A Platform for Ultra Wideband Communication Systems

by

Nathan Ackerman

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2005 June 2005

Author ....
Department of Electrical Engineering and Computer Science
May 19, 2005

Certified by ....
Anantha Chandrakasan
Professor of Electrical Engineering
Thesis Supervisor

Accepted by .... [illegible] ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

[redacted] BARKER

[redacted]

A Platform for Ultra Wideband Communication Systems

by

Nathan Ackerman

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering

Abstract

In this thesis, a web interface for sending and receiving data across an actual UWB channel was designed. In addition, a platform for Ultra Wideband (UWB) communication development was implemented. The UWB communication platform is implemented to provide a unique testing tool for both transmission and reception of UWB signals. In debugging the system, the UWB communication proved invaluable as the platform was able to able to accurately and quickly discover errors. Both new tools should prove extremely useful for developing future UWB front and back ends.

Thesis Supervisor: Anantha Chandrakasan
Title: Professor of Electrical Engineering

Acknowledgments

First and foremost, I would like to thank Professor Anantha Chandrakasan for granting me the opportunity to work on the system described in this thesis and for all of the time that he spent helping me along the way. I have truly enjoyed working under his supervision and I will sincerely miss the work atmosphere that he fosters in his laboratory.

David Wentzloff, Raul Blazquez, Vivienne Sze, Fred Lee, and Nathan Ickes, were all incredibly helpful and were key components to completion the system described in this thesis. I wish to thank them all for their help.

Finally, I would like to thank my parents for their eternal love and support.

Contents

1 Introduction **13**

1.1 Previous work 14

1.2 Goals 16

2 Web Interface **17**

2.1 Design Goals 17

2.2 Hardware Requirements and Setup 18

2.3 Architecture 20

2.3.1 User Handler 21

2.3.2 Background Processor 26

2.3.3 Packet Transmit and Capture 27

3 UWB Platform Architecture **29**

3.1 UWB Development Platform Hardware 32

3.1.1 Standalone Carrier 32

3.1.2 Signal Processing FPGA 34

3.1.3 Control FPGA 35

3.1.4 Custom Carrier 35

3.1.5 Crystal Oscillator 36

3.1.6 Discrete Down Converter 36

3.1.7 Discrete Pulse Generator 36

4 UWB Platform Design **39**

4.1 Control FPGA 39
4.1.1 Opal Kelly interface 39
4.2 FPGA reprogramming 41
4.2.1 Com Port Interface 42
4.2.2 MSP430 Programming 43
4.3 Transmission Interface 45
4.4 Receiver interface 46
4.5 Boot Sequence 48
4.5.1 Programming the Control FPGA 48
4.5.2 Programming the Signal Processing FPGA 48
4.5.3 Initializing the ADC 49
4.5.4 Running PC side software 50

5 Results **53**
5.1 Test Design 53
5.2 Analysis 56

6 Conclusion **59**
6.1 Future Work 60

List of Figures

1-1 Discrete UWB Communication Channel 15

2-1 Web-Based Interface 21
2-2 Screenshot for Web Interface 22
2-3 Screenshot when Waiting 24
2-4 Screenshot of Successful Transmission 25
2-5 Screenshot of Error 26

3-1 Functional Receiver Model 30
3-2 Functional Transmitter Model 31
3-3 Complete UWB Development Platform Model 33
3-4 UWB Development Platform Hardware Connectivity 34

4-1 Wiring Diagram for Programming Interface 42
4-2 UWB Development Platform Transmission Interface 45
4-3 Timing Diagram using Discrete Transmitter Interface 46
4-4 UWB Development Platform Receiver Interface 47

5-1 UWB Communication Channel Testing Setup 54
5-2 Sample Chipscope Output Window 54
5-3 Plot of Sampled UWB Waveform 55
5-4 Graph of Sampled UWB Waveform 56

List of Tables

2.1 Error Codes and Causes 25

5.1 Transmission Statistics for Revised UWB Signal Processing Code . . 57

Chapter 1

Introduction

The amount of research in the field of Ultra Wideband (UWB) communication has increased since the FCC approval of Ultra Wide Band for commercial applications. The FCC limits transmission using UWB to a center frequency between 3.6 to 10.6 Ghz. The maximum power spectral density is limited to -41dBm/Mhz and the bandwidth must be at least 500 MHz[1][2]. The increase in research efforts around UWB as a communication protocol are likely the result of a wireless technology boasting high potential bandwidth and low power consumption.

Physical access to UWB communication channels allow analysis of transmitted data. Without access to an actual UWB communication channel, it is difficult to develop the signal processing algorithms required for retrieving data from UWB signals. As popularity in UWB research grows, it would be beneficial to allow other researchers access to transmit and analyze data across an actual UWB communication channel so that they may also develop signal processing algorithms for recovering data from UWB signals.

After the initial FCC approval of UWB for commercial applications, two UWB communication systems were constructed by the MIT UWB researchers. These systems served as a starting point to demonstrate the proof of concept and were concerned primarily with transmission of data using UWB and not did not measure packet detection rate, packet loss, or overall bandwidth. The first system, constructed entirely using discrete components, demonstrated transmission and reception only in baseband[1]. The second system utilized discrete off the shell components to demonstrate data transmission with a center frequency of 5.355 Ghz. Baseband signals were converted up to 5.355 Ghz, transmitted, and converted back down to baseband for analysis.

Now, as the next iteration of UWB communication systems are under development, tools should be available for determining more than single packet reception. Since it has already been demonstrated that UWB communication systems can transmit data, the next iteration of UWB communication systems should be developed and characterized for performance. Furthermore, the next UWB communication system should be developed maximizing bandwidth and packet detection while minimizing packet loss. To fine tune and evaluate the next iteration of UWB communication systems, a development platform dedicated to testing and debugging UWB systems is required. With a dedicated UWB development platform, new systems can be fine tuned and more compelling end-to-end scenarios can be demonstrated.

1.1 Previous work

Recently, Fred Lee, David Wentzloff, and Raul Blazquez created a functional one-way UWB communication channel utilizing discrete components. UWB pulse information is generated using MATLAB and sent to an arbitrary waveform generator using the local ethernet network. The arbitrary waveform generator creates a modulated baseband signal from the MATLAB file. The output from the arbitrary waveform generator is passed to a passive mixer for up conversion to 5.355 GHz. The output from the mixer is transmitted to a series of discrete components which down convert the signal to baseband. The resulting baseband signal is sampled with an ADC connected to a computer and the resulting samples are processed using MATLAB to recover the original data. A diagram for this system is shown in Figure 1-1.

There are three main limitations to this system. First, the system is not real time. Second, the system utilizes an arbitrary waveform generator to generate the input signal. The interface to the arbitrary waveform generator was not specifically

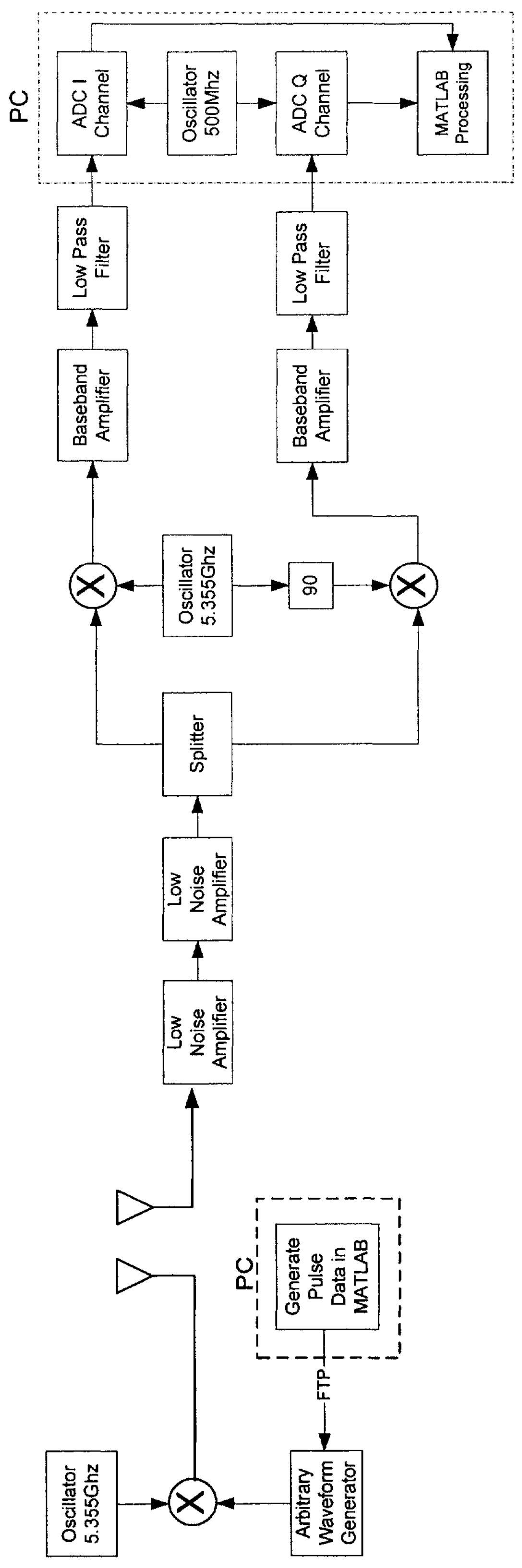


Figure 1-1: Discrete UWB Communication Channel

design for this particular application and the user interface makes setting and sending particular data very difficult and time consuming. Third, the system only provides one way communication. The system does however, allow for capturing raw UWB packets so that UWB demodulation processing algorithms may be developed.

1.2 Goals

The primary goal of this thesis is to design and implement a web interface to allow web users to send and receive data from an actual UWB channel. This will allow other researchers to develop UWB signal processing algorithms based on real data from an actual UWB communication channel. The secondary goal of this thesis is to implement a UWB development platform based on a FPGA, dedicated to the rapid development and characterization of UWB communication systems. Using this platform, users will be able to debug and fine tune current UWB communication channels to fix implementation errors and maximize performance.

Chapter 2

Web Interface

The primary purpose of the web interface is to extend the functionality of a current UWB communication channel to select web users. This will allow researchers to analyze data transmitted over a real UWB channel. The results obtained from the web interface will allow web users to develop UWB signal processing algorithms for recovering data transmitted in a UWB channel. Furthermore, the web interface also serves the MIT UWB laboratory by compiling a large collection of transmitted data, which can be used for UWB signal processing algorithms.

2.1 Design Goals

The design goals for the web interface are as follows.

- Specification of transmission data - The web interface must allow users to upload sample data to send over the UWB communication channel.
- Analysis of received data - The web interface must allow users to download the channel raw channel data from the ADC.
- Scalability - The use of the web interface by one user should not deny the use of the web interface to another user. Furthermore, the web interface should be designed to accommodate a number of users simultaneously.

- Fairness - The web interface should transmit the data and report the results to the user in the order in which the data was uploaded. In other words, the user who first uploads data to send across the channel will receive the results first.

- Customization - The web server should be complete configurable, allowing the system administrator to specify options such as storage location, maximum file upload size, and access list.

- Logging - The web server should log all activity and explicitly log all errors in the system separately.

- Storage of results - The web server should allow for storage of all results to a storage location accessible through the web so that users may access their data at any time. The results will be stored permanently to also allow the MIT UWB researchers to view previously sent transmission data.

- Access control - The system administrator should be able to limit access to the system by email address, allowing universal access, or select access to a specific list of email accounts. This is important as the tool will not be open to the public due to potential licensing issues.

- Basic security - The web interface should be immune to only the most basic forms of mis-use such as improper form completion.

2.2 Hardware Requirements and Setup

The web interface runs on a Windows XP Pro computer, using the Microsoft Web Server. The web interface uses a global IP address which makes the web interface accessible to any web-user[1].

To sample the received UWB signals, the PC contains an acquisition card that supports simultaneous two channel high speed ADC conversion. The PCI card used

[1]If access list is enabled, only select users will have access.

in the design of the web interface is an Acqiris PCI ADC capture card capable of 1Gsample/sec[2].

To send data using UWB, the data must first be transferred from the PC to the hardware responsible for pulse generation and transmission. David Wentzloff has constructed a discrete pulse generator with a 3-wire serial interface which will create a UWB signal centered around an input frequency. The input frequency comes from a local oscillator reference frequency which is set at 5.355 GHz.

The PC interfaces with the discrete pulse generator through the use of a XEM3001 FPGA board[3]. This FPGA shall be known as the transmission FPGA and connects to the PC USB interface. The transmission FPGA also connects to the discrete pulse generator. A programmable crystal oscillator is programmed to produce a 5.355 GHz carrier frequency and is connected to the discrete pulse generator.

The output of the discrete pulse generator is connected to an antenna. On the receiving end, another antenna is connected to a low noise amplifier followed by an additional amplifier. The output from the second low noise amplifier is split and one output from the splitter is multiplied by the 5.355 GHz carrier frequency. The other output from the splitter is multiplied by the 5.355 GHz carrier frequency 90 degrees out of phase. The resulting signals are then fed through baseband amplifiers and low pass filtered. The output from the filters is then fed into the two input channels of the ADC[4] as shown in Figure 2-1. The current implementation does not allow the user to modify any transmission parameters such as pulse repetition frequency or modulation scheme. To allow users to select modulation scheme and other transmission options, the code running on the transmission FPGA and the discrete signal pulse generator would need to be altered.

[2]http://www.acqiris.com/

[3]The Opal Kelly XEM3001 board is explained further in Chapter 4

[4]Both channels are independently sampled using the same sampling clock

2.3 Architecture

Many architectures for the design of the web interface were considered when attempting to meet the design goals of the system. The main limitation of the web interface was that only one process may use the discrete pulse generator and the capture card at the same time.

The initial design spawned a new process when a user would activate the web interface by uploading a data file. This process would obtain a reference to the control FPGA and transmit the data to the transmission FPGA. The process would communicate with the transmission FPGA and instruct the transmission FPGA to send the data to the discrete pulse generator. The process would then initiate a capture using the Acqiris capture card. Once the capture was complete, the process would create a series of webpages displaying the results. Once the process finished the creation of the web-formatted results, the process would direct the user to the index of the result pages and halt execution.

This design was rejected due to scalability. Under this design, two users could not use the system at the same time. A given user would have to wait for another user to completely finish before uploading a file. If a new user did not wait for the current user to finish, the new user might gain control of the discrete pulse generator during the time when the first user was sampling the signal.

To protect the initial system against multiple simultaneous users, a software locking scheme was designed. Again, a new process would spawn each time a user requested to upload a file. This process would then check a lock file to determine if the system was in use. When a process was finished, the process would delete the lock file. If the lock file did not exist, the process would create the file, indicating that the system was busy. When other new processes would spawn, the process would check the lock file to verify that the system was free. If the system was busy, the processes would wait until the lock file was deleted. To address the issue of fairness, a file containing a process ID could be updated such that each new process could open the file,obtain a process ID, and increment the process ID so that the next process would have a different process ID. The problem with this design is that file access is not atomic, so multiple processes could simultaneously open a file and obtain the same process ID, defeating the purpose.

Instead of implementing atomic file reading and writing operations to fix the issue, the final design split the responsibility of user interaction and backend processing into two separate processes known as the user-handler and the backend processor. The backend processor would run in the background and process requests in the order which they were received while the user handler would manage the uploading of files and re-direction of the user to the web-formated results. Using this implementation, locks are not necessary and it is guaranteed that only one process will ever have control over the discrete pulse generator and ADC at a given time. A block diagram of the final structure is shown in Figure 2-1.

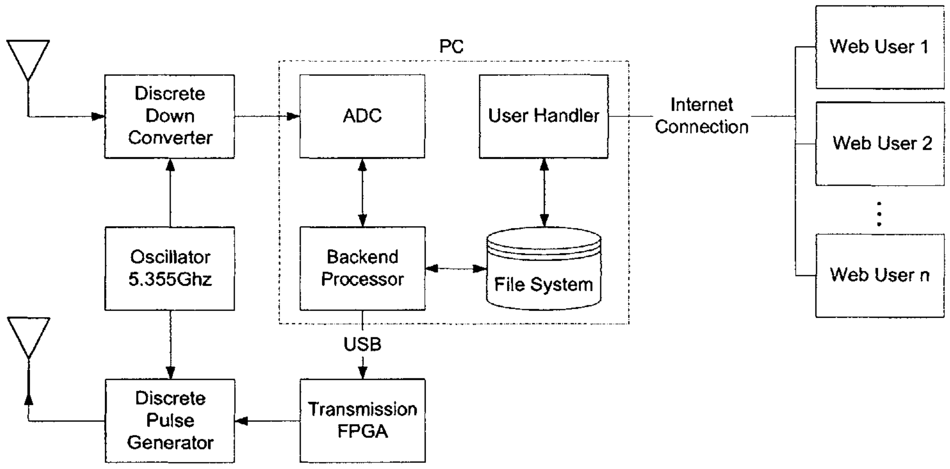
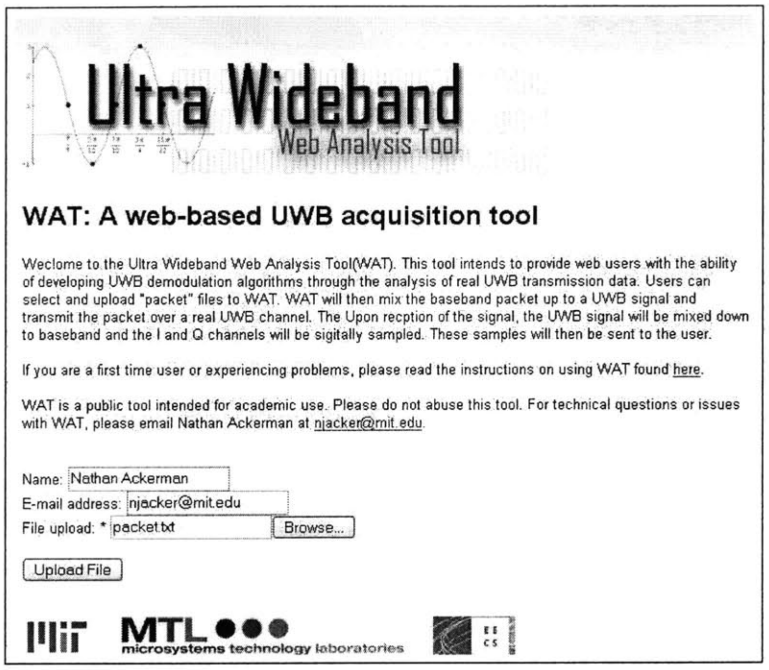


Figure 2-1: Web-Based Interface

2.3.1 User Handler

The role of the user handler is to interface with the user. More specifically, the user handler queues incoming requests to use the web interface for the backend processor and creates/redirects the user to a web-formatted page of results. The web interface contains a simple yet functional graphical user interface (GUI) as shown in Figure 2-2. The web interface GUI allows users to enter their name, email, and a text file representing the desired data to send across the UWB channel. The text file must be a series of ASCII one and zero characters delineated by the space character. Failure to send a text file in this format will result in the display of an error.



WAT: A web-based UWB acquisition tool

Weclome to the Ultra Wideband Web Analysis Tool(WAT). This tool intends to provide web users with the ability of developing UWB demodulation algorithms through the analysis of real UWB transmission data. Users can select and upload "packet" files to WAT. WAT will then mix the baseband packet up to a UWB signal and transmit the packet over a real UWB channel. The Upon recption of the signal, the UWB signal will be mixed down to baseband and the I and Q channels will be sigitally sampled. These samples will then be sent to the user.

If you are a first time user or experiencing problems, please read the instructions on using WAT found here.

WAT is a public tool intended for academic use. Please do not abuse this tool. For technical questions or issues with WAT, please email Nathan Ackerman at njacker@mit.edu.

Name: Nathan Ackerman
E-mail address: njacker@mit.edu
File upload: * packet.txt [Browse...]

[Upload File]

Figure 2-2: Screenshot for Web Interface

After filling out the form, the user will press the submit button. Pressing the submit button will create an instance of the user handler application on the web interface PC. The Common Gateway Interface(CGI) is used to pass information from the web form to the new instance of the user handler. The user handler application is written in the C programming language and will examine the validity of the input fields and the uploaded file. If the access list is not enabled, neither the name or email field is required for proper operation. However, if the access list is enabled, a proper email address will be required for use.

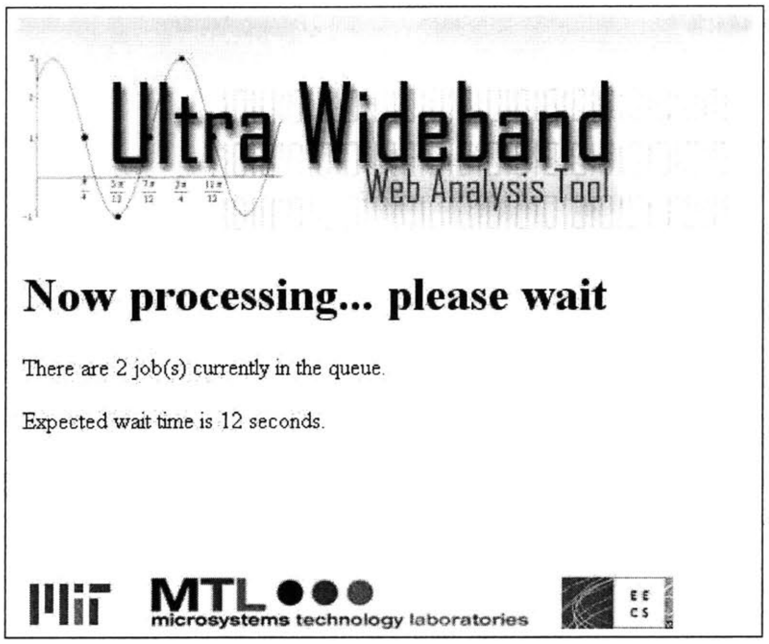
After checking validity of the input fields and the uploaded file, the user handler will create a process identification number (process ID). The process ID is a concatenation of the current date and a random 6 digit number. So for example, a process ID for a user on May 1, 2005 would look something like 05012005_637485. The purpose of the process ID is to uniquely identify each request to send a packet. The random 6 digit number allows for a potential 1 million unique packets to be sent everyday. This limit was chosen arbitrarily and could trivially be extended by using a larger random number.

The user handler creates a directory with the same name as the process ID and copies the original input file to the directory. In addition, the user handler also copies the input file to the processing folder and renames the file with the process ID. The user handler writes the system time and date along with the user, email, and filename to the user log. The user handler also creates an index page in the created directory with links to the output files. While the output files have not been produced, their location is fixed, so making a link is valid. The user handler then examines the number of files currently in the "to process" folder and makes an estimation of the time it will take to process the recent request by multiplying the number of files by the average process time. Experimentally, the average process time was 6 seconds from start to finish of the backend processor with the majority of that time spent on disk access. The user handler then redirects the user to a page which displays text instructing the user to wait for the expected amount of time. In addition, the text also displays how many other requests are pending. This page is written in html and will automatically redirect the user to the html output page after the expected wait time, as calculated by the number of requests pending times 6 seconds. A sample screenshot of the waiting screen can be seen in Figure 2-3.

After the expected amount of wait time, the user will view a html-formatted page with links to the original input data file as well as the raw samples of both the I and Q channels from the ADC as seen in Figure 2-4.

Access List

The access list is a file maintained by the system administrator and contains a space delineated list of email addresses who have access to the system. Using the access list is a administrative option that, when activated, limits access to the system to only a list of email addresses. if the access list is turned on, and a user not on the access



Now processing... please wait

There are 2 job(s) currently in the queue.

Expected wait time is 12 seconds.

Figure 2-3: Screenshot when Waiting

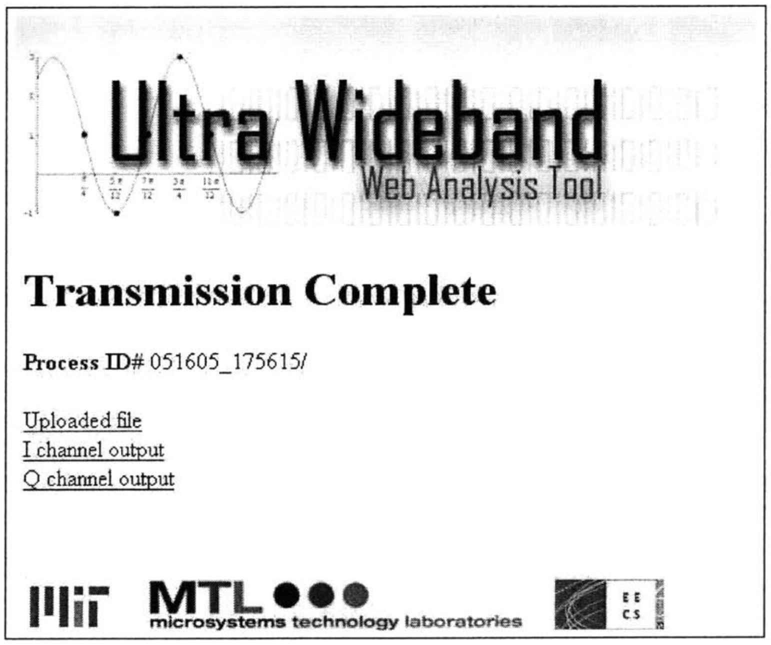
list tries to use the system, the user will be presented with an error.

Error Types

The user handler currently checks for 5 types of errors as shown in Table 2.1. A "file too large" error is displayed if the user attempted to upload a file with a filesize larger than the maximum filesize allowed[5]. An "incorrect file type" error will occur if the uploaded file was not in ASCII format. A "file does not exist" error will occur if a user mistypes the path or name of a file. When this happens, no file will be uploaded to the user handler and the error will be displayed. A "generic error" is designed for future modifications, however it is currently be displayed if a user attempts to upload an empty file or a file that with an incorrect file format. Lastly, an "access denied" error will be displayed if a user not on the access list attempts to use the web interface.

When an error is detected, regardless of the type, the user handler will create a web-formatted error page displaying the error code as shown in Figure 2-5 and

[5]As specified by the system administrator



Transmission Complete

Process ID# 051605_175615/

Uploaded file
I channel output
Q channel output

Figure 2-4: Screenshot of Successful Transmission

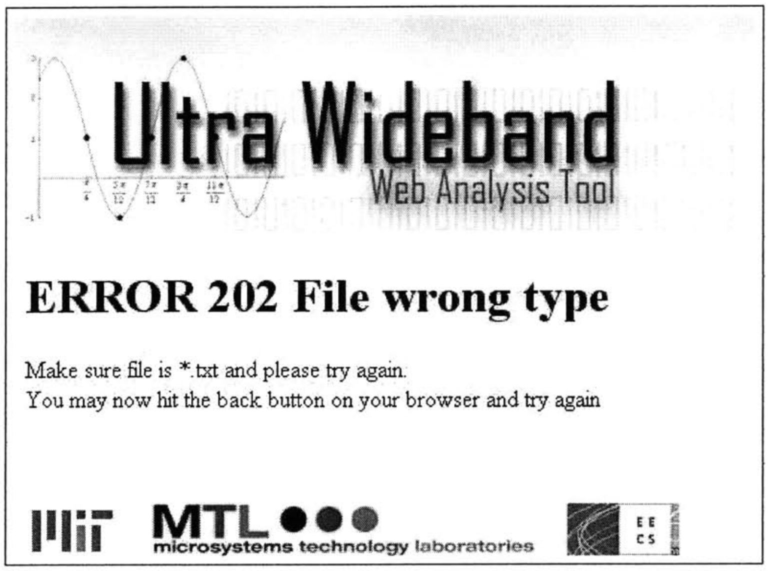
Table 2.1: Error Codes and Causes

Error Code	Cause
201	File too large
202	Incorrect file type
203	File does not exist
204	Generic error
205	Access denied

redirect the user to that page. At that point, the user will be able to press the back button on the web browser and try to use the system again.

Logs

Logs will be created and appended by the user-handler so that the system administrator will have a record of the system use and errors. Currently, two separate logs are implemented. A user log is appended when a new process is requested indicating, the user name, email, time, process ID, and filename. When an error is encountered, the error log is appended with the error code, the user name, email, time, date, and



ERROR 202 File wrong type

Make sure file is *.txt and please try again.
You may now hit the back button on your browser and try again

Figure 2-5: Screenshot of Error

process ID.

2.3.2 Background Processor

The background processor runs when the PC running the web interface boots up. When the background process first starts, the backend processor searches the PC for the transmission FPGA and obtains a software reference to the first available device. The web interface PC should have exactly one transmission FPGA connected to it, so issues concerning multiple or disconnected transmission FPGA modules result in error and such circumstances are assumed not to exist. After obtaining a reference to the connected transmission FPGA, the backend processor will program the transmission FPGA over the USB interface by setting the system clock frequency and configuring the transmission FPGA with web based transmitter bit stream. After this is complete, the background processor is ready to process requests.

The background process will then monitor the contents of a specific folder on the local PC through continuous polling of the folder contents. The folder represents the queue for packets to be sent across the UWB channel. When the folder is empty, there are no packets that are to be sent. When there are packets to be sent over the UWB channel, the folder will contain a file for each packet specifying the data to be sent. As an optimization, when the folder is empty, the process will sleep for 10 milliseconds as to not use the CPU when the CPU is not required. When the background process detects that the "to process" folder is not empty, the process will initiate a packet transmit and capture.

When the background process detects that the folder to process data is not empty, the background process will open the oldest file first. By opening the oldest file first, the backend processor promises to be fair and process the oldest request first. Since the backend processor is the only application interfacing with the discrete pulse generator and capture card, it can be guaranteed that only one packet is transmitted and captured at a time.

2.3.3 Packet Transmit and Capture

To capture a packet, the PC will fill an input buffered pipe on the transmission FPGA with the packet data[6]. The PC will then initiate a capture on the Acqiris card before transmitting the data over the UWB channel to ensure capture of the packet. Interfacing with the Acqiris capture card is accomplished in the C programming language through function calls to a statically linked library. Setup for a two channel capture requires initializing the card, specifying the capture duration in seconds, and then initiating the capture. Immediately after initiating the capture of the packet, the PC will send a trigger to the transmission FPGA instructing the transmission FPGA to transmit the contents of the buffered pipe to the discrete pulse generator. The data from the Acqiris card will be read and stored to a file for analysis on the local filesystem. While this implementation is guaranteed to capture the transmitted packet, the acquisition will also contain extra "silence"[7]. However, the implementation was chosen because it was the simplest implementation of guaranteeing that a given capture will contain the transmitted data.

[6]A more complete overview of the XEM3001 and its functionality is provided in Chapter 4

[7]A long series samples with only noise present

Chapter 3

UWB Platform Architecture

The primary purpose of the UWB development platform is to allow rapid prototyping and performance characterization of a UWB communication channel. Also, the UWB development platform aims to provide testing of logic designs before ASIC[1] fabrication. In order to accomplish this, the UWB development platform must natively contain all components essential for transmission and reception of data over UWB. Furthermore, the UWB development platform must be modular to allow replacement of modules without loss of functionality for testing and characterization.

Every received UWB signal follows the same path and the flow of information can be described by Figure 3-1. Incoming signals are down converted to baseband, digitally sampled, fed into signal processing, and the resulting data is transfered to the PC. Similarly, every transmitted UWB signal is transfered from PC, up converted to a center frequency of 5.355 GHz, and transmitted as shown in Figure 3-2.

Combining both transmission and reception together in one design, a total of 6 functional modules are required for a UWB development platform and are described briefly below.

- Down converter - Incoming UWB signals must first be down converted to baseband before the incoming signal may be digitally sampled.

- Analog to Digital Converter - To sample the down converted signal, an analog

[1]Application specific integrated circuit

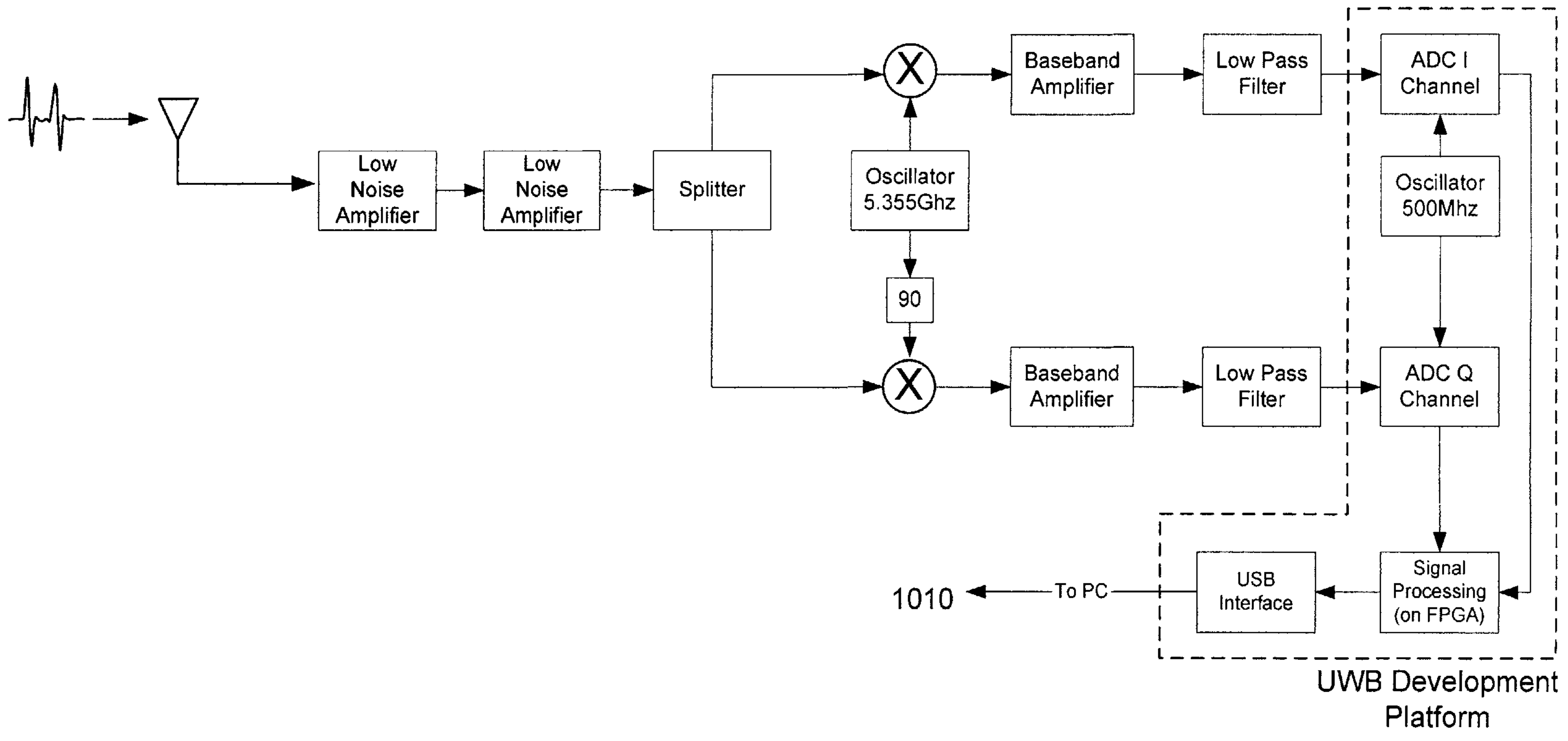


Figure 3-1: Functional Receiver Model

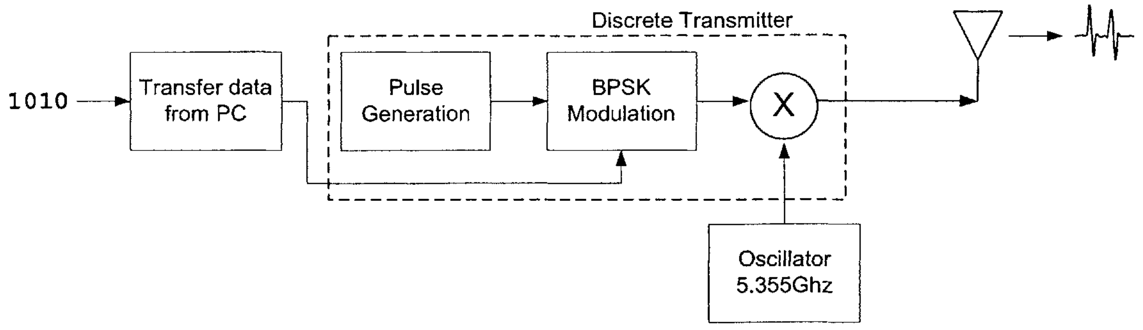


Figure 3-2: Functional Transmitter Model

to digital converted is required to convert the analog waveform into a series of digital samples for extracting the transmitted data.

- Signal Processing - After a series of digital samples representing an analog waveform are acquired, the signal processor extracts the transmitted data from the series of digital samples.

- Transfer of Received Data - Once the data is retrieved from the signal processing module, the data must be transferred to a computer.

- Transmission of Data to Send - The purpose of this module is to provide transportation of the desired data from a PC to the development platform.

- Up converter - The up converter will up convert the baseband signal to 5.355 GHz before transmission over the air.

Regardless of the particular implementation of the modules, general information flow will be similar. The down converter will continuously down convert all received signals to baseband. The baseband output of the down converter will be constantly sampled by the ADC at 500 MHz. The resulting digital samples will be fed into the signal processing module. If a UWB signal is detected, the signal processor will retrieve the data bits from the UWB signal and send them to the module for transportation to the PC. At the same time, the module responsible for receiving desired data to transmit will receive incoming transmission requests from the PC and send the appropriate data to the discrete pulse generator for creation of a UWB signal. With all functional modules combined together, the entire functional diagram for the UWB development platform can be seen in Figure 3-3.

3.1 UWB Development Platform Hardware

As outlined in the functional overview, the hardware used in the design of the UWB development platform must provide the functionality of the 6 functional modules. The hardware selected for the implementation of the UWB development platform was chosen to maximize configuration options as well as performance. A connectivity diagram of the hardware is shown in Figure 3-4. Hardware located on top of one another in the figure indicates electrical connections.

3.1.1 Standalone Carrier

The UWB development platform consists of many separate printed circuit boards connected together. The physical backbone of the physical architecture is a Sundance SMT118 carrier board and known as the standalone carrier. All circuit boards required for the UWB platform are reside on the standalone carrier except for the down converter, which is connected only by a coaxial cable. The standalone carrier contains 3 standard TIM sites denoted CPU, I/O 1, and I/O 2. The standalone carrier has several power supplies and draws power from a single 12V supply. Locally, the standalone carrier converts and distributes supplies of 3.3, 5, 12, and -12 volts. All connected modules draw the appropriate power directly from the standalone carrier[7].

The standalone carrier implements a hardware reset signal that is connected to all TIM sites. Pressing a button on the carrier causes a synchronized global reset to all connected modules. In addition, the standalone carrier implements the com port interface. The com port interface is a bi-directional communication protocol to allow all for communication between connected modules. The standalone carrier implements interconnections between all three sites so that each site may communicate directly with any other site using the com port protocol. Each I/O TIM site has 2

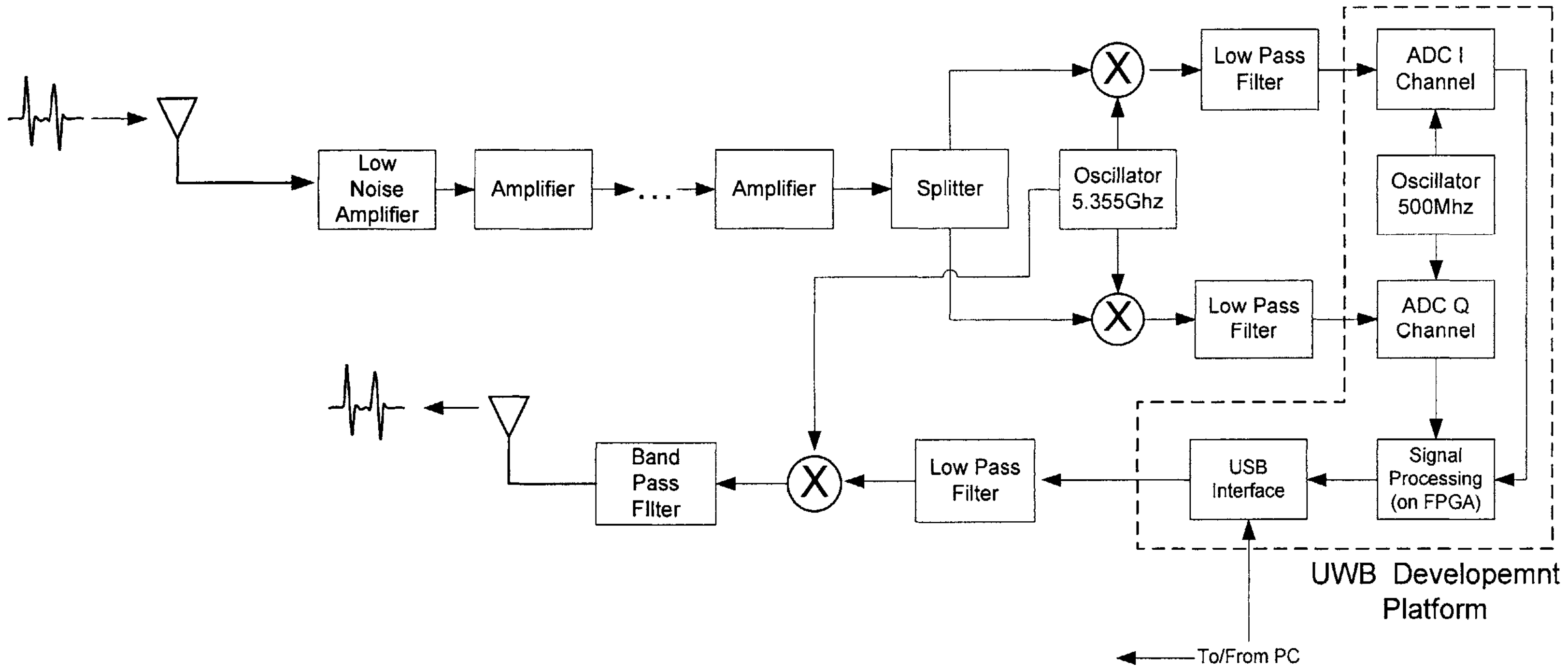


Figure 3-3: Complete UWB Development Platform Model

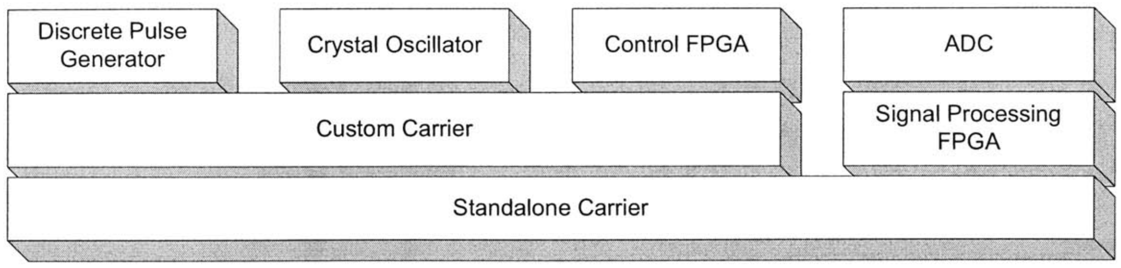


Figure 3-4: UWB Development Platform Hardware Connectivity

bi-directional com ports and the CPU sit has four bi-directional com ports.

3.1.2 Signal Processing FPGA

The CPU TIM site on the standalone carrier holds the Sundance SMT391-VP which serves as the reprogrammable signal processing module for the system. The Sundance SMT391-VP is the name used for the combination of two separate Sundance boards which are the SMT338VP30-6 and the SMT391. The SMT338VP30-6 contains a Virtex2Pro VP30-6 FPGA. All code to retrieve UWB data from digitized sample resides on this FPGA and the FPGA will be commonly referred to as the signal processing FPGA. The FPGA is large with an effective 1 million reprogrammable gates. A large FPGA allows maximum flexibility in design as smaller FPGAs could potentially limit the complexity of signal processing schemes. The signal processing FPGA also has a 16-bit microprocessor connected to it. The microprocessor is used to provide a non-volatile interface for reprogramming the signal processing FPGA. The signal processing FPGA board also has connections for two Sundance high speed buses which connect directly to the FPGA. The Sundance high speed buses are physical interfaces specifically designed for high speed transfer of data[5].

The SMT391 is also manufactured by Sundance Corporation and contains a high speed analog to digital converter capable of digitally sampling two independent input signals at 1 Gsample/sec[3]. For the remainder of the document, the SMT391 will be referenced as the ADC.

The SMT391 and the SMT338 were designed to interface with one another. The connection between the SMT391 and the SMT338 provides the signal processing

FPGA with direct access to the output from the ADC. The ADC is configured through control registers located on the ADC. Configuration of the ADC takes places over a serial interface implemented by the signal processing FPGA. Lastly, the signal Processing FPGA is connected to the com port so that communication between the signal processing FPGA and other modules is possible using the com port interface.

3.1.3 Control FPGA

The XEM3001 provides an interface between the UWB development platform and a PC over the USB bus using USB version 2.0. The XEM3001, manufactured by Opal Kelly, contains a Spartan3 FPGA and a chip implementing the USB 2.0 interface. The Spartan3 is a smaller FPGA than the signal processing FPGA however only a small amount of space is required to store the control logic for the UWB development platform. For the remainder of this document, the XEM3001 will be referenced as the "control FPGA"[4].

The roles of signal processing, overall control, and USB interface are divided between two separate FPGAs where the signal processing FPGA is dedicated to running the signal processing code and the control FPGA is dedicated to running the overall control and implementing the USB interface. While it would be possible to implement all three roles of signal processing, USB interfacing, and control on the same FGPA, two separate FPGAs were used. The justification for using two separate FPGA is to maximize the available space for signal processing algorithms. If the same FPGA were used for all three purposes, the logic required for control and the USB interface would further limit the size, and thus complexity, of signal processing algorithms.

3.1.4 Custom Carrier

The second and third TIM sites contain a custom carrier board to connect the control FPGA and discrete transmitter to the standalone carrier. The custom carrier was designed by Kyle Gilpin specifically for the UWB development platform. The custom carrier routes the com port connections from the standalone carrier TIM sites to the control FPGA so that the control FPGA may communicate with the signal processing FPGA using the com port. The custom carrier also contains a connector for a Sundance high speed bus (SHB) so that the control FPGA can interface with the signal processing FPGA directly over the Sundance high speed bus. The custom carrier also supports connections between the control FPGA, the discrete pulse generator, and the crystal oscillator.

3.1.5 Crystal Oscillator

A programmable crystal oscillator is connected to the custom carrier which in turn, is connected to the control FPGA. This allows the control FPGA to program the crystal oscillator. On power up, the crystal oscillator is programmed to output 5.355 GHz as a reference frequency for the discrete pulse generator and the down converter. Programming of the crystal oscillator is accomplished through a serial interface implemented by the control FPGA.

3.1.6 Discrete Down Converter

The down converter, designed by Fred Lee, is a series of connected printed circuit boards responsible for down converting incoming signals centered around 5.355 GHz to baseband. While the down converter is essential to the operation of the UWB platform, the down converter does not reside on the standalone carrier and is only connected to the UWB platform using a coaxial cable.

3.1.7 Discrete Pulse Generator

The discrete pulse generator, designed by David Wentzloff, converts serial digital data into a UWB signal centered around an input frequency. The discrete pulse generator first generates a baseband pulse train modulated with the desired data using BPSK. The baseband signal is then up converted around the input frequency. The input frequency of discrete pulse generator is connected to the programmable and the serial data interface is connected to the control FPGA.

Chapter 4

UWB Platform Design

4.1 Control FPGA

The control FGPA is responsible for the control signals to coordinate the transfer and reception of UWB signals, initializing the UWB development platform on a power up or system reset, and providing a USB interface to the UWB development platform.

A USB chip connected to the control FPGA permits communication with any USB capable PC. The control FPGA is connected to a programmable crystal oscillator which is used for the local control FPGA system clock. The oscillator is connected to 5 programmable dividers used to generate up to 5 distinct clock signals for use by the control FPGA. The provided C-libraries allow a connected computer to program the FPGA, change the system clock frequency, and communicate with the FPGA while the FPGA is running[4].

4.1.1 Opal Kelly interface

Opal Kelly has designed and implemented three distinct types of Verilog modules designed for communication between the control FPGA and a PC over the USB bus. In addition, Opal Kelly has also compiled a corresponding C-library for interfacing with the Verilog modules. These modules and the library are designed to provide an abstraction barrier around the USB interface so that USB communication is possible by running a Verilog module on the FPGA and referencing C programming language function calls on the PC[8].

The three types of communication modules designed by Opal Kelly are known as wires, triggers, and buffered pipes. Each type of communication is unidirectional so that a given module will only send or receive data. For the remainder of this document, signals going from the PC to the control FPGA will be referenced as as input signals and signals emanating from the control FPGA destined for the PC will be referenced as output signals.

The first type of communication is a wire, which is asynchronous. Querying the value of a wire is a simple asynchronous operation on either the PC or the control FPGA. On a PC, checking or setting the value of a wire requires a function call. In the Verilog code running on the FPGA, checking or setting the value of a wire requires referencing a specific register.

The second type of communication is a trigger. Triggers are synchronous signals that, when activated, remain high for one cycle of the clock when used. Triggers ensure that input signals from the PC are synchronized to a known clock on the FPGA and eliminate the need for separate synchronizers for input signals. To ensure that the trigger is synchronized with the correct clock, the synchronized clock is one of the parameters when instantiating the trigger in Verilog. Since there is not a real clock that C programs are synchronized to, the programmer must call functions to query the value of the trigger at a given time. When the control FPGA initiates an output trigger, the trigger data is placed in a FIFO on the PC automatically. The PC may get the value of a trigger by reading values from the FIFO in the order they were received. Each successive function call will obtain the next value from the FIFO.

Buffered Pipes are the last type of communication module. Buffered pipes serve as synchronous FIFOs. The data in and data out ports have separate clock signals so that the data may be loaded and unloaded into the FIFO synchronously on both ends.

The Control FPGA has three distinct interfaces with respect to the UWB development platform which are as follows.

- Reprogram the signal processing FPGA and initialize the ADC
- Transmission of UWB signals
- Reception of UWB signals

4.2 FPGA reprogramming

The control FPGA itself is programmed over the USB interface using a compiled bit stream. The provided C-library provides a function to program the control FPGA with a bit stream as input. Once the control FPGA has been programmed over the USB interface, the control FPGA is automatically reset and runs.

The signal processing FPGA can be programmed in two different ways and must be reprogrammed every time the UWB development platform is powered on. The signal processing FPGA may be programmed using the JTAG interface or though a connected MSP430 microprocessor. A JTAG connector can be found directly on the signal processing FPGA board for JTAG programming. Programming using the MSP430 microprocessor requires communicating the desired programming file to the MSP430. By configuring the control FPGA to program the signal processing FPGA through the MSP430 microprocessor, the need for a JTAG cable is eliminated[5].

Sundance has provided the software that runs on the MSP430. The MSP430 microprocessor uses non-volatile flash memory to store its programming, so it is never reprogrammed. The code running on the MSP430 implements a the com port protocol for programming the signal processing FPGA. A start key, configuration data, and an end key are sent to the MSP430 using the com port interface to program the signal processing FPGA. The com port is connected to both the MSP430 and the signal processing FPGA but while the FPGA is not programmed, the FPGA will ignore signals on the com port. Once the MSP430 is finished programming the signal processing FPGA, the MSP430 will ignore the com port so that the com port may be used for communication between the signal processing FPGA and the control FPGA. A diagram outlining connections specific to programming between the control FPGA,

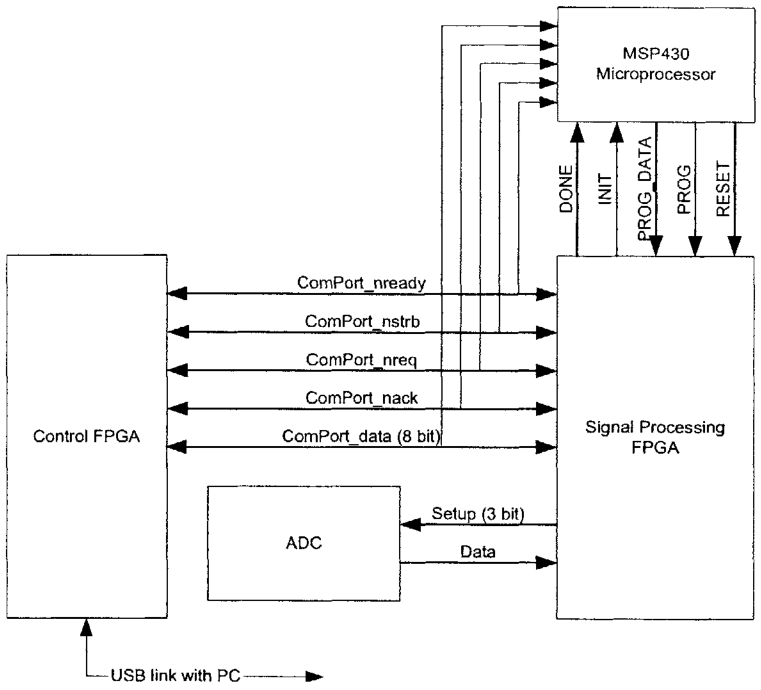


Figure 4-1: Wiring Diagram for Programming Interface

the MSP430 microprocessor, the ADC, and the signal processing FPGA is shown in Figure 4-1.

4.2.1 Com Port Interface

The com port interface is the primary communication protocol for inter-module communication on the UWB platform. The com port interface is used for initial configuration of the signal processing FPGA and communication between the control FPGA and the signal processing FPGA. The com port interface is a simple bi-directional interface implemented with 8 data lines, and 4 control lines. Port arbitration is accomplished through the use of a token and handshakes guarantee the reception of data as well as the transfer of the token.

The com port interface defines communication between a transmitter and a receiver. As such, on a global reset, one side of every communication link must default to receiver and the other must default to transmitter. After a reset, the receiver may request a token change to transmit.

The control lines are labeled nstrb, nack, nreq, and nready and the control logic are all active low signals. The transmitter is responsible for driving the data, nstrb, and nack signals while the receiver drives nready and nreq signals. The signals nready, and nstrb are used for data transfer while the signals nreq and nack are used for transferring ownership of the token.

The transmitter sends data by placing valid data on the 8-bit data bus and then driving nstrb low. When the receiver detects that nstrb has gone low, the receiver will latch the data on the 8 bit data bus. After latching the data, the receiver will drive nready low. The transmitter will then wait until the nready line has been pulled low by the receiver. Once the transmitter has detected that the nready line has been pulled low, the transmitter will pull the nstrb line high and place the next 8 bits of data on the data bus. The receiver will detect the transition on the nstrb signal and will pull nready high once again when it is ready to receive the next word. Only when nready has been pulled high by the receiver will the transmitter pull nstrb low again, indicating the start of another 8 bit datum. Data is sent form the transmitter to the receiver in 32 bit words such that any transfer between a transmitter and receiver is not to be interrupted except on 4 byte boundaries. Only after successful transmission of a 4 byte word can the receiver request the token[7].

To change token ownership, the nreq and nack signals are used. The receiver may, at any 4 byte boundary in the transmission, request token ownership. The receiver will request ownership by pulling the nreq line low. The transmitter will observe the nreq line transitioning low and will pull the nack line low in acknowledgement. The receiver will then pull the nreq line high and wait until the transmitter responds by pulling the nack line high. Once nack has been pulled high, the token is transferred and the roles of the transmitter and receiver are switched.

4.2.2 MSP430 Programming

After power on or a global reset[1], the MSP430 will hold the FPGA in reset and monitor the communications port for a start key. As mentioned previously, programming

[1]Global resets originate from the standalone carrier and are sent to all TIM sites

the FPGA through the MSP430 requires a start key followed by programming data followed by an end key. All data except for the start key or the end key will be ignored by the MSP430. After proper reception of the start key, the MSP430 will forward all incoming data to the FPGA for programming until the end key is received. When the end key is received, the MSP430 will ignore the com port. By default on power up, the MSP430 will initialize as a com port receiver so that any transmitter may start transmitting after a reset.

After a reset, the FPGA will drive the FPGA DONE [2] low, indicating that the FPGA is not programmed. Once the MSP430 receives the start key, the MSP430 will activate the FPGA PROG[3] pin and will wait for the FPGA INIT[4] pin to transition low. Once the FPGA INIT pin transitions low, the FPGA will be ready for programming. The MSP430 pulls the FPGA PROG pin high and waits for the INIT pin to transition high. Once the INIT pin is high, the MSP430 programs the FPGA by passing the comport data straight to the programming pins of the FPGA. The FPGA DONE pin will transition high only after the FPGA has been properly programmed. Proper programming of the FPGA requires the correct number of bytes to be sent for programming. In addition, a CRC verification algorithm protects against programming the FPGA with erroneous programming files. If the transferred file does not pass the CRC checksum, regardless of how many bytes it receives, the FPGA DONE pin will not transition high.

When the FPGA DONE pin is high, the MSP430 will not transfer any data to the FPGA and will continue to hold the FPGA in reset until the end key is received. Proper reception of the end key will cause the MSP430 to drive the FPGA reset pin low, allowing the FPGA to run[6].

If a global reset is applied to the UWB platform after FPGA programming, the signal processing FPGA may be reprogrammed by sending a start key followed by a bit stream and an end key to the MSP430.

Lastly, while the UWB platform does not require the use of a JTAG cable to

[2]FPGA output pin indicating the programming status
[3]FPGA input pin to start programming
[4]FPGA output pin indicating readiness to begin programming

configure the signal processing FPGA, a JTAG cable may still be used. To program the device using a JTAG cable and any Xilinx programming tool, such as Chipscope Pro or iMPACT, may be used. Once the FPGA is programmed using the JTAG, the end key must be sent to the MSP430 to release the com port and take the FPGA out of reset.

4.3 Transmission Interface

The first step in transmission is for the PC to use a trigger to reset the input FIFO and clear any potential previous content on the control FPGA. The desired packet is then created on the PC as an array of bytes and transferred to the control FPGA using a buffered pipe. Once the packet is transferred into the FIFO, the PC sets the value of four wires, which provide two separate 16-bit signals to the FGPA as shown in Figure 4-2. The FIFO on the FPGA contains a bitwise replica of the desired packet. The two 16 bits signals are used to indicate the number of total bits in the preamble and the total number of bits in the packet. This allows for specific preamble and data length since the preamble and payload may be transmitted at different pulse repetition frequencies. Both 16 bit signals are concatenations of 2 separate 8 bit wires which interface from the PC to the Control FPGA. 16 bits are used to represent the payload length and preamble length which places a hard coded limit of preamble and payload length of 65,536 bits.

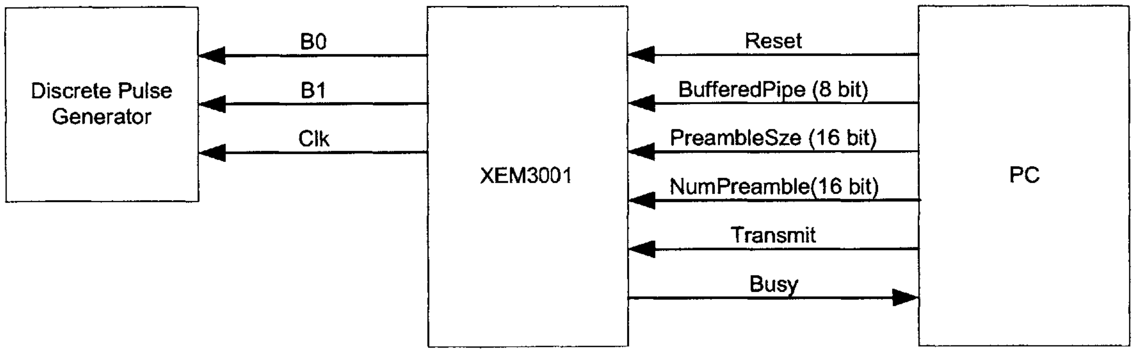


Figure 4-2: UWB Development Platform Transmission Interface

After the length of the preamble and packet is set on wires by the PC, a trans-

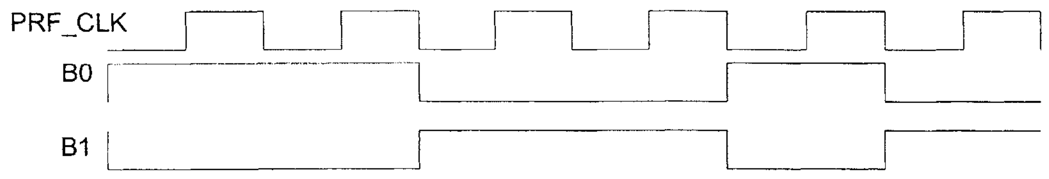


Figure 4-3: Timing Diagram using Discrete Transmitter Interface

mit trigger is sent from the PC to the control FPGA. When the transmit trigger is received, the control FPGA sets a busy wire going from the control FPGA to the PC to be high. The Control FPGA interfaces with the discrete pulse generator and sends the packet, adjusting for the separation of preambles and payload. After initiating a transmit trigger to the FPGA, the PC will suspend execution and wait until the PC observes the busy wire going low. When the busy wire goes low, the packet has been sent and the control FPGA is now ready to accept a new command.

The interface between the Control FPGA and the discrete pulse generator consists of three wires which are PRF_CLK, B0, and B1. Data is set up for the discrete pulse generator on the falling edge of PRF_CLK and sampled by the discrete pulse generator on the rising edge of the clock. On a positive clock edge, if B0 is high and B1 is low, a negative pulse will be sent. Conversely, on the rising edge, if B1 is high and B0 is low, a positive pulse will be sent. If B0 and B1 are both low during the rising edge of the clock, no pulse will be sent. Lastly, if B1 and B0 are both high during the rising edge of the clock, unknown results will occur. Since pulses are sent on the rising edge of the clock, the clock signal input to the discrete pulse generator is the maximum pulse repetition frequency. A sample timing diagram for transmitting a sequence of 0,0,1,1,0,1 can be seen in Figure 4-3.

4.4 Receiver interface

The ADC continuously converts the analog input signal into digital samples and this information is sent to the signal processing FPGA. The signal processing FPGA searches for the preamble in the data. After locating the preamble of a packet, the resulting decoded bits will be transferred to the control FPGA though the following specified interface as shown in Figure 4-4. The signal processing FPGA will drive 6 pins to the control FPGA which are clock, enable, and a 4 bit data signal. The enable signal will transition high at the start of decoded bits and will remain high until the signal processing FPGA is done sending valid data. On the rising edge of the clock signal, valid data will be present on the data pins for the control FPGA to latch.

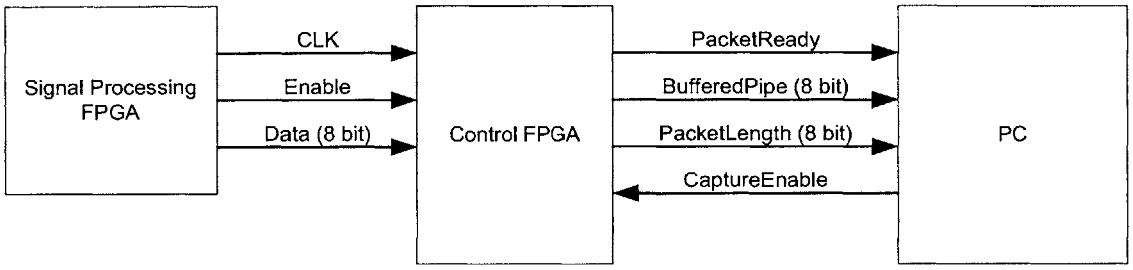


Figure 4-4: UWB Development Platform Receiver Interface

To control the capture of packets, there is an interface implemented between the PC and the control FPGA using three wires. An enable signal is set by the PC and monitored by the control FPGA. When the capture enable signal is high, the control FPGA will latch the incoming data from the signal processing FPGA into a buffered pipe for transfer back to the PC. When enable capture is driven low by the PC, the control FPGA will not latch any data. Once the enable signal from the signal processing FPGA goes low, the control FPGA signals the PC by driving a "done" wire high. The PC will periodically poll the wire and when the wire is high, the PC will set the capture enable wire low and read the data from the buffered pipe. During the transfer, the PC will monitor a third wire indicating the fullness of the buffered pipe as to guarantee that all of the contents are transferred to the PC. After the PC has transferred the contents of the buffered pipe, the PC will drive the capture enable signal high again so that the process may repeat. While this design does not allow the buffered pipe to store multiple packets, the design does allow for flexibility in packet size which proves useful for testing and debugging.

4.5 Boot Sequence

The boot sequence for the UWB platform consists of powering on the corresponding hardware and running the boot loader software. To power on the hardware, simply connect the external power supply to the wall and turn the power switch on. Various LEDs should turn on indicating that the boards have power. Once the UWB platform is powered up and connected to a USB port of a PC, the boot loader software may be started. Launching the software will cause the PC to search for a connected control FPGA device and obtain a reference to the first available control FPGA. The PC is expected to be connected to exactly one UWB platform at a time, so cases of selecting the incorrect control FPGA are not an issue. The remainder of the boot sequence will be accomplished by the boot loader software which invokes the following steps.

- Program the control FPGA
- Transmit a bit stream from the PC to reprogram signal processing FPGA
- Initialize and setup ADC
- Run the desired PC development software to receive and send packets

4.5.1 Programming the Control FPGA

Once the UWB development platform is powered on and connected to a PC, the boot loader will reprogram the control FPGA is using the provided C functions. When the functions return execution to the boot loader, the control FPGA will be programmed and running.

4.5.2 Programming the Signal Processing FPGA

After programming the control FPGA, the signal processing FPGA will need to be reprogrammed. The control FPGA will send the start key, followed by a bit stream , followed by the end key to the MSP430 on the com port using the com port interface.The programming bit stream for the signal processing FPGA is over 1 megabyte in size and would not fit entirely in a single buffered pipe. Instead of using many buffered pipes, two control signals, a buffered pipe, and a trigger are used to transmit the programming file and program the signal processing FPGA. Before sending programming data to the control FPGA, the PC will send a reset signal to the control FPGA. This signal will reset all buffered pipes and ensure that they are empty. Before any programming data is sent to the MSP430, the start key is sent through the PC sending a trigger to the control FPGA. Upon reception of the start key trigger, the control FPGA will send the start key on the com port to the MSP430.

After the start key is sent from the control FPGA to the MSP430, the PC splits the desired programming file into chunks and transmits each chunk to the control FPGA using a buffered pipe. Once the chunk of data is transferred to the control FPGA, the PC will issue another trigger to the control FPGA to program the signal processing FPGA. When the buffered pipe is empty, the control FPGA will set an output wire to the PC indicating that the buffered pipe is empty. The PC will poll for the state of the output wire and will block execution until the output wire to the PC indicates that the buffered pipe is empty. When the pipe is empty, The PC will send another chunk to the control FPGA and trigger the transfer. This process will repeat for the duration of the programming file. During each transfer, the control FPGA will send all of the data is the buffered pipe to the MSP430 over the com port until the buffer is empty.

When the PC has finished transferring the programming file to the control FPGA, the PC will issue a trigger to send the end key. When triggered, the control FPGA will send the end key to the MSP430. Upon reception of the end key, the MSP430 will take the signal processing FPGA out or reset and the signal processing FPGA will run. The MSP430 will then ignore all com port signals so that that control FPGA may communicate directly with the signal processing FPGA via the com port.

4.5.3 Initializing the ADC

On a power up or reset, the ADC is in a standby mode and is not sampling the input. To initialize and control the ADC, a 3 wire serial interface is wired between the signal processing FPGA and the ADC. To program the ADC, the PC sends a series of commands to the control FPGA, which in turn sends a series of commands over the com port to the signal processing FPGA. Code on the signal processing FPGA then programs the ADC to sample continuously at 500 MHz by programming registers on the ADC.

4.5.4 Running PC side software

The last step of the boot loader is to start the appropriate user-level application so that the user may control the UWB development platform using the connected PC. Currently, there are four different user level programs to run and the last role of the boot loader allows the user to select and launch which program to use. The currently available programs are as follows.

- Sending test packets - This program allows the user to specify one or more test packets to send. Users may specify to send a randomly generated test packet, a fixed test packet, or a test packet from a file. Users may also select the preamble, number of repetitions of the preamble, and number of packets to send.

- Send file - This program assumes a two-way UWB communication channel. The send file program attempts to mimic a basic guaranteed delivery protocol on top of UWB. For the send file program, a file is broken up into the number of packets required to send the file. Then, each file is sequentially transmitted. The sender then waits for a set duration to receive the same packet back. If the sender does not receive a packet back, or the packet received does not match the packet sent, the sender will resend the packet until the sender receives the correct packet. Statistics are also tabulated for packet transmission.

- Receiving test packets - This program is the receiver-side counter part for the sending test packet program. The receiving test packet program continuously polls the receiver for received packets. When a packet is received, the user is notified and the contents of the packet are saved to disk for future examination.

- Receive file - This program is the complement to the send file program. When run, the receive file program polls the receiver for received packets. When a packet is received, the packet is saved to disk and then retransmitted so that the sender may receive it and send the next packet.

Chapter 5

Results

At present, only one UWB development platform is available for testing and thus only one way communication can be tested. Two way communication requires two complete UWB platforms. To test one way communication, the hardware responsible for transmission was constructed and packets were sent from the newly created transmitter to UWB development platform. The UWB platform was then able to calculate the percentage of dropped packets and erroneous packets by considering the number of packets sent and the packet data.

5.1 Test Design

Two separate computers in the lab were selected to run the preliminary UWB development platform tests. A transmission FPGA and discrete transmitter were connected to one computer and a UWB platform was connected to the other computer as shown in Figure 5-1. The computer connected to the UWB platform used the existing PC software and the computer connected to the transmitter ran a slightly modified version of the PC software in which configuration of the receiver and signal processing FPGA was eliminated since those components did not exist. After the PC software set up the transmitter and the UWB platform, the UWB platform software was placed in a mode where the software would save to disk each packet that it received. The software running on the transmitter was set up to transmit 1000 packets

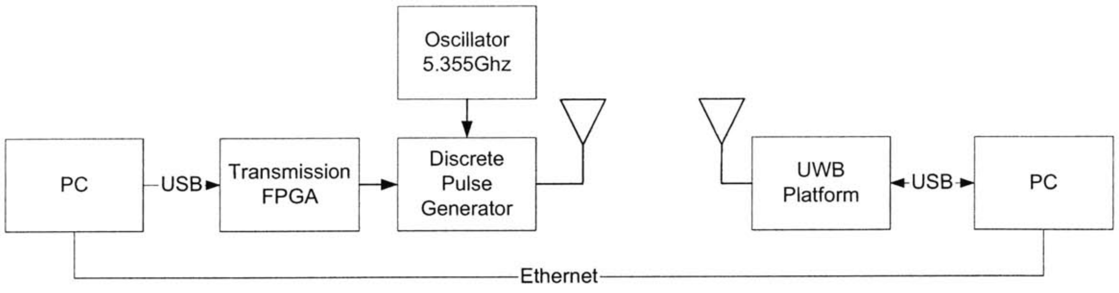


Figure 5-1: UWB Communication Channel Testing Setup

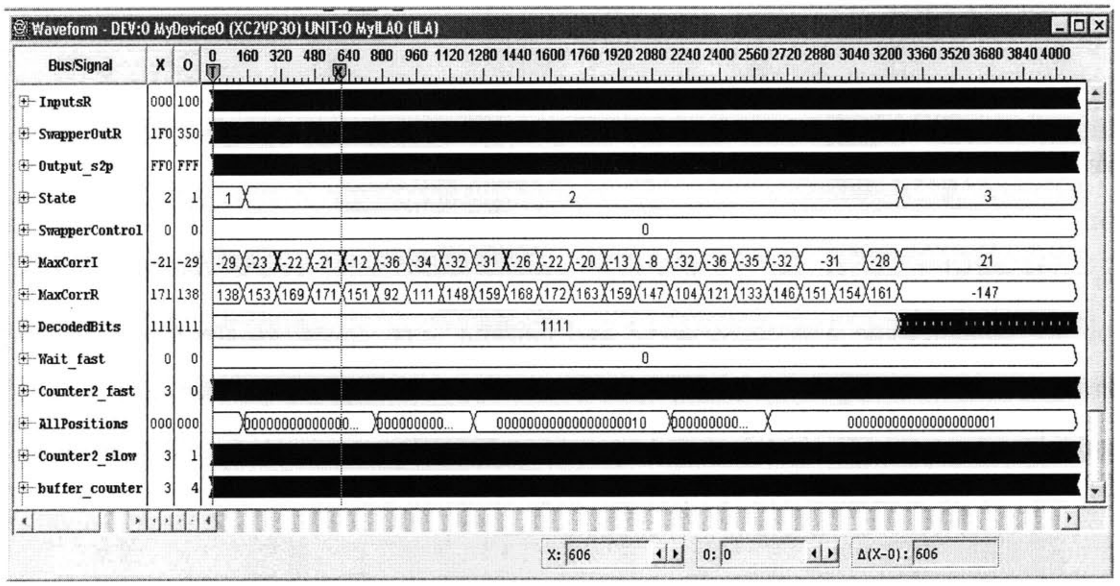


Figure 5-2: Sample Chipscope Output Window

with a random amount of "silence" between packets ranging between 100 and 2000 milliseconds. The minimum wait time was chosen to be so long to ensure that packets were not dropped due to C code inefficiencies. Since the C code must enable packet reception on the receiver for each packet it receives, it is possible that the C could would not enable the acquisition of a packet if packets were sent too quickly.

To calculate the packet loss, the number of received packets was compared against 1000. Of the packets received, it was also important to verify the validity of each packet. A network share was set up on the transmitter so that the contents of each packet sent could be read by the receiver to compare validity. Experiments were

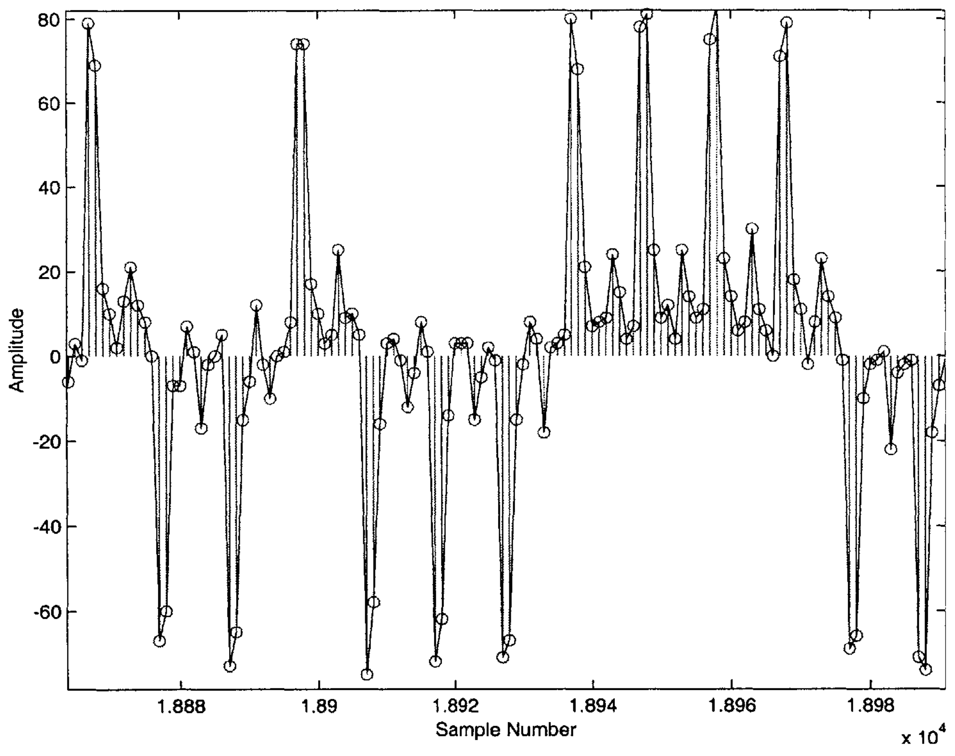


Figure 5-3: Plot of Sampled UWB Waveform

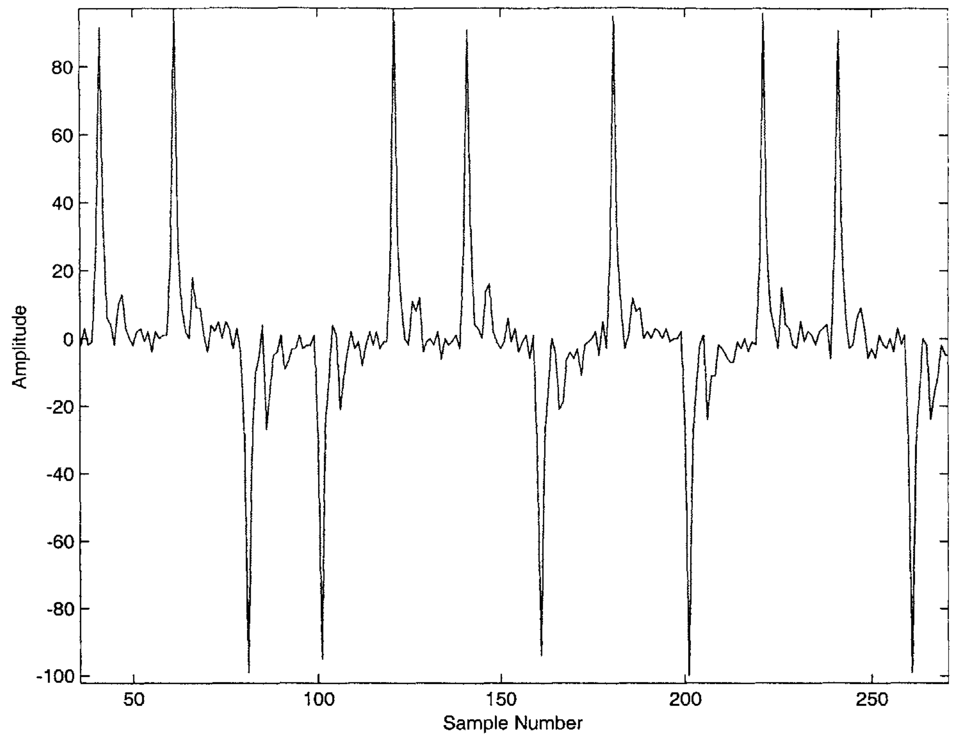


Figure 5-4: Graph of Sampled UWB Waveform

run varying the number of preamble repetitions, and payload size. The tabulated results measured percentage of packets received by the receiver. Also, a percentage of "correct packets" was calculated indicating wheather or not the received packet matched the transmitted data.

5.2 Analysis

When first testing the one-way channel using the described test setup, the packet drop rate and percentage of received bad packets were much higher than anticipated. It was also observed that roughly 20% of packets received contained data that was mostly zero regardless of the input. While there is always the potential for error in packet payload due to background noise, it is highly imporbable that under ideal testing conditions for 20% of packets to have data that was mostly zero.

The first step in debugging the problem using the UWB platform was to analyze

Table 5.1: Transmission Statistics for Revised UWB Signal Processing Code

Payload Len.	**Preamble Repetitions**	**Pkt. Reception**	**Correct Pkt.**
1600	31	73%	97%
400	31	73%	98%
200	31	71%	99%
100	31	71%	99%
1600	31	73%	98%
400	20	60%	99%
200	20	60%	99%
100	20	56%	99%

the raw ADC samples to ensure that the ADC was receiving a good signal. To do this, Chipscope[1] was used to capture a series of samples. The samples were then exported from Chipscope and plotted using MATLAB. Sample MATLAB output of UWB signals can be seen in Figure 5-3 and Figure 5-4. Once the validity of the UWB signals were verified by the MATLAB plots, other internal signals were observed using Chipscope. A sample Chipscope debug window can be seen in Figure 5-2. After careful analysis of control signals using Chipscope, an error was found. The error was corrected and after re-running the tests, there were no more "mostly zero" packets. In addition, both the overall drop rate as well as received bad packet percentage decreased, thus demonstrating the debugging capability of the UWB development platform. Overall statistics for packet transfer can be seen in Table 5.1. Lastly, the probability of UWB packet detection also is lower than anticipated. In the future, Chipscope will likely be used again to narrow down the cause of the packet loss.

[1]http://www.xilinx.com/

Chapter 6

Conclusion

In this thesis, two new tools for debugging and designing UWB communications systems were created. First, a new tool for allowing web based users to obtain data from real UWB communication channels was designed and implemented. This tool will allow web users to develop UWB signal processing algorithms using data from a real UWB communication channel. In addition, a UWB development platform was designed and implemented. The design of the UWB development platform is very modular, allowing for drop in replacements of the core modules responsible for UWB transmission and reception to allow for testing. Furthermore, the interface to the platform is easy to use and compatible with any USB compatible computer. The combination of a modular design and an easy to use interface promises to make the UWB development platform an effective tool for both developing and demonstrating cutting edge UWB communication systems.

After design and assembly of the first UWB development platform was complete, the effectiveness of the platform was immediately tested by characterizing the test setup. Though the initial implementation was able to send information over the UWB channel, the characterization of the system using the platform software indicated a packet drop rate much higher than anticipated. Using the debugging tools associated with the platform, an error was discovered in the underlying implementation of the UWB receiver which accounted for some packet loss. The error was corrected and the packet drop rate decreased as measured by the UWB development platform.

6.1 Future Work

There are three main points for future work.

- Creation of a second UWB development platform - With the construction of a second UWB development platform, a two way channel could be constructed and more compelling demonstrations of UWB capability could be implemented.
- Implement higher level protocols - Higher level protocols such as TCP could be implemented on top of UWB. Implementing TCP would in turn allow for other applications to send and receive data using UWB.
- Construct a new transmitter - The current discrete prototype is limited to a pulse repetition frequency of 50 MHz which in turn corresponds to a maximum potential bandwidth of 50 Mbps. By constructing and using a faster transmitter, an increase in bandwidth would be possible.

Bibliography

[1] Raul Blazquez-Fernandez. Design of synchronization subsystem for an ultra wideband radio. Master's thesis, Massachusetts Institute of Technology, 2003.

[2] Federal Communications Commission. *Ultra-Wideband (UWB) First Report and Order*, February 2002.

[3] Atmel Corporation. *Datasheet for Atmel AT84AD001Dual 8-bit 1Gps ADC*, 2005.

[4] Opal Kellyl Corporation. *XEM3001v2 User's Manual*, 2005.

[5] Sundance Incorporated. *SMT338VP User Manual*, August 2004.

[6] Sundance Incorporated. *SMT398 User Manual*, January 2005.

[7] Texas Instruments. *TIM-40 Module Specification Including TMS320C44 Addendum.*

[8] Opal Kelly. *XEM3001 Application Programmer's Interface (API) Guide.*