A HARDWARE IMPLEMENTATION OF THE IMBEDDED REFERENCE SIGNAL ALGORITHM SYSTEM USING A DIGITAL SIGNAL PROCESSING BOARD

A Dissertation presented to

The Faculty of the

Fritz J. and Dolores H. Russ
College of Engineering and Technology
Ohio University
In Partial Fulfillment
of the Requirement for the Degree
Doctor of Philosophy

by

Mohammed Fahad Alsharekh

June, 2002
Acknowledgement

The author would like to express his gratitude and sincere thanks to Dr. Jeffrey C. Dill for his endless guidance, unceasing patience and friendship. Moreover, the author would like to thank him for technical supports both in hardware and software in this dissertation research.

Sincere thanks also go to the Dr. David W. Motalak who is a member of my dissertation committee for his support, guidance and time during this research. Also, I would like to thank my other committee members: in particular Dr. Frank Van Grass, Dr. Chris Bartone, and Sergio Robert Lopez-Permouyh for their helps, ideas, and time.

I would like to express my deep and sincere thanks for my beloved parents who kept waiting for this moment, my brothers and my sisters for always being there for me. They have supported me in each step I made, even when that meant living thousands of miles apart. And Also, I would like to thank my wife Eman and my children Fahad and Abdulmalik for their endless help, support, and love though this period of time were they being away from their family.

To my friends Hamed Alsharari, Inad Aljarah, Abdulqader Alaqeeli, Ahmed Alsoliam and each person who gave me help in either direct or indirect way during this research, I offer my sincere thanks. Finally, I would like to express my thanks for all peoples who helped me during this research.
Table of Contents

Acknowledgement .......................................................................................................................... iii

List of Tables ........................................................................................................................................ x

List of Figures ....................................................................................................................................... xi

CHAPTER ONE: INTRODUCTION ......................................................................................................... 1

1.1 INTRODUCTION AND PURPOSE .............................................................................................. 1

1.2 TD/CCSK SYSTEM ......................................................................................................................... 2

1.3 RESEARCH OUTLINES ................................................................................................................... 6

CHAPTER TWO: SYSTEM OVERVIEW ................................................................................................. 7

2.1 TRANSMITTED WAVEFORM ......................................................................................................... 7

2.1.1 Addition of Reference Signal ................................................................................................... 7

2.1.2 Example Signal Transmission and Reception ........................................................................... 8

2.2 INNOVATIVE IRS ALGORITHMS ................................................................................................. 16

2.2.1 Frequency-Domain Interleave (FDI) ......................................................................................... 16

2.2.2 Reference Noise Reduction .................................................................................................... 17

2.2.3 Narrowband Excision (NBE) .................................................................................................... 18

CHAPTER THREE: ALGORITHM DESIGN ............................................................................................. 19

3.1 TRANSMITTER DESIGN ................................................................................................................ 19
3.1.1 Symbol CCSK Modulation .......................................................... 21
3.1.2 Supersymbol Buffer ................................................................. 21
3.1.3 Data/Reference Modulation ..................................................... 21
3.1.4 Interleaving ............................................................................... 21
3.1.5 Prime Permutation ................................................................. 21
3.1.6 IFFT ......................................................................................... 22
3.1.7 Transmit Sample Buffer ........................................................... 22

3.2 RECEIVER DESIGN ..................................................................... 23
3.2.1 Received Sample Buffer ............................................................ 23
3.2.2 Polyphase Filter (PPF) ............................................................... 25
3.2.3 FFT ......................................................................................... 25
3.2.4 Anti-Aliasing Filter Compensation ........................................... 25
3.2.5 Narrowband Excision (NBE) .................................................... 25
3.2.6 Data/Reference Demodulation .................................................. 26
3.2.7 Tail Clipping IFFT ................................................................. 26
3.2.8 Tail Clipping (TC) ................................................................. 26
3.2.9 Gradient Adaptive Lattice (GAL) .............................................. 27
3.2.10 Center Clipping/ Prediction to Estimation .................................. 27
3.2.11 Tail Clipping FFT ................................................................. 27
3.2.12 TD/Rake Combiner ............................................................... 27
6.1 INTRODUCTION .................................................................................................................. 48
6.2 BOARD SEARCH’S JOURNEY ............................................................................................. 48
  6.2.1 M67 Flexible DSP/Data Acquisition ........................................................................... 48
  6.2.2 TI’s Board ...................................................................................................................... 49
6.3 BOARD’S DECISION ........................................................................................................... 50
6.4 TMSC6701 EVM BUNDLED BOARD ............................................................................... 51
  6.4.1 'C6701 Processor Features ....................................................................................... 51
6.5 THE TMS320C6000 ARCHITECTURE ........................................................................... 54
  6.5.1 The Central Processing Unit (CPU) ............................................................................ 55
  6.5.2 Pipelined CPU ........................................................................................................... 61
6.6 SOFTWARE TOOLS ............................................................................................................ 62
  6.6.1 Introduction ............................................................................................................... 62
  6.6.2 Software Development Tools (SDTs) ....................................................................... 64

CHAPTER SEVEN: TMS320C6701 EVALUATION MODULE BUNDLED BOARD

7.1 INTRODUCTION ............................................................................................................... 69
7.2 KEY FEATURES OF THE TMS320C6x EVM ................................................................. 69
  7.2.2 External Interfaces ....................................................................................................... 71
7.3 THE TMS320C6701 EVM BOARD ................................................................................ 72
7.4 TMS320C6x EVM HARDWARE FUNCTIONAL OVERVIEW ........................................ 73
7.5 TMS320C6x EVM SOFTWARE FUNCTIONAL OVERVIEW .................................... 77
7.6 OPERATING SCENARIOS ..................................................................................... 80
7.7 SYSTEM REQUIREMENTS ................................................................................. 81
7.8 THE BOARD CONFIDENCE TEST ..................................................................... 82
  7.8.1 Verifying the EVM Installation ................................................................. 83
7.9 CODE COMPOSER STUDIO ............................................................................ 83
  7.9.1 Code Composer Studio Development ....................................................... 84
  7.9.2 Code Generation Tools .............................................................................. 85
  7.9.3 CCS Integrated Development Environment (IDE) .................................... 88
  7.9.4 DSP/BIOS Plug-ins ..................................................................................... 89
  7.9.5 Hardware Emulation and Real-Time Data Exchange (RTDX) .................... 89

CHAPTER EIGHT: THE HARDWARE IMPLEMENTATION AND RESULTS 90

8.1 SOFTWARE SIMULATION .................................................................................. 90
8.2 THE HARDWARE IMPLEMENTATION OF THE IRS ALGORITHM SYSTEM .......... 93
8.3 A HARDWARE PULSE PROJECT ....................................................................... 96
8.4 THE IRS ALGORITHM SYSTEM HARDWARE PROJECT .................................... 101
  8.4.1 The IRS Algorithm System's Transmitter ................................................. 101
  8.4.2 The IRS Algorithm System's Hardware ................................................... 103
8.5 RESULTS .................................................................................................................. 104
  8.5.1 Theoretical Results ......................................................................................... 106
  8.5.2 The IRS Transmitter Hardware’s Project ....................................................... 108
  8.5.3 The IRS Hardware Project’s Results and Discussions ................................. 112
  8.5.4 The IRS Hardware Project’s with Noise Results and Discussions .............. 120

8.6 Statistical Study ........................................................................................................ 122

CHAPTER NINE: CONCLUSION .................................................................................... 125
  9.1 Summary of Results ............................................................................................ 125
  9.2 Future Research .................................................................................................... 126

REFERENCES ................................................................................................................. 128
List of Tables

Table 3.1 Constants Data Element Definition for Transmitter ......................................... 22
Table 3.2 Top-Level Arrays Data Element Definition for Transmitter ............................. 23
Table 3.3 Constants Data Element Definition for Receiver ............................................. 28
Table 3.4 Local Arrays Data Element Definition for Receiver ....................................... 29
Table 3.5 Local Variables Data Element Definition for Receiver .................................. 29
Table 3.6 Top-Level Arrays Data Element Definition for Receiver ............................... 30
Table 4.1 The Total Computational Requirements for the System .................................. 42
Table 6.1 Common Compiler Options ........................................................................... 64
Table 6.2 Common Assembler Options .......................................................................... 65
Table 6.3 Frequently Used Options for the Linker ......................................................... 66
Table 6.4 Common Compiler Sections ........................................................................... 68
Table 7.1 Quad Clock Support Frequencies ................................................................. 70
Table 7.2 Operating Scenarios Application Loading Summary ...................................... 80
Table 8.1 The statistical study summary for different parameters ................................. 124
List of Figures

Figure 1.1 Block diagram of TD/CCSK transmitter with transmitted reference signal .....2

Figure 1.2 Block diagram of TD/CCSK receiver with transmitted reference signal ........3

Figure 2.1 Initial data signal .................................................................................................9

Figure 2.2 Transmitted signal ...............................................................................................10

Figure 2.3 Multipath channel impulse response ..................................................................11

Figure 2.4 Received signal ....................................................................................................12

Figure 2.5 Received signal with additive noise .................................................................13

Figure 2.6 Despread received symbol ..................................................................................14

Figure 2.7 Despread received symbol with noise reduction ...............................................15

Figure 3.1 Transmitter main flow graph ..............................................................................19

Figure 3.2 Interpretation key for transmitter .......................................................................20

Figure 3.3 Receiver main flow graph ...................................................................................24

Figure 4.1 The performance of IRS system against different normal HF channels ........39

Figure 4.2 The performance of IRS system against different disturbed HF channels ....40

Figure 6.1 TMS320C6000 block diagram ............................................................................54

Figure 6.2 The CPU of the TMS320C6000 board .................................................................55

Figure 6.3 Illustration of the program fetch phases .............................................................56

Figure 6.4 The C6701 data paths diagram .........................................................................57
Figure 6.5 Address cross-paths .................................................................58
Figure 6.6 Pipelined Vs. non-pipelined CPU ...........................................62
Figure 7.1 TMS320C6701 EVM layout (top side) .....................................73
Figure 7.2 TMS320C6701 EVM layout (bottom) .......................................73
Figure 7.3 TMS320C6x EVM block diagram ..........................................74
Figure 7.4 TMS320C6x EVM host support software block diagram ..........78
Figure 7.5 TMS320C6x EVM DSP support software block diagram ..........79
Figure 7.6 Code Composer Studio’s components ......................................85
Figure 7.7 Software development flow ....................................................86
Figure 8.1 Pulse project view .................................................................97
Figure 8.2 The dis-assembly window in the pulse project .......................98
Figure 8.3 The output view of the pulse project .......................................99
Figure 8.4 Some of the code composer studio features ..........................100
Figure 8.5 The IRS project building’s output .........................................103
Figure 8.6 The IRS project’s view, building’s output, and disassembly window ....110
Figure 8.8 The IRS system’s output for M=16 samples ...........................114
Figure 8.9 The IRS system’s output for M=32 samples ...........................114
Figure 8.10 The IRS system’s output for M=64 samples ...........................115
Figure 8.11 The IRS system’s output for M=128 samples .......................116
Figure 8.12 The base vector for the IRS’s with M=128 ...........................117
Figure 8.13 The reference vector for the IRS’s with $M=128$ ........................................117
Figure 8.14 The transmitted signal for the IRS’s with $M=128$ ........................................119
Figure 8.15 The input signal and the output signal of the IRS’s with $M=128$ ..............119
Figure 8.16 The IRS system’s output for $M=128$ samples with the noise ..................121
Figure 8.17 The noise signal for $M=128$ samples .........................................................121
Figure 8.18 The transmitted signal plus the noise for the IRS’s with $M=128$ ..........122
Chapter One: Introduction

1.1 Introduction and Purpose

In the last three decades, wireless communication has grown exponentially. This growth requires wireless communication experts, devices, maintenance, and the orientation of a large number of users. These problems are becoming more complicated, especially in areas that are crowded with electromagnetic wave signals. Moreover, the bandwidth is almost always occupied in these areas, which yields interference among the users. Because of these circumstances, wireless communications experts devised a new technique designed especially for coping with interference among users, as well as the bandwidth problems. This scheme is called spread spectrum, where the bandwidth of the transmitted signal is much greater than the bandwidth of the information signal itself. Therefore, this technique generates other modern schemes and causes the competition to increase among companies and individuals who have an interest in such subjects. Thus, the innovation of new techniques is happening every day; however, the invention of the Code Division Multiple Access—or what is called CDMA—brings a furor among the wireless communication experts themselves. CDMA is a multiple access technique and is considered one of the spread spectrum applications. Since in CDMA many users can easily share the bandwidth, the bandwidth problem is almost solved by this technique.
In 1992, Smallcomb [1] proposed a new scheme called Transform Domain Cyclic Code Shift Keying (TD/CCSK), which is one kind of CDMA. In addition, in 1995 Dill proposed a new modulation algorithm called Imbedded Reference Signal (IRS) modulation [2]. This modulation technique is a method for spread spectrum low probability detection communication over HF channels using transform domain signal processing. In this research, one of our goals is to implement the IRS algorithm experimentally and study its performance.

1.2 TD/CCSK System

Comparing the TD/CCSK transmitter and receiver system with the conventional systems, TD implementation exists in CCSK demodulation, as shown in Figures 1.1 & 1.2. In addition, the implementation of the Rake correlation is added to the TD/CCSK receiver to combine the components of the multipath and reduce the complexity of the required DFT operation. After the received signal passes through the first two orientated stages, it yields base band vector \( r \), and this vector is expressed as [1]

![Figure 1.1 Block diagram of TD/CCSK transmitter with transmitted reference signal [1]](image)
The received sequence is transformed to the frequency domain to yield the TD received vector $\mathbf{R}$. Since the TD received vector $\mathbf{R}$ and TD base vector $\mathbf{s}_o$ are both in frequency domain, the TDCCSK demodulation is implemented as a multiplication $\mathbf{R}$ by the complex conjugate of the base vector $\mathbf{s}_o$. Also, the output of the Rake Correlator $\mathbf{C}$ is implemented as a multiplication of the output of the TDCCSK demodulation by the complex conjugate of TD channel impulse response estimate $\mathbf{h}_o^*$, so the output is given by
However, there is no need for the Rake correlator if these conditions have been met [1]:

- The channel is single time-invariant path.
- The channel has unity attenuation.
- The receiver has perfect timing synchronization.

In other words, the TD channel impulse response estimate $H_0$ is all one ($H_0=1$). Therefore, the output vector $C$ is

$$C = R S_0^\ast H_0^\ast$$  \hspace{1cm} (1.2)$$

After taking the inverse Fast Fourier Transform (IFFT) of the output vector $C$, vector $c$ in the time domain will be considered. And this output vector can be represented as

$$c(n) = \sum_{k=0}^{M-1} \left( \sum_{p=0}^{M-1} r(n+p+k) s_o^\ast(p) \right) h_o^\ast(k)$$  \hspace{1cm} (1.4)$$

where, $n=0,1,2,\ldots\ldots,M-1$ And $r$ is the received signal vector with length $M$, $s_o^\ast$ is the conjugate of the base vector of length $M$ in time domain, and $h_o^\ast(k)$ is the conjugate of the impulse response of the channel with length $M$. Assume the case of one pulse at position $m$ of the sequence has been sent and the remaining are zeros. In the case of the ideal channel, the output of the IFFT $c$ should have a strong correlation peak at the position where the pulse has been sent. Thus, the optimum receiver, as picking the largest positive real time of $U_i$, can choose the symbol decision.

$$U_i = \text{Re}[c(n) \delta(n-i)]$$  \hspace{1cm} (1.5)$$
There are several methods to determine the estimate of the channel impulse response \( H_0 \). In one method, the reference base vector \( s_r \) is added before the shift register to the modulated CCSK base vector \( s_m \) and transmits with \( s_m \) at the same time (see Figure (1.1)). The reference vector \( s_r \) is not modulated; however, it is a CCSK base vector. Because the receiver knows the reference base vector \( s_r \), a priori, a copy of it is stored at the receiver. Therefore, no decision feedback is required to update the channel estimator. There are three properties that have been chosen to be the reference base vector \( s_r \) properties[1]:

- Since the reference base vector \( s_r \) is a CCSK base vector,
  \[
  R_{sr}(\tau) = E_s \quad \text{for} \quad \tau = 0
  \]
  \[
  = 0 \quad \text{for} \quad \tau = 1, 2, 3, \ldots \ldots \ M-1 \quad (1.6)
  \]
- The magnitude spectrum of both reference vector \( s_r \) and base vector \( s_o \) are almost the same.
  \[
  |S_0(k)| \approx |S_r(k)|
  \]
  \[
  \quad (1.7)
  \]
  where \( k=0, 1, 2 \ldots M-1 \).
- The reference base vector \( s_r \) and the base vector \( s_o \) are not correlated, so
  \[
  R_{sosr}(\tau) \ll E_s \quad \tau = 0, 1, 2, \ldots \ll M-1
  \]
  and
  \[
  R_{sosr}(\tau) = \sum_{n=0}^{M-1} S_o^*(n) S_r(n+\tau) M
  \]
  \[
  \quad (1.8)
  \]
  where \( R_{sosr}(\tau) \) is the circular cross correlation of the \( S_o(n) \) and \( S_r(n) \).
1.3 Research Outlines

Since we reviewed the TDCCSK system in Chapter One, we are going to cover the IRS system in Chapter Two, but its algorithm design for both the transmitter and the receiver will be discussed at Chapter Three. Moreover, the performance of the system will be studied in Chapter Four; whereas, in Chapter Five will clarify the proposed work of the research. The selection of the DSP board and its study will be covered at Chapters Six and Seven, respectively, and the IRS hardware implementation is covered in Chapter Eight. Finally, Chapter Nine will summarize the dissertation and give some future work suggestions.
Chapter Two: System Overview

2.1 Transmitted Waveform

In the beginning, we have stored on a computer hard disk a sequence of bits, which we want to transmit from one place to another by using a communications system. The transmitter of this system collects a group of source bits (i.e., b) in a buffer and sends them as a symbol. Then the system converts this source of bits to a binary number, encodes them by first generating a zero vector with length of \((2^b)\), and places a single one in the location that represents the symbol. By a cyclic shift of the base vector, the system generates the rest of possible symbol [2].

2.1.1 Addition of Reference Signal

In the communication system, the channel produces distortions to the signal due to a multipath and fading effect in addition to the AWGN, interference, and frequency selective attenuation and phase shift. So the reference signal is added to the transmitter waveform to allow the receiver to measure and compensate for that distortion. The generation of the reference signal is exactly the same as the generation of the data signal. So the base vector used for the reference signal must be orthogonal or near orthogonal to the one used for the data. Finally, the data signal and the reference signal are simply added together for transmission with equal energy.
2.1.2 Example Signal Transmission and Reception

This example illustrates the communications system of the basic waveform. Also it clarifies the basic concepts of transform domain signal generation and the difficulties of the system. In this example, a single a 7-bit data symbol (0110101 or in decimal 53) is used to illustrate the operation of the transmitter, propagation, and the receiver. The transmitted reference signal is generated along with the data signal, propagated via a multipath channel, and detected by the receiver using the received reference signal to perform the rake correlation in the transform domain. The following data is used in the example:

Sample rate: 1KHz  Symbol size: 128 samples/symbol  Signal BW: 1KHz

Data Rate: 54.7 Bits/second  Processing Gain: 12.6 dB

By observing this example, we find it clearly is showing two main problems—the loss in energy caused by the truncation of the multipath energy and the amount of noise present in the reference signal, which degrades the system performance. Therefore, we can conclude from this example the following points (see the Figures 2.1-2.7):

- Some fundamentals difficulties are facing this system.
- The potentional of both transfer domain signaling and the transmitted reference signal clearly is showing in this example.
- The performance of the system is unacceptable.

So we need to add more algorithms to enhance the performance of the system [2].
Figure 2.1 Initial data signal [2]
Figure 2.2 Transmitted signal [2]
Figure 2.3 Multipath channel impulse response [2]
Figure 2.4 Received signal: the received signal after propagation through the multipath channel [2]
Figure 2.5 Received signal with additive noise: the sold curve represents the signal plus noise, and the dashed curve represents the signal only [2]
Figure 2.6 Despread received symbol [2]
Figure 2.7 Despread received symbol with noise reduction [2]
2.2 Innovative IRS Algorithms

To enhance the system performance, some innovation is studied and applied to the system. In addition to the transmitted reference signal and frequency domain rake operation, the following algorithms are innovated: Frequency Domain Interleaving (FDI), reference noise reduction, and narrow band excision.

2.2.1 Frequency-Domain Interleave (FDI)

The length of transmitted symbol should be much larger than the worst-case delay spread of the channel to maximize the performance of the system. However, the length of the transmitted signal is limited to the size of FFT algorithm and to the data rate required of the system. So, we can define the FDI as the method of eliminating the dependency between data rate and CCSK symbol length. FDI method can be done by taking small symbols at the transmitter, combining them into one large (i.e., supersymbol) at the transmitter, and applying IFFT on the large symbol prior to the transmission. At the receiver, return the signal to its FD by applying the long FFT to the received signal and then reorder the samples to form the original set of the small symbol and apply the symbol decision after performing the Q-IFFT to the signal. By doing FDI, we make the data rate and symbol length independent of each other.
2.2.2 Reference Noise Reduction

In order for the transmitted reference signal to provide a good estimate of the channel impulse, the set of the noise reduction signal processing operations should be applied to the reference signal prior to estimating the channel. These operations are:

2.2.2.1 Tail Clipping

After applying the FDI, the size of the symbol enlarges (i.e. \( N = Q \cdot L \) samples/symbol). Tail clipping is the method that zeros out the last \( N - L \) components to reduce the noise in the channel. This method can be accomplished in two ways: first, by applying a low pass filter to the reference signal with the time domain and frequency domain reversed, and second by applying IFFT to despread the reference signal and zeroing out all the components past the tail-clipping threshold.

2.2.2.2 Gradient Adaptive Lattice

It is a further noise reduction operation that estimates the dynamic of the channel impulse response. For time invariant channel, a long average is used to remove most of the noise from the individual channel taps. However, when the channel is changing rapidly (time varying channel), the algorithm accurately tracks the channel.

2.2.2.3 Center Clipping

To identify the modes of the channel impulse and to eliminate all taps that are not part of this mode, the center-clipping algorithm is designed. The center-clipping algorithm is more accurate than tail clipping for removing the noise and is enhanced by
the GAL algorithm because of its identification of the modes of the channel impulse response [2].

2.2.3 Narrowband Excision (NBE)

For a communication system that works in a severe HF environment, two components are required. First, improve the spectral contentment and combat the narrowband interference. Polyphase filter and the dynamic narrowband excision algorithm solve these two problems, respectively.

2.2.3.1 Polyphase Filter

Polyphase filter serves as the window that improves the spectral containment without any major problem. It is implemented as a bank of P-order FIR filters—one for each of the points in the supersymbol vector. Its coefficients are calculated offline and stored in memory.

2.2.3.2 Adaptive Narrowband Excision Algorithm

By using narrowband excision algorithms, all frequency components (with the large amplitude) that contain narrowband interference will be zeroed out dynamically by using a threshold. The method of computing the threshold is based on normalizing the condition mean by assuming some threshold was selected previously. A new threshold is calculated based on the mean of all the components below the old threshold (condition mean) [2].
Chapter Three: Algorithm Design

3.1 Transmitter Design

The main flow graph for the IRS transmitter-processing algorithm is illustrated in Figure 3.1. Each block contained in the figure is assigned with the following: input signal, output signal, data element required, the necessary timing signal, and suggested implementation phase for system development and testing. Figure 3.2 illustrates transmitter key interpretation [2].

Figure 3.1 Transmitter main flow graph [2]
Figure 3.2 Interpretation key for transmitter [2]

Timing Signals:

a represents every sample

b represents every Nth sample (supersymbol rate)

c represents every Mth sample (symbol rate)

Implementation Phase:

1 means basic

2 means permutation

3 means narrowband excision

4 means channel estimator
3.1.1 Symbol CCSK Modulation

This block simply interprets the binary data (one symbol) into decimal number, and then assigns a one in the place where the data is represented. Finally, it takes the FFT of the signal.

3.1.2 Supersymbol Buffer

The operation of this block is the generation of the long sequence (supersymbol) that contains a Q (the tail clipping length) sample.

3.1.3 Data/Reference Modulation

This block is responsible for spreading the reference and the data symbols by performing a component-wise multiply of the input supersymbol vector with the data base vector and then adding the reference signal base vector (all in frequency domain).

3.1.4 Interleaving

This block is responsible for reordering the interleaved components of the individual symbol into a set of concatenated symbols (each of length M samples).

3.1.5 Prime Permutation

To avoid any correlation of the signal vector with any environment, a prime permutation reordering of the interleaved vector components of the supersymbol by using a simple prime permutation algorithm is done. However, the receiver requires being familiar with this reordering in order to reconstruct the original data signal.
3.1.6 IFFT

In order to return the signal to its time domain, this block performs the inverse of FFT of the signal.

3.1.7 Transmit Sample Buffer

To prepare the signal for the transmission, this block performs a parallel-serial buffer. Tables 3.1 and 3.2 give the transmitter data element definitions.

Table 3.1 Constants Data Element Definition for Transmitter

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
<th>Recommended Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Length of symbol</td>
<td>1024</td>
</tr>
<tr>
<td>L</td>
<td>Interleave Ratio</td>
<td>16</td>
</tr>
<tr>
<td>N</td>
<td>Length of supersymbol</td>
<td>M*L= 16384</td>
</tr>
<tr>
<td>ρ</td>
<td>Permutation prime Number</td>
<td>17</td>
</tr>
</tbody>
</table>

Where,

R real numbers

C complex numbers

I integer numbers

S static (computed offline and can be stored in ROM)

D dynamic (recomputed dynamically, and values must be preserved)

T temporary (recomputed dynamically, and values need not be preserved)
Table 3.2 Top-Level Arrays Data Element Definition for Transmitter

<table>
<thead>
<tr>
<th>Top-Level Arrays</th>
<th>Description</th>
<th>Size</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Input Data Symbol</td>
<td>Lx M</td>
<td>CD</td>
</tr>
<tr>
<td>S'</td>
<td>FFT of s</td>
<td>Lx M</td>
<td>CD</td>
</tr>
<tr>
<td>S</td>
<td>Supersymbol Vector</td>
<td>N</td>
<td>CD</td>
</tr>
<tr>
<td>S₀</td>
<td>Data Symbol Base Code</td>
<td>N</td>
<td>CD</td>
</tr>
<tr>
<td>S_r</td>
<td>Reference Symbol Base Code</td>
<td>N</td>
<td>CD</td>
</tr>
<tr>
<td>SY”</td>
<td>Combine Ref/Data Symbol</td>
<td>N</td>
<td>CD</td>
</tr>
<tr>
<td>SY'</td>
<td>Interleaved Supersymbol</td>
<td>N</td>
<td>CD</td>
</tr>
<tr>
<td>SY</td>
<td>Permutated Supersymbols</td>
<td>N</td>
<td>ID</td>
</tr>
<tr>
<td>t</td>
<td>Transmitted Signal</td>
<td>N</td>
<td>ID</td>
</tr>
</tbody>
</table>

3.2 Receiver Design

The main flow graph for the IRS receiver processing algorithms is illustrated in Figure 3.3. Each block contained in the figure is assigned with the following: input signal, output signal, data element required, the necessary timing signal, and suggested implementation phase for system development and testing [2].

3.2.1 Received Sample Buffer

This is a FIFO buffer that collects the most three recent supersymbols. These three supersymbols are the input of the next stage (polyphase filter).
Figure 3.3 Receiver main flow graph [2]
3.2.2 Polyphase Filter (PPF)

As shown earlier, the function of the filter is to improve spectral containment without major processing loss and serves as a window. For each of the N-points in the supersymbols vector, a one of P-order FIR filter is implemented the polyphase filter (nominally N=16384, P=3). However, the coefficients of the FIR filter are calculated offline and stored in memory.

3.2.3 FFT

This block performs a N-Points FFT to transform the received signal into the frequency domain.

3.2.4 Anti-Aliasing Filter Compensation

The phase and the magnitude distortion caused by the anti-aliasing filter are digitally compensated to improve the channel impulse response. The coefficients of the vector are calculated offline and stored in memory.

3.2.5 Narrowband Excision (NBE)

NBE algorithm is done once during the system process per supersymbol. The function of the NBE is determining the threshold and eliminating all frequency domain components that exceed the threshold in the supersymbol that contains narrowband interference.
3.2.6 Data/Reference Demodulation

This block is responsible for despreading the reference and data symbol by performing a component-wise multiply of the input supersymbol vector with a complex conjugate of the reference base-vector. This is accomplished perfectly by using the same base-vector that was used in the modulation process. This block has two output vectors with the same length (i.e., N): one is denoted as D and another one is denoted as RF. The D output represents the despread data vector, whereas the RF output vector represents the despread reference signal.

3.2.7 Tail Clipping IFFT

This block is executed despread reference vector. To facilitate the tail clipping operation, the despread signal is transformed to the time domain to collapse the multipath signal energy into small portion.

3.2.8 Tail Clipping (TC)

The purpose of this block is to eliminate all the components of the reference signal that are known to be beyond the worst case delay spread of the channel. The length of the tail clipping needs a design trade-off since the short one may remove more noise; however, it may remove some signal energy. The length of the tail clipping (Q) should be in the power of 2 for processing efficiently. So for this research, N=16384 samples, Q = 4096, or Q = 2048.
3.2.9 **Gradient Adaptive Lattice (GAL)**

GAL generates the variables that are used for the channel estimate and noise reduction. These variables are used for performing the next step (center clipping), channel prediction, and for Doppler correction.

3.2.10 **Center Clipping/ Prediction to Estimation**

The taps of the tail clipping and channel impulse response are weighted by this algorithm according to how well the GAL is able to predict these taps. The taps will be eliminated if the prediction is poor, but they are weighted according to the GAL predictability factor.

3.2.11 **Tail Clipping FFT**

After applying the noise reduction process, the reference signal should have most of the noise removed and should be returned to its frequency domain to perform the rake correlation.

3.2.12 **TD/Rake Combiner**

This block performs a component-wise multiply of the received data symbol with the channel estimate.

3.2.13 **Coherent Noise Cancellation**

This block improves the system performance by removing the auto-correlation noise from the correlator output.
3.2.14 Prime De-Modulation

In order to reconstruct the original signal, reorder the permuted vector components of the supersymbol. This is nothing more than the inverse of the operation that has been done in the transmitter.

3.2.15 De-Interleave

This block is responsible for reordering the interleaved components of the individual symbols into a set of concatenated symbol (each of the length of M samples).

3.2.16 IFFT

To prepare the received signal to the symbol decision, this block returns the L symbols back to time domain by applying IFFT. Table (3.3-3.6) show data element definition for receiver.

Table 3.3 Constants Data Element Definition for Receiver

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
<th>Recommended Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Length of Symbol</td>
<td>1024</td>
</tr>
<tr>
<td>L</td>
<td>Interleave Ratio</td>
<td>16</td>
</tr>
<tr>
<td>N</td>
<td>Length of Supersymbol</td>
<td>M*L = 16384</td>
</tr>
<tr>
<td>P</td>
<td>Length of Polyphase Filter</td>
<td>3</td>
</tr>
<tr>
<td>Q</td>
<td>Length of Tail Clip Window</td>
<td>4096</td>
</tr>
<tr>
<td>G</td>
<td>Number of GAL Filter Stage</td>
<td>3</td>
</tr>
<tr>
<td>PE</td>
<td>Target Percent Excision</td>
<td>1.0</td>
</tr>
<tr>
<td>ρ</td>
<td>Permutation Prime Number</td>
<td>17</td>
</tr>
</tbody>
</table>
### Table 3.4 Local Arrays Data Element Definition for Receiver

<table>
<thead>
<tr>
<th>Local Arrays</th>
<th>Description</th>
<th>Size</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varphi )</td>
<td>Polyphase Filter Taps</td>
<td>P x N</td>
<td>RS</td>
</tr>
<tr>
<td>( \tau_s )</td>
<td>NBE Update Factor (slope)</td>
<td>5</td>
<td>RS</td>
</tr>
<tr>
<td>( \tau_r )</td>
<td>NBE Update Factor (offset)</td>
<td>5</td>
<td>RS</td>
</tr>
<tr>
<td>( \chi )</td>
<td>Anti-Alias Filter Compensation</td>
<td>N</td>
<td>CS</td>
</tr>
<tr>
<td>( R^2 )</td>
<td>Mag. Squared of R&quot;</td>
<td>N</td>
<td>RT</td>
</tr>
<tr>
<td>( f = )</td>
<td>Forward Error</td>
<td>(G+1) x Q</td>
<td>CD</td>
</tr>
<tr>
<td>( b = )</td>
<td>Backward Error</td>
<td>(G+1) x Q x 2</td>
<td>CD</td>
</tr>
<tr>
<td>( k = )</td>
<td>Prediction Weighting Factor</td>
<td>(G+1) x Q</td>
<td>CD</td>
</tr>
<tr>
<td>( v = )</td>
<td>Update Weighting</td>
<td>(G+1) x Q</td>
<td>CD</td>
</tr>
<tr>
<td>( Z = )</td>
<td>Predictability Factor</td>
<td>Q</td>
<td>RT</td>
</tr>
</tbody>
</table>

### Table 3.5 Local Variables Data Element Definition for Receiver

<table>
<thead>
<tr>
<th>Local Variables</th>
<th>Description</th>
<th>Initial</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_r )</td>
<td>Excision Threshold (Recursive)</td>
<td>1.0</td>
<td>RD</td>
</tr>
<tr>
<td>( T_s )</td>
<td>Excision Threshold (Search)</td>
<td>0.5</td>
<td>RD</td>
</tr>
<tr>
<td>( NCM_r )</td>
<td>Normalized Condition Mean (Recursive)</td>
<td>n/a</td>
<td>RT</td>
</tr>
<tr>
<td>( NCM_s )</td>
<td>Normalized Condition Mean (Search)</td>
<td>n/a</td>
<td>IT</td>
</tr>
<tr>
<td>( N_r )</td>
<td>Not-Excised Count (Recursive)</td>
<td>n/a</td>
<td>IT</td>
</tr>
<tr>
<td>( N_s )</td>
<td>Not-Excised Count (Search)</td>
<td>n/a</td>
<td>IT</td>
</tr>
<tr>
<td>( a )</td>
<td>Auto-Correlation Term {R_n(0)}</td>
<td>n/a</td>
<td>RT</td>
</tr>
</tbody>
</table>
Table 3.6 Top-Level Arrays Data Element Definition for Receiver

<table>
<thead>
<tr>
<th>Top-Level Arrays</th>
<th>Description</th>
<th>Size</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Stored Sampled Received Signal</td>
<td>P x N</td>
<td>CD</td>
</tr>
<tr>
<td>r'</td>
<td>Post PPF Signal {r(n); (P-1)N≤n≤PN-1}</td>
<td>N</td>
<td>CD</td>
</tr>
<tr>
<td>R'</td>
<td>FFT of r'</td>
<td>N</td>
<td>CD</td>
</tr>
<tr>
<td>R''</td>
<td>Post Filter Compensation</td>
<td>N</td>
<td>CD</td>
</tr>
<tr>
<td>R'''</td>
<td>Post Narrowband Excision</td>
<td>N</td>
<td>CD</td>
</tr>
<tr>
<td>rf</td>
<td>IFFT of RF</td>
<td>N</td>
<td>CD</td>
</tr>
<tr>
<td>rf'</td>
<td>Post Tail Clip {rf(n); 0≤n≤Q-1}</td>
<td>Q</td>
<td>CD</td>
</tr>
<tr>
<td>rf''</td>
<td>IFFT of RF''</td>
<td>Q</td>
<td>CD</td>
</tr>
<tr>
<td>rf'''</td>
<td>Zero Padded Version of rf''</td>
<td>N</td>
<td>CD</td>
</tr>
<tr>
<td>RF</td>
<td>Reference Symbol</td>
<td>N</td>
<td>CD</td>
</tr>
<tr>
<td>RF''</td>
<td>FFT of rf''</td>
<td>N</td>
<td>CD</td>
</tr>
<tr>
<td>D</td>
<td>Data Symbol</td>
<td>N</td>
<td>CD</td>
</tr>
<tr>
<td>S_0</td>
<td>Data Symbol Base Code</td>
<td>N</td>
<td>CD</td>
</tr>
<tr>
<td>S_1</td>
<td>Reference Symbol Base Code</td>
<td>N</td>
<td>CD</td>
</tr>
<tr>
<td>S</td>
<td>Combine Ref/Data Symbol Base Code</td>
<td>N</td>
<td>CD</td>
</tr>
<tr>
<td>K'</td>
<td>First Row of $k$</td>
<td>Q</td>
<td>CD</td>
</tr>
<tr>
<td>sy</td>
<td>IFFT of SY</td>
<td>L x M</td>
<td>CD</td>
</tr>
<tr>
<td>SY</td>
<td>Demodulation Symbols</td>
<td>L x M</td>
<td>CD</td>
</tr>
<tr>
<td>U</td>
<td>Inphases Symbols</td>
<td>L</td>
<td>ID</td>
</tr>
<tr>
<td>V</td>
<td>Quadrature Symbols</td>
<td>L</td>
<td>ID</td>
</tr>
<tr>
<td>Samp</td>
<td>Sampled Received Signal</td>
<td>N/a</td>
<td>CD</td>
</tr>
</tbody>
</table>
3.2.17 Symbol Decision

This block determines the most likely transmitted symbol values by choosing the maximum component in each symbol vector.
Chapter Four: System Performance

4.1 Theoretical Performance

4.1.1 TD/CCSK with Transmitted Reference

The vector $s_r$ is denoted as the reference signal vector and is transmitted with the modulated signal vector $s_m$ simultaneously over L diversity channel. Moreover, they have the same conditions (i.e., equal energy). So the baseband sampled received signal vector $r$ is

$$r(n) = \sum_{l=0}^{L-1} \left[ \frac{s_m(l) + s_r(l)}{\sqrt{2}} \right] h^*(n-l) + z(n). \quad (4.1)$$

where $n=0,1,...,L+M-1$

After one symbol period (M points), the received signal is truncated to let $n$ vary from (0 to M-1). So the reference signal and the data would be distorted by this truncation. Thus, the loss due to this truncation is called processing loss and is denoted as ($\varphi_t$) and is given to be [3]

$$\varphi_t = 1 - \frac{L}{2M} \quad (4.2)$$

when $M>>L$, the loss is negligible.

At the receiver, the received signal vector $r$, which is in the time domain, is transformed to frequency domain to yield $R$. Then the transformed $R$ is multiplied by both the FFT of $s_o$ and $s_r$ to yield the output of the TD/CCSK demodulation ($S_o^* R$ and
Thus, the output of the reference TD/CCSK ($S_R^* R$) is used to estimate the TD channel $\hat{H}_o$. For the ideal case, the channel impulse response $H_o$ is well known to the receiver, so $\hat{H}_o = H_o$.

When we expand the output of the data TD/CCSK demodulation, we get [3]

$$R(k)S_o^*(k) = \sqrt{\frac{\varphi_t}{2}} \left( S_m(k)S_o^*(k) + S_r(k)S_o^*(k) \right) H(k) + z(k)S_o^*(k). \quad (4.3)$$

The second term of Equation (4.3) can be cancelled since $S_o$ and $S_r$ are already known to the receiver. So Equation (4.3) becomes

$$R(k)S_o^*(k) = \sqrt{\frac{\varphi_t}{2}} (S_m(k)S_o^*(k)) H(k) + z(k)S_o^*(k). \quad (4.4)$$

According to [1], for ideal transmitted reference case, the decision variable PDFs is found as

$$U_i = \frac{\varphi_t E_s}{2} \text{Re} [R_s(i-m)] + \sqrt{\frac{\varphi_t E_s}{2}} \text{Re} R_{\hat{h}_s}(i). \quad (4.5)$$

Since
\[
\text{Re}[R_h(\tau)] = |G[2L\sigma_h^2, 4L\sigma_h^4]| \quad \text{for } \tau = m
\]
\[
\text{Re}[R_h(\tau)] = |G[0, 2(L - |m - \tau|)\sigma_h^4]| \quad \text{for } m - L < \tau < L = m \quad (4.6)
\]
\[
\text{Re}[R_h(\tau)] = |0| \quad \text{else}
\]

and since

\[
\text{Re} = [R_{hz}(\tau)] = G[0, 2L\sigma_h^2, \sigma_z^2]. \quad (4.7)
\]

where \( G[\bar{X}, \sigma^2] \) is the Gaussian probability density function and is defined as

\[
G[\bar{X}, \sigma^2] \equiv f_X(x) = \frac{1}{\sqrt{2\pi}\sigma_X} e^{-\frac{(x-\bar{X})^2}{2\sigma_X^2}} \quad (4.8)
\]

therefore, the statistics for the decision variable becomes

\[
U_i = G\left[\varphi, E_z L\sigma_h^2, \varphi_i E_i L^2 \sigma_h^4, \frac{\varphi_i E_i N_o L \sigma_h^2}{2}\right] \quad \text{for } (i = m)
\]
\[
U_i = G\left[0, \varphi_i E_i L^2 \sigma_h^4 + \varphi_i E_i LN_o \sigma_h^2\right] \quad \text{else} \quad (4.9)
\]
To find the figures of merit of this scheme, we need to calculate both of these functions \( f_1(\gamma_s) \) and \( f_2(\gamma_s) \) where

\[
f_1(\gamma_s) = \frac{X}{\sqrt{2\sigma_y}} = \frac{\varphi_i E_s L \sigma_h^2}{\sqrt{\varphi_i^2 E_s^2 L \sigma_h^4 + \varphi_i E_s N_o L \sigma_h^2}}
\]

\[
f_1(\gamma_s) = \sqrt{\frac{\varphi_i \gamma_s}{\varphi_i \gamma_s + 2}}
\]

\[f_1(\gamma_s) \quad \text{(4.10)}\]

and

\[
f_2(\gamma_s) = \frac{\sigma_s}{\sigma_y} = \sqrt{\frac{2 \varphi_i^2 E_s^2 L \sigma_h^4 + \varphi_i E_s N_o L \sigma_h^2}{\varphi_i E_s^2 L \sigma_h^4 + E_s N_o L \sigma_h^2}}
\]

\[
f_2(\gamma_s) = \sqrt{1 + \frac{f_1(\gamma_s)^2}{L}}
\]

\[f_2(\gamma_s) \quad \text{(4.11)}\]

In general, due to sending the transmitted reference, there is a loss called the "processing loss" \( \varphi_r \). This loss is given by

\[
\varphi_r = \frac{1}{2}
\]

\[\text{(4.12)}\]

Therefore, the figure of merit \( f_1(\gamma_s) \) is given by
4.2 Analysis of System Performance

The transmitter generates the IRS signal, and then its output samples are stored to the disk file number 1. The input of MITER HF is the samples stored in disk file number 1, and the output of the MITER HF is stored in the disk file number 2. So, the disk file number 2 contains the IRS transmitted signal seen by the receiver. Finally, the data that is stored on disk, file number 2, can be the input of the receiver to test the receiver model [2].

4.2.1 Channel Characteristics

The IRS system was executed against a set of six channels, three normal channels, and the other three are disturbed channels.

The normal channels are:

- Channel I – single mode, with 10μ seconds of delay spread.
- Channel II – single mode, with 300μ seconds of delay spread.
- Channel III – two modes with 10 and 100 μ seconds of delay spread, respectively (100 μ seconds of separation between the modes).

The disturbed channels are:

- Channel IV – 144 μ seconds of delay spread, 12 Hz of Doppler spread.
• Channel V – 500 μ seconds of delay spread, 3 Hz of Doppler spread.

• Channel VI – 215 μ seconds of delay spread, 6 Hz of Doppler spread.

In addition to these channels, there are more channels will be used in the implementation work.

4.2.2 System Configuration

The IRS system configuration is as follows:

• Bandwidth: 1 MHz

• Sampling Rate: 1 MHz

• Supersymbol Size 16,384 samples

• Symbol Size 1,024 samples

• Interleave Ratio 16:1

• Data Rate 9.76 K bits/sec

• Processing Gain 20.1dB

• Polyphase Filter 3 taps

• Tail Clipping 4,096 sample threshold

• Center Clipping Yes

• GAL Yes, 3-taps filters

• Reference Signal equal power
4.2.3 Detailed Performance Data

Figures 4.1-4.2 show the performance of the IRS system against a different HF channel. Each figure includes a combination of simulation data and theoretical data. For the IRS system, the dashed line in the figure represents the theoretically optimum performance of the IRS system. This performance was calculated by assuming an equal power reference signal transmission and the reference is noiseless. However, the solid line shows the realistic theoretical operation range of the IRS system against a particular channel. The * curve represents the actual simulation data of the IRS system. Finally, the+ line represents the performance of the MITER wide-bandwidth HF (WBHF) rake modem against the simulated channel.

We can conclude from these figures the following points. First, the experimental simulation data and the theoretically predicted agree with each other in all cases. Second, the disturbed channel showed the advantages of the IRS system, where the transmitted signal reference allows the receiver to track channel dynamics. Also, due to the M-ary orthogonal coding gain of the IRS system and the benefit of the reference signal, the IRS system outperforms the WBHF rake in all cases [2].

4.3 Computational Requirements

With the advance of rapidly growing technology, most of the IRS limitations have been released. In the past, the IRS limitation was the significant computational load required for the transform domain processing. So we need to study the detailed analysis
Figure 4.1 The performance of the IRS system against different normal HF channels
Figure 4.2 The performance of the IRS system against different disturbed HF channels.
of computational requirements for the IRS system. The analysis of the computational requirements will focus in the receiver algorithm since the computations at the transmitter algorithm are much less than in the receiver algorithms.

Table 4.1 represents the receiver configurations, such as symbol size, data rate, processing gain, and stacking. From the combination of the complex adds and multipliers (C+, C*) and real adds and multiplies (R+, R*) required for each operation, the load was computed for the IRS system. In addition, the computation load is determined by the sampling rate of the system. From Table 4.1, the total processing load is 254 MFLOPS at the base line sample rate of 1 MHz [2].
<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>SIZE</th>
<th>RPS</th>
<th>STACK</th>
<th>C+</th>
<th>C*</th>
<th>R+</th>
<th>R*</th>
<th>TOTAL FLOPS</th>
<th>FLOPS SAMPLE</th>
<th>MFLOPS SEC</th>
<th>FLOP/BIT SEC</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polyphase Filter</td>
<td>16,384</td>
<td>1</td>
<td>1</td>
<td>32,768</td>
<td>0</td>
<td>0</td>
<td>98,304</td>
<td>163,840</td>
<td>10.0</td>
<td>10.0</td>
<td>16.8</td>
<td>3.9</td>
</tr>
<tr>
<td>Supersymbol FFT</td>
<td>16,384</td>
<td>1</td>
<td>1</td>
<td>229,376</td>
<td>114,688</td>
<td>0</td>
<td>0</td>
<td>1,146,680</td>
<td>70.0</td>
<td>70.0</td>
<td>117.4</td>
<td>27.0</td>
</tr>
<tr>
<td>Anti-Aliasing Filter Compensation</td>
<td>16,384</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>16,384</td>
<td>0</td>
<td>0</td>
<td>98,304</td>
<td>6.0</td>
<td>6.0</td>
<td>10.1</td>
<td>2.4</td>
</tr>
<tr>
<td>N-band Excision</td>
<td>16,384</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>147,456</td>
<td>32,768</td>
<td>180,224</td>
<td>11.0</td>
<td>11.0</td>
<td>18.5</td>
<td>4.3</td>
</tr>
<tr>
<td>Despread Data</td>
<td>16,384</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>16,384</td>
<td>0</td>
<td>0</td>
<td>98,304</td>
<td>6.0</td>
<td>6.0</td>
<td>10.1</td>
<td>2.4</td>
</tr>
<tr>
<td>Despread Reference</td>
<td>16,384</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>16,384</td>
<td>0</td>
<td>0</td>
<td>98,304</td>
<td>6.0</td>
<td>6.0</td>
<td>10.1</td>
<td>2.4</td>
</tr>
<tr>
<td>Tall Clipping (FFT)</td>
<td>16,384</td>
<td>1</td>
<td>1</td>
<td>57,344</td>
<td>28,672</td>
<td>0</td>
<td>0</td>
<td>266,720</td>
<td>17.5</td>
<td>17.5</td>
<td>29.4</td>
<td>6.9</td>
</tr>
<tr>
<td>Tall Clipping</td>
<td>16,384</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>16,384</td>
<td>1.0</td>
<td>1.0</td>
<td>1.7</td>
<td>0.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAL</td>
<td>4,096</td>
<td>1</td>
<td>1</td>
<td>61,440</td>
<td>86,016</td>
<td>0</td>
<td>0</td>
<td>638,976</td>
<td>39.0</td>
<td>39.0</td>
<td>63.4</td>
<td>15.4</td>
</tr>
<tr>
<td>C+ Predict to Estimate</td>
<td>4,096</td>
<td>1</td>
<td>1</td>
<td>4,096</td>
<td>8,192</td>
<td>0</td>
<td>0</td>
<td>57,584</td>
<td>3.5</td>
<td>3.5</td>
<td>5.9</td>
<td>1.4</td>
</tr>
<tr>
<td>DRRake Correlation</td>
<td>16,384</td>
<td>1</td>
<td>1</td>
<td>57,344</td>
<td>28,632</td>
<td>0</td>
<td>0</td>
<td>266,720</td>
<td>17.5</td>
<td>17.5</td>
<td>29.4</td>
<td>6.9</td>
</tr>
<tr>
<td>Coherent Noise Cancellation</td>
<td>16,384</td>
<td>1</td>
<td>1</td>
<td>16,384</td>
<td>0</td>
<td>16,384</td>
<td>65,536</td>
<td>114,688</td>
<td>7.0</td>
<td>7.0</td>
<td>11.7</td>
<td>2.8</td>
</tr>
<tr>
<td>Prime Permutation</td>
<td>16,384</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16,384</td>
<td>1.0</td>
<td>1.0</td>
<td>1.7</td>
<td>0.4</td>
</tr>
<tr>
<td>De-interleave</td>
<td>1,024</td>
<td>16</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16,384</td>
<td>1.0</td>
<td>1.0</td>
<td>1.7</td>
<td>0.4</td>
</tr>
<tr>
<td>Symbol FFT</td>
<td>1,024</td>
<td>16</td>
<td>1</td>
<td>10,240</td>
<td>5,120</td>
<td>0</td>
<td>0</td>
<td>819,200</td>
<td>50.0</td>
<td>50.0</td>
<td>83.9</td>
<td>19.7</td>
</tr>
<tr>
<td>Symbol Decision</td>
<td>1,024</td>
<td>16</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16,384</td>
<td>1.0</td>
<td>1.0</td>
<td>1.7</td>
<td>0.4</td>
</tr>
<tr>
<td>Total</td>
<td>4,153,344</td>
<td>254</td>
<td>254</td>
<td>425</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Assumptions:
- Bandwidth: 1 MHz
- Sampling Rate: 1 MHz
- Supersymbol: 16,384 Sample
- Symbol: 1,024 Sample
- Bits per Symbol: 10
- Bits per Supersymbol: 160
- Data Rate: 9,766 Bps/S
- Processing Gain: 20 dB
Chapter Five: Proposed Work

5.1 Hardware Implementation

This proposed work is based on the implementation of the IRS algorithm in DSP hardware. We will benefit from advanced technology to implement and run the IRS algorithm efficiently. It is not necessary to get the board that exactly matches your algorithm needs, since the commercial board can be developed to match the level of the algorithm’s needs. Therefore, in this section we are planning to give guidelines and some ideas for choosing the desired DSP board.

5.1.1 Board Decision and Selection

First we are going to start with some discussion about the processor choice, covered as follows [4]:

- Processor Type: The IRS algorithm is FFT based, so it needs a correlation type that requires a FFT and IFFT for every group of samples.

- Arithmetic Format: Based on the accuracy and the dynamic range, the floating point is the best among the arithmetic format.

- Processing Power: Based on the computational requirements for the IRS algorithm, the processing power is 254 MFLOPS.

- Memory is any device that can be used for storing information, and the number of memory locations required to store the system parameters, input
data, output data, and intermediate computed values, represents the memory requirements for a system. For example, the size of the supersymbol is 16K samples so for a 16K points FFT requires at least 32 Kbytes, which is usually offered by most of the DSP chips.

Then, we are going to narrow our choice of the DSP board by using the following key factors [4]:

- **DSP Board Interface with the PC**: For the fastest in term of the speed is the PCI-Bus. So we select the PCI-Bus as DSP board interface.

- **Algorithm Performance**: The performance of FFT on the chip does not usually represent the performance of FFT in the DSP board for two reasons; the clock rate is slower than the maximum instruction cycle of the chip, and the on-board memory is slower sending data or program instructions to the chip than the maximum rate at which it can receive them.

- **I/O Performance**: The computing of FFT’s in a DSP chip is capable of being faster than the data I/O rate, so we need to compare the chip’s FFT benchmark with I/O data rate. By doing so, the best situation is when I/O data rate is faster than the rating which the chip performs the FFT’s.

- **Software Support**: The time required to develop the IRS algorithm is affected by the following three tools:
• Software support tools include assemblers, linkers, and compiler for writing a code.

• Simulator and debuggers to remove program errors.

• Algorithm library to reduce the amount of code that must be written.

• Expansion Capability: Rather than offering highly budget to design a board that meets all the needs of IRS algorithm’s application, we can modify a commercial board by having a daughter-card conneoter and/or prototyping area. These two upgrading options have the ability to upgrade the commercial board to meet the changing requirements.

• Language Support: Since the algorithm is designed entirely by using a Matlab code, the preferable language is Matlab. However, a C-Language is acceptable because of the existing Matlab compiler.

• Cost and Availability: This research, like any other project, has a specified, time-limited budget, so these two factors will play a role in choosing a board.

5.2 Problem Statement

5.2.1 Implementation of Basic Algorithm on Modern DSP Hardware

Based on the decision that will be taken with the regards to the board, we are planning to have a DSP board that almost matches all desired features listed in the previous section. In this DSP board, we will implement the entire IRS algorithm system using the following steps:
• Implement the transmitter of the system at the hardware with a very low supersymbol's length.

• Upgrade the system length gradually with the different values of the supersymbol length.

• Add the basic receiver to the IRS system's hardware implementation starting with a very low supersymbol length.

• Upgrade the entire system's length gradually with the different values of the supersymbol length.

• Study the performance of the IRS algorithm system experimentally by comparing the input data with the output data each time.

5.2.2 Optimization for a Real-Time Execution Speed

In this part, we are planning to run the IRS algorithm experimentally after we make all the change needed for the commercial board. For example, write and develop the software in the proper language for the IRS algorithm model for the ideal channel. Then study the experimental performance of the IRS algorithm for the ideal channel.

5.2.3 Addition of the Additive White Gaussian Noise (AWGN)

In this part, we are planning to add the noise (AWGN) to the transmitted signal and then receive the signal with the added noise. Furthermore, we will study the performance experimentally again by comparing the input data with the output data at each time.
5.2.4 Revision and Improvement Algorithm

Based on the experimental performance data for the ideal channel, we are going to revise and improve the IRS algorithm to enhance the overall performance of the system. These revision and improvement algorithms depend upon the experimental performance of the IRS for ideal channel.
Chapter Six: Board Architecture

6.1 Introduction

Based on the first part of Chapter Five, we began our search, and it had been limited by the conditions that were given. Although, the market is saturated with the board maker’s companies, we narrowed our search to two, because these companies were able to satisfy all the conditions proposed in the last chapter. These companies are Signalogic and Texas Instruments, and both of them are very well known. In Signalogic, we placed our search in platform called “M67 Flexible DSP/Data Acquisition” [5]. In contrast, in the TI we placed our search in the platform called “TMS320C6701 EVM bundled” [6].

6.2 Board Search’s Journey

In this section, we will go over both of these options, and then discuss the final decision regarding the board. This study will cover the following terms: description, features, specifications, software support, etc.

6.2.1 M67 Flexible DSP/Data Acquisition

M67 Flexible DSP/Data Acquisition board provides a flexible platform for a wide variety of DSP-based data acquisition applications. M67 boards have a high performance up to one GFLOPS besides some other features, such as on the board SRAM and SDRAM, PCI interface. Moreover, M67 boards combine the power of M67
is 32-bit floating-point with C6701 DSP devices with onboard SRAM, and SDRAM, analog and digital I/O modules signal options. M67 boards provide two daughter board sites for wide array of OMNIBUS modules. Moreover, they have telecom modules and provide two daughter board sites for standard OMNIBUS telecom modules. Signalogic off-the-shelf DSP software products support the M67 boards, and they are designed for both DSP-based data acquisition and c6x DSP code development. These software are DSPower, DirectDSP, and Hypersignal Software Support. First, Hypersignal-Macro and Hypersignal –Acoustic software series are offering a number of simulation and real-time instrument functions (including DSP and math functions and more). Then DirectDSP is a window library, which provides both low and high level calls for user-defined C/C++, Visual Basic, and Lab View programs. Also, it controls run, rest, and hold functions, where a large number of the C6xx algorithms and functions are contained in the C6xx Code Interface. Finally, Real-Time Compositor and DSPower provide the signal flow block diagram including simulation and interactive display and instrument functions, and real time code generation. In addition, Signalogic provides unlimited technical support at the time (pre-and-post sale) for M67 boards [5].

6.2.2 TI's Board

C6701 EVM board belongs to TI which allows a designer anywhere in the world to benchmark his system by innovated C6701 EVM bundled with its superior, Code Composer Studio (CCS), software support. It provides a comprehensive development
platform with CCS integrated development environment. C6701 EVM bundled saves engineers time and the expense of building their own application. The board has some features, for instance, a PCI bus interface, SBSRAM and SDRAM, a 16 bit audio code, and embedded JTAG emulation support. Also, it has an external memory interface (EMIF) and peripheral signals that enable it to be expanded with daughter boards. Moreover, C6701 board has a controller interface for power, clock, JTAG, and PCI to provide a reference for interfacing it to many types of memory peripherals. Finally, EVM hardware is divided into 12 sections; for instance, C6000 DSPs, which uses a 167 MHz version of the CPU of the board. In DSP Clock, the board has two different onboard clock sources besides a two clock mode that enables the DSP to operate at four different clock rates. The rest of the hardware is as follow: external memory, expansion interfaces, PCI interface, JTAG emulation, programmable logic, audio interface, power supplies, voltage supervision, and rest control, LED indictors, and user options [6]. (For more information see Chapter Seven.)

6.3 Board’s Decision

After studying both options carefully, we found that both boards have sufficient features and are good platforms to have at your communications lab. Even though the code in our project is written by a Matlab code and M67 Flexible DSP/Data Acquisition board provides a Matlab interface in their software support, we chose the C6701 EVM bundled that have C compiler in their software code because the TI’s technical support
is superior as is their online and hands-on training workshops. Finally, TI software support, which is called code composer studio (CCS), is superior. Therefore, in the following section, we are going to dig into depth in the C6701 EVM bundled board to study the board with the following view: description, features, and installation and system requirements.

6.4 TMSC6701 EVM Bundled Board

Since we chose TMSC6701 EVM Bundled Board from TI, in the next section we will go over its hardware and software details. However, we will go over the processor features first.

6.4.1 'C6701 Processor Features

The 'C6701 processor has a floating-point digital processor that has the following features [7]:

- Highest performance floating point digital signal processing DSP TMS320C6701.
  - Up to 6n second instruction cycle time.
  - Up to 167-MHz clock rate.
  - Eight 32-Bit Instructions cycle.
  - Performance of 1 GFLOPS.
- VelociTI™ Advanced Very Long Instruction Word (VLIW), C67x CPU core
  - Eight highly independent functional units.
- Load-store architecture with 32 32-bit general-purpose registers.
- Instruction packing reduces code size.
- All instructions conditional.

- Instruction set features.
  - Hardware support for IEEE single precision instructions.
  - Hardware support for IEEE double precision instructions.
  - Byte-Addressable (8-, 16-, 32-Bit data).
  - 8-Bit overflow protection.
  - Saturation.
  - Bit-field extract, set, clear.
  - Bit counting.
  - Normalization.

- 1 M-Bit On-Chip SRAM.
  - 512 K-Bit internal program/cache (16 K 32-Bit instructions).
  - 512 K-Bit dual-access internal data (64K Bytes).

- 32-Bit External Memory Interface (EMIF).
  - Glueless interface to synchronous memories: SDRAM and SBSRAM.
  - Glueless interface to asynchronous memories: SRAM and EPROM.
  - 52 M-Byte addressable external memory space.
• Four-Channel bootloading Direct Memory Access (DMA) controller with an auxiliary channel.

• 16-Bit Host Port Interface (HIP)
  ▪ Access to entire memory map.

• Two Multichannel Buffered Serial Ports (McBSPs).
  ▪ Direct interface to T1/E1, MVIP, SCSA framers
  ▪ ST-bus-switching compatible.
  ▪ Up to 256 channels each.
  ▪ AC97-compatible.
  ▪ Serial-Peripheral-Interface (SPI) compatible (Motorola™).

• Two 32-Bit general-purpose timers.

• Flexible Phase-Locked-Loop (PLL) clock generators.

• IEEE-1149.1 (JTAG) boundary-scan-compatible.

• 352-pin Ball Grid Array (BGA) package (GJC suffix).

• 0.18-μm/5-Level Metal Process.
  ▪ CMOS Technology.

• 3.3-V I/Os, 1.8-V Interval (120-, 150-MHz).

• 3.3-V I/Os, 1.9-V Interval (167 MHz only).
6.5 The TMS320C6000 Architecture

TMS320C6000 family is a TI product and has the latest and the highest performance floating point DSP processors available from TI. These processors can operate at 167 MHz (6 n second) with high performance of 1 GFLOPS. Also, the 'C6xx processors benefit from the high efficient C compiler and assembly optimizer. Figure 6.1 shows the TMS320C6000 block diagram, which consists of three main parts: the Central Processing Unit (CPU), memories, and peripherals. These three main elements are connected by the internal buses. Moreover, 'C6701 processor has an External Memory Interface (EMIF), and Host Port Interface (HPI) [8][9].

![Figure 6.1 TMS320C6000 block diagram [9]](image)
6.5.1 The Central Processing Unit (CPU)

The CPU is the heart of the processor and is divided into four following elements, as illustrated in Figure 6.2.

- Program control unit.
- Data paths.
- Control registers.
- Test, emulation, control and interrupt logic.

<table>
<thead>
<tr>
<th>Program Control Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Program fetch</td>
</tr>
<tr>
<td>• Instruction dispatch</td>
</tr>
<tr>
<td>• Instruction decode</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Path 1</th>
<th>Data Path 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Control Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test, Emulation, Control and Interrupt Logic</th>
</tr>
</thead>
</table>

Figure 6.2 The CPU of the TMS320C6000 board [9]

6.5.1.1 Program Control Unit

The program control unit consists of three elements: program fetch unit, instruction dispatch unit, and instruction decode.

In order to retrieve a fetch packet (FP), the following phases are needed:
- PG phase: the CPU generates a fetch address.
- PS phase: the CPU sends the address to the memory.
- PW phase: the CPU waits for the data to be ready.
- PR phase: the CPU reads the opcode.

First, the program addresses is generated in the PG phase (cycle n), and then sent to the program memory in the PS phase (cycle n+1). Then CPU waits for the instruction in the program memory (PW phase) to be ready before the CPU receives the fetch packet at the final stage--this will take one cycle if the program in the internal program memory or eight cycles plus the wait state cycles, otherwise. Figure 6.3 illustrates the program fetch phases. The processors of ‘C67xx have eight functional units and these units can only execute specific instructions. So instruction dispatch unit is used to help the instructions to dispatch to the appropriate units. Alternatively, the instruction opcode will be decoded when it reaches the instruction decode.

* Number of cycles depends on the memory type.

Figure 6.3 Illustration of the program fetch phases [9]
6.5.1.2 CPU Data Paths

The C6701 CPUs data paths are composed of two parts, which are called data path one and data path two, respectively. Each one of these paths have four execution units denoted as .L, .M, .S, .D, and a register file containing 16 32-Bit general purpose register and multiple data communications path. The CPU paths can be divided into the data path and address path. The data path is used for the data transfer between the memory and the register file and the units, whereas the address path is used for transferring the address from the data unit .D to memory.

6.5.1.2.1 Cross Paths

As shown in Figure 6.4, there is a cross-path between the two sides of the processor’s CPU for both data and address. These cross-paths are good, but they have some restrictions. Register file cross-paths is another name for the data cross-paths that allow one operand to cross from one side to the other. However, these points must be noted:

![Figure 6.4 The C6701 data paths diagram [9]](image-url)
• First, only one cross-path is allowed per direction per each execute packet.
• The destination register is always in the same side of the unit used.

The unit .D1 and .D2 generate addresses that are sent to the data address path DA1 or the data address path DA2, as illustrated in Figure 6.5. Generation of the address using one register file and accessing the data in the other register file is the advantage of the address cross-path. However, these points must be noted:

• Only one address cross-path is allowed per direction per each execute packet.
• The destination register for the load (LD) instructions and the source register for the store (ST) instructions should come from the opposite side of the unit when using the address cross-paths.
• Both or either none of the address cross-paths are used if both .D units are used.

Figure 6.5 Address cross-paths [9]
6.5.1.3 Execution L Unit

Both L1 and L2 are 40-bit integer arithmetic and Logic Units (ALUs), and can be used for the following:

- 32/40-bit arithmetic and compare operation.
- 32-bit logical operation.
- Normalization and bit count operations.
- Saturated arithmetic for 32/40 bit-operations.

6.5.1.4 Execution M Unit

Each of M1 and M2 is a hardware multiplier unit for the data path one and data path two respectively. These units produce a 32-bit as a result of 16-bit by 16-bit multiplications [9].

6.5.1.5 Execution S Unit

S1 and S2 contain 32-bit integer ALUs and 40-bit shifters and can be used for the following:

- 32-bit arithmetic, logic and bit field operations.
- 32/40 bit shifters.
- Branches (S only when you use a register).
- Register transfer to and from control registers (S2 only).
- Constant generation.
6.5.1.6 Execution .D Unit

Both data units D1 and D2 may be used for the following:

- Load and store with 5-bit constant offset.
- Load and store with 15-bit constant offset (.D2 only)
- 32-bit additions/subtractions.
- Linear and circular address calculations.

6.5.1.7 Control Registers

The ‘C67xx processor has 13 control registers leads the ‘C62xx processor with three registers to support the floating points operations.

6.5.1.8 Memory

Since the address bus is 32 bits wide at ‘C6701 processor, the total memory address range is 4 Gbytes \(2^{32}\). This memory can be divided into four spaces: internal program memory, internal data memory, internal peripheral memory, and external memory map (CE0, CE1, CE2, CE3). However, external memory map (CE0, CE2, and CE3) support asynchronous (SRAM and ROM) and synchronous (SBSRAM and SDRAM) memory with 8- and 16-bit read and write and 32 read only, whereas the CE1 supports 32-bit read and write or 16-bit read only. In addition, the processor provided with the direct memory access (DMA) controller allows data transfer between the internal memory and the following: external memory, host port (HP), and external
peripherals. The 'C67xx has four DMA channels and one auxiliary channel for the host port interface (HPI) [10].

6.5.1.9 Serial Ports

The 'C6701 processor have two multichannels buffered serial ports (McBSP) that support the full-duplex communications and operate at maximum speed at 40 megabits per second per channel.

6.5.1.10 Host Port Interface (HPI)

HPI is a parallel port that supports 16-bit data and provides a low-cost interface to standard microprocessor buses. Moreover, the internal, external, and memory-mapped peripherals can be used for exchange of data between the HPI and the CPU, whereas, the HPI only can access the memory space via the DMA auxiliary channel. For more features of the 'C6701 process, such as boot functions, internal timers, interrupts, etc, refer to the 'C6701 process the data sheet [7].

6.5.2 Pipelined CPU

The 'C67xx processor has the followings steps for an instruction, fetching, decoding, and execution to enhance the throughput DSP CPUs are designed to be pipelined, which means these steps are carried out at the same time. Figure 6.6 shows the difference in processing time for three instructions for both non-pipelined and pipelined CPU [11].
<table>
<thead>
<tr>
<th>CPU Type</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Pipelined</td>
<td>F₁</td>
<td>D₁</td>
<td>E₁</td>
<td>F₂</td>
<td>D₂</td>
<td>E₂</td>
<td>F₃</td>
<td>D₃</td>
<td>E₃</td>
</tr>
<tr>
<td>Pipelined</td>
<td>F₁</td>
<td>D₁</td>
<td>E₁</td>
<td>F₂</td>
<td>D₂</td>
<td>E₂</td>
<td>F₃</td>
<td>D₃</td>
<td>E₃</td>
</tr>
</tbody>
</table>

Where:
Fᵢ is fetching of instruction i.
Dᵢ is decoding of instruction i.
Eᵢ is execution of instruction i.

Figure 6.6 Pipelined Vs. non-pipelined CPU [11]

6.6 Software Tools

Since it is important for the DSP programmer to understand the use and capabilities of the software development tools, this section will cover the software development tools and the Evaluation Module (EVM) board.

6.6.1 Introduction

The software tools for the TMS320C6000 is for converting a C or assembly file into a DSP executable file including compiler, assembler, linker, debugger, and simulator. The steps for going from a source file (.C for C Language) to an executable file (.out extension) are illustrated in Figure 6.7. As is seen in Figure 6.7, the compiler
and the assembler optimizer convert the C-file (.C) and the linker assembler file (.as), respectively, into an object file (.obj). Also, the assembler box translates the assembly file (.asm) to the object file. The object file, as instructed by the command file (.cmd), is combined into an executable file on C67xx processor at the linker.

---

![Diagram](image.png)

**Figure 6.7 C6x software tools [11]**

All the assembling, linking, complying, and debugging steps have been incorporated into a software tool called Code Composer Studio (CCS) that is an integrated development environment (IDE). Also, CCS provides an easy use graphical user environment for building and debugging both C and assembler on many DSP applications [11].
6.6.2 Software Development Tools (SDTs)

The SDTs are composed of four main modules: compiler, assembler, linker, and simulator and code converter.

6.6.2.1 Compiler

The DSP processor does not understand the c source so it needs to convert it to a language that is executable and understandable for the DSP processor. However, the c programs are used to communicate with the DSP processor because of its portability, ease of use, and popularity. Moreover, optimizing the c compiler for C6x processors can achieve performances more than 70% compared with code written in assembly. Table 6.1 shows the options of the compiler that provide information about the program and the system to the compiler [8].

Table 6.1 Common Compiler Options [9]

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-mv6701</td>
<td>Tells the compiler that the code is for the 'C6701 processor.</td>
</tr>
<tr>
<td>-k</td>
<td>Do not delete the assembly file (*.asm) created by the compiler.</td>
</tr>
<tr>
<td>-g</td>
<td>Symbolic debugging directives are generated in order.</td>
</tr>
<tr>
<td>-I</td>
<td>Specifies the directory where the (#include) file reside.</td>
</tr>
<tr>
<td>-s</td>
<td>Interlists C and assembly source statements.</td>
</tr>
<tr>
<td>-z</td>
<td>Adding this to the command line will evoke the assembler and the linker.</td>
</tr>
</tbody>
</table>
### 6.6.2.2 Assembler

The assembler converts the assembly code into an object file to be executed by the processor. Table 6.2 shows the most common options that the programmer can provide.

#### Table 6.2 Common Assembler Options [9]

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-l</td>
<td>Generates an assembly-listing file.</td>
</tr>
<tr>
<td>-s</td>
<td>Puts labels in the symbolic cross-reference table to be used by the debugger.</td>
</tr>
<tr>
<td>-x</td>
<td>Option automatically evokes the -l option.</td>
</tr>
</tbody>
</table>

### 6.6.2.3 Linker

To have the final executable file, the linker combined the various object files that constitute an application. Also, the library file and the command file that describes the hardware are taken as input by the linker. Table 6.3 illustrates the most common linker options specified by the programmer.
Table 6.3 Frequently Used Options for the Linker [9]

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-o</td>
<td>Names an output file.</td>
</tr>
<tr>
<td>-c</td>
<td>Use auto-initialization at run time.</td>
</tr>
<tr>
<td>-l</td>
<td>Specifies a library file.</td>
</tr>
<tr>
<td>-m</td>
<td>Produces a map file.</td>
</tr>
</tbody>
</table>

6.6.2.3.1 Linker Command File (.cmd)

There are three functions of the linker command file: first, to describe the memory map of the system to be used to the linker, which is specified by “MEMORY {...}”; second, specified by “SECTIONS {...}”, is to tell the linker how to bind each section of the code to a specific section as defined by the memory function; and third, to provide the linker with input and output file and the options of the linker.

6.6.2.4 Directives

Directives are used to specify assembly source sections and to declare data structures. However, the assembly statements appearing as directives do not produce an executable code. Some of assembler directives are as follow:

- .sect “name” directive that defines a section of code or data named “name”.

...
• .int, .long, or .word directive that reserves 32 bits of memory initialized to a value.
• .bute or .half directive that reserves 16 bits of memory initialized to a value.

In the TI Common Object File Format (COFF), the directives .text, .data, and .bss are used to indicate code, initialized constant data, and un-initialized variables, respectively, and other directives are used for different functions, as listed below:
• .set directive is used for assigning a value to a symbol.
• .global or .def directive is used to declare a symbol or module as global.
• .end directive is used to signal the termination of the assembly code.

Moreover, the C compiler creates many sections indicated by the directives, which is shown in Table 6.4 [11].

6.6.2.5 Memory Management

The DSP processor uses either static memory (SRAM) or dynamic memory (DRAM) as an external memory. The SRAM is faster and more expensive than the DRAM because SRAM takes more space on silicon. Since the address bus is 32 bits wide, the total memory space is 4 G bytes ($2^{32}$). According to the memory map, this space is divided as follow: internal program memory (PMEM), internal data memory (DMEM), internal peripherals, and four external memory spaces called CE0, CE1, CE2, and CE3. CE0, CE2, and CE3 all have the same memory range (16 M Bytes), whereas the CE1 has a memory range of 4 M Bytes [11].
Table 6.4 Common Compiler Sections [11]

<table>
<thead>
<tr>
<th>Section Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text</td>
<td>Code.</td>
</tr>
<tr>
<td>.switch</td>
<td>Tables for switch instructions.</td>
</tr>
<tr>
<td>.const</td>
<td>Global and static string literals.</td>
</tr>
<tr>
<td>.cinit</td>
<td>Initial values for global/static vars.</td>
</tr>
<tr>
<td>.bss</td>
<td>Global and static variables.</td>
</tr>
<tr>
<td>.far</td>
<td>Global and static declared far</td>
</tr>
<tr>
<td>.stack</td>
<td>Stack (local variables)</td>
</tr>
<tr>
<td>.system</td>
<td>Memory for malloc fcns (heap)</td>
</tr>
<tr>
<td>.cio</td>
<td>Buffers for stdio functions</td>
</tr>
</tbody>
</table>
Chapter Seven: TMS320C6701 Evaluation Module Bundled Board

7.1 Introduction

The C6x EVM is a peripheral component interconnect (PCI) plug-in-card. There are many factors that let us think seriously about having such a board in the lab and using it for the implementation of our research. For example, the TMS320C6x evaluation module (EVM) is a low-cost, general-purpose platform for the development, analysis, and testing of 'C6x digital signal processor (DSP) algorithms and applications. Also, the 'C6x EVM allows you to evaluate the 'C6x DSP and algorithms to figure out if they meet your application requirements, and its hardware design information and software application programming interfaces (APIs) also provide a reference design that facilitates your own 'C6x-based hardware and software development. The EVM is bundled with many other software, such Code Composer, Windows 95 and NT drivers, host PC and DSP software APIs, example applications with source code, and various utility applications. The software and the hardware bundle provides an integrated package to facilitate the evaluation of the performance of the 'C6x DSP’s and the development of user applications.

7.2 Key Features of the TMS320C6x EVM

The 'C6x EVM has the following key features [12]:

- 'C6701 floating-point DSP.
• Different quad clock support. (See Table 7.1 for more details.)

• Peripheral component interconnect (PCI) interface.

• 256K bytes of 133-MHz synchronous burst static random-access memory.

• 8M bytes of 100-MHz synchronous dynamic RAM (SDRAM).

• Embedded JTAG emulation via PCI or external XDS510 support.

• Access to all DSP memory from the PCI bus via Host port interface.

• Stereo 16-bit audio codec that supports sample rates from 5.5 kHz – 48 kHz.

• Onboard switching voltage regulators.

• Onboard linear voltage regulator for analog 5 V DC.

• Three light emitting diode (LED) indicators.

• Internal PCI operation.

• External desktop operation (requires external power supply and XDS510 or XDS510WS emulator, which are not included in the ’C6x EVM kit).

• Expansion memory and DSP peripheral connectors for daughterboard.

• Plug-and-play PCI device.

Table 7.1 Quad Clock Support Frequencies [12]

<table>
<thead>
<tr>
<th>Processor</th>
<th>OSC A-By-1</th>
<th>OSC A-By-4</th>
<th>OSC B-By-1</th>
<th>OSC B-By-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>’C6701</td>
<td>25MHz</td>
<td>100 MHz</td>
<td>33.25 MHz</td>
<td>133 MHz</td>
</tr>
</tbody>
</table>
7.2.1 User Controls and Indicators

The 'C6x EVM has the following user controls and indicators [12]:

- JTAG emulation (embedded or external).
- DSP oscillator clock selection (OSC A or OSC B – see Table 7-1).
- Clock mode (multiply-by-1 or multiply-by-4 for either oscillator).
- DSP boot mode selection:
  - None (no-boot mode)
  - HPI boot mode
  - Read-only memory (ROM) boot mode (optional daughter-board)
- Little- or big-endian memory addressing.
- Three user-defined dual in-line package (DIP) switches.
- Manual reset.
- Power-on LED indicator.
- Two user-defined LED indicators.

7.2.2 External Interfaces

The 'C6x EVM has the following external interfaces[12]:

- PCI revision 2.1-compliant interface (5 V, 32-bit).
- Molex 4-pin external power connector (5 V, 12 V, –12 V, and GND).
- 2-pin power connector for cooling fan (5 V).
• 14-pin external JTAG header.
• 10-pin complex programmable logic device (CPLD) in-system programmable (ISP) header (not installed).
• Three 3.5-mm audio jacks (line in, line out, and microphone).
• Two low-profile 80-pin (.050-inch) daughterboard connectors (EMIF and DSP peripheral expansion).

7.3 The TMS320C6701 EVM Board

The TMS320C6701 EVM board is a full-size peripheral component interconnect (PCI) designed for use in a PCI expansion slot on the computer’s motherboard. The dimensions of the board are as follow: 4.2 inches wide, 12.28 inches long, and 0.49 inches high (excluding its bracket and retainer). Moreover, the board has ‘C6701 DSP onboard, which allows full-speed verification of ‘C6x source with the include code debugger. Also, the board provides a PCI interface, SBSRAM and SDRAM, a 16-bit audio codec, and embedded JATG emulation support, whereas the connectors on the board provide DSP external memory interface (EMIF) and peripheral signals to expand the memory by using the daughterboard. In addition, the board provides hardware a reference design and a reference for interfacing the DSP to various types of memories and peripherals. The top and the bottom sides, respectively, of the ‘C6701 EVM board are shown in Figures 7.1 and 7.2 [12].
7.4 TMS320C6x EVM Hardware Functional Overview

The ‘C6x EVM hardware divided into 12 functional areas as shown in Figure 7.3.
Figure 7.3 TMS320C6x EVM block diagram [11]
In the next list, we will give a quick overview for these 12 functional [12]:

- **DSP**: The ‘C6x EVM is built around the ‘C6701 DSP that operates up to 1 GFLOPS with a CPU clock of 133MHz.

- **DSP Clocks**: The ‘C6x EVM supports operation with two different onboard clock sources (OSC A, and OSC B) and two clock modes (multiply by 1 and by 4), as seen at Table 7.1.

- **External Memory**: The ‘C6x EVM provides one of each of the following bank: 64K X 32, 133 MHz SBSRAM and provides two of the following banks: 1M X 32, 100MHz SDRAM, besides the asynchronous memory that can be added with a daughterboard by using the expansion memory interface, and the peripheral interface that provides a daughterboard use of the DSP internal peripherals.

- **Expansion Interface**: The ‘C6x EVM provides external memory interface that supports asynchronous memory transfer, and external peripheral interface connectors that provide daughterboard use of the DSP internal peripherals.

- **PCI Interface**: the ‘C6x EVM includes a PCI local bus revision 2.1complaint interface to enable host access to the onboard JTAG controller, DSP host port interface (HPI), and board control/status register.

- **JTAG emulation**: The ‘C6x EVM provides embedded JTAG emulation.
- Programmable logic: the ‘C6x EVM’s CPLD provides the following: the board’s glue-logic and control/status registers.

- Audio Interface: The ‘C6x EVM have a 16-bit audio interface with stereo microphone and line level input and a stereo line level output.

- Power Supplies: The ‘C6x EVM have a voltage regulation to provide 1.8 V or 2.5 V for DSP core; 3.3 V for the DSP I/O, memories, and buffers; and 5 V for audio components. And for all other digital components, the PCI bus or the external connectors are used.

- Voltage Supervision and reset control: The ‘C6x EVM uses a voltage supervisor to control the board voltages and provide a board rest signal.

- User Options: There are 12 onboard DIP switch are provided by the ‘C6x EVM via the PCI bus. Some of these options are boot mode, clock mode, clock select, JTAG select and endian mode.

- LED Indictors: The ‘C6x EVM have three LED indictors—a signal green LED that illuminates whenever 5 V is applied to the board two and red LEDs that can be used for user-defined status.
7.5 TMS320C6x EVM Software Functional Overview

The 'C6x EVM software consists of host and DSP support software. The host support software supplied with the 'C6x EVM board includes the following Win32 host utilities and libraries [12].

- Board configuration utility (evm6xrst.exe): This utility is used to reset and configure the board. [12]
- Code Composer: Code composer is the new software debugger that is used to debug 'C6x software on the board. More information will be provided in the next chapter.
- EVM confidence test utility (evm6xtst.exe): This utility tests the basic operation of the board. It is used to verify the hardware and software installation.
- 'C6x COFF loader utility (evm6xldr.exe): This is used to load and execute 'C6x software on the board.
- EVM Win32 DLL (evm6x.dll): The Win32 host libraries composed of a Windows 95 and a Windows NT version of evm6x.dll that provides user software access for control and communication with the EVM board.
- Example source code: Example code that shows how to use the host Win32 dynamic link library (DLL) functions is provided with the 'C6x EVM.
Figure 7.4 illustrates the block diagram of the EVM host software components and their relationships. The Code Composer is shown at the left side of the figure. To access the JTAG TBC device on the EVM board, the debugger makes direct calls to the low-level (ring 0) drivers. Alternatively, the board configuration utility also makes direct calls to the drivers to access the board option registers on the EVM board.

Figure 7.4 TMS320C6x EVM host support software block diagram [12]

The DSP support software supplied with the EVM board includes the following components:
- McBSP driver (mcbspdrv.c and mcbspdrv.h): This driver is for the multichannels-buffered serial port (McBSP) peripheral.
- Codec library (codec.c and codec.h): This library is a number of routines that configures and controls the operation of the CS4231A audio codec device.
- Board support library (board.c and board.h): This library supplies 'C6x board-specific routines for EVM configuration and control.
- Example source code: code examples are provided to demonstrate the use of the DSP McBSP, codec, and board support functions.

Figure 7.5 illustrates the block diagram of the EVM 'C6x software components and their relationships.

Figure 7.5 TMS320C6x EVM DSP support software block diagram [12]
7.6 Operating Scenarios

The 'C6x EVM can operate in a computer motherboard's full-size PCI slot. Since the 'C6x EVM does not have onboard ROM, the user's applications are loaded by host software using the board's embedded JTAG or DSP HPI interfaces over the PCI bus or by an XDs510 emulator connected to the board 14 pin JTAG header. Table 7.2 summarizes the various ways applications can be loaded into the 'C6x in the PCI operating scenarios. Also, PCI operation Scenario will be explained in the next paragraph.

<table>
<thead>
<tr>
<th>Operating Scenario</th>
<th>JTAG TABS</th>
<th>DSP HPI Via PCI BUS</th>
<th>XDs510 Emulator</th>
<th>ROM Daughterboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCI Slot</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

A Slot host software can control and monitor the 'C6x EVM via memory-mapped registers when the 'C6x EVM is installed in a PCI, and the host software can also access the onboard JTAG test bus controller (TBC) to download code and access the DSP, as well as transfer data to and from the DSP using the 'C6201/6701 HPI. Code Composer--to access the board--uses the JTAG TBC and the 'C6x EVM COFF loader uses the HPI, whereas the XDS510 can be used with the board in a PCI slot but is not
needed because onboard emulation control is provided. If a daughterboard (with nonvolatile memory) is installed, an optional ROM-boot mode can be used in PCI operation [12].

7.7 System Requirements

In this section we will provide the list of the system requirements for operating the 'C6x EVM in a PCI slot configuration [13].

- Hardware checklist:
  - Host Pentium: PC or Pentium-compatible PC
  - Memory: Minimum of 16M bytes (32M bytes recommended)
  - Hard disk space: Minimum of 20M bytes
  - Monitor: Color
  - Slot: One full-length revision 2.1-compliant PCI local bus slot (5 V, 32 bits)
  - Board: TMS320C6x EVM board (included in the kit)
  - Other hardware: CD-ROM drive
  - Cables: 3.5-mm audio cables (optional)

- Software checklist:
  - Operating system: Windows 95 or Windows NT 4.0
  - PC software tools: Microsoft Visual C++ version 4.2 or greater (required for host-side user application development).
Files: required for host-side user application development.

7.8 The Board Confidence Test

The confidence test is testing the major components of the EVM board and its support software. Also it displays board configuration information. The tested EVM hardware components are listed below [12]:

- SBSRAM
- SDRAM
- ’C6x interrupts
- ’C6x timers
- ’C6x DMA
- ’C6x McBSP serial ports
- PCI controller NVRAM (EEPROM)
- Audio codec
- Audio inputs and output
- LEDs
- Endian mode selection

Also the installation of these EVM software components is tested:

- Win32 DLL
- Host low-level board driver
The following section explains how to run the board confidence test utility to verify that the board and software have been installed properly.

### 7.8.1 Verifying the EVM Installation

The following steps are used to run the board confidence test utility to verify the installation of the board and software needed:

- Install the 'C6x EVM board and supplied software.
- Open a command window and change the directory to the conftest subdirectory of the EVM installation directory. This directory is created during the installation process and contains additional files needed for the execution of the confidence test utility.
- Execute the confidence test utility in a command window by entering the following command on the command line: `evm6xtst test.log`. This command starts the confidence test utility for the first EVM board and logs the test results to a file named: `test.log`.

The output from the confidence test should be similar to the sample output provided in [12].

### 7.9 Code Composer Studio

Code Composer Studio (CCS) is a software tool designed to access the TMS320C6701 EVM board. Also CCS is designed to enhance and speed the development process for the board's user, so in this chapter we will have an overview of
the CCS software development process, the components of the CCS, and the files and variables used by the CCS.

7.9.1 Code Composer Studio Development

With a set of debugging and real-time analysis capabilities, CCS extends the generation tools. CCS supports all the following phases listed below [14]:

- Design: conceptual planning.
- Code and build: create project, write source code, and configuration file.
- Debug: syntax checking, probe points, logging, etc.
- Analysis: real-time debugging, statistics, and tracing.

In addition, the CCS includes the following components:

- TMS320C6000 code generation tools.
- Code Composer Studio Integrated Development Environment (IDE).
- DSP/BIOS plug-ins and API.
- RTDX plug-in, host interface, and API.

Figure 7.6 illustrates the CCS's components.
7.9.2 Code Generation Tools

The code generation tools are the bases of the development process provided by the CCS. The software development flow is shown in Figure 7.7, where the most common software development path for C language program is shaded. The remaining portions are peripheral functions that enhance the development process. The generation tools are listed below [14]:

- The C compiler accepts C source code and produces assembly language source code.
• The assembler translates assembly language source files into machine language object files. The machine language is based on common object file format (COFF).

• The assembly optimizer allows you to write linear assembly code without being concerned with the pipeline structure or with assigning registers. It
assigns registers and uses loop optimization to turn linear assembly into highly parallel assembly that takes advantage of software pipelining.

- The linker combines object files into a single executable object module. As it creates the executable module, it performs relocation and resolves external references. The linker accepts relocatable COFF object files and object libraries as input.

- The archiver allows you to collect a group of files into a single archive file called a library. The archiver also allows you to modify a library by deleting, replacing, extracting, or adding members.

- You can use the library-build utility to build your own customized run-time-support library.

- The run-time-support libraries contain ANSI standard run-time-support functions, compiler-utility functions, floating-point arithmetic functions, and I/O functions that are supported by the C compiler.

- The hex conversion utility converts a COFF object file into TI-Tagged, ASCII-hex, Intel, Motorola-S, or Tektronix object format. You can download the converted file to an EPROM programmer.

- The cross-reference lister uses object files to cross-reference symbols, their definitions, and their references in the linked source files.
• The absolute lister accepts linked object files as input and creates .abs files as output. You assemble the .abs files to produce a listing that contains absolute addresses rather than relative addresses. Without the absolute lister, producing such a listing would be tedious and require many manual operations.

7.9.3 CCS Integrated Development Environment (IDE)

As with any other software tools, the programmers need to have some features, such as editing, building, and debugging, so the code composer studio integrated development environment is designed for these features. For example, in the editing features, the CCS allows a programmer to edit both the C and assembly source code. In addition, the programmer has all other feature of editing, such as highlighting of keywords, comments, finding and replace, undo and redo multiple actions, etc. In the application building features, the programmer creates a new project and adds files--C source files, assembly source files, object files, libraries, linker command files, include files--to the project. Also, the CCS can create a full build-rebuild all or an incremental build and can compile individual files beside to the scanning file to build an include file dependency tree for the entire project. Finally, in the application debugging features, the CCS provides support for the following debugging features:

• Setting breakpoints with a number of stepping options.

• Automatically updating windows at breakpoints.
• Watching variables.
• Viewing and editing memory and registers.
• Viewing the call stack.
• Using probe point tools to stream data to and from the target and to gather memory snapshots.
• Graphing signals on the target.
• Profiling execution statistics.
• Viewing disassembled and C instructions executing on target.

7.9.4 DSP/BIOS Plug-ins

DSP/BIOS plug-ins supports real-time analysis in CCS. Also, a configuration file in the CCS can be created to simplify memory mapping. By saving the configuration file, several will be generated that must be linked to the project application. In addition, the configuration file has numerous models that minimize the program size [15], [16].

7.9.5 Hardware Emulation and Real-Time Data Exchange (RTDX)

The DSP board provides an enhanced JTAG link to communicate with on-chip emulation support, which enables CCS to control program execution and monitor real-time program activity. The on-chip emulation hardware has many capabilities--one of these capabilities is RTDX that exposes through the host and DSP APIs allowing for bi-directional real-time communications between the host, and the DSP board [15], [16].
Chapter Eight: The Hardware Implementation and Results

This chapter mainly covers both the software simulation and the hardware implementation and their results. In software, we used the C-Language and the Matlab code, whereas in the hardware, we still use some software to control the DSP target board, such as C-Language, assembly language, linear assembly language, linker command file, and libraries.

8.1 Software Simulation

In the software simulation section, we will study the performance of the IRS algorithm system implemented using C language for both parts--the transmitter and the receiver, whereas the study of the performance of the entire IRS algorithm system implemented using Matlab code was already studied by Dill [2] and was discussed in an earlier chapter. For that study, a Matlab code had been used to represent the IRS algorithm system to study its theoretical performance by using bit error rate ($E_b/N_0$). Figures (4.1-4.2) show the theoretical performance of the IRS algorithm system with different channels.

To deal with the DSP board, we need to have the system represented with C language. So we started with the implementation of the system’s transmitter using C
language as the first step, whereas the channel and the system’s receiver will be later represented using a C-Language. Then we tested the IRS algorithm transmitter code at the Microsoft Visual C++ 6.0 after the run and the debugging process. In addition, we need to make sure about the following after each run of the project:

- Generating a binary random input.
- Generating a base vector (So) randomly.
- Generating a reference vector (Sr) randomly.
- Converting the binary code to decimal value correctly.
- Verifying the FFT function [17] by applying the known input and check the output, then compare it with the Matlab ‘s FFT function.
- Applying the FFT function to the code.
- Processing of the CCSK modulation.
- Applying Frequency Domain Interleaving (FDI) to the transmitter.
- Verifying the transmitted signal.
- Saving the base vector, the reference vector and the transmitted signal at the disk files in the host PC.

In the transmitter code, the output of the transmitter-the transmitted signal- was saved in a disk file on the hard drive of the host PC; then the receiver fetched them to act as its input. In addition to the transmitted output, both the base vector and the reference
vector was saved in disk files and the receiver was fetched them to decode the signal. After the verification of the transmitter, we implemented the IRS basic receiver to decode the transmitted signal. This basic receiver has the CCSK demodulation and Frequency Domain De-Interleaving (FDDI) feature. By comparing the transmitter’s input to the receiver’s output, we made sure that the transmitter was implemented successively and we were ready for the receiver’s C code implementation. Therefore, we implemented the IRS algorithm system receiver by using C language, and by doing so we had the entire system implemented using C code. After we had the whole system implemented in the C code, we tested the system by making a comparison between the input signal of the system’s transmitter with the system’s output signal. We ran the system and did the comparison each time to make sure that the system was working perfectly. Again the things that we checked for the entire system are the following:

- CCSK demodulation.
- FDDI processing.
- Implementation of the IFFT.
- Using the base vector and the reference vector to decode the signal.
- Using a maximum function to determine the highest sample at the correlation process.
• Running the project for different values of the supersymbol length (M) starting with low value (4 samples) and ending with the highest value (16K samples).

• Adding noise function to encrypt the transmitted signal and repeating all the previous steps.

After we were done from the implementation of the entire system by using the C-Language, we were ready to start our journey of hardware implementation.

8.2 The Hardware Implementation of The IRS Algorithm System

In this project, we are planning to implement the IRS algorithm system in the hardware by using the TMS320C6701 EVM bundled board, which is one of the Texas Instrument Company’s products. After we got the DSP board in our communication lab, we installed it in the one of the lab PCs that has Windows 98 as its operating system. Then we installed its software development tools with the following folders besides to the folder that created in the windows directory [14]:

• Bin: Various utility programs

• C6000\bios: Files used when building programs that use the DSP/BIOS API

• C6000\cgtools: The Texas Instruments code generation tools

• C6000\examples: Code examples

• C6000\rtdx: Files for use with RTDX
• C6000\tutorial: The examples you use in this manual

• Cc\bin: Program files for the Code Composer Studio environment

• Cc\gel: GEL files for use with Code Composer Studio

• Docs. Documentation and manuals in PDF format. (If you did not choose to install the complete documentation, see the CD-ROM for manuals in PDF format.)

• Myprojects: Location provided for your copies of the tutorial examples and your project files.

The following directory structure is added to the Windows directory:

• Ti\drivers: Files for various DSP board drivers.

• Ti\plugins: Plug-ins for use with Code Composer Studio.

• Ti\uninstall: Files supporting Code Composer Studio software removal.

While using Code Composer Studio-the board interface-, you work with files that have the following extension:

• Project.mak. Project file used by Code Composer Studio to define a project and build a program

• Program.c. C program source file(s)

• Program.asm. Assembly program source file(s)
• Filename.h. Header files for C programs, including header files for DSP/BIOS API modules
• Filename.lib. Library files
• Project.cmd. Linker command files
• Program.obj. Object files compiled or assembled from your source files
• Program.out. An executable program for the target (fully compiled, assembled, and linked). You can load and run this program with Code Composer Studio.
• Project.wks. Workspace file used by Code Composer Studio to store information about your environment settings
• Program.cdb. Configuration database file created within Code Composer Studio. This file is required for applications that use the DSP/BIOS API and is optional for other applications. The following files are also generated when you save a configuration file:
  • Programcfg.cmd. Linker command file
  • Programcfg.h62. Header file
  • Programcfg.s62. Assembly source file

Of course, after we installed the DSP board, the first thing that we did was run the confidante test and verify its output. Then we went over the provided examples given
in the tutorial section [14]. The first example was the hello project, with a simple project to help us develop a simple program. So we learn from the first project the following: creating a new project, adding files to the project, reviewing the code, building and running program, changing program options, and fixing syntax error, using some of CCS features such as breakpoints, watch window, and making a profile code execution time. After we finished from this example, we went over the rest of the projects given in [14], which really gave us full confidence to deal with the board. Also, we went over some more advanced examples given in [9], [10], and [11], such as audio signal sampling, integer arithmetic, adaptive filtering, real time filtering, and FFT. In general, we obtained also information about the board and the way we should deal with it, so we are ready to having our first example implemented in the DSP board.

8.3 A Hardware Pulse Project

In this simple hardware project, we were aiming to deal with the board by having our own example and applying some other features in addition to test both the board and our ability with the hardware. So we chose the project that generated a pulse randomly as its input and then inverted that input to have the project’s output. For example, if we had the input as [0 0 0 1 0 0 0 0], the output would be as follows [1 1 1 0 1 1 1 1]. In this example, we first generated the input pulse signal randomly and then sent the signal to the hard disk file to be saved there. After that, we fetched the signal to apply the inverter and get the output, which is the input’s inverted signal. The pulse project view is
illustrated in Figure 8.1, and as it clears form the figure, the project has the followings folders:

- Include: this folder has all the files that main source need them.

- Library: this folder has the processor library rts6701.lib, which provides runtime support for the target DSP.

- Source: this folder has the main source program in C language and assembly file (*.sam), which contains assembly instructions needed to set RESET interrupt service fetch packets to branch to the program’s C entry point.

Figure 8.1 Pulse project view
- Linker command file: pulse.cmd, which maps sections to memory.
- DSP/BIOS Config: dsp configuration file—not used in this example.
- GEL Files: this folder is for the general execution language function.

To start the hardware debugging process, we built the project and then got zero errors and zero warnings (good!). Then we loaded the data or the pulse common object file format (COFF) into the DSP target board and got the Dis-Assembly window, which has the assembly code of the instructions and Program Counter (PC). Figure 8.2 shows the Dis-Assembly window in the pulse project and the build view of the same project. The Dis-Assembly window opened by code composer studio because we loaded the pulse program onto the DSP target board.

![Figure 8.2 The dis-assembly window in the pulse project](image-url)
It displays the disassembled instructions and symbolic information needed for the debugging. After we loaded the COFF successively, we run the program to have the output signal and got the results. Figure 8.3 shows the output view of the pulse project for several values of input and output signals.

Now it is time to play with the code composer studio features to plot the input and output signals and show the values of the vectors by viewing their memory addresses or by utilizing the watching window. Figure 8.4 depicts some of the features of code composer studio.

Figure 8.3 The output view of the pulse project

In summary, we generated a random pulse, and sent it to the host PC, and then fetched it back to the main source and inverted it to get the project’s output. Moreover,
we got familiar with the DSP target board and its software development tools, CCS, and learned from the hardware pulse project the following:

Figure 8.4 some of the code composer studio features

- How to generate random vectors.
- How to use the following CCS' features:
  - Creating the new project.
  - Leaning how to write the linker command file.
  - Adding new files to the projects.
  - Learning how to build and run the project and how to load a common object file format onto the DSP target board.
  - Getting familiar with the debugging features such as:
    - Watch window view.
• Memory address view.
• Graph view for any vector.
• Setting breakpoints, probe points, etc.

- When we sent the pulse signal to the host PC, we use the following:
  - A C-Language command (printf).
  - A hardware feature DO I/O.

8.4 The IRS Algorithm System Hardware Project

Now we are completely ready to start our IRS hardware project, so first, we plan to implement the transmitter of the IRS algorithm and test it for several values of the supersymbol’s length. After that, we aim to implement the basic receiver to the IRS algorithm system to study the system’s performance. Finally, we hope to add more features of the IRS algorithm system and upgrade the system gradually, in addition to adding the noise to the transmitted signal.

8.4.1 The IRS Algorithm System’s Transmitter

In the beginning, we prepared with representing the algorithm’s code in C-Language, since we had it in a Matlab code when we studied its theatrical performance, to deal with the DSP target board properly. In addition, we wrote the linker command file to map the memory to the sections. Therefore, it was the time for creating a new project which we called the IRS project, representing the transmitter of the IRS Algorithm. Then we added all the files needed by the software and the hardware, such as
main source, linker command file, library, and all include files by choosing scan for all independencies options. As mentioned in Chapter Seven, the compiler, the assembler, and the linker have many options, so we chose the proper option for each of them by choosing the project \rightarrow options at the toolbar's buttons. Also, we needed to check the destination of the output file where the output file would be saved in addition to selecting its proper name for the output file (*.out). Then it was time to building the project by using the build button and of course we were expecting (0 Errors) as results of the build's output (as seen in Figure 8.5). Finally, we loaded the COFF to the DSP target board -by choosing FILE \rightarrow Load program \rightarrow select the object file- to prepare to run the hardware. By loading the COFF onto the DSP target successively, the CCS opened the Dis-Assembly window to display the disassembled instructions and symbolic information needed for the debugging. To run the project at the hardware, we selected Debug\rightarrow RUN; the DSP target was running, as indicated by the symbol-DSP RUNNING- that appeared at the left lower corner of the CCS window view. Then the DSP target halted and it was clear by substituting-DSP RUNNING- with the following symbol -DSP HALTED-. At this point, we being able to verify the results and view them using CCS's features, such as watch window view, memory address view, graph view, etc.
From this part of the project, the main goal was to make sure that the transmitter of the IRS algorithm was working successively experimentally since we started with very low length of the supersymbol. Then, we gradually upgraded the system and tested it each time we did upgrading. Notice that all the results and the discussion of this hardware project are covered in the “Results” section later. Since we make sure about the system’s transmitter, we are going to implement the basic receiver of the IRS algorithm system in the hardware.

8.4.2 The IRS Algorithm System’s Hardware

In fact, we first began with upgrading the IRS algorithm system’s code represented in C-Language to have both the transmitter and the basic receiver. Then we
upgraded the linker command file to be able to map the memory to the sections properly. Then, we added the new C-Language file to the project and the linker command file, as well. Moreover, we added all the include file to the project to make sure that the IRS algorithm systems project were ready to be implemented with the hardware. After that, we went over the same process -building, loading the COFF, and running- to have the output. With all these processes, we have verified that the input signal and the output signal was the same for several values of the supersymbol’s length. So far, the system transmitter and basic receiver are implemented in the hardware using the DSP board, and based on the input and the output of the system, the IRS algorithm is working perfectly at the hardware. Also, all the results and the dissections with the graphs will be covered in the next section.

8.5 Results

We gave the results and the discussion of the simple pulse hardware project and its main goals in an earlier section of this chapter. However, in this section we will go first over the transmitter of the IRS algorithm hardware project’s results and its importance and we will discuss these results and show some graphs. After that, we will go over the IRS algorithm hardware project’s results and will discuss these results and show some graphs, as well. In the beginning, let us define some of the important symbols or definitions that control the system:
• nn is the length of the data that was chosen by the CCSK Modulation (how many bits that the CCSK can take at the time).

• mm \((mm=2^{nn})\) is the length of each symbol (how many samples at each symbol).

• kk is Number of symbols (how many time we need to take nn bits and generate a symbol with length of mm).

• rr \((rr=nn\times kk)\) is the length of the source (this is the main information that needs to be transmitted, so we will have nn bits kk times).

• M \((M=kk\times mm)\) is the total length of the signal or the supersymbol.

So, each time we need to change the length of the supersymbol, we change the symbols nn and kk and all others will be changed accordantly. Also, we need to clarify some terminologies that we will use in this section:

• Run process consists of the followings three parts:
  
  ▪ Building’s part consists of a compiler, an assembler, and a linker and if we had zero Errors as result of the building’s output, then we would go to the load process.
  
  ▪ Load’s part is a loading the common object file format (COFF) and if we did it properly, we would go to the run process.
  
  ▪ Run’s part is a debugging process.

• Result means have an input signal, output signal, and all dependencies.
• Position means the decimal value of the nn binary bits that had been chosen by the CCSK modulation; for example, if we had the nn=[1 0 1], the decimal value would be 5.

• Input means a vector that has the positions of all the symbols that had been sent and so we should have a kk position at the input vector for each run.

• Output means a vector has the positions of all the symbols that had been received and so we should have a kk positions at the output vector for each run.

• Verifying the result means checking the values of any vector in the project, and this can be by any of the following ways:
  - Watch window view.
  - Memory addresses view.
  - Graph view which the following options:
    - Time/Frequency graph view.
    - Constellation graph view.
    - Eye diagram graph view.
    - Image graph view.

8.5.1 Theoretical Results

In the beginning, we are going to explain how the ISR algorithm system works. First, the CCSK modulation takes nn bits to figures out its decimal value; then it
generates a zeros sequence with length of $mm=2^n$ and puts one at the position that represents the decimal value. Meanwhile it converts the sequence to its frequency domain vector by using the FFT function and saves the output of the FFT function at a new vector that has a supersymbol length of $M (M=kk*mm$, where $kk$ is the number of symbols). This vector will be saved at the first $mm$ position at the supersymbol vector. A new vector called B_2_D will be generated to save the decimal values or the position of the input data. Then we repeat these steps $kk$ times to fill the supersymbol vector. Meanwhile, the IRS system generates both base and reference vectors, respectively, with the same length of the supersymbol and then takes their FFT values. Moreover the system multiplies the information sequence bit-by-bit with the base vector and then adds them to the reference vector bit-by-bit to give the transmitted signal. Nevertheless at the receiver, the system decodes the data and the reference signal by multiplying again bit-by-bit the received signal with the base vector and with the reference vector. The output of this data decoding will be multiplied with the output of the reference decoding to prepare the signal to the symbol decision. Finally, it takes $mm$ samples of that the output of the last step and then takes its IFFT to return it back to its time domain to verify its maximum value’s position to save it at a vector called output position. To verify the system performance, we compare the two B_2_D vector with output position vector value-by-value. Therefore in our research, we generate the C language code for the entire IRS algorithm system and then we test the IRS algorithm transmitter code at the
Microsoft Visual C++ 6.0. At the simulation, we run the system and check the results’ vector several times and for several values of M. Finally, we added the noise to the transmitted signal and repeat the same process to verify the system results before we implement it at the hardware.

8.5.2 The IRS Transmitter Hardware’s Project

As we described earlier, the main goal of the transmitter IRS algorithm hardware project is to implement the system step-by-step to verify that the system works correctly with the hardware—for example, to implement the system using the FFT function and at the next step, which is the receiver, we will implement the system using the IFFT function etc. First, we chose the following values for the project:

- \( nn = 2 \) bits.
- \( kk = 1 \) symbol.
- \( M = 4 \) samples.

Then we made the run process to get the result. Based on the code that has been written in C, the results should be satisfied as long as the length of the supersymbol is quite high enough. This is because the base vector and the reference vector conditions must be met. For a small number of samples, the cross correlation between the base vector and the reference vector will be very high, which causes interference among the symbols. Therefore, the reason of choosing this low number of samples is to check our ability to control the DSP target board, and test the system’s ability, such as dealing with
DSP processor, memories etc. In fact, the purpose of this low number of samples was to clarify that what we obtained from the first case was fully satisfied because it met the following criteria:

- Random function generates random values for several vectors and each vector’s content differs from the others.
- FFT function works properly and gives the expected results when it compared to the FFT Matlab function’s output if we apply the same input.
- CCSK modulation works successively.
- Memory mapping is doing its job perfectly.
- CCS’ features are available, applicable, and doable.
- Most important, the DSP target is working successively.

In general, the transmitter of the IRS hardware’s project is just for checking the system’s memory or how many samples the system can reach without any memory problems.

Since the transmitted signal is still in process (or we don’t implement the receiver yet), we could not verify the results. Figure 8.6 illustrates the IRS project view, building’s output, and assembly window. Since we were satisfied with the last iteration four-samples case, we will upgrade the system to testify the ability of the system in terms of memory with a larger supersymbol’s length. So we chose the following values for upgrading the project:
- nn = 2 bits
- kk = 2 symbols
- M = 8 samples

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>00009114</td>
<td>STW.D1 A4.==A0[0x0]</td>
</tr>
<tr>
<td>00009118</td>
<td>SUB.L1 A0.A0.A0</td>
</tr>
<tr>
<td>00009120</td>
<td>STW.D1 A3.==A0[0x1]</td>
</tr>
<tr>
<td>00009124</td>
<td>07BC5CFA</td>
</tr>
<tr>
<td>00009128</td>
<td>001FEE32</td>
</tr>
<tr>
<td>00009132</td>
<td>0005006A</td>
</tr>
<tr>
<td>00009136</td>
<td>07BC1E22</td>
</tr>
<tr>
<td>00009140</td>
<td>078F7A23</td>
</tr>
<tr>
<td>00009144</td>
<td>078542A</td>
</tr>
<tr>
<td>00009148</td>
<td>07400658</td>
</tr>
<tr>
<td>00009152</td>
<td>01000023</td>
</tr>
<tr>
<td>00009156</td>
<td>090C03A2</td>
</tr>
<tr>
<td>0000915E</td>
<td>0A0C03A2</td>
</tr>
<tr>
<td>00009168</td>
<td>01EC50A4</td>
</tr>
<tr>
<td>00009170</td>
<td>02400666</td>
</tr>
<tr>
<td>00009174</td>
<td>029012AC</td>
</tr>
<tr>
<td>00009178</td>
<td>01C8B422</td>
</tr>
<tr>
<td>00009180</td>
<td>0180006A</td>
</tr>
</tbody>
</table>

Figure 8.6 The IRS project's view, building's output, and disassembly window

Since we were satisfied with the last iteration four-samples case, we will upgrade the system to testify the ability of the system in terms of memory with a larger supersymbol's length. So we chose the following values for upgrading the project:

- nn = 2 bits
- kk = 2 symbols
- M = 8 samples

We upgraded the system length up to 8 samples (four samples was the last length of the system) and the hardware DSP was still working successively and giving zero
errors as a building’s output, which means the transmitter of the IRS system was applicable at hardware with eight samples as supersymbol’s length. Also, we did not have any problem with the memory when we upgraded the length of supersymbol of the transmitter of the IRS algorithm system’s project to sixteen samples. Moreover, this result gives us some confidence to upgrade the system one step further by choosing the supersymbol’s length of 32 samples instead of 16 samples. However, the building’s output had a memory error at this time, and it was not a good result or at least was not as we expected. So we went back to the memory and tried to organize the system to solve the problem. For example, we went over the vectors and their memory space needs and we changed some of unused but reserved memory. These changes were enough to let the transmitter of the IRS system work perfectly; however, that problem was a good indication that we will have more memory problems as we upgrade the system. So we upgraded the transmitter of the IRS system’s length to be 64 samples and we got zero errors at the building’s output. However, this was not as expected because we had a memory problem in the previous iteration which had a lower number of the transmitter of the IRS system’s length compare to this iteration. We did not have problems because when we fixed the problem of the previous case (M=32 sample) we reduced the amount of memory that the system needs, so it, in fact, expanded the memory [18]. Also, we might not have any problem if we upgrade the system one step further, which is what really happened when we upgraded the system’s length to be 128 samples. However, this
was not the case when we upgraded the system’s length up to 256 samples. To solve this problem, we made some changes in the linker command file to make sure that all the memory is used besides checking the system again in terms of memory space. Finally, we stopped at this length because of time limits. We are ready to have the basic receiver implemented in the hardware to test the entire system by doing a comparison between the input signal and the output signal to study the system performance experimentally.

To summarize what we had done so far, we started the hardware implementation of the transmitter of the IRS algorithm system with a very low value of $M=4$ samples supersymbol’s length to make sure that the system was working correctly; then we gradually upgraded the system’s length until we reached the value of $M$ to be 256 samples via these sequence values of $M$ (4,8,16,32,64, and 128 samples). Even though the transmitter was working experimentally for $M=256$ samples, we did not expect the entire IRS system to reach such a number because we should use more variables and functions that need more space-extra memory unless we make some changes to the system or to the project.

8.5.3 The IRS Hardware Project’s Results and Discussions

As it described in Section 8.4.2, the IRS main source’s code was modified to have both the transmitter and the basic receiver of the IRS system and the linker command file was modified. So we have the project ready for the run process with first a very low value of supersymbol’s length, such a four samples, to verify the system’s
receiver at the hardware. After the run, we made sure the DSP worked perfectly and gave results. However, these results were not as needed for the reason discussed earlier in this chapter. So in this iteration, all that we needed was to verify the DSP board working properly, although there were errors in the results. Furthermore, we upgraded the system length to have eight samples, and the results we obtained were not far from the last case, as seen in Figure 8.7. Also, the case of the sixteen samples was almost the same as previous cases. Figure 8.8 illustrates the IRS project for \( M=16 \) samples.

However, for the case of \((32)\) thirty-two samples, the system started to give some correct results since the length of the system started to get bigger and bigger. Figure 8.9 illustrates the IRS project for \( M=32 \) samples.

![Figure 8.7 The IRS system’s output for M=8 samples](image-url)
Figure 8.8 The IRS system's output for M=16 samples

Figure 8.9 The IRS system's output for M=32 samples
Therefore, by upgrading the system’s length to (64) sixty-four samples, the DSP was still working and the errors in the system’s results were being smaller—sometimes zero errors. Figure 8.10 illustrates the IRS project for M=64 samples. However, when we upgraded the system’s length to 128 samples, the building’s output had an error because of the memory. So we went back to the C-code and tried to optimize it again by working out with the size and the amount of memory space reserved for each variable to minimize or reduce the memory, but this was not enough to solve the problem. We needed to have another try, such as checking the linker command file or the CCS’ features to extend the memory. Finally, these iterations were enough to solve the problem and let the DSP board work successively and give good results. Figure 8.11 illustrates the IRS project for M=128 samples.

![Figure 8.10 The IRS system’s output for M=64 samples](image)
Therefore we needed to get some information about the IRS system, such as the transmitted signal, the base vector, the reference vector, input signal, and output signal by using some of the features of the CCS:

- We generate both the base vector and the reference vector randomly with values (1, -1), and then we applied them to the FFT function to have them in frequency domain, as is clear in Figures 8.12 and 8.13, which have the real part of the base vector (upper) and the imaginary part of the base vector (lower) for the IRS's with M=128 and the same for the reference vector.
Figure 8.12 The real part of the base vector (upper) and the imaginary part of the base vector (lower) for the IRS's with $M=128$.

Figure 8.13 The real part of the reference vector (upper) and the imaginary part of the reference vector (lower) for the IRS's with $M=128$. 
• Figure 8.14 illustrates the real part of the transmitted signal and the imaginary part (lower) of the transmitted signal for the IRS's with M=128 samples.

• Figure 8.15 illustrates the symbols positions at the input signal and the symbols’ positions at the output signal of the IRS’s with M=128 samples.
  ▪ It is clear from Figure 8.15 that the input vector has two values—the first one at position (0) was 41 and the second at position (1) was 48, which means we had two symbols (kk=2) each with length of 64 (mm =64 samples)
    ○ The first symbol was all zeros but at position number 41, it had one.
    ○ The second symbol was all zeros but at the position 48, it had one.

• Also, we note that if we had M=128 samples and as we know M=mm*kk and mm=2^nn and we had nn=3 bits, KK=16 symbols. In the case, we had a lot of errors—why?
  ▪ We had eight samples and these samples were not long enough to combat the interferences and the noise at the process of both data decoding and reference decoding, respectively.
  ▪ In this situation, the correlation happened with the signal that had eight samples and with eight samples of the base vectors or the reference vector. So we considered the system with the length of eight samples, which was not string enough to combat the interference.
Figure 8.14 The real part of the transmitted signal (upper) and the imaginary part of the transmitted signal (lower) for the IRS's with $M=128$

Figure 8.15 The input signal (top) and the output signal (lower) of the IRS's with $M=128$ and $kk=2$ symbols
8.5.4 The IRS Hardware Project’s with Noise Results and Discussions

First, we established a noise function to generate both the real and the imaginary noise vector, which represents an AWGN. Then we added the noise to the transmitted signal for both parts—real and imaginary—to test the IRS systems’ ability against the noise. We had the same criteria that we used in the previous section to study the performance experimentally, and based on that criteria let list what we had done so far:

- We applied the noise to the system with different values of M; however,
  - From M=4 samples until M=32 samples there was not enough length to combat both of the interference and the noise.
  - Whereas in the case of M=64 samples, there were two cases:
    - If 2<nn<5, the error was already there without the noise, as discussed in the previous section.
    - If nn=6 bits, sometimes the results existed without any error; however, most of the time it was with errors because of the length of the base vector and the length of the reference vector.
- And for the case of M=128 samples, we had building’s error because of the memory, since we had a new function and variables. However, by optimizing the code the problem, was solved.
Figure 8.16 depicts the IRS System’s output for M=128 samples with the noise added to the system’s transmitted signal, and Figure 8.17 shows the noise signal for both parts: real (top) and imaginary (lower) for M=128 samples.

Figure 8.16 The IRS system’s output for M=128 samples with the noise

Figure 8.17 The noise signal for both parts: real (upper) and imaginary (lower) for M=128 samples
Finally, Figure 8.18 illustrates the real part of the transmitted signal (upper) and the imaginary part of the transmitted signal (lower) plus the noise for the IRS's with $M=128$ at the frequency domain.

Figure 8.18 The real part of the transmitted signal (upper) and the imaginary part of the transmitted signal (lower) plus the noise for the IRS's with $M=128$

### 8.6 Statistical Study

After we have done from the project experiments, we made some statistical studies. These studies are mainly cove the followings:

- Central Processing Unit (CPU): it is the percentage of the instruction cycles that the CPU spends doing application work. Or it is the percentage of the total time that the CPU is:
• Running interrupt service routines, software interrupt, or periodic functions.
• Performing input/output with the host.
• Running any user routine.
• Memory: it is the used memory for each supersymbol value and it has the following parts:
  ▪ Internal Program Random Access Memory (IPRAM).
  ▪ Internal Data Random Access Memory (IDRAM).
• Time: it is the execution time in milliseconds for each supersymbol value.

We rebuilt the experiment for several values of supersymbol length (4-128 samples), and we applied a loop with value of 1000 for the entire system. Then, we used the feature of the CCS—DSP/Bios—to measure the CPU load and illustrated it percentage. For the memory used, we generated the map file to describe the memory used for each section. And for the time execution, we used the C-command to get the execution time in milliseconds. Table 8.1 shows the summary of this statistical study for these parameters.
Table 8.1 The statistical study summary for different parameters

<table>
<thead>
<tr>
<th>Supersymbol Length (Samples)</th>
<th>CPU (%)</th>
<th>Memory</th>
<th>Execution Time (Milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>IPRAM (Bytes)</td>
<td>IDRAM (Bytes)</td>
</tr>
<tr>
<td>4</td>
<td>23.73</td>
<td>31360</td>
<td>38378</td>
</tr>
<tr>
<td>8</td>
<td>25.41</td>
<td>31424</td>
<td>40418</td>
</tr>
<tr>
<td>16</td>
<td>26.60</td>
<td>31456</td>
<td>44498</td>
</tr>
<tr>
<td>32</td>
<td>27.09</td>
<td>31752</td>
<td>46418</td>
</tr>
<tr>
<td>64</td>
<td>27.70</td>
<td>32488</td>
<td>50274</td>
</tr>
<tr>
<td>128</td>
<td>27.83</td>
<td>34240</td>
<td>57954</td>
</tr>
</tbody>
</table>
Chapter Nine: Conclusion

9.1 Summary of Results

In this dissertation, we have proposed a hardware implementation of the IRS algorithm system using a DSP board, TMS320C6701 EVM bundled, produced by Texas Instrument. We have reviewed the IRS algorithm system and theoretically studied its performance carefully and then we looked for a suitable board in which to implement the system. Based on the criteria provided in Chapter Five, we chose TMS320C6701 EVM bundled board that had been designed at TI. After we received it at the communication lab, we went over a lot of examples that were provided by the company to get familiar with the board and its development software tool, CCS. Also we established a simple example--a pulse project--to understand the board carefully. Then to start the hardware project, we prepared to write the code in C language to deal properly with the board and then implemented the system in the board. However, we started our hardware project with the implementation of the transmitter of the IRS system with a low number of the supersymbol’s length to verify the system in the hardware. By increasing the length of the supersymbol gradually, we achieved 256 samples as supersymbol’s length for the hardware implementation of the transmitter only. This part of the implementation—the transmitter hardware part—gave us enough motivation to start the implementation of the entire IRS system, so we have implemented the IRS system at the hardware with
different values of the supersymbol's length until we reached the value of 128 samples. Then we added noise to the system to check its robustness with the same length of 128 samples. This hardware implementation of the IRS algorithm system has the following features of the system: CCSK modulation, frequency interleaving, the addition of the Additive, White Guassian Noise (AWGN), CCSK demodulation, and frequency de-Interleaving.

9.2 Future Research

In this section, we are going to state some ideas for future work related to our research:

- Upgrading the system length: The system supersymbol’s length is upgradeable as discussed earlier in this research and some good points to start with are:
  - Optimizing the code in term of memory or space reserved for each vector used in the code.
  - Minimizing the function used.
  - Using the off chip external memory that provided by the board.
- Adding more features of the IRS system: the IRS Algorithm system has a lot of features, noise reduction, coherent cancellation, Narrowband Excision (NBE), etc, as is clear in Chapter Three, and since we used a few of these features, it is good to add more of them gradually.
• Optimization for a Real-Time Execution Speed: Here it is good to run the IRS algorithm experimentally after all the developments needed have been done to the TI's board. For example, write and develop the software in the proper language for the IRS algorithm model for the ideal channel, and then study the experimental performance of the IRS algorithm for the ideal channel.

• System Performance Evaluation on a Set on Non-Ideal Channel: After the previous steps have been successively accomplished, it is good to have an evaluation and study the performance of the model for a set of non-ideal channels. Since the IRS algorithm was designed for a HF channel, it doesn’t hurt to test this modulated scheme on a variety of a non-ideal channels, such as an indoor, VHF, urban multipath channel (UMC), etc.

• Using the DSP/BIOS plug-in features to support real-time analysis in CCS.

• Implement the IRS using another board, such as [5], which has a Matlab interface or using the math work products [19] that gives us the access into the board and develop its hardware by using Matlab.
References


[19] Math Works,

This dissertation proposes a hardware implementation of the IRS algorithm system by using the TI’s DSP board called ”TMS320C6701 bundled EVM”. This research consists of three parts: the review and the study of the IRS algorithm system, the explanation of the hardware and criteria of chosen a suitable board, and the hardware implementation of the IRS algorithm system. This hardware implementation project is divided into two sections in terms of the implementation: the transmitter and the entire system, transmitter, channel, and receiver. So, we started with the system’s transmitter implementation and then we implemented the system entirely at the hardware.

At the transmitter implementation, we started writing the code in C language and then writing the assembly language for the linker command file, and we began the implementation with a very low value of the supersymbol’s length and upgraded the system length gradually. Of course, these results led us to implement the receiver of the IRS system. So after doing many of modifications to the project, we implemented the