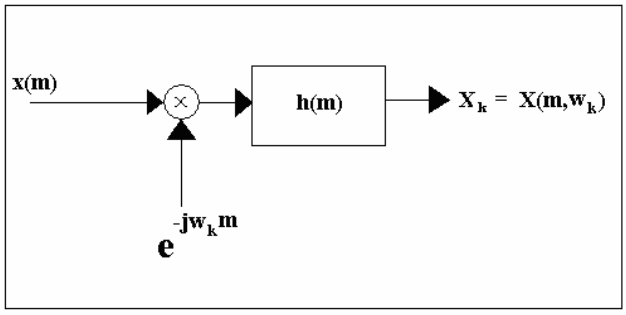


Figure 5.1 Filter method to compute STFT

Figure 5.2 shows *h(m)* as a Hanning window of 256 points, and *x(m)* as a chirp signal with two seconds of duration and frequency 0-500Hz. The result of the STFT Figure 5.3 shows the spectral content of the chirp signal at different intervals of time with step size of 126 points. The filter method has a deficient when low frequencies want to be detected because the first window does not overlap.

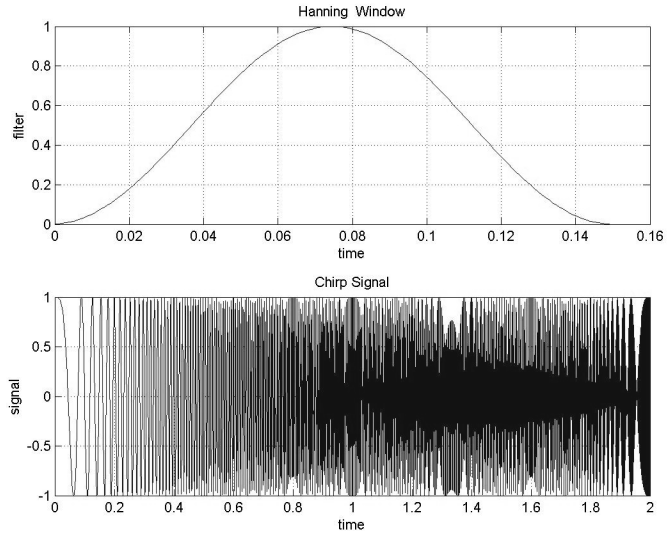


Figure 5.2 Hanning window 256 points and Chirp Signal two seconds 0-500Hz

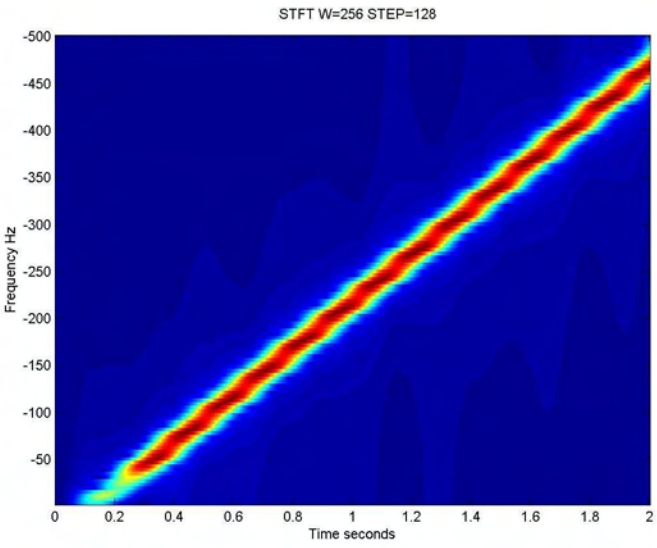


Figure 5.3 STFT belongs to Chirp signal.

Figure 5.4 shows a more realistic application of STFT using a trumpet sound with six seconds of duration, we can observe a main frequency next to 1500Hz and different harmonics separated 500Hz, here we observe the 2D and 3D representation of STFT.

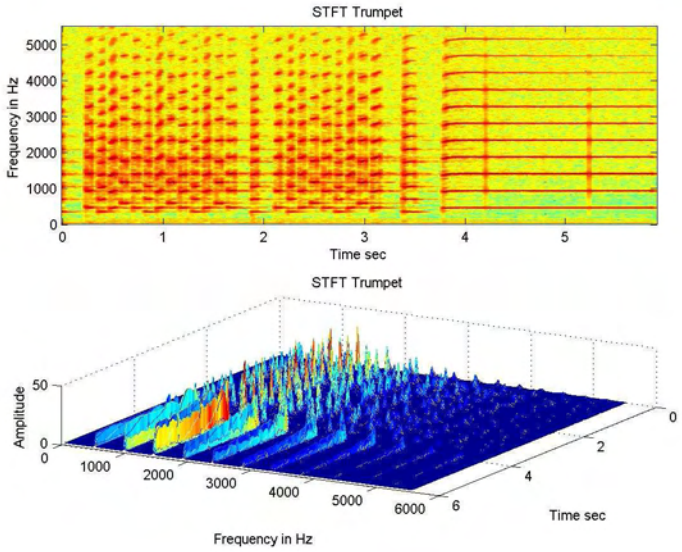


Figure 5.4 2D and 3D STFT that belongs to trumpet sound.

5.2 Ambiguity Function (AF)

Another way to compute join TF is based in the Fourier Transform of cross correlation between transmitted signal and received signal. The name of this function Ambiguity, explains if a poor correlation was found, if the ambiguity between that two signals is high, and vice versa, if the correlation is high, the ambiguity is poor between transmitted and received signal. The AF evaluates the Fourier Transform of the product

$$A(S_\alpha, S_r)[m,k] = \left| \sum_{n=0}^{N-1} S_\alpha[n] S^*_r[\langle n+m \rangle_N] e^{-\frac{j2\Pi kn}{N}} \right|,$$

let

$$S^*_{r;m}[n] = [S^*_r[\langle n+m \rangle_N], \tag{5.3}$$

then

$$S_{w;m}[n] = S_{\alpha}[n] S^{*}{}_{r;m}[n], \tag{5.4}$$

$$A(S_{\alpha}, S_{r})[m,k] = \left| \sum_{n=0}^{N-1} S_{w;m}[n] e^{-\frac{j2\Pi kn}{N}} \right|. \tag{5.5}$$

Where S_{α} and S_{r} are transmitted and received signal respectively, *m* means the index for move in object space, *k* means frequency spectral shift. $S^{*}{}_{r;m}[n]$ means a family of received signals with different time shifting *m* and this shift are cyclic (5.3). $S_{w;m}[n]$ means a signal family of windows equal to Hadamard product between $S_{\alpha}[n]$ transmitted signal and $S^{*}{}_{r;m}[n]$ the family conjugate of received signals shift in time for each *m* (5.4). Finally (5.5) shows the Fourier Transform of each signal family's window $S_{w;m}[n]$. Equation (5.5) also shows, the Fourier Transform of multiplication of two signals that means the correlation of the spectral signal multiplied. The last method to compute the AF is called Frequency Correlation. Figure 5.5 shows the 3-D plot of the AF versus frequency and time delay. The AF is normally used by radar designers to study different waveforms. In this example we simulate a chirp signal transmitted and received for a bandwidth of 3Khz, and the difference between time and frequency is plotted using Matlab®. The algorithms to compute this AF are implemented on of the TMS320C6711 DSP.

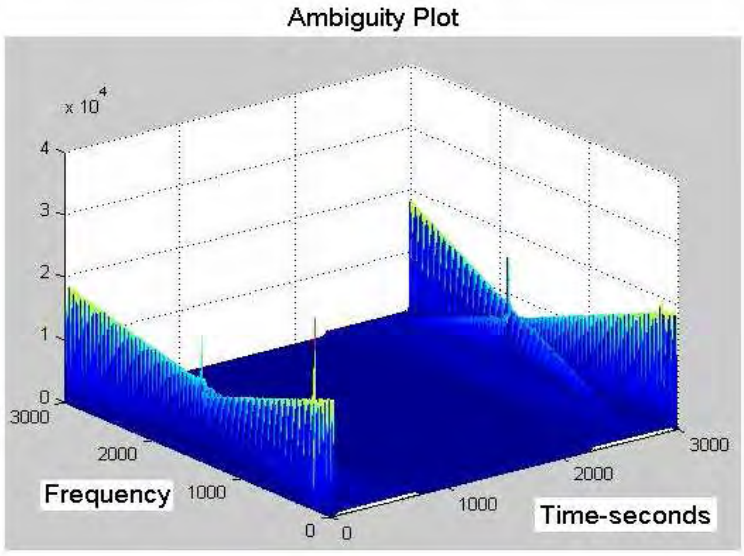


Figure 5.5. AF for Tx and Rx Chirp signal.

5.3 Time-Frequency Hardware Implementation using the DSK320C6711

In order to use a real DSP processor for TF representations, we implemented the STFT and the AF for different signals size on the TMS320C6711 of T.I. and the Code Composer Studio CCS v2.1. These implementations are used to compute the capability of processing and storage of the DSP processors, using double precision variables (64-bits) [14]. We will determine the capabilities of this DSP processor using the time of execution, size of the signal processed and memory consumption; these results will be summarized on Table 1and Table 2. For next sections the STFT and the AF will be implemented.

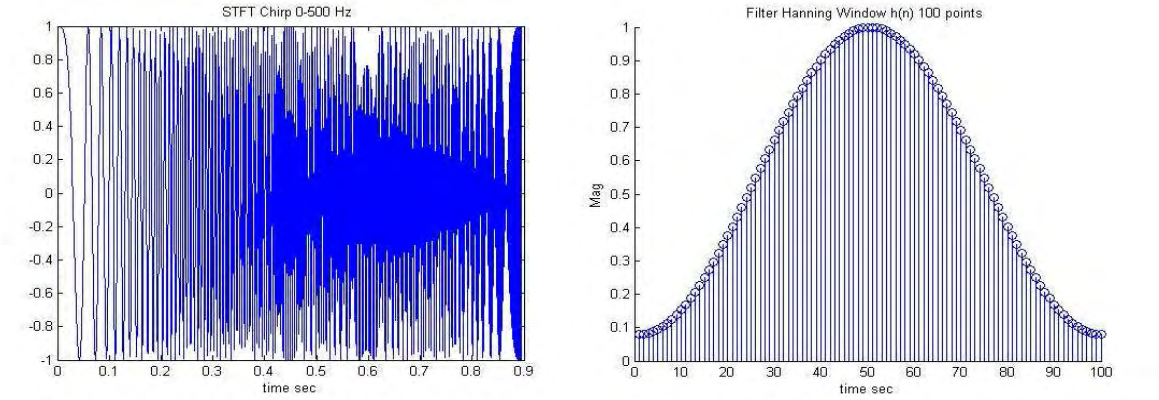
5.3.1 Short Time Frequency Transform (STFT) implementation on the TMS320C6711

For implement this function we will use the following routines implemented using the Code Composer Studio CCS v.2.1 and C language. This algorithm is a modification of the equations explained before using the FFT method for cascade of filters bank.

```
/* Main program */
void main()
{
Input_signal_Padding((double*) X);      // Data input padding to M+L-1
Matrix_modulation((double*) x_padd);  //Data input chirp modulated
FFT_TI();                                          // fft of rows
Haddamart ((double*) h);            // haddamart product with the fft(filter)
In_FFT_TI();                          // Inverse fft
}.
```

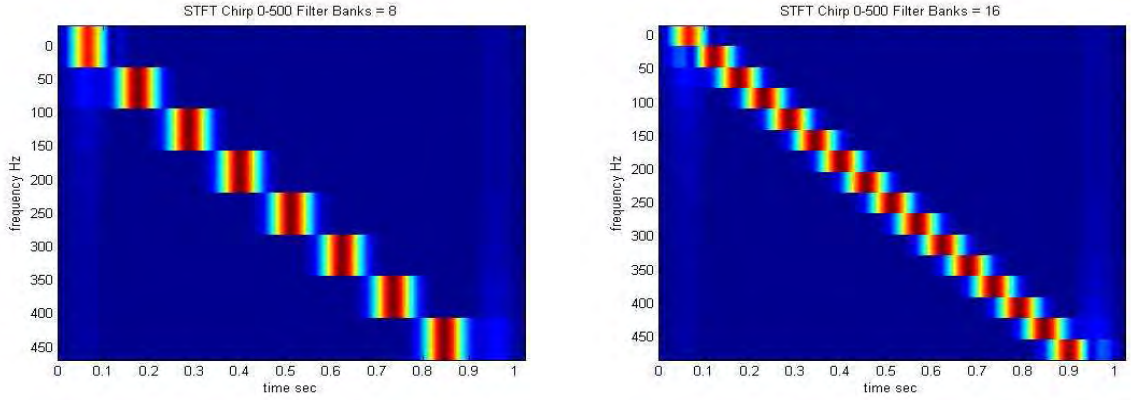
5.3.1.1 Short Time Frequency Transform (STFT) for *N* filters bank x 1024 Signal points

For hardware implementation we simulated a Chirp signal (897-points) using Maltlab®, as shown in Figure 5.6a) and the filter window Figure 5.6 b) correspond a Hanning window (128-points). The computation of the STFT using the filter bank method is performed using the TMS320C6711 processor. Figure 5.7a) represents a STFT of 8 filter banks, Figure 5.7b) represent a STFT of 16 filters bank, Figure 5.7c) represent a STFT of 32 filters bank, Figure 5.7d) represent a STFT of 64, Figure 5.7e) represent a STFT of 128 filters bank filters bank, Figure 5.7f) represent a STFT of 256 filters bank, Figure 5.7g) represent a STFT of 512 filters bank, Figure 5.7h) represent the maximum STFT implemented on the DSP320C6711 a STFT of 1020 filters bank. All the last filters banks had an output matrix N-filter by 1024 points with double precision variables.

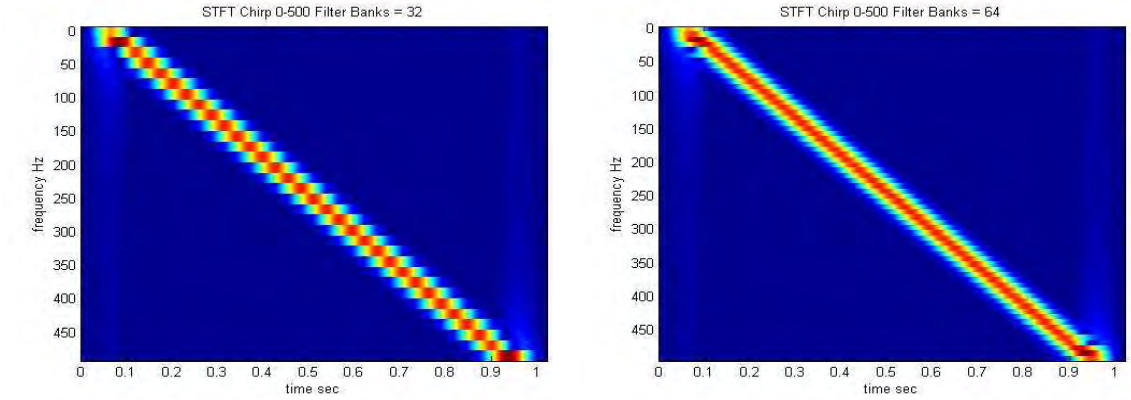


a) b)

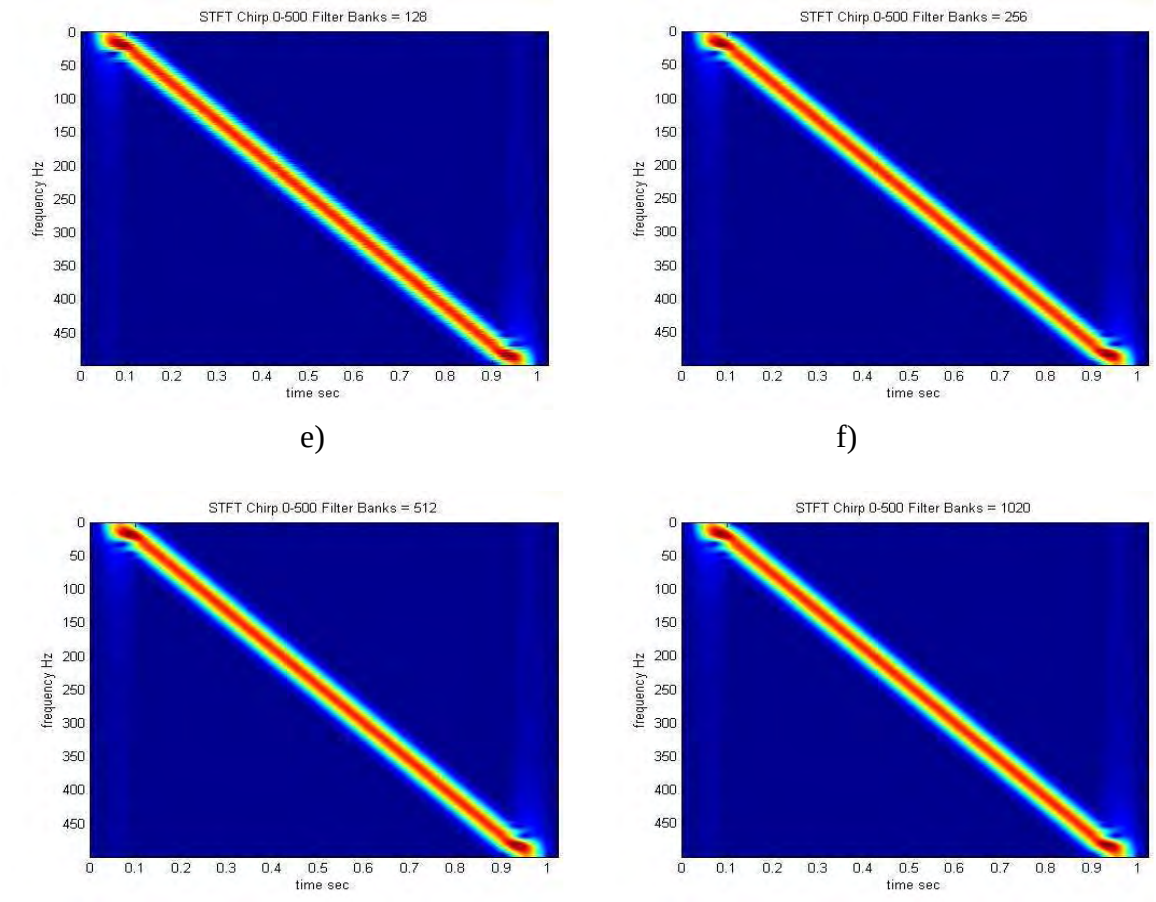
Figure 5.6. Signals to implement the STFT a)Linear Chirp, b) Hanning window filter



a) b)



b) d)



e) f)

g) h)

Figure 5.7. STFT for Chirp Signal of 924 points and different quantity of Filters Banks

5.3.1.2 Short Time Fourier Tranform (STFT) Time Implementations

The table 1 shows the summary of signal points for the Chirp signal, the matrix points for the STFT, the Twiddle Factors spend to compute the FFT , the time consumption of STFT and finally the memory spent in bytes.

STFT Execution Times				
SIGNAL POINTS	MATRIX POINTS	Twiddle Factors	AF Time sec	External Memory Consuption Bytes
897	8 x 1024	512	15 sec	131072
897	16 x 1024	512	31 sec	262144
897	32 x 1024	512	62 sec	524288
897	64 x 1024	512	126 sec	1048576
897	128 x 1024	512	252 sec	2097152
897	256x 1024	512	508 sec	4194304
897	512 x 1024	512	1024 sec	8388608
897	1020 x 1024	512	-out-time-	16711680

Table 1. Summary of STFT implementations on the DSP320C6711

5.3.2 Ambiguity Function (AF) implementation on the TMS320C6711

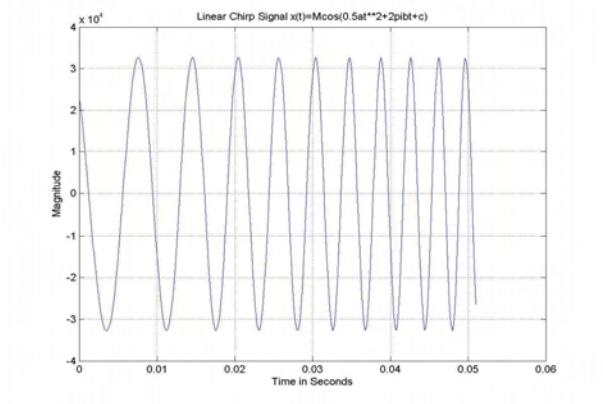
To implement this function we will use the following routines implemented using the Code Composer Studio CCS v.2.1 and C language:

```
/* Main program */
void main()
{
Corr(Sigtx, Sigrx );          // Cross Correlation.
Shift_Signal (Sigrx);         // Matrix of ciclyc shifth received signal. Equation (5.3)
Haddamart (Sigtx);            // Matrix  of Haddamart product with the transmitted signal.
                                 Equation (5.4)

Complex_Complement(); // Complex part introduced to the Matrix

FFT_TI();                     // FFT of the Matrix's Rows . Equation (5.5)
}.
```

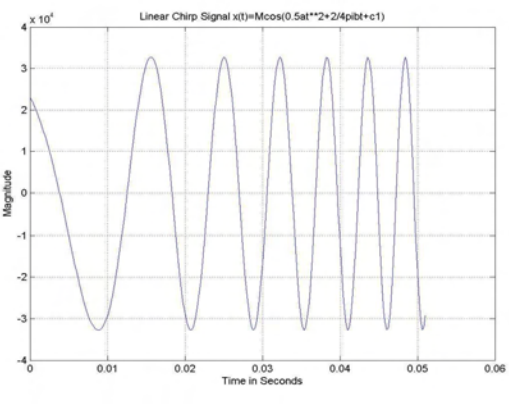
5.3.2.1 Ambiguity Function (AF) for 256x256 points

For a real implementation we simulated using Maltlab® a Chirp signal (256-points) for transmition and the delayed reception signal as used on radar applications. The

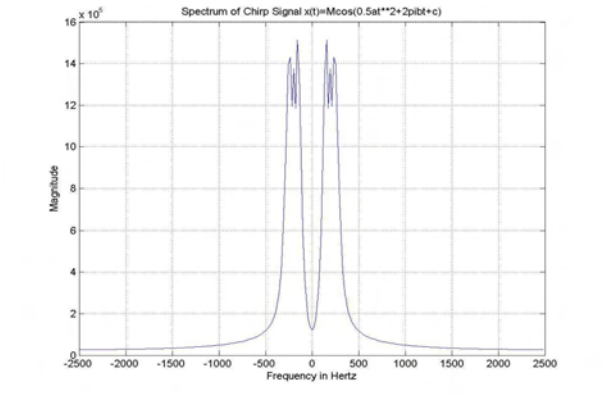
computation of the Cross-Correlation and the AF is performed on the TMS320C6711 processor. Figure 5.8a) represent a 256 points transmitted (Tx) chirp signal. Figure 5.8b) represent a 256 received (Rx) chirp signal. Figures 5.8c) and 5.8d) represent the power spectrum of the Tx and Rx signal. Figure 5.8e) shows the Cross–Correlation between Tx and Rx signal. Finally Figure 5.8f) represent the AF matrix 256x256 where time and frequency maximum represent the difference in time and frequency between the Tx and Rx signal.



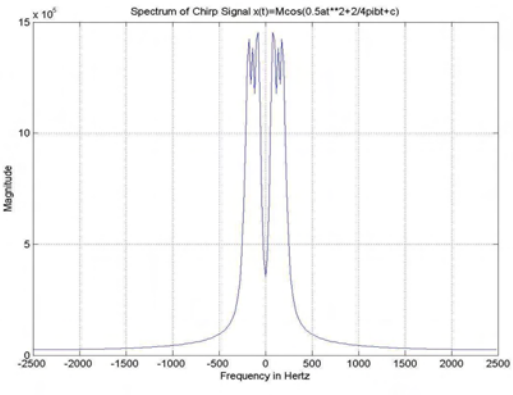
a)



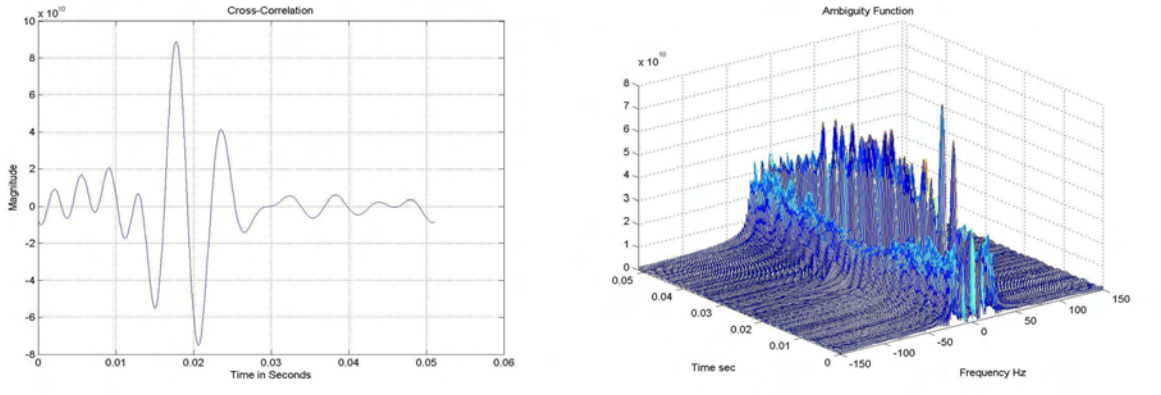
b)



c)



d)

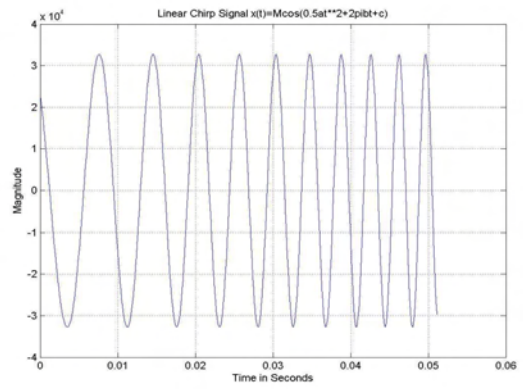


e) f)

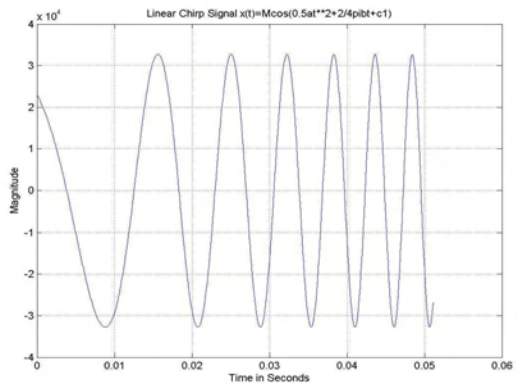
Figure 5.8 Time frequency representation of 256 Tx and Rx Chirp Signal.

5.3.2.2 Ambiguity Function (AF) for 512x512 points

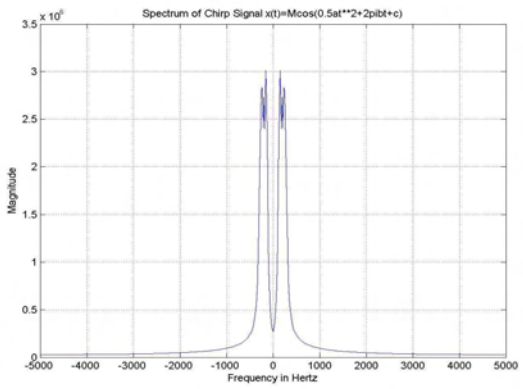
For a real implementation we simulated using Maltlab® a Chirp signal (512-points) for transmition and the delayed reception signal as used on radar applications. The computation of the Cross-Correlation and the AF is performed on the TMS320C6711 processor. Figure 5.7a) represent a 512 points transmitted (Tx) chirp signal. Figure 5.7b) represent a 512 received (Rx) chirp signal. Figures 5.7c) and 5.7d) represent the power spectrum of the Tx and Rx signal. Figure 5.7e) shows the Cross–Correlation between Tx and Rx signal. Finally Figure 5.7f) represent the AF matrix 512x512 where time and frequency maximum represent the difference in time and frequency between the Tx and Rx signal.



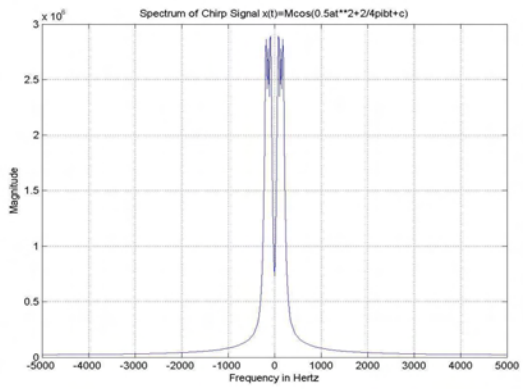
a)



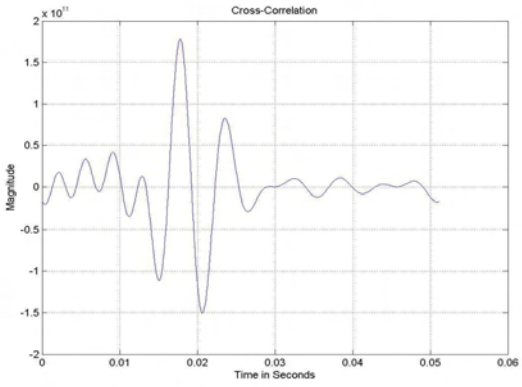
b)



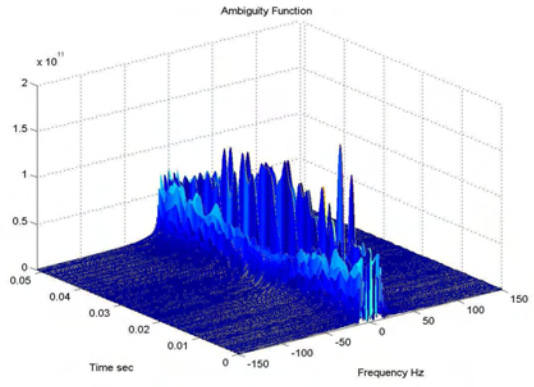
c)



d)



d)

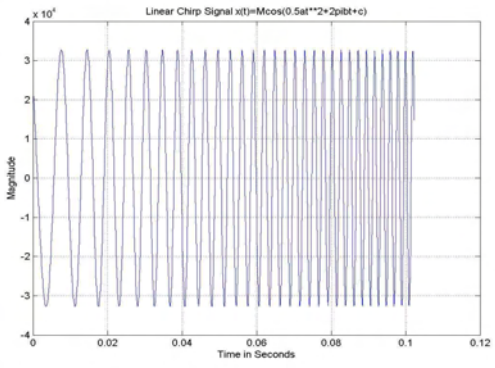


f)

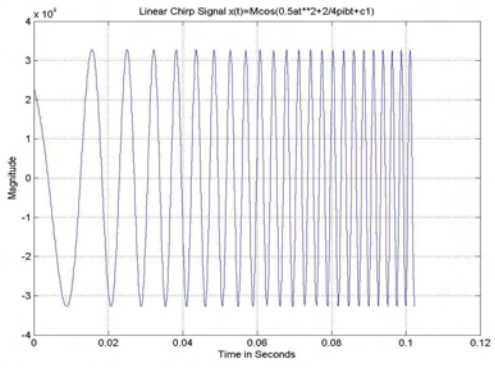
Figure 5.7 Time frequency representation of 512 Tx and Rx Chirp Signal.

5.3.2.3 Ambiguity Function (AF) for 1020x1024 points

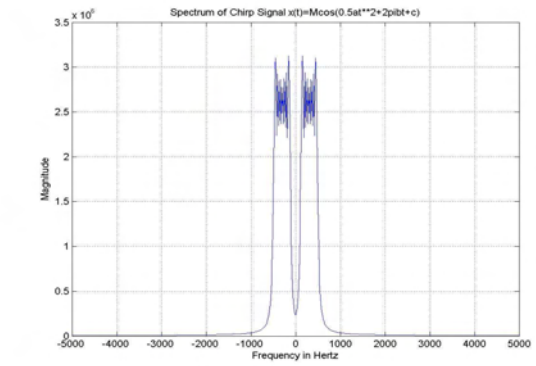
For a real implementation we simulated using Maltlab® a Chirp signal (1024-points) for transmition and the delayed reception signal as used on radar applications. The computation of the Cross-Correlation and the AF is performed on the TMS320C6711 processor. Figure 5.8a) represent a 512 points transmitted (Tx) chirp signal. Figure 5.8b) represent a 512 received (Rx) chirp signal. Figures 5.8c) and 5.8d) represent the power spectrum of the Tx and Rx signal. Figure 5.8e) shows the Cross–Correlation between Tx and Rx signal. Finally Figure 5.8f) represent the AF matrix 1020x1024 where time and frequency maximum represent the difference in time and frequency between the Tx and Rx signal. This is the maximum capability of storage next to 16-Mbytes.



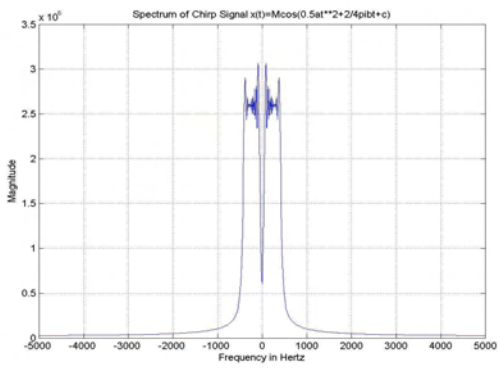
a)



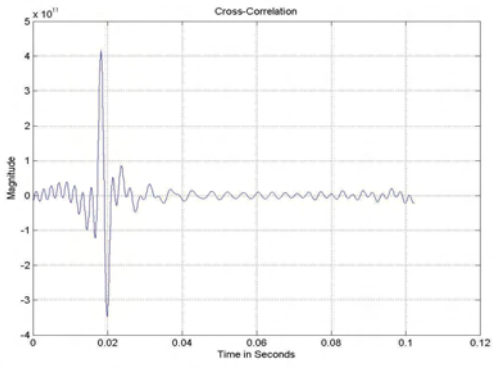
b)



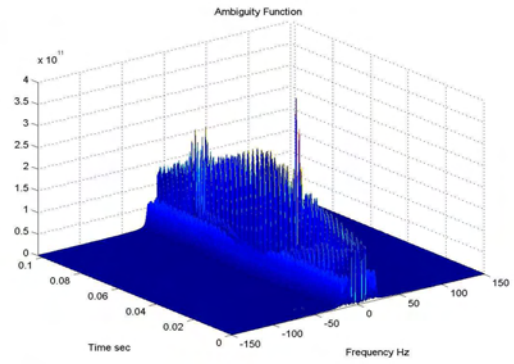
c)



d)



e)



e)

Figure 5.8 Time frequency representation of 1024 Tx and Rx Chirp Signal.

5.3.2.4 Ambiguity Function (AF) Time Implementations

The Table 2., shows the summary of signal points for the Tx signal and Rx, the matrix points for the AF, the Twiddle Factors spend to compute the FFT , the time

consumption for the Cross-Correlation, the time consumption of AF and finally the memory spent in bytes.

Ambiguity Function Times					
SIGNAL POINTS	MATRIX POINTS	Twiddle Factors	Cross-Correlation Time sec	AF Time sec	External Memory Consumption Bytes
256	256 x 256	128	0.217	5.8	1048576
512	512 x 512	256	0.871	20	4194304
1024	1020x1024	512	3.502	80	16777216

Table 2. Summary of AF implementations on the DSP320C6711

Finally on Appendix C, the routines for STFT and AF are shown in C programation language.

Chapter 6

Multirate Sensor Array System based on Kronecker Products

This chapter shows the implementation of multirate concepts (down-sampling) using sensor arrays on digital signal processor (DSP) units such as TMS320C6711 DSK. This implementation based on Kronecker products formulation for mapping from hardware configurations to software algorithms, centers on a scalable and modular approach. The scalable approach to this implementation implies that the function and structure of each algorithmic formulation should adapt to changes in the size of the sensor array and on the length and dimensions of the signal to be processed. The modularity approach implies that each system can be composed by a set of modules with flexible interconnectivity, and reconfigurability will be obtained.

6.1 Computational Sensor Array System

When individual sensors are placed in a regular grid as shown in Figure 6.1, this produces a sample array aperture of the received sound signal (6.1):

$$S_r(t) = \left[S_{r0}(t) S_{r1}(t) .... S_{rN-1}(t)\right], \tag{6.1}$$

where *N* means the number of sensors. The row vector of information $S_{rK}(t)$, stores the intensity of the front sound wave at different instants *t*, transducer from the respectively microphone through the A/D converter. For a specific time *t* we obtain a row vector of

information corresponding of space sampling difference distances *d*. For last considerations, let us work in terms of vectors and matrixes to introduce the Kronecker products, in order to develop algorithms for computational structures such as DSP processors. Other important consideration when we use array of sensors consists in the improvement of the *SNR*, as demonstrated in (4.12). The original $(SNR)_0$ is enhanced by the number of sensors *N*.

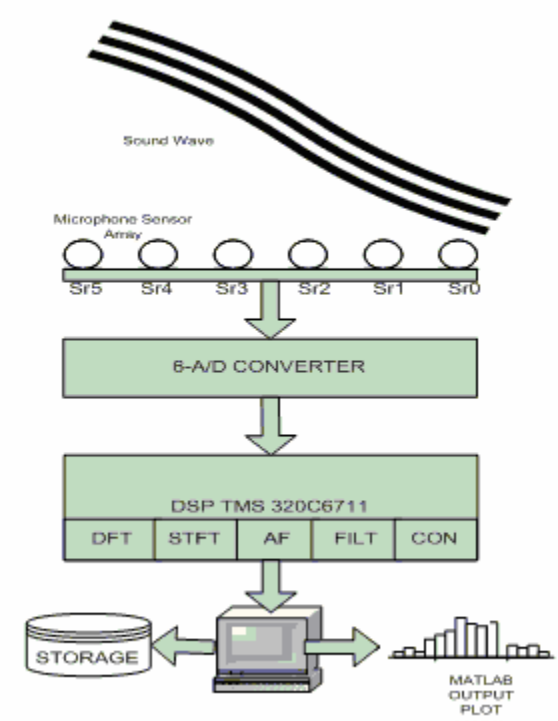


Figure 6.1 Computational Sensor Array System

The computational sensor array system Figure 6.1 consists on the initial concept to develop a physical implementation using a six-sensor array of microphones (Appendix F. for tools utilized). After that, a six-channels A/D converter, and then a DSP processor receive all the digitalized six-channels signals and perform different signal processing operations such as Discrete Fourier Transform (DFT) for spectral analysis, Short Time

Frequency Tranform (STFT) for time-frequency representations, Ambiguity Function (AF) for radar applications, Filter and Convolution for signal analysis. Then a PC interface for final storage, and output plots using Matlab®.

In this Chapter we will implement a physical Beamforming application defined for detecting the direction of arriving (DOA) signal to an array of sensors. The following sections explain a block diagram for the actual implementation of this system.

6.1.1 Data Acquisition Configuration

This system, Figure 6.2, assumes a sound wave arriving to the unit linear array (ULA) of sensors with defined distance *d.* After that, the signal conditioner circuit put the correct offset voltages to the A/D daughter card, then the digitalized signals are obtained for the DSK320C6711 microprocessor, and finally an output *file.h* is stored on PC to be read for Matlab® and plot [15].

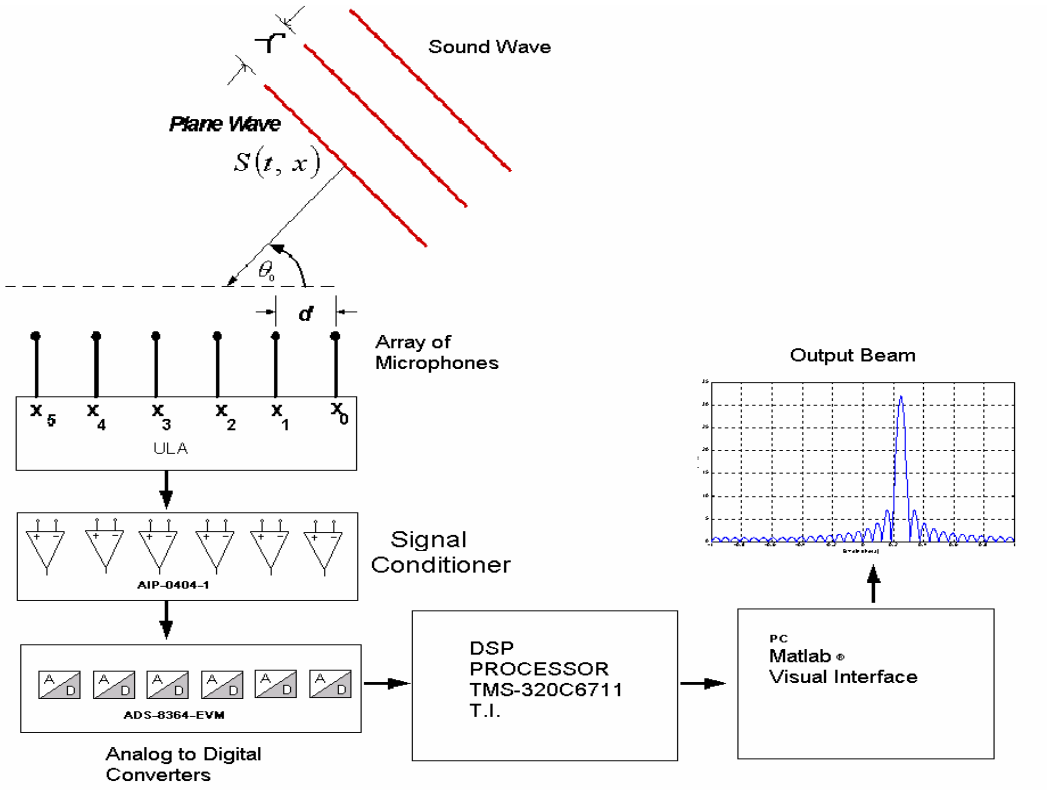
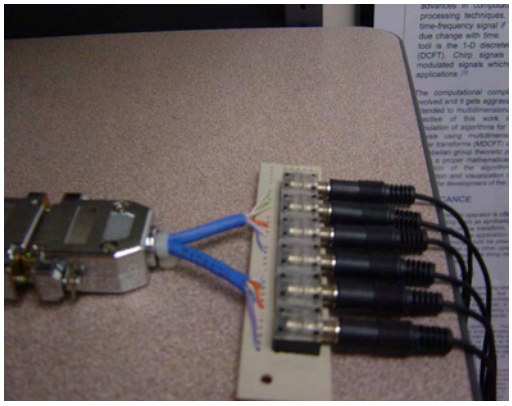


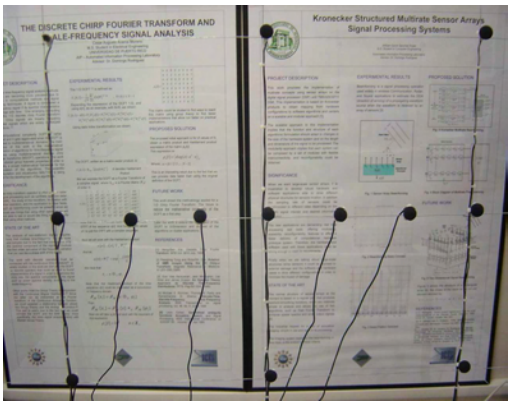
Figure 6.2 Computational Unit Linear Array (ULA) sensor system.

6.1.2 Data Acquisition Implementation

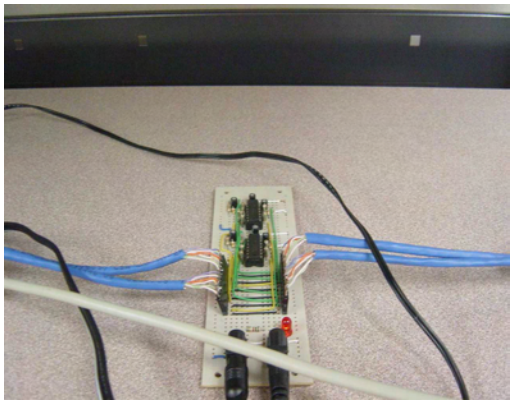
Figure 6.3 shows the physical implementation of unit dimensional array (ULA) using six microphones 6.3a), six analogs inputs 6.3b), a signal conditioner circuit 6.3c), the daughter card ADS8364 with 6 A/D converters and the DSK320C6711 microcontroller 6.3d).



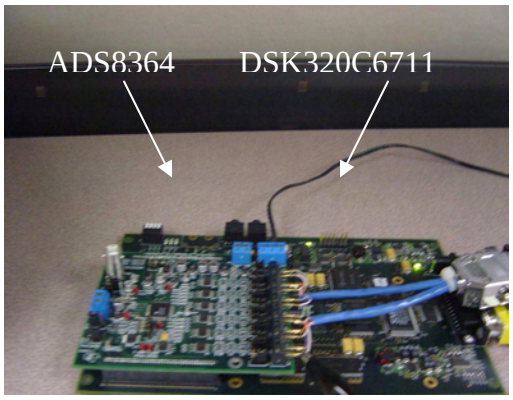
b) Analog Data Input 6ch



a) Unit Dimensional Linear Array



c) Signal Conditioner AIP-0404-01



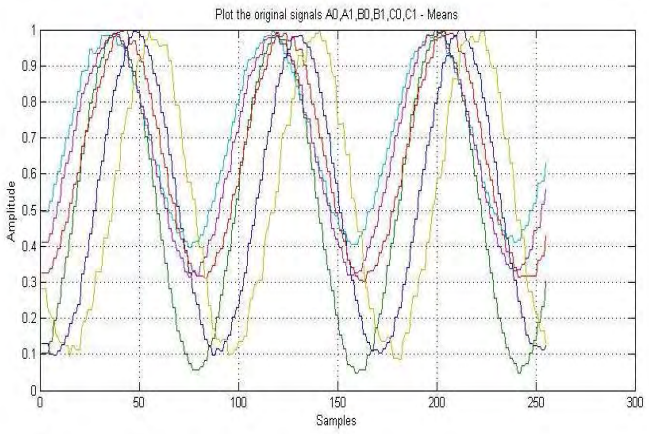
d) A/D and DSK-MICROPROCESSOR

Figure 6.3 Physical computational implementation of unit dimensional array (ULA)

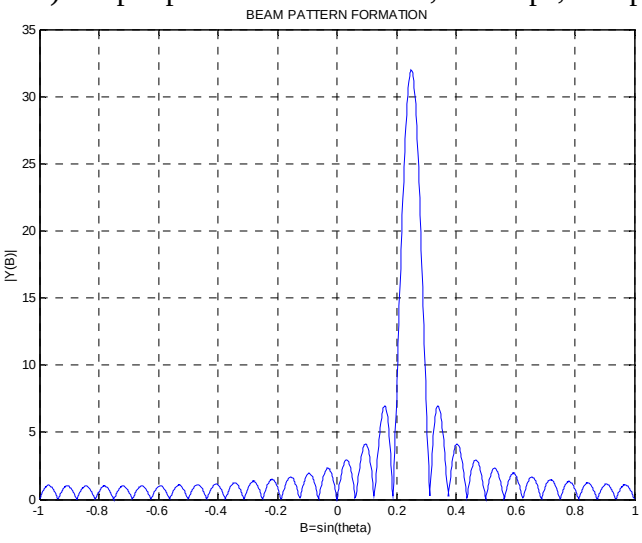
6.1.3 Six channels A/D of sine wave sound

This is a probe sound signal to determine the functionality of the unit dimensional array (ULA) system implemented using the configuration of Figure 6.3. The A/D

converter ADS8364 of Texas Instruments offers 6 channels (A0, A1, B0, B1, C0, C1) of analog to digital conversion, Figure 6.4 shows the six channels sampling at 128Ksps and 256 points, the six channels perform the sampling at the same time, and the storage data of each channel was plotted using a Matlab® as visual interface Figure 6.4a), and Beam pattern Figure 6.4b).



a) Output plot from 6 channels, 128Ksps, 256 points



b) Output detected for conventional Beamforming

Figure 6.4 Six-sensor array Beam forming

6.2 Kronecker products for Multirate Sensor Array Beamforming

Beamforming is a signal processing operation used widely in wireless Communication, Radar, and Sound applications to estimate the DOA of a propagating waveform source when the waveform is received by an array of sensors [19].

When the input array sensors grown to large scale implementations a partial Beamforming is performed decomposing the input vector x of length $L=NM$ in M segments of length N, as

$$\left(F_N \otimes I_M\right)\cdot x. \tag{6.1}$$

Thus, we define a Kronecker parallel factor as a diagonal matrix

$$F_N \otimes I_M = \begin{bmatrix} F_N & 0 & \cdots & 0 \\ 0 & F_N & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \cdots & 0 & F_N \end{bmatrix}. \tag{6.2}$$

Multi-Beamforming is performed collecting and combining the output information, channel by channel, of every DFT, on this process the increasing factor of *SNR* is obtained as demonstrated in equation (4.12). Thus, we obtain a matrix of size NM

$$I_N \otimes U^T{}_M = \begin{bmatrix} 1 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots & 0 & 1 & \ddots & \vdots & \cdots & 0 & 1 & \ddots & \vdots \\ 0 & \ddots & \ddots & 0 & \vdots & \ddots & \ddots & 0 & \cdots & \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 1 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix}. \tag{6.3}$$

The mathematical formulation using Kronecker products for the multi-beamforming (*MB*) operation becomes

$$MB = \left(I_N \otimes U_M^T\right) \cdot \left(F_N \otimes I_M\right) \cdot [\downarrow S] \cdot x, \qquad (6.4)$$

where $[\downarrow S]$ means the down sampling process performed by the Multirate block. Figure 6.5 shows the implementations of the last formulations. The specific block of Multirate pre-processing is shown on Figure 6.6, where the product $[\downarrow S].x$ is implemented.

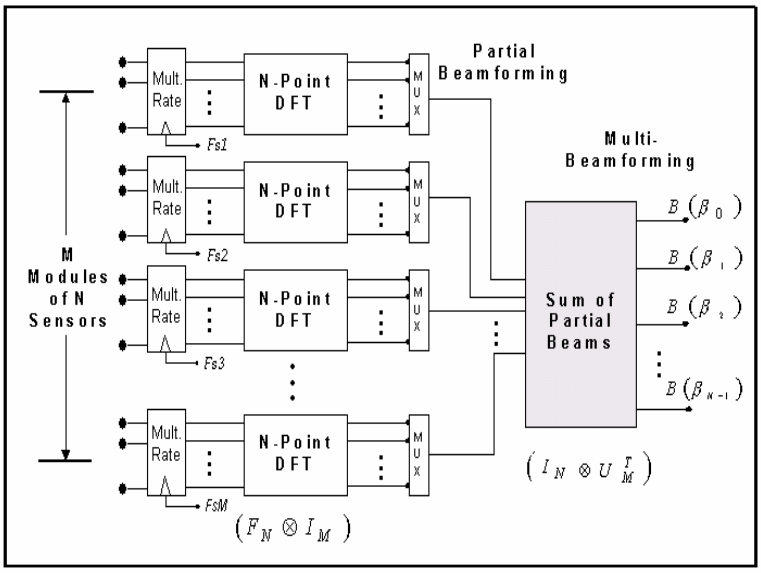


Figure 6.5 Kronecker Multirate Beamforming implementation blocks.

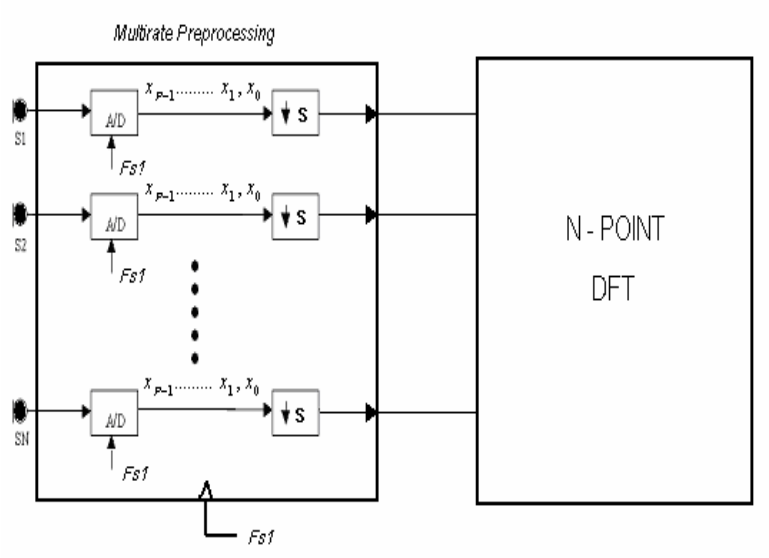


Figure 6.6 Block diagram of Multirate Pre-processing

6.3 Multirate Beamforming with 32 Sensors Array using Kronecker products and implemented on the DSK320C6711 processor.

The implementation of the Beamforming using 32 sensors array (ULA) Figure 6.7, is based on the concepts explained in the section 6.2. The large scale implementation of sensors, in this case 32 sensors, cannot be physical implemented using an array of microphones. For that reason, we simulated the incoming data from 32 sensors using Matlab® and stored on the DSK320C6711 to perform real computation of the Beamforming. The computation performed by the DSP processor, the output matrix *B* stored on the memory of the DSK320C6711 is translated then to the PC and the output Beamforming could be plot using Matlab® as visual interface.

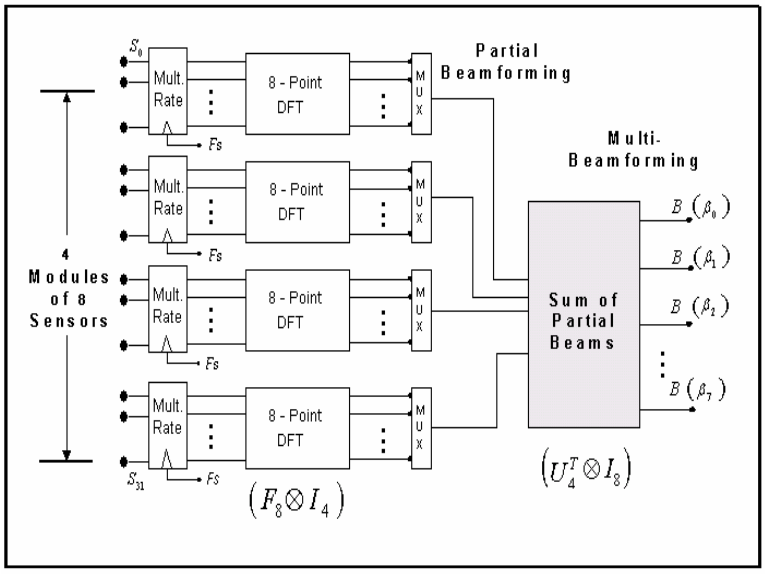


Figure 6.7 Kronecker Multirate Beamforming implementation using 32 Sensors Array

Figure 6.8 shows the output plot of the Beamforming for 32 sensors array without down sampling or *S=0,* the number of samples vectors are *256* each one with *32* positions to built an input matrix of *256*x*32*, then each input vector (*32* positions) is divided in 4 modules of 8 sensors to perform the 8-points Discrete Fourier Transform (DFT) and finally a coherent sum for partial beams. The dimensions of the output matrix *B* is *256*x*8* , the plot of each entire column (256-length) of this matrix represent each of the beams (8 in total) detected as shown in Figure 6.8.

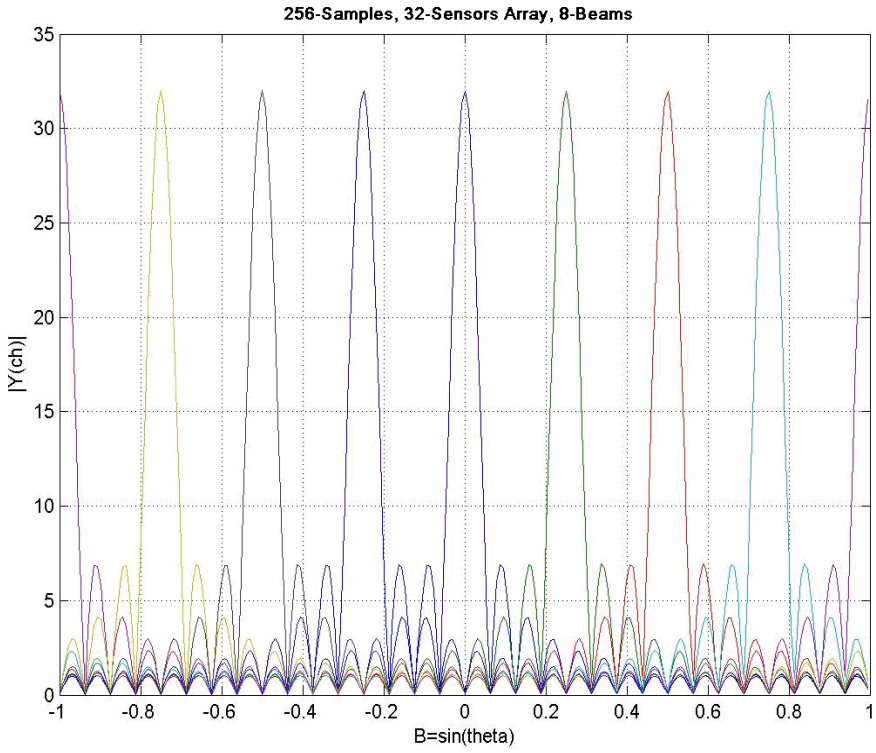


Figure 6.8 Beam Pattern Detected without down sampling, *S=0.*

The output means that the detected beams had to be of incidence angle with respect of sensor array such as 0°, 15°,30°, 45°,90°, 135°,150°, 165°.To diminish the computational effort we can use a down sampling by $S = 2$ in order to reduce the input sample vectors from *256* to *128*, each one with *32* positions to built an input matrix of *128*x*32*, then each input vector (*32* positions) is divided in 4 modules of 8 sensors to perform the 8-points Discrete Fourier Transform (DFT) and finally a coherent sum for partial beams. The dimensions of the output matrix *B* are *128*x*8*. The plot of each entire column (128-length) of this matrix represent each of the beams (8 in total) detected as shown in Figure 6.9.

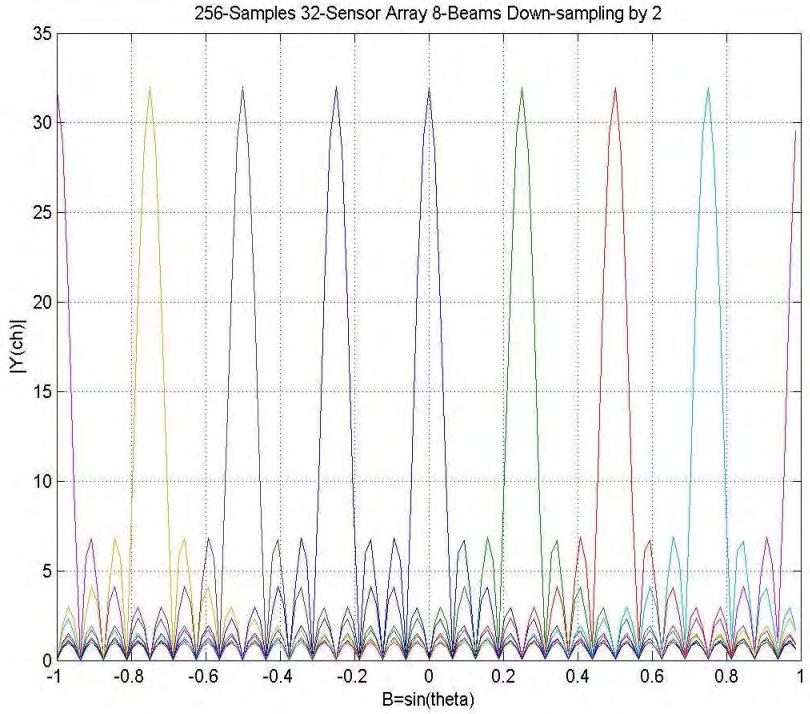


Figure 6.9 Beam Pattern Detected with down sampling, *S=2.*

The output means that the detected beams had to be of incidence angle with respect of sensor array such as 0°, 15°,30°, 45°,90°, 135°,150°or 165°. This output shows the same incidence angles as Figure 6.7. Both plots, Figure 6.7 and Figure 6.8 have the same amplitude $|Y(ch)|$ and the same angles detected, for that reason the down sampling process offers a way to reduce computation time without extremely final resolution affected of the Beamforming system. Figure 6.10 and 6.11 plot the output beam using a down sampling factor *S=4,* and *S=8.*

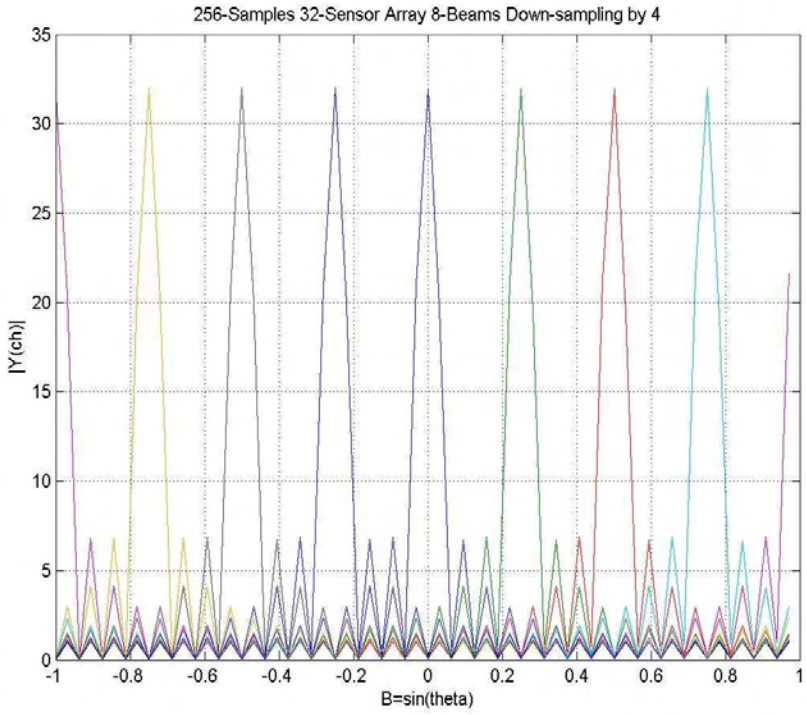


Figure 6.10 Beam Pattern Detected with down sampling, *S=4.*

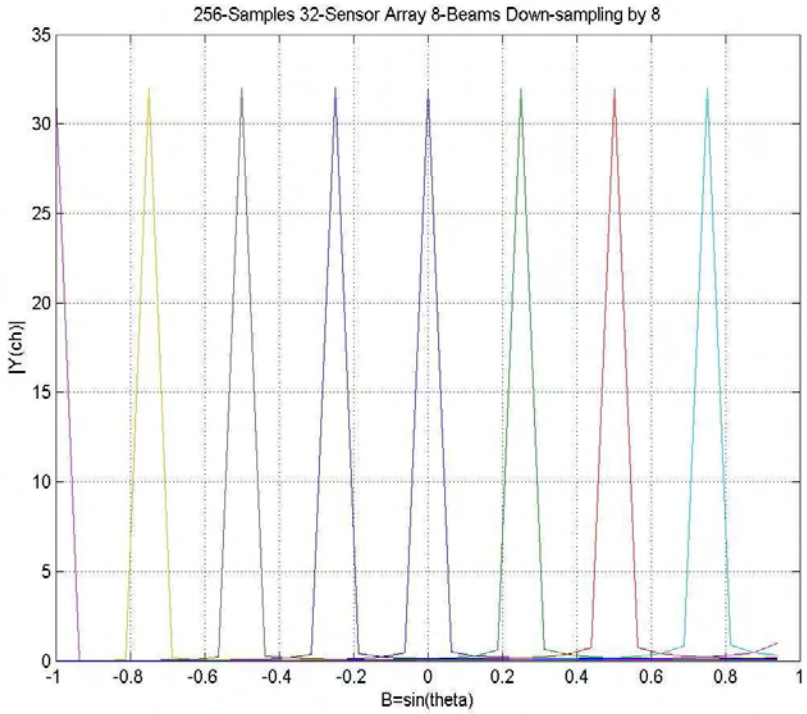


Figure 6.11 Beam Pattern Detected with down sampling, *S=8.*

Table 3 shows the different time of execution for down sampling reduction by *S=0,2,4,8* and different size of the input matrix.

Beamforming Time of Execution					
SENSOR ARRAY	*S*	INPUT MATRIX	*L = M*N*	CLOCK PERIODS	BF TIME sec
32	0	256x32	8*4	214007410	0.214
32	2	128 x 32	8*4	107001190	0.107
32	4	64 X 32	8*4	53482606	0.053
32	8	32 X 32	8*4	26741810	0.027

Table 3. Execution Times for 32 Sensor Array with *S=0,2,4,8* Down samples

Finally, Appendix D shows the algorithm used to compute the Beamforming application on the TMS320C671 DSP processor.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This work presented Kronecker Structures for Multirate Sensor Array signal processing systems. The concepts of Multirate and Sensor array were studied to be implemented on an actual DSP floating point processor TMS320C6711 from T.I. The mathematical framework of this work is based on Kronecker products that shown an important tool to develop and implement modularity and scalability hardware and software on signal processing systems.

The hardware and software tools utilized on this work offered a practical way to evaluate the maximum capabilities of the first floating point (32 bits) DSP processor TMS320C6711 from T.I. in the field of Multirate, Sensor Array and Time-Frequency representations. Tables 1, 2, and 3 summarized the maximum capabilities on memory and time execution utilized by the DSP processor. The DBT tool box for Matlab® offers an interesting approach in the field of Radar applications and sensor array system to use Matlab® as a developed environment.

The scalability and modularity approach based on Kronecker products to formulate software and hardware implementations in the field of Sensor Array shown a useful tool, because the quantity and configurability of Unit Linear Arrays (ULA) sensors could be changed for specific applications or possible external damage of sensors, without change DSP routines.

A complete hardware and software physical implementation using an array of six-microphones was performed with the six channels, 16bits A/D converter, ADS8364 evaluation module of T.I. and the AIP-0404-1 conditioner signal. In order to probe the concepts for a realistic application, a 32 sensor array signal was simulated using Matlab®, and the total evaluation was performed on the DSP to obtain an 8 Beamforming plot. To reduce execution time, a Down-sampling processing was implemented to reduce the data computational effort, Table 3.

Different time-frequency algorithms such as Short Time Fourier Transform (STFT) and Ambiguity Function (AF) for complete characterization of Chirp (Radar) signals were implemented using C language and double precision variables (64-bits) for code execution and storage respectively. In order to understand the advantages and disadvantages of this implementation, different long of signals were utilized and the Tables 1 and 2, showed the maximum capability for storage and output matrixes, obtained with the last transforms.

The floating point architecture of the DSP processor TMS320C6711, offers a new broadband field for applications were computational accuracy is required. This was the case of the STFT, AF and, Cyclic Correlation algorithms. The rising number of bits utilized from Analog to Digital converter opens the possibility to conjugate embedded DSP processors with enough computational power and the accuracy required for specialized applications on signal processing.

7.2 Future Work

In order to compute actual applications using DSP processor, increasing the number of sensors for high scale implementations on Multirate Sensor Array Systems, a multi-DSP processor boards can be studied to obtain a higher computational and memory resource for real time applications in the field of Time frequency representations, and Sensor Array processing.

The rising embedded applications on digital signal processing and sensor array systems open the development possibility of new applications in the field of Power Quality, Automotive Control and, Sound Systems, using well known signal processing techniques. This is because some years ago the computational requirements, the algorithm complexity and A/D's resolution, were not efficient.

Bibliography

[1] **J. R. Jonson, R. W. Jonson, D. Rodríguez, R. Tolimieri,** *"A Methodology for Designing, Modifying, and Implementing Fourier Transform Algorithms on Various Architectures"*.
Journal of Circuits, Systems and Signal Processing Birkhäuser, Vol. 9, No. 4, 1990.

[2] **D. B. Ward, Z. Ding, R. A. Kennedy**, "*Broadband DOA Estimation Using Frequency-Invariant Beam-Space Processing*". IEEE Trans. On Signal Processing, vol. 46, no. 5, pp 1463-1469, May 1998.

[3] **D. B. Ward, R. A. Kennedy, R. C. Williamson** *"Theory and design of broadband sensor arrays with frequency invariant far-field beam patterns"* Journal of Acoustical Society of America, vol 97, no.2, pp. 1023-1034, Feb. 1995.

[4] **M. Ghavami and R. Kohomo**, *"Rectangular arrays for uniform wideband beamforming with adjustable structure"*. In Proc. WPMC'00, Bangkok, pp. 93-97, Nov. 2000.

[5] **A. Quinchanegua, D. Rodríguez**, "*Kronecker DFT Multi-Beamforming Implementation Approach"* . Proceedings of the IASTED International Conference Circuits Signal and Systems May 19-21, Cancún, México May 2003.

[6] **J.C Chen, L.Yip, H. Wang, D. Maniezzo, R.E. Hudson, J. Elson, K. Yao, D. Estrin** *"DSP implementation of a distributed acoustical beamforming on a wireless sensor platform"* . IEEE International Conference on Acoustics, Speech, and Signal Processing ICASP 2003, Hong Kong, China April 2003.

[7] **S. K. Mitra**, "*Digital Signal Processing: A Computer-Based Approach "* Chapter 10. Mac Graw Hill, NY 2001, ISBN 0-07252261-5.

[8] **P. A. Regalia, S. K. Mitra**, *"Kronecker Products, Unitary Matrices and Signal Processing Applications"*. SIAM Review, Vol 31, No. 4, pp 586-613, December 1989.

[9] **G.G. Pechanek, C.J. Glossner, Z. Li, C.H.L. Moller, and S. Vassiliadis**, "*Tensor Product FFT's on M.f.a.s.t.: A Highly Parallel Single Chip DSP*", In Proceeding of DSP95 - Digital Signal Processing and Its Applications, eighth paper, pp 1-10, October 1995, Paris, France.

[10] **S. Unnikrishna Pillai**, *"Array Signal Processing"* Chapter 2.Springer-Verlag, NY 1989, ISBN 0387969519.

[11] **B. E. Nelson**, "*Configurable Computing and Sonar Processing Architectures and Implementations*", Proceedings of the Asilomar conference on Signal, Systems and Computers, pp 56-60 Vol. 1. Monterrey CA, 2001.

[12] **R. B. Mahafza**, *"Radar System Analysis and Design using Matlab"* Chapter1 Chapman & Hall/CRC, 2000, ISBN 1584881828.

[13] **S. Björklund, D. Rejdemyhr** *"A Matlab Tool Box for Radar Array Processing Paper"* IEEE Proceedings of ISSPA, pp 547-550. Brisbane Australia, August 1999.

[14] **W. D. Sánchez, C. A. Aceros, D. Rodríguez** "*Time Frequency Analysis Using Sensors Array Based on Kronecker Products*" Proceedings of the IASTED International conference in Circuits Signal and Systems. Nov 28-Dec.1,2004. Clearwater Beach, FL, USA.

[15] **J. S. Lim, A.V. Oppenheim** "*Advance Topics in Signal Processing "* Chapter 6. Prentice Hall, NJ 1988, ISBN 0130131296.

[16] **M.R. Pornof** "*Time-Frequency Representation of Digital Signals and Systems Based on Short-Time Fourier Analysis*" IEEE transactions ASSP-28 No.1. February1980.

[17] Texas Instruments Inc., DSP Developers' Village, C6000™ Platform http://focus.ti.com/docs/prod/folders/print/tms320c6711.

References

[18] TMS320C62xx DSP Library Programmer's Reference, Literature Number SPRU402A, April 2002. Texas Instruments Inc.

[19] **B. Vrcelj** "Multirate Signal Processing Concepts in Digital Communications" Thesis Ph.D. California Institute of Technology, Pasadena California, 125 pp.

[20] **D.B. Ward, M. S. Brandstein** "*Grid-Based Beamforming for Room-Environment microphone Array*" Proc. 1999 IEEE Workshop ASPAA 17-20, NY, Oct. 1999.

[21] **D. de Vries, M. M. Boone** "*Wave Field Synthesis and Analysis Using Array Technology*" Proc. 1999 IEEE Workshop ASPAA , NY, Oct 17-20. 1999.

[22] **D. B. Ward, T.D. Abhayapala** "*Reproduction of a Plane-Wave Sound Field Using an Array of Loudspeakers*" IEEE Transactions on Speech and Audio Processing, Vol. 9, No. 6, September 2001.

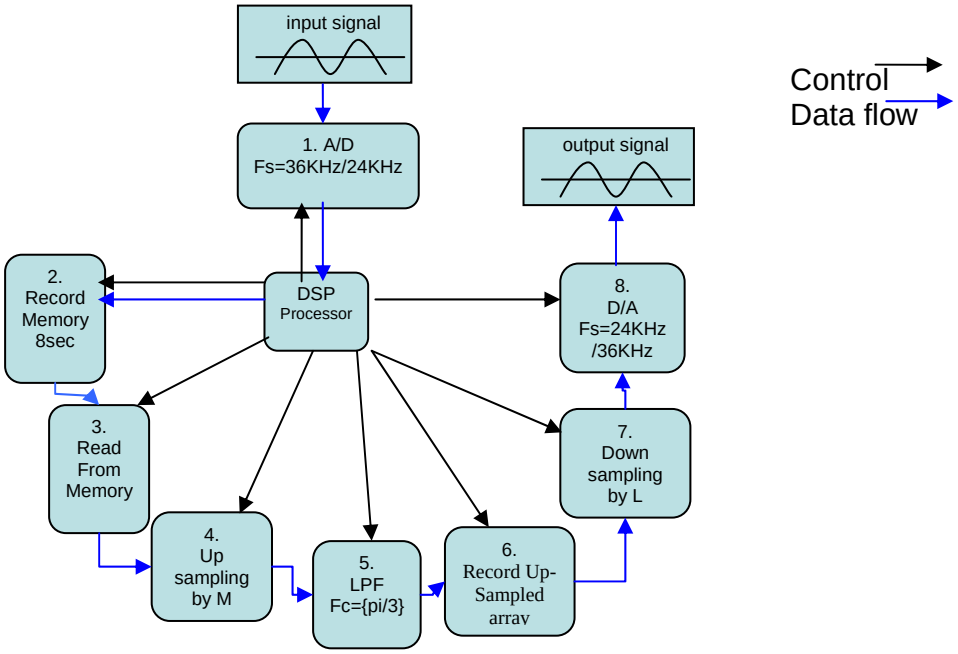
[23] **C. A. Aceros, W.D. Sánchez and D. Rodríguez** "*The Discrete Fourier Transform and Scale-Frequency Signal Analysis*" Proceedings of the IASTED International conference in Circuits Signal and Systems. Nov 28-Dec.1,2004 Clearwater Beach, FL, USA.

A. User Guide to Multirate System

Introduction

This project consists in the utilization of the TSM320C6711 DSP processor and the PCM3003 audio daughter card to implement multi-rate system. The project consists of a fractional rate change by 2/3 and 3/2 samples from the input signal. Based on the concepts of up-sampling, down-sampling and FIR (Finite Impulse Response) filters, we are going to develop the following processes which are the base of a Multi-rate system.

System Flow Chart



9. Input signal: Generator, microphone or music with maximum frequency of 4KHz.

10. A/D converter: Sampler input signal at a rate of 36KHz or 24KHz.
11. Record Memory: It stores the sampled signal into a buffer array of size 36Ksps or 24Ksps * 8sec.
12. Read from Memory: With a pointer we are able to recover the samples stored in the buffer memory.
13. Up-sampling by M: introduces (M-1) zeros between each sample read from the buffer-memory.
14. LPF: It removes spectral images, interpolates and limits frequency for the down-sampling stage.
15. Down-sampling by L: Reads every L sample from the output memory.
16. D/A converter: it converts from digital to analog with a sampling rate of 24 KHz or 36 KHz.

The output signal is going to be reproduced with the same frequency as the original if the D/A converter operates at frequency 24 KHz or 36KHz. This is obtained from the following relation: $Fsout = \frac{36KHz * 2}{3} = 24KHz$. $Fsout = \frac{24KHz * 3}{2} = 36KHz$

We are going to implement all of the above mentioned stages in the DSP board and the Code Composer Studio development software.

Hardware Settings

1. The DSK board, TMS 320c6711, should be connected to the computer through the parallel port and be connected to the power supply.

2. The DSK board should have installed the Audio Daughter Card, PCM3003, for this particular project. The Audio Daughter Card should be installed on the DSK according to the instruction of the manufacturer.

3. The Audio Daughter Card should have the following jumper configuration:

JP1, JP2, JP4, JP9, JP10 disabled

JP3 connection from the Audio Daughter Card with the DSK.

JP5 MCLK short pins 1 and 2.

JP6 FSCTRL short pins 3 and 4.

JP7 DGND short pins 1 and 2.

JP8 DVDD short pins 1 and 2.

JP11 BITRATE short pins 5 and 6.

JP12 SAMPLERATE short pins 3 and 4.

Note:

The jumper configuration depends on the square solder on the bottom side of the Audio Daughter Card. It should be noted that the pin #1 is the square solder. The complete jumper configuration is illustrated in **Figure 1.**

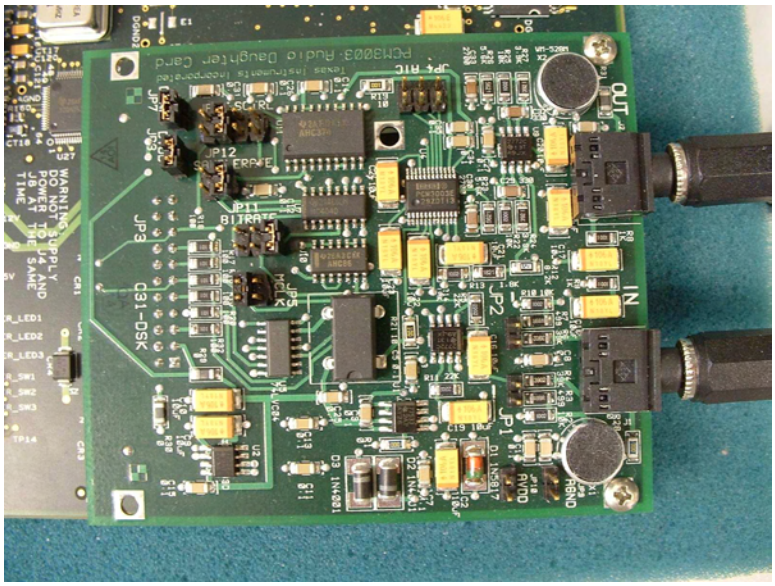


Figure 1 Audio Daughter Card jumper configuration.

4. The project uses standard stereo plugs with both right and left channels connected to the Audio Daughter Card.

Project Creation

There are two ways to create the *Multirate* project from an acquired file.

Fast and Easy Way:

1. Select the given folder named *Mulirate* and save it on the following path:

    c:\ti\myprojects\

the final path to the stored file is:

    c:\ti\myprojects\Multirate

2. Open the CCS and Select Project => Open Project and search for the Multirate folder and open it. Then choose the file: *Multirate.pjt* and open it.

3. Now jump to the section: **Building and Running the Project** for building and running the *Multirate* project.

Step by step project creation

1. To create the project file *Multirate.pjt*. Select Project => New. Type *Multirate* for project name as shown in figure 2a. This project file is saved in *Multirate* (the folder you created in c:\ti\myprojects). The .pjt file stores project information on build options, source file names, and dependencies.

2. To add files to project. Select Project => Add files to Project. Look in *Multirate.* Files of type C Source Files. Open the C sources files *Filter.c , interrupts.c , mcbsp1.c , Miltirate System.c , stereo.c and, switches.c.* Open (to add to project) one file at a time; or place the cursor to one of these files, then to the other while holding the Shift key, and press Open. Click on the "+" symbol on the left of the Project Files window within CCS to expand and verify that the C sources files have been added to the project.

3. Select Project => Add Files to Project. Look in *Multirate.* Use the pulldown menu for Files of type: and select ASM Source Files. Double-click on the assembly source file *vectors.asm* to open/add it to the project.

4. Repeat step 3 but select Files of type: Linker Command File, and add the linker command file *lnk.cmd* to the project.

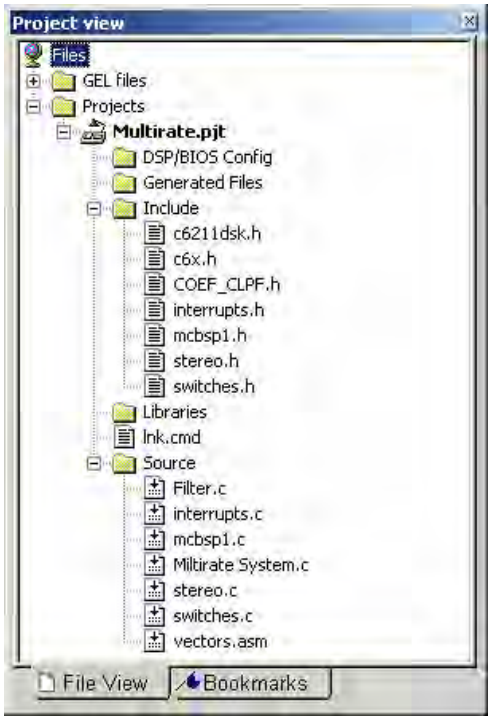
5. Verify that the linker command (.cmd) file, the project (.pjt) file, the C (.c) files, and the assembly (.asm) files have been added to the project.

6. Note that there are no "includes" files yet. Select Project => Scan All Dependencies. This add/includes the headers files: *c6211dsk.h , COEF_CLPF.h , interrupts. , mcbsp1.h, stereo.h , switches.h , and c6x.h.* The last one is included in the CCS files, the others have to be copied (transferred) from the accompanying disk supplied by us. finally the figure 2b shows all the included files of this project.

Compiler Option: Select Project => Build Options. **Figure 3** shows CCS window Build Options for the compiler. Look at the figure and fill with the exact values presented at the **Figure 3a**.

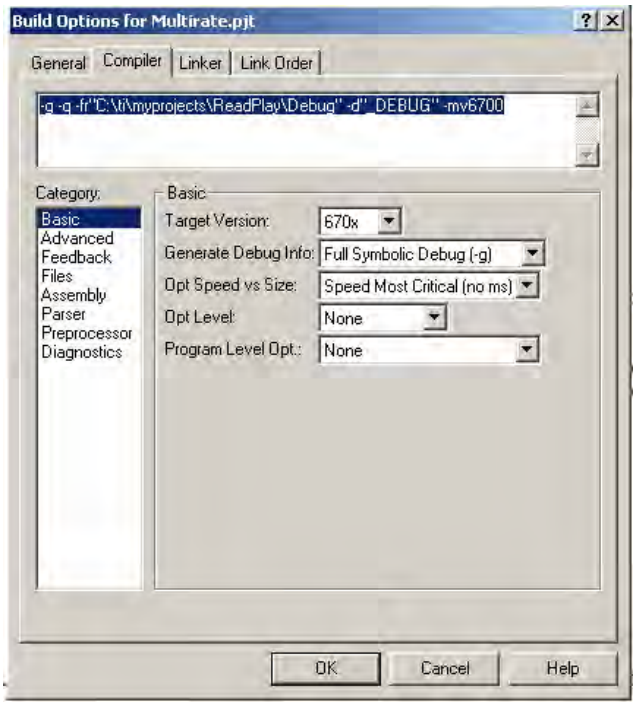
Linker Option: Click on Linker (from CCS Build Options) and select Absolute Executable (for Output Module), *Multirate.out (*for Output Filename), and Run-time Autoinitialization (for Autoinit Model). The output filename defaults to the name of the *.pjt* filename. The linker option should be displayed as in **Figure 3b**.

(a)

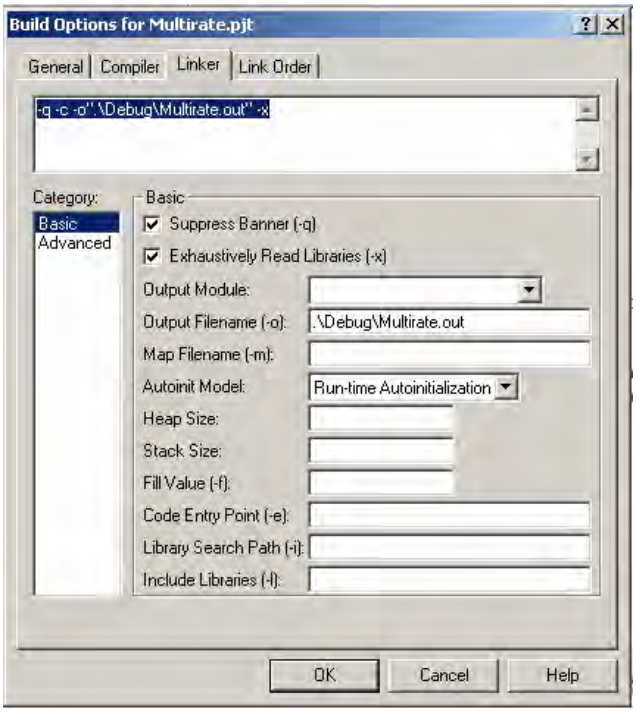


(b)

Figure 2 CCS Project View window for *Multiraet:* (a) creating project; (b) project files



(a)



(b)

Figure 3 CCS Build options: (a) compiler; (b) linker

Building and Running the Project

The project *Multirate* can now be built and run.

1. Build this project as *Multirate.* Select Project => Rebuild All. Or press the toolbar with the three down arrows. This compiles and assembles all the C files and assembles the assembly file *vectors.asm.* If the compilation was successful, the executable file *Multirate.out* is created that can be loaded into the C6711 processor and run. Note that the commands for compiling, assembling, and linking are performed with the Build option. **Figure 4** shows several windows within CSS for the project *Multirate.*

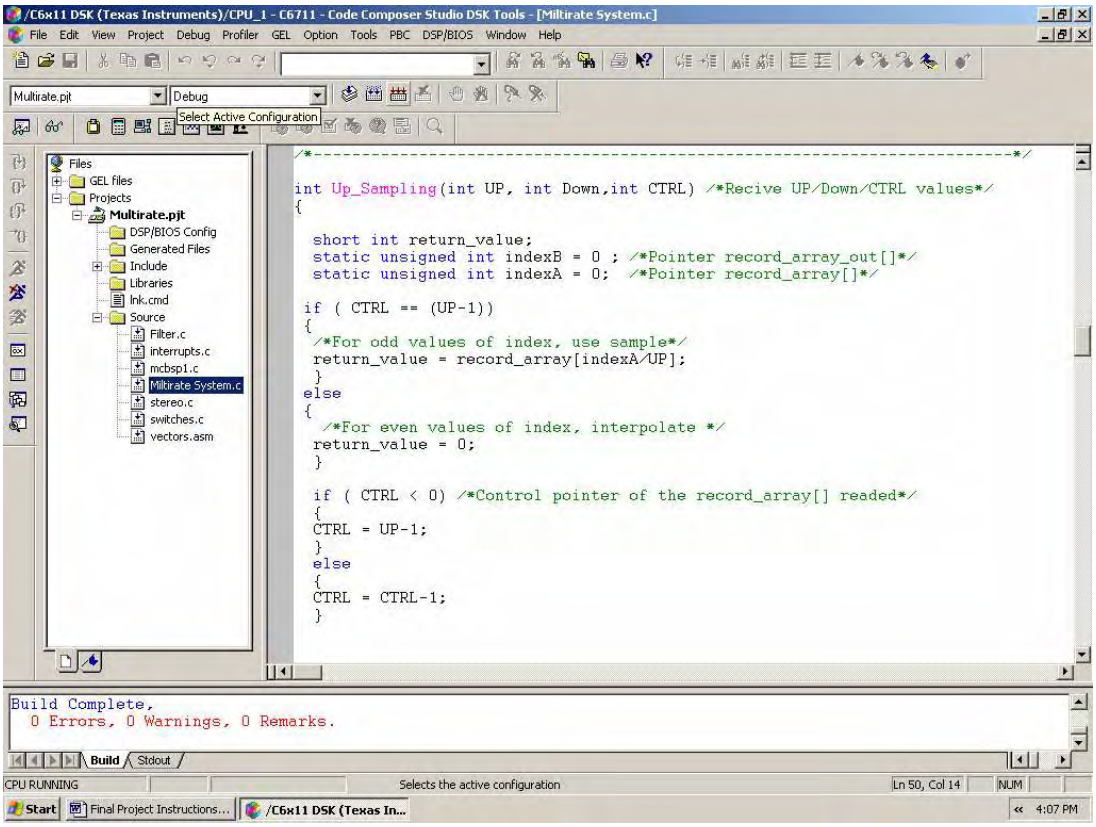


Figure 4 Windows for project *Multirate.pjt*

2. Select Debug => Reset CPU in order to clear and initialize all the registers on the DSK. Then Select File => Load Program in order to load *Multirate.out* by clicking on it. It should be in the project *Multirate* folder. Connect an input signal from the signal generator or audio source to the IN connector (j1) on the **Daughter Board** of the DSK, also connect a speaker and the oscilloscope to the OUT connector (j2) on the **Daughter Board** of the DSK. This prepares the Multirate implementation on the DSK for use.

NOTE: Before the Run command on the CCS, the user should put all the user switches on the DSK to zero, or to the down position. Depending on the multi-rate desired, the user may select switches (USER_SW1) = '1', (USER_SW2) = '0' and (USER_SW3) = '0' for 2/3 rate, Fs(in) = 36 KHz and Fs(out) = 24 KHz. Or select switches (USER_SW1) = '0', (USER_SW2) = '1' and (USER_SW3) = '0' for 3/2 rate, Fs(in) = 24 KHz and Fs(out) = 36 KHz . If the program is run with the switch = 0 the rate will be selected as default as 2/3 and Fs = 24 KHz.

3. Select Debug => Run. Or use the toolbar with the "running man". Now track the following routine for the switches in the DSK to run the Multi-rate system. It should be noted that the User Selectable Switches represent a 3 Bits binary number where the LSB, (Lowest Significant Bit), is the switch (USER_SW1), and the MSB, (Maximum Significant Bit), is the switch (USER_SW3).

Follow the next steps and compare them with the **Figure 5** .

a. Select switches (USER_SW1) = ‘0’, (USER_SW2) = ‘0’ and (USER_SW3) = ‘0’ for “Straight through, no recording” state. This switch value correspond to the “0” value in decimal base. This routine loops back the input signal to the output of the DSK Daughter board.

b. Select switches (USER_SW1) = ‘1’, (USER_SW2) = ‘0’ and (USER_SW3) = ‘0’ for “Manual recording in progress” state. This switch value correspond to the “1” value in decimal base. This routine takes the digital out of the ADC and stores it on the RAM memory, called “record_array”, on the DSK. **Wait** with the switch in that state until the message “Buffer Full” appears in the **Stdout** window on the CCS.

c. Select switches (USER_SW1) = ‘0’, (USER_SW2) = ‘0’ and (USER_SW3) = ‘0’ to return to the “Straight through, no recording” state.

d. Select switches (USER_SW1) = ‘1’, (USER_SW2) = ‘1’ and (USER_SW3) = ‘0’ for “Normal playback” state. This switch value correspond to the “3” value in decimal base. This routine plays back the recorded sound from the memory, “record_array”, of the DSK at the same Sampling Frequency as the input sampling rate.

e. Select switches (USER_SW1) = ‘0’, (USER_SW2) = ‘0’ and (USER_SW3) = ‘0’ to return to the “Straight through, no recording” state.

f. Select switches (USER_SW1) = ‘0’, (USER_SW2) = ‘1’ and (USER_SW3) = ‘0’ for “Up-Sampling Process” state. This switch value correspond to the “2”

value in the decimal base. This routine performs the Up-sampling process and filter the stored samples on the “record_array” and stores the results on the memory called “record_array_out”.

g. Select switches (USER_SW1) = ‘0’, (USER_SW2) = ‘0’ and (USER_SW3) = ‘0’ to return to the “Straight through, no recording” state.

h. Select switches (USER_SW1) = ‘0’, (USER_SW2) = ‘0’ and (USER_SW3) = ‘1’ for “Playback Up_Sampling” state. This switch value correspond to the “4” value in the decimal base. This routine plays back the recorded sound from the memory “record_array_out”of the DSK at the same Sampling Frequency as the input sampling rate.

i. Select switches (USER_SW1) = ‘1’, (USER_SW2) = ‘0’ and (USER_SW3) = ‘1’ for “Playback Down_Sampling” state. This switch value correspond to the “5” value in the decimal base. This routine performs the Down-sampling process and plays the result at the new sampling frequency corresponding to the desired Up/Down sampling rate chosen at the beginning.

j. Select switches (USER_SW1) = ‘1’, (USER_SW2) = ‘1’ and (USER_SW3) = ‘1’ for “record_arrays_clear” state. This switch value correspond to the “7” value in the decimal base. This routine performs a clean-up of the memory used in the process, “record_array” and “record_array_out”.

k. Select switches (USER_SW1) = ‘0’, (USER_SW2) = ‘0’ and (USER_SW3) = ‘0’ to return to the “Straight through, no recording” state. Now the DSK is ready for another recording and Multi-rate process with the same selected rate.

NOTE: If the user wants to use the other rate, the user must stop the program and re-load the *Multirate.out* file again. Follow the **step 2** used earlier. Here the user may select the rate again.

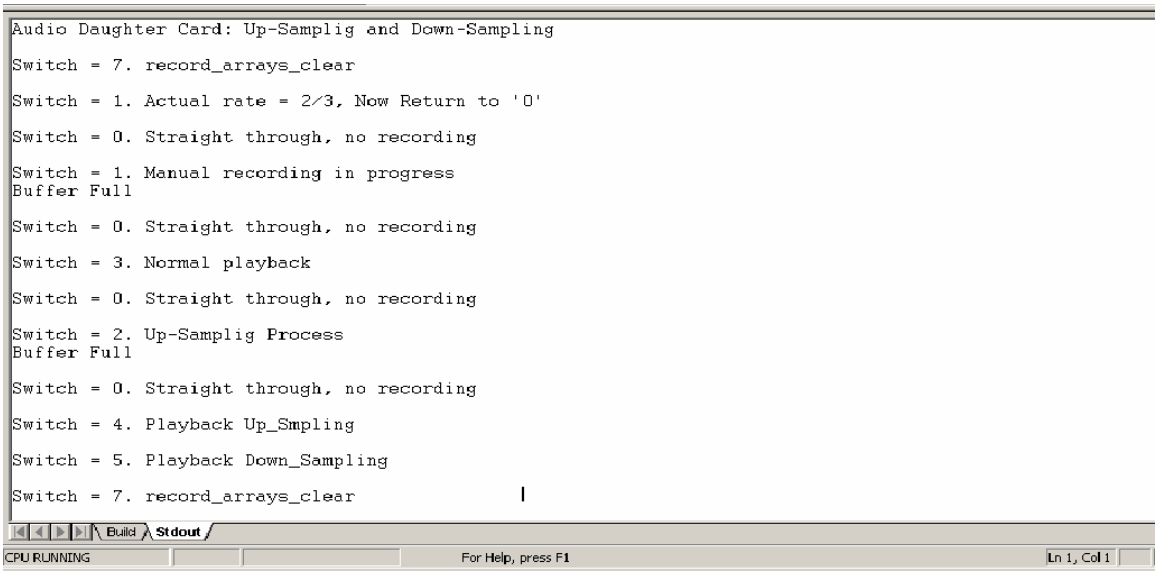
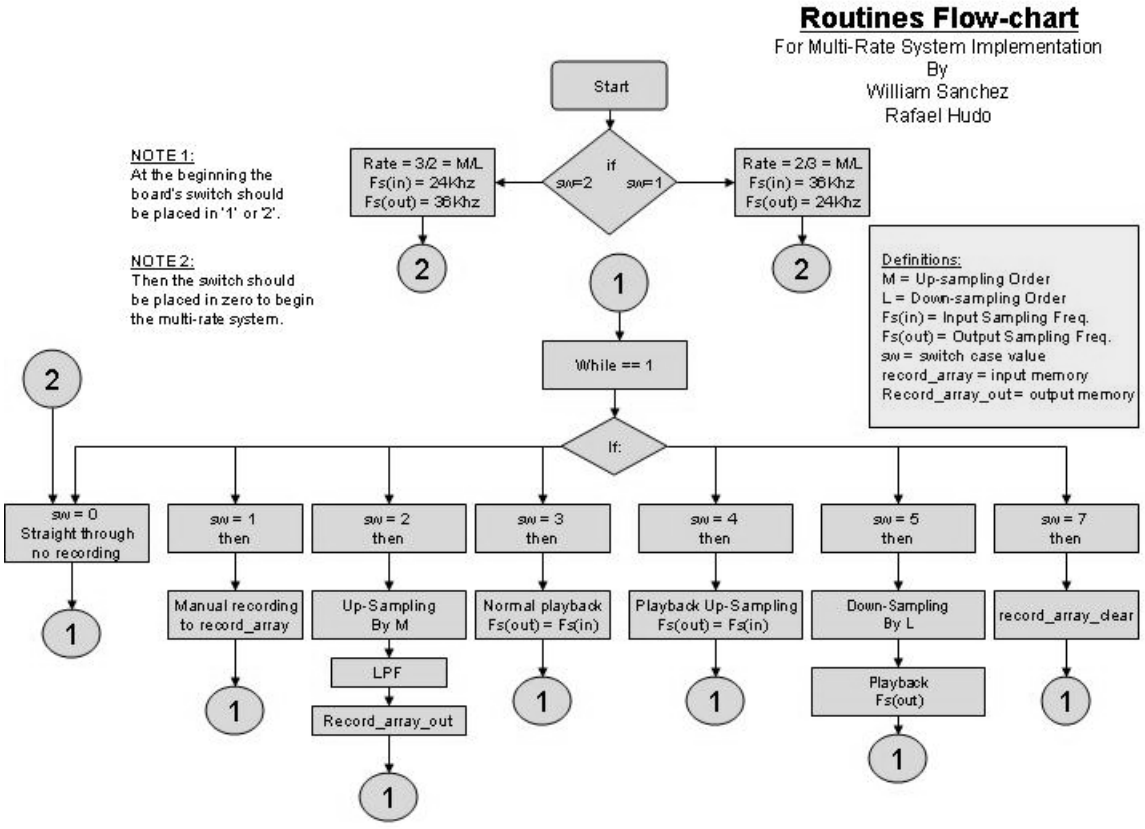


Figure 5 Standard Out window of the *Multirate* program.

Routines Flow Chart



Project Results

The Following figures shows, the final results of the Multirate Project.

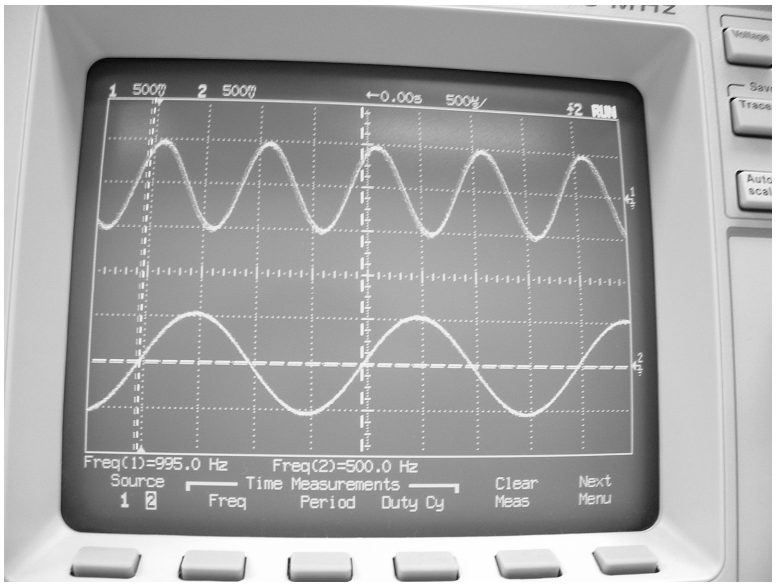
Figure A

Channel 1: Original Input Signal (Vpp = 500mv and 1 KHz).

Channel 2: Up-Sampled Signal by 2

Fs(in) = 36 KHz.

Fs(out) = 36 KHz.



The picture shows that the signal in channel 2, up-sampled by 2, has half the frequency of the original input signal, presented in channel 1, because the Up_Sampling Process increases the size of the original input signal by 2. To show that the Up_Sampling process works, we selected Fs (out) with equal value as Fs (in) so that the output signal,

with two times the samples as the original signal, comes out with half the frequency of the original signal. It is important to know that the output signal has been filtered by LPF with cut frequency of pi/3 to interpolate the zero value samples added with the original samples and to eliminate the spectral images generated by the Up-sampler.

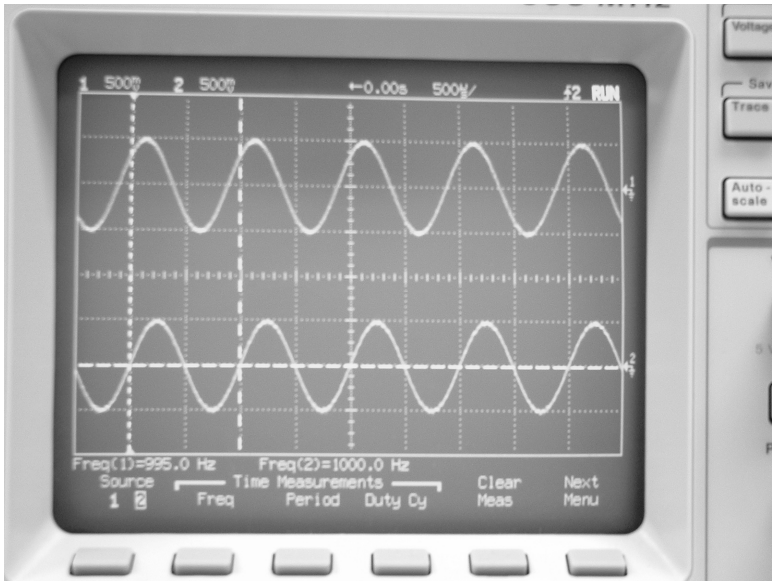
Figure B

Channel 1: Original Input Signal (Vpp = 500mv and 1 KHz).

Channel 2: Signal after Up/Down sampling processes, rate = 2/3

Fs(in) = 36 KHz.

Fs(out) = 24 KHz.



The picture shows that the signal in channel 2, after the Up/Down sampling processes, has the same frequency that of the original recorded signal. The reason for this is that the input signal's sampling frequency, Fs(in) = 36 KHz, is multiplied by the Up/Down sampling rate = 2/3, resulting in a output sampling frequency, Fs(out) = 24 KHz. To

show that the Up/Down sampling processes works, we selected Fs(out) = 24 KHz in order to reproduce correctly the input signal. The signal has been filtered to eliminate the spectral images and possible aliasing created by the Up-sampler and Down-sampler respectively.

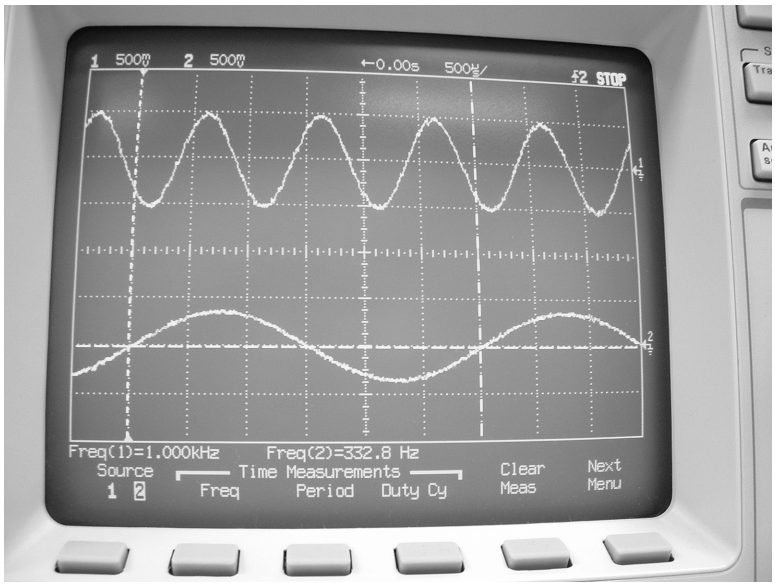
Figure C

Channel 1: Original Input Signal (Vpp = 500mv and 1 KHz).

Channel 2: Up-Sampled Signal by 3

Fs(in) = 24 KHz.

Fs(out) = 24 KHz.



The picture shows that the signal in channel 2, up-sampled by 3, has 1/3 the frequency of the Original Input Signal, presented in channel 1, because the Up_Sampling Process increases the size of the Original Input Signal by 3. To show that the Up_Sampling process works, we selected Fs (out) with equal value as Fs (in) so that the output signal, with three times the samples as the original signal, comes out with 1/3 the frequency of

the original signal. It is important to know that the output signal has been filtered by LPF with cut frequency of pi/3 to interpolate the zero value samples added with the original samples and to eliminate the spectral images generated by the Up-sampler.

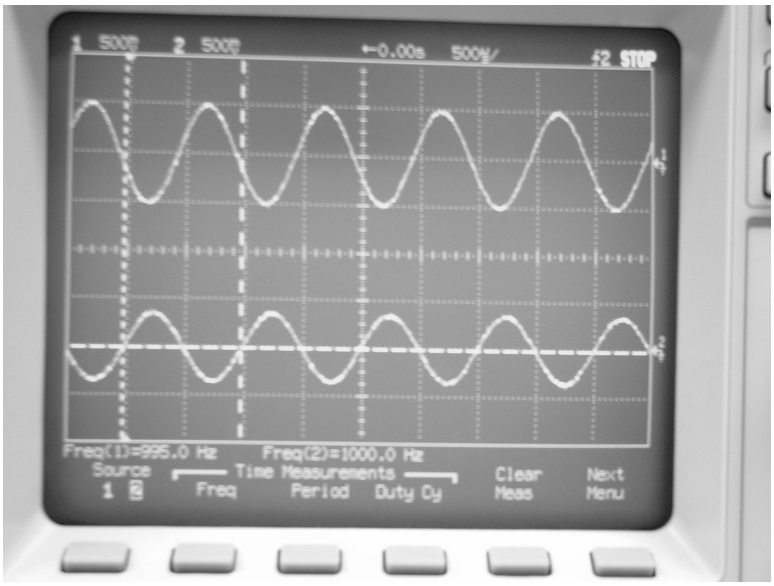
Figure D

Channel 1: Original Input Signal (Vpp = 500mv and 1 KHz).

Channel 2: Signal after Up/Down sampling processes, rate = 3/2

Fs(in) = 24 KHz.

Fs(out) = 36 KHz.

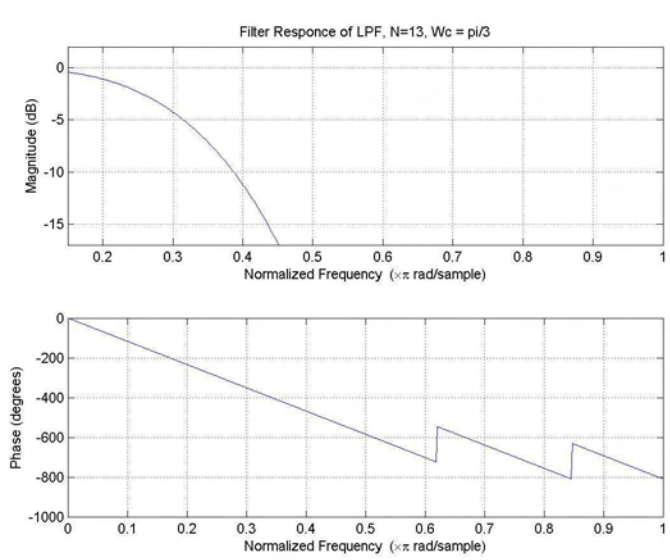


The picture shows that the signal in channel 2, after the Up/Down sampling processes, has the same frequency that of the original recorded signal. The reason for this is that the input signal's sampling frequency, Fs(in) = 24 KHz, is multiplied by the Up/Down sampling rate = 3/2, resulting in a output sampling frequency, Fs(out) = 36 KHz. To show that the Up/Down sampling processes works, we selected Fs(out) = 36 KHz in

order to reproduce correctly the input signal. The signal has been filtered to eliminate the spectral images and possible aliasing created by the Up-sampler and Down-sampler respectively.

Filter Design with MatLab

We used MatLab as a tool for the filter design in this project for interpolation and spectral images rejection and as anti-aliasing. We design a finite impulse response, FIR, filter with order of 13 and a cut frequency of л/3. We used the FIR1 command to generate the coefficients used by the implementation that are located in the file *COEF_CLPF.h.*



Conclusion:

The Multi-rate system implementation on the TI's TMS320C6711 DSP board was completed successfully. We were able to implement a system capable of adapting an output signal from an initial system working at a sampling frequency of 24 KHz, to a system working at a sampling frequency of 36 KHz. Also the implementation is capable of adapting the two systems, the sender system and the receiving system, with samples frequencies of 36 KHz and 24 KHz respectively. This is proof that we could implement successfully the Up-sampling and Down-sampling concepts, also the interpolation by the use of FIR filters.

We learned about the different features of the DSK board like the User Switches, the DSK Ram memory, the register for control like TIMER0 and TIMER1 to change the sampling rate, and the Audio Daughter Card for high frequency sampling and flexibility with the sampling rates frequencies by controlling it by software.

Most importantly we gained knowledge about the C++ programming for DSP applications using the CCS, Code Composer Studio, software tool, also the experience in designing a complete DSP application, which will be helpful on our development as engineers.

B. DBT Tool box Example

```
%function dbtex1
%DBTEX1 An example of a main program using  conventional beamforming on
%simulated signals from an ULA.
% *  DBT, A Matlab Toolbox for Radar Signal Processing  *
% (c) FOA 1994-2000. See the file dbtright.m for copyright notice.
%  Start        : 951221 Svante Björklund (svabj).
%  Latest change: $Date: 2000/10/16 15:39:42 $ $Author: svabj $.
%  $Revision: 1.2 $
% ****************************************************************************

% ------------------------------------------------------------------------ %
% Parameters.
% ------------------------------------------------------------------------ %

lambda  = 0.03;         % wavelength.
D    = 0.45*lambda;     % Element separation.
T    = 24;           % Number of snapshots.
K    = 32;           % Number of digital antenna channels.

theta   = d2r([25 32])';    % Target angles. The number of targets is
                %  given by the number of target angles.
phi = zeros(size(theta));   % Target angles.
SNR = [10 5]';      % Signal to noise ratio in dB at each
                %  antenna element!?!
alpha   = d2r([0 16])';     % Start phases of the target signals.
dalpha  = d2r([34 -13])';   % A constant phase shift between snapshots.
                %  Means targets movements at constant velocity.
dist=Inf*ones(size(theta)); % Distances to the sources.
tgtModel        = 'const';  % Target type to simulate.
noiseModel      = 'rndnw';  % Noise type to simulate.
%noiseModel         = 'rndn';   % Noise type to simulate.
MMu = 2;            % Number of targets that MUSIC believes in.

% ------------------------------------------------------------------------ %
% Commands.
% ------------------------------------------------------------------------ %
ant = defant('isotropULA',[K,D]);       % Define the antenna.

sig = compsim4(ant, lambda, T, tgtModel, [theta, ...
  phi, SNR, alpha, dalpha, dist, ...        % Generate simulated received
  eye(size(theta,1))], noiseModel, eye(K));     % antenna signals.


%spectl = sdoaspc('music',sig,[],MMu);      % Estimate the DOA-spectrum
                        %  with MUSIC.

R = ecorrm(sig);                % Estimate the antenna signals
                                %  correlation matrix.

spect2 = sdoaspc('cbf',R);          % Estimate the DOA-spectrum
                                    %  with conventional beamform.
figure,splot2(spect2);
```

C. Time Frequency Algorithms

```
/*********************************************************************
*
* AIP Laboratory
*
* AF.C
*
* DESCRIPTION
*  This program compute the Ambiguity Function between the Tx and Rx
*  signals.
*
*  DEVICE: DSK320C6711 T.I.
*  CCS v. 2.1
*--------------------------------------------------------------------
* HISTORY
*     Rev 1.00 - Sep/2004  Created by MS. William D. Sánchez R.
*                                     Dr. Domingo Rodríguez - Advisor
*
*********************************************************************/
// Included Files//
#include "math.h"       // math.h header for mathematical operations.
#include "dataTW256.h"  // Twiddle Factors to compute a N-point FFT.
#include "Sgtx256.h"    // Signal transmmitted.
#include "Sgrx256.h"    // Signal Received.
#include "Xcorr256.h"   // Correlation File initialized with "0.0".
#include "c6211dsk.h"   // c6211dsk. header for dsk function routines
//////////////////////////////////////////////////////////////////////

#define N 256           // Points of the Signals and FFT points



//Storage Matrix//
far double Ambiguity_Mtx[N][2*N]; // far - location, of the Ambiguity
Function [Rows][ComplexColumns].
//////////////////////////////////////////////////////////////////////

/*--------------------------------------------------------------------
* NAME: Coplex_Complement(void)
* DESCRIPTION: Perform the complement to R = (R + j0) Real data stored

*                    in Ambiguity_Mtx[m][n],
* ARGUMENTS:   Uses the general Variable Ambiguity_Mtx
*
*********************************************************************/

void Complex_Complement(void)
{

int n,m;

for (m=0; m<N; m+=1)
      {
          for (n=(N-1); n>=0;n-=1)
```

```
                    {
                     Ambiguity_Mtx[m][2*n] = Ambiguity_Mtx[m][n];
                     Ambiguity_Mtx[m][2*n+1] = 0;
                     }

      }
}
////////////////////////////////////////////////////////////////////////

/*---------------------------------------------------------------------
* NAME: FFT_TI (void)
* DESCRIPTION: Perform the FFT of each row of the Ambiguity_Mtx[m]
*
* ARGUMENTS:   Uses the general Variable Ambiguity_Mtx
*
*********************************************************************/

void FFT_TI (void)
{
int m;
double *pointer1;

//bit_rev((float *)w, N>>1);

for (m=0; m<N; m+=1)
      {

       pointer1 = Ambiguity_Mtx[m];
       DSPF_sp_cfftr2_dit((double*) pointer1,(float*) w, N);
       bit_rev((double *)pointer1, (2*N)>>1);

      }

}

////////////////////////////////////////////////////////////////////////

/*---------------------------------------------------------------------
* NAME: Haddamart (double Stx[])
* DESCRIPTION: Perform the Haddamart product between Tx and the Matrix
family of
*                     shifthed Rx signal
* ARGUMENTS:   The Tx signal vector.
*
*********************************************************************/

void Haddamart (double Stx[])
{
int n,m;

for (m=0; m<N; m+=1)
      {
            for (n=0; n<N;n+=1)
                  {
```

```
                    Ambiguity_Mtx[m][n]=
(Ambiguity_Mtx[m][n]*Stx[n]);//Fam_Shift[m][n]=
(Fam_Shift[m][n]*Stx[n]);
                    }
      }

}

////////////////////////////////////////////////////////////////////////

/*---------------------------------------------------------------------
* NAME: Shift_Signal (double Srx[])
* DESCRIPTION: Create a Matrix of Cyclic Shifth Rx signal
* ARGUMENTS:   The Rx signal vector
*
**********************************************************************/

void Shift_Signal (double Srx[])
{
int n,m,i;

for (m=0; m<N; m+=1)
      {
            for (n=0; n<N;n+=1)
                  {
                   i=(n+m)%N;
                   Ambiguity_Mtx[m][n]= (Srx[i]);//Fam_Shift[m][n]=
(Srx[i]);
                   }
      }

}
////////////////////////////////////////////////////////////////////////

/*---------------------------------------------------------------------
* NAME: Corr(double Stx[], double Srx[])
* DESCRIPTION:Perform the Cross-correlation between the Tx and Rx
Signal.
* ARGUMENTS:  The Tx signal vector and    the Rx signal vector.
*
**********************************************************************/

void Corr(double Stx[], double Srx[])
{
int n,m,i;

for (m=0; m<N; m+=1)
      {
            for (n=0; n<N;n+=1)
                  {
                   i=(n+m)%N;
                   Xcorr[m]= (Stx[n]*Srx[i])+Xcorr[m];
                   }
      }
```

```
}

//////////////////////////////////////////////////////////////////////

/* Main program */
void main()
{

Corr(Sigtx, Sigrx );  //Cross-correlation between the Tx and Rx Signal.

Shift_Signal (Sigrx); //Cyclic Shifth Rx signal

Haddamart (Sigtx);    // Haddamart Product

Complex_Complement(); // R = (R+j0)

FFT_TI();                // FFT  Matrix's rows  Ambiguity_Mtx.


}
//////////////////////////////////////////////////////////////////////




/*********************************************************************
*
* AIP, Laboratory.
*
* STFT.C
*
* DESCRIPTION
*  This program compute the Short Time Fourier Transform
*  (STFT)of a Shirp Signal, Using the method of indirect
*  filtering (FFT) .
*
* DEVICE: DSK320C6711 T.I.
* CCS v. 2.1
*--------------------------------------------------------------------
* HISTORY
*     Rev 1.00 - Sep/2004  Created by MS. William D. Sánchez R.
*                                                 Dr. Domingo Rodríguez -
Advisor
*
*********************************************************************/

/* Define Variables */
#define N  256                    // number of filters banks
#define P  1024                   // Points of the signal Padded
#define L  128                    // Points of the filter without
padding
#define M  P-L                    // Point of the input Chirp Signal
```

```
#define fs 1000             // frequency sampling
#define D  1.024            // time durantion in seconds
////////////////////////////////////////////////////////////////////

//Storage Matrix//
far double xk[N][2*P];// far - location, of the STFT
//[Rows][ComplexColumns].
////////////////////////////////////////////////////////////////////

// Included Files//
#include "math.h"             // math.h header for mathematical
operations.
#include "c6211dsk.h"         // c6211dsk. header for dsk function
routines
#include "dataTW1024.h"       // Twiddle Factors to compute a P-point
FFT.
#include "chirpSTFT_896.h"  // Chirp input Signal with  M-points.
#include "fft_h_padd_128.h" // FFT ( of the L-points filter with pading
//                                          // to P-points).
////////////////////////////////////////////////////////////////////
```

```
/*--------------------------------------------------------------------
* NAME: Haddamart (double *fft_h)
* DESCRIPTION: Perform the Haddamart product between aech row of the
*                    modulated Array and fft_h. FFT(xk[m][n]) .* FFT_h
*
* ARGUMENTS:   The fft_h coeficientes = FFT (h_padded).
*
*********************************************************************/
void Haddamart (double *fft_h)
{
int n,m;
double y1,y2,y3,y4;

for (m=0; m<N; m+=1)
      {
            for (n=0; n<P;n+=1)
                  {

                   y1 = xk[m][2*n]*fft_h[2*n];
                   y2 = -1*(xk[m][2*n+1]*fft_h[2*n+1]);
                   y3 = xk[m][2*n]*fft_h[2*n+1];
                   y4 = xk[m][2*n+1]*fft_h[2*n];

                   xk[m][2*n]= y1+y2;
                   xk[m][2*n+1]= y3+y4;

                   }
```

```
      }

}
////////////////////////////////////////////////////////////////////////
/*--------------------------------------------------------------------
* NAME: FFT_TI (void)
* DESCRIPTION: Perform the FFT of each row on the modulated Array
*                    FFT(xk[row][n].
*
* ARGUMENTS:   Uses the general variable matrix xk[m][n].
*
**********************************************************************/

void FFT_TI (void)
{
int m;
double *pointer1;

//bit_rev((float *)w, N>>1);

for (m=0; m<N; m+=1)
      {

       pointer1 = xk[m];       // pointer to the rows of the xk matrix.
       DSPF_sp_cfftr2_dit((double*) pointer1,(float*) w, P);
       bit_rev((double *)pointer1, (2*P)>>1);

      }

}
////////////////////////////////////////////////////////////////////////
/*--------------------------------------------------------------------
* NAME: In_FFT_TI (void)
* DESCRIPTION: Inverse FFT, using the complex complement of the
*              Twidel factor. IFFT(FFT(xk[m][n]) .* FFT_h)/P.
*
* ARGUMENTS:   Uses the general variable matrix xk[m][n].
*
**********************************************************************/
void In_FFT_TI (void)
{
int m,n;
double *pointer1;

//bit_rev((float *)w, N>>1);

for (m=0; m<N; m+=1)
      {

       pointer1 = xk[m];            // pointer to the rows of the xk
matrix.
       In_DSPF_sp_cfftr2_dit((double*) pointer1,(float*) w, P);
       bit_rev((double *)pointer1, (2*P)>>1);

      }
```

```
for (m=0; m<N; m+=1)     // The output has to be divided by P-points.
      {
            for (n=0; n<2*P; n+=1)
            {
            xk[m][n]=(xk[m][n])/P;
            }
      }
}
////////////////////////////////////////////////////////////////////////
/*--------------------------------------------------------------------
* NAME: Matrix_modulation(double *x)
* DESCRIPTION: Modulation of the input Chirp Signal with e^(wk)
*
* ARGUMENTS:   Uses the input Chirp Signal stored on x[n].
*
*********************************************************************/
void Matrix_modulation(double *x)
{
int k,n,t;
double wk;
double pi = 4.0*atan(1.0);

t = fs*D;

for (k=0; k<N; k+=1)
      {
            wk = (2*pi*k)/N;

            for (n=0; n<t; n+=1)
                  {
                        xk[k][2*n] = cos(wk*n)*x[2*n];
                        xk[k][2*n+1] = (sin(wk*n)*x[2*n])*-1;
//Modulation Matrix

                  }
      }
}



/* Main program */
void main()
{

Matrix_modulation((double*) X); //Modulation of input Chirp Signal.

FFT_TI();                     // FFT of rows of the Modulation Matrix.

Haddamart ((double*) h);        // Haddamart product between each  row
//                              // of Modulation Matrix and fft_h.

In_FFT_TI();                    // inverse fft, we don't have magnitude

}
```

D. **Multirate** Beamforming Algorithm.

```
/************************************************************************
*
* AIP, Laboratory.
*
* BeamForming.C
*
* DESCRIPTION
*  This program perform a "P" beam-forming from "N" sensor array.
*  with the S Down-Sampling factor.
*
*
* DEVICE: DSK320C6711 T.I.
* CCS v. 2.1
*-----------------------------------------------------------------------
* HISTORY
*     Rev 1.00 - Sep/2004  Created by MS. William D. Sánchez R.
*                                                 Dr. Domingo Rodríguez -
Advisor
*
************************************************************************/

/* Define Variables */

#include "math.h"
#include "c6211dsk.h"


/* Define sample rate */
float Fs=40000.0;

#define N  32       // number of Sensors Array.
#define L_d 2       // lambda/d (distance).
#define p   2       // pth output of DFT.
#define B0   p*L_d/N // Covered beams, p = 0,1,2,3,....N-1.
#define ch p
#define r p
#define B 256       // Number of vectors sampling over all sensors.
#define P 8           //points of fft to evaluate the number of beams.
#define S 1         // Down-Sampling factor.
double FFT_Vec[2*P];// Vector for store rows and perform FFT.
////////////////////////////////////////////////////////////////////////
// Included Files//
#include "dataTW8.h"      // Twiddle Factor to Compute the P-FFT
#include "Beam_256_32.h"  // Input Signal Simulated from Matlab
(Matrix_complex (B x N).
#include "FFT_Sum_256_8.h" // Output Matrix_complex (B x P). each
colomn represent a entire beam.
////////////////////////////////////////////////////////////////////////
```

```
/*---------------------------------------------------------------------
* NAME: Sum_FFT_Coherent(double* x, int fila)
* DESCRIPTION: Perform the Coherent Sum for each row
*
* ARGUMENTS:   Pointer to the row in after fft *x and, the column in
process.
*
**********************************************************************/

void Sum_FFT_Coherent(double* x, int fila)
{
int m,n;

m= fila;

      for (n=0; n<2*P; n+=1)
      {
      FFT_Sum[m][n]=    x[n]+FFT_Sum[m][n];
      }

}
//////////////////////////////////////////////////////////////////////
/*---------------------------------------------------------------------
* NAME: FFT_TI_Beam (void)
* DESCRIPTION: For each N-points rows divide by P-points and perform
the P-points fft.
*                    with the S Down-Sampling factor. to reduce
computations.
* ARGUMENTS:   Uses the general variable matrix Beam_M and FFT_Vec.

*
**********************************************************************/

void FFT_TI_Beam (void)
{
int m,n,i,k,j;
double *pointer1;


i=0;
k=1;
j=0;

//bit_rev((float *)w, N>>1);

for (m=0; m<B; m+=S)
      {
       for (n=i; n<N; n+=1)
       {
       FFT_Vec[2*j]= Beam_M[m][2*n];
       FFT_Vec[2*j+1]= Beam_M[m][2*n+1];
       j++;
```

```
        if (n == k*P-1)
             {
                   j=0;
                   k++;
                   i+=k*P;
                   pointer1 = FFT_Vec;
                   DSPF_sp_cfftr2_dit((double*) pointer1,(float*) w, P);
                   bit_rev((double*)pointer1, (2*P)>>1);
                   Sum_FFT_Coherent((double*)pointer1, m);
             };

        }
       i=0;
       k=1;
       }

}
////////////////////////////////////////////////////////////////////////


/* Main program */
void main()
{

FFT_TI_Beam();// Routine to compute the "B" sampling vectors to compute
              // "P" Beamforming. with the "S" Down-Sampling factor.

}
```

E. **Kronecker Properties Examples.**

Let

$$A=\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \quad , \quad B=\begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \quad , \quad C=\begin{bmatrix} c_{0,0} & c_{0,1} \\ c_{1,0} & c_{1,1} \end{bmatrix} \qquad D=\begin{bmatrix} d_{0,0} & d_{0,1} \\ d_{1,0} & d_{1,1} \end{bmatrix}$$

- **Scalar Multiplication:** If α is a scalar, then

$$A \otimes (\alpha B) = \alpha (A \otimes B)$$

$$A \otimes \begin{bmatrix} \alpha b_{0,0} & \alpha b_{0,1} \\ \alpha b_{1,0} & \alpha b_{1,1} \end{bmatrix} = \alpha \begin{bmatrix} b_{0,0}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} & b_{0,1}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \\ b_{1,0}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} & b_{1,1}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \alpha b_{0,0}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} & \alpha b_{0,1}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \\ \alpha b_{1,0}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} & \alpha b_{1,1}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \end{bmatrix} = \alpha \begin{bmatrix} b_{0,0}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} & b_{0,1}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \\ b_{1,0}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} & b_{1,1}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \end{bmatrix}$$

- **Distributive Law:** The Kronecker product is distributive with respect to addition

$$(A+B) \otimes C = A \otimes C + B \otimes C$$

$$\begin{bmatrix} a_{0,0}+b_{0,0} & a_{0,1}+b_{0,1} \\ a_{1,0}+b_{1,0} & a_{1,1}+b_{1,1} \end{bmatrix} \otimes C = \begin{bmatrix} c_{0,0}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} & c_{0,1}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \\ c_{1,0}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} & c_{1,1}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \end{bmatrix} + \begin{bmatrix} c_{0,0}\begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} & c_{0,1}\begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \\ c_{1,0}\begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} & c_{1,1}\begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \end{bmatrix}$$

.

$$\begin{bmatrix} c_{0,0}\begin{bmatrix} a_{0,0}+b_{0,0} & a_{0,1}+b_{0,1} \\ a_{1,0}+b_{1,0} & a_{1,1}+b_{1,1} \end{bmatrix} & c_{0,1}\begin{bmatrix} a_{0,0}+b_{0,0} & a_{0,1}+b_{0,1} \\ a_{1,0}+b_{1,0} & a_{1,1}+b_{1,1} \end{bmatrix} \\ c_{1,0}\begin{bmatrix} a_{0,0}+b_{0,0} & a_{0,1}+b_{0,1} \\ a_{1,0}+b_{1,0} & a_{1,1}+b_{1,1} \end{bmatrix} & c_{1,1}\begin{bmatrix} a_{0,0}+b_{0,0} & a_{0,1}+b_{0,1} \\ a_{1,0}+b_{1,0} & a_{1,1}+b_{1,1} \end{bmatrix} \end{bmatrix} =$$

$$\begin{bmatrix} c_{0,0}\left(\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix}+\begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix}\right) & c_{0,1}\left(\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix}+\begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix}\right) \\ c_{1,0}\left(\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix}+\begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix}\right) & c_{1,1}\left(\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix}+\begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix}\right) \end{bmatrix}$$

- **Associative Law**: The Kronecker product is associative

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C .$$

$$\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \otimes \begin{bmatrix} c_{0,0}\begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} & c_{0,1}\begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \\ c_{1,0}\begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} & c_{1,1}\begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} b_{0,0}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} & b_{0,1}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \\ b_{1,0}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} & b_{1,1}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \end{bmatrix} \otimes C$$

$$\begin{bmatrix}\begin{bmatrix}b_{0,0}c_{0,0}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix} & b_{0,1}c_{0,0}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix}\\ b_{1,0}c_{0,0}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix} & b_{1,1}c_{0,0}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix}\end{bmatrix} & \begin{bmatrix}b_{0,0}c_{0,1}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix} & b_{0,1}c_{0,1}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix}\\ b_{1,0}c_{0,1}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix} & b_{1,1}c_{0,1}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix}\end{bmatrix}\\ \begin{bmatrix}b_{0,0}c_{1,0}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix} & b_{0,1}c_{1,0}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix}\\ b_{1,0}c_{1,0}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix} & b_{1,1}c_{1,0}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix}\end{bmatrix} & \begin{bmatrix}b_{0,0}c_{1,1}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix} & b_{0,1}c_{1,1}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix}\\ b_{1,0}c_{1,1}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix} & b_{1,1}c_{1,1}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix}\end{bmatrix}\end{bmatrix} =$$

$$\begin{bmatrix}c_{0,0}\begin{bmatrix}b_{0,0}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix} & b_{0,1}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix}\\ b_{1,0}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix} & b_{1,1}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix}\end{bmatrix} & c_{0,1}\begin{bmatrix}b_{0,0}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix} & b_{0,1}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix}\\ b_{1,0}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix} & b_{1,1}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix}\end{bmatrix}\\ c_{1,0}\begin{bmatrix}b_{0,0}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix} & b_{0,1}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix}\\ b_{1,0}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix} & b_{1,1}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix}\end{bmatrix} & c_{1,1}\begin{bmatrix}b_{0,0}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix} & b_{0,1}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix}\\ b_{1,0}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix} & b_{1,1}\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

- **Identity Product**: Given I_{rc}, the $r \times c$ identity matrix,

$$I_{rc} = I_r \otimes I_c$$

$$I_4 = I_2 \otimes I_2$$

$$\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \otimes \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$$

$$\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}1\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} & 0\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\\ 0\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} & 1\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\end{bmatrix}$$

- **Transpose**: The transpose for both matrix and tensor operations is useful for manipulating symmetric matrices (e.g. the Fourier matrix), where the original matrix and the transpose are equal.

$$(AB)^T = B^T A^T$$

$$(A \otimes B)^T = A^T \otimes B^T$$

$$\left(\begin{bmatrix} b_{0,0}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} & b_{0,1}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \\ b_{1,0}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} & b_{1,1}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \end{bmatrix}\right)^T = \begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix}^T \otimes \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix}^T$$

$$\left(\begin{bmatrix} \begin{bmatrix} a_{0,0}b_{0,0} & a_{0,1}b_{0,0} \\ a_{1,0}b_{0,0} & a_{1,1}b_{0,0} \end{bmatrix} & \begin{bmatrix} a_{0,0}b_{0,1} & a_{0,1}b_{0,1} \\ a_{1,0}b_{0,1} & a_{1,1}b_{0,1} \end{bmatrix} \\ \begin{bmatrix} a_{0,0}b_{1,0} & a_{0,1}b_{1,0} \\ a_{1,0}b_{1,0} & a_{1,1}b_{1,0} \end{bmatrix} & \begin{bmatrix} a_{0,0}b_{1,1} & a_{0,1}b_{1,1} \\ a_{1,0}b_{1,1} & a_{1,1}b_{1,1} \end{bmatrix} \end{bmatrix}\right)^T = \begin{bmatrix} a_{0,0} & a_{1,0} \\ a_{0,1} & a_{1,1} \end{bmatrix} \otimes \begin{bmatrix} b_{0,0} & b_{1,0} \\ b_{0,1} & b_{1,1} \end{bmatrix}$$

$$\left(\begin{bmatrix} a_{0,0}b_{0,0} & a_{0,1}b_{0,0} & a_{0,0}b_{0,1} & a_{0,1}b_{0,1} \\ a_{1,0}b_{0,0} & a_{1,1}b_{0,0} & a_{1,0}b_{0,1} & a_{1,1}b_{0,1} \\ a_{0,0}b_{1,0} & a_{0,1}b_{1,0} & a_{0,0}b_{1,1} & a_{0,1}b_{1,1} \\ a_{1,0}b_{1,0} & a_{1,1}b_{1,0} & a_{1,0}b_{1,1} & a_{1,1}b_{1,1} \end{bmatrix}\right)^T = \begin{bmatrix} b_{0,0}\begin{bmatrix} a_{0,0} & a_{1,0} \\ a_{0,1} & a_{1,1} \end{bmatrix} & b_{1,0}\begin{bmatrix} a_{0,0} & a_{1,0} \\ a_{0,1} & a_{1,1} \end{bmatrix} \\ b_{0,1}\begin{bmatrix} a_{0,0} & a_{1,0} \\ a_{0,1} & a_{1,1} \end{bmatrix} & b_{1,1}\begin{bmatrix} a_{0,0} & a_{1,0} \\ a_{0,1} & a_{1,1} \end{bmatrix} \end{bmatrix}$$

$$\left(\begin{bmatrix} a_{0,0}b_{0,0} & a_{0,1}b_{0,0} & a_{0,0}b_{0,1} & a_{0,1}b_{0,1} \\ a_{1,0}b_{0,0} & a_{1,1}b_{0,0} & a_{1,0}b_{0,1} & a_{1,1}b_{0,1} \\ a_{0,0}b_{1,0} & a_{0,1}b_{1,0} & a_{0,0}b_{1,1} & a_{0,1}b_{1,1} \\ a_{1,0}b_{1,0} & a_{1,1}b_{1,0} & a_{1,0}b_{1,1} & a_{1,1}b_{1,1} \end{bmatrix}\right)^T = \begin{bmatrix} \begin{bmatrix} a_{0,0}b_{0,0} & a_{1,0}b_{0,0} \\ a_{0,1}b_{0,0} & a_{1,1}b_{0,0} \end{bmatrix} & \begin{bmatrix} a_{0,0}b_{1,0} & a_{1,0}b_{1,0} \\ a_{0,1}b_{1,0} & a_{1,1}b_{1,0} \end{bmatrix} \\ \begin{bmatrix} a_{0,0}b_{0,1} & a_{1,0}b_{0,1} \\ a_{0,1}b_{0,1} & a_{1,1}b_{0,1} \end{bmatrix} & \begin{bmatrix} a_{0,0}b_{1,1} & a_{1,0}b_{1,1} \\ a_{0,1}b_{1,1} & a_{1,1}b_{1,1} \end{bmatrix} \end{bmatrix}$$

- **Mixed Product Rule**: Let A and C be $M \times M$ and B and D be $N \times N$ matrices. Thus,

$$(A \otimes B)(C \otimes D) = AC \otimes BD$$

$$A = \begin{bmatrix} a_{0,0} \end{bmatrix} \quad , \quad B = \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \quad , \quad C = \begin{bmatrix} c_{0,0} \end{bmatrix} \quad D = \begin{bmatrix} d_{0,0} & d_{0,1} \\ d_{1,0} & d_{1,1} \end{bmatrix}$$

$$\begin{bmatrix} b_{0,0}a_{0,0} & b_{0,1}a_{0,0} \\ b_{1,0}a_{0,0} & b_{1,1}a_{0,0} \end{bmatrix} \begin{bmatrix} d_{0,0}c_{0,0} & d_{0,1}c_{0,0} \\ d_{1,0}c_{0,0} & d_{1,1}c_{0,0} \end{bmatrix} = \begin{bmatrix} a_{0,0} \end{bmatrix} \begin{bmatrix} c_{0,0} \end{bmatrix} \otimes \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \begin{bmatrix} d_{0,0} & d_{0,1} \\ d_{1,0} & d_{1,1} \end{bmatrix}$$

$$\begin{bmatrix} (b_{0,0}a_{0,0})(d_{0,0}c_{0,0}) + (b_{0,1}a_{0,0})(d_{1,0}c_{0,0}) & (b_{0,0}a_{0,0})(d_{0,1}c_{0,0}) + (b_{0,1}a_{0,0})(d_{1,1}c_{0,0}) \\ (b_{1,0}a_{0,0})(d_{0,0}c_{0,0}) + (b_{1,1}a_{0,0})(d_{1,0}c_{0,0}) & (b_{1,0}a_{0,0})(d_{0,1}c_{0,0}) + (b_{1,1}a_{0,0})(d_{1,1}c_{0,0}) \end{bmatrix} =$$
$$\begin{bmatrix} a_{0,0} \end{bmatrix} \begin{bmatrix} c_{0,0} \end{bmatrix} \otimes \begin{bmatrix} (b_{0,0}d_{0,0}) + (b_{0,1}d_{1,0}) & (b_{0,0}d_{0,1}) + (b_{0,1}d_{1,1}) \\ (b_{1,0}d_{0,0}) + (b_{1,1}d_{1,0}) & (b_{1,0}d_{0,1}) + (b_{1,1}d_{1,1}) \end{bmatrix}$$

$$\begin{bmatrix} (b_{0,0}a_{0,0})(d_{0,0}c_{0,0}) + (b_{0,1}a_{0,0})(d_{1,0}c_{0,0}) & (b_{0,0}a_{0,0})(d_{0,1}c_{0,0}) + (b_{0,1}a_{0,0})(d_{1,1}c_{0,0}) \\ (b_{1,0}a_{0,0})(d_{0,0}c_{0,0}) + (b_{1,1}a_{0,0})(d_{1,0}c_{0,0}) & (b_{1,0}a_{0,0})(d_{0,1}c_{0,0}) + (b_{1,1}a_{0,0})(d_{1,1}c_{0,0}) \end{bmatrix} =$$
$$\begin{bmatrix} \left((b_{0,0}d_{0,0}) + (b_{0,1}d_{1,0})\right)\left(a_{0,0}c_{0,0}\right) & \left((b_{0,0}d_{0,1}) + (b_{0,1}d_{1,1})\right)\left(a_{0,0}c_{0,0}\right) \\ \left((b_{1,0}d_{0,0}) + (b_{1,1}d_{1,0})\right)\left(a_{0,0}c_{0,0}\right) & \left((b_{1,0}d_{0,1}) + (b_{1,1}d_{1,1})\right)\left(a_{0,0}c_{0,0}\right) \end{bmatrix}$$

One useful identity which follows (3.8), given $D(l \times s)$, then

$$A \otimes D = (AI_N) \otimes (I_l D) = (A \otimes I_l)(I_N \otimes D)$$

$$A \otimes D = (a_{0,0}[1]) \otimes \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d_{0,0} & d_{0,1} \\ d_{1,0} & d_{1,1} \end{bmatrix} \right) = \left(a_{0,0} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \left([1] \otimes \begin{bmatrix} d_{0,0} & d_{0,1} \\ d_{1,0} & d_{1,1} \end{bmatrix} \right)$$

$$(a_{0,0}[1]) \otimes \left(\begin{bmatrix} d_{0,0} & d_{0,1} \\ d_{1,0} & d_{1,1} \end{bmatrix} \right) = \left(\begin{bmatrix} a_{0,0} & 0 \\ 0 & a_{0,0} \end{bmatrix} \right) \left(\begin{bmatrix} d_{0,0} & d_{0,1} \\ d_{1,0} & d_{1,1} \end{bmatrix} \right)$$

$$\left(\begin{bmatrix} d_{0,0}a_{0,0} & d_{0,1}a_{0,0} \\ d_{1,0}a_{0,0} & d_{1,1}a_{0,0} \end{bmatrix} \right) = \left(\begin{bmatrix} a_{0,0} & 0 \\ 0 & a_{0,0} \end{bmatrix} \right) \left(\begin{bmatrix} d_{0,0} & d_{0,1} \\ d_{1,0} & d_{1,1} \end{bmatrix} \right)$$

F. Signal Processing Tools

This chapter concentrates on the main characteristics of hardware and software tools utilized around the implementation process of this thesis. We will start explaining the Digital Starter Kid DSK320C6711 of T.I. After that, we will review the most relevant characteristics of the DSP processor TMS320C6711 of T.I. In addition will see the Daughter Card PCM3003 of T.I., the Analog to Digital converter ADS8364 of T.I., the signal conditioner card AIP-0404-1 and the software of development Code Composer Studio based on C language CCS 2.1v.

1. Digital Starter Kit DSK320C6711

The TMS320C6711 DSP Starter Kit (DSK) Figure 6, developed jointly with Spectrum Digital, is a low-cost improvement platform designed to speed the development of high precision applications based on TI´s TMS320C6000 floating point DSP generation [17]. The kit uses a parallel port to connect to PC. The Code Composer Studio v2.1 was utilized.

The C6711 DSK tools include the latest fast simulators from TI and access to the Analysis Toolkit via Update Advisor which features the Cache Analysis tool and Multi-event Profiler.

The C6711 DSK allows downloading and stepping through code quickly and uses Real Time Data Exchange (RTDX™) for improved Host and Target communications.

The DSK includes the Fast Run Time Support libraries and utilities, such as Flashburn to program flash, Update Advisor to download tools, utilities and software and a power on self test and diagnostic utility to ensure the DSK is operating correctly.

The full contents of the kit include:

- C6711 DSP Development Board with 64K Flash and 16MB SDRAM
- C6711 DSK Code Composer Studio™ v2.1
- Quick Start Guide
- Technical Reference
- Customer Support Guide
- Parallel Cable
- Universal Power Supply
- AC Power Cord(s)

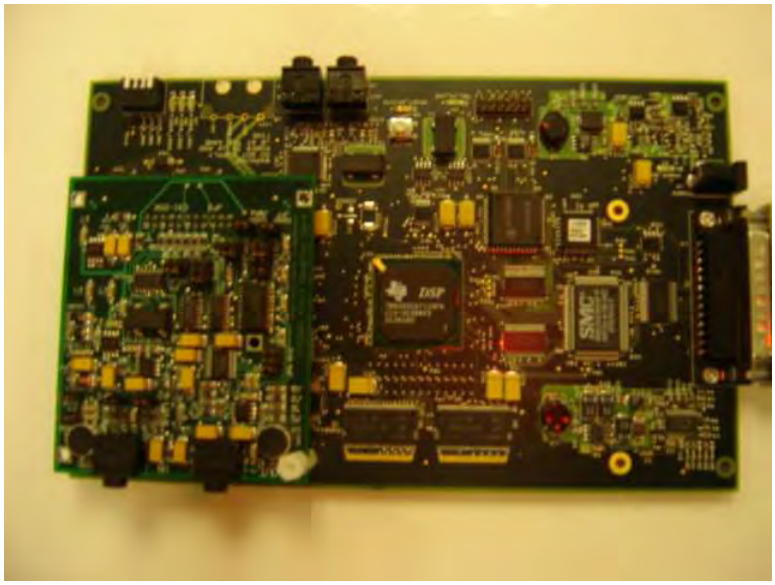


Figure 6 Digital Starter Kid DSK320C6711 of Texas Instruments

1.2 TMS320C6711 DSP Processor

The C6711 device, Figure 7, is based on the high-performance, advanced VelociTI™ very-long-instruction-word (VLIW) architecture developed by Texas Instruments (TI) [17]. It makes this DSP an excellent choice for multichannel and multifunction applications. With performance of up to 900 million floating-point operations per second (MFLOPS) at a clock rate of 150 MHz, the C6711 device offers cost-effective solutions to high-performance DSP programming challenges. The C6711 DSP possesses the operational flexibility of high-speed controllers and the numerical capability of array processors. This processor has 32 general-purpose registers of 32-bit word length and eight highly independent functional units. The eight functional units provide four floating-/fixed-point ALUs, two fixed-point ALUs, and two floating-/fixed-point multipliers. The C6711 can produce two MACs per cycle for a total of 300 MMACS.

With performance of up to 1200 million floating-point operations per second (MFLOPS) at a clock rate of 200 MHz or 1350 MFLOPS at a clock rate of 250 MHz (for 6711D), the C6711 device also offers cost-effective solutions to high-performance DSP programming challenges. The C6711 DSP also possesses the operational flexibility of high-speed controllers and the numerical capability of array processors. This processor has 32 general-purpose registers of 32-bit word length and eight highly independent functional units. The eight functional units provide four floating-/fixed-point ALUs, two fixed-point ALUs, and two floating-/fixed-point multipliers. The C6711 can produce two MACs per cycle for a total of 400 MMACS.

The C6711 DSP also has application-specific hardware logic, on-chip memory, and additional on-chip peripherals. This C6711 uses a two-level cache-based architecture and has a powerful and diverse set of peripherals. The Level 1 program cache (L1P) is a 4-Kbit direct mapped cache and the Level 1 data cache (L1D) is a 64-Kbit 2-way set-associative cache. The Level 2 memory/cache (L2) consists of a 64-Kbit memory space that is shared between program and data space. L2 memory can be configured as mapped memory, cache, or combinations of the two.

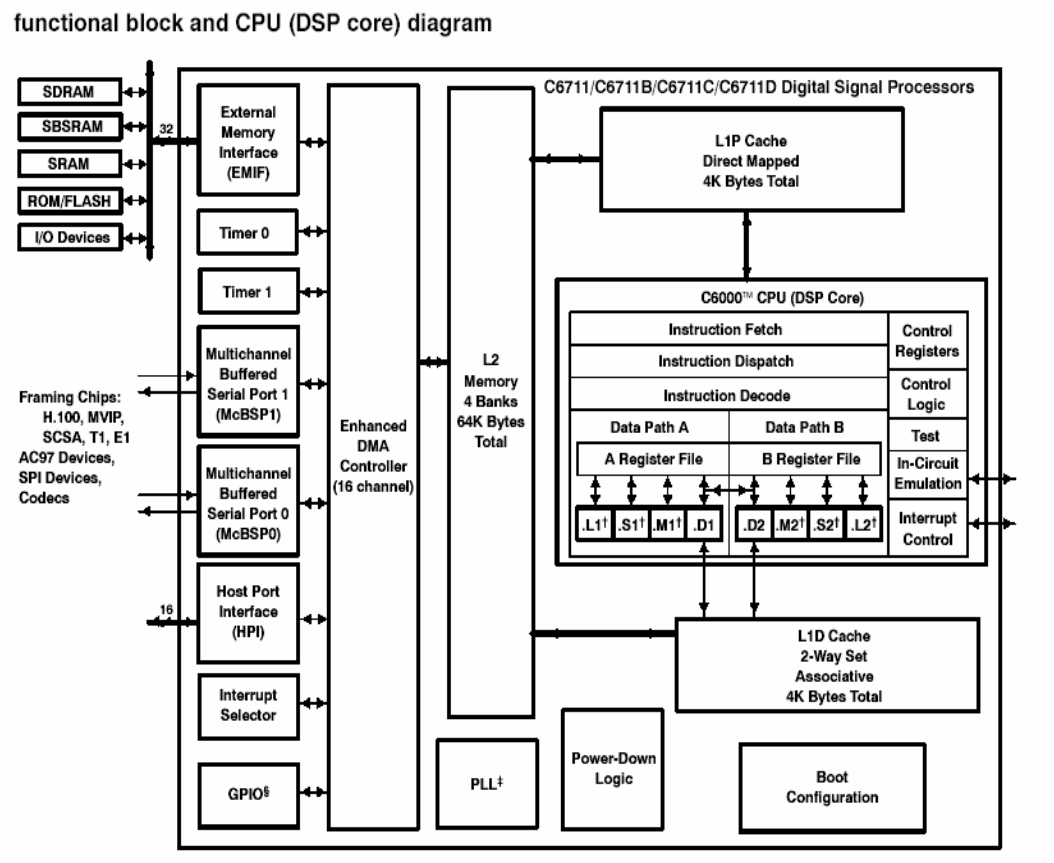


Figure 7. Functional Block Diagram and CPU for DSP320C6711. [spru190 source].

The peripheral set includes two multichannel buffered serial ports (McBSPs), two general-purpose timers, a host-port interface (HPI), and a glueless external memory interface (EMIF) capable of interfacing to SDRAM, and asynchronous peripherals.

The C6711 has a complete set of development tools which includes: C compiler, an assembly optimizer to simplify programming and scheduling, and a Windows™ debugger interface for visibility into source code execution.

1.3 Audio Daughter Card PCM3003

PCM3003 Figure 8, shows a low cost single chip stereo audio CODECs (analog-to-digital and digital-to-analog converters) with single-ended analog voltage input and output. The ADCs and DACs employ delta-sigma modulation with 64X oversampling. The ADCs include a digital decimation filter, and the DACs include an 8X oversampling digital interpolation filter [17]. The DACs also include digital attenuation, de-emphasis, infinite zero detection and soft mute to form a complete subsystem.

PCM3003 operates with left-justified and right-justified formats. PCM3003 provides a power-down mode that operates on the ADCs and DACs independently. Fabricated on a highly advanced CMOS process, PCM3003 is suitable for a wide variety of cost-sensitive consumer applications where good performance is required. The PCM3003's functions include

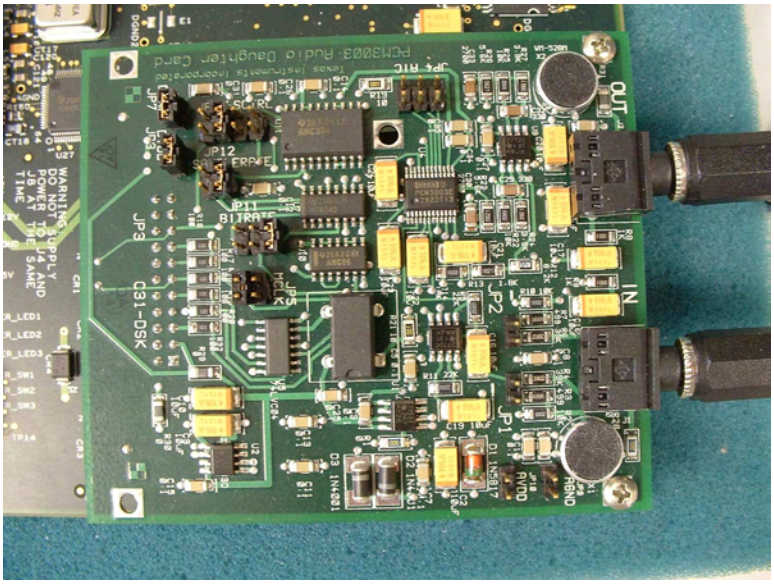


Figure 8. Audio Daughter Card PCM3003 of Texas Instruments

deemphasize, power down, and audio data format selections, which are controlled by hardware.

Hardware Features

- Board Size: 3.5" x 3" Inches
- PCM3003 - Burr Brown 16-/20-Bit Single-Ended Analog Input/Output Stereo Audio Codec (TI Lit. # SPAS079)
- Compatible with TI C31 and C6711 DSKs (attaches via header connector)
- Line-in/out stereo mini audio jacks
- 2 electrets microphones
- Sample rate controlled by 12.288 MHz Oscillator or by DSP timer output pin.
- Separate Analog/Digital power regulators and ground planes for high-resolution audio.
- Jumper Configurations
- 20/16-bit codec selection

- Clock sample rates
- Jack/Microphone selection
- Line/Microphone input gain control selection
- Oscillator/DSP Timer selection
- Sampling rate Jumpers
- Software Features

1.4 Evaluation Module ADS8364EVM

ADS8364 Figure 9, shows a high-speed, low power, dual 16-bit A/D converter that operates from independent 5-V AVdd and DVdd supplies [17]. The digital output is delivered through a built-in buffer circuit that can be powered from DVdd or separate 2.7-V to 5.5-V (BVdd) sources. This allows for flexibility when designing within mixed voltage environments.

The ADS8364EVM includes the following features:

- Full-featured evaluation board for the ADS8364 250-kHz, 16-bit, 6-channel, simultaneous sampling A/D converter
- Analog inputs can be configured as single-ended or differential
- Direct connection to C5000 and C6000 DSK platforms through the 80-pin interface connectors
- Built-in reference
- High-speed parallel interface

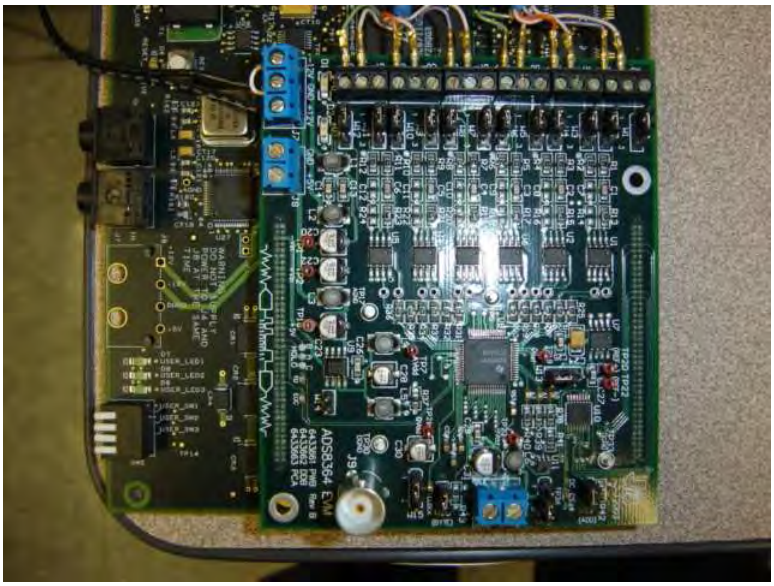


Figure 9 Six-Analog to Digital Converter ADS8364

1.5 Signal Conditioner Adapter AIP-0404-1

AIP-0404-1 Figure 10, shows a hardware prototype developed at the AIP-Laboratory and fully implemented to perform the signal conditioner between the 6-channels (A/D converters) of the ADS8364EVM and the external analog signals in our case a sensor array of 6-microphones [17]. It is developed to introduce a 2.5 V-DC offset and gain factor of 5, in order to offer a correct input signal to the A/Ds converter references. This allows the possibility to connect different input sensors not only microphones to the A/Ds converters. The device AIP-0404-1 is based on the LM3900 OpAmp (Operational Amplifier), this offers good characteristics of input and output impedances. The schematic connection is shown in Figure 11.

The AIP-0404-1 includes the following features:

- Six-channels A/D signal conditioner adapter.
- Six-Analog inputs and outputs.
- Direct connection to the ADS8364EVM and external analog six-sensors.
- Only one 5 VDC input supply.
- Six input plug connectors.

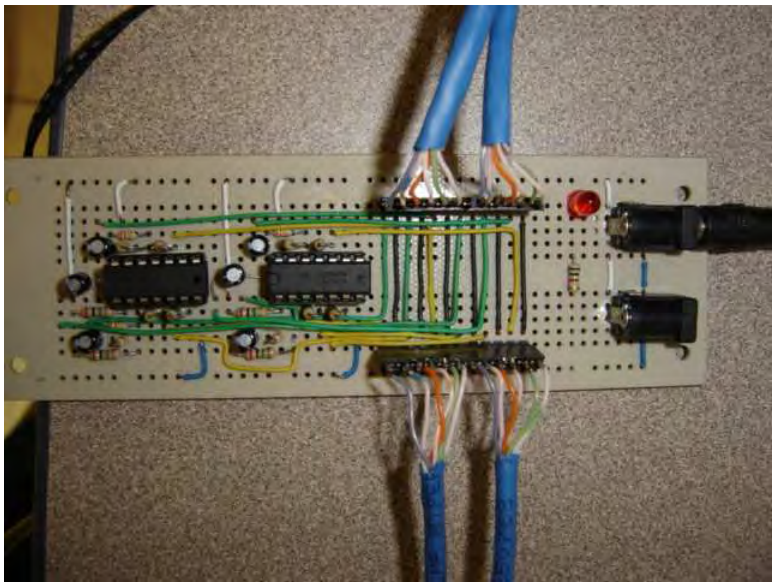


Figure 10. Signal Conditional Adapter for six-channels AIP-0404-1

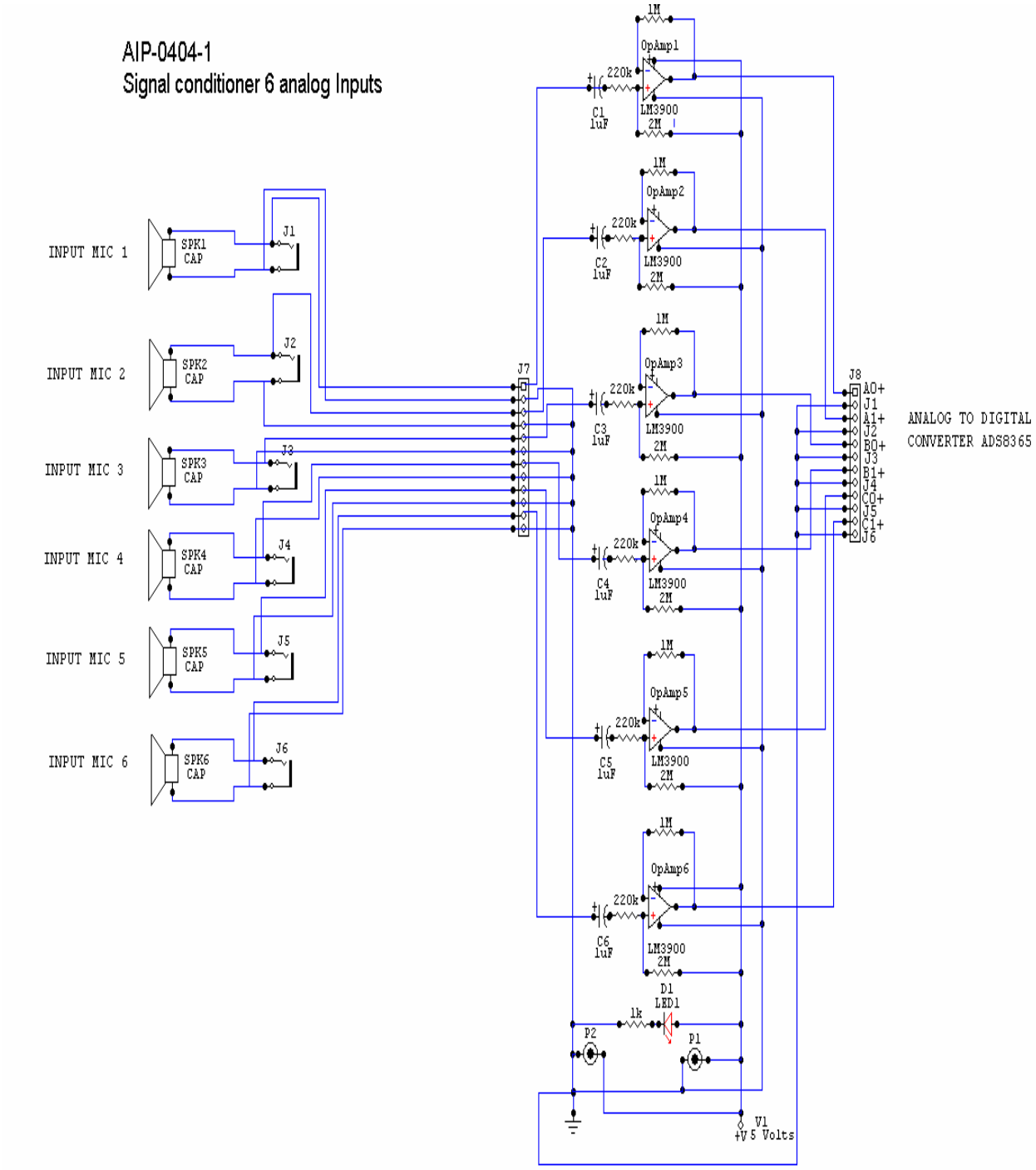


Figure 11. Signal Conditional Adapter Schematic Connection

2. Software Development Tool

Code Composer Studio™ (CCStudio) Development Tools are a key element of the eXpressDSP Software and Development Tools strategy from Texas Instruments. CCStudio delivers all of the host tools and runtime software support for your TMS320 DSP based real-time embedded application to market faster [17]. Familiar tools and interfaces allow users to get started faster than ever before and add functionality to their application thanks to sophisticated productivity tools. CCStudio's easy to use development environment allows DSP designers of all experience levels to move quickly through each phase of the application development process including design, code and build, debug, analyze and optimize. The fully integrated development environment includes real-time analysis capabilities, easy to use debugger, C/C++ Compiler, Assembler, linker, editor, visual project manager, simulators, XDS560 and XDS510 emulation drivers and DSP/BIOS support.

Code Composer Studio's fully integrated Host Tools include:

- TMS320 DSPs C/C++ compiler, assembler, linker and visual linker with optimization feedback
- XDS560™ high speed emulation drivers
- XDS510™ emulation drivers
- Simulators for full devices, CPU only and CPU plus memory for optimal performance
- Integrated Visual Project Manager with source control interface, multi-project support and the ability to handle 1000's of project files

- Editor with CodeMaestro™ technology to simplify the creation of C/C++ programs
- Source Code Debugger common interface for both simulator and emulator targets
- C/C++/Assembly language support
- Simple breakpoints
- Advanced and Hardware breakpoints (Hardware target only)
- Probe points for data injection/extraction
- Pin Connect, Port Connect for simulating real world interfaces (Simulator target only)
- Advanced Watch Window
- Symbol Browser
- DSP/BIOS™ Host Tooling Support (Configure, Real-time analysis and Debug)
- RTDX™ data transfer for real time data exchange between host and target
- Parallel Debug Manager to support multi-processor board debug and analysis
- Profiler to understand code performance
- Update Advisor to keep your system current with the latest releases from TI (requires active subscription)
- Data ConverterPlug-in to auto configure support for Texas Instruments Mixed Signal products
- Online Context Sensitive help
- Online Tutorial for getting started

- CCStudio also delivers critical time saving software for your target application consisting of:
- DSP/BIOS™ Kernel for the TMS320C5000 DSPs (DSP/BIOS™ license included with purchase of Code Composer Studio)
- Pre-emptive multi-threading
- Interthread communication
- Interrupt Handling
- Chip Support Library
- TMS320 DSP Algorithm Standard to enable software reuse
- Chip Support Libraries to simplify device configuration
- DSP Libraries for optimum DSP functionality
- Reference Frameworks - production quality starter code to get you coding faster
- TMS320 DSP Algorithm Standard Developer Kit v2.1 with
- Analysis Toolkit to analyze code performance, including multi-event profiler, code coverage and cache analysis