AN OPERATOR APPROACH TO THE IMPLEMENTATION OF SIGNAL PROCESSING ALGORITHMS ON THE TMS320C6713 DIGITAL SIGNAL PROCESSOR

by

Inerys Otero Pagán

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

MASTER OF ENGINEERING in COMPUTER ENGINEERING

UNIVERSITY OF PUERTO RICO MAYAGÜEZ CAMPUS 2011

Approved by:

Néstor Rodríguez , PhD Member, Graduate Committee

Nayda Santiago, PhD Member, Graduate Committee

Domingo Rodríguez, PhD President, Graduate Committee

Héctor Rosario, PhD Representative of Graduate Studies

Erick Aponte, PhD Chairperson of the Department Date

Date

Date

Date

Date

ABSTRACT

This report details the results of research regarding what can be done to enhance the experience of users in the process of implementing digital signal processing algorithms with several programming tools and devices. The research process resulted in a guide that takes users with any level of expertise in the TMS320C6713 digital signal processing unit, and guides them in a step by step manner, so that they can use the tool or device effectively.

RESUMEN

Este reporte detalla los resultados de la investigación en relación a que se puede hacer para mejorar la experiencia de los usuarios en el proceso de implementar algoritmos de procesamiento digital de señales en diferentes herramientas y dispositivos de programación. Como resultado de la investigación se generó una guía que lleva a los usuarios de cualquier tipo de experiencia con el sistema de procesamiento digital de señales TMS320C6713, paso a paso, de forma tal que puedan usar esta o cualquier otra herramienta similar con mayor eficacia.

•

To my family . . .

ACKNOWLEDGEMENTS

I want to express gratitude to my advisor Prof. Domingo Rodríguez who gave me his unconditional support a helped me to regain my self-confidence. I also want to thank my AIP laboratory partners Gozalo Vaca, David Márquez and Abigail Fuentes for their support and friendship. I want to express special gratitude to Héctor O. Santiago who showed me that I have the strength to achieve this difficult goal and to my family that has always been with me through this difficult but gratifying process.

The Grant from NSF CISE-CNS Grant No. 0424546 provided the funding and the resources for the development of this research under de WALSAIP project.

Table of Contents

A	BSTRACT		II
RI	ESUMEN.		III
A	CKNOWL	EDGEMENTS	V
T	ABLE OF	CONTENTS	VI
т.		Γ	VIII
		1	····· V III
FI	GURE LI	ST	IX
1	INTRO	DUCTION	1
	1.1 Mot	IVATION	2
	1.2 Sum	MARY OF FOLLOWING CHAPTERS	2
2	SIGNA	L PROCESSING FUNDAMENTALS	4
3	TMS32	0C6713 DSP DEVELOPMENT SYSTEM	
	3.1 Cod	e Composer Studio IDE (CCS)	
	3.2 CCS	INSTALLATION AND SUPPORT	
	3.3 CCS	SETUP AND INITIALIZATION	
	3.3.1	Selecting Simulation Environment	
	3.3.2	Selecting Emulation Environment	
	3.4 GEN	ERAL ALGORITHM IMPLEMENTATION ON THE BOARD	
	3.4.1	Types of Useful Files	
	3.4.2	DSK Support Tools	
	3.5 PRO	GRAMMING EXAMPLES TO TEST THE DSK TOOLS	
	3.3.1 2.5.2	Example 1. Hello World:	
	252	Example 2. Fust Fourier Transform (FFT) (Created Froject Version)	
	3.3.3 3.5.1	Example 3. Fusi Fourier Transform (FFT) (Creating the Froject Version) Frample 4. Corner Turning (Created Project Version)	
	355	Example 5. Corner Turning (Creating the Project Version)	
	5.5.5 GT GT G	Example 5. Corner Furning (Creating the Project Version)	
4	SIGNA	L OPERATOR FORMULATIONS FOR MATLAB IMPLEMENTATION	
	4.1 LINE	AR SHIFT INVARIANCE SYSTEMS	
	4.1.1	Matrix Representation of LSI-FIR Systems	
	4.1.2	Spectral Properties of LSI-FIR systems	
	4.2 CYC	LIC MATRIX	
	4.5 DISC	RETE FOURIER TRANSFORM	
	4.4 UIH	AMADD DDODUCT	
	4.5 HAD	AMARD FRODUCT	
	4.0 CON	Discrete Filter	
	462	Response of a Filter to a Finite Signal	
	4.6.3	Finite Response Filters to a Finite Impulse	
5	IMAG	ING FORMATION ALGORITHM	
	5.1 SAR	Imaging Formation Design	
	5.2 IMAG	GE FORMATION RESULTS OBTAINED	
	5.2.1	TMS320C6713 Emulation results for 128x128 Raw Data	
	5.2.2	TMS320C6713 Emulation results for 256x256 Raw Data	

5.2.3 TMS320C6713 Emulation results for 512x512 Raw	• Data
5.3 EXAMPLE. IMAGING FORMATION (CREATING THE PROJEC	<i>T VERSION</i>)135
6 CONCLUSION AND FUTURE WORK	
REFERENCES	
APPENDIX A. TMS320C6713 DSP ATRIBUTES	

Table List

Tables	Page
Table 1: TMS320C6713 DSK Features	156

Figure List

Figure 1: Sample of Programming Environment: "Code Composer Studio"	28
Figure 2: Code Composer Studio (CCS) v3.3 Installation Wizard	29
Figure 3: CCS System Requirements Verification	30
Figure 4: CCS License Agreement	31
Figure 5: CCS Instalation Type Selection	32
Figure 6: CCS Destination Folder	33
Figure 7: Code Composer Studio v3.3 Installation	34
Figure 8: CCS Installation Progress	35
Figure 9: Finished CCS Installation	
Figure 10: CCS Emulation Drivers Main Menu Window	37
Figure 11: CCS 3.1 Planinum Driver	
Figure 12: CCS 3.1 Emulation Drivers Installation Window	
Figure 13: CCS Emulation Drivers Setup Type Selection	40
Figure 14: Selection of Destination Location for CCS 3.1 Emulation Drivers	41
Figure 15: CCS 3.1 Emulation Drivers Installation Progress	42
Figure 16: CCS 3.1 Emulation Drivers Installation Progress	43
Figure 17: CCS Emulation Drivers Installation Closure	44
Figure 18: CCS 3.1 Planinum Driver	45
Figure 19: CCS Emulation Drivers Main Menu Window	46
Figure 20: Location of CCS in Windows XP	47
Figure 21: Code Composer Studio Setup	48
Figure 22: Selecting Simulation Environment	50
Figure 23: TMS320C6713 "Digital Starter Kit" (DSK)	51
Figure 24: TMS320C6713 DSP Board	51
Figure 25: Emulation Environment Selection	53
Figure 26: Establish the Connection between the CCS and the TMS320C6713 DSP	54
Figure 27: Window for the creation of a New Project	58
Figure 28: Project Folders	59
Figure 29: Project Files	61
Figure 30: Setting the Target Version	62
Figure 31: Specifying the Chip Architecture	63
Figure 32: Libraries Nedded for the Project	64
Figure 33: Compiling Results	65
Figure 34: Results Obtained after Run the Algorithm "hello world"	65
Figure 35: FFT Files	66
Figure 36: Open FFT Project	67
Figure 37: FFT Project Selection	67
Figure 38: CCS Environment for FFT Example	68
Figure 39: FFT Project Compiling Results	69
Figure 40: "Load Program" Location	69

Figure 41: "FFTproject.out" File Location	70
Figure 42: Downloading the "FFTproject.out" File to the TMS320C6713 DSP	70
Figure 43: Results Obtained after Run the FFT Algorithm	71
Figure 44: FFT Project Files	72
Figure 45: Creating a New Project	73
Figure 46: Window for the Creation of a New Project	74
Figure 47: FFT Project Folder	74
Figure 48: FFT Project Files	75
Figure 49: Adding Files to the Project	76
Figure 50: Project Files	77
Figure 51: Build Option Setting Location	78
Figure 52: Setting the Target Version	78
Figure 53: Memory Model Type Selection	79
Figure 54: Specifying the Chip Architecture	80
Figure 55: Libraries Needed for the Project	81
Figure 56: FFT Project Compiling Results	81
Figure 57: "Load Program" Location	82
Figure 58: "FFTproject.out" File Location	82
Figure 59: Downloading the "FFTproject.out" File to the TMS320C6713 DSP	83
Figure 60: Results Obtained after Run the FFT Algorithm	84
Figure 61: Corner Turning Files	85
Figure 62: Open "Corner Turning" Project	86
Figure 63: Corner_Turning Project Selection	87
Figure 64: CCS Environment for Corner Turning Example	87
Figure 65: Corner Turning Compiling Results	
Figure 66: "Load Program" Location	88
Figure 67: "Corner_Turning.out" File Location	
Figure 68: Downloading the Corner_Turning.out File to the TMS320C6713 DSP	89
Figure 69: Results Obtained after Run the Algorithm "Corner_Turning"	90
Figure 70: Corner Turning Files	91
Figure 71: Creating a New Project	92
Figure 72: Window for the Creation of a New Project	93
Figure 73: Corner_Turning Project Folder	93
Figure 74: Corner Turning Project Files	
Figure 75: Adding Files to the Project	
Figure 76: Project Files	
Figure 77: Build Option Setting Location	
Figure 78: Setting the Target Version	
Figure /9: Memory Model Type Selection	
Figure 80: Specifying the Chip Architecture	
Figure 81: Libraries Needed for the Project	100
Figure 82: Compiling Results	100
Figure 83: "Load Program" Location	101

Figure 84: Corner_Turning.out File Location	102
Figure 85: Downloading the Corner_Turning.out File to the TMS320C6713 DSP	102
Figure 86: Corner Turning Input Data and Validation Files	103
Figure 87: Results Obtained after Run the Algorithm "Corner_Turning"	104
Figure 88: Discrete System Block Diagram	119
Figure 89: Discrete Filter Block Diagram	120
Figure 90: FIR Filter Block Diagram	121
Figure 91: FIR Filter Block Diagram	121
Figure 92: Averaging Filter Block Diagram	122
Figure 93: Range and Azimuth Direction	125
Figure 94: SAR Image Formation Diagram Procedure	127
Figure 95: Raw Data	128
Figure 96: Data Compressed in Range	129
Figure 97: Applying Corner Turning to Data Compressed in Range Direction	129
Figure 98: Data Compressed in Azimuth Direction	130
Figure 99: Raw Data	131
Figure 100: Data Compressed in Range	131
Figure 101: Applying Corner Turning to Data Compressed in Range Direction	132
Figure 102: Data Compressed in Azimuth Direction	132
Figure 103: Raw Data	133
Figure 104: Data Compressed in Range	133
Figure 105: Applying Corner Turning to Data Compressed in Range Direction	134
Figure 106: Data Compressed in Azimuth Direction	134
Figure 107: ImagingFormation Files	136
Figure 108: Creating a New Project	137
Figure 109: Window for the Creation of a New Project	138
Figure 110: "ImageFormation" Project Folder	138
Figure 111: Image Formation Project Files	140
Figure 112: Adding Files to the Project	142
Figure 113: Project Files	143
Figure 114: Build Option Setting Location	144
Figure 115: Setting the Target Version	145
Figure 116: Memory Model Type Selection	146
Figure 117: Building options for Linker→Basic	147
Figure 118: Specifying the Chip Architecture	148
Figure 119: Libraries Needed for the Project	149
Figure 120: "ImageFormation" Project Compiling Results	149
Figure 121: "Load Program" Location	151
Figure 122: "ImageFormation.out" File Location	151
Figure 123: Downloading the "ImageFormation.out" File to the TMS320C6713 DSP	152

1 INTRODUCTION

At the time of working with a new algorithm design and development project, the task of being able to connect the integrated efforts of software and hardware design usually takes a lot of time and in most cases it requires the efficient management of many resources. Algorithms developed for a specific architecture should work well after the testing and refining processes are completed. Problems emerge when trying to use these same algorithms over other architectures. To use them on a new architecture, for instance, they may require a lot of changes or practically develop a new hardware/software integration scheme. The problem addressed in this project deals with the need to develop a system level design approach to assist in the design and development of a certain class of signal processing algorithms. In particular, this class of algorithms represents finite dimensional linear shift invariant systems. This type of systems always admits a matrix representation and, hence, can be treated as finite dimensional operators. Signal algebra methods can then be used to study the properties of these operators in order to arrive at desirable algorithm formulations for integrated hardware/software implementations on a targeted architecture. The development of an appropriate system level design approach for algorithm design and development could contribute to the task of software reuse on different architectures with a reduced amount of code alteration.

The linear operator nature of the class of systems addressed in this proposal allows for the representation of these systems using and iconic or block diagram approach. In this context, a typical finite dimensional shift invariant system may be represented as 3-tuple entity: 1) a set of causal input signals of finite order, 2) an operator, linear transformation, or agent, and 3) a set of output signals. The operator, linear transformation, or agent acts on a given element of the set of input signals and it produces an element of the set of output signals. The fundamental purpose during this research project was to develop working tools to allow TMS320c6713 DSK future users to work in a more efficient and rapid manner. As part of this research project, a user's guide on implementing digital signal processing (DSP) application programs for the SDK6713 board was designed. The main purpose was to study and analyze the learning process involved as a specific number of individuals followed the guide step by step, in order to interact and use Code Composer Studio v3.3 IDE to develop different application examples for the DSP board. The level of difficulty in learning how to implement the application program, and becoming familiarized with Code Composer and the DSP board itself, was taken into consideration for this particular study.

1.1 Motivation

The main motivation to work in this project was that during the literature review process I realized that there exists a big gap between the software and hardware area, and how to use different algorithms in different architectures without any major problems. I also noticed that this issue is a common concern in the engineering and research areas. For these reasons I think that my work will be a great contribution for the field of signal processing algorithm design and development and it will serve as a starting reference point for future investigations.

1.2 Summary of Following Chapters

This document is organized as follow: Chapter 2 presents important signal processing fundamentals theory which is essential to understand de development of this project. Chapter 3 presents a description of the TMS320C6713 Digital Starter Kit (DSK) and its development environment Code Composer Studio v3.3. It also includes a detailed TMS320C6713 user guide that describes how to use the Code Composer Studio for the creation of the following project examples: *Hello World!*, *Fast Fourier Transform (FFT) and Corner Turning*.

Chapter 4 presents a description of signal operators formulation. Some of these operators are used in the implementation of the image formation advanced algorithm. Chapter 5 presents the SAR Image Formation design description and the TMS320C6713 DSP User Guide for this example. Chapter 6 presents the conclusion of the project and potential future projects.

2 SIGNAL PROCESSING FUNDAMENTALS

Digital Signal Processing:

Digital Signal Processing is defined as the treatment of signals using digital electronics technology and digital computation techniques, in an algorithmic manner, to extract information important or relevant to a user. The diagram below depicts a basic digital signal processing system conformed of three basic components: an analog-to-digital (A/D) conversion system, a digital processor system, and a digital-to-analog (D/A) conversion system. The digital processor system takes a digital signal as input and produces another digital signal as output. An analog-to-digital system converts a continuous-domain signal or analog signal into a digital signal into an analog signal or continuous-domain signal. A continuous-domain signal is normally referred to as a continuous-domain signal or simply a continuous signal since it can describe the variations or scales of a physical quantity such as pressure, temperature, or sound as a function of time. Examples of continuous-time signals such as speech signals abound all around us.



Continuous-domain Signal or Analog Signal:

A continuous-domain signal or analog signal denotes a function x whose value x(t) is defined for every value t of a set D called the domain of the function.

Discrete-time Signal Processing:

Discrete-time Signal Processing is a more general treatment of signals, which includes digital signal processing, using other technologies such as surface acoustic wave (SAW) devices and charged-coupled devices (CCDs) as well as analog computation techniques such as optical and biological computing.

Discrete Signal:

A discrete signal or discrete function has as its domain a discrete set such as the set of integers \mathbb{Z} . The number of elements in the discrete set serving as the domain of the discrete signal may be finite or infinite. As an example of a discrete signal we have the following function

$$x = \{x[n] = 2^n, n \in \mathbb{Z}\} = \left\{\dots, -\frac{1}{8}, -\frac{1}{4}, -\frac{1}{2}, 1, 2, 4, 8, \dots\right\}$$

A signal which is discrete is also called a *sequence*. As an example of a finite sequence, we provide the following function over the finite set $\mathbb{Z}_4 = \{0, 1, 2, 3\}$:

$$x = \left\{ x[n] = \cos \frac{2\pi n}{4}, \qquad n \in \mathbb{Z}_4 \right\} = \{1, 0, -1, 0\}$$

A discrete signal can be obtained from a continuous signal by making the time axis a discrete set. That is, if we have a continuous signal $x: \mathbb{R} \to \mathbb{C}$

$$t \rightarrow \mathbf{x}(t) = e^{+j2\pi f_0 t}$$
, $j = \sqrt{-1}$, f_0 is a constant.

Digital Signal

A digital signal has as its range a finite discrete set.

Causal Discrete Signal:

It is a sequence $\{x[n]\}$ such that x[n] = 0 for n < 0.

Discrete Finite Causal Signals:

Let $Z_N = \{0, 1, 2, ..., N-1\}$. Example $Z_5 = \{0, 1, 2, 3, 4\}$.

A sequence $\{y[n]\}$ is causal and finite if $\{y[n], n \in \mathbb{Z}_N\}$. In this case we say that the signal has length N.

Discrete System:

A discrete system T takes as input a discrete signal, say $\{x[n]\}$ and it produces as output another discrete signal, say y[n].

Block Diagram Representation of a Discrete System:

A discrete system is usually represented using a rectangular figure, called a black box. To the left of the box an inward directed arrow is attached to indicate the input signal to the system. To the right of the box an outward directed arrow is attached to indicate the output signal produced by the system. Two modalities are commonly used to describe the input and output signals as depicted in the diagrams below. The diagram on the left describes the input and output signals as sets but does not identify the domain of the signals. The diagram on the right depicts an arbitrary element of the input and output signals and provides the domains where theses signals are evaluated.

Discrete Linear System:

The system T is linear if:

$$T\{ax_1[n] + bx_2[n]\} = aT\{x_1[n]\} + bT\{x_2[n]\}$$

Simplified condition:

1. Additivity or Superposition: a = b = 1

$$T\{x_1[n] + x_2[n]\} = T\{x_1[n]\} + T\{x_2[n]\}$$

2. Homogeneity: b = 0

$$T\{ax_1[n]\} = aT\{x_1[n]\}$$

For the system to be linear it must satisfy, both, the additivity and homogeneity conditions.

Example: Squarer Discrete System

$$x[n], n \in \mathbb{Z}$$
System
T
$$y[n] = T\{x[n]\} = x[n] \bullet x[n] = x^{2}[n]$$

Check the homogeneity condition:

1.
$$T\{x_1[n]\} = x_1^2[n]$$

 $aT\{x_1[n]\} = ax_1^2[n]$
2. Let $g[n] = ax_1[n]$
 $T\{g[n]\} = g^2[n]$

Substituting for $g[n] = ax_1[n]$, we obtain

$$T\{ax_1[n]\} = (ax_1[n])^2 = a^2 x_1^2[n]\}$$

Therefore the system is not linear.

Discrete Shift Invariant or Time Invariant System:

A system T is shift invariant or time invariant if it satisfies the following condition:

$$y[n-n_0] = T\{x[n-n_0]\}$$

Discrete Filter:

A discrete filter T is a system, which is, both, linear and time invariant. *Note:* Any discrete signal can be expressed as a sum of delayed unit sample functions:

$$x[n] = \sum_{k=-\infty}^{\infty} x[k] \delta[n-k]$$

Finite Impulse Response Filter:

It is any filter whose impulse response signal is of final duration, that is, it has duration equal to, say N_h , an arbitrary but fixed length.

Causal Filter:

A filter T is called causal if the impulse response signal of the filter is a causal signal.

$$h[n] = \begin{cases} h[n], & n \ge 0\\ 0, & n < 0 \end{cases}$$

RC-Filter:

The figure below depicts an example of an electric circuit modeling a continuous passive RC-filter. The filter is called continuous or analog due to the fact that it operates as a rule which assigns to an input signal, x(t), $t \in \mathbf{R}$ an output signal, y(t), $t \in \mathbf{R}$. It is called RC since all the components in the circuit are made up of either resistors or capacitors. Each resistance element in the circuit models a dissipative load. Also, each capacitive element in the circuit models an energy storage load. The overall circuit is conformed by two basic first

order filters coupled in cascade. A first order continuous passive filter may be described by a first order differential equation with constant coefficients.



General Continuous Filters:

In general, a continuous passive filter with input the signal x(t) and output the signal y(t) may be represented in terms of a differential equation of the form:

$$a_{M} \frac{d^{M}}{d_{t}^{M}}(y(t)) + a_{M-1} \frac{d^{M-1}}{d_{t}^{M-1}}(y(t)) + \dots + a_{o} y(t) = b_{N} \frac{d^{N}}{d_{t}^{N}} x(t) + b_{N-1} \frac{d^{N-1}}{d_{t}^{N-1}} x(t) + \dots + b_{o} x(t)$$

This can also be expressed as follows using summation expressions:

$$\sum_{m=0}^{M} a_m \frac{d^m}{d_t^m} y(t) = \sum_{n=0}^{N} b_n \frac{d^n}{d_t^n} x(t)$$

The input signal x(t) is also called the forcing function of the continuous filter.

Discrete Filters:

Discrete filters may be represented using difference equations of the form

$$\sum_{k=0}^{N} d_k y[n-k] = \sum_{k=0}^{M} p_k x[n-k],$$

where the sequence x[n], $n \in Z$, represents and arbitrary input signal, the sequence y[n], $n \in Z$ represents the output signal, and d_k , p_k are complex scalars. The output signal y[n], $n \in Z$ can be expressed in terms of the input signal and past values of the output signal.

Discrete Filter Implementation:

A large class of discrete filters can be expressed in terms of a difference equation of the form:

$$\sum_{k=0}^{M} d_k y[n-k] = \sum_{k=0}^{N} b_k x[n-k]$$

This is the only type of filters that we will study in this primer.

Filter Operators: The diagrams below represent operators to implement all filters



Discrete Time Fourier Transform:

Let x[n] be a discrete signal. Its discrete-time Fourier transforms is defined as follows

$$F\{x[n]\} = DTFT\{x[n]\} = \sum_{n \to -\infty}^{\infty} x[n]e^{-j\omega n}, \omega \in \mathbb{R}; j = \sqrt{-1}$$

Remember that $e^{-j\omega n} = \cos \omega n - j \sin \omega n$. This implies that the DTFT of the signal x[n] is a complex function signal.

Periodic Property of the DTFT

Example: The DTFT of a Signal is Always Periodic Modulo 2π

A signal $X(\omega)$ is periodic with period ω_p if the following condition is satisfied:

$$X(\omega + \omega_p) = X(\omega).$$

Define $X(\omega) = \Im\{x[n]\} = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}; \omega \in \Re$

If we let ω go to $\omega + \omega_p$ by changing the argument of $X(\omega)$, we get

$$X(\omega + \omega_p) = \sum_{n = -\infty}^{\infty} x[n] e^{-j(\omega + \omega_p)n} = \sum_{n = -\infty}^{\infty} x[n] e^{-j\omega n} e^{-j\omega_p n}$$

Allow $\omega_p = 2\pi$

Then,
$$e^{-j\omega_p n} = e^{-j2\pi n} = \cos(2\pi n) - j\sin(2\pi n), n \in \mathbb{Z}$$

We then have the following result:

$$X(\omega + \omega_p)\Big|_{\omega_p = 2\pi} = \sum_{n = -\infty}^{\infty} x[n]e^{-j\omega n} = X(\omega)$$

Discrete Fourier Transform:

This is only defined for finite discrete signals, say of length N.

Let x[n] be a discrete signal of length N. Its DFT is given by the following equation:

$$X(\omega) \bigg|_{\omega = \omega_k = \frac{2\pi k}{N}} = X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\omega_k n} = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi k n}{N}}, k \in \mathbb{Z}_N$$

The DFT can be represented in matrix form:

$$X = F_N x$$

When x is a column vector and is the input signal, X is a column vector and it is the output signal or transformed signal and F_N is a square matrix of order N called the Fourier matrix.

Periodic Discrete Signals:

A signal x[n] is said to be periodic, with fundamental period N, if the following condition is satisfied:

$$x[n+qN] = x[n], \text{ for } q \in \mathbb{Z}$$

Example:



The signal x[n] has a fundamental period equal to N. In this case N = 4:

Let q = 1

$$x[n+4] = x[n]$$

For $n = -3$
$$x[-3+4] = x[-3]$$
$$\therefore x[-3] = x[1]$$

Observation:

Any periodic signal x[n] with fundamental period N, can uniquely be represented by a causal signal x[n], of length equal to N, whose values are equal to the N values of the periodic signal in its fundamental period.

Cyclic or Circular Convolution of Periodic Signals:

Given two periodic signals, say x[n] and h[n], with the same fundamental period N, the cyclic or circular convolution of x[n] and h[n] is a new periodic signal

$$y[n] = x[n]O_Nh[n]$$

with fundamental period also equal to N and which is defined by the following equation

$$y[n] = \sum_{k=0}^{N-1} x[k]h[n-k]; n \in \mathbb{Z}_{N}.$$

Circular or Cyclic Convolution of Periodic signals using Causal Representations:

Let x[n] and h[n] be two periodic signals with fundamental period N. Let x[n] and h[n] be their causal representations, respectively. The circular or cyclic convolution of the causal representation is a new causal signal, of length N, and denoted by y[n].

The signal y[n] is given by

$$y[n] = \sum_{k=0}^{N-1} x[k]h[< n-k >_N]; n \in \mathbb{Z}_N$$

The symbol $\langle p \rangle_N$ denotes the remainder of *p* after being divided by *N*. This is sometimes called "*p* modulo *N*". The periodic signal *y*[*n*] is obtained from its causal representation *y*[*n*] by repeating the causal signal *y*[*n*], starting at the fundamental period.

Observation:

- The efficiency of computing a cyclic convolution operation can be improved using a Fast Fourier Transform (FFT) algorithm. An FFT algorithm is an efficient method for computing the DFT.
- Any linear convolution can be computed using a cyclic convolution operation. Remember that the filters only do linear convolution.
- 3. The Discrete Time Domain Convolution Theorem states that the DFT of the cyclic convolution of two discrete signals is equal to the product of the DFT of each of the individual signals.

Inverse DTFT:

Let $X(\omega)$ be the DTFT of the signal x[n]. We can recover the signal x[n] from its Fourier transform by using the formula (IDTFT):

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(\omega) e^{+j\omega n} d\omega.$$

Example:

Obtain the DTFT of $x[n] = \alpha^n u[n], |\alpha| < 1$.

Solution:

$$X(\omega) = DTFT\{x[n]\} = \sum_{n=-\infty}^{\infty} \alpha^n \mu[n] e^{-j\omega n} = \sum_{n=0}^{\infty} \alpha^n e^{-j\omega n}$$

Expanding, we get

$$X(\omega) = 1 + \alpha e^{-j\omega} + \alpha^2 e^{-j2\omega} + \alpha^3 e^{-j3\omega} + \cdots$$
$$X(\omega) = \sum_{n=0}^{\infty} (\alpha e^{-j\omega})^n$$

Let
$$b = \alpha e^{-j\omega}$$

 $X(\omega) = \sum_{n=0}^{\infty} b^n = 1 + b + b^2 + b^3 + \cdots$
 $X(\omega) - bX(\omega) = 1$
 $(1-b)X(\omega) = 1$
 $\therefore X(\omega) = \frac{1}{1-b} = \frac{1}{1-\alpha e^{-j\omega}}$

Filter Design: First-order



FIR

 $h_D[n] = \begin{cases} h[n], & n \in \mathbf{Z}_{N} \\ 0, & \text{otherwise} \end{cases}$

FIR Filter Design: Windowing Technique

Given the DTFT $X(\omega)$ of an arbitrary signal x[n], the signal can be recovered from its spectrum using the following formula for inverse DTFT:

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(\omega) e^{+j\omega n} d\omega; \ n \in \mathbb{Z}$$

If the signal $X(\omega)$ is the frequency response of a filter, then $X(\omega) = H(\omega)$. The impulse response is then obtained from the frequency response as follows:

$$h[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} H(\omega) e^{+j\omega n} d\omega, \ n \in \mathbb{Z}$$
$$h: \mathbb{Z} \to C$$

Low-pass FIR Filter Design:

- 1. Select an ideal filter with a prescribed frequency response.
- 2. Take the inverse DTFT to obtain an infinite response.
- 3. Multiply in the time domain by a window with the desired order or length. Allow this first window to be rectangular.
- 4. Multiply the result of part 3 by a new window to improve the desired frequency response.

Fast Fourier Transform:

It is an algorithm to compute the discrete Fourier transform in an efficient manner. There are many fast Fourier transform algorithms. We will concentrate on the algorithms designed by John Tukey and James Cooley in 1965 and are commonly known as Cooley – Tukey FFT algorithms.

Cooley – Tukey FFT algorithms:

The objective is to develop an efficient algorithm to compute the matrix-vector

operation:

 $X = f_n x$

The direct computation of this matrix-vector operation required N^2 multiplications and N(N-1) additions.

Example: N = 4

$$\mathbf{X} = f_4 \mathbf{x} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w_4 & w_4^2 & w_4^3 \\ 1 & w_4^2 & 1 & w_4^2 \\ 1 & w_4^3 & w_4^2 & w_4 \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix} = \begin{bmatrix} \mathbf{X}[0] \\ \mathbf{X}[1] \\ \mathbf{X}[2] \\ \mathbf{X}[3] \end{bmatrix}$$

$$w_{4} = e^{-j\frac{2\pi}{4}}$$
$$w_{4}^{6} = e^{-j\frac{2\pi6}{4}} = \underbrace{e^{-j\frac{2\pi2}{4}}}_{w_{4}^{2}} \cdot \underbrace{e^{-j\frac{2\pi4}{4}}}_{1}$$

For $N = 2^{M}$, a power of 2, the Cooley-Tukey algorithm reduces the number of multiplications to $N \log_2 N$.

Example:

Ν	Direct Method	Cooley-Tukey Algorithm
1024	$(1024)^2$ multiplications	$1024 \underbrace{\log_2 1024}_{10} = (10)1024$

Cooley-Tukey Algorithm Technique:

Additive property of the DFT:

Example: N = 4

$$\mathbf{X} = F_4 \mathbf{x} = F_4 \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix}$$

1. We will represent x as a sum of two vectors: $x[n] = x_e[n] + x_0[n]$, $n \in Z_4$

$$x = \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix} = \begin{bmatrix} x[0] \\ 0 \\ x[2] \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ x[1] \\ 0 \\ x[3] \end{bmatrix}$$
$$x_e \qquad x_0$$

2. We will use the linearity property of the DFT $F_4 x = F_4 (x_e + x_0) = F_4 x_e + F_4 x_0$

sparse matrix

$$F_{4}x_{e} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w_{4} & w_{4}^{2} & w_{4}^{3} \\ 1 & w_{4}^{2} & 1 & w_{4}^{2} \\ 1 & w_{4}^{3} & w_{4}^{2} & w_{4} \end{bmatrix} \begin{bmatrix} x[0] \\ 0 \\ x[2] \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & w_{4}^{2} & 0 \\ 1 & 0 & w_{4}^{2} & 0 \end{bmatrix} \begin{bmatrix} x[0] \\ 0 \\ x[2] \\ 0 \end{bmatrix} = \begin{bmatrix} x[0] + x[2] \\ x[0] + w_{4}^{2}x[2] \\ x[0] + w_{4}^{2}x[2] \\ x[0] + w_{4}^{2}x[2] \end{bmatrix}$$

$$F_{4}x_{e} = \begin{bmatrix} 1 & 1 \\ 1 & w_{4}^{2} \\ 1 & 1 \\ 1 & w_{4}^{2} \end{bmatrix} \begin{bmatrix} x[0] \\ x[2] \end{bmatrix} = \begin{bmatrix} x[0] + x[2] \\ x[0] + w_{4}^{2}x[2] \\ x[0] + x[2] \\ x[0] + w_{4}^{2}x[2] \end{bmatrix}$$
$$w_{4}^{2} = e^{-j\frac{2\pi^{2}}{4}} = e^{-j\pi} = \cos \pi - j \sin \pi = -1$$



Butterfly Block Diagram (Flow Diagram)

Representation of the FFT:



$$F_4 x_e = \begin{bmatrix} F_2 \\ F_2 \end{bmatrix} [x[0] \quad x[2]]$$

We want to compute

$$\checkmark F_4 x = F_4 x_e + F_4 x_0$$

16 multiplications

12 summations

1.
$$F_4 x_e = \begin{bmatrix} F_2 \\ F_2 \end{bmatrix} \begin{bmatrix} x[0] \\ x[2] \end{bmatrix} = F_4 \begin{bmatrix} x[0] \\ 0 \\ x[2] \\ 0 \end{bmatrix}$$

2.
$$F_4 x_0 = F_4 \begin{bmatrix} 0 \\ x[1] \\ 0 \\ x[3] \end{bmatrix}$$

In general, we want to know

DFT
$$\{x[n-n_0]\} = \sum_{n=0}^{N-1} x[n-n_0] w_n^{Kn}$$

 $m = n - n_0; n = m + n_0$

DFT
$$\{x[n - n_0]\} = \sum_{m=-n_0}^{m=(N-1-n_0)} x[m] W_n^{K(m+n_0)}$$

= $W_N^{Kn_0} \sum_{m=-n_0}^{m=N-1-n_0} x[m] W_N^{Km}$



Example:



Remainder
$$\left(\frac{P}{N}\right) = \langle p \rangle_N = \langle p + qN \rangle_N$$

$$<-3>_4=<-3+q4>_4=<-3+4>_4=<1>_4=1$$

$$x[-3] \leftrightarrow x[1]$$

$$G[K] = W_N^{Kn_0} \left(\sum_{m=0}^{N-1} x[n] \cdot W_N^{Km} \right)$$

Hadamard product
DFT
$$\{x[n-n_0]\} = W_N^{Kn_0} \bullet X[K]$$

 $W_N^{Kn_0} = e^{-j\frac{2\pi Kn_0}{N}}$

Express $F_4 x_0$ in matrix form.

$$G[k] \rightarrow long. N$$

$$G[k] = W_N^{K_{no}} \cdot X[k]$$

$$G[k] = W_N^{K_{no}} \bullet (F_4 x)$$

$$F_4 x_0 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w_4 & w_4^2 & w_4^3 \\ 1 & w_4^2 & 1 & w_4^2 \\ 1 & w_4^3 & w_4^2 & w_4 \end{bmatrix} \begin{bmatrix} 0 \\ x[3] \\ 0 \\ x[3] \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & w_4 & 0 & w_4^3 \\ 0 & w_4^2 & 0 & w_4^2 \\ 0 & w_4^3 & 0 & w_4 \end{bmatrix} \begin{bmatrix} 0 \\ x[3] \\ 0 \\ x[3] \end{bmatrix}$$

Compacting, we get

$$F_{4}x_{0} = \begin{bmatrix} 1 & 1 \\ w_{4} & w_{4}^{3} \\ w_{4}^{2} & w_{4}^{2} \\ w_{4}^{3} & w_{4} \end{bmatrix} \begin{bmatrix} x[1] \\ x[3] \end{bmatrix} = \begin{bmatrix} x[1] + x[3] \\ w_{4}x[1] + w_{4}^{3}x[3] \\ w_{4}^{2}x[1] + w_{4}^{2}x[3] \\ w_{4}^{3}x[1] + w_{4}x[3] \end{bmatrix}$$

We know that

DFT_N {
$$x[< n - n_0 >_N]$$
} = W_N^{Kn₀} · X[K]
Example: $N = 4$, $x[n] = {x[0], x[1], x[2], x[3]}$
 $y[n] = x[< n - n_0 >_N]$; $n_0 = 2$
 $y[n] = x[< n - 2 >_4]$; $n \in Z_4$
 $y[0] = x[< 0 - 2 >_4] = x[2]$
 $y[1] = x[<1 - 2 >_4] = x[3]$
 $y[2] = x[<2 - 2 >_4] = x[0]$
 $y[3] = x[<3 - 2 >_4] = x[1]$

$$_N = _N$$

$$_{N} = \text{Remainder}\left(\frac{P + qN}{N}\right) =$$

Remainder
$$\left(\frac{P}{N}\right)$$
 + Remainder $\left(\frac{qN}{N}\right)$
<1>₄=1
<5>₄=<1+4>₄=<1>₄ + <4>₄
<9>₄=<1+2·4>₄=<1>₄ + <8>₄
<21>₄=<1+5·4>₄=<1>₄ + <8>₄
<-21>₄=<-21+2·11> =1

$$y[n] = \{x[2], x[3], x[0], x[1]\}$$

$$\begin{bmatrix} x[0]\\x[1]\\x[2]\\x[3] \end{bmatrix} \rightarrow \begin{bmatrix} x[3]\\x[0]\\x[1]\\x[2] \end{bmatrix} \rightarrow \begin{bmatrix} x[2]\\x[3]\\x[0]\\x[1] \end{bmatrix} \rightarrow \begin{bmatrix} x[1]\\x[2]\\x[3]\\x[0] \end{bmatrix}$$

$$F_{4}\begin{bmatrix}x[1]\\0\\x[3]\\0\end{bmatrix} = \begin{bmatrix}F_{2}\\F_{2}\end{bmatrix} \cdot \begin{bmatrix}x[1]\\x[3]\end{bmatrix} = DFT_{4}\{s[n]\} = S[K]$$
We want

$$F_{4}\begin{bmatrix} 0\\x[1]\\0\\x[3]\end{bmatrix} = DFT_{4} \{s[< n - n_{0} >_{4}]\}; n_{0} = 1$$

$$F_{4}\begin{bmatrix}0\\x[1]\\0\\x[3]\end{bmatrix} = \mathbf{W}_{4}^{Kn_{0}} \cdot \mathbf{S}[k]; k \in \mathbb{Z}_{4}$$

If
$$n_0 = 1$$

$$w_4^{kn_0} = \begin{bmatrix} 1 \\ w_4 \\ w_4^2 \\ w_4^3 \end{bmatrix}$$

$$F_{4}\begin{bmatrix}0\\x[1]\\0\\x[3]\end{bmatrix} = \begin{bmatrix}1\\w_{4}\\w_{4}^{2}\\w_{4}^{3}\end{bmatrix} \bullet \begin{bmatrix}\mathbf{S}[0]\\\mathbf{S}[1]\\\mathbf{S}[2]\\\mathbf{S}[3]\end{bmatrix}$$

$$\therefore F_4 \begin{bmatrix} 0\\x[1]\\0\\x[3] \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0\\0 & w_4 & 0 & 0\\0 & 0 & w_4^2 & 0\\0 & 0 & 0 & w_4^3 \end{bmatrix} \bullet \begin{bmatrix} F_2\\F_2 \end{bmatrix} \begin{bmatrix} x[1]\\x[3] \end{bmatrix}$$

Remember:

$$F_{2} = \begin{bmatrix} 1 & 1 \\ 1 & w_{4}^{2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & w_{2} \end{bmatrix}$$
$$\begin{bmatrix} 1 & 1 \\ w_{4} & w_{4}^{3} \\ w_{4}^{2} & w_{4}^{2} \\ w_{4}^{3} & w_{4} \end{bmatrix} \begin{bmatrix} x[1] \\ x[3] \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & w_{4} & 0 & 0 \\ 0 & 0 & w_{4}^{2} & 0 \\ 0 & 0 & 0 & w_{4}^{3} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & w_{4}^{2} \\ 1 & 1 \\ 1 & w_{4}^{2} \end{bmatrix} \begin{bmatrix} x[1] \\ x[3] \end{bmatrix}$$

$$F_4 x = F_4 x_e + F_4 x_0$$

3 TMS320C6713 DSP DEVELOPMENT SYSTEM

The possibilities to develop an application on the DSP C6713 are varied. There are different compiling high-level languages to DSP's. The tools used to compile and download programs to the DSP are MATLAB[®], Labview, Visual Basic and Visual C++. Those tools are interfaced with the DSP using RTDX (Real Time Data Exchange).

3.1 Code Composer Studio IDE (CCS)

This is an Integrated Development Environment from Texas Instruments used to build and debug applications developed in C or Assembly languages (see *Figure 1*). Some of the special features of this environment are the possibility of reviewing variables or registers from the DSK and also it is useful for exchange data between the board and other programming languages such us Labview and MATLAB[®].

CCS is used to calculate the quantity of floating point operations executed during any process in order to evaluate the algorithm implementation performance. CCS IDE can be used for reviewing results of an implementation due to the possibility of checking memory map.

🥐 /C6713 DSK/CPU_1 - C671x - Code Composer Studio	
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help	
首は日本目に、2011年1月1日日日日日日日日日日日日日日日日日日日日日日日日日日日日日日日日日	• •
Projects	
GEL StartUp Complete.	
GEL Output	- - -
LE For Help, press F1	1

Figure 1: Sample of Programming Environment: "Code Composer Studio".

3.2 CCS Installation and Support

The development environment is provided by Texas Instruments with the DSK board. Insert the installation CD into the CD-ROM drive with the board disconnected. The CD is labeled as "Code Composer Studio TM IDE Platinium v3.3". It is not required to connect the card using the USB port at the time of installation.

In the Texas Instruments web page (<u>www.ti.com</u>) it is possible to access technical documentation, download libraries, discussion groups and technical conferences.

<image>

 Code Composer Studio v3.3 Setup

 Welcome to the Code Composers

 Studio v3.3 Installation Wizard

A storage to the code of t

The following figure is the first window that appears when you insert the installation CD.

Figure 2: Code Composer Studio (CCS) v3.3 Installation Wizard

In this windows click "Next" to proceed with the installation.

CCS installation wizard will check your system in order to verify that it has the minimum requrements for installation.

뤻 Code Composer Studio v3.3 Setup	
System Requirements The following minimum system configuration is required	Ecomposer studio
Operating System OK - Microsoft Windows 2000 (or higher) Detected	
OK - Internet Explorer	
Memory OK - 3326 MB Detected	
Display Resolution OK - 1600 x 1200, 32 Bit Detected	
	<pre> Back Next > Cancel</pre>

Figure 3: CCS System Requirements Verification

Note: If your system does not meet the minimum requirements the software may not function.

Click "Next" if your system has all the minimum requirements.

In order to complete the installation process it is needed to accept the license agreement. Select the option "*I accept the License Agreement*", then select the button "*Next*" (see *Figure 4*).

🔂 Code Composer Studio v3.3 Setup		
License Agreement You must agree with the license displayed below to proceed	Ecomposer studio	
Code Composer Studio 3.3 Software License	Agreement	
Important - Please read the following license agreement care agreement. After you read this license agreement, you will be ask the terms of this license agreement. Do not click "I accept the unless: (1) you are authorized to accept and agree to the terms of yourself and your company; and (2) you intend to enter into and to b binding agreement on behalf of yourself and your company.	fully. This is a legally binding ed whether you accept and agree to e terms of the license agreement" this license agreement on behalf of e bound by the terms of this legally	
Important - Read carefully: This Code Composer Studio 3.3 ("CCS") Software License Agreement ("Agreement") is a legal agreement between you (either an individual or entity) and Texas Instruments Incorporated ("TI"). The software programs included herein (the "Licensed CCS Programs") consist of the following materials: (1) the TI proprietary materials (the "Proprietary Programs"), which are subject to the licensing terms set forth below, (2) the GNU materials, which are subject to the terms set forth below, (2) the GNU materials, which is included with those materials, (3) the Xerces materials, which are subject to the terms set forth in the in the Apache Software License, Version 1.0, a copy of which is included with those materials, (4) the Xalan materials, which are subject \checkmark		
 I accept the License Agreement I do not accept the License Agreement 	nt	
Print < Back	Next > Cancel	

Figure 4: CCS License Agreement

The CCS has three types of installation: *Typical Install, Debugger-Only Install and Custom Install.*

The *Typical Install* is the recommended installation for users without experience. In this type of installation the most common application features will be installed. Select "*Typical Install*" and then select "**Next**".



Figure 5: CCS Instalation Type Selection

The installation creates a folder with the name C:\CCStudio_v3.3\ by default. The CCS icon should be on the desktop and it is called *CCStudio v3.3* by default.

🐻 Code Composer Studio v3.3 Setup	
Destination Folder Select the folder to install your application into	Ecomposer [®]
Code Composer Studio v3.3 will be installed in the following folder. To inst the Browse button, and select another folder.	tall into a different folder, click
You can choose not to install Code Composer Studio v3.3 by clicking Can Wizard.	icel to exit the Installation
C Destination Folder	
C:\CCStudio_v3.3	Browse
Size	
Disk Space Required: 987MB	
(Click the "Disk Cost" button to see all available volumes and s	pace requirements/
Disk Cost Cack	Next > Cancel

Figure 6: CCS Destination Folder

Once the folder is created the program is ready to be install. Click "**Install Now**" to proceed with the installation.

ট Code Composer Studio v3.3 Setup			
	Ready to Install 0 Studio v3.3	Code Composer	
	Are you ready to have the Code Composer Studio v3.3 Installation Wizard begin the install?		
	Product:	Code Composer Studio v3.3	
	Install Mode:	Typical	
	Disk Space Required:	987MB	
	Destination Folder:	C:\CCStudio_v3.3	
TEXAS INSTRUMENTS	Once you have verified the install button to install Code Composer S	ation options, click the "Install Now" tudio v3.3.	
	< Back	Install Now Cancel	

Figure 7: Code Composer Studio v3.3 Installation

This window is presenting the installation progress.



Figure 8: CCS Installation Progress

Once the installation is over, click "Finish" to complete the installation procedure.



Figure 9: Finished CCS Installation

Code Composer Studio v3.3 Platinum installs all the drivers needed to work in the simulation stage, but does not have the drivers needed to complete the emulation stage. After installing the program Code Composer Studio v3.3 Platinum, proceed to install the drivers CD labeled as "*Code Composer Studio 2.x/3.x Driver Disk for Digital Spectrum Emulators*", that allow the users to complete the emulation stage. This software is included in the SPI525 PCI JTAG Emulator package.

Note: *Code Composer Studio 2.x/3.x Driver Disk for Digital Spectrum Emulators* only will be installed if the user wants to do implementations using the TMS320C6713 DSP (emulation stage).

Insert the CD labeled as "*Code Composer Studio 2.x/3.x Driver Disk for Digital Spectrum Emulators*". On the first window select "*CCS V3.1x PRODUCTS*" (see *Figure 10*).



Figure 10: CCS Emulation Drivers Main Menu Window

In the following window select "*CCS 3.1 Platinum Drivers*" in order to begin the installation process.



Figure 11: CCS 3.1 Planinum Driver

The InstallShield Wizard window for SD CCS 3.1 Emulation Drivers appears to continue the installation. Click "**Next**" to proceed.



Figure 12: CCS 3.1 Emulation Drivers Installation Window

The *Typical* option is the recommended installation for users without experience. In this type of installation the most common application features will be installed. Select "*Typical*" and then select "**Next**".

SD CCS 3.1 Em	ulation Drivers - InstallShield Wizard	×
Setup Type Select the setup	p type to install.	
Click the type o	f setup you prefer, then click Next.	
Typical	Program will be installed with the most common options. Recommended for most users.	
C Compact	Program will be installed with minimum required options.	
C Custom	You may select the options you want to install. Recommended for advanced users.	
InstallShield		
	< Back Next > Cancel	

Figure 13: CCS Emulation Drivers Setup Type Selection

Change *"Destination Folder"* from the direction *C:\CCStudio_v3.1* to *C:\CCStudio_v3.3*. To change the folder click "Browse..." and select the folder located at *C:\CCStudio_v3.3*.

SD CCS 3.1 Emulation Drivers - InstallShield Wizard	X
Choose Destination Location Select folder where setup will install files.	
Setup will install SD CCS 3.1 Emulation Drivers in the following folder.	
To install to this folder, click Next. To install to a different folder, click Browse and select another folder.	
Destination Folder C:\CCStudio_v3.3	
InstallShield]
< Back Next > Cancel	

Figure 14: Selection of Destination Location for CCS 3.1 Emulation Drivers

In the following window you have the opportunity to go back and verify all the previous settings. If you are satisfied with the settings click "**Next**" to begin the installation.

SD CCS 3.1 Emulation Drivers - InstallShield Wizard	
Start Copying Files Review settings before copying files.	1
Setup has enough information to start copying the program files. If you want to review or change any settings, click Back. If you are satisfied with the settings, click Next to begin copying files.	
Current Settings:	
Installing Typical Feature Set	
InstallShield Cance	el

Figure 15: CCS 3.1 Emulation Drivers Installation Progress

Figure 16 presents the software installation progress window.



Figure 16: CCS 3.1 Emulation Drivers Installation Progress

In order to finish the CCS Emulation Drivers installation, click "Finish".



Figure 17: CCS Emulation Drivers Installation Closure

Once the installation process is finished, select *Main Menu* in the Emulation Drivers window (see *Figure 18*).



Figure 18: CCS 3.1 Planinum Driver

Select "*Exit*" to finish the emulation drivers installation procedure.



Figure 19: CCS Emulation Drivers Main Menu Window

3.3 CCS Setup and Initialization

- To setup the Code Composer Studio V3.3 environment in a Windows XP system, click on Start and select All Programs → Texas Instruments → Code Composer Studio 3.3 → Setup CCStudio v3.3 (see Figure 20).
- If the Setup CCStudio v3.3 icon is located at the Desktop, this application can also be accessed by double clicking on this icon.



Figure 20: Location of CCS in Windows XP

The following setup window for CCS will appear:

Code Composer Studio Setup						
File Edit View Help						
System Configuration	Available Factory Boards	Family	Platform	Endianness	My System	
		All		All		
My System	ARM11 - VPOM2420 Platform Simu.	ARM11	simulator	*		
	ARM11 - VPOM2430 Platform Simu.	. ARM11	simulator	* =		
	ARM7 - VPOM2420 Platform Simul	ARM7	simulator	*		
	RRM7 Simulator, Big Endian	ARM7	simulator	big 📃		
	ARM7 Simulator, Little Endian	ARM7	simulator	little		
	ARM7 XDS510 Emulator	ARM7	×ds510	*		
	RRM7 XDS560 Emulator	ARM7	×ds560	*		
	RRM9 XDS510 Emulator	ARM9	×ds510	*		
	RRM9 XDS560 Emulator	ARM9	×ds560	*		
	ARM926EJ-S Simulator Little Endian	ARM9	simulator	little		
	F240 XDS510 Emulator	C24xx	×ds510	*		
	F240 XDS560 Emulator	C24xx	xds560	*		
	F2401 XDS510 Emulator	C24xx	×ds510	*		
	F2401 XDS560 Emulator	C24xx	×ds560	*		
	F2402 XDS510 Emulator	C24xx	×ds510	*		
	F2402 XDS560 Emulator	C24xx	xds560	*		
	F2403 XDS510 Emulator	C24xx	xds510	*		
	F2403 XDS560 Emulator	C24xx	xds560	*		
	F2406 XDS510 Emulator	C24xx	×ds510	*		
	F2406 XDS560 Emulator	C24xx	×ds560	*		
	F2407 XDS510 Emulator	C24xx	×ds510	*		
	F2407 XDS560 Emulator	C24xx	xds560	*		
	F241 XD5510 Emulator	C24xx	xds510	*		
	E241 XDS560 Emulator	C24xx	xds560	*		
	E243 XDS510 Emulator	C24xx	xds510	*		
	E243 XDS560 Emulator	C24xx	xds560	*		
	C27xx Cycle Accurate Simulator	C27xx	simulator	*		
	C27xx XDS510 Emulator	C27xx	vds510	*		
	C27xx XDS510 Emulator	C27vv	xds560	*		
	E2810 Device Simulator	C28vv	cipulator	*		
	E2810 YDS510 Emulator	C2077	vdc510	*		
		C20XX	X05310	·		
	🔤 Factory Boards 🔤 Custom Boa	rds 🐘 C	reate Board			>
Save & Quit Remove Re	emove All << Add << Add Multiple				Modify Properties	
					J	
ag a device driver to the left to add a b	oard to the system.					

Figure 21: Code Composer Studio Setup

- Here, the programming environment must be selected by the user: simulation or emulation.
 - **Simulation** implies that the application program developed can be compiled and executed, without physically connecting the target board to the computer.
 - **Emulation** implies that the target must be connected to the computer in order to compile and execute the application program.

3.3.1 Selecting Simulation Environment

- As mentioned previously, simulation implies that the application program developed can be compiled and executed, without physically connecting the target board to the computer.
- To conduct a simulation analysis, the user must access the Setup Code Composer Studio v3.3 tool, and follow these subsequent steps:
 - Next to Available Factory Boards, under Family, select the option C67xx.
 - Under **Platform**, select **simulator**.
 - Under Endianness, select little.
 - Under Available Factory Boards, a list of possible simulators should appear. Here, C6713 Device Cycle Accurate Simulator should be selected, by a single click, then pressing the Add button, located at the middle bottom (see the bottom figure). The simulator can also be selected by double clicking on the simulator board.
 - Next, **press Save & Quit**. Note: if there are any other boards **under System Configuration**, proceed to remove them. This is done by selecting each board and hitting the delete key. Only the **C6713 Device Simulator** must be selected.
 - A prompt window will appear, asking the user if he/she wishes to save the changes made to system configuration. The button **Yes** should be selected.
 - A second prompt window will appear, asking the user if Code Composer Studio should start on exit. The user should press **Yes**.

🐬 Code Composer Studio Setup			
File Edit View Help			
System Configuration	Current Proccesor Type	Driver Location	TMS320C6713
System Configuration My System C6713 Device Cycle Accurate Simulator MS320C6713	Current Procesor Type	Driver Location C:\CCStudio_v3.3\drivers\tisimC6xxx.dvr	Device Type: CPU Summary: Tips: - Use init6713sim.gel - Select Functional Simulator for faster simulation speeds. - Turning off resource conflict detection improves simulation speed. - Reserved Memory access detection can be turned off, if you are sure of memory usage, as, switching it off makes simulation speed faster Summary: Supports L1D, L1P, L2 Cache, EDMA, QDMA, Timer(2), EMIF supports interfacing with Async and SDRAM Memory Models, McBSP(2), McASP(2), Interrupt Selector. Does not support HP
	Factory Boards	Custom Boards 🦛 Create Board	IIC. GEL File: C:\CCStudio_v3.3\cc\gel\init6713sim.ge Master/Slave: N/A Device: C6713 Simulator Type: Cycle Accurate
Save & Quit Remove All	<< Add << A	Add Multiple	Modify Properties
elect the system node to add a new board to the :	system configuration.		1

Figure 22: Selecting Simulation Environment

3.3.2 Selecting Emulation Environment

If the user desires to work in the emulation environment, the DSP board should be connected to the PC or work station at this point. First, the power supply should be connected to the board through the power jack. Next, the DSP board should be connected to the PC or work station via the USB port (see *Figure 24*).



Figure 23: TMS320C6713 "Digital Starter Kit" (DSK)



To set up the emulation environment, the user should access the Setup Code Composer Studio v3.3 tool by going to *All Programs* \rightarrow *Texas Instruments* \rightarrow *Code Composer Studio* 3.3 \rightarrow *Setup CCStudio* v3.3 and follow these subsequent steps:

- Next to Available Factory Boards, under Family, select the option C67xx.
- Under **Platform**, select **dsk**.
- Under Endianness, select little.
- Under Available Factory Boards, the option C6713 DSK-USB and/or C6713 DSK should appear.
- Here, C6713 DSK-USB or C6713 DSK should be selected, by a single click, then pressing the Add button, located at the middle bottom. The emulator can also be selected by double clicking on the emulator board.
- Next, press Save & Quit. Note: if there are any other boards under System Configuration, proceed to remove them. This is done by selecting each board and hitting the delete key. Only the C6713 DSK-USB or the C6713 DSK must be selected.
- A prompt window will appear, asking the user if Code Composer Studio should start on exit. The user should press Yes.

St Code Company Studio Satur			
File Edit View Help			
System Configuration	Current Proccesor Type	Driver Location	CPU_1
My System C6713 DSK C713 Market C6713 NSK C713 Market C6713 NSK C714 Market C714	TM5320C6×1×	C:\CCStudio_v3.3\drivers\sdgo6713dsk.dvr	Device Type: CPU GEL File: C:\CCStudio_v3.3\cc\gel\DSK6713.gel Master/Slave: N/A
Save & Quit Remove Remove All	tem configuration	dd Multiple	Modify Properties

Figure 25: Emulation Environment Selection

Previous to start CCS operation the board should be connected to the power and also the PC by USB connection.

- Once CCS is launched, go *Debug* \rightarrow *Connect*, in order to establish connection with the board.



Figure 26: Establish the Connection between the CCS and the TMS320C6713 DSP

3.4 General Algorithm Implementation on the Board

The process in an algorithm implementation on the board is:

1. Create a project, add it the C or assembly programs and the libraries nedded for the program.

- 2. Build your project → 🛗
- 3. Download the project to the board.
- 4. Run the project $\rightarrow \overset{\sim}{
 ightarrow}$
- 5. Evaluate results and correct errors.
- 6. In case of errors in the results return to the step two.

3.4.1 Types of Useful Files

Each program that is constructed using "Code Composer Studio" will be working with a number of files with different extensions:

- Namefile.pjt: to create and build a project.
- Namefile.c: C source program created by the user. There could be one or more depending on the application.
- Namefile.asm: Assembly source program created by the user. There could be one or more depending on the application.
- Namefile.h: Header support file.
- Namefile.lib: Library file.
- Namefile.cmd: Linker command file that maps sections to memory in the DSP.
- Namefile.obj: Files created after compiling the project.
- Namefile.out: Executable file created by the linker to be loaded on the processor.
- Namefile.cdb: Configuration file when using DSP/BIOS.

3.4.2 DSK Support Tools

The following support files are frequently used when a project is created:

- **C6713dskinit.c:** Includes functions for initializing the DSK, the codecs for the serial ports and the I/O of the target board.
- **C6713dskinit.h:** Provides description of the functions used to initialize target board.
- **C6713dsk.cmd:** File used for the memory organization and distribution of the DSP.
- Vectors_intr.asm: Assembly source file used for managing interrupts.
- **Vectors_poll.asm:** Assembly source file used for managing access to ports through "polling".
- **rts6700.lib: dsk6713bsl.lib; csl6713.lib; rtdx.lib:** Support libraries needed for the DSP target board and data interchange in "real-time".

3.5 Programming Examples to test the DSK Tools

The following program example illustrates the features of the CCS and the DSK board. This example shows step by step how to create a project to compile and download to the DSK TMS320C6713. Be sure to place the files included with this guide in C:\CCStudio_v3.3\MyProjects, before starting the examples.

3.5.1 Example 1. Hello World!

AIM:

This example helps us begin to understand the functionality of the CCS and the TMS320C6713 DSP.

EQUIPMENT:

PC	- Windows XP Operating System
Software	- CCStudio V3.3
Hardware	- TMS320C6713 DSP

PROGRAM:

Creating the Project:

In this section is shown how to create a project, adding the necessary files to build a project using "Code Composer Studio".

1. Select *Project* → *New*. In the filename, type the name "hello" of the new project and click "Save".

This project file (.pjt) is saved in the folder "hello" (within C:\CCStudio_v3.3\MyProjects\hello). *Figure 27* shows how create a new project and in the *Figure 28* the project view files.

Project Creation	on 🔀
Project Name:	hello
Location:	C:\CCStudio_v3.3\MyProjects\hello\
Project Type:	Executable (.out)
Target	TMS320C67XX
	Finish Cancel Help

Figure 27: Window for the creation of a New Project

🦑 /C6713 DSK/CPU_1 - C671x - Code	Composer Studio	×
File Edit View Project Debug GEL Optic	on Profile Tools DSP/BIOS Window Help	
🏙 📽 🖬 👗 🖪 💼 👘 🗠 👘	- 新路路路路路 - ● 😢 🦛 # 軒軒 軒軒 巨 亜 1 ◆ 36 36 👟 💣	
hello.pit 💽 Debug		
🔊 🚳 🗅 🗐 🛤 🗏 🔤 🖻 🗖		
Image: Second system Image: Second system Image: Second		
GEL StartUp Complete.		
GEL Output		-
MALTED	LE	

Figure 28: Project Folders

Select File → New → Source File, copy the following source code (.C), click File → Save As and save the file as "hello.c" in the following path C:\CCStudio_v3.3\MyProjects\hello.

<u>C source code:</u>

```
#include <std.h>
// ====== main =======
void main()
{
    puts("hello world!\n");
    return; }
```

- 3. Select *Project* → *Add files to project*. Add the file "hello.c" created in the previous step.
- 4. Copy and Paste the file *vectors_poll.asm*, located in the path C:\CCStudio_v3.3\MyProjects\Support_files_6713, to the folder "hello". Repeat step 3 to add to the project the ".asm" source file *vectors_poll.asm*. Repeat again and select files ".cmd", *C6713dsk.cmd* to add to the project.
- 5. Similarly as the previous step the following ".lib" files should be added: *rts6700.lib*, *dsk6713bsl.lib* and with the chip support library file *csl6713.lib*.
- 6. Select *Project* \rightarrow *Scan All Files Dependencies*. Verify that all the files that are shown in the *Figure 29* were added to the project.


Figure 29: Project Files

- 7. Once all of the files are added to the project, the project must be built. This is done by going to *Project* → *Build Options*. This option is used to properly set up the compiler and linker, based on the characteristics of the TMS320C6713 DSP board. Several settings should to be chosen or written, and the option OK is selected after all settings are verified.
- 8. Under *Compiler* → *Category* → *Basic*
 - The target version: *C671x* (*-mv6710*) should be highlighted

Build Options for hello.pjt (Debug) 🛛 🛛 🛛 🖓 🔀							
General Compile	General Compiler Linker DspBiosBuilder Link Order						
-g -fr''\$(Proi_dir)'	-g -fr''\$(Proi_dir)\Debug'' -d''_DEBUG'' -mv6710						
Category: Basic Advanced Feedback Files Assembly Parser Preprocessor Diagnostics	Basic Target Version: Generate Debug Info: Opt Speed vs Size: Opt Level: Program Level Opt.:	C671× (-mv6710) ▼ Full Symbolic Debug (-g) ▼ Speed Most Critical (no -ms) ▼ None ▼ None	•				

Figure 30: Setting the Target Version

- 9. Under *Compiler* → *Preprocessor*:
 - In *Pre-Define Symbol*, the following should be written: *CHIP_6713*. This specifies the DSP chip that the target board utilizes.

Build Options f	or hello.pjt (Debug)			? 🔀
General Compile	^{er} Linker DspBiosBuild	er Link	Order	
-g -fr''\$(Proi_dir)\	Nebug" -d"_DEBUG" -m∖	6710		~
Category:	Preprocessor			
Advanced Feedback Files Assembly Parser Preprocessor Diagnostics	Pre-Define Symbol (-d): Undefine Symbol (-u): Preprocessing:	CHIP_6	713 5a)	•
			Cancel	Help

Figure 31: Specifying the Chip Architecture

10. Under *Linker* → *Libraries*:

- In *Included Libraries* (-1), these libraries must be specified: rts6700.lib; dsk6713bsl.lib; csl6713.lib

Build Options for hello.pjt (Debug)	?×
General Compiler Linker DspBiosBuilder Link Order	
-c -m".\Debug\hello.map" -o".\Debug\hello.out" -w -x	~
Category: Libraries Basic Image: Exhaustively Read Libraries (-x) Search Path (-i): Incl. Libraries (-l): Incl. Libraries (-l): rts6700.lib; dsk6713bsl.lib; csl6713.lib	
OK Cancel H	elp

Figure 32: Libraries Nedded for the Project

11. Now the user may click OK once all the previous building option settings have been established.

Compiling and Debugging the Project

In this step the C compilation and linker to build a project.

1. Click on the "**rebuild all**" button that is in the upper part of the CCS environment and verifies that you have 0 errors.



Figure 33: Compiling Results

Note: If there are errors in your code, they will be listed with the corresponding line numbers. Correct them and rebuild your project.

- 2. Select *File* → *Load Program*. Choose the file "hello.out" that is located in the following path: C:\CCStudio_v3.3\MyProjects\hello\Debug.
- 3. Click on the "run" button *that* is located on the left side of the CCS environment.

Results Obtained:

On the "Stdout" a message "hello world!" is printed and then the program is finalized.

hello world!
GEL Output Build Stdout
🧏 💿 HALTED: s/w breakpoint

Figure 34: Results Obtained after Run the Algorithm "hello world"

3.5.2 Example 2. Fast Fourier Transform (FFT) -- (*Created Project Version*) Code Developed by Rulph Chassaing[1]

AIM:

FFT algorithm takes a given input signal and returns its Fourier transform.

EQUIPMENT:

PC	- Windows XP Operating System
Software	- CCStudio V3.3 Platinium
Hardware	- TMS320C6713 DSP

Figure 35 is presenting the files needed for the creation of the FFT project. The folder is located at C:\CCStudio_v3.3\MyProjects\FFTproject_files.

🗁 FFTproject_files		
File Edit View Favorites Tools Help		
G Back 🝷 🕥 🕤 🏂 🔎 Search 🎼 Fol	lders 🕼 🎲 🗙	9
Address 🛅 C:\CCStudio_v3.3\MyProjects\FFTproject_fi	les	
Name 🔺	Size	Туре
C6713dsk.cmd	1 KB	Windows NT Comm
🖺 c6713dskinit.c	5 KB	C File
🖺 C6713dskinit.h	9 KB	H File
📼 csl6713.lib	111 KB	LIB File
🖺 dsk6713.h	2 KB	H File
🖺 dsk6713_aic23.h	9 KB	H File
👼 dsk6713bsl.lib	21 KB	LIB File
🖺 FFTproject.c	6 KB	C File
📼 rts6700.lib	336 KB	LIB File

Figure 35: FFT Files

This section show how to open a project using "Code Composer Studio".

1. Click *Project* → *Open*. Look and click on the file *FFTproject.pjt* in the following path: C:\CCStudio_v3.3\MyProjects\FFTproject.

File	Edit	View	Project	Debug	GEL	Option	Profile	Tools	DSP/BIOS	Window	Help		
睝	2		New.								2	in 741	.
, R	60	Ö	Use E Export Add F	xternal M rt to Mak Files to Pr	1akefil efile oject.	e		1	₩ ₩ ;	<u> </u>	2	• 4	<u>2</u>
{ ¹ }	0	Files	Save Close					bly					×
ው ው ው ው ው ው ው ው ው ው ው ው ው ው ው ው ው ው ው		GE	Source Comp Build Rebu Stop Build Confi Build File S	e Contro ile File Build Clean gurations Options, pecific Op	;		Ctrl F7 F7	- 4BC0 4BC0 4BC0 4BC0 4BD0 4BD0 4BD0 4BD0 4BD0 4BC0 4BF4	92108 92305 91A5E 91800 000000 402305 402305 402305 400004 00555 001100 400006	3940 54F4 2A2A 106A 1000 52E4 1000 2E92 1264 5000	[[[A1] A1] A1] A1]	
≫ ≫ ∞ ⊡ ₽ ₽ ₽	-		Proje Show Show Scan Rece i	ct Depen Project I File Dep All File D ht Projec	dencie Depen enden epende t Files	is dencies cies encies		J					

Figure 36: Open FFT Project

Project Ope	n		? 🗙
Look in: 隘	FFTproject		* 💷 •
C Debug	:.C5_ :.pjt		
File name:	FF I project.pjt		Upen
Files of type:	Project Files (*.pjt)	▼	Cancel
			Help

Figure 37: FFT Project Selection

 Double click on "*FFTproject.pjt*" on the left side of CCS and click on Source to see the files. Then double click on "*FFTproject.c*". Your environment should looks like *Figure 38* and should have all the files that are on the left side.

Note: Verify that the options in *Project* \longrightarrow *Build Options*, are correct (Steps from 7 to 11, Example 1: hello world)



Figure 38: CCS Environment for FFT Example

Compiling and Debugging the Project

Click on the "rebuild all" button that is in the upper part of the CCS environment and review that you have 0 errors.

Build Complete, O Errors, 2 Warnings,	O Remarks.		
GEL Output Build			
MALTED		LE	Symbol definition not found.
			

Figure 39: FFT Project Compiling Results

- **Note**: If there are errors in your code, they will be listed with the corresponding line numbers. Correct them and rebuild your project.
 - 3. Select *File* → *Load Program*. Choose the file "*FFTproject.out*" that is located in the following path: C:\CCStudio_v3.3\MyProjects\FFTproject\Debug.



Figure 40: "Load Program" Location



Figure 41: "FFTproject.out" File Location



Figure 42: Downloading the "FFTproject.out" File to the TMS320C6713 DSP

4. Click on the "run" button *is* that is located on the left side of the CCS environment.

Results Obtained:

On the "Stdout" are printed messages of the program process until the execution is done. In the Debug folder three data files are generated: *input_signal_real_DSP256pts.txt*, *input_signal_imag_DSP256pts.txt* and *transform_DSP256pts.txt*.

- input_signal_real_DSP256pts.txt This file contains the real part of the input signal.
- input_signal_imag_DSP256pts.txt This file contains the imaginary part of the input signal.
- transform_DSP256pts.txt This file contains the Fourier transform of a given input signal.



Figure 43: Results Obtained after Run the FFT Algorithm.

3.5.3 Example 3. Fast Fourier Transform (FFT) -- (*Creating the Project Version*) Code Developed by Rulph Chassaing

AIM:

FFT algorithm takes a given input signal and returns its Fourier transform.

EQUIPMENT:

PC	- Windows XP Operating System
Software	- CCStudio V3.3 Platinium
Hardware	- TMS320C6713 DSP

Figure 44 is presenting the files needed for the creation of the FFT project. The folder is located at C:\CCStudio_v3.3\MyProjects\FFTproject_files.



Figure 44: FFT Project Files

Creating the Project:

This section show how to create a project, adding the necessary files to build a project using "Code Composer Studio".

1. Select *Project* → *New*. In the filename, type the name "**FFTproject**" of the new project and click "**Save**".

This project file (.pjt) is saved in the folder "**FFTproject**" (within **C:\CCStudio_v3.3\MyProjects\FFTproject**). *Figure 46* shows how create a new project and in the *Figure 47* presents where the folder is created.

🦉 /C6713	DSK	JCPU_1	- C671	x - (Code Co	mposer	Studio)		
File Edit V	View	Project	Debug	GEL	Option	Profile	Tools	DSP/BIOS	Window	He
🏠 🚅 🛯	II.	New.								1
	_	Open								
		Use E	xternal N	1akefil	e		1	🏙 🛗 🖥	🖌 🕀	
,		Expor	rt to Maki	efile						
👳 60°	8	Add F	iles to Pr	oject.						
		Save								
🕑 👰 F	iles	Close					ibly			
₽	📕 GE 📄 Pr	Sourc	e Contro	I		•	DAD 4	0010A	A7A	
{}+		Comp	ilo Eilo			CHLE7	-DAD8	20000	1590	
7h		Duild	uic i lic			cum 7 57	DADC	00008	000	
111		- Dulla				F7	DAEO	00000	1790	
0 ⁴		Rebu	IIA All				DAE 4	02000	IOA8	
		Stop	Build				DAE 8	00006	000	
⇒α		Build	Clean				DAEC	02306	2E6	
		Confi	gurations				DAFO	00006	000	
{ * }		Build	Options.,				DAF 4	02108	05A	

Figure 45: Creating a New Project

Verify if the following option is selected:

Target \rightarrow *TMS320C67XX*, and then click **Finish**.

Project Creation	m	X
Project Name:	FFTproject	
Location:	C:\CCStudio_v3.3\MyProjects\FFTproj]
Project Type:	Executable (.out)	_
Target	TMS320C67XX	-
		·
	Finish C	ancel Help

Figure 46: Window for the Creation of a New Project



Figure 47: FFT Project Folder

- 2. Copy the following files from C:\CCStudio_v3.3\MyProjects\FFTproject_files to C:\CCStudio_v3.3\MyProjects\FFTproject:
 - C6713dsk.cmd
 - C6713dskinit.c
 - C6713dskinit.h
 - FFTproject.c
 - csl6713.lib
 - dsk6713.h
 - dsk6713_aic23.h
 - dsk6713bsl.lib
 - rts6700.lib

FFTproject_files				
File Edit View Favorites Tools Help				
🚱 Back 🝷 💮 🚽 🏂 🔎 Search 🞼	Folders	B 🕉 🗙	₽	
Address 🛅 C:\CCStudio_v3.3\MyProjects\FFTproje	iect_files			
Name 🔺		Size	Туре	Dal
C6713dsk.cmd		1 KB	Windows NT Comm	2/2
🗐 c6713dskinit.c		5 KB	C File	2/2
📃 C6713dskinit.h		9 KB	H File	- 1/2
🔤 csl6713.lib		111 KB	LIB File	- 10/
📃 dsk6713.h		2 KB	H File	- 11/
📃 dsk6713_aic23.h		9 KB	H File	- 11/
🔤 dsk6713bsl.lib		21 KB	LIB File	- 12/
FFTproject.c		6 KB	C File	6/5,
🔤 rts6700.lib		336 KB	LIB File	- 10/

Figure 48: FFT Project Files

- 3. Select *Project* \rightarrow *Add files to project*. Add the following files:
 - C6713dsk.cmd
 - C6713dskinit.c
 - FFTproject.c
 - csl6713.lib
 - rts6700.lib
 - dsk6713bsl.lib

()	C6713 DSK/CPU_1 - C671	x - Code Composer Studio	
File	Edit View Project Debug	GEL Option Profile Tools DS	P/BIOS Win
1	🖻 🖬 X 🖻 💼 🗠	CM	•
FFT	project.pjt 💽 Debu	g 💽 💽	i 🛗 👗
X	66' 📋 📰 🔜 🖂	■ £	
{ }	Files	🗷 Disassembly	
(유		◆ 00004BF0 0 00004BF4 0)00C0362)0008000
(9	Dependent Pr	Add Files to Project	100C0362 120008C0
₿ ₽	DSP/BIOS Col	Export to Makefile)0006000)00000000
->{}		Save)00000000)00000000
(*) X	Source	Build	100000000
滏		Build (Selection only) Stop Build	
×	-	Clean Clean (Selection only)	
		Project Dependencies	

Figure 49: Adding Files to the Project

4. Select *Project* \longrightarrow *Scan All Files Dependencies*. Verify that all the files that are shown in the *Figure 50* were added to the project.



Figure 50: Project Files

5. Once all of the files are added to the project, the project must be built. This is done by going to *Project* → *Build Options*. This option is used to properly set up the compiler and linker, based on the characteristics of the TMS320C6713 DSP board. Several settings should to be chosen or written, and the option OK is selected after all settings are verified.

🦉 /C6	713 DSK/CP	U_1 - C671x - Code Com	poser Studio - [FFTproject.c]
🔶 File	: Edit View	Project Debug GEL Option	n Profile Tools DSP/BIOS Window Help
FFT pro	≩ 🛃 🐰 (oject.pit 661 🗂 🗐	New Open Use External Makefile Export to Makefile Add Files to Project	A`A`A`A`A_A`A`@ ?? \$\$ \$4 []
{ ? }	Piles	Save Close	roject.c FFT implementation calling a C-coded FFT f
8 8	⊕ 🔁 GEL fili ⊡ 🔁 Projec	Source Control	<pre>de "dsk6713_aic23.h" fs=DSK6713 AIC23 FREQ 8KHZ;</pre>
()		Compile File Build	CtrlF7 F7 de <stdio.h></stdio.h>
<u>0</u> ,		Rebuild All Stop Build	de <math.h> Iude <time.h></time.h></math.h>
70	±	Build Clean	e PTS 256
(•}		Configurations Build Options	e PI 3.14159265358979
Ä		File Specific Options	f struct {float real,imag;} COMPLEX; FT(COMPLEX *Y, int n, COMPLEX *w); //FFT pro
×		Project Dependencies Show Project Dependencies Show File Dependencies	le timeval_diff(struct timeval *a, struct timeval *, x[PTS]; //intermediate buffer i; //general purpose index va
õX.	🔛	Scan All File Dependencies Recent Project Files	<pre>K w[PIS]; //twiddle constants stored LEX samples[PTS]; //primary working buf. K data_in[PTS]; //input data</pre>
5 5			<pre>FILE *entrada_real; FILE *entrada_imag; FILE *trans;</pre>
			int main ()

Figure 51: Build Option Setting Location

- 6. Under *Compiler* → *Category* → *Basic* The target version: C671x (-mv6710) should be highlighted

Build Options f	or FFTproject.pjt	(Debug)	? ×
General Compile	er Linker DspBiosBu	iilder Link Order	1
-g -fr''\$(Proi_dir)'	.Debug" -d"_DEBUG" -	mv6710	<
Category: Basic Advanced Feedback Files Assembly Parser Preprocessor Diagnostics	Basic Target Version: Generate Debug Info: Opt Speed vs Size: Opt Level: Program Level Opt.:	C671x (-mv6710) Full Symbolic Debug (-g) ▼ Speed Most Critical (no -ms) ▼ None None	×

Figure 52: Setting the Target Version

7. Under *Compiler* \rightarrow *Category* \rightarrow *Advanced*:

- In *Memory Models* select *Far* (*-mem_model:data=far*).
- Verify that Endianness is selected to be *Little Endian*.

Build Options for FFTproject.pjt (Debug) 🛛 🔹 💽								
General Compiler Linker DspBiosBuilder Link Order								
-g -fr''\$(Proi_dir)	-g-fr''\$(Proj_dir)\Debug''-d''_DEBUG''-mv6710mem_model:data=far							
Category:	- Advanced							
Basic	RTS Modifications:	Defns No RTS Funcs	3 💌					
Advanced Feedback	Auto Inline Threshold (-o):						
Files	Endianness:	Little Endian 💌						
Parser	Memory Models:	Far (mem_model:dat	ta=far) 💌					
Diagnostics	RTS Calls:	Use Memory Model	-					
	Aliasing:	Default	•					
	🔲 Interrupt Threshold (-	mi):						
	🔲 🔲 Speculate Threshold	(-mh):	_					
	🔲 🔲 Turn Off Software Pij	pelining (-mu)						
	🔲 🔲 Old 6400 Alignment (Compatibility (-mb)						
Turn Off Reorder of Associative Floating Pt Ops (-mc)								
	Use Function Subsections (-mo)							
Historic C Pointer to Const Alias Disambiguation (-ox)								
-								
	OK	Cancel	Help					

Figure 53: Memory Model Type Selection

- 8. Under *Compiler* → *Category* → *Preprocessor*:
 - In *Pre-Define Symbol*, the following should be written: *CHIP_6713*. This specifies the DSP chip that the target board utilizes.

Build Options f	or FFTproject.pjt (Debug)	? ×			
General Compile -g -fr''\$(Proi_dir)	er Linker DspBiosBuilder Link Order NDebug''-d''CHIP_6713''-mv6710mem_model:data=far	< >			
Category: Preprocessor Basic Include Search Path (-i); Advanced Pre-Define Symbol (-d); Files Undefine Symbol (-d); Assembly Preprocessor Parser Preprocessor Diagnostics Continue with Compilation (-ppa)					
	OK Cancel	Help			

Figure 54: Specifying the Chip Architecture

- 9. Under *Linker* \rightarrow *Libraries*:
 - In *Included Libraries* (-l), these libraries must be specified: rts6700.lib; dsk6713bsl.lib; csl6713.lib

Build Options for FFTproject.pjt (Debug)	?×
General Compiler Linker DspBiosBuilder Link Order	
-c -m".\Debug\FFTproject.map" -o".\Debug\FFTproject.out" -w -x -l"its6700.lib" -l"dsk6713bsl.lib" -l"csl6713.lib"	<
Category: Libraries Basic ✓ Libraries Advanced Advanced Search Path (-i): Incl. Libraries (-l): Its6700.lib;dsk6713bsl.lib;csl6713.lib	
OK Cancel	Help

Figure 55: Libraries Needed for the Project

10. Now the user may click **OK** once all the previous building option settings have been established.

Compiling and Debugging the Project

Click on the "rebuild all" button that is in the upper part of the CCS environment and review that you have 0 errors.

Build Complete, O Errors, 2 Warnings, O	Remarks.		
GEL Output Build			
MALTED		LE	Symbol definition not found.

Figure 56: FFT Project Compiling Results

- **Note**: If there are errors in your code, they will be listed with the corresponding line numbers. Correct them and rebuild your project.
 - 1. Select *File* → *Load Program*. Choose the file "*FFTproject.out*" that is located in the following path: C:\CCStudio_v3.3\MyProjects\FFTproject\Debug.

۲	/C6713 DSK/CPU_1 - C67	71x - Code	e Comp	oser Stu	idio -	[FFTproje	ect.c *]
0	File Edit View Project De	ebug GEL	Option	Profile	Tools	DSP/BIOS	Window
FF	New Open Close Save Save Save As	Ctrl+O Ctrl+S		•	\$ 🖽	. ■ ≚	1 6 8 () ()
74	Save All		_	//FFT]	proj	ect.c I	FT ii
9 9 9 9	Load Program Reload Program Load Symbols Reload Symbols Unload Symbols	Ctrl+L Ctrl+Shift+	L •	#inclu Uint32 #inclu	ude 2 fs ude	"dsk67: =DSK67: <stdio< th=""><th>13_ai) 13_AI) .h></th></stdio<>	13_ai) 13_AI) .h>
ው ት የት	Load GEL Data Workspace		;	#fncf //#inc //#inc #defin #defin	<i>clude</i> <i>clude</i> ne P ne P	e <i>(time)</i> e <i>(sys)</i> TS 256 I 3.14:	(/ ///////////////////////////////////
Ż	Difference between files Merge Files		-1	typede void I	ef s FFT()	truct {	floa *Y,
浴祭	Print	Ctrl+P		∕∕doul float	ble x[P	<i>timeval</i> TS];	_dif.
「同日間」	Recent Source Files Recent Workspaces Recent Program Files Recent Symbols Recent GEL Files			short COMPLE COMPLE COMPLE FILE	i; EX w <i>FLEX</i> EX d *ent: *ont:	[PTS]; <i>sample</i> ata_in rada_re	<i>s/PT</i> [PTS] al;
§. ()	Launch Setup Exit		-	FILE *	*tra:	ns;	109,
				int ma {	ain	0	

Figure 57: "Load Program" Location

Load Progra	m		? 🗙
Look in: 🔎	Debug		¥ ∎
FFTproject	out		
File name:	FFT project.out		Open
Files of type:	*.out	•	Cancel
			Help

Figure 58: "FFTproject.out" File Location

💽 File E	dit View Project Debug GEL	Option Profile To	ols DSP/BIOS '	Window Help	
Loading	Program			8 8 % %	4
	Loading Program		± 🔏 -	0 🕺 🕒 🖉	4
	jects\FFTproject\Debug\FFT	project.out			
•••	Loading sections.		10802		ZF
			18000		NO
	text: 22752 of 50496 at 0v220	-	20028		NO MV
	.text. 32732 01 30436 at 0x220	Cance	80068		MV
P 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Documents DSP/BIOS Config Generated Files Include Soft Status discraiss discraiss.lib discraiss.lib	00000018 0000001C 0000024 0000028 0000028 0000028 0000028 0000034 0000034	COCC3666 000C3666 020C3664 00004000 30000612 0008000 028C3626 000029C2 20000192 3FFFFF12	[!B0] [B0] [!B0]	LD LD NC B. NC LD SU B.
	FFTproject.c	0000003C 00000040 00000044	008008C0 308C67A0 02903636 328F87A0	[!B0]	ZE AN ST

Figure 59: Downloading the "FFTproject.out" File to the TMS320C6713 DSP

2. Click on the "run" button *that* is located in the left side of the environment CCS.

Results Obtained:

On the "Stdout" are printed messages of the program process until the execution is done. In the Debug folder three data files are generated: *input_signal_real_DSP256pts.txt*, *input_signal_imag_DSP256pts.txt* and *transform_DSP256pts.txt*.

- input_signal_real_DSP256pts.txt This file contains the real part of the input signal.
- input_signal_imag_DSP256pts.txt This file contains the imaginary part of the input signal.
- transform_DSP256pts.txt This file contains the Fourier transform of a given input signal.

🦉 /C6	5713 DSK/CPU_1 - C671x - Cod	de Composer Studio - [Disassembly (abort)]	. 🗆 🗙
🔄 File	e Edit View Project Debug GEL	. Option Profile Tools DSP/BIOS Window Help	- 8 ×
1	3 ⊟ X № @ ∽ ~		/* %)
FFTpr	oject.pjt 🗾 Debug		
892			
(4)	Piles		<u>^</u>
{} }	GEL files Projects	0000C744 00000090 B.S1 0xC744 (PC+4 = 0	OxOOC
{}	🗄 🍰 FFTproject.pjt (Deb	0000C748 00008000 NOP 5	
()	Dependent Project	0000C74C 0000000 NOP	
<u>a</u>	Documents	0000C754 00000000 NOP	
	Generated Files	0000C758 0000000 NOP	
3 0	🕀 🧰 Include	00000760 cinit, etext, etext, cinit:	
(4)	E-C Libraries	00000C760 00000230 .word 0x00000230	
æ	dsk6713bsl.lib	0000C764 80001414 [A1] LDBU.D1T1 *A0[0],A0	
**	📩 rts6700.lib	00000768 0000000 NOP	_
*	🖻 🔄 Source	0000C770 0000000 NOP	
<u> </u>	FFTproject.c	0000C774 0000000 NOP	
õx	C6713dsk.cmd	UUUUC778 UUUUUUUU NOP 000000770 00000012 B S2 cipit (PC+0 -	0~00
		00000C780 00000000 NOP	UAUC
商		0000C784 0000001 NOP	
5		0000C788 00000000 NOP	
		0000C790 0000000 NOP	
		0000C794 0000000 NOP	
		00000C798 00000022 .word 0x0000022	
		0000C7A4 00000000 NOP	
		0000C7A8 0000000 NOP	
	<	00000C7B4 00000024 LDB.D1T1 *-A0[0],A0	
-		0000C7B8 0000002 NOP	~
	30		
done	1		<u> </u>
done)		
aone	1		
	► ► GEL Output \ Build \ Std		
📡 💿	HALTED: s/w breakpoint	LE For Help, press F1	//

Figure 60: Results Obtained after Run the FFT Algorithm.

3.5.4 Example 4. Corner Turning -- (*Created Project Version*) Code Developed by Abigail Fuentes and Inerys Otero.

AIM:

This example helps us begin to understand the functionality of the CCS and the TMS320C6713 DSP. The Corner Turning algorithm allows the users to obtain the transpose of an input matrix.

EQUIPMENT:

PC	- Windows XP Operating System
Software	- CCStudio V3.3 Platinium
Hardware	- TMS320C6713 DSP

Figure 61 is presenting the files needed for the creation of the Corner Turning project. The folder is located at C:\CCStudio_v3.3\MyProjects\Corner_Turning_files

Corner_Turning_files					
File Edit View Favorites Tools Help					
🚱 Back 🝷 🕥 🕤 🏂 🔎 Search 🎼 Fo	ders		×	4	•
Address 🛅 C:\CCStudio_v3.3\MyProjects\Corner_Turn	ing_file	s			
Name 🔺		:	5ize	Туре	
ata				File Fo	lder
C6713dsk.cmd		1	КB	Windo	ws NT Comm
🗐 c6713dskinit.c		5	5 KB	C File	
🗒 C6713dskinit.h 9 KB H File					
📃 cornerTurning.c		1	KB	C File	
📼 csl6713.lib		111	KB	LIB File	•
🗐 dsk6713.h		2	2 KB	H File	
🗐 dsk6713_aic23.h		ç	9 KB	H File	
🔟 dsk6713bsl.lib		21	KB	LIB File	•
📃 main.c		2	2 KB	C File	
📼 rts6700.lib		336	5 KB	LIB File	•

Figure 61: Corner Turning Files

This section show how to open a project using "Code Composer Studio".

1. Click *Project* → *Open*. Look and click on the file *Corner_Turning.pjt* in the following path: C:\CCStudio_v3.3\MyProjects\Corner_Turning.



Figure 62: Open "Corner Turning" Project

Project Ope	n		? 🗙
Look in: ଢ	Corner_Turning	- 🗢 🖻) 💣 🎟 -
Corner_Tu Debug	rning.CS_ rning.pjt		
File name:	Corner_Turning.pit		Open
Files of type:	Project Files (*.pjt)	•	Cancel
			Help

Figure 63: Corner_Turning Project Selection

2. Double click on "*Corner_Turning.pjt*" on the left side of CCS and click on **Source** to see the files. Then double click on "*Corner_Turning.c*". Your environment should look like *Figure 64* and should have all the files that are on the left side.

Note: Verify that the options in *Project* \rightarrow *Build Options*, are correct (Steps from 7 to 11)



Figure 64: CCS Environment for Corner Turning Example

Compiling and Debugging the Project

Click on the "rebuild all" button that is in the upper part of the CCS environment and review that you have 0 errors.

Build Complete, O Errors, 2 W	arning	s, O) Rema	arks.				
GEL Outp	ut 👌 Build ,							
🙀 💿 HALTED					LE	5	iymbol definition not found.	
			~	T	<i>a</i> 11			

Figure 65: Corner Turning Compiling Results

- **Note**: If there are errors in your code, they will be listed with the corresponding line numbers. Correct them and rebuild your project.
 - 3. Select *File* → *Load Program*. Choose the file "Corner_Turning.out" that is located in the following path: C:\CCStudio_v3.3\MyProjects\Corner_Turning\Debug.

	/C6713 DSK/CPU_1 - C	671x - Code	e Comp	ioser St	udio -	[ma
٠	File Edit View Project	Debug GEL	Option	Profile	Tools	DSF
1	New					
Ca	Open Close	Ctrl+O	F	•	۵ 🕸	i 📥
, E	Save Save As	Ctrl+S				
74	Save All		_			
Ð	Load Program Reload Program	Ctrl+L Ctrl+Shift+	L.	#incl Uint3	u <mark>de</mark> 12 fs	"ds =D\$
Ð	Load Symbols Reload Symbols			#incl #incl	ude ude	<si <1:</si
- (*] - 771	Unload Symbols		•	#incl #incl	ude ude	<ma< td=""></ma<>
	Load GEL Data			#incl	ude	< c1
-78 - da	Workspace			#defi	ne N	12
Ä	Difference between files Merge Files			float float	inp tra	ut) nsp
20	Print	Ctrl+P		FILE	* aen	era
1	Recept Source Files			void	COTD	er

Figure 66: "Load Program" Location

Load Progra	m		? 🔀
Look in: 🔎	Debug	- 🗧 🛍	
Corner_Tu	rning.out		
File name:	Corner_Turning.out		Open
Files of type:	*.out	•	Cancel
			Help

Figure 67: "Corner_Turning.out" File Location



Figure 68: Downloading the Corner_Turning.out File to the TMS320C6713 DSP

4. Click on the "run" button that is located in the left side of the environment CCS.

Results Obtained:

On the "Stdout" there are printed messages of the program process until the execution is done. In the Debug folder a data file is generated with the transposed matrix.

🥐 /C6713 DSK/CPU_1 - C671x - Code C	iomposer Studio - [Disassembly (abort)]	
🔄 File Edit View Project Debug GEL Op	ption Profile Tools DSP/BIOS Window Help	_ 8 ×
🎦 🎽 🖬 X 🗈 🖻 🗠 🗠	- 윩 胬 慃 锔 镉 饧 ● № 唯 唯 雌 雌 匡 王 * 涔 涔 冬 ●	
Comer_Turning.pjt 💌 Debug	🚽 🖉 凿 省 🕭 🗶	
💭 60' 🗋 🖩 🛤 🖬 🛤 🖬 🕰		
Addition Content Content	0000E720 abort, C\$\$EXIT: 0000E724 0000000 000E724 0000000 000E725 0000000 000E726 0000000 000E727 0000000 000E728 0000000 000E729 0000000 000E720 0000000 000E720 0000000 000E721 0000000 000E723 0000000 000E724 0000000 000E725 0000000 000E726 0000000 000E740 0000200 000E750 0000000 000E750 00000000 000E750 <th></th>	
done Reading input data from fil Performing Corner Corner Turning operation fi Program execution done	le data input file Turning Operation inished: Creating data output file	_
		- - -
ALTED: s/w breakpoint	LE	Ln 1, Col 1

Figure 69: Results Obtained after Run the Algorithm "Corner_Turning".

3.5.5 Example 5. Corner Turning -- (*Creating the Project Version*) Code Developed by Abigail Fuentes and Inerys Otero.

AIM:

This example helps us begin to understand the functionality of the CCS and the TMS320C6713 DSP. The Corner Turning algorithm allows the users to obtain the transpose of an input matrix.

EQUIPMENT:

PC	- Windows XP Operating System
Software	- CCStudio V3.3 Platinium
Hardware	- TMS320C6713 DSP

Figure 70 is presenting the files needed for the creation of the Corner Turning project. The folder is located at C:\CCStudio_v3.3\MyProjects\Corner_Turning_files.

Corner_Turning_files		
File Edit View Favorites Tools Help		
🌀 Back 🝷 🕥 🕤 🏂 🔎 Search 🎼 Fo	olders 🕼 🎲 🗙 🗳	9
Address 🛅 C:\CCStudio_v3.3\MyProjects\Corner_Turn	ing_files	
Name 🔺	Size T	уре
🚞 data	Fil	e Folder
C6713dsk.cmd	1 KB - W	indows NT Comm
🗐 c6713dskinit.c	5KB C	File
🗐 C6713dskinit.h	9 KB H	File
🗐 cornerTurning.c	1 KB C	File
📼 csl6713.lib	111 KB LI	B File
🗐 dsk6713.h	2 KB H	File
🗐 dsk6713_aic23.h	9 KB H	File
📾 dsk6713bsl.lib	21 KB LI	B File
🗒 main.c	2 KB C	File
國 rts6700.lib	336 KB LII	B File

Figure 70: Corner Turning Files

Creating the Project:

This section show how to create a project, adding the necessary files to build a project using "Code Composer Studio".

1. Select *Project* → *New*. In the filename, type the name "Corner_Turning" of the new project and click "Save".

This project file (.pjt) is saved in the folder "Corner_Turning" (within C:\CCStudio_v3.3\MyProjects\Corner_Turning). *Figure 72* shows how create a new project and in the *Figure 73* presents where the folder is created.



Figure 71: Creating a New Project

Verify if the following option is selected:

Target \rightarrow *TMS320C67XX*, and then click Finish to continue.

Project Creation	on 🔀
Project Name:	Corner_Turning
Location:	C:\CCStudio_v3.3\MyProjects\Corner_
Project Type:	Executable (.out)
Target	TM\$320C67XX
	Finish Cancel Help

Figure 72: Window for the Creation of a New Project

🗁 MyProjects		
File Edit View Favorites Tools Help		
Search in Folders in the search in the searc	×	9
Address 🛅 C:\CCStudio_v3.3\MyProjects		
Name 🔺	Size	Туре
Corner_Turning		File Folder
Corner_Turning_files		File Folder
🛅 hello		File Folder
_		

Figure 73: Corner_Turning Project Folder

- 2. Copy the following files from C:\CCStudio_v3.3\MyProjects\Corner_Turning_files to C:\CCStudio_v3.3\MyProjects\Corner_Turning:
 - C6713dsk.cmd
 - C6713dskinit.c
 - C6713dskinit.h
 - cornerTurning.c
 - csl6713.lib
 - dsk6713.h
 - dsk6713_aic23.h
 - dsk6713bsl.lib
 - main.c
 - rts6700.lib

Corner_Turning_files		
File Edit View Favorites Tools Help		
🚱 Back 🔹 📀 🕤 🏂 🔎 Search 🎼 Fo	ılders 🕼 🏂 🗙	₽
Address 🛅 C:\CCStudio_v3.3\MyProjects\Corner_Turn	ing_files	
Name 🔺	Size	Туре
🗀 data		File Folder
C6713dsk.cmd	1 KB	Windows NT Comm
📋 c6713dskinit.c	5 KB	C File
📋 C6713dskinit.h	9 KB	H File
📋 cornerTurning.c	1 KB	C File
📼 csl6713.lib	111 KB	LIB File
🗒 dsk6713.h	2 KB	H File
🗐 dsk6713_aic23.h	9 KB	H File
🖬 dsk6713bsl.lib	21 KB	LIB File
🎒 main.c	2 KB	C File
🖬 rts6700.lib	336 KB	LIB File

Figure 74: Corner Turning Project Files

- 3. Select *Project* \rightarrow *Add files to project*. Add the following files to the project:
 - C6713dsk.cmd
 - C6713dskinit.c
 - cornerTurning.c
 - csl6713.lib
 - dsk6713bsl.lib
 - main.c
 - rts6700.lib



Figure 75: Adding Files to the Project

4. Select *Project* → *Scan All Files Dependencies*. Verify that all the files that are shown in *Figure 76* were added to the project.



Figure 76: Project Files

5. Once all of the files are added to the project, the project must be built. This is done by going to *Project* → *Build Options*. This option is used to properly set up the compiler and linker, based on the characteristics of the TMS320C6713 DSP board. Several settings should to be chosen or written, and the option OK is selected after all settings are verified.
| 🧳 / | C <mark>6713 DS</mark>
Edit View | CCPU_1 - C671x - Code C
Project Debug GEL Option | <mark>ompose</mark>
n Profile | r <mark>Studio</mark>
Tools D: | SP/BIOS | Window | Help | | |
|--|-------------------------------------|--|----------------------------------|-----------------------------------|----------------------------------|--------------------------|-------------------|------------------------|--|
| Com | මේ ∎
her_Turning.p
රෝ [ට් | New
Open
Use External Makefile
Export to Makefile
Add Files to Project
Save | | \$ E | 3 # # 2 | .
 | %****
≰
● ♪ | ¥a %4 <i>⊕</i> №
 | ς(⁴) Ξ(₂₀ Ξ(+ Ξ(+ Ξ)) |
| (1)
(1)
(1)
(1)
(1)
(1)
(1)
(1)
(1)
(1) | Files | Close
Source Control | | JAD4
JAD8 | 0010A
20000 | A7A
590 | [B0] | | |
| 6
7 | | Build
Rebuild All
Stop Build | F7 | DADC
DAEO
DAE4
DAE8 | 00008 | 000
790
0A8
000 | | 2 | |
| 70
(†)
≩ | E | Build Clean
Configurations
Build Options
File Sperific Options | | DAEC
DAF0
DAF4
DAF8 | 02306
00006
02108
02306 | 2E6
000
05A
2E6 | | | |
| 浴
答 | E | Project Dependencies
Show Project Dependencies
Show File Dependencies | | | | | | 2 | |
| ∞
■
\$\$ | | Scan All File Dependencies
Recent Project Files | | · | | | | | |
| 6
• | - | | | | | | | | |

Figure 77: Build Option Setting Location

- 6. Under *Compiler* \rightarrow *Category* \rightarrow *Basic*
 - a. The target version: C671x (-mv6710) should be highlighted

Build Options for Corner_Turning.pjt (Debug) 🛛 🛛 🛛 💽								
General Comp	General Compiler Linker DspBiosBuilder Link Order							
-g -fr''\$(Proi_di	-g -fr''\$(Proj_dir)\Debug'' -d''_DEBUG'' -mv6710							
Category: Basic Advanced Feedback Files Assembly Parser Preprocessor Diagnostics	Basic Target Version: C671x (-mv6710) ▼ Generate Debug Info: Full Symbolic Debug (-g) ▼ Opt Speed vs Size: Speed Most Critical (no -ms) ▼ Opt Level: None ▼ Program Level Opt.: None]						

Figure 78: Setting the Target Version

- 7. Under *Compiler* \rightarrow *Category* \rightarrow *Advanced*:
 - In *Memory Models* select *Far* (-*mem_model:data=far*).
 - Verify that Endianness is selected to be *Little Endian*.

Build Options for Corner_Turning, pjt (Debug)								
General Compiler Linker DspBiosBuilder Link Order								
-g -fr''\$(Proi_dir)\Debug'' -d''_DEBUG'' -mv6710mem_model:data=far								
Category: Advanced Basic Advanced Feedback Files Assembly Parser Preprocessor Diagnostics RTS Calls: Use Memory Model • Aliasing: Default • Interrupt Threshold (-mi): Speculate Threshold (-mi): Speculate Threshold (-mi): Turn Off Software Pipelining (-mu) Old 6400 Alignment Compatibility (-mb) Turn Off Reorder of Associative Floating Pt Ops (-mc) Use Function Subsections (-mo)								

Figure 79: Memory Model Type Selection

- 8. Under Compiler → Category → Preprocessor:
 In Pre-Define Symbol, the following should be written: CHIP_6713. This specifies the DSP chip that the target board utilizes.

Build Options for Corner_Turning.pjt (Debug) 🛛 🔹 💽									
General Compiler Linker DspBiosBuilder Link Order									
-g -fr"\$(Proi_dir)\Debug" -d"_DEBUG" -mv6710mem_model:data=far									
Lategory: Basic Advanced Feedback Files Assembly	Category: Preprocessor Basic Include Search Path (-i): Advanced Pre-Define Symbol (-d): Files Undefine Symbol (-u):								
Preprocessor Diagnostics	Continue with Compilation (-ppa)								
OK Cancel Help									

Figure 80: Specifying the Chip Architecture

- 9. Under *Linker* → *Libraries*:
 - In *Included Libraries* (-1), these libraries must be specified: rts6700.lib; dsk6713bsl.lib; csl6713.lib

Build Options	for Corner_Tu	rning.pjt (De	ebug)	? 🛛				
General Comp	oiler Linker Dsp&	3iosBuilder Lin	k Order					
-c -m".\Debug\Corner_Turning.map" -o".\Debug\Corner_Turning.out" -w -x								
Category: Basic Libraries Advanced Libraries (-4): Search Path (-i): Incl. Libraries (-4): csl6713.lib; dsk6713bsl.lib; rts6700.lib								
		OK	Cancel	Help				

Figure 81: Libraries Needed for the Project

10. Now the user may click **OK** once all the previous building option settings have been established.

Compiling and Debugging the Project

In this step the C compilation and linker to build a project are performed.

1. Click on the "rebuild all" button that is in the upper part of the CCS environment and verifies that you have 0 errors.



Figure 82: Compiling Results

Note: If there are errors in your code, they will be listed with the corresponding line numbers. Correct them and rebuild your project.

2. Select *File* → *Load Program*. Choose the file "Corner_Turning.out" that is located in the following path: C:\CCStudio_v3.3\MyProjects\Corner_Turning\Debug.

*	C6713 DSK/CPU	_1 - C6	71x -	Code	Comp	ioser St	udio
�	File Edit View P	roject D)ebug	GEL	Option	Profile	Too
裮	New			_	• Г		
	Open		Ctrl+C)			an.
C	Close		Child			-	8
5	Save As		Cur+3)			
60	Save All						
권					-1		
n	Load Program		Ctrl+L		- 1		
0	Reload Program		Ctrl+5	hirt+L			
()?	Load Symbols Deload Symbols					#incl	ud
6	Lipload Symbols					Uint3	32
7	onioda symbols				÷.	#incl	ud
	Load GEL				- 8	#incl	ud
78	Data					#incl	ud
sh	Workspace				<u> </u>	#incl	ua nd
~	Difference betwe	en files			- 8	#INCI	uu
2	Merge Files				- 8	#defi	ne
浴	Print		Ctrl+P)	-1	c1 4	
2	Decept Source Fil				. .	float	- 1
	Recent Workspace	es es				11040	· U.
öx	Recent Program F	-iles				FILE	* g
	Recent Symbols					void	cŏ
褒	Recent GEL Files				II		
5	Launch Setup				-1	void	ma
	Evit				-11	i	nt
4					-1	i	nt
						1.1.11	

Figure 83: "Load Program" Location

Load Progra	m		? 🗙
Look in: 🗀	Debug	- 🗢 主	
Corner_Tu	rning.out		
File name:	Corner_Turning.out		Open
Files of type:	*.out	•	Cancel
			Help

Figure 84: Corner_Turning.out File Location

🈻 /C6	5713 DSK/CPU_1 - C671x - 0	de Composer Studio - [Dis	assembly]
💽 File	Edit View Project Debug (. Option Profile Tools DSF	9/BIOS Window He
Loadi	ng Program		- 8 8 1
	Loading Program		i 🔏 🕘 🏽 🛛
	rner_Turning\Debug\Cori	_Turning.out	
	Loading sections.) Cancel	08C2 8000 0000 0028 0068
100 to	Documents DSP/BIOS Config Generated Files Include Ibraries Ibraries ds6713.lib dsk6713bsl.lib dx6713bsl.lib	00000018 0000 0000001C 0200 00000020 0000 00000024 3000 00000028 0000 0000002C 0280 00000020 0000 00000034 2000	3666 3664 4000 0612 [!E 8000 3626 29C2 0192 [E

Figure 85: Downloading the Corner_Turning.out File to the TMS320C6713 DSP

- 3. CopythefollowingfilesfromC:\CCStudio_v3.3\MyProjects\Corner_Turning_files\datatoC:\CCStudio_v3.3\MyProjects\Corner_Turning\Debug:
 - Edatain.txt
 - Sdatain.txt



Figure 86: Corner Turning Input Data and Validation Files

4. Click on the "run" button *that* is located in the left side of the environment CCS.

Results Obtained:

On the "Stdout" are printed messages of the program process until the execution is done. In the Debug folder a data file is generated with the transposed matrix.

V /C6713 DSK/CPU_1 - C671x - Code Composer Studio - [Disassembly (abort)]								
File Edit View Project Debug GEL	Option Profile Tools DSP/BIOS Window Help	- 8 ×						
	_ 商商簡簡簡簡 録 録 ♥ 健 健 離 離 桂 桂 本 渉 渉 ◆ ●							
Corner_Turning.pjt 💽 Debug								
💭 🚳 🗂 🗒 🗮 🔤 🖻 🗖								
Image: Second	OUNDER/20 abort, C\$SEXIT: NOP 0000E720 0000000 000E724 0000000 0000E724 0000000 0000E724 0000000 0000E726 0000000 0000E727 0000000 0000E728 0000000 0000E729 0000000 0000E730 0000000 0000E740 etext,cinit. cinit: 0000E740 etext,etext,cinit. cinit: 0000E740 etext,etext,cinit. cinit: 0000E740 word 0x00000230 0000E740 word 0x00000230 0000E740 word 0x00000230 0000E740 word 0x00000230 0000E740 NOP 0x00000230 0000E750 0000000 NOP 0000E750 0000000 NOP 0000E750 0000000 NOP 0000E764 0000000 NOP 0000E778 0000000 NOP 0000E780 0000000 NOP 0000E780							
Corner Turning operation Program execution done	finished: Creating data output file							
HALTED: s/w breakpoint	IE For Help, press F1							

Figure 87: Results Obtained after Run the Algorithm "Corner_Turning".

4 SIGNAL OPERATOR FORMULATIONS FOR MATLAB IMPLEMENTATION

This chapter presents a set of linear finite dimensional signal operators which are fundamentals in the development of signal processing algorithms. The signal operators are formulated with respect to the standards basis Δ_{\Box} to facilitate their matrix implementation. In this contest, the signal operators admit easy implementation in a MATLAB environment, due to the fact that MATLAB stands for MATrix LABoratory and facilitates the implementation of algorithms expressed in matrix-vector form.

4.1 Linear Shift Invariance Systems

4.1.1 Matrix Representation of LSI-FIR Systems

In this section we discuss the representation of LSI-FIR through matrices. Since each N dimensional LSI-FIR system $T_h: L(Z_N) \to L(Z_N)$ represents a linear transformation on the space $L(Z_N)$, T_h is determined by its action on a set of basis vectors (signals) spanning $L(Z_N)$. If we choose as reference the standard basis set $\{\delta_{\{j\}} : j \in Z_N\}$, then each signal $T_h(\delta_{\{j\}}) \in L(Z_N)$ can be uniquely expressed as a linear combination of the basis set. We write

$$T_{h}\left\{\delta_{\{k\}}\right\} = \sum_{j \in \mathbb{Z}_{N}} h[j,k]\delta_{\{j\}}$$

where the set of scalars

$$\{h[j,k]: j \in Z_N\}, k \in Z_N$$

represents the vector coordinates of the given signal $T_h\left\{\delta_{\{k\}}\right\}, k \in \mathbb{Z}_N$, with respect to the standard basis set. The signal $T_h\left\{\delta_{\{k\}}\right\}$ can be written as

$$T_{h}\left\{\delta_{\{k\}}\right\} = \sum_{j \in \mathbb{Z}_{N}} T_{h}\left\{\delta_{\{k\}}\right\} \left[j\right]\delta_{\{j\}}$$

where

$$T_{h}\left\{\delta_{\{k\}}\right\}\left[j\right] = \sum_{m \in \mathbb{Z}_{N}} h[m]\left(S_{N}^{m}\left\{\delta_{\{k\}}\right\}\right)\left[j\right] = \sum_{m \in \mathbb{Z}_{N}} h[m]\delta_{\{k+m\}}\left[j\right]$$
$$= \sum_{m \in \mathbb{Z}_{N}} h[m]\delta[j-k-m]=h[j-k]=\left(S_{N}^{k}\left\{h\right\}\right)[j]$$

Thus, we write

$$T_{h} \left\{ \delta_{\{k\}} \right\} = \sum_{j \in \mathbb{Z}_{N}} h[j,k] \delta_{\{j\}} = \sum_{j \in \mathbb{Z}_{N}} h[j-k] \delta_{\{j\}}$$
$$= \sum_{j \in \mathbb{Z}_{N}} \left(S_{N}^{k} \{h\} \right) [j] S_{N}^{j} \{\delta\} = T_{\left(S_{N}^{k} \{h\} \right)} \{h\} = S_{N}^{k} \{h\}$$

Next, we define the matrix H_N as follows

$$H_{N} = \left[h[j,k]\right]_{j,k \in \mathbb{Z}_{N}} = \left[h[j-k]\right]_{j,k \in \mathbb{Z}_{N}}$$

The matrix H_N , thus, have the following form

$$H_{N} = \begin{bmatrix} h[0] & h[N-1] & h[N-2] & \cdots & h[1] \\ h[1] & h[0] & h[N-1] & \cdots & h[2] \\ h[2] & h[1] & h[0] & \cdots & h[3] \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ h[N-1] & h[N-2] & h[N-3] & \cdots & h[0] \end{bmatrix}$$

We notice that the columns of H_N are formed by shifted versions of the coordinate vector representation of the signal h; that is, we can write H_N as

$$H_{N} = \left[I_{N}h, S_{N}\{h\}, S_{N}^{2}\{h\}, \dots, S_{N}^{N-1}\{h\} \right]$$

where S_N is the matrix representing the shift operator S_N ; and h is the coordinate vector representation of the signal h.

We would like to describe in more details how the matrix H_N , representing the system T_h , is obtained. Starting with expression above, we rewrite

$$\left(\delta_{\{k\}} \right) = \sum_{j \in \mathbb{Z}_N} h[j,k] S_N^j \{\delta\}, \quad h[j,k] \in C = h[0,k] \delta_{\{0\}} + h[1,k] \delta_{\{1\}} + \ldots + h[N-1,k] \delta_{\{N-1\}}$$

Evaluating this expression at different values of $k \in Z_N$ results in the following set of identities:

$$T_{h}\left\{\delta_{\{0\}}\right\} = h[0,0]\delta_{\{0\}} + h[1,0]\delta_{\{1\}} + \dots + h[N-1,0]\delta_{\{N-1\}} + h[N-1,0]\delta_{\{N-1\}} + h[0,1]\delta_{\{0\}} + h[1,1]\delta_{\{1\}} + \dots + h[N-1,1]\delta_{\{N-1\}} \\ \vdots \\ T_{h}\left\{\delta_{\{N-1\}}\right\} = h[0,N-1]\delta_{\{0\}} + h[1,N-1]\delta_{\{1\}} + \dots + h[N-1,N-1]\delta_{\{1\}} + \dots + h[N-1,N-1]\delta_{\{N-1\}}$$

We write these identities in an array form:

$$\begin{bmatrix} T_h \left\{ \delta_{\{0\}} \right\} \\ T_h \left\{ \delta_{\{1\}} \right\} \\ \vdots \\ T_h \left\{ \delta_{\{N-1\}} \right\} \end{bmatrix} = \begin{bmatrix} h[0,0] & h[1,0] & \dots & h[N-1,0] \\ h[0,1] & h[1,1] & \dots & h[N-1,1] \\ \vdots & \vdots & \vdots & \vdots \\ h[0,N-1] & h[1,N-1] & \dots & h[N-1,N-1] \end{bmatrix}$$
$$\bullet \begin{bmatrix} \delta_{\{0\}} \\ \delta_{\{1\}} \\ \vdots \\ \delta_{\{N-1\}} \end{bmatrix}$$

We know obtain a vector-matrix representation of a cyclic convolution opration described in 6.5. Given a system T_h and a signal $f \in L(Z_N)$, the response $g = T_h\{f\}$ is obtained as follows

$$g = T_h \left\{ f \right\} = T_h \left\{ \sum_{k \in \mathbb{Z}_N} f[k] \delta_{\{j\}} \right\}$$
$$= \sum_{k \in \mathbb{Z}_N} f[k] T_h \left\{ \delta_{\{k\}} \right\} = \sum_{j \in \mathbb{Z}_N} g[j] \delta_{\{j\}}$$

Expanding the above sum, we obtain

$$T_h\{f\} = f[0]T_h\{\delta_{\{0\}}\} + f[1]T_h\{\delta_{\{1\}}\} + \ldots + f[N-1]T_h\{\delta_{\{N-1\}}\}$$

where

$$f[0]T_{h}\left\{\delta_{\{0\}}\right\} = f[0]h[0,0]\delta_{\{0\}} + f[0]h[1,0]\delta_{\{1\}} + \dots + f[0]h[N-1]\delta_{\{N-1\}}$$

$$f[1]T_{h}\left\{\delta_{\{10\}}\right\} = f[1]h[0,1]\delta_{\{0\}} + f[1]h[1,1]\delta_{\{1\}} + \dots + f[1]h[N-1,1]\delta_{\{N-1\}}$$

$$\vdots$$

$$f[N-1]T_{h}\left\{\delta_{\{N-1\}}\right\} = f[N-1]h[0,N-1]\delta_{\{0\}} + f[N-1]h[1]\delta_{\{N-1\}} + \dots + f[1]h[N-1,1]\delta_{\{N-1\}}$$

The addition of the above set of equations produces the following expression

$$g = T_h \left\{ f \right\} = \sum_{j \in \mathbb{Z}_N} g \left[j \right] \delta_{\left\{ j \right\}}, \quad f \in L \left(\mathbb{Z}_N \right)$$

$$= \left(f[0]h[0,0] + f[1]h[0,1] + \ldots + f[N-1]h[0,N-1] \right) \delta_{\{0\}} \\+ \left(f[0]h[1,0] + f[1]h[1,1] + \ldots \right) \\+ f[N-1]h[1,N-1] \right) \delta_{\{1\}} + \ldots \\+ \left(f[0]h[N-1,1] + f[1]h[N-1,1] + \ldots \right) \\+ f[N-1]h[N-1,N-1] \right) \delta_{\{N-1\}} \\= T_h \left\{ f \right\} = \sum_{j \in \mathbb{Z}_N} \left(\sum_{k \in \mathbb{Z}_N} f[k]h[j,k] \right) \delta_{\{j\}}$$

where

$$g[m] = T_h \left\{ f \right\} = \left(\sum_{j \in \mathbb{Z}_N} \sum_{k \in \mathbb{Z}_N} f[k] h[j,k] \right) \delta_{\{j\}}[m]$$
$$= \sum_{K \in \mathbb{Z}_N} f[k] h[m,k], m \in \mathbb{Z}_N$$

in vector notation, we have

$$\begin{bmatrix} g[0] \\ g[0] \\ \vdots \\ g[j] \\ \vdots \\ g[N-1] \end{bmatrix} = \begin{bmatrix} \sum_{k=0}^{N-1} f[k]h[0,k] \\ \sum_{k=0}^{N-1} f[k]h[1,k] \\ \vdots \\ \sum_{k=0}^{N-1} f[k]h[j,k] \\ \vdots \\ \sum_{k=0}^{N-1} f[k]h[N-1,k] \end{bmatrix}$$

Factoring out the vector f form above, we obtain the following matrix-vector representation

$$\begin{bmatrix} g[0] \\ g[1] \\ \vdots \\ g[j] \\ \vdots \\ g[N-1] \end{bmatrix} = \begin{bmatrix} h[0,0] & h[0,1] & \cdots & h[0,k] & \cdots & h[0,N-1] \\ h[1,0] & h[1,1] & \cdots & h[1,k] & \cdots & h[1,N-1] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h[j,0] & h[j,1] & \cdots & h[j,k] & \cdots & h[j,N-1] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h[N-1,0] & h[N-1,1] & \cdots & h[N-1,k] & \cdots & h[N-1,N-1] \end{bmatrix}$$

$$\begin{bmatrix} f[0] \\ f[1] \\ \vdots \\ f[k] \\ \vdots \\ f[N-1] \end{bmatrix}$$

Recalling that $h[j,k] == h[j-k], j,k \in \mathbb{Z}_N$, we write

$$\begin{bmatrix} g[0] \\ g[1] \\ \vdots \\ g[j] \\ \vdots \\ g[N-1] \end{bmatrix} = \begin{bmatrix} h[0] & h[N-1] & \cdots & h[N-k] & \cdots & h[1] \\ h[1] & h[0] & \cdots & h[1-k] & \cdots & h[2] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h[j] & h[j-1] & \cdots & h[j-k] & \cdots & h[j+1] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h[N-1] & h[N-2] & \cdots & h[N-1-k] & \cdots & h[0] \end{bmatrix} \\ \begin{bmatrix} f[0] \\ f[1] \\ \vdots \\ f[k] \\ \vdots \\ f[N-1] \end{bmatrix}$$

The above matrix-vector operation $g = H_N(f)$ represents the cyclic convolution operation $g = f * h = T_h\{f\}$, where we have the same symbols \Box and \Box denote, both, the coordinate vector representation of the signals \Box and \Box , respectively, as well as the signals themselves; and the matrix H_N represents the system T_h :

$$H_{N} = \left[T_{h} \left\{ \delta_{\{0\}} \right\}, T_{h} \left\{ \delta_{\{1\}} \right\}, \dots, T_{h} \left\{ \delta_{\{N-1\}} \right\} \right]$$
$$= \left[T_{\delta_{\{0\}}} \left\{ h \right\}, T_{\delta_{\{1\}}} \left\{ h \right\}, \dots, T_{\delta_{\{N-1\}}} \left\{ h \right\} \right]$$
$$= \left[I_{N} T_{h} \left\{ \delta \right\}, S_{N} T_{h} \left\{ \delta \right\}, \dots, S_{N}^{N-1} T_{h} \left\{ \delta \right\} \right]$$

Here, again, we have used commas to separate the vectors; and we have used the same notation used for the signals in order to denote the coordinate vector representation of the signals. The computation of the cyclic convolution operation

$$g = f * h = T_h \{f\}, \quad f, h \in L(Z_N)$$

is now performed by substitution into the defining equation

$$g = T_h \left\{ f \right\} = T_h \left(\sum_{k=0}^{N-1} f[k] \delta_{\{k\}} \right)$$

and proceed in the following manner

$$T_{h}\left\{f\right\} = T_{h}\left\{\sum_{k \in Z_{N}} f[k]\delta_{\{k\}}\right\}$$
$$= \sum_{k \in Z_{N}} f[k]T_{h}\left\{\delta_{\{k\}}\right\}$$
$$= \sum_{k \in Z_{N}} f[k]\left(\sum_{j \in Z_{N}} h[j-k]\delta_{\{j\}}\right)$$
$$= \sum_{j \in Z_{N}}\left(\sum_{k \in Z_{N}} h[j-k]f[k]\right)\delta_{\{j\}}$$

Evaluating $g \in L(Z_N)$ at a particular index value $j \in Z_N$ results in

$$g[j] = T_h\{f\}[j] = \sum_{j \in Z_N} \left(\sum_{k \in Z_N} h[j-k]f[k] \right) \delta_{\{j\}}[j]$$
$$= \sum_{j \in Z_N} \left(\sum_{k \in Z_N} h[j-k]f[k] \right) \delta = \sum_{k \in Z_N} h[j-k]f[k]$$

4.1.2 Spectral Properties of LSI-FIR systems

In this section we will describe the spectral properties of LSI-FIR systems. A shift invariant linear operator acting on an *N*- dimensional vector space may be reprented in the frequency domain by using the concepts of eigen-functions (eigenvectors) and eigenvalues. The eigenvalues correspond to the natural frequencies encountered in the spectral representation of the impulse response signal of a given LSI-FIR system. We will be more explicit later on in describing the relationship existing between the eigenvalues (and their associated eigenfunctions) of a given LSI-FIR operator T_h and the frequency section describing some properties of the system $T_{\delta_{[1]}}$ which are essentially the same as the properties of the shift operator S_N . The simplest LSI-FIR system, apart from the trivial system, i.e., the system represented by the identity operator I_N , is the system represented by the shift operator S_N . The system is sometimes called the unit delay system because its digital electronics hardawre implementation may be accomplished by using a single delay element. We use the same symbol S_N to denote the matrix representation of the shift operator S_N . This matrix reprentation is now given. Recalling that

$$T_{\delta_{\{1\}}} = \sum_{j \in \mathbb{Z}_N} \delta_{\{1\}} [j] S_N^j = S^1 = S_N$$

we have,

$$T_{\delta_{\{1\}}}\left\{\delta_{\{k\}}\right\} = \delta_{\{1\}} * \delta_{\{k\}} = S_N\left\{\delta_{\{k\}}\right\} = \delta_{\{k+1\}}$$

The matrix S_N representing the shift operator S_N is obtain by allowing the vector reprentation (with respect to the standard basis set $\{\delta_{\{k\}}: k \in Z_N\}$) of the signal $T_{\delta_{\{1\}}}\{\delta_{\{k\}}\}, k \in Z_N$, become the columns of the matrix S_N :

$$S_{N} = \left[T_{\delta_{\{1\}}} \left\{ \delta_{0} \right\}, T_{\delta_{\{1\}}} \left\{ \delta_{0} \right\}, \dots, T_{\delta_{\{1\}}} \left\{ \delta_{0} \right\}, \dots, T_{\delta_{\{1\}}} \left\{ \delta_{0} \right\}, \dots, \delta_{0} \right\} \right]$$
$$= \left[\delta_{0} \left\{ 1 \right\}, \delta_{0} \right\}, \dots, \delta_{0} \left\{ 0 \right\} \right]$$

where we have separeted by commas the columns of S_N for legibility. The matrix S_N becomes

		0	0	 0	1
		1	0	 0	0
S_N	=	0	1	 0	0
		:	÷	 ÷	:
		0	0	 1	0

An important property of the S_N operator matrix is that any LSI-FIR system T_h may be represented by a matrix H_N which can be written as a sum of powers of the matrix S_N premultiplied by a diagonal matrix $D_{h[j]}$:

$$H_{N} = \sum_{j \in \mathbb{Z}_{N}} D_{h[j]} S_{N}^{j} = \sum_{j \in \mathbb{Z}_{N}} \left(h[j] \otimes S_{N}^{j} \right)$$

where

$$D_{h[j]} = \begin{bmatrix} h[j] & & \\ & h[j] & \\ & & \ddots & \\ & & & & h[j] \end{bmatrix} , \ j \in Z_N$$

4.2 Cyclic Matrix

A cyclic matrix of order N is a N x N matrix of the form

$$H_{N} = \begin{bmatrix} H_{0} & H_{N-1} & \dots & H_{1} \\ H_{1} & H_{0} & \dots & H_{2} \\ \vdots & \vdots & \ddots & \vdots \\ H_{N-2} & H_{N-3} & \cdots & H_{N-1} \\ H_{N-1} & H_{N-2} & \cdots & H_{0} \end{bmatrix}$$

Notice that the input of each column is exactly the same as the previous column, but they are shifted one position downward. In this case our matrix is cycled downward and has the previous form.

4.3 Discrete Fourier Transform

Given a finite succession x[n], where $0 \le n \le N-1$, the discrete Fourier transform of x[n] is defined as the succession given by

$$x^{k}[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N}$$

where $0 \le k \le N - 1$.

It is common to call Wn=e^-j2pi/N and rewrite the discrete Fourier transform x[n] as

$$x^{k}[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \ 0 \le k \le N-1$$

For a finite succession y[k], where $0 \le k \le N-1$, the inverse discrete Fourier transform of y[k] is given by

$$y^{\vee}[n] = \frac{1}{N} \sum_{n=0}^{N-1} y[n] W_N^{-kn}, \ 0 \le n \le N-1.$$

4.4 Other Operators and Properties

The first operator studied in this section is the reflection operator, which have important and interesting properties.

The reflection operator over the space of unidimentional signals is defined by

$$\begin{array}{rcl} \mathfrak{R}_N & : \ l^2(\mathbb{Z}_N) & \to \ l^2(\mathbb{Z}_N) \\ x & \mapsto \ \mathfrak{R}_N\{x\} = x^{(-)}, \end{array}$$

where

$$\Re_N\{x\}[k] = x^{(-)}[k] = x[\langle N - k \rangle_N] = x[\langle -k \rangle_N].$$

Lets calculate the Rn matrix of the reflection operator with respect to the standard base, this is

$$R_N = [\Re_N \{\delta_{\{0\}}\} \ \Re_N \{\delta_{\{0\}}\} \ \cdots \ \Re_N \{\delta_{\{N-1\}}\}].$$

now,

$$\begin{bmatrix} (\Re_N \{\delta_{\{0\}}\})[n] \end{bmatrix} = \begin{bmatrix} \delta_{\{0\}}[\langle -n \rangle_N] \end{bmatrix} = \begin{bmatrix} 1\\0\\0\\\vdots\\0 \end{bmatrix},$$
$$\begin{bmatrix} (\Re_N \{\delta_{\{1\}}\})[n] \end{bmatrix} = \begin{bmatrix} \delta_{\{1\}}[\langle -n \rangle_N] \end{bmatrix} = \begin{bmatrix} 0\\0\\0\\\vdots\\1 \end{bmatrix},$$
$$\vdots$$
$$\begin{bmatrix} (\Re_N \{\delta_{\{N-1\}}\})[n] \end{bmatrix} = \begin{bmatrix} \delta_{\{N-1\}}[\langle -n \rangle_N] \end{bmatrix} = \begin{bmatrix} 0\\1\\0\\\vdots\\0 \end{bmatrix}.$$

•

So the matrix of the reflection operator is

$$R_N = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

which again we see is a cyclic matrix.

4.5 Hadamard Product

The Hadamard product over the space I²(Zn) of unidimensional signals is defined as

$$\bigcirc : l^2(\mathbb{Z}_N) \times l^2(\mathbb{Z}_N) \longrightarrow l^2(\mathbb{Z}_N) (x, y) \longmapsto x \bigcirc_N y,$$

where

$$(x \odot_N y)[n] = x[n]y[n].$$

So notice that if

$$x = \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix} \quad y \quad y = \begin{bmatrix} y[0] \\ y[1] \\ \vdots \\ y[N-1] \end{bmatrix},$$

Then

$$x \odot_N y = \begin{bmatrix} x[0]y[0] \\ x[1]y[1] \\ \vdots \\ x[N-1]y[N-1] \end{bmatrix}.$$

Hadamard product satisfies the following properties:

- 1. $x \odot_N y = y \odot_N x$, for all $x, y \in l^2(\mathbb{Z}_N)$.
- 2. $x \odot_N (y \odot_N z) = (x \odot_N y) \odot_N z$, for all $x, y, z \in l^2(\mathbb{Z}_N)$.
- 3. $x \odot_N (y + z) = x \odot_N y + x \odot_N z$, for all $x, y, z \in l^2(\mathbb{Z}_N)$.
- 4. $\alpha(x \odot_N y) = (\alpha x) \odot_N y = x \odot_N (\alpha y)$, for all $x, y \in l^2(\mathbb{Z}_N)$ and all $\alpha \in \mathbb{C}$.

4.6 Convolution as a Fundamental Objective

The main objective of this section is the convolution operation as a basic tool in the description of linear systems.

Given a finite signal and a discrete system, find the system output. Remember that all finite signal must be discrete and its domain is a discrete and finite set. If we represent a discrete system as a block diagram the following is obtained:



Figure 88: Discrete System Block Diagram

Observation

Discrete signal is defined as a *vector*. Finite signal is defined as *finite dimension vector*. As a notation, the finite signals are represented as finite dimension vectors in column format.

Example

$$x: Z_{4} \to C$$

$$n \mapsto x[n] = e^{\frac{-j2\pi n}{4}}$$

$$x = \{x[0], x[1], x[2], x[3]\} \Leftrightarrow x = \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix}$$

4.6.1 Discrete Filter

A discrete filter is any system that satisfies the conditions of invariance and linearity.

4.6.2 **Response of a Filter to a Finite Signal**



Figure 89: Discrete Filter Block Diagram

Unitary Impulse:
$$\delta[n] = \begin{cases} 1, n = 0, n \in Z_N \\ 0, n \neq 0, n \in Z_N \end{cases}$$

We represent a vector as follows:

$$\begin{split} & \delta : Z_N \to C \\ & n & \mapsto \delta[n] \\ & \delta = \{\delta[0], \delta[1], \dots, \delta[N-1]\} \quad \Leftrightarrow \quad \delta = \begin{bmatrix} \delta[0] \\ \delta[1] \\ \vdots \\ \delta[N-1] \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \end{split}$$

4.6.3 Finite Response Filters to a Finite Impulse

This type of filter is known in English by its acronym FIR (Finite Impulse Response).

Example

$$T\{\delta[n]\} = h[n], n \in Z_L$$
FIR Filter
$$T$$

$$h: Z_M \to C$$

Figure 90: FIR Filter Block Diagram

Observation:

Every discrete filter with a finite response to an impulse is characterized by its impulse response. This means that everything you need to now regarding this filter is known, and even more, we can get the response of this filter to any input arbitrary but finite.

$$\begin{array}{c} \delta[n], n \in Z_{L} \\ \hline x[n], n \in Z_{L} \\ \end{array} \begin{array}{c} \text{FIR Filter} \\ T \\ y[n] = T\{\delta[n]\} = h[n] \\ \hline y[n] = T\{x[n]\} \\ \end{array}$$

Figure 91: FIR Filter Block Diagram

Example 1

The Finite Response Averaging Filter to a unitary impulse

$$h[n] = \begin{bmatrix} h[0] \\ h[1] \\ \vdots \\ h[M-1] \end{bmatrix} = \begin{bmatrix} 1/M \\ 1/M \\ \vdots \\ 1/M \end{bmatrix}$$

Example 2

Averaging Filter with input $\delta[n-2]$

$$\delta[n-2], n \in \mathbb{Z}_{L}$$
Averaging Filter
$$f$$

$$g[n] = T\{\delta[n-2]\}$$

Figure 92: Averaging Filter Block Diagram

$$\delta[n-2] = S[n]$$
$$S : Z_L \to C$$
$$n \mapsto S[n]$$

$$S = \begin{bmatrix} S[0] \\ S[1] \\ \vdots \\ S[L-1] \end{bmatrix} = \begin{bmatrix} \delta[-2] \\ \delta[-1] \\ \delta[0] \\ \delta[1] \\ \vdots \\ \delta[L-3] \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Observation

All finite signals with dimension L can be represented as a lineal combination of displaced unitary impulse: $x[n] = \sum_{k=0}^{L-1} x[k]\delta[n-k]$.

Example

Represent the signal x[n] = x[n+1], $n \in \mathbb{Z}_4$ as a sum of displaced unitary impulses.

$$x: Z_{4} \to C$$

$$n \mapsto x[n] = n+1$$

$$x[n] = \begin{vmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{vmatrix} = \begin{vmatrix} 1 \\ 2 \\ 3 \\ 4 \end{vmatrix}$$



5 IMAGING FORMATION ALGORITHM

Synthetic aperture radar (SAR) imaging processing consists of forming an image of a landscape or terrain surface using active sensing. In active sensing, an antenna transmits and receives a series of pulse signals reflected from an area of interest. For SAR processing, the antenna is placed on a moving platform, such as an aircraft or satellite. Hence a large surface area can be covered by sections. For each section, the antenna is maintained fixed, keeping that specific area illuminated, which is called a *footprint*. The antenna transmits pulse signals to that region and receives pulses that are reflected back from the surface. The signals that are reflected from the surface area form a reflectivity pattern. A convolution operation is performed between the reflectivity pattern and the impulse response function that characterizes the image formation system. This operation produces a two-dimensional raw *data*. This data is spread in two distinct directions: in the *azimuth* direction, which is defined to be in the same direction parallel to the antenna, and in the *range* direction, which is perpendicular to the azimuth direction (see *Figure 93*). This data requires further processing since the objects present in section of the surface cannot be clearly distinguished. To obtain a better image two types of data compression are applied to the raw data, which are: range compression and azimuth compression.



Figure 93: Range and Azimuth Direction

First a range compression is performed. For this process each row of the raw data is convolved with a range reference function. The range reference function (RRF) is formulated taking into consideration the sampling rate, the duration of the transmitted pulse signal and the frequency modulation (FM) rate of the radar pulse:

$$RRF = e^{+j\theta t^2}, \ \theta = \pi * FM_{rate},$$

where θ is the phase of the range reference function.

A transposed operation is applied on a resulting data obtained of the range compression. The algorithm that is used to execute the transposed operation is known as Corner Turning. Then an azimuth compression is performed. In the azimuth compression, the data compressed in range is convolved with an azimuth reference function. This function is characterized by the duration in which the target is maintained illuminated by the antenna beam, the phase variation detected in the received signal, and the pulse repetition frequency (PRF):

$$ARF = e^{-j\theta}, \ \theta = -2\pi f_{nc}t + \pi k_a t^2,$$

where the θ is the phase of the azimuth reference function which changes with the varying frequency f_{nc} .

5.1 SAR Imaging Formation Design

SAR imaging formation was implemented on the TMS320C6713 DSP board using the design procedure that was followed by Ana Ramirez for the implementation in MATLAB. The Code Composer Studio V3.3 was used to develop the SAR imaging program application in C language. Such program application included the implementation of range compression and azimuth compression algorithms.

The first step for SAR imaging formation in hardware consisted of obtaining the range reference function, the azimuth reference functions, and the raw data. These were obtained by executing the MATLAB program *main.m* created by Ana Ramírez, and then executing the program *CreatingReferenceFunction.m*. This was done in order to generate the corresponding .txt files containing the real and imaginary parts of the complex the reference functions and raw data. Such files were used as input for the SAR imaging formation application program.

In the following figure, a block diagram is presented, which illustrates the overall design procedure to implement the SAR imaging formation in hardware.



Figure 94: SAR Image Formation Diagram Procedure

5.2 Image Formation Results obtained

For the TMS320C6713 DSP board, the range and azimuth compression algorithms were implemented and applied to the raw data provided. The imaging formation results are demonstrated, where raw data of sizes 128x128, 256x256 and 512x512 were processed. The resulting images obtained from range and azimuth compressions on the DSP board were generated in MATLAB from the output files that were created during the imaging formation process.



5.2.1 TMS320C6713 Emulation results for 128x128 Raw Data

Figure 95: Raw Data



Figure 96: Data Compressed in Range



Figure 97: Applying Corner Turning to Data Compressed in Range Direction



Figure 98: Data Compressed in Azimuth Direction



5.2.2 TMS320C6713 Emulation results for 256x256 Raw Data

Figure 99: Raw Data



Figure 100: Data Compressed in Range



Figure 101: Applying Corner Turning to Data Compressed in Range Direction



Figure 102: Data Compressed in Azimuth Direction 132


5.2.3 TMS320C6713 Emulation results for 512x512 Raw Data





Figure 104: Data Compressed in Range



Figure 105: Applying Corner Turning to Data Compressed in Range Direction



Figure 106: Data Compressed in Azimuth Direction

5.3 Example. Imaging Formation -- (*Creating the Project Version*) Code Developed by Abigail Fuentes and Inerys Otero.

AIM:

Synthetic aperture radar (SAR) imaging formation was implemented on the TMS320C6713 digital signal processing (DSP) board. In order to obtain an image formation of a desired surface from raw data, a range compression is first applied to the raw data. The compressed data is then transposed, where such operation is known as corner turning; finally an azimuth compression is applied to the transposed data in order to obtain the final image.

EQUIPMENT:

PC	- Windows XP Operating System
Software	- CCStudio V3.3 Platinium
Hardware	- TMS320C6713 DSP

Main source files needed for application program

The image formation application program is implemented in these two source files:

- *ImageFormation.c* This is the principal program where all the variables are initialized, input data files are read, and output files are created after performing the image formation operation.
- **RangeCompression.c** This function performs the range compression using the range reference function *range_reference_real.txt*, for the real part of the data and *range_reference_imagl.txt* for the imaginary part.
- *AzimuthCompression.c* This function performs the azimuth compression using the nine different azimuth reference functions for both real and imaginary part.
- *cornerTurning.c* This is the actually function that performs the corner turning operation.
- *create_complex_matrix.c* This function joints the real and imaginary part in one complex matrix
- *readingAzimuthFunctions.c* Reads the azimuth functions necessary for the azimuth compression.
- *FFTAzimuth.c* Computes one dimensional fast Fourier transform for the azimuth compression.

- *FFTRange.c* Computes one dimensional fast Fourier transform for the azimuth compression.
- *IFFTRange.c C*omputes one dimensional inverse fast Fourier transform (IFFT) for the range compression.
- *IFFTAzimuth.c C*omputes one dimensional inverse fast Fourier transform for the azimuth compression.
- *divide.c* This function is used to implement the IFFT for both *IFFTRange.c* and *IFFTAzimuth.c*.
- *Separate_matrix.c* This function separates the real and imaginary part of complex matrix
- *ImageFormation_resultsDSP.m* This program provides the image obtained from the range and azimuth compression.

Figure 107 is presenting the files needed for the creation of the Imaging Formation project. The folder is located at C:\CCStudio_v3.3\MyProjects\ImageFormation_files.



Figure 107: ImagingFormation Files

Creating the Project:

This section shows how to create a project, adding the necessary files to build a project using "Code Composer Studio".

1. Select *Project* \longrightarrow *New*. In the filename, type the name "**FFTproject**" of the new project and click "**Save**".

This project file (.pjt) is saved in the folder "**ImageFormation**" (within C:\CCStudio_v3.3\MyProjects\ImageFormation. *Figure 109* shows how to create a new project and *Figure 110* presents where the folder is created.

🦸 /C6713 DSK/CPU_1 - C671x - Code Composer Studio										
File E	dit View	Project	Debug	GEL	Option	Profile	Tools	DSP/BIOS	Window	He
웥 🛛	2 E	New.					-			
		Open								
		Use E	xternal N	1akefil	e		1	🎬 🋗 🖥	6	
· ·		Expo	rt to Mak	efile						
R.	66° 🛛 🗋	Add F	iles to Pr	oject.						
		Save								_
{ }	🔮 Files	Close					bly			
₽ ₽	🕀 🛄 G	e , Sourc	e Contro	I		,	DAD 4	0010A	A7A	
{}			ila Eila			awl cz	-DAD8	20000	590	
a		L Durit	lie Flie				DADC	00008	000	
6		Build				F7	DAEO	00000	790	
		Rebu	Id All				DAE 4	02000	0A8	
		Stop	Build				DAE 8	00006	000	
⇒n		Build	Clean				DAEC	02306	2E6	
		Confi	nurations				-DAF O	00006	000	
{ + }		Build	Options				DAF 4	02108	05A	

Figure 108: Creating a New Project

Verify if the following option is selected:

Target \rightarrow *TMS320C67XX*, and then click **Finish**.

Project Creation	m 🔀
Project Name:	ImageFormation
Location:	dio_v3.3\MyProjects\ImageFormation\
Project Type:	Executable (.out)
Target	TMS320C67XX
	Finish Cancel Help

Figure 109: Window for the Creation of a New Project



Figure 110: "ImageFormation" Project Folder 138

2. Copy the following C:\CCStudio_v3.3\MyProjects\ImageFormation_files C:\CCStudio_v3.3\MyProjects\ImageFormation: files

from to

- C6713dsk.cmd
- C6713dskinit.c
- C6713dskinit.h
- csl6713.lib
- dsk6713.h
- dsk6713_aic23.h
- dsk6713bsl.lib
- rts6700.lib
- AzimuthCompression.c
- bitrev.c
- cornerTurning.c
- create_complex_matric.c
- digitrev_index.c
- divide.c
- FFTAzimuth.c
- FFTRange.c
- Icfftr2_dif.c
- IFFTAzimuth.c
- IFFTRange.c
- ImageFormation.c
- RangeCompression.c
- readingAzimuthFunctions.c
- separate_matrix.c



Figure 111: Image Formation Project Files

- 3. Select *Project* → *Add files to project*. Add the following files:
 - C6713dsk.cmd
 - C6713dskinit.c
 - csl6713.lib
 - dsk6713bsl.lib
 - rts6700.lib
 - AzimuthCompression.c
 - bitrev.c
 - cornerTurning.c
 - create_complex_matric.c
 - digitrev_index.c
 - divide.c
 - FFTAzimuth.c
 - FFTRange.c
 - Icfftr2_dif.c
 - IFFTAzimuth.c
 - IFFTRange.c
 - ImageFormation.c
 - RangeCompression.c
 - readingAzimuthFunctions.c
 - separate_matrix.c



Figure 112: Adding Files to the Project

4. Select *Project* → *Scan All Files Dependencies*. Verify that all the files that are shown in *Figure 113* were added to the project.



Figure 113: Project Files

5. Once all of the files are added to the project, the project must be built. This is done by going to *Project* → *Build Options*. This option is used to properly set up the compiler and linker, based on the characteristics of the TMS320C6713 DSP board. Several settings should to be chosen or written, and the option OK is selected after all settings are verified.



Figure 114: Build Option Setting Location

- 6. Under *Compiler* → *Category* → *Basic*
 - a. The target version: C671x (-mv6710) should be highlighted.

Build Options f	or ImageFormation	n.pjt (Debug)	? 🗙
General Compile	er Linker DspBiosBu Debug'' -d''_DEBUG'' -	ilder Link Orde m∨6710	er	~
Laregory: Basic Advanced Feedback Files Assembly Parser Preprocessor Diagnostics	Basic Target Version: Generate Debug Info: Opt Speed vs Size: Opt Level: Program Level Opt.:	C671x (-mv671 Full Symbolic C Speed Most C None None	0) V Debug (-g) V itical (no -ms)	
		OK (Cancel	Help

Figure 115: Setting the Target Version

- 7. Under Compiler → Category → Advanced:
 In Memory Models select Far (-mem_model:data=far).
 Verify that Endianness is selected to be Little Endian.

Build Options for ImageFormation.pjt (Debug)				
General Compiler Linker DspBiosBuilder Link Order				
-g-fr''\$(Proi_dir)\Debug''-d''_DEBUG''-mv6710mem_model:data=far				
Category:	Advanced			
Basic	RTS Modifications:	Defns No RTS Funcs	7	
Advanced Feedback	Auto Inline Threshold (-c	i:	-	
Files	Endianness:	Little Endian 💌		
Parser	Memory Models:	Far (mem_model:data=far)	-	
Preprocessor Diagnostics	RTS Calls:	Use Memory Model 🔻		
-	Aliasing:	Default		
	🔲 Interrupt Threshold (-mi):		
	🔲 Speculate Threshold	i (-mh):		
	🔲 Turn Off Software Pi	pelining (-mu)		
	□ Old 6400 Alignment Compatibility (-mb)			
	Turn Off Reorder of Associative Floating Pt Ops (-mc)			
	Use Function Subsections (-mo)			
Historic C Pointer to Const Alias Disambiguation (-ox)				
	10	Cancel H	lelp	

Figure 116: Memory Model Type Selection

- 8. Under *Compiler* → *Category* → *Preprocessor*:
 - In *Pre-Define Symbol*, the following should be written: *CHIP_6713*. This specifies the DSP chip that the target board utilizes.
- 9. Under Linker \rightarrow Basic:
 - In Heap Size (-heap) and in Stack Size (-stack), writes 32000.



Figure 117: Building options for Linker → Basic

Build Options f	or ImageFormation	.pjt (De	ebug)	? 🗙
General Compile -g -fr''\$(Proi_dir)' Category: Basic	er Linker DspBiosBu Debug''-d''_DEBUG''-r Preprocessor Include Search Path (-	lder Lin nv6710 i):	k Order mem_model:data	=far
Advanced Feedback Files Assembly Parser Preprocessor Diagnostics	Pre-Define Symbol (-d) Undefine Symbol (-u): Preprocessing: Continue with Com	i CHIP_	6713 ppa)	X
		эк 🛛	Cancel	Help

Figure 118: Specifying the Chip Architecture

- 10. Under *Linker* → *Libraries*:
 - In *Included Libraries* (-1), these libraries must be specified: rts6700.lib; dsk6713bsl.lib; csl6713.lib



Figure 119: Libraries Needed for the Project

11. Now the user may click **OK** once all the previous building option settings have been established.

Compiling and Debugging the Project

Click on the "rebuild all" button that is in the upper part of the CCS environment and review that you have 0 errors.



Figure 120: "ImageFormation" Project Compiling Results

- **Note**: If there are errors in your code, they will be listed with the corresponding line numbers. Correct them and rebuild your project.
 - 11. CopythefollowingfilesfromC:\CCStudio_v3.3\MyProjects\ImageFormation_files\datatoC:\CCStudio_v3.3\MyProjects\ImageFormation\Debug:
 - **raw_data_real128.txt**, **raw_data_imag128.txt** These files were previously generated using MATLAB, as input raw data. Each of these files contains a 128x128 square matrix.
 - **Range and azimuth reference functions** These files are needed to execute the range and azimuth compression:
 - range_reference_real.txt
 - range_reference_imag.txt
 - azimuth128function1real.txt
 - azimuth128function1imag.txt
 - azimuth128function2real.txt
 - azimuth128function2imag.txt
 - azimuth128function3real.txt
 - azimuth128function3imag.txt
 - azimuth128function4real.txt
 - azimuth128function4imag.txt
 - azimuth128function5real.txt
 - azimuth128function5imag.txt
 - azimuth128function6real.txt
 - azimuth128function6imag.txt
 - azimuth128function7real.txt
 - azimuth128function7imag.txt
 - azimuth128function8real.txt
 - azimuth128function8imag.txt
 - azimuth128function9real.txt
 - azimuth128function9imag.txt
 - ImageFormation_resultsDSP.m
 - 12. Select *File* → *Load Program*. Choose the file "*ImageFormation.out*" that is located in the following path: C:\CCStudio_v3.3\MyProjects\ImageFormation\Debug.



Figure 121: "Load Program" Location

Load Progra	m			? 🛛
Look in: ଢ	Debug	•	🕁 🔁) 💣 🎟 -
	nation.out			
File name:	ImageFormation.out			Open
Files of type:	×.out		-	Cancel
				Help

Figure 122: "ImageFormation.out" File Location



Figure 123: Downloading the "ImageFormation.out" File to the TMS320C6713 DSP

13. Click on the "run" button *that* is located in the left side of the environment CCS.

Results Obtained:

Once the image formation application program has finished execution, the following .dat files are created in the directory C:\CCStudio_v3.3\MyProjects\ImageFormation\Debug: dataAzimuth_imag.dat, dataAzimuth_real.dat, DataAzimuthCompressed_imag.dat, DataAzimuthCompressed_real.dat, dataRange_imag.dat, dataRange_real.dat. Run the ImageFormation_resultsDSP.m file using MATLAB to see the resulting images.

6 CONCLUSION AND FUTURE WORK

The TMS320C6713 User's Guide resulted to be extremely helpful in the process of getting acquainted with the DSP unit and Code Composer Studio. Through the User's Guide I was able to learn rapidly and efficiently how to implement different programs and algorithms using the DSP unit.

SAR image formation algorithms were successfully implemented on the TMS320C6713 DSP boards. Images were successfully obtained from the data compression techniques, using raw data supplied by the AIP laboratory. For the TMS320C6713 DSP board, image formation for raw data of sizes 128x128, 256x256, and 512x512 was achieved. For raw data of size 512x512, the images were formed with more details and could be appreciated better, in comparison with raw data of smaller sizes.

I expected that my research project will help future users to bridge the existent gap between the DSP and MATLAB by the further development of tools and examples similar to the one described in this work.

REFERENCES

- [1] Ana B. Ramirez Silva, María Rodríguez, and Domingo Rodríguez, "TMS320C6713 User's Guide". University of Puerto Rico, Mayagüez Campus, 2007.
- [2] Ana Beatriz Ramirez Silva, "On Implementing Time-Frequency Representations on Hardware/Software Computational Structures for SAR Aplications". University of Puerto Rico, Mayagüez Campus, June 2006.
- [3] R. Chassaing, Digital Signal Processing and Application with the C6713 and C6416
 DSK.: Wiley-Interscience, John Wiley & Sons, Inc., 2005.
- [4] G. Franceschetti, R. Lanari, and E.S. Marzouk, "Efficient and high precision spacevariant processing of SAR data," in *Aerospace and Electronic Systems, IEEE Transactions on*, Jan. 1995, pp. 227-237.
- [5] L.J. van Bokhoven, J.P.M. Voeten, and M.C.W. Geilen, "Software synthesis for system level design using process execution trees," in *EUROMICRO Conference*, 1999.
 Proceedings. 25th, 1999, pp. 463-467 vol.1.
- [6] Guido Arnout, "C for System Level Design," in *Design, Automation and Test in Europe Conference and Exhibition 1999. Proceedings*, 2002, p. 384.
- [7] H.D. Patel, S.K. Shukla, and R.A. Bergamaschi, "Heterogeneous Behavioral Hierarchy for System Level Designs," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, 2006.
- [8] D.D. Gajski, "New Strategies for System Level Design," in VLSI Design, Automation and Test, 2006 International Symposium on, CA, 2006.

- [9] Inc. The MathWorks, "Embedded MATLAB™ User's Guide". MA: The MathWorks, Inc., 2007.
- [10] W. Tibboel, V. Reyes, M. Klompstra, and D. Alders, "System-Level Design Flow Based on a Functional Reference for HW and SW," in *Design Automation Conference*, 2007. DAC '07. 44th ACM/IEEE, June 2007, pp. 23-28.
- [11] M. di Bisceglie, M. Di Santo, C. Galdi, R. Lanari, and N. Ranaldo, "Synthetic Aperture Radar Processing with GPGPU," in *Signal Processing Magazine*, *IEEE*, March 2010, pp. 69-78.
- [12] M.G. Morrow, T.B. Welch, and C.H.G. Wright, "A Host Port Interface Board to Enhance the TMS320C6713 DSK," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, May 2006.
- [13] Texas Instruments Inc., *TMS320C6713 Floating-Point Digital Signal Processor*. Texas: Texas Instruments Incorporated, November 2005.

APPENDIX A. TMS320C6713 DSP ATRIBUTES

Digital Signal Processor (DSP) is used for a wide range of applications such as image processing, speech recognition, control, medicine, spectrography, communications, seismography and others. The wide range of applications is due to the real-time processing with that they are concerned. Some advantages of using DSP is because they are less affected by environmental conditions, are easy to use, flexible and economical in comparison with the analogous devices.

The primary tool for designing a DSP application program is the "Digital Starter Kit (DSK) from Texas Instruments, Inc. The DSK package is useful to developers and it is made up by Code Composer Studio (CCS) and a development board (TMS320C6713 DSK).

This starter kit is useful for developers because they can test the performance of the algorithms implemented before the mass production of devices for specific applications. Besides, DSK has connections for peripherals (Audio, memory or JTAG connectors for example) to simulate the input and output signals to the processor. This tool is compatible with PCs and requires a USB connection to program it.

TMS320C6713 DSK Features

On next table there are some basic attributes of the TMS320C6713 Digital Started Kit:

FEATURES	VALUE
Clock Frequency	225 MHz
SDRAM Memory	16 MB
FLASH Memory	256 KB
Architecture	VLIW (Very-Long-Instruction-Word)
I/O Audio Stereo	2 for input and 2 for output

Table 1: TMS320C6713 DSK Features

Other special characteristics available on the DSK are:

- The board has an analog to digital converter (ADC) and a digital to analog converter (DAC).
- The McASP channels have a special input filter for anti-aliasing to eliminate erroneous signals and an output filter to smooth or reconstruct the processed output signal.
- A daughter card expansion with 80-pin connector provided for external peripheral and external memory interfaces.
- Four user dip switches.
- Voltage regulators that provide 1.26V for the DSP and 3.3 V for the memory and peripherals.

TMS320C6713 DSP Architecture

The TMS320C6713 DSP internal memory has two-level cache architecture. The first level has 4KB of program cache and 4KB data cache and the second level has 256 KB shared between program and data memory. There are in two different banks with two different busses of 32 bits to be accessed independently.

The CPU of the DSP has eight independent functional units divided in two paths, which are useful for multiply operations (.M), logical and arithmetical operations (.L), for bit manipulations (.S) and loading/storing (.D).