ABSTRACT

A Low-Cost Pulsed Fourier Transform Network Analyzer

Brandon J. Herrera, M.S.E.C.E.

Mentor: B. Randall Jean, Ph.D.

Characterizing the frequency response of a material or device over microwave frequencies is a common practice for an RF engineer. New industrial and consumer applications are being developed based on the measurement of the electrical response in the RF and microwave frequency region of various materials ranging from blood glucose to concrete mixtures. Current measurement instrumentation carries a minimum cost in the tens of thousands of dollars which reduces the economical viability of many new possible applications.

This thesis documents the development a low-cost pulsed Fourier transform network analyzer which can be used to measure the electrical properties of materials. New silicon-germanium integrated circuits allow for the pulse system to be implemented for under $300.

A Low-Cost Pulsed Fourier Transform Network Analyzer

by

Brandon J. Herrera, B.S.E.C.E.

Approved by the Department of Electrical and Computer Engineering

_____

Kwang Y. Lee, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree
of
Master of Science in Electrical and Computer Engineering

Approved by the Thesis Committee:

_____

B. Randall Jean, Ph.D., Chairperson

_____

Charles Baylis, Ph.D.

_____

Dennis A. Johnston, Ph.D.

Accepted by the Graduate School
August 2011

_____

J. Larry Lyon, Ph.D., Dean

Page bearing signatures is kept on file in the Graduate School.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

80/20 – Eighty percent to twenty percent

AC – Alternating Current

ADC – Analog to digital converter

ADS – Advanced Design System

ASIC – Application Specific Integrated Circuit

BJT – Bipolar Junction Transistor

BOM – Bill of Materials

CSV – Comma-separated values

DAC – Digital to analog converter

DC – Direct Current

DDS – Direct Digital Synthesis

DSA – Digitally Stepped Attenuator

DSP – Digital Signal Processing

ECL – Emitter-Coupled Logic

FFT – Fast Fourier Transform

GMS – Guided Microwave Spectrometry

I^2C – Inter-Integrated Circuit

IC – Integrated Circuit

IF – Intermediate Frequency

LF – Low Frequency

LNA – Low Noise Amplifier

MUT – Material Under Test

PCB – Printed Circuit Board

Pot – Potentiometer

PTH – Plated Through Hole

RAM – Random Access Memory

RF – Radio Frequency

SATA – Serial ATA

SiGe – Silicon Germanium

SMA – Subminiature version A

SMT – Surface Mount Technology

SNA – Scalar Network Analyzer

SNR – Signal-to-Noise Ratio

SPI – Serial Peripheral Interface Bus

SRD – Step Recovery Diode

TTT – Time Transit Tomography

UI – User Interface

USB – Universal Serial Bus

UWB – Ultra-Wide Band

VGA – Variable Gain Amplifier

VNA – Vector Network Analyzer

ACKNOWLEDGMENTS

I would like to thank Dr. Jean for guiding, advising, and keeping me focused. I would also like to thank my mother for her support and encouragement, and my father for selling glucose meters that don't exist yet. My sincerest gratitude goes to Chelsea for her understanding and patience with my late nights and endless rants. And thanks to the undergraduate research assistants: Joel, Josh M. and Josh D, whom have all since graduated. I also need to thank Eric, Melanie, Matt and Dr. Marks for helping provide the underlying motivations for this work. Thank you to the ECE Department for expensive toys and a fun atmosphere along with Mr. Orr and Mr. Hromadka.

Finally, thank you to anyone else who ever listened, cared, or even pretended to care about this project that diminished so much of my sociability.

For me and you,

and Tara, my salesman

CHAPTER ONE

Introduction

Microwave engineering often requires the characterization of a device or material over a broad frequency range. Many new applications are being developed where the broadband electrical response of a material can be mapped back to specific properties of that material. Examples of these measurements include determining the water to cement ratio in concrete for strength determination or non-invasive blood glucose sensing for diebetics [1]. In laboratory settings these measurements are often accomplished with a vector network analyzer. Vector network analyzers (VNA) are desktop systems which typically cost in the tens of thousands of dollars and provide an extremely high level of accuracy. To make these new applications economically viable, a low cost measurement system to replace the laboratory VNA is needed.

Vector network analyzers operate in the frequency domain by measuring the change in amplitude and phase of a sinusoid that is passed through a material or device under test. The frequency of this sinusoid excitation signal is then swept over the bands of interest permitting the magnitude and phase response of the material under test to be measured. Precise generation of sine waves over a wide bandwidth in the microwave region is a difficult task which leads to the high costs for VNA's. An alternative method to measure the transfer function of a material is to generate an ultra-wide band (UWB) pulse which contains a broad range of frequency information. A sequence of such pulses can then be used to replace the frequency swept sine waves for material characterization in the time domain.

This thesis presents the design of a pulsed Fourier transform network analyzer (PFTNA) show in Fig. 1. The PFTNA uses an UWB pulse train for frequency content generation to offer a measurement solution with costs that are two orders of magnitude less than VNA's. The measurement system is intended for embedded use in low-cost consumer or industrial products such as a non-invasive blood glucose sensor or concrete analyzer. The proposed PFTNA's performance specifications, while not expected to match the accuracy and bandwidth associated with laboratory grade vector network analyzers, are more than adequate to address a wide range of industrial and biological applications. Part of this work was previously published [2].

Fig. 1: Pulsed Fourier Transform Network Analyzer

CHAPTER TWO

Background

Motivation

A Non-Invasive Blood Glucose Sensor

One ongoing research project at Baylor University has been focused on developing a non-invasive blood glucose sensor for diabetics. The sensor has demonstrated good correlation for small test groups [3-4] over a limited range of glucose values, as shown on the Clarke error grid in Fig. 2.

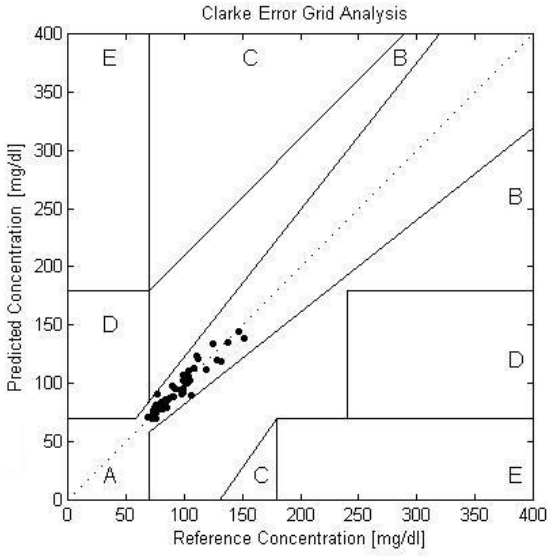

Fig. 2: Clarke error grid of glucose sensor data

The Clarke error grid has five regions of correlation based on the health consequences to the patient if a treatment decision is made from the data. The test data shown here from the non-invasive blood glucose sensor fall in the "A" region which indicates the error

range for a reported blood glucose concentration that is considered valid for making safe treatment decisions.

To advance the research and development of the glucose sensor, a larger sample population is needed, i.e., a large group of people must participate in calibration tests. To obtain access to large groups of diabetics, these tests must conveniently be conducted at an offsite facility. The measurement setup for an early prototype of the sensor is shown in Fig. 3 and includes the sensor, a spiral microstrip transducer as seen on the right in the photograph, and a vector network analyzer, shown on the left, connected together with SMA cables.

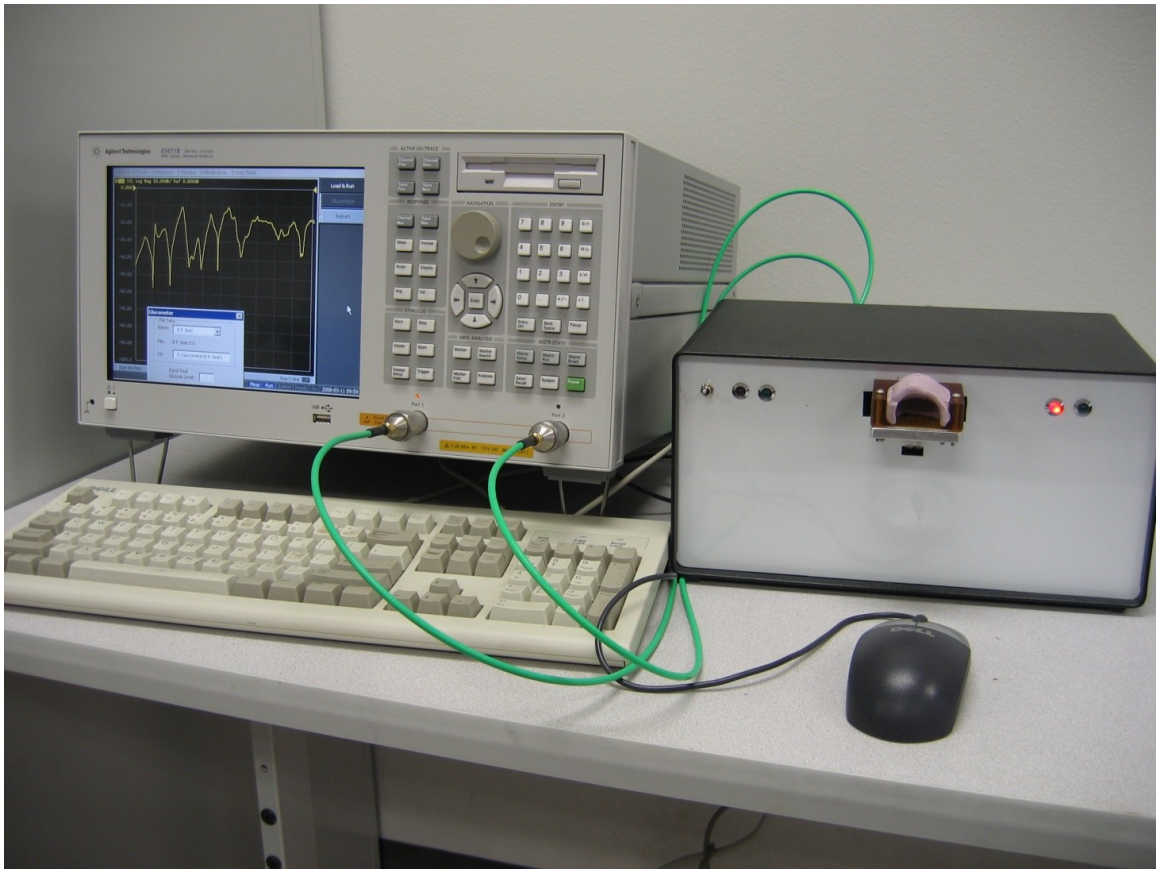

Fig. 3. Non-invasive blood glucose sensor test measurement setup

However, the VNA used in this measurement setup is complex and expensive and cannot be operated efficiently by untrained personnel, nor is it possible to dedicate the instrument offsite for long periods of time to the data collection task. A new measurement platform is needed to replace the VNA. Ideally, this new measurement platform can be implemented for a much lower cost such that it could then be easily replicated and distributed to various hospitals and clinics for testing the sensor on large sample populations.

The non-invasive blood glucose sensor operates by producing shifts in the frequency-domain response signal as the permittivity of a patient's thumb changes. For correlation of the sensor's frequency response to an actual glucose value, the position in frequency of features such as peaks and nulls are more important than amplitude of those features. Phase is currently not used in the correlation, but may be used in future applications. One performance objective of the PFTNA developed in this thesis, motivated by the requirements of this particular application, is to accurately locate the position of peaks and nulls in frequency.

A Concrete Sensor

Also being studied is a sensor to measure the water to cementitious ratio (W/C) of fresh concrete. This application could be deployed using a specialized frequency dependent structure inside cement trucks to measure the water percentage of the cement inside of the truck. Fig. 4 shows the strength of concrete based on the W/C ratio with different mixtures of aggregate.

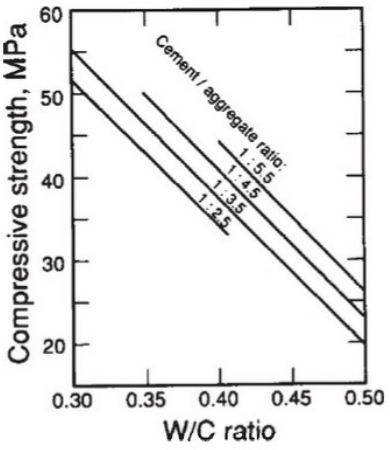

Fig. 4: Effect of aggregate concentration on strength of concrete [5]

Currently the sensor's measurements are also taken with a vector network analyzer which is cost-prohibitive for this imbedded application. To produce accurate results the PFTNA needs to be able to measure highly lossy materials with at least a 2 GHz bandwidth. Fig. 5 shows sample data taken with a VNA of concrete with varying W/C ratios.

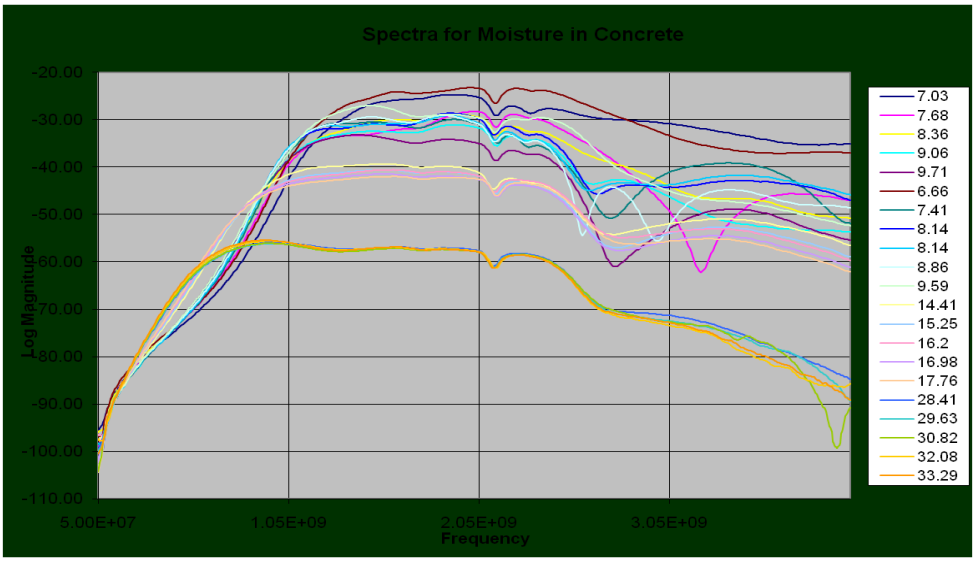

Fig. 5: Sample concrete data

The sensor configuration exhibits about 20dB minimum insertion loss for concrete, with an S_{21} variation of around 60 dB over the frequency measurement range. This typical application provides motivation for specifying the minimum transmitter power and measurement dynamic range for the Pulsed Fourier Transform Network Analyzer. When W/C measurement capability is fully developed, it will represent a huge economic impact. Currently many concrete designs use more cement than necessary to guarantee strength requirements as a safeguard to assure desired performance. As an estimate, 15% - 18% of the industry's costs are spent reworking concrete. The sensor would also have a large environmental impact as cement production accounts for 5% of annual $C0_2$ production in the world [6].

Time Transit Tomography

Another application for the PFTNA involves a new modality of imaging that was developed using UWB pulses dubbed Time Transit Tomography (TTT). This method creates tomographic projections using the time delay of the ultra-wide band pulses as they pass through a material of varying permittivity. The modality is able to use classic tomographic reconstruction techniques to obtain images with minimal hardware requirements. The current test bed is shown in Fig. 6 with the rotating center plate for the material under test (MUT) and the traversing biconical antennas on the edges. Materials with larger permittivity will increase the delay between the two antennas.

The new imaging technique was initially conceived as an alternative modality for medical imaging applications. For example, it could offer a safer imaging modality for such applications as breast cancer detection. The imaging technique requires a system

that can measure transit time delay with at least 10 ps of accuracy. Fig. 7 shows a TTT reconstructed image of two jars of oil with a few centimeters of separation.

The tomographic reconstruction shown in Fig. 7 was obtained using a vector network analyzer and the prototype apparatus shown in Fig. 6. The VNA can provide a transit time delay measurement by performing an inverse FFT operation on the swept-frequency measurement of the complex forward transmission S-parameter, S_{21}. This measurement mode can provide a minimum resolution in the time delay measurement of 12 ps on an 8 GHz VNA. The study of TTT is another ongoing research project at Baylor University and provides another example of an application that would benefit from a low cost time domain network analyzer.

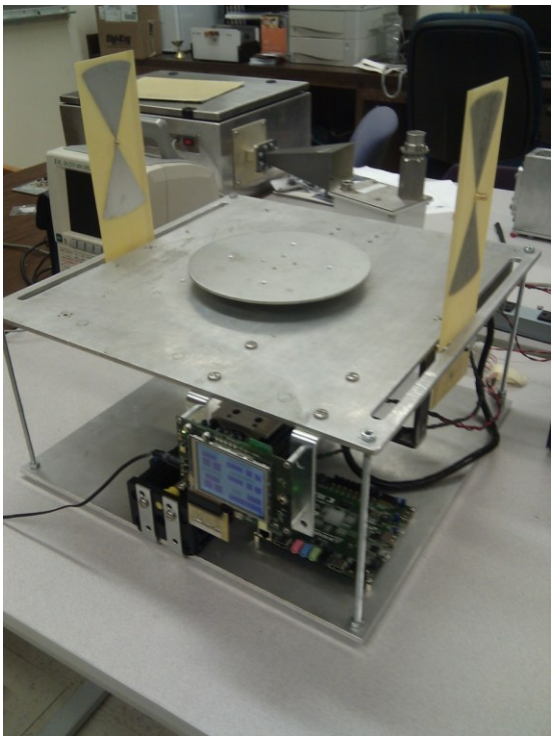

Fig. 6: Time Transit Tomography Table

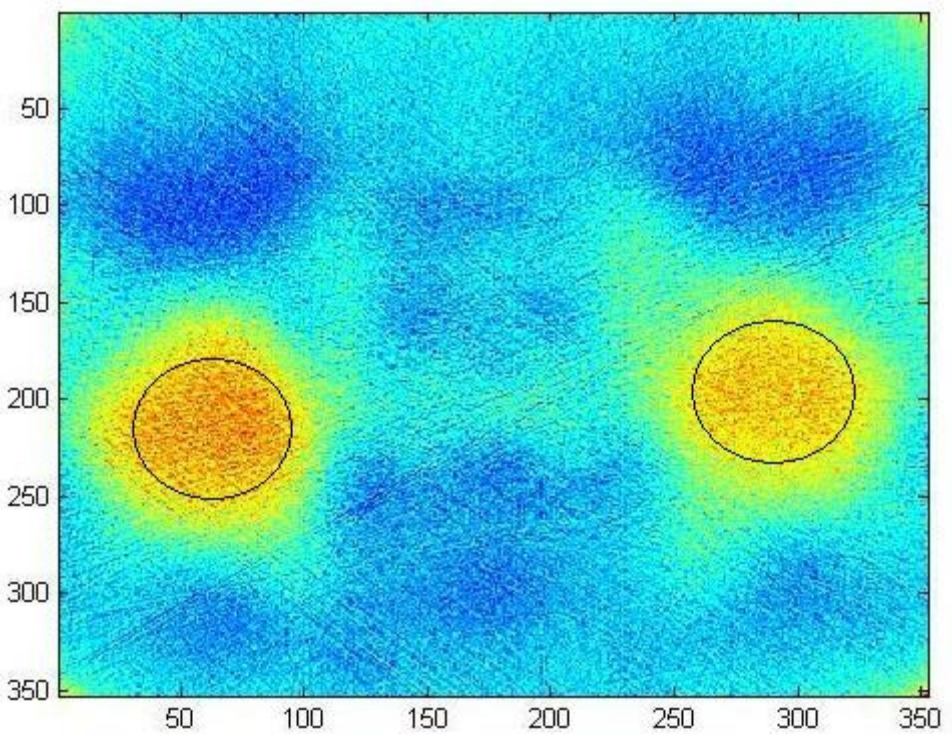

Fig. 7: Reconstructed image of jars of oil

A Commercial Example: The MarginProbe

A current consumer application using UWB microwave measurements is the MarginProbe™ from Dune Medical Devices shown in Fig. 8. The MagrinProbe is currently for sale in Europe for detection of tumorous tissues during operations. It features an open ended coaxial probe used for coupling electromagnetic fields into tissue. The system then uses a vector network analyzer from Agilent Technologies to make the measurement and a separate computer to analyze the data. The network analyzer is sitting vertically in the bottom of the system's console. If an embedded alternative for the network analyzer existed the cost of the device could be lowered and the size also reduced.

Fig. 8: The MarginProbe system from Dune Medical Devices

When the costs of making UWB systems decrease a large number of possible applications will become economically feasible. The methodology of UWB pulse generation is described in the subsequent section.

Pulse Spectroscopy

An alternative to swept-frequency excitation of a material is to generate all of the desired frequencies simultaneously. Theoretically an impulse or Dirac delta function has infinite bandwidth and would therefore produce all of the frequencies desired in the response. However, realized pulses have finite bandwidth and finite energy content with measureable, i.e. non instantaneous, rise and fall times, and an extended duration. To adequately determine the response of a system out to multiple GHz, these pulses must have rise times on the order of tens of picoseconds. The difficulty of this task can be conveyed by the fact that light travels only 2.99 mm every 10 picoseconds.

After generation, the ultra-wide band pulses are then transmitted to the material under test (MUT). In the case of the non-invasive blood glucose sensor, the MUT is a subject's thumb pressed upon a two-port spiral microstrip printed circuit board (PCB) displayed in Fig. 9.

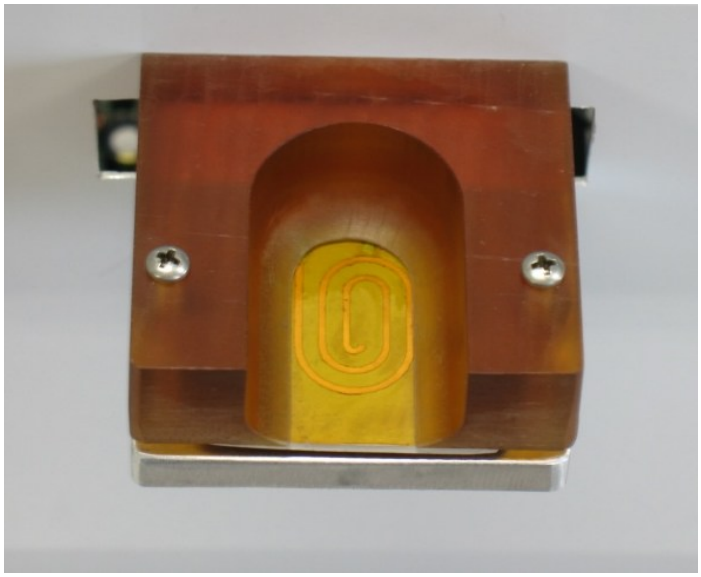

Fig. 9. Spiral resonant sensor used for non-invasive blood glucose sensing

The MUT could also be concrete in a Guided Microwave Spectrometry (GMS) waveguide filled with concrete or other materials in the case of the concrete sensor. As the pulses travel through the MUT they become dispersed. Dispersion is the distortion in the characteristics of a signal that is produced when different frequencies contained in the signal travel at different velocities. Dispersion occurs when the permittivity of the material under test is not constant, but varies as a function of frequency. As an example, Fig. 10 graphs the complex permittivity of water which is a highly dispersive material at microwave frequencies. Note that permittivity is also referred to as the dielectric constant, but it can only be assumed as a constant in more simplistic applications.

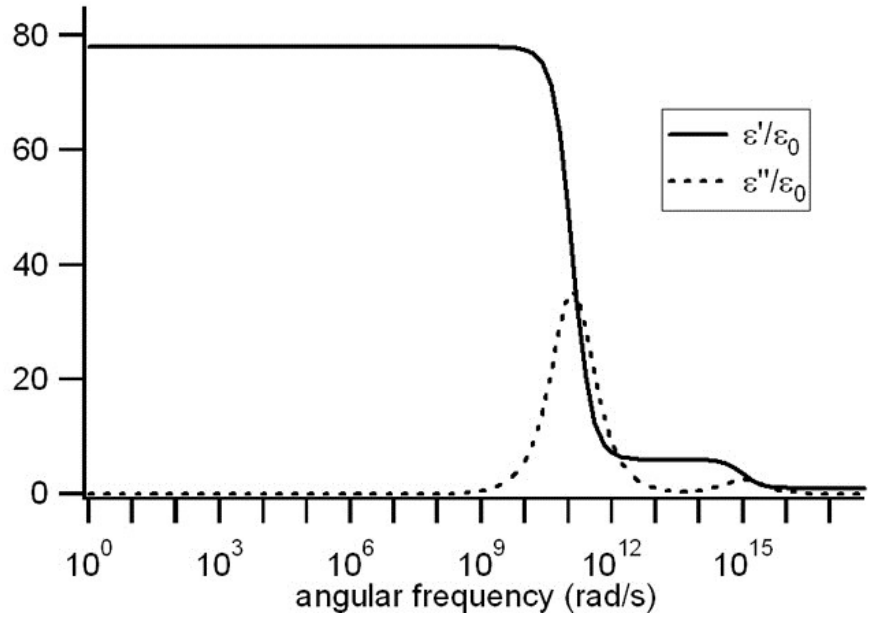

Fig. 10. Complex permittivity of water over frequency

To use the information contained in the dispersed pulses, it is convenient to convert them from the time domain to the frequency domain. To accomplish this task digitally, the dispersed pulses are first sampled and digitized by an analog-to-digital converter. A fast Fourier transform (FFT) is used to convert the time domain samples to a frequency domain representation. The Fourier transforms are often represented in log magnitude format. The frequency response of the MUT can be characterized by the difference between the frequency spectrum of the pulse transmitted pulse and the frequency spectrum of the pulse leaving the MUT. This response can be referred to as the transfer function, impulse response, or S_{21} by various fields of electrical engineering.

The bandwidth of the electrical response produced by pulse is limited by the frequency spectrum of the transmitted pulse. The bandwidth is limited to frequencies where the spectrum of the transmitted pulse is significantly greater than the noise floor. The magnitude of the pulse's spectrum decreases with frequency, which also reduces the

system's dynamic range as frequency increases. A good metric for the bandwidth of the PFTNA for a number of important applications is the 10 dB bandwidth, which is defined as the frequency where the magnitude of the pulse's spectrum falls to 10 dB lower than the maximum magnitude.

By way of illustration, consider a rectangle function as a simple model for pulses of the type being used in the present system. Taking the Fourier transform,

$$\mathcal{F}(\omega) = \int_{-\infty}^{\infty} f(t) e^{-2\pi j\omega} \tag{1}$$

of a rectangular pulse,

$$f(t) = \text{rect}(\frac{t}{t_0}) \tag{2}$$

results in a sinc function.

$$\mathcal{F}(\omega) = \frac{1}{\sqrt{2\pi t_0^2}} \text{sinc}\left(\frac{\omega * t_0}{2\pi}\right) \tag{3}$$

Equation 3 quantifies the frequency information contained in the pulse and would be the ideal result from a discretized Fourier transform such as the FFT. The sinc function has nulls where no frequency content is transmitted when the argument of the sinc function equals an integer multiple of π. Thus the frequency response of the MUT cannot be determined near those frequencies. This property is a limitation with pulse spectroscopy, but can be mitigated by designing the nulls to occur at frequencies higher than the frequencies of interest. The position of the first null can be increased in frequency by minimizing t_0 in the argument which decreases the width of the rectangle function.

For a more accurate analysis of the frequency response of a real pulse, a slightly more complicated model is used shown in Fig. 11.

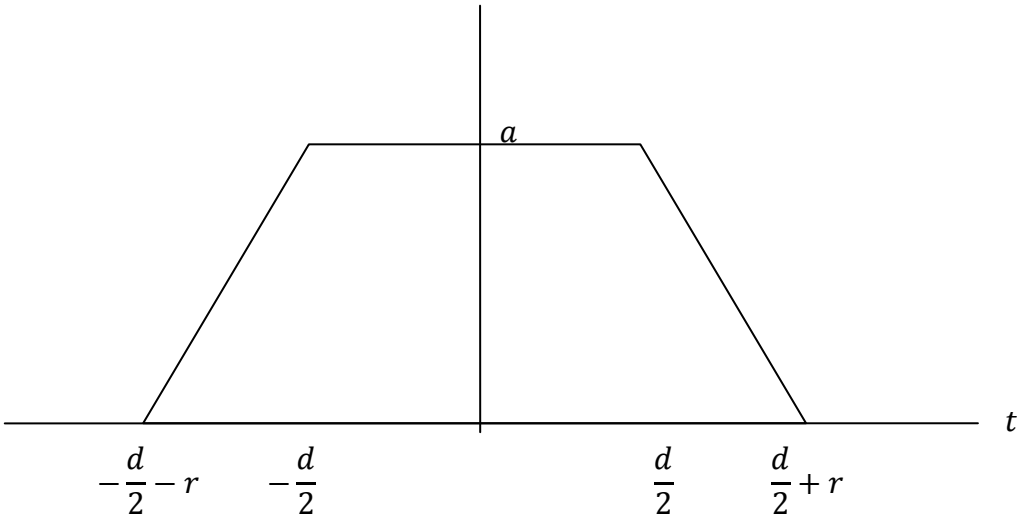

Fig. 11. Model of the Pulses

The parameter d represents the hold time of the pulse, r is the transition times, and a is the amplitude as defined in the piecewise function (4).

$$f(t)=\begin{cases}\frac{a}{r}*t+a*\left(1+\frac{d}{2r}\right), & for-\frac{d}{2}-r\le t\le-\frac{d}{2}\\ a, & for-\frac{d}{2}\le t\le\frac{d}{2}\\ -\frac{a}{r}*t+a*\left(1+\frac{d}{2r}\right), & for\frac{d}{2}\le t\le\frac{d}{2}+r\end{cases} \tag{4}$$

The Fourier Transform of the model is shown in equation (5).

$$\mathcal{F}(\omega)=-a*(d+r)*\text{sinc}\left(\frac{(d+r)\omega}{2}\right)*\text{sinc}\left(\frac{r\omega}{2}\right) \tag{5}$$

The derivation of (5) is included in Appendix B. As the argument of the sinc function gets smaller, the first null position increases in frequency. From (5), the first sinc function with frequency scaling factor $(d+r)/2$ will always have a first null lower in frequency than the second sinc function with $d/2$ as the scaling factor. Since the sinc

function with $(d + r)/2$ for the scaling term is the limiting factor, pulse transition times and duration have approximately equal importance in achieving increased pulse bandwidth.

The pulse model in Fig. 11 uses linearly sloped sides to model the rise and fall of the pulse and was produced by piecewise functions in MATLAB. For the current applications of PFTNA, a bandwidth of at least 2 GHz is required. The pulse in Fig. 12 demonstrates transition times of 100 ps and a hold time of 200 ps which results in a first lobe bandwidth of 3.2 GHz. This performance would have excellent utility for a number of applications provided that hardware can be designed and fabricated with 100 ps transition times and a 200 ps hold time.

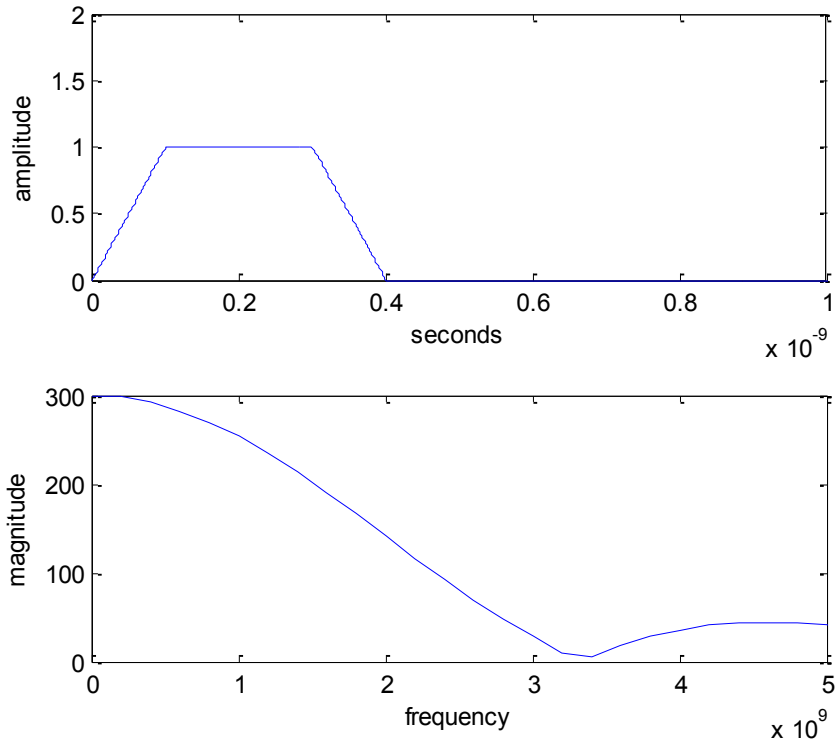

Fig. 12. Frequency content of a test pulse with 100 ps transition times and a 200 ps duration

Further analysis was done to determine the relationship and importance between the pulse transition times and hold time. Fig. 13 compares two pulses with the same total pulse length while varying the transition and hold times. The pulse in red has 100 ps transition times and a 400 ps hold time. The pulse in blue has 200 ps transition times and a 200 ps hold time. The pulse in blue, which has 100 ps slower transition times but a 200ps shorter hold time, has bandwidth increased by 500 MHz. This result means that decreasing the hold time by 200 ps is more important than decreasing the transition times by 100 ps, unless more amplitude is desired.

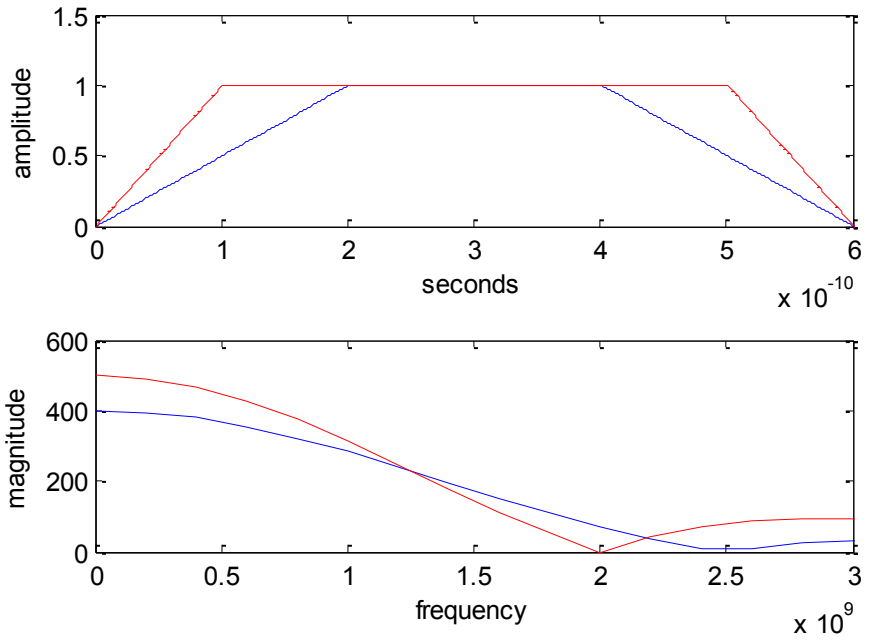

Fig. 13. Frequency content of a test pulse with 100 ps transition times and a 400 ps duration and a pulse with 200 ps transition times and a 200 ps duration

Fig. 14 shows a comparison of two additional pulses. The pulse in blue is the same from Fig. 13 with 200 ps rise times with a 200 ps hold time. The pulse in red has 100 ps transition times and a 300 ps hold time. These pulses were chosen to compare the tradeoff between 100 ps in transition times with 100 ps in hold time. The resulting FFT

shows that the pulses offer similar frequency performance which verifies the conclusion from (5). Simply stated, pulse transition times are of equal importance to the pulse hold time. This analysis was done to discover which variables were more important for the design of the pulse generator.

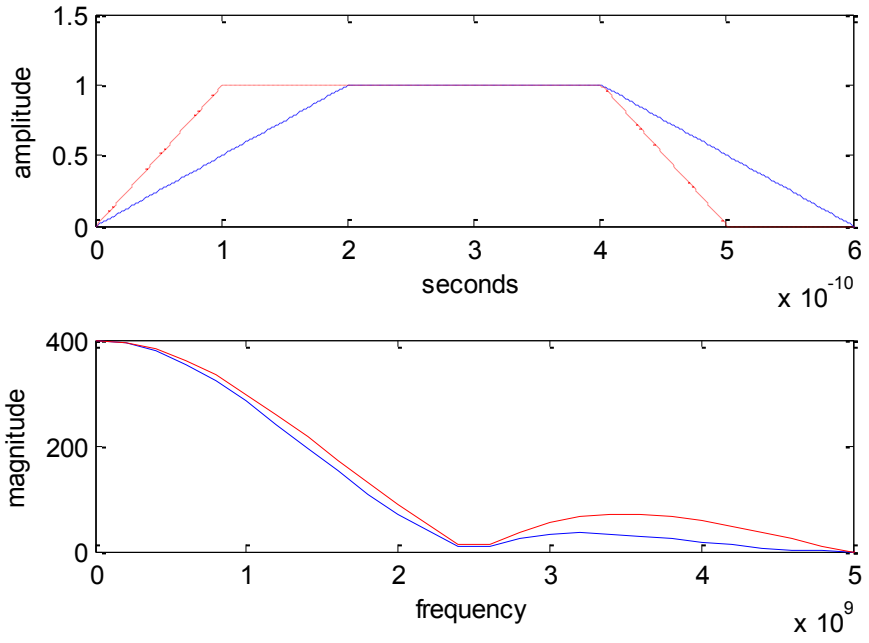

Fig. 14. Frequency content of a test pulse with 100 ps transition times and a 400 ps duration and a pulse with 200 ps transition times and 300 ps duration

Extended Time Sampling

Generating an UWB pulse is only half of the requirement of a pulse transceiver. For example, to receive an UWB pulse with 2.5 GHz of bandwidth, a 5 Gs/s analog to digital converter (ADC) would be required to satisfy the Nyquist criterion. Currently, the fastest ADC available on DigiKey is only 3 Gs/s and costs $815. A design involving an RF ADC would not be a cost effective or adequate solution so an alternative method is needed. Equivalent time sampling (ETS) is a technique that may be applied to repetitive signals to achieve an equivalent sampling rate that is significantly greater than the actual

sampling rate of an ADC. The ETS technique triggers the receiver at a slightly slower frequency than the generator. As the receiver triggering clock slowly drifts out of phase from the generator triggering clock, the point sampled from the RF input signal shifts with the phase difference. Each generated pulse provides a single sample point in the extended time domain. Fig. 15 shows an example of ETS.

The top panel of Fig. 15 shows the generated pulse sequence with a 1 kHz pulse repetition frequency, and the middle section shows the receiver trigger signal at 0.9 kHz. These frequencies are much slower than the actual frequencies and are only used for illustration purposes. The 100 Hz offset between the generation and the receiver causes the receiver to extend the pulse in time by a factor of 10. This factor is called the extended time factor (ETF) and is governed by:

$$\frac{f_0}{\Delta f} = \frac{1\ k\text{Hz}}{100\ \text{Hz}} = 10 \tag{6}$$

Thus the generated pulse that was 1ms wide is now 10 ms. This effect is shown by the bottom section of Fig. 15. The blue dots are the points where the receiver is triggered and the dotted blue line is an equivalent representation of the original pulse that is now 10 times as wide. Fig. 16 shows an overlay of the ETS pulse with the generated pulse.

The receiver features a sub-sampling circuit which provides a sample and hold function. The extended time factor determines how many individual samples are contained in the reconstructed pulse. The horizontal steps are then filtered with an operational amplifier which cleans the signal. The actual frequencies used are described later in the Sub-Sampling Circuit section of Chapter 3.

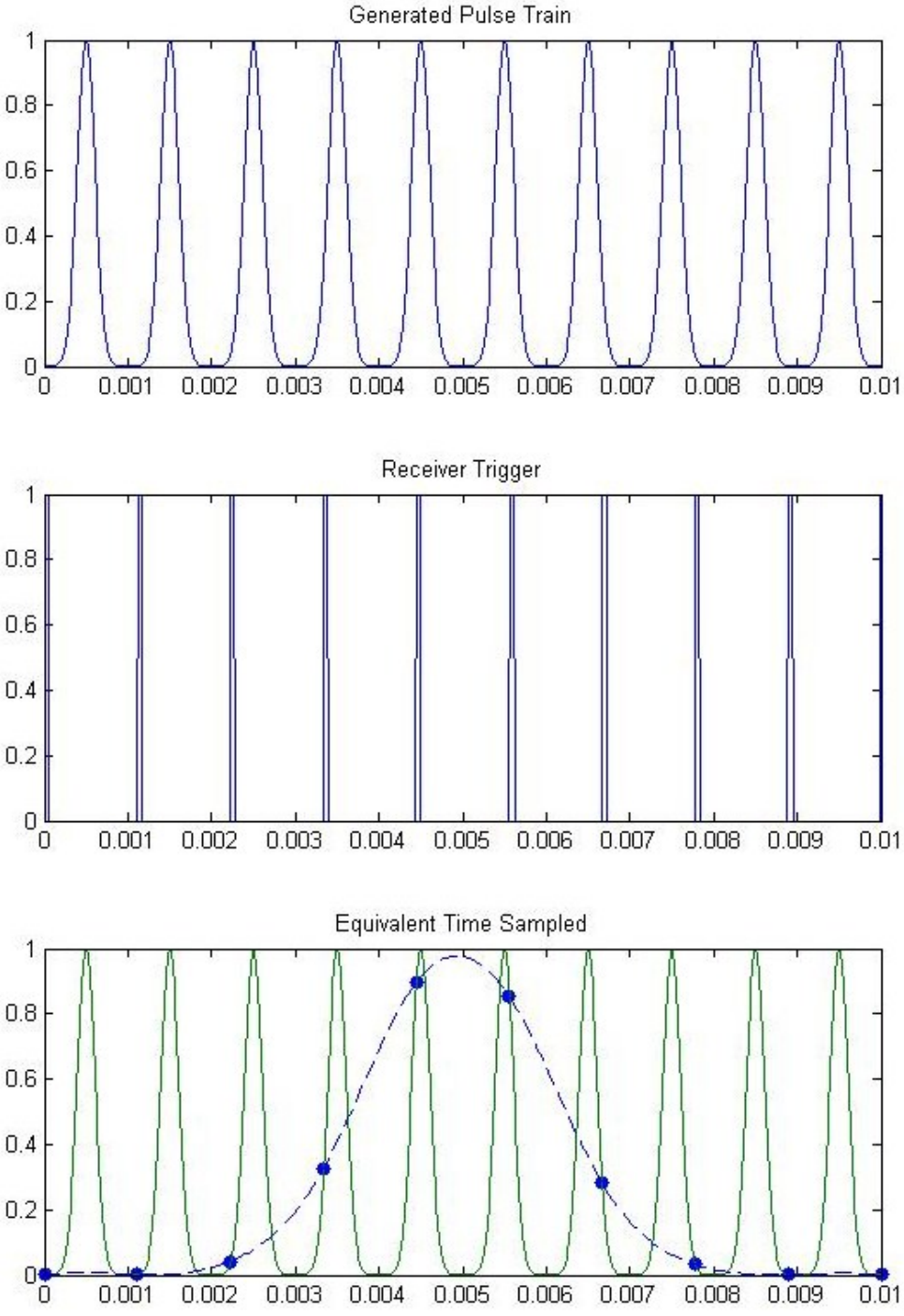

Fig. 15: Equivalent Time Sampling

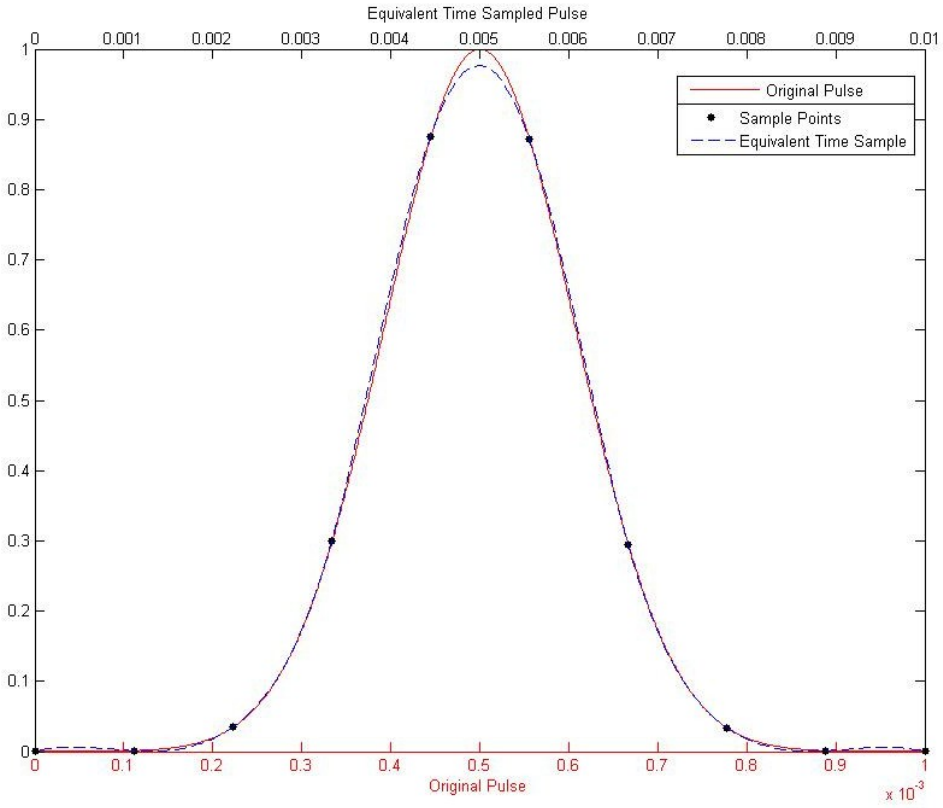

Fig. 16: ETS Overlay

CHAPTER THREE

The Pulse Transceiver

The pulse transceiver produces both the UWB pulses and samples the delayed and dispersed received pulses. A separate sub-system which provides the control signals, data collection and processing, and user interface is described in Chapter 4. The current pulse transceiver design consists of the three subsystems shown in Fig. 17.

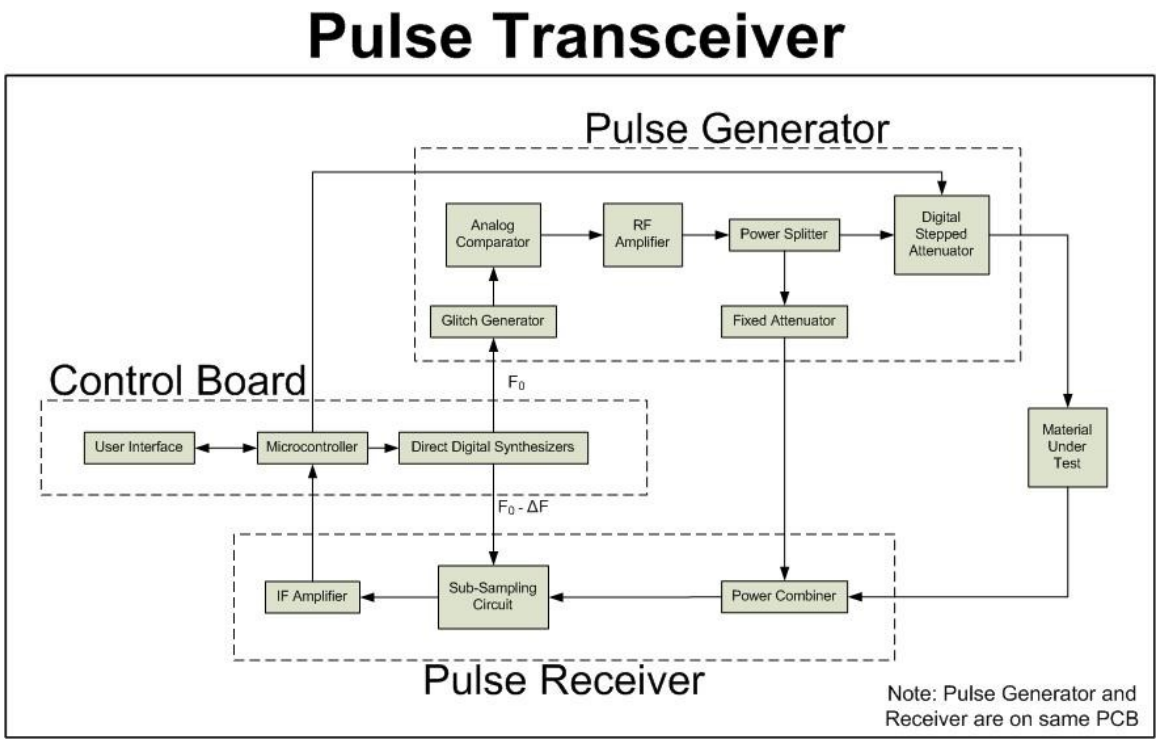

Fig. 17: Block diagram of the Pulse Transceiver

The generator produces the UWB pulse sequence with important specifications of pulse rise time, width and amplitude. Pulse rise times of less than 100 picoseconds are desired for large bandwidth with pulse amplitudes of at least 1 volt. The receiver expands the dispersed pulse in time using extended time sampling techniques to allow

sampling by a low frequency ADC. Specifications desired of the receiver focus on dynamic range, signal-to-noise ratio and signal reconstruction accuracy.

The organization of this chapter follows the design process. Initially individual sub-systems were designed and tested separately. Labcenter Electronics' the Proteus PCB Design Software version 7.7 was used for schematic capture and printed circuit board (PCB) layout design.

Pulse Generator

Design

The design of the pulse generator uses a high speed SiGe analog comparator, an ADCMP580, which has a specified application for pulse spectroscopy. The ADCMP580 from Analog devices has a differential output with specifications of 400 mV single ended or 800 mV differential amplitude and 80/20 rise times of 37 ps. Only one output of the comparator is used to drive the pulse generator. The inverted output is terminated in 50 Ohms.

To achieve small pulse widths, a high speed emitter-coupled logic (ECL) flip-flop is used to trigger the comparator as demonstrated in Fig. 18. The input to the D-flip-flop is tied high and the Q-output is routed to the active high reset. When the clock is asserted, the flip-flop Q-output will drive high and immediately reset itself. This configuration results in generating a pulse or "glitch" with widths based on the propagation delay of the flip-flop. The comparator is then used to sharpen the rise times of the glitch.

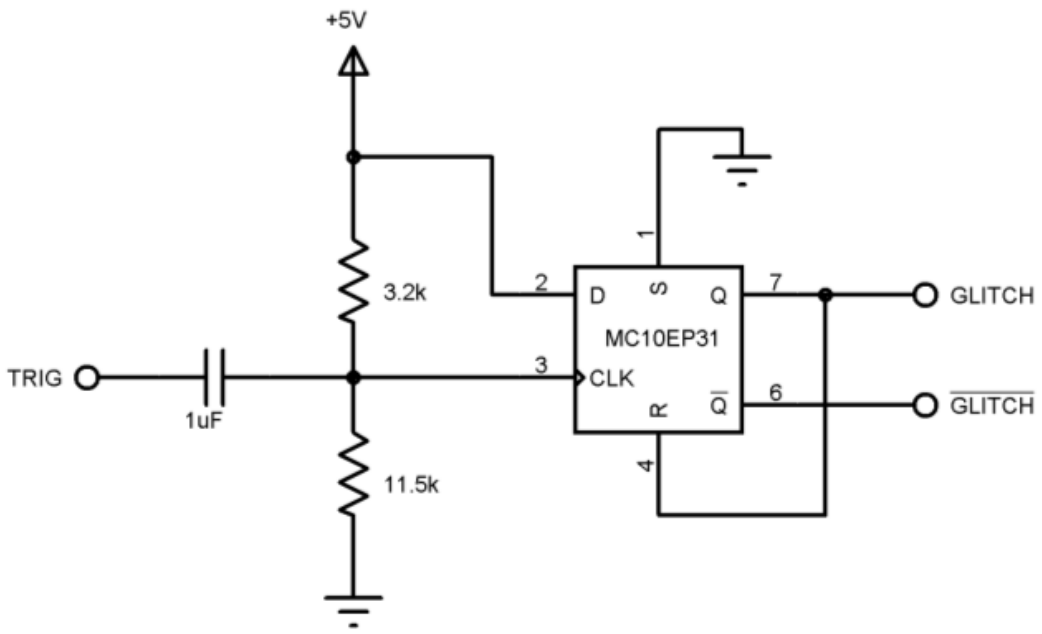

Fig. 18. Schematic of the glitch generator

An InGaP broadband high frequency amplifier from Skyworks is used to increase the amplitude of the output of the comparator. This amplifier provides 20 dB of gain with a gain flatness of ±1.5dB from 0 to 6 GHz. The SKY65017-70LF which dissipates 500mW of power was designed to be cascadable in wireless base stations, thus it has good stability. A few amplifiers from RFMD and Triquent that were evaluated for this application did not have unconditional stability and were found to oscillate in this circuit design. The Skyworks amplifier was chosen for its high flat gain and stability. If it is desired to design a handheld, battery powered system, there are lower power alternative solutions. With a 400mV pulse single ended output from the comparator, the amplified pulse with 20dB of gain should have an output power of 25dBm. However, the SKY65017-70LF's 1dB compression point is 20dBm. This operating condition means that the amplifier is being driven into compression. Compression of sinusoidal signals causes the peaks of the signals to be flattened which add higher frequency harmonic

content. Compressing a pulse will cause the pulse shape to be squarer, which is actually a more desirable shape.

RF amplifiers usually receive the DC bias voltage through the same pin as the RF output. This configuration requires a circuit to be designed to prevent any of the RF output power from reaching the DC power supply. Usually for an amplifier's bias network, a single inductor is used. However the parasitics of a single inductor will allow some high frequency signals to undesirably pass into the power network at the inductor's self-resonant frequency. For UWB, the bias network was designed with three inductors for RF chokes, each with different and non-overlapping self-resonant frequencies. The theory is that when the parasitics of one inductor allow RF to pass, the other inductors will still have a large impedance.

The initial pulse generator prototype was designed with a power splitter on the output. The splitter's coupled port allows a low amplitude pulse to go directly to the receiver whereas the splitter's through-port is applied to the material under test. The input of the receiver could then have a power combiner to recombine the coupled pulse from the generator and the dispersed pulse from the material under test. This architecture produces an attenuated copy of the transmitted pulse along with the dispersed pulse from the MUT at the input to the receiver. The SMA cables used to connect the MUT to the pulse transceiver add delay which ensures that the dispersed pulse will be received after the coupled transmit pulse. The attenuated replica of the transmitted pulse serves as a calibration pulse for every measurement.

The broadband power splitters and combiners necessary for the calibration path can only be implemented with resistors without adding significant costs or size to the

system. These resistive splitters add considerable loss to the pulses. To obtain a configuration with minimal losses, Table 1 was compiled to compare various power splitter designs for the generator and receiver. Three power splitters were chosen for analysis: the common 6dB coupler, the Owen resistive splitter [7], and the Adam resistive splitter [8].

Table 1: Resistive Power Splitters Performance Comparison

Generator Splitter:	Owen (-1/-31)	Owen(-3/-20.5)	Adam(-2/-18.7)	Adam(-3/-15)
Receiver Splitter:	6dB Coupler (-6/-6)	Owen(-3/-20.5)	Adam(-2/-18.7)	Adam(-3/-15)
Transmit Path Losses	7 dB	6 dB	4 dB	6 dB
Calibration Path Losses	36.5 dB	41 dB	37.4 dB	30 dB

The parentheses indicate the splitter's attenuation in the through- and coupled-ports. The total attenuation of the calibration path is desired to be nominally -35 dB, with minimum attenuation present in the transmit path. The attenuation value of -35 dB was chosen for the coupled port so as to match the expected losses of a typical MUT so that the amplitude of the calibration pulse would be close to the amplitude of the dispersed pulse. The second and forth columns in Table 1 represent two choices for splitter designs that provide close to the 35dB of attenuation in the calibration path. Of these two designs, the two Adams splitters provide an extra 3dB of power in the transmit path over using an Owen with a 3dB coupler. This analysis of possible combinations of power splitters allowed for a design with a 3dB increase in the transmit pulse's power.

The two Adams splitters from column 4 were used to create a through-port signal with an amplitude loss of 4 dB and a coupled-port signal that is attenuated by 38 dB. The

broadband splitter is matched to 50 Ω and has an ideal or theoretical efficiency of 80%. The calibration section in this chapter further discusses the power splitters.

Fabrication

To obtain frequency performance in the multiple GHz range, careful high frequency design practices are used in the layout designs. These techniques include the use of ample decoupling and bypass capacitors to reduce power supply noise and transient load fluctuations. A top ground fill is used to create a secondary top ground plane along with a bottom ground plane. The ground plane on the top is important for the integrated circuits whereas the ground plane on the bottom is important for the microstrip traces. These planes are tied together using multiple through-hole connections or vias. The vias connecting the two ground planes help to provide each integrated circuit with a low impedance path to ground. The lateral spacing between the signal layer ground and the high frequency transmission lines is increased as much as possible to reduce parasitic capacitance. Rows of plated through-holes (PTH) are also used to isolate the high frequency microstrip transmission lines. These techniques are demonstrated in the layout in Fig. 19.

Several power zones were used for each power supply, provided there was enough physical space on the PCB. The 4-layer control board described in Chapter 4 uses seven power planes, which are separated for the analog and digital supplies. For two layer designs, adding bottom power planes remove some of the ground plane. Care was taken to minimize ground loops when adding power zones. The power planes do not extend to the edge of the PCB to reduce coupling.

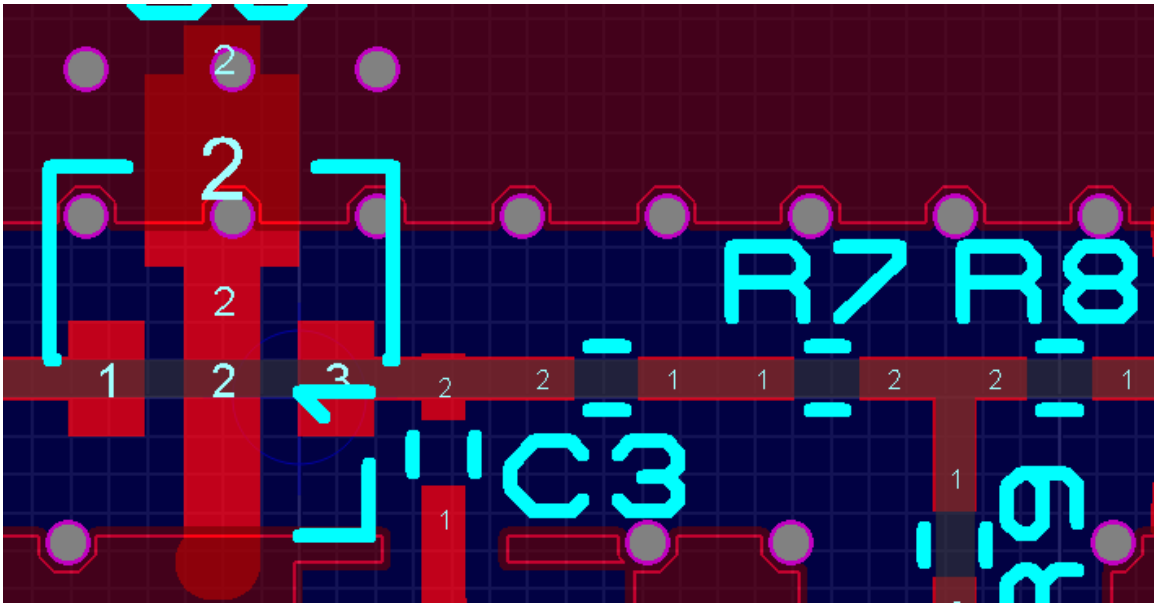

Fig. 19: A section of the pulse generator layout

Rogers RT/Duroid 6010 was chosen as the substrate for its high permittivity, low cost, and wide use in RF commercial applications. A two layer design was chosen to reduce fabrication difficulties and costs. The RT/Duroid 6010 has a relative permittivity of 10.2, and 25 mil substrate thickness was chosen to result in 50 Ohm line widths of 23 mils. This trace width is a good compromise between keeping the trace size small and providing ease of fabrication, as smaller line widths are more susceptible to impedance variations from manufacturing. The control printed circuit boards were fabricated using FR4.

The boards were fabricated using an LPKF Protomat S62 rapid prototyping machine with a vacume table and fiducial recognition system shown on the left in Fig. 20. The through-holes were plated using a LPKF MiniContact RS plating system shown on the right in Fig. 20.

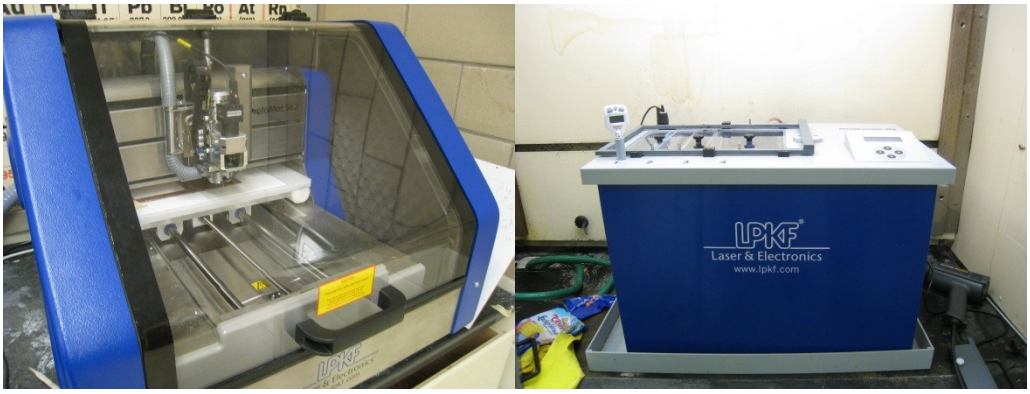

Fig. 20: In-house PCB Prototyping Systems

After plating, the board was tinned using Liquid Tin by MG Chemicals. A solder paste stencil was milled out of Kapton Film with the Protomat S62. The stencil was aligned to the PCB by hand and solder paste was then applied to the stencil and PCB. The components were placed by hand and soldered using a hotplate.

A divide-and-conquer method was used in the circuit development process, where portions of the pulse transceiver were prototyped individually. This method allowed for easier testing and diagnosing of problems. Several prototype boards needed to be fabricated as Baylor's surface mount technology (SMT) manufacturing process was refined. The most common defects experienced were short circuits underneath the leadless IC's. Short circuits were minimized with improved solder paste placement and outsourced laser cut stencils. Several non-working PCBs are shown in Fig. 21 to illustrate the difficulties in fabrication due to a lack of SMT assembly machines. The through-hole plating process also added to the PCB's total copper thickness which increased the trace width variance. This problem was alleviated in later prototypes by using more expensive rectangular endmill bits.

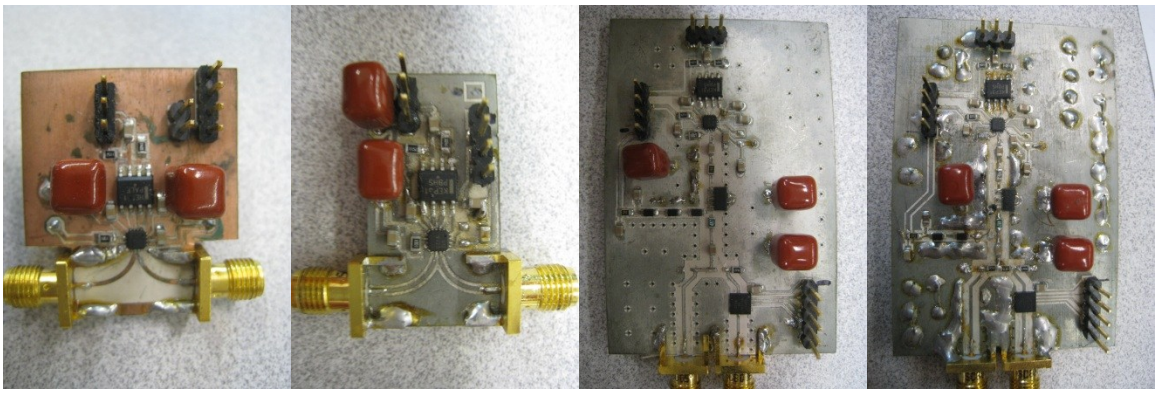

Fig. 21: Printed circuit boards with manufacturing errors

Initial prototyping was done without the availability of the through-hole plating system, a situation which added difficulty and significant time to prototype construction. The initial lack of plating capability did not provide the option of vias underneath the integrated circuits. Vias directly underneath IC's are very effective for dissipating heat and reducing ground impedance as the ground plane extends underneath most of the IC's.

The pulse generator in Fig. 22 achieved rise times of 90 ps, with pulse peak amplitudes of 1.6 V as shown in Fig. 23.

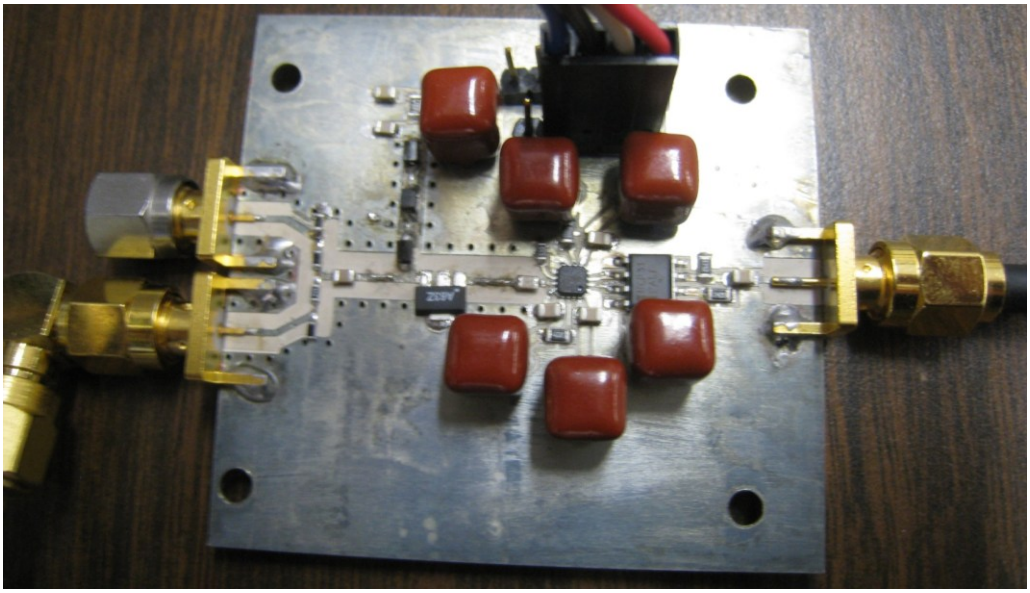

Fig. 22: Pulse Generator

Some low frequency error is noticed extending after the duration of the pulse. This error was identified as being due to reflections caused by an impedance mismatch in the calibration path. This particular pulse generator prototype is an early design with an Owen resistive splitter instead of the more efficient Adam splitter. Fig. 23 is the Fourier transform of the pulse, which has usable frequency content out to 4 GHz with the first null of the sinc function occurring at 2 GHz. The position of the null is in good agreement with the value predicted by the pulse model developed in Chapter 2.

The slight dip or notch in the spectrum of the signal at 650MHz was found to be caused by the bias network. Three series connected inductors having different resonance frequencies were evaluated in the amplifier bias network in an attempt to provide broadband RF rejection. However, further simulation in Agilent's Advanced Design System (ADS) showed that this network was ineffective in preventing specific resonant frequencies from passing through the inductors and into the power supply. One of these frequencies was 650 MHz and can be seen affecting the pulse spectrum. A more traditional design, the one now implemented in the completed pulse transceiver board, uses one high Q inductor with a resonance frequency greater than 6 GHz.

Table 2 compares the measurements of this pulse generator with some of the best performing published designs using different technologies.

Table 2: Comparison of Pulse Generators

Work	Pulse Rise Time (ps)	Pulse Width (ps)	Pulse Amplitude (V)	Technology
[9]	50	140	1.1 V	SRD
[10]	200*	600	0.28 V	BJT
[11]	--	500	0.673 V	CMOS
This Work	90	540	1.6 V	CMP

*Estimated from published figures

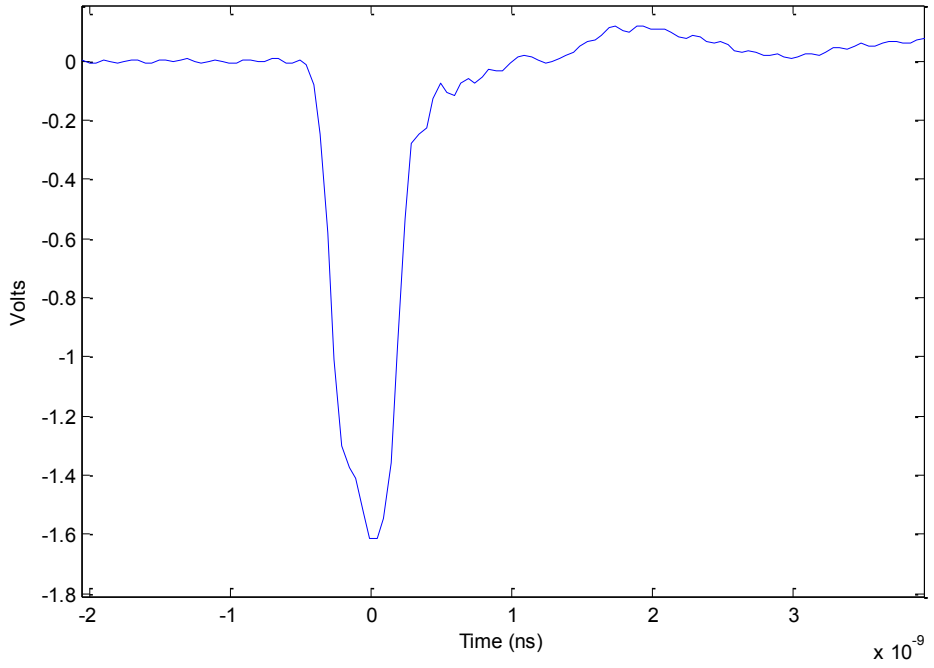

Fig. 23: A pulse from the pulse generator

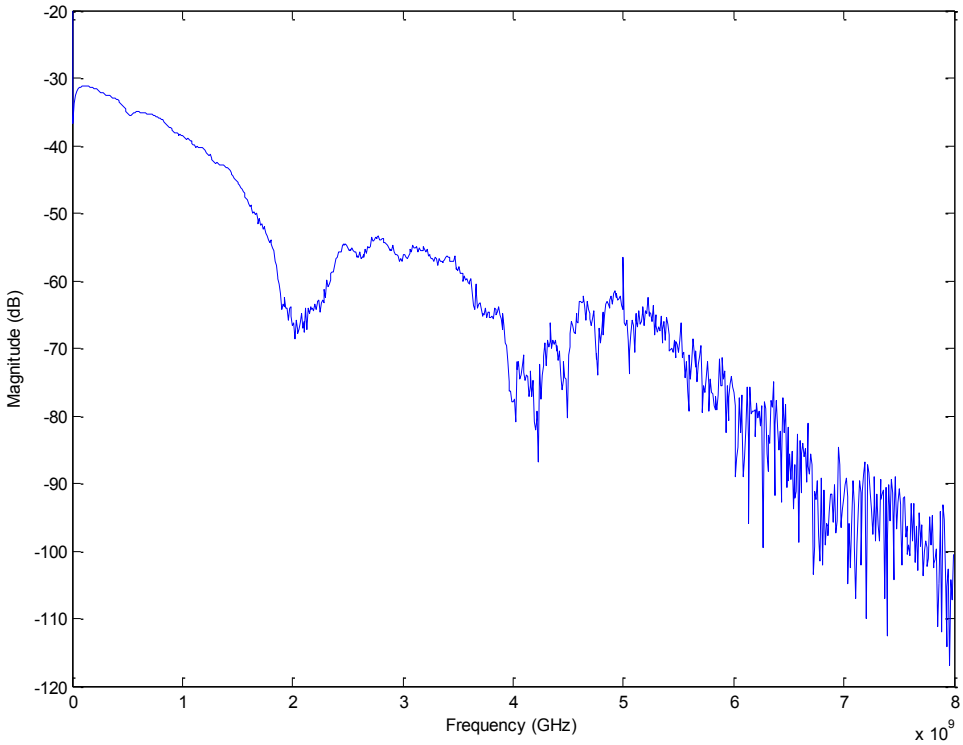

Fig. 24: Fourier transform of the transmitted pulse from Fig

Sub-Sampling Circuit

The sub-sampling circuit implements the sample and hold function necessary for equivalent time sampling. This process is described by Jean [12]. Equivalent time sampling uses two independent frequencies to control the timing of the trigger signals of the source and the sampling circuit. In the initial prototypes, the source timing frequency is 4 MHz. The sampling frequency is chosen to have a difference frequency of 64 Hz which results in the extended time factor as defined in (6).

$$\frac{f_0}{\Delta f} = \frac{4\ \text{MHz}}{64\ \text{Hz}} = 62500 \tag{6}$$

This extended time factor means that frequency content at 3.5 GHz is scaled down to 56 kHz and can be sampled by an ADC with a minimum sampling frequency of 112 kS/s. The extended time factor can be controlled to provide a tradeoff between the receiver's bandwidth and the FFT frequency resolution. For example, a 120 kS/s ADC and a 62500 ETF with a 1024 point FFT has a frequency resolution of 7.3 MHz with 3.75 GHz of bandwidth. If the ETF is increased to 100000, the bandwidth is increased to 6 GHz, but the frequency resolution has also been increased to 11.7 MHz. The larger frequency resolution means that any peaks, nulls or other features identified will have a frequency error of ± 5.85 MHz. This error could be mitigated by expanding the number of points in the FFT along with the associated computational time and cost.

The sub-sampling circuit shown in Fig. 25 is based on Zhang [13] which offers an improvement upon Han [14] by using a fully balanced structure. However, in Zhang's work the sampling circuit was triggered using a strobe step generator connected to the sampling switch circuit through a wideband balun. The circuit produces a sampling signal that has a 100 ps gating duration, but it also introduces significant ringing after the

triggering pulses. This ringing response could inadvertently bias the diodes to an ON state and introduce error into the sampled voltage.

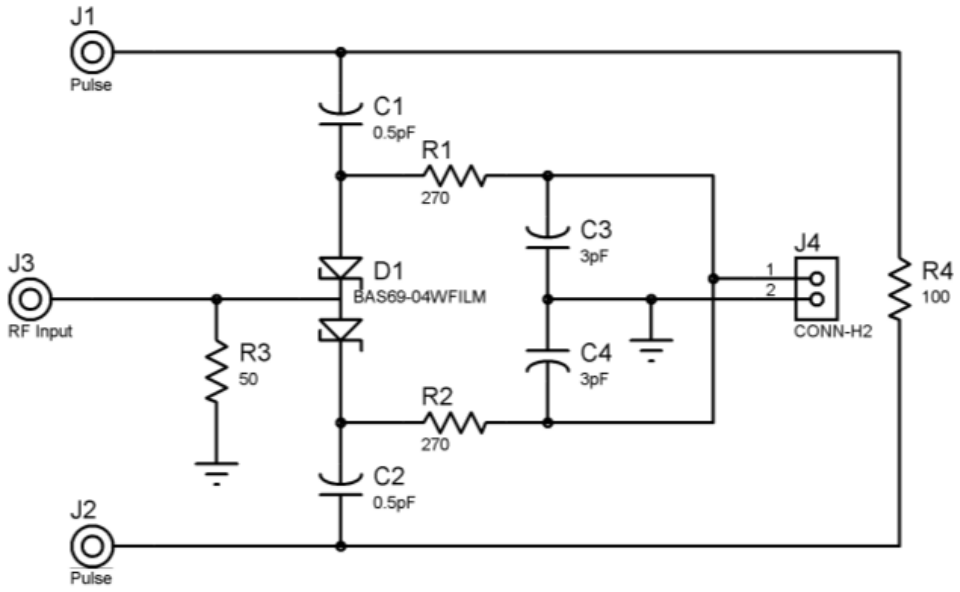

Fig. 25. Schematic of sub-sampling circuit

The triggering pulses for the sub-sampling circuit in the present design are generated by an analog comparator, which already has a balanced output. The same comparator that was chosen for the pulse generator is used to trigger the sub-sampling circuit. The comparator's balanced output eliminates the need for a balun, which are difficult to design for large bandwidths. A high speed ECL flip-flop triggers the sub-sampling circuit comparator as described above for the pulse generator. Here both the inverting and non-inverting outputs of the comparator are used. Blocking capacitors are placed on both outputs to maintain circuit symmetry, although technically only one is required to remove the DC bias in the inverting output.

Fig. 25 shows the sampling circuit used in this work. HSMS-286C Schottky detector diodes were chosen for their low series resistance (6 ohms) and high sensitivity.

R3 and R4 are used for impedance matching. C1 and C2 are the RF holding capacitors which form an RC discharge circuit with the diode resistance. The RF time constant is set at 36 ps to provide a small sampling window. The inside resistors and capacitors (R1, R2, C3, C4) also form a RC discharge circuit to hold the voltage for the intermediate frequency (IF). The IF time constant is set to approximately 810 ps so as to maintain the voltage level from sample to sample.

The sub-sampling circuit needs two differential pulse signals to forward bias the sampling diodes at the proper sampling time. The prototype of the triggering circuit, shown in Fig. 26, was constructed independently to verify its operation. The output from the sample-control comparator with external DC blocks is shown in Fig. 27. As expected, the output amplitude is 400 mV and the ringing is minimal.

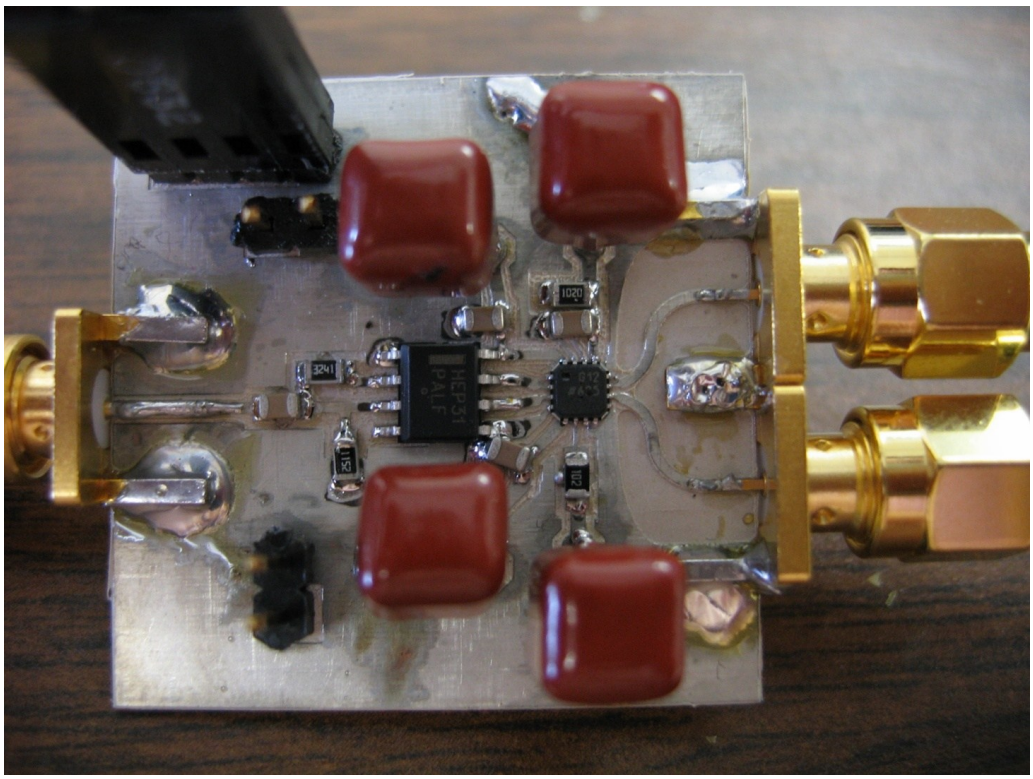

Fig. 26: Sub-Sampling Circuit Triggering Board

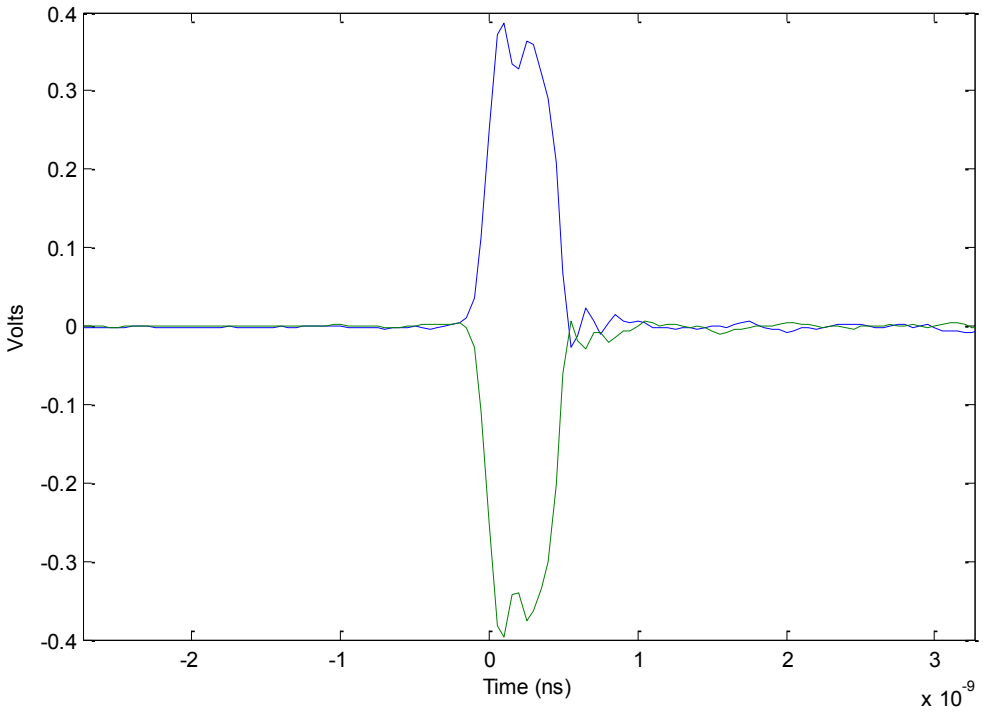

Fig. 27: Output pulses from the Sub-Sampling Circuit Triggering Board

The sub-sampling circuit in Fig. 28 was designed using size 0603 surface mount parts.

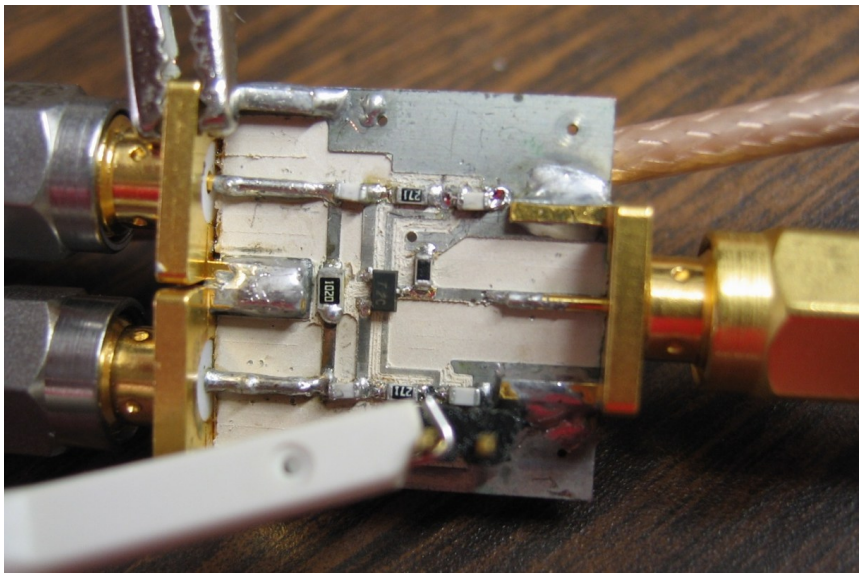

Fig. 28: Sub-Sampling Circuit

The width of these parts closely matched the width of the 50 Ω microstrip transmission lines and reduced trace impedance mismatches. Fig. 29 shows an overlay of the pulse from the pulse generator in red, as measured by a LeCroy Wavemaster 8500 oscilloscope, with the sub-sampled pulse from the receiver measured with a Yokogawa 1640 oscilloscope. The sub-sampling circuit output demonstrates good agreement with the transmitted pulse in amplitude and the correct expected extended time scale. However, a low frequency discrepancy can be noticed at the beginning of the pulse. The source of this discrepancy can be traced to the arrangement of the component elements. The measurement setup involved a chain of the pulse generator, sub-sampling circuit, a clock generation board, and the sub-sampling triggering board. Power was individually connected to the all of the boards except the sub-sampling board, which is passive. Thus the only ground path for the sub-sampling board was connected through the SMA connectors, which caused the error noticed at the beginning of the pulse.

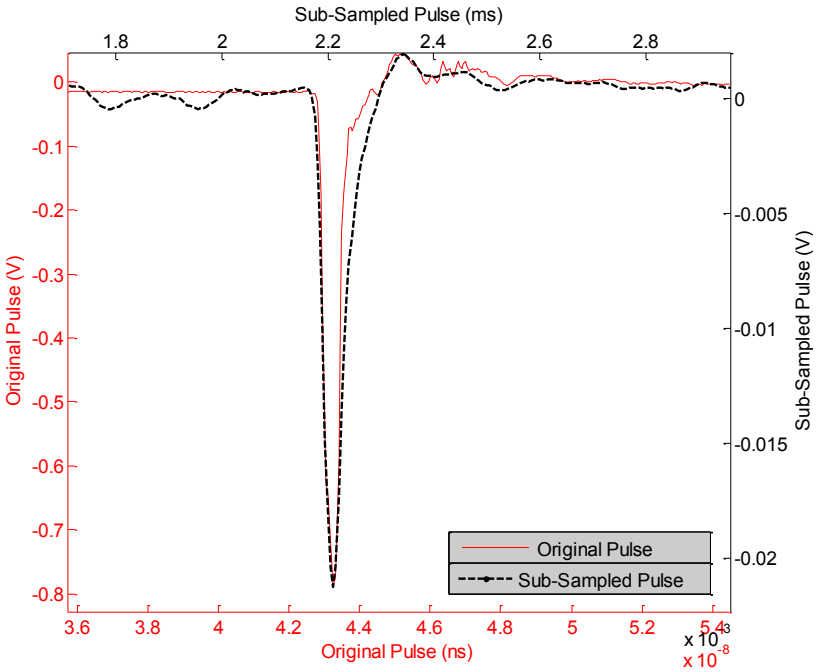

Fig. 29: Comparison of Transmitted Pulse to Sub-Sampled Pulse

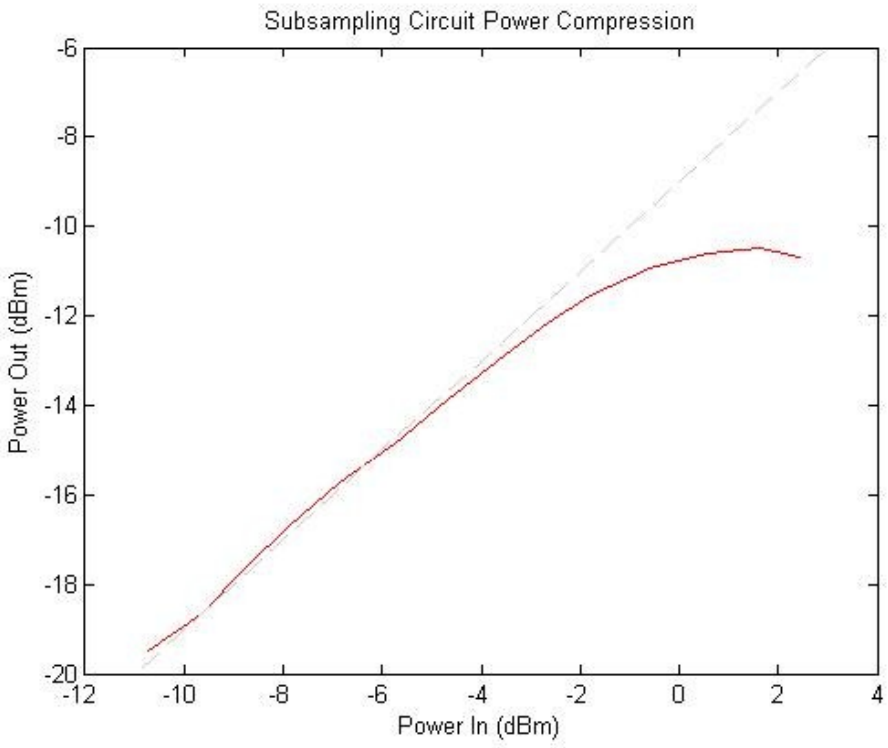

Fig. 30: Sub-sampling circuit power compression

Pulse Transceiver Revision A

Once acceptable performance with the individual circuit elements was achieved, the designs for the pulse generator, sub-sampling circuit, and sub-sampling triggering circuit were then combined into a single board. A digital stepped attenuator (DSA) was also added in series with the output of the generator as a means to increase the dynamic range of the transceiver via control of the pulse amplitude. The attenuator is controlled with 5 parallel bits, giving 32 steps with 1 dB of resolution. The attenuator allows for the measurement of a low loss material by preventing receiver saturation by adding up to 31 dB of attenuation to the output pulse. The sub-sampling circuit begins to saturate at -4 dBm, so if the signal path is lossy, such as the case for concrete in a GMS waveguide,

then the DSA is set to zero attenuation. If the PFTNA was designed specifically for measuring high loss materials, then the DSA does not need to be included in the design.

For the layout, the signal isolation through-holes are spaced closer than half a wavelength for the highest frequency in the pulse and an extra row is added around the sub-sampling circuit as shown in Fig. 31.

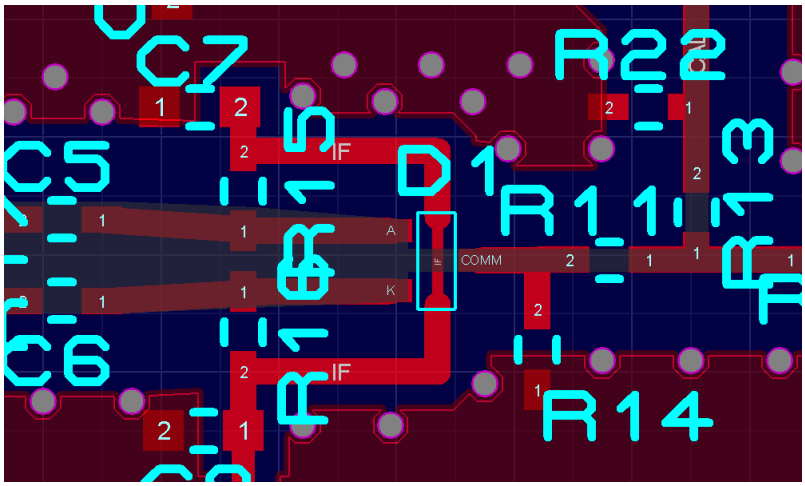

Fig. 31. Portion of the Pulse Transceiver RF PCB layout focusing on the sub-sampling circuit

These closely spaced holes provide isolation between the transmit pulse output and the input channel of the receiver. The sub-sampling circuit is sensitive to very small voltages and its close proximity to the pulse generator with a single board design increases its susceptibility to crosstalk. Fig. 32 shows the entire pulse transceiver after fabrication.

Normally the output pulse quickly follows the calibration pulse, depending upon the delay based on cable lengths. The pulse shown in Fig. 33 is the internal calibration pulse and was obtained by terminating the transceiver's ports in 50 ohms which eliminated the second transmitted pulse.

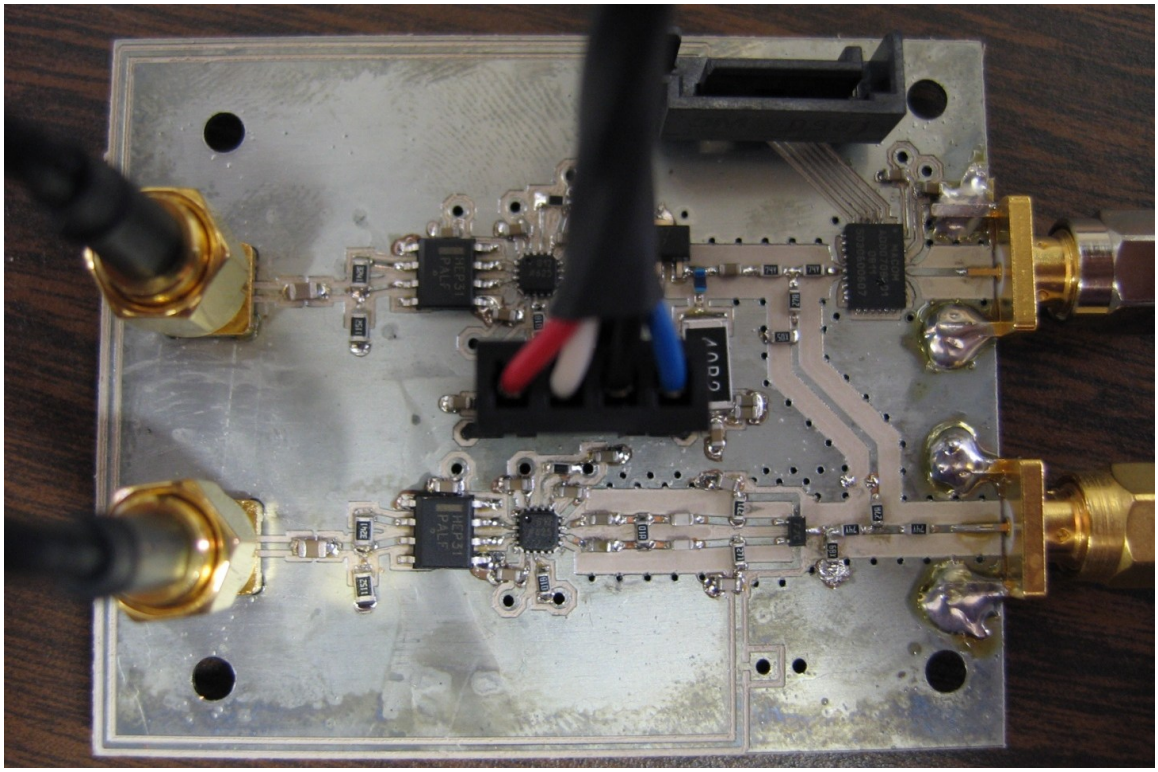

Fig. 32: Fabricated Pulse Transceiver

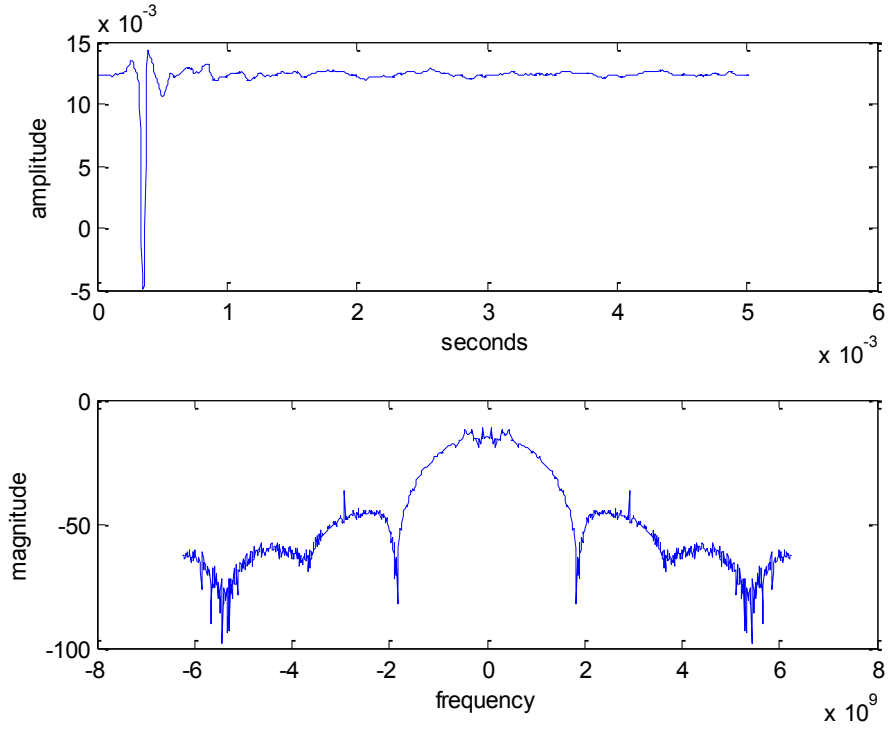

Fig. 33: Calibration pulse and FFT

The 50 ohm termination was used for this calibration to include the low frequency error occurring after the pulse. This low frequency error is clearly visible in the time domain following the pulse as well as in the spectrum of the calibration pulse in second section of Fig. 33. For a test measurement, water in a GMS waveguide was used as the MUT shown in Fig. 34.

Fig. 34: Guided Microwave Spectrometry Waveguide

Fig. 35 shows the time and frequency domain of a single dispersed pulse from the GMS waveguide. This data was obtained by truncating the beginning of the signal which contained the calibration pulse. This calibration pulse is the exact same pulse featured in the first millisecond in Fig. 33. Note that some error from the calibration pulse can be seen preceeding the dispersed pulse.

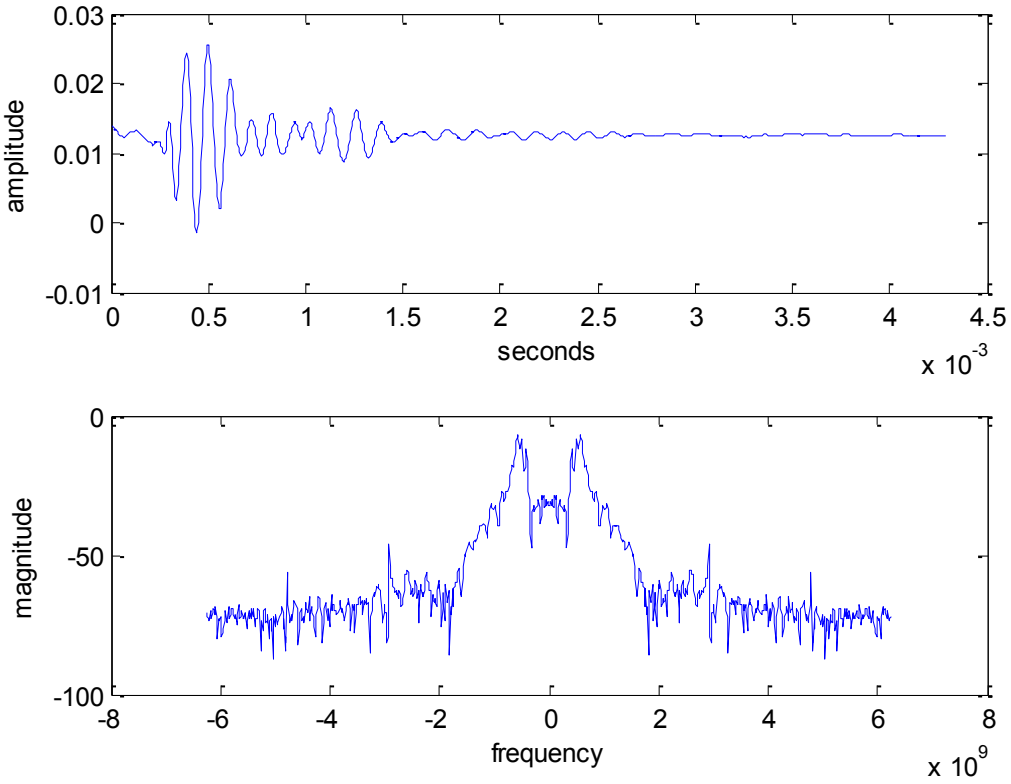

Fig. 35. Dispersed pulse and FFT

Fig. 36 shows a comparison between the responses measured by a VNA and the pulse transceiver for a GMS waveguide filled with tap water. The pulse transceiver data was produced by subtracting the spectrum of the dispersed pulse in Fig. 35 from the spectrum of the calibration pulse in Fig. 33. The data has fairly good agreement up to 1.5 GHz. Since the first null in the spectrum of the transmitted pulse occurs at 2 GHz, performance will degrade close to that frequency. The discrepancy between DC and 400 MHz can be attributed to the VNA having a significantly lower noise floor. At the lower frequencies the GMS waveguide filled with water has a high level of attenuation which approaches the noise floor of the pulse transceiver.

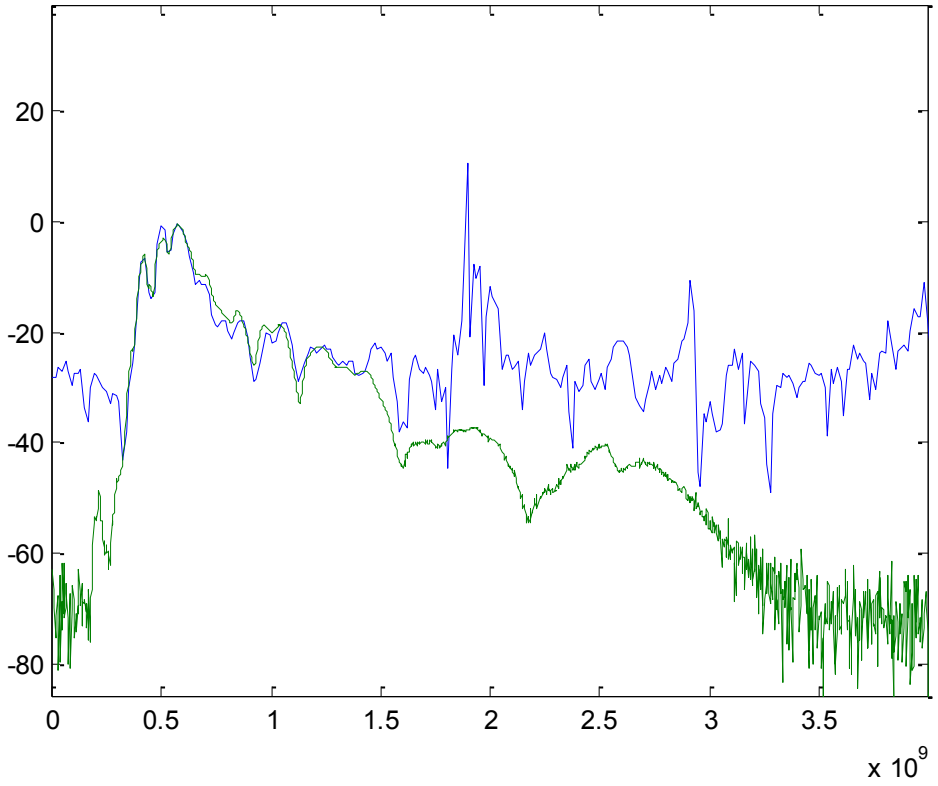

Fig. 36. Comparison of dispersed water from the pulse transceiver and VNA

Fig. 37 shows a comparison between the pulse transceiver and a VNA for the measurement of a 650 MHz low pass filter. Again there is some error in the lower frequency range due to some reflections from the calibration pulse. The low pass filter's cutoff frequency shows very good agreement between the two measurement systems. However, the graph clearly illustrates the noise floor shortcomings of the system, which degrade to only 35 dB of loss at higher frequencies. The HSMS-286C diodes only have 55 dBm of tangential sensitivity which sets the unamplified noise floor of the system. Tangential sensitivity is the lowest powerlevel that the pulse can be detected, but does not guarantee any spectrum accuracy.

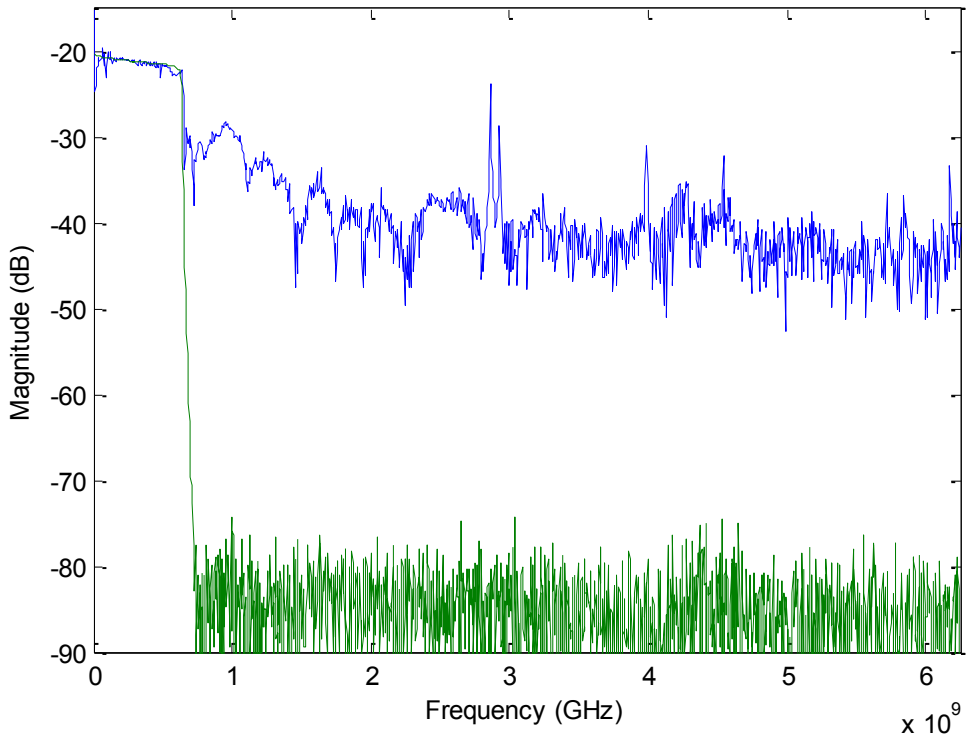

Fig. 37. Comparison of a VNA and Pulse Transceiver on a low pass filter

A test of the accuracy of the pulse transceiver was conducted by measuring external attenuators of 3, 6, 10, 20 and 30 dB. During this test the digital stepped attenuator (DSA) was varied to maintain a constant input power level to the receiver. The DSA allowed the pulse from each external attenuator under test to have a large amplitude while not saturating the receiver. The results of this test in Fig. 38 show some deterministic error present in the measurements. This error can be attributed to reflections in the calibration path. The reflections then overlap the second measurement pulse and cannot be removed. Since most of the reflections are not actually transmitted to the MUT, they are completely erroneous. The error could have been calibrated out, but future designs will attempt to eliminate it.

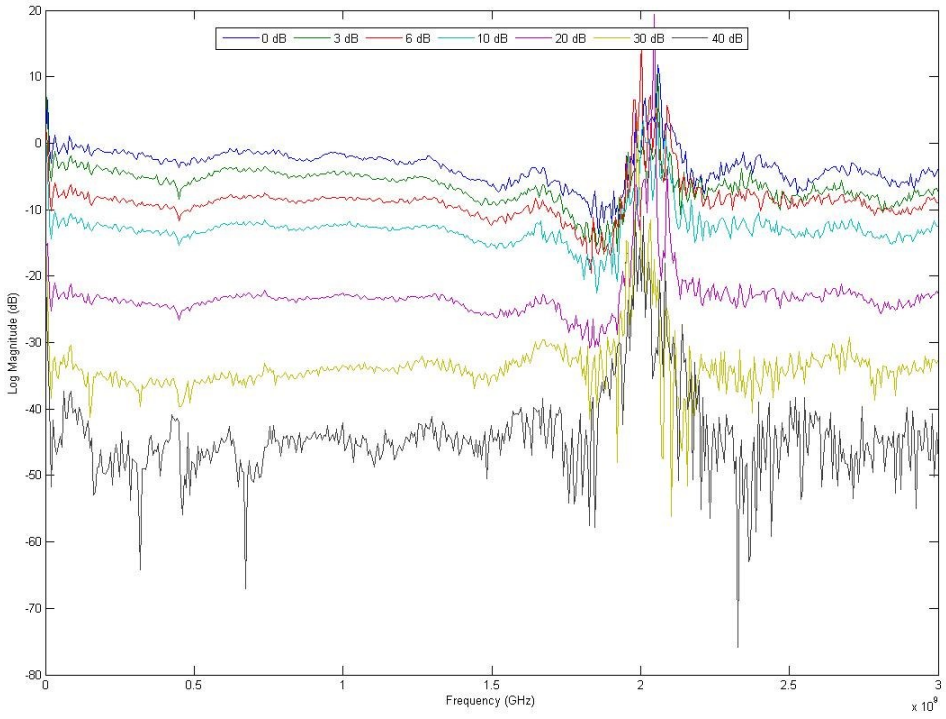

Fig. 38. Accuracy test of the pulse transceiver against external attenuators

Stub Test

To reduce the error, the two Adams splitters were unsoldered from the pulse transceiver, which removed the calibration pulse. A pulse for calibration was acquired by directly connecting the input and output of the transceiver using a 6-inch SMA cable in place of a MUT. In performing this test, another source of the error was found to be the small inductance caused by using an unshielded twisted pair carrying the IF signal from the pulse transceiver to the control board. The problem was alleviated by shortening the IF cable. Future designs will incorporate an operation amplifier circuit with low input bias currents on the pulse transceiver board to maintain a high impedance source load for the sub-sampling circuit.

A circuit was fabricated to mimic the electrical performance of a new non-invasive blood glucose sensor. This circuit was designed with three nulls in frequency occurring at 25MHz, 500MHz and 1GHz. The network was then measured with a VNA and compared to a measurement taken by the pulse transceiver. The result of this comparison is shown in Fig. 39.

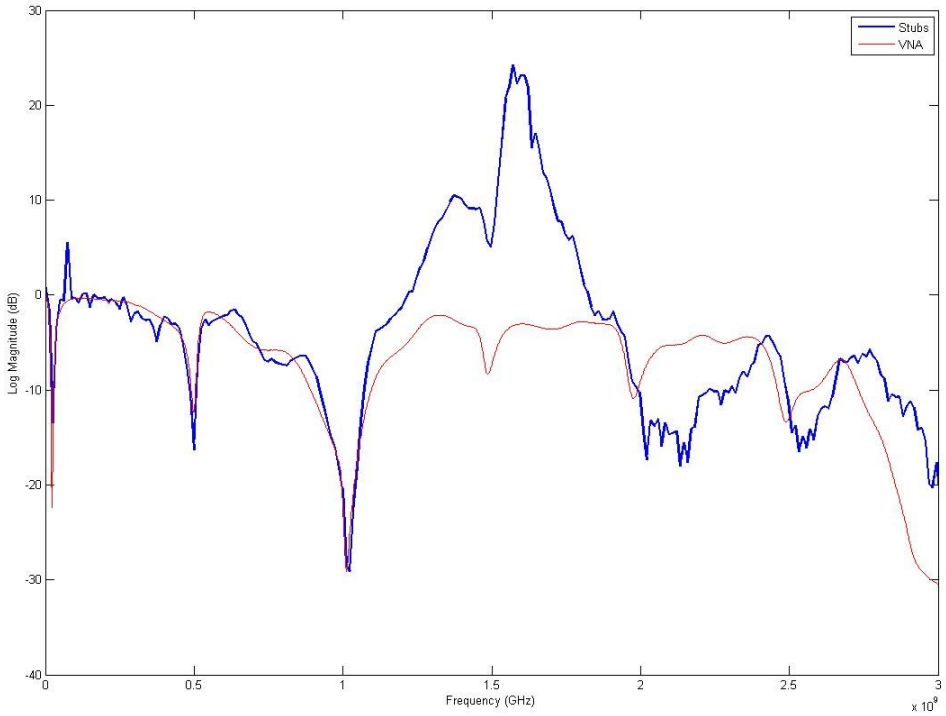

Fig. 39. Comparison of a stub network measured with a VNA and the pulse transceiver

Table 3 tabulates a comparison of the null positions in frequency as well as the associated error of the pulse transceiver. The pulse transceiver demonstrates exceptional performance as the frequency error of the high frequency nulls is less than one percent. Note that because the pulse transceiver is using 2048 FFTs the accuracy of each frequency point is limited to ±6.1MHz. A discrete Fourier transform (DTF) could be

used to compute more points specifically around the nulls to increase the accuracy. The large error between 1.2 and 1.8 GHz is due to null in the pulse's spectrum.

Table 3: Stub location comparison

	Vector Network Analyzer	Pulse Transceiver	Error
Null 1	22.8 MHz	24.41 MHz	7.06%
Null 2	499 MHz	500.5 MHz	0.30%
Null 3	1011 MHz	1019 MHz	0.79%

Pulse Transceiver Revision B

The calibration path in revision A provided an attenuated pulse from the generator with every measurement. Small impedance mismatches in the calibration path created ringing of the calibration pulse that overlapped onto the received pulse. The overlap created systematic frequency error that decreased the accuracy of the system. To remove the error, the calibration path was de-soldered and calibration was accomplished externally with a cable, just as a vector network analyzer's through-port calibration.

Since this system is designed to be embedded in a consumer application it is necessary for it to have the ability to self-calibrate and minimize any extra steps the consumer would have to take. The calibration path was redesigned as a switched configuration using two AS196-307LF single pole dual throw switches. The switches are spec'd from 300 kHz to 6 GHz with 0.9-1.6 dB of insertion loss and 30-55 dB of isolation, but with a P1dB of only 17 dBm. This signal compression point is slightly lower than ideal and the switch will reduce the amplitude of the output pulse, but it's inclusion in the design was considered an acceptable tradeoff.

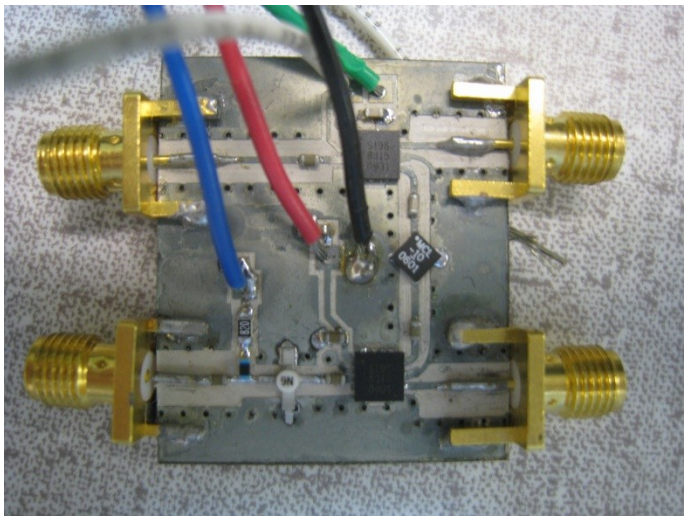

Fig. 40: UWB Switch Prototype Board

The high-speed flip-flops in revision A were dissipating a considerable amount of heat that was not expected for normal operation. The flip-flops use 5V ECL logic, and are designed to drive a 50 ohm load referenced to 3V instead of ground. With the flip-flops driving the comparator's 50 ohm load referenced to ground they were sourcing an extra 120 mA of current to a total of 144 mA, just slightly under an absolute maximum rating of 150 mA. This extra heat had to dissipate in one small ground pin, as the package used for the flip-flop did not have a thermal pad. The heat caused the ground solder joint to fail every few weeks, making the design unacceptable.

The design change for the heating problem involved the use of a "Y" connection where the termination pins of the two comparator inputs are tied together. When tied together, a common mode voltage is allowed to develop on the termination pins that can then be pulled down with a resistor. This configuration is shown in Fig. 41.

To layout the new "Y" connection, the RF termination had to be moved to the bottom side of the board. This move necessitated two 0.5mm vias for the RF termination

ports, which is generally not advisable for high frequency traces. The size of the PCB, seen to be larger than necessary from a circuit performance perspective, was designed to fit inside a 1457N1202 aluminum case from Hammond Manufacturing rendered in Fig. 42.

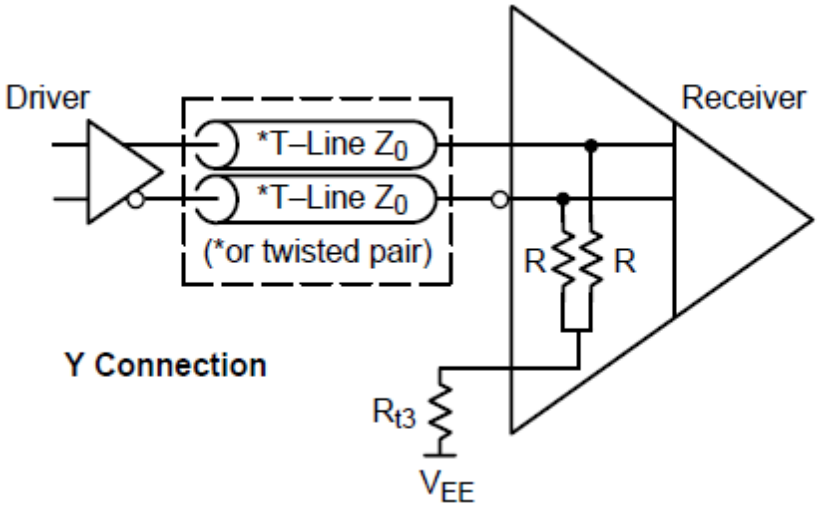

Fig. 41: Method of termination for an ECL receiver

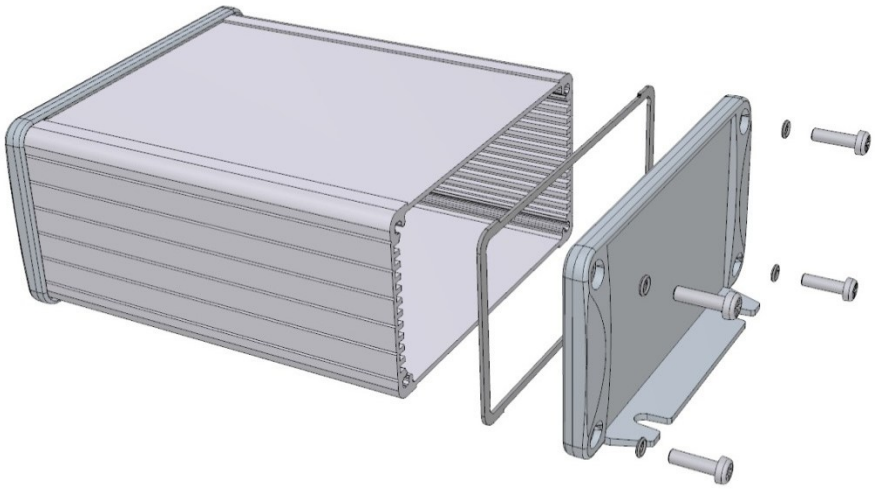

Fig. 42: Aluminum case for the FTNA

Fig. 43 shows the fabricated Pulse Transceiver Revision B printed circuit board. Jumpers were added for the switches and the digital stepped attenuator which added the ability to independently control them. In normal operation with a separate control board these jumpers are to be left open.

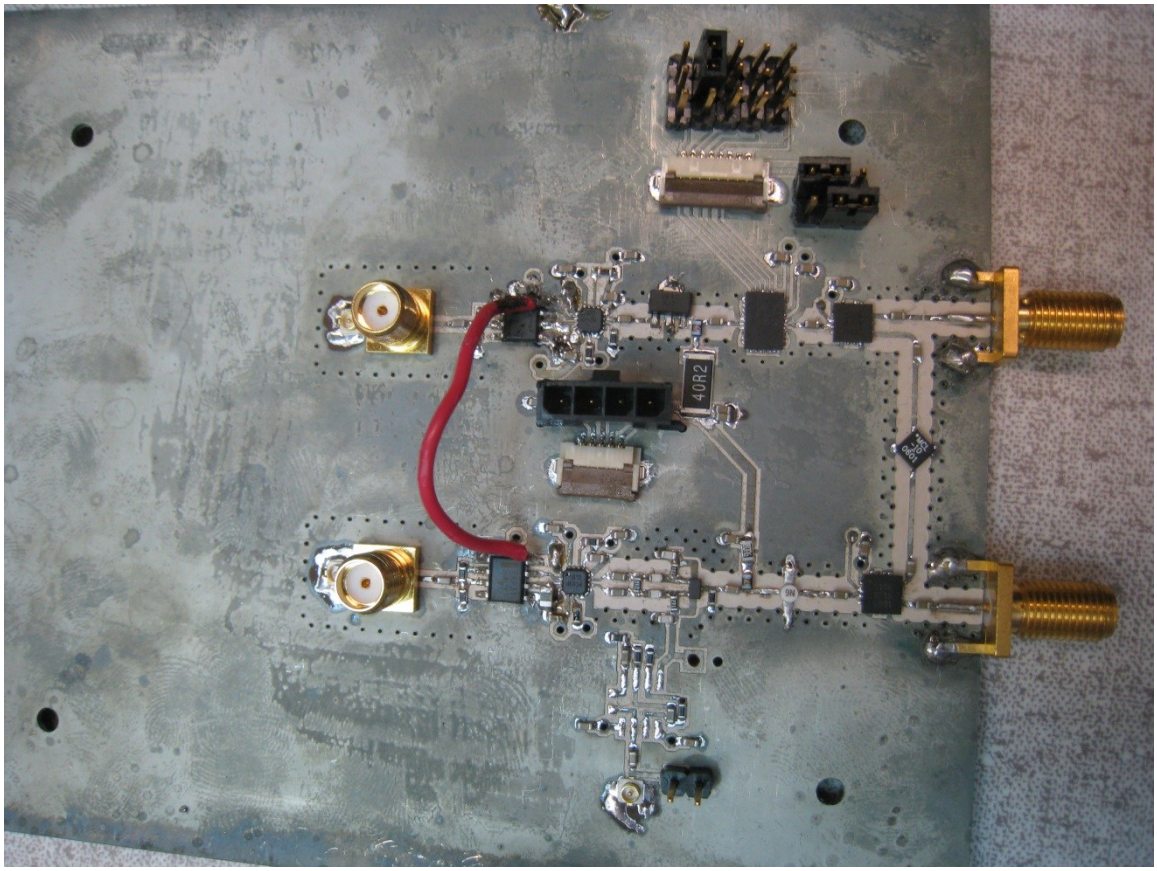

Fig. 43: Fabricated Pulse Transceiver Rev B

Fig. 44 shows the pulse generated by Revision B. Note the erroneous shape of the waveform after the pulse. It was determined that the error was due to reflections from the comparator termination. The termination's via inductance caused significant ringing, a problem which necessitated a design change for this portion of the circuit.

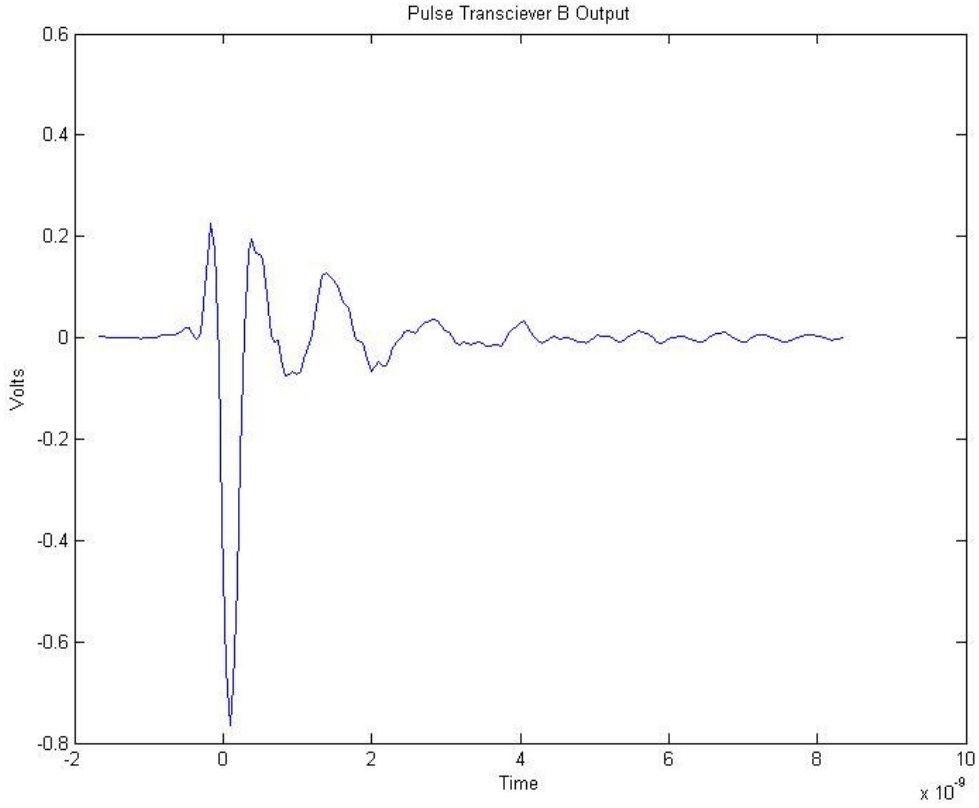

Fig. 44: Pulse Transceiver Revision B

Pulse Transceiver Revision C

Since the solution to remedy the flip-flop overheating issue implemented in revision B caused reflections, an alternative design was needed. ECL has varying logic levels depending on the supply voltage. For example, if the supply voltage is 5V, then the logic levels are referenced to 3V. If the circuit is powered at 3.3V, then the logic is referenced to 1.3V. 3.3V logic was then chosen to reduce the overheating caused by having the comparator referenced to ground. Removing the "Y" termination from revision B and going back to simply grounded terminations as in revision A, while changing the flip-flop's supply voltage reduced the flip-flop's sourced current to 78 mA. This value would nearly halve the extra power dissipation caused by the ground

termination. The change would not eliminate all excess heat, but substantially reduce it without compromising the quality of the generated pulse.

The power conditioning ICs were included into the design and moved to the center of the PCB, since most of the board is not heavily populated, as shown in Fig. 45. This design would reduce any inductance or noise for the pulse transceiver's power supplies due to electromagnetic interference (EMI) from the previously used external cables.

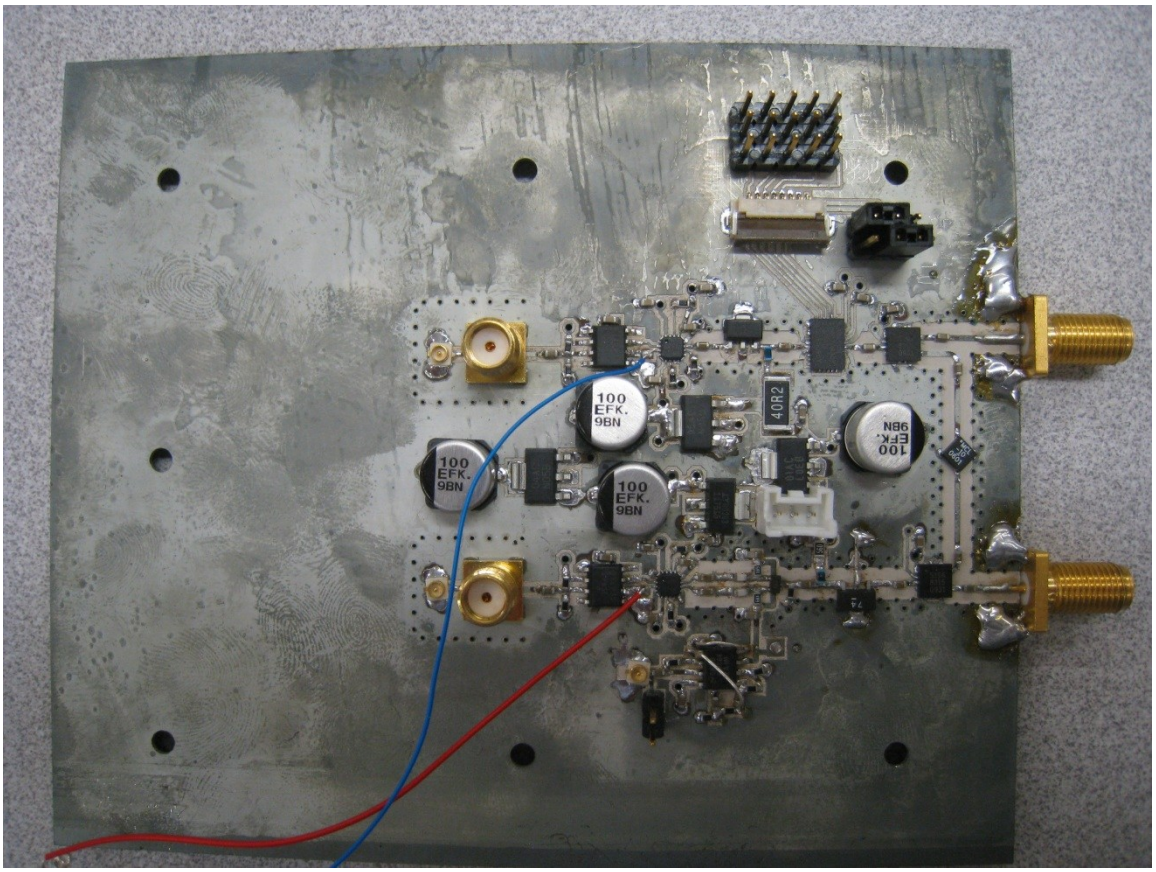

Fig. 45: Fabricated Pulse Transceiver Rev C

Revision C included a two stage operational amplifier circuit that was initially on the control daughter board discussed in Chapter 4. The low voltages from the sub-

sampling circuit are especially vulnerable to EMI and needed to be amplified before exposed to board-to-board cables. It was also found that the added cable impedance negatively altered the sub-sampling circuit's performance. Another performance degradation was attributed to the bias current of the operational amplifier on the control daughter board as the sub-sampling circuit cannot source very much current. This limitation led to the selection of the LMC662 from National Semiconducter for the opamp used in revision C due to its 2fA bias current.

Since five voltage regulators were moved onto the RF board, the board dissipates a large amount of heat. This heat was found to negatively affect the performance of the diodes in the sub-sampling circuit by causing a decrease in the amplitude of the sub-sampled pulse by nearly half as shown by Fig. 46.

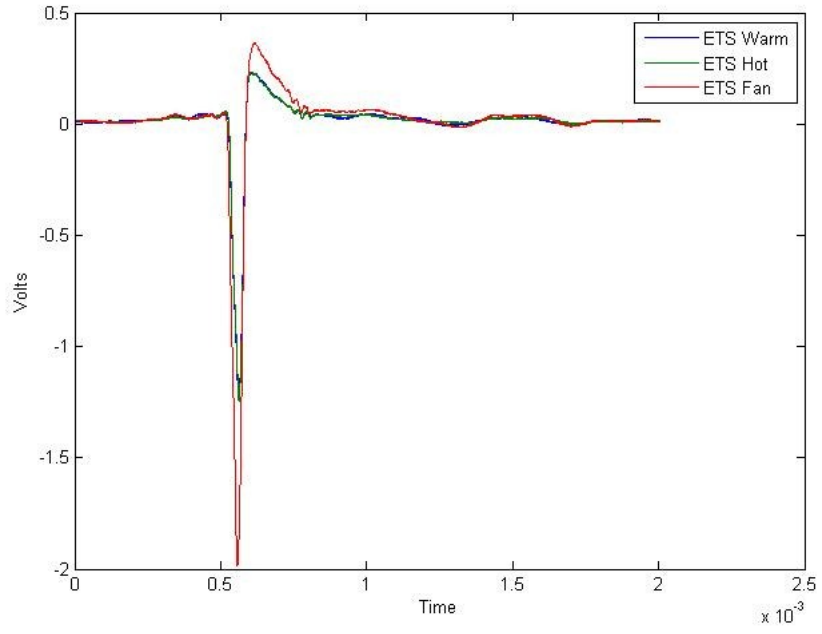

Fig. 46: Pulse transceiver output over temperature

In considering this effect, it was thought that the diodes were not biased with enough voltage to adequately forward bias them. Fig. 47 shows the diodes IV curves on a log scale. The difference between a 400 mV forward bias and a 350 mV forward bias causes a large decrease in forward current.

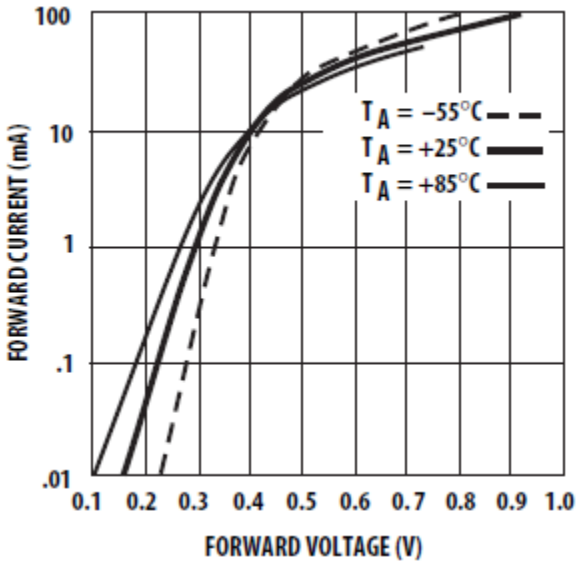

Fig. 47: HSMS-286C IV curve

A small PCB was made with two amplifiers that were used to increase the bias voltage. This PCB was used with older prototype boards to see if a larger bias voltage reduced the diode's temperature sensitivity. However, it was determined that increasing the bias voltage did not relieve the temperature amplitude fluctuations enough to warrant the added complexity of two additional amplifiers.

This triggering for the pulse transceiver revision C was slightly modified to greatly increase the generated pulse's bandwidth. Rather than use the differential output of the self-resetting flip-flop, only the positive terminal, Q, is routed to the non-inverting input of the comparator. A DC voltage is then applied to the inverting input of the

comparator to trigger it at the peak of the flip-flop's output "glitch". The flip-flop's "glitch" is significantly narrower towards the peak amplitude than the width at the median voltage levels. The DC voltage can then directly control the comparator's pulse width while not affecting the pulse rise and fall times. Fig. 48 shows the time domain pulse with varying DC levels. Note that the pulse width can now become shorter than the comparator's rise times, which begins to lower the pulse's amplitude.

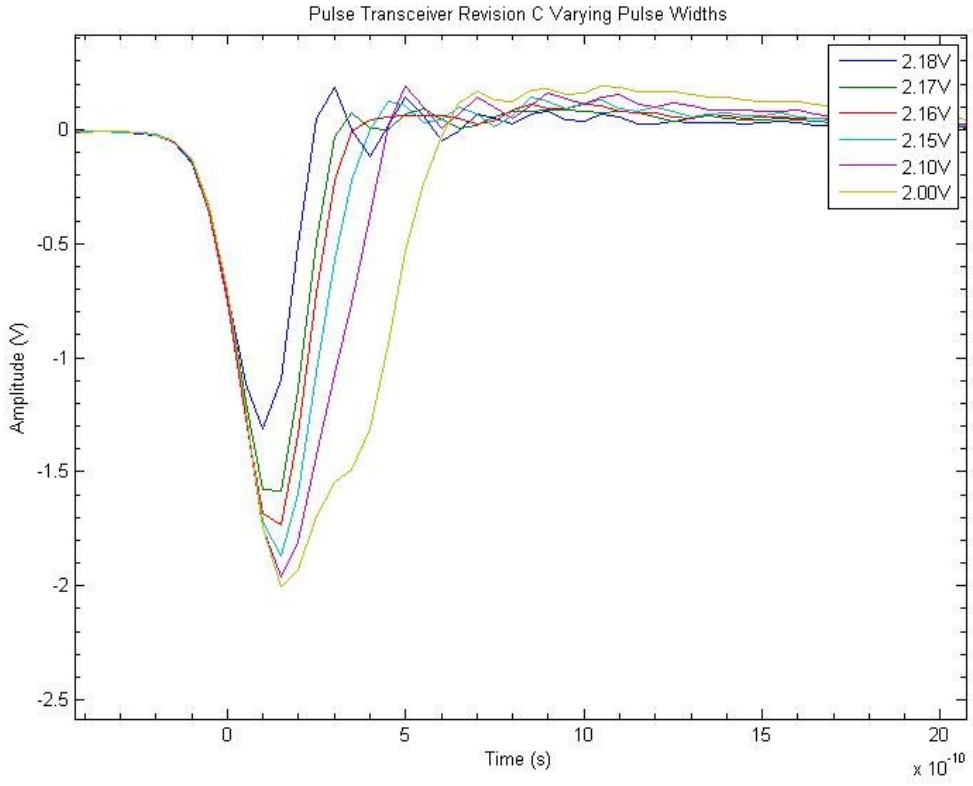

Fig. 48: Pulse Transceiver Revision C Varying Pulse Widths

The frequency spectrum of the pulses in Fig. 48 is shown in Fig. 49. The 2.00 V trigger level's spectrum is close to the spectrum from pulse transceiver revision A with a first null at 2.1 GHz and a 1.4 GHz 10 dB bandwidth. The 2.18 V trigger level's spectrum is significantly flatter with a first null that is beyond the range of the 5 GHz

LeCroy Oscilloscope and has a 3.5 GHz 10dB bandwidth. This pulse more than doubles the bandwidth of the pulse generator, but also sacrifices 10 dB in output amplitude. However, the pulse width is now tunable and could offer increased amplitude with decreased bandwidth if desired.

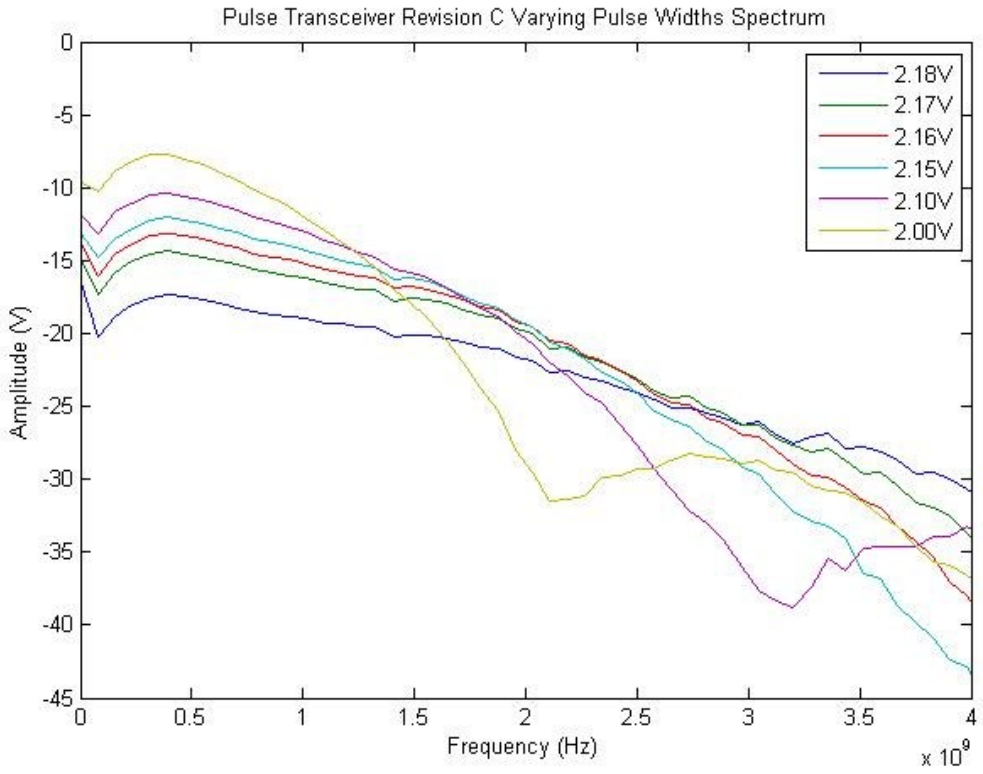

Fig. 49: Pulse Transceiver Revision C Varying Pulse Widths Spectrum

Fig. 50 shows a comparison measuring water inside of a GMS waveguide using an Agilent E5071B VNA, a prototype Fourier Transform Network Analyzer, and the Pulsed Fourier Transform Network Analyzer spectrum as captured by the Rev C PCB. The figure shows that the pulse transceiver has 40-55 dB of dynamic range and fairly good accuracy. The estimated costs of the three systems are $30000, $2000, and $200

respectively. Some of the measurement discrepancy among the three systems can be attributed to varying water levels as each measurement was taken on separate days.

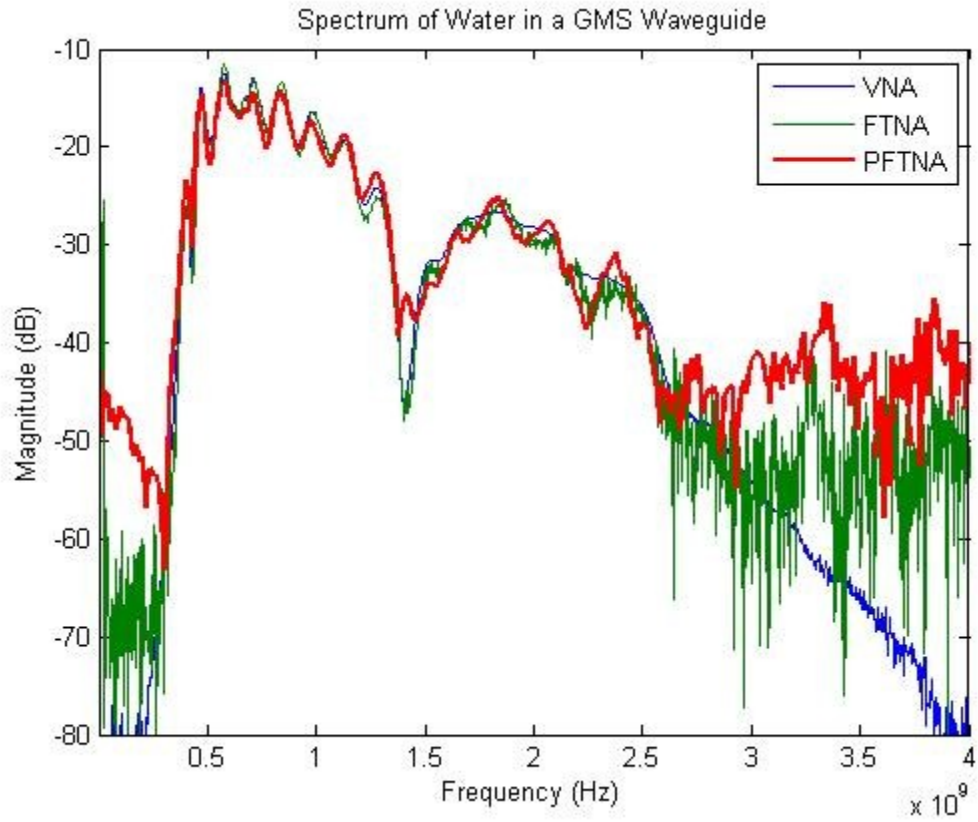

Fig. 50: Spectrum of Water in a GMS Waveguide

The full schematics, layout, and bill of materials (BOM) for the pulse transceiver revision C are provided in Appendix A.

CHAPTER FOUR

Control

Fig. 51 shows an initial control prototype for the pulse transceiver revision A. This prototype consisted of the PIC32 Starter Kit, a PIC32 I/O Expansion Board, a Graphics PICtail Plus Daughter Board with 3.2inch Display Kit from Microchip Technology, and a custom control daughter board. The Starter Kit features a PIC32MX360F512L microcontroller and was used for the control and digital signal processing (DSP). The I/O Expansion board provided a connector for the Graphics daughter board as well as connectors for the control daughter board. The Graphics Daughter Board was used for an SSD1926 LCD Graphic Controller from Solomon Systech which drives a 3.2 inch 320x230 LCD from Truly.

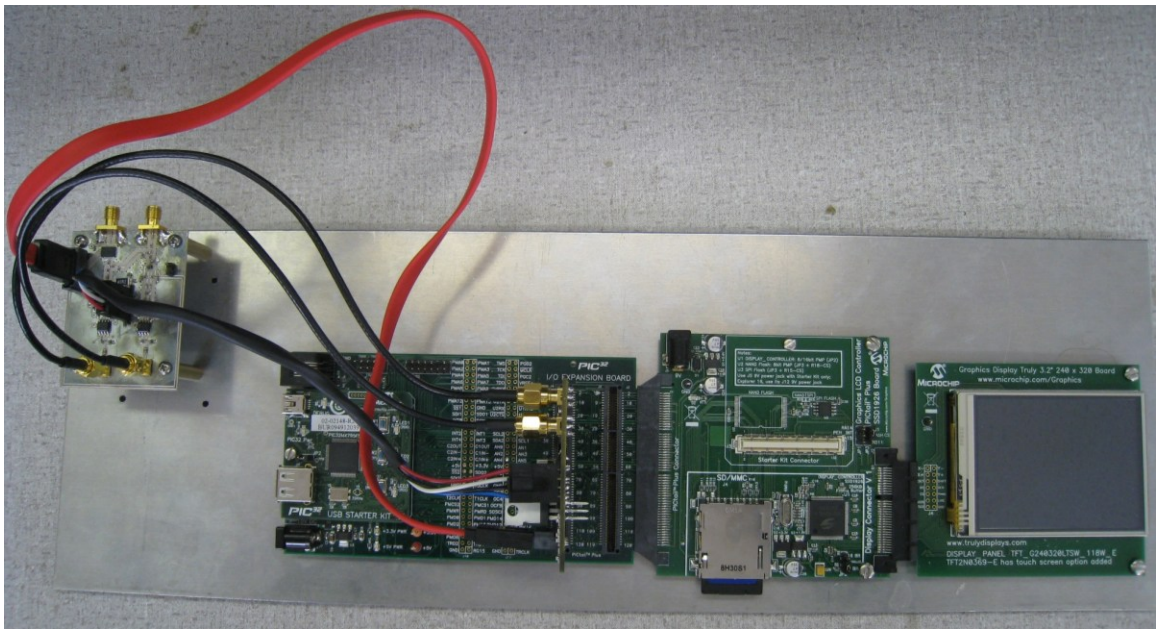

Fig. 51. Pulsed Transceiver System Prototype

An external DDS board made by PECO Manufacturing, Inc. was used in the early stages of circuit development for the equivalent time sampling before a suitable DDS design was developed.

Control Daughter Board

The control daughter board shown in Fig. 52 was designed to provide power, intermediate frequency (IF) amplification, and the two equivalent time sampling clocks.

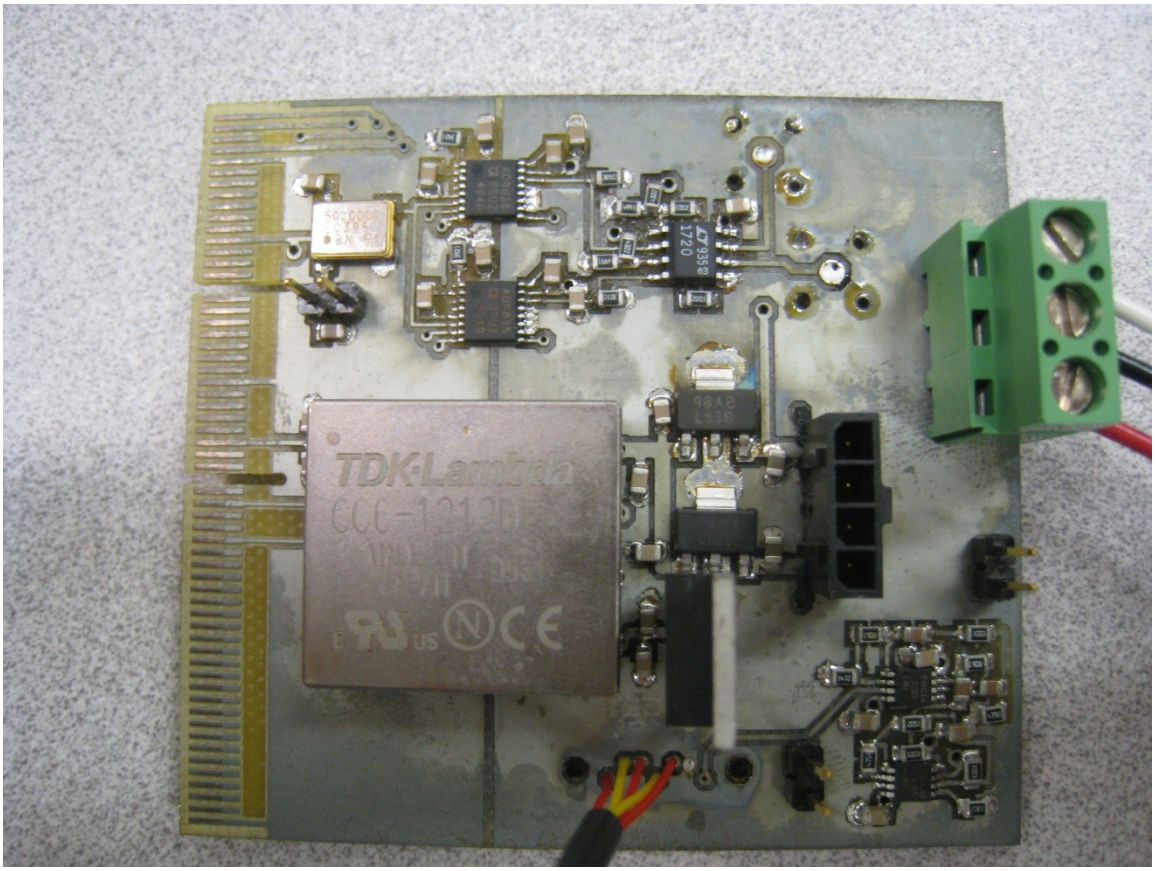

Fig. 52: Control Daughter Board

This board plugged into a PIC32 I/O Expansion board from Microchip. The IF amplification was accomplished with a three-stage operational amplifier circuit. The first stage was a non-inverting amplifier with a gain of 1.5 that provides a high impedance

load for the sub-sampling circuit. The AC-coupled second-stage non-inverting amplifier removed any DC from the IF signal. AC coupling was chosen for the second stage, rather than for the first stage, so that a high capacitance load would not be presented to the sub-sampling circuit. The third stage was another non-inverting amplifier. A DC voltage was added to the signal in the third stage to center the bi-polar dynamic range of the received signal in the middle of the voltage window of the microcontroller's ADC. The first two stages of this design were moved to the pulse transceiver PCB as previously described.

The control daughter board also provided power conditioning for the analog IC's on the high frequency board. The board can use any DC source from 9 to 15 volts such as an AC-to-DC wall transformer. The input power supply voltage is routed to a DC/DC switching regulator which produces ±12 V. The ±12V are routed to independent linear regulators which produce each necessary output voltage. This design eliminates the need for a large and expensive external linearly regulated power supply.

Two AD9835 direct digital synthesis (DDS) IC's from Analog Devices were used to create the timing signals necessary for equivalent time sampling. These IC's produce sine waves that are used as the input to two Schmitt trigger circuits, which convert them into square waves. The two DDS chips are programmed independently via a Serial Peripheral Interface Bus (SPI). They share common clock and data signals but have separate frame synchronization signals. This shared arrangement requires the use of two fewer pins on the microcontroller and simplifies the communication programming.

The early design of the DDS clock generation circuits exhibited large amounts of clock jitter. This jitter was due to several harmonic frequencies produced by the DDS

chips and their 50 MHz clock. To prevent the jitter, a high order low-pass filter was found to be necessary on each output. An evaluation board for a different, but similar, DDS chip from Analog Devices employed 5th order inverse Chebyshev filters. The hoped for simplification attempted by the omission of output filters from early designs of the control board was found to be generally inadvisable.

Direct Digital Synthesis Prototypes

Three different prototype boards for direct digital synthesis were designed, fabricated and evaluated for their effectiveness to reduce the clock jitter. The first design was using two AD9834 and a dual input ADCMP563 high speed comparator. This design increased the DDS clock speed by 25 MHz and added two 5th order low-pass Butterworth filters. The increased DDS clock speed and added filters were not enough to reduce the clock jitter to within acceptable limits.

Another design prototype used two AD9952s with two 7th order inverse Chebyshev filters. These DDS IC's can be clock at 400 MHz but are driven by a 25 MHz oscillator for this design. The clock signal was routed on the bottom of the PCB. Unfortunately, the clock trace got too close to the one of the DDS outputs on the bottom and added excessive amounts of 25MHz jitter. The PCB was meant to be a quick prototype to test the design and then integrate it into a larger PCB, so separate analog and digital power suppliers were not used. It turned out that separate power supplies are essential for generating a clean set of timing signals.

The next design iteration used a 25MHz crystal with the AD9952 onboard oscillator to provide the DDS clocking signals. The DDS chips were positioned facing each other with the crystal equal distance in trace length from both IC's. Separate analog

and digital power supplies were used with bypass capacitors. Since the layout was limited to two layers there was not room for a large power zone on the bottom layer. The conclusion from the design iteration was that a layout was needed with separate ground, power and signal planes.

Further investigation into the clocking jitter problem revealed that any frequency jitter from the reference oscillator or crystal will increase the spurious harmonics by 24dB on the DDS output due to the phase lock loop. The addition to the harmonics is quantified by (6) where x is the PLL multiplier and $N(x)$ is the added harmonic content in decibels. This relationship makes a low-cost sub-picosecond clock generator difficult to design because a high frequency source is needed to drive the integrated circuits.

$$N(x) = 20 * \log(x) \tag{6}$$

Another attempt at generating the clocking signals necessary for equivalent time sampling was not attempted until the design of the control board. That design is fully functional and described in the following section.

Control Board

To reduce the size of the initial prototype system in Fig. 51, a separate control board was designed to run the pulse transceiver revision C PCB independently. The control board provides the following major functions: power generation, user interface, analog and digital signal processing, data collection, and equivalent time sampling clock generation. The full schematics, layout, and bill of materials are provided in Appendix A.

For the user interface, the control board contains the SSD1926 LCD graphics driver. Communication between it and the PIC32 is accomplished with the Parallel

Master Port (PMP) which is a 16 bit data bus with read and write control lines. The control board has a jumper, J23, which can set the SSD1926 to operate in 8 bit mode. The LCD is the TFT-G240320LTSW-118W-E with the TFT2N0369-E resistive touch screen option from Truly Semiconductors. It is the same 3.2 inch 320x230 LCD that was used in the initial prototype. The LCD is affixed to the bottom side of the control board which is ultimately the top of the PFTNA housing. A 28 pin FPC connector, the FH12A-40S-0.5SH(55) from Hirose, is used for the data signals between the SSD1926 and the LCD, and the SFW4R-3STE1LF from LCI is used as the connector for the 4 wire resistive touch screen. Note that the two connectors do not have mating contacts on both the top and bottom of the connector, so the user cannot freely interchange cables. The two cables for the LCD and touch screen are routed through two cutouts in the PCB. Fig. 53 shows the top side of the control board with the pulse transceiver connected and suspended in the top of the figure.

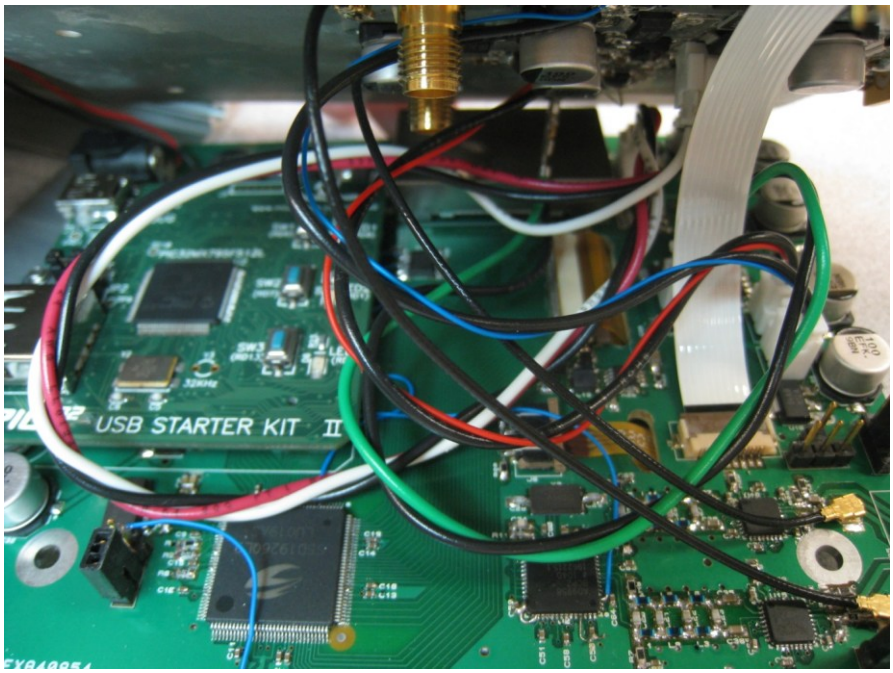

Fig. 53: Top view of the Control Board

The control board has a two stage operation amplifier circuit to maximize the dynamic range of the A/D converter for pulse receiver's output. This circuit uses a 10 bit digital to analog converter (DAC), the DAC7574 from Texas Instruments, to set the DC voltage level, and a 50kΩ 8 bit digital potentiometer (pot), the ISL95810 from Intersil, to control the gain. A voltage divider was used between the DAC and the non-inverting input of the second opamp stage so that the output DC offset has zero gain. When the pot's resistance between the wiper and "low" terminal is set to 10kΩ the circuit has unity gain. Equation 7 approximates the output of the operational amplifier circuit.

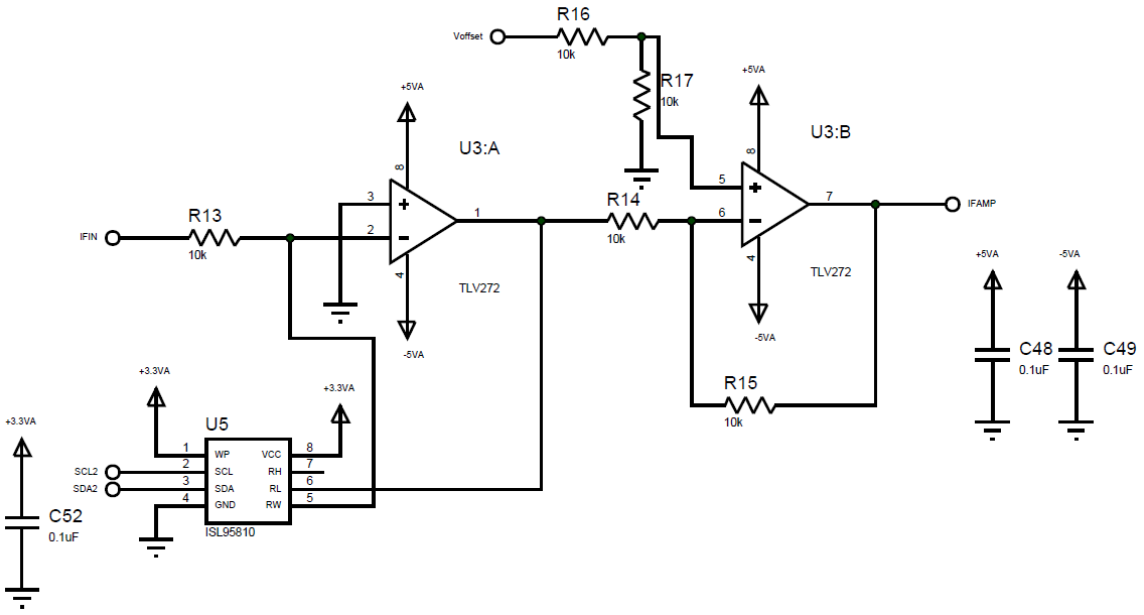

Fig. 54: Intermediate Frequency Signal Processing Circuit Schematic

$$IF_{out} = IF_{in} * \frac{R_{pot}}{10k} + V_{DAC} \tag{7}$$

Both the DAC and pot are controlled via an Inter-Integrated Circuit (I^2C) bus. Thus the microcontroller, and ultimately the user, has full control over the DC offset voltage and gain of the signal. The DAC used for the DC offset also produces the two

voltages needed to set the trigger level for the comparators on the pulse transceiver. The PA series of connectors from JST were chosen for the two DC voltages and IF output signal. A twisted ground wire accompanied each signal as an electromagnetic interference shield.

The control board also features the DS1721 temperature sensor from Maxim Integrated Products. This integrated circuit will allow for the possibility of temperature compensation if needed. It is also controlled with the same I^2C bus. The control board design also includes a 32Mbit flash memory integrated circuit from Microchip Technology. The SST25VF032 uses an SPI bus and can be clocked up to 80MHz. This memory is used to store calibration data and user settings.

The equivalent time sampling trigger signals uses a single direct digital synthesis IC, the AD9958, which was dual outputs, but no waveform-shaping comparators. Switching to a single DDS chip design eliminates some layout problems. It also simplified synchronization of the initiation of the time cycle defined by the two signals. Syncing is essential when pulse delay is to be measured, which is necessary for time transit tomography applications. The two differential sinusoidal waves are then connected to two differential 7^{th} order 135 MHz low pass Chebyshev filters. Two AD9515 clock buffering circuits are used to convert the sinusoidal waves at the output of the filters into square waves. The clock buffering circuits are capable of dividing the clock frequency by a factor 2 to 32. Using the clock division feature allows the DDS ICs to operate at higher frequencies than is required for the system clock which reduced the system clock jitter. Two shielded ultra miniature coaxial cables were used to route the clocking signals to the pulse transceiver PCB.

The layout for the control board is a 4 layer design with a dedicated power plane and ground plane for the two inner layers. The two outer layers are signal layers and the top component layer also has a ground fill. The fabrication of this PCB was unsuccessfully attempted in-house, and was outsourced to PCB FAB Express. The layout design has separate analog and digital power supplies and power zones which meet in the middle for analog ICs that need digital communication. The dimensions of the PCB were designed to match the pulse transceiver revision C and fit inside of the 1457N1202 case from Hammond Manufacturing. Fig. 55 shows the PCB for the four layer control board.

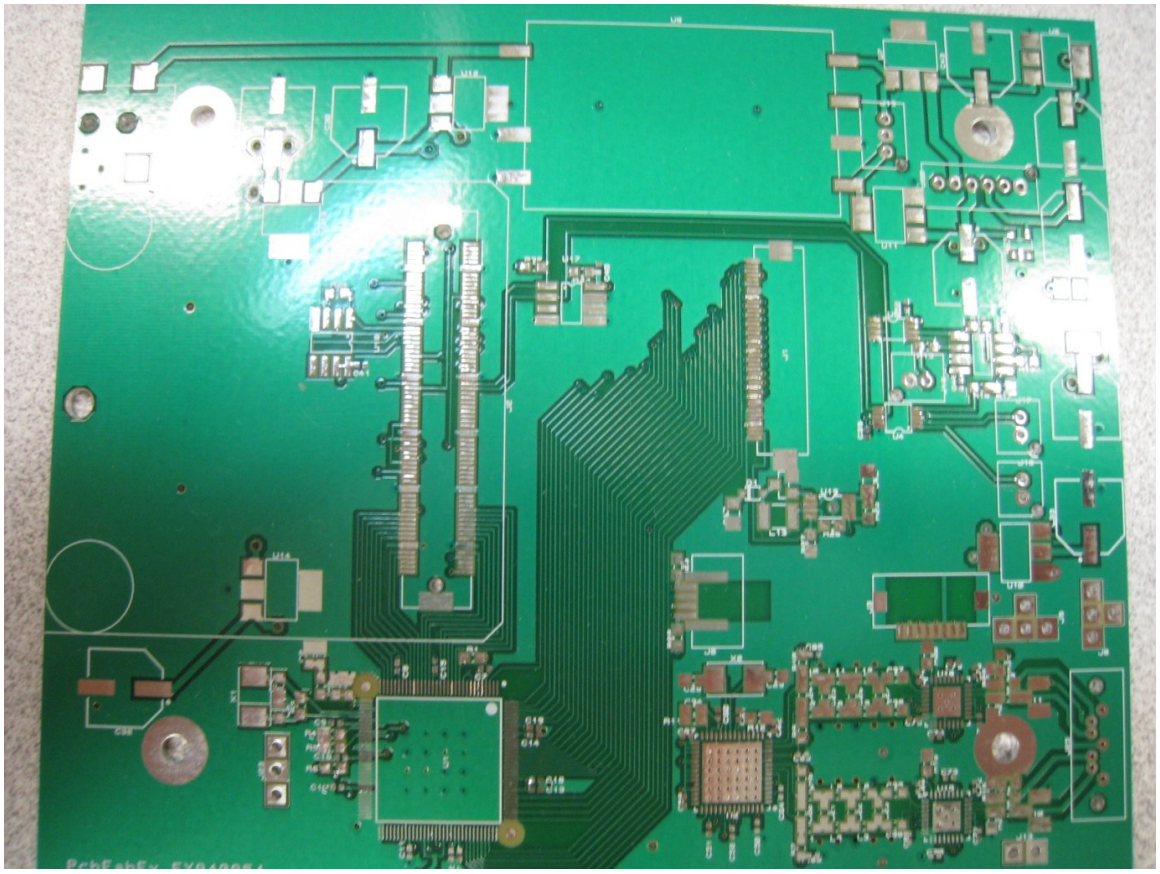

Fig. 55: Control Board PCB

Microcontroller

A PIC32 microcontroller from Microchip was chosen for data processing. This microcontroller is low cost and provides ample processing power for the required digital signal processing (DSP) operations. The available PIC32 development kits share a common connector which allows for easy upgrading, as was done when the PIC32MX795F512L came out replacing the PIC32MX360F512L. Microchip microcontrollers are prolific in the marketplace and have a well-established base of free software libraries. These libraries allow for easier implementation of a user interface and the DSP functions. The schematics for the PIC development kit from Microchip are included in Appendix A.

The following software was written in C using Microchip's MPLAB version 8.70 for the IDE. The compiler was C32 v1.12a. Note that this version has a linker error related to the peripheral library macros/functions for the 32-bit timers, TMR23 and TMR45. A fix is available from Microchip's C32 website and is necessary to compile the code. The code was written with the use of Microchip's Application Libraries released October 19, 2010 specifically using the Graphics Library v2.11, MCHPFSUSB Framework v2.8, and the Memory Disk Drive v1.2.7.

The analog-to-digital converter (ADC) capture period is programmed to be clocked from the PIC's Timer3, which has been set at 200 kHz. This clock rate allows for a maximum RF input frequency of 6.25 GHz to be sampled without aliasing.

Digital Signal Processing

The FFT algorithm used to convert the time-domain signals to the frequency domain is included in Microchip's DSP library, but written by MIPS Technologies. The

algorithm operates on 32 bit fixed-point numbers and the length of the FFT is controllable. Fixed-point number representation is commonly used with microcontrollers that do not have a floating point processor. The PIC32 MIPS library uses Q31 format for fixed-point numbers which has 31 fractional bits and one sign bit. The Q31 fixed-point format scales numbers to range from -1 to 1, requiring a 10-bit integer from the ADC to be converted to a purely fractional number. This conversion is accomplished with a scaling factor of 1/2048 which will scale binary integers from 0 to 1024 to 0 to 0.5 in fixed-point format. The ratio 1/2048 was chosen as the scaling number to avoid clipping of fixed point values yet minimize precision loss. Clipping could still occur, for example, if the ADC output of three sequential samples are each valued at 1024. Each value gets converted to 0.5 and the resulting FFT should have a maximum value of 1.5. However Q31 has a maximum value of 1, thus the output of the FFT will clip. Since clipping is a possibility, the fixed-point format conversion algorithm checks if clipping occurred and will report the event if the microcontroller is running in debug mode. Each power of two that the scaling factor is decreased by, halves the probability of clipping, but reduces the FFT's dynamic range by 1 bit or 6dB.

For the specified sampling frequency of 200 kHz, each bin of the 2048 length FFT represents a frequency window of 97.65Hz, and when multiplied by the extended time factor, gives an effective bandwidth in true signal space of 6.1MHz. Triggering is simply accomplished by waiting for the input signal to reach a preset reference voltage. A circular buffer is used to store the samples of the input signal before a trigger is detected. Once a trigger occurs a small number of samples are captured after the trigger and used with the circular buffer to reconstitute the complete calibration pulse. After the

calibration pulse is received, the digitized dispersed pulse is then directly written into an array. The amount of averaging can be varied but is set at 32 for the data presented in this thesis.

User Interface Software

The user interface (UI) software, written in C, was created by using software components available in Microchip's free Graphics Library. The graphics library is a part of the Microchip Applications Library which also contains functions for the Universal Serial Bus (USB). The program originated from modified example code included with the graphics library. The main user input screen that was written for a non-invasive blood glucose sensor application is shown in Fig. 56.

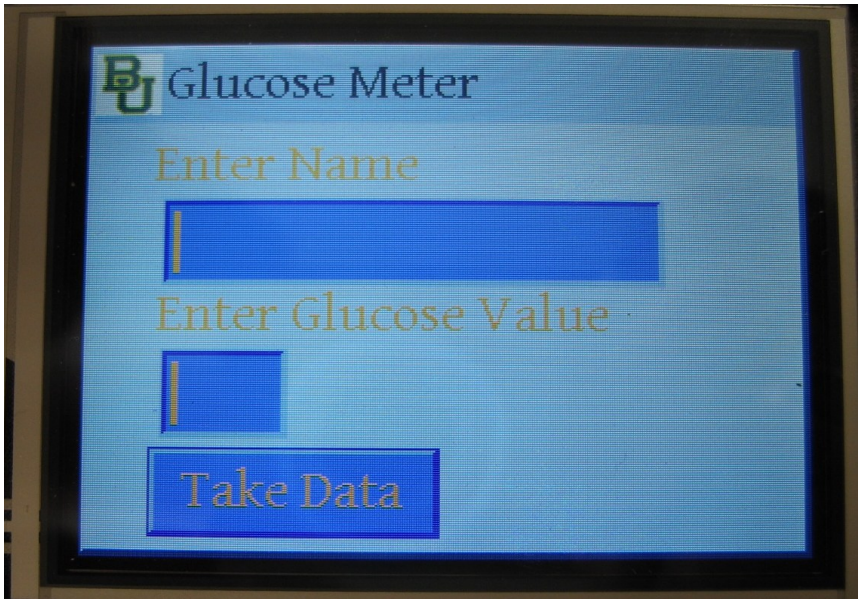

Fig. 56. Screen Shot of the User Interface

If the user touches the name or glucose value input blocks, the program brings up a separate screen for easy text or numerical entry.

Another user interface was created for non-specific user applications shown in Fig. 57. This user interface provides the user with the ability to enter specimen numbers and control the output power. The output filename can also be controlled for data collection.

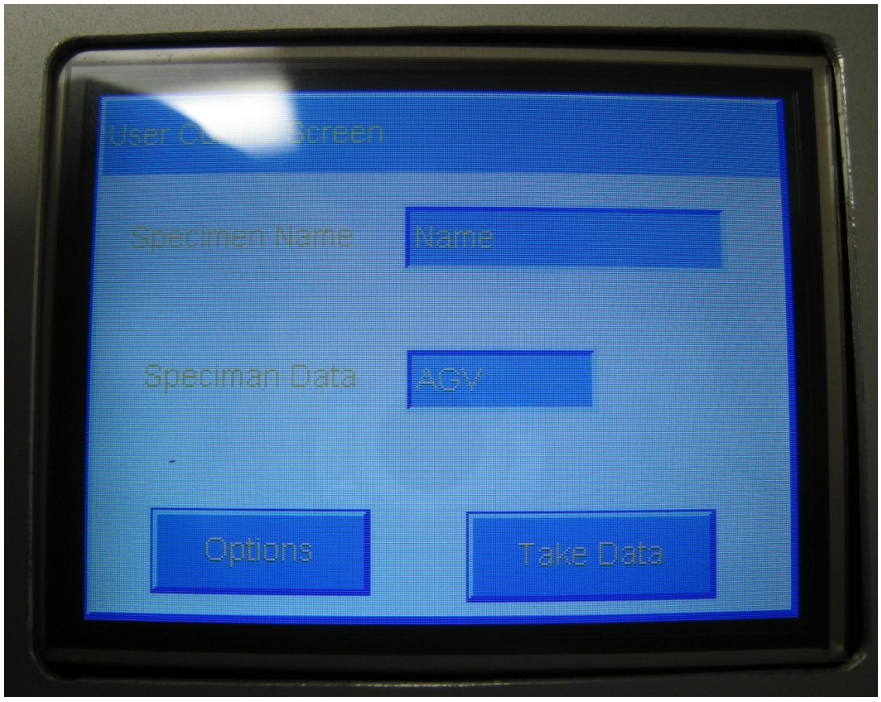

Fig. 57: PFTNA Non-specific Application User Interface

An additional user interface was coded for the PFTNA without a specific application shown in Fig. 58. This interface displays the time and frequency domain waveforms while providing the user with specific control over the hardware. There are controls to adjust the output power, gain, DC offset, and trigger level. The user can also save the data internally and externally. The system can be calibrated by the Cal button which allows the user to control if the pulse is switched to the SMA ports or to the receiver internally.

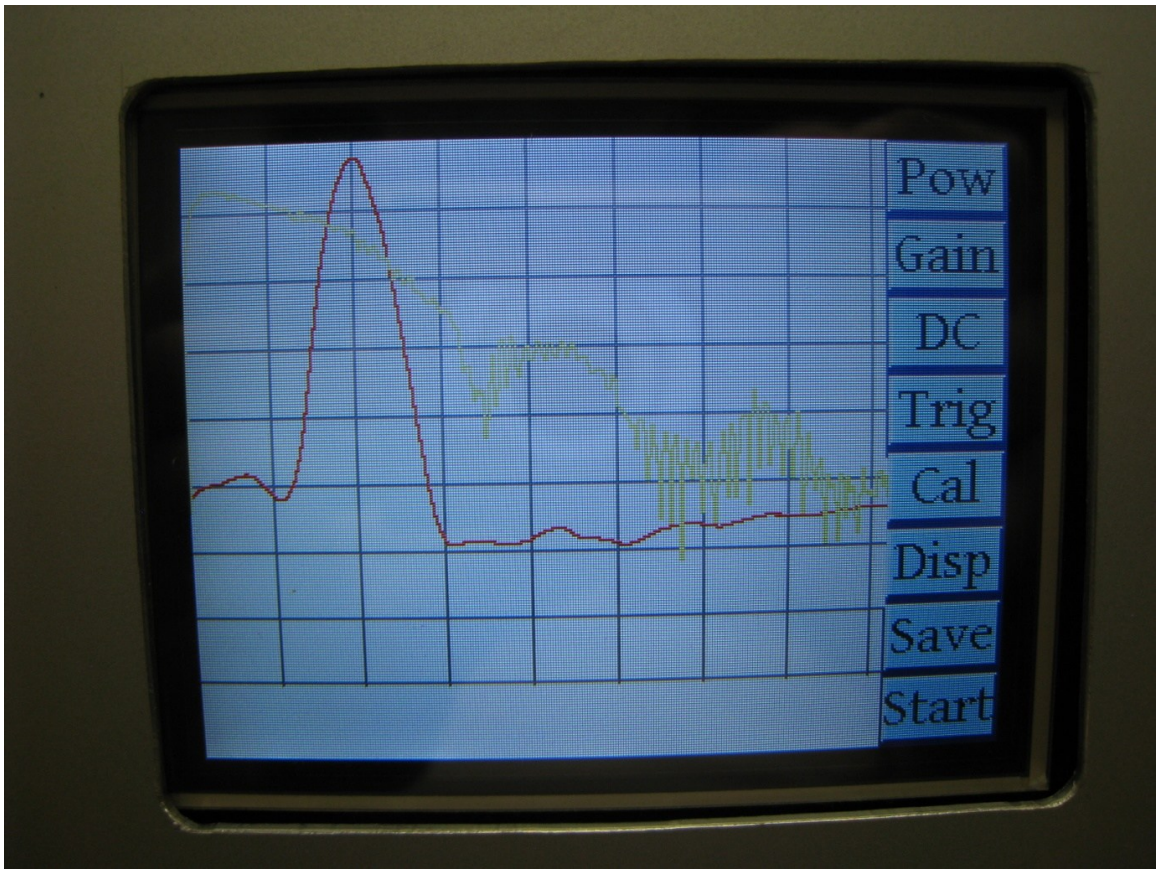

Fig. 58: PFTNA Full Featured User Interface

Data Collection

Data is stored onto a USB flash drive in comma-separated values (CSV) format for ease of access for further manipulation and post analysis. This choice of data storage implementation allows multiple units to be deployed offsite and researchers can then easily retrieve the data to analyze. Once a calibration algorithm has been written for a particular application, this data storage feature can be removed.

CHAPTER FIVE

Conclusions and Final Recommendations

Future Design Recommendations

For a future revision of the pulse transceiver, the power conditioning should be provided by the control board on a separate PCB. This design change will reduce the heat dissipated by the pulse transceiver PCB which will lower the diode temperature. As the diodes heat, they increase the conversion loss of the sub-sampling circuit, directly reducing its voltage output. The control board was designed for this possible change and includes power generation and condition for all of the voltages needed by the pulse transceiver PCB. The control board contains a separate header J20 specifically for this purpose.

To further alleviate the performance degradation from heating, a study of possible alternative diodes should be conducted. A preliminary search has yielded several possible alternatives: BAS70-04W.115, SMS7621-005LF, and BAS70-04W E6327. Each alternate diode should undergo full temperature sweep tests monitoring output voltage at several different frequencies.

The termination for the high-speed, pulse sharpening comparators used in the pulse transceiver board should be altered. The termination for the secondary port connected to the DAC on the control board should be left floating. This will change the input impedance from 50 ohms to 1.5k ohms. The impedance change will greatly reduce the current produced by the DAC's allowing them to reach their full voltage output. The change should also allow for smaller pulse widths to be supported, since high frequency

noise will be reflected and attenuated by the high impedance. The benefits of this change were verified on revision B, but could not be implemented on revision C due to the layout restrictions.

The equivalent time sampling clocks transmitted between the two PCBs in the PFTNA are single ended and are not shielded. To further reduce clock jitter they should be changed to differential signals with shielded cables. This change will minimize any common mode noise produced by EMI radiating from the digital circuits. The control board was designed with this change in mind and provides both differential ETS triggering signals with a separate connector J25. This connector is a common SATA connector, which has two individually shielded twisted pairs of wire.

Increased broadband performance could be obtained by using a comparator with faster rise times. In March of 2010 Hittite released the HMC675LC3C, which has rise times of 27 ps. This comparator shares the same package and pin layout as the current comparator. It could be fairly easily incorporated into the design in place of the Analog Devices ADCMP580 with only a change in voltage supplies. The -3 V supply needed by the Hittite device was included in the design of the control board to accommodate this improvement. However, because the comparator is driven with lower voltage supplies, its input will no longer be compatible with 3.3V PECL logic. The flip-flops would also have to be replaced.

An alternative flip-flop is the NB7V52M from ON Semiconductor. It supports a differential input signal that would be consistent with the previous recommendation of differential clocking signals. The NB7V52M also has a propagation delay reduction of 134 ps from the currently used flip-flops, which would further increase the bandwidth of

the system. The upgrade to the comparator and flip-flop should only be made to the pulse generation section of the transceiver. As an estimate, the performance increases should double the 10 dB bandwidth to well over 4 GHz. The costs increase due to these two integrated circuits is $35 in bulk quantities.

The dynamic range of the system is likely adversely affected by crosstalk from the pulse generator and pulse receiver sharing the same printed circuit board. Any possible increase in dynamic range is more difficult to test as it would involve fabricating two RF PCBs. Another Fourier transform network analyzer made this change and the dynamic range of that system was increased by 10dB.

The glitch generator could also be redesigned to shorten the duration of the pulse which would increase system bandwidth. One possible alternate design would use a two input AND gate with the pulse-sequence system clock as one input. An inverter on the clock would create a delay that could then be connected to the other input of the AND gate. The AND gate would then produce a glitch as wide as the propagation delay of the inverter.

Final Thoughts

The Pulsed Fourier Transform Network Analyzer documented in this thesis and shown in Fig. 59 provides more than 2.5 GHz of accurate measurement capability. The cost of the PFTNA is extremely low when compared with current commercial measurement systems. A fully embedded system with no user interface could cost under $80 with the fully featured version falling under $300. The low-costs allow for the development of a large range of consumer and industrial applications that were previously not economically viable for investigation. There are also several

recommended design changes that would further enhance the performance of the system. This Pulsed Fourier Transform Network Analyzer fully meets the design requirements of the current research applications and could be ready for commercialization within a year.

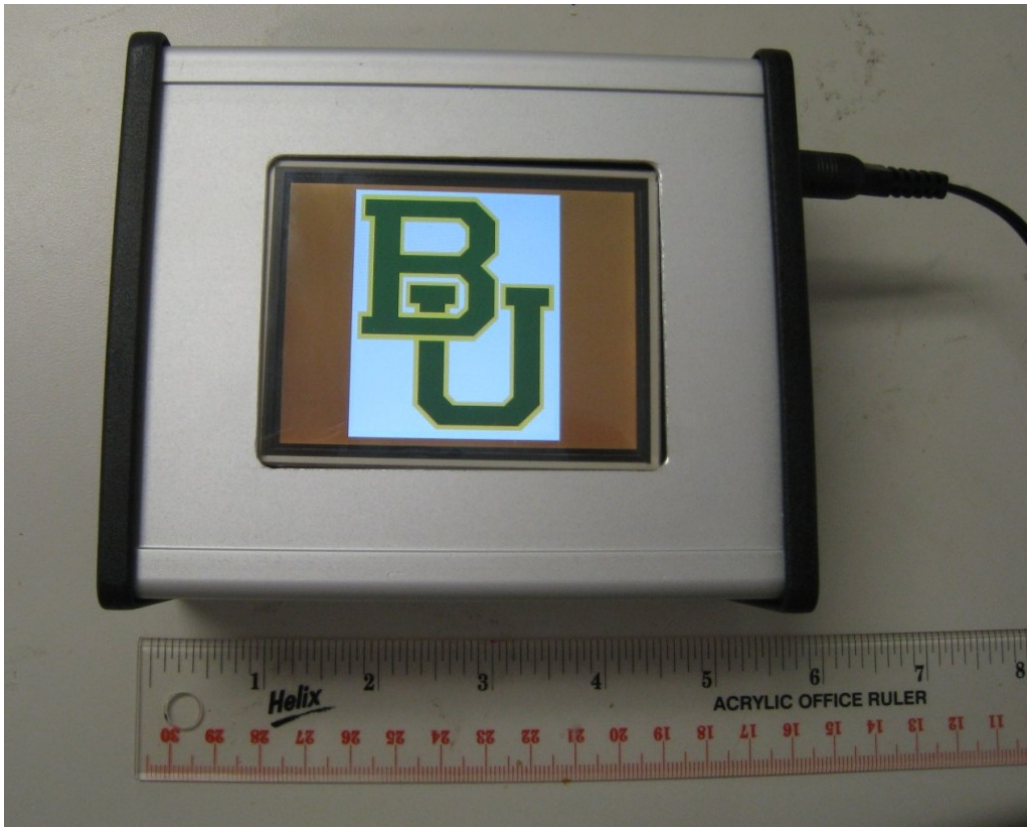

Fig. 59: Final PFTNA Prototype

APPENDICES

APPENDIX A

Circuit Schematics, Layouts, and BOM

Only the schematics, layouts and bill of materials for the pulse transceiver revision, control board, and the PIC32 Starter Kit are included in this appendix. The previous revisions, prototypes, and individual components of the transceiver that were manufactured for testing purposes have been left out.

Pulse Transceiver RF Revision C Printed Circuit Board

This is the high frequency PCB on Rogers RT/Duriod that produces the UWB pulse train and the equivalent time sampled received pulse.

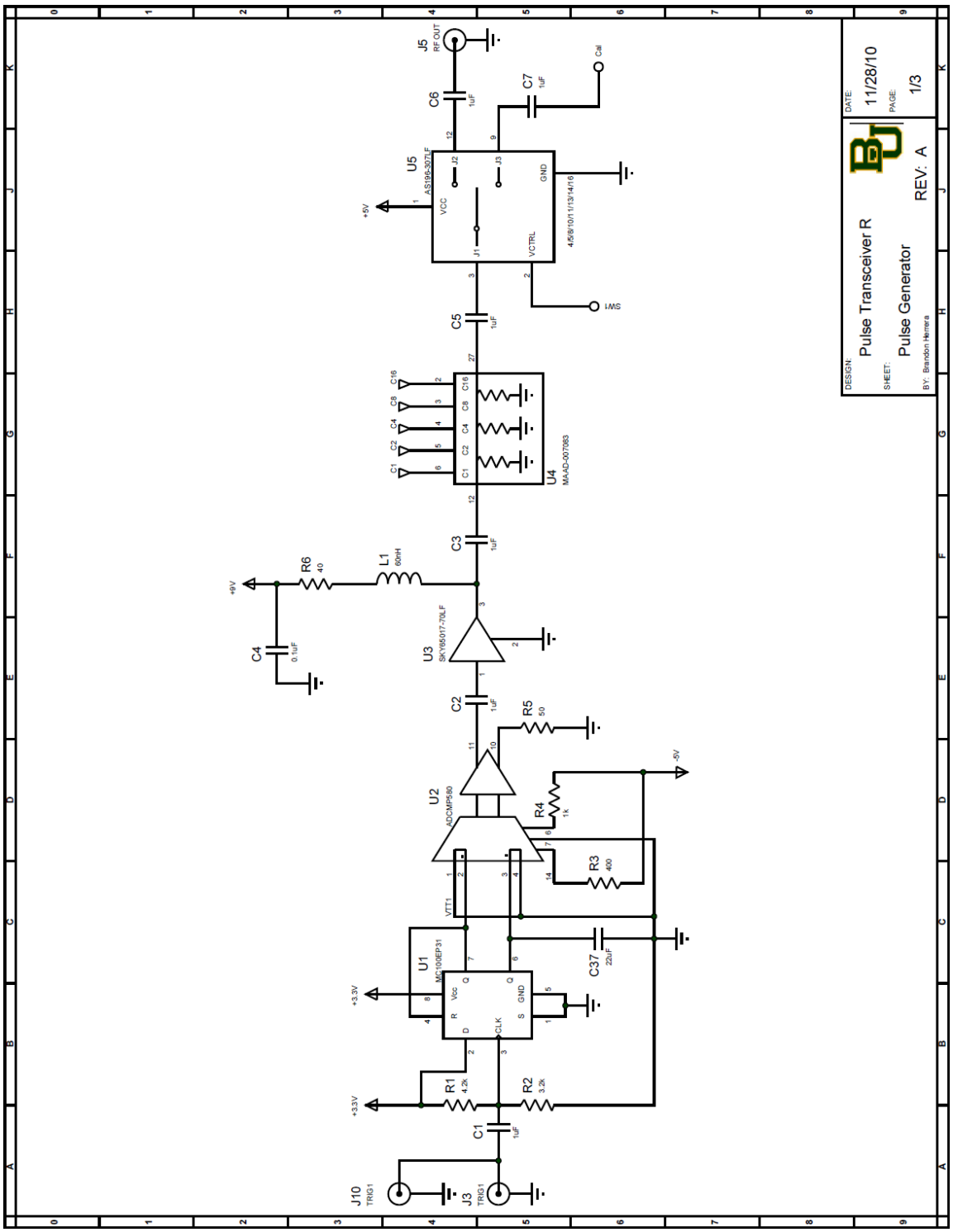

Fig. A.60: Schematic of the Pulse Transceiver Page 1

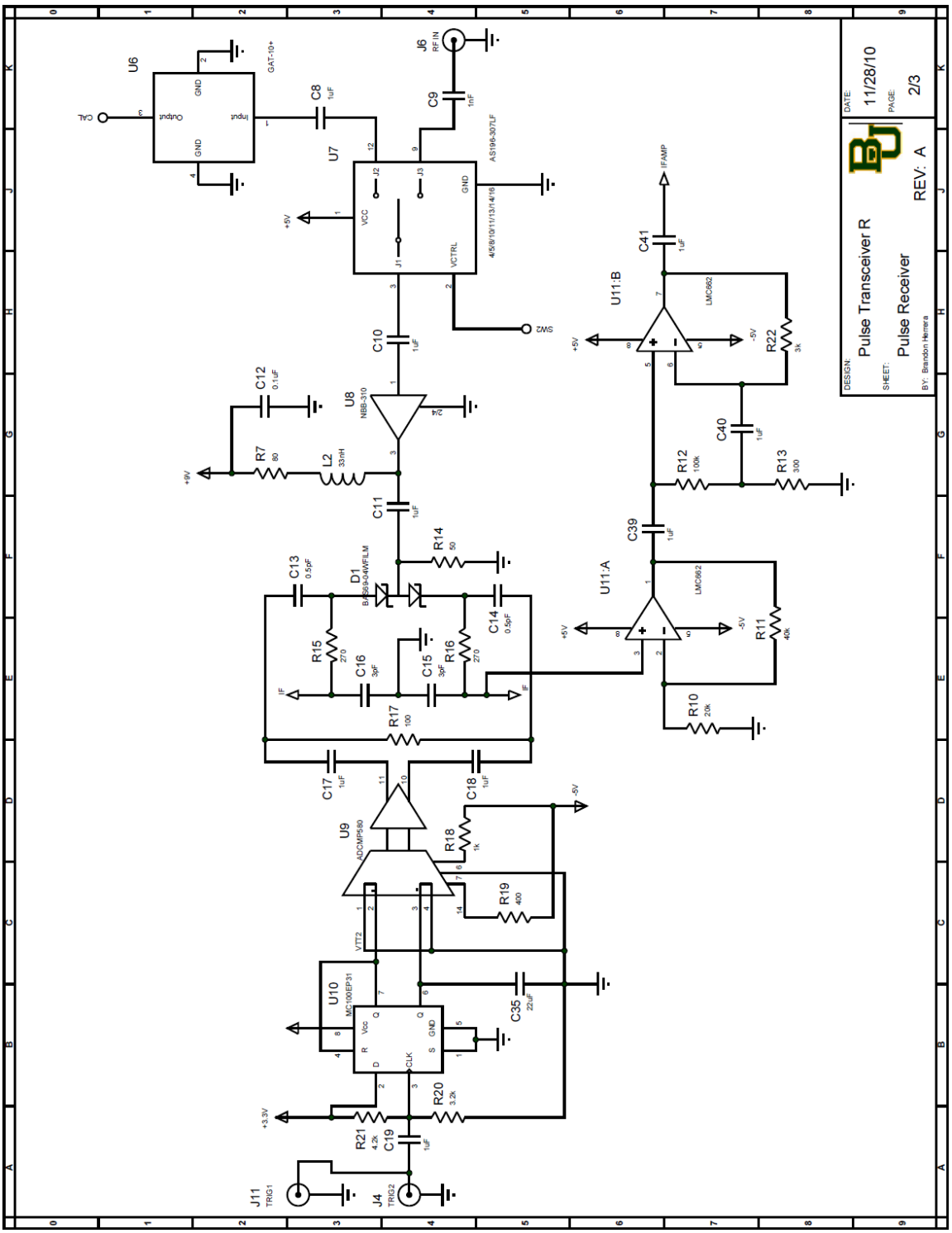

Fig. A.61: Schematic of the Pulse Transceiver Page 2

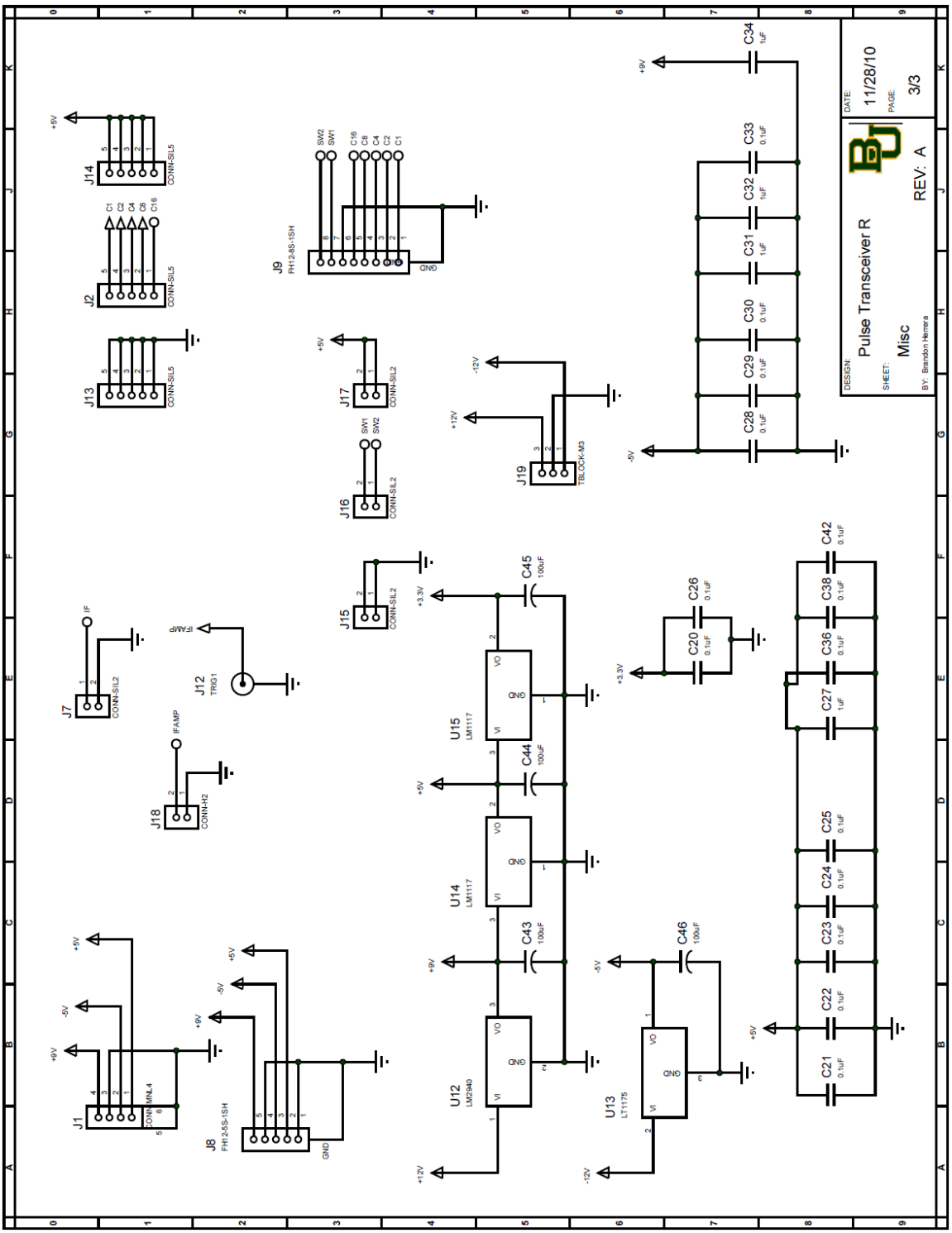

Fig. A.62: Schematic of the Pulse Transceiver Page 3

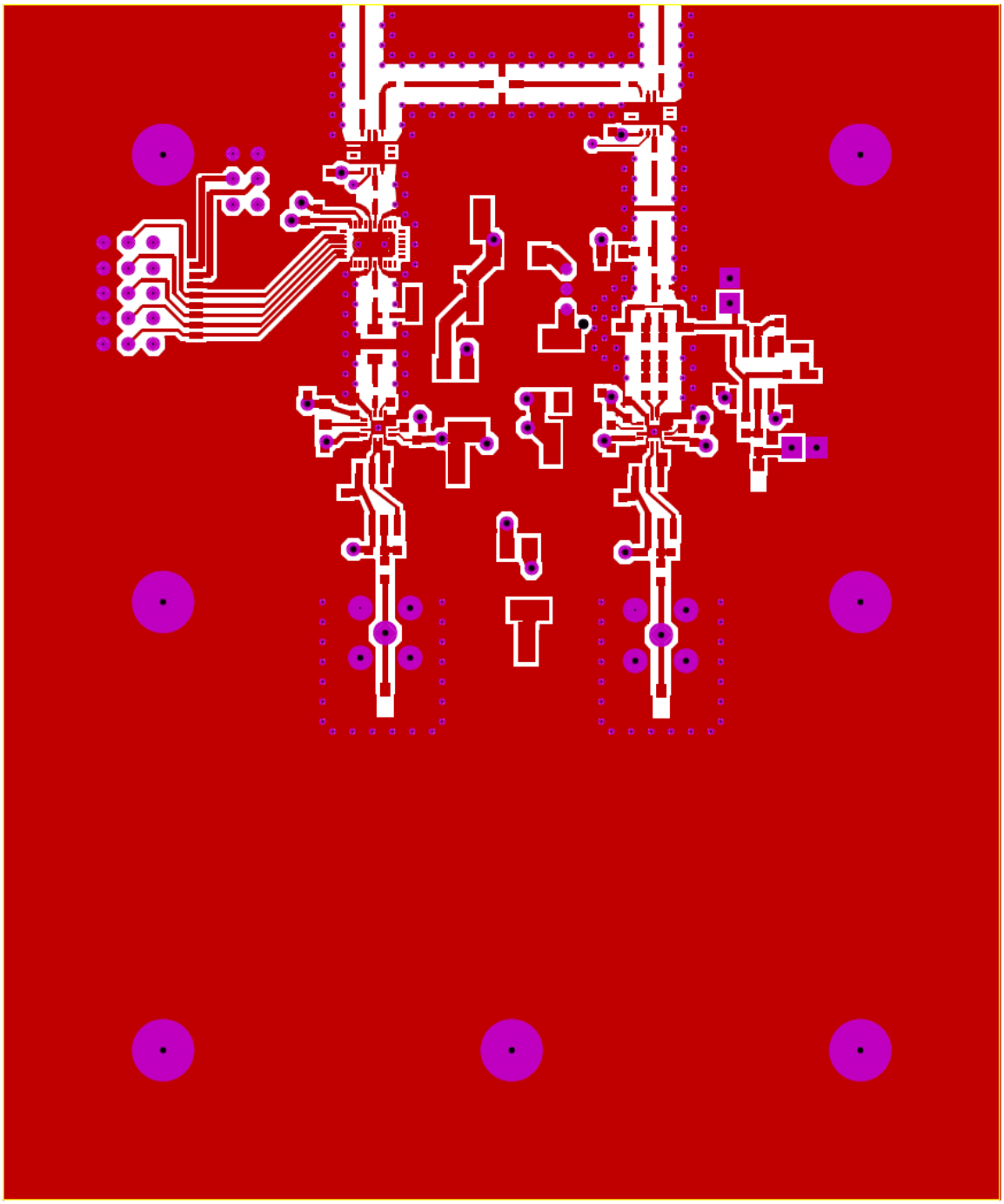

Fig. A.63: Layout of the Pulse Transceiver Top

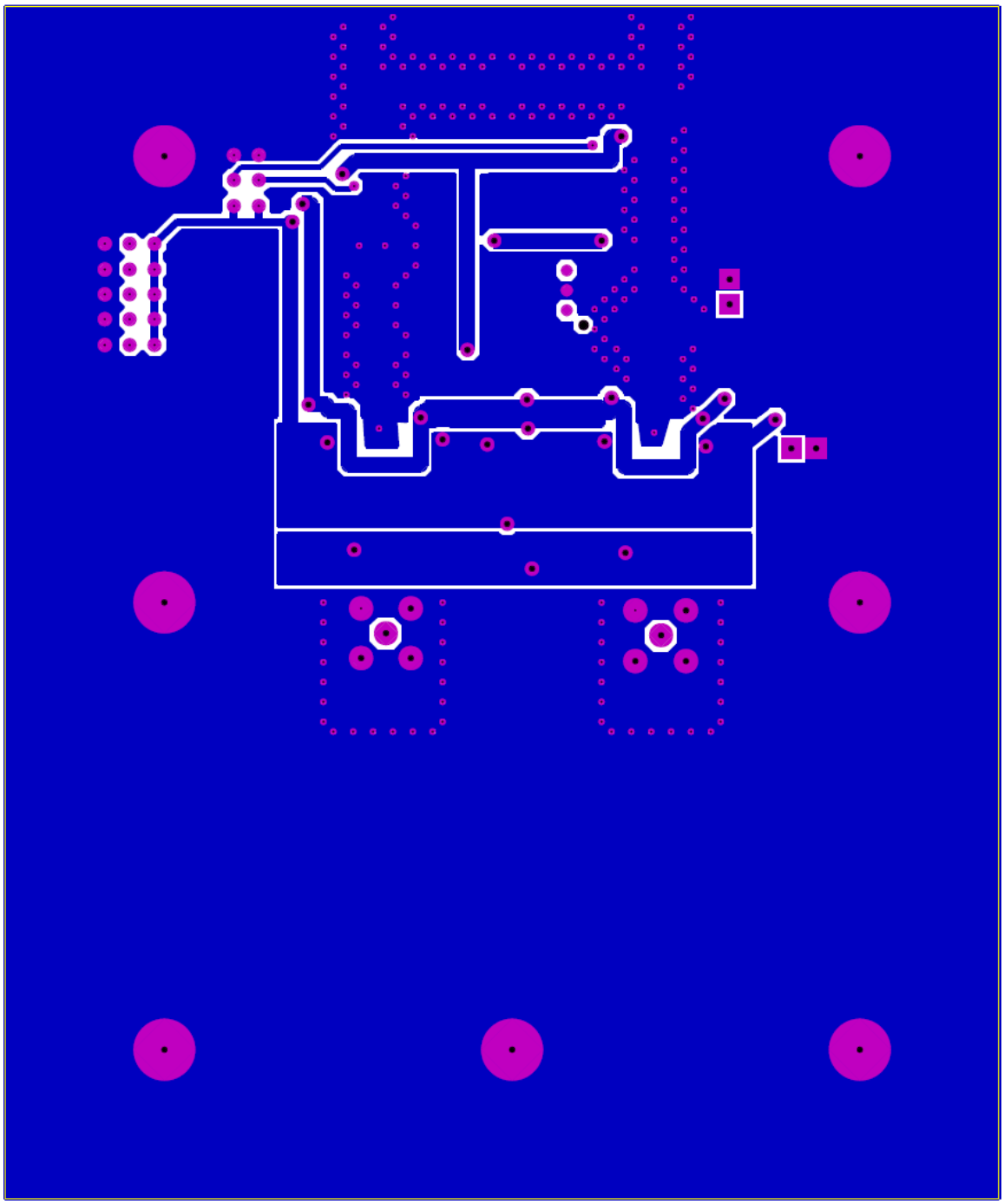

Fig. A.64: Layout of the Pulse Transceiver Bottom

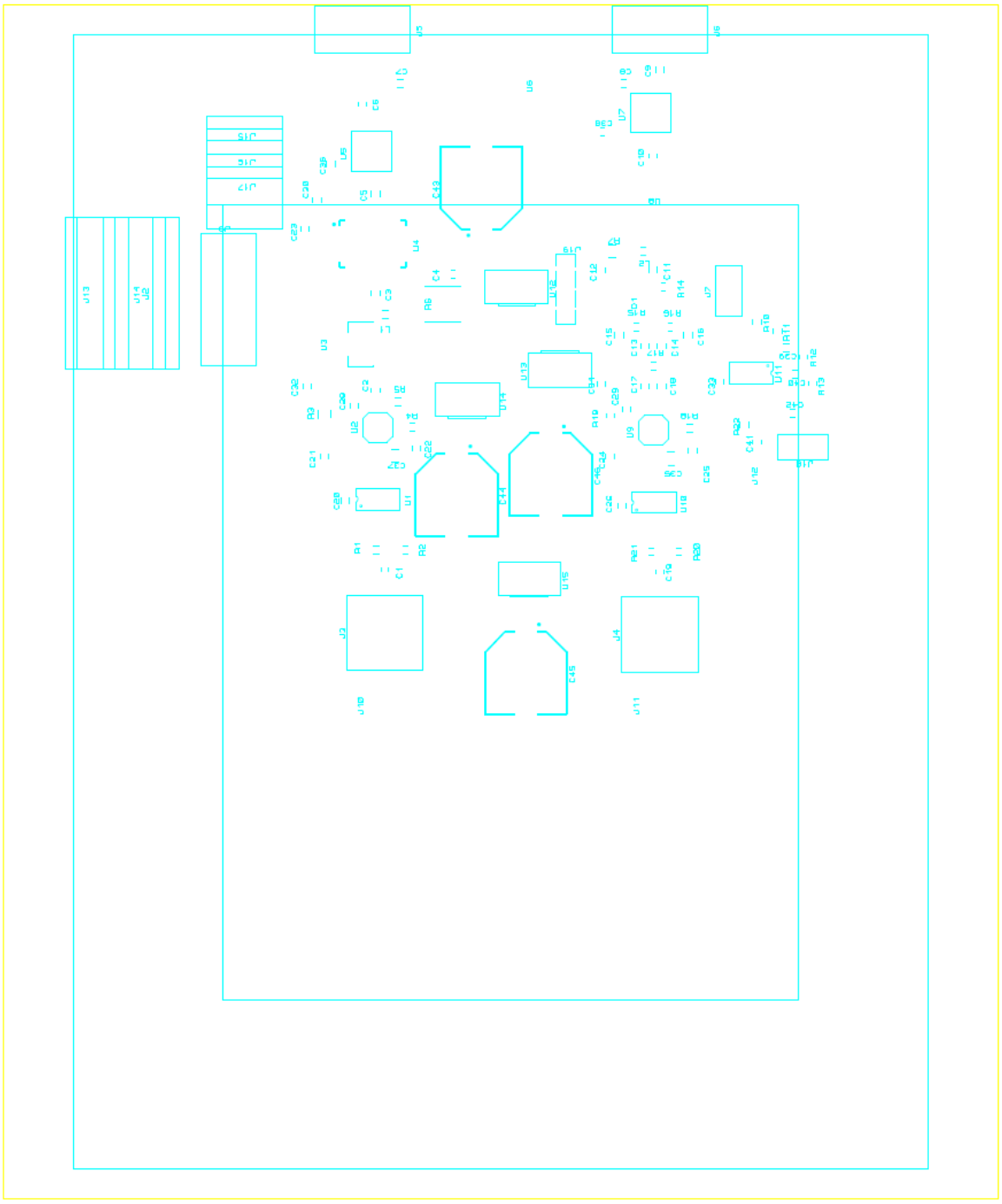

Fig. A.65: Layout of the Pulse Transceiver Silk Screen

Table 4: Pulse Transceiver Revision C: Bill of Materials

Reference	Value	Package	MFR	MFR P/N	Vendor	Vendor P/N	Proto Cost	Bulk Cost
R1,R21	86	603						
R2,R20	130	603						
R3	400	805						
R4,R18	1k	603						
R5	50	603						
R6	40	2512						
R7	80	805						
R10	20k	603						
R11	40k	603						
R12	100k	603						
R13	300	603						
R14	50	0603-23mil						
R15,R16	270	603						
R17	100	603						
R19	400	603						
R22	3k	603						
C1,C17-C19,C27,C31,C32,C34,C39-C41	1uF	603						
C2,C3,C5-C8,C10,C11	1uF	0603-23MIL						
C4,C12,C20-C26,C28-C30,C33,C36,C38,C42	0.1uF	603						
C9	1nF	0603-23MIL						
C13,C14	2.2pF	603						

Table 4 continued

Reference	Value	Package	MFR	MFR P/N	Vendor	Vendor P/N	Proto Cost	Bulk Cost
C15,C16	6pF	603						
C35,C37	22uF	805						
C43-C46	100uF	CAPAE830X620						
U1,U10	MC100EP31	SO8	On Semiconductor	MC100EP31DG	Mouser	863-MC100EP31DG	$9.68	$5.80
U2,U9	ADCMP580	16 PIN LFCSP_VQ	Analog Devices	MC100EP31DG	Digikey	ADCMP580BCZ-WP-ND	$20.68	$11.68
U3	SKY65017-70LF	SOT89	TriQuint Semiconductor	SKY65017-70LF	Digikey	863-1063-1-ND	$1.63	$1.30
U4	MAAD-007083	PQFN50P400X600X80-32	MACOM					
U5,U7	AS196-307LF	LPCC 4X4	Skyworks	AS196-307LF	Digikey	863-1007-1-ND	$2.58	$1.67
U6	GAT-10+		Mini-Circuits					
U8	NBB-310	MICRO-X	RFMD					
U11	LMC662	SO8	National Semiconductor	LMC662AIMX/NOPB	Digikey	LMC662AIMX/NOPBCT-ND	$2.38	$1.08
U12	LM2940	SOT223-3	National Semiconductor					
U13	LT1175	SOT223-3	Linear Technology					
U14,U15	LM1117	SOT223-3	National Semiconductor					
D1	HSMS-286C-TR1G	SOT-323-D3	STMicroelectronics	HSMS-286C-TR1G	Digikey	516-1822-1-ND	$1.44	$0.62
J1	CONN-MNL4	4POSM-N-L					$0.99	
J2,J13,J14	CONN-SIL5	SIL-100-05						
J3	TRIG1	SMA-TH						
J4	TRIG2	SMA-TH						
J5	RF OUT	SMA-EDGE-23MIL	Emerson Network Power Connectivity Solutions	142-0701-871	Digikey	J610-ND	$7.68	
J6	RF IN	SMA-EDGE-23MIL	Emerson Network Power Connectivity Solutions	142-0701-871	Digikey	J610-ND	$7.68	$3.48

Table 4 continued

Reference	Value	Package	MFR	MFR P/N	Vendor	Vendor P/N	Proto Cost	Bulk Cost
J7	CONN-SIL2	CONN-SIL2						
J8	FH12-5S-1SH	FH12-5S-1SH						
J9	FH12-8S-1SH	FH12-8S-1SH						
J10,J11	TRIG1	UMC	Emerson Network Power Connectivity Solutions	128-0711-201	Digikey	J983CT-ND	$0.93	$0.44
J12	TRIG1	UMC						
J15-J17	CONN-SIL2	SIL-100-02						
J18	CONN-H2	CONN-SIL2						
J19	TBLOCK-M3	B03B						
L1	60nH	0603-23MIL						
L2	33nH	603						

Control Board Printed Circuit Board

This is the 4 layer control board that provides the DSP, power generation, user interface, signal conditioning, and equivalent time sampling triggering signals.

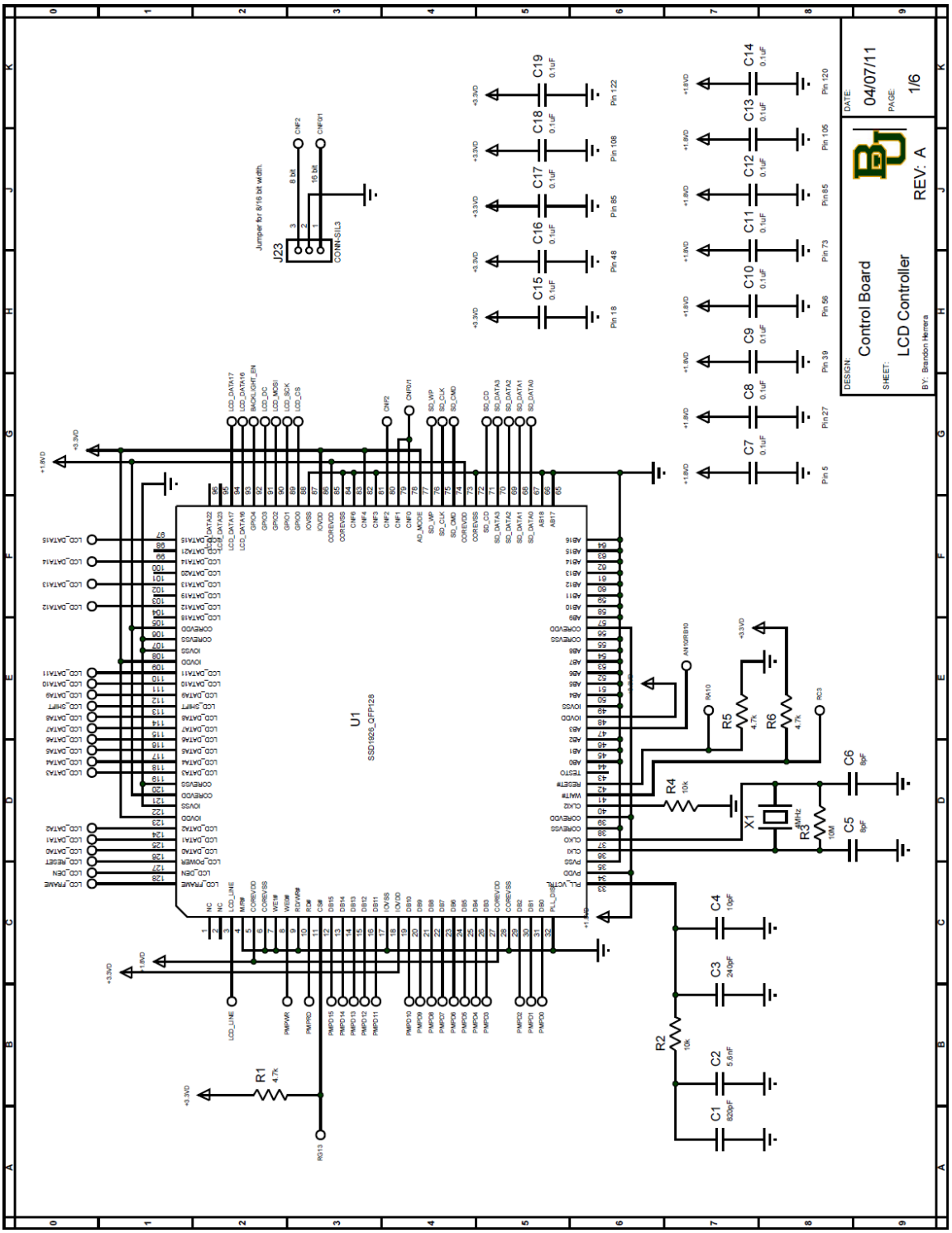

Fig. A.66: Schematic of the Control Board Page 1

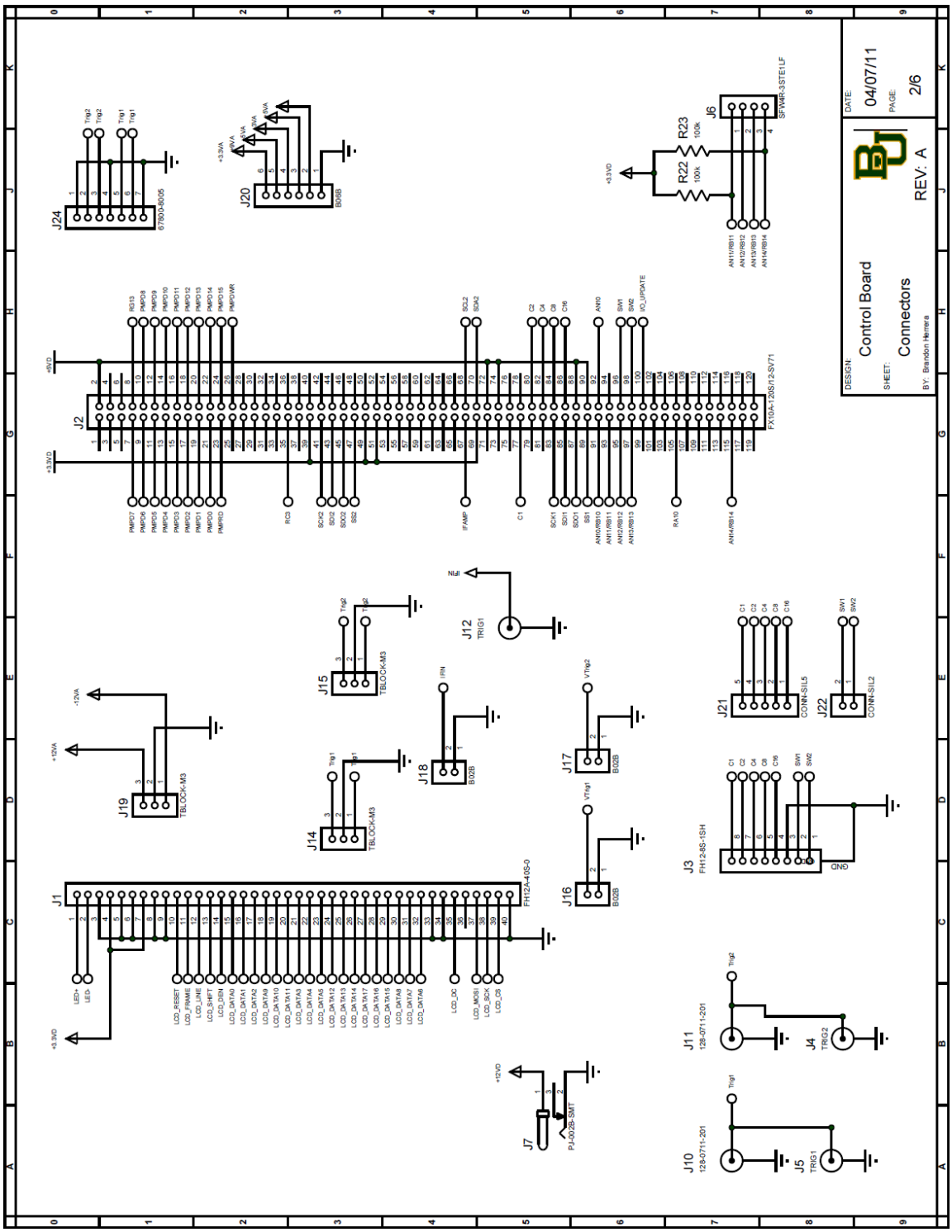

Fig. A.67: Schematic of the Control Board Page 2

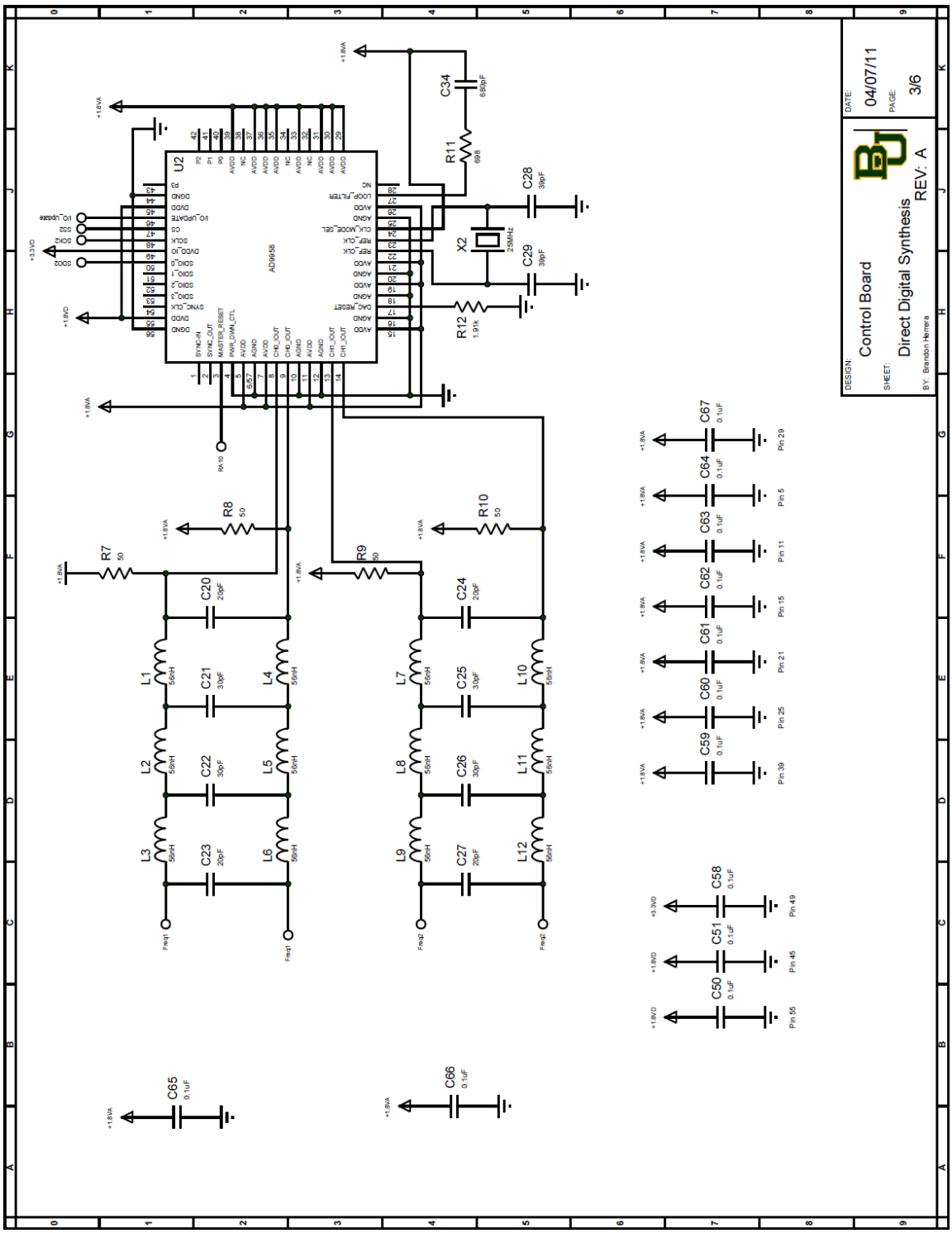

Fig. A.68: Schematic of the Control Board Page 3

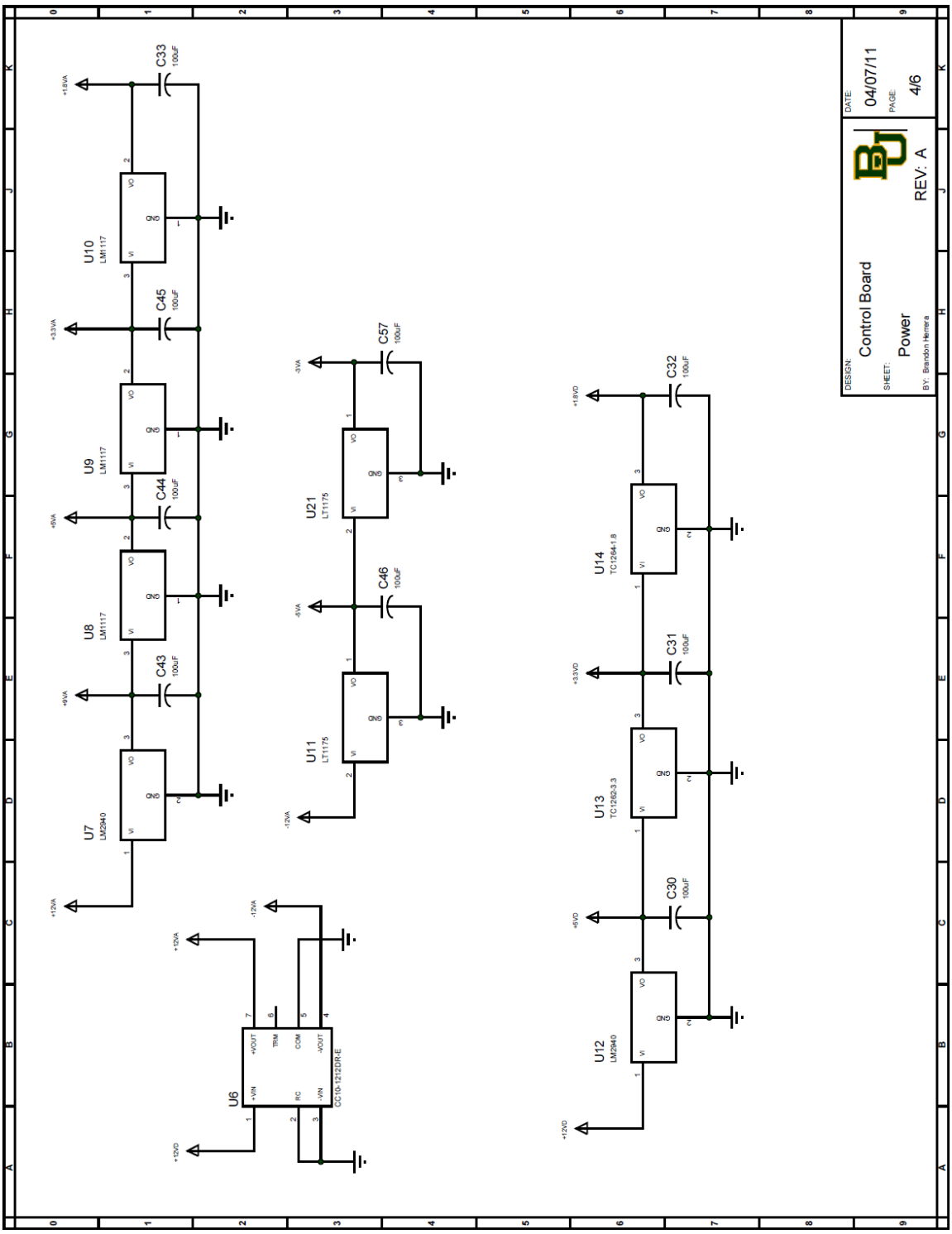

Fig. A.69: Schematic of the Control Board Page 4

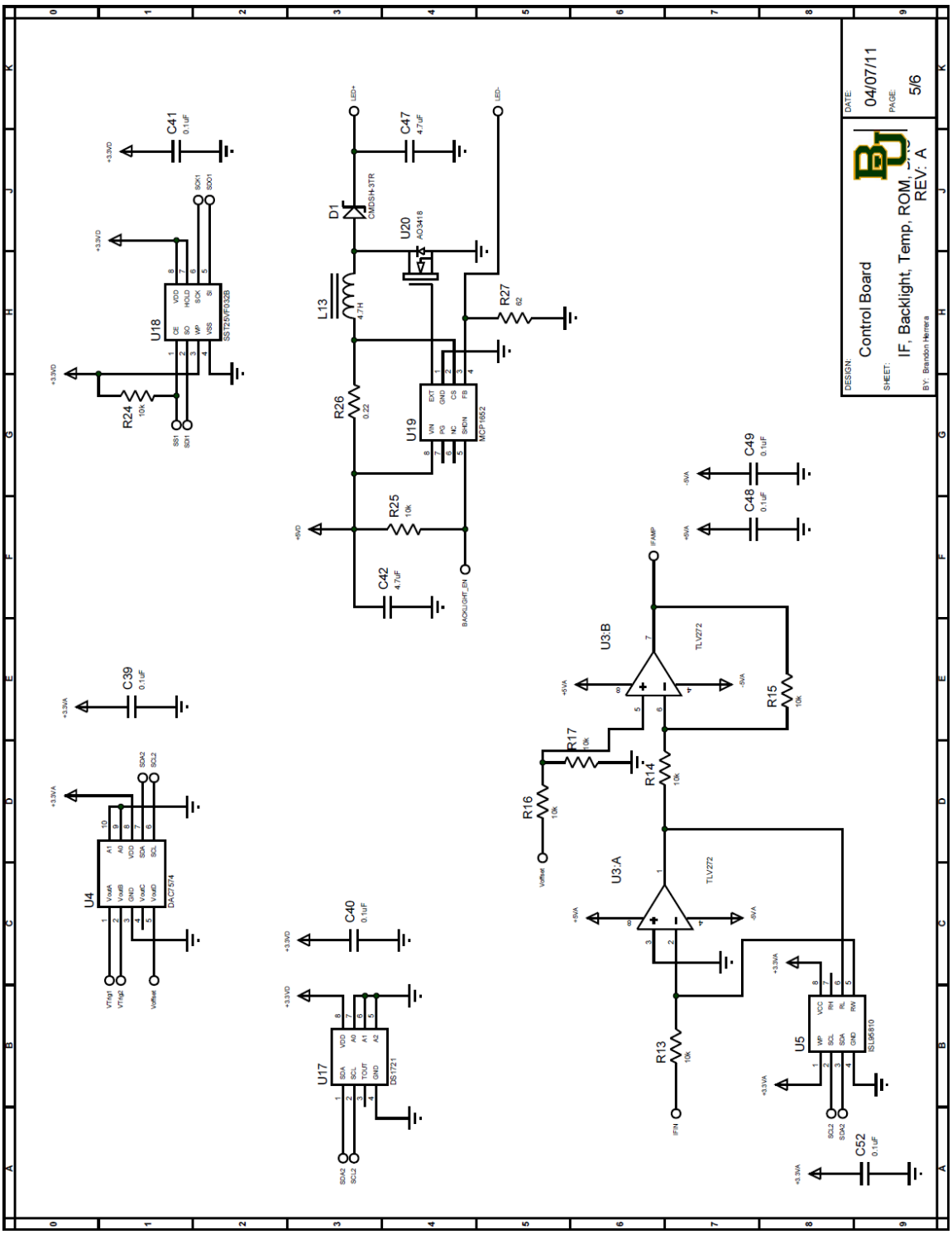

Fig. A.70: Schematic of the Control Board Page 5

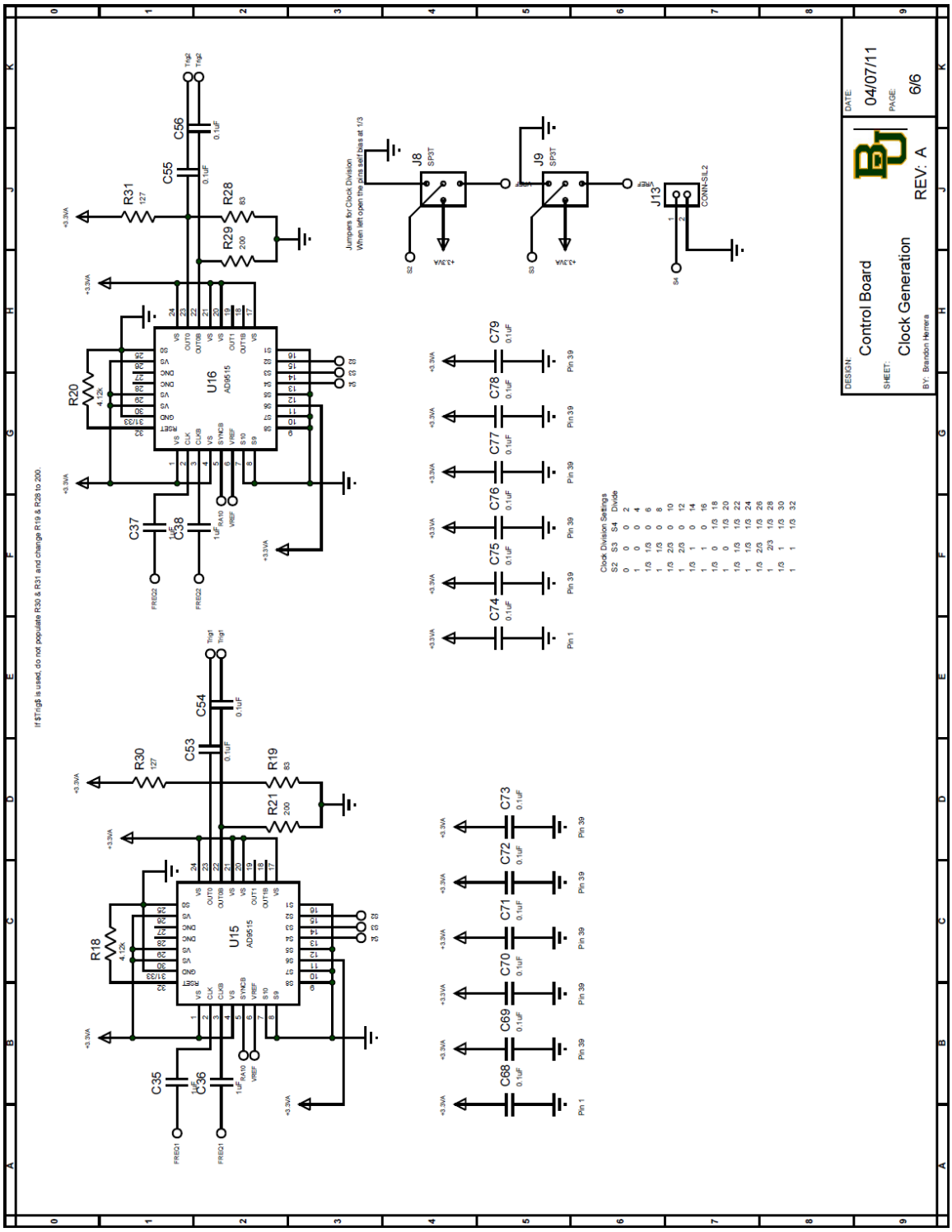

Fig. A.71: Schematic of the Control Board Page 6

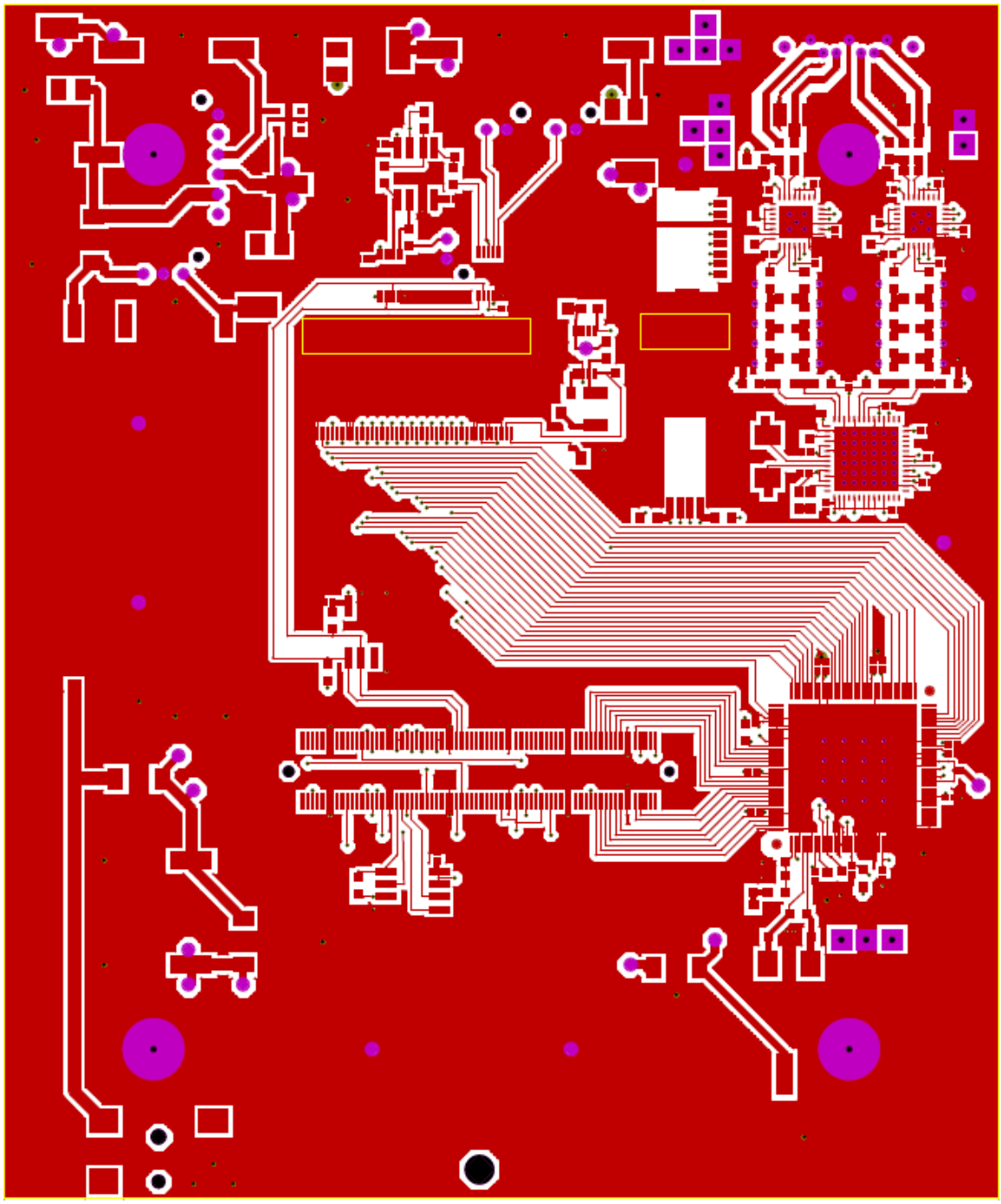

Fig. A.72: Layout of the Control Board Top

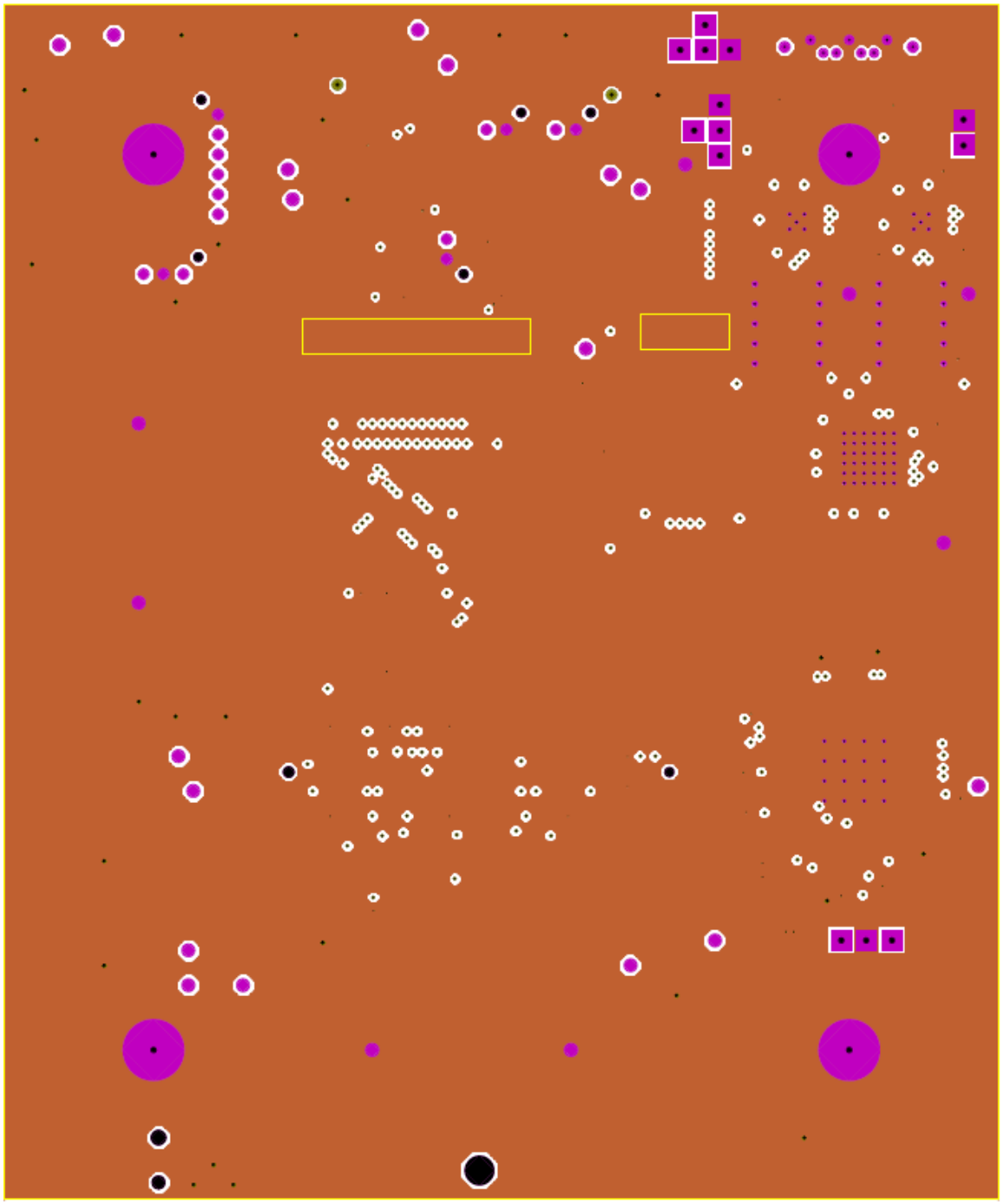

Fig. A.73: Layout of the Control Board Inner 1

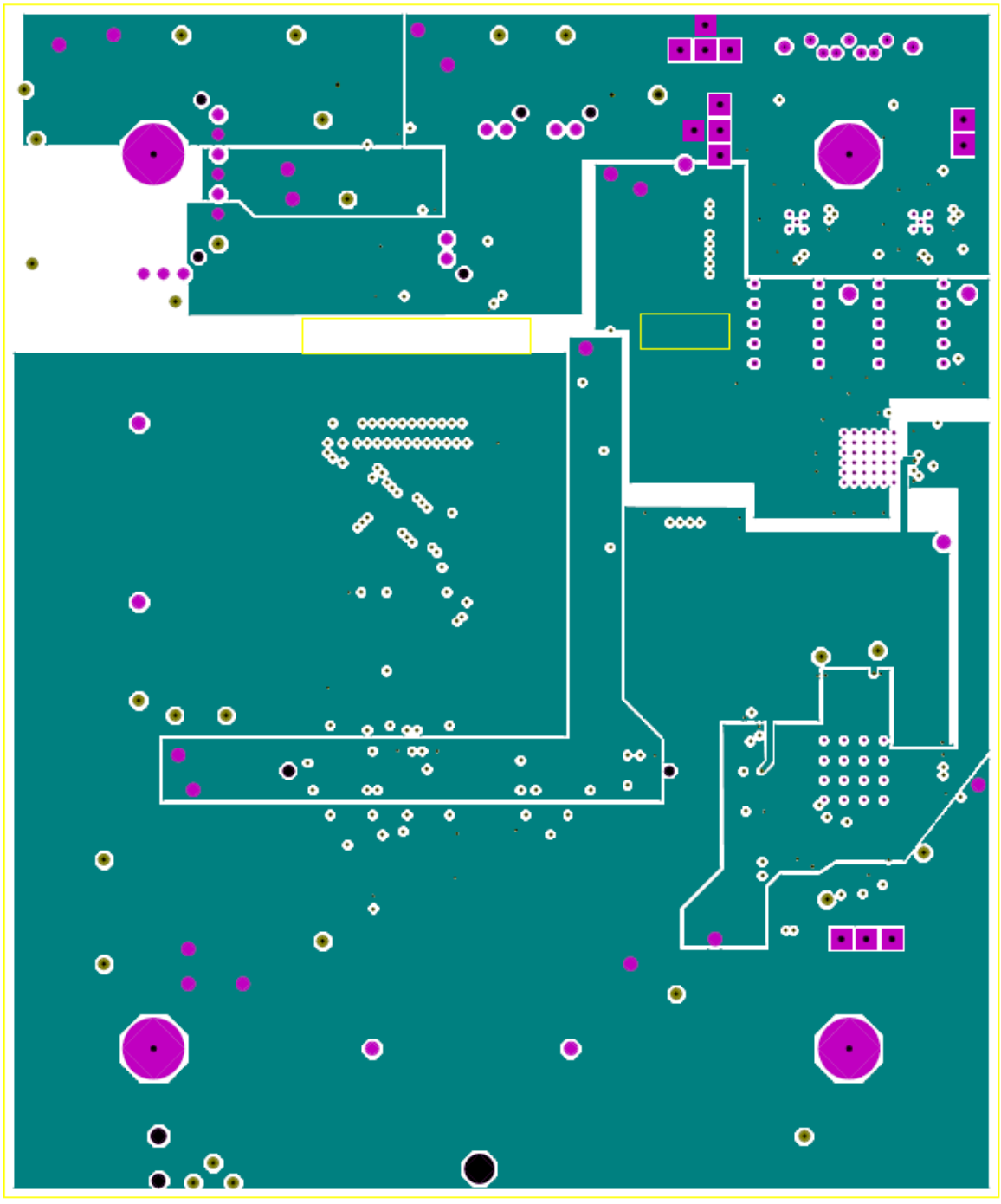

Fig. A.74: Layout of the Control Board Inner 2

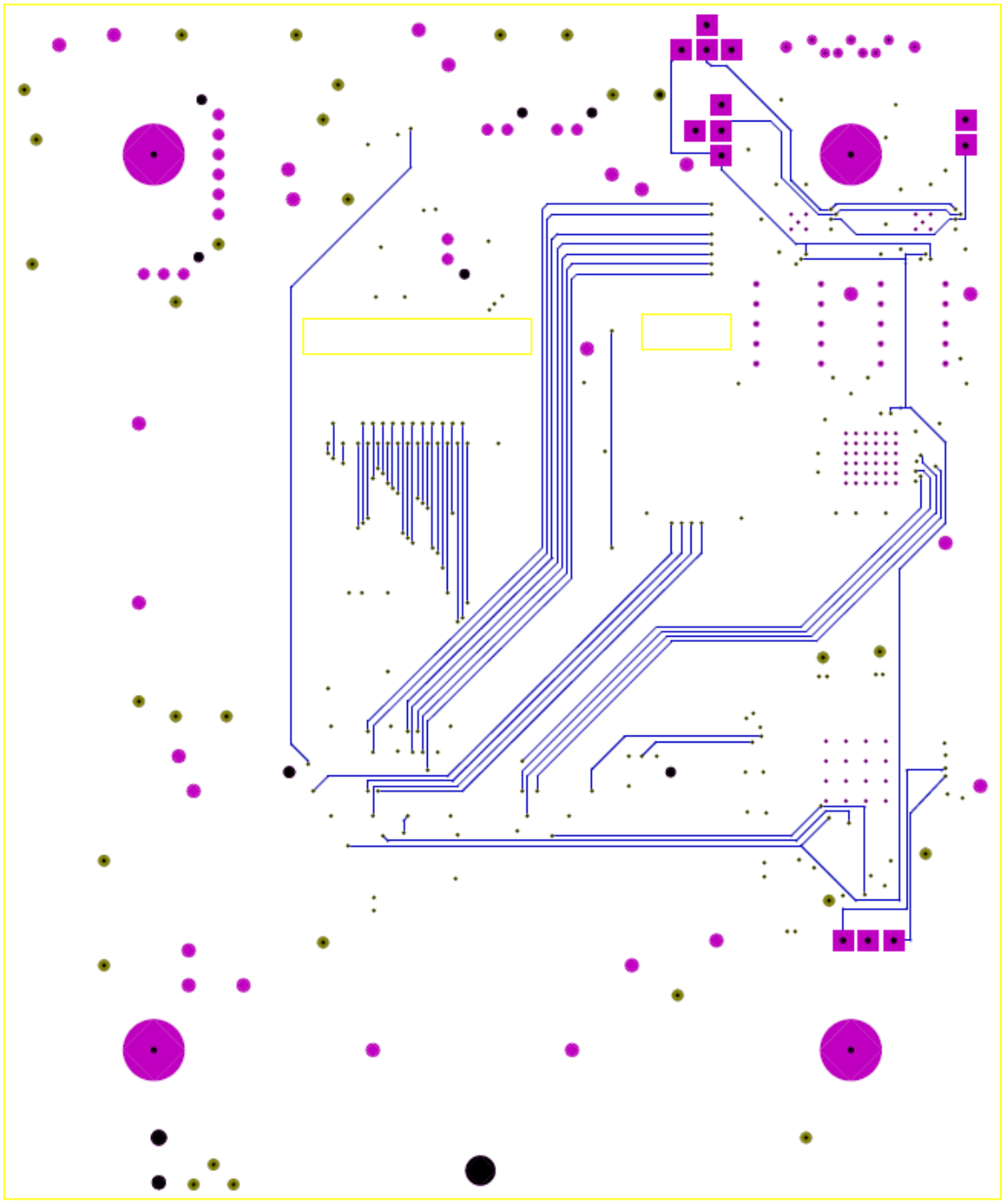

Fig. A.75: Layout of the Control Board Bottom

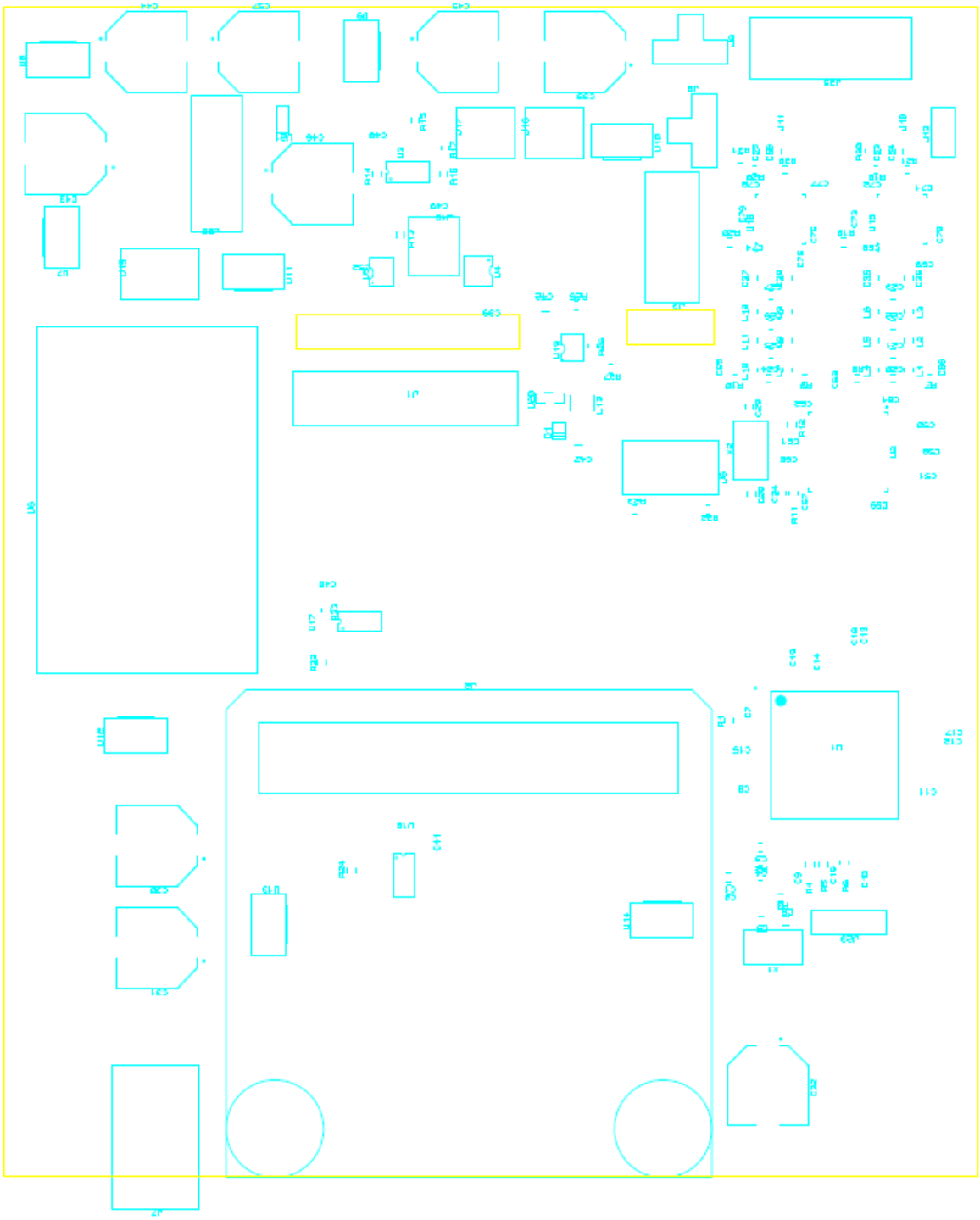

Fig. A.76: Layout of the Control Board Top Silk Screen

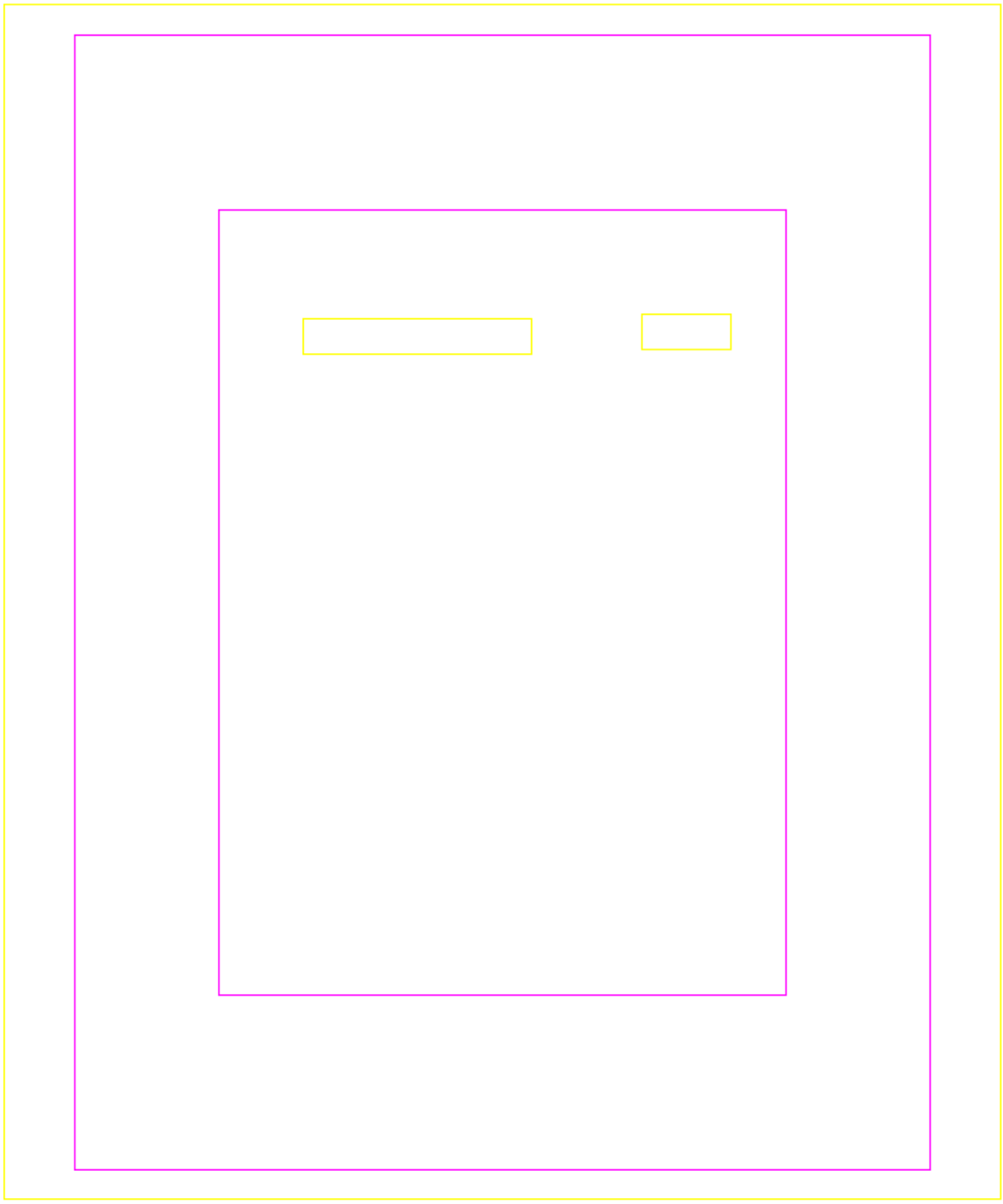

Fig. A.77: Layout of the Control Board Bottom Silk Screen

Table 5: Control Board: Bill of Materials

Reference	Value	Package	MFR	MFR P/N	Vendor	Vendor P/N	Proto Cost	Bulk Cost
R1,R5,R6	4.7k	603	Panasonic	ERJ-3GEYJ472V	Digikey	P4.7KGCT-ND	$0.03	$0.00
R2,R4,R13-R17,R24,R25	10k	603	Panasonic	ERJ-3EKF1002V	Digikey	P10.0KHCT-ND	$0.04	$0.00
R3	10M	603						
R7-R10	50	603						
R11	698	603						
R12	1.91k	603						
R18,R20	4.12k	603						
R19,R28	83	603						
R21,R29	200	603						
R22,R23	100k	603	Panasonic	ERJ-3EKF1003V	Digikey	P100KHCT-ND	$0.04	$0.00
R26	0.22	603						
R27	62	603						
R30,R31	127	603						
R32,R33	1k	603						
C1	820pF	603	Murata	GRM188R72A821KA01D	Digikey	490-1471-1-ND	$0.26	$0.02
C2	5.6nF	603	Murata	GRM188R71H562KA01D	Digikey	490-1507-1-ND	$0.24	$0.02
C3	240pF	603	Murata	GRM1885C1H241JA01D	Digikey	490-1436-1-ND	$0.25	$0.03
C4	10pF	603	Murata	GRM1885C2A100RA01D	Digikey	490-3279-1-ND	$0.19	$0.02
C5,C6	8pF	603	Murata	GRM1885C1H8R0DZ01D	Digikey	490-1399-1-ND	$0.14	$0.01

Table 5 continued

Reference	Value	Package	MFR	MFR P/N	Vendor	Vendor P/N	Proto Cost
C7-C19,C39-C41,C48-C52,C58-C79	0.1uF	CAPC1005X55					
C20,C23,C24,C27	20pF	603	Murata	GRM1885C1H200JA01D	Digikey	490-1410-1-ND	$0.18	$0.01
C21,C22,C25,C26	30pF	603	Murata	GRM1885C1H300JA01D	Digikey	490-1414-1-ND	$0.21	$0.02
C28	39pF	603					
C29	39pF	603	Murata	GRM1885C1H390JA01D	Digikey	490-1417-1-ND	$0.14	$0.01
C30-C33,C43-C46,C57	100uF	CAPAE830X620					
C34	680pF	603					
C35-C38	1uF	603					
C42,C47	4.7uF	805					
C53-C56	0.1uF	603					
U1	SSD1926_QFP128	QFP40P1600X1600X160-128	Solomon Systech	SSD1926	Microchip Direct	IC00409	$12.48	$12.48
U2	AD9958	QFN50P800X800X90-57	Analog Devices	AD9958BCPZ	Digikey	AD9958BCPZ-ND	$31.44	$25.89
U3	TLV272	SO8	Texas Instruments	TLV272IDR	Digikey	296-10655-1-ND	$1.15	$0.50
U4	DAC7574	MSOP10	Texas Instruments	DAC7574IDGSR	Digikey	296-15227-1-ND	$14.15	$8.61
U5	ISL95810	TSSOP8	Intersil	ISL95810UIU8Z-T	Digikey	ISL95810UIU8Z-TCT-ND	$3.39	$1.72

Table 5 continued

Reference	Value	Package	MFR	MFR P/N	Vendor	Vendor P/N	Proto Cost	Reference
U6	CC10-1212DR-E	CC10-1212DR	TDK-Lambda	CC10-1212DR-E	Digikey	445-3329-ND	$24.72	$16.44
U7,U12	LM2940	SOT223-3	National Semiconductor					
U8-U10	LM1117	SOT223-3	National Semiconductor					
U11	LT1175	SOT223-3	Linear Technology					
U13	TC1262-3.3	SOT223-3						
U14	TC1264-1.8	SOT223-3						
U15,U16	AD9515	QFN50P500X500X80-33B	Analog Devices	AD9515BCPZ	Digikey	AD9515BCPZ-ND	$9.86	$6.16
U17	DS1721	SO8	Maxim	DS1721U+	Digikey	DS1721U+-ND	$4.60	$2.23
U18	SST25VF032B	SO8	Microchip	SST25VF032B-80-4I-S2AF	Digikey	SST25VF032B-80-4I-S2AF-ND	$2.45	$1.69
U19	MCP1652	TSSOP8	Microchip	MCP1652R-E/MS	Digikey	MCP1652R-E/MS-ND	$1.50	$1.04
U20	AO3418	SOT23-N	Alpha & Omega	AO3418	Digikey	785-1012-1-ND	$0.54	$0.11
U21	MIC5270	SOT23-5						
D1	CMDSH-3TR	SOD-323						
J1	FH12A-40S-0	FH12A-40S-0.5SH(55)	Hirose Electric	FH12A-40S-0.5SH(55)	Digikey	HFK140CT-ND	$3.07	$1.91
J2	FX10A-120S/12-SV71	FX10A-120S/12-SV(71)	Hirose Electric	FX10A-120S/12-SV(71)	Digikey	H11234-ND	$7.22	$4.48
J3	FH12-8S-1SH	FH12-8S-1SH						
J4	TRIG2	SMA-TH						
J5	TRIG1	SMA-TH						
J6	SFW4R-3STE1LF	SFW4R-3STE1LF	FCI	SFW4R-3STE1LF	Digikey	609-1885-1-ND	$1.00	$0.34
J7	PJ-002B-SMT	PJ-002B-SMT	CUI	PJ-002B-SMT	Digikey	CP-002BPJCT-ND	$1.16	$0.72

Table 5 continued

Reference	Value	Package	MFR	MFR P/N	Vendor	Vendor P/N	Proto Cost	Reference
J8,J9	SP3T	SP3T						
J10,J11	128-0711-201	UMC	Emerson	128-0711-201	Digikey	J983CT-ND	$0.94	$0.43
J12	TRIG1	UMC						
J13,J22	CONN-SIL2	CONN-SIL2						
J14	DIL28	TRANS 28 DIL						
J16-J18	B02B	B02B						
J19	TBLOCK-M3	B03B						
J20	B06B	B06B						
J21	CONN-SIL5	CONN-SIL5						
J23	CONN-SIL3	CONN-SIL3						
J24	67800-8005	SATA	Molex				$0.57	
J25	CONN-SIL7	SATA						
L1-L12	56nH	603	Murata	LQW18AN56NJ00D	Digikey	490-1177-1-ND	$0.28	$0.15
L13	4.7H	1210	TDK	FLF3215T-4R7M	Digikey	445-4846-1-ND	$0.60	$0.25
X1	4MHz	ABM7	NDK	NX8045GB 4MHZ AT-W	Digikey	644-1138-1-ND	$1.50	$0.76
X2	25MHz	ABM7						

PIC32 USB Starter Kit II

This is the debugger microcontroller board for the PIC32. It is manufactured by Microchip and screws into the control board.

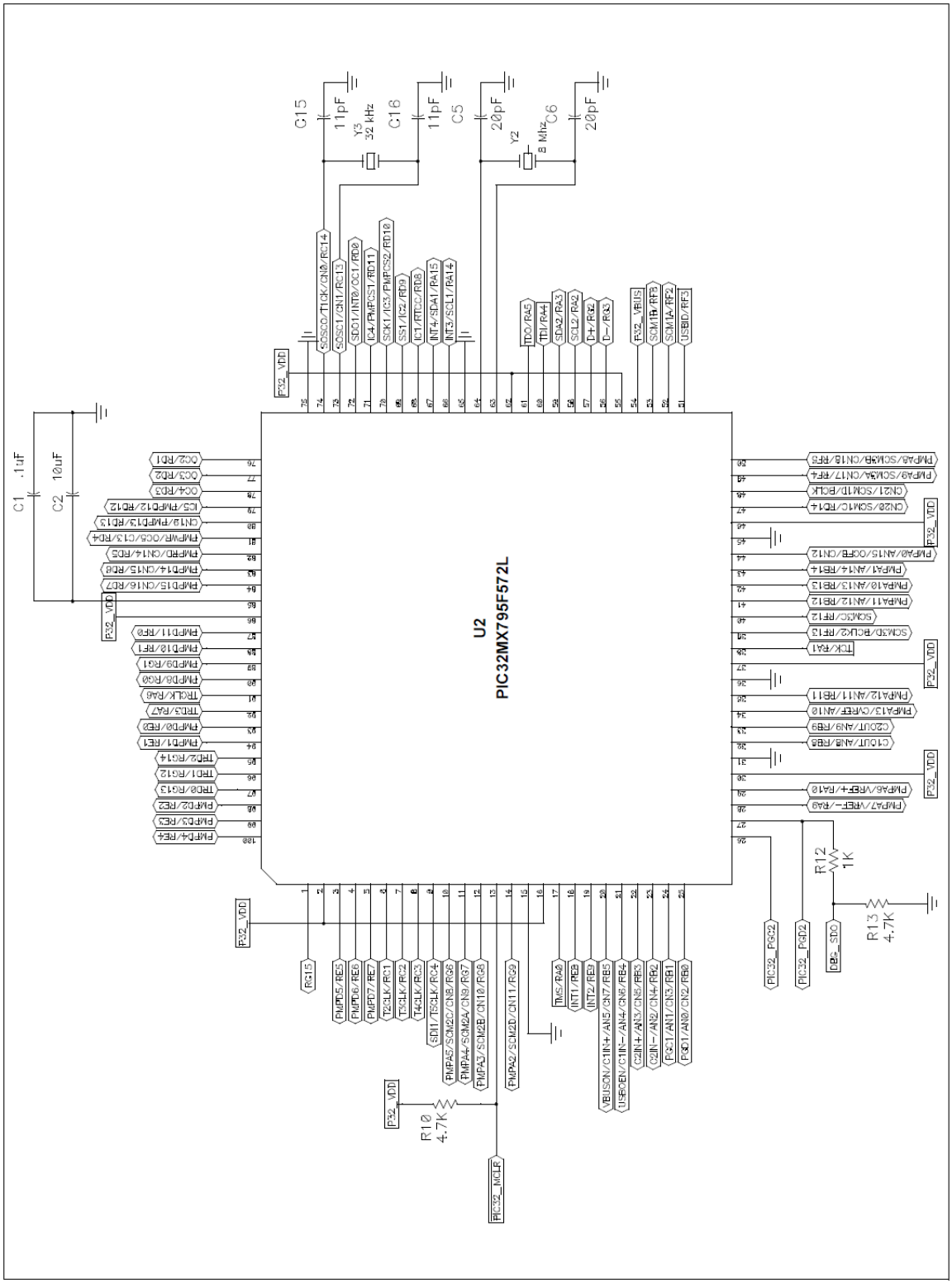

Fig. A.78: PIC32 USB Starter Kit II Schematic Page 1

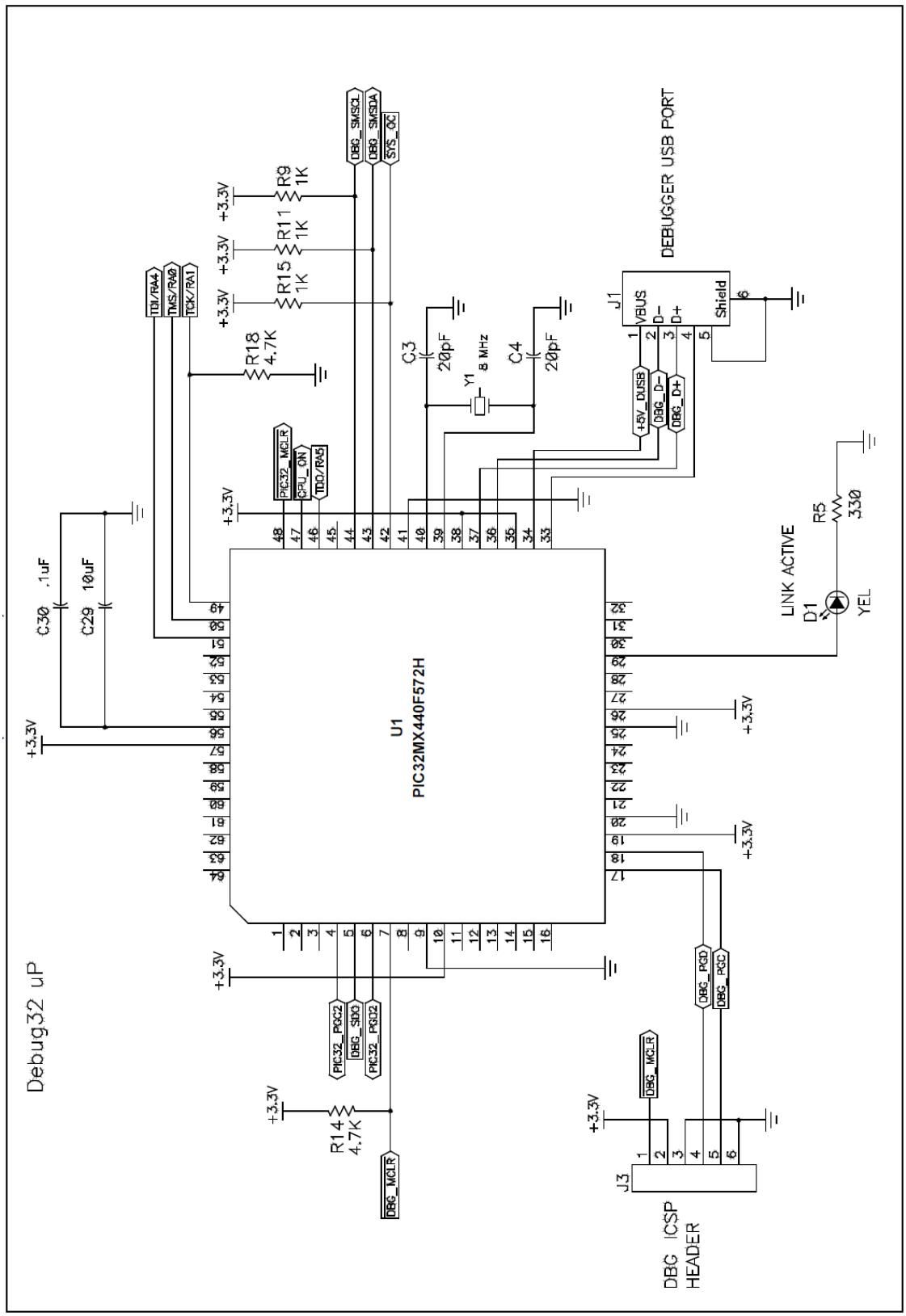

Fig. A.79: PIC32 USB Starter Kit II Schematic Page 2

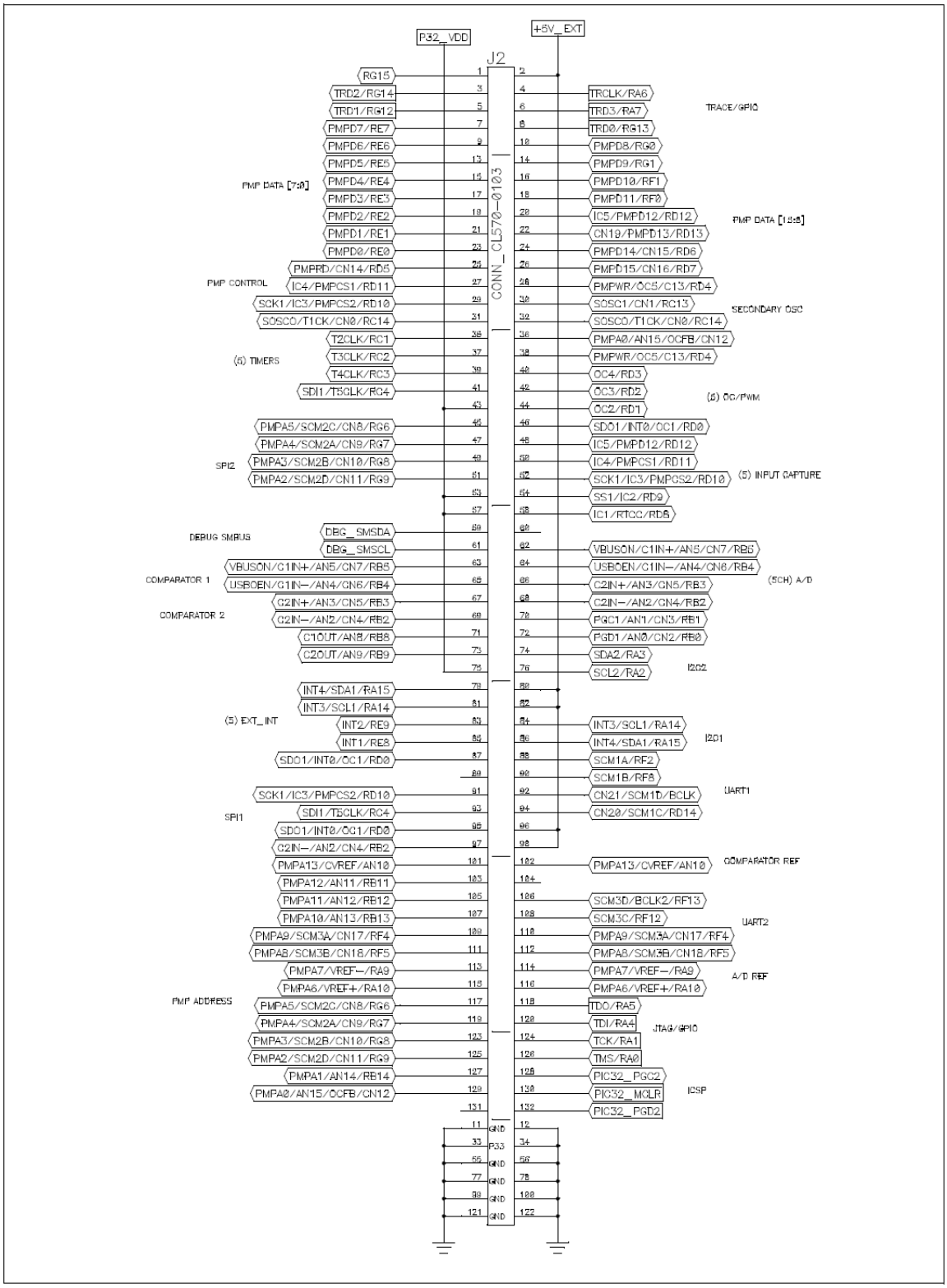

Fig. A.80: PIC32 USB Starter Kit II Schematic Page 3

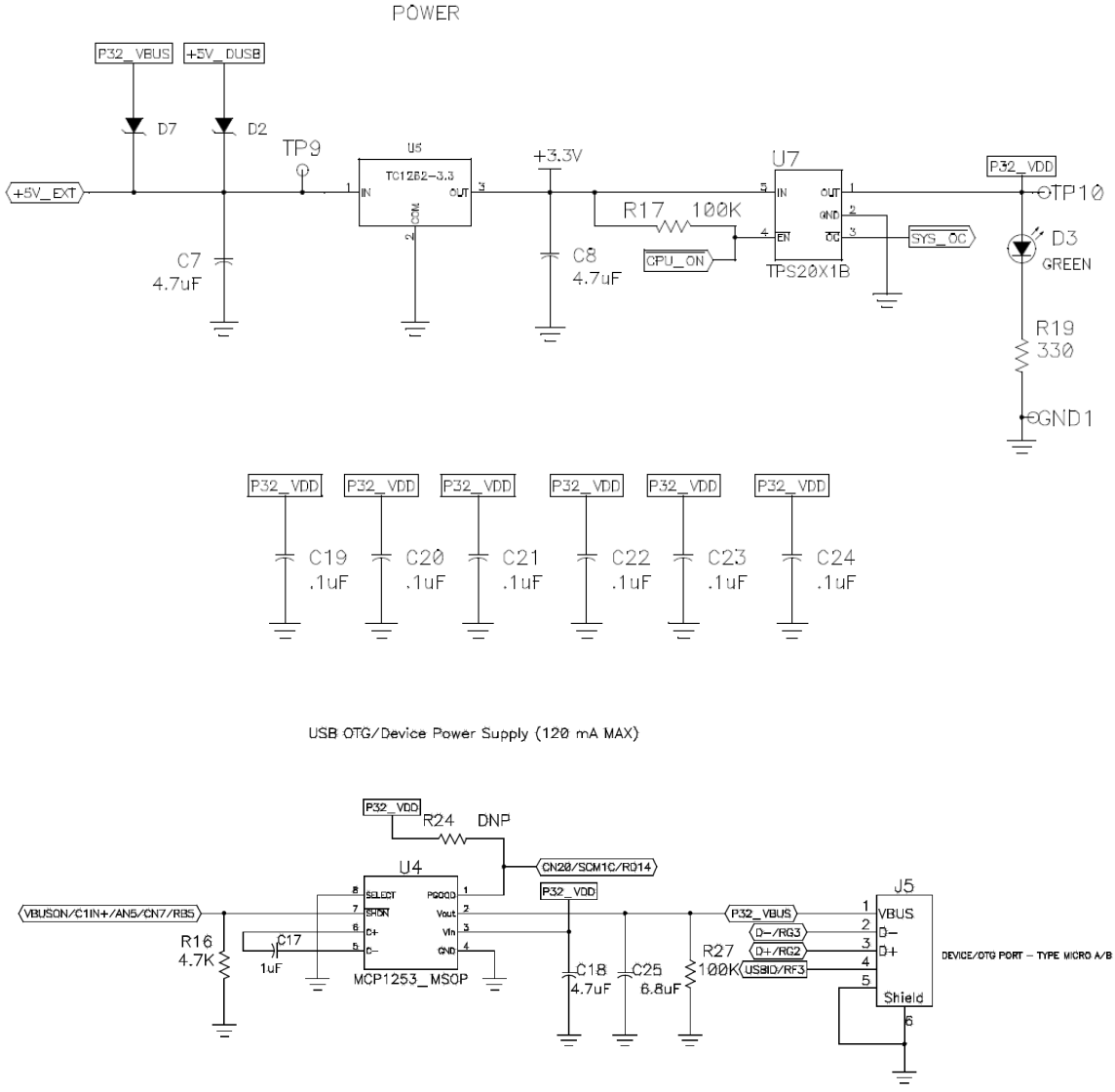

Fig. A.81: PIC32 USB Starter Kit II Schematic Page 4

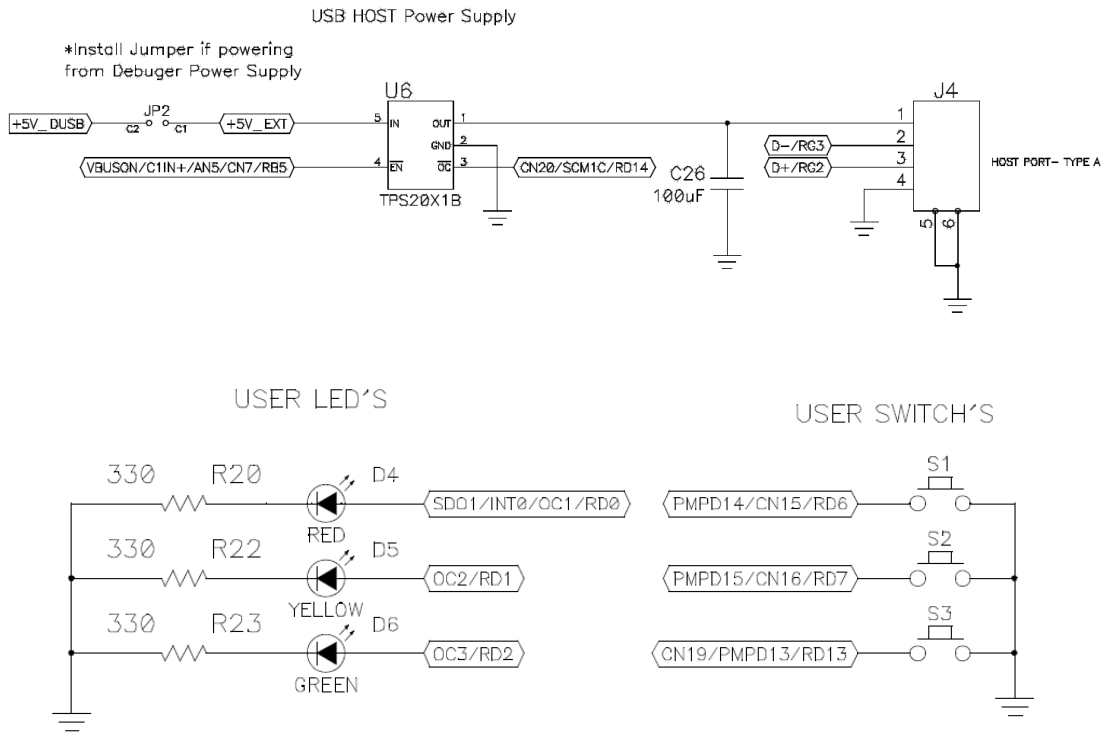

Fig. A.82: PIC32 USB Starter Kit II Schematic Page 5

APPENDIX B

The Fourier Transform of the Pulse Model

This appendix contains the derivation of the Fourier transform for the piecewise function used as a model for a pulse from Chapter 2.

$$f(t)=\begin{cases}\dfrac{a}{r}*t+a*\left(1+\dfrac{d}{2r}\right), & for-\dfrac{d}{2}-r\le t\le-\dfrac{d}{2}\\ a, & for-\dfrac{d}{2}\le t\le\dfrac{d}{2}\\ -\dfrac{a}{r}*t+a*\left(1+\dfrac{d}{2r}\right), & for\dfrac{d}{2}\le t\le\dfrac{d}{2}+r\end{cases}$$

$$\mathcal{F}(\omega)=\int_{-\infty}^{\infty}f(t)e^{-j\omega t}$$

$$\mathcal{F}(\omega)=\int_{-\frac{d}{2}-r}^{-\frac{d}{2}}a\left(1+\frac{d}{2r}\right)e^{-j\omega t}dt+\int_{-\frac{d}{2}-r}^{-\frac{d}{2}}\frac{a}{r}te^{-j\omega t}dt+\int_{-\frac{d}{2}}^{\frac{d}{2}}a\,e^{-j\omega t}dt$$

$$+\int_{\frac{d}{2}}^{\frac{d}{2}+r}a\left(1+\frac{d}{2r}\right)e^{-j\omega t}dt-\int_{\frac{d}{2}}^{\frac{d}{2}+r}\frac{a}{r}te^{-j\omega t}dt$$

$$\mathcal{F}(\omega)=\frac{-a}{j\omega}\left(1+\frac{d}{2r}\right)e^{-j\omega t}\Big|_{-\frac{d}{2}-r}^{-\frac{d}{2}}+\frac{-at}{j\omega r}e^{-j\omega t}\Big|_{-\frac{d}{2}-r}^{-\frac{d}{2}}+\frac{a}{\omega^2 r}e^{-j\omega t}\Big|_{-\frac{d}{2}-r}^{-\frac{d}{2}}+\frac{-a}{j\omega}e^{-j\omega t}\Big|_{-\frac{d}{2}}^{\frac{d}{2}}$$

$$+\frac{-a}{j\omega}\left(1+\frac{d}{2r}\right)e^{-j\omega t}\Big|_{\frac{d}{2}}^{\frac{d}{2}+r}+\frac{at}{j\omega r}e^{-j\omega t}\Big|_{\frac{d}{2}}^{\frac{d}{2}+r}-\frac{a}{j\omega^2}e^{-j\omega t}\Big|_{\frac{d}{2}}^{\frac{d}{2}+r}$$

$$\mathcal{F}(\omega)=\frac{-a}{j\omega}\left(1+\frac{d}{2r}\right)e^{j\omega\frac{d}{2}}+\frac{a}{j\omega}\left(1+\frac{d}{2r}\right)e^{j\omega\left(\frac{d}{2}+r\right)}+\frac{ad}{2j\omega r}e^{j\omega\frac{d}{2}}$$

$$-\frac{a}{j\omega r}\left(\frac{d}{2}+r\right)e^{j\omega\left(\frac{d}{2}+r\right)}+\frac{a}{r\omega^2}e^{j\omega\frac{d}{2}}-\frac{a}{r\omega^2}e^{j\omega\left(\frac{d}{2}+r\right)}-\frac{a}{j\omega}e^{-j\omega\frac{d}{2}}$$

$$+\frac{a}{j\omega}e^{j\omega\frac{d}{2}}+\frac{-a}{j\omega}\left(1+\frac{d}{2r}\right)e^{-j\omega\left(\frac{d}{2}+r\right)}+\frac{a}{j\omega}\left(1+\frac{d}{2r}\right)e^{-j\omega\frac{d}{2}}$$

$$+\frac{a}{j\omega r}\left(\frac{d}{2}+r\right)e^{-j\omega\left(\frac{d}{2}+r\right)}-\frac{a}{j\omega r}\frac{d}{2}e^{-j\omega\frac{d}{2}}-\frac{a}{r\omega^2}e^{-j\omega\left(\frac{d}{2}+r\right)}+\frac{a}{r\omega^2}e^{-j\omega\frac{d}{2}}$$

$$\mathcal{F}(\omega) = \frac{-2a}{\omega}\left(1+\frac{d}{2r}\right)\sin\left(\frac{d\omega}{2}\right) + \frac{2a}{\omega}\left(1+\frac{d}{2r}\right)\sin\left(\frac{d\omega}{2}+r\omega\right) + \frac{ad}{\omega r}\sin\left(\frac{d\omega}{2}\right)$$

$$- \frac{2ad}{\omega r}\left(\frac{d}{2}+r\right)\sin\left(\frac{d\omega}{2}+r\omega\right) + \frac{2a}{r\omega^2}\cos\left(\frac{d\omega}{2}\right) - \frac{2a}{r\omega^2}\cos\left(\frac{d\omega}{2}+r\omega\right)$$

$$+ \frac{2a}{\omega}\sin\left(\frac{d\omega}{2}\right)$$

$$\mathcal{F}(\omega) = \frac{2a}{r\omega^2}\left[\cos\left(\frac{d\omega}{2}\right) - \cos\left(\frac{d\omega}{2}+r\omega\right)\right]$$

$$\mathcal{F}(\omega) = \frac{2a}{r\omega^2}\left[-2\sin\left(\frac{d\omega}{2}+r\omega\right)\sin\left(\frac{d\omega}{2}\right)\right]$$

$$\mathcal{F}(\omega) = -a * (d+r) * \mathrm{sinc}\left(\frac{(d+r)\omega}{2}\right) * \mathrm{sinc}\left(\frac{r\omega}{2}\right)$$

APPENDIX C

Software

This appendix contains the software for the PFTNA. The following files from the Microchip Applications Library are not included in the code, but are necessary to have in the project: FSIO.c,usb.h, usb_ch9.h, usb_common.h, usb_hal.h, usb_host.h, usb_host.h, usb_host_local.h, usb_host_msd.h usb_host_msd_scsi.h, usb_host.c, usb_host_msd.c, usb_host_msd_scsi.c, button.c, button.h, DisplayDriver.c, DisplayDriver.h, DisplayDriverInterface.h, editbox.c, editbox.h, gfxepmp.c, GOL.c, GOL.h, GOLFontDefault.c, Graphics.h, Palette.c, Palette.h, picture.c, picture.h, Primitive.c, Primitive.h, radiobutton.c, radiobutton.h, ScanCodes.h, slider.c, slider.h, statictext.c, statictext.h, window.c, window.h, timedelay.c, timedelay.h, and GenericTypeDefs.h.

Description of Files

The AD9958.h & AD9958.c files contain the functions that program the direct digital synthesis IC.

The DAC7574.h & DAC7574.c files contain the functions that run the DAC IC.

The DSP.h & DSP.c files contain the digital signal processing functions as well as the ADC.

The FSconfig.h file contains the configuration for the file system.

The GraphicsConfig.h file contains the configuration settings for the graphics driver.

The HardwareProfile_CONTROL_BOARD.h file contains all of the hardware pin mappings.

The ISL95810.h & ISL95810.c files contain the functions for the digital potentiometer.

The Main.h & Main.c files are the main program.

The SST25VF032.h & SST25VF032.c files contain the functions for programming the 32Mbit flash memory.

The Touchscreen.h & TouchScreen.c files contain the functions that initiate and run the touchscreen.

The Usb_config.h & usb_config.c files configure the USB.

The UserInterface.h & UserInterface.c files contain the functions that draw out the user interface as well as control the button functions.

```
/************************************************************************
* Company: Baylor University
* Engineer: Brandon Herrera
* File Name:  AD9958.h
* Project Name:  PFTNA
* Target Device:       Control Board Rev A
* Compiler: C32
* Dependencies: M.A.L.
* Description: This is the main oscilloscope like user interface for
*                     the Pulsed Fourier Transform Network Analyzer.
*
************************************************************************
* Date         Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* 6/6/11     File Created
* 7/9/11    Initial Release
************************************************************************/
#ifndef _AD9958_H
#define _AD9958_H

#include <plib.h>
#include "GenericTypeDefs.h"
#include "HardwareProfile.h"
#include "TimeDelay.h"

// Defines the SPI channel registers
#define DDS_SPISTAT     SPI2STAT
#define DDS_SPISTATbits SPI2STATbits
#define DDS_SPICON      SPI2CON
#define DDS_SPICONbits  SPI2CONbits
#define DDS_SPIBRG      SPI2BRG
#define DDS_SPIBUF      SPI2BUF

/************************************************************************
* Macros DDSSCLow()
*
* Overview:  this macro lowers the DDS chip select pins
*
* Input: none
*
* Output: none
*
************************************************************************/
#define DDSCSLow()    DDS_CS_LAT = 0;

/************************************************************************
* Macros DDSCSHigh()
*
* Overview:  this macro raises the DDS chip select pins
*
```

```
* Input: none
*
* Output: none
*
*********************************************************************/
#define DDSCSHigh()    DDS_CS_LAT = 1;

/*********************************************************************
* Macros DDSUpdateHigh()
*
* Overview:  this macro raises the DDS I/O Update pins
*
* Input: none
*
* Output: none
*
*********************************************************************/
#define DDSUpdateHigh()    DDS_UPDATE_LAT = 1;

/*********************************************************************
* Macros DDSUpdateLow()
*
* Overview:  this macro raises the DDS I/O Update pins
*
* Input: none
*
* Output: none
*
*********************************************************************/
#define DDSUpdateLow()    DDS_UPDATE_LAT = 0;

/*********************************************************************
* Function DDSInit(void)
*
* Overview:  this function initiates the direct digital synthesis chips
*
* Input: none
*
* Output: none
*
*********************************************************************/
void DDSInit(void);

/*********************************************************************
* Function DDSSet(void)
*
* Overview:  this function holds the DDS chips for no output
*
* Input: none
*
* Output: none
*
*********************************************************************/
void DDSSet(void);
```

```
/*********************************************************************
* Function DDSRelease(void)
*
* Overview:  this function lets the DDS chips run again
*
* Input: none
*
* Output: none
*
*********************************************************************/
void DDSRelease(void);

/*********************************************************************
* Function DDSProgram(void)
*
* Overview:  this function programs the direct digital synthesis chips
*
* Input: none
*
* Output: none
*
*********************************************************************/
void DDSProgram(void);

/*********************************************************************
* Function DDSPut(BYTE data)
*
* Overview:  this function sends a byte
*
* Input: byte to be sent
*
* Output: none
*
*********************************************************************/
void DDSPut(BYTE data);


#endif //_AD9958_H
```

```
/*****************************************************************
* Company: Baylor University
* Engineer: Brandon Herrera
* File Name:  AD9958.c
* Project Name:  PFTNA
* Target Device:       Control Board Rev A
* Compiler: C32
* Dependencies: M.A.L.
* Description: This is the main oscilloscope like user interface for
*                     the Pulsed Fourier Transform Network Analyzer.
*
*****************************************************************
* Date         Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* 6/6/11     File Created
* 7/9/11    Initial Release
*****************************************************************/
#include "AD9958.h"


/*****************************************************************
* Function DDSInit(void)
*
* Overview:  this function initiates the direct digital synthesis chips
*
* Input: none
*
* Output: none
*
*****************************************************************/
void DDSInit(void)
{
/*
    DDS_SPISTAT = 0;
    DDS_SPICON = 0;  //Reset the SPI
    DDS_SPIBRG = 0;
    DDS_SPICONbits.MSTEN = 1;
    DDS_SPICONbits.CKP = 0;
    DDS_SPICONbits.CKE = 1;
    DDS_SPICONbits.SMP = 1;
    DDS_SPIBRG = 1;
    DDS_SPICONbits.ON = 1;

      DDS_UPDATE_TRIS = 0;
    DDS_UPDATE_LAT = 0;

    DDS_CS_LAT = 1;
    DDS_CS_TRIS = 0;

   // DDS_SCK_TRIS = 0;
```

```
   // DDS_SDO_TRIS = 0;
*/


OpenTimer3(T3_ON|T3_PS_1_1|T3_SOURCE_INT, 0x0190);

SpiChnOpen(2,SPI_CONFIG_MODE8|SPI_CONFIG_FSP_WIDE|SPI_CONFIG_MSTEN|SPI_
CONFIG_ON|SPI_CONFIG_MSSEN|SPI_OPEN_CKE_REV,80);

      DDS_UPDATE_TRIS = 0;
    DDS_UPDATE_LAT = 0;

      DDS_RST_LAT = 0;
      DDS_RST_TRIS = 0;
}


/*********************************************************************
* Function DDSPut(BYTE data)
*
* Overview:  this function sends a byte
*
* Input: byte to be sent
*
* Output: none
*
*********************************************************************/
void DDSPut(BYTE data)
{

    // Wait for free buffer
    while(!DDS_SPISTATbits.SPITBE);
    DDS_SPIBUF = data;

    // Wait for data byte
    while(!DDS_SPISTATbits.SPIRBF);

}

/*********************************************************************
* Macros DDSGet()
*
* Overview:  this macros gets a byte from SPI
*
* Input: none
*
* Output: none
*
*********************************************************************/
#define DDSGet()     DDS_SPIBUF


/*********************************************************************
```

```
* Function: void DDSWriteByte(BYTE data, DWORD address)
*
* Overview: this function writes a byte to the address specified
*
* Input: data to be written and address
*
* Output: none
*
*********************************************************************/
/*void DDSWriteByte(BYTE data, DWORD address)
{
    SST25WriteEnable();
    SST25CSLow();

    SPIPut(SST25_CMD_WRITE);
    SPIGet();

    DDSPut(((DWORD_VAL) address).v[2]);
    DDDSGet();

    SPIPut(((DWORD_VAL) address).v[1]);
    SPIGet();

    SPIPut(((DWORD_VAL) address).v[0]);
    SPIGet();

    SPIPut(data);
    SPIGet();

    SST25CSHigh();

    // Wait for write end
    while(SST25IsWriteBusy());
}

*/

/*********************************************************************
* Function DDSProgram(void)
*
* Overview:  this function programs the direct digital synthesis chips
*
* Input: none
*
* Output: none
*
*********************************************************************/
void DDSProgram(void)
{
int i = 0;
/*
DDSCSLow();
DDSPut(0b00000000);
DDSPut(0b01000000);
```

```
DDSPut(0b00000001);
DDSPut(0b11010000);
DDSPut(0b00000000);
DDSPut(0b00000000);

DDSPut(0b00000100);
DDSPut(0b01000001);
DDSPut(0b10001001);
DDSPut(0b00110111);
DDSPut(0b01001100);

while(SST25IsWriteBusy());

DDSCSHigh();
DelayMs(2);
DDSUpdateHigh();

DDSUpdateLow();

*/
    DDS_RST_LAT = 1;
                while(i<30)                         //short pause
                {
                  i++;
                }
                i=0;

    DDS_RST_LAT = 0;

      DelayMs(1);

    DDS_CS_LAT = 0;

    SpiChnPutC(2, 0b00000001);  //Set PLL
    SpiChnPutC(2, 0b11010000);
    SpiChnPutC(2, 0b00000000);
      SpiChnPutC(2, 0b00000000);

    SpiChnPutC(2, 0x00000010);   //AutoClear All
    SpiChnPutC(2, 0b00000000);
    SpiChnPutC(2, 0b00000000);

    SpiChnPutC(2, 0x00);   //Select Channel 0
    SpiChnPutC(2, 0b01110000);


    SpiChnPutC(2, 0b00000100);
      SpiChnPutC(2, 0x3D);  //120
      SpiChnPutC(2, 0x70);
      SpiChnPutC(2, 0xA3);
      SpiChnPutC(2, 0xD7);

    SpiChnPutC(2, 0b00000011);   //AutoClear
```

```
    SpiChnPutC(2, 0b00000000);
    SpiChnPutC(2, 0b00000011);
    SpiChnPutC(2, 0b00100101);

    SpiChnPutC(2, 0x00);   //Select channel 1
    SpiChnPutC(2, 0b10110000);

    SpiChnPutC(2, 0b00000100);
      SpiChnPutC(2, 0x3D);  //119.9992
      SpiChnPutC(2, 0x70);
      SpiChnPutC(2, 0x88);
      SpiChnPutC(2, 0xFF);

    SpiChnPutC(2, 0b00000011);   //AutoClear
    SpiChnPutC(2, 0b00000000);
    SpiChnPutC(2, 0b00000011);
    SpiChnPutC(2, 0b00100101);



       while(SPI2STATbits.SPIBUSY)    //waits until channel in not busy
                {
                }

    DDS_UPDATE_LAT = 1;
                while(i<30)                         //short pause
                {
                  i++;
                }
                i=0;

    DDS_UPDATE_LAT = 0;
    DDS_CS_LAT = 1;
}

/*********************************************************************
* Function DDSSet(void)
*
* Overview:  this function holds the DDS chips for no output
*
* Input: none
*
* Output: none
*
*********************************************************************/
void DDSSet(void)
{
      int i = 0;


    DDS_CS_LAT = 0;

    SpiChnPutC(2, 0x00000000);
    SpiChnPutC(2, 0b11000000);
```

```
      SpiChnPutC(2, 0b00000011);
      SpiChnPutC(2, 0b01100110);

       while(SPI2STATbits.SPIBUSY)    //waits until channel in not busy
                {
                }

    DDS_CS_LAT = 1;

    DDS_UPDATE_LAT = 1;
                while(i<30)                         //short pause
                {
                  i++;
                }
                i=0;

    DDS_UPDATE_LAT = 0;

}

/*********************************************************************
* Function DDSRelease(void)
*
* Overview:  this function lets the DDS chips run again
*
* Input: none
*
* Output: none
*
*********************************************************************/
void DDSRelease(void)
{
      int i = 0;
    DDS_CS_LAT = 0;

    SpiChnPutC(2, 0x00000000);
    SpiChnPutC(2, 0b11000000);

      SpiChnPutC(2, 0b00000011);
      SpiChnPutC(2, 0b00100100);


       while(SPI2STATbits.SPIBUSY)    //waits until channel in not busy
                {
                }

    DDS_CS_LAT = 1;

    DDS_UPDATE_LAT = 1;
                while(i<30)                         //short pause
                {
                  i++;
                }
```

```
                i=0;

    DDS_UPDATE_LAT = 0;

}
```

```
/********************************************************************
* Company: Baylor University
* Engineer: Brandon Herrera
* File Name:  DAC7574.h
* Project Name:  PFTNA
* Target Device:       Control Board Rev A
* Compiler: C32
* Dependencies: M.A.L.
* Description: This is the main oscilloscope like user interface for
*                    the Pulsed Fourier Transform Network Analyzer.
*
********************************************************************
* Date         Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* 6/6/11     File Created
* 7/9/11    Initial Release
********************************************************************/
#ifndef _DAC7574_H
#define _DAC7574_H

#include <plib.h>
#include "TimeDelay.h"
#include "HardwareProfile.h"
#include "GenericTypeDefs.h"

// The I2C address for the DAC
#define DACAddress 0b1001100

// The addresses for each of the 4 channels
#define Channel_A 0b00010000
#define Channel_B 0b00010010
#define Channel_C 0b00010100
#define Channel_D 0b00010110

/********************************************************************
* Macros DACStop()
*
* Overview:  this macros releases the I2C channel
*
* Input: none
*
* Output: none
*
********************************************************************/
#define DACStop() I2CStop(I2C2);

/********************************************************************
* Function: void DACInit(void)
*
* Overview: this function initiates the digital to analog converter
```

```
*
* Input: none
*
* Output: none
*
*********************************************************************/
void DACInit(void);

/*********************************************************************
* Function: void DACPut(BYTE data)
*
* Overview: this function puts a byte to the DAC
*
* Input: byte to be written to the DAC
*
* Output: none
*
*********************************************************************/
void DACPut(BYTE data);

/*********************************************************************
* Function: void DACProgram(BYTE channel, BYTE UData, BYTE LData)
*
* Overview: this function programs the appropriate channel of the DAC
*
* Input: the channel to write to and the upper and lower bytes of data
*        to write
*
* Output: none
*
*********************************************************************/
void DACProgram(BYTE channel, BYTE UData, BYTE LData);

#endif //_DAC7574_H
```

```
/************************************************************************
* Company: Baylor University
* Engineer: Brandon Herrera
* File Name:  DAC7574.c
* Project Name:  PFTNA
* Target Device:       Control Board Rev A
* Compiler: C32
* Dependencies: M.A.L.
* Description: This is the main oscilloscope like user interface for
*                    the Pulsed Fourier Transform Network Analyzer.
*
************************************************************************
* Date         Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* 6/6/11     File Created
* 7/9/11    Initial Release
************************************************************************/
#include "DAC7574.h"

/************************************************************************
* Function: void DACInit(void)
*
* Overview: this function initiates the digital to analog converter
*
* Input: none
*
* Output: none
*
************************************************************************/
void DACInit(void)
{
      WORD actualClock;
      I2CConfigure(I2C2,I2C_STOP_IN_IDLE|I2C_ENABLE_SLAVE_CLOCK_STRETCH
ING);
      actualClock = I2CSetFrequency(I2C2, GetPeripheralClock(),
I2C_CLOCK_FREQ);
      if ( abs(actualClock-I2C_CLOCK_FREQ) > I2C_CLOCK_FREQ/10 )
      {
          DBPRINTF("  Clock frequency (%d) error exceeds 10\%\r\n",
actualClock);
      }
      else
      {
          //DBPRINTF("  Clock frequency = %ld Hz\n", actualClock);
      }

      I2CEnable(I2C2, TRUE);

}
```

```
/*********************************************************************
* Function: void DACPut(BYTE data)
*
* Overview: this function puts a byte to the DAC
*
* Input: byte to be written to the DAC
*
* Output: none
*
*********************************************************************/
void DACPut(BYTE data)
{
      BYTE result;
      while(!I2CTransmitterIsReady(I2C2));

            result = I2CSendByte(I2C2, data);


       if(result != I2C_SUCCESS)
       {
           DBPRINTF("Error Byte 1 %d\n", result);
       }

      while (!I2CTransmissionHasCompleted(I2C2));

          if (!I2CByteWasAcknowledged(I2C2))
          {
              DBPRINTF("No ack");
          }

}

/*********************************************************************
* Function: void DACProgram(BYTE channel, BYTE UData, BYTE LData)
*
* Overview: this function programs the appropriate channel of the DAC
*
* Input: the channel to write to and the upper and lower bytes of data
to
*        write
*
* Output: none
*
*********************************************************************/
void DACProgram(BYTE channel, BYTE UData, BYTE LData)
{
      BYTE result;
    if (I2CBusIsIdle(I2C2))
    {
        result = I2CStart(I2C2);
    }

    if(result != I2C_SUCCESS)
    {
```

```
        DBPRINTF("Error Start %d\n", result);
    }
Delay10us(5);

DACPut((DACAddress << 1)|I2C_WRITE);

DACPut(channel);

DACPut(UData);
DACPut(LData);

DACStop();

DelayMs(1);
}
```

DSP.h

```
/*********************************************************************
* Company: Baylor University
* Engineer: Brandon Herrera
* File Name:  DSP.h
* Project Name:  PFTNA
* Target Device:       Control Board Rev A
* Compiler: C32
* Dependencies: M.A.L.
* Description: This is the main oscilloscope like user interface for
*                    the Pulsed Fourier Transform Network Analyzer.
*
*********************************************************************
* Date         Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* 6/6/11     File Created
* 7/9/11    Initial Release
*********************************************************************/
#ifndef _DSP_H
#define _DSP_H

#include "dsplib_def.h"
#include <plib.h>
#include "GenericTypeDefs.h"
#include "HardwareProfile.h"
#include "TimeDelay.h"
#include "main.h"
#include "math.h"
#include "limits.h"
#include "dsplib_dsp.h"
#include "fftc.h" // pre-computed coefficients
//    #include "dsplib_dsp.h"

// The FFT twiddle factors
#define fftc fft32c4096

/*********************************************************************
* Function: void ADCInit(void)
*
* Overview: this function initiates the analog to digital converter
*
* Input: none
*
* Output: none
*
*********************************************************************/
void ADCInit(void);

/*********************************************************************
* Function: void TakeData(void)
*
```

```
* Overview: this function takes the data
*
* Input: none
*
* Output: none
*
*****************************************************************/
void TakeData(void);

/*****************************************************************
* Function: void DoFFT(void)
*
* Overview: this function initiates the digital to analog converter
*
* Input: data is the 10 bit integer array from the ADC
*
* Output: FFT output in 64bit floating point format
*
*****************************************************************/
void DoFFT(short data[], double fft_dispersed[]);



#endif //_DSP_H
```

```
/*********************************************************************
* Company: Baylor University
* Engineer: Brandon Herrera
* File Name:  DSP.c
* Project Name:  PFTNA
* Target Device:       Control Board Rev A
* Compiler: C32
* Dependencies: M.A.L.
* Description: This is the main oscilloscope like user interface for
*                    the Pulsed Fourier Transform Network Analyzer.
*
*********************************************************************
* Date         Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* 6/6/11     File Created
* 7/9/11    Initial Release
*********************************************************************/
#include "DSP.h"

////////////////////////////////////////////////////////////////////
//                              EXTERNAL VARIABLES
////////////////////////////////////////////////////////////////////
extern unsigned short cir_buff[50];
extern unsigned short test_data[4096];
extern double fout[2048];
extern int trigger;
extern int averaging;
extern int N;
extern unsigned short Trig_Level;

#define SYS_FREQ        (80000000L)

/*********************************************************************
* Function: void ADCInit(void)
*
* Overview: this function initiates the analog to digital converter
*
* Input: none
*
* Output: none
*
*********************************************************************/
void ADCInit(void)
{
TRISBbits.TRISB9 = 1;
AD1PCFGbits.PCFG9 = 0;
//OpenTimer3(T3_ON|T3_PS_1_1|T3_SOURCE_INT, 0x0320);  //800
//OpenTimer3(T3_ON|T3_PS_1_1|T3_SOURCE_INT, 0x0190);  //400
//OpenTimer3(T3_ON|T3_PS_1_1|T3_SOURCE_INT, 0x00C8);  //200
OpenTimer3(T3_ON|T3_PS_1_1|T3_SOURCE_INT, 0x0064);  //100
```

```
 // Configure the device for maximum performance but do not change the
PBDIV
 // Given the options, this function will change the flash wait states
and
 // enable prefetch cache but will not change the PBDIV. The PBDIV
value
 // is already set via the pragma FPBDIV option above..
 //SYSTEMConfig(SYS_FREQ, SYS_CFG_WAIT_STATES | SYS_CFG_PCACHE);

 // configure and enable the ADC
 CloseADC10(); // ensure the ADC is off before setting the
configuration

 // define setup parameters for OpenADC10
 //     Turn module on | ouput in integer | trigger mode auto | enable
autosample
 #define PARAM1  ADC_MODULE_ON | ADC_FORMAT_INTG | ADC_CLK_TMR |
ADC_AUTO_SAMPLING_ON

 // define setup parameters for OpenADC10
 //     ADC ref external    | disable offset test    | disable scan
mode | perform 1 samples | use dual buffers | use alternate mode
 #define PARAM2  ADC_VREF_AVDD_AVSS | ADC_OFFSET_CAL_DISABLE |
ADC_SCAN_OFF | ADC_SAMPLES_PER_INT_1 | ADC_ALT_BUF_OFF |
ADC_ALT_INPUT_OFF

 // define setup parameters for OpenADC10
 //      use ADC internal clock | set sample time
 //#define PARAM3  ADC_CONV_CLK_SYSTEM | ADC_SAMPLE_TIME_20 |
ADC_CONV_CLK_9Tcy2
 #define PARAM3  ADC_CONV_CLK_SYSTEM | ADC_SAMPLE_TIME_30 |
ADC_CONV_CLK_3Tcy2

 // define setup parameters for OpenADC10
 //               set AN4 and AN5 as analog inputs
 #define PARAM4 ENABLE_AN9_ANA


 // define setup parameters for OpenADC10
 // do not assign channels to scan
 #define PARAM5 SKIP_SCAN_ALL

 // use ground as neg ref for A | use AN4 for input A      |  use
ground as neg ref for A | use AN5 for input B

 // configure to sample AN4 & AN5
 SetChanADC10( ADC_CH0_NEG_SAMPLEA_NVREF | ADC_CH0_POS_SAMPLEA_AN9 |
ADC_CH0_NEG_SAMPLEB_NVREF | ADC_CH0_POS_SAMPLEB_AN9); // configure to
sample AN4 & AN5
 OpenADC10( PARAM1, PARAM2, PARAM3, PARAM4, PARAM5 ); // configure ADC
using the parameters defined above
 //ConfigIntADC10(ADC_INT_ON|ADC_INT_PRI_5|ADC_INT_SUB_PRI_2);
 EnableADC10(); // Enable the ADC
```

```
}


/*********************************************************************
* Function: void TakeData(void)
*
* Overview: this function takes the data
*
* Input: none
*
* Output: none
*
*********************************************************************/
void TakeData(void)
{
int count=0;
 int i;
int pos=0;
int trigger_pos;
int j;

mT1ClearIntFlag();
ConfigIntTimer1(T1_INT_OFF | T1_INT_PRIOR_3);  //disable touch screen
ConfigIntTimer45(T45_INT_OFF | T45_INT_PRIOR_2);
mT1ClearIntFlag();

DisableIntT5;
DisableIntT1;

ADCInit();

  // DATA AQUSITION
  for (i=0;i<averaging;i++)
  {
   // Capture 50 samples in a circular buffer till it triggers.
   while (!trigger)
   {
//    while(!AD1CON1bits.DONE)
    while ( ! mAD1GetIntFlag() )
          {
              // wait for the first conversion to complete so there
will be vaild data in ADC result registers
          }

    cir_buff[pos]=ReadADC10(0);
    mAD1ClearIntFlag();
      count++;

    //if (cir_buff[pos] < 350)  // This is the trigger detection
    if ( (cir_buff[pos] < Trig_Level) | (count > 9000) )  // This is
the trigger detection
    {
     trigger=1;
     trigger_pos=pos+1;
```

```
            count=0;
     }
     //pos=(pos+1)%50;
       pos++;
       if (pos == 50)
            {
                  pos=0;
                  count++;

            }

   }


   // Capture the Dispersed Pulse.
   for (j=50;j<N;j++)
   {

    while ( ! mAD1GetIntFlag() )
          {
              // wait for the first conversion to complete so there
will be vaild data in ADC result registers
          }
    test_data[j]=ReadADC10(0)+test_data[j];
    mAD1ClearIntFlag();
   }
   trigger=0;

   // Reconstruct the circular buffer into the first part of the cal
data.
   for (j=0;j<50;j++)
   {
    test_data[j]=cir_buff[(j+trigger_pos)%50]+test_data[j];
      cir_buff[(j+trigger_pos)%50]=0;
   }
  }

  // Divide for the average
  for (j=0;j<N;j++)
  {
   test_data[j]=test_data[j]/averaging;
  }

      DoFFT(test_data,fout);

      EnableIntT5;
      EnableIntT1;
      ConfigIntTimer1(T1_INT_ON | T1_INT_PRIOR_3);  //Enable touch
screen
      ConfigIntTimer45(T45_INT_ON | T45_INT_PRIOR_2);
}
  // END DATA AQUSITION.

/*********************************************************************
```

```
* Function: void DoFFT(void)
*
* Overview: this function initiates the digital to analog converter
*
* Input: data is the 10 bit integer array from the ADC
*
* Output: FFT output in 64bit floating point format
*
*********************************************************************/
void DoFFT(short data[], double fft_dispersed[])
{
      //unsigned short test[1024];
//test[0]=1024;

//test[2]=1024;


      int log2N = 12;

      int32c din[4096];
      int32c dout[4096];
      int32c scratch[4096];

      double realX;
      double imagX;

      double sumX;
      double sqX;

      int j;
      int i;
/*
  for (j=0;j<N;j++)
  {
      test[j]=0;
}
test[0]=1024;
test[1]=1024;
*/
 // mean=mean/samples; //calculate the mean


  // Convert to Q31
  for (j=0;j<N;j++)
  {
   din[j].re=data[j];
      din[j].im = 0;
   //din[j].re=din[j].re-mean;  //subtract the mean
   din[j].re= (int)((double)din[j].re * (double)2147483648 /
(double)8192);
   //din[j].re=din[j].re * Window[j];  //Apply the window
  }

  mips_fft32(dout, din, fftc, scratch, log2N);  //FFT
```

```
  dout[0].re=dout[1].re;   //remove DC
  dout[0].im=dout[1].im;

  for (j=0;j<(N/2);j++)
  {
   if ((dout[j].re == 0x7FFFFFFF) || (dout[j].re == 0xFFFFFFFF) ||
(dout[j].im == 0x7FFFFFFF) || (dout[j].im == 0xFFFFFFFF))
   {
    DBPRINTF("Q31 %d \n",j);
   }
   realX=(double)dout[j].re / (double)2147483648 * (double)8192;
//Convert from Q15 to floating point.
   imagX=(double)dout[j].im / (double)2147483648 * (double)8192;

   sumX=fabs(realX)*fabs(realX) + fabs(imagX)*fabs(imagX);  //sum the
squares
   sqX=sqrt(sumX);            //square root

   fft_dispersed[j]=(float)sqX;
   //fft_dispersed[(j+N/2)%N]=(float)sqX;
  }

  for (i=0;i<(N/2);i++)
  {
   //if (fout[i] == 0) {
   // fout[i]=0.01;
  // }
   fft_dispersed[i]=10*log10f(fft_dispersed[i]);
  }

}
```

```
/********************************************************************************
 *
 *            Microchip Memory Disk Drive File System
 *
 ********************************************************************************
 * FileName:        FSconfig.h
 * Dependencies:    None
 * Processor:       PIC18/PIC24/dsPIC30/dsPIC33
 * Compiler:        C18/C30
 * Company:         Microchip Technology, Inc.
 * Version:         1.0.0
 *
 * Software License Agreement
 *
 * The software supplied herewith by Microchip Technology Incorporated
 * (the "Company") for its PICmicro® Microcontroller is intended and
 * supplied to you, the Company's customer, for use solely and
 * exclusively on Microchip PICmicro Microcontroller products. The
 * software is owned by the Company and/or its supplier, and is
 * protected under applicable copyright laws. All rights are reserved.
 * Any use in violation of the foregoing restrictions may subject the
 * user to criminal sanctions under applicable laws, as well as to
 * civil liability for the breach of the terms and conditions of this
 * license.
 *
 * THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
 * WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 * TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 * IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 * CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 *
********************************************************************************/


#ifndef _FS_DEF_


#include "HardwareProfile.h"

/********************************************************************************/
/*  Note:  There are likely pin definitions present in the header file    */
/*       for your device (SP-SPI.h, CF-PMP.h, etc).  You may wish to    */
/*       specify these as well                                          */
/********************************************************************************/

// The FS_MAX_FILES_OPEN #define is only applicable when Dynamic
// memeory allocation is not used (FS_DYNAMIC_MEM not defined).
// Defines how many concurent open files can exist at the same time.
// Takes up static memory. If you do not need to open more than one
// file at the same time, then you should set this to 1 to reduce
```

```
// memory usage
#define FS_MAX_FILES_OPEN     2
/*********************************************************************
**/

// The size of a sector
// Must be 512, 1024, 2048, or 4096
// 512 bytes is the value used by most cards
#define MEDIA_SECTOR_SIZE           512
/*********************************************************************
**/

/*
*********************************************************************
*******************************/
/************** Compiler options to enable/Disable Features based on
user's application ******************/
/*
*********************************************************************
*******************************/

// Uncomment this to use the FindFirst, FindNext, and FindPrev
#define ALLOW_FILESEARCH
/*********************************************************************
**/
/*********************************************************************
**/

// Comment this line out if you don't intend to write data to the card
#define ALLOW_WRITES
/*********************************************************************
**/

// Comment this line out if you don't intend to format your card
// Writes must be enabled to use the format function
#define ALLOW_FORMATS
/*********************************************************************
**/

// Uncomment this definition if you're using directories
// Writes must be enabled to use directories
#define ALLOW_DIRS
/*********************************************************************
**/

// Allows the use of FSfopenpgm, FSremovepgm, etc with PIC18
//#define ALLOW_PGMFUNCTIONS
/*********************************************************************
**/

// Allows the use of the FSfprintf function
// Writes must be enabled to use the FSprintf function
#define ALLOW_FSFPRINTF
```

```
/*****************************************************************
**/

// If FAT32 support required then uncomment the following
#define SUPPORT_FAT32

// Allows the use of the FSGetDiskProperties() function to get
//   the size, free space, etc. of a drive
#define ALLOW_GET_DISK_PROPERTIES
/*
*****************************************************************
****************************** */



#if defined( __C30__ )
    // Select how you want the timestamps to be updated
    // Use the Real-time clock peripheral to set the clock
    // You must configure the RTC in your application code
    #define USEREALTIMECLOCK
    // The user will update the timing variables manually using the
SetClockVars function
    // The user should set the clock before they create a file or
directory (Create time),
    // and before they close a file (last access time, last modified
time)
    //#define USERDEFINEDCLOCK
    // Just increment the time- this will not produce accurate times
and dates
    //#define INCREMENTTIMESTAMP
#elif defined (__PIC32MX__)
    // Select how you want the timestamps to be updated
    // Use the Real-time clock peripheral to set the clock
    // You must configure the RTC in your application code
    //#define USEREALTIMECLOCK
    // The user will update the timing variables manually using the
SetClockVars function
    // The user should set the clock before they create a file or
directory (Create time),
    // and before they close a file (last access time, last modified
time)
    #define USERDEFINEDCLOCK
    // Just increment the time- this will not produce accurate times
and dates
    //#define INCREMENTTIMESTAMP
#endif



// Warnings
#ifdef USE_PIC18
      #ifdef USEREALTIMECLOCK
```

```
            #error The PIC18 architecture does not currently support
Real-time clock and calander mode
      #endif
#endif

#ifdef ALLOW_PGMFUNCTIONS
      #ifndef USE_PIC18
            #error The pgm functions are unneccessary when not using
PIC18
      #endif
#endif
#ifndef USEREALTIMECLOCK
    #ifndef USERDEFINEDCLOCK
        #ifndef INCREMENTTIMESTAMP
            #error Please enable USEREALTIMECLOCK, USERDEFINEDCLOCK, or
INCREMENTTIMESTAMP
        #endif
    #endif
#endif

/*********************************************************************
**/
// Define FS_DYNAMIC_MEM to use malloc for allocating
// FILE structure space.  uncomment all three lines
/*********************************************************************
**/
#if 0
      #define FS_DYNAMIC_MEM
      #ifdef USE_PIC18
            #define FS_malloc SRAMalloc
            #define FS_free         SRAMfree
      #else
            #define FS_malloc malloc
            #define FS_free         free
      #endif
#endif


// Function definitions
// Associate the physical layer functions with the correct physical
layer
#define USE_USB_INTERFACE               // USB host MSD library
#ifdef USE_SD_INTERFACE_WITH_SPI       // SD-SPI.c and .h

    #define MDD_MediaInitialize     MDD_SDSPI_MediaInitialize
    #define MDD_MediaDetect         MDD_SDSPI_MediaDetect
    #define MDD_SectorRead          MDD_SDSPI_SectorRead
    #define MDD_SectorWrite         MDD_SDSPI_SectorWrite
    #define MDD_InitIO              MDD_SDSPI_InitIO
    #define MDD_ShutdownMedia       MDD_SDSPI_ShutdownMedia
    #define MDD_WriteProtectState   MDD_SDSPI_WriteProtectState

#elif defined USE_CF_INTERFACE_WITH_PMP       // CF-PMP.c and .h
```

```
    #define MDD_MediaInitialize     MDD_CFPMP_MediaInitialize
    #define MDD_MediaDetect         MDD_CFPMP_MediaDetect
    #define MDD_SectorRead          MDD_CFPMP_SectorRead
    #define MDD_SectorWrite         MDD_CFPMP_SectorWrite
    #define MDD_InitIO              MDD_CFPMP_InitIO
    #define MDD_ShutdownMedia       MDD_CFPMP_ShutdownMedia
    #define MDD_WriteProtectState   MDD_CFPMP_WriteProtectState
    #define MDD_CFwait              MDD_CFPMP_CFwait
    #define MDD_CFwrite             MDD_CFPMP_CFwrite
    #define MDD_CFread              MDD_CFPMP_CFread

#elif defined USE_MANUAL_CF_INTERFACE         // CF-Bit transaction.c
and .h

    #define MDD_MediaInitialize     MDD_CFBT_MediaInitialize
    #define MDD_MediaDetect         MDD_CFBT_MediaDetect
    #define MDD_SectorRead          MDD_CFBT_SectorRead
    #define MDD_SectorWrite         MDD_CFBT_SectorWrite
    #define MDD_InitIO              MDD_CFBT_InitIO
    #define MDD_ShutdownMedia       MDD_CFBT_ShutdownMedia
    #define MDD_WriteProtectState   MDD_CFBT_WriteProtectState
    #define MDD_CFwait              MDD_CFBT_CFwait
    #define MDD_CFwrite             MDD_CFBT_CFwrite
    #define MDD_CFread              MDD_CFBT_CFread

#elif defined USE_USB_INTERFACE               // USB host MSD library

    #define MDD_MediaInitialize     USBHostMSDSCSIMediaInitialize
    #define MDD_MediaDetect         USBHostMSDSCSIMediaDetect
    #define MDD_SectorRead          USBHostMSDSCSISectorRead
    #define MDD_SectorWrite         USBHostMSDSCSISectorWrite
    #define MDD_InitIO();
    #define MDD_ShutdownMedia       USBHostMSDSCSIMediaReset
    #define MDD_WriteProtectState   USBHostMSDSCSIWriteProtectState

#endif

#endif
```

```
/*****************************************************************************
 * Microchip Graphics Library
 *****************************************************************************
 * FileName:        GraphicsConfig.h
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:          MPLAB C30/C32
 * Company:         Microchip Technology Incorporated
 *
 * Software License Agreement
 *
 * Copyright © 2010 Microchip Technology Inc.  All rights reserved.
 * Microchip licenses to you the right to use, modify, copy and distribute
 * Software only when embedded on a Microchip microcontroller or digital
 * signal controller, which is integrated into your product or third party
 * product (pursuant to the sublicense terms in the accompanying license
 * agreement).
 *
 * You should refer to the license agreement accompanying this Software
 * for additional information regarding your rights and obligations.
 *
 * SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
 * KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY
 * OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR
 * PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR
 * OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION,
 * BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT
 * DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL,
 * INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA,
 * COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY
 * CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
 * OR OTHER SIMILAR COSTS.
 *
 * Date        Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *****************************************************************************/
#ifndef _GRAPHICSCONFIG_H
#define _GRAPHICSCONFIG_H

//////////////////// COMPILE OPTIONS AND DEFAULTS ////////////////////

/*****************************************************************
* Overview: Blocking and Non-Blocking configuration selection. To
*                       enable non-blocking configuration USE_NONBLOCKING_CONFIG
*                       must be defined. If this is not defined, blocking
*                       configuration is assumed.
*
*****************************************************************/
    #define USE_NONBLOCKING_CONFIG   // Comment this line to use blocking configuration

/*****************************************************************************
```

```
* Overview: Using Palettes, different colors can be used with the same
*                 bit depth.
*
*********************************************************************/
//    #define USE_PALETTE // Comment this line if Palette is not
required.

    #ifdef USE_PALETTE
        //#define USE_PALETTE_EXTERNAL // Uncomment this line if
Palette must be stored in External Memory.
    #endif

/*********************************************************************
* Overview: Keyboard control on some objects can be used by enabling
*                 the GOL Focus (USE_FOCUS)support.
*
*********************************************************************/

#define USE_FOCUS

/*********************************************************************
* Overview: Input devices used defines the messages that Objects will
*                 process. The following definitions indicate the usage
of
*                 the different input device:
*                 - USE_TOUCHSCREEN - enables the touch screen support.
*                 - USE_KEYBOARD - enables the key board support.
*
*********************************************************************/

#define USE_TOUCHSCREEN             // Enable touch screen support.
#define USE_KEYBOARD                // Enable key board support.

/*********************************************************************
* Overview: To save program memory, unused Widgets or Objects can be
*                 removed at compile time.
*
*********************************************************************/
#define USE_GOL                 // Enable Graphics Object Layer.

//#define USE_BUTTON            // Enable Button Object.

//#define USE_WINDOW                // Enable Window Object.
//#define USE_CHECKBOX              // Enable Checkbox Object.
//#define USE_RADIOBUTTON           // Enable Radio Button Object.
//#define USE_EDITBOX               // Enable Edit Box Object.
//#define USE_LISTBOX               // Enable List Box Object.
//#define USE_SLIDER            // Enable Slider or Scroll Bar Object.

//#define USE_PROGRESSBAR           // Enable Progress Bar Object.
//#define USE_STATICTEXT            // Enable Static Text Object.
//#define USE_PICTURE               // Enable Picture Object.
//#define USE_GROUPBOX              // Enable Group Box Object.
//#define USE_ROUNDDIAL             // Enable Dial Object.
```

```
//#define USE_METER                       // Enable Meter Object.
//#define USE_TEXTENTRY             // Enable TextEntry Object.
//#define USE_CUSTOM            // Enable Custom Control Object (an
example to create customized Object).

/*************************************************************
* Overview: To enable support for unicode fonts, USE_MULTIBYTECHAR
*                 must be defined. This changes XCHAR definition. See
XCHAR
*                 for details.
*
*************************************************************/

//#define USE_MULTIBYTECHAR

/*************************************************************
* Overview: Font data can be placed in two locations. One is in
*                 FLASH memory and the other is from external memory.
*                 Defininig one or both enables the support for fonts
located
*                 in internal flash and external memory.
*     - USE_FONT_FLASH - Font in internal flash memory support.
*     - USE_FONT_EXTERNAL - Font in external memory support.
*
*************************************************************/
#define USE_FONT_FLASH          // Support for fonts located in
internal flash

//#define USE_FONT_EXTERNAL         // Support for fonts located in
external memory

/*************************************************************
* Overview: Similar to Font data bitmaps can also be placed in
*                 two locations. One is in FLASH memory and the other
is
*                 from external memory.
*                 Defininig one or both enables the support for
bitmaps located
*                 in internal flash and external memory.
*     - USE_BITMAP_FLASH - Font in internal flash memory support.
*     - USE_BITMAP_EXTERNAL - Font in external memory support.
*
*************************************************************/
#define USE_BITMAP_FLASH        // Support for bitmaps located in
internal flash

//#define USE_BITMAP_EXTERNAL       // Support for bitmaps located in
external memory

/*************************************************************
* Overview: Define the malloc() and free() for versatility on OS
*           based systems.
*
*************************************************************/
```

```
      #define GFX_malloc(size)      malloc(size)
      #define GFX_free(pObj)            free(pObj)

// GDD_GraphicsConfig.h should be included as the last line in this
file (GraphicsConfig.h).
// The file checks if a certain definition used by GDD has been defined
yet or not,
// if not, it will define it. This allows end users to manually
enable/disable other definitions.
#include "GDD_GraphicsConfig.h"

#endif // _GRAPHICSCONFIG_H
```

```
/********************************************************************
* Company: Baylor University
* Engineer: Brandon Herrera
* File Name:  HardwareProfile_CONTROL_BOARD.h
* Project Name:  PFTNA
* Target Device:       Control Board Rev A
* Compiler: C32
* Dependencies: M.A.L.
* Description: This is the main oscilloscope like user interface for
*                    the Pulsed Fourier Transform Network Analyzer.
*
********************************************************************
* Date         Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* 6/6/11     File Created
* 7/9/11    Initial Release
********************************************************************/
#ifndef __HARDWARE_PROFILE_H
    #define __HARDWARE_PROFILE_H

/********************************************************************
* GetSystemClock() returns system clock frequency.
*
* GetPeripheralClock() returns peripheral clock frequency.
*
* GetInstructionClock() returns instruction clock frequency.
*
********************************************************************/

/********************************************************************
* Macro: #define  GetSystemClock()
*
* Overview: This macro returns the system clock frequency in Hertz.
*                 * value is 8 MHz/2 x 18 PLL for PIC32
*
********************************************************************/
        #define GetSystemClock()    (80000000ul)


/********************************************************************
* Macro: #define  GetPeripheralClock()
*
* Overview: This macro returns the peripheral clock frequency
*                 used in Hertz.
*                 * value for PIC32 is
<PRE>(GetSystemClock()/(1<<OSCCONbits.PBDIV)) </PRE>
*
********************************************************************/
        #define GetPeripheralClock()    (GetSystemClock() / (1 <<
OSCCONbits.PBDIV))
```

```
/*********************************************************************
* Macro: #define  GetInstructionClock()
*
* Overview: This macro returns instruction clock frequency
*                 used in Hertz.
*                 * value for PIC24 is <PRE>(GetSystemClock()/2) </PRE>
*                 * value for PIC32 is <PRE>(GetSystemClock()) </PRE>
*
*********************************************************************/
        #define GetInstructionClock()   (GetSystemClock())


/*********************************************************************
* PIC Device Specific includes
*********************************************************************/
#ifdef __PIC32MX
      #include <p32xxxx.h>
#endif

/*********************************************************************
* UART SETTINGS
*********************************************************************/
      #define BAUDRATE2               115200UL
    #define BRG_DIV2                4
    #define BRGH2                   1


//Defines for microchip specific code
#define MULTI_MEDIA_BOARD_DM00123
#define USE_16BIT_PMP
#define PIC32_USB_SK_DM320003_1
#define DISPLAY_CONTROLLER SSD1926
#define DISPLAY_PANEL TFT_G240320LTSW_118W_E
#define COLOR_DEPTH 16
//End Auto Generated Code

#define USE_GFX_PMP



/*********************************************************************
* GRAPHICS SETTINGS
*********************************************************************/
#define DISP_ORIENTATION            90
#define DISP_HOR_RESOLUTION         240
#define DISP_VER_RESOLUTION         320
#define DISP_DATA_WIDTH             18
#define DISP_INV_LSHIFT
#define DISP_HOR_PULSE_WIDTH    25
#define DISP_HOR_BACK_PORCH     5
#define DISP_HOR_FRONT_PORCH    10
#define DISP_VER_PULSE_WIDTH    4
```

```
#define DISP_VER_BACK_PORCH     0
#define DISP_VER_FRONT_PORCH    2

 /*******
 * PIC32 will use the SET and CLR SFRs for the I/O.
 * These are atomic settings that are recommended when
 * modifying SFRs
 ********/
// Definitions for reset pin
#define DisplayResetConfig()        TRISACLR = _TRISA_TRISA10_MASK
#define DisplayResetEnable()        LATACLR = _LATA_LATA10_MASK
#define DisplayResetDisable()       LATASET = _LATA_LATA10_MASK

// Definitions for RS pin
#define DisplayCmdDataConfig()      AD1PCFGSET = _AD1PCFG_PCFG10_MASK,
TRISBCLR = _TRISB_TRISB10_MASK
#define DisplaySetCommand()         LATBCLR = _LATB_LATB10_MASK
#define DisplaySetData()            LATBSET = _LATB_LATB10_MASK

// Definitions for CS pin
#define DisplayConfig()             TRISGCLR = _TRISG_TRISG13_MASK
#define DisplayEnable()             LATGCLR = _LATG_LATG13_MASK
#define DisplayDisable()            LATGSET = _LATG_LATG13_MASK

// Definitions for FLASH CS pin
#define DisplayFlashConfig()
#define DisplayFlashEnable()
#define DisplayFlashDisable()

// Definitions for POWER ON pin
#define DisplayPowerConfig()
#define DisplayPowerOn()
#define DisplayPowerOff()

/*********************************************************************
* DDS SETTINGS
*********************************************************************/

#define DDS_CS_TRIS   TRISGbits.TRISG9   //SS1
#define DDS_CS_LAT    LATGbits.LATG9    //SS1
#define DDS_SCK_TRIS  TRISGbits.TRISG6
#define DDS_SDO_TRIS  TRISGbits.TRISG8
#define DDS_UPDATE_TRIS  TRISFbits.TRISF4
#define DDS_UPDATE_LAT  LATFbits.LATF4

#define DDS_RST_LAT LATAbits.LATA6
#define DDS_RST_TRIS TRISAbits.TRISA6


/*********************************************************************
* FLASH ETTINGS
*********************************************************************/
#define USE_SST25_SPI1
#define SST25_CS_TRIS   TRISBbits.TRISB2   //SS1
```

```
#define SST25_CS_LAT    LATBbits.LATB2    //SS1
#define SST25_SCK_TRIS  TRISDbits.TRISD10
#define SST25_SDO_TRIS  TRISDbits.TRISD0
#define SST25_SDI_TRIS  TRISCbits.TRISC4

/*********************************************************************
* DSA ETTINGS
*********************************************************************/

#define DSA_C1_TRIS   TRISEbits.TRISE8
#define DSA_C1_LAT    LATEbits.LATE8

#define DSA_C2_TRIS   TRISFbits.TRISF2
#define DSA_C2_LAT    LATFbits.LATF2

#define DSA_C4_TRIS   TRISFbits.TRISF8
#define DSA_C4_LAT    LATFbits.LATF8

#define DSA_C8_TRIS   TRISDbits.TRISD15
#define DSA_C8_LAT    LATDbits.LATD15

#define DSA_C16_TRIS   TRISDbits.TRISD14
#define DSA_C16_LAT    LATDbits.LATD14

#define Switch_TX_TRIS   TRISFbits.TRISF13
#define Switch_TX_LAT    LATFbits.LATF13

#define Switch_RX_TRIS   TRISFbits.TRISF12
#define Switch_RX_LAT    LATFbits.LATF12


            #define       USE_XC2C64A
                  #define       USE_SST25VF016                    //
use the SPI Flash
       #define       USE_RESISTIVE_TOUCH              // use the
resistive touch

      #define I2C_CLOCK_FREQ 50000

/*********************************************************************
* TOUCHSCREEN ETTINGS
*********************************************************************/

// ADC Status
#define TOUCH_ADC_DONE              AD1CON1bits.DONE

#define ADC_XPOS    ADC_CH0_POS_SAMPLEA_AN11
#define ADC_YPOS    ADC_CH0_POS_SAMPLEA_AN14

// X port definitions
#define ADPCFG_XPOS AD1PCFGbits.PCFG11
#define LAT_XPOS    LATBbits.LATB11
#define TRIS_XPOS   TRISBbits.TRISB11
#define LAT_XNEG    LATBbits.LATB13
```

```
#define TRIS_XNEG   AD1PCFGbits.PCFG13 = 1, TRISBbits.TRISB13

// Y port definitions
#define ADPCFG_YPOS AD1PCFGbits.PCFG14
#define LAT_YPOS    LATBbits.LATB14
#define TRIS_YPOS   TRISBbits.TRISB14
#define LAT_YNEG    LATBbits.LATB12
#define TRIS_YNEG   AD1PCFGbits.PCFG12 = 1, TRISBbits.TRISB12

/*********************************************************************
* BUTTON ETTINGS
*********************************************************************/

      typedef enum
{
    HW_BUTTON_PRESS = 0,
    HW_BUTTON_RELEASE = 1
}HW_BUTTON_STATE;

#define HardwareButtonInit()        (CNPUESET = _CNPUE_CNPUE16_MASK |
_CNPUE_CNPUE15_MASK | _CNPUE_CNPUE19_MASK)
#define GetHWButtonTouchCal()       (PORTDbits.RD6)
#define GetHWButtonProgram()        (HW_BUTTON_RELEASE)
#define GetHWButtonScanDown()       (HW_BUTTON_RELEASE)
#define GetHWButtonScanUp()         (HW_BUTTON_RELEASE)
#define GetHWButtonCR()             (PORTDbits.RD6)
#define GetHWButtonFocus()          (PORTDbits.RD7 & PORTDbits.RD13)

#define GetHWButton1()             (PORTDbits.RD6)
#define GetHWButton2()             (PORTDbits.RD7)
#define GetHWButton3()             (PORTDbits.RD13)

/*********************************************************************
* SWITCH ETTINGS
*********************************************************************/
#define TX_TRIS TRISFbits.TRISF5
#define RX_TRIS TRISFbits.TRISF4


/*********************************************************************
* RTCC DEFAULT INITIALIZATION (these are values to initialize the RTCC
*********************************************************************/
#define RTCC_DEFAULT_DAY        15      // 15th
#define RTCC_DEFAULT_MONTH      10      // October
#define RTCC_DEFAULT_YEAR       10      // 2010
#define RTCC_DEFAULT_WEEKDAY    05      // Friday
#define RTCC_DEFAULT_HOUR       10      // 10:10:01
#define RTCC_DEFAULT_MINUTE     10
#define RTCC_DEFAULT_SECOND     01



#include <plib.h>
```

```
/*********************************************************************
* Configuration for the CPLD
*********************************************************************/
#ifdef USE_16BIT_PMP
#define GRAPHICS_HW_CONFIG     CPLD_GFX_CONFIG_16BIT
#else
#define GRAPHICS_HW_CONFIG     CPLD_GFX_CONFIG_8BIT
#endif

/*********************************************************************
* PIC32 LEDs
*********************************************************************/
typedef enum
{
    LED_1,
    LED_2,
    LED_3,
}MMB_LED;

extern inline void __attribute__((always_inline)) SetLEDDirection(void)
{
   PORTSetPinsDigitalOut(IOPORT_D, (BIT_0 | BIT_1 | BIT_2));

}

extern inline void __attribute__((always_inline)) TurnLEDOn(MMB_LED
led)
{
    if(led == LED_1)
      PORTSetBits(IOPORT_D, BIT_0);

    if(led == LED_2)
      PORTSetBits(IOPORT_D, BIT_1);

    if(led == LED_3)
      PORTSetBits(IOPORT_D, BIT_2);


}

extern inline void __attribute__((always_inline)) TurnLEDOff(MMB_LED
led)
{
    if(led == LED_1)
      PORTClearBits(IOPORT_D, BIT_0);

    if(led == LED_2)
      PORTClearBits(IOPORT_D, BIT_1);

    if(led == LED_3)
      PORTClearBits(IOPORT_D, BIT_2);

}
```

```
extern inline void __attribute__((always_inline)) ToggleLED(MMB_LED
led)
{
    if(led == LED_1)
      PORTToggleBits(IOPORT_D, BIT_0);

    if(led == LED_2)
      PORTToggleBits(IOPORT_D, BIT_1);

    if(led == LED_3)
      PORTToggleBits(IOPORT_D, BIT_2);

}

extern inline void __attribute__((always_inline)) TurnLEDAllOn(void)
{
      PORTSetBits(IOPORT_D, BIT_0);
      PORTSetBits(IOPORT_D, BIT_1);
      PORTSetBits(IOPORT_D, BIT_2);

}

extern inline void __attribute__((always_inline)) TurnLEDAllOff(void)
{
      PORTClearBits(IOPORT_D, BIT_0);
      PORTClearBits(IOPORT_D, BIT_1);
      PORTClearBits(IOPORT_D, BIT_2);

}

#endif // __HARDWARE_PROFILE_H
```

```
/*****************************************************************
* Company: Baylor University
* Engineer: Brandon Herrera
* File Name:  ISL95810.h
* Project Name:  PFTNA
* Target Device:       Control Board Rev A
* Compiler: C32
* Dependencies: M.A.L.
* Description: This is the main oscilloscope like user interface for
*                    the Pulsed Fourier Transform Network Analyzer.
*
*****************************************************************
* Date         Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* 6/6/11     File Created
* 7/9/11    Initial Release
*****************************************************************/
#ifndef _ISL95810_H
#define _ISL958104_H

#include <plib.h>
#include "TimeDelay.h"
#include "HardwareProfile.h"
#include "GenericTypeDefs.h"

// I2C address for the POT
#define POTAddress 0b0101000

// Commands for memory type to write to
#define Non_Volatile 0x00
#define Volatile_Only 0x80

/*****************************************************************
* Marco: POTStop()
*
* Overview: this macro closes the I2C channel
*
* Input: none
*
* Output: none
*
*****************************************************************/
#define POTStop() I2CStop(I2C2);

/*****************************************************************
* Function: void PotPut(BYTE data)
*
* Overview: this function writes a byte of data to the potentiometer
*
* Input: byte to be written
```

```
*
* Output: none
*
*********************************************************************/
void POTPut(BYTE data);

/*********************************************************************
* Function: void PotInit(void)
*
* Overview: this function initiates the potentiometer
*
* Input: write to Volitile_Only or Non_Volitile memory
*
* Output: none
*
*********************************************************************/
void POTInit(BYTE memory);

/*********************************************************************
* Function: void POTProgram(BYTE data)
*
* Overview: this function programs the potentiometer
*
* Input: 8 bit setting for the POT
*
* Output: none
*
*********************************************************************/
void POTProgram(BYTE data);

#endif //_ISL958104_H
```

```
/*****************************************************************
* Company: Baylor University
* Engineer: Brandon Herrera
* File Name:  ISL95810.h
* Project Name:  PFTNA
* Target Device:       Control Board Rev A
* Compiler: C32
* Dependencies: M.A.L.
* Description: This is the main oscilloscope like user interface for
*                    the Pulsed Fourier Transform Network Analyzer.
*
*****************************************************************
* Date         Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* 6/6/11     File Created
* 7/9/11    Initial Release
*****************************************************************/
#include "ISL95810.h"


/*****************************************************************
* Function: void PotPut(BYTE data)
*
* Overview: this function writes a byte of data to the potentiometer
*
* Input: byte to be written
*
* Output: none
*
*****************************************************************/
void POTPut(BYTE data)
{
      BYTE result;
      while(!I2CTransmitterIsReady(I2C2));

            result = I2CSendByte(I2C2, data);


       if(result != I2C_SUCCESS)
       {
           DBPRINTF("Error Byte 1 %d\n", result);
       }

      while (!I2CTransmissionHasCompleted(I2C2));

          if (!I2CByteWasAcknowledged(I2C2))
          {
              DBPRINTF("No ack");
          }
```

```
}

/*********************************************************************
* Function: void PotInit(void)
*
* Overview: this function initiates the potentiometer
*
* Input: write to Volitile_Only or Non_Volitile memory
*
* Output: none
*
*********************************************************************/
void POTInit(BYTE memory)
{
      BYTE result;

    if (I2CBusIsIdle(I2C2))
    {
        result = I2CStart(I2C2);
    }

    if(result != I2C_SUCCESS)
    {
        DBPRINTF("Error Start %d\n", result);
    }
      Delay10us(5);

      POTPut((POTAddress << 1)|I2C_WRITE);

      POTPut(0x02);

      POTPut(memory);

      POTStop();

      DelayMs(1);

}

/*********************************************************************
* Function: void POTProgram(BYTE data)
*
* Overview: this function programs the potentiometer
*
* Input: 8 bit setting for the POT
*
* Output: none
*
*********************************************************************/
void POTProgram(BYTE data)
{
      BYTE result;

      DisableIntT5;
```

```
      DisableIntT1;

    if (I2CBusIsIdle(I2C2))
    {
        result = I2CStart(I2C2);
    }

    if(result != I2C_SUCCESS)
    {
        DBPRINTF("Error Start %d\n", result);
    }
Delay10us(5);

POTPut((POTAddress << 1)|I2C_WRITE);

POTPut(0x00);

POTPut(data);

POTStop();

DelayMs(1);

EnableIntT5;
EnableIntT1;

}
```

```
/*********************************************************************
* Company: Baylor University
* Engineer: Brandon Herrera
* File Name:  Main.h
* Project Name:  PFTNA
* Target Device:       Control Board Rev A
* Compiler: C32
* Dependencies: M.A.L.
* Description: This is the main oscilloscope like user interface for
*                    the Pulsed Fourier Transform Network Analyzer.
*
*********************************************************************
* Date         Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* 6/6/11     File Created
* 7/9/11    Initial Release
*********************************************************************/
#ifndef _MAIN_H_
    #define _MAIN_H_

//////////////////////////////// INCLUDES ////////////////////////////////

    #include <plib.h>
    #include "GenericTypeDefs.h"
    #include "Graphics/Graphics.h"
    #include "SST25VF032.h"
    #include "TouchScreen.h"
      #include "AD9958.h"
      #include "DAC7574.h"
      #include "ISL95810.h"
      #include "DSP.h"


////////////////////////////////////////////////////////////////////////
//                                STATES
////////////////////////////////////////////////////////////////////////
typedef enum
{
    CREATE_STARTUP    = 0,
    DISPLAY_STARTUP,
    CREATE_MAIN,
    DISPLAY_MAIN,
    CREATE_LETTERINPUT,
      DISPLAY_LETTERINPUT,
      CREATE_OPTIONS,
      DISPLAY_OPTIONS,
      CREATE_UISMALL,
      DISPLAY_UISMALL,
      UPDATE_GRAPH,
} SCREEN_STATES;
```

```
// State Prototypes

WORD    StartupMsgCallback(WORD objMsg, OBJ_HEADER *pObj, GOL_MSG
*pMsg);

WORD    MainMsgCallback(WORD objMsg, OBJ_HEADER *pObj, GOL_MSG *pMsg);

//WORD    NumberInputMsgCallback(WORD objMsg, OBJ_HEADER *pObj, GOL_MSG
*pMsg);

WORD    LetterInputMsgCallback(WORD objMsg, OBJ_HEADER *pObj, GOL_MSG
*pMsg);

WORD    OptionsMsgCallback(WORD objMsg, OBJ_HEADER *pObj, GOL_MSG
*pMsg);

WORD MainDrawCallback();

//#define N 1024

#endif
```

```
/********************************************************************
* Company: Baylor University
* Engineer: Brandon Herrera
* File Name:  Main.c
* Project Name:  PFTNA
* Target Device:       Control Board Rev A
* Compiler: C32
* Dependencies: M.A.L.
* Description: This is the main oscilloscope like user interface for
*                    the Pulsed Fourier Transform Network Analyzer.
*
********************************************************************
* Date         Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* 6/6/11     File Created
* 7/9/11    Initial Release
********************************************************************/
#include "Main.h"
#include "HardwareProfile.h"
#include "HardwareProfile2.h"
#include "TouchScreen.h"
#include "GDD_Screens.h"
#include "USB/usb.h"
#include "USB/usb_host_msd.h"
#include "USB/usb_host_msd_scsi.h"
#include "MDD File System/FSIO.h"
#include "AD9958.h"
//#include "db_utils.h"


// Configuration bits
#pragma config UPLLEN = ON          // USB PLL Enabled
#pragma config FPLLMUL = MUL_20     // PLL Multiplier
#pragma config UPLLIDIV = DIV_2     // USB PLL Input Divider
#pragma config FPLLIDIV = DIV_2     // PLL Input Divider
#pragma config FPLLODIV = DIV_1     // PLL Output Divider
#pragma config FPBDIV = DIV_1       // Peripheral Clock divisor
#pragma config FWDTEN = OFF         // Watchdog Timer
#pragma config WDTPS = PS1          // Watchdog Timer Postscale
#pragma config FCKSM = CSDCMD       // Clock Switching & Fail Safe
Clock Monitor
#pragma config OSCIOFNC = OFF       // CLKO Enable
#pragma config POSCMOD = HS         // Primary Oscillator
#pragma config IESO = OFF           // Internal/External Switch-over
#pragma config FSOSCEN = ON         // Secondary Oscillator Enable (KLO
was off)
#pragma config FNOSC = PRIPLL       // Oscillator Selection
#pragma config CP = OFF             // Code Protect
#pragma config BWP = OFF            // Boot Flash Write Protect
#pragma config PWP = OFF            // Program Flash Write Protect
```

```
#pragma config ICESEL = ICS_PGx2    // ICE/ICD Comm Channel Select


////////////////////////////////////////////////////////////////////////
//                            LOCAL PROTOTYPES
////////////////////////////////////////////////////////////////////////
void            TickInit(void);                 // starts tick counter
void            ADCSampleInit(void);
void              SetAttenuator(int);

////////////////////////////////////////////////////////////////////////
//                            GLOBAL VARIABLES
////////////////////////////////////////////////////////////////////////
XCHAR Input_Name_Text[15] = "Name";
XCHAR* pText_Input;
XCHAR DSAstr[2] = "0";
XCHAR DATAstr[30] = "0";
WORD DSA = 31;
FSFILE * myFile;
//BYTE myData[512];
size_t numBytes;
BYTE DCOFF = 0b10000000;  //AD for DC Offset
BYTE Gain = 51;   //Pot Setting for Gain
unsigned short Trig_Level = 350;

int N=4096;

unsigned short cir_buff[50];
short test_data[4096];
double fout[2048];
double fcal[2048];
int trigger=0;
int averaging = 16;
BOOL UseCal = 0;
BOOL Save_Data = 0;

      //int32c din[4096];
      //int32c dout[4096];
      //int32c scratch[4096];

volatile BOOL deviceAttached;

WORD temp = 0;

//SCREEN_STATES screenState = CREATE_STARTUP; // current state of
screen state machine
//SCREEN_STATES screenStateReturn = CREATE_STARTUP;

SCREEN_STATES screenState = CREATE_UISMALL; // current state of screen
state machine
SCREEN_STATES screenStateReturn = CREATE_UISMALL;

/* */
int main(void)
```

```
{

    INTEnableSystemMultiVectoredInt();
    SYSTEMConfigPerformance(GetSystemClock());
//SYSTEMConfig(80000000uL, SYS_CFG_WAIT_STATES | SYS_CFG_PCACHE);



      DBINIT();
      DBPRINTF("hello");


// configure for multi-vectored mode
INTConfigureSystem(INT_SYSTEM_CONFIG_MULT_VECTOR);
INTClearFreeze();
// enable interrupts
INTEnableInterrupts();

INTSetVectorPriority( INT_CORE_TIMER_VECTOR,
                      INT_PRIORITY_LEVEL_5);



      DSA_C1_TRIS = 0;
      DSA_C1_LAT = 1;

      DSA_C2_TRIS = 0;
      DSA_C2_LAT = 1;

      DSA_C4_TRIS = 0;
      DSA_C4_LAT = 1;

      DSA_C8_TRIS = 0;
      DSA_C8_LAT = 1;

      DSA_C16_TRIS = 0;
      DSA_C16_LAT = 1;

      Switch_TX_TRIS = 0;
      Switch_TX_LAT = 0;

      Switch_RX_TRIS = 0;
      Switch_RX_LAT = 1;


    GOL_MSG msg;                    // GOL message structure to
interact with GOL

    Nop();

    ////////////////////////////////////////////////////////////////////
```

```
    GOLInit();                      // initialize graphics library &
                                    // create default style scheme for
GOL
          #if defined (USE_SST25VF016)
            SST25Init();            // initialize GFX3 SST25 flash SPI
        #endif

    TouchInit();                    // initialize touch screen
    HardwareButtonInit();           // Initialize the hardware buttons
      DDSInit();
      DACInit();
      POTInit(Volatile_Only);
      TickInit();

      DDSProgram();
      //DACProgram(Channel_A,0b01100100,0b01100000);
      //DACProgram(Channel_A,0b01100100,0b11110000);
      DACProgram(Channel_A,0b11111111,0b11110000);
      DACProgram(Channel_B,0b01110000,0b10000000);
      DACProgram(Channel_D,0b10000000,0b00000000);

      POTProgram(Gain);
      ADCSampleInit();

SetLEDDirection();
//DBPRINTF("while");
int i=0;

/*
while(GetHWButton3() != HW_BUTTON_PRESS)
//while(1)
{
if (GetHWButton1() == HW_BUTTON_PRESS)
{
      //DBPRINTF("%d",GetReg(0x00));
//    SST25WriteWord(GRAPHICS_LIBRARY_VERSION, ADDRESS_VERSION);
//    temp=GetReg(0x00);
//    DBPRINTF("%d",SST25ReadWord(ADDRESS_VERSION));
 //         if(0x01 == temp)
//        if(GRAPHICS_LIBRARY_VERSION ==
SST25ReadWord(ADDRESS_VERSION))
//        {
//                ToggleLED(LED_3);
 //       }

//TRISAbits.TRISA10 = 0;
DelayMs(500);

DDSProgram();

DelayMs(5);

}
```

```
if (GetHWButton2() == HW_BUTTON_PRESS)
{
//DBPRINTF("%d",temp);
 while(GetHWButton1() != HW_BUTTON_PRESS)
{
DDS_CS_LAT = 0;
   DDS_UPDATE_LAT = 1;
Delay10us(1);
    DDS_UPDATE_LAT = 0;
DelayMs(9);
}
}

if (GetHWButton3() == HW_BUTTON_PRESS)
{
      TakeData();
      DBPRINTF("done");
// while(GetHWButton1() != HW_BUTTON_PRESS)
//{
//DDSSet();
//DelayMs(1);
//DDSProgram();
//DelayMs(20);
//}
}
// DBPRINTF("b");
//GetReg(0x00);
//ToggleLED(LED_3);


}

*/
      //DBPRINTF("%d",GetReg(0x00));


           // Force a touchscreen calibration by pressing the switch

          if(GetHWButton3() == HW_BUTTON_PRESS)
          {
              TouchCalibration();
              TouchStoreCalibration();
          }



        if(GRAPHICS_LIBRARY_VERSION != SST25ReadWord(ADDRESS_VERSION))
        {
            TouchCalibration();
            TouchStoreCalibration();
        }
```

```
      deviceAttached = FALSE;

    //Initialize the stack
    USBInitialize(0);

    // Load touch screen calibration parameters from memory
    TouchLoadCalibration();

    GDDDemoCreateFirstScreen();

    while(1)
    {
            USBTasks();

        if(GOLDraw())               // Draw GOL object
        {
            TouchGetMsg(&msg);      // Get message from touch screen
           GOLMsg(&msg);           // Process message


      if(Save_Data)
      {

            DisableIntT5;
            DisableIntT1;

        if(USBHostMSDSCSIMediaDetect())
        {
            deviceAttached = TRUE;

            //now a device is attached
            //See if the device is attached and in the right format
            if(FSInit())
            {
                //Opening a file in mode "w" will create the file if it
doesn't
                //  exist.  If the file does exist it will delete the
old file
                //  and create a new one that is blank.
                myFile = FSfopen("data.csv","w");

                        for (i=0;i<4096;i++)
                        {
                        sprintf(DATAstr, "%d,", test_data[i]);
//Convert to string

                //Write some data to the new file.
                FSfwrite(DATAstr,1,5,myFile);
                  }

                //Always make sure to close the file so that the data
gets
                //  written to the drive.
```

```
                FSfclose(myFile);
                        Save_Data=0;
                //Just sit here until the device is removed.
                //while(deviceAttached == TRUE)
                //{
                //    USBTasks();
                //}
            }
        }

            //end demo code

            EnableIntT5;
            EnableIntT1;

      }
        }
    }//end while
}

/////////////////////////////////////////////////////////////////////////////
//////
// Function: WORD GOLMsgCallback(WORD objMsg, OBJ_HEADER* pObj,
GOL_MSG* pMsg)
// Input: objMsg - translated message for the object,
//        pObj - pointer to the object,
//        pMsg - pointer to the non-translated, raw GOL message
// Output: if the function returns non-zero the message will be
processed by default
// Overview: it's a user defined function. GOLMsg() function calls it
each

//           time the valid message for the object received
/////////////////////////////////////////////////////////////////////////////
//////
WORD GOLMsgCallback(WORD objMsg, OBJ_HEADER *pObj, GOL_MSG *pMsg)
{
    WORD    objectID;

    objectID = GetObjID(pObj);

    GDDDemoGOLMsgCallback(objMsg, pObj, pMsg);

    // Add additional code here...
    switch(screenState)
    {
        case DISPLAY_STARTUP:
            return (StartupMsgCallback(objMsg, pObj, pMsg));

        case DISPLAY_MAIN:
            return (MainMsgCallback(objMsg, pObj, pMsg));

        //case DISPLAY_NUMBERINPUT:
        //    return (NumberInputMsgCallback(objMsg, pObj, pMsg));
```

```
        case DISPLAY_LETTERINPUT:
            return (LetterInputMsgCallback(objMsg, pObj, pMsg));

            case DISPLAY_OPTIONS:
                  return (OptionsMsgCallback(objMsg, pObj, pMsg));

            case DISPLAY_UISMALL:
                  return (UISmallMsgCallback(objMsg, pObj, pMsg));

        default:
            return (1); // process message by default
    }

    return (1);
}

////////////////////////////////////////////////////////////////////
// Function: WORD GOLDrawCallback()
// Output: if the function returns non-zero the draw control will be
passed to GOL
// Overview: it's a user defined function. GOLDraw() function calls it
each
//           time when GOL objects drawing is completed. User drawing
should be done here.
//           GOL will not change color, line type and clipping region
settings while

//           this function returns zero.
////////////////////////////////////////////////////////////////////
WORD GOLDrawCallback(void)
{
      EDITBOX* pEb;
      XCHAR* pChar;
      SLIDER* pSlider;

    GDDDemoGOLDrawCallback();

    // Add additional code here...
switch(screenState)
    {
        case CREATE_STARTUP:
            CreateStartup();                    // create selection
screen
            screenState = DISPLAY_STARTUP;     // switch to next state
            return (1);                              // draw objects
created

        case DISPLAY_STARTUP:
            //StartupDrawCallback();
            return (1);                              // redraw objects
if needed

        case CREATE_MAIN:
```

```
            CreateMain();
                  // Set the correct name


                  pEb = (EDITBOX*)GOLFindObject(Main_OBJ_EDITBOX_Name);
                  EbSetText(pEb,(CHAR*)Input_Name_Text);
                  SetState(pEb, EB_DRAW);

            screenState = DISPLAY_MAIN;         // switch to next state
            return (1);

        case DISPLAY_MAIN:
            MainDrawCallback();
            return (1);

        //case CREATE_NUMBERINPUT:
            //CreateNumberInput();
            //screenState = DISPLAY_NUMBERINPUT;  // switch to next
state
            //return (1);

        //case DISPLAY_NUMBERINPUT:
            //NumberInputDrawCallback();
            //return (1);

        case CREATE_LETTERINPUT:
            CreateLetterInput();
            screenState = DISPLAY_LETTERINPUT;
            return (1);

        case DISPLAY_LETTERINPUT:
           // LetterInputDrawCallback();
            return (1);

            case CREATE_OPTIONS:
                  CreateOptions();

                  pSlider =
(SLIDER*)GOLFindObject(Options_OBJ_SLIDER_DSA);
                  pEb =
(EDITBOX*)GOLFindObject(Options_OBJ_EDITBOX_DSA);
                  SldSetPos(pSlider,DSA);
                  EbSetText(pEb,(XCHAR*)DSAstr);   //Write to text box
                  SetState(pEb, EB_DRAW);
                  SetState(pSlider, SLD_DRAW);

                  screenState = DISPLAY_OPTIONS;
                  return (1);

            case DISPLAY_OPTIONS:
                  return (1);

            case CREATE_UISMALL:
                  CreateUISmall();
```

```
                    screenState = DISPLAY_UISMALL;

                case UPDATE_GRAPH:
                    DrawGraph();
                    screenState = DISPLAY_UISMALL;
                    return (1);

                case DISPLAY_UISMALL:
                    return (1);

            default:
                break;
        }

        return (1);
}

WORD StartupMsgCallback(WORD objMsg, OBJ_HEADER *pObj, GOL_MSG *pMsg)
{
      screenState= CREATE_MAIN;

}

WORD MainMsgCallback(WORD objMsg, OBJ_HEADER *pObj, GOL_MSG *pMsg)
{
      if(objMsg == BTN_MSG_RELEASED)
      {
            switch(GetObjID(pObj))
            {
                  case Main_OBJ_BUTTON_Options:
                        screenState = CREATE_OPTIONS;
                        return 1;

                  case Main_OBJ_BUTTON_Data:
                        TakeData();
                        screenState = CREATE_UISMALL;
                        return 1;

                  case Main_OBJ_EDITBOX_Name:
                        screenState = CREATE_LETTERINPUT;
                        screenStateReturn = CREATE_MAIN;
                        pText_Input = (CHAR*)Input_Name_Text;
                        return 1;

                  case Main_OBJ_EDITBOX_AGV:
                        //screenState = CREATE_NUMBERINPUT;
                        return 1;
                  default:
                        return 1;
            }
      }
}

WORD MainDrawCallback()
```

```
{

}

//WORD NumberInputMsgCallback(WORD objMsg, OBJ_HEADER *pObj, GOL_MSG
*pMsg)
//{
//    screenState= CREATE_MAIN;
//
//}

WORD LetterInputMsgCallback(WORD objMsg, OBJ_HEADER *pObj, GOL_MSG
*pMsg)
{
      EDITBOX* pEb;
      XCHAR* pChar;

      pEb = (EDITBOX*)GOLFindObject(LetterInput_Input_Display);

      if(objMsg == BTN_MSG_RELEASED)
      {
            switch(GetObjID(pObj))
            {
                  case LetterInput_OBJ_BUTTON_Del:
                        if (strlen(pText_Input)>0)
                        {
                              Input_Name_Text[(strlen(pText_Input)-1)]
= '\0';
                        }
                        EbSetText(pEb,pText_Input);
                        SetState(pEb, EB_DRAW);
                        return 1;

                  case LetterInput_OBJ_BUTTON_Enter:
                        screenState = screenStateReturn;
                        return 1;

                  default:
                        //EbAddChar(pEb,'Q');

                        pChar = BtnGetText(pObj);
                        strcat(pText_Input,pChar);
                        //EbSetText(pEb,pChar);
                        EbSetText(pEb,pText_Input);
                        SetState(pEb, EB_DRAW);
                        return 1;


            }
      }
      //screenState= CREATE_MAIN;
}
```

```
WORD OptionsMsgCallback(WORD objMsg, OBJ_HEADER *pObj, GOL_MSG *pMsg)
{
      EDITBOX* pEb;
      SLIDER* pSlider;
      XCHAR* pChar;


      pEb = (EDITBOX*)GOLFindObject(Options_OBJ_EDITBOX_DSA);
      pSlider = (SLIDER*)GOLFindObject(Options_OBJ_SLIDER_DSA);


      switch(GetObjID(pObj))
      {
            case Options_OBJ_BUTTON_Back:
                  if(objMsg == BTN_MSG_RELEASED)
                  {
                        screenState = CREATE_MAIN;
                  }
                  return 1;

            case Options_OBJ_BUTTON_Time:
                  if(objMsg == BTN_MSG_RELEASED)
                  {
                        screenState = CREATE_MAIN;
                  }
                  return 1;

            case Options_OBJ_SLIDER_DSA:
                  DSA = (XCHAR)SldGetPos(pSlider);  //Get the position
                  SetAttenuator(DSA);
                  sprintf(DSAstr, "%d", DSA);      //Convert to string
                  EbSetText(pEb,(XCHAR*)DSAstr);   //Write to text box
                  SetState(pEb, EB_DRAW);

            default:
                  return 1;
      }


}

//WordLetterInputDrawCallback()

/*****************************************************************
* Function SetAttenuator(void)
*
* Overview:  this function sets the DSA
*
* Input: none
*
* Output: none
*
*****************************************************************/
void SetAttenuator(int attenation)
```

```
{
      DSA_C1_LAT = (attenation & 0b00001);
      DSA_C2_LAT = ((attenation & 0b00010) >> 1);
      DSA_C4_LAT = ((attenation & 0b00100) >> 2);
      DSA_C8_LAT = ((attenation & 0b01000) >> 3);
      DSA_C16_LAT = ((attenation & 0b10000) >> 4);
}

/*****************************************************************
  Function:
    BOOL USB_ApplicationEventHandler( BYTE address, USB_EVENT event,
                void *data, DWORD size )

  Summary:
    This is the application event handler.  It is called when the stack
has
    an event that needs to be handled by the application layer rather
than
    by the client driver.

  Description:
    This is the application event handler.  It is called when the stack
has
    an event that needs to be handled by the application layer rather
than
    by the client driver.  If the application is able to handle the
event, it
    returns TRUE.  Otherwise, it returns FALSE.

  Precondition:
    None

  Parameters:
    BYTE address    - Address of device where event occurred
    USB_EVENT event - Identifies the event that occured
    void *data      - Pointer to event-specific data
    DWORD size      - Size of the event-specific data

  Return Values:
    TRUE    - The event was handled
    FALSE   - The event was not handled

  Remarks:
    The application may also implement an event handling routine if it
    requires knowledge of events.  To do so, it must implement a
routine that
    matches this function signature and define the
USB_HOST_APP_EVENT_HANDLER
    macro as the name of that function.
  *****************************************************************/

BOOL USB_ApplicationEventHandler( BYTE address, USB_EVENT event, void
*data, DWORD size )
{
```

```
    switch( event )
    {
        case EVENT_VBUS_REQUEST_POWER:
            // The data pointer points to a byte that represents the
amount of power
            // requested in mA, divided by two.  If the device wants
too much power,
            // we reject it.
            return TRUE;

        case EVENT_VBUS_RELEASE_POWER:
            // Turn off Vbus power.
            // The PIC24F with the Explorer 16 cannot turn off Vbus
through software.

            //This means that the device was removed
            deviceAttached = FALSE;
            return TRUE;
            break;

        case EVENT_HUB_ATTACH:
            return TRUE;
            break;

        case EVENT_UNSUPPORTED_DEVICE:
            return TRUE;
            break;

        case EVENT_CANNOT_ENUMERATE:
            //UART2PrintString( "\r\n***** USB Error - cannot enumerate
device *****\r\n" );
            return TRUE;
            break;

        case EVENT_CLIENT_INIT_ERROR:
            //UART2PrintString( "\r\n***** USB Error - client driver
initialization error *****\r\n" );
            return TRUE;
            break;

        case EVENT_OUT_OF_MEMORY:
            //UART2PrintString( "\r\n***** USB Error - out of heap
memory *****\r\n" );
            return TRUE;
            break;

        case EVENT_UNSPECIFIED_ERROR:   // This should never be
generated.
            //UART2PrintString( "\r\n***** USB Error - unspecified
*****\r\n" );
            return TRUE;
            break;

        default:
```

```
            break;
    }

    return FALSE;
}


////////////////////////////////////////////////////////////////////////
// Function: Timer3 ISR
// Input: none
// Output: none
// Overview: increments tick counter. Tick is approx. 1 ms.
////////////////////////////////////////////////////////////////////////

#define __T1_ISR    __ISR(_TIMER_1_VECTOR, ipl3)
/* */
void __T1_ISR _T1Interrupt(void)
{
    // Clear flag
    mT1ClearIntFlag();

      TouchProcessTouch();
}


#define __T45_ISR    __ISR(_TIMER_45_VECTOR, ipl2)
/* */
void __T45_ISR _T45Interrupt(void)
{
    // Clear flag

    mT5ClearIntFlag();
    mT4ClearIntFlag();
      TakeData();
//    screenState = UPDATE_GRAPH;
DrawGraph();
ToggleLED(LED_1);
//    DisableIntT45;
//DisableIntT4;
//DisableIntT5;
}


#define __ADC_ISR    __ISR(_ADC_VECTOR, ipl5)
void __ADC_ISR _ADCInterrupt(void)
{


      mAD1ClearIntFlag();
}


////////////////////////////////////////////////////////////////////////
// Function: void TickInit(void)
```

```
// Input: none
// Output: none
// Overview: Initilizes the tick timer.
////////////////////////////////////////////////////////////////////////

/*********************************************************************
 * Section: Tick Delay
 ********************************************************************/
#define SAMPLE_PERIOD       500 // us
#define TICK_PERIOD                 (GetPeripheralClock() *
SAMPLE_PERIOD) / 5000000

/* */
void TickInit(void)
{

    // Initialize Timer1
    OpenTimer1(T1_ON | T1_PS_1_8, TICK_PERIOD);
    ConfigIntTimer1(T1_INT_ON | T1_INT_PRIOR_3);

}


//#define ADCSAMPLE_PERIOD       800000000 // us
//#define ADCTICK_PERIOD                  (GetPeripheralClock() *
ADCSAMPLE_PERIOD) / 156250
#define ADCTICK_PERIOD              100000
/* */
void ADCSampleInit(void)
{

    // Initialize Timer2
    OpenTimer45(T45_ON | T45_PS_1_256, ADCTICK_PERIOD);
    ConfigIntTimer45(T45_INT_ON | T45_INT_PRIOR_2);

}
```

```
/*****************************************************************************
 *
 * Basic access to SPI Flash SST25VF016 located on
 * Graphics LCD Controller PICtail Plus SSD1926 Board.
 *
 *****************************************************************************
 * FileName:        SST25VF016.h
 * Dependencies:    Graphics.h
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:          MPLAB C30, MPLAB C32
 * Linker:          MPLAB LINK30, MPLAB LINK32
 * Company:         Microchip Technology Incorporated
 *
 * Software License Agreement
 *
 * Copyright © 2008 Microchip Technology Inc.  All rights reserved.
 * Microchip licenses to you the right to use, modify, copy and distribute
 * Software only when embedded on a Microchip microcontroller or digital
 * signal controller, which is integrated into your product or third party
 * product (pursuant to the sublicense terms in the accompanying license
 * agreement).
 *
 * You should refer to the license agreement accompanying this Software
 * for additional information regarding your rights and obligations.
 *
 * SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
 * KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY
 * OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR
 * PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR
 * OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION,
 * BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT
 * DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL,
 * INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA,
 * COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY
 * CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
 * OR OTHER SIMILAR COSTS.
 *
 * Date             Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 01/07/09                 ...
 *****************************************************************************/
#ifndef _SST25VF016_H
    #define _SST25VF016_H

    #if defined(__dsPIC33F__)
        #include <p33Fxxxx.h>
    #elif defined(__PIC24H__)
        #include <p24Hxxxx.h>
    #elif defined(__PIC32MX__)
        #include <plib.h>
    #else
```

```
        #include <p24Fxxxx.h>
    #endif

      #include "GenericTypeDefs.h"
      #include "HardwareProfile.h"
      #include "TimeDelay.h"

/******************************************************************************
* SST25 SPI Channel
******************************************************************************
*/
#ifdef USE_SST25_SPI1

#define SST25_SPISTAT     SPI1STAT
#define SST25_SPISTATbits SPI1STATbits
#ifdef __PIC32MX__
#define SST25_SPICON      SPI1CON
#define SST25_SPICONbits  SPI1CONbits
#else
#define SST25_SPICON1     SPI1CON1
#define SST25_SPICON1bits SPI1CON1bits
#define SST25_SPICON2     SPI1CON2
#define SST25_SPICON2bits SPI1CON2bits
#endif
#define SST25_SPIBRG      SPI1BRG
#define SST25_SPIBUF      SPI1BUF

#elif defined (USE_SST25_SPI2)

#define SST25_SPISTAT     SPI2STAT
#define SST25_SPISTATbits SPI2STATbits
#ifdef __PIC32MX__
#define SST25_SPICON      SPI2CON
#define SST25_SPICONbits  SPI2CONbits
#else
#define SST25_SPICON1     SPI2CON1
#define SST25_SPICON1bits SPI2CON1bits
#define SST25_SPICON2     SPI2CON2
#define SST25_SPICON2bits SPI2CON2bits
#endif
#define SST25_SPIBRG      SPI2BRG
#define SST25_SPIBUF      SPI2BUF

#elif defined (USE_SST25_SPI1A)

#define SST25_SPISTAT     SPI1ASTAT
#define SST25_SPISTATbits SPI1ASTATbits
#ifdef __PIC32MX__
#define SST25_SPICON      SPI1ACON
#define SST25_SPICONbits  SPI1ACONbits
#else
#define SST25_SPICON1     SPI1ACON1
#define SST25_SPICON1bits SPI1ACON1bits
#define SST25_SPICON2     SPI1ACON2
```

```
#define SST25_SPICON2bits SPI1ACON2bits
#endif
#define SST25_SPIBRG      SPI1ABRG
#define SST25_SPIBUF      SPI1ABUF

#elif defined (USE_SST25_SPI2A)

#define SST25_SPISTAT     SPI2ASTAT
#define SST25_SPISTATbits SPI2ASTATbits
#ifdef __PIC32MX__
#define SST25_SPICON      SPI2ACON
#define SST25_SPICONbits  SPI2ACONbits
#else
#define SST25_SPICON1     SPI2ACON1
#define SST25_SPICON1bits SPI2ACON1bits
#define SST25_SPICON2     SPI2ACON2
#define SST25_SPICON2bits SPI2ACON2bits
#endif
#define SST25_SPIBRG      SPI2ABRG
#define SST25_SPIBUF      SPI2ABUF

#define SST25_SCK_TRIS  TRISGbits.TRISG6
#define SST25_SDO_TRIS  TRISGbits.TRISG8
#define SST25_SDI_TRIS  TRISGbits.TRISG7

#elif defined (USE_SST25_SPI3A)

#define SST25_SPISTAT     SPI3ASTAT
#define SST25_SPISTATbits SPI3ASTATbits
#ifdef __PIC32MX__
#define SST25_SPICON      SPI3ACON
#define SST25_SPICONbits  SPI3ACONbits
#else
#define SST25_SPICON1     SPI3ACON1
#define SST25_SPICON1bits SPI3ACON1bits
#define SST25_SPICON2     SPI3ACON2
#define SST25_SPICON2bits SPI3ACON2bits
#endif
#define SST25_SPIBRG      SPI3ABRG
#define SST25_SPIBUF      SPI3ABUF

#else
#error "Please define a SPI channel for SPI Flash"

#endif

/*****************************************************************
**
* SST25 Commands
*****************************************************************
*/
    #define SST25_CMD_READ  (unsigned)0x03
    #define SST25_CMD_WRITE (unsigned)0x02
    #define SST25_CMD_WREN  (unsigned)0x06
```

```
    #define SST25_CMD_RDSR  (unsigned)0x05
    #define SST25_CMD_ERASE (unsigned)0x60
    #define SST25_CMD_EWSR  (unsigned)0x50
    #define SST25_CMD_WRSR  (unsigned)0x01
    #define SST25_CMD_SER   (unsigned)0x20

/*********************************************************************
**
* Macro: SST25CSLow()
*
* Preconditions: CS IO must be configured as output
*
* Overview: this macro pulls down CS line
*
* Input: none
*
* Output: none
*
*********************************************************************
*/
    #define SST25CSLow()    SST25_CS_LAT = 0;

/*********************************************************************
**
* Macro: SST25CSHigh()
*
* Preconditions: CS IO must be configured as output
*
* Overview: this macro set CS line to high level
*
* Input: none
*
* Output: none
*
*********************************************************************
*/
    #define SST25CSHigh()   SST25_CS_LAT = 1;

/*********************************************************************
**
* Function: SST25Init()
*
* Overview: this function setups SPI and IOs connected to SST25
*
* Input: none
*
* Output: none
*
*********************************************************************
*/
void    SST25Init(void);

/*********************************************************************
**
```

```
* Function: BYTE SST25IsWriteBusy(void)
*
* Overview: this function reads status register and chek BUSY bit for
write operation
*
* Input: none
*
* Output: non zero if busy
*
*****************************************************************
*/
BYTE    SST25IsWriteBusy(void);

/*****************************************************************
**
* Function: void SST25WriteByte(BYTE data, DWORD address)
*
* Overview: this function writes a byte to the address specified
*
* Input: byte to be written and address
*
* Output: none
*
*****************************************************************
*/
void    SST25WriteByte(BYTE data, DWORD address);

/*****************************************************************
**
* Function: BYTE SST25ReadByte(DWORD address)
*
* Overview: this function reads a byte from the address specified
*
* Input: address
*
* Output: data read
*
*****************************************************************
*/
BYTE    SST25ReadByte(DWORD address);

/*****************************************************************
**
* Function: void SST25WriteWord(WODR data, DWORD address)
*
* Overview: this function writes a 16-bit word to the address specified
*
* Input: data to be written and address
*
* Output: none
*
*****************************************************************
*/
void    SST25WriteWord(WORD data, DWORD address);
```

```
/*************************************************************************
**
* Function: WORD SST25ReadWord(DWORD address)
*
* Overview: this function reads a 16-bit word from the address
specified
*
* Input: address
*
* Output: data read
*
*************************************************************************
*/
WORD    SST25ReadWord(DWORD address);

/*************************************************************************
**
* Function: SST25WriteEnable()
*
* Overview: this function allows writing into SST25. Must be called
*           before every write/erase command
*
* Input: none
*
* Output: none
*
*************************************************************************
*/
void    SST25WriteEnable(void);

/*************************************************************************
**
* Function: BYTE SST25WriteArray(DWORD address, BYTE* pData, nCount)
*
* Overview: this function writes data array at the address specified
*
* Input: flash memory address, pointer to the data array, data number
*
* Output: return 1 if the operation was successfull
*
*************************************************************************
*/
BYTE    SST25WriteArray(DWORD address, BYTE *pData, WORD nCount);

/*************************************************************************
**
* Function: void SST25ReadArray(DWORD address, BYTE* pData, nCount)
*
* Overview: this function reads  data into buffer specified
*
* Input: flash memory address, pointer to the buffer, data number
*
```

```
*****************************************************************************
*/
void    SST25ReadArray(DWORD address, BYTE *pData, WORD nCount);

/*****************************************************************************
**
* Function: void SST25ChipErase(void)
*
* Overview: chip erase
*
* Input: none
*
*****************************************************************************
*/
void    SST25ChipErase(void);

/*****************************************************************************
**
* Function: void SST25ResetWriteProtection()
*
* Overview: this function reset write protection bits
*
* Input: none
*
* Output: none
*
*****************************************************************************
*/
void    SST25ResetWriteProtection(void);

/*****************************************************************************
**
* Function: void SST25SectorErase(DWORD address)
*
* Overview: this function erases a 4Kb sector
*
* Input: address within sector to be erased
*
* Output: none
*
*****************************************************************************
*/
void    SST25SectorErase(DWORD address);
#endif //_SST25VF016_H
```

```
/*********************************************************************
*
* Basic access to SPI Flash SST25VF016 located on
* Graphics LCD Controller PICtail Plus SSD1926 Board.
*
*********************************************************************
* FileName:        SST25VF016.c
* Dependencies:    SST25VF016.h
* Processor:       PIC24F, PIC24H, dsPIC, PIC32
* Compiler:          MPLAB C30 V3.00, MPLAB C32
* Linker:          MPLAB LINK30, MPLAB LINK32
* Company:         Microchip Technology Incorporated
*
* Software License Agreement
*
* Copyright © 2008 Microchip Technology Inc.  All rights reserved.
* Microchip licenses to you the right to use, modify, copy and distribute
* Software only when embedded on a Microchip microcontroller or digital
* signal controller, which is integrated into your product or third party
* product (pursuant to the sublicense terms in the accompanying license
* agreement).
*
* You should refer to the license agreement accompanying this Software
* for additional information regarding your rights and obligations.
*
* SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
* KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY
* OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR
* PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR
* OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION,
* BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT
* DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL,
* INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA,
* COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY
* CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
* OR OTHER SIMILAR COSTS.
*
* Author          Date      Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Anton Alkhimenok                01/07/09          ...
*********************************************************************/
#include "SST25VF016.h"

#if defined (USE_SST25VF016)

/*********************************************************************
* Function: SST25Init
*
* Overview: this function setup SPI and IOs connected to SST25
*
* Input: none
```

```
*
* Output: none
*
*****************************************************************************/
void SST25Init(void)
{

            #if defined (EXPLORER_16)
/*****************************************************************
* For Explorer 16 RD12 is connected to EEPROM chip select.
* To prevent a conflict between this EEPROM and SST25 flash
* the chip select of the EEPROM SPI should be pulled up.
*****************************************************************************
*/
    // Set IOs directions for EEPROM SPI
    EEPROM_CS_LAT = 1;              // set initial CS value to 1 (not
asserted)
      EEPROM_CS_TRIS = 0;                 // set CS pin to output
            #endif // #if defined (EXPLORER_16)

    // Initialize SPI
        #ifdef __PIC32MX
    SST25_SPISTAT = 0;
    SST25_SPICON = 0;
    SST25_SPIBRG = 0;
    SST25_SPICONbits.MSTEN = 1;
    SST25_SPICONbits.CKP = 0;
    SST25_SPICONbits.CKE = 1;
    SST25_SPICONbits.SMP = 1;
    SST25_SPIBRG = 1;
    SST25_SPICONbits.ON = 1;
    #else
        #if defined (GFX_PICTAIL_V3)
    SST25_SPISTAT = 0;
    SST25_SPICON1 = 0x001b;
    SST25_SPICON1bits.MSTEN = 1;
    SST25_SPICON2 = 0;
    SST25_SPICON1bits.MODE16 = 0;
    SST25_SPICON1bits.CKE = 0;
    SST25_SPICON1bits.CKP = 1;
    SST25_SPICON1bits.SMP = 1;
    SST25_SPISTATbits.SPIEN = 1;
        #elif defined (PIC24FJ256DA210_DEV_BOARD)
    SST25_SPISTAT = 0;
    #if defined (SPI_CLK_SPEED_8MHZ)      // use 8 MHZ
            SST25_SPICON1bits.SPRE = 6;
            SST25_SPICON1bits.PPRE = 3;
          SST25_SPICON1bits.CKE = 1;
      SST25_SPICON1bits.CKP = 0;
      SST25_SPICON1bits.SMP = 0;
      #else                                                 // use 16
MHZ
            SST25_SPICON1bits.SPRE = 7;
            SST25_SPICON1bits.PPRE = 3;
```

```
          SST25_SPICON1bits.CKE = 0;
          SST25_SPICON1bits.CKP = 1;
          SST25_SPICON1bits.SMP = 1;
      #endif
    SST25_SPICON1bits.MSTEN = 1;
    SST25_SPICON2 = 0;
    SST25_SPICON1bits.MODE16 = 0;
    SST25_SPISTATbits.SPIEN = 1;
        #endif
      #endif
    // Set IOs directions for SST25 SPI
    #if defined (GFX_PICTAIL_V3) || defined (MULTI_MEDIA_BOARD_DM00123)

    SST25_CS_LAT = 1;
    SST25_CS_TRIS = 0;
    #ifndef __PIC32MX__
    SST25_SCK_TRIS = 0;
    SST25_SDO_TRIS = 0;
    SST25_SDI_TRIS = 1;
    #if defined(__PIC24FJ256GB210__)
      SST25_SDI_ANS = 0;
      #endif
    #endif

    #elif defined (PIC24FJ256DA210_DEV_BOARD)
    SST25_CS_LAT = 1;
    SST25_CS_TRIS = 0;
      SST25_SDI_ANS = 0;
    SST25_SDO_ANS = 0;
    SST25_SCK_TRIS = 0;
    SST25_SDO_TRIS = 0;
    SST25_SDI_TRIS = 1;
      #endif

        #if defined(__dsPIC33FJ128GP804__) ||
defined(__PIC24HJ128GP504__)
    AD1PCFGL = 0xFFFF;
    RPOR9bits.RP18R = 11;                   // assign RP18 for SCK2
    RPOR8bits.RP16R = 10;                   // assign RP16 for SDO2
    RPINR22bits.SDI2R = 17;                 // assign RP17 for SDI2
        #elif defined(__PIC24FJ256GB110__) ||
defined(__PIC24FJ256GA110__) || defined (__PIC24FJ256GB210__)
    __builtin_write_OSCCONL(OSCCON & 0xbf); // unlock PPS
    RPOR10bits.RP21R = 11;                  // assign RP21 for SCK2
    RPOR9bits.RP19R = 10;                   // assign RP19 for SDO2
    RPINR22bits.SDI2R = 26;                 // assign RP26 for SDI2
    __builtin_write_OSCCONL(OSCCON | 0x40); // lock   PPS
        #elif defined(__PIC24FJ256DA210__)
    __builtin_write_OSCCONL(OSCCON & 0xbf); // unlock PPS
    #if defined (USE_SST25_SPI2)
          RPOR1bits.RP2R = 11;                // assign RP2 for SCK2
          RPOR0bits.RP1R = 10;                // assign RP1 for SDO2
          RPINR22bits.SDI2R = 0;              // assign RP0 for SDI2
      #elif defined (USE_SST25_SPI1)
```

```
          RPOR1bits.RP2R = 8;                 // assign RP2 for SCK1
          RPOR0bits.RP1R = 7;                 // assign RP1 for SDO1
          RPINR20bits.SDI1R = 0;              // assign RP0 for SDI1
      #endif
    __builtin_write_OSCCONL(OSCCON | 0x40); // lock   PPS
        #endif

    SST25ResetWriteProtection();
}

/*********************************************************************
**
* Function SPIPut(BYTE data)
*
* Overview:  this function sends a byte
*
* Input: byte to be sent
*
* Output: none
*
*********************************************************************
*/
void SPIPut(BYTE data)
{
        #ifdef __PIC32MX

    // Wait for free buffer
    while(!SST25_SPISTATbits.SPITBE);
    SST25_SPIBUF = data;

    // Wait for data byte
    while(!SST25_SPISTATbits.SPIRBF);
        #else

    // Wait for free buffer
    while(SST25_SPISTATbits.SPITBF);
    SST25_SPIBUF = data;

    // Wait for data byte
    while(!SST25_SPISTATbits.SPIRBF);
        #endif
}

/*********************************************************************
**
* Macros SPIGet()
*
* Overview:  this macros gets a byte from SPI
*
* Input: none
*
* Output: none
*
```

```
*********************************************************************
*/
#define SPIGet()     SST25_SPIBUF

/*********************************************************************
**
* Function: void SST25WriteByte(BYTE data, DWORD address)
*
* Overview: this function writes a byte to the address specified
*
* Input: data to be written and address
*
* Output: none
*
*********************************************************************
*/
void SST25WriteByte(BYTE data, DWORD address)
{
    SST25WriteEnable();
    SST25CSLow();

    SPIPut(SST25_CMD_WRITE);
    SPIGet();

    SPIPut(((DWORD_VAL) address).v[2]);
    SPIGet();

    SPIPut(((DWORD_VAL) address).v[1]);
    SPIGet();

    SPIPut(((DWORD_VAL) address).v[0]);
    SPIGet();

    SPIPut(data);
    SPIGet();

    SST25CSHigh();

    // Wait for write end
    while(SST25IsWriteBusy());
}

/*********************************************************************
**
* Function: BYTE SST25ReadByte(DWORD address)
*
* Overview: this function reads a byte from the address specified
*
* Input: address
*
* Output: data read
*
*********************************************************************
*/
```

```
BYTE SST25ReadByte(DWORD address)
{
    BYTE    temp;
    SST25CSLow();

    SPIPut(SST25_CMD_READ);
    SPIGet();

    SPIPut(((DWORD_VAL) address).v[2]);
    SPIGet();

    SPIPut(((DWORD_VAL) address).v[1]);
    SPIGet();

    SPIPut(((DWORD_VAL) address).v[0]);
    SPIGet();

    SPIPut(0);
    temp = SPIGet();

    SST25CSHigh();
    return (temp);
}

/*********************************************************************
**
* Function: void SST25WriteWord(WODR data, DWORD address)
*
* Overview: this function writes a 16-bit word to the address specified
*
* Input: data to be written and address
*
* Output: none
*
*********************************************************************
*/
void SST25WriteWord(WORD data, DWORD address)
{
        #if defined(__dsPIC33FJ128GP804__) ||
defined(__PIC24HJ128GP504__)
    AD1PCFGLbits.PCFG6 = 1;
    AD1PCFGLbits.PCFG7 = 1;
    AD1PCFGLbits.PCFG8 = 1;
        #endif
    SST25WriteByte(((WORD_VAL) data).v[0], address);
    SST25WriteByte(((WORD_VAL) data).v[1], address + 1);
}

/*********************************************************************
**
* Function: WORD SST25ReadWord(DWORD address)
*
* Overview: this function reads a 16-bit word from the address
specified
```

```
*
* Input: address
*
* Output: data read
*
*****************************************************************************
*/
WORD SST25ReadWord(DWORD address)
{
    WORD_VAL    temp;

        #if defined(__dsPIC33FJ128GP804__) ||
defined(__PIC24HJ128GP504__)
    AD1PCFGLbits.PCFG6 = 1;
    AD1PCFGLbits.PCFG7 = 1;
    AD1PCFGLbits.PCFG8 = 1;
        #endif
    temp.v[0] = SST25ReadByte(address);
    temp.v[1] = SST25ReadByte(address + 1);

    return (temp.Val);
}

/*****************************************************************************
**
* Function: SST25WriteEnable()
*
* Overview: this function allows write/erase SST25. Must be called
* before every write/erase command.
*
* Input: none
*
* Output: none
*
*****************************************************************************
*/
void SST25WriteEnable(void)
{
    SST25CSLow();
    SPIPut(SST25_CMD_WREN);
    SPIGet();
    SST25CSHigh();
}

/*****************************************************************************
**
* Function: BYTE SST25IsWriteBusy(void)
*
* Overview: this function reads status register and chek BUSY bit for
write operation
*
* Input: none
*
* Output: non zero if busy
```

```
*
*********************************************************************
*/
BYTE SST25IsWriteBusy(void)
{
    BYTE    temp;

    SST25CSLow();
    SPIPut(SST25_CMD_RDSR);
    SPIGet();

    SPIPut(0);
    temp = SPIGet();
    SST25CSHigh();

    return (temp & 0x01);
}

/*********************************************************************
**
* Function: BYTE SST25WriteArray(DWORD address, BYTE* pData, nCount)
*
* Overview: this function writes a data array at the address specified
*
* Input: flash memory address, pointer to the data array, data number
*
* Output: return 1 if the operation was successfull
*
*********************************************************************
*/
BYTE SST25WriteArray(DWORD address, BYTE *pData, WORD nCount)
{
    DWORD   addr;
    BYTE    *pD;
    WORD    counter;

    addr = address;
    pD = pData;

    // WRITE
    for(counter = 0; counter < nCount; counter++)
    {
        SST25WriteByte(*pD++, addr++);
    }

    // VERIFY
    for(counter = 0; counter < nCount; counter++)
    {
        if(*pData != SST25ReadByte(address))
            return (0);
        pData++;
        address++;
    }
```

```
    return (1);
}

/*********************************************************************
**
* Function: void SST25ReadArray(DWORD address, BYTE* pData, nCount)
*
* Overview: this function reads data into buffer specified
*
* Input: flash memory address, pointer to the data buffer, data number
*
*********************************************************************
*/
void SST25ReadArray(DWORD address, BYTE *pData, WORD nCount)
{
    SST25CSLow();

    SPIPut(SST25_CMD_READ);
    SPIGet();

    SPIPut(((DWORD_VAL) address).v[2]);
    SPIGet();

    SPIPut(((DWORD_VAL) address).v[1]);
    SPIGet();

    SPIPut(((DWORD_VAL) address).v[0]);
    SPIGet();

    while(nCount--)
    {
        SPIPut(0);
        *pData++ = SPIGet();
    }

    SST25CSHigh();
}

/*********************************************************************
**
* Function: void SST25ChipErase(void)
*
* Overview: chip erase
*
* Input: none
*
*********************************************************************
*/
void SST25ChipErase(void)
{
    SST25WriteEnable();

    SST25CSLow();
```

```
    SPIPut(SST25_CMD_ERASE);
    SPIGet();

    SST25CSHigh();

    // Wait for write end
    while(SST25IsWriteBusy());
}

/*****************************************************************************
**
* Function: void SST25ResetWriteProtection()
*
* Overview: this function reset write protection bits
*
* Input: none
*
* Output: none
*
*****************************************************************************
*/
void SST25ResetWriteProtection(void)
{
    SST25CSLow();

    SPIPut(SST25_CMD_EWSR);
    SPIGet();

    SST25CSHigh();

    SST25CSLow();

    SPIPut(SST25_CMD_WRSR);
    SPIGet();

    SPIPut(0);
    SPIGet();

    SST25CSHigh();
}

/*****************************************************************************
**
* Function: void SST25SectorErase(DWORD address)
*
* Overview: this function erases a 4Kb sector
*
* Input: address within sector to be erased
*
* Output: none
*
*****************************************************************************
*/
void SST25SectorErase(DWORD address)
```

```
{
    SST25WriteEnable();
    SST25CSLow();

    SPIPut(SST25_CMD_SER);
    SPIGet();

    SPIPut(((DWORD_VAL) address).v[2]);
    SPIGet();

    SPIPut(((DWORD_VAL) address).v[1]);
    SPIGet();

    SPIPut(((DWORD_VAL) address).v[0]);
    SPIGet();

    SST25CSHigh();

    // Wait for write end
    DelayMs(100);
    while(SST25IsWriteBusy());
}

#endif // #if defined (USE_SST25VF016)
```

```
/*****************************************************************************
 * Simple 4 wire touch screen driver
 *****************************************************************************
 * FileName:        TouchScreen.h
 * Dependencies:    Graphics.h
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:          MPLAB C30, MPLAB C32
 * Linker:          MPLAB LINK30, MPLAB LINK32
 * Company:         Microchip Technology Incorporated
 *
 * Software License Agreement
 *
 * Copyright © 2008 Microchip Technology Inc.  All rights reserved.
 * Microchip licenses to you the right to use, modify, copy and distribute
 * Software only when embedded on a Microchip microcontroller or digital
 * signal controller, which is integrated into your product or third party
 * product (pursuant to the sublicense terms in the accompanying license
 * agreement).
 *
 * You should refer to the license agreement accompanying this Software
 * for additional information regarding your rights and obligations.
 *
 * SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
 * KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY
 * OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR
 * PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR
 * OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION,
 * BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT
 * DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL,
 * INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA,
 * COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY
 * CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
 * OR OTHER SIMILAR COSTS.
 *
 * Date             Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 01/08/07                 ...
 * 06/06/07                 Basic calibration and GOL messaging are added
 * 01/14/08         Graphics PICtail Version 2 support is added
 * 02/05/08         Portrait orientation is added
 * 02/07/08         PIC32 support is added
 * 05/27/08         More robust algorithm
 * 01/07/09         Graphics PICtail Version 3 support is added
 * 07/27/10         Added support for parallel flash
 *                                  (only on PIC24FJ256DA120 Development Board)
 * 09/30/10         Added automatic X and Y direction detection based on the display
 *                                  orientation of the two panels:
 *                                  - DISPLAY_PANEL == TFT_G240320LTSW_118W_E
 *                                  - DISPLAY_PANEL == PH480272T_005_I11Q
 *****************************************************************************/
#ifndef _TOUCHSCREEN_H
```

```
    #define _TOUCHSCREEN_H

    #include "Graphics/Graphics.h"

    #if defined (GFX_PICTAIL_V1)

//#define SWAP_X_AND_Y
        #define FLIP_X

//#define FLIP_Y
    #elif defined (GFX_PICTAIL_V2)

            #include "EEPROM.h"

//#define SWAP_X_AND_Y
//#define FLIP_X
        #define FLIP_Y

    #elif defined(GFX_PICTAIL_V3) || defined
(PIC24FJ256DA210_DEV_BOARD)
            #if defined (USE_SST39LF400)
                  #include "SST39LF400.h"       // used in
PIC24FJ256DA210 Development Board only, selectable with SPI Flash
            #else
                  #include "SST25VF016.h"       // used in
GFX_PICTAIL_V3 and PIC24FJ256DA210 Development Board
            #endif

            #if (DISPLAY_PANEL == TFT_G240320LTSW_118W_E)
                  #if (DISP_ORIENTATION == 0)
                        #define SWAP_X_AND_Y
                        #define FLIP_Y
                  #elif (DISP_ORIENTATION == 180)
                        #define SWAP_X_AND_Y
                        #define FLIP_X
                  #elif (DISP_ORIENTATION == 270)
                        #define FLIP_X
                        #define FLIP_Y
                  #endif
            #endif

            #if (DISPLAY_PANEL == PH480272T_005_I11Q)
                  #if (DISP_ORIENTATION == 90)
                        #define SWAP_X_AND_Y
                        #define FLIP_X
                  #elif (DISP_ORIENTATION == 180)
                        #define FLIP_X
                        #define FLIP_Y
                  #elif (DISP_ORIENTATION == 270)
                        #define SWAP_X_AND_Y
                        #define FLIP_Y
                  #endif
            #endif
    #endif
```

```
// Default calibration values
    #define YMINCAL 0x74
    #define YMAXCAL 0x284
    #define XMINCAL 0x2f
    #define XMAXCAL 0x39f

// Max/Min ADC values for each direction
extern volatile WORD    _calXMin;
extern volatile WORD    _calXMax;
extern volatile WORD    _calYMin;
extern volatile WORD    _calYMax;

    #if defined (GFX_PICTAIL_V1) || defined (GFX_PICTAIL_V2)

// Addresses for calibration and version values in EEPROM on Explorer
16
        #define ADDRESS_VERSION (unsigned)0x7FFE
        #define ADDRESS_XMIN    (unsigned)0x7FFC
        #define ADDRESS_XMAX    (unsigned)0x7FFA
        #define ADDRESS_YMIN    (unsigned)0x7FF8
        #define ADDRESS_YMAX    (unsigned)0x7FF6
    #else

// Addresses for calibration and version values in SPI Flash on
Graphics PICtail 3 & PIC24FJ256DA210 Development Board.
// Or Addresses for calibration and version values in Parallel Flash on
PIC24FJ256DA210 Development Board.
            #if defined (USE_SST25VF016)
              #define ADDRESS_VERSION (unsigned long)0x003FFFFE
              #define ADDRESS_XMIN    (unsigned long)0x003FFFFC
              #define ADDRESS_XMAX    (unsigned long)0x003FFFFA
              #define ADDRESS_YMIN    (unsigned long)0x003FFFF8
              #define ADDRESS_YMAX    (unsigned long)0x003FFFF6
        #elif defined (USE_SST39LF400)
              #define ADDRESS_VERSION (unsigned long)0x0003FFFE
              #define ADDRESS_XMIN    (unsigned long)0x0003FFFC
              #define ADDRESS_XMAX    (unsigned long)0x0003FFFA
              #define ADDRESS_YMIN    (unsigned long)0x0003FFF8
              #define ADDRESS_YMAX    (unsigned long)0x0003FFF6
            #else
        #endif


    #endif

// Current ADC values for X and Y channels and potentiometer R6
extern volatile SHORT   adcX;
extern volatile SHORT   adcY;
extern volatile SHORT   adcPot;

/*****************************************************************
* Function: void TouchProcessTouch(void)
*
```

```
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: Process the detection of touch on the ADC
*
* Note: none
*
********************************************************************/
void TouchProcessTouch(void);


/*********************************************************************
* Function: void TouchInit(void)
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: sets ADC
*
* Note: none
*
********************************************************************/
void                    TouchInit(void);

/*********************************************************************
* Function: SHORT TouchGetX()
*
* PreCondition: none
*
* Input: none
*
* Output: x coordinate
*
* Side Effects: none
*
* Overview: returns x coordinate if touch screen is pressed
*           and -1 if not
*
* Note: none
*
********************************************************************/
SHORT                   TouchGetX(void);

/*********************************************************************
```

```
* Function: SHORT TouchGetY()
*
* PreCondition: none
*
* Input: none
*
* Output: y coordinate
*
* Side Effects: none
*
* Overview: returns y coordinate if touch screen is pressed
*           and -1 if not
*
* Note: none
*
*********************************************************************/
SHORT                   TouchGetY(void);

/*********************************************************************
* Function: void TouchGetMsg(GOL_MSG* pMsg)
*
* PreCondition: none
*
* Input: pointer to the message structure to be populated
*
* Output: none
*
* Side Effects: none
*
* Overview: populates GOL message structure
*
* Note: none
*
*********************************************************************/
void                    TouchGetMsg(GOL_MSG *pMsg);

/*********************************************************************
* Function: void TouchCalibration()
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: Runs the calibration routine.
*
* Note: none
*
*********************************************************************/
void                    TouchCalibration(void);
```

```
/*********************************************************************
* Function: void TouchStoreCalibration(void)
*
* PreCondition: EEPROMInit() must be called before
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: stores calibration parameters into EEPROM
*
* Note: none
*
*********************************************************************/
void                    TouchStoreCalibration(void);

/*********************************************************************
* Function: void TouchLoadCalibration(void)
*
* PreCondition: EEPROMInit() must be called before
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: loads calibration parameters from EEPROM
*
* Note: none
*
*********************************************************************/
void                    TouchLoadCalibration(void);

/*********************************************************************
* Macros: ADCGetX()
*
* PreCondition: none
*
* Input: none
*
* Output: ADC result
*
* Side Effects: none
*
* Overview: returns ADC value for x direction if touch screen is
pressed
*           and -1 if not
*
* Note: none
*
*********************************************************************/
```

```
    #define ADCGetX()   adcX

/*****************************************************************
* Macros: ADCGetY()
*
* PreCondition: none
*
* Input: none
*
* Output: ADC result
*
* Side Effects: none
*
* Overview: returns ADC value for y direction if touch screen is
pressed
*           and -1 if not
*
* Note: none
*
*****************************************************************/
    #define ADCGetY()   adcY

/*****************************************************************
* Macros: ADCGetPot()
*
* PreCondition: none
*
* Input: none
*
* Output: ADC result
*
* Side Effects: none
*
* Overview: returns ADC value for potentiometer
*
* Note: none
*
*****************************************************************/
    #define ADCGetPot() adcPot
#endif
```

```
/*****************************************************************************
*
* Simple 4 wire touch screen driver
*
*****************************************************************************
* FileName:        TouchScreen.c
* Dependencies:    TouchScreen.h
* Processor:       PIC24, PIC32, dsPIC, PIC24H
* Compiler:          MPLAB C30, MPLAB C32
* Linker:          MPLAB LINK30, MPLAB LINK32
* Company:         Microchip Technology Incorporated
*
* Software License Agreement
*
* Copyright © 2008 Microchip Technology Inc.  All rights reserved.
* Microchip licenses to you the right to use, modify, copy and distribute
* Software only when embedded on a Microchip microcontroller or digital
* signal controller, which is integrated into your product or third party
* product (pursuant to the sublicense terms in the accompanying license
* agreement).
*
* You should refer to the license agreement accompanying this Software
* for additional information regarding your rights and obligations.
*
* SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
* KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY
* OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR
* PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR
* OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION,
* BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT
* DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL,
* INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA,
* COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY
* CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
* OR OTHER SIMILAR COSTS.
*
* Date              Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* 01/08/07                  ...
* 06/06/07                  Basic calibration and GOL messaging are added
* 02/05/08          new PICtail support, portrait orientation is added
* 02/07/08          PIC32 support
* 05/27/08          More robust algorithm
* 01/07/09          Graphics PICtail Version 3 support is added
* 06/25/09          dsPIC & PIC24H support
* 06/29/09                  Added event EVENT_STILLPRESS to support continuous press
* 04/15/10                  Qualified EVENT_STILLPRESS to be set only when
*           an actual touch is occuring.
*           Removed timer and timer ISR, made the touch detection
*           a function.
* 07/27/10          Added support for parallel flash
```

```
*                               (only on PIC24FJ256DA120 Development Board)
*****************************************************************************/
#include "TouchScreen.h"

#if defined (USE_RESISTIVE_TOUCH)

#if defined (USE_25LC256)
      #include "EEPROM.h"
#elif defined (USE_SST25VF016)
      #include "SST25VF016.h"
#elif defined (USE_SST39LF400)
      #include "SST39LF400.h"
#else
      #warning "Touchscreen is being used but calibration data will not
be saved or retrieved. Enable the memory that will hold the calibration
data in the Hardware Peofile."
#endif

////////////////////// LOCAL PROTOTYPES ///////////////////////////
void    TouchGetCalPoints(void);

#define WAIT_UNTIL_FINISH(x)    while(!x)

////////////////////// GLOBAL VARIABLES ///////////////////////////
#define PRESS_THRESHOULD    256 // between 0-0x03ff the lesser this value the lighter the
screen must be pressed
#define CALIBRATION_DELAY   300 // delay between calibration touch points

    // Max/Min ADC values for each derection
volatile WORD     _calXMin = XMINCAL;
volatile WORD       _calXMax = XMAXCAL;
volatile WORD       _calYMin = YMINCAL;
volatile WORD       _calYMax = YMAXCAL;

// Current ADC values for X and Y channels
volatile SHORT      adcX = -1;
volatile SHORT      adcY = -1;
volatile SHORT      adcPot = 0;

typedef enum
{
    SET_X,
    RUN_X,
    GET_X,
    RUN_CHECK_X,
    CHECK_X,
    SET_Y,
    RUN_Y,
    GET_Y,
    CHECK_Y,
    SET_VALUES,
    GET_POT,
    RUN_POT
} TOUCH_STATES;
```

```
volatile TOUCH_STATES state = SET_X;

void TouchProcessTouch(void)
{
    static SHORT    tempX, tempY;
    SHORT           temp;

    switch(state)
    {

        case SET_VALUES:
            if(!TOUCH_ADC_DONE)
                break;

            if((WORD) PRESS_THRESHOULD < (WORD) ADC1BUF0)
            {
                adcX = -1;
                adcY = -1;
            }
            else
            {
                adcX = tempX;
                adcY = tempY;
            }
        // If the hardware supports an analog pot, if not skip it
        #ifdef ADC_POT
            state = RUN_POT;

       case RUN_POT:
            #if defined(__dsPIC33F__) || defined(__PIC24H__)
            AD1CHS0 = ADC_POT;      // switch ADC channel
            #else
            AD1CHS = ADC_POT;       // switch ADC channel
            #endif
            AD1CON1bits.SAMP = 1;   // run conversion
            state = GET_POT;
            break;

        case GET_POT:
            if(!AD1CON1bits.DONE)
                break;

            adcPot = ADC1BUF0;
        #endif
            state = SET_X;

        case SET_X:
            #if defined(__dsPIC33F__) || defined(__PIC24H__)
            AD1CHS0 = ADC_XPOS;     // switch ADC channel
            #else
            AD1CHS = ADC_XPOS;      // switch ADC channel
            #endif
```

```
    TRIS_XPOS = 1;
    TRIS_YPOS = 1;
    TRIS_XNEG = 1;
    LAT_YNEG = 0;
    TRIS_YNEG = 0;

    AD1CON1bits.SAMP = 1;    // run conversion
    state = CHECK_X;
    break;

case CHECK_X:
case CHECK_Y:
    if(!TOUCH_ADC_DONE)
        break;

    if((WORD) PRESS_THRESHOULD > (WORD) ADC1BUF0)
    {
          if (state == CHECK_X)
          {
          LAT_YPOS = 1;
          TRIS_YPOS = 0;
          tempX = -1;
          state = RUN_X;
        }
        else
        {
              LAT_XPOS = 1;
          TRIS_XPOS = 0;
            tempY = -1;
              state = RUN_Y;
          }
    }
    else
    {
        adcX = -1;
        adcY = -1;
            #ifdef ADC_POT
              state = RUN_POT;
          #else
                state = SET_X;
          #endif
        break;
    }

case RUN_X:
case RUN_Y:
    AD1CON1bits.SAMP = 1;
    state = (state == RUN_X) ? GET_X : GET_Y;
    // no break needed here since the next state is either GET_X or GET_Y

case GET_X:
case GET_Y:
    if(!TOUCH_ADC_DONE)
```

```
            break;

        temp = ADC1BUF0;
        if (state == GET_X)
          {
              if(temp != tempX)
              {
                  tempX = temp;
                  state = RUN_X;
                  break;
              }
          }
          else
          {
              if(temp != tempY)
              {
                  tempY = temp;
                  state = RUN_Y;
                  break;
              }
           }

        if (state == GET_X)
              TRIS_YPOS = 1;
        else
              TRIS_XPOS = 1;
        AD1CON1bits.SAMP = 1;
        state = (state == GET_X) ? SET_Y : SET_VALUES;
        break;

    case SET_Y:
        if(!TOUCH_ADC_DONE)
            break;

        if((WORD) PRESS_THRESHOULD < (WORD) ADC1BUF0)
        {
            adcX = -1;
            adcY = -1;
                #ifdef ADC_POT
                  state = RUN_POT;
              #else
              state = SET_X;
            #endif
            break;
        }

        #if defined(__dsPIC33F__) || defined(__PIC24H__)
        AD1CHS0 = ADC_YPOS;      // switch ADC channel
        #else
        AD1CHS = ADC_YPOS;       // switch ADC channel
        #endif
        TRIS_XPOS = 1;
        TRIS_YPOS = 1;
```

```
            LAT_XNEG = 0;
            TRIS_XNEG = 0;
            TRIS_YNEG = 1;

            AD1CON1bits.SAMP = 1;    // run conversion
            state = CHECK_Y;
            break;

        default:
            state = SET_X;
    }

}

/*****************************************************************
* Function: void TouchInit(void)
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: sets ADC
*
* Note: none
*
*****************************************************************/
void TouchInit(void)
{
    // Initialize ADC
    AD1CON1 = 0x080E0;      // Turn on, auto-convert
    AD1CON2 = 0;            // AVdd, AVss, int every conversion, MUXA only
    AD1CON3 = 0x1FFF;       // 31 Tad auto-sample, Tad = 256*Tcy
    #if defined(__dsPIC33F__) || defined(__PIC24H__)
    AD1PCFGL = 0;           // All inputs are analog
    AD1PCFGLbits.PCFG11 = AD1PCFGLbits.PCFG12 = 1;
    #else
    #if !(defined(__PIC24FJ256DA210__) || defined(__PIC24FJ256GB210__))
    AD1PCFG = 0;            // All inputs are analog
    #endif
    #endif
    AD1CSSL = 0;            // No scanned inputs

}

/*****************************************************************
* Function: SHORT TouchGetX()
*
* PreCondition: none
*
* Input: none
```

```
*
* Output: x coordinate
*
* Side Effects: none
*
* Overview: returns x coordinate if touch screen is pressed
*       and -1 if not
*
* Note: none
*
*****************************************************************/
SHORT TouchGetX(void)
{
    long    result;

    #ifdef SWAP_X_AND_Y
    result = ADCGetY();
    #else
    result = ADCGetX();
    #endif
    if(result >= 0)
    {
        #ifdef SWAP_X_AND_Y
        result = (GetMaxX() * (result - _calYMin)) / (_calYMax -
_calYMin);
        #else
        result = (GetMaxX() * (result - _calXMin)) / (_calXMax -
_calXMin);
        #endif
        #ifdef FLIP_X
        result = GetMaxX() - result;
        #endif
    }

    return (result);
}

/*****************************************************************
* Function: SHORT TouchGetY()
*
* PreCondition: none
*
* Input: none
*
* Output: y coordinate
*
* Side Effects: none
*
* Overview: returns y coordinate if touch screen is pressed
*       and -1 if not
*
* Note: none
*
*****************************************************************/
```

```
SHORT TouchGetY(void)
{
    long    result;

    #ifdef SWAP_X_AND_Y
    result = ADCGetX();
    #else
    result = ADCGetY();
    #endif
    if(result >= 0)
    {
        #ifdef SWAP_X_AND_Y
        result = (GetMaxY() * (result - _calXMin)) / (_calXMax -
_calXMin);
        #else
        result = (GetMaxY() * (result - _calYMin)) / (_calYMax -
_calYMin);
        #endif
        #ifdef FLIP_Y
        result = GetMaxY() - result;
        #endif
    }

    return (result);
}

/*****************************************************************
* Function: void TouchGetMsg(GOL_MSG* pMsg)
*
* PreCondition: none
*
* Input: pointer to the message structure to be populated
*
* Output: none
*
* Side Effects: none
*
* Overview: populates GOL message structure
*
* Note: none
*
*****************************************************************/
void TouchGetMsg(GOL_MSG *pMsg)
{
    static SHORT    prevX = -1;
    static SHORT    prevY = -1;

    SHORT           x, y;

    x = TouchGetX();
    y = TouchGetY();
    pMsg->type = TYPE_TOUCHSCREEN;
    pMsg->uiEvent = EVENT_INVALID;
```

```
if(x == -1)
{
    y = -1;
}
else
{
    if(y == -1)
        x = -1;
}

if((prevX == x) && (prevY == y) && (x != -1) && (y != -1))
{
    pMsg->uiEvent = EVENT_STILLPRESS;
    pMsg->param1 = x;
    pMsg->param2 = y;
    return;
}

if((prevX != -1) || (prevY != -1))
{
    if((x != -1) && (y != -1))
    {

        // Move
        pMsg->uiEvent = EVENT_MOVE;
    }
    else
    {

        // Released
        pMsg->uiEvent = EVENT_RELEASE;
        pMsg->param1 = prevX;
        pMsg->param2 = prevY;
        prevX = x;
        prevY = y;
        return;
    }
}
else
{
    if((x != -1) && (y != -1))
    {

        // Pressed
        pMsg->uiEvent = EVENT_PRESS;
    }
    else
    {

        // No message
        pMsg->uiEvent = EVENT_INVALID;
    }
}
```

```
    pMsg->param1 = x;
    pMsg->param2 = y;
    prevX = x;
    prevY = y;
}

/*****************************************************************
* Function: void TouchStoreCalibration(void)
*
* PreCondition: EEPROMInit() must be called before
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: stores calibration parameters into EEPROM
*
* Note: none
*
*****************************************************************/
void TouchStoreCalibration(void)
{
      #if defined (USE_25LC256)
      EEPROMWriteWord(_calXMin, ADDRESS_XMIN);
      EEPROMWriteWord(_calXMax, ADDRESS_XMAX);
      EEPROMWriteWord(_calYMin, ADDRESS_YMIN);
      EEPROMWriteWord(_calYMax, ADDRESS_YMAX);
      EEPROMWriteWord(GRAPHICS_LIBRARY_VERSION, ADDRESS_VERSION);
    #elif defined (USE_SST39LF400)
            WORD tempArray[12];

            SST39LF400Init(tempArray);

          SST39LF400SectorErase(ADDRESS_XMIN); // erase 4K sector
            SST39LF400WriteWord(_calXMin, ADDRESS_XMIN);
            SST39LF400WriteWord(_calXMax, ADDRESS_XMAX);
            SST39LF400WriteWord(_calYMin, ADDRESS_YMIN);
            SST39LF400WriteWord(_calYMax, ADDRESS_YMAX);
            SST39LF400WriteWord(GRAPHICS_LIBRARY_VERSION,
ADDRESS_VERSION);

            SST39LF400DeInit(tempArray);

      #elif defined (USE_SST25VF016)
            SST25SectorErase(ADDRESS_XMIN); // erase 4K sector
            SST25WriteWord(_calXMin, ADDRESS_XMIN);
            SST25WriteWord(_calXMax, ADDRESS_XMAX);
            SST25WriteWord(_calYMin, ADDRESS_YMIN);
            SST25WriteWord(_calYMax, ADDRESS_YMAX);
            SST25WriteWord(GRAPHICS_LIBRARY_VERSION, ADDRESS_VERSION);
      #else
```

```
            #error "Touch screen is being used but calibration data
cannot be saved!"
      #endif
}

/*****************************************************************
* Function: void TouchLoadCalibration(void)
*
* PreCondition: EEPROMInit() must be called before
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: loads calibration parameters from EEPROM
*
* Note: none
*
*****************************************************************/
void TouchLoadCalibration(void)
{
      #if defined (USE_25LC256)
      _calXMin = EEPROMReadWord(ADDRESS_XMIN);
      _calXMax = EEPROMReadWord(ADDRESS_XMAX);
      _calYMin = EEPROMReadWord(ADDRESS_YMIN);
      _calYMax = EEPROMReadWord(ADDRESS_YMAX);
    #elif defined (USE_SST39LF400)
            WORD tempArray[12];

            SST39LF400Init(tempArray);

            _calXMin = SST39LF400ReadWord(ADDRESS_XMIN);
            _calXMax = SST39LF400ReadWord(ADDRESS_XMAX);
            _calYMin = SST39LF400ReadWord(ADDRESS_YMIN);
            _calYMax = SST39LF400ReadWord(ADDRESS_YMAX);

            SST39LF400DeInit(tempArray);

      #elif defined (USE_SST25VF016)
          _calXMin = SST25ReadWord(ADDRESS_XMIN);
          _calXMax = SST25ReadWord(ADDRESS_XMAX);
          _calYMin = SST25ReadWord(ADDRESS_YMIN);
          _calYMax = SST25ReadWord(ADDRESS_YMAX);
      #else
            #error "Touch screen is being used but calibration data is
not accessible!"
      #endif
}

/*****************************************************************
* Function:  void TouchCalibration()
*
```

```
* PreCondition: InitGraph() must be called before
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: calibrates touch screen
*
* Note: none
*
********************************************************************/
void TouchCalibration(void)
{
    static const XCHAR  scr1StrLn1[] =
{'I','M','P','O','R','T','A','N','T','.',0};
    static const XCHAR  scr1StrLn2[] =
{'P','e','r','f','o','r','m','i','n','g',' ','t','o','u','c','h',0};
    static const XCHAR  scr1StrLn3[] = {'s','c','r','e','e','n','
','c','a','l','i','b','r','a','t','i','o','n','.',0};
    static const XCHAR  scr1StrLn4[] =
{'T','o','u','c','h','p','o','i','n','t','s','
','E','X','A','C','T','L','Y',0};
    static const XCHAR  scr1StrLn5[] = {'a','t',' ','t','h','e','
','p','o','s','i','t','i','o','n','s',' ','s','h','o','w','n',0};
    static const XCHAR  scr1StrLn6[] = {'b','y','
','a','r','r','o','w','s','.',0};
    static const XCHAR  scr1StrLn7[] = {'T','o','u','c','h','
','s','c','r','e','e','n',' ','t','o',0};
    static const XCHAR  scr1StrLn8[] =
{'c','o','n','t','i','n','u','e','.',0};

#if defined (PIC24FJ256DA210_DEV_BOARD)
    static const XCHAR  scr2StrLn1[] = {'H','o','l','d',' ','S','1','
','b','u','t','t','o','n',' ','a','n','d',0};
    static const XCHAR  scr2StrLn2[] = {'p','r','e','s','s','
','M','C','L','R',' ','r','e','s','e','t',' ','t','o',0};
    static const XCHAR  scr2StrLn3[] = {'R','E','P','E','A','T','
','t','h','e',' ','c','a','l','i','b','r','a','t','i','o','n',0};
    static const XCHAR  scr2StrLn4[] =
{'p','r','o','c','e','d','u','r','e','.',0};
#else
    static const XCHAR  scr2StrLn1[] = {'H','o','l','d',' ','S','3','
','b','u','t','t','o','n',' ','a','n','d',0};
    static const XCHAR  scr2StrLn2[] = {'p','r','e','s','s','
','M','C','L','R',' ','r','e','s','e','t','(','S','1',')','
','t','o',0};
    static const XCHAR  scr2StrLn3[] = {'R','E','P','E','A','T','
','t','h','e',' ','c','a','l','i','b','r','a','t','i','o','n',0};
    static const XCHAR  scr2StrLn4[] =
{'p','r','o','c','e','d','u','r','e','.',0};
#endif
    SHORT               x, y;
```

```
    SHORT               textHeight, textStart;

    SetFont((void *) &FONTDEFAULT);

    textHeight = GetTextHeight((void *) &FONTDEFAULT);
    textStart =  (GetMaxY() - (textHeight*8)) >> 1;

    SetColor(WHITE);
    ClearDevice();

    SetColor(BRIGHTRED);
    WAIT_UNTIL_FINISH(OutTextXY((GetMaxX()-GetTextWidth((XCHAR
*)scr1StrLn1, (void *) &FONTDEFAULT))>>1,  \
                                            textStart, (XCHAR
*)scr1StrLn1));
    SetColor(BLACK);
    WAIT_UNTIL_FINISH(OutTextXY((GetMaxX()-GetTextWidth((XCHAR
*)scr1StrLn2, (void *) &FONTDEFAULT))>>1,  \
                                            textStart + (1*textHeight),
(XCHAR *)scr1StrLn2));
    WAIT_UNTIL_FINISH(OutTextXY((GetMaxX()-GetTextWidth((XCHAR
*)scr1StrLn3, (void *) &FONTDEFAULT))>>1,  \
                                            textStart + (2*textHeight),
(XCHAR *)scr1StrLn3));
    WAIT_UNTIL_FINISH(OutTextXY((GetMaxX()-GetTextWidth((XCHAR
*)scr1StrLn4, (void *) &FONTDEFAULT))>>1,  \
                                            textStart + (3*textHeight),
(XCHAR *)scr1StrLn4));
    WAIT_UNTIL_FINISH(OutTextXY((GetMaxX()-GetTextWidth((XCHAR
*)scr1StrLn5, (void *) &FONTDEFAULT))>>1,  \
                                            textStart + (4*textHeight),
(XCHAR *)scr1StrLn5));
    WAIT_UNTIL_FINISH(OutTextXY((GetMaxX()-GetTextWidth((XCHAR
*)scr1StrLn6, (void *) &FONTDEFAULT))>>1,  \
                                           textStart + (5*textHeight),
(XCHAR *)scr1StrLn6));
    SetColor(BRIGHTRED);
    WAIT_UNTIL_FINISH(OutTextXY((GetMaxX()-GetTextWidth((XCHAR
*)scr1StrLn7, (void *) &FONTDEFAULT))>>1,  \
                                           textStart + (6*textHeight),
(XCHAR *)scr1StrLn7));
    WAIT_UNTIL_FINISH(OutTextXY((GetMaxX()-GetTextWidth((XCHAR
*)scr1StrLn8, (void *) &FONTDEFAULT))>>1,  \
                                           textStart + (7*textHeight),
(XCHAR *)scr1StrLn8));

    // Wait for touch
    do
    {
        x = ADCGetX();
        y = ADCGetY();
    } while((y == -1) || (x == -1));
```

```
DelayMs(CALIBRATION_DELAY);

SetColor(WHITE);
ClearDevice();

SetColor(BRIGHTRED);

#ifdef SWAP_X_AND_Y
WAIT_UNTIL_FINISH(Line(GetMaxX() - 5, 5, GetMaxX() - 5, 15));
WAIT_UNTIL_FINISH(Line(GetMaxX() - 4, 5, GetMaxX() - 4, 15));
WAIT_UNTIL_FINISH(Line(GetMaxX() - 6, 5, GetMaxX() - 6, 15));

WAIT_UNTIL_FINISH(Line(GetMaxX() - 5, 5, GetMaxX() - 15, 5));
WAIT_UNTIL_FINISH(Line(GetMaxX() - 5, 4, GetMaxX() - 15, 4));
WAIT_UNTIL_FINISH(Line(GetMaxX() - 5, 6, GetMaxX() - 15, 6));

WAIT_UNTIL_FINISH(Line(GetMaxX() - 5, 6, GetMaxX() - 15, 16));
WAIT_UNTIL_FINISH(Line(GetMaxX() - 5, 4, GetMaxX() - 15, 14));
WAIT_UNTIL_FINISH(Line(GetMaxX() - 5, 5, GetMaxX() - 15, 15));

#else
WAIT_UNTIL_FINISH(Line(5, 5, 5, 15));
WAIT_UNTIL_FINISH(Line(4, 5, 4, 15));
WAIT_UNTIL_FINISH(Line(6, 5, 6, 15));

WAIT_UNTIL_FINISH(Line(5, 5, 15, 5));
WAIT_UNTIL_FINISH(Line(5, 4, 15, 4));
WAIT_UNTIL_FINISH(Line(5, 6, 15, 6));

WAIT_UNTIL_FINISH(Line(5, 6, 15, 16));
WAIT_UNTIL_FINISH(Line(5, 4, 15, 14));
WAIT_UNTIL_FINISH(Line(5, 5, 15, 15));
#endif
_calXMin = 0xFFFF;
_calXMax = 0;
_calYMin = 0xFFFF;
_calYMax = 0;

TouchGetCalPoints();

SetColor(WHITE);
ClearDevice();

SetColor(BRIGHTRED);

#ifdef SWAP_X_AND_Y
WAIT_UNTIL_FINISH(Line(5, 5, 5, 15));
WAIT_UNTIL_FINISH(Line(4, 5, 4, 15));
WAIT_UNTIL_FINISH(Line(6, 5, 6, 15));

WAIT_UNTIL_FINISH(Line(5, 5, 15, 5));
WAIT_UNTIL_FINISH(Line(5, 4, 15, 4));
WAIT_UNTIL_FINISH(Line(5, 6, 15, 6));
```

```
    WAIT_UNTIL_FINISH(Line(5, 6, 15, 16));
    WAIT_UNTIL_FINISH(Line(5, 4, 15, 14));
    WAIT_UNTIL_FINISH(Line(5, 5, 15, 15));

    #else
    WAIT_UNTIL_FINISH(Line(5, GetMaxY() - 5, 5, GetMaxY() - 15));
    WAIT_UNTIL_FINISH(Line(4, GetMaxY() - 5, 4, GetMaxY() - 15));
    WAIT_UNTIL_FINISH(Line(6, GetMaxY() - 5, 6, GetMaxY() - 15));

    WAIT_UNTIL_FINISH(Line(5, GetMaxY() - 5, 15, GetMaxY() - 5));
    WAIT_UNTIL_FINISH(Line(5, GetMaxY() - 4, 15, GetMaxY() - 4));
    WAIT_UNTIL_FINISH(Line(5, GetMaxY() - 6, 15, GetMaxY() - 6));

    WAIT_UNTIL_FINISH(Line(5, GetMaxY() - 6, 15, GetMaxY() - 16));
    WAIT_UNTIL_FINISH(Line(5, GetMaxY() - 4, 15, GetMaxY() - 14));
    WAIT_UNTIL_FINISH(Line(5, GetMaxY() - 5, 15, GetMaxY() - 15));
    #endif
    TouchGetCalPoints();

    SetColor(WHITE);
    ClearDevice();

    SetColor(BRIGHTRED);

    #ifdef SWAP_X_AND_Y
    WAIT_UNTIL_FINISH(Line(GetMaxX() / 2 - 5, GetMaxY() - 5, GetMaxX()
/ 2 - 5, GetMaxY() - 15));
    WAIT_UNTIL_FINISH(Line(GetMaxX() / 2 - 4, GetMaxY() - 5, GetMaxX()
/ 2 - 4, GetMaxY() - 15));
    WAIT_UNTIL_FINISH(Line(GetMaxX() / 2 - 6, GetMaxY() - 5, GetMaxX()
/ 2 - 6, GetMaxY() - 15));

    WAIT_UNTIL_FINISH(Line(GetMaxX() / 2 - 5, GetMaxY() - 5, GetMaxX()
/ 2 - 15, GetMaxY() - 5));
    WAIT_UNTIL_FINISH(Line(GetMaxX() / 2 - 5, GetMaxY() - 4, GetMaxX()
/ 2 - 15, GetMaxY() - 4));
    WAIT_UNTIL_FINISH(Line(GetMaxX() / 2 - 5, GetMaxY() - 6, GetMaxX()
/ 2 - 15, GetMaxY() - 6));

    WAIT_UNTIL_FINISH(Line(GetMaxX() / 2 - 5, GetMaxY() - 6, GetMaxX()
/ 2 - 15, GetMaxY() - 16));
    WAIT_UNTIL_FINISH(Line(GetMaxX() / 2 - 5, GetMaxY() - 4, GetMaxX()
/ 2 - 15, GetMaxY() - 14));
    WAIT_UNTIL_FINISH(Line(GetMaxX() / 2 - 5, GetMaxY() - 5, GetMaxX()
/ 2 - 15, GetMaxY() - 15));

    #else
    WAIT_UNTIL_FINISH(Line(GetMaxX() - 5, GetMaxY() / 2 - 5, GetMaxX()
- 5, GetMaxY() / 2 - 15));
    WAIT_UNTIL_FINISH(Line(GetMaxX() - 4, GetMaxY() / 2 - 5, GetMaxX()
- 4, GetMaxY() / 2 - 15));
    WAIT_UNTIL_FINISH(Line(GetMaxX() - 6, GetMaxY() / 2 - 5, GetMaxX()
- 6, GetMaxY() / 2 - 15));
```

```
    WAIT_UNTIL_FINISH(Line(GetMaxX() - 5, GetMaxY() / 2 - 5, GetMaxX()
- 15, GetMaxY() / 2 - 5));
    WAIT_UNTIL_FINISH(Line(GetMaxX() - 5, GetMaxY() / 2 - 4, GetMaxX()
- 15, GetMaxY() / 2 - 4));
    WAIT_UNTIL_FINISH(Line(GetMaxX() - 5, GetMaxY() / 2 - 6, GetMaxX()
- 15, GetMaxY() / 2 - 6));

    WAIT_UNTIL_FINISH(Line(GetMaxX() - 5, GetMaxY() / 2 - 6, GetMaxX()
- 15, GetMaxY() / 2 - 16));
    WAIT_UNTIL_FINISH(Line(GetMaxX() - 5, GetMaxY() / 2 - 4, GetMaxX()
- 15, GetMaxY() / 2 - 14));
    WAIT_UNTIL_FINISH(Line(GetMaxX() - 5, GetMaxY() / 2 - 5, GetMaxX()
- 15, GetMaxY() / 2 - 15));
    #endif
    TouchGetCalPoints();

    SetColor(WHITE);
    ClearDevice();

    SetColor(BLACK);
    WAIT_UNTIL_FINISH(OutTextXY((GetMaxX()-GetTextWidth((XCHAR
*)scr2StrLn1, (void *) &FONTDEFAULT))>>1,  \
                                          textStart + (1*textHeight),
(XCHAR *)scr2StrLn1));
    WAIT_UNTIL_FINISH(OutTextXY((GetMaxX()-GetTextWidth((XCHAR
*)scr2StrLn2, (void *) &FONTDEFAULT))>>1,  \
                                          textStart + (2*textHeight),
(XCHAR *)scr2StrLn2));
    WAIT_UNTIL_FINISH(OutTextXY((GetMaxX()-GetTextWidth((XCHAR
*)scr2StrLn3, (void *) &FONTDEFAULT))>>1,  \
                                          textStart + (3*textHeight),
(XCHAR *)scr2StrLn3));
    WAIT_UNTIL_FINISH(OutTextXY((GetMaxX()-GetTextWidth((XCHAR
*)scr2StrLn4, (void *) &FONTDEFAULT))>>1,  \
                                          textStart + (4*textHeight),
(XCHAR *)scr2StrLn4));
    SetColor(BRIGHTRED);
    WAIT_UNTIL_FINISH(OutTextXY((GetMaxX()-GetTextWidth((XCHAR
*)scr1StrLn7, (void *) &FONTDEFAULT))>>1,  \
                                          textStart + (6*textHeight),
(XCHAR *)scr1StrLn7));
    WAIT_UNTIL_FINISH(OutTextXY((GetMaxX()-GetTextWidth((XCHAR
*)scr1StrLn8, (void *) &FONTDEFAULT))>>1,  \
                                          textStart + (7*textHeight),
(XCHAR *)scr1StrLn8));

    // Wait for touch
    do
    {
        x = ADCGetX();
        y = ADCGetY();
    } while((y == -1) || (x == -1));

    DelayMs(CALIBRATION_DELAY);
```

```
    SetColor(BLACK);
    ClearDevice();
}

/*****************************************************************
* Function: void TouchGetCalPoints(void)
*
* PreCondition: InitGraph() must be called before
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: gets values for 3 touches
*
* Note: none
*
*****************************************************************/
void TouchGetCalPoints(void)
{
    static const XCHAR  calStr[] =
{'C','A','L','I','B','R','A','T','I','O','N',0};
    XCHAR               calTouchLeft[] = {'3','
','t','o','u','c','h','e','s',' ','l','e','f','t',0};
    SHORT               counter;
    SHORT               x, y;
    WORD                ax[3], ay[3];

    SetFont((void *) &FONTDEFAULT);

    SetColor(BRIGHTRED);

    WAIT_UNTIL_FINISH
    (
        OutTextXY
            (
                (GetMaxX() - GetTextWidth((XCHAR *)calStr, (void *)
&FONTDEFAULT)) >> 1,
                (GetMaxY() - GetTextHeight((void *) &FONTDEFAULT)) >>
1,
                (XCHAR *)calStr
            )
    );

    for(counter = 0; counter < 3; counter++)
    {

        SetColor(BRIGHTRED);

        calTouchLeft[0] = '3' - counter;
```

```
        WAIT_UNTIL_FINISH
        (
            OutTextXY
                (
                    (GetMaxX() - GetTextWidth(calTouchLeft, (void *)
&FONTDEFAULT)) >> 1,
                    (GetMaxY() + GetTextHeight((void *) &FONTDEFAULT))
>> 1,
                    calTouchLeft
                )
        );

        // Wait for press
        do
        {
            x = ADCGetX();
            y = ADCGetY();
        } while((y == -1) || (x == -1));

        ax[counter] = x;
        ay[counter] = y;

        // Wait for release
        do
        {
            x = ADCGetX();
            y = ADCGetY();
        } while((y != -1) && (x != -1));

        SetColor(WHITE);

        WAIT_UNTIL_FINISH
        (
            OutTextXY
                (
                    (GetMaxX() - GetTextWidth(calTouchLeft, (void *)
&FONTDEFAULT)) >> 1,
                    (GetMaxY() + GetTextHeight((void *) &FONTDEFAULT))
>> 1,
                    calTouchLeft
                )
        );

        DelayMs(CALIBRATION_DELAY);
    }

    for(counter = 0; counter < 3; counter++)
    {
        if(_calXMax < ax[counter])
            _calXMax = ax[counter];

        if(_calYMin > ay[counter])
            _calYMin = ay[counter];
```

```
        if(_calYMax < ay[counter])
            _calYMax = ay[counter];

        if(_calXMin > ax[counter])
            _calXMin = ax[counter];
    }
}

#endif  // #if defined (USE_RESISTIVE_TOUCH)
```

```
/*
********************************************************************************

Software License Agreement

Copyright © 2007-2008 Microchip Technology Inc.  All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute Software
only when embedded on a Microchip microcontroller or digital signal controller
that is integrated into your product or third party product (pursuant to the
sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for
additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

********************************************************************************
*/

// Created by the Microchip USBConfig Utility, Version 2.0.0.0, 11/18/2008, 8:08:56

#ifndef _usb_config_h_
#define _usb_config_h_

#if defined(__PIC24F__)
    #include <p24fxxxx.h>
#elif defined(__18CXX)
    #include <p18cxxx.h>
#elif defined(__PIC32MX__)
    #include <p32xxxx.h>
    #include "plib.h"
#else
    #error No processor header file.
#endif

#define _USB_CONFIG_VERSION_MAJOR 2
#define _USB_CONFIG_VERSION_MINOR 0
#define _USB_CONFIG_VERSION_DOT   0
#define _USB_CONFIG_VERSION_BUILD 0

// Supported USB Configurations
```

```
#define USB_SUPPORT_HOST

// Hardware Configuration

//USB_PING_PONG__FULL_PING_PONG
#define USB_PING_PONG_MODE  USB_PING_PONG__FULL_PING_PONG

// Host Configuration

#define NUM_TPL_ENTRIES 1
#define USB_NUM_CONTROL_NAKS 200
#define USB_SUPPORT_INTERRUPT_TRANSFERS
#define USB_NUM_INTERRUPT_NAKS 3
#define USB_SUPPORT_BULK_TRANSFERS
#define USB_NUM_BULK_NAKS 20000
//#define USB_SUPPORT_ISOCHRONOUS_TRANSFERS
#define USB_INITIAL_VBUS_CURRENT (100/2)
#define USB_INSERT_TIME (250+1)
#define USB_HOST_APP_EVENT_HANDLER USB_ApplicationEventHandler

// Host Mass Storage Client Driver Configuration

//#define USB_ENABLE_TRANSFER_EVENT

#define USB_MAX_MASS_STORAGE_DEVICES 1

// Helpful Macros

#define USBTasks()                  \
    {                               \
        USBHostTasks();             \
        USBHostMSDTasks();          \
    }

#define USBInitialize(x)            \
    {                               \
        USBHostInit(x);             \
    }


#endif
```

```
/*
********************************************************************************

Software License Agreement

Copyright © 2007-2008 Microchip Technology Inc.  All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute Software
only when embedded on a Microchip microcontroller or digital signal controller
that is integrated into your product or third party product (pursuant to the
sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for
additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

********************************************************************************
*/

// Created by the Microchip USBConfig Utility, Version 2.0.0.0, 11/18/2008, 8:08:56

#include "GenericTypeDefs.h"
#include "HardwareProfile.h"
#include "USB/usb.h"
#include "USB/usb_host_msd.h"
#include "USB/usb_host_msd_scsi.h"

// ******************************************************************************
// Media Interface Function Pointer Table for the Mass Storage client driver
// ******************************************************************************

CLIENT_DRIVER_TABLE usbMediaInterfaceTable =
{
    USBHostMSDSCSIInitialize,
    USBHostMSDSCSIEventHandler,
    0
};

// ******************************************************************************
// Client Driver Function Pointer Table for the USB Embedded Host foundation
```

```
//
************************************************************************
******

CLIENT_DRIVER_TABLE usbClientDrvTable[] =
{
    {
        USBHostMSDInitialize,
        USBHostMSDEventHandler,
        0
    }
};

//
************************************************************************
******
// USB Embedded Host Targeted Peripheral List (TPL)
//
************************************************************************
******

USB_TPL usbTPL[] =
{
    { INIT_CL_SC_P( 8ul, 6ul, 0x50ul ), 0, 0, {TPL_CLASS_DRV} } //
Thumbdrives
};
```

```
/*********************************************************************
* Company: Baylor University
* Engineer: Brandon Herrera
* File Name:  UserInterface.h
* Project Name:  PFTNA
* Target Device:       Control Board Rev A
* Compiler: C32
* Dependencies: M.A.L.
* Description: This is the main oscilloscope like user interface for
*                    the Pulsed Fourier Transform Network Analyzer.
*
*********************************************************************
* Date         Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* 6/6/11     File Created
* 7/9/11    Initial Release
*********************************************************************/
#ifndef _USERINTERFACE_H
#define _USERINTERFACE_H

#include "Graphics/Graphics.h"
#include "main.h"
#include <plib.h>
#include "GenericTypeDefs.h"
#include "HardwareProfile.h"
#include "TimeDelay.h"
#define gentium_8_200_0_0_0_0_SIZE     2360
extern FONT_FLASH gentium_8_200_0_0_0_0;

/*********************************************************************
* USERINTERFACE TEXT
*********************************************************************/
const XCHAR UISmall_OBJ_BUTTON_OPtext[] = "Pow";
const XCHAR UISmall_OBJ_BUTTON_Gtext[] = "Gain";
const XCHAR UISmall_OBJ_BUTTON_DCtext[] = "DC";
const XCHAR UISmall_OBJ_BUTTON_TRtext[] = "Trig";
const XCHAR UISmall_OBJ_BUTTON_Ctext[] = "Cal";
const XCHAR UISmall_OBJ_BUTTON_DPtext[] = "Disp";
const XCHAR UISmall_OBJ_BUTTON_SVtext[] = "Save";
const XCHAR UISmall_OBJ_BUTTON_STtext[] = "Start";
const XCHAR UISmall_OBJ_BUTTON_INTtext[] = "Int";
const XCHAR UISmall_OBJ_BUTTON_EXTtext[] = "Ext";
const XCHAR UISmall_OBJ_BUTTON_CALtext[] = "Cal";
const XCHAR UISmall_OBJ_BUTTON_CLRtext[] = "Clr";
 GOL_SCHEME* pscheme;

/*********************************************************************
* Function: void CreateUISmall(void)
*
* Overview: this function creates the main oscilloscope like user
```

```
*           interface
*
* Input: none
*
* Output: none
*
*****************************************************************************/
void CreateUISmall (void);

/*****************************************************************************
* Function: void CreateGraphData(void)
*
* Overview: this function converts the input data to lcd coordinates
*
* Input: none
*
* Output: none
*
*****************************************************************************/
void CreateGraphData (void);

/*****************************************************************************
* Function: void DrawGraph(void)
*
* Overview: this function draws the lcd coordniates
*
* Input: none
*
* Output: none
*
*****************************************************************************/
void DrawGraph (void);

/*****************************************************************************
* Function: void UISmallMsgCallback(WORD objMsg, OBJ_HEADER *pObj,
GOL_MSG *pMsg)
*
* Overview: this function dictates what happens when a touch is
detected
*
* Input: Any messages from the object, graphics library and the object
that was touched
*
* Output: none
*
*****************************************************************************/
WORD    UISmallMsgCallback(WORD objMsg, OBJ_HEADER *pObj, GOL_MSG
*pMsg);

// These are the object numbers for the various GOL objects
#define UISmall_OBJ_BUTTON_OP   201
#define UISmall_OBJ_BUTTON_G   202
#define UISmall_OBJ_BUTTON_DC   203
#define UISmall_OBJ_BUTTON_TR   204
```

```
#define UISmall_OBJ_BUTTON_C   205
#define UISmall_OBJ_BUTTON_DP  206
#define UISmall_OBJ_BUTTON_SV   207
#define UISmall_OBJ_BUTTON_ST   208

#define UISmall_OBJ_SLIDER_DC 209
#define UISmall_OBJ_SLIDER_OP 210
#define UISmall_OBJ_SLIDER_G  211
#define UISmall_OBJ_SLIDER_TR  212
#define UISmall_OBJ_BUTTON_INT   213
#define UISmall_OBJ_BUTTON_EXT   214
#define UISmall_OBJ_BUTTON_CAL   215
#define UISmall_OBJ_BUTTON_CLR   216

/*********************************************************************
* Macro: WAIT_UNTIL_FINISH()
*
* Overview: this macro waits for drawing to finish
*
* Input: none
*
* Output: none
*
*********************************************************************/
#define WAIT_UNTIL_FINISH(x)    while(!x)

/*********************************************************************
* BUTTON STATES
*********************************************************************/
typedef enum
{
      DISPLAY_MSG  = 0,
    CREATE_OP,
    DISPLAY_OP,
    CREATE_G,
    DISPLAY_G,
    CREATE_DC,
    DISPLAY_DC,
      CREATE_TR,
      CREATE_C,

} SCREEN_SUB_STATES;


#endif //_USERINTERFACE_H
```

```
/*********************************************************************
* Company: Baylor University
* Engineer: Brandon Herrera
* File Name:  UserInterface.c
* Project Name:  PFTNA
* Target Device:       Control Board Rev A
* Compiler: C32
* Dependencies: M.A.L.
* Description: This is the main oscilloscope like user interface for
*                    the Pulsed Fourier Transform Network Analyzer.
*
*********************************************************************
* Date         Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* 6/6/11     File Created
* 7/9/11    Initial Release
*********************************************************************/
#include "UserInterface.h"
// dimensions for graph area
#define ORIGIN_X        0
#define ORIGIN_Y        0
#define PANEL_LEFT      ORIGIN_X
#define PANEL_RIGHT     ORIGIN_X + 270
#define PANEL_TOP       ORIGIN_Y
#define PANEL_BOTTOM    ORIGIN_Y + 209

#define GR_LEFT      (PANEL_LEFT)
#define GR_RIGHT     (PANEL_RIGHT)
#define GR_TOP       (PANEL_TOP)
#define GR_BOTTOM    (PANEL_BOTTOM)

/*********************************************************************
* EXTERNAL VARIABLES
*********************************************************************/
extern unsigned short test_data[];
extern double fout[];
extern double fcal[];


short Graph_Data[280];
short Graph_Data_Old[280];

short FFT_Graph_Data[280];
short FFT_Graph_Data_Old[280];

extern WORD DSA;
extern BYTE DCOFF;
extern BYTE Gain;
extern WORD Trig_Level;
```

```
extern BOOL UseCal;
extern BOOL Save_Data;

extern SCREEN_STATES screenState; //
extern SCREEN_STATES screenStateReturn;


SCREEN_SUB_STATES screenSubState = DISPLAY_MSG; // current state of
screen state machine

/*****************************************************************
* Function: void CreateUISmall(void)
*
* Overview: this function creates the main oscilloscope like user
*           interface
*
* Input: none
*
* Output: none
*
*****************************************************************/
void CreateUISmall (void)
{
      short x, y;
    short *ptr;
      short sy, ey;
    GOLFree();  // free memory for the objects in the previous linked
list and start new list
      SetColor(WHITE);
      ClearDevice();

      if(pscheme != NULL) free(pscheme);
      pscheme = GOLCreateScheme();
/*
      defscheme->Color0 = 19583;
      defscheme->Color1 = 64969;
      defscheme->ColorDisabled = 55070;
      defscheme->CommonBkColor = 55166;
      defscheme->EmbossDkColor = 6172;
      defscheme->EmbossLtColor = 44765;
      defscheme->TextColor0 = 64969;
      defscheme->TextColor1 = 31;
      defscheme->pFont = (void*)&gentium_8_200_0_0_0_0;
*/
      BUTTON *pUISmall_OBJ_BUTTON_OP;
      pUISmall_OBJ_BUTTON_OP =
BtnCreate(UISmall_OBJ_BUTTON_OP,270,0,320,29,0,BTN_DRAW,NULL,(XCHAR*)UI
Small_OBJ_BUTTON_OPtext,pscheme);

      BUTTON *pUISmall_OBJ_BUTTON_G;
      pUISmall_OBJ_BUTTON_G =
BtnCreate(UISmall_OBJ_BUTTON_G,270,30,320,59,0,BTN_DRAW,NULL,(XCHAR*)UI
Small_OBJ_BUTTON_Gtext,pscheme);
```

```
      BUTTON *pUISmall_OBJ_BUTTON_DC;
      pUISmall_OBJ_BUTTON_DC =
BtnCreate(UISmall_OBJ_BUTTON_DC,270,60,320,89,0,BTN_DRAW,NULL,(XCHAR*)U
ISmall_OBJ_BUTTON_DCtext,pscheme);

      BUTTON *pUISmall_OBJ_BUTTON_TR;
      pUISmall_OBJ_BUTTON_TR =
BtnCreate(UISmall_OBJ_BUTTON_TR,270,90,320,119,0,BTN_DRAW,NULL,(XCHAR*)
UISmall_OBJ_BUTTON_TRtext,pscheme);

      BUTTON *pUISmall_OBJ_BUTTON_C;
      pUISmall_OBJ_BUTTON_C =
BtnCreate(UISmall_OBJ_BUTTON_C,270,120,320,149,0,BTN_DRAW,NULL,(XCHAR*)
UISmall_OBJ_BUTTON_Ctext,pscheme);

      BUTTON *pUISmall_OBJ_BUTTON_DP;
      pUISmall_OBJ_BUTTON_DC =
BtnCreate(UISmall_OBJ_BUTTON_DP,270,150,320,179,0,BTN_DRAW,NULL,(XCHAR*
)UISmall_OBJ_BUTTON_DPtext,pscheme);

      BUTTON *pUISmall_OBJ_BUTTON_SV;
      pUISmall_OBJ_BUTTON_SV =
BtnCreate(UISmall_OBJ_BUTTON_SV,270,180,320,209,0,BTN_DRAW,NULL,(XCHAR*
)UISmall_OBJ_BUTTON_SVtext,pscheme);

      BUTTON *pUISmall_OBJ_BUTTON_ST;
      pUISmall_OBJ_BUTTON_ST =
BtnCreate(UISmall_OBJ_BUTTON_ST,270,210,320,239,0,BTN_DRAW,NULL,(XCHAR*
)UISmall_OBJ_BUTTON_STtext,pscheme);

      SLIDER *pUISmall_OBJ_SLIDER_DC;
      SLIDER *pUISmall_OBJ_SLIDER_OP;
      SLIDER *pUISmall_OBJ_SLIDER_G;
      SLIDER *pUISmall_OBJ_SLIDER_TR;
      BUTTON *pUISmall_OBJ_BUTTON_INT;
      BUTTON *pUISmall_OBJ_BUTTON_EXT;
      BUTTON *pUISmall_OBJ_BUTTON_CAL;
      BUTTON *pUISmall_OBJ_BUTTON_CLR;

    switch(screenSubState)
    {
            case CREATE_DC:
                  pUISmall_OBJ_SLIDER_DC =
SldCreate(UISmall_OBJ_SLIDER_DC,0,210,269,239,SLD_DRAW,256,1,DCOFF,psch
eme);
                  //screenSubState = DISPLAY_DC;
                  return;

            case CREATE_OP:
                  pUISmall_OBJ_SLIDER_OP =
SldCreate(UISmall_OBJ_SLIDER_OP,0,210,269,239,SLD_DRAW,31,1,DSA,pscheme
);
                  //screenSubState = DISPLAY_OP;
                  return;
```

```
            case CREATE_G:
                  pUISmall_OBJ_SLIDER_G =
SldCreate(UISmall_OBJ_SLIDER_G,0,210,269,239,SLD_DRAW,256,1,Gain,pschem
e);
                  //screenSubState = DISPLAY_OP;
                  return;

            case CREATE_TR:
                  pUISmall_OBJ_SLIDER_TR =
SldCreate(UISmall_OBJ_SLIDER_TR,0,210,269,239,SLD_DRAW,1024,1,Trig_Leve
l,pscheme);
                  //screenSubState = DISPLAY_OP;
                  return;

            case CREATE_C:

pUISmall_OBJ_BUTTON_INT =
BtnCreate(UISmall_OBJ_BUTTON_INT,0,210,66,239,0,BTN_DRAW,NULL,(XCHAR*)U
ISmall_OBJ_BUTTON_INTtext,pscheme);
pUISmall_OBJ_BUTTON_EXT =
BtnCreate(UISmall_OBJ_BUTTON_EXT,67,210,133,239,0,BTN_DRAW,NULL,(XCHAR*
)UISmall_OBJ_BUTTON_EXTtext,pscheme);
pUISmall_OBJ_BUTTON_CAL =
BtnCreate(UISmall_OBJ_BUTTON_CAL,134,210,200,239,0,BTN_DRAW,NULL,(XCHAR
*)UISmall_OBJ_BUTTON_CALtext,pscheme);
pUISmall_OBJ_BUTTON_CLR =
BtnCreate(UISmall_OBJ_BUTTON_CLR,201,210,267,239,0,BTN_DRAW,NULL,(XCHAR
*)UISmall_OBJ_BUTTON_CLRtext,pscheme);
                  return;
          default:
              return;
        }

}

/*****************************************************************
* Function: void CreateGraphData(void)
*
* Overview: this function converts the input data to lcd coordinates
*
* Input: none
*
* Output: none
*
*****************************************************************/
void CreateGraphData(void)
{
      short x, y;
    short *ptr;
      short sy, ey;
      //ptr = test_data;
    for(x = 0; x <= 280; x++)
    {
```

```
            Graph_Data_Old[x]=Graph_Data[x];
            Graph_Data[x]=test_data[x]/4;

            if(UseCal)
            {
                  FFT_Graph_Data_Old[x]=FFT_Graph_Data[x];
                  FFT_Graph_Data[x]=(fout[x]-fcal[x])*-5.25+52.5;
            }
            else
            {
                  FFT_Graph_Data_Old[x]=FFT_Graph_Data[x];
                  FFT_Graph_Data[x]=fout[x]*-5.25+52.5;
            }

      }

}

/*********************************************************************
* Function: void DrawGraph(void)
*
* Overview: this function draws the lcd coordniates
*
* Input: none
*
* Output: none
*
*********************************************************************/
void DrawGraph(void)
{
      ConfigIntTimer45(T45_INT_OFF | T45_INT_PRIOR_2);
      CreateGraphData();
      short x, y;
    short *ptr;
      short sy, ey;
      SetColor(WHITE);
      ptr = Graph_Data_Old;
    sy = *ptr++;
    for(x = GR_LEFT; x <= GR_RIGHT; x++)
    {
        ey = *ptr++;
        if(ey > sy)
        {
            for(y = sy + GR_TOP; y < ey + GR_TOP + 1; y++)
                PutPixel(x, y);
        }
        else
        {
            for(y = ey + GR_TOP; y < sy + GR_TOP + 1; y++)
                PutPixel(x, y);
        }
        sy = ey;
      }
      ptr = FFT_Graph_Data_Old;
```

```
    sy = *ptr++;
    for(x = GR_LEFT; x <= GR_RIGHT; x++)
    {
        ey = *ptr++;
        if(ey > sy)
        {
            for(y = sy + GR_TOP; y < ey + GR_TOP + 1; y++)
                PutPixel(x, y);
        }
        else
        {
            for(y = ey + GR_TOP; y < sy + GR_TOP + 1; y++)
                PutPixel(x, y);
        }
        sy = ey;
      }


    SetColor(LIGHTGRAY);
    for(x = GR_LEFT + ((GR_RIGHT - GR_LEFT) >> 3); x < GR_RIGHT; x +=
(GR_RIGHT - GR_LEFT) >> 3)
        WAIT_UNTIL_FINISH(Bar(x, GR_TOP, x, GR_BOTTOM));

    for(y = GR_TOP + ((GR_BOTTOM - GR_TOP) >> 3); y < GR_BOTTOM; y +=
(GR_BOTTOM - GR_TOP) >> 3)
        WAIT_UNTIL_FINISH(Bar(GR_LEFT, y, GR_RIGHT, y));


      SetColor(BRIGHTRED);
      ptr = Graph_Data;
    sy = *ptr++;
    for(x = GR_LEFT; x <= GR_RIGHT; x++)
    {
        ey = *ptr++;

        if(ey > sy)
        {
            for(y = sy + GR_TOP; y < ey + GR_TOP + 1; y++)
                PutPixel(x, y);
        }
        else
        {
            for(y = ey + GR_TOP; y < sy + GR_TOP + 1; y++)
                PutPixel(x, y);
        }
        sy = ey;
      }

      SetColor(YELLOW);
      ptr = FFT_Graph_Data;
    sy = *ptr++;
    for(x = GR_LEFT; x <= GR_RIGHT; x++)
    {
        ey = *ptr++;
```

```
        if(ey > sy)
        {
            for(y = sy + GR_TOP; y < ey + GR_TOP + 1; y++)
                PutPixel(x, y);
        }
        else
        {
            for(y = ey + GR_TOP; y < sy + GR_TOP + 1; y++)
                PutPixel(x, y);
        }
        sy = ey;
      }


ConfigIntTimer45(T45_INT_ON | T45_INT_PRIOR_2);
}

/*****************************************************************
* Function: void UISmallMsgCallback(WORD objMsg, OBJ_HEADER *pObj,
GOL_MSG *pMsg)
*
* Overview: this function dictates what happens when a touch is
detected
*
* Input: Any messages from the object, graphics library and the object
that was touched
*
* Output: none
*
*****************************************************************/
WORD UISmallMsgCallback(WORD objMsg, OBJ_HEADER *pObj, GOL_MSG *pMsg)
{
      SLIDER* pSlider;
      switch(GetObjID(pObj))
      {
      //    if(objMsg == BTN_MSG_RELEASED)
      //    {
            case UISmall_OBJ_BUTTON_OP:
                  if(objMsg == BTN_MSG_RELEASED)
                  {
                        screenSubState = CREATE_OP;
                        screenState = CREATE_UISMALL;
                  }
                  return 1;

            case UISmall_OBJ_BUTTON_G:
                  if(objMsg == BTN_MSG_RELEASED)
                  {
                        screenSubState = CREATE_G;
                        screenState = CREATE_UISMALL;
                  }
                  return 1;
```

```
            case UISmall_OBJ_BUTTON_DC:
                  if(objMsg == BTN_MSG_RELEASED)
                  {
                        screenSubState = CREATE_DC;
                        screenState = CREATE_UISMALL;
                  }
                  return 1;

            case UISmall_OBJ_BUTTON_TR:
                  if(objMsg == BTN_MSG_RELEASED)
                  {
                        screenSubState = CREATE_TR;
                        screenState = CREATE_UISMALL;
                  }
                  return 1;


            case UISmall_OBJ_BUTTON_C:
                  if(objMsg == BTN_MSG_RELEASED)
                  {
                        screenSubState = CREATE_C;
                        screenState = CREATE_UISMALL;
                  }
                  return 1;

            case UISmall_OBJ_BUTTON_DP:

                  return 1;

            case UISmall_OBJ_BUTTON_SV:
                  if(objMsg == BTN_MSG_RELEASED)
                  {
                        Save_Data=1;
                  }
                  return 1;


            case UISmall_OBJ_BUTTON_ST:
            //    TakeData();
                  screenState = CREATE_UISMALL;
                  return 1;


            case UISmall_OBJ_SLIDER_OP:
                  pSlider =
(SLIDER*)GOLFindObject(UISmall_OBJ_SLIDER_OP);
                  DSA = (XCHAR)SldGetPos(pSlider);  //Get the position
                  SetAttenuator(DSA);
                  //sprintf(DSAstr, "%d", DSA);      //Convert to
string
                  //EbSetText(pEb,(XCHAR*)DSAstr);   //Write to text
box
                  //SetState(pEb, EB_DRAW);
                  return 1;
```

```
            case UISmall_OBJ_SLIDER_DC:
                  pSlider =
(SLIDER*)GOLFindObject(UISmall_OBJ_SLIDER_DC);
                  DCOFF = (XCHAR)SldGetPos(pSlider);  //Get the
position
                  DACProgram(Channel_D,DCOFF,0b00000000);
//                TakeData();
                  //screenState = CREATE_UISMALL;
                  //sprintf(DSAstr, "%d", DSA);      //Convert to
string
                  //EbSetText(pEb,(XCHAR*)DSAstr);   //Write to text
box
                  //SetState(pEb, EB_DRAW);
                  return 1;

            case UISmall_OBJ_SLIDER_G:
                  if(objMsg == OBJ_MSG_PASSIVE)
                  {
                        pSlider =
(SLIDER*)GOLFindObject(UISmall_OBJ_SLIDER_G);
                        Gain = (XCHAR)SldGetPos(pSlider);  //Get the
position
                        POTProgram(Gain);
                  }
                  return 1;

            case UISmall_OBJ_SLIDER_TR:
                  if(objMsg == OBJ_MSG_PASSIVE)
                  {
                        pSlider =
(SLIDER*)GOLFindObject(UISmall_OBJ_SLIDER_TR);
                        Trig_Level = (XCHAR)SldGetPos(pSlider);  //Get
the position
                  }
                  return 1;

            case UISmall_OBJ_BUTTON_INT:
                  if(objMsg == BTN_MSG_RELEASED)
                  {
                        Switch_TX_LAT = 1;
                        Switch_RX_LAT = 0;
                  }
                  return 1;

            case UISmall_OBJ_BUTTON_EXT:
                  if(objMsg == BTN_MSG_RELEASED)
                  {
                        Switch_TX_LAT = 0;
                        Switch_RX_LAT = 1;
                  }
                  return 1;

            case UISmall_OBJ_BUTTON_CAL:
```

```
                if(objMsg == BTN_MSG_RELEASED)
                {
                        memcpy(fcal,fout,2048);
                        UseCal = 1;
                }
                return 1;

        case UISmall_OBJ_BUTTON_CLR:
                if(objMsg == BTN_MSG_RELEASED)
                {
                        UseCal = 0;
                }
                return 1;

        default:
                return 1;

    }
}
```

BIBLIOGRAPHY

[1] E.C. Green, “Design of a Microwave Sensor for Non-Invasive Determination of Blood-Glucose Concentration.” M.S. thesis, Dept. of ECE, Baylor University, Waco, TX, 2005.

[2] Herrera, B.J.; Jean, B.R., "A low cost ultra-wideband pulse transceiver," *System Theory (SSST), 2010 42nd Southeastern Symposium on* , vol., no., pp.72-74, 7-9 March 2010

[3] Jean, B.R.; Green, E.C.; McClung, M.J., "A microwave frequency sensor for non-invasive blood-glucose measurement," *Sensors Applications Symposium, 2008. SAS 2008. IEEE* , vol., no., pp.4-7, 12-14 Feb. 2008

[4] M. J. McClung, “Calibration Methodology for a Microwave Non-Invasive Glucose Sensor,” M.S. thesis, Dept. of ECE, Baylor University, Waco, TX, 2008.

[5] I. Sorka, *Concrete in Hot Environments*. London: Chapman & Hall, 2004, p. 128.

[6] World Business Council for Sustainable Development. The Cement Sustainability Initiative: Progress report. June, 2002.

[7] Chris Owen, “Owen Resistive Splitter,” May 2007. [Online]. Available: http://www.microwaves101.com/encyclopedia/Resistive_splitter2.cfm [Accessed: Dec 10, 2009].

[8] G. Adams, “Designing Resistive Unequal Power Dividers,” *High Frequency Electronics*, pp. 48-50, Mar., 2007.

[9] Jianping Lai; Qing Hao; Jingzhao She; Zhenghe Feng, "A low cost trigger frequency alterable ultra-wide band ambipolar pulses generator," *Microwave and Millimeter Wave Technology, 2008. ICMMT 2008. International Conference on* , vol.1, no., pp.216-219, 21-24 April 2008

[10] Wun-Bi Lin; Ying-Te Liu; Fu-Chiarng Chen, "A New Ultra-Wideband Monocycle Pulse Generator Using Second-Order Transient Circuit," Microwave Conference, 2008. EuMC 2008. 38th European , vol., no., pp.1585-1588, 27-31 Oct. 2008.

[11] Sanghoon Sim; Dong-Wook Kim; Songcheol Hong, "A CMOS UWB Pulse Generator for 6–10 GHz Applications," *Microwave and Wireless Components Letters, IEEE* , vol.19, no.2, pp.83-85, Feb. 2009

[12] Jean, Buford R., F. L. Whitehead and J. L. Daniewicz, "Ultra Wideband Pulse Dispersion Spectrometry Method and Apparatus Providing Multi-Component Composition Analysis", U. S. Patent 6,987,393, January 17, 2006.

[13] Cemin Zhang; Fathy, A.E.; Mahfouz, M., "Performance Enhancement of a Sub-Sampling Circuit for Ultra-Wideband Signal Processing," *Microwave and Wireless Components Letters, IEEE* , vol.17, no.12, pp.873-875, Dec. 2007

[14] Jeongwoo Han; Cam Nguyen, "Coupled-slotline-hybrid sampling mixer integrated with step-recovery-diode pulse generator for UWB applications," *Microwave Theory and Techniques, IEEE Transactions on* , vol.53, no.6, pp. 1875-1882, June 2005.