DESIGN AND IMPLEMENTATION OF A SAMPLING SWEPT TIME DELAY SHORT PULSE (SSTDSP) WIRELESS CHANNEL SOUNDER FOR LMDS

by

Christian James Rieser

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirement for the degree of

Master of Science
in
Electrical Engineering

APPROVED:

_____	_____
Dr. Dennis Sweeney, Co-Chair	Dr. Jeff H. Reed, Co-Chair
_____	_____
Dr. Charles Bostian	Dr. Tim Pratt

July 17, 2001
Blacksburg, VA

Keywords: Broadband, Wireless, Channel Sounder, SSTDSP, LMDS, Direct Digital Synthesis, Ultra Wide Band, Global Positioning System, Digital Signal Processing

DESIGN AND IMPLEMENTATION OF A SAMPLING SWEPT TIME DELAY SHORT PULSE (SSTDSP) WIRELESS CHANNEL SOUNDER FOR LMDS

Christian James Rieser

(ABSTRACT)

This thesis describes the theoretical development, design, and implementation of a novel measurement system, called a Sampling Swept Time Delay Short Pulse (SSTDSP) wireless channel sounder, capable of real time in field performance characterization of high speed fixed wireless links. The SSTDSP sounder has been designed to provide vital performance metrics for fixed point high data rate applications in the 28 GHz LMDS band at a fraction of the cost and complexity of existing wideband channel sounders.

The SSTDSP sounder monitors the behavior of the LMDS channel by sampling the impulse response of the channel in real time. This digitized impulse response is used to assemble a power delay profile and render real-time channel performance metrics such as the mean excess delay, RMS delay spread, maximum excess delay for a given multipath threshold, and coherence bandwidth. The SSTDSP sounder is capable of recording these metrics through three modes of operation - continuous channel monitoring, single instant channel snapshot, or data logging. Swept time delay time dilation processing is combined with precise sample and hold gating to reduce the analog to digital converter sampling rate required to digitize the nanosecond short pulses from 2 Gsps to 1 Msps, while retaining the required effective Nyquist sampling rate of 2 Gsps. This dramatically reduces the memory, digital signal processing, and data logging storage requirements as well as the overall cost of the sounder system.

The thesis presents the theory behind channel sounding and discusses whether there is a "bounce path" available to LMDS. Several existing channel sounding methods are compared for this application. A number of specific design and performance criteria from

each of these methods are synthesized to produce the Sampling Swept Time Delay Short Pulse Sounder architecture. The design and implementation process used to realize the SSTDSP sounder is presented, including a system overview, module details, and algorithm development details. A calibration and measurement test procedure is outlined and system verification results are presented.

Current work in progress on the test platform and future improvements to the modular system are outlined, as well as conclusions and future implications of the system.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Content and Organization of Thesis

This thesis describes the design and implementation of a novel measurement system, called a Sampling Swept Time Delay Short Pulse (SSTDSP) wireless channel sounder, capable of characterizing high speed fixed wireless links in the field and in real time. The SSTDSP sounder has been designed to provide vital performance metrics for fixed point high data rate applications in the 28 GHz LMDS band, at a fraction of the cost and complexity of existing wideband wireless channel sounders.

The theory that provides the foundation for this innovative system design is presented along with initial results. Chapter 1 gives the motivation for the SSTDSP design by providing a specific problem definition. Chapter 2 discusses wireless signal propagation at LMDS frequencies and the theory behind channel sounding and compares a number of existing sounding methods. Selected design and performance criteria from each of these methods are synthesized to produce the Sampling Swept Time Delay Short Pulse Sounder. Chapter 3 introduces the Sampling Swept Time Delay Short Pulse system and describes the design process and implementation of the SSTDSP system. Chapter 4 outlines a test procedure, methods for calibration and verification, and current work in progress on the test platform as well as future improvements to the modular system. Chapter 5 outlines future implications of the system and concludes the thesis. Appendix A details the SSTDSP sounder module interfaces, operation, and key parameters. Appendix B indicates the specific target programming language used for each SSTDSP sounder algorithm and provides step by step SSTDSP sounder algorithm processing flow

details. Appendix C provides photos of the SSTDSP sounder implementation and source code for all algorithms developed in this thesis. Appendix D provides details of the test procedure for taking SSTDSP sounder measurements.

1.2 Contributions

This thesis is concerned with the design and implementation of a wireless channel sounding system that provides high performance at a fraction of the cost, size, and complexity of existing channel sounding methods. As such, this thesis makes the following contributions:

- Presentation of a new channel sounding method introduced by Sweeney et.al that reduces the cost, size, and data processing and archival requirements for wideband channel measurement, while retaining resolution and accuracy. This method combines short pulse radio frequency (RF) transmission with sampling swept time delay sampling by a digital signal processor (DSP) to allow channel characterization in real time in the field.
- Design of a modular system architecture that realizes this new sounding method. The interaction and tradeoffs made between RF and DSP techniques are explored.
- Development of the hardware and software modules that implement this new channel sounder architecture in a cost and size efficient way. Digital hardware and algorithms were implemented by the author, while RF components were either bought off the shelf or implemented by Dr. Dennis Sweeney.

1.3 Motivation

In recent years, the demand for high speed data communications has increased dramatically. The growth of the internet has required an increase in the communications infrastructure, capable of providing a vast array of services to a variety of locations. Most recently, demand for carrier class data pipes to remote locations has prompted significant investment in broadband wireless services and equipment. To jumpstart high speed

wireless technology development, the Federal Communications Commission (FCC) has allocated a number of frequency bands specifically for multimedia communications, including voice, video, and data. The Local Multipoint Distribution Services (LMDS) band at 28 GHz has been designated to provide these bandwidth intensive services.

Virginia Tech recognized the research and commercial potential for the immense wireless spectrum provided to LMDS band licensees and invested substantial resources into acquiring Block A licenses in a number of geographical locations near Virginia Tech. These licenses allow Virginia Tech to design, test, and operate experimental and production fixed point wireless equipment in the frequency range of 27.5 GHz to 28.35 GHz. The university is required to deploy LMDS technology in order to retain their license of that spectrum. To effectively deploy systems for this new frequency band, extensive knowledge of the wireless channel is beneficial. This thesis focuses on the implementation of a novel digital measurement system, called the Sampling Swept Time Delay Short Pulse (SSTDSP) Wireless Channel Sounder, capable of characterizing wireless links in the field that operate over the entire 850 MHz Block A LMDS band. This information can then be used in conjunction with Geographic Information Systems (GIS) to permit optimal rapid deployment of LMDS network nodes.

1.4 Problem Definition

The performance of any wireless system is inherently limited by the usable channel bandwidth. Information about the behavior of the communications channel allows the engineer to determine the channel capacity and overcome a number of design challenges [1]. Channel measurement equipment can be used to monitor and analyze the behavior of the wireless channel, rendering the usable bandwidth and information about the propagation characteristics of the channel being studied.

Most state of the art wireless channel measurement tools are designed for relatively narrow channels compared to the 850 MHz LMDS Block A spectrum band, often channel

bandwidths on the order of less than 100 MHz. These existing measurement systems can therefore leverage traditional real-time sampling and digital processing techniques, which in turn reduce the overall cost and operation of these narrowband channel sounders. However, to directly sample the impulse response of the entire LMDS Block A spectrum band, by Nyquist's theory an analog to digital (A/D) converter running at close to 2 Gsps is required [2]. As of this writing, the cost of such a system is prohibitive, especially since a processor and storage space capable of handling a 2Gsps throughput becomes a necessity. This thesis focuses on leveraging alternative processing methods to reduce the complexity and cost of implementing such a wideband digital measurement system.

The SSTDSP sounder transmits an impulse like signal, or Ultra Wide Band (UWB) pulse shape, over the LMDS channel and uses the SSTDSP method to economically and efficiently digitize the received channel impulse response in the time domain. Currently LMDS is considered a line of sight propagation channel with minimal multipath interference because of the millimeter wavelength. One of the research applications of the SSTDSP sounder is to determine whether LMDS propagation through non line of sight or reflected single "bounce paths" can be used to extend the effective coverage of an LMDS communications network beyond the line of sight communications currently possible. The SSTDSP digital channel impulse response can be used to analyze non line of sight or reflected single bounce path network node topologies. The digital impulse response is used to assemble the power delay profile and calculate a number of key metrics that allow us to determine the sustainable bandwidth over that link [2]. By providing this information to Geographic Information Systems (GIS) applications operating in the field, an optimum network topology can be calculated [3]. In addition, the status of the channel can be monitored and radio characteristics optimized to provide control of error correction coding, modulation, and power levels. The challenge now is to leverage theory and practice to provide the theoretical underpinning that enables implementation of this optimal system in the most cost-effective way.

Chapter 2

Background

Chapter 2 presents the necessary background and research on which the SSTDSP sounder was developed and is divided into two major sections: Propagation at LMDS and Channel Sounding.

2.1 Propagation at LMDS

The following section presents the fundamental theory behind propagation of wireless signals in the LMDS band.

2.1.1 What is LMDS?

To better understand the need for the SSTDSP sounder system, further explanation of LMDS is necessary. The acronym LMDS stands for "Local Multipoint Distribution Service" and was created by the Federal Communications Commission (FCC) to describe a two-way digital wireless communications medium that can carry voice, data, and video simultaneously [4]. The primary Block A LMDS spectrum license obtained by Virginia Tech ranges from 27.5 GHz to 28.35 GHz, a total of 850 MHz of continuous spectrum. The primary LMDS Block B license defined by the FCCC extends from 31.075 to 31.225 GHz, a total of 150 MHz of continuous spectrum. LMDS is considered a millimeter frequency, given its approximately 11mm wavelength. When properly deployed, LMDS networks may utilize this large block of spectrum for high-speed communications and

data transfer up to several Gbit/sec. Equipment manufactors are currently claiming that they have product offerings that can provide 622 Mbps. In comparison, upcoming 3G mobile wireless systems may permit transfer speeds of several Mb/sec. This allows LMDS fixed wireless point to point and point to multi-point links to be used as effective "wireless fiber" backbones reaching geographic areas that are uneconomical to reach with landline connections such as fiber optics and cable.

The network topology of an LMDS system is similar to that of mobile cellular radio networks, however the size of the cells are much smaller, often four to five kilometers. The size of these cells is a function of the range of the LMDS equipment, which is heavily influenced by the high propagation path loss and large noise power inherent to millimeter systems such as LMDS. The higher frequencies present in LMDS systems increase the propagation path loss and receiver noise figure; however, the true limitation of broadband wireless systems that utilize LMDS is the large system bandwidth. In the link budgets developed and calculated in this thesis, the large system bandwidth increased the kTB noise power. This phenomenon is further explained in the following sections.

Geographic regions are divided into cells, each with a centrally located hub and multiple receivers or transceivers used by subscribers. Point to point links can be used to link hub to hub using highly directional antennas, while point to multipoint links are used to link hubs with sectored antennas to the local subscribers with highly directional antennas. This user/hub topology can be considered a "star architecture," where one main node serves as the gateway to the outside world. LMDS systems could also be configured in a "mesh architecture," where each node can talk directly to each other without going through a hub. This is more complicated to implement requiring network coordination and innovation multiple access and routing schemes.

2.1.2 What is Unique about LMDS?

The following three things are unique about LMDS:

1. Immense bandwidth for communications services
2. LMDS Operators are not restricted to offer a certain type of service
3. Propagation is profoundly influenced by obstructions and precipitation.

These characteristics mean that LMDS deployment and technology development is both attractive and extremely challenging. Block A LMDS licenses have at their disposal a huge expanse of communications capacity to allot, with the freedom to utilize it however they see fit, provided their signal gets through. Hence the motivation for a measurement system that allows real time characterization of LMDS propagation paths in the field. Given this channel information, GIS site surveys and optimal network deployments can be planned and completed allowing full realization of the potential and promise that LMDS communications link hold. The SSTDSP sounder system allows researchers and system planners to determine optimum LMDS node placement and whether it is possible to use a "bounce" of the signal to fill in a coverage region where a line of sight (LOS) link is not available. The next section provides further explanation and theoretical underpinning for this postulation.

2.1.3 Wireless Propagation At LMDS Frequencies

Discussion of several fundamental aspects of millimeter wave propagation will help shed light on the behavior of LMDS millimeter wave wireless links. Due to the effect that path length, link geometry, and shadowing obstructions have on millimeter wave systems, changes in distance can cause the channel to be extremely time varying and unpredictable. As such, the majority of millimeter wave systems are proposed for more stable fixed line of sight (LOS) links such as LMDS. The small wavelength of millimeter-wave frequency links also mean that most reflective surfaces will look rough with respect to this wavelength, causing random scattering of the incident wave in both

specular and non-specular directions, resulting in both coherent and incoherent field components. This random scattering effect is often seen when millimeter waves propagate through vegetation. Leaves, which act as random scatterers, tending to depolarize, beam broaden, and attenuate propagating signals. Rain and other precipitation can also cause excess attenuation, beam bending, depolarization, and multipath propagation due to reflection [5].

2.1.4 Free Space Path Loss for LMDS

An important source of attenuation in any wireless system is the path loss, or difference between the transmitted power and received power expressed in dB calibrating out various system and antenna gains [5]. The free space path loss is a reduction in flux density per the inverse square relationship ($1/d^2$) that occurs as the transmitted electromagnetic wave spreads out over a radius d. The power attenuation in free space is given by the Friis free space path loss [6], where d is distance, and λ is wavelength

$$PL_{fs} = 20\log\left(\frac{4\pi d}{\lambda}\right)$$

This holds true only for distances in the far-field, or Fraunhofer region of the transmitter antenna. This region satisfies the following relationships [7] where D is the largest dimension of the antenna and d_f is the Fraunhofer distance

$$d >> D$$

$$d >> \lambda$$

$$d >> d_f = \frac{2d}{\lambda}$$

LMDS links often operate at distances of several kilometers, which is much greater than the Fraunhofer distances for such systems, which is less than 200 meters, given antennas dimensions of one meter or less. Most LMDS systems operate at ranges of between two to five kilometers. The free space path loss for such systems ranges from 127 dB to 135 dB respectively. Free space loss can be combined with the other system attenuations and gains to form a link budget for the LMDS link under consideration.

2.1.5 Link Budget for LMDS

A link budget allows wireless system designers to estimate the received power, noise power of the receiver, and the subsequent signal to noise ratio for their system. The received signal power is given by

$$P_r[dBW] = P_t[dBW] + G_t[dB] + G_r[dB] - PL(d)[dB]$$

Where P_t is the transmitted power, G_t and G_r are transmitter and receiver gains respectively, d is the transmitter-receiver separation, and $PL(d)$ is the path loss in the far field, which for LMDS is equal to the free space path loss given by $PL_{fs}(d)$. For a LMDS system with P_t=0.0 dBW, G_t=15.0 dB, G_r=15.0 dB, and $PL(5km)$=135.3 dB, the received power

$$P_r = -105.3\ dBW.$$

The noise power of the receiver is given by

$$P_n[dB] = 10 \log (k) + 10 \log (T) + 10 \log (B)$$

Where k is the Boltzmann constant, which equals 1.38×10^{-23} W/K/Hz, T is the effective noise temperature of the receiver system in Kelvin (K), and B is the receiver bandwidth in Hz. For an LMDS receiver with T = 750 K and suggested LMDS channel bandwidth of B = 80 MHz, the noise power is

$$P_n = -120.8\ dB$$

We can then calculate the signal to noise ratio in the receiver by

$$SNR[dB] = P_r[dB] - P_n[dB]$$

For this particular example the signal to noise ratio is

$$SNR = 15.5\ dB$$

This is a very basic example - other losses or gains due to atmospheric losses or additional amplifier gains may be included in the calculation of the link budget. Table 1 below shows a typical spreadsheet calculation for an LMDS link budget.

Sample link budget for an LMDS system

Carrier Frequency		27.92500 GHz
Wavelength		0.01074 meter
Power Budget		
Transmitted power, Pt	1 Watt	0.0 dBW
Tx Antenna Gain, Gt		15.0 dB
Rx Antenna Gain, Gr		15.0 dB
Path Loss, Lp	5000 meter	-135.3 dB
Received Power		**-105.3 dBW**
Noise Budget		
Boltzmann Const		-228.6 dBW/K/Hz
Noise Temp.	750 Kelvin	28.8 dBK
Noise bandwidth	80.0 MHz	79.0 dBHz
Noise power		**-120.8 dBW**
Signal to Noise (SNR)		15.5 dB

Table 2.1: Sample LMDS Link Budget

Chapter 3 provides a detailed link budget for the SSTDSP sounder.

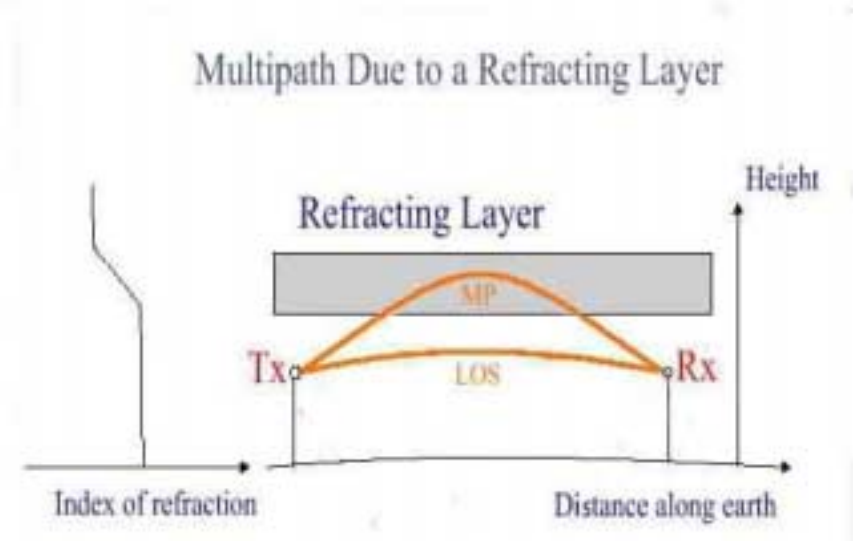
2.1.6 Refraction, Reflection, Scattering, and Diffraction

Several words should be said about typical wireless propagation environments. Very rarely will a receiver only detect the original transmitted electromagnetic (EM) wave. Not only will the line of sight (LOS) signal be received, but also a number of copies of the signal may arrive at the receiver delayed in time and possibly altered. These waves have traveled to the receiver over paths other than the LOS path, and are called "multipath signals" and are often considered interference. Such multipath components are usually created because of refraction, reflection, scattering, or diffraction of the transmitted EM

wave. This section outlines those four phenomenon and their relationship to wireless channel behavior.

Refraction

Under certain weather conditions, the atmosphere can cause the propagation paths of transmitted EM waves to "bend," thereby possibly producing additional multipath components at the receiver. This wave bending or "refraction" is governed by the atmosphere's "index of refraction." Figure 2.1 below illustrates this phenomenon.



Excerpted from [38]

Figure 2.1: Multipath propagation due to refractive index change.

An increase in multipath interference in LMDS channels due to refraction is most noticeable during weather related changes in the atmosphere, and as such tend to be sporadic and random over time.

Reflection

When an EM wave impinges on a surface with finite conductivity, part of the energy is reflected and part of the energy is transmitted into the surface. If the EM wavelength is

much larger than the surface irregularities, the surface will appear smooth. However, if the wavelength is close in size to the surface irregularities, the surface appears rough. If the wavelength is smaller than the surface irregularities, it can often cause the energy to be absorbed, preventing reflection. When a reflecting surface appears smooth, the relationship between the reflected wave and the transmitted wave is governed by the reflection and transmission coefficients given by Snell's law [8].

$$\theta_r = \theta_i$$

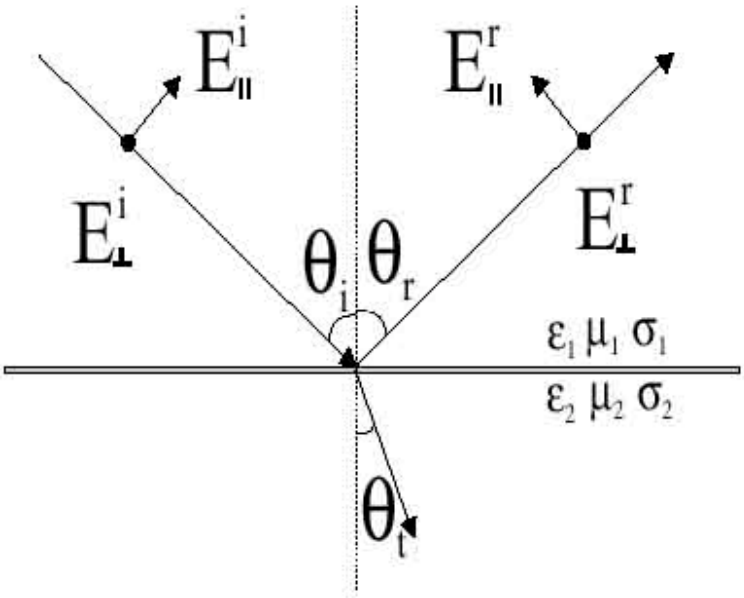
$$E_r = \Gamma E_i$$

$$\Gamma_{\parallel} = \frac{E_r}{E_i} = \frac{\eta_2 \cos(\theta_t) - \eta_1 \cos(\theta_i)}{\eta_2 \cos(\theta_t) + \eta_1 \cos(\theta_i)}$$

$$\Gamma_{\perp} = \frac{E_r}{E_i} = \frac{\eta_2 \cos(\theta_i) - \eta_1 \cos(\theta_t)}{\eta_2 \cos(\theta_i) + \eta_1 \cos(\theta_t)}$$

$$\eta_i = \sqrt{\frac{\mu_i}{\varepsilon_i}}$$

Where θ_r is the angle of reflection, θ_t is the angle of incidence, E_r is the reflected field intensity, and Γ is the reflection coefficient which equals $\Gamma_{//}$ or $\Gamma_{\perp}$ for parallel or perpendicular polarization, and η_i is the intrinsic impedance of the ith medium (i=1,2). Such geometry ensures that the angle of reflection equals the angle of incidence, which produces "specular reflection" or often just "reflection," shown below in Figure 2.2.



Excerpted from [5]

Figure 2.2: Geometry of specular reflection over a smooth surface.

This thesis is particularly interested in designing a measurement system that will indicate whether reflected signal "bounce path" LMDS signals can be used to extend the coverage of an LMDS system past the existing line of sight coverage.

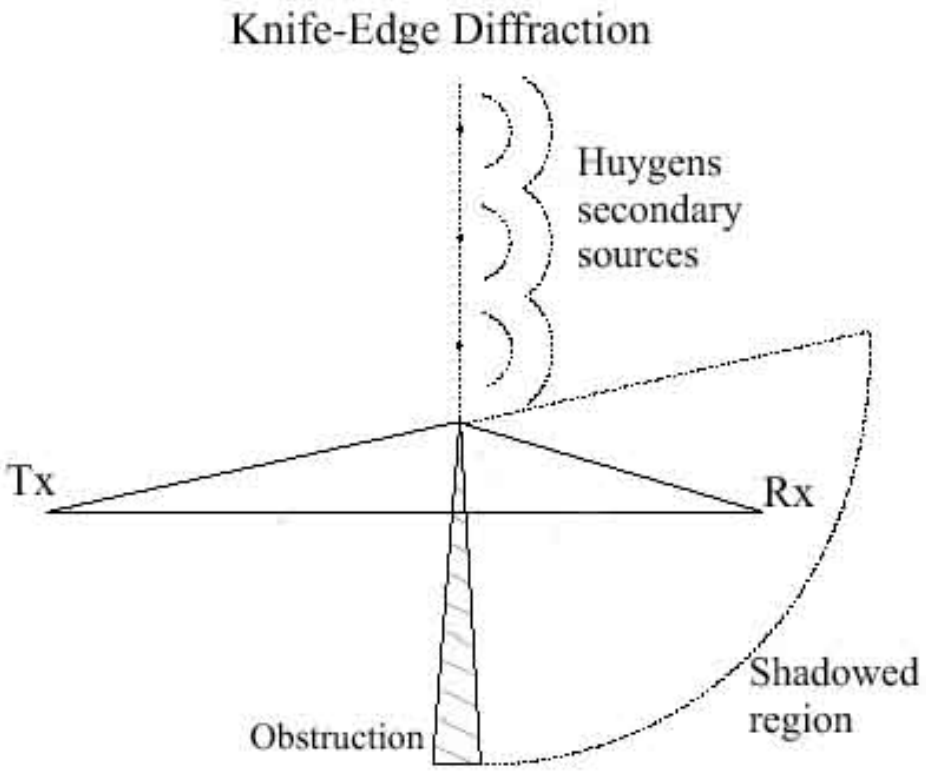
Scattering

When the surface is rough, the impinging wave is not only scattered in the specular direction, but also in other directions. This form of incoherent reflection is called "scattering" or diffuse reflection. As surface roughness increases, specular reflection decreases and incoherent scattering increases. Scattering at LMDS frequencies tends to depolarize, beam broaden, and attenuate propagating signals, which necessitates careful system design to compensate for these phenomenum. Both specular reflections and incoherent scattering can cause multipath inference; however, this thesis will be primarily interested in using reflection to "fill in" LMDS coverage regions.

Diffraction

Diffraction describes how radio waves can propagate around obstacles such as buildings, hills, and other geographical obstructions to the LOS path and allow coverage of users who are blocked from the LOS path to the transmitter. These users are said to be in the "shadow region" of the transmitter. Such diffracted multipath components are considered useful since they can often provide additional system coverage. These diffracted LMDS signals might also be used to extend the coverage of an LMDS system past the existing line of sight coverage. The discussion over whether non line of sight LMDS extended coverage can be better achieved using reflection or refraction is still a topic of debate. [39]

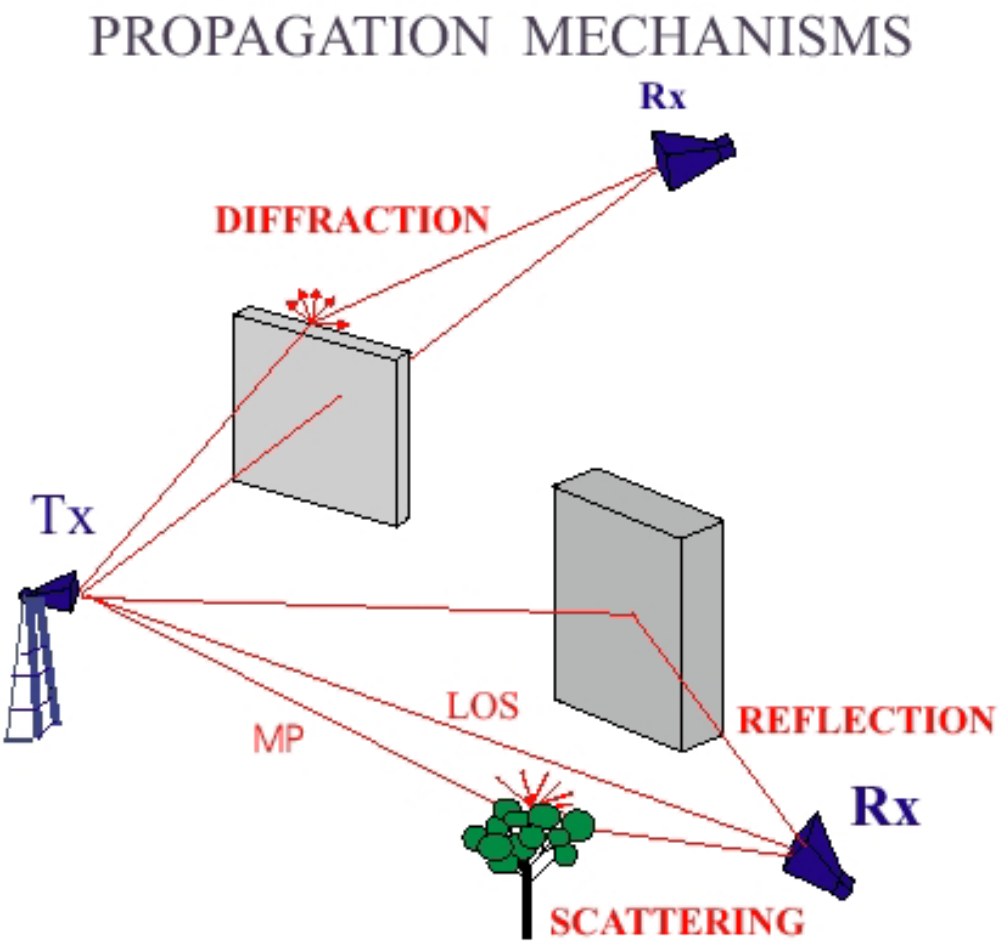
Diffraction can be explained by Huygen's principle, which states that each point on a wavefront can be considered as the source of a secondary wavelet, and these wavelets combine to form a new wavefront in the direction of propagation [9]. Diffraction is caused by the propagation of the secondary wavelets in the shadow region. Figure 2.3 below provides an example of such propagation called "knife edge diffraction." This figure clearly illustrates how the diffracted waves provide signal coverage in the shadow region. This produces an effective gain in received power over the free space path loss, which effectively extends the coverage region of the transmitter into the shadow region.



Excerpted from [5]

Figure 2.3: Knife edge diffraction showing coverage in the shadow region.

The relationship between diffraction, reflection, and scattering is illustrated below in Figure 2.4 [5].



Excerpted from [5]

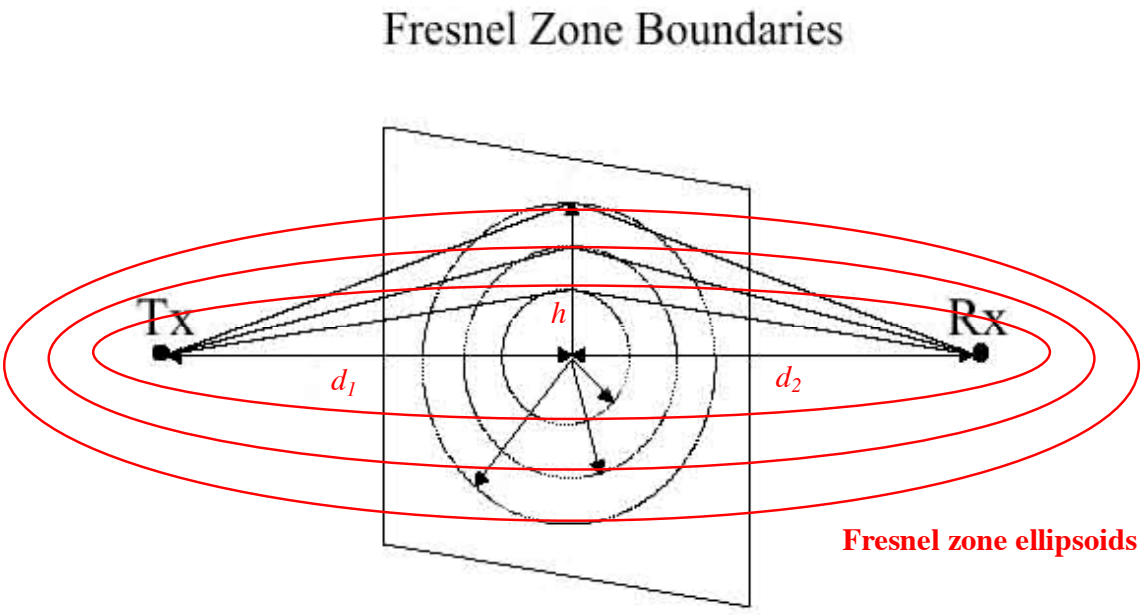
Figure 2.4: Fundamental Propagation mechanisms.

2.1.7 Fresnel Zone Clearance

The presence of diffraction along a given path is highly dependent on the link geometry. Fresnel zones explain the concept of diffraction loss as a function of the path difference around an obstruction and can be used to describe the level of diffraction throughout regions of space.

Fresnel zones represent successive regions where secondary waves have a path length which are $n\lambda/2$ greater than the total path length of a line of sight (LOS) path, where n is

the Fresnel zone and λ is the wavelength. If a transparent plane is placed at the location of the obstacle, the secondary wavelets will form concentric circles on the plane which represent the loci of the origins of secondary wavelets that propagate to the receiver, such that the total path length increases by wavelength/2 for successive circles. These circles provide the boundaries for Fresnel zones. A family of ellipsoids may be constructed between the transmitter and receiver by joining all the points for which the excess path delay is an integer multiple of half wavelengths [6]. These ellipsoids represent Fresnel zones, as illustrated in Figure 2.5 below.



Excerpted from [5]

Figure 2.5: Illustration of Fresnel Zone Boundaries and Fresnel Zones.

Successive Fresnel zones have the effect of alternately providing constructive and destructive interference to the total received signal.

The level of diffraction loss corresponds to the amount of blockage of the volume contained within the Fresnel zones. To avoid diffraction loss and therefore loss in coverage, point to point system designers should ensure that in addition to LOS path clearance, 56% of the first Fresnel zone is kept clear from obstruction [10].

In a LOS path, the Fresnel zone clearance affects the multipaths seen by the LMDS receiver and as such is a very important parameter when determining the multipath profile of an LMDS channel.

To determine the required multipath component resolution required to resolve a given propagation path, the excess path length (Δ)must be calculated. The excess path length is defined as the difference between the direct line of sight path and the diffracted path [6]. Consider a transmitter and receiver separated in free space as shown in Figure 2.5 separated by an obstructing screen of height h with infinite width placed at a distance d_1 from the transmitter and d_2 from the receiver. Assuming that $h << d_1$ and $h >> \lambda$, then

$$\Delta = \frac{h^2}{2}\frac{(d_1 + d_2)}{d_1 d_2}$$

If a building serving as the obstruction has height h = 50 m and is located at a distance d_1 = 2500 m from the transmitter and d_2 = 2500 m from the receiver, then the excess path length is 1 m.

2.1.8 Is There a Bounce at LMDS?

The SSTDSP sounder has been designed to allow investigation into whether there is a way to extend coverage of LMDS systems by using a "bounce path" to reach non-line of sight users. A bounce path is essentially a multipath component used for communication. Reflection or diffraction may produce this multipath component.

The following paragraphs were excerpted from [11]

> **"As part of an effort to validate Geographic Information Systems (GIS) predictions of LMDS coverage, the Center for Wireless Telecommunications (CWT) set up a propagation experiment at VA Tech (2). This experiment compared predicted and measured CW signal strength at various points on the VA Tech campus at 900 MHz and 28 GHz.**

> **GIS more accurately predicted the 28 GHz coverage than the 900 MHz coverage. As expected, there was considerable bouncing of the 900 MHz signal but the 28 GHz signal unexpectedly "filled in" areas that were not expected to be visible from the transmitter. Most objects are "rough" at LMDS wavelengths and therefore there will not be specular reflections. However, the signal powers were high enough to wonder if communications might be possible over non-line of sight paths, and, if so, how much bandwidth could be supported?**
>
> **The question is can GIS do more than predict line of sight or viewshed? Can it be applied to predict a "comshed." That is a region of supportable bandwidth that can be seen from a transmitter. This region may or may not be LOS from the transmitter. Wideband channel measurement will be necessary to answer this question." [11]**

The question of the existence of a LMDS bounce path was one of the motivating factors behind the design and implementation of the SSTDSP sounder. The following chapter discusses how and why the SSTDSP sounder method was synthesized from existing channel sounding methods, along with the underlying theory behind channel sounding.

2.2 Channel Sounding

The following section presents the underlined theory of channel sounding including a historical perspective and comparison of several common channel sounding techniques. In addition, the theory that serves as the foundation for the Sampling Swept Time Delay Short Pulse (SSTDSP) sounder is synthesized from the existing channel sounding techniques.

2.2.1 Channel Modeling at LMDS

Wireless channel models best describe the effects that the wireless channel has on the link between transmitter and receiver. The most common way to model such channels is as a linear filter, which may or may not be time varying. If the channel is a mobile or portable radio link, it will inevitably vary with time. While a stationary transmitter and receiver may not experience the time varying effects that would be present if they were moving, the channel between the stationary transmitter and receiver may still vary with time with changing path conditions like those resulting from moving obstructions (cars, tree leaves, etc.). Since the LMDS wireless channel is considered a fixed channel due to the stationary nature of the LMDS transmitter and receiver nodes and most links are line of sight or free from time varying obstructions, time variability is of little concern. Therefore, we will consider the LMDS channel to be a Linear Time Invariant (LTI) filter.

LTI filters can be characterized in two different but related ways: as a complex transfer function in the frequency domain or the impulse function in the time domain. Either of these functions completely describes the behavior of the filter and can be used to determine the filter output for any arbitrary input. The SSTDSP sounder measures the impulse response of the LMDS channel in the time domain and treats it as a LTI filter response. This filter response can be used to characterize the behavior and performance of the LMDS link path under study by predicting the maximum usable bandwidth and calculating the frequency response.

The impulse response $h(t,\tau)$ of a filter is defined as the output of the filter when a unit impulse $\delta(t)$ is impressed on the input. The LMDS channel can be though of as a filter placed between the transmitter and receiver which can vary with absolute time t and can be defined using relative time τ. Since the received signal in a multipath channel consists of a series of attenuated, time-delayed, phase shifted replicas of the transmitted signal, the baseband impulse response of a multipath channel can be expressed as

$$h(t,\tau) = \sum_{i=0}^{N-1} a_i(t,\tau)\exp\{j2\pi f_c \tau_i(t) + \varphi_i(t,\tau)\}\delta(\tau - \tau_i(t))$$

where $a_i(t,\tau)$ and $\tau_{i,}(t)$ are the real amplitudes and excess delays respectively of the ith multipath component at time t and N is the total number of possible multipath components based on the discretized time bins, including the first arriving component. The phase $2\pi f_c \tau_i(t) + \varphi_i(t,\tau)$ represents the phase shift due to free space propagation of the ith multipath component plus any addition phase shifts in the channel, while the term $\delta(\tau - \tau_i)$ is an impulse introduced into the channel at $\tau_i = \tau_0$ [12].

If the channel impulse response is considered time invariant, or is at least wide sense stationary over a small-scale time or distance interval, as are LMDS links, the channel impulse response may be simplified to

$$h(\tau) = \sum_{i=0}^{N-1} a_i \exp\{-j\theta_i\}\delta(\tau - \tau_i)$$

where a_i and τ_i are the real amplitudes and excess delays respectively of the ith multipath component, N is the total number of possible equally spaced multipath components, including the first arriving component, and θ_i represents all phase shifts for the ith multipath component lumped together.

Figure 2.6 illustrates the baseband LTI channel impulse response equation.

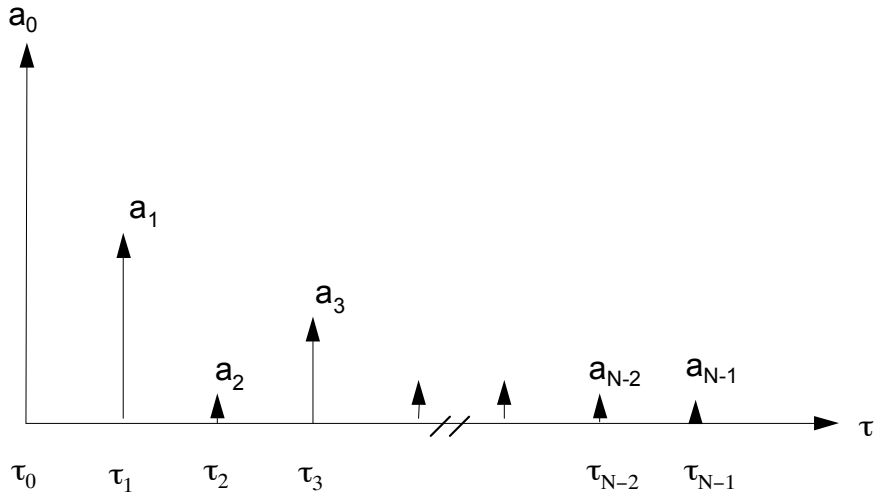


Figure 2.6: LTI baseband channel impulse response.

Due to difficulty implementing true delta functions, a probing pulse $p(\tau)$ which approximates an impulse function $\delta(\tau - \tau_i)$ is used at the transmitter

$$p(\tau) \sim \delta(\tau - \tau_i)$$

to sound the channel and measure or predict $h(\tau)$. For small scale channel modeling, the "power delay profile" of the channel is found by taking the spatial average of $|h(\tau)|^2$ over a local area. By making several measurements of $|h(\tau)|^2$ over a local area, it is possible to build an ensemble of power delay profiles, each one representing a possible small-scale multipath channel state [13]. For channel sounders that use wideband $p(\tau)$ probing pulses that can resolve multipaths, the small scale received power is simply the sum of the powers received in each multipath component. Therefore, since the amplitudes of individual multipath components do not fluctuate widely in a local area, the received power of a wideband probing pulse will not fluctuate significantly when a receiver is moved around a local area [14]. The SSTDSP sounder has been designed to operate when no data is being transmitted over the wideband LMDS channel. This is achieved by assigning the SSTDSP sounder its own time slot within the Time Division Multiple Access (TDMA) LMDS frame scheme. TDMA is a multiple access scheme used in wireless telecommunications systems that permits multiple access of the same frequency channel by allocating each user an amount of time to transmit on the channel within a given repeating time frame.

If a signal $c(\tau)$ is transmitted through an LTI channel filter with impulse response $h(\tau)$, the received signal $r(\tau)$ can be calculated by convolving $c(\tau)$ with $h(\tau)$

$$\mathrm{r}(\tau) = \sum_{i=0}^{N-1} c(\tau_i) h(\tau - \tau_i)$$

For the case where $c(\tau)$ is an impulse, then $r(\tau)$ will be the impulse response $h(\tau)$.

Work done by Cox [15], [16] showed that provided $p(\tau)$ has a time duration much smaller than the resolution (time bin delay) between multipath components, then $p(\tau)$ can be treated as an impulse and each multipath component within the impulse response can be

treated as an impulse corresponding to a particular path delay. Since a transmitted impulse convolved with the channel impulse response is simply the channel impulse response, the signal seen at the receiver will be the channel impulse response. This means that $r(\tau)$ and can be directly sampled by the receiver at each time delay bin to determine the relative multipath component signal strengths and recover the channel impulse response. Cox's work eliminates the need to deconvolve $p(\tau)$ from the received signal $r(\tau)$ in order to recover $h(\tau)$ and the relative multipath signal strengths. This would require taking the Fourier transform of the received signal $r(\tau)$, multiplying that with the reciprocal of the Fourier transform of $p(\tau)$ in the frequency domain $B(f)$, and then converting $H(f)$ back to the time domain using the inverse Fourier transform to recover the impulse response $h(\tau)$. The following equations illustrate this process.

$$\Im(r(\tau)) = \Im(p(\tau) * H(\tau)) \Leftrightarrow R(f) = P(f)H(f)$$

$$H(f) = \frac{R(f)}{P(f)} = R(f)\frac{1}{P(f)}$$

$$b(\tau) \Leftrightarrow \Im^{-1}\left(\frac{1}{P(f)}\right) = B(f)$$

$$h(\tau) = r(\tau)*b(\tau) \Leftrightarrow \Im^{-1}\left(H(f)\right)$$

The received power delay profile $P(\tau)$ is given by

$$P(\tau) \sim k\,|h(\tau)|^2$$

Where k relates the transmitted power in the probing pulse $p(\tau)$ to the total power received in a multipath delay profile. Since wideband channel sounders such as the SSTDSP sounder can treat wideband probing signals such as $p(\tau)$ as impulse functions, they can directly measure the power delay profile of the channel without deconvolving $p(\tau)$ from the received signal $r(\tau)$ to determine relative multipath signal strengths in $h(\tau)$. In practical terms, this means a very short pulse can be used to sound the channel and the received signal can be directly sampled to recover multipath components. This method is leveraged in the SSTDSP sounder.

To determine the frequency response of the channel, the Fourier transform of the impulse response $h(\tau)$ must be calculated. The SSTDSP sounder uses digital sampling in discrete time to capture the impulse response. Hence the finite data sequence $\{a_0, a_1, \ldots, a_{N-1}\}$ represents the amplitudes of multipath components a_i within a channel impulse response, where each component at a time index of $i=\{0,1,\ldots N-1\}$ represents a different path delay. Each time index i is separated by a time bin delay, equal to $\tau_1 - \tau_0 = \tau_1$. These values represent the result of sampling the continuous time impulse response $h(\tau)$ at times $\tau=\{0, T_s \ldots, (N-1)T_s\}$ where T_s is the sampling interval.

The discrete Fourier transform (DFT) is then used to calculate the transform sequence G_k which represents the frequency response H(f) of the finite length impulse response $h(\tau)$ represented by the sequence $\{a_0, a_1, \ldots, a_{N-1}\}$.

$$g_n = h(iT_s) = a_i, \quad i=n$$

$$G_k = \sum_{n=0}^{N-1} g_n \exp\{-\frac{j2\pi}{N} kn\}, k = 0,1,\ldots,N-1$$

Conversely, the original finite data sequence g_n can be recovered from the transform sequence G_k by computing the Inverse Discrete Fourier Transform (IDFT)

$$g_n = \frac{1}{N} \sum_{k=0}^{N-1} G_k \exp\{\frac{j2\pi}{N} kn\}, n = 0,1,\ldots,N-1$$

The DFT and IDFT form a transform pair.The digital signal and data processing algorithms in the SSTDSP sounder use an efficient form of the DFT called the Fast Fourier Transform (FFT) to calculate the frequency domain equivalent of the channel impulse response [17].

Now that the LTI discrete impulse response has been established as a model for the LMDS channel, we will investigate several channel metrics that can be calculated from the power delay profile.

2.2.2 Channel Metrics for LMDS

Several channel metrics can be calculated from the power delay profile. The mean excess delay, rms delay spread, maximum excess delay or excess delay spread, and coherence bandwidth allow wireless system designers to grossly quantify the multipath channel. The time dispersive properties of wideband multipath channels are most commonly quantified by their mean excess delay and rms delay spread [6]. The mean excess delay is the first moment of the power delay profile $P(\tau)$ and is defined to be:

$$\bar{\tau} = \frac{\sum_k a_k^2 \tau_k}{\sum_k a_k^2} = \frac{\sum_k P(\tau_k)\tau_k}{\sum_k P(\tau_k)}$$

Where a_k are the amplitudes of multipath components, $P(\tau_k)$ is the power, and τ_k is the delay at a time index of $k=\{0,1,\ldots N-1\}$. The mean excess delay value can be thought of the "average" multipath delay. The rms delay spread is the square root of the second central moment of the power delay profile $P(\tau)$ and is defined to be:

$$\sigma_\tau = \sqrt{\left(\overline{\tau^2} - \bar{\tau}^2\right)}$$

Where

$$\overline{\tau^2} = \frac{\sum_k a_k^2 \tau_k^2}{\sum_k a_k^2} = \frac{\sum_k P(\tau_k)\tau_k^2}{\sum_k P(\tau_k)}$$

The delay values τ_k and power values $P(\tau_k)$ are measured relative to the first detectable signal in the power delay profile, occurring at τ_0=0. The rms delay spread is effectively the variance around the mean excess delay and is used to estimate the coherence bandwidth.

The maximum excess delay for a given multipath threshold *X dB* of the power delay profile $P(\tau)$ is defined to be the time delay during which the energy of the measured multipath component falls *X dB* below the peak value of the power delay profile. Mathematically this value can be expressed as $\tau_X - \tau_0$, where τ_0 is the first arriving signal

and τ_X is the maximum delay at which a multipath component is no more than *X dB* down from the strongest arriving multipath component. For blocked paths, the strongest multipath component may not necessarily arrive at τ_0. The value of τ_X is often referred to as excess delay spread for a given threshold *X dB*. Since the SSTDSP sounder will be used to measure line-of-sight paths or to investigate the use of non line of sight reflections or "bounce paths", the digital processing algorithms assume that the largest impulse occurs in the first time bin for either line of sight or bounce path links. This first impulse therefore corresponds to either the line of sight path or a non-line of sight bounce path. This algorithm may be altered for different sounding scenarios.

The coherence bandwidth is defined as a statistical measure of the range of frequencies over which the channel responses have approximately equal gain and linear phase, or the bandwidth over which the frequencies are correlated. The coherence bandwidth can be measured indirectly from wideband measurements by taking the Fourier transform of the average power delay profile [16], or measured directly from frequency sweeping measurements using a Vector Network Analyzer (VNA) with Sampling Swept Frequency Oscillator [18], [19], [20]. The coherence bandwidth provides a loose bound that helps define the "usable bandwidth" in a particular fixed point LMDS communications link path.

The coherence bandwidth B_c can also be related to the rms delay spread for correlation levels of 0.9 and 0.5 respectively by the following loose bounds:

$$1/50\sigma_\tau < B_c < 1/5\sigma_\tau$$

The SSTDSP sounder calculates the coherence bandwidth using the above bounds and provides a second reference by plotting the Fourier transform of the average power delay profile. Repeated tests of the SSTDSP algorithm have confirmed that the estimated B_c can be visually confirmed by the 3dB bandwidth shown on the plot of the FFT of the power delay profile, which is expected.

Now that the LTI discrete impulse response has been established as a model for the LMDS channel and channel metrics have been established, we will investigate ways of obtaining the impulse response and power delay profile by channel sounding.

2.2.3 Channel Sounding History and Methods

The best and most effective way to obtain the impulse response and power delay profile of a channel is to employ wideband sounding techniques. Taking a number of simultaneous or sequential narrowband measurements may be less costly and complex than using signals that occupy a wide bandwidth because the waveforms in narrowband measurements occupy much less bandwidth than wideband signals and therefore require less costly equipment to capture. However, the limitations of narrowband measurements have prompted researchers to develop a number of wideband sounding techniques aimed at retaining the performance and detail provided by wideband measurements, but at a fraction of the cost. The SSTDSP method introduced in this thesis is a synthesis of such wideband techniques. This section will therefore focus on wideband sounding techniques.

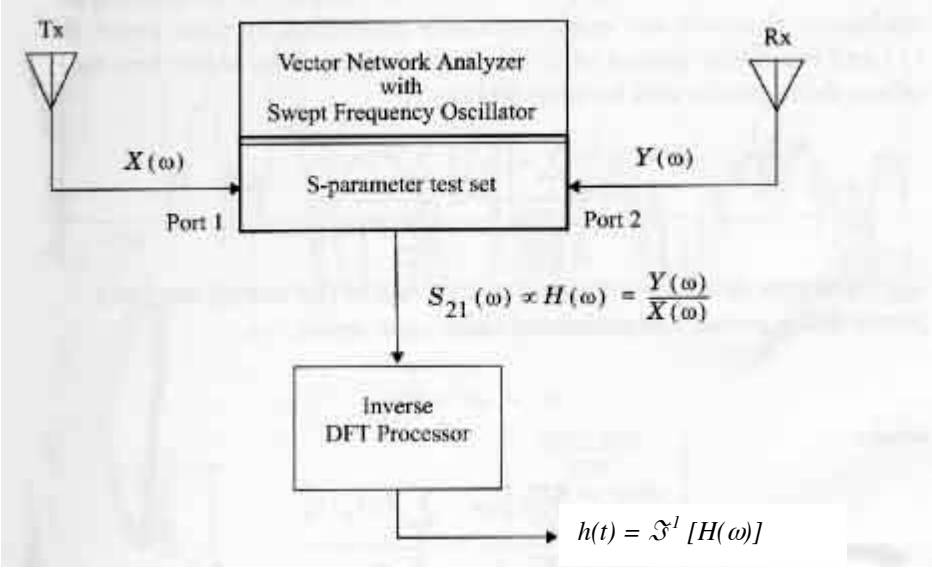
Wideband wireless channel sounding techniques are currently segmented into the following categories:

1. **Swept frequency domain methods**
2. **Periodic pulse methods**
3. **Pulse compression methods based on convolution and correlation**

The SSTDSP method synthesizes aspects of the periodic pulse method with the swept time delay processing inherent to the pulse compression method based on correlation. These various channel sounding techniques will now be discussed in further detail.

Swept frequency domain methods

The dual relationship between time domain and frequency domain channel sounding techniques, permits channel impulse response measurements to be taken in the frequency domain. Figure 2.7 below shows a swept frequency domain channel sounder used for measuring channel impulse responses [6].



Excerpted from [6]

Figure 2.7: Swept frequency domain channel impulse response measurement system.

A vector network analyzer (VNA) controls a synthesized frequency sweeper, and an S-parameter test set is used to monitor the frequency response of the channel. The sweeper scans the frequency band under study centered on a carrier by stepping through discrete frequencies steps and transmitting a known signal level on port 1 and monitoring the received signal on port 2. These signal levels allow the analyzer to determine the transmissivity response $S_{21}(\omega)$ over the measured frequency range. The transmissity response is a frequency domain representation of the channel impulse response and is converted to the time domain using the Inverse Discrete Fourier Transform (IDFT), producing a bandlimited version of the impulse response.

This method can indirectly provide amplitude and phase information in the time domain, however it is cumbersome and costly to operate because of the precise calibration and stringent synchronization between transmitter and receiver that are required. Often the tight timing requirements necessitate direct connection between transmitter and receiver which limits the usable range of the system to short distance channel sounding measurement campaigns, such as indoor channel sounding.

The number and spacings of the frequency steps impact the time resolution of impulse response measurement. The sweep bandwidth of the VNA

$$B_{sw} = 2 / \Delta\tau$$

is based on the desired time resolution $\Delta\tau$ of the impulse response. For a given sweep bandwidth of B_{sw} and total number of data points N, the frequency step $\Delta f = B_{sw} / N$ determines the unambigiuous period $T_p = 1 / \Delta f$ of the measurement such that aliasing of multipath components does not occur.

The sweep time should be chosen such that the channel remains invariant during the entire sweep. This hinders use of the frequency domain sounder in mobile wireless channels that vary quickly with time. A faster sweep time can be accomplished by reducing the number of frequency steps which increases the frequency step size for a given sweep bandwidth B_{sw}. However, this reduces the time resolution and excess delay range in the time domain. Since better resolution or longer unambiguous range results in a longer sweep time, a balance of measurement resolution, unambiguous range, and measurement speed is required. The swept frequency domain system has been used successfully by Pahlavan [18] and Zaghloul, et. Al. [19], [20].

Due to the cost, complexity, and limited range inherent to the system, the swept frequency domain sounding method was not a practical approach to characterizing the wideband LMDS channel. Other more practical time domain channel sounding methods will now be presented.

Periodic Pulse Sounding Methods

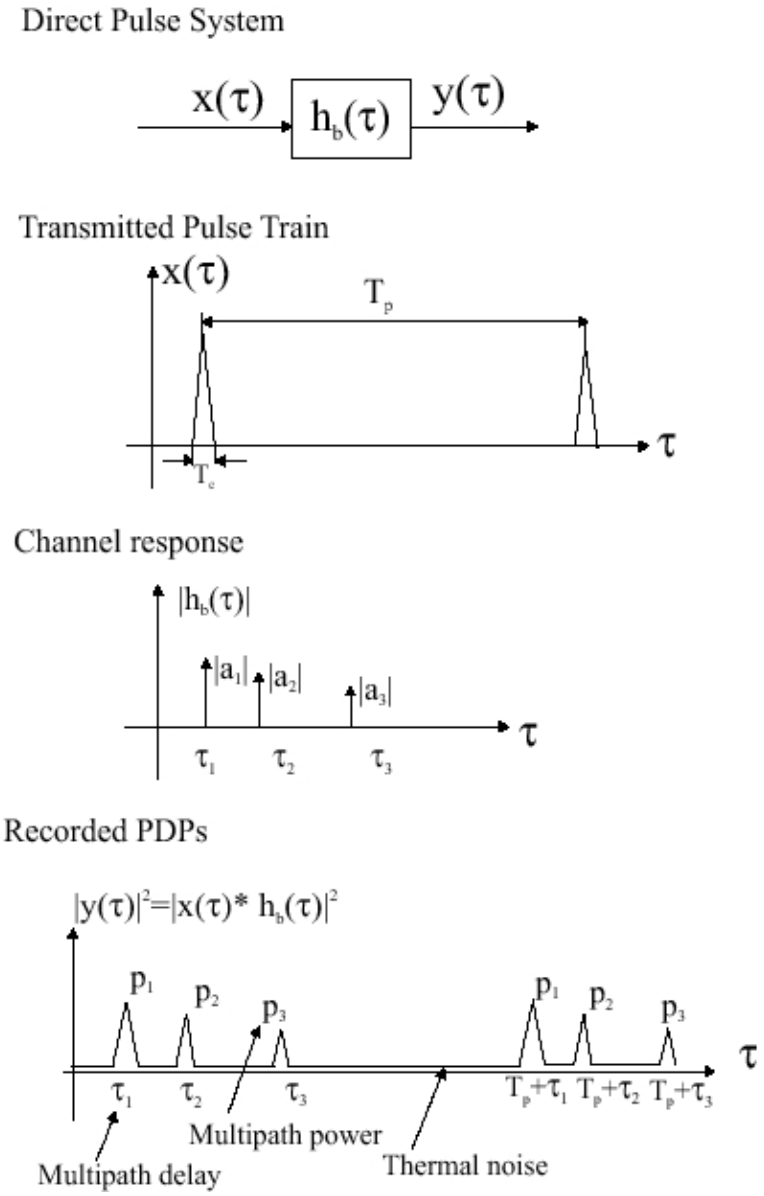
The quickest and least complex way of observing the impulse response of a wireless channel is to stimulate the channel with an impulse or a waveform that approximates an impulse. The periodic short pulse waveform p(t) is treated as impulse and has a pulse width that corresponds to half the bandwidth of the of the spectrum to be characterized. The waveform p(t) is used to initiate the channel impulse response. This method of channel sounding is called "period pulse sounding" or "direct RF pulse sounding."

Periodic pulse techniques have been in use since before World War II to measure characteristics of radio paths. In 1952 Delange at Bell Laboratories studied a line of sight channel using 1 watt pulses at 4 GHz with a pulse duration of 3 ns at half amplitude. He observed multipath transmission effects during fading periods indicating delay spreads of 7ns over a 35 km path [21]. The first reported study of impulse responses using a periodic pulse sounder in a mobile radio propagation channel was by Young and Lacy [22] in urban New York city at 450 MHz using a sounder with pulse duration of 0.5us (equivalent spatial resolution = 150 m). A further study was conducted by Turin [12] in San Francisco using essentially the same method as Young and Lacy, but with a 1us duration pulse (300m spatial resolution) and carrier frequencies of 488 MHz, 1280 MHz, and 2920 MHz. Later Van Rees [23, 24] conducted similar experiments from a moving vehicle at 10W peak power, 910 MHz carrier frequency, and pulse repetition period of 100us, using pulse durations ranging from 50 ns to 200 ns, capable of resolving path length differences as small as 15 m to 60 m.

The transmitted pulse duration determines the time bin resolution within the power delay profile. Each time bin of duration τ_{bb} in the power delay profile corresponds to a path length delay. By multiplying τ_{bb} by the speed of light, the multipath resolution can be calculated. Path length differences for each multipath component can be calculated by multiplying the excess delay time relative to the first pulse in the impulse response by the speed of light.

The periodic pulse sounding system relies on the fact that the received signal is the convolution of the transmitted impulse train with the channel impulse response, which produces the receive impulse response. This system can be thought of as a wide band bistatic radar, sending out periodic ranging pulses of pulse width τ_{bb} and watching for received echoes of the original pulse, which represent different received multipath components that can be captured to produce the impulse response and power delay profile.

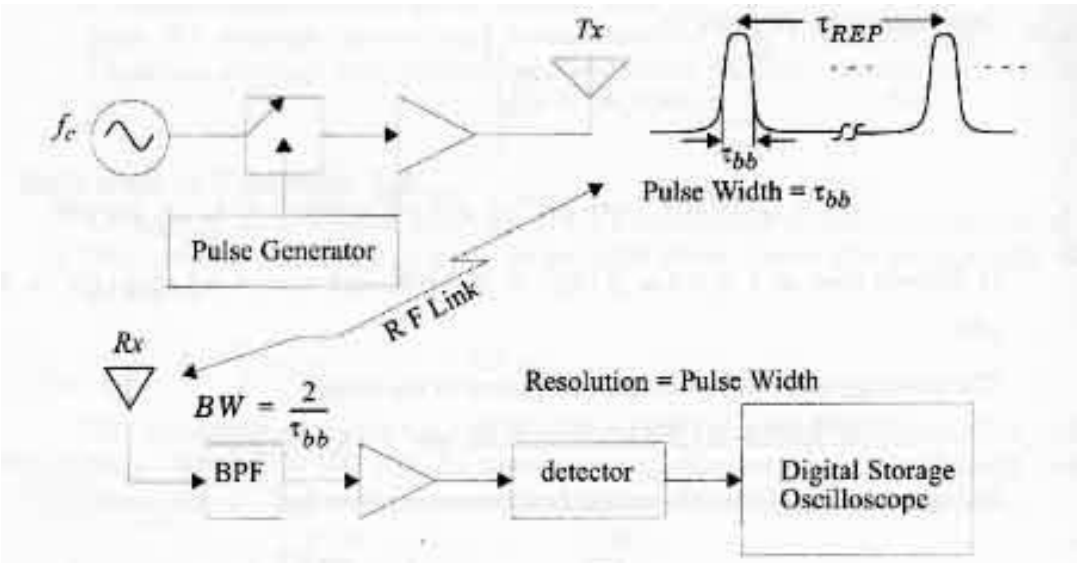
Figure 2.8 below illustrates this convolution process, where the channel response $|h_b(\tau)|$ is convolved with transmitted periodic pulse train with pulse repetition period of T_p, pulse width of T_c, and relative time bin delays of τ_i, where $i=0,1,\ldots N$.



Excerpted from [5]

Figure 2.8: Principle of operation of a periodic pulse sounder.

The periodic pulse sounding system is shown in Figure 2.9 below.



Excerpted from [6]

Figure 2.9: Periodic pulse sounder system.

The transmitter usually generates a carrier that is gated on and off by a pulse generator, which is then amplified before being radiated over the channel by the antenna. The received band pass signal is then amplified, detected with an envelope detector, and displayed and stored on a high speed oscilloscope. This provides an immediate visualization of the channel impulse response. The minimum resolvable delay between multipath components is equal to the probing pulse width τ_{bb} and the measurable bandwidth is $BW=2/\tau_{bb}$.

This system has a number of limitations including susceptibility to interference and noise because of the wide passband filter required for multipath time resolution and the fact that the system relies on the ability of the oscilloscope to trigger on the first arriving signal. If the first signal is blocked or fades, the system may not trigger properly. While the phases of the individual multipath components in a periodic pulse sounder are not recovered by an envelope detector, this can be overcome using a coherent detector. The periodic pulse sounder system is also limited by its requirement for a high peak-to-mean power ratio that allows adequate detection of multipath echoes. Typically a video detector is used for detection, which limits the system's dynamic range and therefore

ability to detect weak echoes. This method of channel characterization also requires costly high speed sampling scopes to digitize the data.

The SSTDSP sounder addresses these limitations through a combination of novel digital signal processing, ultrawideband sampling methods, and swept time delay processing. Swept time delay processing will be introduced in greater detail in the following section on pulse compression sounders.

Pulse Compression Methods

The basis for all pulse compression systems is contained in the theory of linear systems [25]. It is known that if white noise *n(t)* is applied to the input of a linear system and the output *w(t)* is cross correlated with a delayed replica of the input $n(t-\tau)$, then the resulting cross correlation coefficient is proportional to the impulse response of the system, $h(\tau)$ evaluated at the delay time. This can be shown as follows:

$$E[n(t)n^*(t-\tau)] = R_n(\tau) = N_0\,\delta(\tau)$$

Where $R_n(\tau)$ is the autocorrelation function of the noise, and N_0 is the single-sided noise power spectral density. The output of the linear channel is given by the convolutional relationship:

$$w(t) = \int h(\xi)n(t-\xi)d\xi$$

So the cross-correlation of the output and the delayed input is given by,

$$\begin{aligned} E[w(t)n^*(t-\tau)] &= E[\int h(\xi)n(t-\xi)n^*(t-\tau)d\xi] \\ &= \int h(\xi)\mathrm{R}_{\mathrm{n}}(\tau-\xi)d\xi \\ &= \mathrm{N}_0 h(\tau) \end{aligned}$$

Therefore, the impulse response of a linear system can be evaluated using white noise, and some method of correlation processing. Due to difficulty generating white noise, most systems use deterministic waveforms that have noise like character, such as maximal-length pseudo-random binary sequences (m-sequences), usually called pseudo-noise (PN) codes. These codes are extremely popular in communications, navigation, and

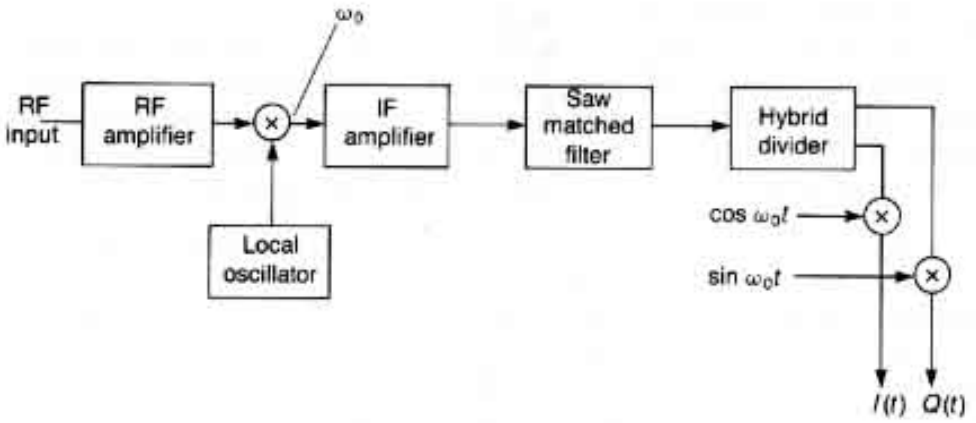
ranging systems because they are easily generated using inexpensive logic-based linear feedback shift registers and have favorable periodic autocorrelation properties [25],[26],[27].

Pulse compression by convolution matched filter technique

One way to affect pulse compression is to use a filter that is matched to the sounding waveform. This method, called the convolution matched-filter technique, has been used in a study by Bajwa et. al. at 436 MHz using a Surface Acoustic Wave (SAW) device for the matched filter [28]. The SAW filter is matched to the specific sequence at the transmitter, which means there is no requirement for local regeneration of the m-sequence at the receiver in order to produce pulse compression. Therefore this method of sounding can be considered asynchronous and does not require rather expensive and complex timing references. The system operates in real time, producing a series of channel response snapshots at the output of the matched filter, producing an effective 1:1 mapping of time delays in the time domain.

The disadvantages of this method of channel sounding are numerous. Due to the high rate of incoming real time information, expensive equipment is required to record the data. In addition, either very large data archival devices or specialized circuits for reducing the recording bandwidth are needed to store the data. Variances in the manufacturing processes of SAW devices, difficulty implementing long sequences, and generation of spurious acoustic signals can often produce the unwanted presence of multiple reflections, bi-directional re-radiation and scattering of the surface acoustic waves, limiting the performance of the SAW devices within the system. This in turn can result in sidelobes appearing in the matched filter, limiting the system's sensitivity to weak echoes.

Figure 2.2.5 below provides a block diagram of a pulse compression system using a matched filter, where ω_0 is the center frequency, $I(t)$ is the inphase output, and $Q(t)$ is the quadrature output.



Excerpted from [10]

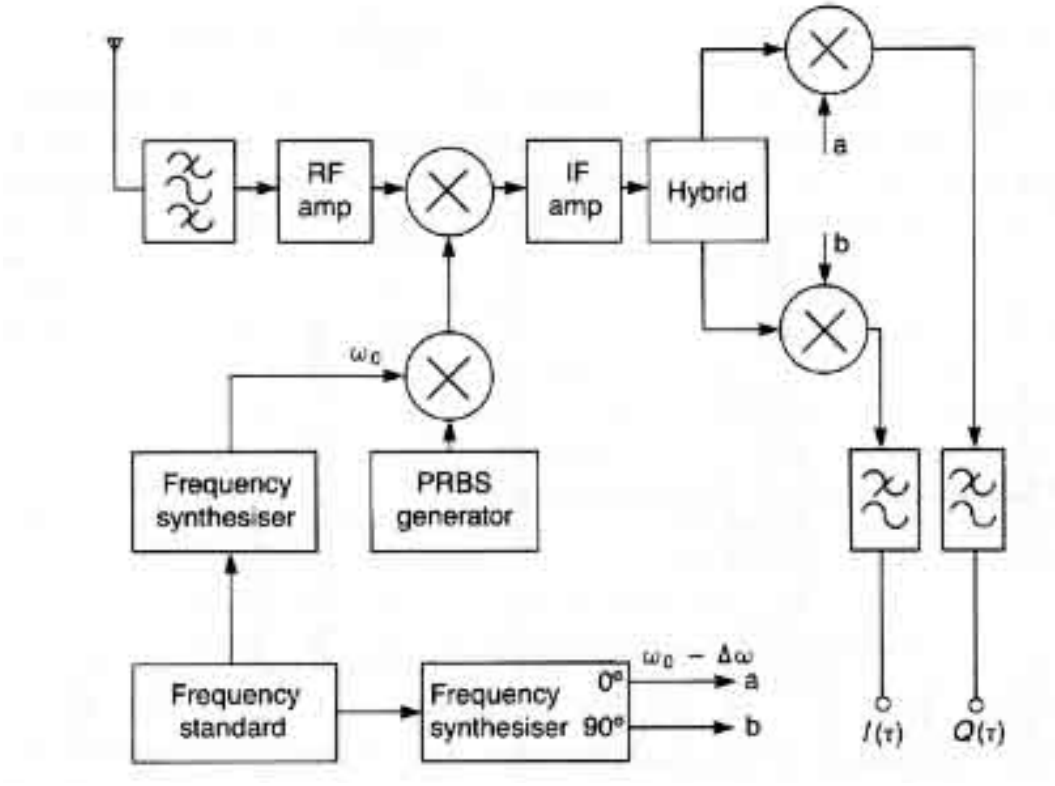
Figure 2.10: Block diagram of a pulse compression system using a matched filter.

Pulse compression by swept time delay cross correlation technique

Pulse compression sounders can also be based on correlation processing instead of convolutional processing. To implement real time correlation processing similar to the real time convolution processing previous described would require a bank of correlators with infinitesimally different time delay lags. This is unrealistic [10]. Instead, correlation processing is implemented using swept time delay techniques at the receiver which correlate an incoming signal with a copy of the m-sequence used in the transmitter, only clocked at a rate slightly slower than the transmit rate.

This results in time scaling, often called "time dilation", which reduces the data capture and archival needs of the system through the effective bandwidth compression that is achieved. The time scaling factor is determined by the difference in clocking rates at the transmitter and receiver. Such systems are called "swept time delay cross correlator (STDCC) sounders" or "spread spectrum sliding correlator sounders."

Figure 2.11 below provides a block diagram of a pulse compression system using cross-correlation, where ω_0 is the center frequency, $\Delta\omega$ is frequency offset, *I(t)* is the inphase output, and *Q(t)* is the quadrature output.



Excerpted from [10]

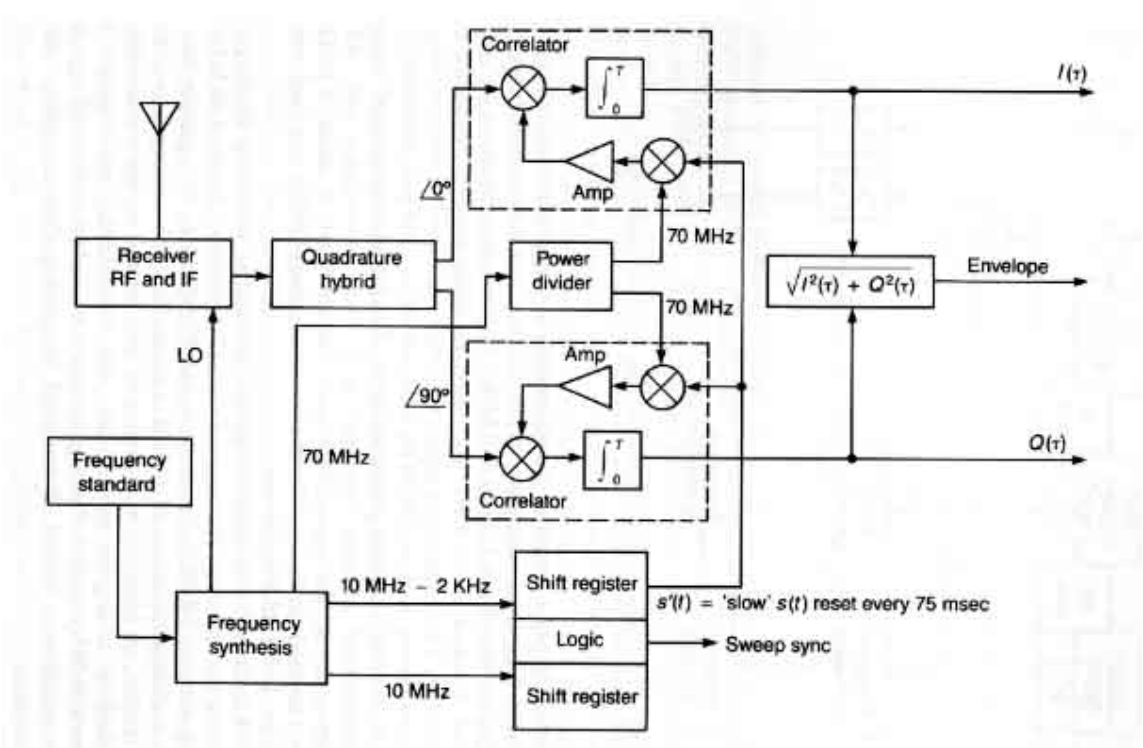
Figure 2.11: Block diagram of a pulse compression system using cross-correlation.

The STDCC sounder technique was first used by Cox to measure mobile channels in New York at 910 MHz [15]. A 511-bit m-sequence, clocked at 10 MHz, was used to phase reversal modulate a 70 MHz carrier. The modulated signal was then translated to the frequency band under study by mixing it with an 840 MHz local oscillator. This signal was then amplified and transmitted from an omnidirectional antenna at a fixed site with an average radiated power of 10 W. All frequencies were derived from a stable 5 MHz frequency standard.

The receiver amplified the received signal and then translated it down to the 70 MHz IF using an 840 MHz local oscillator. The 70 MHz IF was then split in a wideband quadrature hybrid, and applied to two corrrelators. In each correlator, a copy of the transmitted m-sequence clocked at a slightly slower rate (9.998 MHz) phase-reversal

modulated a 70 MHz carrier, which was then multiplied with the IF signal from the quadrature hybrid. A low pass integrating filter completed the cross-correlator. By demodulating the received signal in quadrature demodulators and using stable frequency and time standards, Cox was able to simultaneously measure time delays and extract Doppler shifts in a multipath mobile channel for the first time ever in a single channel sounder.

Figure 2.12 below provides a block diagram of Cox's STDCC sounder, where *Envelope* is the envelope of the received signal, *LO* is the local oscillator source, *T* is the chip period, *I(t)* is the inphase output, and *Q(t)* is the quadrature output.



Excerpted from [10]

Figure 2.12: Channel Sounder receiver as used by Cox.

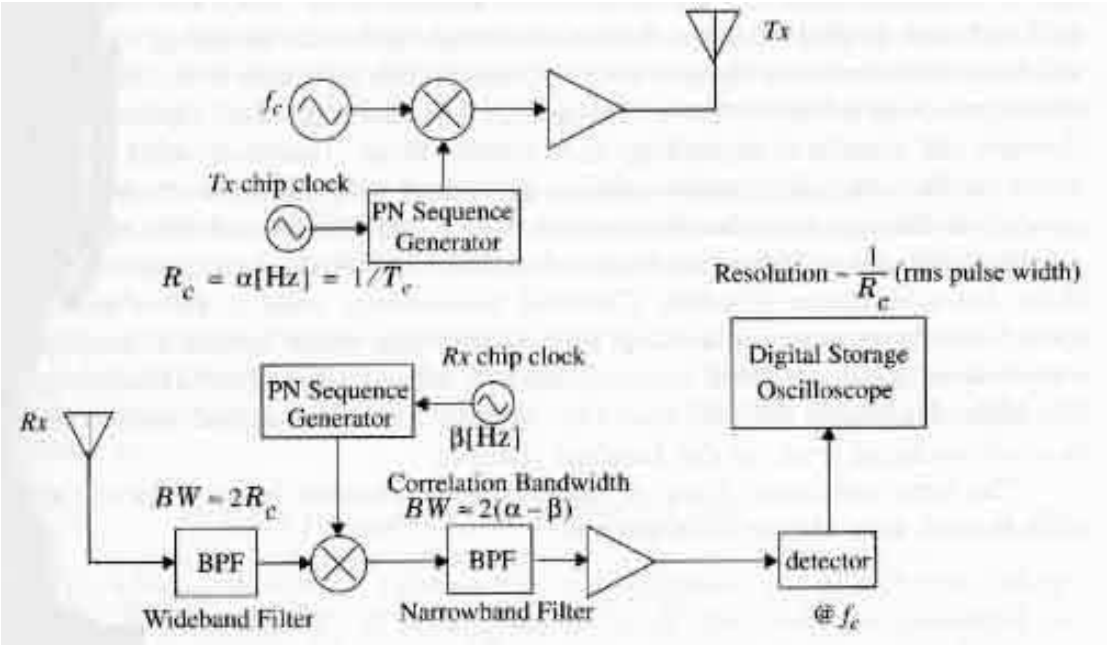
A number of further studies have been done using variants of Cox's system in the mobile radio [29], [30], [31],[32] and microwave [21] fields, often employing envelope detectors instead of quadrature demodulators because only the received envelope was being investigated.

A generic block diagram for a variant of the STDCC sounder called the "spread spectrum sliding correlator" is shown below in Figure 2.13 to better illustrate the system operation and "time dilation" processing. In this diagram, T_c is the chip duration, $R_c = 1/T_c$ is the chip rate, and f_c is the carrier frequency. The power spectrum envelope of the transmitted spread spectrum signal is given by [33] as

$$S(f) = \left[\frac{\sin(\pi(f - f_c)T_c}{\pi(f - f_c)T_c} \right]^2$$

And the null to null bandwidth is

$$BW = 2\,R_c$$



Excerpted from [6]

Figure 2.13: Spread Spectrum Channel impulse response measurement system.

The following equations are provided to shed light on the "time dilation" process inherent to this system. The system can resolve multipath components that are greater than $2\,T_c$ seconds apart, given the time resolution $(\Delta\tau)$ of multipath components for a spread spectrum system as

$$\Delta\tau = 2T_c = \frac{2}{R_c}$$

If α is the transmitter chip clock rate in Hz and β is the receiver chip clock rate in Hz, then the time dilation factor, or sliding factor, is

$$\gamma = \frac{\alpha}{\alpha - \beta}$$

For a maximum length PN sequence where *n* is the number of shift registers in the sequence generator [15], the sequence length is

$$L = 2^n - 1$$

Equivalent time measurements occur every time the two sequences are maximally correlated given by

$$\Delta T = T_c \gamma L = \frac{\gamma L}{R_c}$$

This means that the relative rate of the two codes slipping past each other, Cox's swept time delay processing, determines the rate of information that must be captured and stored. This allows narrowband processing of wideband signals, reducing the complexity and cost of sounder systems. This processing method is leveraged in the SSTDSP sounder.

The equivalent time measurements are related to the actual time delays of the multipath components by the time dilation factor γ and is given by

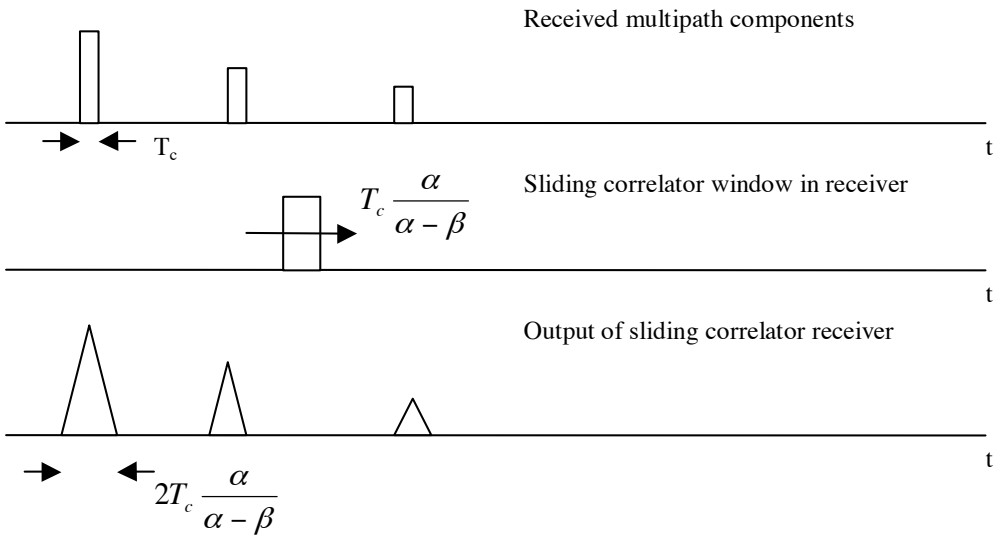
$$ActualPropagationTime = \frac{ObservedTime}{\gamma}$$

This system requires that the sequence length *(L)* must have a period (τ_{Pnseq}) greater than the longest multipath propagation delay, where

$$\tau_{Pnseq} = T_c L$$

By multiplying τ_{Pnseq} by the speed of light, the maximum unambiguous range of incoming multipath signal components can be calculated.

Figure 2.14 below illustrates the swept time delay (sliding correlation) process.



Excerpted from [11]

Figure 2.14: Swept time delay (sliding correlation) process.

The STDCC sounder method is robust in peak power limited situations because it is able to reject passband interference, providing significant processing gain in the spread spectrum receiver. When the PN code of the faster transmitted chip clock catches up with the PN code of the slower chip clock, the two sequences become aligned providing maximum correlation. However, when the two sequences are not maximally correlated, mixing the incoming signal with the unsynchronized receiver chip sequence will spread the signal into bandwidth at least as large as the receiver's PN sequence. This will allow the narrowband filter of bandwidth

$$BW = 2\,(\alpha - \beta)$$

that follows the correlator to reject most all of the incoming signal power and passband interference. This produces processing gain in the spread spectrum receiver, given by

$$PG = \frac{2R_c}{R_{bb}} = \frac{2\tau_{bb}}{T_c} = \frac{(S/N)_{out}}{(S/N)_{in}}$$

where $t_{bb} = 1/R_{bb}$ is the period of the baseband information. For the STDCC sounder R_{bb} is equal to the frequency offset of the PN sequence clocks at the transmitter and receiver:

$$R_{bb} = \alpha - \beta$$

The STDCC sounder's processing gain, significantly reduced processing and storage requirements, and ability to reject passband interference provide substantial performance gains over the periodic pulse sounding method.

However, the STDCC system design is not optimal for monitoring extremely wideband channels, such as LMDS. When small multipath resolution is needed to monitor bandwidths in excess of several hundred MHz, the resulting chip rates that are required by the STDCC system become difficult and expensive to implement. To implement 0.5 ns time bins necessary to resolve a wideband LMDS channel with a STDCC or "spread spectrum sliding correlator " sounder, a chipping rate of 4GHz is required. In addition, for the correlation process to work, the transmitter and receiver chip clocks must be precisely synchronized. Many systems use expensive highly stable Rubidium atomic clocks to meet this design requirement.

Ideally, one would like to combine the simplicity and inherent wideband capabilities of the periodic pulse sounder with the reduced processing and storage requirements of the STDCC sounder, while ensuring that the system was robust to wideband interference and did not require multi-GHz correlator circuits. The resulting system design should also be cost effective.

A New Approach: Sampling Swept Time Delay Short Pulse (SSTDSP) Sounder

The Sampling Swept Time Delay Short Pulse (SSTDSP) sounder originally proposed by Sweeney et. al. [11] and further developed in this thesis leverages the sampling swept time delay processing element of the STDCC sounder, however eliminates the need for the high speed chip clock by combining short pulses at the transmitter with precise sample gating at the receiver. This eliminates the interference problems associated with the video detector found in most periodic pulse sounders and ensures that the dynamic range of the system is limited only by the sample gate isolation. The resulting Sampling Swept Time Delay Short Pulse (SSTDSP) sounder design is significantly simpler than the STDCC system, higher performance, and less expensive.

The SSTDSP sounder system is shown below in Figure 2.15.

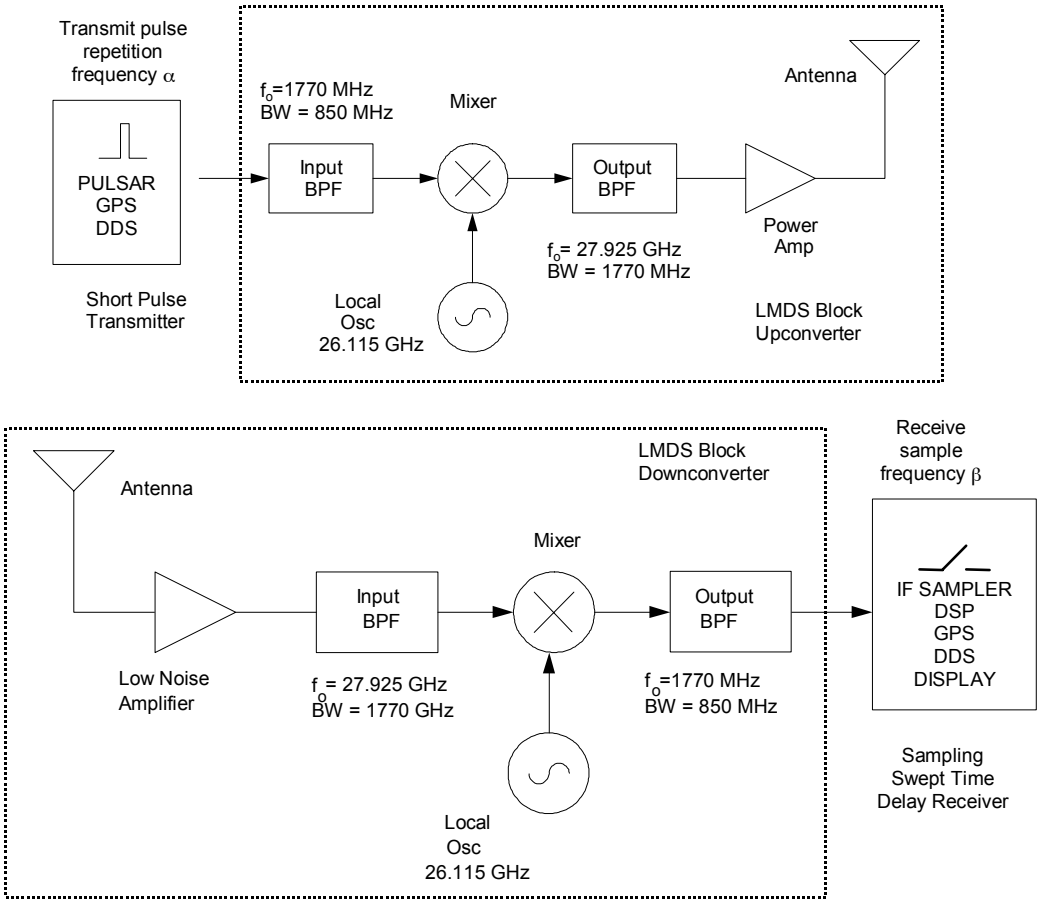


Figure 2.15: SSTDSP Wireless Channel Sounder System

The SSTDSP sounder uses direct digital frequency synthesis (DDS) driven by a cost effective stable 10 MHz global position system (GPS) disciplined frequency and location standard to derive the transmit pulse repetition frequency (prf) α and receive sampling frequency β. The SSTDSP transmitter generates a short pulse of duration τ_{bb} and pulse repetition frequency of α, given a bandwidth $BW=2/\tau_{bb}$ to be characterized and unambiguous range in meters of incoming multipath signal components of

$$UnambiguousRange = \frac{1}{\alpha}c = \frac{c}{\alpha}$$

where c is the speed of light ($2.99*10^8$ m/s).

To characterize the multipath behavior of links that utilize the 850 MHz wide LMDS Block A bandwidth and have an unambiguous range of incoming multipath signal components of 299 m, τ_{bb} will be no more than 2.35 ns and α can be set to 1MHz. The resulting 1 MHz pulse train is then transmitted over the LMDS channel. The SSTDSP receiver sees the envelope of the periodic impulse train convolved with the response of the channel, which produces a periodic waveform which contains the envelope of the response of the LMDS channel each period, as explained in earlier sections on periodic pulse sounders.

The SSTDSP receiver utilizes swept time delay processing to reduce the digital sample and processing rate needed to record and reconstruct the envelope of the channel response, while maintaining the effective Nyquist sampling rate needed to properly detect the short pulses. By sliding across the pulse repetition window $T_c = 1/\alpha$ in small time increments or bins of size $(\Delta\tau)$ that are no more than half the pulse width τ_{bb} by Nyquist's theory

$$\Delta\tau = \frac{T_c}{\gamma} <<< \frac{\tau_{bb}}{2} \qquad where,\ \gamma = \frac{\alpha}{\alpha - \beta}\ is\ the\ "time\ dilation\ factor"$$

the processing and data archival rate requirements needed to capture the channel response are reduced from around a 1 Gsps to 1 Msps, allowing use of low cost analog to digital (A/D) converters to digitize the data. This method of swept time delay processing reduces the sampling rate of the A/D converters, however requires a very fast sample and hold to precede the low rate A/D converters. For extremely short pulses, an IF sample and hold circuit directly samples the downconverted band pass channel response at the IF output of the LMDS radio and then holds the value for the A/D to digitize, store, and process. A digital signal processor (DSP) then takes the digitized sample and stores it in an SRAM memory buffer of length $N=\gamma$.

After N passes of the A/D, the entire channel response contained in the period T_c has been digitally captured and stored. The envelope of this channel response is then interpolated to recover the impulse response of the channel, and then scaled and ordered to produce

the power delay profile. The power delay profile is then utilized to calculate the channel metrics of mean excess delay, rms delay spread, maximum excess delay given a multipath threshold, and the coherence bandwidth. The total time (T_{tot}) it takes to complete the measurement is

$$T_{tot} = N T_c$$

For a pulse repetition frequency α=1 MHz and time bin size of 0.25 ns, the total time to take the SSTDSP measurement is 4 ms and the size of the power delay profile is 4000x24 bit words long, or about 12 kbytes. By setting α=1 MHz, a maximum excess delay of 1 us and an unambiguous range of incoming multipath signal components of 299 meters can be resolved, which should be sufficient for LMDS channel measurements. Given that S_{bits} is the size in bits of one word, the SSTDSP data logging rate (R_{data}) in bits per second (bps) to the host processor is equal to

$$R_{data} = (N S_{bits}) / T_{tot} = S_{bits} \alpha$$

This means that the SSTDSP sounder can run in the field in real time, rendering wideband multipath channel measurements with multipath component resolution of less than a foot. This information will allow researchers to determine the best network topology for rapid deployment of LMDS nodes and if there is a "bounce" path available to the network designer. The SSTDSP sounder has been designed in a modular fashion, which permits rapid reconfiguration in the field through software and programmable hardware to allow system parameters to be changed to permit a wide variety of wireless channel sounding experiments.

The following chapter discusses the design and implementation of the first prototype SSTDSP sounder test platform.

Chapter 3

Design and Implementation

3.1 System requirements

The SSTDSP sounder described in Chapter 2 was implemented with a number of system requirements in mind. Since changes in one aspect of the system design often affect the performance of another, a number of design tradeoffs were made. The system parameters and requirement tradeoffs are discussed in detail below. A brief review of computations required to calculate the channel metrics is also provided for reference.

3.1.1 Review of system parameters

The following system parameters were developed in Chapter 2 and are summarized below for reference. Given the ***speed of light (c=2.99*10^8 m/s)*** an ***unambiguous range of incoming multipath signal components (RANGE)*** in meters of the links to be sounded and a ***characterization bandwidth (BW)*** in Hz to be characterized, the ***short pulse duration (τ_{bb})*** in seconds is:

$$\tau_{bb} = \frac{2}{BW}$$

and the ***transmitted pulse repetition frequency (α)*** in Hz is

$$\alpha \leq \frac{c}{RANGE}.$$

The ***pulse repetition window (T_c)*** in seconds is

$$T_c = 1/\alpha$$

and the required Nyquist ***time bin size (Δτ)*** in seconds is

$$\Delta\tau \le \frac{\tau_{bb}}{2}.$$

Given the time bin size *(Δτ)*, the ***time dilation factor (γ)*** can be found

$$\gamma = \frac{T_c}{\Delta\tau} \qquad where,\ \gamma = \frac{\alpha}{\alpha - \beta}\ is\ the\ "time\ dilation\ factor".$$

From the time dilation factor *(γ)*, the ***receiver ADC sample frequency (β)*** in Hz is found to be

$$\beta = \alpha(1 - \frac{1}{\gamma})$$

The length of the ***DSP SRAM memory buffer (N)*** in words is equal to the time dilation factor *(γ)*

$$N = \gamma$$

And the ***total time for SSTDSP measurement (T_{tot})*** in seconds is equal to

$$T_{tot} = N\,T_c$$

Given that S_{bits} is the size in bits of one word, the ***SSTDSP real time data logging rate (R_{data})*** in bits per second (bps) to the host processor is equal to

$$R_{data} = (N\,S_{bits}) / T_{tot} = S_{bits}\,\alpha$$

The number of memory words on the DSP *(N)* available for data buffering, the sampling rate of the ADCs *(β)*, and the data transfer rate to the host processor *(R_{data})* are the three parameters that limit the performance of the SSTDSP sounder. The time dilation factor directly affects the memory buffer size, which is highly dependent on the transmitted pulse repetition frequency and the required time bin size. The required time bin size is a function of the pulse width, which is a function of the bandwidth of spectrum to characterized by channel sounding.

The dynamic range of the system determines the difference in amplitude of the multipath components that can be measured. The SSTDSP sounder scales all incoming pulses relative to the maximum pulse in the power delay profile. The sounder's digital processing algorithms assume that the strongest pulse is the first one, since the LMDS channel is typically line of sight (LOS) or a possibly a non-LOS bounce path. The

original pulse output power, transmit antenna gain, path loss, receiver antenna gain, and receiver gain will dictate how much gain is needed after the IF sample and hold circuit to ensure the peak pulse voltage is mapped to +2 volts on the ADC. As indicated in the link budget in the next section, around 25 dB of gain could be required. The dynamic range of the ADC found in the RX DSP Module sets the maximum dynamic range that can be recorded by the SSTDSP sounder digital hardware. The ADC's dynamic range is governed by the number of bits available to the converter, quantization noise, and nonlinearities.

As a rule of thumb, for each bit of resolution in an ADC, you gain 6 dB of dynamic range. So the 14 bit ADCs used in the SSTDSP sounder will be capable of detecting a maximum of 84 dB of dynamic range. However, since the ADCs are designed for a +-2.5 volt swing, the sign bit reduces this effective resolution to a maximum dynamic range of 13 bits or 78 dB. Realistically, due to quantization noise nonlinearities and the ambient noise floor, this dynamic range will be even less. The SSTDSP IF Sampler only utilizes a +-2V swing of the DSP, which reduces the effective bits to 12 bits, or 72 dB dynamic range, since the number of bits (BITS) to represent 0 to +2 volts given .3mV increments is

$$BITS = \log_2(2/(2.5/2\text{^}13)) = 12.7 \sim 12 \text{ bits, or } 12*6=72 \text{ dB}$$

This means that the ADC will have approximately 72 dB of dynamic range to record the incoming pulse. Realistically, as shown by the link budget in the next section, the maximum dynamic range in the SSTDSP sounder will be influenced by the signal to noise ratio *(SNR)*, which is 25.5 dB for the particular system gain settings provided in the next section. The system gain settings can be varied somewhat to adjust the SSTDSP sounder's *SNR*. A *SNR* of 25.5 dB means that pulses lower than 25.5 dB down from the peak pulse will appear in the noise floor. Since the ADC has a resolution of approximately 6 dB of dynamic range per bit, the ADC will be only utilizing approximately 2 bits to differentiate between the peak pulse value and the receiver noise floor. This has implications for the multipath threshold. The maximum dynamic range of the SSTDSP sounder will be equal to the *Carrier to Noise Ratio (C/N)*, with the minimum multipath threshold level 25.5 dB down from the peak pulse.

The maximum dynamic range of the SSTDSP system is governed by the noise in the receiver. If there is sufficient gain prior to the RX IF Sampler Module, ensuring it is using the maximum dynamic input range of the ADC, the noise figure *(NF)* of the receiver will be set by the Low Noise Amplifier (LNA) found in the RX LMDS Radio Module.

The receiver gain is distributed in several places in the RX LMDS Radio Module and RX IF Sampler, including gain in the LMDS downconverter, gain in the IF circuitry, gain/loss in the IF sampler, and an adjustable gain post IF Sampler. The gain block following the IF Sampler is controlled so that the largest pulse in the power delay profile is normalized to the maximum +2 V input of the ADC. In addition, the sampling efficiency of the IF Sampler may affect the receiver gain, since the system can be lossy because the amount of energy transferred during the IF sample is very small. Given the channel sounding bandwidth *(BW)*, the short pulse duration *(τ_{bb})* can be calculated to be

$$\tau_{bb} = \frac{2}{BW}, \; where \; BW = \frac{2}{\tau_{bb}}$$

therefore the effective bandwidth *(BW_{eff})* at the intermediate frequency (IF*)* due to the time dilation processing is

$$\gamma\tau_{bb} = \frac{2}{BW_{eff}}$$

$$BW_{eff} = \frac{2}{\gamma\tau_{bb}} = \frac{BW}{\gamma}$$

which is a fraction of the full channel sounding bandwidth. Due to the time dilation process, the maximum effective system bandwidth is the transmit pulse repetition frequency, which for the SSTDSP system is at most 1 MHz. Therefore the noise power output of the RX IF Sampler Module in dB is defined to be

$$P_N = 10\log(kTB) + A + NF$$

where *NF* is the noise figure of the LMDS LNA in dB, *k* is the Boltzmann constant which equals 1.38×10^{-23} W/K/Hz, *T* is the effective noise temperature of the receiver system in

Kelvin (K), B is the double sided receiver bandwidth in Hz, and A is the gain of the specific system being evaluated in dB.

The accuracy of the frequency standards at the transmitter and receiver directly affects the accuracy of the SSTDSP system measurements. Highly stable frequency and time standards are required to allow the swept time processing to occur properly. Any small frequency difference between standards found in the transmitter and receiver drift causes a slow drift between the transmitter and receiver systems. This phenomenum results in a relative shift in timing such that echoes with the same path delay no longer occupy the same time resolution cell. The maximum time of field trial operation (T_{max}) before resynchronization of the sounder is required is a function of how long it takes the a standard to drift one time resolution bin $(\Delta\tau)$. The frequency difference between the transmit and receive standards must be of the order of $(\Delta\tau) / T_{max}$ over a T_{max} period. For the current SSTDSP setup, if $\Delta\tau = 0.25$ns and $T_{max} = 4$ ms, this frequency difference must be on the order of 6.25×10^{-8} and the stability has to be good enough to maintain this difference over the 4 ms period. The SSTDSP sounder achieves this by resynchronizing an oven oscillator to a highly stable and accurate GPS time reference every second through use of a digital phase lock loop. This novel global position system (GPS) disciplined frequency synthesis module was implemented at a fraction of the cost of existing stable frequency standards.

The multipath resolution of the SSTDSP system is a function of the short pulse duration (τ_{bb}). Given the speed of light $(c=2.99*10^8$ m/s$)$, the multipath resolution (MP_{res}) in meters of the SSTDSP sounder is

$$MP_{res} = \tau_{bb}\, c$$

Given a sounding bandwidth of 850 MHz for the LMDS Block A band, the current SSTDSP sounder has a multipath component path length difference resolution of about 0.703m. As mentioned in Chapter 2, if a building serving as the obstruction has height h = 50 m and is located at a distance d_1 = 2500 m from the transmitter and d_2 = 2500 m from the receiver, then the excess path length is 1 m, within the resolution of the SSTDSP sounder to permit single "bounce path" measurements.

3.1.2 Tradeoff analysis

The SSTDSP sounder implementation aims to balance several fundamental tradeoffs governed by the system parameters above. Given a specific spectrum bandwidth to sound and therefore a fixed pulse width and required time bin, a tradeoff between the unambiguous range of incoming multipath signal components, the amount of DSP memory, and the measurement time must be made. The larger the unambiguous range of incoming multipath signal components, the larger the memory buffer must be for a given sounding bandwidth, and the longer it will take to complete the measurement, if the entire pulse repetition period is swept. It is possible to circumvent this tradeoff by insuring that the SSTDSP sounder measurement is always initiated in conjunction with the original sounding pulse. This allows the sounder to capture the first portion of the pulse repetition period corresponding with the maximum excess delay given a certain multipath threshold, thereby reducing the memory buffer and measurement time needed.

A tradeoff also exists between the SSTDSP data logging rate *(R_{data})* and the transmitted pulse repetition frequency *(α)*, which determines the unambiguous range of incoming multipath signal components. The longer the unambiguous range of incoming multipath signal components, the smaller the pulse repetition frequency, which reduces the required data logging rate.

The specific tradeoffs of parameters for the current SSTDSP sounder were calculated based on the best fit for the LMDS channel monitoring application. A pulse width of 4ns was chosen, with a bin time size of 0.25ns. The unambiguous range of incoming multipath signal components of the system was determined to be no more than 299 meters for LMDS links with a maximum excess delay of no more than 1 us, so a transmitted pulse repetition frequency of 1 MHz was chosen. This required a time dilation factor of 4000 and resulting DSP SRAM memory buffer of 4000x24bit words. The required receiver sample frequency was 0.999750 MHz, which is within the tuning capabilities of the DDS. The total measurement time is 4 ms and the required real time data transfer speed is 24 Mbps which is within the range of the high speed DSP Host Port

Interface (HPI). During the measurement period, the channel must be stationary. The current sounder takes single impulse response snapshots and transfers them to the host computer for processing and display. However, given the I/O data rate above, real time streaming of data is possible if the RX Host Module has a peripheral interface fast enough to interface to the DSP HPI port.

3.1.3 Channel Metric calculation considerations

The computations required to calculate the various channel metrics from the power delay profile $P(\tau)$ are derived in Chapter 2. Due to computational limitations present in fixed point digital signal processors, the first generation of the SSTDSP sounder uses the ADC and DSP to capture and store the impulse response of the channel in SRAM buffer memory. This buffer is then transferred to a floating point host computer to render the power delay profile and channel metrics in MATLAB. Provided enough computing power is present in the MIPS (millions of instructions per second) budget for the DSP, these functions may eventually be transitioned fully to the DSP. For a transmit pulse repetition frequency α = 48 kHz, the current SSTDSP sounder DSP algorithm uses a fraction of the current MIPs budget. High precision algorithms that permit floating point accuracy on a fixed point DSP could therefore be applied in a future SSTDSP system upgrade. However, given a transmit pulse repetition frequency of α = 1 MHz, almost all of the DSP MIPs budget is used just to capture and store the channel data samples. Additional detail on how to calculate the sounder's DSP MIPs budget is provided in the RX DSP Module section.

The next section provides a discussion of the power considerations inherent to a short pulse sounding system.

3.2 Power considerations

Since the SSTDSP sounder depends on sending and detecting short pulses of energy that serve as probing impulses that are used to observe the impulse response of the channel, a

quick word should be said about the power considerations surrounding the generation of these pulses.

3.2.1 Peak and Average Power of a Short Pulse

A signal can be quantified by its peak power and its average power. Since the SSTDSP sounder utilizes short pulses, the peak power of the pulse train that is transmitted through the channel is much larger than the average power of the pulse train. The peak power of the pulse will also be larger than the average power of the pulse, but should be distinguished from the peak and average power of the pulse train. It is important to note that peak and average power values differ depending on the averaging period.

Given that a transmitted single cycle pulse with peak voltage *(V_{peak})*, root mean square (rms) voltage *(V_{rms})*, and load impedence *(R)*, the peak power of the single cycle pulse *(P_{peak})* over the short pulse duration is given by

$$P_{peak} = \left(\frac{V_{peak}}{\sqrt{2}} \right)^2 \frac{1}{R} = \frac{V_{rms}{}^2}{R}, \; since \; V_{rms} = \frac{V_{peak}}{\sqrt{2}}.$$

If a P_{peak} = 1 W over one pulse is desired and the antenna appears as R=50 Ω, then the maximum transmitted rms voltage of the pulse *(V_{rms})* should be $\sqrt{50}$ =7.07 volts.

The average power *(P_{avg})* of a transmitted short pulse of duration *(τ_{bb})* and pulse repetition period *(T_c)* is

$$P_{avg} = \frac{P_{peak} \tau_{bb}}{T_c}$$

This means that given a P_{peak} = 1 W, a short pulse duration of τ_{bb}=4ns, and a pulse repetition period of T_c=1us, the average power output P_{avg} = 4 mW. The SSTDSP sounder LMDS radio transmitter is limited in peak power output to 1 W; however, because of the short pulse duration, a power meter and other devices will see an average transmit power of 4 mW. Provided the devices are not transmitting at the instant the pulsar fires, they will see the low average power output of the sounder as minor interference.

3.2.2 Peak to average power ratio

The peak to average power ratio *($Ratio_{peaktoavg}$)* of the short pulse waveform is defined as

$$Ratio_{peaktoavg} = \frac{P_{peak}}{P_{avg}}.$$

Given the SSTDSP sounder parameters above, the peak to average ratio would be 250. The peak to average ratio is useful for providing a quick sense of the power characteristics of a given signaling waveform.

3.2.3 Link Budget

The SSTDSP sounder link budget is used to calibrate and estimate the sounder's performance, given an unambiguous range of incoming multipath signal components, sounding spectrum bandwidth, and numerous other operating system parameters. Chapter 2 provides additional details regarding link budget calculations.

Figure 3.1 and Table 3.1 below depict a SSTDSP sounder block diagram with appropriate labels and a sample LMDS link budget, given a 5 km transmitter and receiver separation, 1 W transmit power output, and 1 MHz receive noise bandwidth. The 1 MHz noise bandwidth is defined by the bandwidth of the incoming time dilated data from the RX IF Sampler Module. While the received short pulses may occupy up to 850 MHz of LMDS Block A spectrum, the noise bandwidth seen by the input of the post IF Sampler gain block is approximately the sounding bandwidth divided by the time dilation factor. The time dilation occurring in the time domain results in a bandwidth reduction in the frequency domain, effectively reducing the SSTDSP sounder noise bandwidth.

The link budget was calculated to determine the necessary IF Sampler gain needed to ensure that the incoming pulse train is scaled correctly to maximize the ADC dynamic range. For this example, the link budget indicates that the LMDS receiver will provide a peak amplitude of -27.3 dBm to the IF sampler. The link budget estimates 2 dB of loss

due the IF Sampler implementation, producing a peak power input into the IF Sampler gain block of -29.3 dBm. The IF Sampler gain block is then adjusted to 24.2 dB to scale the peak pulse voltage to the +-2 V level of the ADC.

Currently this is done in the SSTDSP sounder using manually adjustable gain; however, the gain can be automatically controlled by the digital processing backend. The DSP utilizes the ADC to capture incoming samples and then uses a circular matched filter to determine the value and time bin of the maximum pulse within the impulse response memory buffer. This value can be scaled to automatically control a digital to analog (D/A) converter, which sets the gain level to ensure the maximum pulse within the pulse repetition period corresponds to the +-2 V quantization level within the ADC.

The link budget also indicates that the Signal to Noise Ratio (SNR) entering the ADC is 25.5 dB. The SNR entering the ADC ultimately limits the maximum dynamic range of the SSTDSP system, which in this case is 25.5 dB.

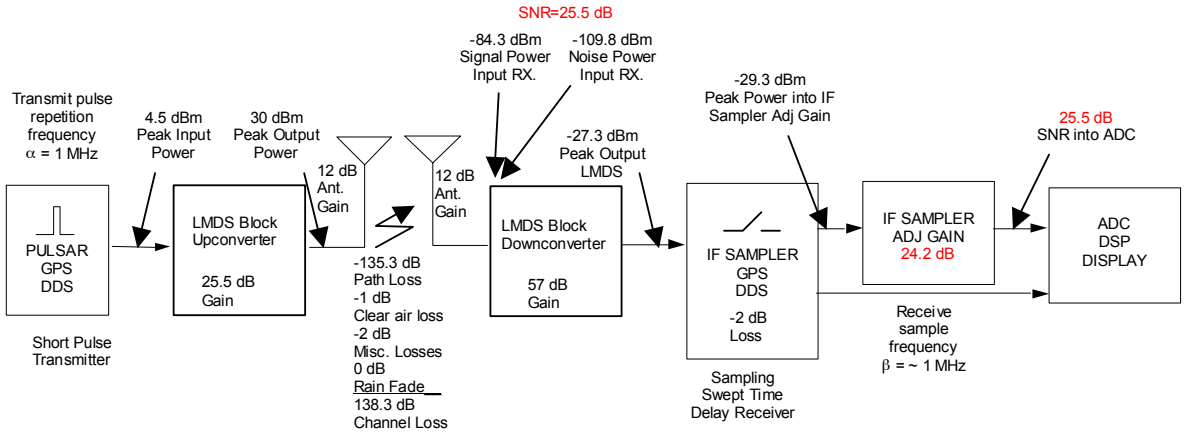


Figure 3.1: Block Diagram for Sample SSTDSP Sounder Link Budget for LMDS.

Table 3.1: Sample SSTDSP Sounder Link Budget for LMDS.

LINK BUDGET FOR SSTDSP SOUNDER AT LMDS

Total available LMDS bandwidth	850.00 MHz
Carrier Frequency	27.92500 GHz
Wave length	0.01074 meter

Power Budget		
Pulse amplitude	0.3754 V	4.5 dBm
Peak transmitted power, Pt	1 Watt	30 dBm
Tx Antenna Gain, Gt		12.0 dB
Path Loss, Lp	5000.0 meter	-135.3 dB
Clear air atm. Loss		-2.0 dB
Misc. Losses		-1.0 dB
Rain fade		0.0 dB
Receiver antenna gain, Gr		12 dB
Signal Power at input of RX		**-84.3 dBm**
Noise Budget		
Boltzmann Const		-198.6 dBm/K/Hz
Noise Temp.	750 Kelvin	28.8 dBK
Noise bandwidth input of IF Sampler	1.00 MHz	60.0 dBHz
Noise Power at input of RX		**-109.8 dBm**
Signal to Noise Ratio (SNR) for A/D input		25.5 dB

IF Sampler Gain Calculation	
Power at Input of RX antenna	-96.3 dBm
Rx Antenna Gain, Gr	12.0 dB
Rx Downconverter Gain	20.0 dB
RX LMDS IF circuit Gain	37.0 dB
LMDS Receiver Output Power	**-27.3 dBm**

Loss due to IF sampler		-2.0 dB
Peak Power into IF Sampler gain block		**-29.3 dBm**
Peak voltage into IF sampler gain block	50 ohms	0.0076 V
Receive voltage required at A/D		2.0 V
Required IF Sampler gain		24.2 dB

The following section provides a system overview of the SSTDSP sounder testing platform.

3.3 System overview

The SSTDSP sounder system consists of a transmitter and receiver, each containing several modules. The SSTDSP sounder was designed using systems engineering methods and a modular methodology to allow rapid reconfiguration and upgrades and provide implementation flexibility. Each module appears as a "black box" to the other modules, providing specific interfaces and functionality that is independent of the specific implementation. This permits a number of different vendor hardware platforms to be used, provided they meet the interface and functionality specifications.

3.3.1 List of SSTDSP sounder modules and system block diagram

The transmit side of the SSTDSP sounder consists of five modules:

1. TX Host Control Module
2. TX Frequency and Location Module
3. TX Pulsar Module
4. TX LMDS Radio Module
5. TX Power Module

The receive side of the SSTDSP sounder consists of six modules

1. RX LMDS Radio Module
2. RX IF Sampler Module
3. RX DSP Module
4. RX Frequency and Location Module
5. RX Host Module
6. RX Power Module

Figure 3.2 below provides a system block diagram of the SSTDSP sounder and illustrates the interfaces between each of the modules.

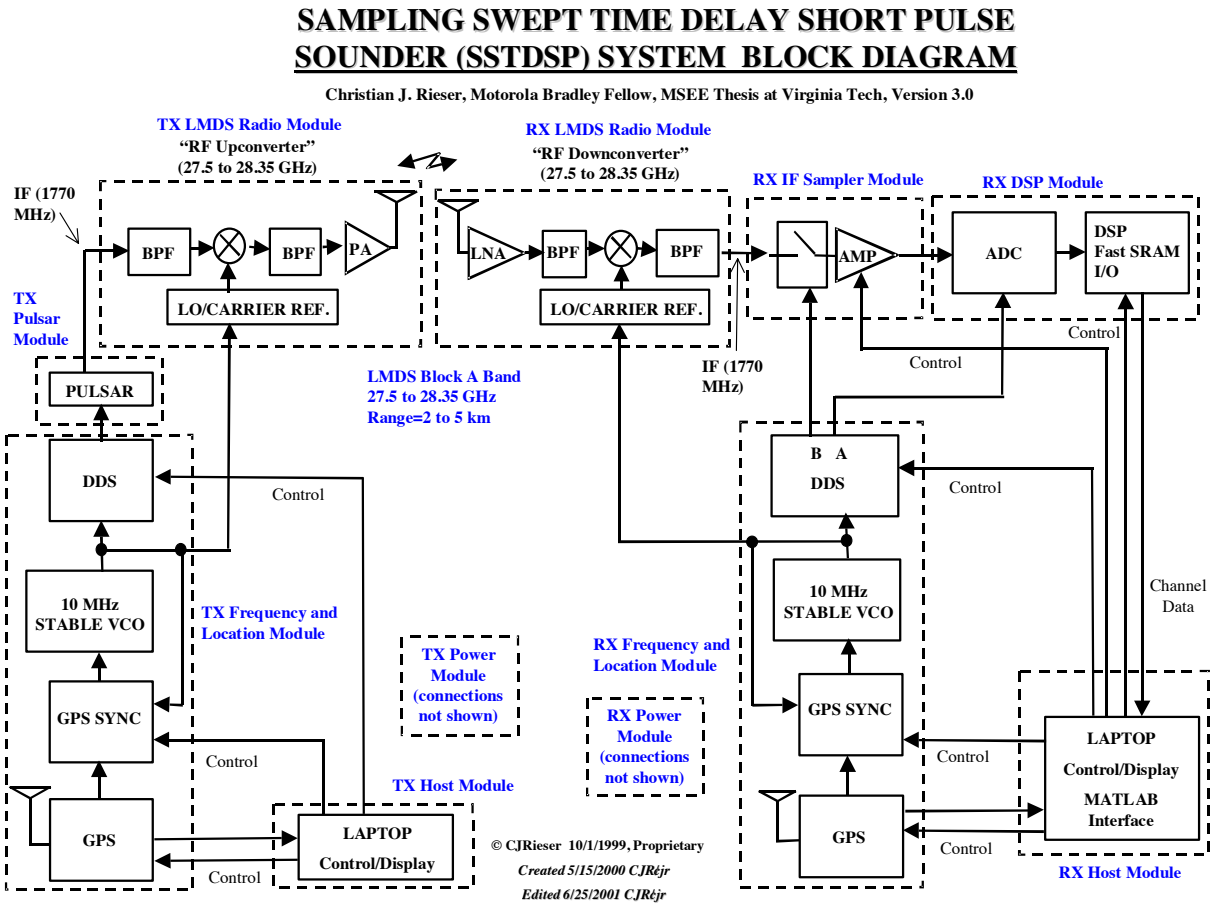


Figure 3.2: SSTDSP sounder system block diagram

3.3.2 List of SSTDSP sounder algorithms and scripts

A number of custom digital processing algorithms were created for the SSTDSP sounder. These algorithms are listed below.

1. **"processdata.m"**
2. **"startup.m"**
3. **"sounderparameters.m"**
4. **"soundersettings.m"**
5. **"AtoD1.asm"**
6. **"make.cmd"**
7. **"capturedata.cmd"**
8. **"exportdata.cmd"**
9. **"sounderdataconverter.exe"**
10. **"sounderdataprocess.m"**

These algorithms were tested in an incremental fashion by verifying that given a known input to the algorithm, the expected output was obtained. This greatly simplified debugging the software code. Most computational algorithms were prototyped in MATLAB first and the transitioned to C and DSP assembly language as needed. Off the shelf software was used to program and control the direct digital frequency synthesis (DDS) board, global position system (GPS) board, and digital signal processing (DSP) board.

Numerous custom interface scripts were written to automate the 9 steps of the SSTDSP channel sounding procedure. These scripts are listed below. Steps taken at the transmitter are preceded by TX and steps taken at the receiver are preceded by RX.

1. **"TX_STEP1_setdds.cmd"**
2. **"TX_STEP2_startgps.cmd"**
3. **"RX_STEP1_configuresounder.cmd"**
4. **"RX_STEP2_editassembly.cmd"**

5. **"RX_STEP3_editdspcontrol.cmd"**
6. **"RX_STEP4_SETSWITCHTOA_setdds.cmd"**
7. **"RX_STEP5_SETSWITCHTOB_setdds.cmd"**
8. **"RX_STEP6_startgps.cmd"**
9. **"RX_STEP7_SETSWITCHTOC_aquireconvertdata.cmd"**
10. **"RX_STEP8_PRESSSPACEBARINMATLAB_processdata.cmd"**

Source code for both the algorithms and scripts can be found in Appendix C.

3.3.3 Functional description of SSTDSP sounder hardware

At a high level, the sounder functions as follows. On both the transmit and receive sides of the SSTDSP sounder, the TX and RX Frequency and Location Modules use a highly stable reference signal derived from the Global Position System (GPS). This means that the transmitter and receiver clocks are effectively derived from the same, highly stable clock. Clock stability is needed between the transmitter and receiver in order for the SSTDSP channel sounding method to work since this process depends on precise offsets in transmit and receive frequencies to produce the necessary time dilation. The TX and RX Frequency and Location Module design acheives this at a low cost while also providing a precise global position location of the transmitter and receiver. This location information is then utilized in Geographic Information Systems (GIS) applications to compute LMDS coverage regions, given an LMDS link topology. The SSTDSP sounder software has been configured for LMDS channel measurements and therefore assumes that the strongest pulse in the transmit pulse repetition period is the first pulse. This represents either a line-of-sight (LOS) or a non-LOS bounce path.

The TX Frequency and Location Module provides a transmit pulse repetition frequency (prf) of α which is used to drive the Pulsar Module. The Pulsar Module is configured to produce the necessary short pulse needed to sound the spectrum bandwidth (BW) under investigation. The duration of the short pulse over the wireless channel is controlled by

the impulse response of the Band Pass Filter (BPF) found in the TX and RX LMDS Radio Modules which have been tailored for the LMDS frequency band under study. The Pulsar Module outputs a short pulse of duration $\tau_{bb} = 2 / BW$ every pulse repetition period $T_c = 1/\alpha$. This pulse train is then upconverted to the LMDS frequency band by the TX LMDS Radio Module. The pulse train propagates through the LMDS channel and is received at varying time offsets at the SSTDSP sounder receiver, depending on the multipath profile of the given LMDS channel.

On the SSTDSP sounder receiver side, the RX LMDS Radio Module downconverts the received band pass LMDS signal. As described in Chapter 2, the envelope of this band pass signal is effectively the impulse response of the channel, since the sounding short pulse is treated as an impulse. The envelope of the received IF band pass signal can be treated as the transmitted periodic short pulse train convolved with the response of the LMDS channel. The RX IF Sampler Module samples this incoming waveform with subnanosecond accuracy at a sample rate β, derived from the RX Frequency and Location Module, and then holds the resulting value of that time bin for long enough for the DSP's ADC to sample the value. Shortly after the RX IF Sampler captures this value, the value is sampled and converted to a digital value by the ADC running at the same sample rate β within the RX DSP Module. This is done because the ADC sample window is not small enough to resolve the subnanosecond time bins needed to capture the short pulse. The sample instant of the IF Sampler and ADC occur at the same sample frequency β, but are offset in phase, or time, to ensure the ADC captures the correct amplitude that was sampled and held by the very fast IF Sampler. The DSP within the RX DSP Module then buffers this value to memory. This process is continued N times, where

$$N = \gamma = \frac{\alpha}{\alpha - \beta}$$

until the entire channel response is buffered into memory. The entire channel response must be buffered into memory because the current SSTDSP sounder implementation has no way of determining when the original short pulse was received. Therefore, the DSP buffers all of the data for a transmit pulse repetition period and calculates the envelope of the band pass signal to recover the impulse response. The sounder algorithms then sorts

and normalizes the data so that the largest pulse is reordered and placed in the first time impulse response time bin. This assumption is made since the channel is a LOS or non-LOS bounce path LMDS channel. If the SSTDSP sounder measurement was initiated with the reception of the original transmitted short pulse, then the DSP would only need to capture the multipath components present between the original transmitted short pulse and the maximum excess delay, instead of the entire transmit pulse repetition period. Such measurement initiation synchronization is not currently implemented in the SSTDSP sounder.

The DSP has the option of computing the power delay profile and channel metrics in parallel with data capture, after the entire impulse response buffer is full, or it can send the impulse response buffer to the RX Host Module where the power delay profile and channel metrics are computed. The current implementation of the SSTDSP sounder RX DSP Module uses the last option utilizing the MATLAB Interface within the RX Host Module to process the impulse response and produce the power delay profile and channel metrics.

Regardless of the process by which they are computed, the power delay profile and channel metrics are provided to the RX Host Module every time the SSTDSP measurement process is complete, or every $T_{tot} = N T_c$ seconds. This information can then be used by external devices that are affected by the LMDS channel, such as modems, to tailor their configuration to the current channel conditions, including optimizing coding and network protocol settings. In addition to providing control functions to the RX DSP Module and RX Frequency and Location Module, the RX Host Module also provides feedback control to the RX IF Sampler amplifier gain block to ensure that the peak pulse amplitude within the impulse response corresponds to the peak value of the ADC voltage swing. This feedback optimizes the dynamic range of the ADC.

All power to the transmitter hardware is provided by TX Power Module and all power to the receiver is provided by the RX Power Module.

Specific implementation details of the SSTDSP software algorithms and hardware modules are described in the next two sections.

3.4 Module details

The following section provides addition implementation details about each of the SSTDSP modules. Each section provides the following information about the particular module and the specific implementation realized in this thesis:

1. System function
2. Implementation and design tradeoffs

Appendix A details the SSTDSP sounder module interfaces, operation, and key parameters. Photos of each module can be found in Appendix C.

3.4.1 TX Host Control Module

System function

The TX Host Control Module provides control signals to program the transmit DDS, GPS Sync board, and GPS boards within the TX Frequency and Location Module. It also receives data from the GPS unit within the TX Frequency and Location Module and displays the GPS receiver data, including location, acquired satellite signals, heading, status of GPS receiver, and time.

Implementation and design tradeoffs

The design process for implementing the TX Host Module involved finding a Windows 95 or later laptop that had a DB9 serial port, DB25 parallel port, CDROM for installing software, and 5 MB of hard drive space to install the two required software programs. A high performance computer was not needed at the transmitter, because minimal control and processing was occurring at the transmitter.

The TX Host Module was implemented by the author using a Toshiba laptop provided by CWT. This laptop was configured with a Pentium processor at 133 MHz with 128 kB of cache, 32 MB of RAM, 2 MB of video RAM, and a 2 GB hard drive. The following software packages were installed on the laptop, as shown in Table 3.2.

Table 3.2: TX Host Module software.

Company	Software	Version	Function
Analog Devices	AD9852 Customer Evaluation Software	1.7.1	DDS Control
Apollocom	OutlookPlus GPS	1.0 Build 8	GPS Control
Hilgraeve	HyperTerminal	1999	GPS Sync Control

These software packages were used to control various transmitter hardware blocks, as indicated in Table 3.2.

3.4.2 TX and RX Frequency and Location Modules

System function

The TX Frequency and Location Module and RX Frequency and Location Module implementations are virtually identical and provide a 10 MHz temperature controlled frequency standard locked to a highly accurate stable global position system (GPS) reference signal. The GPS disciplined processing ensures stability between the frequency standards in the TX and RX Frequency and Location Modules. The transmit frequency standard is utilized by the direct digital frequency synthesis (DDS) function within the TX Frequency and Location Module to derive the necessary pulse repetition frequency α. Similarly, the DDS present in the RX Frequency and Location Module produces the required sample frequency β that is required for successful time dilation in the receiver required by the SSTDSP channel sounding method.

Implementation and design tradeoffs

The design process for implementing the TX Frequency and Location Module involved implementing the four functional blocks listed above. A primary goal was to implement this module with operational flexibility and at a fraction of the cost of commercially available GPS referenced frequency standards. This was achieved by utilizing low cost off the shelf components and innovative implementation techniques, resulting in an implementation that cost several hundred dollars instead of several thousand dollars. Table 3.3 below indicates the hardware solution used by the author to implement each block.

Table 3.3: TX Frequency and Location Module hardware.

Block	Company	Hardware	Function
GPS	Motorola	OnCore GPS UT+	Produce 1 PPS from GPS and location information for GIS
GPS Control	A&A Engineering	GPS Sync Board	Generate VCO Control Voltage
10 MHz Ref	HP	10 MHz VCO	Generate 10MHz reference
DDS	Analog Devices	AD9852 Eval Kit	Generate transmit frequency α and receive frequency β and β_δ

A GPS card with precise timing capabilities was needed that produced both the highly stable 1 PPS second GPS reference (10 ns jitter or less) and location information. The Motorola Oncore UT+ was chosen because of its sub 10 ns accuracy without Selective Availability and in Position Hold (stationary location) mode. Selectivity Availability (SA) was an intentional degradation by the US government of the timing and location signals seen by commercial GPS receivers. SA introduces jitter into the GPS signals, thereby preventing high resolution timing and location without proper government equipment and codes. Recently the US government turned off SA, thereby significantly increasing GPS receiver accuracy.

The OutlookGPSPlus commercial software was recommended by Motorola to permit control and monitoring of the GPS OnCore UT+ hardware platform operation. It provides a graphical user interface and data logging capabilities. A 10 MHz temperature controlled

ovenized voltage controlled oscillator (VCXO) from HP was selected to be the highly stable frequency standard. The 10 MHz frequency was chosen because it is a common reference frequency and could be used as a reference in both the DDS circuit and the 960 MHz carrier reference in the LMDS radios.

The VCXO was disciplined to the highly stable GPS atomic clock reference signal by a novel low cost digital phase lock loop GPS Sync circuit published by Brooks Shera [35]. This circuit phase locks the 10 MHz VCXO to the highly stable 1 PPS GPS reference signal by using a digital phase lock loop phase difference accumulator that calculates the phase difference between the 1 PPS signal and a divided down frequency derived from the 10 MHz reference. The resulting phase count difference is calculated by a microprocessor, converted to an analog VCXO control voltage by a digital to analog converter (DAC), and used as a control voltage input to the 10 MHz VCXO to keep the VCXO phase locked to the GPS atomic clocks. Detailed circuit diagrams for the GPS Sync board implementation are published at [36].

The 10 MHz VCXO output is fed to the DDS to produce the required transmit frequency α and receive sample frequencies β and β_δ. An Analog Devices AD9852 CMOS 300 MHz Complete DDS evaluation board was used in the transmitter. Two AD9852 CMOS 300 MHz Complete DDS evaluation boards were used in the receiver to provide the phase offset A and B frequency channels needed to ensure the proper sample instant timing relationship between the IF Sampler and ADC in the DSP Module. This relationship mandates that the IF Sampler sample and hold its value prior to the ADC digitization of the value. The ADC sample window is 100 ns, which means that the IF Sampler must sample prior to the conversion start of the ADC and then hold that value for at least the 100 ns sample window of the ADC.

The AD9852 DDS chip was chosen because of its microHertz tuning capababilities. The time dilation within the SSTDSP sounding method is limited by the resolvable frequency tuning resolution difference, $\Delta f = \alpha - \beta$. The AD9852 provides down to microHertz Δf

resolution. This is more than adequate for the pulse repetition period and time bin resolution required for most LMDS channels.

For a pulse repetition frequency α=1 MHz and time bin resolution of $\Delta\tau$=0.25 ns, the required time dilation is γ=4000, requiring a receive sample frequency of β=.999750 MHz. A 250 Hz difference between α and β must exist. This requirement is easily handled by the AD9852. Given the time bin size $\Delta\tau$=0.25 ns required for LMDS channel sounding, a frequency tuning resolution Δf = 1 mHz, and no limitations on the size of the memory buffer, the smallest supportable transmit pulse repeition frequency is α = 2 kHz, where

$$\beta = \alpha(1-\frac{1}{\gamma}),\ \gamma = \frac{T_c}{\Delta\tau} = \frac{1}{\alpha\Delta\tau} = N = \frac{\alpha}{\alpha-\beta}$$

$$\Delta f = \alpha - \beta = \alpha - \alpha(1-\frac{1}{\gamma}) = \alpha(1-(1-\frac{1}{\gamma})) = \alpha\frac{1}{\gamma} = \alpha\alpha\Delta\tau = \alpha^2\Delta\tau$$

$$\alpha = \sqrt{\frac{\Delta f}{\Delta\tau}}$$

This requires a memory buffer of N = 2e6 words, which is well beyond the 1e6 words available on the current DSP board. If the memory limitation of the current DSP board is taken into account (N≤1e6 words), where

$$N = \frac{1}{\alpha\Delta\tau}$$

$$\alpha = \frac{1}{N\Delta\tau}$$

$$\Delta f = \alpha\frac{1}{\gamma} = \alpha\frac{1}{N}$$

then α = 4 kHz and the minimum required tuning range of the DDS is Δf = 4 mHz, which is met by the 1 mHz DDS tuning limits. Remember that the transmit pulse repetition frequency α determines unambiguous range of incoming multipath signal components and measurement time. Therefore we can say that the maximum unambiguous range of incoming multipath signal components, or minimum pulse repetition period, is currently limited by the amount of memory on the current DSP

board. If the memory requirements are lifted, then the tuning range of the DDS limits the maximum unambiguous range of incoming multipath signal components and minimum pulse repetition period.

The "AD9852_54 Consumer Evaluation Software" software package was provided by Analog Devices to permit in-field programming of the AD9852 evaluation board.

3.4.3 TX Pulsar Module

System function

The TX Pulsar Module provides a short pulse of duration $\tau_{bb} = 2 / BW$ and pulse amplitude A_{tx} = -25.5 dBm at the input of the TX LMDS Radio Module. This ensures the peak pulse power used to sound the LMDS channel bandwidth (BW) does not exceed the 1 W TX LMDS Radio Module power output limit, as discussed in the previous link budget. Since this pulse occupies the given bandwidth, it is treated as an impulse traveling through the channel, producing the impulse response of the channel at the receiver. The short pulse is driven by the transmit pulse repetition frequency α provided by the TX Frequency and Location Module.

Implementation and design tradeoffs

The design process for implementing the TX Pulsar Module involved investigating methods that would produce short pulses with pulse width durations ranging from 1 ns to 25 ns at a low cost. Several methods were considered, including utilizing tunnel diodes, gate feedback circuits, and avalanche transistor breakdown. The avalanche transistor breakdown methodology was chosen to implement the short pulse circuit because of its simplicity and availability of parts.

The TX Pulsar Module was implemented by Dr. Dennis Sweeney using a custom circuit design derived from [40]. Figure 3.3 below shows the basic circuit topology provided by Dr. Sweeney [37].

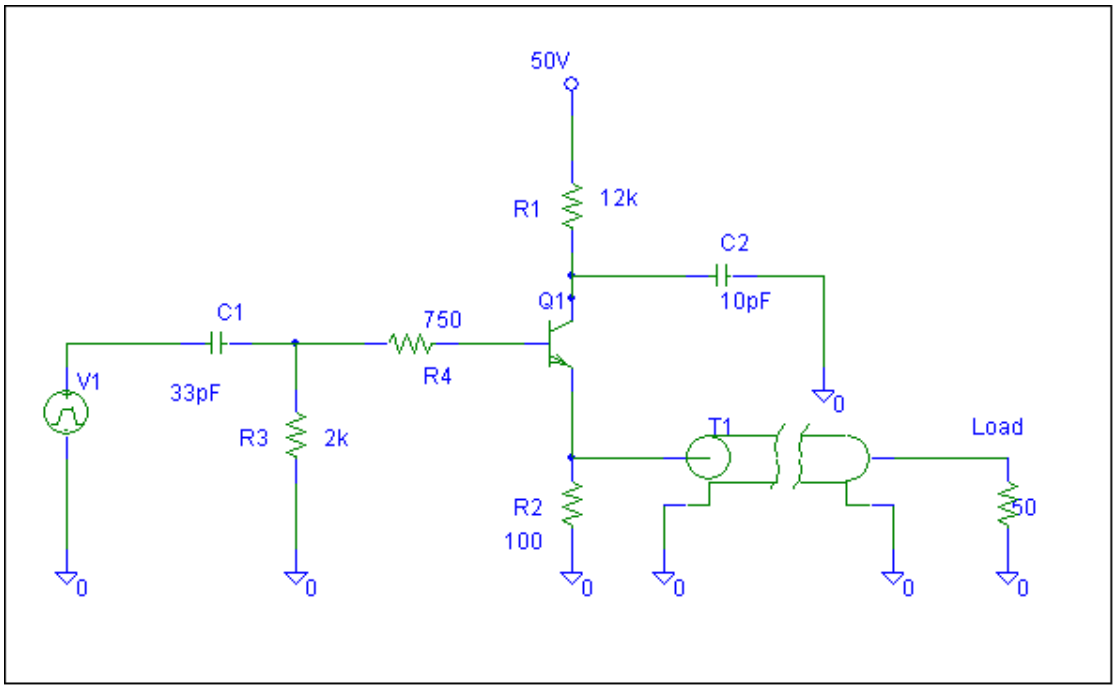


Figure 3.3: TX Pulsar Module circuit diagram.

The following text was excerpted Dr. Sweeney's design notes for the TX Pulsar Module and describes the basic operation of the current TX Pulsar Module implementation.

> "[In Figure 3.3] The transistor, Q1, is normally biased off. This is accomplished by returning the bias resistor R3 to ground. The collector voltage is set so it exceeds the transistor breakdown voltage, but as long as the device is biased off, no collector current flows. The positive edge of the input pulse is differentiated by the small value capacitor C1 and drives the base of Q1 positive. The instant that the base of Q1 goes positive, the transistor goes into avalanche breakdown. Due to the avalanche, the transistor goes short circuit in a very short period of time and the charge in C2 is dumped into R2 and the load resistor. The avalanche causes the load voltage to rise in about 0.5nsec. As the charge is dissipated, the

output voltage slowly decays, the transistor becomes non-conducting, and C2 recharges through R1. The transistor can then be fired again."[37]

The resulting pulse occupies a very wide spectrum and as such must be band limited before supplying it to the IF input of the TX LMDS Radio Module. The following text from Dr. Sweeney's design notes discuss how he implemented this necessary filtering.

"The LMDS upconverter that follows the pulsar circuit is a bandpass device. Therefore, prior to being passed to the LMDS upconverter, the pulse is applied to a bandpass filter to band limit it as shown in Figure 3.4. This filter must be carefully chosen to produce a well-defined pulse. Based on the output of simulation, the impulse responses of conventional Butterworth and Chebyshev filters predicted that the pulse would be smeared. Initial testing with an available Chebyshev filter produced unacceptable results, confirming the simulation. To combat the pulse smearing, a transitional 6 dB Gaussian filter was designed and simulated. This filter has a Gaussian frequency response in the passband and a near Butterworth roll off in the stopband, which represents a reasonable trade off between good time domain characteristics and a well-defined frequency spectrum. The simulated output of a Butterworth filter is shown in Figure 3.5, while Figure 3.6 shows the same output from the Transitional filter. The spectrum of the short pulse transmitter is shown in Figure 3.7."

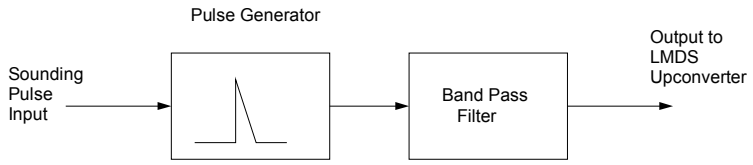


Figure 3.4: TX Pulsar Module.

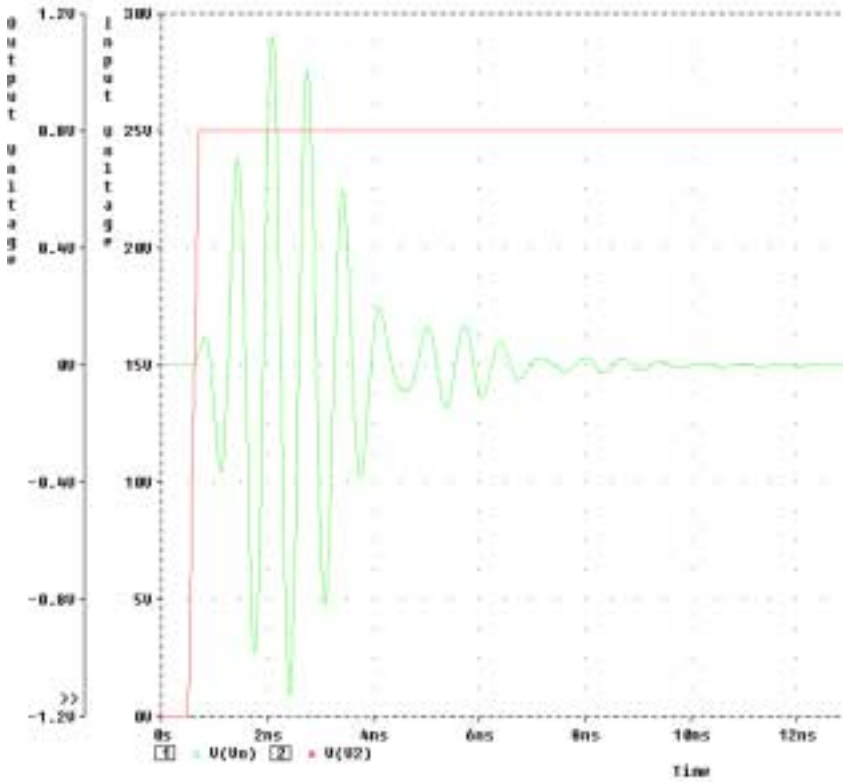


Figure 3.5: Short pulse transmitter output: Butterworth filter. Notice the extended "tail" on the Butterworth filtered pulse.

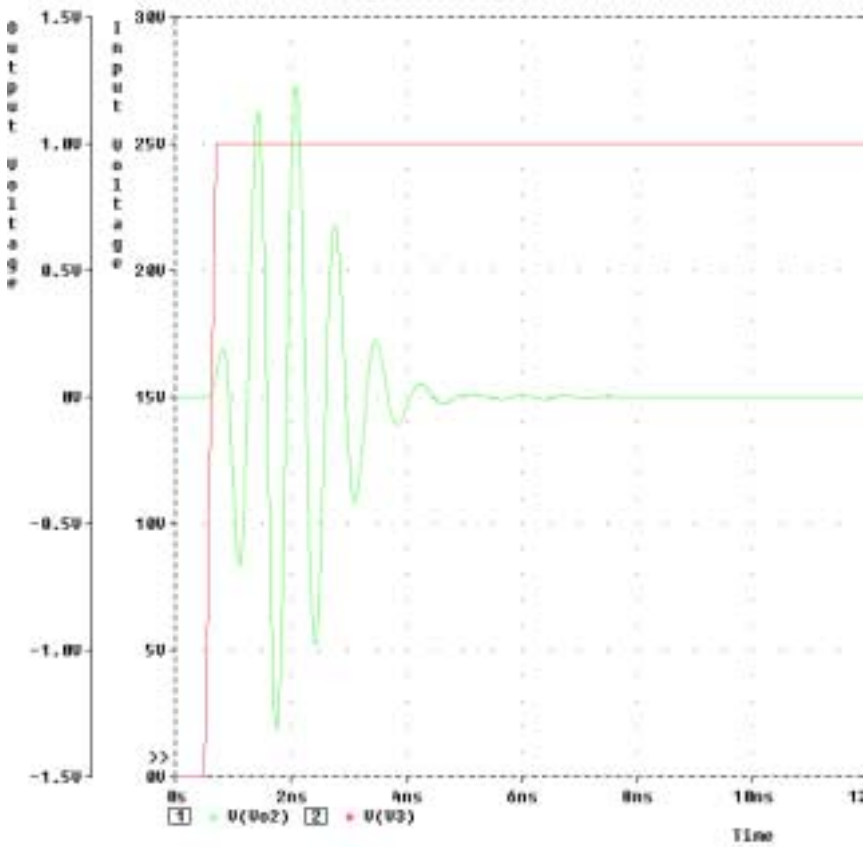


Figure 3.6: Short pulse transmitter output: Transitional filter.

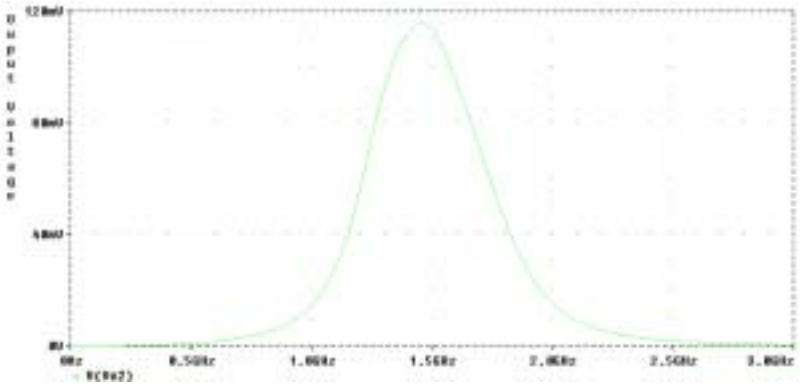


Figure 3.7: Short pulse transmitter spectrum with Transitional filter.

The specific filtering used is dependent on the particular channel sounding bandwidth under study.

3.4.4 TX LMDS Radio Module

System function

The TX LMDS Radio Module radio frequency (RF) up converts and amplifies the short pulse produced by the TX Pulsar Module to permit wireless transmission and reception of the short pulse in the LMDS Block A spectrum bandwidth.

Implementation and design tradeoffs

The TX LMDS Radio Module requires an IF frequency input port that meets or exceeds the channel sounding bandwidth (BW), thereby permitting the entire short pulse to be up converted to the LMDS band without distortion. In addition, the radio must be able to output at least 1 W of power.

The TX LMDS Radio Module was implemented by the author by obtaining a Motorola Spectrapoint LMDS TX1 Radio, which was readied for operation with the assistance of Dr. Dennis Sweeney, Shital Chheda, and Cindy Dillard. These radios were chosen because they could be readily customized with different channel sounding bandwidth filters depending on the channel measurement campaign needed. The current implementation uses the supplied 630 MHz filters and an IF of 1770 MHz. The radio outputs 1 W (30 dBm) of power with a 4.5 dBm 1770 MHz IF input, providing 25.5 dB of gain in the transmitter prior to the transmit antenna. Specific details of the transmitter radio implementation can not be disclosed due to proprietary restrictions.

The Spectrapoint transmitter radio requires a 960 MHz frequency reference used to lock the internal 26.15 GHz local oscillator. This reference was implemented by Dr. Dennis Sweeney by designing a custom board that multipled a 10 MHz reference up to 960 MHz. The 10 MHz reference is currently implemented using a separate 10 MHz oscillator, however the GPS disciplined 10 MHz VCXO could also be used.

3.4.5 RX LMDS Radio Module

System function

The RX LMDS Radio Module radio frequency (RF) amplifies and down converts the impulse response of the LMDS Block A spectrum bandwidth (BW) and provides that waveform at the IF output port of the RX LMDS Radio Module.

Implementation and design tradeoffs

The RX LMDS Radio Module requires an IF frequency output port that meets or exceeds the channel sounding bandwidth (BW), thereby permitting the entire short pulse to be down converted from the LMDS band without distortion. In addition, the radio must have enough sensitivity to be able to receive signals with -97dBm of power.

The RX LMDS Radio Module was implemented by the author by obtaining Motorola Spectrapoint LMDS RX1 Radios, which were readied for operation with the assistance of Dr. Dennis Sweeney, Shital Chheda, and Cindy Dillard. These radios were chosen because they could be readily customized with different channel sounding bandwidth filters depending on the channel measurement campaign needed. The current implementation uses the supplied 200 MHz filters and an IF of 1770 MHz. However, these filters could be replaced with 630 MHz filters, since the output of the down converter block will be the entire 630 MHz of transmitted spectrum. The radio has several stages of gain. The power gain through the downconverter is approximately 20 dB and the power gain through the IF circuitry of the receiver is 37 dB. This provides an aggregate receiver gain of 57 dB following the receive antenna. Specific details of the receiver radio implementation can not be disclosed due to proprietary restrictions.

The Spectrapoint receiver radio requires a 960 MHz frequency reference used to lock the internal 26.15 GHz local oscillator. This reference was implemented by Dr. Dennis Sweeney by designing a custom board that multipled a 10 MHz reference up to 960 MHz. The 10 MHz reference is currently implemented using a separate 10 MHz oscillator, however the GPS disciplined 10 MHz VCXO could also be used.

3.4.6 RX IF Sampler Module

System function

The RX IF Sampler Module samples, holds, and applies fixed gain to the impulse response waveform of the LMDS channel that appears at the output of the RX LMDS Radio Module. This allows a slower less expensive ADC to follow the fast IF Sampler and digitize the resulting samples. The IF Sampler uses a feedback loop to set the fixed gain to ensure that the peak pulse amplitude within a pulse repetition period corresponds with the maximum voltage swing of the ADC, as discussed in the previous link budget. The IF Sampler is driven by the receive sampling frequency β_δ provided by the RX Frequency and Location Module.

Implementation and design tradeoffs

The design process for implementing the TX Pulsar Module involved investigating methods that would permit very fast (less than 1 ns) sample and hold of the impulse response waveform present at the RX IF Output of the RX LMDS Radio Module. Two methods were considered - implementing the high speed sampler function on a custom board using chips made by Alpha or utilizing a S6 Tektronics Sampler Head Plugin. The S6 Tektronics Sampler Head Plugin methodology was chosen to implement the high speed sampler circuit because of its simplicity, availability of parts, and it enabled rapid prototyping. A custom adjustable fixed gain block circuit was also implemented.

The RX IF Sampler Module was implemented by Dr. Dennis Sweeney using a custom circuit design. Figure 3.8 below shows the basic circuit topology provided by Dr. Sweeney [37].

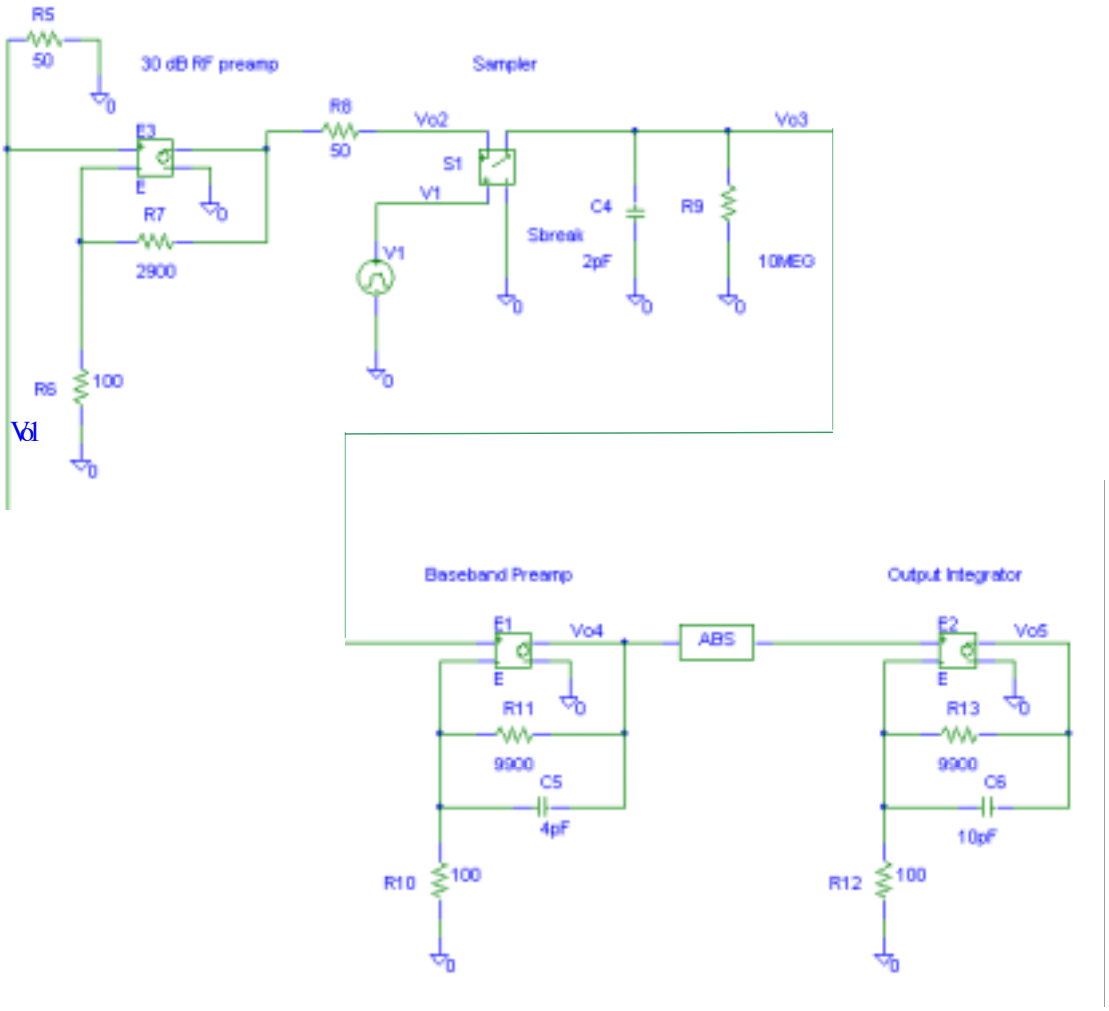


Figure 3.8: RX IF Sampler Module circuit diagram.

The following text was excerpted from Dr. Sweeney's design notes for the RX IF Sampler Module and describes the basic operation of the current RX IF Sampler Module implementation.

> "The receiver circuitry for the sounder is more complex than the transmitter circuitry. In order to reduce the design and implementation time, existing circuitry was adapted. The receiver consists of a high speed sampler and an integrator shown in Figure 3.9.

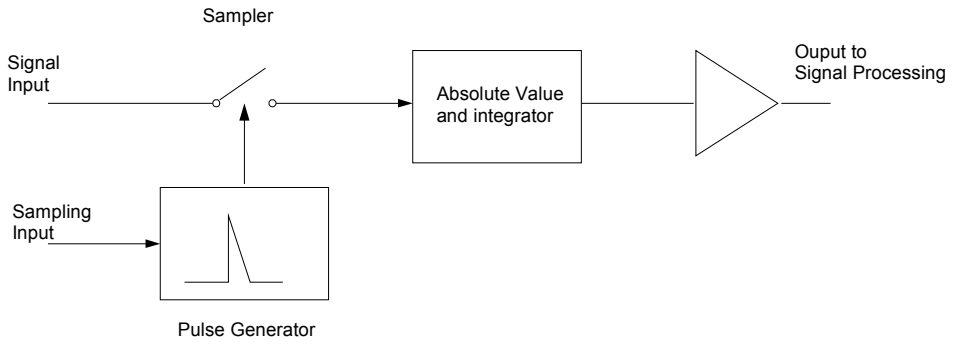


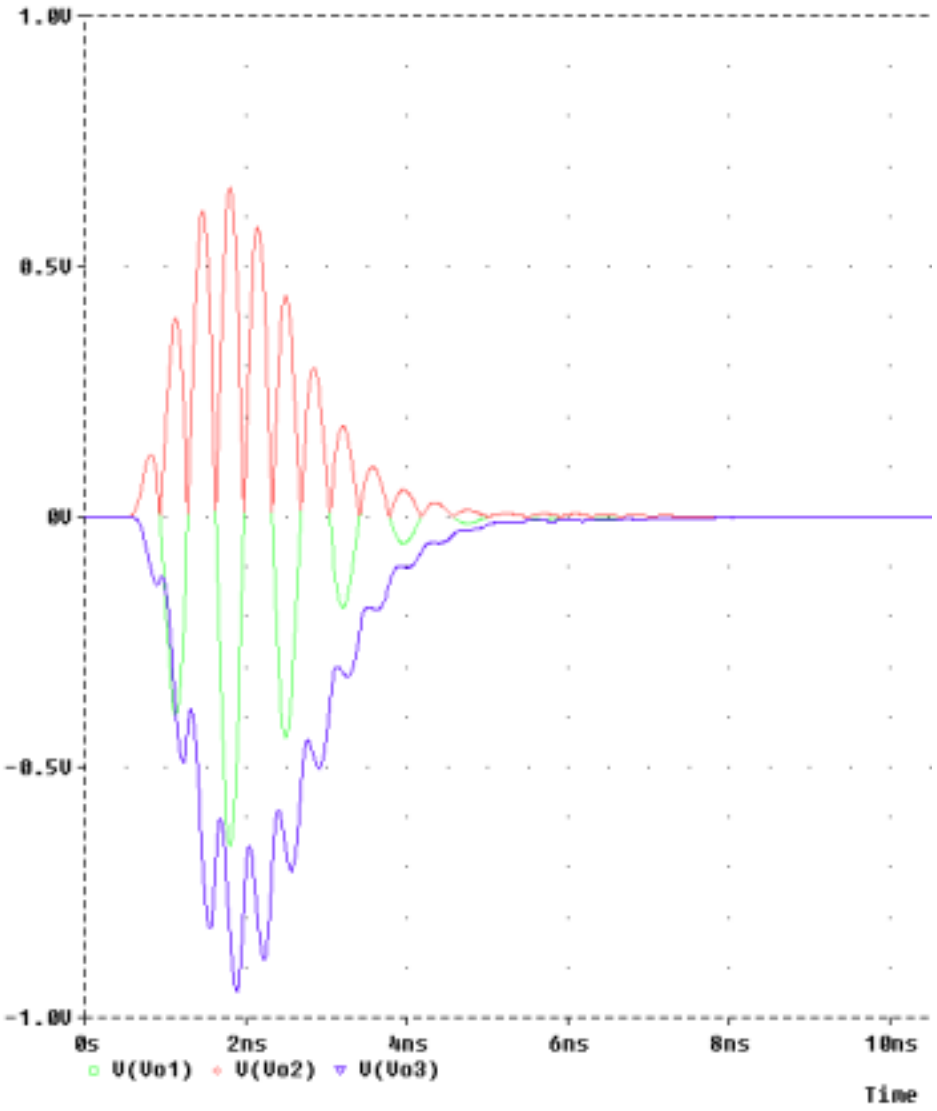
Figure 3.9: Short pulse receiver front end.

An S-6 sampler and pulse generator from a Tektronics 7000 series sampling oscilloscope has been adapted for the short pulse receiver. An external interface circuit has been designed to connect to the preexisting S-6 module, which already contains operational high frequency circuitry.

Ideally, the receiver reproduces the output shown by the transitional filter in Figure 3.6, only expanded in time from a time scale of nanoseconds to microseconds, due to time dilation. The block diagram shown in Figure 3.9 turns the transitional filter output in Figure 3.6 back into a pulse. The actual circuit used to realize the block diagram is shown in Figure 3.8, while the operation of the circuit is detailed in Figure 3.10. Notice that this simulation is operating directly at the output and not after the time dilation. The absolute value operation and integration takes place in the DSP

The S-6 sampler was mounted in a shielded enclosure with power lead filtering to provide isolation from noise, since the close proximity of an AM broadcast station prevented operation at low signal levels.

A high gain AGC amplifier was constructed to follow the sampler. The output is adjusted to produce approximately 2 volts (V) peak output. This is consistent with the 2.5 V peak maximum input for the A/D converter. This amplifier also required shielding to prevent stray signal pickup. " [37]



Excerpted from [37]

Figure 3.10: Pulse detection and output.
The red and green trace is the original pulse, the red trace is the absolute value, and the blue is the output from a leaky integrator.

The specific fixed gain needed in the RX IF Sampler module is dependent on the amount of path loss and attenuation the peak pulse experiences within a pulse repetition period. This gain control can be achieved manually or using a digital feedback loop to appropriately adjust the RX IF Sampler gain block to ensure the peak pulse amplitude corresponds with the maximum +-2 volt swing of the ADC.

3.4.7 RX DSP Module

System function

The RX DSP Module digitizes the incoming sample values present at the output of the RX IF Sampler Module, buffers those impulse response values to memory, and transfers the entire buffer to the RX Host Module when the SSTDSP sounding measurement is complete to permit computation of the power delay profile and channel metrics. The power delay profile and channel metrics calculation may also be implemented in the RX DSP Module to allow direct real time communication between the RX DSP Module and other external devices like LMDS modems. The current SSTDSP sounder implementation utilizes the RX Host Module to calculate the power delay profile and channel metrics because this enabled rapid algorithm prototyping through MATLAB.

Implementation and design tradeoffs

The design process for implementing the RX DSP Module involved implementing the two functional blocks listed above. A primary goal was to implement this module with operational flexibility and at a fraction of the cost of commercially available high speed data acquisition boards. This was achieved by utilizing low cost off the shelf components and innovative implementation techniques, resulting in an implementation that cost several hundred dollars instead of several thousand dollars. Table 3.4 below indicates the hardware solution used by the author to implement each block.

Table 3.4: RX DSP Module hardware.

Block	Company	Hardware	Function
ADC	Linear Technology	LTC1414, 14 bit 2.2 Msps max Sampling ADC	Sample the incoming value present at the RX IF Sampler output and digitize the value at the RX Frequency β.
DSP SRAM I/O	Motorola	Custom DSP board designed and implemented by the author at Motorola	Buffer the incoming ADC values in Fast SRAM to assemble an impulse response and transmit the Fast SRAM

		utilizing the Motorola 56311 DSP, 1Mx24 bit Fast SRAM, and several I/0 ports including Parallel Command Converter, JTAG, RS-232, and HPI	buffer to the RX Host Module when the SSTDSP measurement is complete. May also calculate the power delay profile and channel metrics

An ADC chip with high precision external sample triggering capabilities was needed to allow the RX Frequency and Location Module to drive the sample instant of the ADC at the RX Frequency β. The Linear Technologies LTC1414 14 bit 2.2 Msps max Sampling ADC was chosen because of its excellent dynamic range (78 dB at 1.1 MHz) and a low sample jitter (3 ps, rms). The sampling rate of the ADC determines the bounds of the transmit pulse repetition frequency α. The LTC1414 can operate with full dynamic range given sampling frequencies ranging from 1 kHz to 1 MHz, which means the current SSTDSP sounder pulse repetition frequency α can range from 1 kHz to 1 MHz.

A DSP with adequate processing power, on chip I/O peripherals, and a data path capable of handing the 14 bit ADC values was needed. The processing budget, or MIPs (million of instructions per second) budget, is determined by the number of calculations that must occur during the transmit pulse repetition period (T_c) and/or SSTDSP measurement period (T_{tot}). Since α = 1 MHz is the fastest pulse repetition frequency that is supported by the current SSTDSP implementation, the number of calculations per pulse repetition period (N_{calc}) that a 100 MIPs DSP can perform in T_c = 1 us, is N_{calc} = 100 operations, where:

$$N_{calc} = T_c MIPs = \frac{1}{\alpha} MIPs$$

Since the SSTDSP DSP algorithm requires less than 100 operations to be performed between samples, any DSP that has a MIPs budget of 100 MIPs or greater can be used. As the transmit frequency α decreases, the number of operations the DSP can perform

given a fixed MIPs budget increases. For a transmit frequency α = 48 kHz and MIPs = 100, then N_{calc} = 2083.

Therefore, the required MIPs for the RX DSP module is bounded by the transmit pulse repetition frequency α and the number of SSTDSP operations needed between samples which can vary depending on the computations done by the DSP. If all instruction cycles are counted in clock cycles, the current code requires at most 99 clock cycles between samples to acquire and store the incoming values in the impulse response Fast SRAM buffer (See Appendix C for DSP code and specific clock cycles needed per assembly instruction macro). Therefore, if the DSP runs at α = 1 MHz, no additional calculations such computing the power delay profile or channel metrics are possible. However, if the SSTDSP sounder runs at α = 48 kHz, an additional 2083-99=1984 clock cycles are available to do computations such as assembling the power delay profile and calculating the channel metrics.

The Motorola 56311 DSP was chosen because of it 150 MIPs (million of instructions per second) processing power, 24 bit processing path, and built in I/O peripheral interfaces capable of providing a Parallel Command Converter function, JTAG, RS-232, and a high speed data transfer port interface. The Motorola 56311 DSP chip has the following features, shown in Table 3.5:

Table 3.5: Motorola 56311 DSP features.

Features
• 255 Effective MIPS, 150 Core MIPS
• Programmable Enhanced Filter Coprocessor (EFCOP), 150 MIPS
• 128K words (24-bit) on-chip SRAM
• 5 memory switch options
- P memory: 32K, 48K, 64K, 80K, or 96K
- D memory: 96K, 80K, 64K, 48K, or 32K
• 2 ESSI ports, SCI port, HI08 Host Interface, 3 Timers, 6 DMA channels
• 18-bit Address, 24 or 16-bit Data external memory interface
• 1.8v core and 3.3v I/O

- 0.7mA/MIPS based on simulation
- Small 196-pin PBGA package: 15 X 15mm, 1mm ball pitch
- Upward migration from DSP56307, DSP56309 or DSP56303 in PBGA

The RX DSP Module board was implemented with a LTC1414 ADC, a Motorola 56311 DSP, a 1Mx24 bit high speed external Asynchronous Fast Static Random Access Memory (Fast SRAM), a RS-232 serial port using the SCI (serial connect interface) port, a Parallel Command Converter and JTAG test interface, and a high speed adaptable 8 bit data HPI (Host Port Interface).

The Motorola 56300 ADS DSP Software Development package was recommended by Motorola to permit DSP code development and debugging, binary executable download, on-chip execution, and control and monitoring of the RX DSP Module hardware platform operation. It provides a graphical user interface and data logging capabilities.

3.4.8 RX Host Module

System function

The RX Host Control Module calculates SSTDSP system parameters based on user input through the MATLAB Interface. It then provides control signals to program the receive DDS, GPS Sync board, and GPS boards within the RX Frequency and Location Module and DSP within the RX DSP Module. It also receives data from the GPS unit within the RX Frequency and Location Module and displays the GPS receiver data, including location, acquired satellite signals, heading, status of GPS receiver, and time. In addition, the RX Host Module receives channel data from the RX DSP Module. The RX Host Control Module then takes this channel data and computes the power delay profile and channel metrics, displaying them graphically through the MATLAB Interface.

Implementation and design tradeoffs

The design process for implementing the RX Host Module involved finding a Windows 2000 Professional laptop or tower that had a DB9 serial port, DB25 parallel port, CDROM for installing software, and 10 GB of hard drive space to install the five required programs and reference material, provided in SSTDSP software package. A high performance computer was needed at the receiver, because a great deal of control and processing was occurring at the receiver.

The RX Host Module was implemented by the author using a Dell Workstation running a Pentium 3 processor at 650MHz with 256 kB of cache, 192 MB of RAM, 8 MB of video RAM, and a 10 GB hard drive provided by CWT. The following software packages were installed on the workstation as part of the SSTDSP software package, as shown in Table 3.6.

Table 3.6: RX Host Module software.

Company	Software	Version	Function
Analog Devices	AD9852 Customer Evaluation Software	1.7.1	DDS Control
Apollocom	OutlookPlus GPS	1.0 Build 8	GPS Control
Hilgraeve	HyperTerminal	1999	GPS Sync Control
Motorola	DSP56300 ADS	6.3.4	DSP Control
Mathworks	MATLAB	5.1.0.421	Process Channel Data

These software packages were used to control various receiver hardware blocks, as indicated in Table 3.6.

3.4.9 TX and RX Power Module

System function

The TX and RX Power Modules supply the required power needed by the hardware components found in the transmitter and receiver, respectively.

Implementation and design tradeoffs

The TX and RX Power Modules were implemented using a power strip, numerous "wall wart" power supplies that matched the required voltages, and a custom interface box designed and implemented by the author that permits the power from the various "wall wart" power supplies to the appropriate voltage input terminals found on the various modules.

The next section discusses the numerous SSTDSP algorithms in further detail.

3.5 Algorithm details

The following section provides addition implementation details about each of the SSTDSP algorithms. Each section provides the following information about the particular algorithm and the specific implementation realized in this thesis:

1. System function
2. Algorithm flowchart

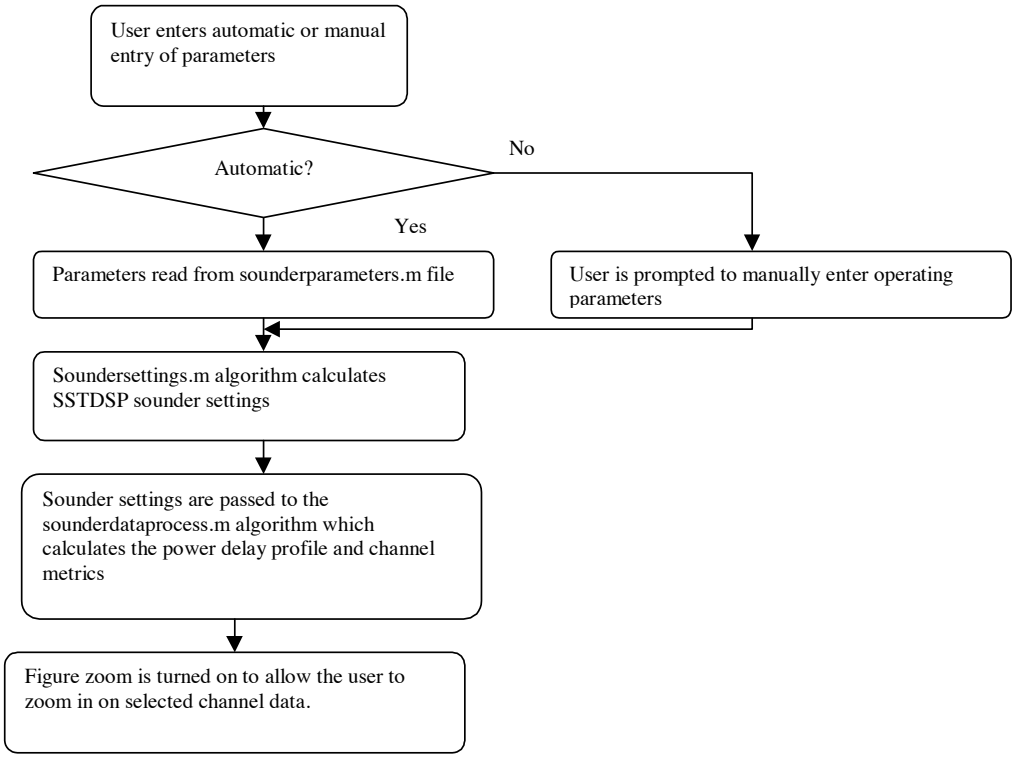
Appendix B indicates the specific target programming language used for each SSTDSP sounder algorithm and provides step by step SSTDSP sounder algorithm processing flow details. Source code for the algorithms can be found in Appendix C.

3.5.1 "processdata.m"

System function

This algorithm serves as the primary MATLAB Interface to the SSTDSP sounder.

Algorithm flowchart

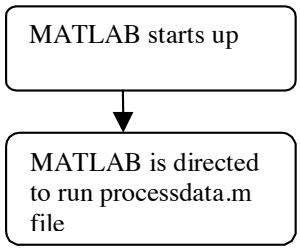


3.5.2 "startup.m"

System function

This algorithm permits automatic launch of SSTDSP MATLAB Interface.

Algorithm flowchart

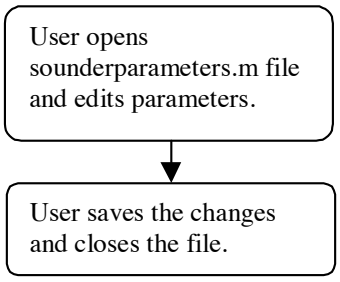


3.5.3 "sounderparameters.m"

System function

This algorithm allows the user to predefine SSTDSP sounder parameters file to allow rapid SSTDSP sounder measurement campaigns.

Algorithm flowchart

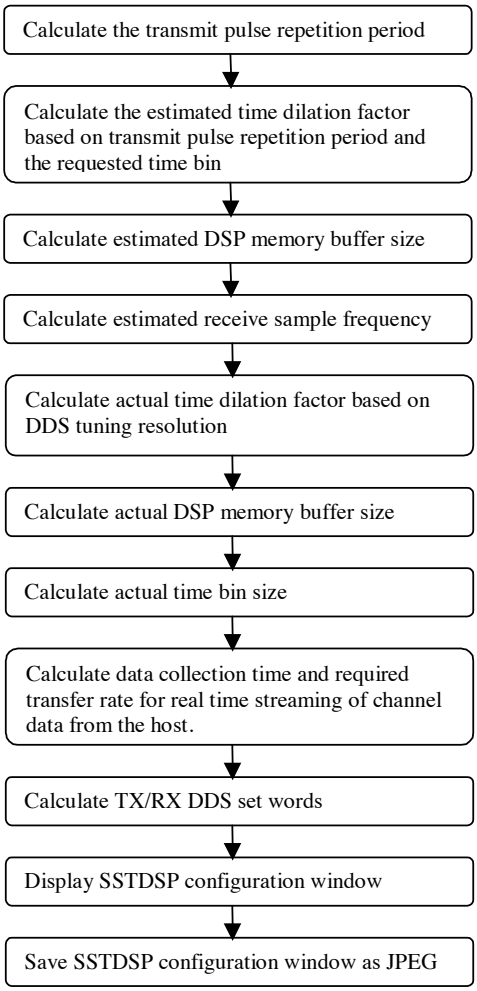


3.5.4 "soundersettings.m"

System function

This algorithm computes SSTDSP system configuration settings based on user input, including the actual time bin size that can be supported by the mHz tunable DDS, the required receive sample frequency, and the size of the DSP memory buffer needed and corresponding hexadecimal buffer end address. The algorithm also utilizes the data collection time in seconds and minutes, the required data rate to permit real time streaming of impulse response data from the DSP to the RX Host Module, the transmit DDS set word in hexadecimal and binary, and the receive DDS set word in hexadecimal and binary.

Algorithm flowchart

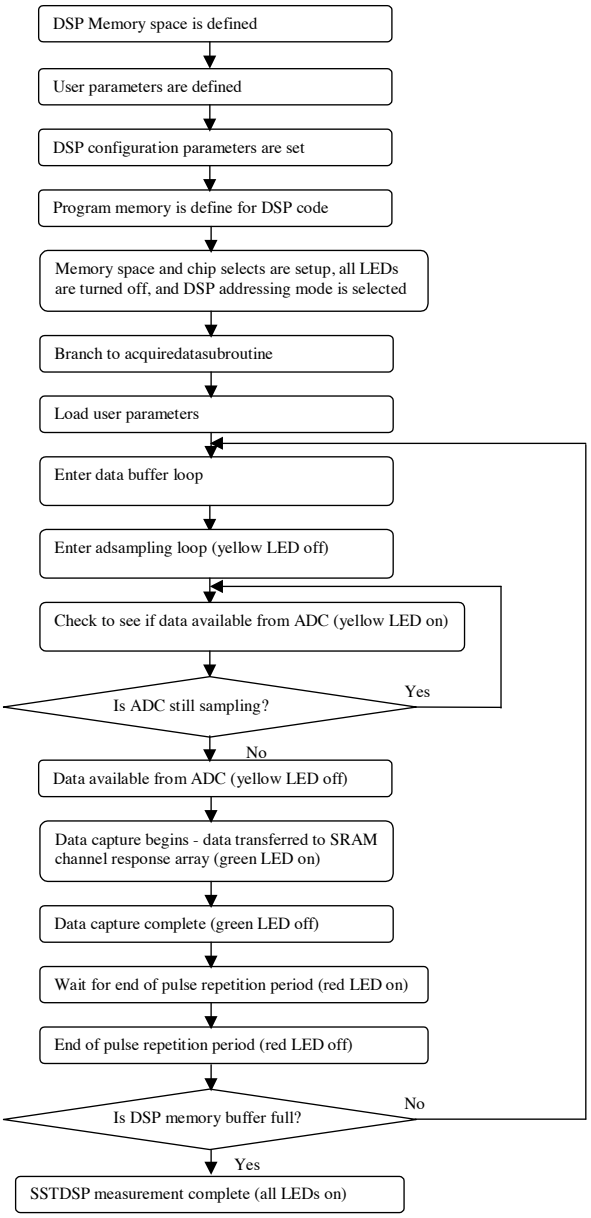


3.5.5 "AtoD1.asm"

System function

This algorithm configures and controls the DSP in the RX DSP Module as well as facilitates ADC data capture and DSP memory buffering.

Algorithm flowchart

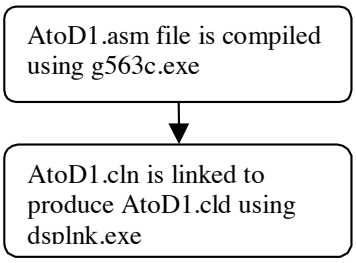


3.5.6 "make.cmd"

System function

This algorithm compiles and assembles the real time DSP code found in "AtoD1.asm" to a binary executable file to be run on the DSP.

Algorithm flowchart

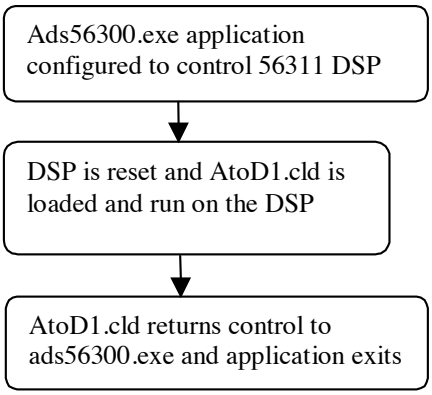


3.5.7 "capturedata.cmd"

System function

This algorithm initiates DSP channel data capture using the Motorola "ads56300.exe" application.

Algorithm flowchart

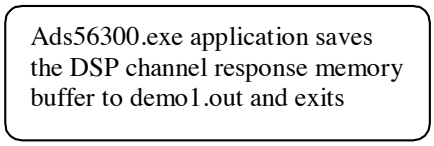


3.5.8 "exportdata.cmd"

System function

This algorithm transfers the DSP impulse response memory buffer from the DSP to an ASCII file in RX Host Module.

Algorithm flowchart

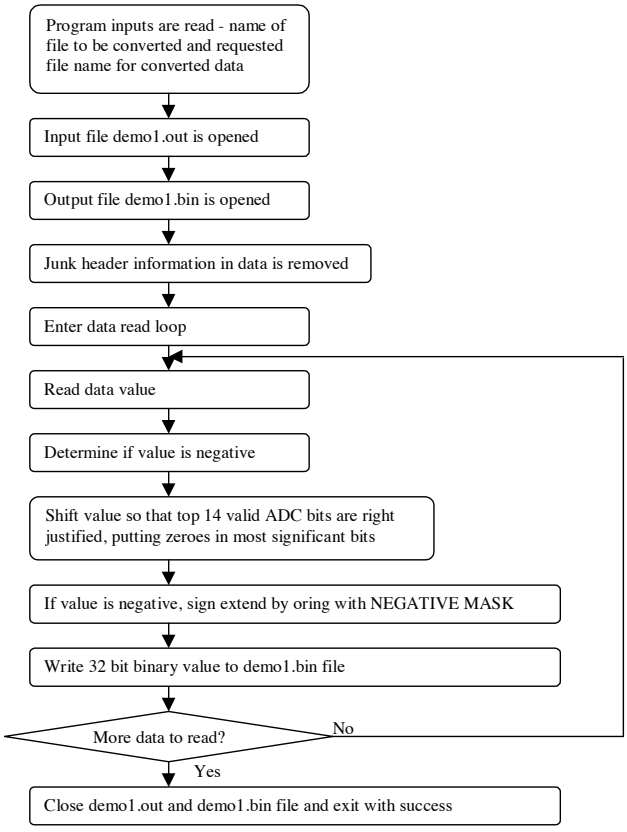


3.5.9 "sounderdataconverter.cpp"

System function

This algorithm converts the ASCII memory buffer file "demo1.out" to binary file format "demo1.bin" needed by MATLAB. Thanks go to Todd Eshler for rapidly implementing this utility program.

Algorithm flowchart

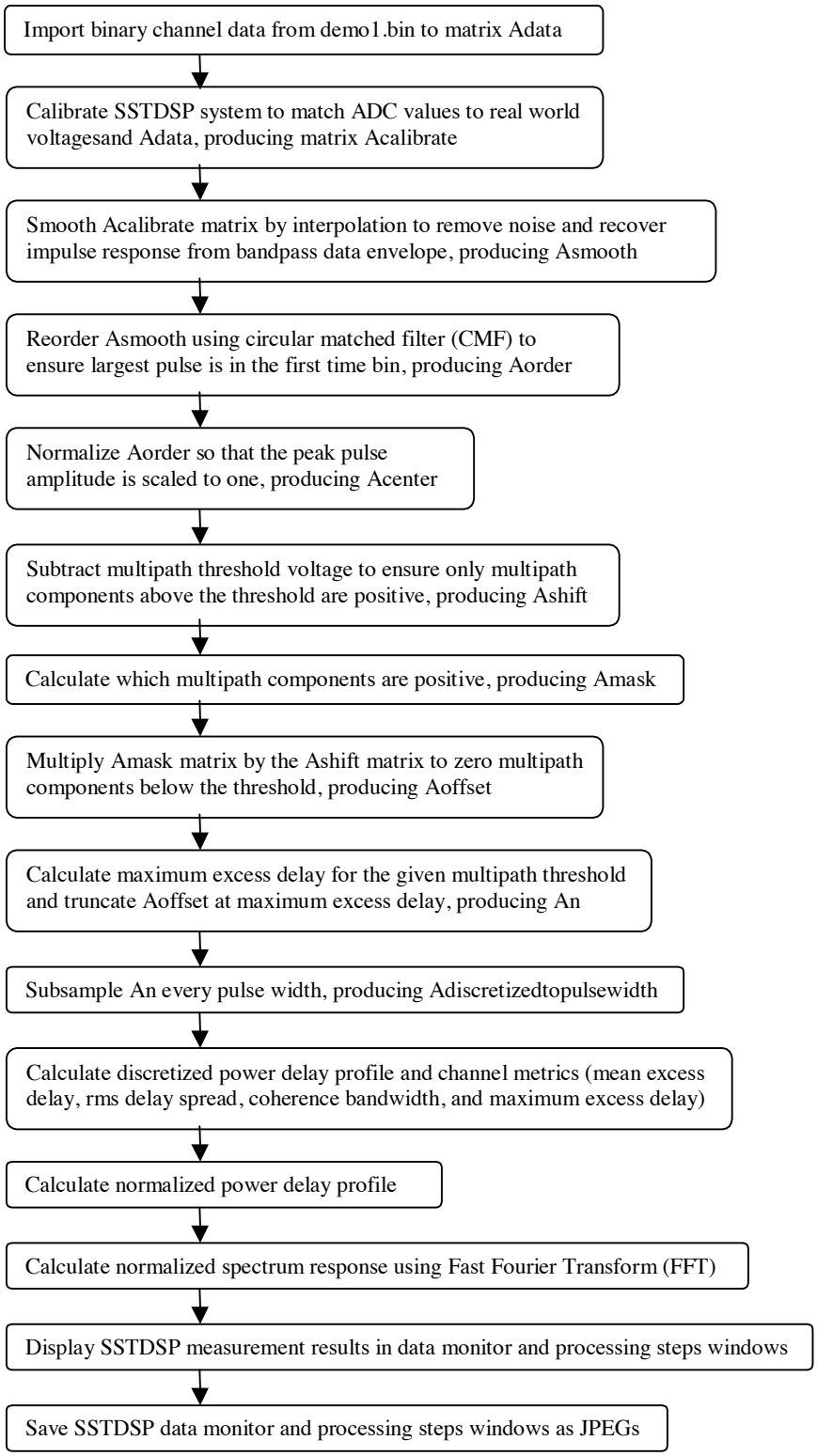


3.5.10 "sounderdataprocess.m"

System function

This algorithm imports the binary memory buffer file into MATLAB Interface, implements a novel Circular Matched Filter (CMF) algorithm to find the peak pulse value and position in pulse repetition frame, and utilizes the CMF to set the necessary IF Sampler gain level and reorder the channel data to ensure the original pulse appears in the first time bin. The algorithm then scales the impulse response, computes the Power Delay Profile (PDP), computes the channel metrics, computes the frequency response of the channel, and displays all relevant information through the graphical MATLAB Interface.

Algorithm flowchart



3.6 Script details

The following section provides addition implementation details about each of the SSTDSP scripts. Each section indicates the script's system function within the SSTDSP sounder. Source code for the scripts can be found in Appendix C.

3.6.1 "TX_STEP1_setdds.cmd"

System function

Launches AD9852_54 Eval Software so that user can program the DDS for the correct transmit frequency. Programming words are supplied by the MATLAB Interface.

3.6.2 "TX_STEP2_startgps.cmd"

System function

Launches OutlookgGPSPlus so that user can monitor GPS operation.

3.6.3 "RX_STEP1_configuresounder.cmd"

System function

Launches MATLAB and calculates necessary operating parameters based on user input.

3.6.4 "RX_STEP2_editassembly.cmd"

System function

Launches Notepad so user can edit DSP assembly code to allocate correct memory buffer size on DSP board. Correct memory buffer length is supplied in hexadecimal and binary by the MATLAB Interface. User must save the document and close.

3.6.5 "RX_STEP3_editdspcontrol.cmd"

System function

Launches Notepad so user can edit DSP control script with correct memory buffer length to permit correct memory buffer transfer from DSP board to RX Host Module. The MATLAB Interface supplies the correct memory buffer length in hexadecimal and binary format. User must save the document and close which recompiles necessary binary DSP executable code.

3.6.6 "RX_STEP4_SETSWITCHTOA_setdds.cmd"

System function

Informs user to set switch box present at the sounder receiver to position "A" so that the RX Host Module can launch and control the DDS through the AD9852_54 Eval Software. This software permits the user to program the DDS for the correct receiver frequency. Programming words are supplied by the MATLAB Interface.

3.6.7 "RX_STEP5_SETSWITCHTOB_setdds.cmd"

System function

Informs user to set switch box to position "A" and launches AD9852_54 Eval Software so that user can program the DDS for the correct receiver frequency. Programming words are supplied by the MATLAB Interface.

3.6.8 "RX_STEP6_startgps.cmd"

System function

Launches OutlookgGPSPlus so that user can monitor GPS operation.

3.6.9 "RX_STEP7_SETSWITCHTOC_aquireconvertdata.cmd"

System function

Informs user to set switch box to "C" and launches the script that acquires and converts SSTDSP channel measurement data. This is actually when the measurement takes place. User may be prompted to overwrite a file if data measurements already are present in the "D:\BroadbandSounder\captureddata" directory, indicated by a preexisting "demo1.out" file.

3.6.10 "RX_STEP8_PRESSSPACEBARINMATLAB_processdata.cmd"

System function

Informs user to press the space bar in MATLAB to complete data processing and display power delay profile and channel metrics.

The following chapter discusses channel measurement and results using the SSTDSP sounder.

Chapter 4

Channel Measurement and Results

4.1 Review of SSTDSP sounder measurements

The following measurements taken by the SSTDSP sounder are reviewed for reference. The sounder produces the digital power delay profile, mean excess delay, rms delay spread, coherence bandwidth bounds, and maximum excess delay.

4.2 Calibration and verification process

The SSTDSP sounder was calibrated through the MATLAB Interface to ensure that the binary quantization levels recorded by the ADC properly map to the voltages present at the input to the ADC. This was done by inputting a square wave of known peak voltage and observing the max binary quantization level provided by the ADC in the DSP impulse response memory buffer. This peak quantization value (PEAKQUANTIZATION) for a given peak input voltage to the ADC (PEAKVOLTAGE) was used to calculate the fixed calibrate ratio:

$$CALIBRATERATIO = \frac{PEAKVOLTAGE}{PEAKQUANTIZATION}$$

This fixed ADC calibration ratio is then used to scale all incoming data from the DSP impulse response memory buffer. This calibration is needed to ensure that the values processing by the SSTDSP software algorithms correspond to real world measurement quantities provided by the SSTDSP hardware block. The hardware found in the SSTDSP DSP Module, TX and RX Frequency and Location Modules, and TX and RX LMDS

Radio Module hardware have fixed operation or are self calibrating, as in the case of the GPS unit, and therefore do not require any user calibration.

The SSTDSP sounder baseband transmitter operation was verified by configuring the TX Frequency and Location Module for a known transmit pulse repetition frequency and observing the TX Pulsar Module output on an oscilloscope. In a similar fashion the SSTDSP sounder baseband receiver operation was verified by connecting the output of the baseband SSTDSP transmitter to the input of the baseband SSTDSP receiver. The RX Frequency and Location Module was then configured for the receive sample frequency provided by the MATLAB SSTDSP configuration window, given the user system parameters. The original pulse repetition period should be displayed by the MATLAB Interface as well as corresponding channel metrics, verifying correct SSTDSP sounder operation.

Since the original input was a pulse train without any multipath, the mean excess delay and rms delay spread should be zero, the coherence bandwidth should be the full sounding bandwidth, and the maximum excess delay should be the short pulse duration. These quantities were verified in Figures 4.1, 4.2, and 4.3 below using a 3 us short pulse duration with transmit pulse repetition frequency of 48 kHz.

Figure 4.1 displays several important sounder parameters including the transmitted pulse repetition frequency α, receiver sample frequency β, and size of DSP memory buffer needed N in words. Other important sounder parameters shown in Figure 4.1 include the time bin resolution that sets the multipath component resolution, sounding pulse width that is calculated from the requested sounding bandwidth, the data collection time, and DDS binary and hexadecimal set words.

Figure 4.2 shows a SSTDSP sounder measurement of sounding pulse with duration 3 us and transmit pulse repetition frequency of 48 kHz. Figure 4.3 shows the processing steps that are required to produce the power delay profile from the raw channel data captured by the SSTDSP sounder. The first plot indicates the raw channel data that was captured,

the second plot illustrates the data after being smoothed by interpolation, the third plot shows the data normalized to one and ordered so the strongest multipath component is in the first time bin, and the fourth plot shows the data dicretized for metric calculations.

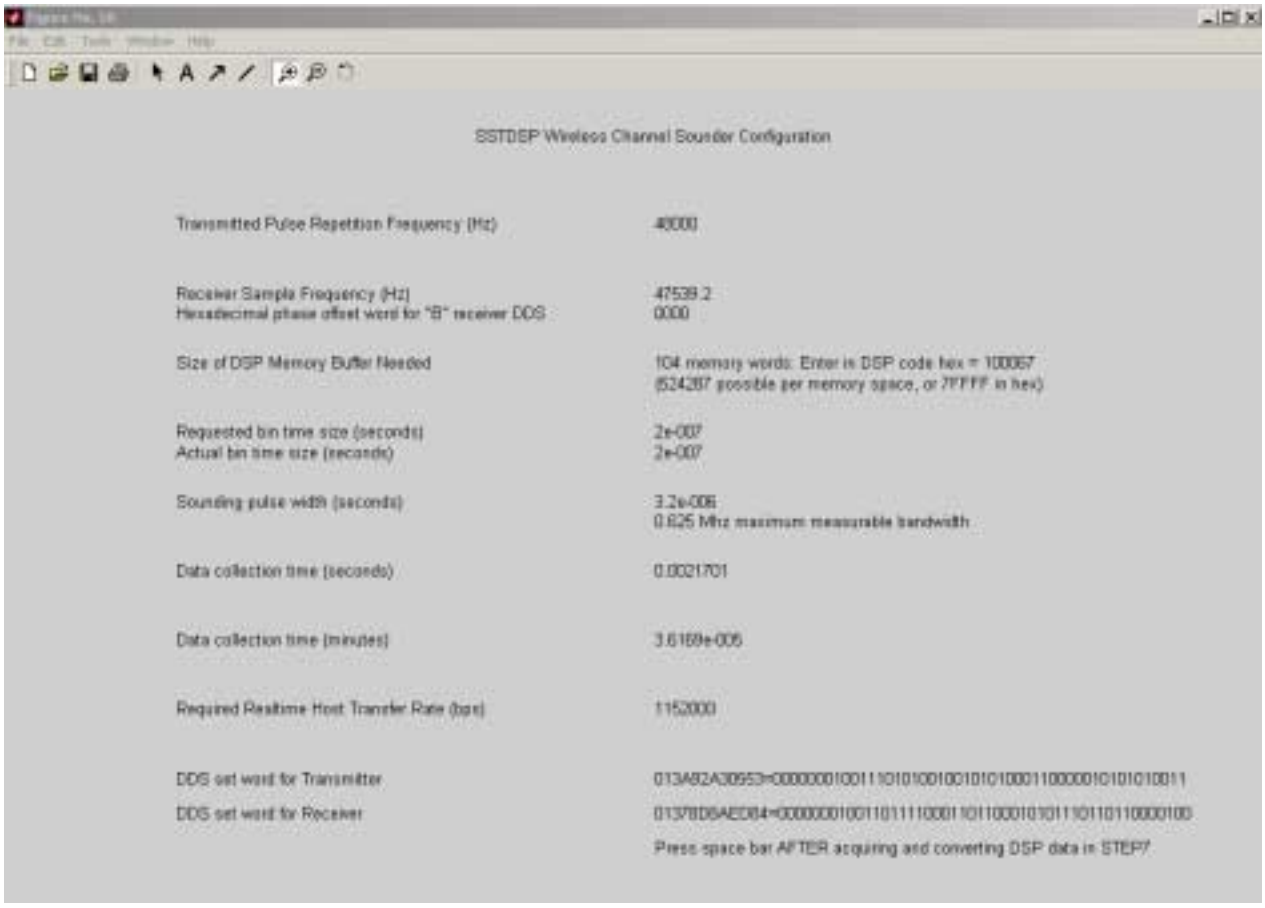


Figure 4.1: SSTDSP Wireless Channel Sounder Configuration verification output.

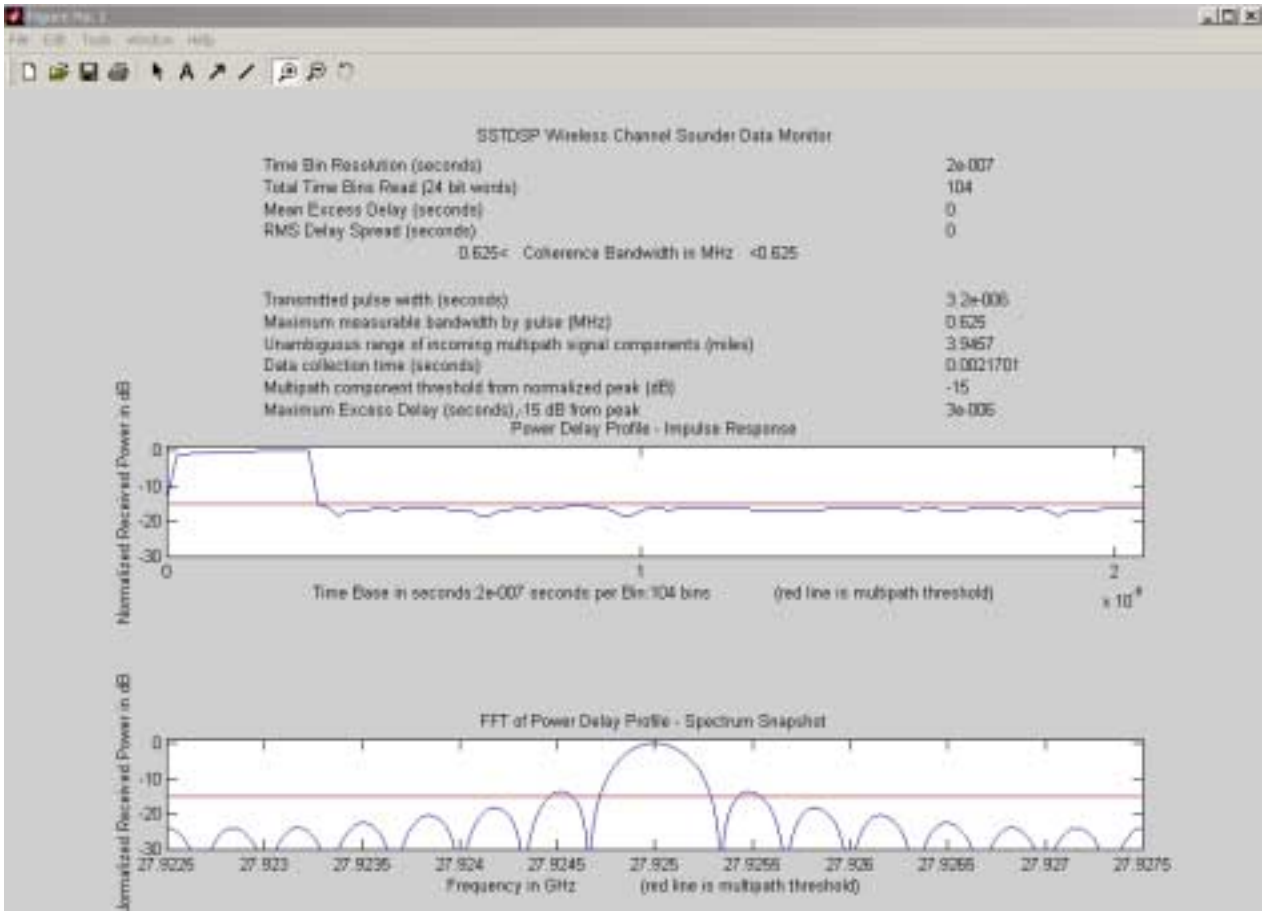


Figure 4.2: SSTDSP Wireless Channel Sounder Data Monitor verification output.

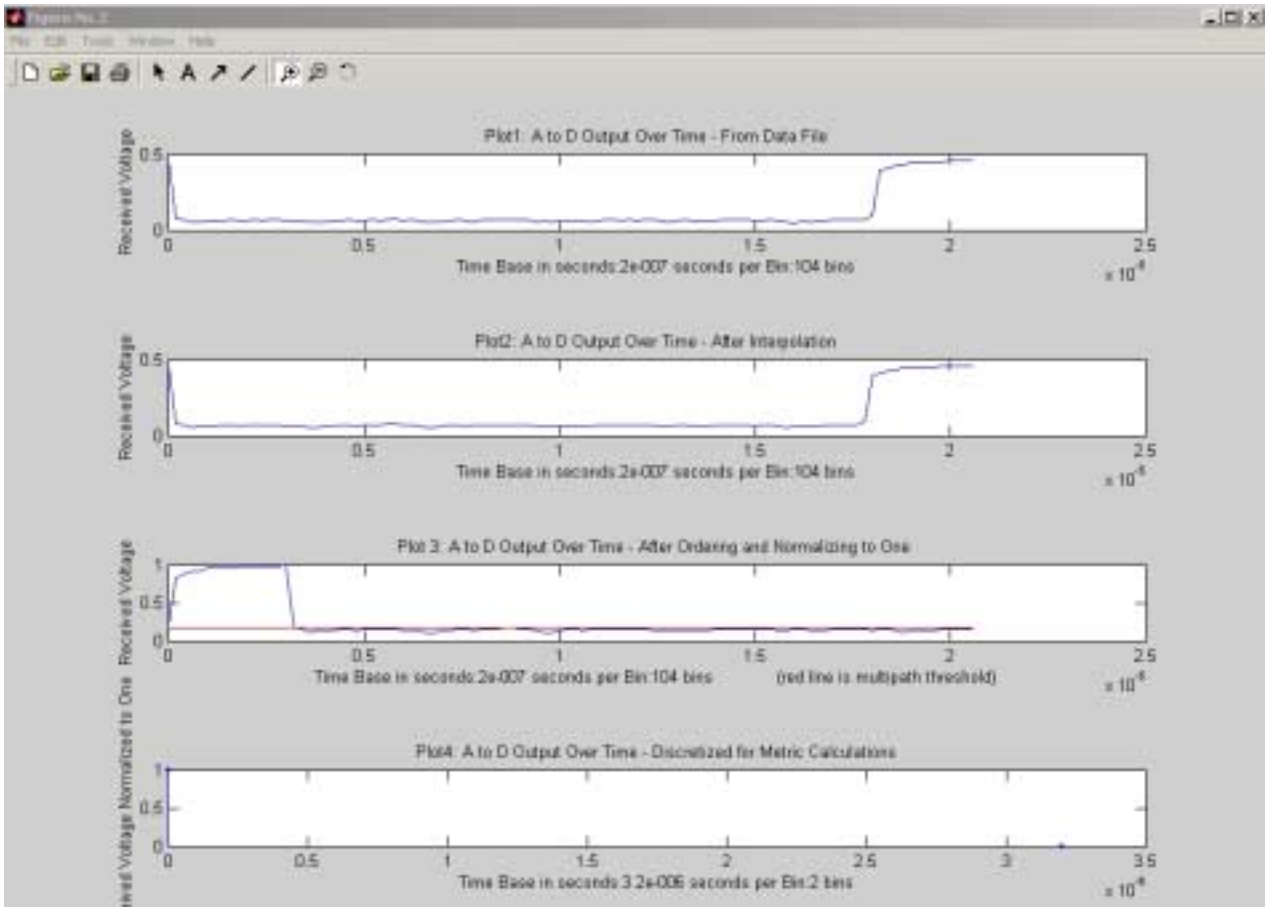


Figure 4.3: SSTDSP sounder data process steps verification output.

These SSTDSP sounder outputs verify the correct baseband operation of the SSTDSP sounder. The following baseband modules were designed and implemented by the author and are currently fully operational: TX Host Module, TX and RX Frequency and Location Modules, RX DSP Module, and the RX Host Module. All SSTDSP software algorithms were completed by the author and are fully functional, with the exception of the DSP memory data converter program that was implemented and verified by Todd Eshler. The baseband TX Pulsar Module was implemented and verified by Dr. Dennis Sweeney. The baseband RX IF Sampler Module implementation is currently being completed and verified by Dr. Sweeney.

Verification of the passband operation of the SSTDSP system in the LMDS band can not be completed until the RX IF Sampler design Dr. Sweeney implemented is fully

functional. Radio channel measurements without the RX IF Sampler and only the RX DSP ADC are not possible because of the 100 ns sampling window of the ADC and lack of peak impulse gain control and scaling to ensure the maximum dynamic range of the ADC is utilized. Any signal with duration close to or less than 100 ns can not be reliably digitized because of the large sampling window of the ADC. The RX IF Sampler mitigates that 100 ns section limitation by sampling and holding the passband value that has been downconverted and appears at the 1770 MHz IF output of the RX LMDS Radio Modules in less than a nanosecond. This permits the ADC, which has a relatively course sample window of 100 ns, to follow up and sample the value held during the remaining receive sample period.

4.3 Test procedure for taking SSTDSP sounder measurements

The following test procedure should be used to take SSTDSP channel measurements. The SSTDSP sounder measurement procedure consists of three major steps:

1. Setup and configuration
2. Data capture and storage
3. Data analysis and display

Appendix D outlines these three steps in greater detail.

4.4 Presentation and analysis of measurement data

This section presents and analyzes channel data that was captured and processed by the SSTDSP sounder.

Since the RX IF Sampler design implemented by Dr. Sweeney is not fully operational, radio channel measurements can not be taken due to course sampling window limitations inherent to the ADC, as previously discussed. Therefore no power delay profiles or

channel metrics are currently available for display in this thesis. Using an arbitrary waveform generator triggered by the TX Frequency and Location module to simulate multipath signals entering the receiver ADC was considered. However, the arbitrary waveform generator had internal circuitry limitations that appear to prevent it from generating its programmed multipath waveform from a precise external clock. The TX Frequency and Location Module that was disciplined to GPS could have been used as the stable frequency referense, however due to the internal circuitry limitations of the arbitrary waveform generator being used, this option of generating multipath channel data for the ADC to read was not pursued.

The following section discusses the current work in progress on the SSTDSP sounder.

4.5 Work in progress

4.5.1 Completion of RX IF Sampler and LMDS channel measurement

All parts of the baseband portion of the SSTDSP sounder are currently operational and their operation has been verified, with the exception of the RX IF Sampler Module that Dr. Sweeney is implementing. Once Dr. Sweeney completes the implementation of the RX IF Sampler, the RX IF Sampler's proper operation can be verified at baseband as follows.

The MATLAB Interface can be used calculate the correct SSTDSP sounding settings, given a transmit short pulse duration of 3 us, transmit pulse repetition period of 48 kHz, and time bin size of 200 ns. This long pulse duration and time bin can be easily captured by the RX DSP Module ADC, which has a course sample window of 100ns. The TX Frequency and Location Module can then be configured using the information provided in the SSTDSP sounder configuration window of the MATLAB Interface. The output of

the TX Frequency and Location Module can then be connected to the externally triggered variable width pulse generator in the lab which has been set to output a 2 volt peak pulse amplitude. The output of the pulse generator can be connected to the input of the RX DSP Module using coaxial cable and a properly set variable attenuator. The pulse generator and attenuator should be set so that maximum pulse amplitude is +2 volts, which can be verified using an oscilloscope. A SSTDSP measurement can then be taken and saved to disk. This can be done by copying and renaming the measurement screen capture files "output.jpg", "output2.jpg", and "output3.jpg" found in "D:\BroadbandSounder\captureddata".

Then the RX IF Sampler can be inserted in the system to verify it is correctly operating. The attenuator that is in between the transmit pulse generator and RX IF Sampler input should be initially configured with no attenuation. The initial RX IF Sampler gain setting should be set with no gain. As the attenuation is increased by the attenuator "channel", the RX IF Sampler Module gain should be gradually increased to ensure the peak pulse amplitude is being displayed through the RX Host Module display as +2 volts in the sounder data processing window within the MATLAB Interface.

To insert the RX IF Sampler Module in the SSTDSP sounder system, the coaxial cable and attenuator from the sounder transmitter should be connected to the input of the RX IF Sampler. The output of the RX IF Sampler should then be connected to the RX DSP Module, as indicated by the system block diagram in Chapter 3. Care should be taken to ensure the RX IF Sampler sample trigger input is properly connected to the "B" output of the RX Frequency and Location Module. The "A" output of the RX Frequency and Location Module should be connected to the sample trigger input of the RX DSP Module. Another SSTDSP measurement should be taken.

The measurement results taken with the RX IF Sampler Module in the system should be then be compared with the measurement results without the RX IF Sampler in the system. In both cases, only the 3 us pulse width should be displayed and the mean excess delay and rms delay spread should be zero because there is no multipath present in the coax

cable connection. In addition, the coherence bandwidth should be the sounding bandwidth of the short pulse, 667 kHz, and the maximum excess delay should be 3 us.

Once the RX IF Sampler Module operation has been verified at baseband, the entire SSTDSP system can be reconnected as indicated in the system block diagram in Chapter 3 and taken outside to verify channel data is being captured.

4.5.2 SSTDSP sounder measurement campaign

Once valid LMDS channel data has been taken by the SSTDSP sounder and verified, an extensive LMDS channel measurement campaign can be pursued. In addition, one of the more intriguing experiments that can be conducted using the SSTDSP sounder is to investigate whether a "bounce path" exists for LMDS systems and can be used to effectively extend LMDS coverage beyond light of sight. To answer this question first and SSTDSP measurement should be taken with the TX and RX LMDS Radio Modules positioned with line of sight paths. Then the TX and RX LMDS Radio Modules should be positioned so that line of sight communication is not present but a single "bounce path" exists between the TX and RX LMDS Radio Modules. The results of the line of sight SSTDSP measurement should then be compared with the results of the single "bounce path" measurement.

4.5.3 Future SSTDSP sounder upgrades and improvements

The SSTDSP sounder system has been designed in a modular fashion to facilitate sounder upgrades and allow rapid configuration in the field. Several improvements to the system may be pursued. Currently the TX Pulsar Module is designed to produce a fixed duration short pulse, which fixes the sounding bandwidth that can be measured by the SSTDSP sounder. This may be upgraded in the future to permit variable SSTDSP

sounding bandwidths. Currently SSTDSP measurements are taken manually by running the measurement scripts found in the TX and RX Host Module. This process can be automated by using external devices like an LMDS modem to initiate an SSTDSP sounder measurement using the SSTDSP measurement scripts, permitting the modem to query the LMDS channel and optimize its operating parameters such as coding rate based on the channel status. This upgrade is currently being developed by Virginia Tech as part of the NSF Digital Government Project at Virginia Tech which enables disaster relief teams to rapidly deploy broadband LMDS wireless communications networks. Finally, the current SSTDSP sounder could be further miniaturized by combining the baseband processing functionality on a single board design and repackaging the resulting hardware in a smaller form factor. This would permit the university to offer the SSTDSP channel monitoring technology to broadband wireless equipment manufacturers that were interesting in integrating the broadband wireless channel monitoring technology into their product offerings.

Chapter 5

Conclusion and Future Implications

This thesis presented the theoretical underpinnings, design, and implementation of the Sampling Swept Time Delay Short Pulse (SSTDSP) wireless channel sounder. The mechanics of LMDS signal propagation were explored as well as the theory behind wireless channel sounding. A historical perspective of channel sounding as well as a comparison of existing channel sounding methodologies was outlined. The SSTDSP channel sounding method was synthesized from these preexisting methods to provide optimal wideband wireless channel sounding performance at a low cost.

System requirements were established and a detailed tradeoff analysis pursued. A SSTDSP sounder architecture was presented at the systems level and then developed in detail for each module and algorithm within the modular SSTDSP system. A SSTDSP channel measurement test procedure was outlined along with information on how to calibrate and verify correct operation of the SSTDSP baseband processing system. Finally, current work in progress on the SSTDSP sounder, future planned improvements, and plans for a complete measurement campaign were outlined.

The specific contributions of the author included presentation of the new SSTDSP channel sounding method introduced by Sweeney *et.al* that reduces the cost, size, and data processing and archival requirements for wideband channel measurement, while retaining the resolution and accuracy. In addition the author designed the modular system architecture that realizes this new sounding method and developed the hardware and software modules that implement this new channel sounder architecture in a cost and size efficient way. Digital hardware and algorithms were implemented by the author, while RF components were either bought off the shelf or implemented by Dr. Dennis Sweeney.

The SSTDSP sounder has been designed in a modular fashion to permit flexible reconfiguration of the measurement system in the field based on channel measurement requirements and to facilitate rapid technology upgrades to the system. In addition, the baseband portion of the SSTDSP system has been designed to permit low cost built in test functionality to be added to LMDS systems. The SSTDSP sounder built in test functionality will be utilized in the LMDS system being developed by Virginia Tech for the NSF Digital Government program, which enables disaster relief teams to rapidly deploy broadband LMDS wireless communications networks using channel metrics and geographic information systems (GIS) applications. In addition, the channel metrics provided by the SSTDSP sounder will be used in the NSF Digital Government system to permit control of the error correction coding and networking protocols based on the status of the channel. This will ensure optimal performance of the ad-hoc broadband network.

Future applications of the SSTDSP system include more detailed LMDS channel measurement campaigns and investigating the existence of using an LMDS "bounce path" for communications. The implications of ultra wide band (UWB) signaling techniques and communication modulation formats could also be studied.

Appendix A

Module Interfaces, Operation, and Key Parameters

A.1 TX Host Control Module

Interfaces

The primary interfaces to the TX Host Module are listed in Table A.1 and displayed in Figure A.1 below.

Table A.1: TX Host Module interfaces.

Block	Inputs	User Interface	Outputs	User Interface
LAPTOP Control/Display	1. GPS Position Info 2. User Notes	DB9 Serial Bus Keyboard	1. GPS Control 2. GPS Sync Control 3. DDS Control 4. Graphical Interface	DB9 Serial Bus DB9 Serial Bus DB25 Parallel Bus Screen

Figure A.1: TX Host Module interface diagram

Operation and key parameters

The TX Host Module primarily functions as a host to the DDS and GPS control and monitoring applications. The laptop used in the Host Module should be powered on and booted into Windows. From Windows the user should launch scripts TX_STEP1 and TX_STEP2 found in the **"D:\BroadbandSounder\run"** directory. This will launch the DDS and GPS software packages.

The AD9852_54 Customer Evaluation Software permits the user to configure the DDS to output the correct pulse repetition frequency α. To configure the DDS, the user should load the file:

"D:\BroadbandSounder\applications\ad9852_54\startsoundertx.54s"

and input the correct "DDS set word for the Transmitter" indicated on the sounder configuration window in MATLAB, then press enter to register the change. The user should then save that setup under a new unique name, and then load that same setup. This ensures that DDS is set. Figures A.2 and Figure A.3 below illustrate the correct DDS software settings, less the correct frequency word input. Note that in Figure A.3, the amplitude should always be full scale and the comparator should NOT be powered down. The comparator is used to generate the necessary CMOS level square wave at frequency α.

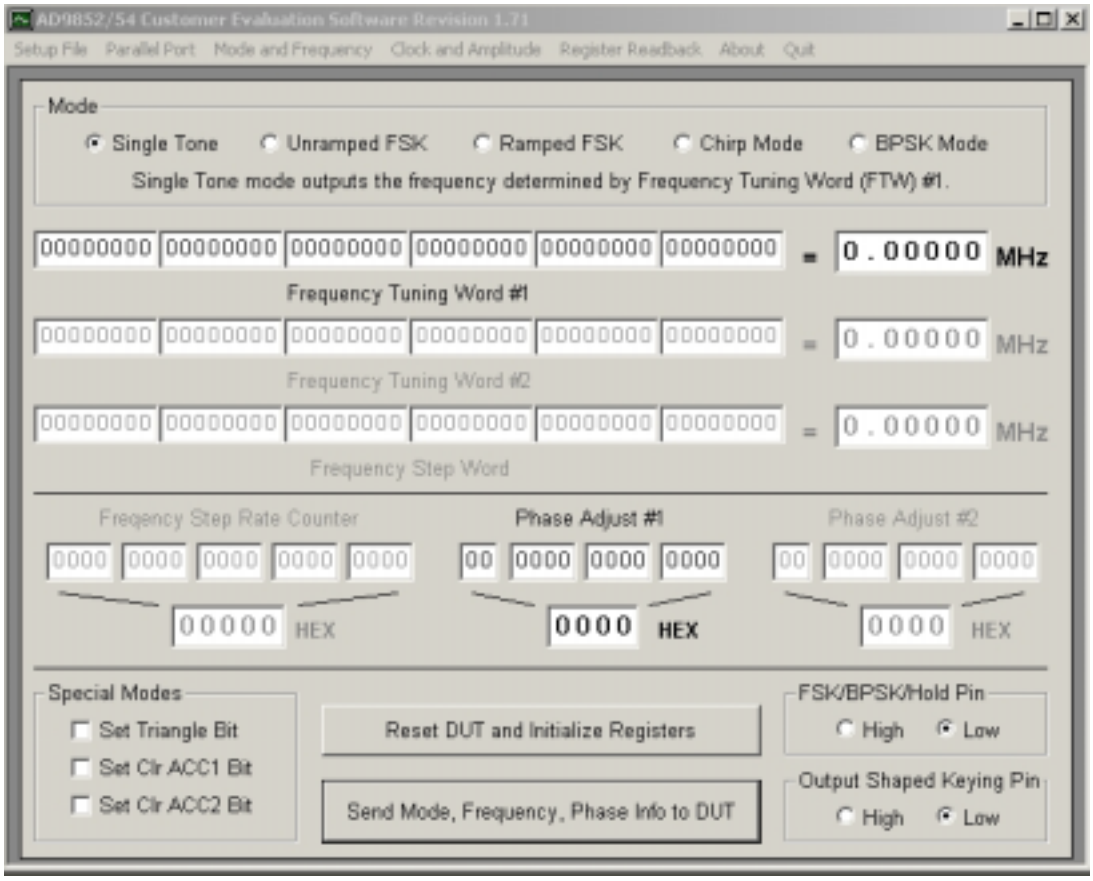


Figure A.2: DDS Configuration Screen One prior to customization.

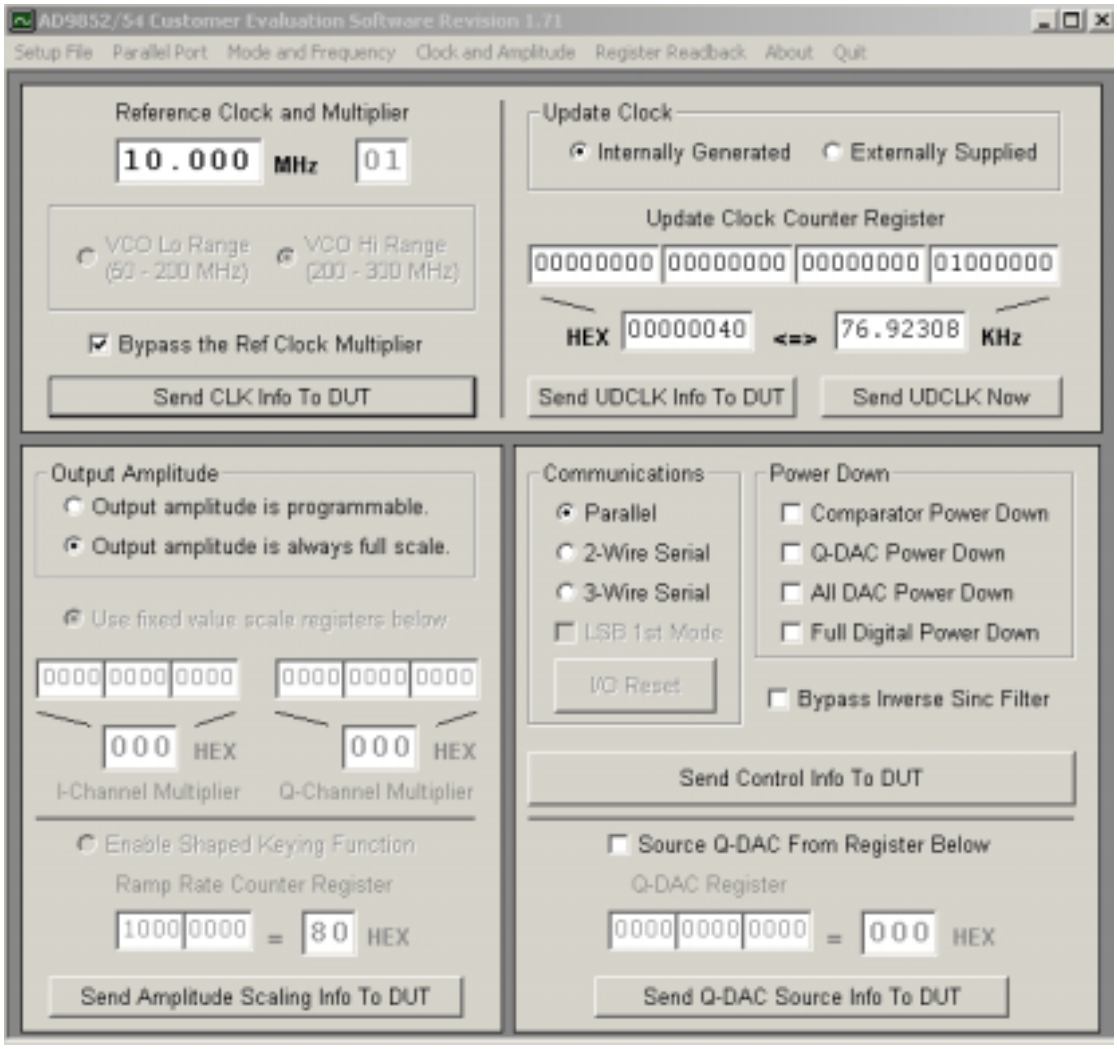


Figure A.3: DDS Configuration Screen Two prior to customization.

Launching the OutlookGPSPlus software enables the user to monitor incoming GPS signals, determine the currently acquired satellites, and record location and time measurements for each SSTDSP measurement. Figure A.4 below shows a screen shot of the OutlookGPSPlus software, which indicates the received signal strengths and messages of the current GPS satellites that the GPS unit is locked onto. In addition, a precise ones second clock is provided along with visualization of the satellite orbits.

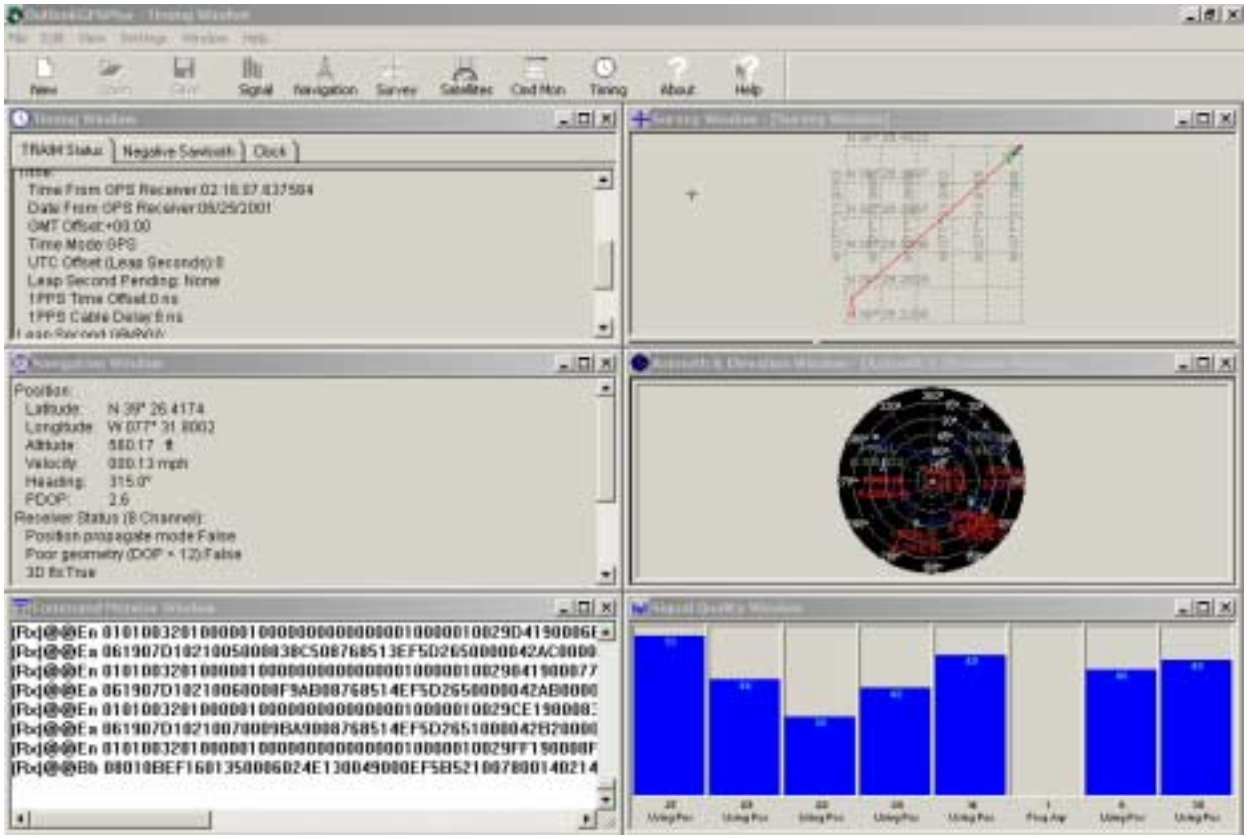


Figure A.4: OutlookGPSPlus software screen shot.

A.2 TX and RX Frequency and Location Modules

Interfaces

The primary interfaces to the TX and RX Frequency and Location Modules are listed in Table A.2 and displayed in Figure A.5 below. The only two differences between the TX and RX modules are the programmed DDS frequencies (α and β) and the additional β_δ frequency output "B" present in the RX module that is phase delayed from the β frequency output "A." This phase delay is needed in the receiver to ensure that the IF Sampler and ADC sample timing instants are properly synchronized. The sample trigger is received by the IF Sampler and ADC at the same time; however, the actual sample instants for the IF Sampler and ADC may be different due to different gate delays and circuit differences. Therefore, in order to ensure that both the IF Sampler and ADC are

sampling the same data, the sample triggers must be properly offset in time by a phase offset, hence the need for the "A" and "B" DDS phase offset sample trigger channels.

Table A.2: TX and RX Frequency and Location Module interfaces.

Block	Inputs	User Interface	Outputs	User Interface
GPS	1. GPS Control 2. GPS Antenna 3. Power	DB9 Serial Bus OSX RF connector + 12 Volt DC In	1. GPS Position Info 2. 1 PPSec Output	Custom interface Custom interface
GPS Sync	1. PPSec Output 2. 10 MHz Output 3. Power	Custom Interface SMA Coax In + 12 Volt DC In	1. 10 MHz Control 2. GPS Sync Monitor	Custom Interface DB9 Serial out
10 MHz Stable VCXO	1.10 MHz Control 2. Power	Custom Interface + 24 Volt DC In	1. 10 MHz Output	SMA Coax Out
DDS	1.10 MHz Output 2. DDS Control 3. Power	SMA Coax Out DB25 Parallel In + 12 Volt DC In	1.TX Frequency α or RX Frequency β 2. RX Frequency β_δ	SMA Coax Out SMA Coax Out

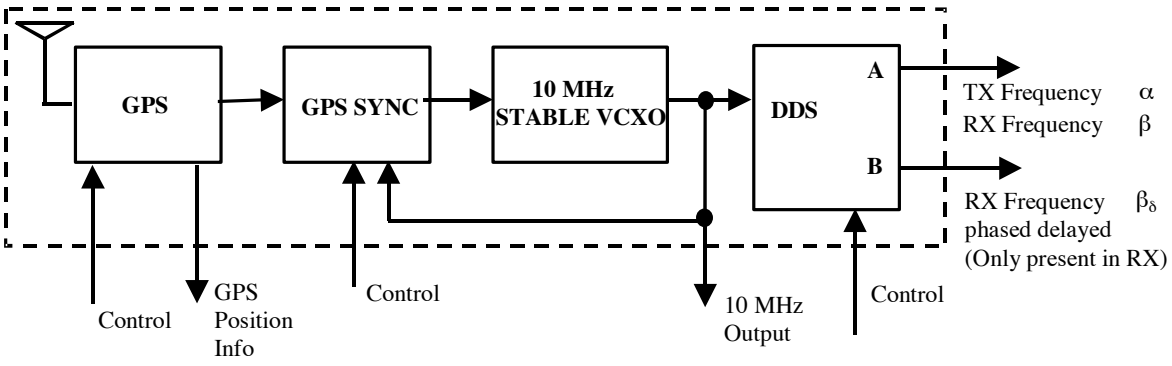


Figure A.5: TX and RX Frequency and Location Module interface diagram.

Operation and key parameters

The TX and RX Frequency and Location Modules are controlled by the TX Host Module and RX Host Module, respectively. Information on configuring the frequency and

location modules is provided in the "Operation and key parameters" within the TX Host Module and RX Host Module section.

Proper operation of the TX and RX Frequency and Location Modules requires powering on the module and placing the GPS antennas so that they can see most of the sky and therefore are in view of the GPS constellation. The GPS unit will automatically function on its own as will the 10 MHz VCXO and GPS Sync board; however, the DDS unit requires configuration each time a different SSTDSP measurement campaign is started. Information on DDS configuration is provided in the sections previously indicated.

A.3 TX Pulsar Module

Interfaces

The primary interfaces to the TX Pulsar Module are listed in Table A.3 and displayed in Figure A.6 below.

Table A.3: TX Pulsar Module interfaces.

Block	Inputs	User Interface	Outputs	User Interface
Pulsar	1.DDS Output 2. Power	SMA Coax In +60 Volt DC In	1. Pulsar Output	SMA Coax Out

Figure A.6: TX Pulsar Module interface diagram.

Operation and key parameters

The TX Pulsar Module requires no configuration by the user. Once the system is powered on and the DDS within the TX Frequency and Location Module is configured, the TX Pulsar Module will start transmitting pulses to the TX LMDS Radio module. The two most important parameters of the TX Pulsar Module include the short pulse of duration $\tau_{bb} = 2 / BW$ and pulse amplitude $A_{tx} = -25.5$ dBm input to the IF signal port of the TX LMDS Radio Modules.

A.4 TX LMDS Radio Module

Interfaces

The primary interfaces to the TX LMDS Radio Module are listed in Table A.4 and displayed in Figure A.7 below.

Table A.4: TX LMDS Radio Module interfaces.

Block	Inputs	User Interface	Outputs	User Interface
TX LMDS Radio Module	1.Pulsar Output 2. 10 MHz reference 3. Power	BNC SMA +-15 Volt DC +9.2 Volt DC +-5 Volts DC	1. LMDS Block A Wireless Spectrum	Built in high gain antenna

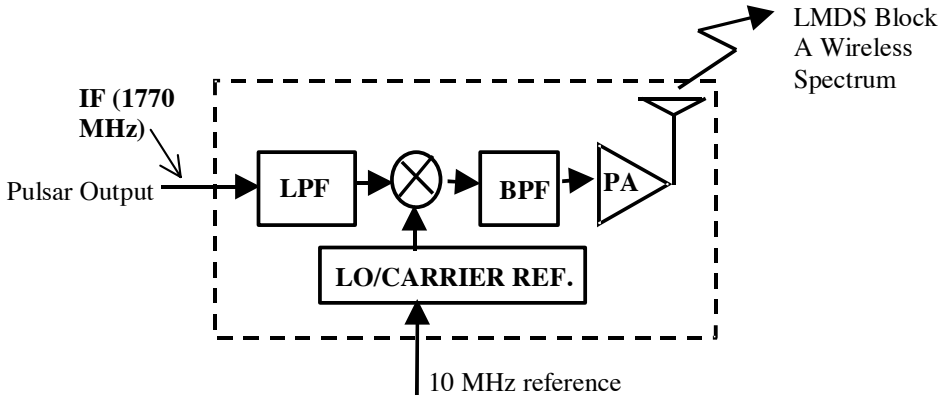


Figure A.7: TX LMDS Radio Module interface diagram.

Operation and key parameters

The TX LMDS Radio Module requires no configuration by the user. The system does allow customization of the local oscillator (LO) frequency, called an "agile LO," using dip switch settings provided by the manufacturer to permit changing the carrier frequency and spectrum under study. Once the system is powered on and directed along the transmission path of interest, it will up convert and amplify the short pulses that are supplied at the IF input. These pulses are then radiated by the high gain antennas built into the Spectrapoint radios. The key parameters of the TX LMDS Radio module are the system's filter bandwidth (BW), IF frequency and power output. The power output is a

integral part of the link budget and ensures the pulse amplitude is not completely attenuated by path loss, while at the same time not exceeding required limits.

A.5 RX LMDS Radio Module

Interfaces

The primary interfaces to the RX LMDS Radio Module are listed in Table A.5 and displayed in Figure A.8 below.

Table A.5: RX LMDS Radio Module interfaces.

Block	Inputs	User Interface	Outputs	User Interface
RX LMDS Radio Module	1.LMDS Block A Wireless Spectrum 2. 10 MHz reference 3. Power	Built in high gain antenna SMA +-15 Volts DC +12 Volts DC +5 Volts DC	1. RX IF Output	BNC

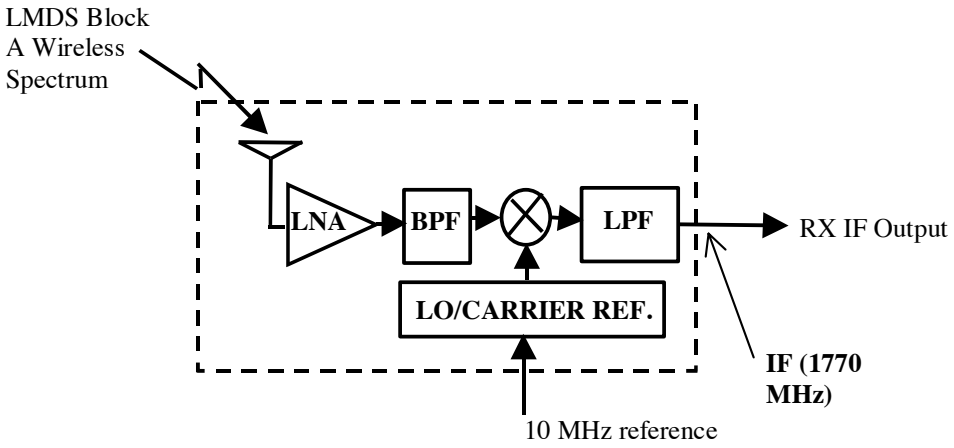


Figure A.8: RX LMDS Radio Module interface diagram.

Operation and key parameters

The RX LMDS Radio Module requires no setup by the user. The system does allow customization of the local oscillator (LO) frequency, called an "agile LO," using dip switch settings provided by the manufacturer to permit changing the carrier frequency

and spectrum under study. Once the system is powered on and directed along the reception path of interest, it will receive signal energy through the high gain receive antennas, amplify and down convert the LMDS channel frequency spectrum, and supply the resulting impulse response waveform to the IF output port. This impulse response is then captured by the IF Sampler through the SSTDSP process. The key parameters of the RX LMDS Radio module are the system's filter bandwidth (BW), IF frequency, receiver sensitivity, and the IF port power output. The IF port power output is a integral part of the link budget and determines how much additional gain is needed in the IF Sampler. The peak pulse amplitude at the output of the IF Sampler should match with the peak voltages swing of the ADC in the DSP Module.

A.6 RX IF Sampler Module

Interfaces

The primary interfaces to the RX IF Sampler Module are listed in Table A.6 and displayed in Figure A.9 below.

Table A.6: RX IF Sampler Module interfaces.

Block	Inputs	User Interface	Outputs	User Interface
RX IF Sampler Module	1.RX IF Output 2. RX Frequency β_δ 3. IF Sampler Gain Control 4. Power	SMA BNC Custom +-50 Volts DC In +15 Volts DC In -12 Volts DC In	1. IF Sampler Output	BNC

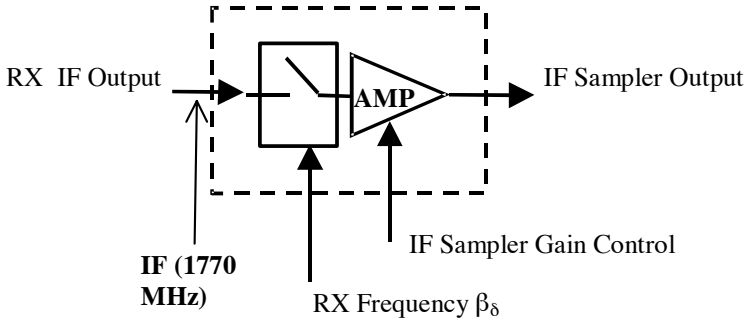


Figure A.9: RX IF Sampler Module interface diagram.

Operation and key parameters

The RX IF Sampler Module requires no configuration by the user. Once the system is powered on and the DDS within the RX Frequency and Location Module is configured, the RX IF Sampler Module will start sampling, holding, and amplifying the incoming impulse response waveform in time present at the RX IF Output of the RX LMDS Radio Module. The two most important parameters of the RX IF Sampler Module include the sample window duration $\tau_{samplewindow} \leq \Delta\tau$ and RX IF Sampler gain value (AMP) that ensures that the peak pulse amplitude corresponds to the maximum +- 2 volt swing at the input of the ADC in the DSP Module.

A.7 RX DSP Module

Interfaces

The primary interfaces to the RX DSP Module are listed in Table A.7 and displayed in Figure A.10 below.

Table A.7: RX DSP Module interfaces.

Block	**Inputs**	**User Interface**	**Outputs**	**User Interface**
RX DSP Module	1.IF Sampler Output	SMA	1. Channel Data	Parallel Command Converter
	2. RX Frequency β	SMA		JTAG
	3. DSP Control	Parallel Command Converter		DB9 RS-232
				Host Port
	4. Power	+-12 Volt DC In		

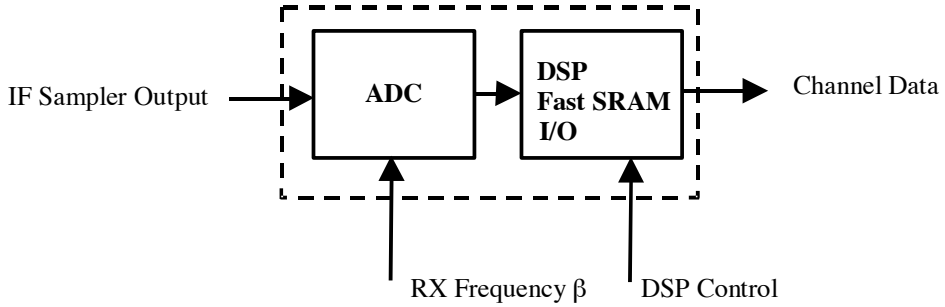


Figure A.10: RX DSP Module interface diagram.

Operation and key parameters

The RX DSP Module is controlled by the RX Host Module. The Motorola 56300 ADS DSP Software Development package is used to initiate communications between the RX DSP Module and the RX Host Module, download the binary executable file for the real time DSP algorithm "AtoD1.asm," and initiate a SSTDSP measurement. Other functions include monitoring the DSP's status and then upload the resulting Fast SRAM impulse response buffer to the RX Host Module when the SSTDSP measurement is complete.

Proper operation of the RX DSP Module requires powering on the module and following the SSTDSP measurement process. The DSP board will automatically function on its own once the acquire and convert data script is run. The key parameters for the RX DSP Module are the DSP MIPs rating, the maximum ADC sample rate, and the ADC dynamic range and input voltage swing. the size of the Fast SRAM memory buffer. Other key parameters include the 24 bit data path, the I/O peripherals data transfer rate, and internal DSP clock rate which bounds the maximum I/O data transfer rate to the RX Host Module. Information on DSP assembly program configuration is provided in the "Operation and key parameters" within the RX Host Module section.

A.8 RX Host Module

Interfaces

The primary interfaces to the RX Host Module are listed in Table A.8 and displayed in Figure A.11 below.

Table A.8: RX Host Module interfaces.

Block	Inputs	User Interface	Outputs	User Interface
LAPTOP	1. GPS Position Info	DB9 Serial Bus	1. GPS Control	DB9 Serial Bus
Control/Display	2. Channel Data	DB25 Parallel bus	2. GPS Sync Control	DB9 Serial Bus
	3. User Notes	Keyboard	3. DDS Control	DB25 Parallel Bus
			4. DSP Control	DB25 Parallel Bus
			5. AMP Control	Custom Interface
			6. Graphical Interface	Screen

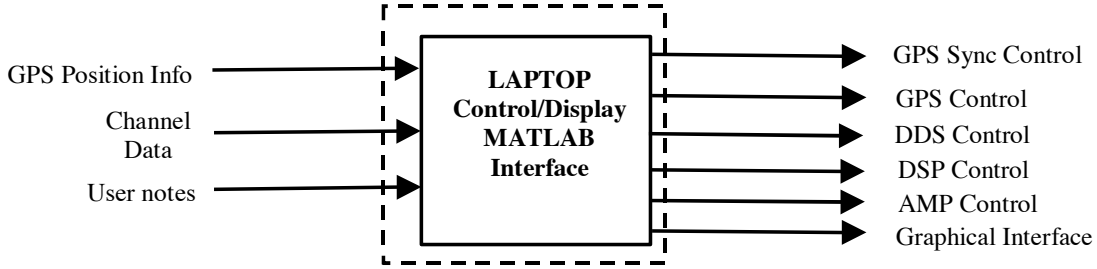


Figure A.11: RX Host Module interface diagram.

Operation and key parameters

The RX Host Module functions as a host to the DSP, DDS, and GPS control and monitoring applications. It also currently computes the power delay profile and channel metrics, providing screen shot capability and data logging. The workstation used in the Host Module should be powered on and booted into Windows. From Windows the user should launch scripts RX_STEP1 through RX_STEP8 found in the **"D:\BroadbandSounder\run"** directory, taking note to follow the correct steps in order and switch the parallel port switch between "A","B", and "C" when instructed to do so. This is necessary because the current RX Host Module workstation only has one working parallel port. This will launch the DDS and GPS software packages as well as the MATLAB Interface and the DSP interface software at the appropriate time.

The AD9852_54 Customer Evaluation Software permits the user to configure the DDS units to output the correct receiver sample frequency β and β_δ. To configure the both the "A" and "B" DDS units, the user should load the file:

"D:\BroadbandSounder\applications\ad9852_54\startsounderrx.54s"

and input the correct "DDS set word for the Receiver" indicated on the sounder configuration window in MATLAB, then press enter to register the change. For the "B" DDS, the user should also enter the necessary ADC-RX IF Sampler phase offset indicated by the MATLAB Interface to ensure proper sample instant synchronization. The user should then save that setup under a new unique name, and then load that same setup. This ensures that DDS is set. Figures A.12 and Figure A.13 below illustrate the correct DDS software settings, less the correct frequency word input and necessary phase offset for the DDS "B" board. Note that in Figure A.13, the amplitude should always be full scale and the comparator should NOT be powered down. The comparator is used to generate the necessary CMOS level square wave at frequency β and β_δ.

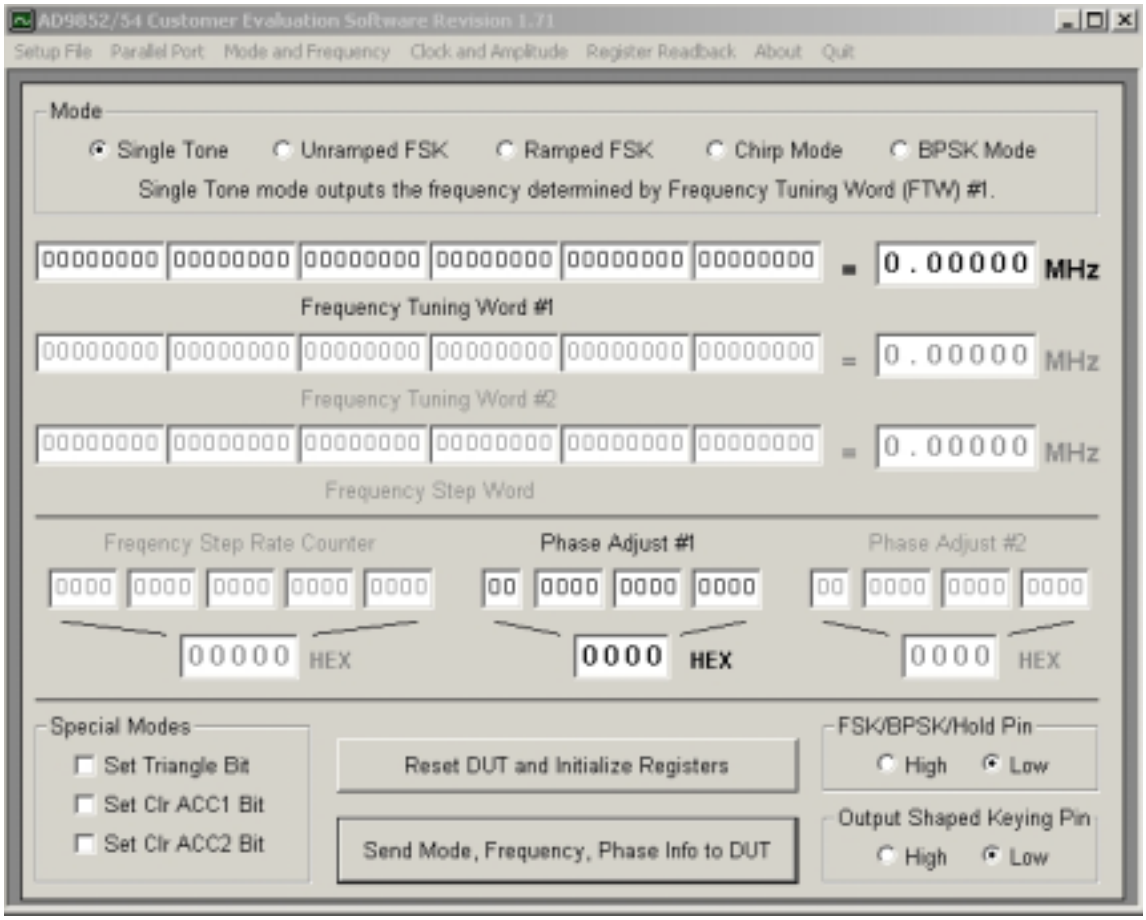


Figure A.12: DDS Configuration Screen One prior to customization.

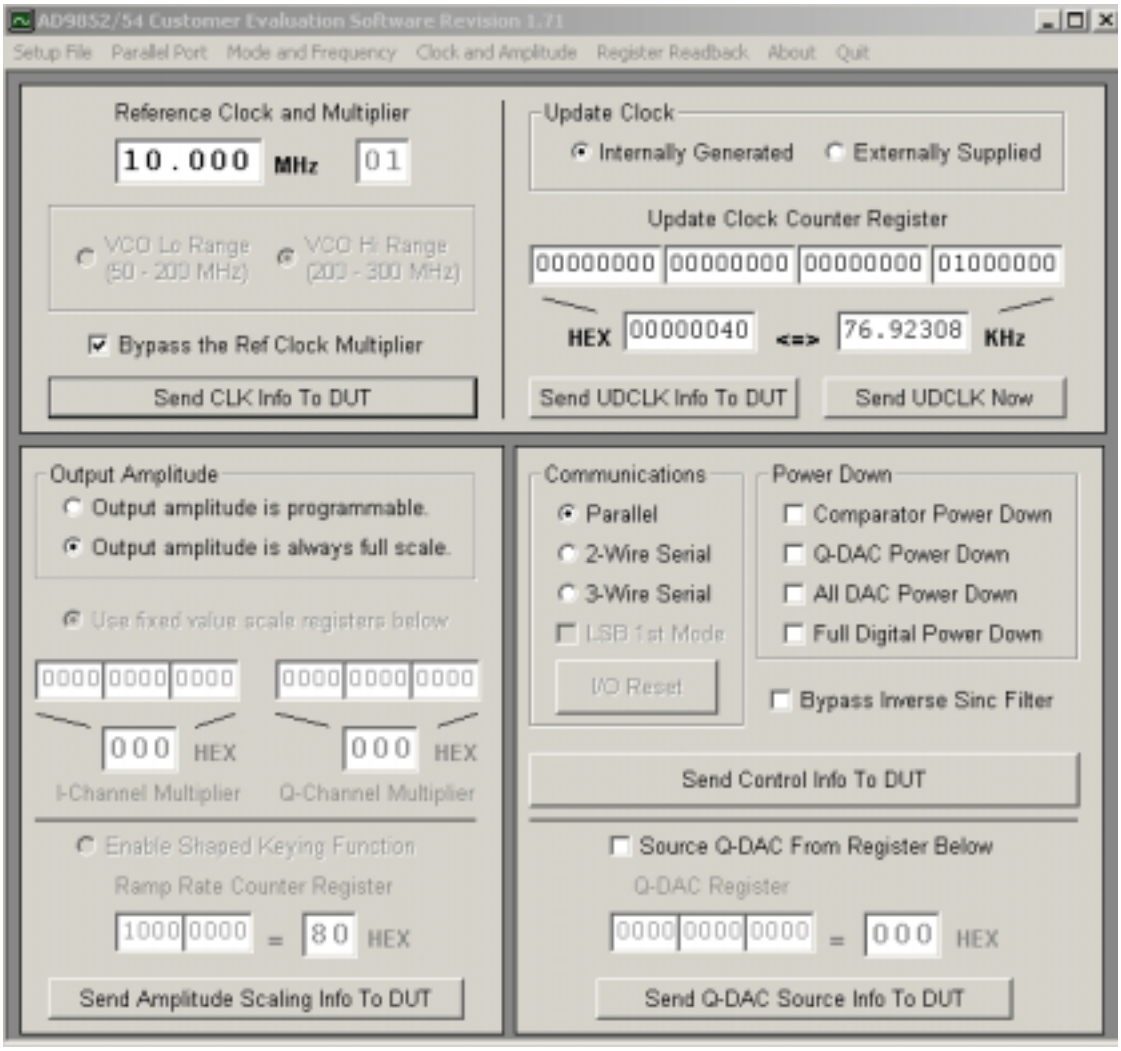


Figure A.13: DDS Configuration Screen Two prior to customization.

Launching the OutlookGPSPlus software enables the user to monitor incoming GPS signals, determine the currently acquired satellites, and record location and time measurements for each SSTDSP measurement. Figure A.14 below shows a screen shot of the OutlookGPSPlus software.

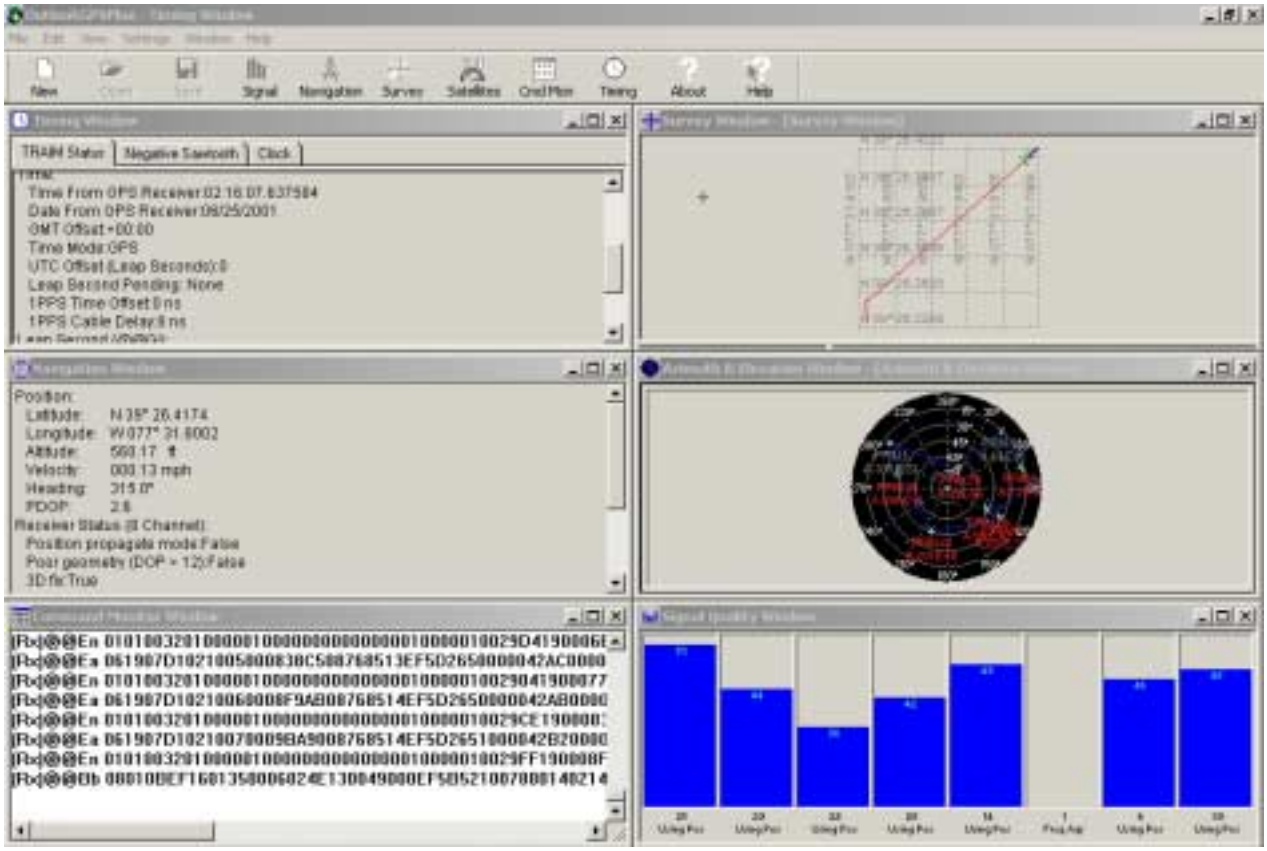


Figure A.14: OutlookGPSPlus software screen shot.

Launching the MATLAB Interface allows SSTDSP sounder configuration. The user can choose to automatically calculate SSTDSP sounder configuration settings using the "sounderparameters.m" file, or manually enter parameters and generate SSTDSP sounder configuration settings. Figures A.15 and A.16 below provide screen shots of this process.

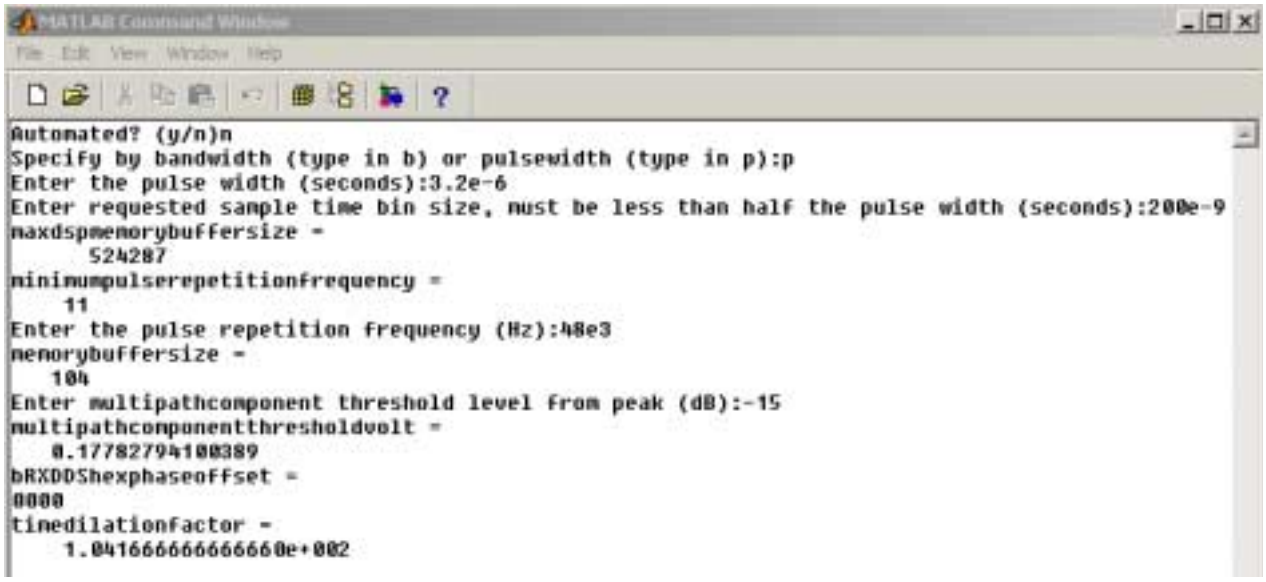


Figure A.15: Screen shot showing manual entry of measurement parameters.

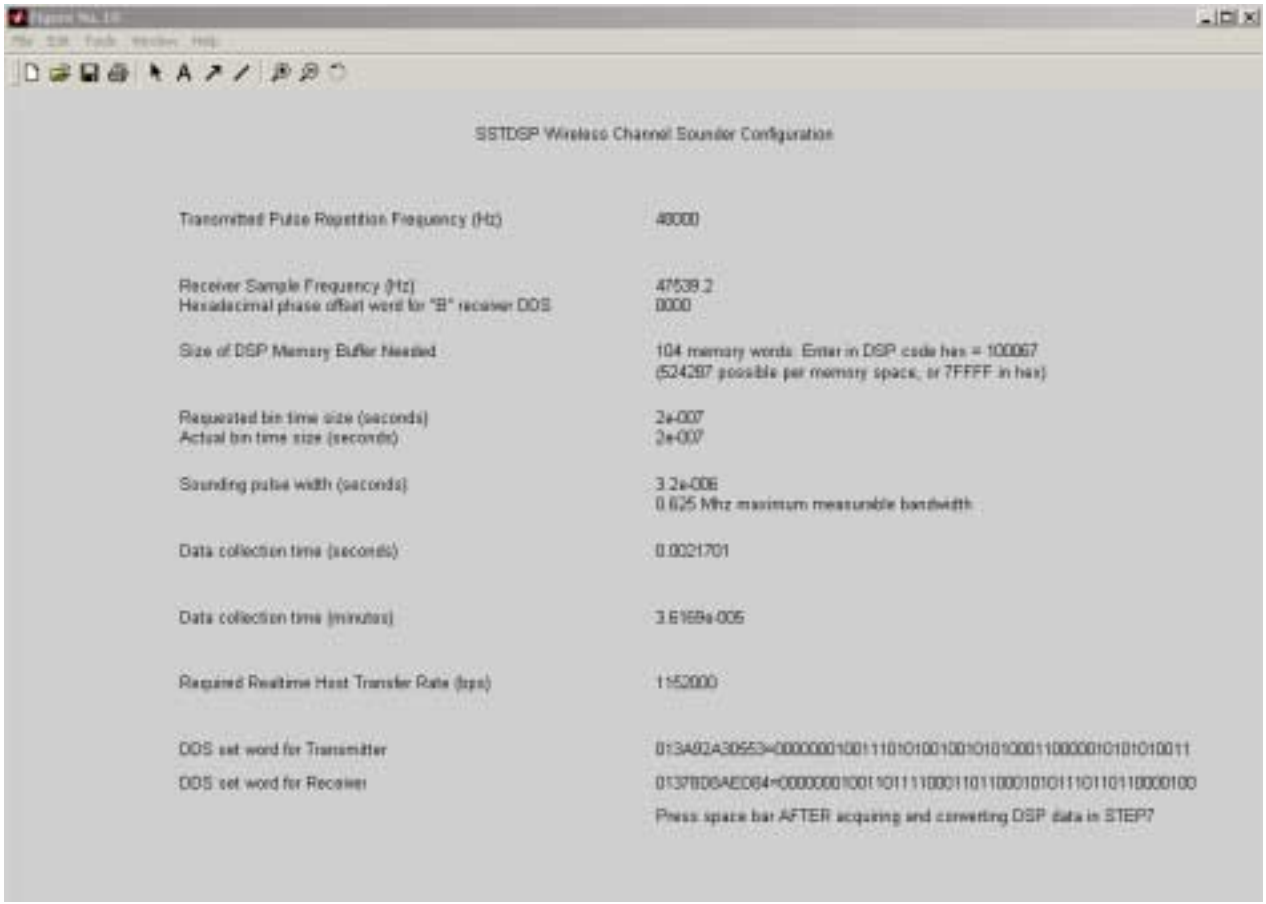


Figure A.16: Screen shot showing SSTDSP sounder configuration information.

After acquiring and converting data in Step 7, pressing spacebar in the MATLAB Interface initiates power delay profile and channel metric processing. Figures A.17 and A.18 provide screen shots of the resulting output. Detailed discussion of the various SSTDSP algorithms is provided in the next section.

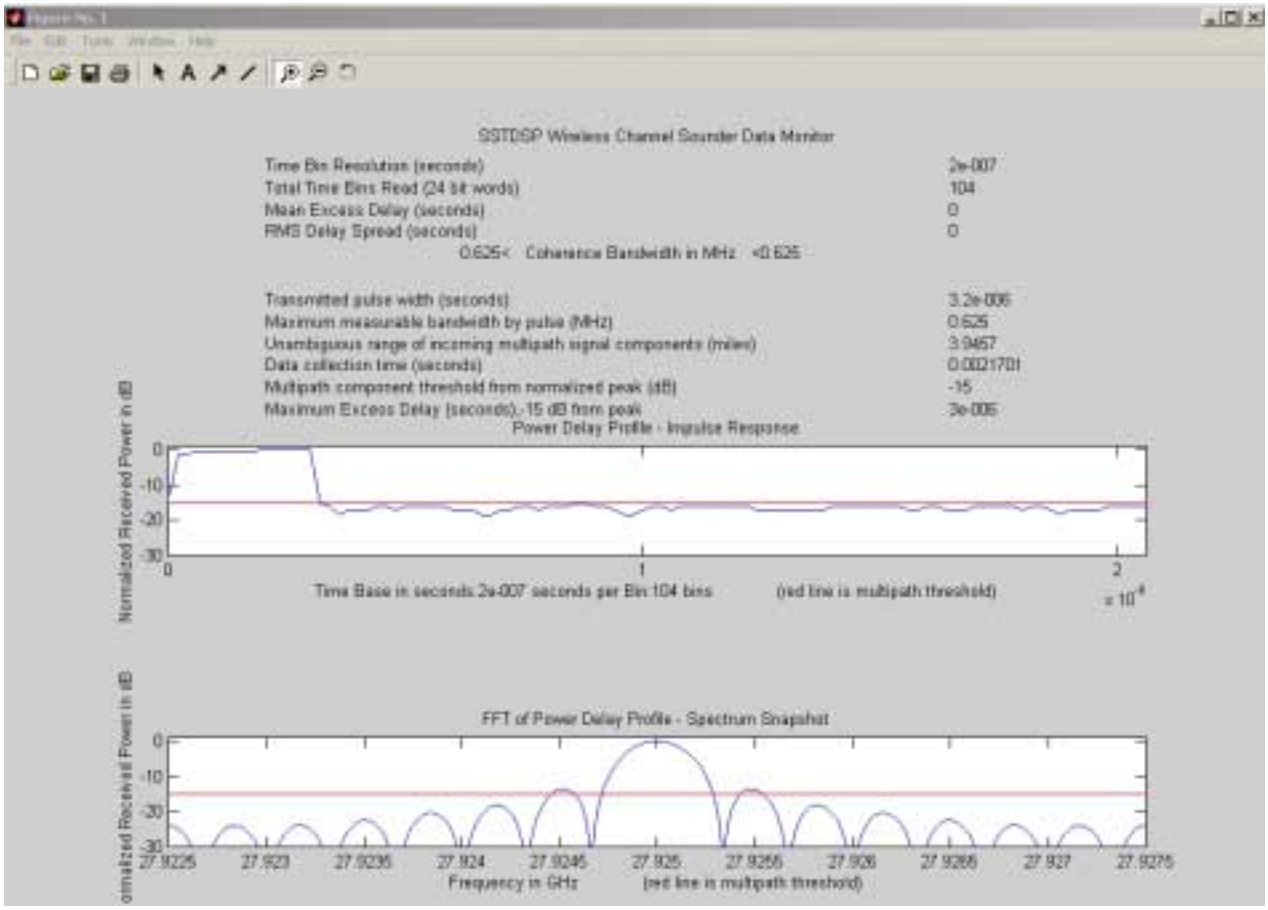


Figure A.17: Screen shot showing SSTDSP power delay profile and channel metrics.

Figure A.18: Screen shot showing SSTDSP impulse response processing steps.

Launching Steps 2 and 3 ensure that the DSP assembly and memory upload code is configured for the correct memory buffer size. In Step 2 the user must edit the DSP assembly code by inserting the correct hexadecimal memory address, save the file, and close the file. In Step 3 the user must edit the DSP control code by inserting the correct hexadecimal memory address, save the file, and close the file. Closing the file in Step 3 also recompiles and links the DSP code, which renders a binary file that can be transferred to the DSP for execution.

Launching the DSP interface software in step 7 loads the executable SSTDSP DSP binary code into the RX DSP Module, initiates an SSTDSP channel sounding measurement,

captures and stores the resulting digital impulse response. The digital impulse response is then uploaded to the RX Host Module, which then converts the memory buffer data to the required binary format required to import the memory buffer into MATLAB.

A.9 TX and RX Power Module

Interfaces

The primary interfaces to the TX and RX Power Modules are listed in Table A.9 and Table A.10 and displayed in Figure A.19 and A.20 below.

Table A.9: TX Power Module interfaces.

Block	Inputs	User Interface	Outputs	User Interface
TX Power Module	120 V AC	3 prong plug	+60 V DC	Cable to TX Pulsar Module
			+24 V DC	Cable to TX Frequency and Location Module
			+15 V DC	Cable to TX LMDS Radio
			-15 V DC	Cable to TX LMDS Radio
			+12 V DC	Cable to TX Frequency and Location Module
			+9.2 V DC	Cable to TX LMDS Radio
			+5 V DC	Cable to TX LMDS Radio
			-5 V DC	Cable to TX LMDS Radio

Table A.10: RX Power Module interfaces.

Block	Inputs	User Interface	Outputs	User Interface
RX Power Module	120 V AC	3 prong plug	+50 V DC	Cable to RX IF Sampler Module
			-50 V DC	Cable to RX IF Sampler Module
			+24 V DC	Cable to RX Frequency and Location Module
			+15 V DC	Cable to RX LMDS Radio
			-15 V DC	Cable to RX LMDS Radio
			+12 V DC	Cable to RX LMDS Radio, RX DSP Module, and RX Frequency and Location Module
			-12 V DC	Cable to RX IF Sampler Module and RX DSP Module
			+5 V DC	Cable to RX LMDS Radio

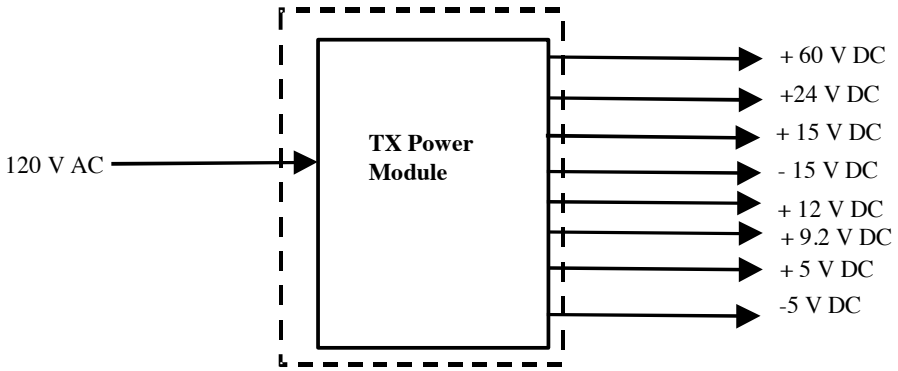


Figure A.19: TX Power Module interface diagram.

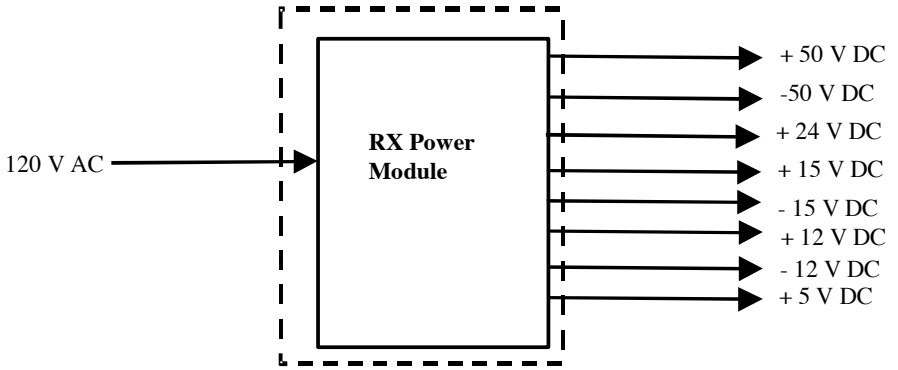


Figure A.20: RX Power Module interface diagram.

Operation and key parameters

The TX and RX Power Modules requires no configuration by the user. Powering on the power strip provides power to all modules. If certain hardware modules of the SSTDSP sounder appear not to be powered on, the user should verify the key voltage parameters are correct for that module.

Appendix B

Algorithm Language and Processing Flow Details

B.1 "processdata.m"

Targeted programming language

MATLAB.

Algorithm processing flow

1. Close all windows and specify SSTDSP sounder software directory.
2. Begin SSTDSP sounder configuration - user indicates automatic or manual entry of operating parameters.
3. If user indicated automatic entry of parameters, then the parameters are read in from the "sounderparameters.m" file, otherwise the user is prompted for parameters. Parameters include short pulse duration, time bin duration, transmit pulse repetition frequency, and multipath threshold in dB relative to peak pulse voltage normalized to 1 volt.
4. User entries are verified to ensure they comply with appropriate Nyquist criteria and DSP memory buffer limitations.
5. Algorithm then passes these parameters to "soundersettings.m" subrountine, which calculates a number of key SSTDSP operational settings, including the actual time bin size that can be supported by the mHz tunable DDS, the required receive sample frequency, the size of the DSP memory buffer needed and corresponding hexadecimal buffer end address, the data collection time in seconds and minutes, the required data rate to permit real time streaming of impulse response data from the DSP to the RX Host Module, the transmit DDS set word in hexadecimal and binary, and the receive DDS set word in hexadecimal and binary.

6. These parameters are then stored and transferred to the "sounderdataprocess.m" algorithm that calculates the power delay profile, mean excess delay, rms delay spread, coherence bandwidth upper and lower bounds, SSTDSP unambiguous range of incoming multipath signal components, and resulting spectrum snapshot that corresponds to the current channel condition.
7. The algorithm then turns on the zoom feature for all figures to allow the user to investigate the channel data during specific time periods and frequencies .

B.2 "startup.m"

Targeted programming language

MATLAB.

Algorithm processing flow

1. Upon launch of MATLAB, this file directs MATLAB to automatically execute "processdata.m" to permit rapid SSTDSP measurements to be taken.

B.3 "sounderparameters.m"

Targeted programming language

MATLAB.

Algorithm processing flow

1. The user opens and edits this file, entering required short pulse duration, time bin duration, transmit pulse repetition frequency, and multipath threshold in dB relative to peak pulse voltage normalized to 1 volt parameter values.
2. User then saves and closes this file, which will be read in by MATLAB Inferface when user indicates automatic parameter initilization.

B.4 "soundersettings.m"

Targeted programming language

MATLAB.

Algorithm processing flow

1. Close all windows and set screen values to ensure MATLAB window is properly positioned. Store required hexadecimal RX "B" DDS phase offset value.
2. Calculate and store the transmit pulse repetition period
3. Calculate the time dilation factor based on the transmit pulse repetition period and the request time bin size. The time dilation factor is the number of time bins in one transmit pulse repetition period
4. Set the estimated DSP memory buffer size in words equal to the time dilation factor.
5. Calculate the estimated receive sample frequency based on the estimated memory buffer size.
6. Round the receive sample frequency to the nearest mHz, which ensures the receive sample frequency is within the DDS's mHz tuning resolution.
7. Calculate the actual time dilation factor based on supported transmit pulse repetition frequency and receive sample frequency.
8. Set the required DSP buffer memory size in words equal to the time dilation factor and indicate the corresponding ending hexadecimal address in DSP memory. The starting hexadecimal address in DSP memory is always 10000 in hexa decimal.
9. Calculate the actual time bin size.
10. Set the DSP word size to 24 bits, calculate the data collection time in second and minutes, and calculate the required data transfer rate in bits per second (bps) needed to permit real time streaming of channel data from the DSP to the RX Host Module.
11. Set the DDS phase accumulator resolution to 48 bits and DDS system clock equal to 10 MHz. Calculate the required transmit and receive DDS set words in hexadecimal and binary.
12. Code is included to allow automatic editing of the DSP memory dump scripts, however this code is not used because MATLAB does not correctly save a new line a

carriage return needed by the DSP script to automatically execute within the debugger.

13. Open SSTDSP configuration figure and position correctly on screen.
14. Store and display the following parameters and labels: transmit pulse repetition frequency, receive sample frequency, hexadecimal phase offset for RX "B" DDS, size of DSP memory needed in decimal and hexadecimal, requested time bin size, actual time bin size, transmit short pulse duration, maximum sounding bandwidth, data collection time in seconds and minutes, data transfer rate required to permit real time streaming of channel data to the RX Host Module, transmit DDS set word in hexadecimal and binary, and receive DDS set word in hexadecimal and binary.
15. The configuration window is then saved in JPEG format to the file "D:\BroadbandSounder\captureddata\output1.jpg."
16. MATLAB pauses while user performs RX_STEP2 through RX_STEP7.
17. After RX_STEP7 is complete and the user presses space bar, the algorithm returns to "processdata.m"

B.5 "AtoD1.asm"

Targeted programming language

Motorola 56311 DSP assembly.

Algorithm processing flow

1. DSP memory space is defined.
2. User parameters are defined including memory buffer size, beginning hexadecimal memory buffer address, and ending hexadecimal memory buffer address.
3. DSP configuration parameters are set including DSP control registers, timer control registers, led on and off values, PCTL, AARO, AAR1, AAR2, AAR3, BCR, ADC 0 through 3 mapped register addresses, and the ADC status mapped register. PCTL stands for PLL control register, AAR stands for address attribute register, BCR stands for bus control register, and ADC stands for analog to digital converter.
4. Program memory is then defined for the main DSP routine.

5. Memory space and chip selects for the ADCs and SRAM are setup, all LEDs are turned off, and the DSP addressing mode is selected.
6. The DSP algorithm then branches to the acquiredatasubroutine.
7. The acquiredatasubroutine first stores the DSP memory buffer size, starting memory buffer address in the current memory location register, length of the memory buffer +1, and zeroes the accumulator count.
8. The databuffer loop is then entered. This loop continues until the defined impulse response DSP memory buffer is full, then returning to the main DSP program.
9. After entering the databuffer loop, the adsampling loop is entered
10. Upon entering the adsampling loop, the yellow LED is turned off.
11. The status register is then stored and trigger bit extracted and tested.
12. The yellow LED is then turned on to indicate that the DSP is checking to see if data is available from the ADCs.
13. If the trigger bit is zero, jump back up to the beginning of the adsampling loop, because the ADC is still sampling. Otherwise ADC has finished sampling and proceed. This assumes that the time between ADC samples is MUCH larger than the sample conversion time and therefore the ADC conversion will be done before the next sample, otherwise the code would also have to check to make sure the ADC conversion was done by inverting the DONE signal within the status register and anding it with the trigger.
14. Given that the ADC is done sampling, the DSP memory buffer index count register is incremented in the accumulator and stored to a register.
15. The length of the DSP memory buffer +1 is moved to a register and compared to the current DSP memory buffer index count register.
16. If they are equal then the DSP memory buffer is full and the databuffer loop is exited, returning control back to the main program.
17. If the DSP memory buffer is not full, then the yellow LED is turned off to indicate that data is available from the ADCs, the green LED is turned on to indicate data capture has begun, and the ADC data in the AD0 register is stored.
18. The ADC channel data value is then transferred to the memory location indicated by the current memory location register and the current memory location register is

incremented. Code is included to also stored the status register in the DSP memory buffer, however this is used only for debugging purposes.

19. The green LED is then turned off to indicate data capture is complete. The red LED is turned on to indicate that the DSP has now entered the wait loop and is waiting for the pulse repetition period to complete.
20. The status register is stored and the trigger bit is tested. Since the ADC sampler samples when the trigger is 0, if the trigger is still 1, the pulse repetition period is not over yet and the algorithm loops back and checks the trigger bit again. Once the trigger bit changes to 0 indicating the ADC is sampling, the wait loop is broken and the algorithm returns to the beginning of the adsampling loop to wait until the sample is complete and then store the next sample. Remember the databuffer loop is broken when the DSP impulse response memory buffer is full.
21. This process repeats N times until the the DSP impulse response memory buffer is full. Once the memory buffer is full, the databuffer loop is broken and the acquireddatasubroutine returns to the main program.
22. The main program turns on all three LEDs to indicate the SSTDSP measurement is complete and exits to the debugger to allow the debugger to export the memory buffer to a file.

B.6 "make.cmd"

Targeted programming language

Windows 2000 script.

Algorithm processing flow

1. The "AtoD1.asm" file is compiled to produce the file "AtoD1.cln" using the Motorola "g563c.exe" compiler.
2. The "AtoD1.cln" file is linked to produce the binary DSP executable file "AtoD1.cld" using the "dsplnk.exe" linker.

B.7 "capturedata.cmd"

Targeted programming language

Motorola "ads56300.exe" script.

Algorithm processing flow

1. The "ads56300.exe" application is configured to control the 56311 DSP.
2. The DSP is then reset and the "AtoD1.cld" binary executable is loaded and run on the DSP.
3. After the "AtoD1.cld" returns control to the "ads56300.exe" application, the application exits.

B.8 "exportdata.cmd"

Targeted programming language

Motorola "ads56300.exe" script.

Algorithm processing flow

1. The "ads56300.exe" application saves the DSP impulse response memory buffer to the file "D:\BroadbandSounder\captureddata\demo1.out" and exits.

B.9 "sounderdataconverter.cpp"

Targeted programming language

C.

Algorithm processing flow

1. Variables are defined: HEADERSIZE indicates the number of junk ASCII characters at the start of the "demo1.out" DSP memory dump file, JUNKSPACE indicates the number of junk bits in the 24 bit DSP word (only the upper 14 bits of the 24 bit word are valid for the 14 bit ADC), MASK defines an 8 bit binary mask, NEGATIVEBIT is a mask used to determine if the 24 bit value is two's complement negative,

NEGATIVEMASK is a mask used to sign extend the negative 24 bit numbers to negative 32 bits numbers, and FOREVER is defined to be one.

2. Input arguments to the main program are the number of command line arguments and the actual command line arguments in a string. There should always be three command line arguments: argv[0]="sounderdataconverter", argv[1]="demo1.out" which is an ASCII memory dump from the DSP memory buffer, and argv[2]="demo1.bin" which is a 32 bit valued memory buffer binary file to be read into MATLAB.
3. Several variables are declared by the main program. These include the integer value read from the "demo1.out" file, a 32 bit binary value to be written to the "demo1.bin" file, and a counter variable. In addition a junkheader variable used to remove the ASCII junk header at the beginning of "demo1.out", the input file stream, an output file stream, and a variable that determines if the memory value in the buffer is negative and must be two's complement sign extended are created.
4. The program checks to make sure that three command line arguments were received and if not returns an error and exits
5. The file indicated by the argv[1] (which should be "demo1.out") is opened.
6. If the input file does not exist or can't be opened, an error is returned and the program exits.
7. A binary output file with the name indicated in argv[2] (which should be "demo1.bin.") is opened.
8. If the output file can't be opened, an error is returned and the program exits.
9. The junk header information is removed from the front of the inputfile stream.
10. An infinite loop is entered that is broken when no more hexadecimal values can be read in, which occurs at the end of the "demo1.bin" data file.
11. The variable that indicates whether a value is negative is initialized to false and the first hexadecimal memory buffer value (max 24 bit) is read in the integer value variable. Hexadecimal values are separated by spaces in the DSP ASCII memory dump file "demo1.out."
12. If a valid hexadecimal value was not read indicating the end of the "demo1.out" file, the infinite loop breaks and the program exits, otherwise the algorithm proceeds.

13. The integer value is tested to see if it is negative and if it is the negative variable is set to true.
14. The integer is shifted so that the top 14 valid ADC bits are right justified. This inserts zeroes in the most significant bits.
15. If the value was negative, the value is sign extended by oring it with the NEGATIVEMASK.
16. Since the 32 bit binary value to be written to the "demo1.bin" file can only be written to 8 bits at a time, a loop is entered that masks, shifts, and stores the lower 8 bits of the value variable into the 32 bit binary value to be written to the "demo1.bin" file until the lower 32 bits of the value variable have been stored.
17. The 32 bit binary value is then written to the "demo1.bin" file and the loop repeats until all values have been converted from 24 bit signed binary numbers to 32 bit signed binary numbers to be read into MATLAB as integers, at which point the loop exits and the program exits.

B.10 "sounderdataprocess.m"

Targeted programming language

MATLAB.

Algorithm processing flow

1. Set screen values to ensure MATLAB window is properly positioned
2. Import 32 bit binary signed integer valued DSP impulse response memory buffer from file "D:\BroadbandSounder\captureddata\demo1.bin" into MATLAB interface as matrix Adata. Close "demo1.bin" file.
3. Calibrate the SSTDSP system to match real world parameters by calculating the maximum sounding bandwidth, scaling the Adata matrix to match the real world voltage waveform (shifted in time), and taking the absolute value of the scaled Adata matrix to produce Acalibrate. Given a known peak voltage input to the ADC, the resulting ADC quantization integer can be scaled to ensure the quantization values match with the real world input voltages to the ADC. This scaled waveform is then

full wave rectified to permit envelope detection. A time reference matrix is calculated that provides both a time index and real time bin vector values that correspond to the time axis on an impulse response plot or power delay profile. In addition, a time step matrix is defined that samples the original short pulse duration several times within the short pulse duration.

4. Smooth the Acalibrate impulse response matrix using level quantization per step and interpolation to eliminate noise. The Acalibrate matrix must be zero extended to produce a matrix that has a length that is a multiple of the step size in time bins. By finding the maximum value within a time step and writing that value to the entire step, a "stair-stepped" version of the signal is produced, eliminating unwanted noise spikes. This "stair-stepped" signal is then smoothed using interpolation between the peak points of each time step. The interpolation algorithm inserts NaN (not a number) values in the Asmooth matrix when out of range numbers are incountered. The matrix is searched from the first value forward and when an NaN value is encountered, it is replaced with the value of the previous matrix entry. If the NaN occurs in the first position it is replaced with a zero, since no valid previous value can be read. The resulting matrix is Asmooth.
5. Reorder Asmooth using the circular matched filter algorithm (CMF) created by the author. The CMF reorders the Asmooth matrix to produce the matrix Aorder. The CMF algorithm preorders the Asmooth matrix in four different preordering scenerios to ensure that in at least one of the scenerios the peak pulse waveform does not overlap between the first and last bin. These four preordering scenerio matrixes are each convolved with a matched waveform that has the same duration as the original short pulse, resulting in a matched filtering or correlation operation. A square pulse is currently being used as the matched waveform, however this can be changed to better match the transmit waveform. The scenerio with the maximum correlation value is chosen as the best preordering scenerio and the location of the peak correlation is noted. The maximum correlation value is used to properly set the fixed gain per SSTDSP measurement in the RX IF Sampler Module. Given the location of the peak correlation for the chosen preordering scenerio and knowledge of the original short pulse width, the chosen preordering scenerio is appropriately shifted so that the peak

pulse occupies the short pulse width time bin. The resulting value is the impulse response matrix Aorder.

6. Normalize Aorder so that peak pulse amplitude is scaled to one, producing Acenter.
7. Subtract the multipath component threshold voltage from Acenter to ensure only multipath components above the multipath threshold are positive valued, producing Ashift. This can be done because the multipath component threshold in dB is referenced to a normalized peak pulse amplitude of one.
8. Calculate the Amask matrix indicating which multipath components in Ashift are above the multipath component threshold (positive=1) or below the multipath component threshold (negative=0).
9. Multiply the Amask by the Ashift to produce the Aoffset matrix, which zeros all multipath components below the threshold.
10. Determine the maximum excess delay by searching from the end of Aoffset until a non-zero valued time bin is found, indicating the last multipath component above the threshold. Calculate the maximum excess time corresponding to that time bin location and truncate Aoffset to produce An, which only includes multipath components out to the maximum excess delay.
11. Subsample the An impulse response matrix using level quantization per short pulse width. The An matrix must be zero extended to produce a matrix that has a length that is a multiple of the short pulse width in time bins. By finding the maximum value within a short pulse width and storing that value to the a new discretization matrix, the Adiscretizedtopulsewidth matrix is created that has samples separated by a short pulse width in time, instead of the original time bin, which is smaller. The Adiscretizedtopulsewidth matrix is stored to the matrix A, which is then used to calculate the discretized power delay profile and channel metrics. The discretized power delay profile has been truncated to the maximum excess delay and discretized so that the separation between samples is a short pulse duration instead of the original time bin.
12. Calculate discretized power delay profile, which is used to compute the mean excess delay, rms delay spread, and coherence bandwidth upper and lower bounds.
13. Open SSTDSP summary figure and position correctly on screen.

14. Display SSTDSP sounder configuration information including actual time bin size, total time bins read from DSP memory file (which should match the time dilation factor and DSP memory buffer size previously indicated), mean excess delay, rms delay spread, coherence bandwidth bounds, short pulse duration, measurable bandwidth, unambiguous range of incoming multipath signal components, data measurement time in seconds, multipath component threshold from normalized peak in dB, and maximum excess delay.
15. Calculate and plot normalized power delay profile. The red line on the plot indicates the relation between the multipath threshold from normalized peak pulse used to determine the maximum excess delay and the normalized power delay profile.
16. Calculate and plot normalized spectrum response by taking Fast Fourier Transform (FFT) of time domain impulse response. The red line on the plot indicates the relation between the multipath threshold from normalized peak pulse and the normalized spectrum response.
17. Store channel metrics and export to a text file "D:\BroadbandSounder\captureddata\metric.out".
18. Open SSTDSP processing steps figure and position correctly on screen.
19. Plot calibrated DSP memory buffer impulse response Acalibrate versus the time matrix. This is the raw impulse response data buffer that was assembled by the DSP and has been scaled to match real world voltages. In addition the waveform has been full wave rectified using the digital processing absolute value function.
20. Plot smoothed impulse response Asmooth versus time matrix. This is the scaled impulse response data buffer that has been smoothed using level quantization and interpolation.
21. Plot ordered and normalized impulse response Acenter versus time matrix. This is the final impulse response matrix used to calculate the power delay profile and discretized power delay profile used for channel metric calculations. The red line on the plot indicates the relation between the multipath threshold from normalized peak pulse used to determine the maximum excess delay and the normalized impulse response.

22. Plot the discretized and truncated impulse response Aavg versus the ptime matrix. This final discretized and truncated impulse response matrix is used to calculate the discretized power delay profile and channel metrics.
23. The summary window and processing steps window are then saved in JPEG format to the file "D:\BroadbandSounder\captureddata\output2.jpg." and "D:\BroadbandSounder\captureddata\output3.jpg.", respectively.
24. MATLAB pauses while user observes and records necessary SSTDSP sounding channel measurement data.

Appendix C

Module Photos and Algorithm Source Code

C.1 Photo of SSTDSP Sounder test setup

Figure C.1: Photo of SSTDSP test setup in CWT Broadband Wireless Lab. Transmit cart on right and receive cart on left in both photos. Extra test equipment is shown for field test debugging including an arbitrary signal generator, oscilloscope, and variable short pulse width generator.

C.2 Photo of TX and RX LMDS Radio Module field testing

Figure C.2: Photo of TX and RX LMDS Radio Module field testing.

C.3 Photo of TX and RX Frequency and Location Module

Figure C.3: Photo of TX and RX Frequency and Location Module.

C.4 Photo of RX DSP Module

Figure C.4: Photo of RX DSP Module.

C.5 Photo of TX LMDS Radio Module

Figure C.5: Photo of TX LMDS Radio Module.

C.6 Photo of RX LMDS Radio Module

Figure C.6: Photo of RX LMDS Radio Module.

C.7 Photo of DDS board within TX and RX Frequency and Location Module

Figure C.7: Photo of DDS board within TX and RX Frequency and Location Module.

C.8 Photo of RX IF Sampler Module

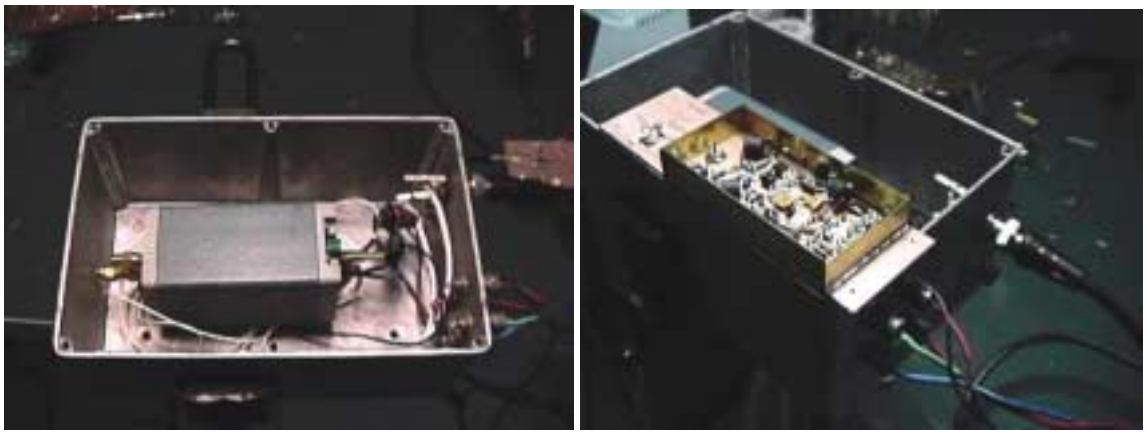


Figure C.8: Photo of RX IF Sampler Module.

C.9 Photo of TX Pulsar Module and 100 MHz test filter

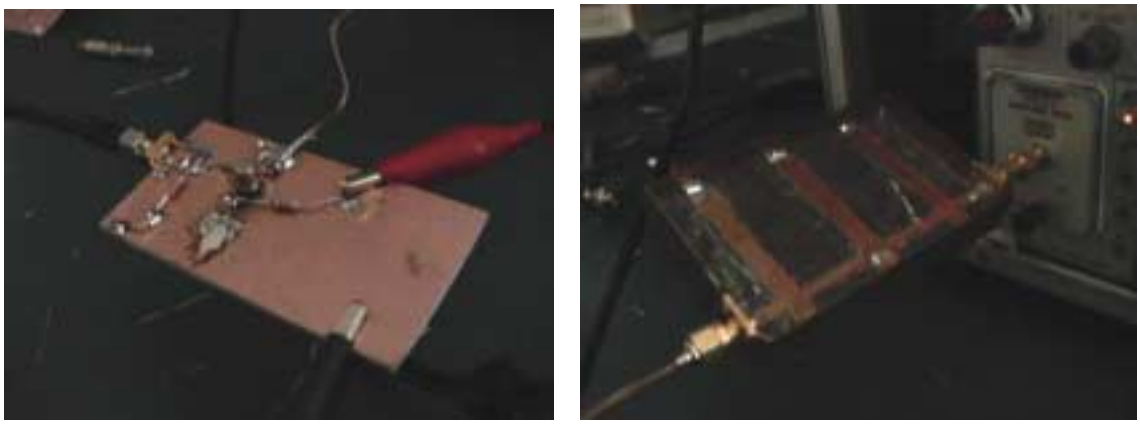


Figure C.9: Photo of TX Pulsar Module and 100 MHz test filter.

C.10 Photo of Output of 100 MHz test filter given TX Pulsar Module input

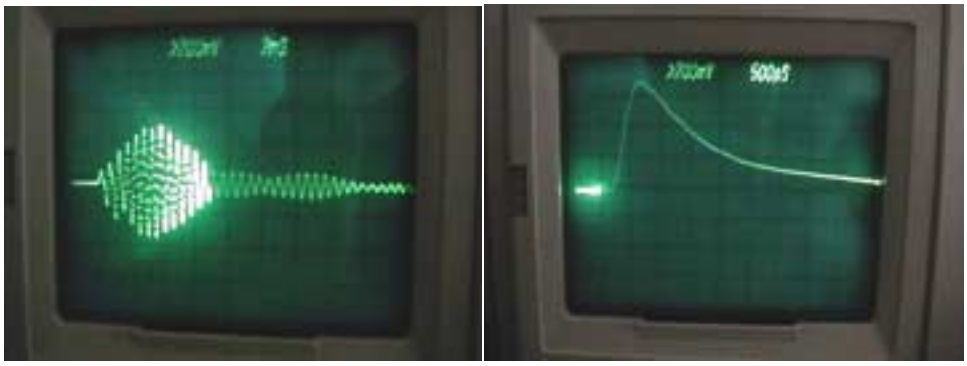


Figure C.10: Photo of Output of 100 MHz test filter given TX Pulsar Module input.

C.11 Photo of TX and RX Power Module

Figure C.11: Photo of TX and RX Power Module.

C.12 Source code for "processdata.m"

```
% Christian J. Rieser, MSEE Virginia Tech
% SSTDSP Wireless Channel Sounder
% MATLAB Interface for SSTDSP Sounder
% Version 4.0
% Processes digital data and provides a Graphical User Interface to the sounder
% 6/26/2001
% This function is used to process the Processes digital data and provides a Graphical User Interface to the sounder
% It prompts the user for all inputs, including pulsewidth or bandwidth, pulse repetition frequency, and multipathcomponentthreshold


close all % close all MATLAB windows
clc % clear MATLAB command window



%
% PARAMETERS
%

% specify drive for SSTDSP software and set path
cd d:\
path(path,'\BroadbandSounder\code\matlabcode');



%
% USER PARAMETER INPUT
```

```
%

% begin SSTDSP configuration calculation
statusbit=0;
while statusbit<1
  status=input('Automated? (y/n)','s');                                    % user indicated whether parameters will be entered manually or automatically

  if strcmp(status,'y')                                                    % read in "sounderparameters.m" file
    statusbit=2;                                                           % exit loop once parameters are read in
    [requestedbintimesize,pulsewidthsec, txpulserepetitionrate, multipathcomponentthresholddb]=sounderparameters; % read in parameters
  elseif strcmp(status,'n')                                                % allow user to enter parameters manually
    statusbit=2;                                                           % exit loop once parameters are manually entered
      nyquistratio=1;                                                      % ensure data entry occurs at least once
      bandwidthratio=1;                                                    % ensure data entry occurs at least once
      while nyquistratio<2 | bandwidthratio<2                              % tests to see if data has been entered correctly
      method=input('Specify by bandwidth (type in b) or pulsewidth (type in p):','s');
      if strcmp(method,'b')                                                % determine short pulse duration needed based on bandwidth to be studyed
            nyquistratio=3;                                                % set other test parameter to allow exit from loop
            bandwidthunderstudy=input('Enter the amount of bandwidth to be characterized (Hz):'); % enter sounding bandwidth
            pulsewidthsec=2/bandwidthunderstudy                            % calculate required short pulse duration
        requestedbintimesize=input('Enter requested sample time bin size, must be less than half the pulse width (seconds):');
        bandwidthratio=pulsewidthsec/requestedbintimesize;                 % determined if Nyquist is met
            if bandwidthratio<2
            disp('NYQUIST ERROR: Requested Time Bin Size must be less than 1/2 the Pulse Width');
            end
      elseif strcmp(method,'p')                                            % enter short pulse duration directly
            bandwidthratio=3;                                              % set other test parameter to allow exit from loop
            pulsewidthsec=input('Enter the pulse width (seconds):');
            requestedbintimesize=input('Enter requested sample time bin size, must be less than half the pulse width (seconds):');
            nyquistratio=pulsewidthsec/requestedbintimesize;               % determine if Nyquist is met
            if nyquistratio<2
```

```
            disp('NYQUIST ERROR: Requested Time Bin Size must be less than 1/2 the Pulse Width');
            end
        else
            disp('Please enter either the letter b or p');
        end
    end

    % calculate maximum DSP memory buffer size
    maxdspmemorybuffersize=hex2dec('17FFFF')-hex2dec('100000')

    % set required memory buffer size in words larger than max memory buffer size in words to allow entry to loop
    memorybuffersize=10e6;

            % calculate minimum transmit pulse repetition period
    minimumpulserepetitionfrequencyhz=round(1/(maxdspmemorybuffersize*requestedbintimesize))+1

    while memorybuffersize>maxdspmemorybuffersize                    % test to see if required memory buffer size in words fits in available DSP memory
        txpulserepetitionrate=input('Enter the pulse repetition frequency (Hz):');
        pulserepetitionperiod=1/txpulserepetitionrate;                % calculate transmit pulse repetition perioid
        memorybuffersize=round(pulserepetitionperiod/requestedbintimesize)    % calculate required memory buffer size in words
        if memorybuffersize>maxdspmemorybuffersize
            disp('MEMORY BUFFER TOO LARGE FOR DSP: Enter a higher pulse repetition frequency')
        end
    end
    multipathcomponentthresholddb=input('Enter multipathcomponent threshold level from peak (dB):');
        else
        disp('Please enter either the letter y or n');
        end
end

% set input impedence to 1 ohm since taking power delay profile after relative scaling of peak pulse to 1 volt, range 0 -> 1 volt
```

```
impedence=1;

% convert multipath threshold to voltage between 0 and 1, since maxvoltage in impulse response will be scaled to 1 volt
multipathcomponentthresholdvoltage=sqrt(impedence*10^(multipathcomponentthresholddb/10))



%
% SOUNDER CONFIGURATION
%

% sounder configuration algorithm
[TXF, RXF, MEMBUFFERSIZE, MEMBUFFERSIZEHEX, actualbintimesize, datacollectiontimesec, datacollectiontimemin, hostdatarateriquiredbps, txddssetwordhex, txddssetwordbin,
rxddssetwordhex, rxddssetwordbin]=soundersettings(requestedbintimesize, pulsewidthsec, txpulserepetitionrate);



%
% SOUNDER DATA PROCESS
%

% sounder power delay profile and channel metric processing algorithm
[A, bins, MXD, RDS, BcHighMhz, BcLowMhz,
Maximumdetectablebandwidthmhz,sounderrangemiles]=sounderdataprocess(actualbintimesize,pulsewidthsec,multipathcomponentthresholdvoltage,TXF,datacollectiontimesec,impedence);

% turn on zoon in all three plots
zoom(1,'ON')
zoom(2,'ON')
zoom(10,'ON')

% pause to allow user to see MATLAB messages if needed, clear, and option to automatically exit MATLAB
```

```
pause
clc
%quit
```

C.13 Source code for "sounderparameters.m"

```
function [requestedbintimesize,pulsewidthsec, txpulserepetitionrate, multipathcomponentthresholddb]=sounderparameters

% Christian J. Rieser, MSEE Virginia Tech
% SSTDSP Wireless Channel Sounder
% SSTDSP sounder parameter file
% Version 4.0
% Provides user defined SSTDSP sounder parameters
% 6/26/2001
% This function allows the user to set necessary SSTDSP sounder parameters for rapid measurements



%
% PARAMETERS
%

% requested impulse response time bin size, must be at least half the duration of the short pulse duration
requestedbintimesize=200e-9;

% requested short pulse duration
pulsewidthsec=3.2e-6;

% transmit pulse repetition frequency
```

```
txpulserepetitionrate=48e3;

% minimum multipath threshold used to determine maximum excess delay
multipathcomponentthresholddb=-15
```

C.14 Source code for "startup.m"

```
cd d:\broadbandsounder\code\matlabcode
processdata
```

C.15 Source code for "soundersettings.m"

```
function [TXF, RXF, MEMBUFFERSIZE, MEMBUFFERSIZEHEX, actualbintimesize, datacollectiontimesec, datacollectiontimemin, hostdatarateruiredbps, txddssetwordhex, txddssetwordbin,
rxddssetwordhex, rxddssetwordbin]=soundersettings(requestedbintimesize, pulsewidthsec, txpulserepetitionrate, bRXDDShexphaseoffset)

% Christian J. Rieser, MSEE Virginia Tech
% SSTDSP Wireless Channel Sounder
% DDS and DSP Configuration Algorithm
% Version 4.0
% Calculates the necessary information for Sounder DDS and DSP configuration
% 6/26/2001
% This function is used to calculate the necessary setting for the DDS and DSP in the sounder
% It requires inputs of requestedbintimesize and short pulse width in seconds and the txpulserepetitionrate in Hz

close all    % close all windows
%clc
```

```
%
% PARAMETERS
%

screenpercent=.90;     % ensure SSTDSP MATLAB GUI fits in screen

bRXDDShexphaseoffset='0000' % "B" RX DDS phase offset word in 4 digit hexadecimal



%
% CALCULATE CONFIGURATION
%

% calculate the transmit pulse repetition period
txpulserepetitionperiod=1/txpulserepetitionrate;

% store transmit pulse repetition frequency
TXF=txpulserepetitionrate;

% calculate the time dilation factor, this is the total number of time bins in one transmit pulse repetition period
estimatedtimedilationfactor=txpulserepetitionperiod/requestedbintimesize;

% set memory buffer size equal to the estimated time dilation factor
estimatedMEMBUFFERSIZE=estimatedtimedilationfactor;

% calculate the estimated receive sample frequency
estimatedRXF=(estimatedMEMBUFFERSIZE-1)/estimatedMEMBUFFERSIZE*TXF;
```

```
% ensure that the frequency offset between the transmit frequency and receive frequency is within the mHz tuning resolution of the DDS
RXF=round(estimatedRXF*1000)/1000;

% calculate the actual time dilation factor
timedilationfactor=TXF/(TXF-RXF)

% set the required DSP memory buffer size in words equal to the actual time dilation factpr
MEMBUFFERSIZE=timedilationfactor;

% determine the DSP memory buffer size in words in hexadecimal
MEMBUFFERSIZEHEX=dec2base(round(MEMBUFFERSIZE-1),16,5);

% calculate actual impulse response time bin size
actualbintimesize=txpulserepetitionperiod/MEMBUFFERSIZE;

% set DSP word size to 24 bits
dspword=24;

% calculate SSTDSP data measurement time in seconds
datacollectiontimesec=MEMBUFFERSIZE*txpulserepetitionperiod;
datacollectiontimemin=datacollectiontimesec/60;                    % calculate the SSTDSP data measurement time in minutes
hostdataraterequiredbps=MEMBUFFERSIZE*dspword/datacollectiontimesec;       % calculate the required data rate to permit real time streaming of channel data to the RX Host Module

N=48;                                                                  % 48 bit phase accumulator resolution
SYSCLK=10e6;                                                           % 10MHz GPS Disclipined Frequency Reference
tx=dec2bin((TXF*2^N)/SYSCLK);                                          % calculate TX DDS binary set word
txtemp=bin2dec(tx);                                                    % converts TX DDS binary set word to decimal
txddssetwordhex=dec2base(txtemp,16,12);                % converts TX DDS decimal set word to 12 digit hexadecimal
txddssetwordbin=dec2base(txtemp,2,48);                 % converts TX DDS 12 digit hexadecimal to 48 bit binary word required to set DDS
rx=dec2bin((RXF*2^N)/SYSCLK);                                          % calculate RX DDS binary set word
rxtemp=bin2dec(rx);                                                    % converts TX DDS binary set word to decimal
```

```
rxddssetwordhex=dec2base(rxtemp,16,12);                    % converts TX DDS decimal set word to 12 digit hexadecimal
rxddssetwordbin=dec2base(rxtemp,2,48);                     % converts TX DDS 12 digit hexadecimal to 48 bit binary word required to set DDS

% code that permits automatically editing DSP memory dump file
% can't use because MATLAB does not correctly save new line or carriage return needed for DSP script to be executed

%bufferlabel1=char('save x:100000..1');
%buffervalue1=MEMBUFFERSIZEHEX;
%bufferlabel2=char(' "\BroadbandSounder\captureddata\demo1.out"');
%bufferlabel=strcat(bufferlabel1,buffervalue1,bufferlabel2);

%delete '\BroadbandSounder\code\dspcode\rundsp2.cmd';
%fid = fopen('\BroadbandSounder\code\dspcode\rundsp2.cmd','w');
%fprintf(fid,'%s\r',bufferlabel);
%fclose(fid);



%
% DISPLAY
%

% open and position SSTDSP configuration window

fig10=figure(10);
screensize=get(0,'ScreenSize');
positionwindow=ceil([1 1 1 screenpercent].*screensize);
set(fig10,'Position',positionwindow);
axis off;

% store configuration window labels and values in text strings
```

```
tTXFlabel=char('Transmitted Pulse Repetition Frequency (Hz)');
tTXFvalue=num2str(TXF);                                                    % store transmit pulse repetition frequency

tRXFlabel=char('Receiver Sample Frequency (Hz)');
tRXFvalue=num2str(RXF);                                                    % store receive sample frequency

tRXphaseoffsetlabel=char('Hexadecimal phase offset word for "B" receiver DDS');
tRXphaseoffsetvalue=bRXDDShexphaseoffset;                                  % store hexadecimal phase offset for "B" RX DDS

tMEMBUFFERSIZElabel=char('Size of DSP Memory Buffer Needed ');
tMEMBUFFERSIZEvalue=num2str(round(MEMBUFFERSIZE));                         % store DSP buffer memory size
tequalslabel=char(' memory words: Enter in DSP code hex = 1');
tMEMBUFFERSIZEHEXvalue=MEMBUFFERSIZEHEX;                                   % store DSP buffer memory size in hexadecimal
tMEMBUFFERSIZEstr=strcat(tMEMBUFFERSIZEvalue, tequalslabel, tMEMBUFFERSIZEHEXvalue);
tMEMBUFFERSIZETOTALlabel=char('(524287 possible per memory space, or 7FFFF in hex)');

trequestedbintimesizelabel=char('Requested bin time size (seconds)');
trequestedbintimesizevalue=num2str(requestedbintimesize);                  % store requested bin time size

tactualbintimesizelabel=char('Actual bin time size (seconds)');
tactualbintimesizevalue=num2str(actualbintimesize);                        % store actual time bin size

tpulsewidthseclabel=char('Sounding pulse width (seconds)');
tpulsewidthsecvalue=num2str(pulsewidthsec);                                % store short pulse duration
tpulsewidthsecMhzfrequencyvalue=num2str(2/pulsewidthsec/1e6);              % store maximum sounding bandwidth
tpulsewidthsecMhzfrequencylabel=char(' Mhz maximum measurable bandwidth');
tpulsewidthsecMhzfrequency=strcat(tpulsewidthsecMhzfrequencyvalue, tpulsewidthsecMhzfrequencylabel);

tdatacollectiontimeseclabel=char('Data collection time (seconds)');
tdatacollectiontimesecvalue=num2str(datacollectiontimesec);                % store SSTDSP data measurement time in seconds
```

```
tdatacollectiontimeminlabel=char('Data collection time (minutes)');
tdatacollectiontimeminvalue=num2str(datacollectiontimemin);                    % store SSTDSP data measurement time in minutes

thostdataraterequiredbpslabel=char('Required Realtime Host Transfer Rate (bps)');
thostdataraterequiredbpsvalue=num2str(hostdataraterequiredbps);                % store data transfer rate required to stream real time channel data to RX Host Module

ttxddssetwordlabel=char('DDS set word for Transmitter');
ttxddssetwordhexvalue=txddssetwordhex;                                         % store TX DDS hexadecimal set word
tequalslabel=char('=');
ttxddssetwordbinvalue=txddssetwordbin;                                         % store TX DDS binary set word
ttxddssetwordstr=strcat(ttxddssetwordhexvalue, tequalslabel, ttxddssetwordbinvalue);

trxddssetwordlabel=char('DDS set word for Receiver');
trxddssetwordhexvalue=rxddssetwordhex;                                         % store RX DDS hexadecimal set word
tequalslabel=char('=');
trxddssetwordbinvalue=rxddssetwordbin;                                         % store RX DDS binary set word
trxddssetwordstr=strcat(trxddssetwordhexvalue, tequalslabel, trxddssetwordbinvalue);


% display SSTDSP configuration information on figure
title('SSTDSP Wireless Channel Sounder Configuration');
text(.01,.9,tTXFlabel);
text(.5,.9,tTXFvalue);
text(.01,.8,tRXFlabel);
text(.5,.8,tRXFvalue);
text(.01,.77,tRXphaseoffsetlabel);
text(.5,.77,tRXphaseoffsetvalue);
text(.01,.7,tMEMBUFFERSIZElabel);
text(.5,.7,tMEMBUFFERSIZEstr);
text(.5,.67,tMEMBUFFERSIZETOTALlabel);
text(.01,.6,trequestedbintimesizelabel);
```

```
text(.5,.6,trequestedbintimesizevalue);
text(.01,.57,tactualbintimesizelabel);
text(.5,.57,tactualbintimesizevalue);
text(.01,.50,tpulsewidthseclabel);
text(.5,.50,tpulsewidthsecvalue);
text(.5,.47,tpulsewidthsecMhzfrequency);
text(.01,.4,tdatacollectiontimeseclabel);
text(.5,.4,tdatacollectiontimesecvalue);
text(.01,.3,tdatacollectiontimeminlabel);
text(.5,.3,tdatacollectiontimeminvalue);
text(.01,.2,thostdatararequiredbpslabel);
text(.5,.2,thostdatararequiredbpsvalue);
text(.01,.1,ttxddssetwordlabel);
text(.5,.1,ttxddssetwordstr);
text(.01,.05,trxddssetwordlabel);
text(.5,.05,trxddssetwordstr);
text(.5,.0,'Press space bar AFTER acquiring and converting DSP data in STEP7'); % inform user to press space bar after completing step 7



%
% SAVE CONFIGURATION
%

% save current configuration window in JPEG format
saveas(fig10,'\BroadbandSounder\captureddata\output1.jpg','jpg');
pause
```

C.16 Source code for "sounderdataprocess.m"

```
function [Adata, COUNT, MXD, RDS, BcHighMhz, BcLowMhz, bandwidthoccupiedbypulseMhz,
sounderrangemiles]=sounderdataprocess(bintime,pulsewidthsec,multipathcomponentthresholdvoltage,TXF,datacollectiontimesec,impedence)

% Christian J. Rieser, MSEE Virginia Tech
% SSTDSP Wireless Channel Sounder
% Data Processing Algorithm
% Version 4.0
% Post processes data from sounder
% 6/26/2001
% This function is used to post process the data received from the sounder
% It requires inputs of the bin time resolution and pulse width in seconds
% multipath threshold voltage in volts, the transmitted pulse repetition frequency
% and the data collection time in seconds

%close all % close all MATLAB windows
%clc % clear MATLAB command window
%clf % clear all figures



%
% PARAMETERS
%

screenpercent=.90;
```

```
% indicate the number of samples taken per transmit short pulse duration
samplesperpulse=15;



%
% IMPORT DATA
%

% open and read in signed 32 bit int captured data from DSP memory file

fid=fopen('\BroadbandSounder\captureddata\demo1.bin','r','l');
[Adata, COUNT]=fread(fid,inf,'int32');
fclose(fid);



%
% CALIBRATE
%

% calculate the sounding bandwidth
bandwidthoccupiedbypulseMhz=2/pulsewidthsec/1e6;

% calibrate algorithm to map ADC values to voltages
% send in a know peak voltage and determine what the corresponding ADC int value is
%maxvalue=max(abs(Adata'))
calibratevoltage=.3;
calibratenumber=620429312;
calibrateratio=calibratevoltage/calibratenumber;

% scale incoming ADC int values to correct real world voltage on ADC and take ABS of impulse response
```

```
Acalibrate=abs(Adata'.*calibrateratio);

% calculate the time index array
Acalibratetimeindex=[0:length(Acalibrate)-1];

% calculate the real time array from the time index array
time=Acalibratetimeindex*bintime;



%
% SMOOTH
%

% calculate the length of a sample step within the time index array in bins
% each short pulse duration consists of pulsewidthsec/bintime bins
step=round(pulsewidthsec/bintime/samplesperpulse)

% calculate the subsampled time index array
Asmoothtimeindex=[0:step:length(Acalibrate)-1];

% zero extend the impulse response array to an even multiple of the step size in bins
Azeroextended=[Acalibrate zeros(1,step-mod(length(Acalibrate),step)+1)];

% find maximum value in the zero extended impulse response for each step
% and store that value to the entire step, this disregards fluctuations
% in amplitude due to noise and produces a "stair step" version of the signal

j=1;
for k=1:step:length(Acalibrate)
   Asmoothtemp(j)=max(Azeroextended(k:k+step));
```

```
  k;
      j=j+1;
end

Asmoothtemplength=length(Asmoothtemp)
Asmoothtimeindexlength=length(Asmoothtimeindex)

% interpolate between stair step points to produce a smooth envelope of the pulse
% the MATLAB interpolate function often produces NaN when a value is calculated that is out of range
% these NaN numbers must be removed from the array and replaced by valid signal estimations
Asmoothinterp=interp1(Asmoothtimeindex,Asmoothtemp,Acalibratetimeindex,'nearest'); % check this option for NaN conditions

% display the number of NaN numbers in the interpolation
disp('Number of NaN before correction')
sum(isnan(Asmoothinterp))

% display end of array - likely location for out of range estimations that produce NaN values
disp('Check end of interpolation for NaN')
Asmoothinterp(length(Asmoothinterp)-10:length(Asmoothinterp))

% remove NaN estimations from interpolation
% determine if an array element is NaN (0=NaN, 1=not NaN)
% sorting these values puts the NaN values and their corresponding index in Asmoothinterp
% at the front of the array truefalsevalues and locationof truefalsevalues
% truefalsevalues is an array mask that indicated if an element is NaN=0 or not NaN=1
[truefalsevalues,indexoftruefalsevaluesinAsmoothinterp]=sort(isfinite(Asmoothinterp));

% by multiplying the inverse of the truefalsevalues array mask by the indexoftruefalsevaluesinAsmoothinterp
% we can keep all the indexes of NaN valued array elements and zero the indexes of non NaN valued array elements
locationofnan=not(truefalsevalues).*indexoftruefalsevaluesinAsmoothinterp;
```

```
% we now search through the locationofnan matrix for non zero index locations and when we encounter a
% non zero index location indicating presence of an NaN, we replace the NaN with the value of the previous
% index location within Asmooth. If the locationofnan=1, then we store a zero, since there is no previous valid
% NaN corrected value to refer because we are correcting the Asmooth matrix from the first index forward.

Asmooth=Asmoothinterp;
i=1;
while locationofnan(i)>0
  if locationofnan(i)==1
    Asmooth(locationofnan(i))=0;
    disp('NaN at index 1, location = ')
    locationofnan(i)
  elseif locationofnan(i)>1
    Asmooth(locationofnan(i))=Asmooth(locationofnan(i)-1);
    disp('NaN at index greater than one, location = ')
    locationofnan(i)
  end
  i=i+1;
end

% display the number of NaN after NaN correction - should be zero
disp('Number of NaN aftercorrection')
sum(isnan(Asmooth))



%
% ORDER
%

% find the largest pulse within the pulse repetition period window memory buffer
```

```
% calculate the short pulse width in index time bins
pulsewidthintimebins=round(pulsewidthsec/bintime)

% calculate the matched waveform of length corresponding to the number of index time bins within a short pulse
% in this case all ones are used because we are assuming a square pulse
% this matched waveform can be customized once the exact short pulse waveform is known
matchedwaveform=ones(1,pulsewidthintimebins);

% confirm that the matched waveform length is the same
matchedwaveformlength=length(matchedwaveform)

% reorder Asmooth so that the peak pulse is first in the impulse response
% using a circular matched filter correlator to determine peak pulse location
Aorder=circularmatchedfiltercorrelator(Asmooth,matchedwaveform);

% verify that the reordering did not insert an NaN values
disp('Number of NaN after ordering')
sum(isnan(Aorder))




%
% QUANTIZE
%

% zero noise floor

% determine the maximum value within Aorder
maxAorder=max(Aorder);
```

```
% set Acenter equal to Aorder, Acenter is used to calculate the non discretized power delay profile
Acenter=Aorder/maxAorder;

% scale Aorder so that values range from 0 to 1 and subtract the corresponding relative multipath threshold voltage
Ashift=Acenter-multipathcomponentthresholdvoltage;

% calculate multipath component mask based on which values are positive, since only multipath component values
% that are above the relative multipath component voltage threshold are positive. we indicate multipath components
% above the multipath component threshold with a 1 and multipath components below the multipath threshold with a 0.
% these 1 and 0 values produce a mask that can be used to zero out multipath components below the multipath component threshold
Amask=sign(floor(sign(Ashift)+.5)+1);

% this mask is then multiplied by the Ashift to render only multipath components above the threshold and zero
% out all multipath components below the threshold
Aoffset=Ashift.*Amask;

% set Acenterthresholded equal to Aoffset, Acenterthresholded is used to calculate the discretized
% power delay profile and channel metrics
Acenterthresholded=Aoffset;

% store number of bins in Acenter, length of the original time matrix, and number of bins in Acenterthresholded
bins=length(Acenter)
binsintime=length(time)
binsinAcenterthresholded=length(Acenterthresholded)

% determine the maximum excess delay given the multipath threshold by starting from the end of the
% Acenterthresholded pulse repetition period and counting back until a non zero, i.e. valid multipath
% component is found. the resulting index provides the index of the maximum excess delay and last
% multipath component above threshold
i=0
```

```
while Acenterthresholded(length(Acenterthresholded)-i)==0
  i=i+1;
end

% calculate index of last multipath above threshold based on zero count.
if i==0
   lastmultipathcomponentabovethreshold=length(Acenterthresholded)-(i) % Check if (i-1) is needed
else
   lastmultipathcomponentabovethreshold=length(Acenterthresholded)-(i-1) % Check if (i-1) is needed
end

% calculate the time value of the maximum excess delay
maximumexcessdelay=time(lastmultipathcomponentabovethreshold-1)

% compute the new impulse response matrix, truncated to the last multipath component above threshold
An=Acenterthresholded(1:lastmultipathcomponentabovethreshold);

% zero extend the truncated impulse response array to an even multiple of the pulse width size in bins
Anextended=[An zeros(1,pulsewidthintimebins-mod(length(An),pulsewidthintimebins)+1)];

% find maximum value in the zero extended truncated impulse response for each pulse width
% and store that value to the entire pulse width, this disregards fluctuations
% in amplitude due to noise and produces a "stair step" version of the signal

j=1;
for k=1:pulsewidthintimebins:length(An)
  Adiscretizedtopulsewidth(j)=max(Anextended(k:k+pulsewidthintimebins));
  k;
      j=j+1;
end
```

```
% store Adiscretizedtopulsewidth to A for discretized power delay profile and channel metric calculations
A=Adiscretizedtopulsewidth;



%
% CALCULATE POWER DELAY PROFILE AND CHANNEL METRICS
%

% test vectors for channel metrics
%Acenterthresholded=[1 1 zeros(1,length(Acenter)-2)];
%A=[1 1];

%Acenterthresholded=[A(1:2) zeros(1,length(Acenter)-2)];
%A=A(1:2);

% determine max voltage in discretized truncated impulse response matrix
[maxvoltage,maxvoltagebin]=max(A)

% test vectors for channel metrics
%bintime=1e-6;
%maxvoltage=1;
%A=[.1 .3162 .3162 0 0 1 zeros(1,length(A))]; % mxd=4.38us, rds=1.37us, bc.5=146kHz

% algoritm to calculate channel metrics

% calculate the length of truncated impulse response
pbins=length(A);

% create equivalent index vector with bin size equal to pulse width
Abins=linspace(0,pbins-1,pbins);
```

```
% use equivalent index vector to calculate equilvalent time bin vector
ptime=Abins*pulsewidthsec;

% scale truncated impulse response to values between 0 and 1
Aavg=A/maxvoltage;

% calculate power delay profile
avgsq=Aavg.*Aavg/impedence;

% calculate channel metrics
sumnumT=sum(avgsq.*Abins);
sumdenomT=sum(avgsq);
T=sumnumT/sumdenomT;
MXD=T*pulsewidthsec                    % Mean Excess Delay
timesq=Abins.*Abins;
sumnumt=sum(avgsq.*timesq);
sumdenomt=sumdenomT;
t=sumnumt/sumdenomt;
RDS=sqrt(t-T^2)*pulsewidthsec          % RMS Delay Spread
BcLow=1/(50*RDS);
BcLowMhz=BcLow/1e6                             % Coherence Bandwidth Lower Bound (MHz)
BcHigh=1/(5*RDS);
BcHighMhz=BcHigh/1e6                       % Coherence Bandwidth Upper Bound (MHz)
if isinf(BcLowMhz) & isinf(BcHighMhz)
  BcLowMhz=bandwidthoccupiedbypulseMhz;
  BcHighMhz=bandwidthoccupiedbypulseMhz;
end
```

```
%
% DISPLAY SOUNDER RESULTS SUMMARY
%

% SSTDSP Sounder summary slide

% open and position SSTDSP configuration window
fig1=figure(1);

screensize=get(0,'ScreenSize');
positionwindow=ceil([1 1 1 screenpercent].*screensize);
set(fig1,'Position',positionwindow);



%
% sounder results and configuration information
%

subplot(5,1,1);
axis off;

% store data monitor window labels and values in text strings
tbintimelabel=char('Time Bin Resolution (seconds)');
tbintimevalue=num2str(bintime);                                    % store actual time bin size

tbinslabel=char('Total Time Bins Read (24 bit words)');
tbinsvalue=num2str(COUNT);                                         % store total time bins read in from DSP memory
                                                                   % should match size of DSP memory buffer needed
                                                                   % in sounder configuration window
```

```
tMXDlabel=char('Mean Excess Delay (seconds)');
tMXDvalue=num2str(MXD);                                          % store mean excess delay

tRDSlabel=char('RMS Delay Spread (seconds)');
tRDSvalue=num2str(RDS);                                          % store rms delay spread

tBcLowMhzvalue=num2str(BcLowMhz);                                % store lower bound of coherence bandwidth when frequency correlation = 0.9
tBclabel=char('<  Coherence Bandwidth in MHz  <');
tBcHighMhzvalue=num2str(BcHighMhz);                              % store upper bound of coherence bandwidth when frequency correlation = 0.5
tBc=strcat(tBcLowMhzvalue, tBclabel, tBcHighMhzvalue);

% display SSTDSP sounding results on figure
title('SSTDSP Wireless Channel Sounder Data Monitor');
text(.1,.9,tbintimelabel);
text(.8,.9,tbintimevalue);
text(.1,.7,tbinslabel);
text(.8,.7,tbinsvalue);
text(.1,.5,tMXDlabel);
text(.8,.5,tMXDvalue);
text(.1,.3,tRDSlabel);
text(.8,.3,tRDSvalue);
text(.3,.1,tBc);

subplot(5,1,2);
axis off;

% store data monitor window labels and values in text strings
tpulsewidthseclabel=char('Transmitted pulse width (seconds)');
tpulsewidthsecvalue=num2str(pulsewidthsec);                      % store transmited short pulse duration

tbandwidthoccupiedbypulseMhzlabel=char('Maximum measurable bandwidth by pulse (MHz)');
```

```
tbandwidthoccupiedbypulseMhzvalue=num2str(bandwidthoccupiedbypulseMhz);          % store measureable bandwidth

speedoflight=1/1e-9;                                                              % Light travels one foot per nanosecond
feetpermile=5280;                                                                 % store number of feet per mile
TXFrepetitiontimeperiod=1/TXF;                                                    % calculate transmit pulse repetition period
sounderrangemiles=TXFrepetitiontimeperiod*speedoflight/feetpermile;               % calculate unambiguous range of incoming multipath signal components in mile

tsounderrangemileslabel=char('Unambiguous range of incoming multipath signal components (miles)');
tsounderrangemilesvalue=num2str(sounderrangemiles);                               % store unambiguous range of incoming multipath signal components in miles

tdatacollectiontimeseclabel=char('Data collection time (seconds)');
tdatacollectiontimesecvalue=num2str(datacollectiontimesec);                       % store sounder data collection time in sec

multipathcomponentthresholddb=10*log10((multipathcomponentthresholdvoltage)^2/impedence); % calculate sounder multipath threshold in dB
multipathcomponentthresholddbvalue=multipathcomponentthresholddb;
tmultipathcomponentthresholddblabel=char('Multipath component threshold from normalized peak (dB)');
tmultipathcomponentthresholddbvalue=num2str(multipathcomponentthresholddbvalue);

tmaximumexcessdelaylabel1=char('Maximum Excess Delay (seconds), ');
tmultipathcomponentthresholddbvalue;
tmaximumexcessdelaylabel2=char(' dB from peak ');
tmaximumexcessdelayvalue=num2str(maximumexcessdelay);
tmaximumexcessdelaylabel=strcat(tmaximumexcessdelaylabel1,tmultipathcomponentthresholddbvalue,tmaximumexcessdelaylabel2);

% display SSTDSP sounding results on figure
text(.1,1.0,tpulsewidthseclabel);
text(.8,1.0,tpulsewidthsecvalue);
text(.1,.8,tbandwidthoccupiedbypulseMhzlabel);
text(.8,.8,tbandwidthoccupiedbypulseMhzvalue);
text(.1,.6,tsounderrangemileslabel);
text(.8,.6,tsounderrangemilesvalue);
```

```
text(.1,.4,tdatacollectiontimeseclabel);
text(.8,.4,tdatacollectiontimesecvalue);
text(.1,.2,tmultipathcomponentthresholddblabel);
text(.8,.2,tmultipathcomponentthresholddbvalue);
text(.1,.0,tmaximumexcessdelaylabel);
text(.8,.0,tmaximumexcessdelayvalue);



%
% power delay profile plot
%

subplot(5,1,3);

logAcenter=10*log10(abs((Acenter)).^2/impedence);                    % calculate power delay profile


plot(time,logAcenter, 'b-',time,multipathcomponentthresholddb*ones(1,length(time)),'r-');
axis([min(time) max(time) (multipathcomponentthresholddb*2) (max(logAcenter)+1)]);
txlabel1=char('Time Base in seconds: ');
txvalue1=num2str(bintime);
txlabel2=char(' seconds per Bin:  ');
txvalue2=num2str(bins);
txlabel3=char(' bins            (red line is multipath threshold)');
txlabel=strcat(txlabel1,txvalue1,txlabel2,txvalue2,txlabel3);

title('Power Delay Profile - Impulse Response');
xlabel(txlabel);
ylabel('Normalized Received Power in dB');
```

```
%
% plot of spectrum
%

subplot(5,1,5);

% set bounds for FFT
% determine the number of bins in thresholded and truncated power delay profile
lengthAcenterthresholded=length(Acenterthresholded)

% determine the number of bins for 850 MHz bandwidth = 41667
halfnanosecondbinsfor850mhzbandwidth=round(1/48e3/.5e-9)

% if length of thresholded and truncated power delay profile is less
% than 41667, make it 41667, otherwise if its greater than 41667, use the
% length of the thresholded truncated power delay profile

if lengthAcenterthresholded<halfnanosecondbinsfor850mhzbandwidth
  N=halfnanosecondbinsfor850mhzbandwidth
else
  N=length(Acenterthresholded)
end

% perform the FFT to convert impulse response to frequency spectrum
fftspectrum=abs(fft(Acenterthresholded,N));

% determine maximum frequency
maxspectrum=max(fftspectrum)
```

```
% scale frequency to normalize to one
spectrum=fftshift(fftspectrum)./maxspectrum;

% calculate power of spectrum
lspectrum=10*log10((spectrum).^2/impedence);

% calculate frequency vector
fbins=length(lspectrum)

% calculate center frequency
centerfrequency=(28.35+27.5)/2

% calculate frequency span given frequency vector and bandwidth scaling
freqspace=linspace(centerfrequency-.5*1/bintime/1e9,centerfrequency+.5*1/bintime/1e9,fbins);

% plot frequency response versus frequency
plot(freqspace,lspectrum, 'b-',freqspace,multipathcomponentthreshholddb*ones(1,length(freqspace)),'r-')
axis([min(freqspace) max(freqspace) -30 (max(lspectrum)+1)]);
title('FFT of Power Delay Profile - Spectrum Snapshot')
xlabel('Frequency in GHz            (red line is multipath threshold)')
ylabel('Normalized Received Power in dB')

% store channel metrics to strings for export to a file
mxdvalue=num2str(MXD);
mxdspace=char('_');
rdsvalue=num2str(RDS);
rdsspace=char('_');
bclowmhzvalue=num2str(BcLowMhz);
bclowmhzspace=char('_');
bchighmhzvalue=num2str(BcHighMhz);
bchighmzspace=char('_');
```

```
channelmetrics=strcat(mxdvalue,mxdspace,rdsvalue,rdsspace,bclowmhzvalue,bclowmhzspace,bchighmhzvalue,bchighmzspace);

% export channel metrics to a file
delete '\BroadbandSounder\captureddata\metric1.out';
fid2 = fopen('\BroadbandSounder\captureddata\metric1.out','w');
fwrite(fid2,channelmetrics);
fclose(fid2);



%
% DISPLAY SOUNDER PROCESSING STEPS
%

% Waveform information

fig2=figure(2);

screensize=get(0,'ScreenSize');
positionwindow=ceil([1 1 1 screenpercent].*screensize);
set(fig2,'Position',positionwindow);



%
% IMPULSE RESPONSE FROM DATA FILE
%

subplot(7,1,1);
axis off;
```

```
% plot impulse response directly from DSP memory
%stem(time,abs(Acalibrate),'b.-');
plot(time,abs(Acalibrate));
txlabel1=char('Time Base in seconds: ');
txvalue1=num2str(bintime);
txlabel2=char(' seconds per Bin:  ');
txvalue2=num2str(bins);
txlabel3=char(' bins ');
txlabel=strcat(txlabel1,txvalue1,txlabel2,txvalue2,txlabel3);

title('Plot1: A to D Output Over Time - From Data File');
xlabel(txlabel);
ylabel('Received Voltage');



%
% IMPULSE RESPONSE AFTER INTERPOLATION
%

subplot(7,1,3);

% plot impulse response after interpolation and smoothing
%stem(time,abs(Asmooth),'b.-');
plot(time,abs(Asmooth));
txlabel1=char('Time Base in seconds: ');
txvalue1=num2str(bintime);
txlabel2=char(' seconds per Bin:  ');
txvalue2=num2str(bins);
txlabel3=char(' bins ');
txlabel=strcat(txlabel1,txvalue1,txlabel2,txvalue2,txlabel3);
```

```
title('Plot2: A to D Output Over Time - After Interpolation');
xlabel(txlabel);
ylabel('Received Voltage');



%
% IMPULSE RESPONSE AFTER ORDERING AND NORMALIZING TO ONE
%

subplot(7,1,5);

% plot impulse response after ordering and normalizing to one
%stem(time,abs(Acenter),'b.-');
plot(time,abs(Acenter), 'b-',time,multipathcomponentthresholdvoltage*ones(1,length(time)),'r-');
txlabel1=char('Time Base in seconds: ');
txvalue1=num2str(bintime);
txlabel2=char(' seconds per Bin:  ');
txvalue2=num2str(bins);
txlabel3=char(' bins            (red line is multipath threshold)');
txlabel=strcat(txlabel1,txvalue1,txlabel2,txvalue2,txlabel3);

title('Plot 3: A to D Output Over Time - After Ordering and Normalizing to One ');
xlabel(txlabel);
ylabel('Received Voltage');



%
% IMPULSE RESPONSE AFTER QUANTIZATION
```

```
%

subplot(7,1,7);

% plot impulse response after quantization
stem(ptime,Aavg,'b.-');
%plot(ptime,Aavg);
txlabel1=char('Time Base in seconds: ');
txvalue1=num2str(pulsewidthsec);
txlabel2=char(' seconds per Bin:  ');
txvalue2=num2str(pbins);
txlabel3=char(' bins ');
txlabel=strcat(txlabel1,txvalue1,txlabel2,txvalue2,txlabel3);

title('Plot4: A to D Output Over Time - Discretized for Metric Calculations');
xlabel(txlabel);
ylabel('Received Voltage Normalized to One');



%
% SAVE CONFIGURATION
%

% save current configuration window in JPEG format
saveas(fig1,'\BroadbandSounder\captureddata\output2.jpg','jpg');
saveas(fig2,'\BroadbandSounder\captureddata\output3.jpg','jpg');
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [rotateddatavector]=circularmatchedfiltercorrelator(datavector,matchedwaveform)

% Christian J. Rieser, MSEE Virginia Tech
% SSTDSP Wireless Channel Sounder
% Circular Matched Filter Algorithm
% Version 4.0
% Processes data so that largest pulse is always first
% 6/26/2001
% This function is used to process the data vector so that the largest pulse waveform is always first
% It requires inputs of a datavector and matched waveform

%clc
%a=[.5 0 .5 0 0 0 0 0 0 0 0 0 .3 0 .2 .2 0 .2];
%b=[1 1 1 1]

% store original data vector to a
a=datavector;

% store original data vector to b
b=matchedwaveform;

% find length of data vector
datalength=length(a);

% find index at 1/2 way point of data vector
```

```
middlewayofdata=round(datalength/2)

% find index at 1/4 way point of data vector
quarterwayofdata=round(datalength/4)

% find index at 3/4 way point of data vector
threequarterswayofdata=round(datalength*3/4)

% preorder data vector to ensure if a peak pulse is sandwiched between the first bins and last bins
% it will still be recognized as the peak pulse in the matched filtering operation
% effectively allows "circular matched filtering" we assume that the peak pulse occupies less than 1/2 of
% the transmit pulse repetition period.

% four preordering scenerios, a1->a4

% store original vector to a1
a1=a;

% store original vector circularly shifted 1/2 length of the orgininal vector length
a2=[a(middlewayofdata:length(a)) a(1:middlewayofdata-1)];

% store original vector circularly shifted 1/4 length of the orgininal vector length
a3=[a(quarterwayofdata:length(a)) a(1:quarterwayofdata-1)];

% store original vector circularly shifted 3/4 length of the orgininal vector length
a4=[a(threequarterswayofdata:length(a)) a(1:threequarterswayofdata-1)];

% test code
%sbins=[0:length(a)-1];
%figure(20)
%plot(sbins,a1)
```

```
%stem(sbins,a1,'b.-');
%figure(21)
%plot(sbins,a2)
%stem(sbins,a2,'b.-');
%figure(22)
%plot(sbins,a3)
%stem(sbins,a3,'b.-');
%figure(23)
%plot(sbins,a4)
%stem(sbins,a4,'b.-');

% convolve matched pulse vector and data vector (matched filter) to find peak pulse in data vector a1->a4
% store peak correlation value v1->v4 and location l1->l4 for impulse response reordering and gain scaling operation
% this allows effective circular matched filtering and ensures that if a peak pulse is sandwiched between
% the first and last time bins, it can be recognized and reordered by this algorithm. This algorithm compares
% the peak correlations of the three different scenerios and picks the preordering scenerio that ensures the
% original peak pulse is not sandwiched between the first and last bins.
[v1,l1]=max(conv(a1,b));
[v2,l2]=max(conv(a2,b));
[v3,l3]=max(conv(a3,b));
[v4,l4]=max(conv(a4,b));

% store peak correlation values and locations for all four preordering scenerios in matrix
valuematrix=[v1 v2 v3 v4]
locationmatrix=[l1 l2 l3 l4]

% find peak correlation of all four preordering scenerios
[maxv,maxl]=max(valuematrix)

% store max correlation and position of max correlation
v=maxv
```

```
l=locationmatrix(maxl)

% store optimal preordering scenerio based on maximum peak correlation
switch maxl
        case 1, a=a1;
   case 2, a=a2;
   case 3, a=a3;
   case 4, a=a4;
   otherwise, disp('CORRELATION ERROR');
end

% use preordered data vector and knowledge of the position of the maximum correlation peak
% and length of the original matched filter waveform to shift peak pulse to the first time bin
% since the beginning of the peak pulse starts length(matchedwaveform) indexes before the peak
% correlation value, we just circularly shift the preordered data vector the position of the
% peak pulse - length of the matched waveform back to the first time bin. since the peak
% correlation occurs when the entire peak pulse is visible to the matched waveform, we will
% always pick the best scenerio that ensures we can see the entire peak pulse and have valid
% peak correlation information that permits us to shift the peak pulse back to the first time bin
if l<length(b)
   c=[a(length(a)+(l-length(b)):length(b)) a(1:length(a)+(l-length(b)-1))];
   disp('op1')
else
   c=[a(l-length(b)+1:length(a)) a(1:l-length(b))];
   disp('op2')
end

rotateddatavector=c;
```

C.17 Source code for "sounderdataconverter.cpp"

```
//Christian J. Rieser, MSEE Virginia Tech
//SSTDSP Wireless Channel Sounder
//Sounderdataconverter for SSTDSP Sounder
//Version 4.0
//6/26/2001
//This algorithm was written by Todd Eshler and comments added by Christian Rieser
//
//Processes ASCII DSP memory dump and produces a file with 32 bit binary int buffer values to be read by MATLAB
//Proper usage on the command line is "sounderdataconverter demo1.out demo1.bin"

#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <iomanip>

//use std namespace
using namespace std;

//size of inputfile header in char's
#define HEADER_SIZE 17

//number of junk bits for each value in inputfile
#define JUNKSPACE 10
```

```
//8 bit (1 byte) mask
#define MASK 0x000000FF

//mask to determine if value is negative
#define NEGATIVEBIT 0x00800000

//mask needed to sign extend negative values
#define NEGATIVEMASK 0xFFFFC000

//define forever to be 1
#define FOREVER 1


//argc tells how many input arguments (should be 3) on command line and argv[]
//contains those command line input arguments
//argv[0] should be sounderdataconverter
//argv[1] should be demo1.out DSP memory dump
//argv[2] should be demo1.bin binary file to be read into MATLAB

int main(int argc, char * argv[])
{
        //value read from inputfile
        int value;

        //8 bits to an unsigned char, therefore an array
        //of 4 unsigned chars produces a 32 bit binary variable
        unsigned char writevalue[4];

        //counter variable
        int i;
```

```
//junk char used to remove HEADER SIZE char header from inputfile
char junkheader;

//input file stream
ifstream inputfile;

//output file stream
ofstream outputfile;

//define negative to be false
bool negative = false;

//if sounderdataprocess does not receive enough input arguments, give error and exit
if ( argc != 3 )
{
        cerr<<"Usage: "<<argv[0]<<" [inputfile] [outputfile]"<<endl;
        return EXIT_FAILURE;
}

//open the file indicated by the argv[1], should be DSP memory dump file "demo1.out"
inputfile.open(argv[1]);


//if input file does not exist or can not be opened, give an error and exit
if (!inputfile.is_open())
{
        cerr<<"Could not open inputfile"<<endl;
        return EXIT_FAILURE;
}
```

```
//open a binary output file with file name found in argv[2], should be "demo1.bin"
outputfile.open(argv[2],ios::outlios::binary);


//if output file can not be opened, give an error and exit
if (!outputfile.is_open())
{
	cerr<<"Could not open outputfile"<<endl;
	return EXIT_FAILURE;
}

//remove header from input file
for(i=0; i<HEADER_SIZE;i++)
	inputfile.get(junkheader);

while(FOREVER)
{
	//set negative to false
	negative = false;

	//read in a single value with hex base
	inputfile>>hex>>value;

	//if no more values to read and convert, break and exit infinite while loop
	if ( inputfile.fail() )
		break;

	//detect if current value is negative
	if ( value & NEGATIVEBIT )
		negative = true;
```

```
		//get rid of JUNKSPACE bits in input value by shifting the top 14 bits
		//of the 24 bit word to the bottom 14 bits of the 24 bit word
		//this puts zeroes in the upper bits
		value=value>>JUNKSPACE;

		//if the value is negative, sign extend the negative number using the NEGATIVEMASK
		if (negative)
			value = value | NEGATIVEMASK;

		//loop through and grab lower 32 bits of each value
		for(i=3;i>-1;i--)
		{
			//grab eight bits at a time to write to output file
			writevalue[i]= value & MASK;

			//remove the eight bits that were just copied.
			value=value>>8;
		}

		//write a 32 bit binary value to file
		outputfile.write((const char *)writevalue,4);

}

//close input file
inputfile.close();

//close output file
outputfile.close();

//exit success
```

```
        return EXIT_SUCCESS;

}
```

C.18 Source code for "AtoD1.asm"

```
page    132,66
;*************************************************************************
;
;   AtoD.ASM - Store AtoD Data on the Sounder Board
;   Christian J. Rieser
;   SSTDSP DSP data capture and buffering code
;   MSEE Virginia Tech
;   6/27/2001
;
;   56311 DSP setup code provided by Motorola with DSP
;   SSTDSP processing routines written by Christian J. Rieser
;
;*************************************************************************

;*************************************************************************
;
; THIS SECTION IS FOR USER PARAMETERS
;
;*************************************************************************
```

```
PXMemStart equ  $100000
PXMemEnd   equ  $100067   ; <---------EDIT HERE AFTER $

;PXMemEnd  equ  $17FFFF

PXMemSize  equ  PXMemEnd-PXMemStart+1
PXMemcount equ  PXMemSize+1
Startcount equ          $000000

;*************************************************************************
;
; THIS SECTION IS FOR DSP CONFIGURATION PARAMETERS
;
;*************************************************************************

;--- DSP56311 Control Registers (X I/O SPACE)
IPRC    equ     $FFFFFF             ; Interrupt Priority Core Register
IPRP    equ     $FFFFFE             ; Interrupt Priority Peripheral Register
BCR     equ     $FFFFFB              ; Bus Control Register
PCTL    equ     $FFFFFD              ; PLL Control Register
AAR0    equ     $FFFFF9              ; Address Attribute Register #0
AAR1    equ     $FFFFF8              ; Address Attribute Register #1
AAR2    equ     $FFFFF7              ; Address Attribute Register #2
AAR3    equ     $FFFFF6              ; Address Attribute Register #3

;--- Timer Control Registers (X I/O SPACE)
TPCR        equ         $FFFF82                     ; Current Timer Prescaler Counter (R)
TPLR        equ         $FFFF83                     ; Timer Prescaler Preload Value (R/W)
TCR2        equ         $FFFF84                     ; Timer 2 Counter Value (R)
TCPR2       equ         $FFFF85                     ; Timer 2 Compare Value (R/W)
TLR2        equ         $FFFF86                     ; Timer 2 ReLoad Value (W)
```

```
TCSR2   equ     $FFFF87                 ; Timer 2 Contol/Status Register (R/W)
TCR1    equ     $FFFF88                 ; Timer 1 Counter Value (R)
TCPR1   equ     $FFFF89                 ; Timer 1 Compare Value (R/W)
TLR1    equ     $FFFF8A                 ; Timer 1 ReLoad Value (W)
TCSR1   equ     $FFFF8B                 ; Timer 1 Control/Status Register (R/W)
TCR0    equ     $FFFF8C                 ; Timer 0 Counter Value (R)
TCPR0   equ     $FFFF8D                 ; Timer 0 Compare Value (R/W)
TLR0    equ     $FFFF8E                 ; Timer 0 ReLoad Value (W)
TCSR0   equ     $FFFF8F                 ; Timer 0 Control/Status Register (R/W)

LED_ON  EQU     $002800
;-------------------------------------------------------------------------
; The above value disables a Timer (uses it as a GPIO), Sets its direction
; to output, and sends a 1 to turn on the LED
;-------------------------------------------------------------------------

LED_OFF EQU     $000800
;-------------------------------------------------------------------------
; The above value disables a Timer (uses it as a GPIO), Sets its direction
; to output, and sends a 0 to turn off the LED
;-------------------------------------------------------------------------

;--- PCTL value = 0x0F0004
pllmul  equ     4                       ; bits 0-11, VCO Mult = 5; (4+1)*19.6608MHz=98.304MHz
lowdiv  equ     0                       ; bit 12-14, Low Power Divider = 1
crystal equ     0                       ; bit 15, No, Crystal not less than 200kHz
disXTAL equ     $010000                 ; bit 16, Yes, disable crystal use
pllstop equ     $020000                 ; bit 17, Yes, PLL runs during STOP
enpll   equ     $040000                 ; bit 18, Yes, enable PLL operation
disclk  equ     0                       ; bit 19, No, enable CORE clock output
;disclk  equ     $080000                ; bit 19, Yes, disable CORE clock output
```

```
prediv  equ     0                       ; bits 20-23, Pre-Divider = 1
PCTL_value equ  prediv+lowdiv+pllmul+crystal+disXTAL+pllstop+enpll+disclk


;--- AAR0 value = 0x100439                          ; SRAM /CS
acctype0 equ    1                       ; Asynchronous External Memory access type = 0x1
aahigh0  equ    0                       ; Enable AA0 pin to be low when selected
aap0     equ    $8                      ; Yes, Enable AA0 pin on ext 'P' accesses
aax0     equ    $10                     ; Yes, Enable AA0 pin on ext 'X' accesses
aay0     equ    $20                     ; Yes, Enable AA0 pin on ext 'Y' accesses
aswap0   equ    0                       ; No, Enable address bus swap
enpack0  equ    0                       ; No, Enable packing/unpacking logic
nadd0    equ    $000400                 ; Compare 4 address bits
msadd0   equ    $100000                 ; Most significant portion of address,
                                                        ; $100000 - $1fffff, to compare.
                                                        ; (0001,xxxx,xxxx,xxxx,xxxx,xxxx)
AAR0_value equ  acctype0+aahigh0+aap0+aax0+aay0+aswap0+enpack0+nadd0+msadd0

;--- AAR1 value = 0xF20721                  ; A/D /CS
acctype1 equ    1                       ; Asynchronous External Memory access type = 0x1
aahigh1  equ    0                       ; Enable AA1 pin to be low when selected
aap1     equ    0                       ; No, Disable AA1 pin on ext 'P' accesses
aax1     equ    0                       ; No, Diable AA1 pin on ext 'X' accesses
aay1     equ    $20                     ; Yes, Enable AA1 pin on ext 'Y' accesses
aswap1   equ    0                       ; No, Disable address bus swap
enpack1  equ    0                       ; No, Disable packing/unpacking logic
nadd1    equ    $000700                 ; Compare 7 address bits
msadd1   equ    $F20000                 ; Most significant portion of address,
                                ; $F20000 - $F3ffff, to compare.
                                ; (1111,001x,xxxx,xxxx,xxxx,xxxx)
AAR1_value equ  acctype1+aahigh1+aap1+aax1+aay1+aswap1+enpack1+nadd1+msadd1
```

```
;--- AAR2 value = 0x14061D                    ; SRAM A18
acctype2 equ    1                ; Asynchronous External Memory access type = 0x1
aahigh2  equ    $4               ; Enable AA2 pin to be high when selected
aap2     equ    $8               ; Yes, Enable AA2 pin on ext 'P' accesses
aax2     equ    $10              ; Yes, Enable AA2 pin on ext 'X' accesses
aay2     equ    0                ; No, Enable AA2 pin on ext 'Y' accesses
aswap2   equ    0                ; No, Enable address bus swap
enpack2  equ    0                ; No, Enable packing/unpacking logic
nadd2    equ    $000600          ; Compare 5 address bits
msadd2   equ    $140000          ; Most significant portion of address,
                                 ; $100000 - $17ffff, to compare.
                                 ; (0001,01xx,xxxx,xxxx,xxxx,xxxx)
AAR2_value equ  acctype2+aahigh2+aap2+aax2+aay2+aswap2+enpack2+nadd2+msadd2

;--- AAR3 value = 0x180525                    ; SRAM A19
acctype3 equ    1                ; External Memory access type = 0x1
aahigh3  equ    $4               ; Enable AA3 pin to be high when selected
aap3     equ    0                ; No, Enable AA3 pin on ext 'P' accesses
aax3     equ    0                ; No, Enable AA3 pin on ext 'X' accesses
aay3     equ    $20              ; Yes, Enable AA3 pin on ext 'Y' accesses
aswap3   equ    0                ; No, Enable address bus swap
enpack3  equ    0                ; No, Enable packing/unpacking logic
nadd3    equ    $000500          ; Compare 5 address bits
msadd3   equ    $180000          ; Most significant portion of address,
                                 ; $180000 - $1fffff, to compare.
                                 ; (0001,1xxx,xxxx,xxxx,xxxx,xxxx)
AAR3_value equ  acctype3+aahigh3+aap3+aax3+aay3+aswap3+enpack3+nadd3+msadd3


;--- BCR value = 0x0248A2
```

```
aaa0ws  equ     $000002              ; Address Attribute Area 0 w/s = 2
aaa1ws  equ     $0000A0              ; Address Attribute Area 1 w/s = 5
aaa2ws  equ     $000800              ; Address Attribute Area 2 w/s = 2
aaa3ws  equ     $004000              ; Address Attribute Area 3 w/s = 2
defws   equ     $020000              ; Default Address Area w/s = 2
busss   equ     0                    ; Bus state status = 0
enblh   equ     0                    ; Enable Bus Lock Hold = 0
enbrh   equ     0                    ; Enable Bus Request Hold = 0
BCR_value equ   aaa0ws+aaa1ws+aaa2ws+aaa3ws+defws+busss+enblh+enbrh

;--- A/D Port Addresses
AD0     equ     $F20000
AD1     equ     $F20001
AD2     equ     $F20002
AD3     equ     $F20003
Status  equ     $F20007
;    D8  = Done #0
;    D9  = Done #1
;    D10 = Done #2
;    D11 = Done #3
;    D15 = Done (Done0 & Done1 & Done2 & Done3)
;    D23 = Trigger

; A/D Done Interrupt Enable  $F20005
; A/D Done Interrupt Disable $F20006


;*****************************************************************************
;
; THIS SECTION SETS UP DSP PROGRAM MEMORY
;
```

```
;*************************************************************************

                org     p:0

                jmp             main      ; Vector 0


;*************************************************************************
;
; THIS SECTION IS THE MAIN DSP PROGRAM
;
;*************************************************************************

        org     p:$100

main

;-------------------------------------------------------------------------
; Set up the memory space and chip selects for both A/D and SRAM.
;-------------------------------------------------------------------------

        movep   #PCTL_value,x:PCTL  ; PLL 4 x 19.6608MHz = 98.304MHz
        movep   #AAR0_value,x:AAR0  ; maps SRAM from p/x:$100000-$17FFFF
        movep   #AAR1_value,x:AAR1  ; maps A/D from y:$F20000-x:$F3FFFF
        movep   #AAR2_value,x:AAR2  ; maps AAR2 as A18
        movep   #AAR3_value,x:AAR3  ; maps AAR3 as A19
        movep   #BCR_value,x:BCR    ; 1 wait states for all areas,

;-------------------------------------------------------------------------
; Turn off all LEDS.
;-------------------------------------------------------------------------
```

```
        movep   #LED_OFF,x:TCSR0      ; Turn OFF Red LED
        movep   #LED_OFF,x:TCSR1      ; Turn OFF Yellow LED
        movep   #LED_OFF,x:TCSR2      ; Turn OFF Green LED

;---------------------------------------------------------------------
; Select addressing mode.
;---------------------------------------------------------------------

        move    #-1,m0                ; Select Linear Addressing Mode
        move    m0,m1
        move    m1,m2

;---------------------------------------------------------------------
; Branch to subrountine that captures and buffers channel data.
;---------------------------------------------------------------------

                bsr             acquiredatasubroutine     ; Go capture and buffer the data

;---------------------------------------------------------------------
; Turn on all LEDS to indicate data capture and buffering is complete.
;---------------------------------------------------------------------

        movep   #LED_ON,x:TCSR0       ; Turn ON Red LED
        movep   #LED_ON,x:TCSR1       ; Turn ON Yellow LED
        movep   #LED_ON,x:TCSR2       ; Turn ON Green LED

;---------------------------------------------------------------------
; Break to allow DSP debugger to download DSP memory to RX Host Module.
; Eventually this break will be replaced by a SCI RS-232 data transfer subroutine
;---------------------------------------------------------------------
```

```
        debug                           ; break to allow user download of memory buffer through debugger

;-------------------------------------------------------------------------
; Subroutine that captures and buffers channel data

acquiredatasubroutine:

        move    #PXMemSize,n0           ; Get P/X memory size
        move    #PXMemStart,r0          ; Get starting address for fill
        move    #PXMemcount,n1                  ; Store length of buffer +1
        move    #Startcount,A                   ; Initialize Accummulator to zero

        dor forever,databuffer          ; loop forever until buffer is full and loop is broken

adsampling                                      ; main data capture loop (all clock cycles are worst case)

    movep       #LED_OFF,x:TCSR1        ; Turn off the Yellow LED (6 clock cycles)
        move            y:Status,x0     ; Store Status register (3 clock cycles)
        btst            #23,x0                  ; Store Trigger Status in carry bit of condition register x000 (2 clock cycles)
    movep       #LED_ON,x:TCSR1         ; Turn on the Yellow LED (6 clock cycles)
        jcc             adsampling              ; Jump if trigger=0=cc, AD is sampling (4 clock cycles)

        add             #000001,A       ; increment DSP buffer index (2 clock cycles)
        move            A,r1                    ; store new index value in r1 register (3 clock cycles)
        move            n1,x0                   ; move length of DSP memory buffer +1 into x0 register (3 clock cycles)
        cmp             x0,A                    ; compare current index with length of DSP buffer +1 (2 clock cycles)
        brkeq                           ; if equal DSP buffer is full and break out of databuffer loop (5 clock cycles)

    movep       #LED_OFF,x:TCSR1    ; Turn off the Yellow LED (6 clock cycles)
    movep       #LED_ON,x:TCSR2         ; Turn on the Green LED (6 clock cycles)
```

```
            move            y:AD0,x0            ; AD done sampling, trigger=1, get A/D Channel #0 Data (3 clock cycles)
        move        x0,x:(r0)+          ; Save off channel data into buffer (3 clock cycles)
;       move        y:Status,x0         ; Get A/D Status Data
;       move        x0,x:(r0)+          ; Save off status data into buffer
        movep       #LED_OFF,x:TCSR2            ; Turn off the Green LED (6 clock cycles)

wait                                                ; wait loop to make sure data capture is complete
        movep       #LED_ON,x:TCSR0             ; Toggle the Red LED (6 clock cycles)
            move            y:Status,x0         ; Store Status register (3 clock cycles)
            btst            #23,x0                  ; Store Trigger Status in carry bit of condition register x000 (2 clock cycles)
            jcs             wait                    ; Check to see if trigger=1=cs, next sample not ready (4 clock cycles)
        movep       #LED_OFF,x:TCSR0            ; Turn off the Red LED (6 clock cycles)
            jmp             adsampling              ; jump to beginning of adsampling loop (3 clock cycles)

databuffer

    rti                                         ; return from interrupt (+15 clock cycles)


;=============================================================================


    end     main
```

C.19 Source code for "capturedata.cmd"

```
device dv0 cc0 tms0 pos0 56311
```

```
force s
load  "D:\BroadbandSounder\code\dspcode\atod1.cld"
go
q
```

C.20 Source code for "exportdata.cmd"

```
save x:100000..100067 "D:\BroadbandSounder\captureddata\demo1.out"
q
```

C.21 Source code for "make.cmd"

```
g563c atod1.asm
dsplnk atod1.cln
```

C.22 Source code for "RX_STEP1_configuresounder.cmd"

```
cd D:\BroadbandSounder\applications\MATLABr12
"MATLAB R12.lnk"
cd D:\
cd D:\BroadbandSounder
```

C.23 Source code for "RX_STEP2_editassembly.cmd"

```
cd D:\BroadbandSounder\code\dspcode
notepad atod1.asm
cd D:\
cd D:\BroadbandSounder
```

C.24 Source code for "RX_STEP3_editdspcontrol.cmd"

```
cd D:\BroadbandSounder\code\dspcode
notepad exportdata.cmd
make
cd D:\
cd D:\BroadbandSounder
```

C.25 Source code for "RX_STEP4_SETSWITCHTOA_setdds.cmd"

```
cd "D:\BroadbandSounder\applications\AD9852_54\"
AD9852_54 D:\BroadbandSounder\code\ddscode\sounderx.54
cd D:\BroadbandSounder
```

C.26 Source code for "RX_STEP5_SETSWITCHTOB_setdds.cmd"

```
cd "D:\BroadbandSounder\applications\AD9852_54\"
AD9852_54 D:\BroadbandSounder\code\ddscode\sounderx.54
cd D:\BroadbandSounder
```

C.27 Source code for "RX_STEP6_startgps.cmd"

```
cd D:\
cd D:\BroadbandSounder\applications\OutlookGPSPlus\
OutlookGPSPlus
cd D:\
cd D:\BroadbandSounder
```

C.28 Source code for "RX_STEP7_SETSWITCHTOC_acquireconvertdata.cmd"

```
cd D:\
cd D:\BroadbandSounder\applications\Motorola\Dsp\dsp\bin\
ads56300.exe -d parallel:1 D:\BroadbandSounder\code\dspcode\capturedata.cmd
ads56300.exe -d parallel:1 D:\BroadbandSounder\code\dspcode\exportdata.cmd
cd D:\
cd D:\BroadbandSounder
cd D:\
cd D:\BroadbandSounder\captureddata
sounderdataconverter demo1.out demo1.bin
```

```
cd D:\BroadbandSounder
```

C.29 Source code for "RX_STEP8_PRESSSPACEBARINMATLAB_processdata.cmd"

```
cd D:\
cd D:\BroadbandSounder\
```

C.30 Source code for "TX_STEP1_setdds.cmd"

```
cd "D:\BroadbandSounder\applications\AD9852_54\"
AD9852_54 D:\BroadbandSounder\code\ddscode\sounderx.54
cd D:\BroadbandSounder
```

C.31 Source code for "TX_STEP2_startgps.cmd"

```
cd D:\
cd D:\BroadbandSounder\applications\OutlookGPSPlus\
OutlookGPSPlus
cd D:\
cd D:\BroadbandSounder
```

Appendix D

Details of Test procedure for taking SSTDSP sounder measurements

D1. Step 1: Setup and configuration

The following steps must be followed to complete setup and configuration of the sounder.

1. Position sounder transmitter and receiver at desired node locations in the field. Optimal communications at LMDS will occur with line of sight paths; however, the sounder can be used to investigate LMDS "bounce paths."
2. Power on the sounder transmitter and receiver by turning on the power strip in the TX and RX Power Modules.
3. Activate the TX and RX Host Modules by booting the Windows workstation computer located at the transmitter and receiver, respectively.
4. Determine the sounder configuration settings by running the "RX_STEP1_configuresounder.cmd" script at the receiver. All sounder measurements scripts can be found in the directory "D:\BroadbandSounder\run."
5. Run the script "RX_STEP2_editassembly.cmd" which opens up the file "AtoD1.asm." Using the hexadecimal memory buffer size indicated on the sounder configuration window in MATLAB, edit the line that says "PXMemEnd equ $100067 ; <---------EDIT HERE AFTER $" and replace the quantity after the $ (which is 100067 in this case) with the hexadecimal memory buffer size value. Save the document and exit.
6. Run the script "RX_STEP3_editdspcontrol.cmd" which opens up the file "exportdata.cmd" Using the hexadecimal memory buffer size indicated on the

sounder configuration window in MATLAB, edit the line that says "save x:100000..100067 "D:\BroadbandSounder\captureddata\demo1.out"" and replace the quantity after the .. (which is 100067 in this case) with the hexadecimal memory buffer size value. Save the document and exit. This recompiles the DSP binary executable code.

7. Activate the TX Frequency and Location Module by running the "TX_STEP1_setdds.cmd" script. Using the "Setup File: Load Setup File" DDS software menu load the file "D:\BroadbandSounder\applications\startsoundertx.54s". Enter the binary transmit DDS set word indicated in the sounder configuration window in MATLAB and press enter. Using the "Setup File: Save Setup File" DDS software menu, save this DDS setting as a new file name of the format "sounderTXtransmitpulserepetitionfrequency.54s", where transmitpulserepetitionfrequency is the transmit pulse repetition frequency indicated in the MATLAB configuration window. Load that same file again to ensure that the DDS has been set. Once a transmit pulse repetition frequency has been saved, if it is needed in the future, it may be loaded directly from the DDS software.
8. Run the script "TX_STEP2_startgps.cmd" to activate the GPS monitoring software at the transmitter.
9. The sounder transmitter is now transmitting short pulses and upconverting them to the LMDS band. The sounder receiver is therefore downconverting the spectrum response and rendering the impulse response.
10. The sounder receiver must now be configured to be able to capture the baseband channel impulse response using the SSTDSP sounding method.
11. Activate the RX Frequency and Location Module by turning the parallel port switch box at the receiver to position "A" and running the "RX_STEP4_SETSWITCHTOA_setdds.cmd" script. Using the "Setup File: Load Setup File" DDS software menu load the file "D:\BroadbandSounder\applications\startsounderrx.54s". Enter the binary receive DDS set word indicated in the sounder configuration window in MATLAB and press enter.

12. Using the "Setup File: Save Setup File" DDS software menu, save this DDS setting as a new file name of the format "sounderRXAreceivetimebininseconds.54s", where receivetimebininseconds is the time bin in seconds indicated in the MATLAB configuration window. Load that same file again to ensure that the DDS has been set. Once a receive time bin setting has been saved, if it is needed in the future, it may be loaded directly from the DDS software.
13. Turn the parallel port switch box at the receiver to position "B" and run the "RX_STEP4_SETSWITCHTOB_setdds.cmd" script. Using the "Setup File: Load Setup File" DDS software menu load the file "D:\BroadbandSounder\applications\startsounderrx.54s". Enter the binary receive DDS set word indicated in the sounder configuration window in MATLAB and press enter. Also enter the hexadecimal phase offset for the "B" receiver DDS indicated on the MATLAB sounder configuration window and press enter.
14. Using the "Setup File: Save Setup File" DDS software menu, save this DDS setting as a new file name of the format "sounderRXBreceivetimebininseconds.54s", where receivetimebininseconds is the time bin in seconds indicated in the MATLAB configuration window. Load that same file again to ensure that the DDS has been set. Once a receive time bin setting has been saved, if it is needed in the future, it may be loaded directly from the DDS software.
15. Run the script "RX_STEP6_startgps.cmd" to activate the GPS monitoring software at the transmitter.

D2. Step 2: Data capture and storage

1. Turn the parallel port switch box at the receiver to "C" and run the "RX_STEP7_SETSWITCHTOC_acquireconvertdata.cmd" script. This automatically initiates the digital channel measurement by the RX DSP Module, buffers the resulting digitized impulse response in the RX DSP Module, downloads the digital impulse response to the RX Host Module, and converts the digital impulse response to a binary file to be analyzed and displayed by the RX Host Module.

D3. Step 3: Data analysis and display

1. Press the space bar in MATLAB to analyze and display the channel data. The "RX_STEP8_PRESSSPACEBARINMATLAB_processdata.cmd" script does not actually process anything, but instead reminds the user to complete the SSTDSP measurement by pressing the space bar. This automatically calculates the power delay profile, channel frequency response, and channel metrics and then displays them through the RX Host Module screen.

The SSTDSP sounder channel measurement is now complete. As the reader can see, the lengthiest steps required to complete the SSTDSP channel measurement are localized around the initial setup and configuration of the SSTDSP sounder. Once the initial setup is complete, the actual data gathering and analysis occurs very rapidly.

Bibliography

[1] T. Cover and J. Thomas, *Elements of Information Theory,* John Wiley & Sons, 1991.

[2] R. N. Bracewell, *The Fourier Transform and Its Applications,* WCB/McGraw-Hill, 2nd ed. Revised, 1986.

[3] L. L. Peterson and B. S. Davie, *Computer Networks, A Systems Approach,* Morgan Kaufmann Publishers, 2nd ed., 2000.

[4] Virginia Tech LMDS Research Effort, http://www.lmds.vt.edu

[5] H. Xu, "Terrestrial radio wave propagation at millimeter-wave frequencies" PhD Dissertation, Virginia Tech, May 1, 2000.

[6] T. S. Rappaport, *Wireless Communications: Principles and Practice*. New Jersey: Prentice Hall Inc., 1996.

[7] W. L. Stutzman and and G. A. Thiele, *Antenna Theory and Design*. New York: Wiley, 1981.

[8] C. A. Balanis, *Advanced Engineering Electromagnetics*. New York: John Wiley and Sons, 1989.

[9] M. Born and E. Wolfe, *Principles of Optics: Electromagnetic Theory of Propagation, Interference, and Diffraction of Light*. January 1998.

[10] J. D. Parsons, *The Mobile Radio Propagation Channel*. New York: John Wiley and Sons, 1994.

[11] D. Sweeney, "Short pulse channel sounder for LMDS and other wideband channel measurement", Center for Wireless Telecommunications (CWT) internal white paper, Virginia Tech, 1999.

[12] G. L. Turin *et al,* "A statistical model of urban multipath propagation," *IEEE Transactions on Vehicular Technology,* Vol. VT-21, pp. 1-9, February 1972.

[13] T. S. Rappaport *et al*., "Statistical channel impulse response models for factory and open plan building radio communication system design," *IEEE Transactions on Communications,* Vol. COM-39, No. 5, pp. 794-806, May 1991.

[14] T. S. Rappaport, "Characterization of UHF multipath radio channels in factory buildings," *IEEE Transactions on Antennas and Propagation,* Vol. 37, No. 8, pp. 1058-1069, August 1989.

[15] D. C. Cox, "Delay-doppler characteristics of multipath delay spread and average excess delay for 910 MHz urban mobile radio paths," *IEEE Transactions on Antennas and Propagation,* Vol. AP-20, No. 5, pp. 625-635, September 1972.

[16] D. C. Cox and R. P. Leck, "Distributions of multipath delay spread and average excess delay for 910 MHz urban mobile radio paths," *IEEE Transactions on Antennas and Propagation,* Vol. AP-23, No. 5, pp. 206-213, March 1975.

[17] S. Haykin, *Communications Systems*. John Wiley & Sons, 3rd ed, 1994.

[18] K. Pahlavan and A. H. Levesque, *Wireless Information Networks,* Chapter 5, John Wiley & Sons, New York, 1995.

[19] H. Zaghloul, G. Morrison, and M. Fattouche, "Frequency response and path loss measurements of indoor channels," *Electronics Letters,* Vol.27, No. 12, pp. 1021-1022, June 1991.

[20] H. Zaghloul, G. Morrison, and M. Fattouche, "Comparison of indoor propagation channel characteristics at different frequencies," *Electronic Letters,* Vol. 27, No. 22, pp. 2077-2079, October 1991.

[21] R. F. Linfield, R. W. Hubbard, and L. E. Pratt, "Transmission channel characteristics by impulse response measurements", U. S. Department of Commerce, Office Telecommunications (OT) Rep. 76-96, 1976.

[22] W. R. Young and L. Y. Lacy, "Echoes in transmission at 450 megacycles from land-to-car radio units", *Proc. IRE,* Vol. 38, pp.255-258, 1950

[23] J. Van Rees, "Measurements of the impulse response of a wideband radio channel at 910 MHz from a moving vehicle", *Electronic Letters,* Vol. 22, No. 5, pp. 246-247, 1986.

[24] J. Van Rees, "Measurements of the wideband radio channel characteristics for rural, residental, and suburban areas", *IEEE Transactions on Vehicular Technology,* Vol. VT-36, No. 1, pp. 2-6 (1987).

[25] A. Papoulis, *Probability, Random Variables, and Stocastic Processes,* McGraw-Hill, 1965.

[26] D. V. Sarwate and M. B. Pursley, "Crosscorrelation properties of pseudorandom and related sequences", *Proc. IEEE,* Vol. 68, No. 5, pp. 593-619, 1980.

[27] M. K. Simon, J. K. Omura, R. A. Scholtz, and B. K. Levitt, *Spread Spectrum Communications,* Computer Science Press, Rockville, MD, 3 volumes, 1985.

[28] A. S. Bajwa and J. D. Parsons, "Small-area characterization of UHF urban and suburban mobile radio propagation", *IEE Proc.,* Vol. 129, Pt. F, No. 2, pp. 102-109, 1982.

[29] D. L. Nielson, "Microwave propagation measurements for mobile digital radio application," *IEEE Transactions on Vehicular Technology,* Vol. VT-27, No. 3, pp. 117-131, 1978.

[30] D. M. J. Devasirvatham, "Time delay spread and signal level measurements of 850 MHz radio waves in building environments," *IEEE Transactions on Antennas and Propagation,* Vol. AP-34, No. 11, pp. 1300-1305, 1986.

[31] P. F. Sass, "Propagation measurements for UHF spread spectrum mobile communications," *IEEE Transactions on Vehicular Technology,* Vol. VT-32, No. 2, pp. 168-176, 1983.

[32] P. W. Huish and E. Gurdenli, "Radio channel measurement and predictions for future mobile radio systems", *British Telecommunications Technology Journal,* Vol. 6, No. 1, pp. 43-53, 1988.

[33] R. C. Dixon, *Spread Spectrum Systems,* 2nd edition, John Wiley and Sons Inc., New York, 1984.

[34] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Prentice Hall, 1975.

[35] Brooks Shera, "A GPS Based Frequency Standard," QST, July 1998, pg 36-44.

[36] Brooks Shera website, http://www.rt66.com/~shera/index_fs.htm

[37] D. Sweeney, "Short pulse LMDS channel sounder", Center for Wireless Telecommunications (CWT) internal status report, Virginia Tech, April 2001.

[38] C. Ruthroff, "Multipath fading on line-of-sight microwave radio systems as a function of path length and frequency", *Bell System Technical Journal,* Vol. 50, pp. 2375-2398, September 1971.

[39] P. A. Tenerelli, "Diffraction by building corners at 28 GHz: Measurements and modeling" MSEE Thesis, Virginia Tech, June 3, 1998.

[40] H. M. Rein, "Subnanosecond pulse generator with variable pulse width using avalanche transistors," *Electronic Letters,* Vol. 11, No. 1, pp. 21-23, January 1975.