

**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

Date

Nayda Santiago, PhD
Member, Graduate Committee

Date

Domingo Rodríguez, PhD
President, Graduate Committee

Date

Héctor Rosario, PhD
Representative of Graduate Studies

Date

Erick Aponte, PhD
Chairperson of the Department

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 and 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 the WALSAIP project.

Table of Contents

ABSTRACT.....	II
RESUMEN.....	III
ACKNOWLEDGEMENTS.....	V
TABLE OF CONTENTS.....	VI
TABLE LIST.....	VIII
FIGURE LIST.....	IX
1 INTRODUCTION.....	1
1.1 MOTIVATION.....	2
1.2 SUMMARY OF FOLLOWING CHAPTERS.....	2
2 SIGNAL PROCESSING FUNDAMENTALS.....	4
3 TMS320C6713 DSP DEVELOPMENT SYSTEM.....	27
3.1 CODE COMPOSER STUDIO IDE (CCS).....	27
3.2 CCS INSTALLATION AND SUPPORT.....	29
3.3 CCS SETUP AND INITIALIZATION.....	46
3.3.1 <i>Selecting Simulation Environment</i>	49
3.3.2 <i>Selecting Emulation Environment</i>	51
3.4 GENERAL ALGORITHM IMPLEMENTATION ON THE BOARD.....	55
3.4.1 <i>Types of Useful Files</i>	55
3.4.2 <i>DSK Support Tools</i>	56
3.5 PROGRAMMING EXAMPLES TO TEST THE DSK TOOLS.....	57
3.5.1 <i>Example 1. Hello World!</i>	57
3.5.2 <i>Example 2. Fast Fourier Transform (FFT) -- (Created Project Version)</i>	66
3.5.3 <i>Example 3. Fast Fourier Transform (FFT) -- (Creating the Project Version)</i>	72
3.5.4 <i>Example 4. Corner Turning -- (Created Project Version)</i>	85
3.5.5 <i>Example 5. Corner Turning -- (Creating the Project Version)</i>	91
4 SIGNAL OPERATOR FORMULATIONS FOR MATLAB IMPLEMENTATION.....	105
4.1 LINEAR SHIFT INVARIANCE SYSTEMS.....	105
4.1.1 <i>Matrix Representation of LSI-FIR Systems</i>	105
4.1.2 <i>Spectral Properties of LSI-FIR systems</i>	112
4.2 CYCLIC MATRIX.....	114
4.3 DISCRETE FOURIER TRANSFORM.....	115
4.4 OTHER OPERATORS AND PROPERTIES.....	116
4.5 HADAMARD PRODUCT.....	118
4.6 CONVOLUTION AS A FUNDAMENTAL OBJECTIVE.....	119
4.6.1 <i>Discrete Filter</i>	120
4.6.2 <i>Response of a Filter to a Finite Signal</i>	120
4.6.3 <i>Finite Response Filters to a Finite Impulse</i>	121
5 IMAGING FORMATION ALGORITHM.....	124
5.1 SAR IMAGING FORMATION DESIGN.....	126
5.2 IMAGE FORMATION RESULTS OBTAINED.....	128
5.2.1 <i>TMS320C6713 Emulation results for 128x128 Raw Data</i>	128
5.2.2 <i>TMS320C6713 Emulation results for 256x256 Raw Data</i>	131

5.2.3	<i>TMS320C6713 Emulation results for 512x512 Raw Data</i>	133
5.3	EXAMPLE. IMAGING FORMATION -- (<i>CREATING THE PROJECT VERSION</i>)	135
6	CONCLUSION AND FUTURE WORK	153
	REFERENCES	154
	APPENDIX A. TMS320C6713 DSP ATRIBUTES	156

Table List

Tables	Page
<i>Table 1: TMS320C6713 DSK Features</i>	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	36
Figure 10: CCS Emulation Drivers Main Menu Window	37
Figure 11: CCS 3.1 Planinum Driver	38
Figure 12: CCS 3.1 Emulation Drivers Installation Window	39
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

<i>Figure 41: “FFTproject.out” File Location</i>	70
<i>Figure 42: Downloading the “FFTproject.out” File to the TMS320C6713 DSP</i>	70
<i>Figure 43: Results Obtained after Run the FFT Algorithm.</i>	71
<i>Figure 44: FFT Project Files</i>	72
<i>Figure 45: Creating a New Project</i>	73
<i>Figure 46: Window for the Creation of a New Project</i>	74
<i>Figure 47: FFT Project Folder</i>	74
<i>Figure 48: FFT Project Files</i>	75
<i>Figure 49: Adding Files to the Project</i>	76
<i>Figure 50: Project Files</i>	77
<i>Figure 51: Build Option Setting Location</i>	78
<i>Figure 52: Setting the Target Version</i>	78
<i>Figure 53: Memory Model Type Selection</i>	79
<i>Figure 54: Specifying the Chip Architecture</i>	80
<i>Figure 55: Libraries Needed for the Project</i>	81
<i>Figure 56: FFT Project Compiling Results</i>	81
<i>Figure 57: “Load Program” Location</i>	82
<i>Figure 58: “FFTproject.out” File Location</i>	82
<i>Figure 59: Downloading the “FFTproject.out” File to the TMS320C6713 DSP</i>	83
<i>Figure 60: Results Obtained after Run the FFT Algorithm.</i>	84
<i>Figure 61: Corner Turning Files</i>	85
<i>Figure 62: Open "Corner Turning" Project</i>	86
<i>Figure 63: Corner_Turning Project Selection</i>	87
<i>Figure 64: CCS Environment for Corner Turning Example</i>	87
<i>Figure 65: Corner Turning Compiling Results</i>	88
<i>Figure 66: “Load Program” Location</i>	88
<i>Figure 67: “Corner_Turning.out” File Location</i>	89
<i>Figure 68: Downloading the Corner_Turning.out File to the TMS320C6713 DSP</i>	89
<i>Figure 69: Results Obtained after Run the Algorithm "Corner_Turning".</i>	90
<i>Figure 70: Corner Turning Files</i>	91
<i>Figure 71: Creating a New Project</i>	92
<i>Figure 72: Window for the Creation of a New Project</i>	93
<i>Figure 73: Corner_Turning Project Folder</i>	93
<i>Figure 74: Corner Turning Project Files</i>	94
<i>Figure 75: Adding Files to the Project</i>	95
<i>Figure 76: Project Files</i>	96
<i>Figure 77: Build Option Setting Location</i>	97
<i>Figure 78: Setting the Target Version</i>	97
<i>Figure 79: Memory Model Type Selection</i>	98
<i>Figure 80: Specifying the Chip Architecture</i>	99
<i>Figure 81: Libraries Needed for the Project</i>	100
<i>Figure 82: Compiling Results</i>	100
<i>Figure 83: “Load Program” Location</i>	101

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

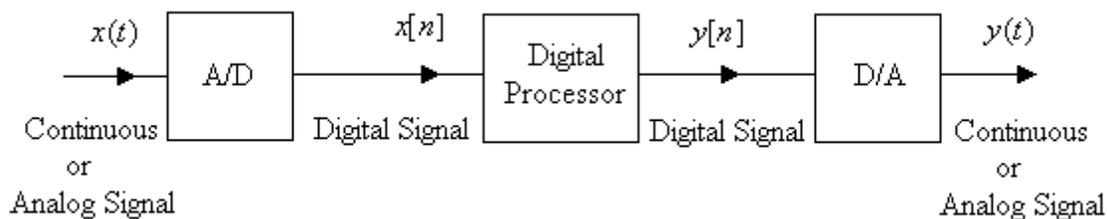
This document is organized as follow: Chapter 2 presents important signal processing fundamentals theory which is essential to understand de development of this project. Chapter 3 presents a description of the TMS320C6713 Digital Starter Kit (DSK) and its development environment Code Composer Studio 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. A digital-to-analog system performs an inverse operation; that is, it converts a digital signal into an analog signal or continuous-domain signal. A continuous-domain signal is normally referred to as a continuous-time 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 $\mathbb{Z}_4 = \{0, 1, 2, 3\}$:

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

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 $x: \mathbb{R} \rightarrow \mathbb{C}$

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

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

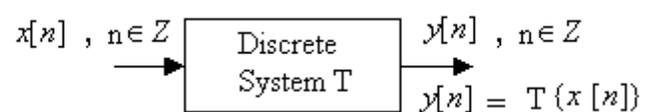
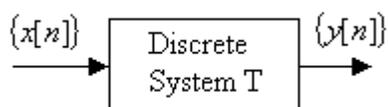
A sequence $\{y[n]\}$ is causal and finite if $\{y[n], n \in Z_N\}$. In this case we say that the signal has length N .

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 these 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: $a = b = 1$

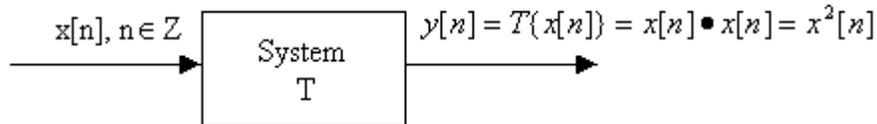
$$T\{x_1[n] + x_2[n]\} = T\{x_1[n]\} + T\{x_2[n]\}$$

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



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:

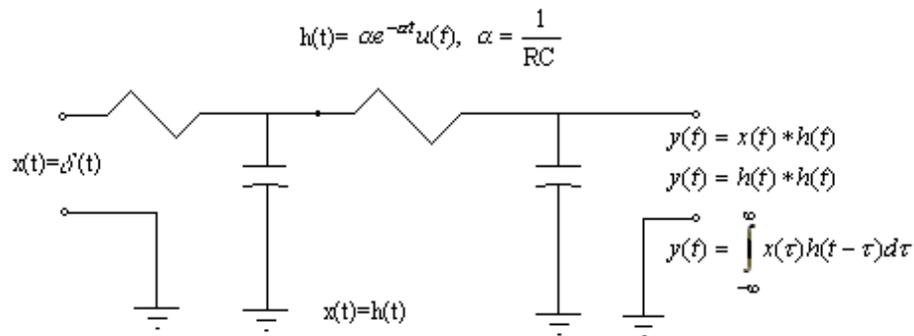
A filter T is called causal if the impulse response signal of the filter is a causal signal.

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

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:

$$a_M \frac{d^M}{d_t^M} (y(t)) + a_{M-1} \frac{d^{M-1}}{d_t^{M-1}} (y(t)) + \dots + a_0 y(t) = b_N \frac{d^N}{d_t^N} x(t) + b_{N-1} \frac{d^{N-1}}{d_t^{N-1}} x(t) + \dots + b_0 x(t)$$

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 an 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.

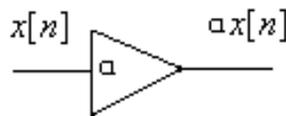
Discrete Filter Implementation:

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 transform is defined as follows

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

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:

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

$$\text{Define } X(\omega) = \mathfrak{F}\{x[n]\} = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}; \omega \in \mathfrak{R}$$

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:

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

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

$$y[n] = x[n] \circledast h[n],$$

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

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

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

$$y[n] = \sum_{k=0}^{N-1} x[k]h[\langle n-k \rangle_N]; n \in Z_N$$

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:

1. 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.
2. 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

$$X(\omega) = 1 + \alpha e^{-j\omega} + \alpha^2 e^{-j2\omega} + \alpha^3 e^{-j3\omega} + \dots$$

$$X(\omega) = \sum_{n=0}^{\infty} (\alpha e^{-j\omega})^n$$

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

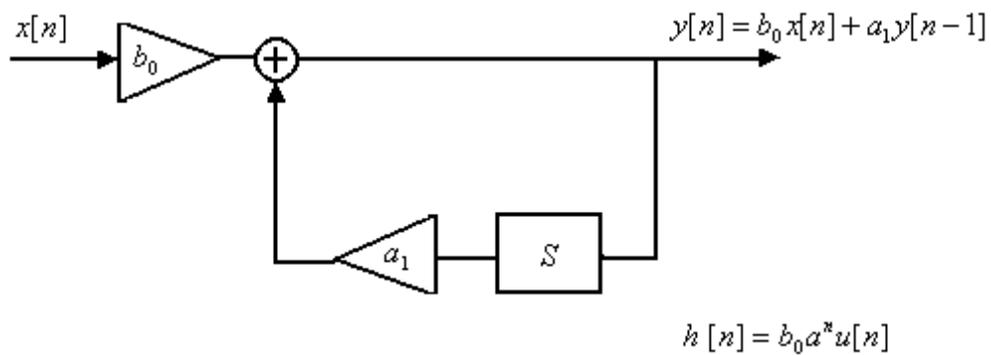
$$X(\omega) = \sum_{n=0}^{\infty} b^n = 1 + b + b^2 + b^3 + \dots$$

$$X(\omega) - bX(\omega) = 1$$

$$(1 - b)X(\omega) = 1$$

$$\therefore X(\omega) = \frac{1}{1 - b} = \frac{1}{1 - \alpha e^{-j\omega}}$$

Filter Design: First-order



FIR

$$h_D[n] = \begin{cases} h[n], & n \in Z_N \\ 0, & \text{otherwise} \end{cases}$$

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:

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

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:

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

$$h: Z \rightarrow C$$

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: $N = 4$

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

$$w_4 = e^{-j\frac{2\pi}{4}}$$

$$w_4^6 = e^{-j\frac{2\pi 6}{4}} = \underbrace{e^{-j\frac{2\pi 2}{4}}}_{w_4^2} \cdot \underbrace{e^{-j\frac{2\pi 4}{4}}}_1$$

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

Example:

N	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: $N = 4$

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

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

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

$x_e \qquad x_o$

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

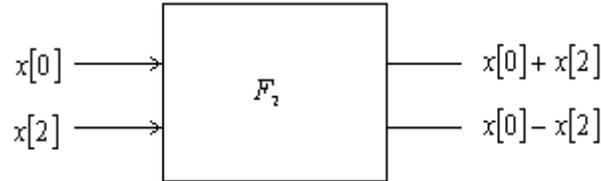
sparse matrix

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

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

$$w_4^2 = e^{-j\frac{2\pi 2}{4}} = e^{-j\pi} = \cos \pi - j \sin \pi = -1$$

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x[0] \\ x[2] \end{bmatrix} = \begin{bmatrix} x[0] + x[2] \\ x[0] - x[2] \end{bmatrix}$$

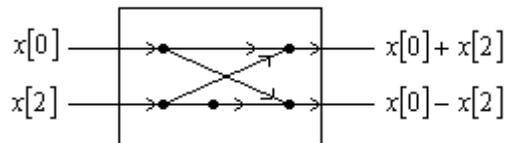


$$F_2 = [w_2^{kn}]_{k,n \in \mathbb{Z}_2} = \begin{bmatrix} 1 & 1 \\ 1 & w_2 \end{bmatrix}$$

$$w_2 = e^{-j\frac{2\pi}{2}} = -e^{-j\pi} = -1$$

Butterfly Block Diagram (Flow Diagram)

Representation of the FFT:



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

We want to compute

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

16 multiplications

12 summations

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

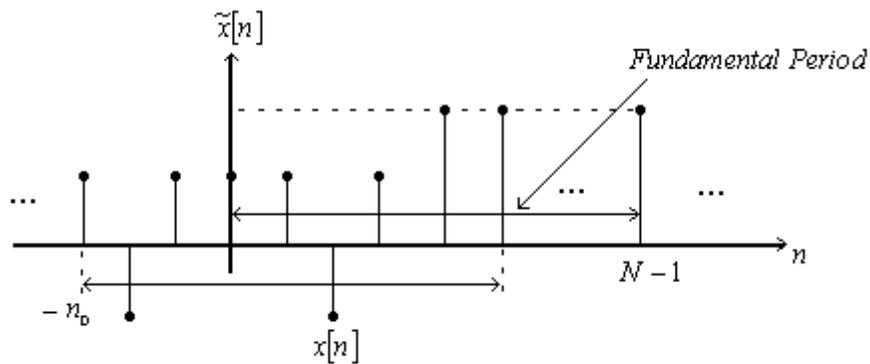
$$2. F_4 x_0 = F_4 \begin{bmatrix} 0 \\ x[1] \\ 0 \\ x[3] \end{bmatrix}$$

In general, we want to know

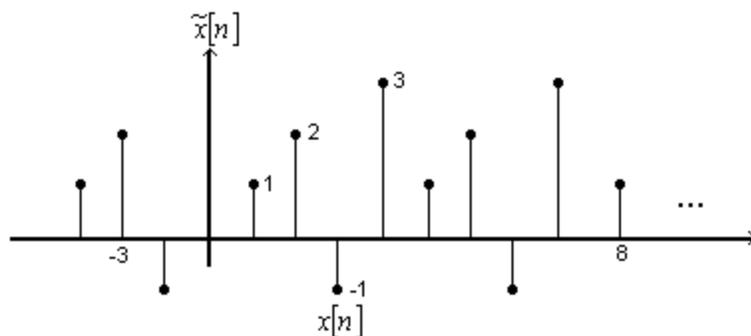
$$\text{DFT } \{x[n - n_0]\} = \sum_{n=0}^{N-1} x[n - n_0] W_n^{Kn}$$

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

$$\begin{aligned} \text{DFT } \{x[n - n_0]\} &= \sum_{m=-n_0}^{m=N-1-n_0} x[m] W_n^{K(m+n_0)} \\ &= W_N^{Kn_0} \sum_{m=-n_0}^{m=N-1-n_0} x[m] W_N^{Km} \end{aligned}$$



Example:



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

$$\langle -3 \rangle_4 = \langle -3 + q4 \rangle_4 = \langle -3 + 4 \rangle_4 = \langle 1 \rangle_4 = 1$$

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

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

Hadamard product

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

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

Express $F_4 x_0$ in matrix form.

$$G[k] \rightarrow \text{long. } N$$

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

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

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

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

$$\text{DFT}_N \{x[\langle n - n_0 \rangle_N]\} = \mathbf{W}_N^{Kn_0} \cdot \mathbf{X}[K]$$

Example: $N = 4$, $x[n] = \{x[0], x[1], x[2], x[3]\}$

$$y[n] = x[\langle n - n_0 \rangle_N]; n_0 = 2$$

$$y[n] = x[\langle n - 2 \rangle_4]; n \in Z_4$$

$$y[0] = x[\langle 0 - 2 \rangle_4] = x[2]$$

$$y[1] = x[\langle 1 - 2 \rangle_4] = x[3]$$

$$y[2] = x[\langle 2 - 2 \rangle_4] = x[0]$$

$$y[3] = x[\langle 3 - 2 \rangle_4] = x[1]$$

$$\langle p \rangle_N = \langle p + qN \rangle_N$$

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

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

$$\langle 1 \rangle_4 = 1$$

$$\langle 5 \rangle_4 = \langle 1 + 4 \rangle_4 = \langle 1 \rangle_4 + \langle 4 \rangle_4$$

$$\langle 9 \rangle_4 = \langle 1 + 2 \cdot 4 \rangle_4 = \langle 1 \rangle_4 + \langle 8 \rangle_4$$

$$\langle 21 \rangle_4 = \langle 1 + 5 \cdot 4 \rangle_4 = \langle 1 \rangle_4 + \langle 20 \rangle_4$$

$$\langle -21 \rangle_{11} = \langle -21 + 2 \cdot 11 \rangle = 1$$

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

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

$$F_4 \begin{bmatrix} x[1] \\ 0 \\ x[3] \\ 0 \end{bmatrix} = \begin{bmatrix} F_2 \\ F_2 \end{bmatrix} \cdot \begin{bmatrix} x[1] \\ x[3] \end{bmatrix} = DFT_4 \{s[n]\} = S[K]$$

We want

$$F_4 \begin{bmatrix} 0 \\ x[1] \\ 0 \\ x[3] \end{bmatrix} = DFT_4 \{s[\langle n - n_0 \rangle_4]\}; n_0 = 1$$

$$F_4 \begin{bmatrix} 0 \\ x[1] \\ 0 \\ x[3] \end{bmatrix} = W_4^{kn_0} \cdot S[k]; k \in Z_4$$

If $n_0 = 1$

$$W_4^{kn_0} = \begin{bmatrix} 1 \\ w_4 \\ w_4^2 \\ w_4^3 \end{bmatrix}$$

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

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

Remember:

$$F_2 = \begin{bmatrix} 1 & 1 \\ 1 & w_4^2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & w_2 \end{bmatrix}$$

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

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

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 as 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.

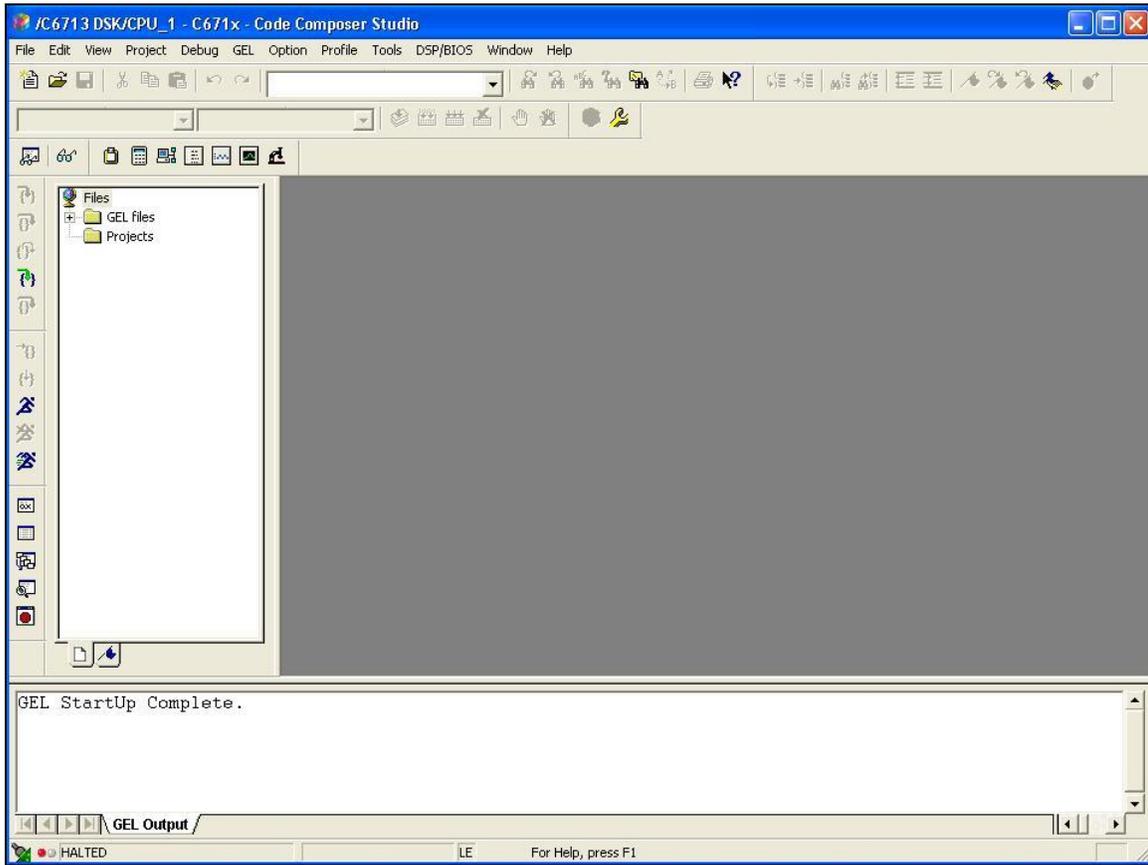


Figure 1: Sample of Programming Environment: "Code Composer Studio".

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 Platinum 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 (www.ti.com) it is possible to access technical documentation, download libraries, discussion groups and technical conferences.

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

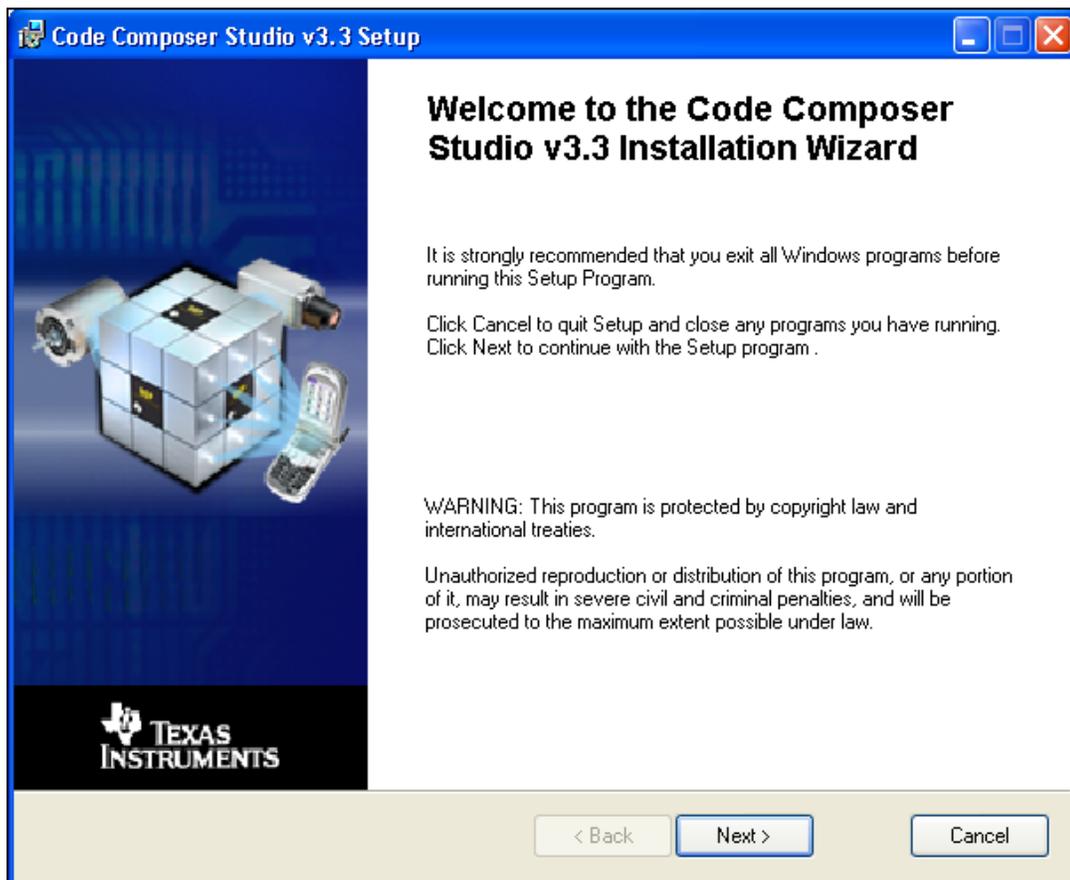


Figure 2: Code Composer Studio (CCS) v3.3 Installation Wizard

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 requirements for installation.

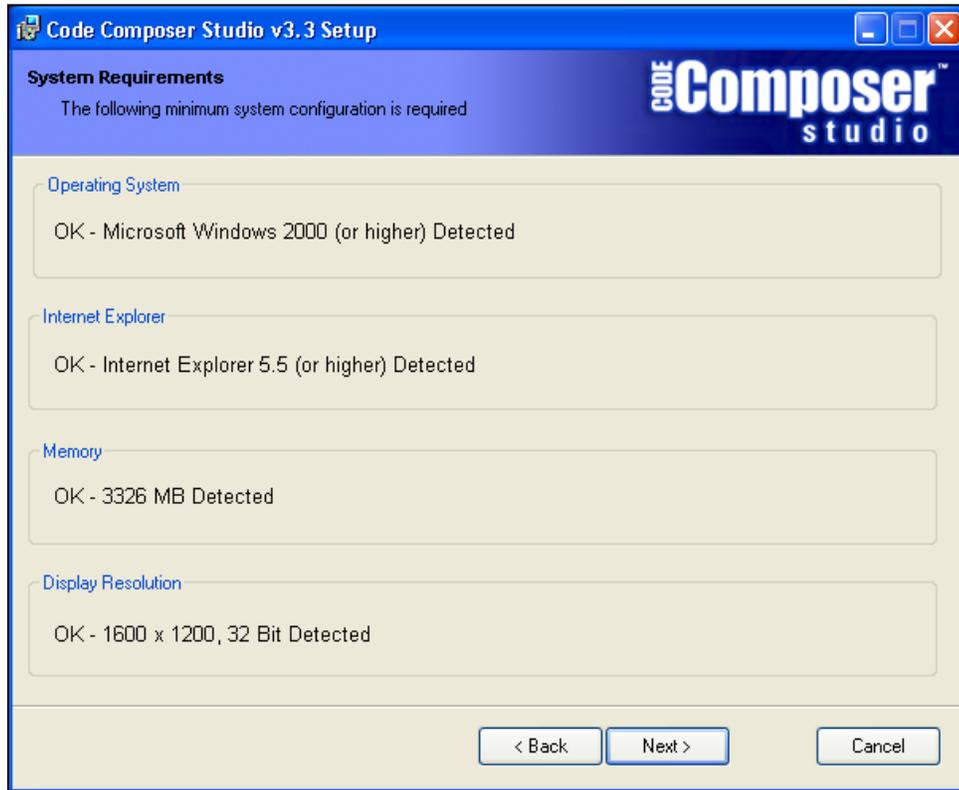


Figure 3: CCS System Requirements Verification

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*).

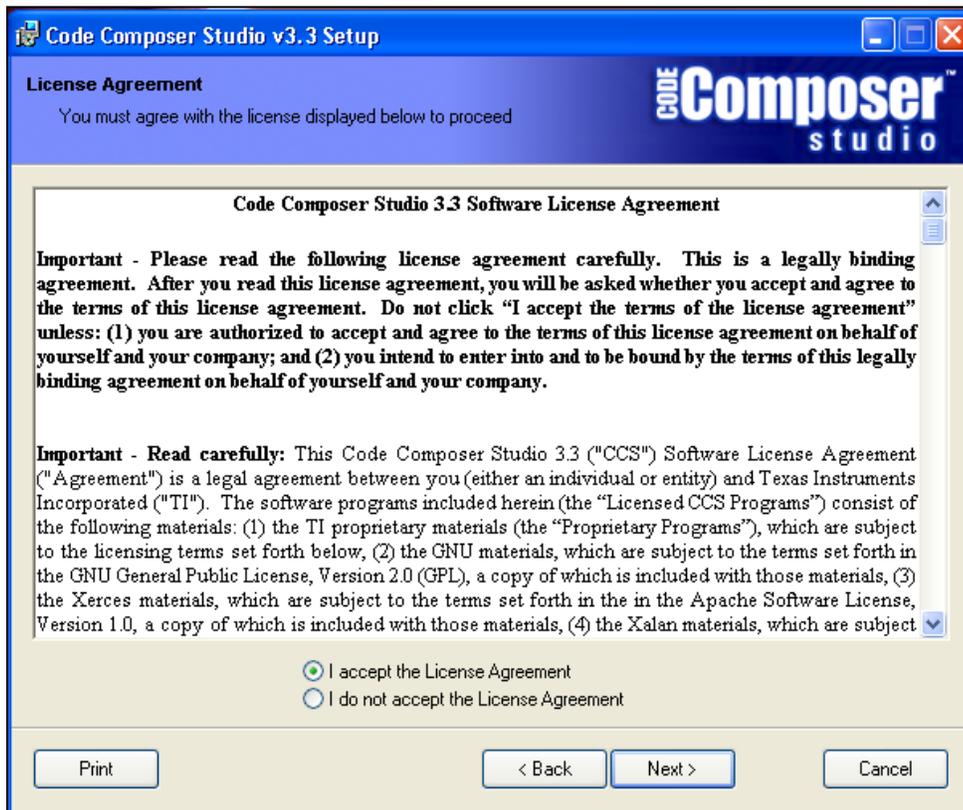


Figure 4: CCS License Agreement

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**”.

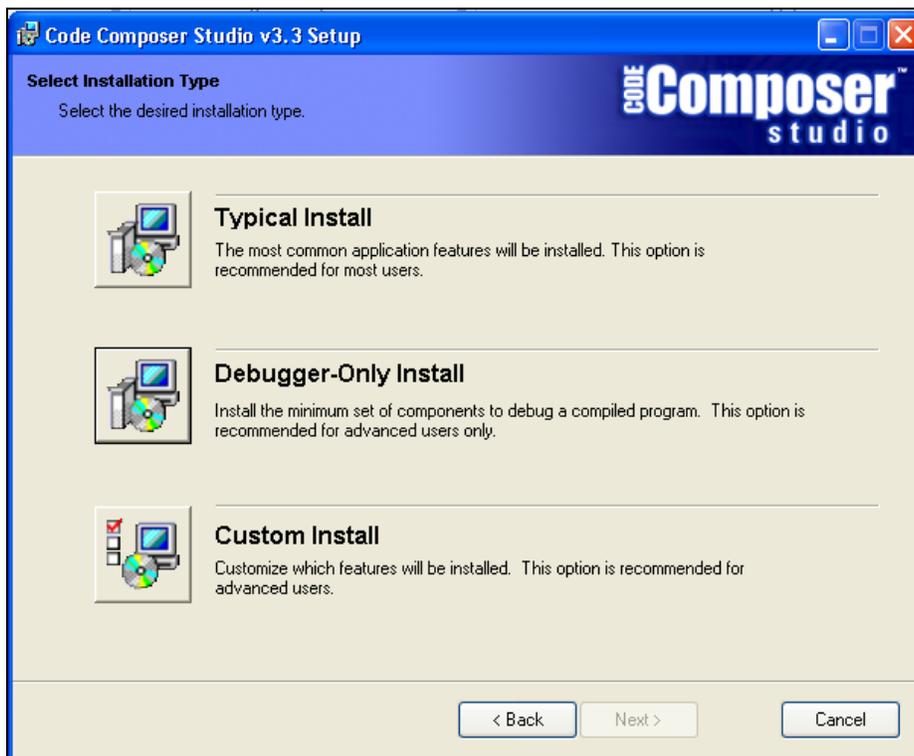


Figure 5: CCS Installation Type Selection

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.

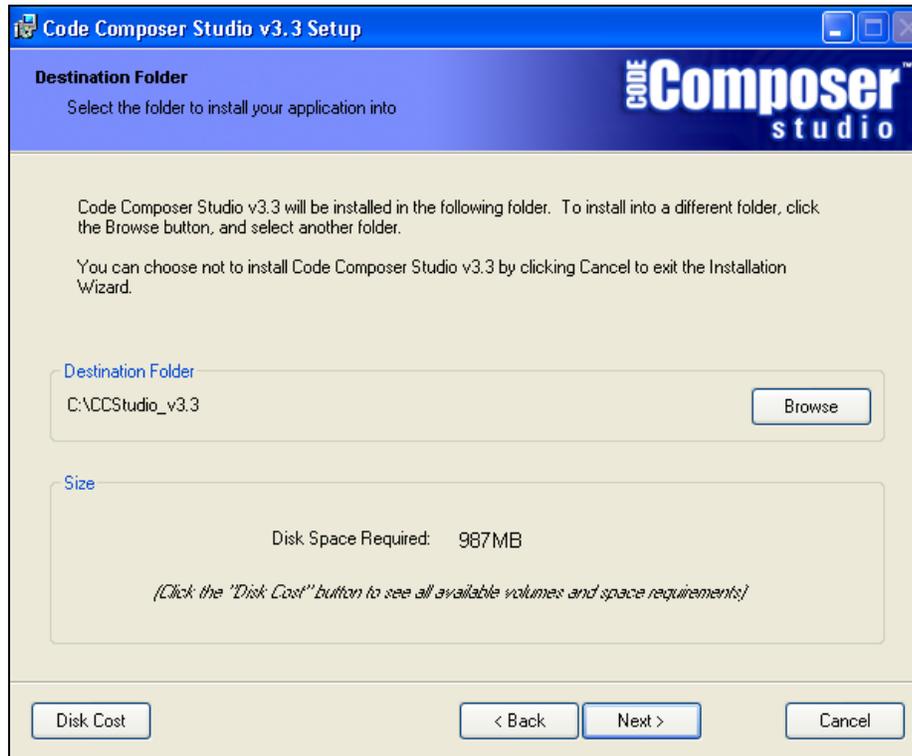


Figure 6: CCS Destination Folder

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



Figure 7: Code Composer Studio v3.3 Installation

This window is presenting the installation progress.

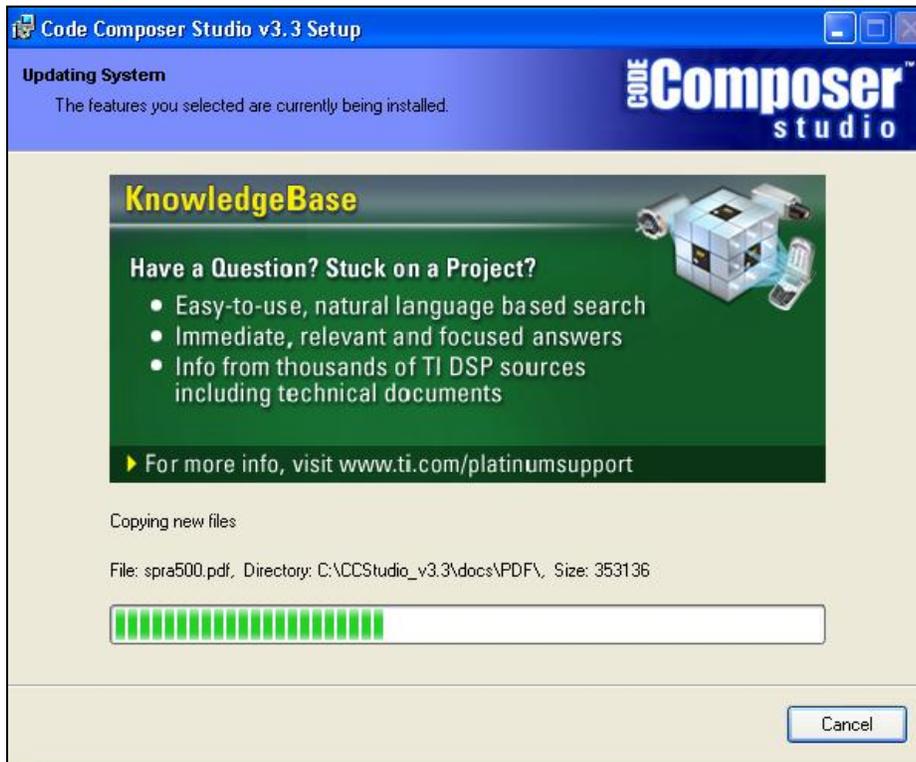


Figure 8: CCS Installation Progress

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*).

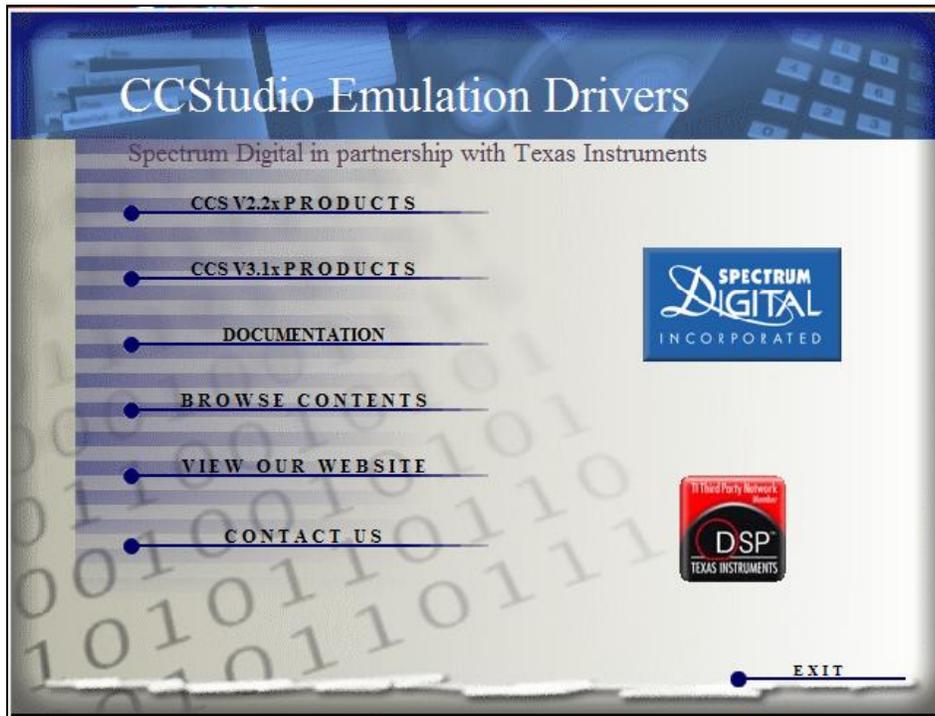


Figure 10: CCS Emulation Drivers Main Menu Window

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

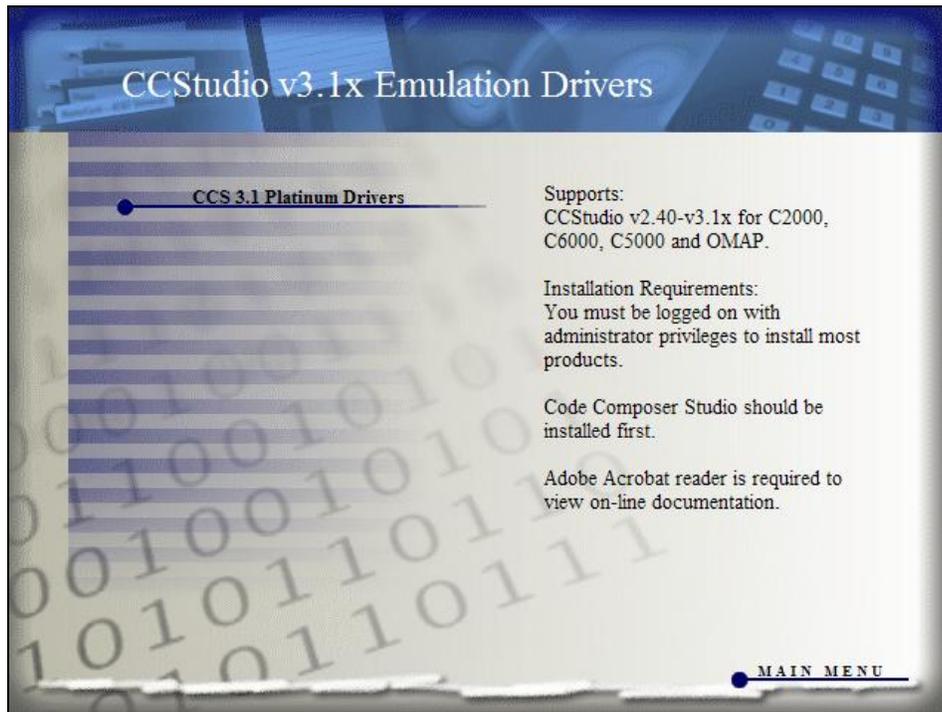


Figure 11: CCS 3.1 Planinum Driver

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

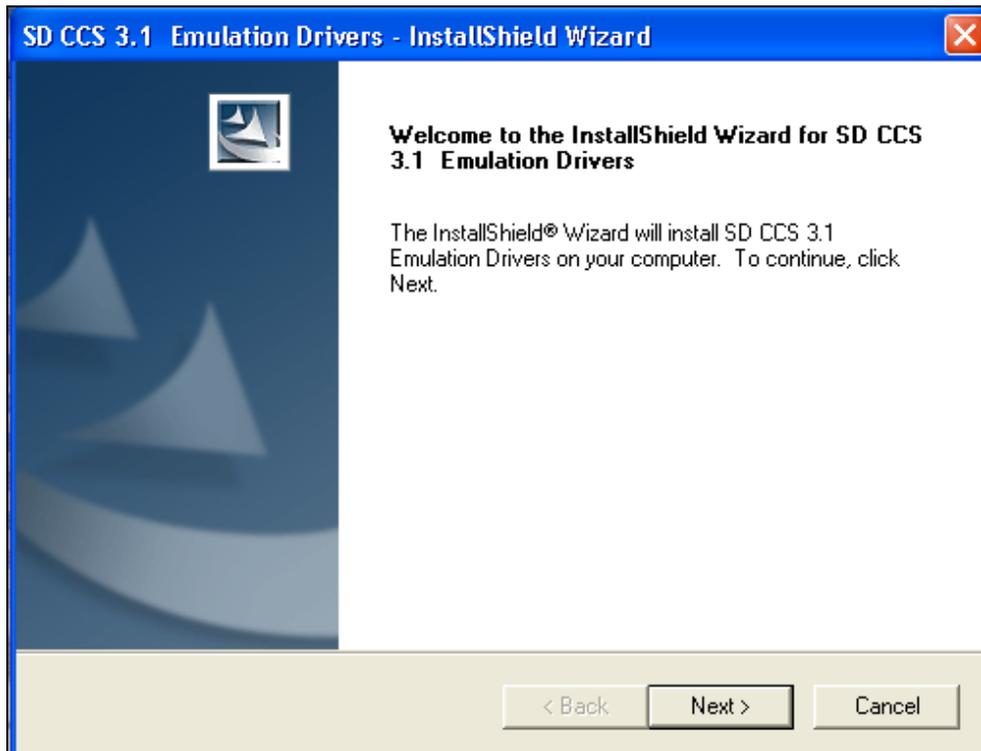


Figure 12: CCS 3.1 Emulation Drivers Installation Window

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**”.

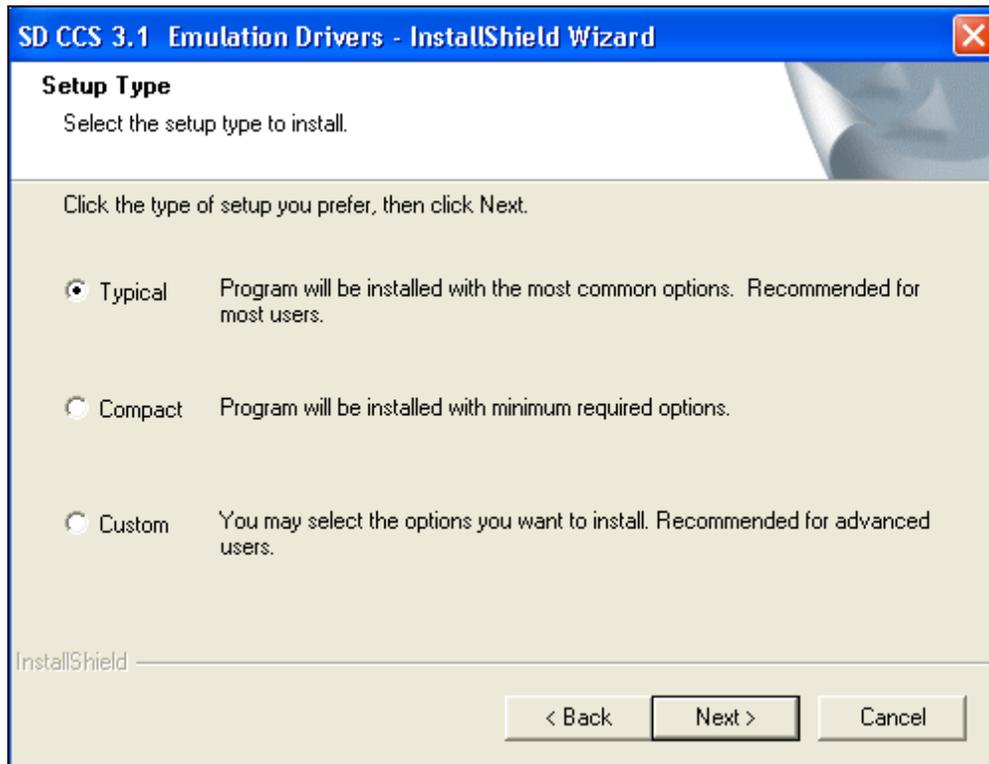


Figure 13: CCS Emulation Drivers Setup Type Selection

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*.

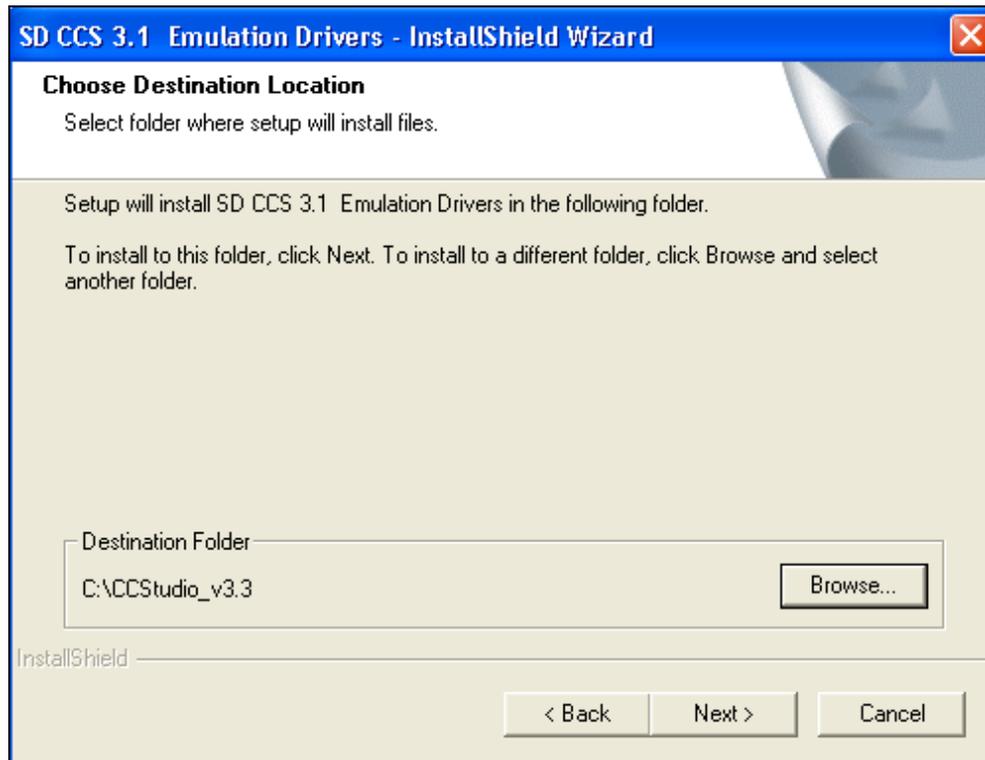


Figure 14: Selection of Destination Location for CCS 3.1 Emulation Drivers

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.

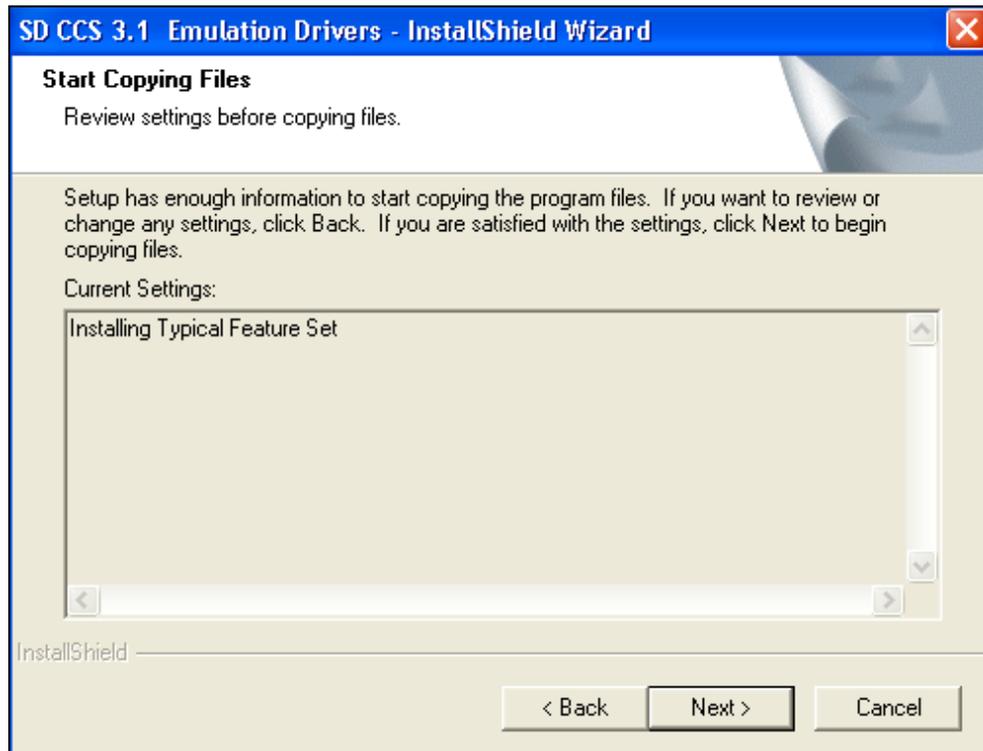


Figure 15: CCS 3.1 Emulation Drivers Installation Progress

Figure 16 presents the software installation progress window.

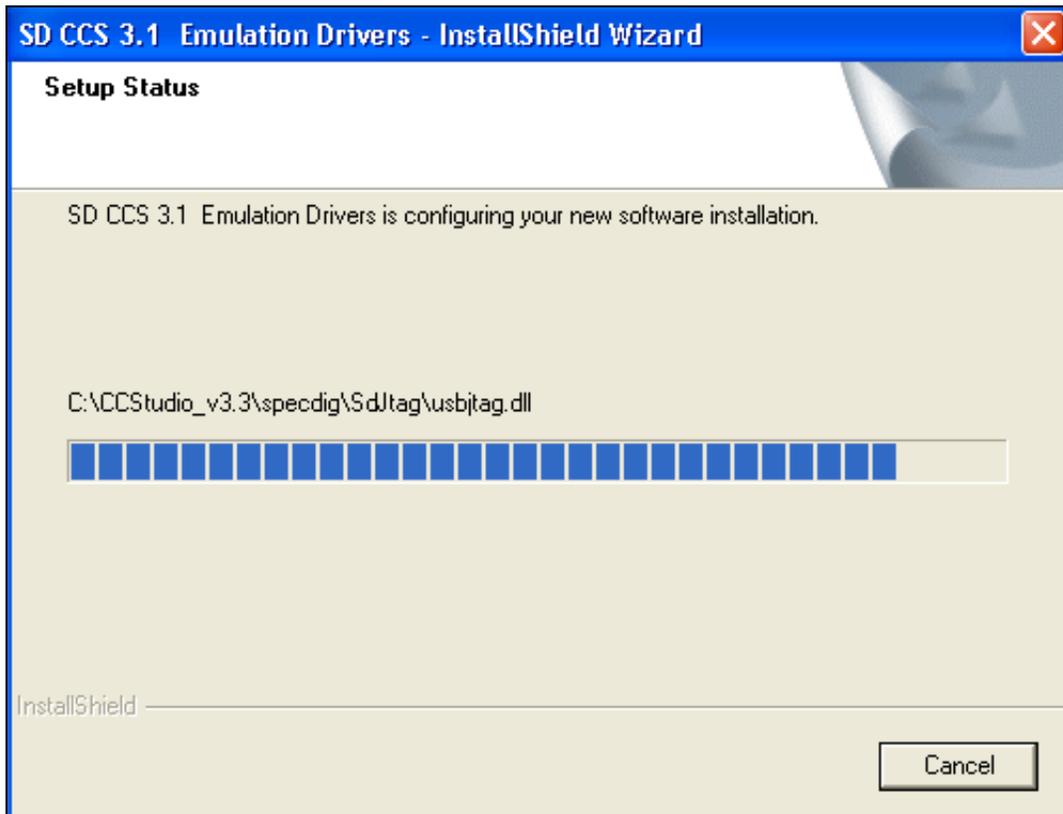


Figure 16: CCS 3.1 Emulation Drivers Installation Progress

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

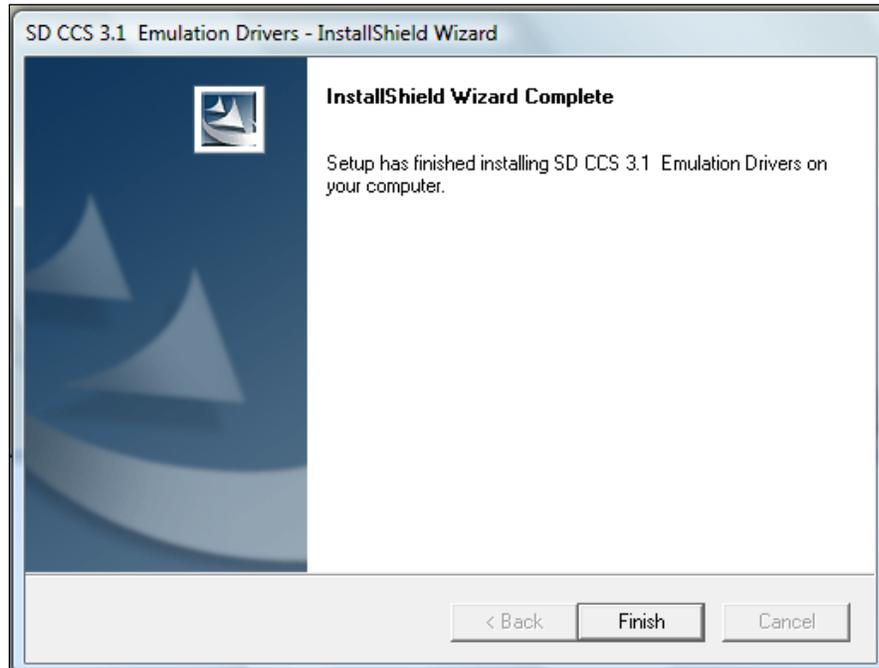


Figure 17: CCS Emulation Drivers Installation Closure

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

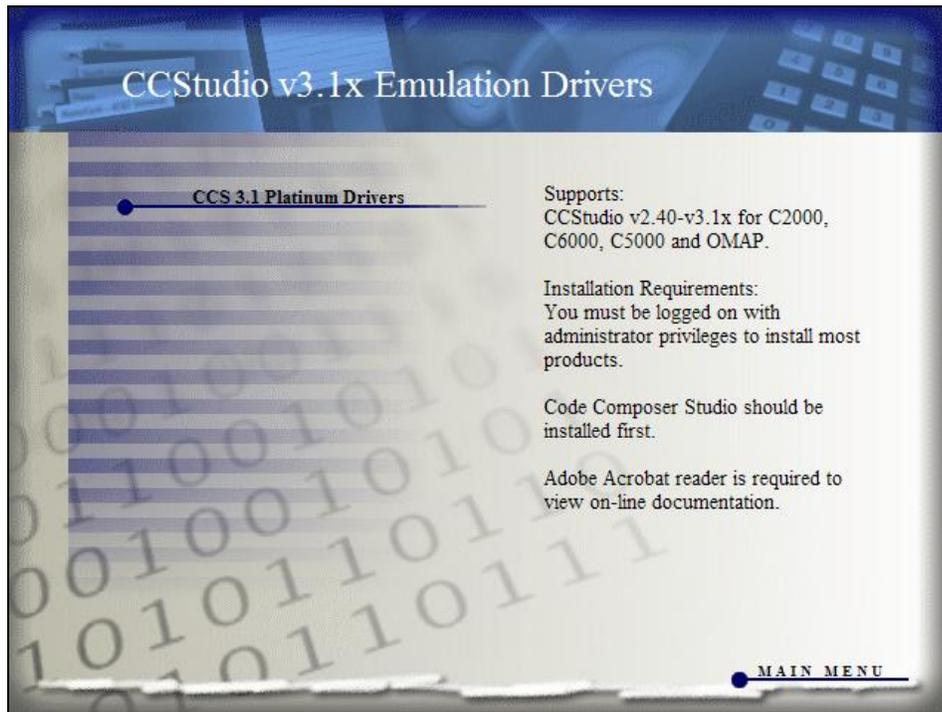


Figure 18: CCS 3.1 Planinum Driver

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

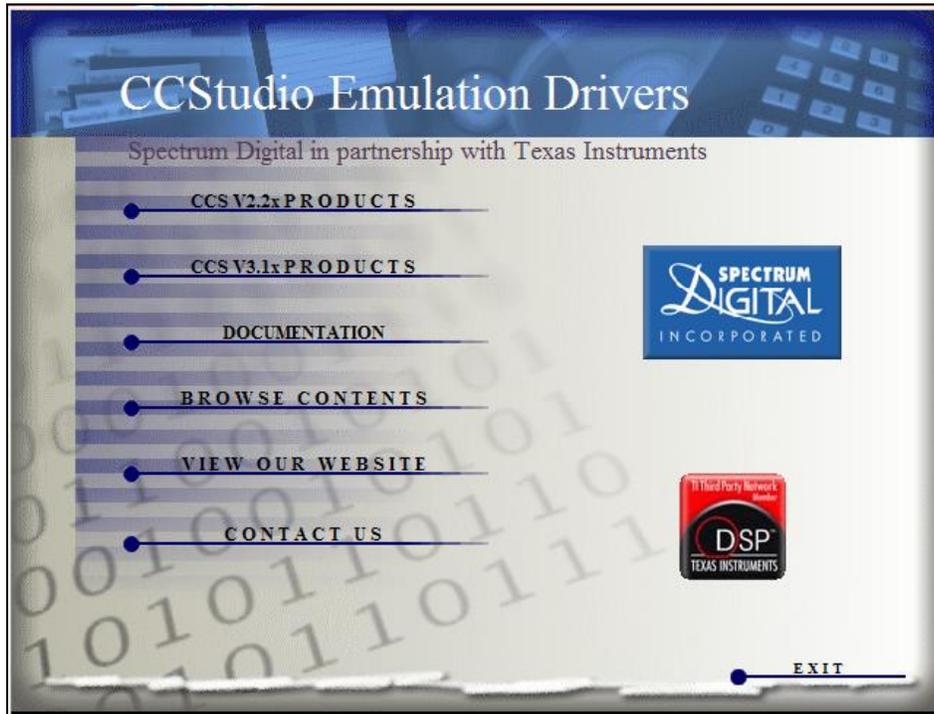


Figure 19: CCS Emulation Drivers Main Menu Window

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.

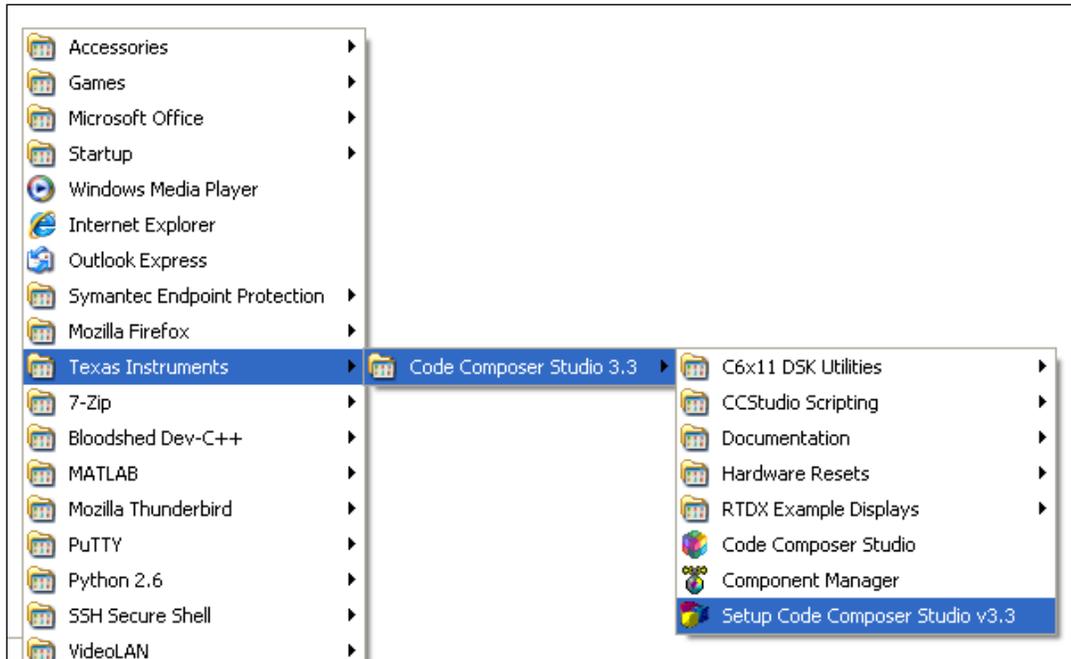


Figure 20: Location of CCS in Windows XP

The following setup window for CCS will appear:

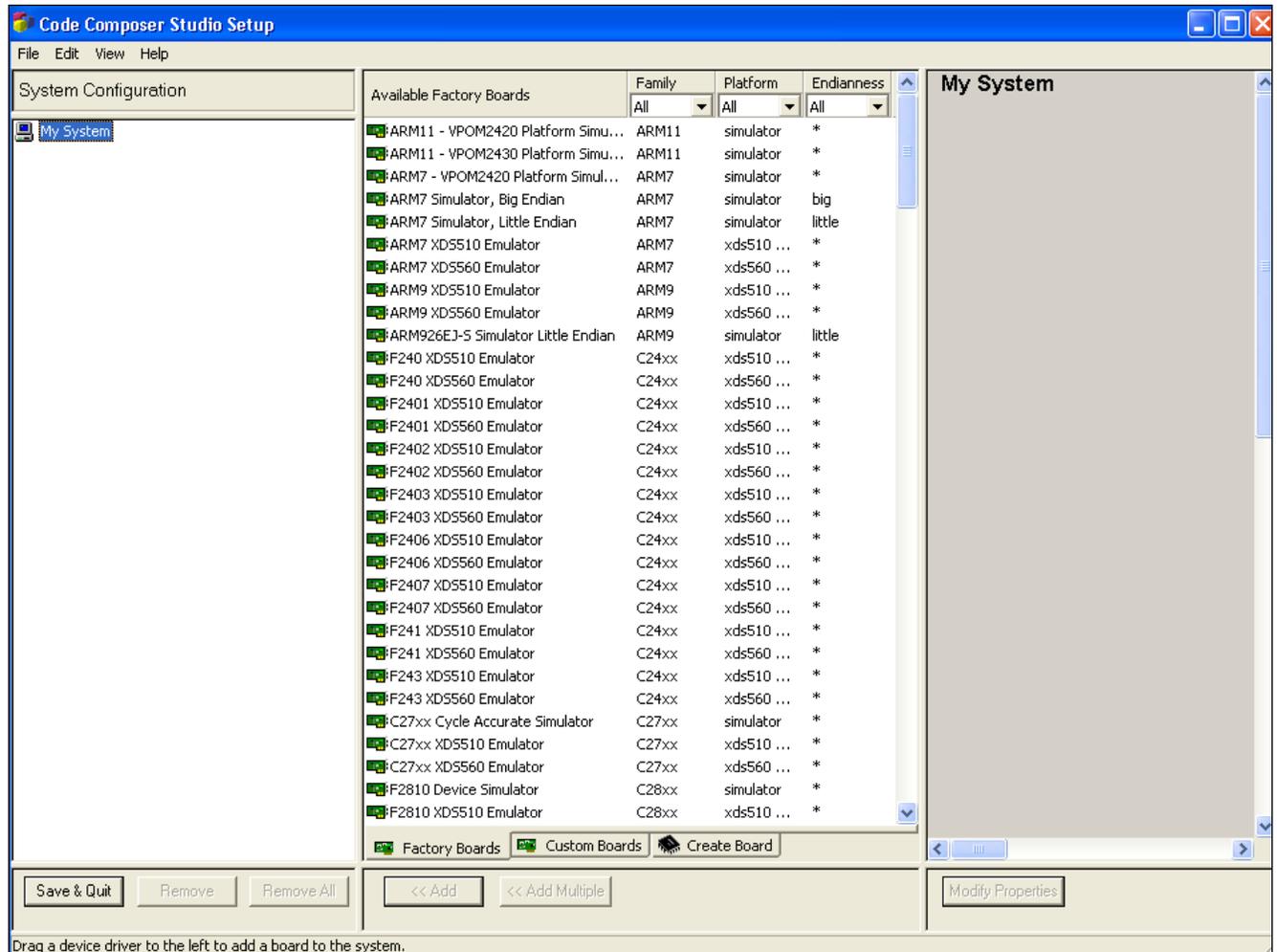


Figure 21: Code Composer Studio Setup

- 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**.

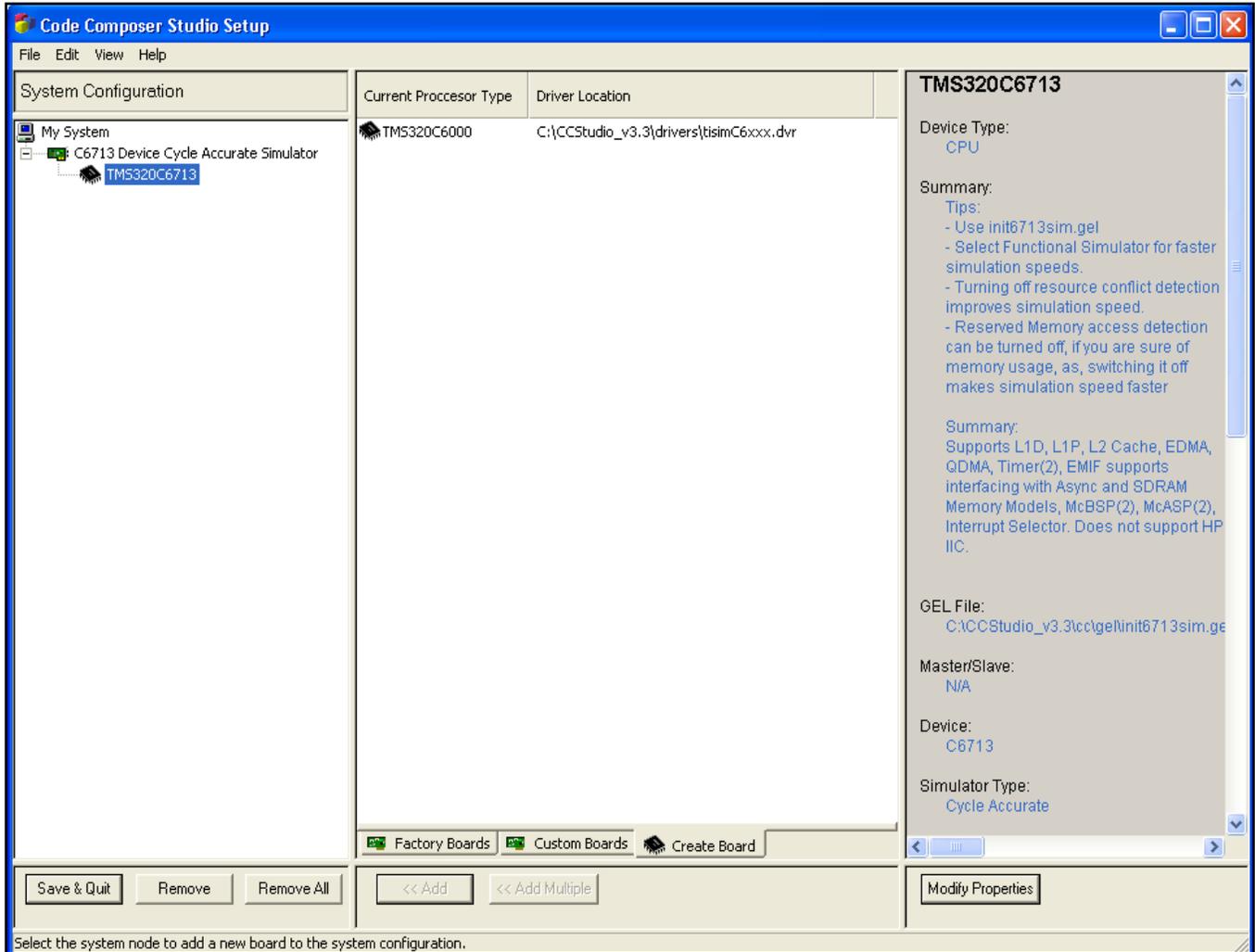


Figure 22: Selecting Simulation Environment

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*).

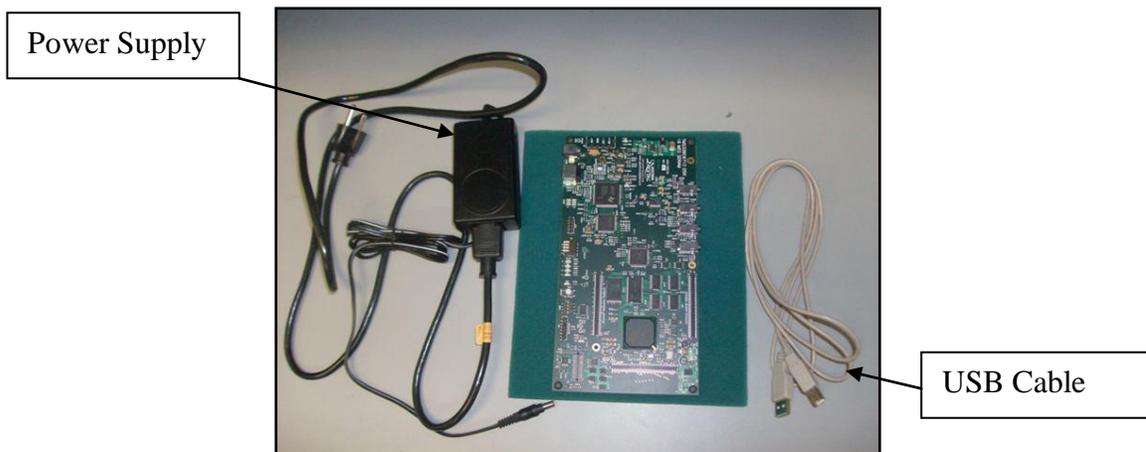


Figure 23: TMS320C6713 "Digital Starter Kit" (DSK)

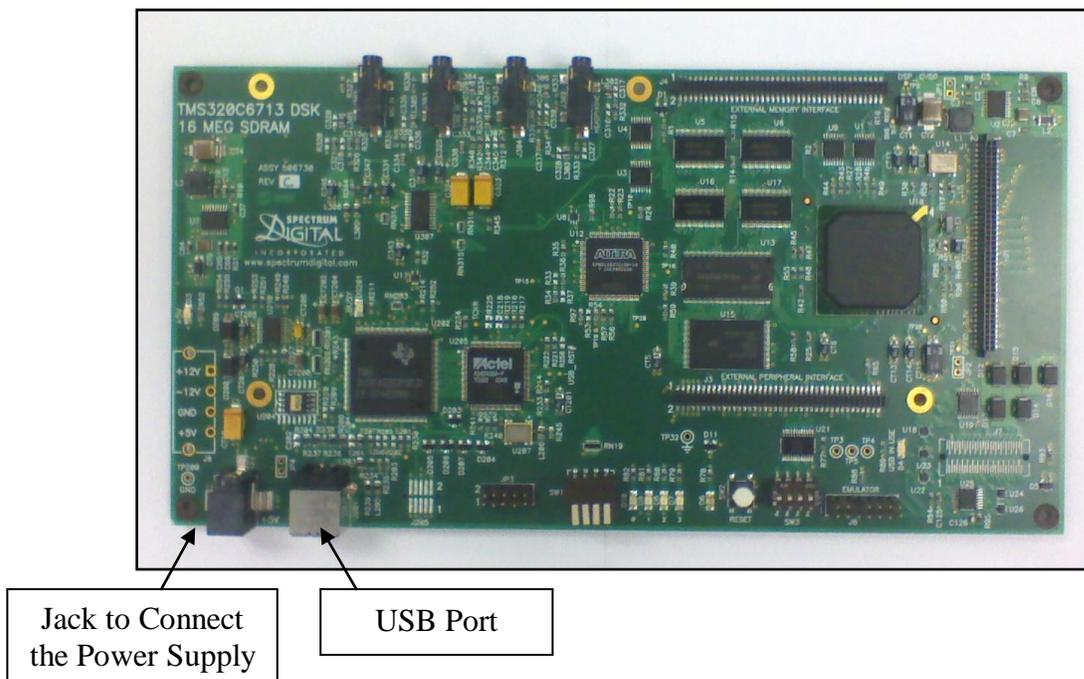


Figure 24: TMS320C6713 DSP Board

To set up the emulation environment, the user should access the Setup Code Composer Studio v3.3 tool by going to *All Programs* → *Texas Instruments* → *Code Composer Studio 3.3* → *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**.

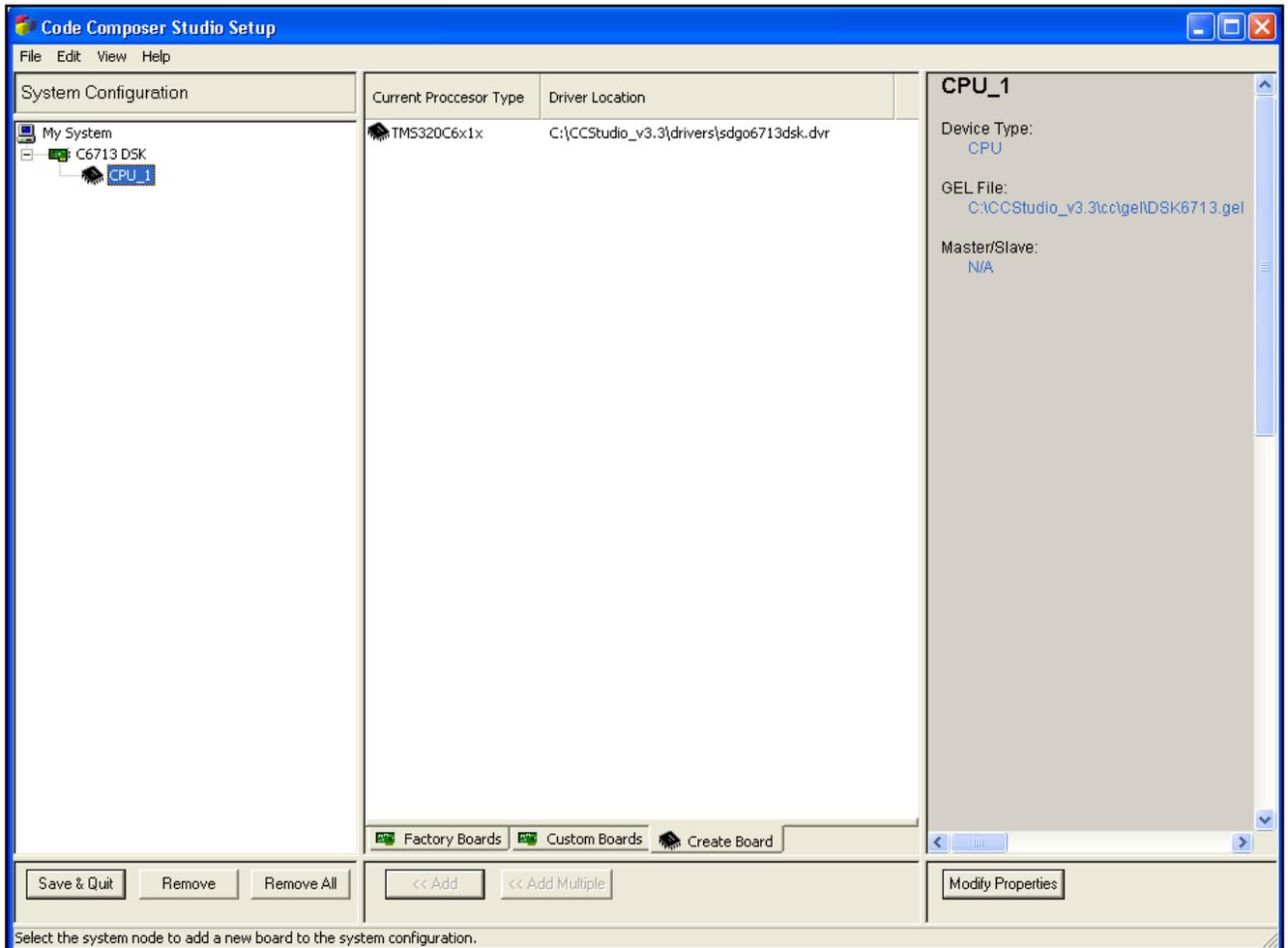


Figure 25: Emulation Environment Selection

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** → **Connect**, in order to establish connection with the board.

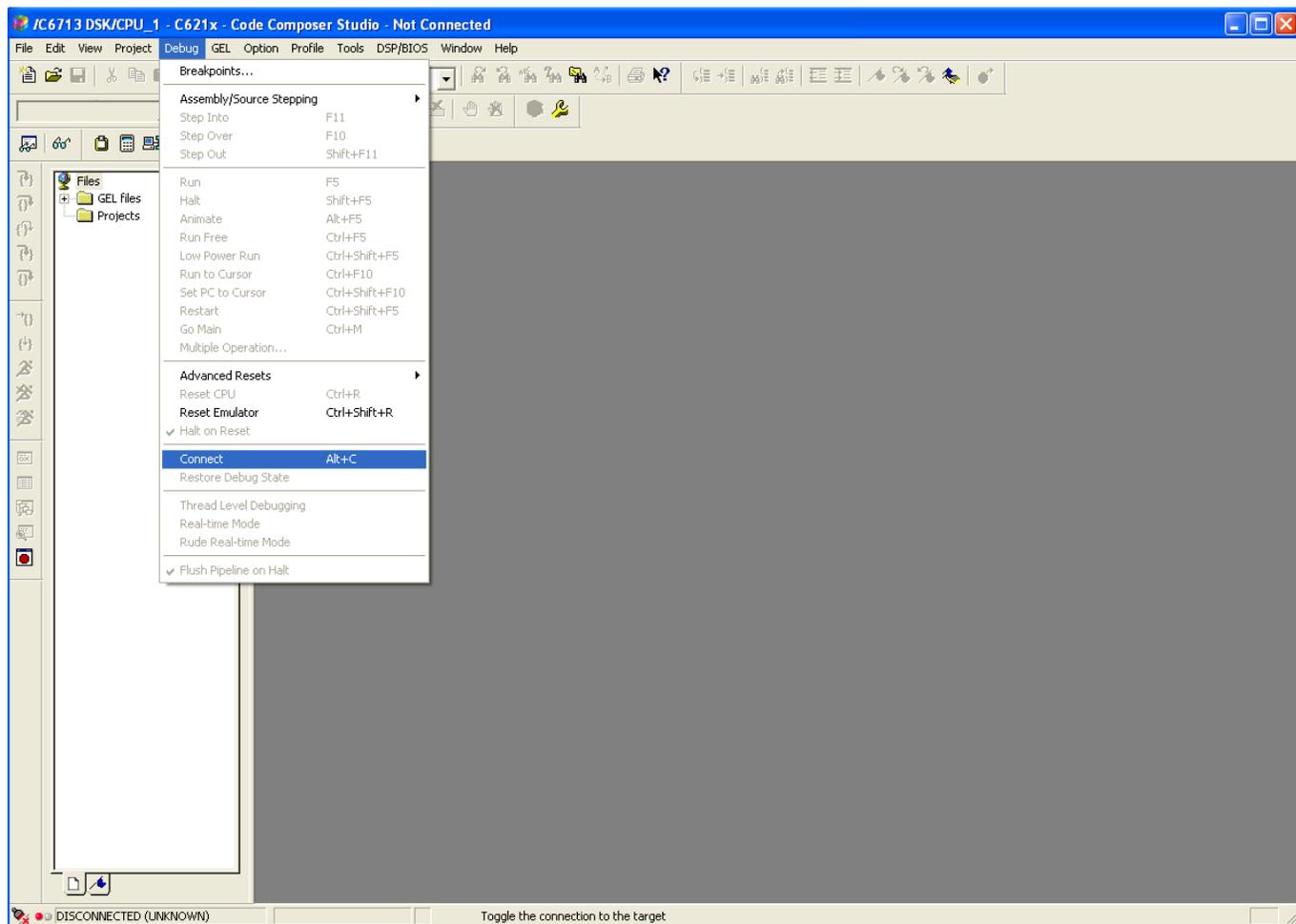


Figure 26: Establish the Connection between the CCS and the TMS320C6713 DSP

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 needed for the program.
2. Build your project → 
3. Download the project to the board.
4. Run the project → 
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:

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

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 (.pj) 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.

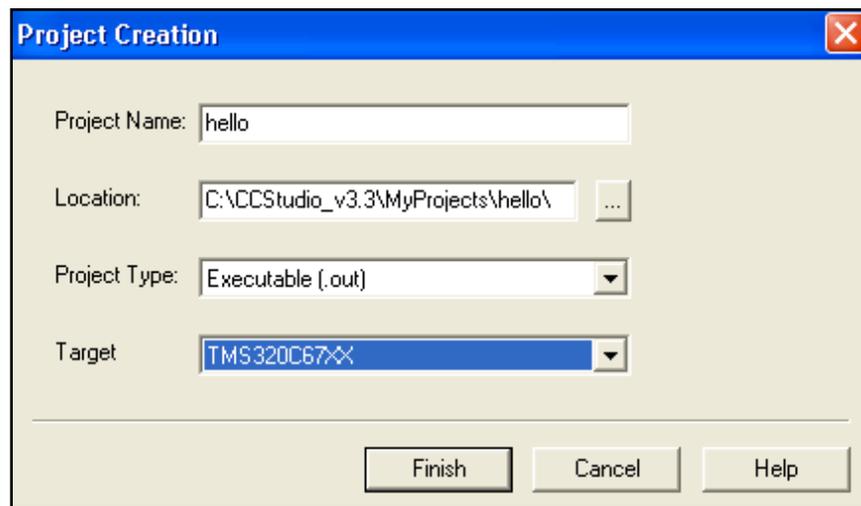


Figure 27: Window for the creation of a New Project

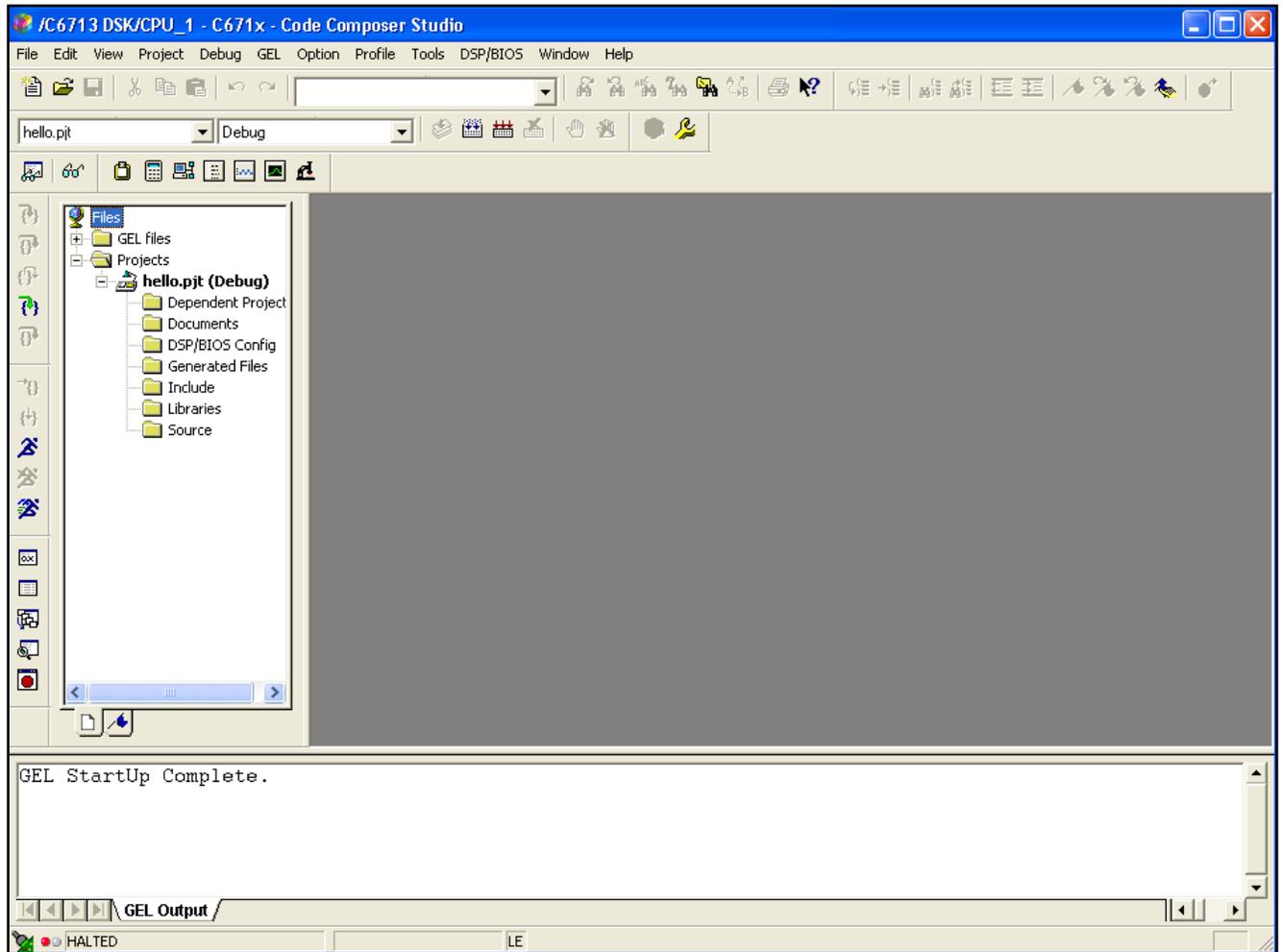


Figure 28: Project Folders

2. 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**.

C source code:

```
#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 **cs16713.lib**.
6. Select **Project** → **Scan All Files Dependencies**. Verify that all the files that are shown in the **Figure 29** were added to the project.

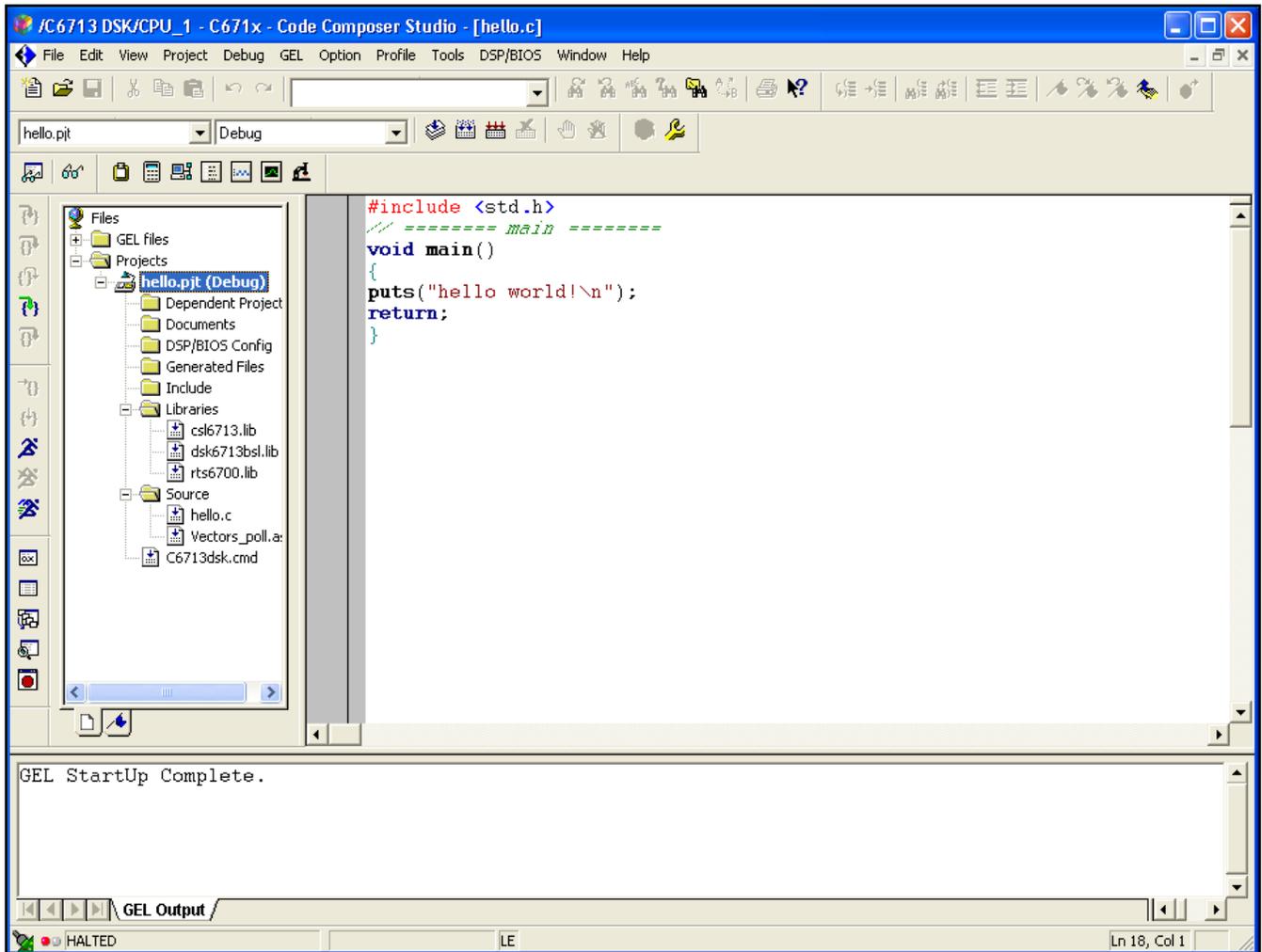


Figure 29: Project Files

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

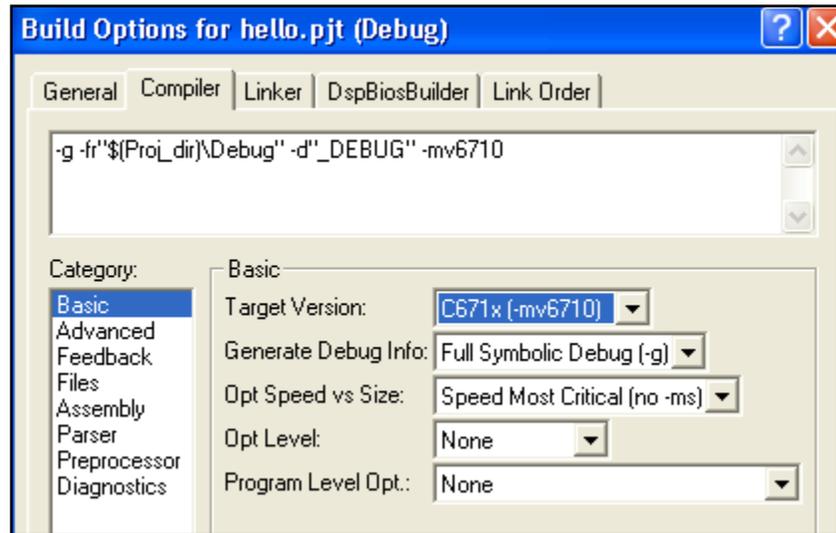


Figure 30: Setting the Target Version

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.

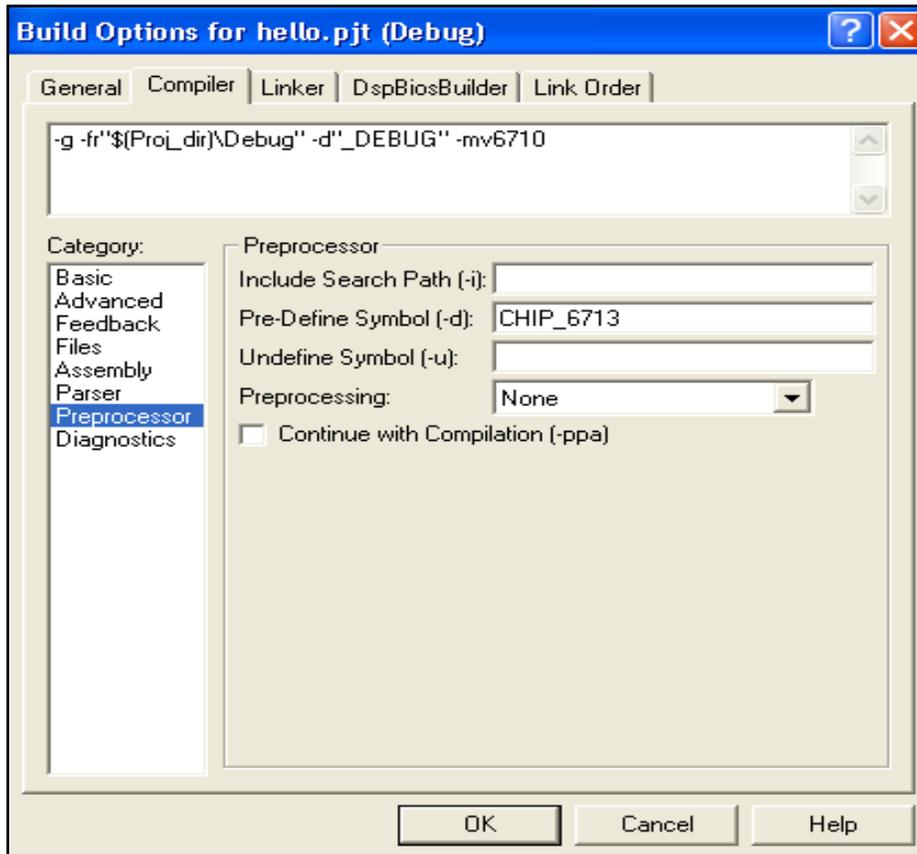


Figure 31: Specifying the Chip Architecture

10. Under *Linker* → *Libraries*:

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

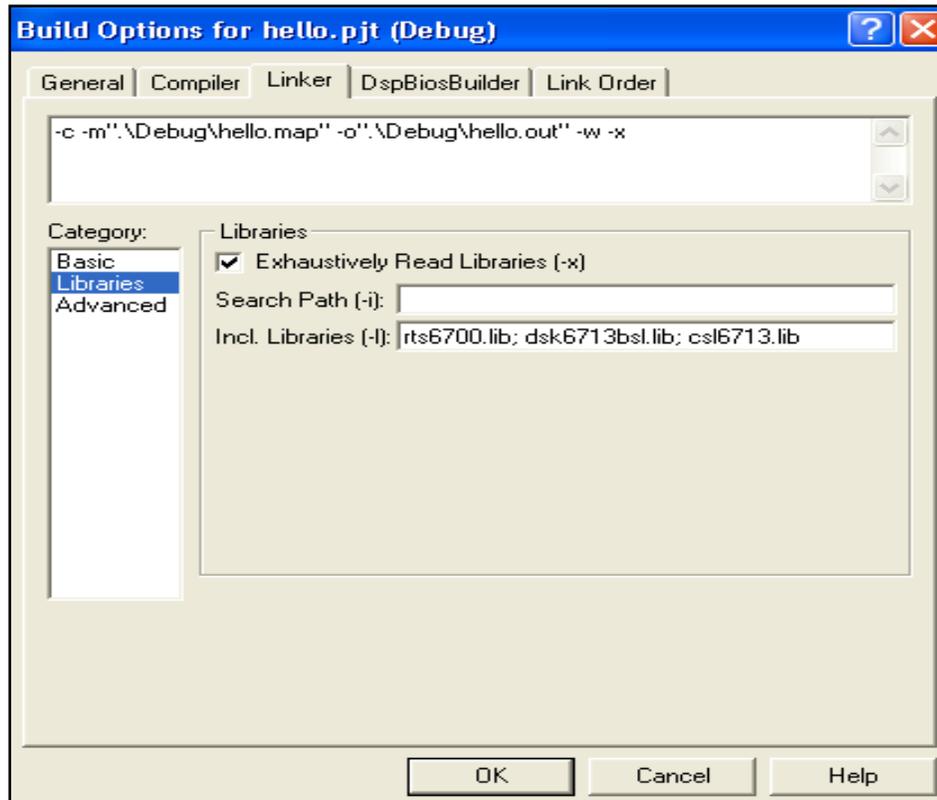


Figure 32: Libraries Needed for the Project

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

Compiling and Debugging the Project

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.

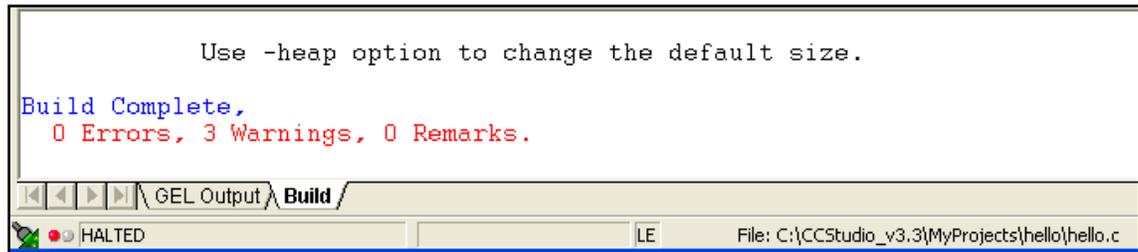


Figure 33: Compiling Results

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.

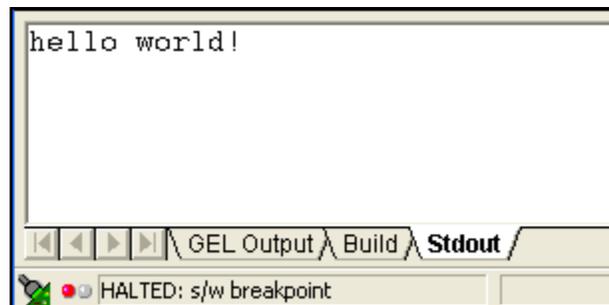


Figure 34: Results Obtained after Run the Algorithm "hello world"

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 Platinum
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**.

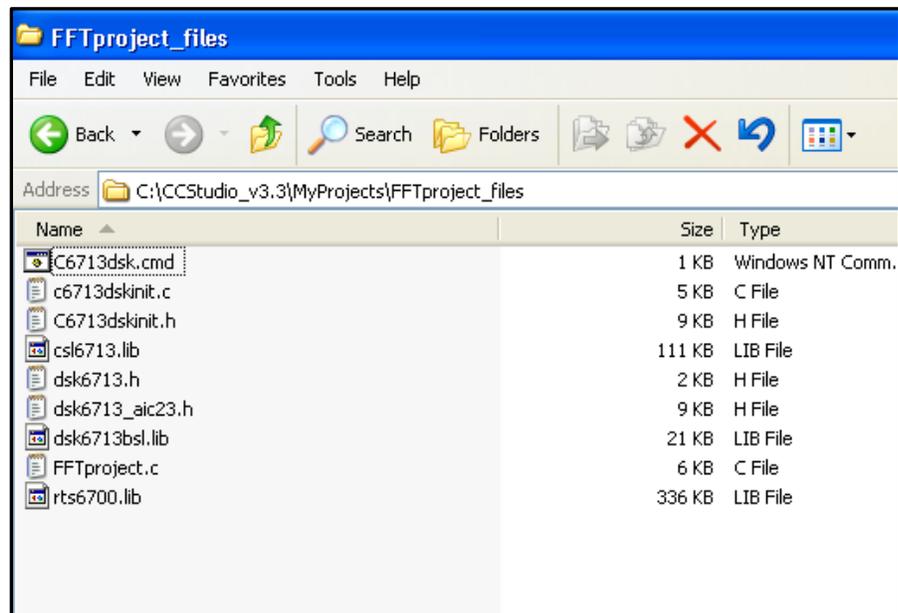


Figure 35: FFT Files

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**.

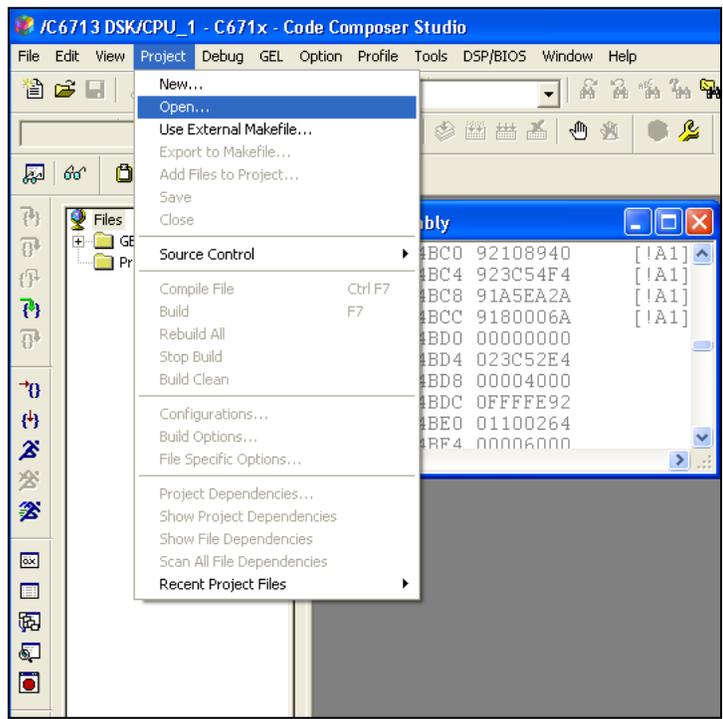


Figure 36: Open FFT Project

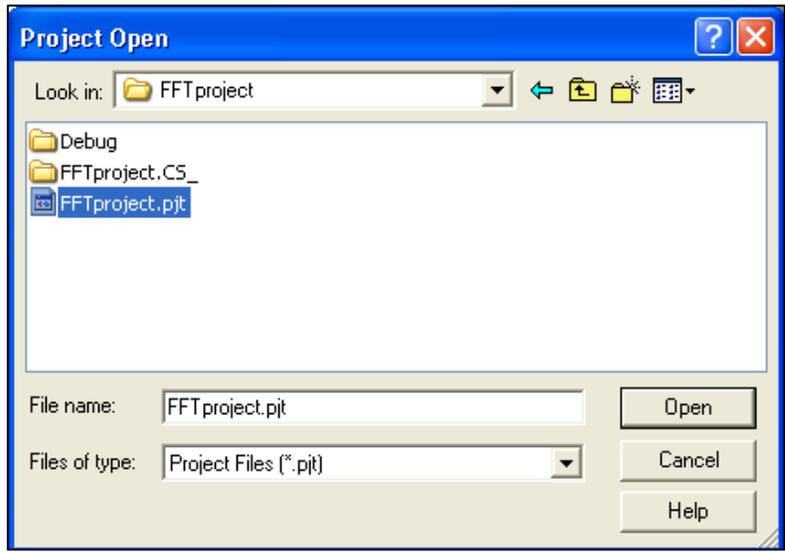


Figure 37: FFT Project Selection

2. 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 look like **Figure 38** and should have all the files that are on the left side.

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

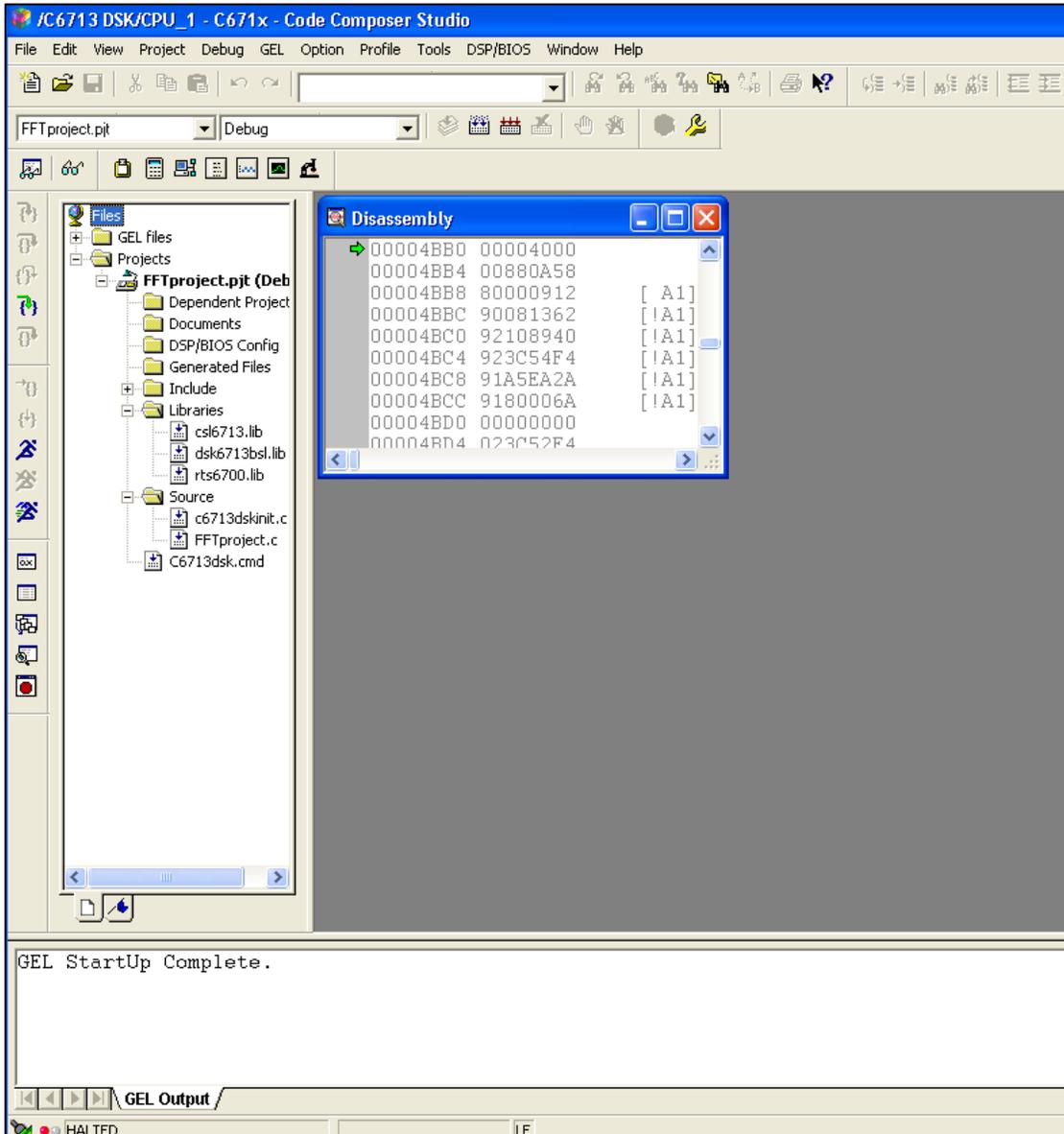


Figure 38: CCS Environment for FFT Example

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.

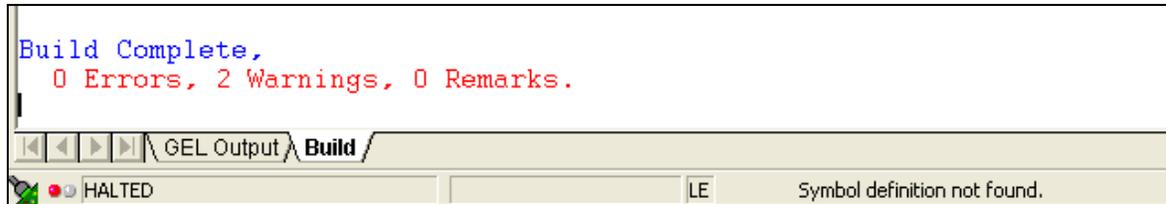


Figure 39: FFT Project Compiling Results

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`.

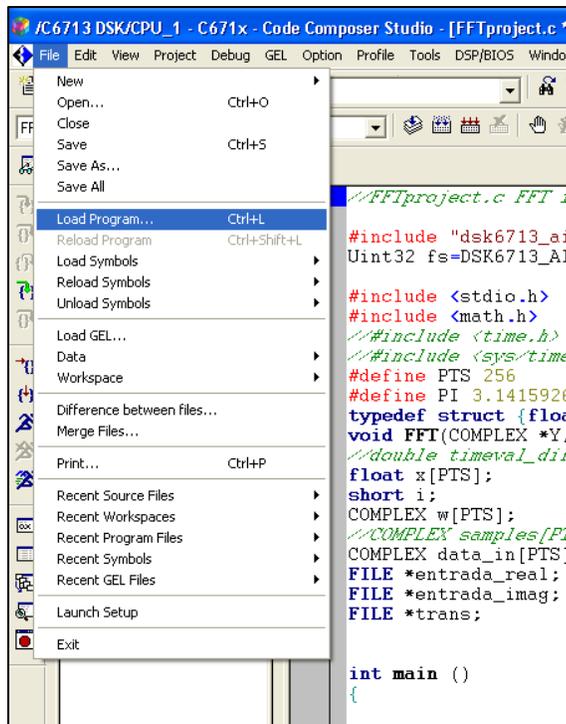


Figure 40: “Load Program” Location

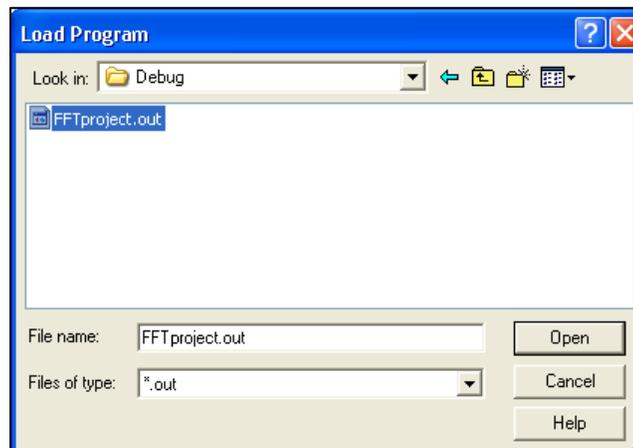


Figure 41: “FFTproject.out” File Location

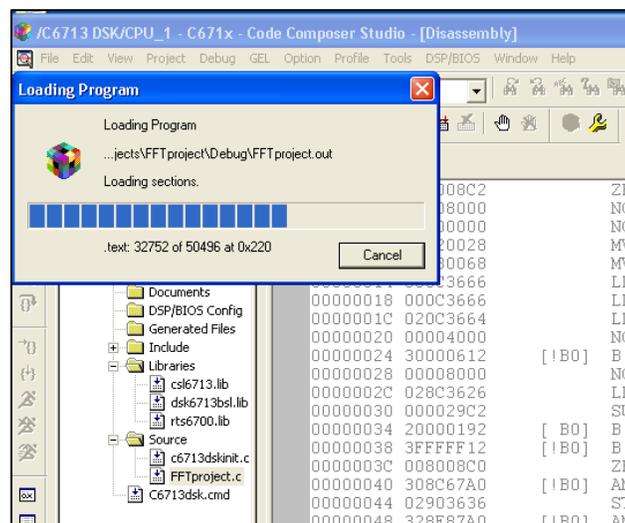


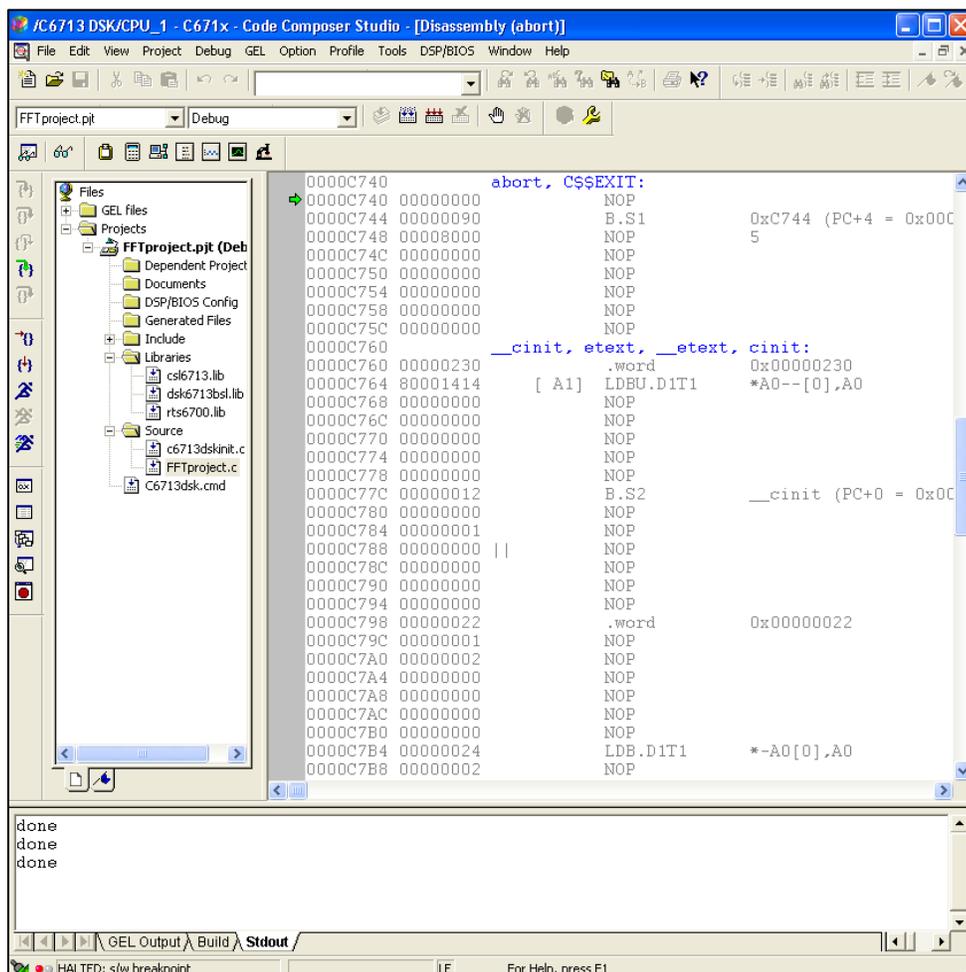
Figure 42: Downloading the “FFTproject.out” File to the TMS320C6713 DSP

4. Click on the “run” button  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.



The screenshot displays the Code Composer Studio interface. The main window shows assembly code for the 'abort, C\$EXIT:' function. The code consists of several NOP instructions and a B.S1 instruction. The address range is from 0000C740 to 0000C7B8. The left pane shows the project structure for 'FFTproject.pjt'. The bottom pane shows the 'Stdout' window with the output 'done' printed three times.

```
0000C740 abort, C$EXIT:
0000C740 00000000 NOP
0000C744 00000090 B.S1 0xC744 (PC+4 = 0x00C748)
0000C748 00008000 NOP
0000C74C 00000000 NOP
0000C750 00000000 NOP
0000C754 00000000 NOP
0000C758 00000000 NOP
0000C75C 00000000 NOP
0000C760 __cinit, etext, __etext, cinit:
0000C760 00000230 .word 0x00000230
0000C764 80001414 [ A1] LDBU.D1T1 *-A0--[0],A0
0000C768 00000000 NOP
0000C76C 00000000 NOP
0000C770 00000000 NOP
0000C774 00000000 NOP
0000C778 00000000 NOP
0000C77C 00000012 B.S2 __cinit (PC+0 = 0x00C780)
0000C780 00000000 NOP
0000C784 00000001 NOP
0000C788 00000000 ||
0000C78C 00000000 NOP
0000C790 00000000 NOP
0000C794 00000000 NOP
0000C798 00000022 .word 0x00000022
0000C79C 00000001 NOP
0000C7A0 00000002 NOP
0000C7A4 00000000 NOP
0000C7A8 00000000 NOP
0000C7AC 00000000 NOP
0000C7B0 00000000 NOP
0000C7B4 00000024 LDB.D1T1 *-A0[0],A0
0000C7B8 00000002 NOP
```

done
done
done

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 Platinum
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**.

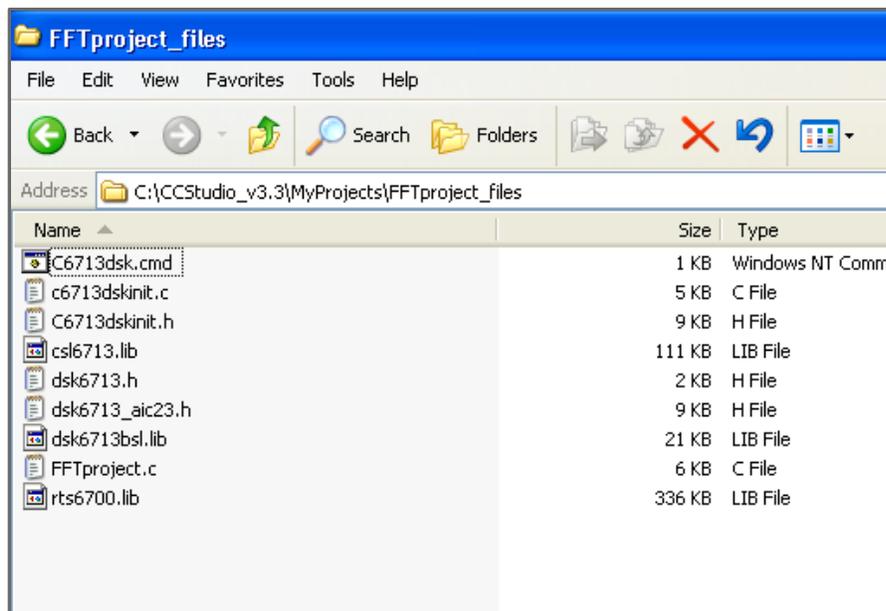


Figure 44: FFT Project Files

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 (.pj) 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.

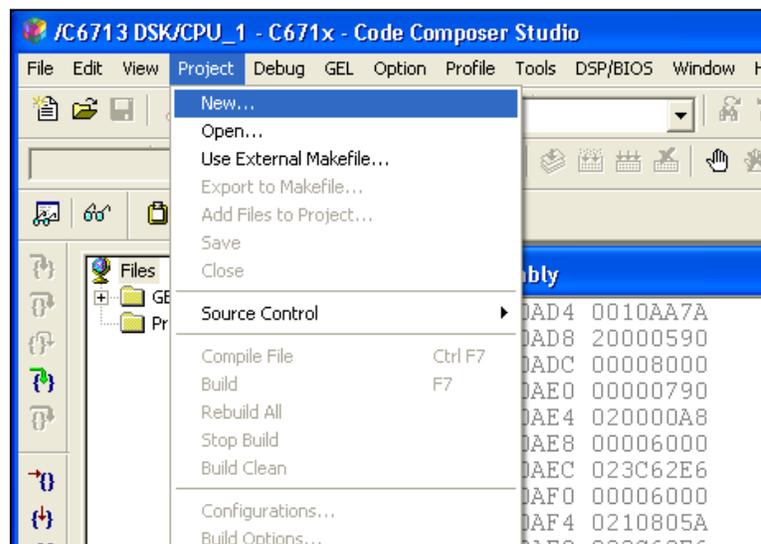


Figure 45: Creating a New Project

Verify if the following Project option is selected:

Target → **TMS320C67XX**, and then click **Finish**.

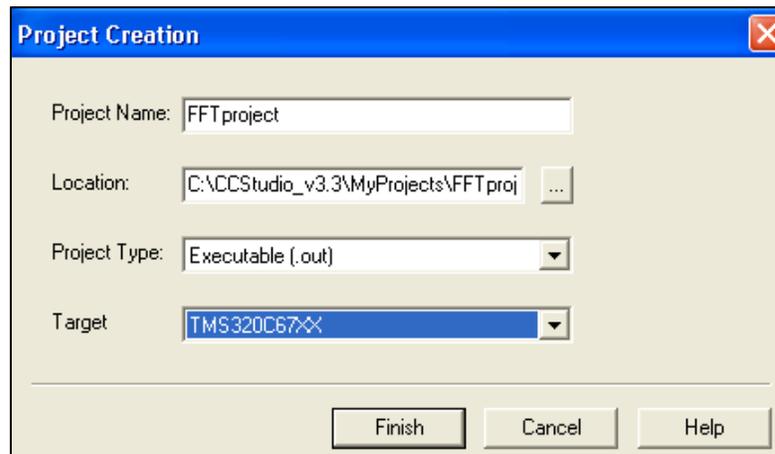


Figure 46: Window for the Creation of a New Project

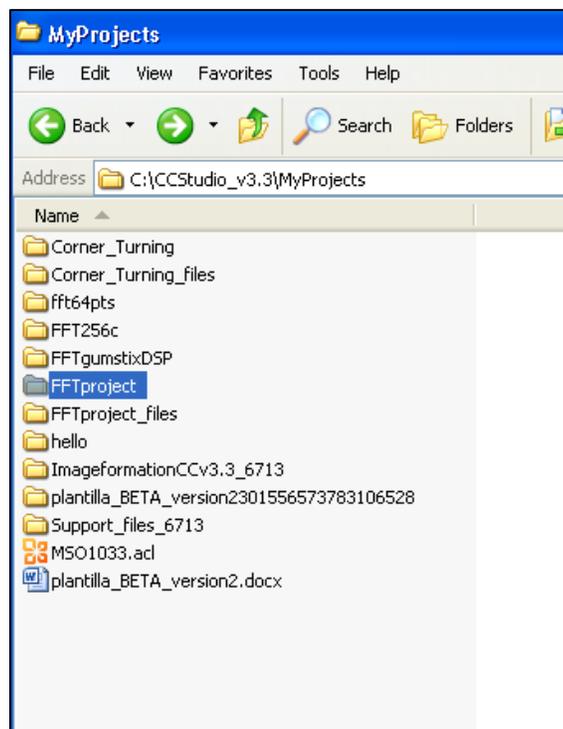


Figure 47: FFT Project Folder

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

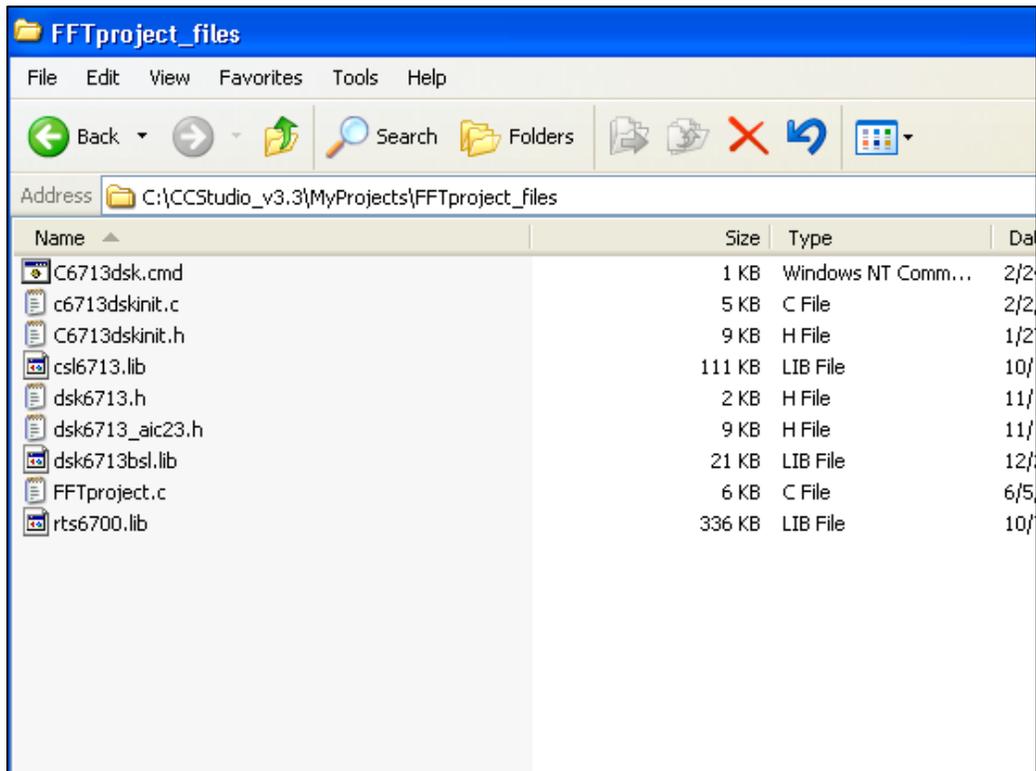


Figure 48: FFT Project Files

3. Select **Project** → **Add files to project**. Add the following files:

- C6713dsk.cmd
- C6713dskinit.c
- FFTproject.c
- csl6713.lib
- rts6700.lib
- dsk6713bsl.lib

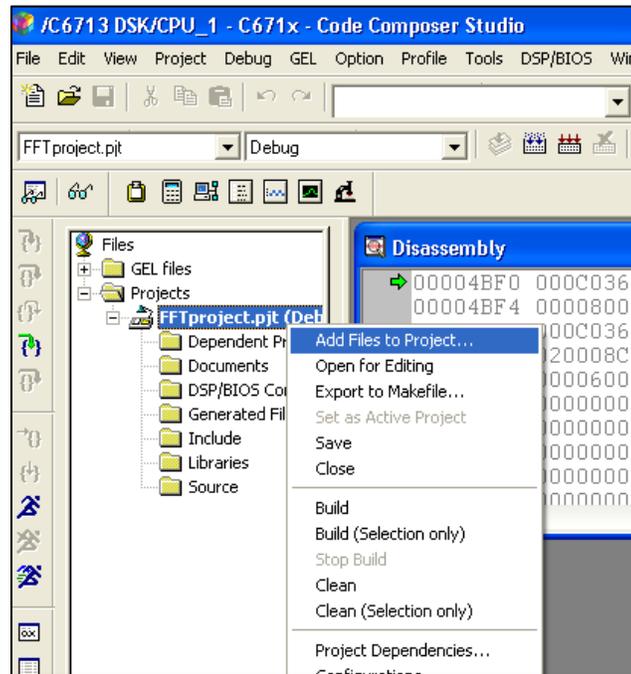


Figure 49: Adding Files to the Project

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

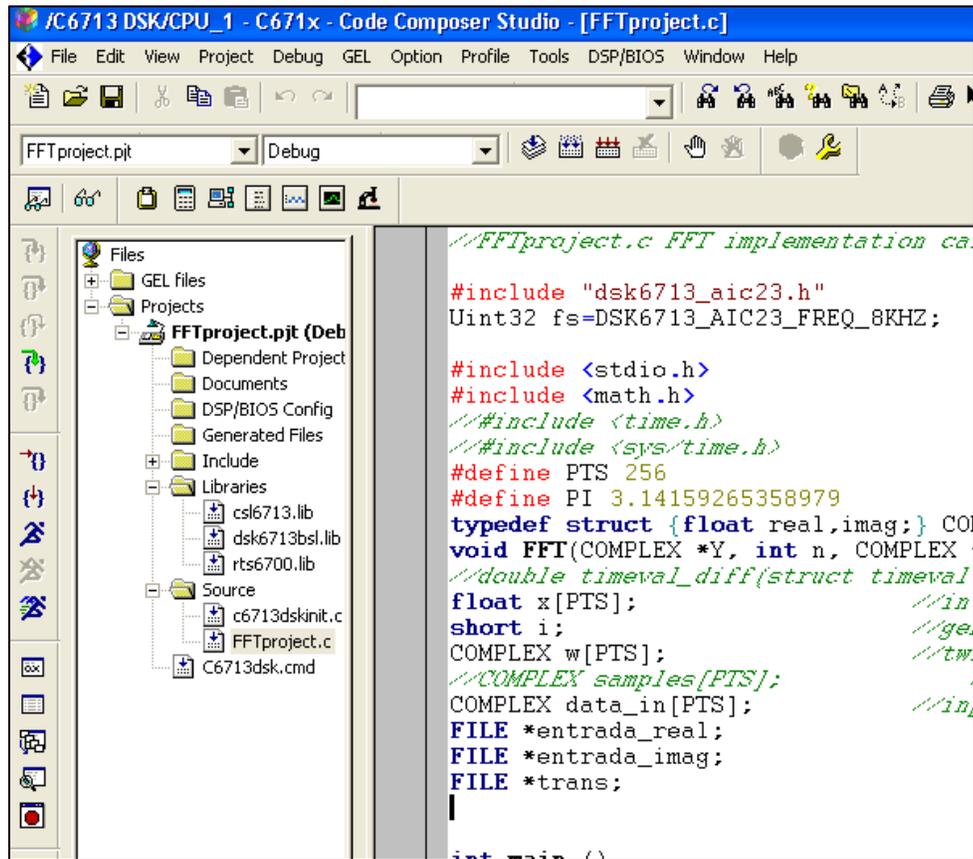


Figure 50: Project Files

- 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 be chosen or written, and the option OK is selected after all settings are verified.

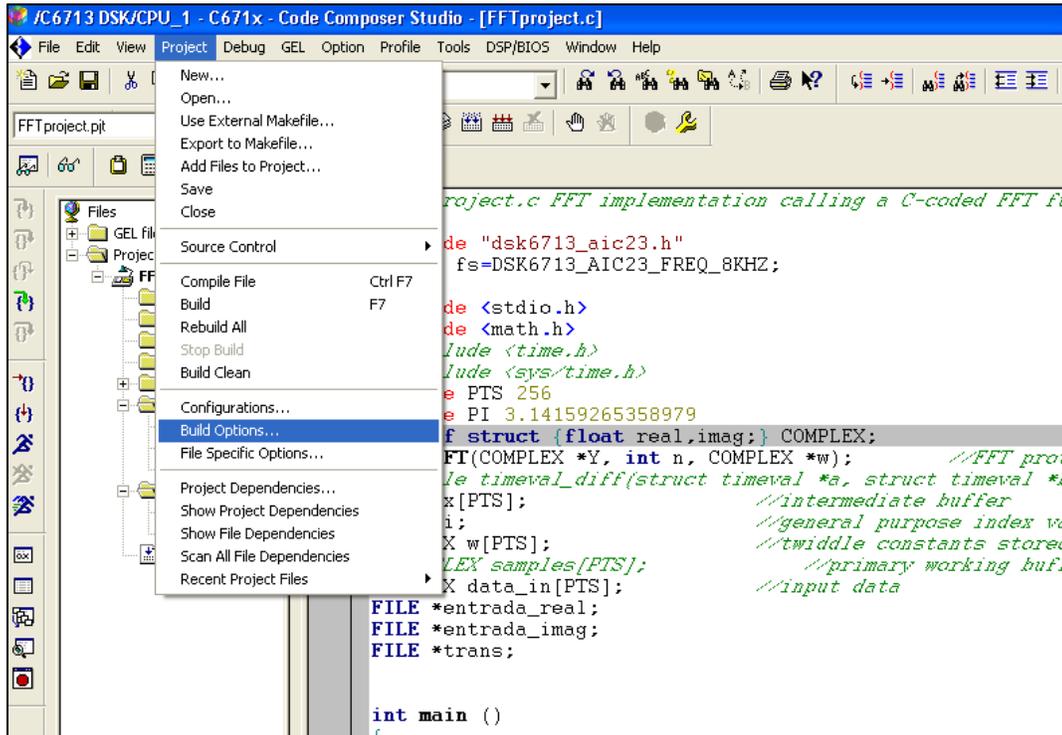


Figure 51: Build Option Setting Location

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

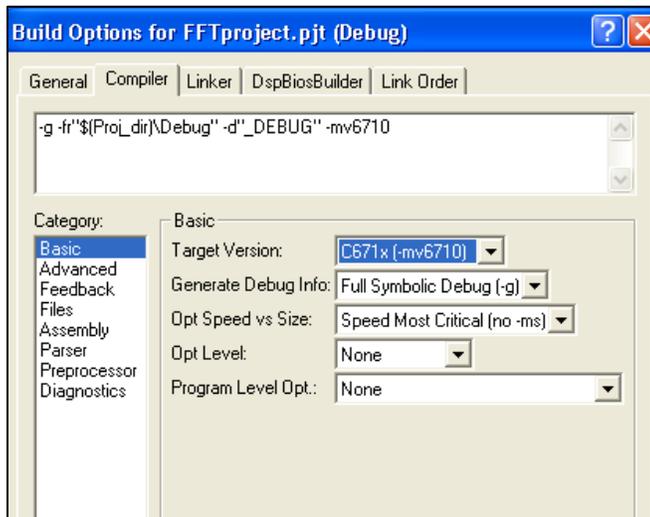


Figure 52: Setting the Target Version

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

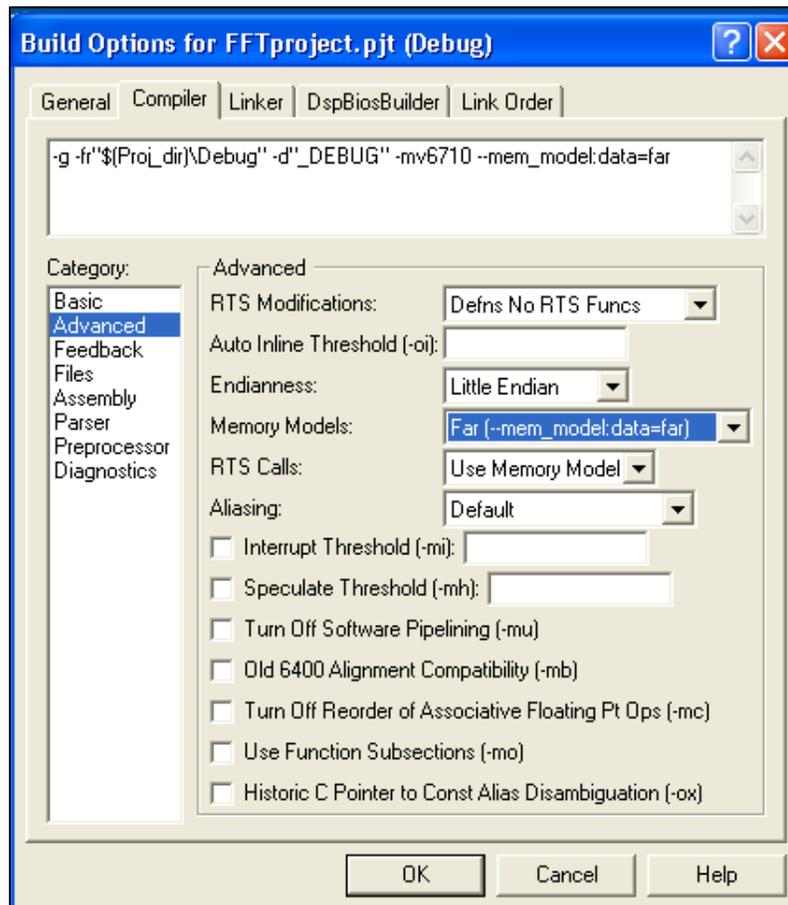


Figure 53: Memory Model Type Selection

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.

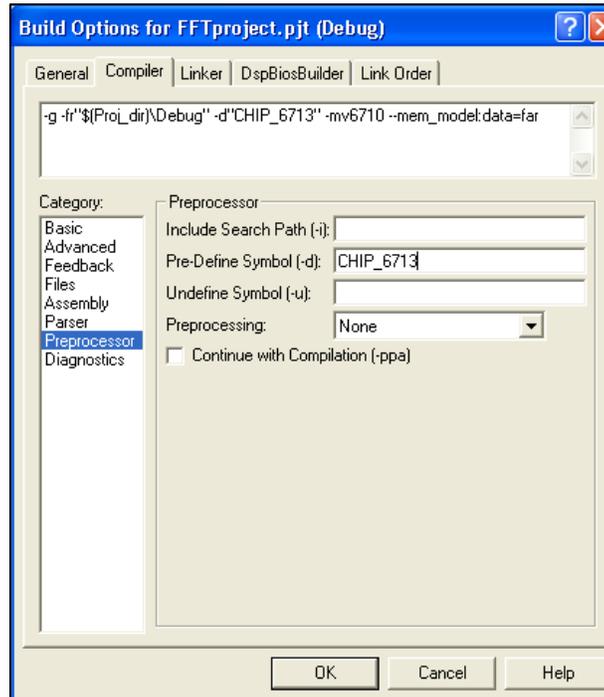


Figure 54: Specifying the Chip Architecture

9. Under *Linker* → *Libraries*:
 - In *Included Libraries (-l)*, these libraries must be specified: **rts6700.lib**;
dsk6713bsl.lib; **cs16713.lib**

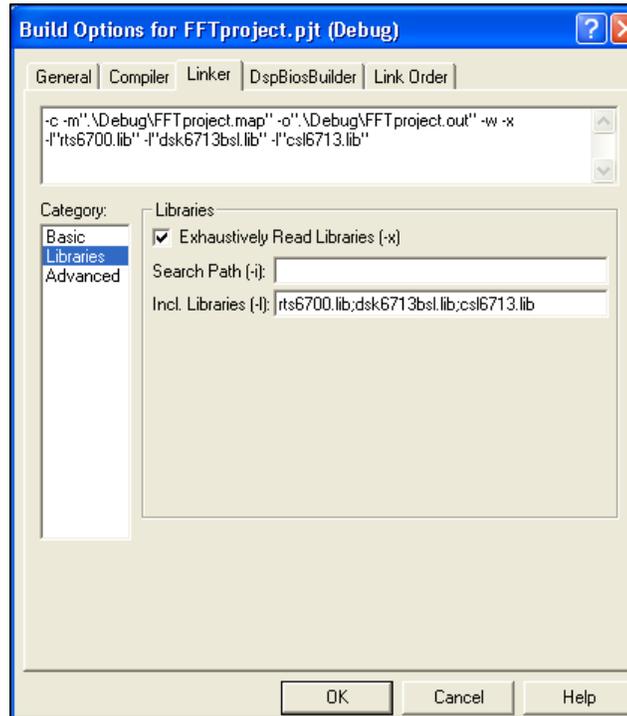


Figure 55: Libraries Needed for the Project

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.

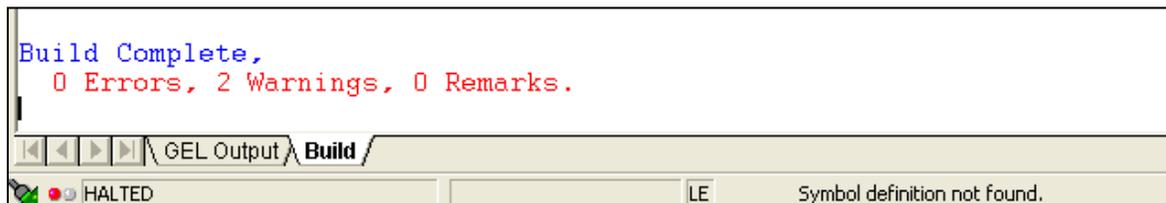


Figure 56: FFT Project Compiling Results

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**.

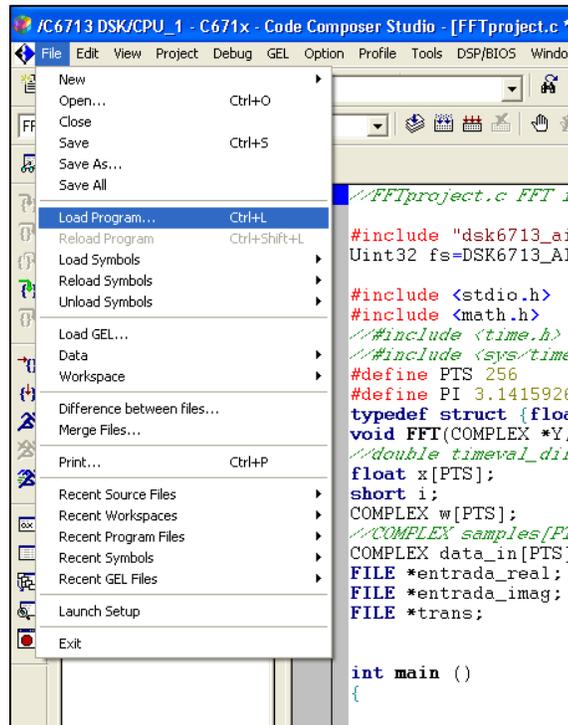


Figure 57: “Load Program” Location

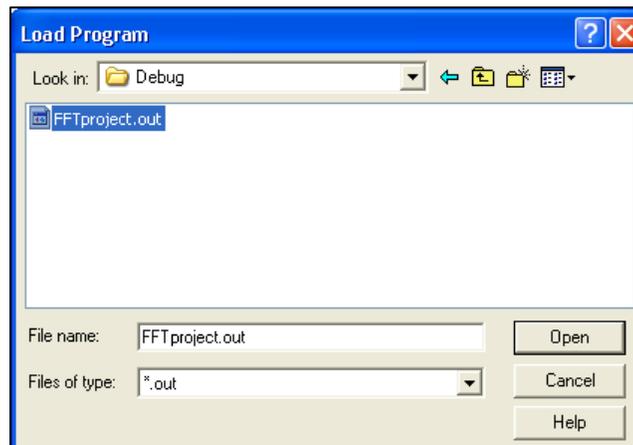


Figure 58: “FFTproject.out” File Location

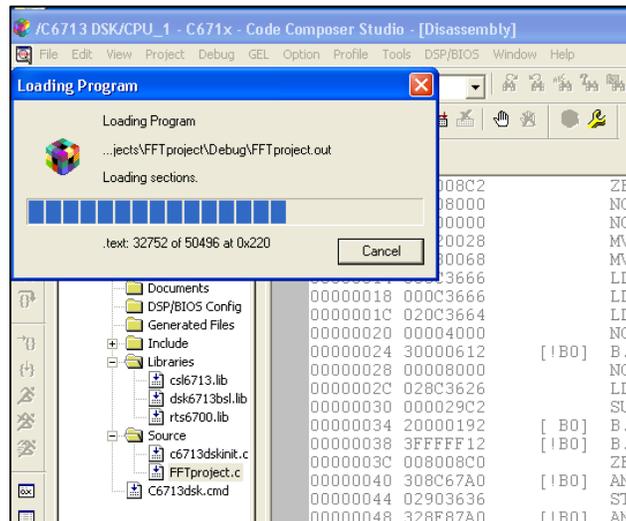


Figure 59: Downloading the “FFTproject.out” File to the TMS320C6713 DSP

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.

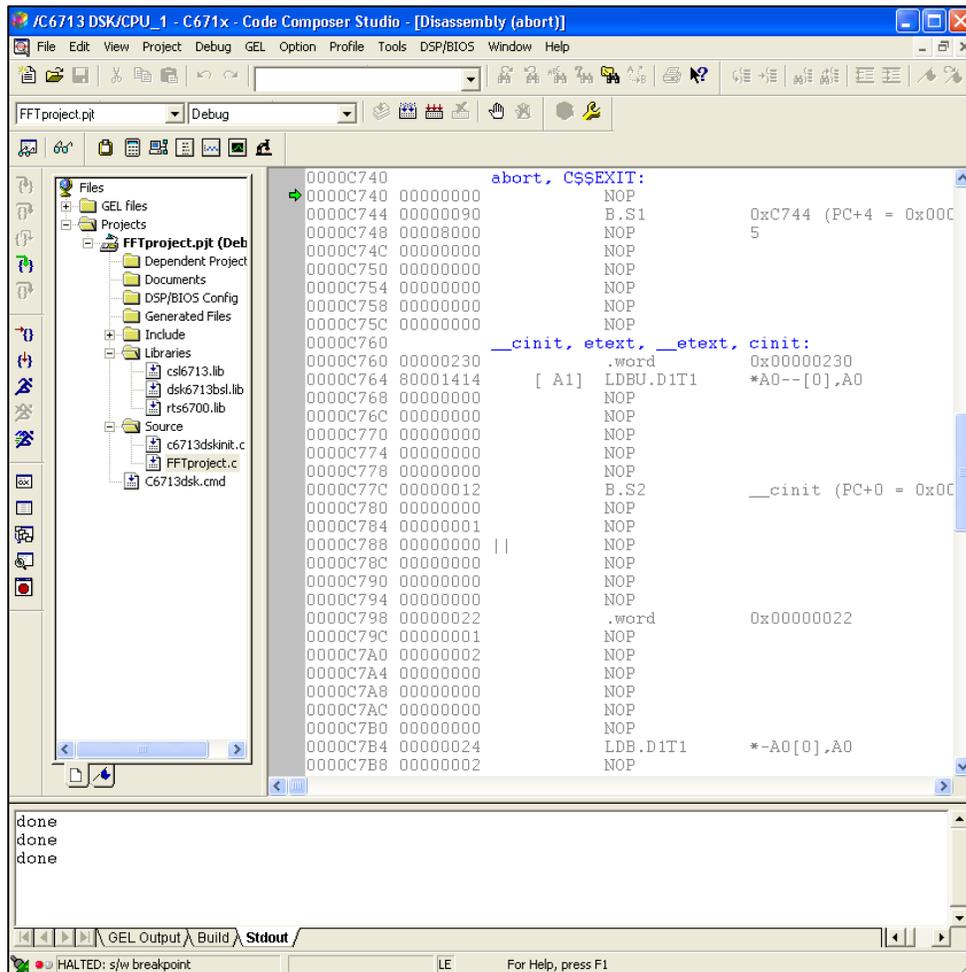


Figure 60: Results Obtained after Run the FFT Algorithm.

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 Platinum
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**

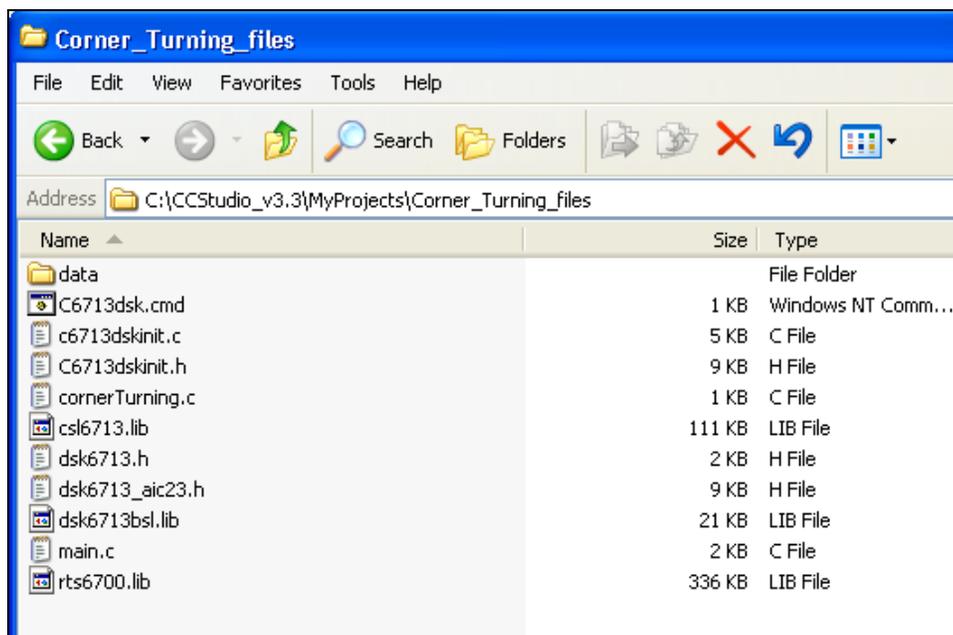


Figure 61: Corner Turning Files

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**.

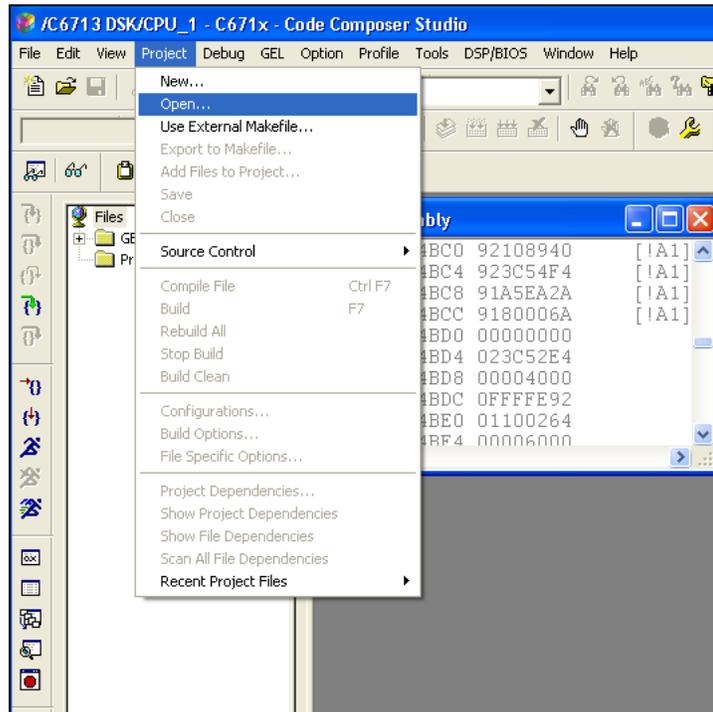


Figure 62: Open "Corner Turning" Project

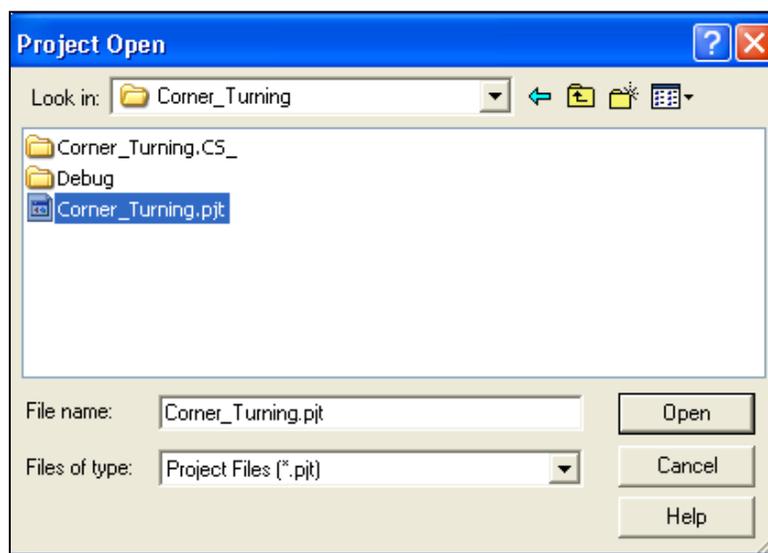


Figure 63: Corner_Turning Project Selection

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** → **Build Options**, are correct (Steps from 7 to 11)

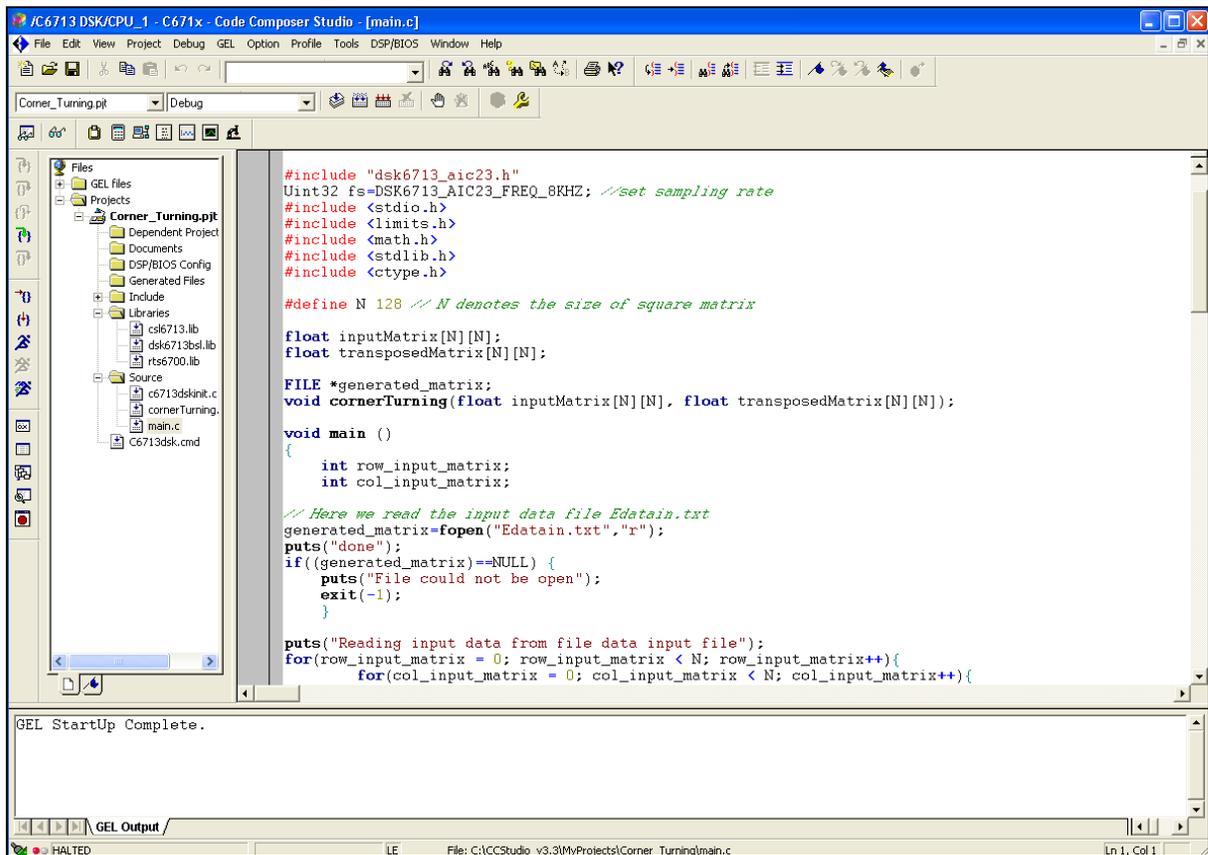


Figure 64: CCS Environment for Corner Turning Example

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.

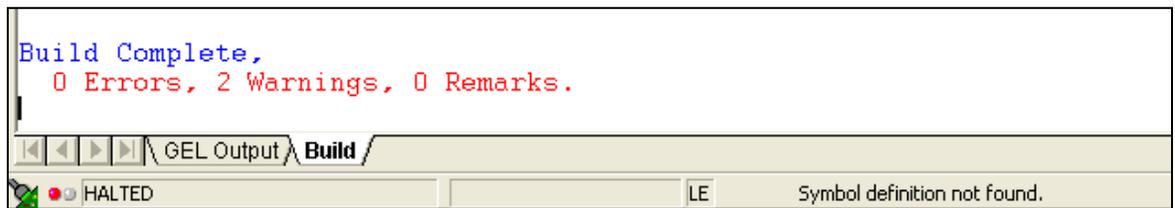


Figure 65: Corner Turning Compiling Results

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.

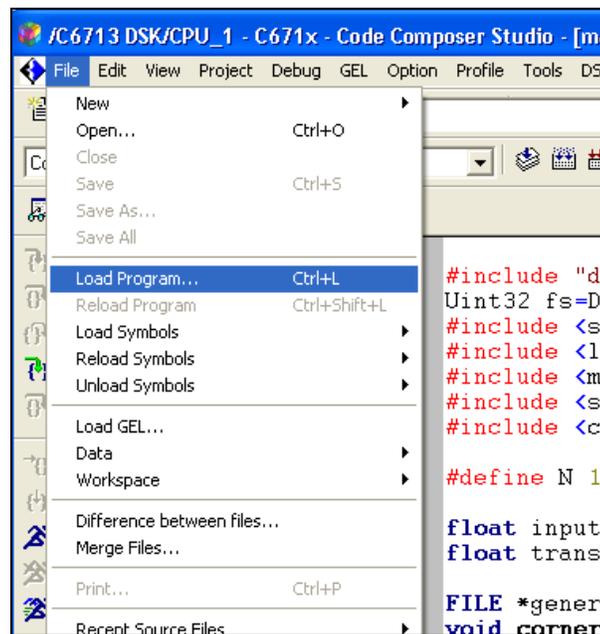


Figure 66: “Load Program” Location

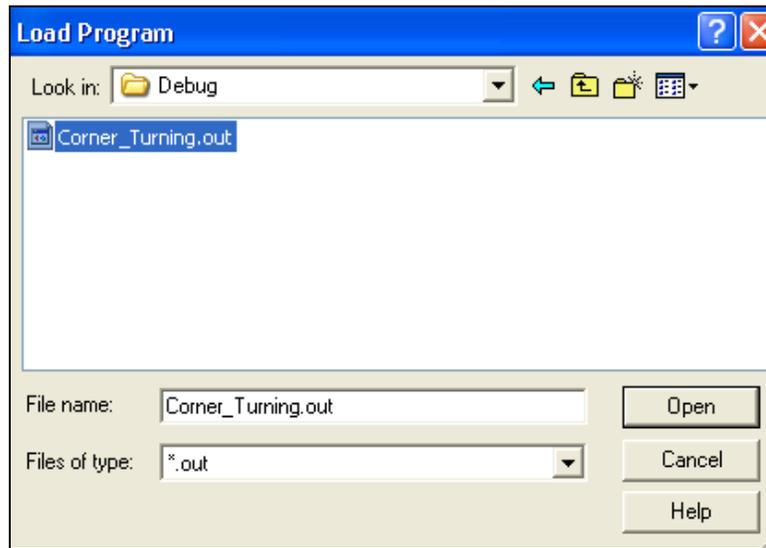


Figure 67: “Corner_Turning.out” File Location

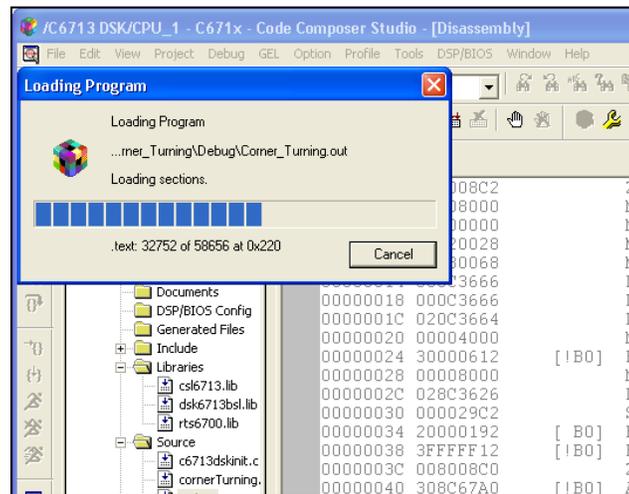


Figure 68: Downloading the Corner_Turning.out File to the TMS320C6713 DSP

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.

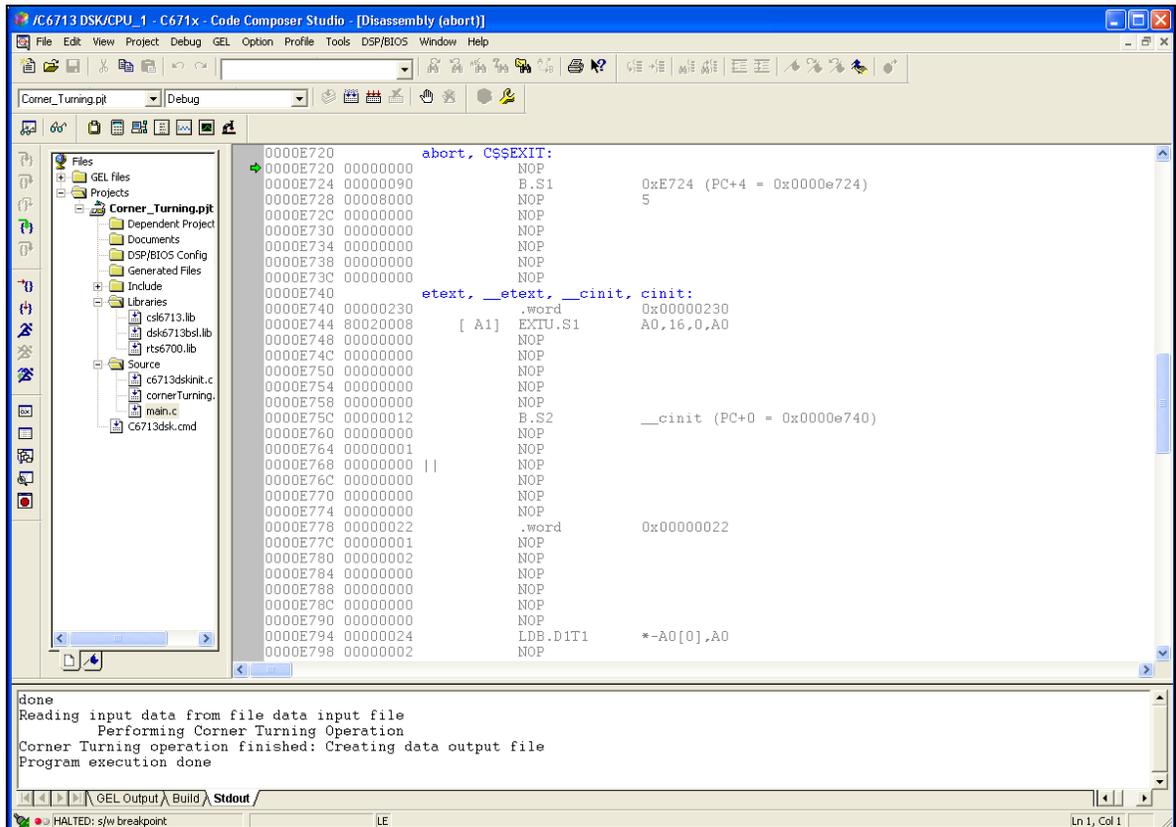


Figure 69: Results Obtained after Run the Algorithm "Corner_Turning".

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 Platinum
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**.

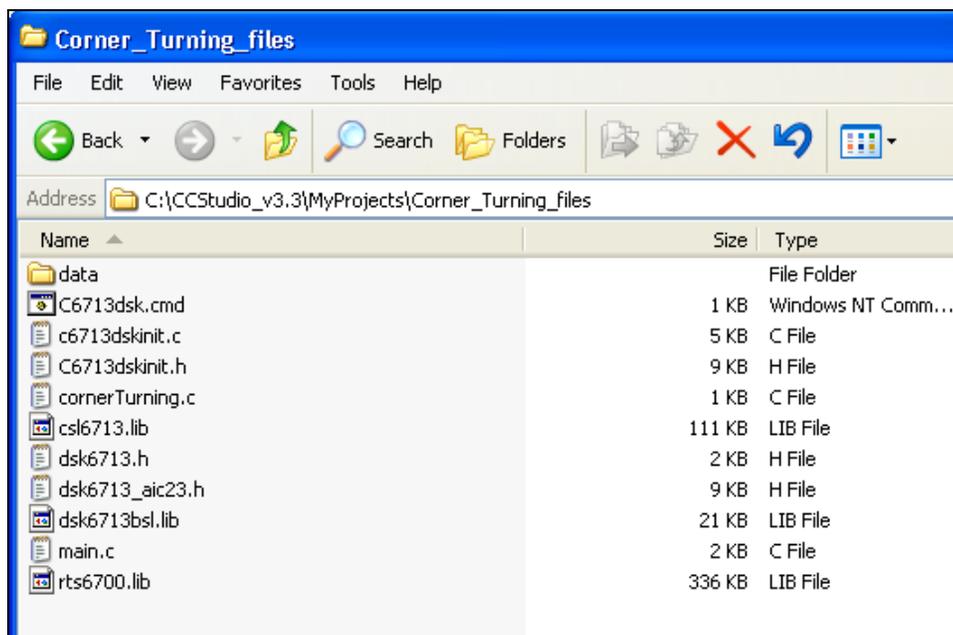


Figure 70: Corner Turning Files

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 (.pj) 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.

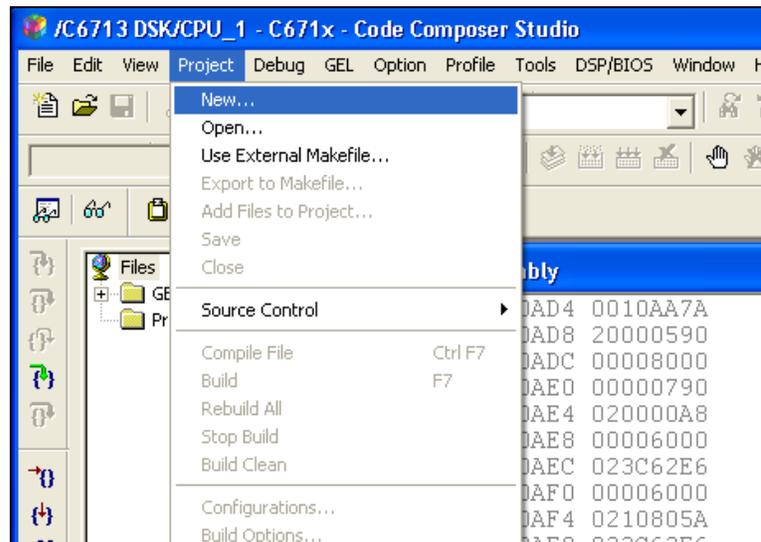


Figure 71: Creating a New Project

Verify if the following option is selected:

Target → **TMS320C67XX**, and then click Finish to continue.

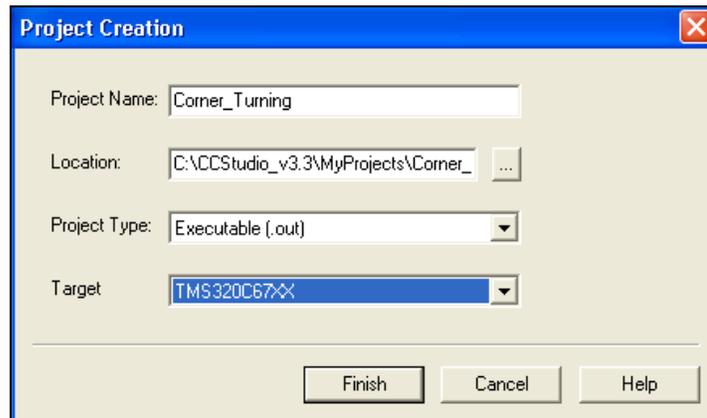


Figure 72: Window for the Creation of a New Project

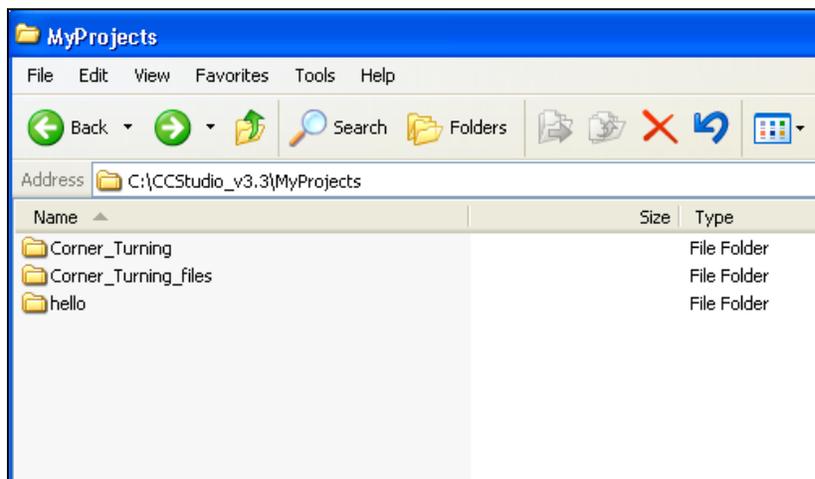


Figure 73: Corner_Turning Project Folder

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

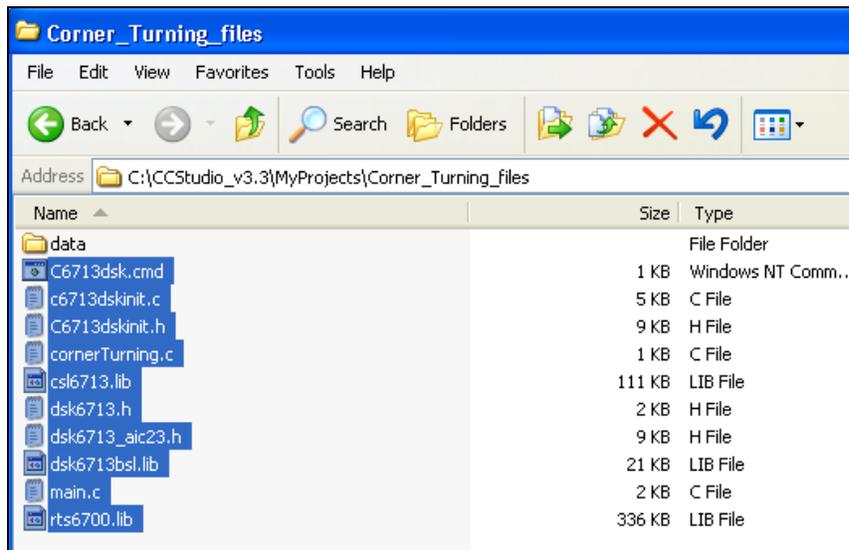


Figure 74: Corner Turning Project Files

3. Select **Project** → **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

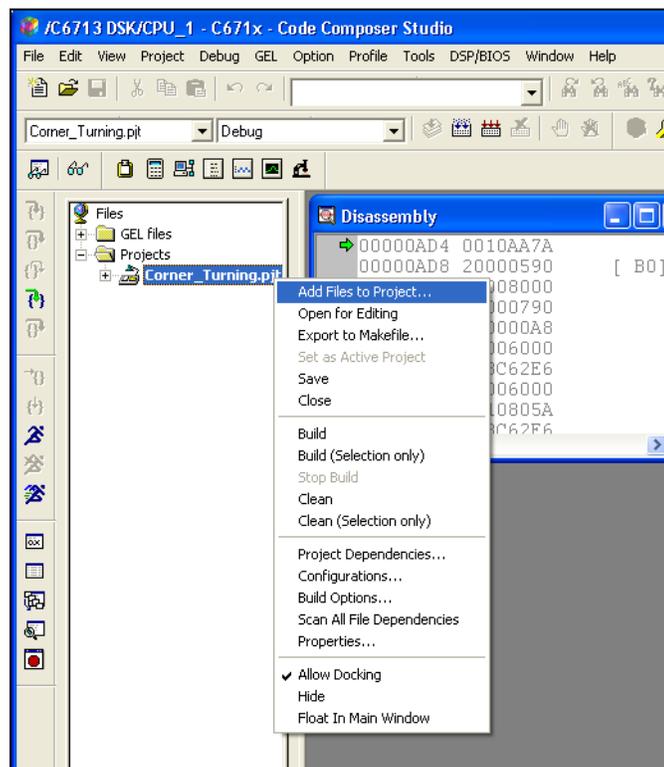


Figure 75: Adding Files to the Project

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

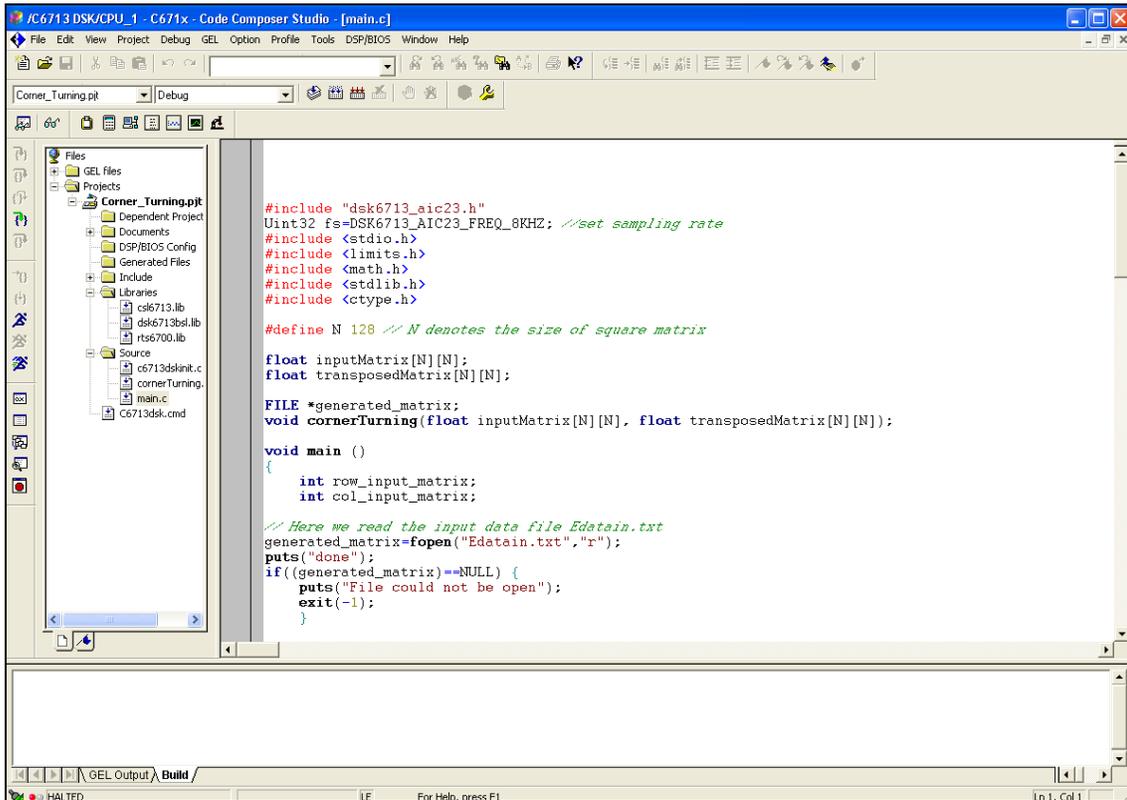


Figure 76: Project Files

- 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.

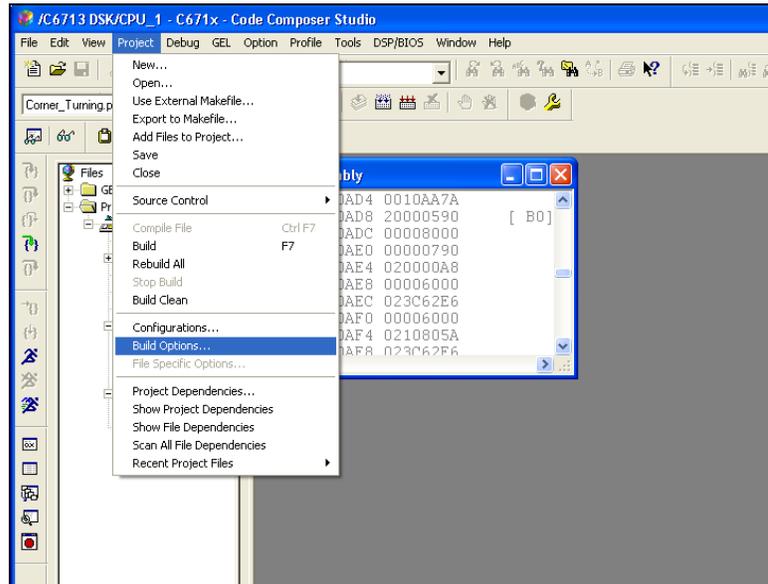


Figure 77: Build Option Setting Location

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

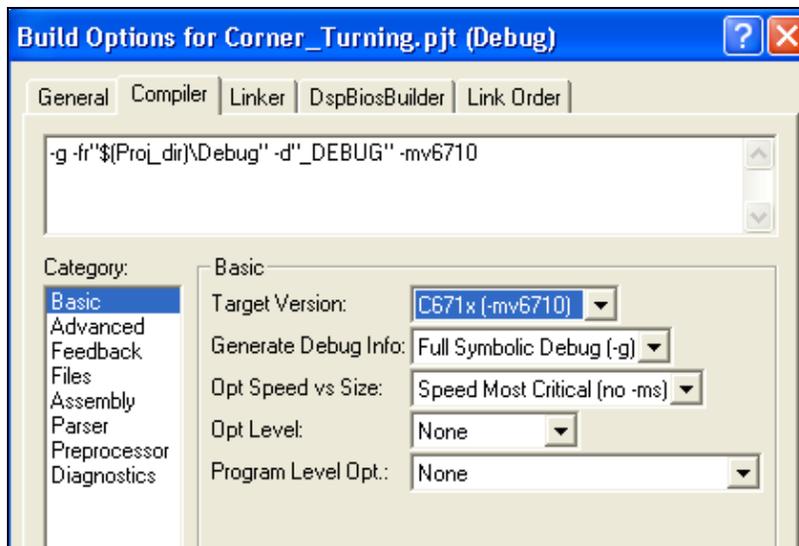


Figure 78: Setting the Target Version

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

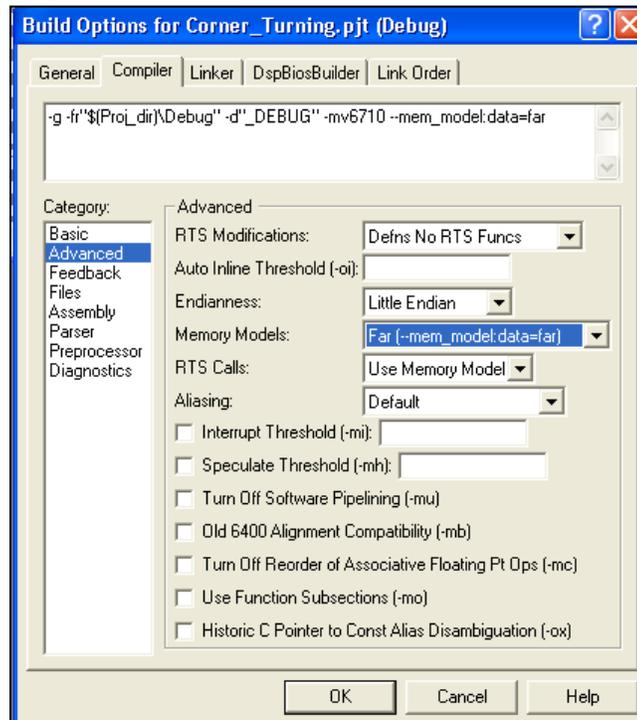


Figure 79: Memory Model Type Selection

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.

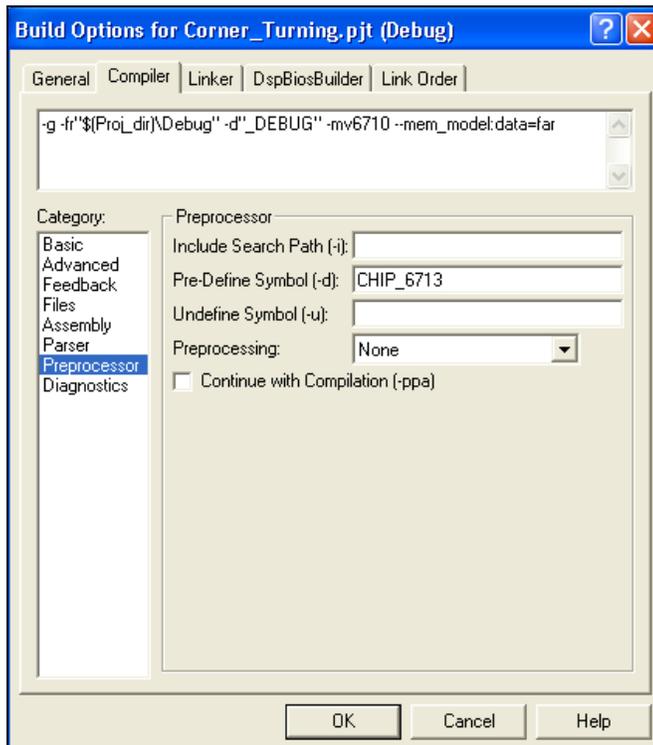


Figure 80: Specifying the Chip Architecture

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



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.

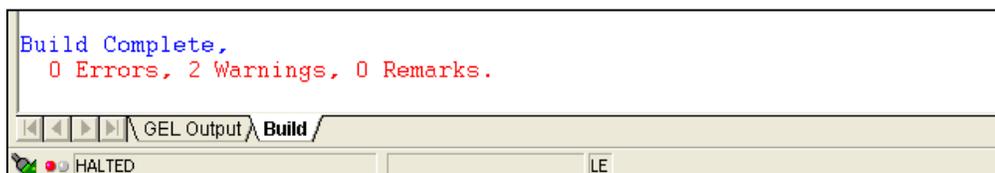


Figure 82: Compiling Results

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**.

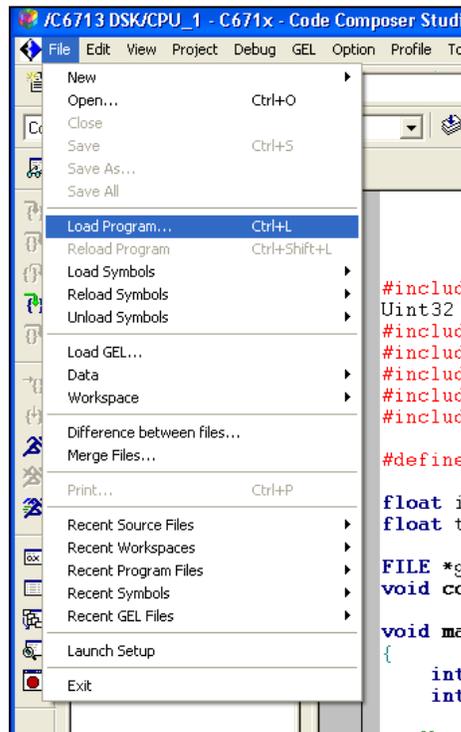


Figure 83: “Load Program” Location

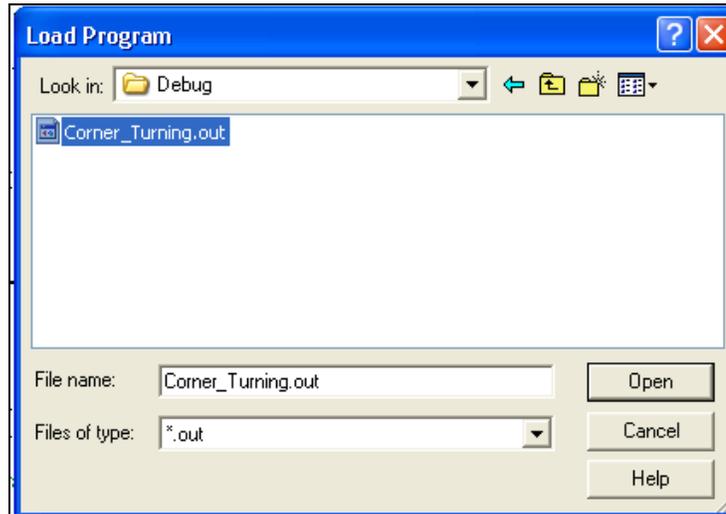


Figure 84: Corner_Turning.out File Location

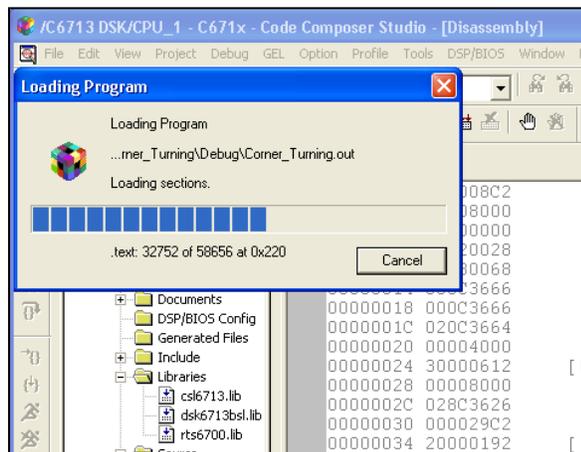


Figure 85: Downloading the Corner_Turning.out File to the TMS320C6713 DSP

3. Copy the following files from
C:\CCStudio_v3.3\MyProjects\Corner_Turning_files\data
C:\CCStudio_v3.3\MyProjects\Corner_Turning\Debug:

- Edatain.txt
- Sdatain.txt

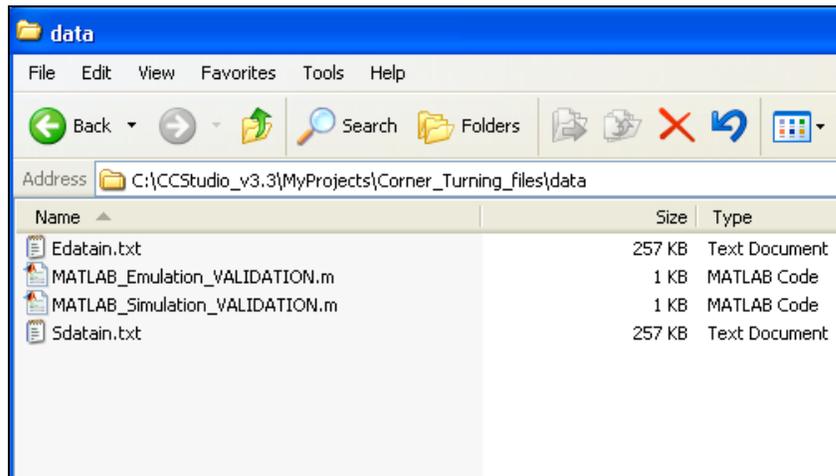


Figure 86: Corner Turning Input Data and Validation Files

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.

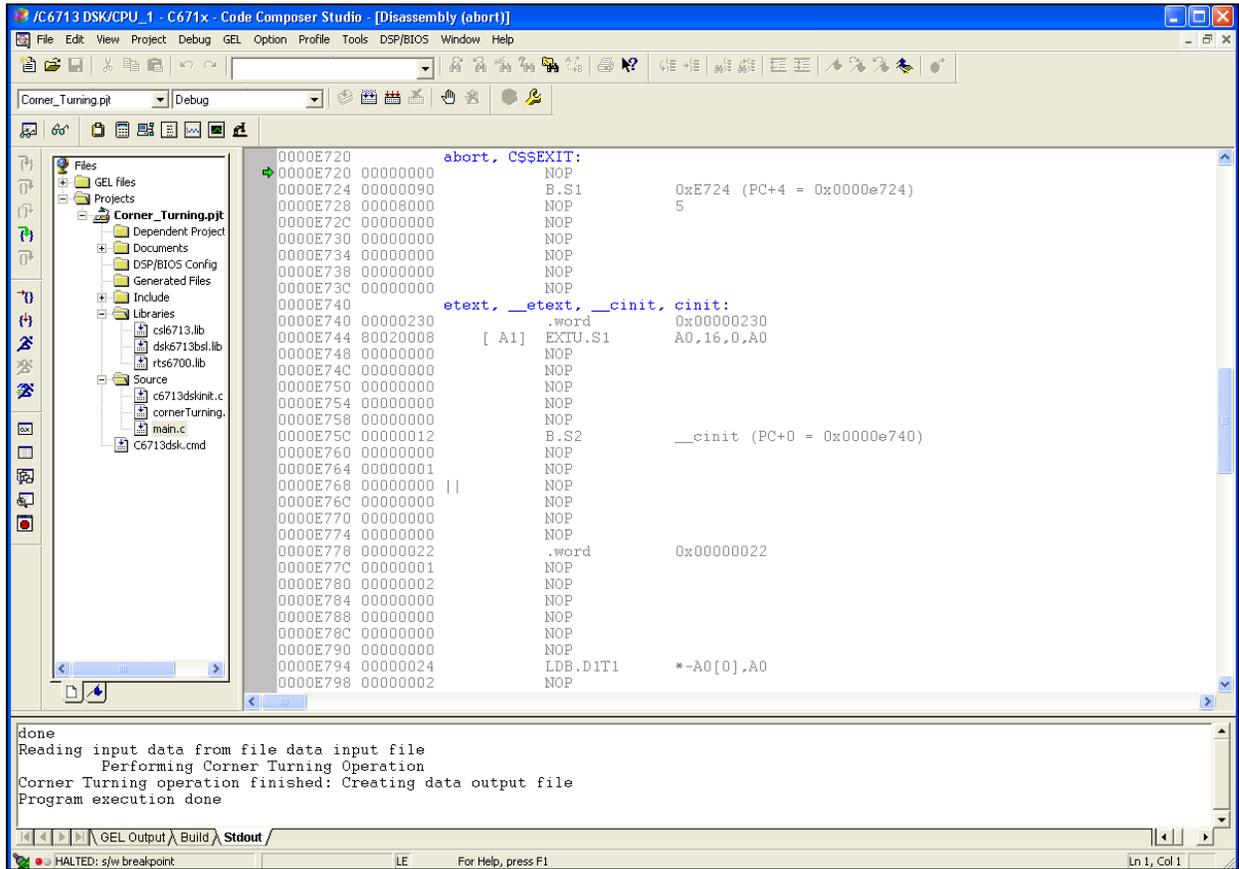


Figure 87: Results Obtained after Run the Algorithm "Corner_Turning".

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_{\square} 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 $T_h: L(Z_N) \rightarrow L(Z_N)$ 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

$$T_h \left\{ \delta_{\{k\}} \right\} = \sum_{j \in Z_N} h[j, k] \delta_{\{j\}}$$

where the set of scalars

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

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

$$T_h \{ \delta_{\{k\}} \} = \sum_{j \in \mathcal{Z}_N} T_h \{ \delta_{\{k\}} \} [j] \delta_{\{j\}}$$

where

$$\begin{aligned} T_h \{ \delta_{\{k\}} \} [j] &= \sum_{m \in \mathcal{Z}_N} h[m] (S_N^m \{ \delta_{\{k\}} \}) [j] = \sum_{m \in \mathcal{Z}_N} h[m] \delta_{\{k+m\}} [j] \\ &= \sum_{m \in \mathcal{Z}_N} h[m] \delta [j-k-m] = h[j-k] = (S_N^k \{ h \}) [j] \end{aligned}$$

Thus, we write

$$\begin{aligned} T_h \{ \delta_{\{k\}} \} &= \sum_{j \in \mathcal{Z}_N} h[j, k] \delta_{\{j\}} = \sum_{j \in \mathcal{Z}_N} h[j-k] \delta_{\{j\}} \\ &= \sum_{j \in \mathcal{Z}_N} (S_N^k \{ h \}) [j] S_N^j \{ \delta \} = T_{(S_N^k \{ h \})} \{ h \} = S_N^k \{ h \} \end{aligned}$$

Next, we define the matrix H_N as follows

$$H_N = [h[j, k]]_{j, k \in \mathcal{Z}_N} = [h[j-k]]_{j, k \in \mathcal{Z}_N}$$

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

$$H_N = [I_N h, S_N \{ h \}, S_N^2 \{ h \}, \dots, S_N^{N-1} \{ h \}]$$

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

$$\begin{aligned} (\delta_{\{k\}}) &= \sum_{j \in Z_N} h[j, k] S_N^j \{\mathcal{D}\}, \quad h[j, k] \in C \\ &= h[0, k] \delta_{\{0\}} + h[1, k] \delta_{\{1\}} + \dots + h[N-1, k] \delta_{\{N-1\}} \end{aligned}$$

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

$$\begin{aligned} T_h \{ \delta_{\{0\}} \} &= h[0,0] \delta_{\{0\}} + h[1,0] \delta_{\{1\}} + \dots \\ &\quad + h[N-1,0] \delta_{\{N-1\}} \\ T_h \{ \delta_{\{1\}} \} &= h[0,1] \delta_{\{0\}} + h[1,1] \delta_{\{1\}} + \dots \\ &\quad + h[N-1,1] \delta_{\{N-1\}} \\ &\quad \vdots \\ T_h \{ \delta_{\{N-1\}} \} &= h[0, N-1] \delta_{\{0\}} + h[1, N-1] \delta_{\{1\}} \\ &\quad + \dots + h[N-1, N-1] \delta_{\{N-1\}} \end{aligned}$$

We write these identities in an array form:

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

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

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

Expanding the above sum, we obtain

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

where

$$\begin{aligned} f[0]T_h\{\delta_{\{0\}}\} &= f[0]h[0,0]\delta_{\{0\}} + f[0]h[1,0]\delta_{\{1\}} + \dots \\ &\quad + f[0]h[N-1,0]\delta_{\{N-1\}} \\ f[1]T_h\{\delta_{\{1\}}\} &= f[1]h[0,1]\delta_{\{0\}} + f[1]h[1,1]\delta_{\{1\}} + \dots \\ &\quad + f[1]h[N-1,1]\delta_{\{N-1\}} \\ &\quad \vdots \\ f[N-1]T_h\{\delta_{\{N-1\}}\} &= f[N-1]h[0,N-1]\delta_{\{0\}} + f[N-1]h[1,N-1]\delta_{\{1\}} \\ &\quad + \dots + f[N-1]h[N-1,N-1]\delta_{\{N-1\}} \end{aligned}$$

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

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

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

where

$$\begin{aligned} g[m] &= T_h\{f\} = \left(\sum_{j \in Z_N} \sum_{k \in Z_N} f[k]h[j,k] \right) \delta_{\{j\}}[m] \\ &= \sum_{K \in Z_N} f[k]h[m,k], m \in Z_N \end{aligned}$$

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 from 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 $h[j, k] = h[j - k]$, $j, k \in Z_N$, we write

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

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 \square and \square denote, both, the coordinate vector representation of the signals \square and \square , respectively, as well as the signals themselves; and the matrix H_N represents the system T_h :

$$\begin{aligned} H_N &= \left[T_h\{\delta_{\{0\}}\}, T_h\{\delta_{\{1\}}\}, \dots, T_h\{\delta_{\{N-1\}}\} \right] \\ &= \left[T_{\delta_{\{0\}}}\{h\}, T_{\delta_{\{1\}}}\{h\}, \dots, T_{\delta_{\{N-1\}}}\{h\} \right] \\ &= \left[I_N T_h\{\delta\}, S_N T_h\{\delta\}, \dots, S_N^{N-1} T_h\{\delta\} \right] \end{aligned}$$

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

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

and proceed in the following manner

$$\begin{aligned} T_h\{f\} &= T_h\left\{\sum_{k \in Z_N} f[k]\delta_{\{k\}}\right\} \\ &= \sum_{k \in Z_N} f[k]T_h\{\delta_{\{k\}}\} \\ &= \sum_{k \in Z_N} f[k]\left(\sum_{j \in Z_N} h[j-k]\delta_{\{j\}}\right) \\ &= \sum_{j \in Z_N} \left(\sum_{k \in Z_N} h[j-k]f[k]\right)\delta_{\{j\}} \end{aligned}$$

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

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

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 represented 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 hardware 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 representation 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 representation (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 :

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

$$= \left[\delta_{\{1\}}, \delta_{\{2\}}, \dots, \delta_{\{N-1\}}, \delta_{\{0\}} \right]$$

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

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

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 pre-multiplied by a diagonal matrix $D_{h[j]}$:

$$H_N = \sum_{j \in Z_N} D_{h[j]} S_N^j = \sum_{j \in Z_N} (h[j] \otimes S_N^j)$$

where

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

4.2 Cyclic Matrix

A cyclic matrix of order N is a $N \times N$ matrix of the form

$$H_N = \begin{bmatrix} H_0 & H_{N-1} & \cdots & H_1 \\ H_1 & H_0 & \cdots & H_2 \\ \vdots & \vdots & \ddots & \vdots \\ H_{N-2} & H_{N-3} & \cdots & H_{N-1} \\ H_{N-1} & H_{N-2} & \cdots & H_0 \end{bmatrix}$$

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 $0 \leq n \leq N-1$, the discrete Fourier transform of $x[n]$ is defined as the succession given by

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

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

It is common to call $W_N = e^{-j2\pi/N}$ and rewrite the discrete Fourier transform $x^{\wedge}[k]$ as

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

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

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

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 unidimensional signals is defined by

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

where

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

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

$$R_N = [\mathfrak{R}_N\{\delta_{\{0\}}\} \quad \mathfrak{R}_N\{\delta_{\{1\}}\} \quad \cdots \quad \mathfrak{R}_N\{\delta_{\{N-1\}}\}].$$

now,

$$[(\mathfrak{R}_N\{\delta_{\{0\}}\})[n]] = [\delta_{\{0\}}[\langle -n \rangle_N]] = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

$$[(\mathfrak{R}_N\{\delta_{\{1\}}\})[n]] = [\delta_{\{1\}}[\langle -n \rangle_N]] = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix},$$

⋮

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

So the matrix of the reflection operator is

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

which again we see is a cyclic matrix.

4.5 Hadamard Product

The Hadamard product over the space $l^2(\mathbb{Z}_N)$ of unidimensional signals is defined as

$$\begin{aligned} \odot: l^2(\mathbb{Z}_N) \times l^2(\mathbb{Z}_N) &\rightarrow l^2(\mathbb{Z}_N) \\ (x, y) &\mapsto x \odot_N y, \end{aligned}$$

where

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

So notice that if

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

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:

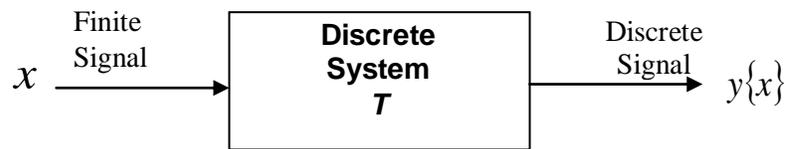


Figure 88: Discrete System Block Diagram

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

$$x : Z_4 \rightarrow C$$

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

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

Correspond

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

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

We represent a vector as follows:

$$\begin{aligned} \delta : Z_N &\rightarrow C \\ n &\mapsto \delta[n] \end{aligned}$$

$$\delta = \{\delta[0], \delta[1], \dots, \delta[N-1]\} \quad \Leftrightarrow \quad \delta = \begin{bmatrix} \delta[0] \\ \delta[1] \\ \vdots \\ \delta[N-1] \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

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

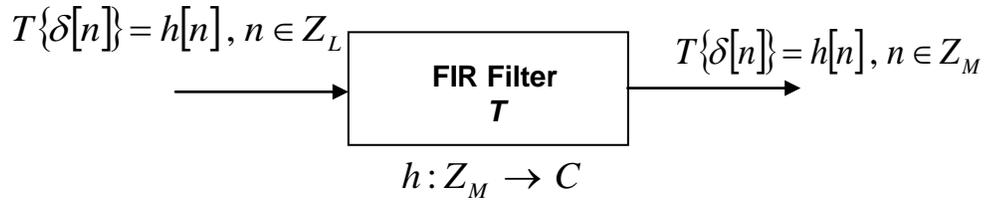


Figure 90: FIR Filter Block Diagram

Observation:

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

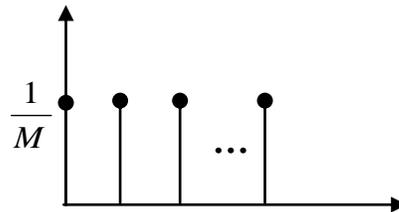


Figure 91: FIR Filter Block Diagram

Example 1

The Finite Response Averaging Filter to a unitary impulse

$$h[n] = \begin{cases} \frac{1}{M}, & n \in Z_M \\ 0, & n \notin Z_M \end{cases}$$



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

Example 2

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



Figure 92: Averaging Filter Block Diagram

$$\delta[n-2] = S[n]$$

$$S : Z_L \rightarrow C$$

$$n \mapsto S[n]$$

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

Observation

All finite signals with dimension L can be represented as a lineal combination of

displaced unitary impulse: $x[n] = \sum_{k=0}^{L-1} x[k] \delta[n-k]$.

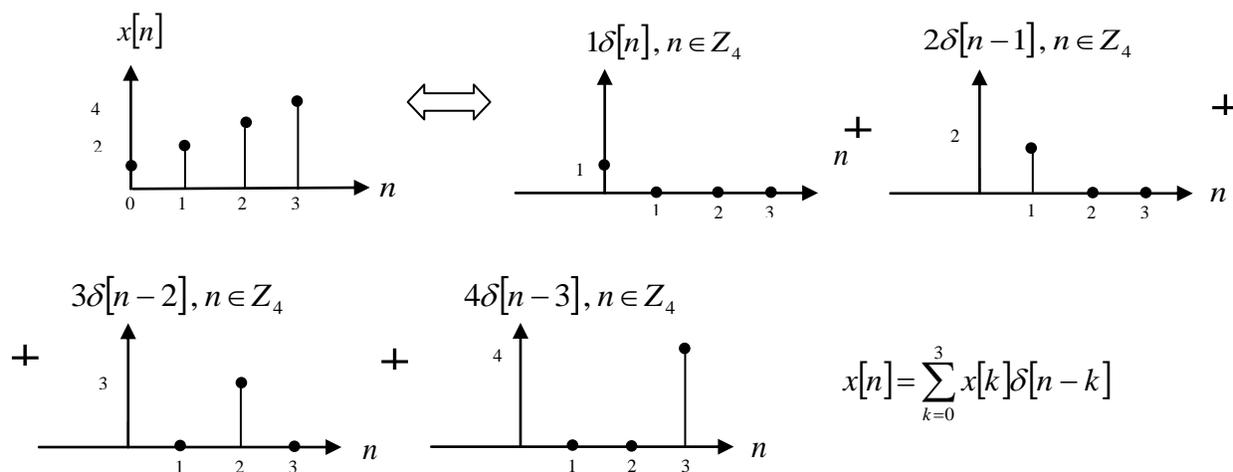
Example

Represent the signal $x[n] = x[n+1]$, $n \in Z_4$ as a sum of displaced unitary impulses.

$$x: Z_4 \rightarrow \mathbb{C}$$

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

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



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*.

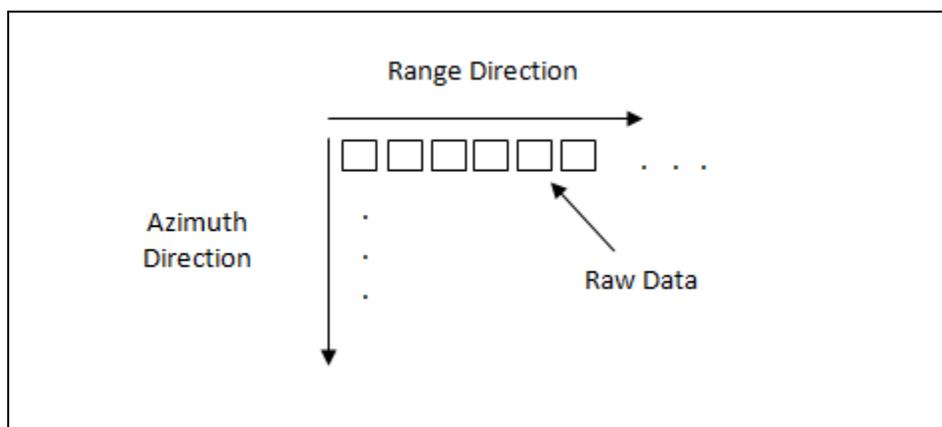


Figure 93: Range and Azimuth Direction

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:

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

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.

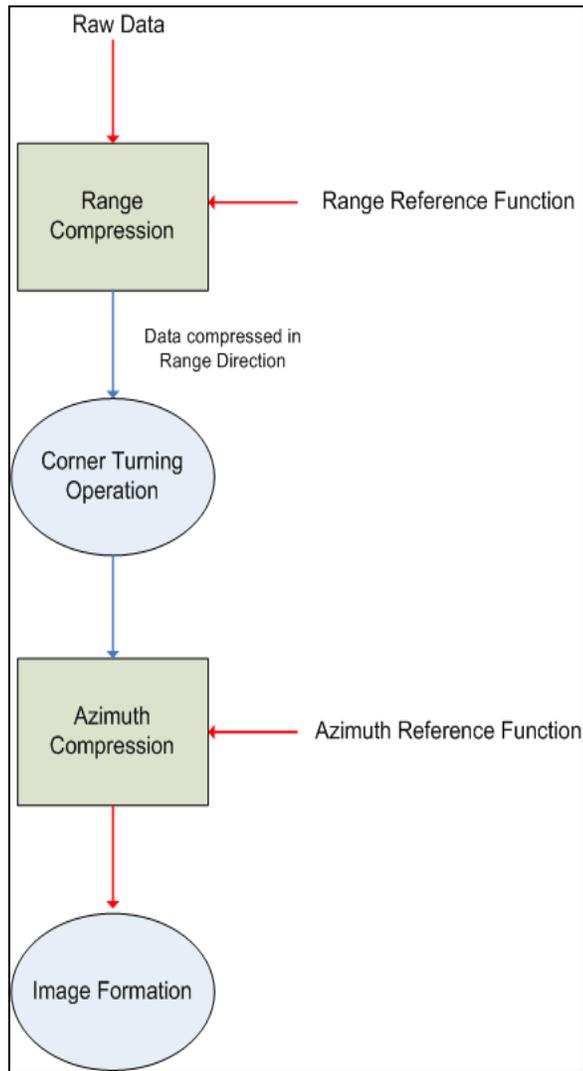


Figure 94: SAR Image Formation Diagram Procedure

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

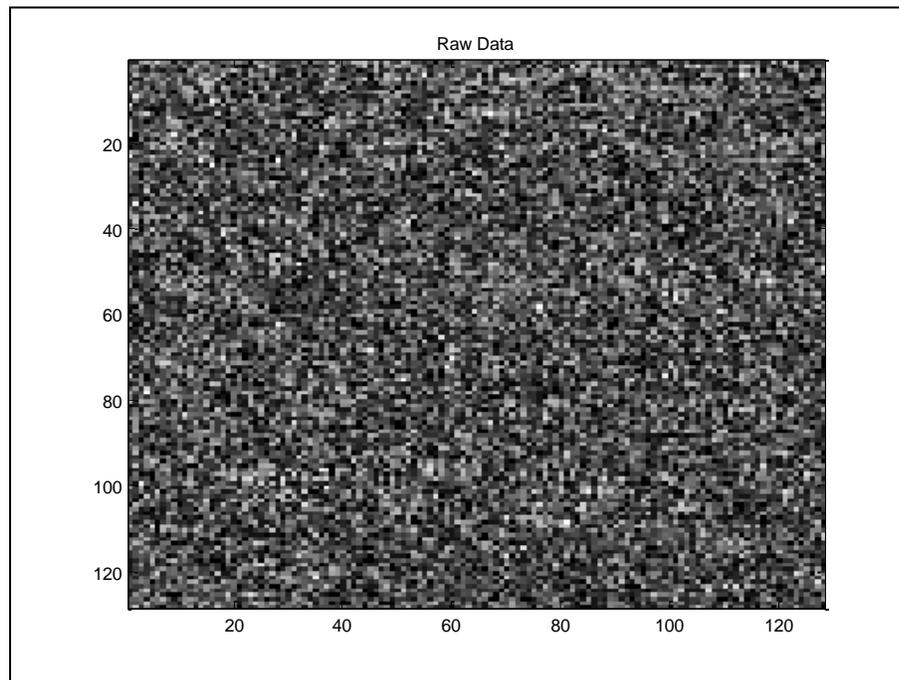


Figure 95: Raw Data

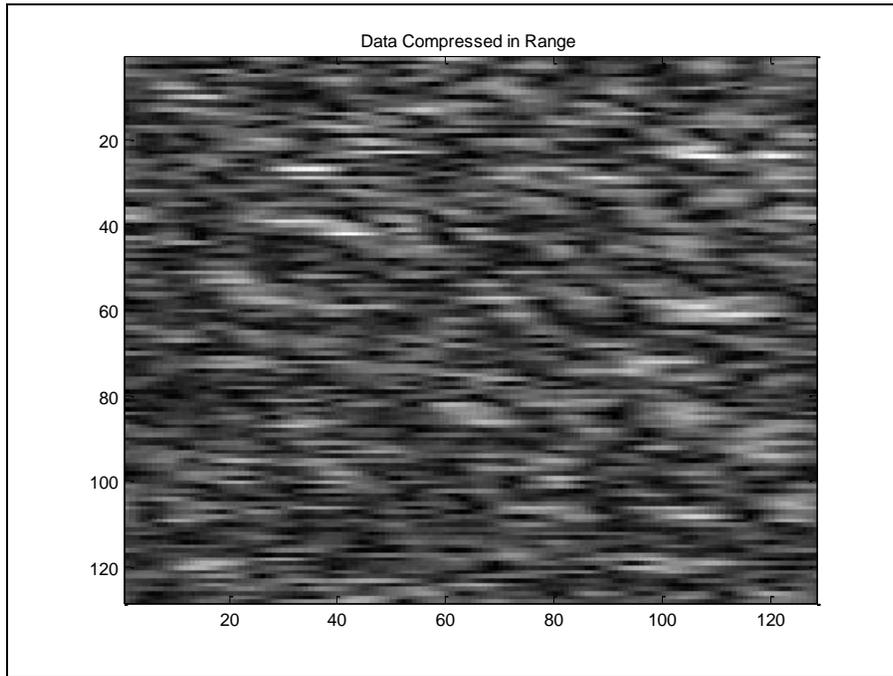


Figure 96: Data Compressed in Range

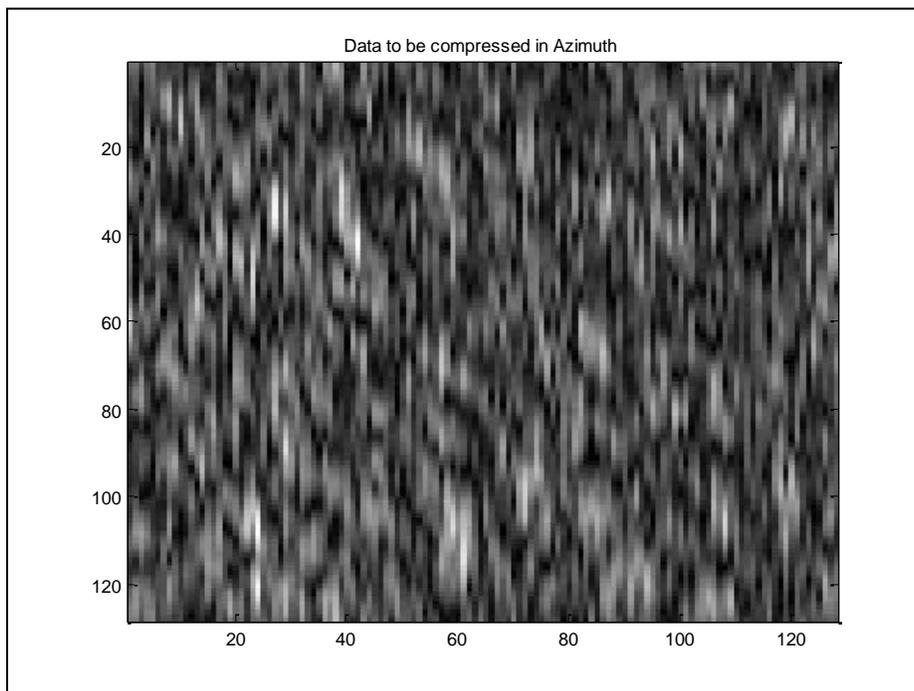


Figure 97: Applying Corner Turning to Data Compressed in Range Direction

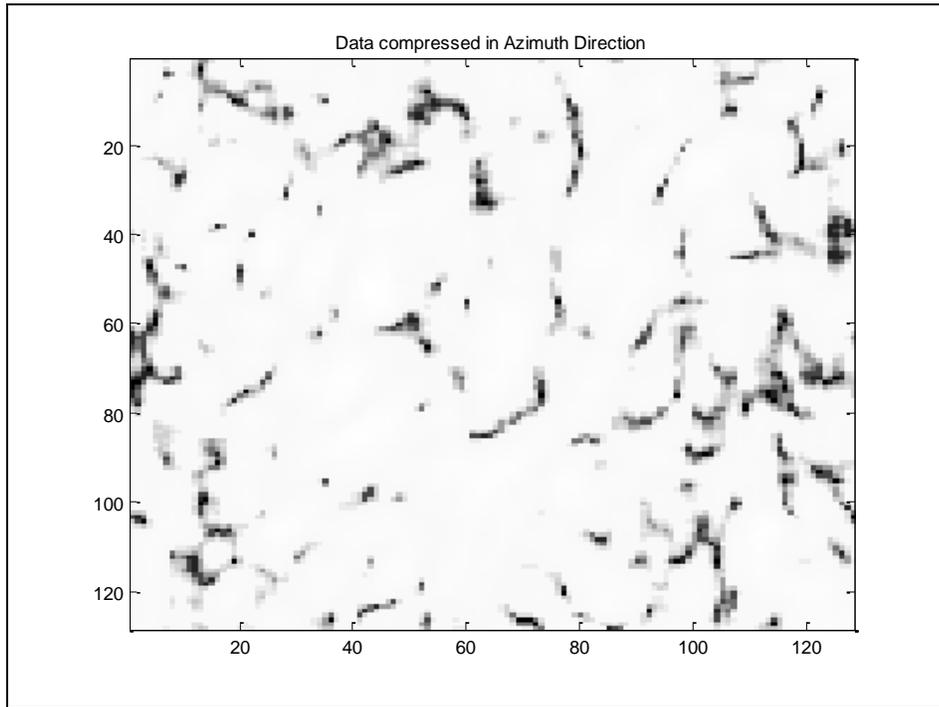


Figure 98: Data Compressed in Azimuth Direction

5.2.2 TMS320C6713 Emulation results for 256x256 Raw Data

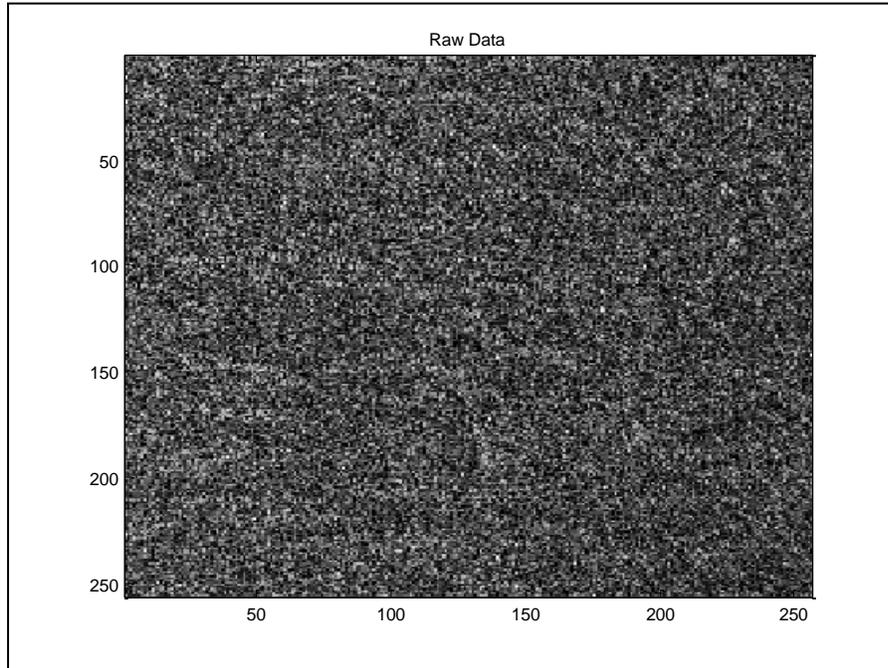


Figure 99: Raw Data

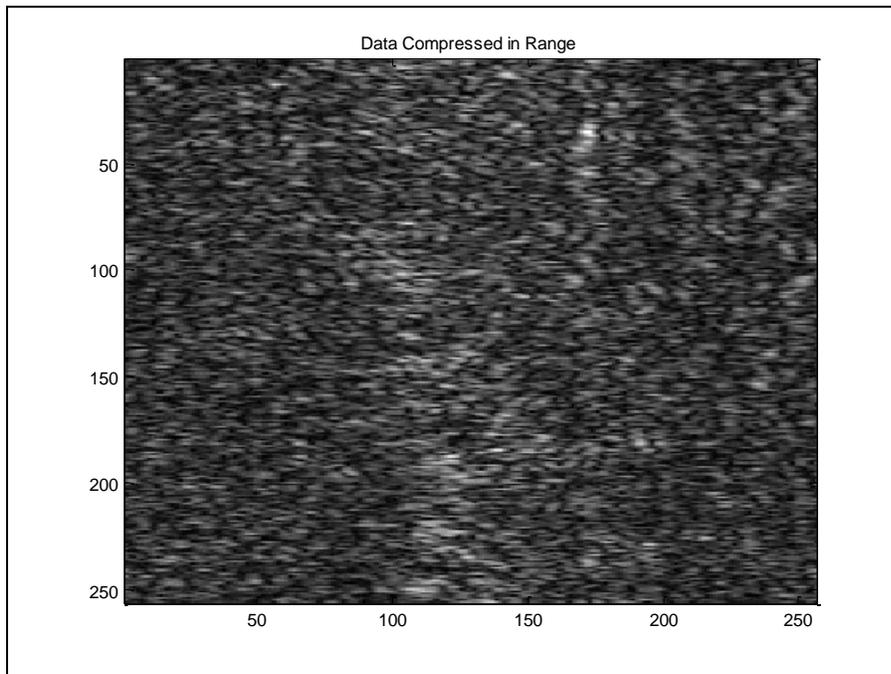


Figure 100: Data Compressed in Range

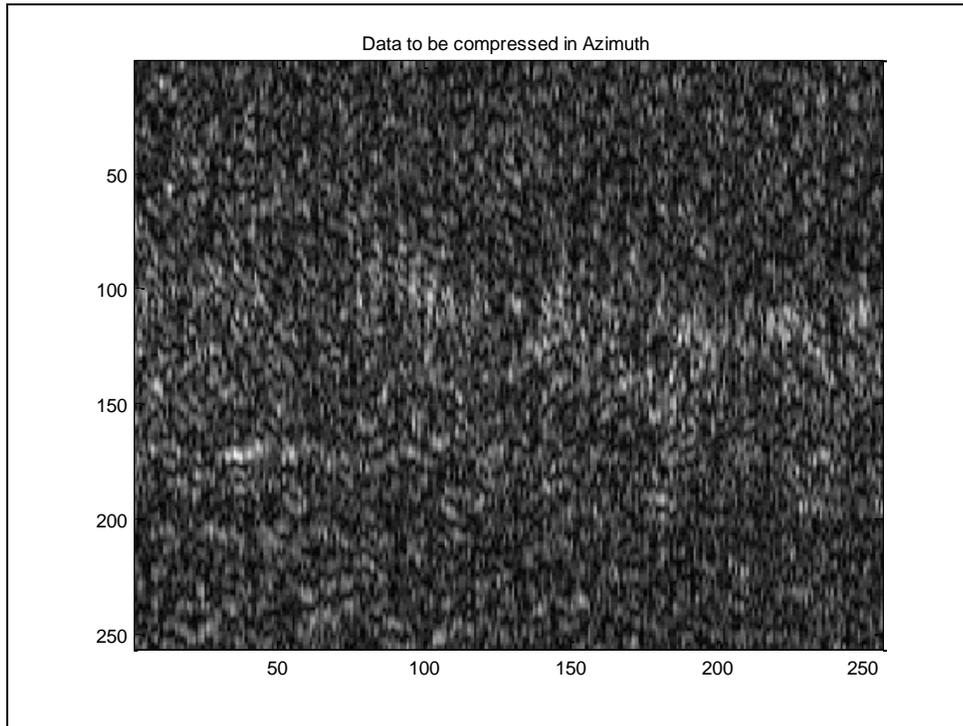


Figure 101: Applying Corner Turning to Data Compressed in Range Direction

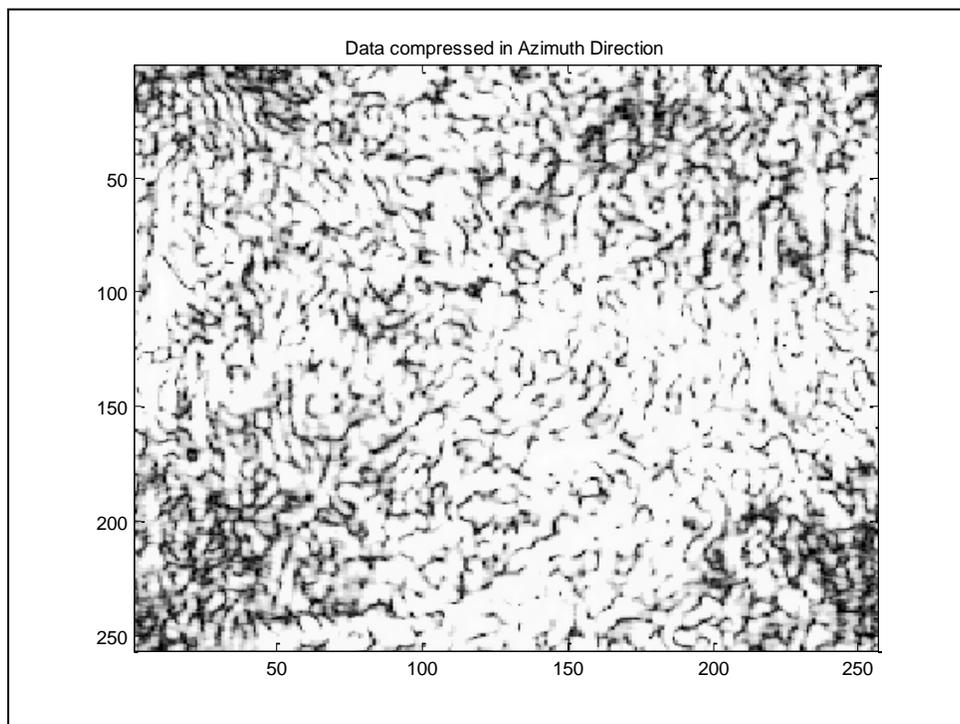


Figure 102: Data Compressed in Azimuth Direction

5.2.3 TMS320C6713 Emulation results for 512x512 Raw Data

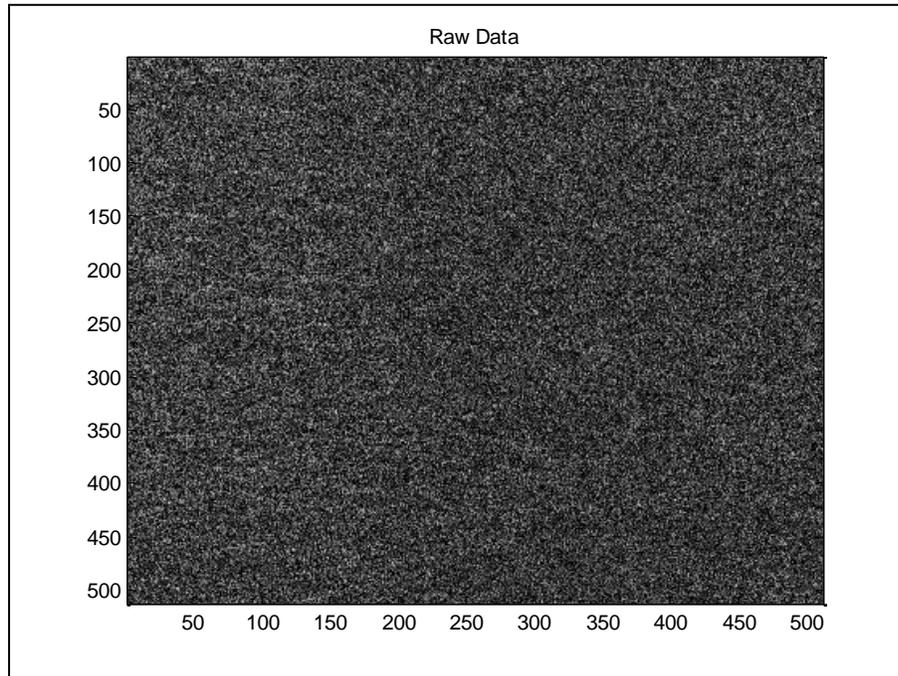


Figure 103: Raw Data

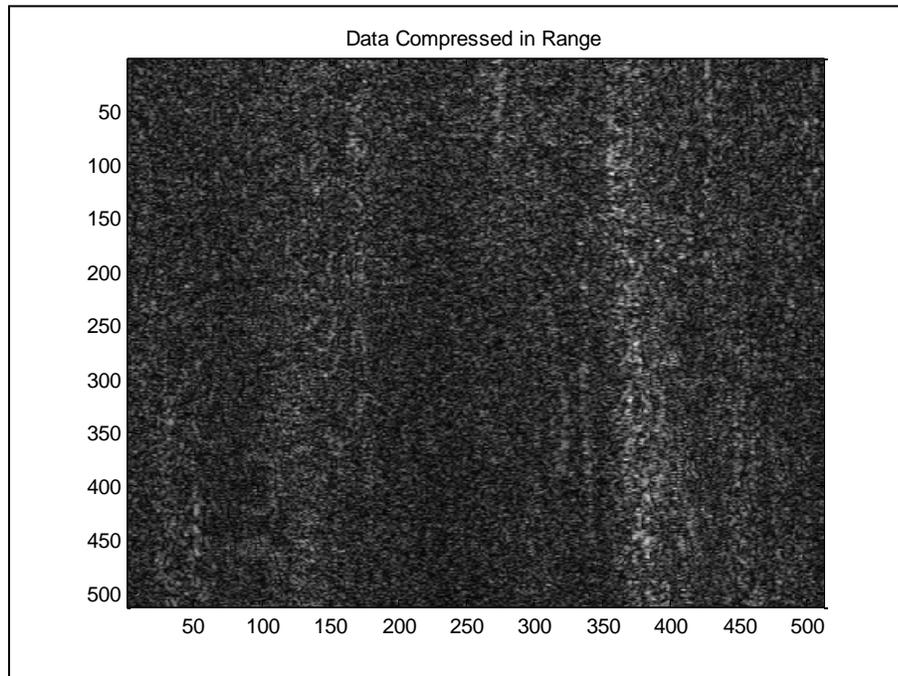


Figure 104: Data Compressed in Range

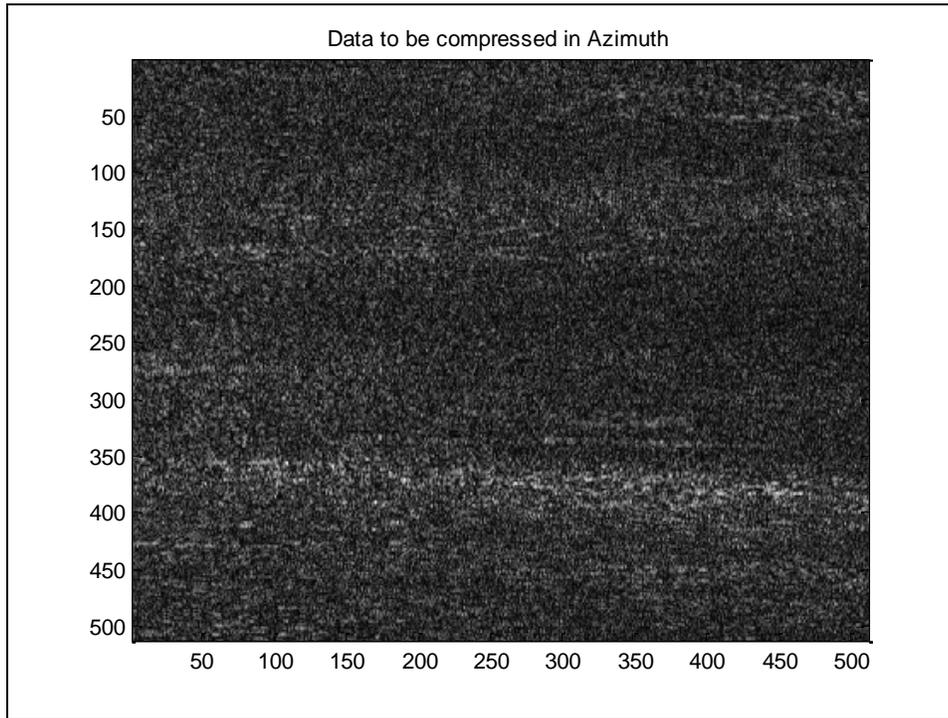


Figure 105: Applying Corner Turning to Data Compressed in Range Direction

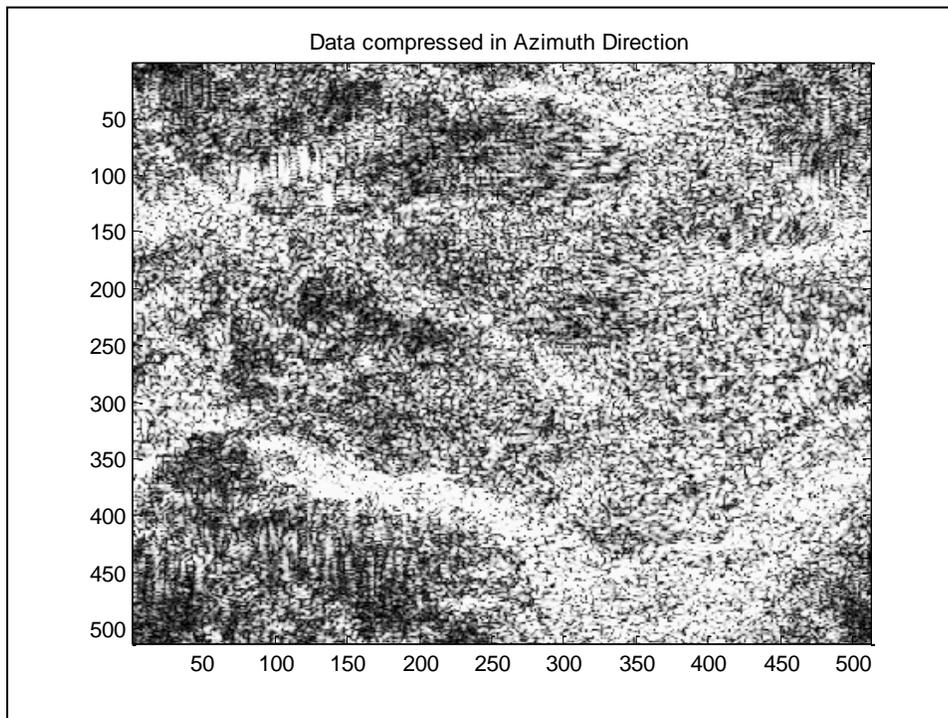


Figure 106: Data Compressed in Azimuth Direction

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 Platinum
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* – Computes one dimensional inverse fast Fourier transform (IFFT) for the range compression.
- *IFFTAzimuth.c* – Computes 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**.

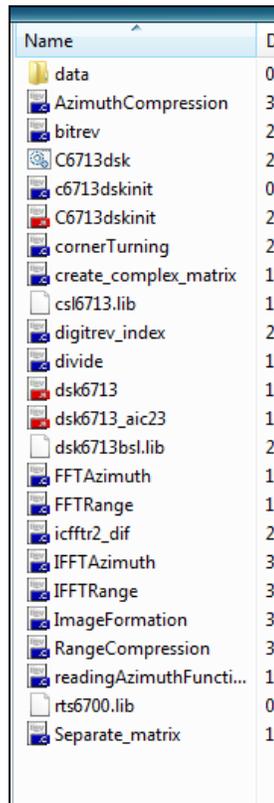


Figure 107: ImagingFormation Files

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** → **New**. In the filename, type the name “**FFTproject**” of the new project and click “**Save**”.

This project file (.pj) 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.

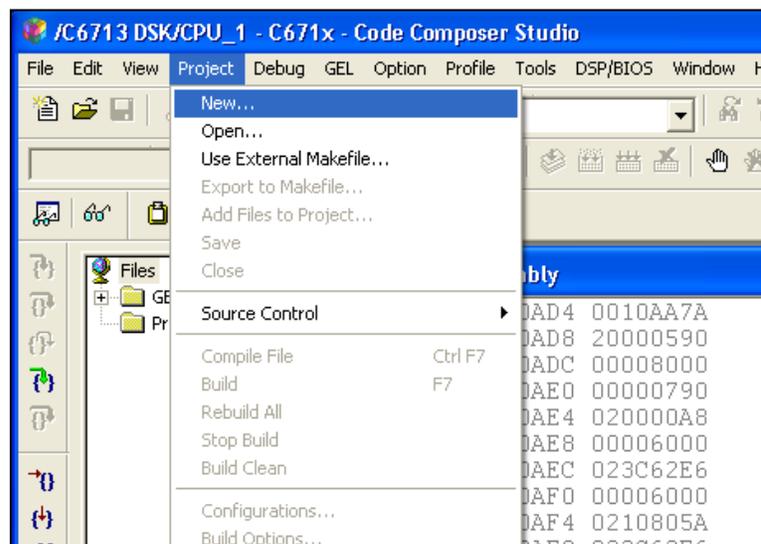


Figure 108: Creating a New Project

Verify if the following option is selected:

Target → **TMS320C67XX**, and then click **Finish**.

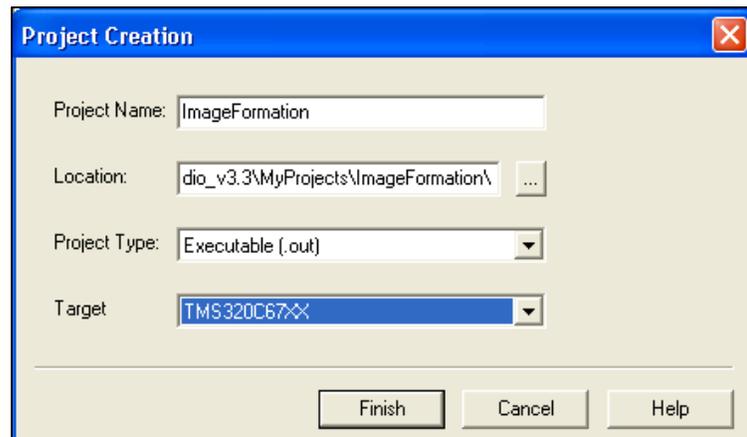


Figure 109: Window for the Creation of a New Project

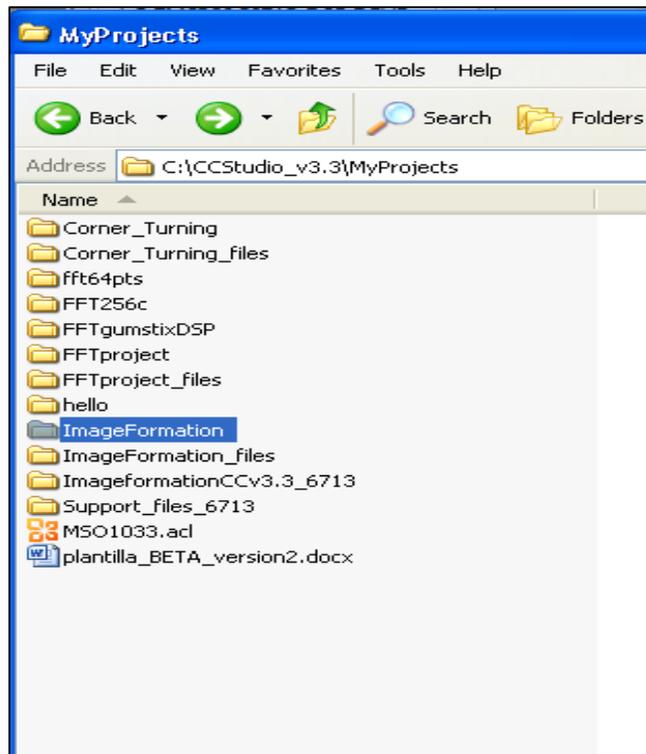


Figure 110: “ImageFormation” Project Folder

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

- 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

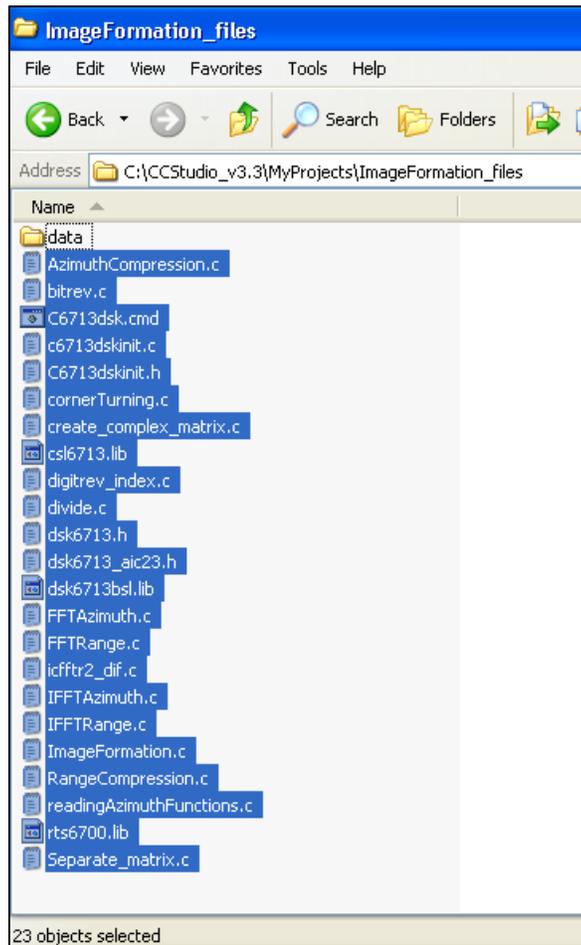


Figure 111: Image Formation Project Files

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_matrix.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

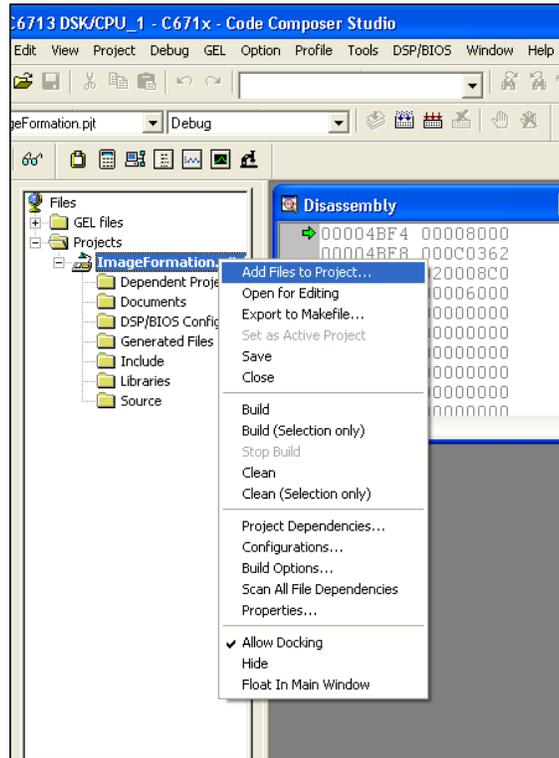


Figure 112: Adding Files to the Project

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

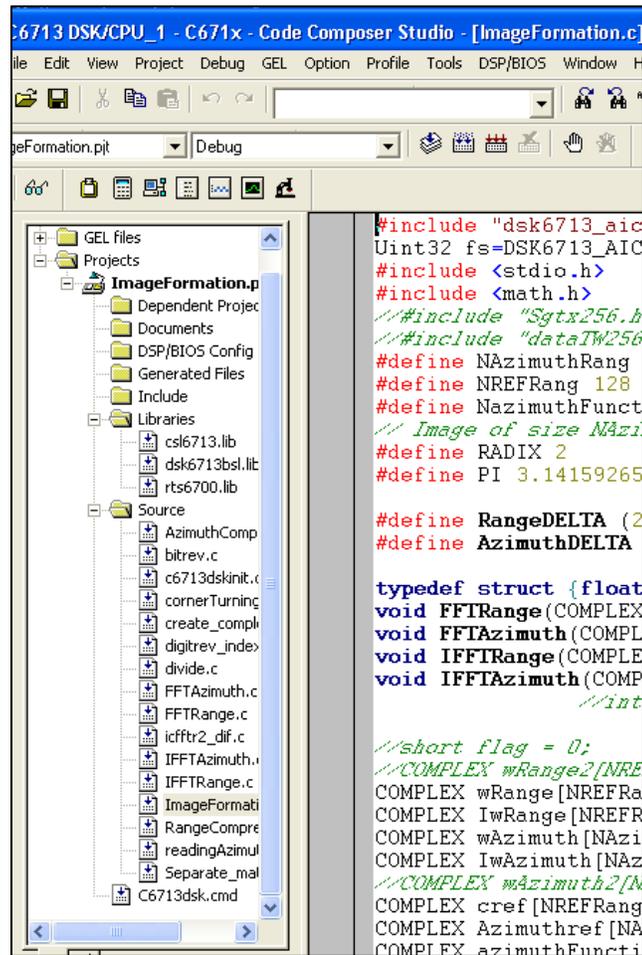


Figure 113: Project Files

- 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 be chosen or written, and the option OK is selected after all settings are verified.

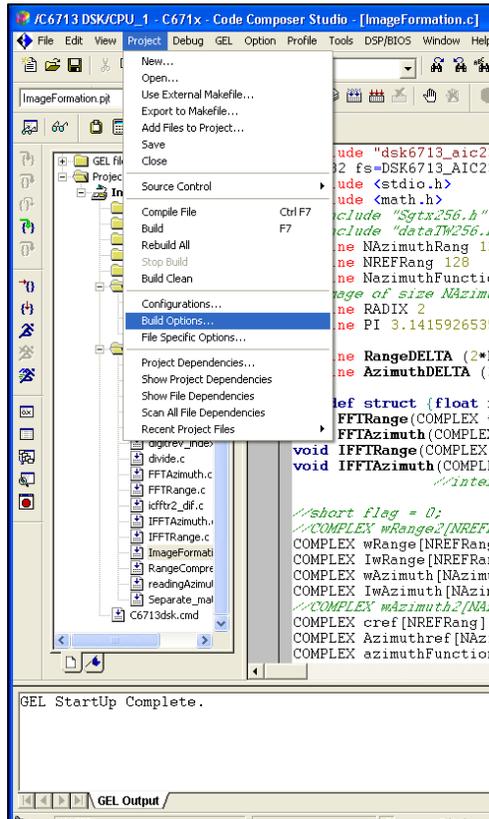


Figure 114: Build Option Setting Location

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

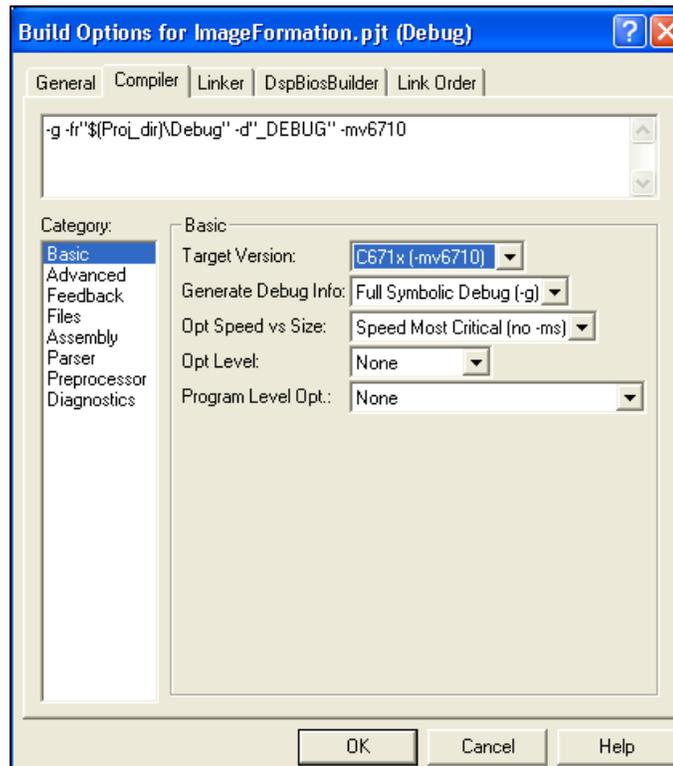


Figure 115: Setting the Target Version

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

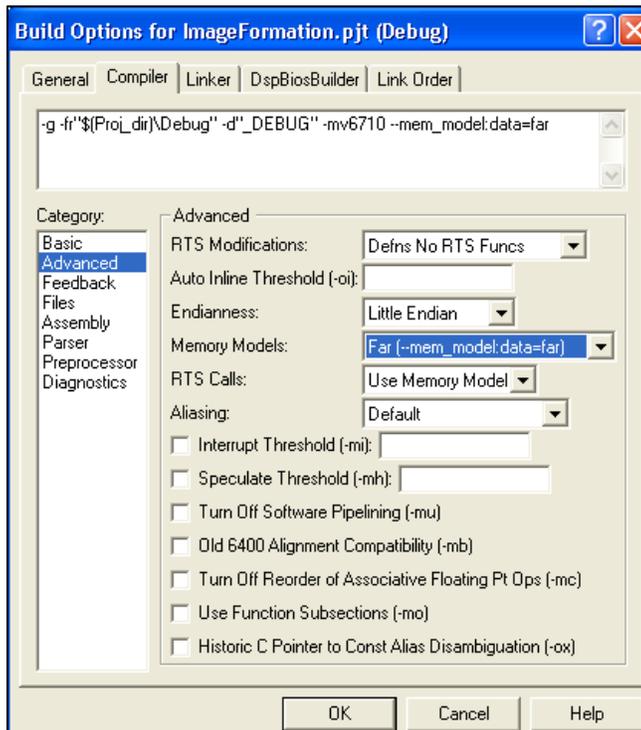


Figure 116: Memory Model Type Selection

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** → **Basic**:
 - In **Heap Size (-heap)** and in **Stack Size (-stack)**, writes **32000**.

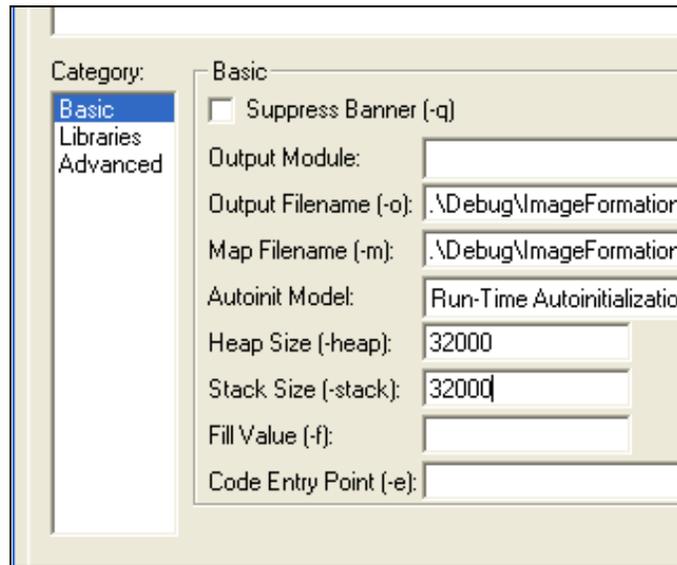


Figure 117: Building options for Linker → Basic

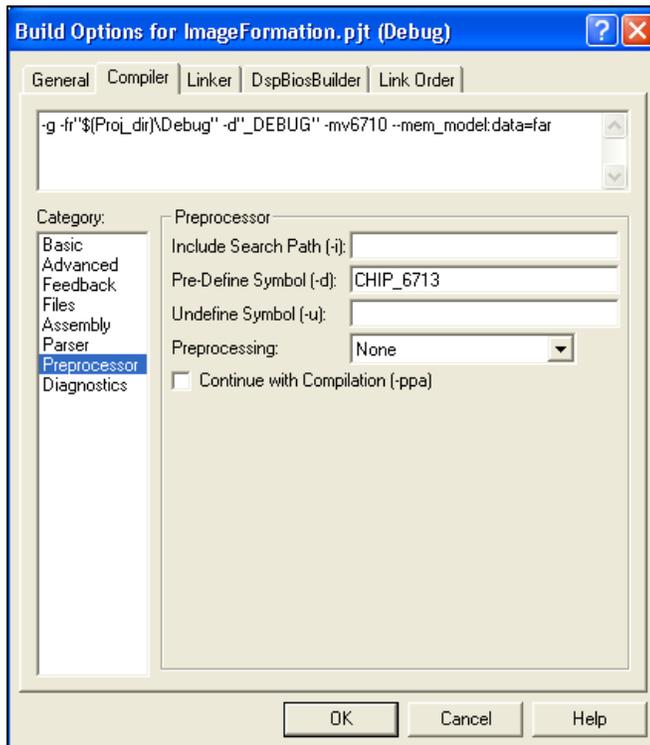


Figure 118: Specifying the Chip Architecture

10. Under *Linker* → *Libraries*:

- In *Included Libraries (-l)*, these libraries must be specified: **rts6700.lib**;
dsk6713bsl.lib; **cs16713.lib**

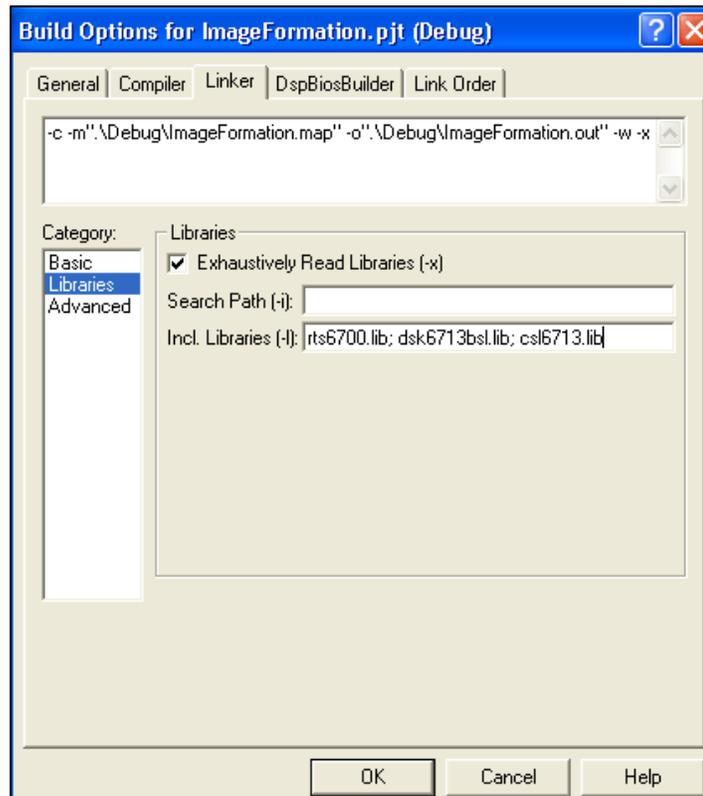


Figure 119: Libraries Needed for the Project

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.

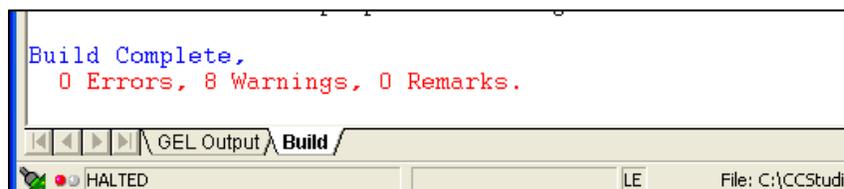


Figure 120: “ImageFormation” Project Compiling Results

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

11. Copy the following files from
C:\CCStudio_v3.3\MyProjects\ImageFormation_files\data to
C:\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.

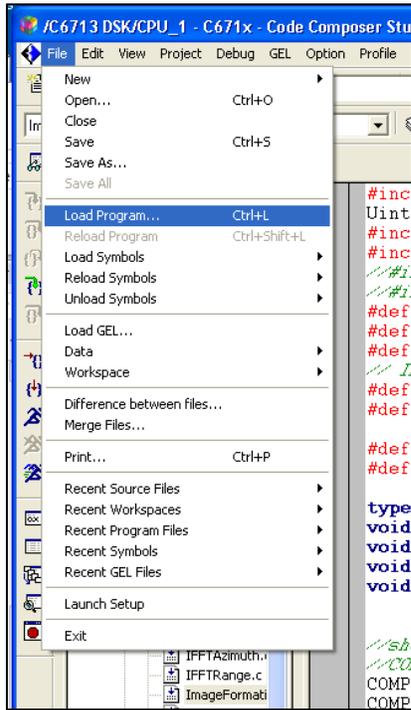


Figure 121: “Load Program” Location

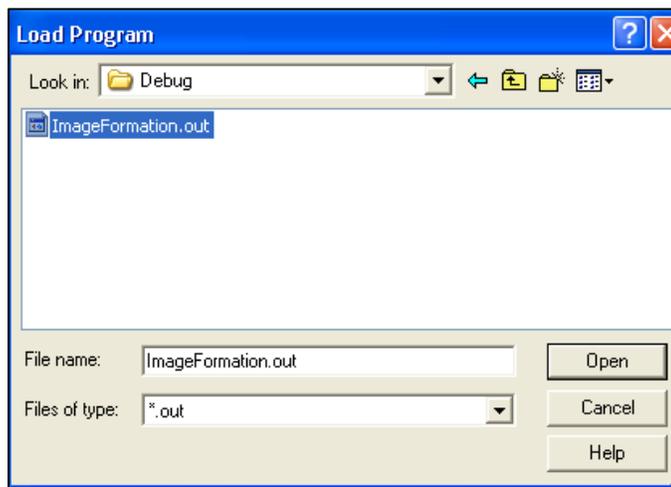


Figure 122: “ImageFormation.out” File Location

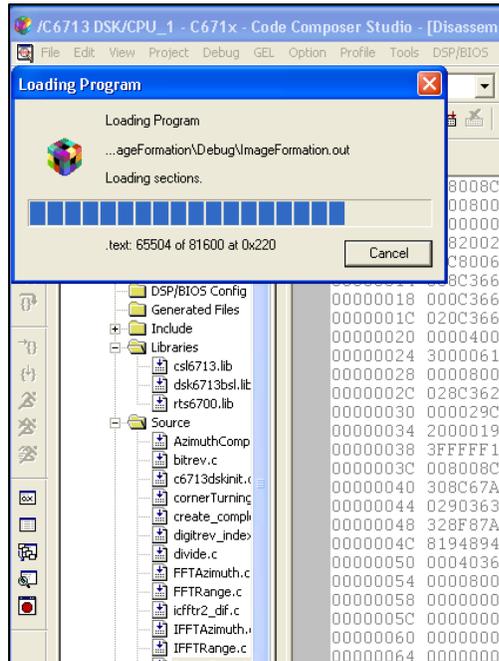


Figure 123: Downloading the “ImageFormation.out” File to the TMS320C6713 DSP

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 Applications*". 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 space-variant 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 ATTRIBUTES

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:

Table 1: TMS320C6713 DSK Features

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

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).