Missouri S&T Library and Learning Resources

Scholars' Mine

Masters Theses

Student Theses and Dissertations

Fall 2007

A software defined radio for research into cognitive radio

Jason Michael Pratt

Follow this and additional works at: https://scholarsmine.mst.edu/masters_theses

Part of the Electrical and Computer Engineering Commons

Department:

Recommended Citation

Pratt, Jason Michael, "A software defined radio for research into cognitive radio" (2007). *Masters Theses*. 4592.
https://scholarsmine.mst.edu/masters_theses/4592

This thesis is brought to you by Scholars' Mine, a service of the Missouri S&T Library and Learning Resources. This work is protected by U. S. Copyright Law. Unauthorized use including reproduction for redistribution requires the permission of the copyright holder. For more information, please contact scholarsmine@mst.edu.

A SOFTWARE DEFINED RADIO FOR RESEARCH INTO COGNITIVE RADIO

by

JASON MICHAEL PRATT

A THESIS

Presented to the Faculty of the Graduate School of the

UNIVERSITY OF MISSOURI-ROLLA

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

2007

Approved by

_______________________________ Kurt Kosbar, Advisor

_______________________________ Steven Grant, Committee Member

_______________________________ Daryl Beetner, Committee Member

ABSTRACT

Software defined radios are capable of transmitting and receiving signals over a broad range of frequencies, and can alter their carrier frequency, modulation scheme, and power level by changing the software controlling the radio. Because of these capabilities, a software defined radio is the necessary platform for research into cognitive radio. The aim of cognitive radio is to more fully utilize the RF spectrum by allowing wireless communication systems to scan the spectrum for unused frequency bands and dynamically establish communication over one of those unused bands. This thesis gives a brief introduction into cognitive radio, and currently existing software defined radio systems. It then describes a software defined radio designed at the IFT/UMR Telemetry Learning Center for future research into cognitive radio. It also gives the results of testing done on the radio to verify its operation.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Kurt Kosbar, for his guidance in and support through the design and testing of this radio. I would also like to thank my committee members, Dr. Steven Grant and Dr. Daryl Beetner for their assistance and for agreeing to be on my committee. Thanks also to Sarah Seguin, who helped greatly in the testing of the software defined radio. I would also like to thank the University of Missouri – Rolla and the School of Engineering for the Chancellor's Fellowship, which greatly helped in continuing my education. Lastly, I would like to thank my parents for their continued support throughout my college career and especially through my graduate degree program.

TABLE OF CONTENTS

Page

ABSTRACT iii

ACKNOWLEDGMENTS iv

LIST OF ILLUSTRATIONS vii

SECTION

1. INTRODUCTION 1
    1.1. COGNITIVE RADIO 1
    1.2. EXISTING SOFTWARE DEFINED RADIO PLATFORMS 2
        1.2.1. The Joint Tactical Radio System 2
        1.2.2. The Universal Software Radio Peripheral 2
2. SOFTWARE DEFINED RADIO DESIGN 3
    2.1. OVERVIEW 3
    2.2. RECEIVER RF FRONT END 4
        2.2.1. Antenna 4
        2.2.2. Low Noise and Variable Gain Amplifiers 4
        2.2.3. I/Q Demodulator 5
        2.2.4. Frequency Synthesizer and Voltage Controlled Oscillator 5
        2.2.5. Anti-Aliasing Filters 6
    2.3. ANALOG TO DIGITAL CONVERSION 7
        2.3.1. Analog to Digital Converters 8
        2.3.2. Digital Down Converters 8
        2.3.3. Field Programmable Gate Array 8
    2.4. TRANSMITTER RF FRONT END 10
        2.4.1. Antenna 10
        2.4.2. Variable Gain Amplifiers 10
        2.4.3. I/Q Modulator 10
        2.4.4. Frequency Synthesizer and Voltage Controlled Oscillator 10
        2.4.5. Reconstruction Filters 10
    2.5. DIGITAL TO ANALOG CONVERSION 11

2.6. DIGITAL SIGNAL PROCESSING 12
2.7. MICROCONTROLLER 13
2.7.1. USB Controller 13
2.7.2. LCD Display 14
2.8. PBC LAYOUT 15
3. SOFTWARE DEFINED RADIO TESTING 16
3.1. ANTENNA 16
3.2. VOLTAGE CONTROLLED OSCILLATOR 18
3.3. RECEIVER 19
3.4. TRANSMITTER 21
3.5. MICROCONTROLLER 24
4. ANALYSIS 26
4.1. LINK BUDGET ANALYSIS 26
4.2. DYNAMIC RANGE 28
4.3. SUGGESTED MODIFICATIONS 28
5. CONCLUSION 29
APPENDICES
A. SOFTWARE DEFINED RADIO SCHEMATICS 30
B. SOFTWARE DEFINED RADIO PCB LAYOUT 43
C. VOLTAGE CONTROLLED OSCILLATOR TESTING RESULTS 48
D. RECEIVER DIGITAL SIGNAL PROCESSOR CODE 52
E. TRANSMITTER DIGITAL SIGNAL PROCESSOR CODE 84
F. TRANSMITTER TESTING SETUP PICTURES 113
G. MICROCONTROLLER CODE 116
H. SOFTWARE DEFINED RADIO PARTS LIST 164
I. SOFTWARE DEFINED RADIO PICTURES 169
BIBLIOGRAPHY 171
VITA 174

LIST OF ILLUSTRATIONS

Page

Figure 2.1: SDR System Overview, Copyright 2007, International Foundation for Telemetering, Used by Permission [1] ... 4

Figure 2.2: PLL Designed by ADIsimPLL 3.0 ... 7

Figure 2.3: FPGA Program Design ... 9

Figure 2.4: PSpice Differential Current to Single-Ended Voltage ... 11

Figure 2.5: PSpice Simulation Results ... 11

Figure 2.6: LabVIEW GUI for USB Communication ... 14

Figure 3.1: BB85 Antenna S11 Curve ... 17

Figure 3.2: BB85 Antenna Reflection Efficiency ... 18

Figure 3.3: VCO Output at 1.6 GHz ... 19

Figure 3.4: LNA Output 20MHz Input ... 20

Figure 3.5: VGA Output ... 20

Figure 3.6: Data Received by Receiver DSP ... 21

Figure 3.7: Transmitting only Carrier Frequency ... 23

Figure 3.8: Transmitter DSP Transmitting 1.17 kHz Sine Wave with 1.6 GHz Carrier ... 23

Figure 3.9: Transmitter DSP Transmitting $e^{\{j2\pi(1.17\ kHz)t\}}$ with a 1.6 GHz Carrier ... 24

Figure 3.10: Successful LCD Display Test ... 25

1. INTRODUCTION

1.1. COGNITIVE RADIO

Current wireless communication systems are restricted in the bands they can transmit over, and the modulation schemes they can use. These restrictions mean that, as demands increase on wireless system performance, there is not enough bandwidth in a system's portion of the spectrum to meet those demands. For example, cell phones now come with the ability to access the internet, but the connection speeds are slow due to the limited spectrum that cell phones are allowed to operate over. However, a large portion of the spectrum is not being used in any given geographic location, including portions of the spectrum that are licensed, but not being utilized. The most notable example of this underutilization is the UHF television bands (Channels 14 to 83), many of which are never used in a given location [2]. If new wireless communication systems could determine which of these licensed bands are not being used and establish communication over them, it would solve the bandwidth problems of many wireless systems and better utilize the radio frequency (RF) spectrum. This type of system is called a cognitive radio system [2].

A cognitive radio system would be able to scan the RF spectrum to determine the portions of the spectrum that are not being utilized (called *white spaces* [3]) and then establish communication with one or more other radios over those bands. They would also be able to change their modulation technique and power level to optimally use the RF spectrum. However, current wireless systems use application specific hardware to communicate over specific frequencies with a specific modulation technique. Therefore, cognitive radios would need a new system that would perform most of the signal processing in software, thus allowing the system to dynamically change its carrier frequency and modulation scheme based on the current utilization of the spectrum. Such systems are called software defined radios (SDR) [2]. Software defined radios first convert a received signal to an intermediate frequency (IF) and then use a wideband analog to digital converter to capture a large portion of the spectrum. This data is then

---

[1] Portions of this work will be presented at the 2007 International Telemetry Conference [1].

passed through a digital down converter to extract the desired band and convert it down to base-band and sent to a digital signal processor (DSP), or general purpose processor for processing [3 - 6].

1.2. EXISTING SOFTWARE DEFINED RADIO PLATFORMS

There are already SDRs in existence and in development. Two such systems are the military's Joint Tactical Radio System (JTRS) [7,8] and a system called the Universal Software Radio Peripheral (USRP) designed by Ettus Research [9].

1.2.1. The Joint Tactical Radio System. The JTRS is a system still under development to replace the military's existing radio systems, many of which are unique to a certain branch of the military and incompatible with other branches. The JTRS is a software defined radio meant to replace all legacy radio systems and bring compatibility between the branches' communication systems. The JTRS consists of several platforms designed to meet a particular operating environment, such as platforms for ground troops, planes and ships. These platforms are all SDRs and are loaded with software to determine which bands they can operate over. However, these systems are not used for cognitive radio. They simply change software to communicate over established protocols [7,8].

1.2.2. The Universal Software Radio Peripheral. The USRP, however, is designed for research into different types of wireless communication. The system consists of a main board which communicates over USB to a computer and has slots for up to 4 daughterboards, which act as the RF front end for the system. The main board handles the analog to digital and digital to analog conversion of the signals (with an operation bandwidth of up to 16 MHz) and communication with a computer. The portion of the spectrum the device can transmit and receive over is determined by which daughterboards are used with the system, but can range from DC to 2.9 GHz [9]. This device was not used because it required a computer to be connected to the device to perform the signal processing required for different modulation techniques instead of being able to perform the signal processing in the SDR's hardware. It also has a lower bandwidth than the system outlined in this thesis.

2. SOFTWARE DEFINED RADIO DESIGN

2.1. OVERVIEW

An overview of the design of the SDR developed at the International Foundation for Telemetering (IFT) / University of Missouri – Rolla (UMR) Telemetry Learning Center is shown in Figure 2.1. This SDR is capable of setting its carrier frequency independently on both the transmitting and receiving sides. The carrier frequency can be varied from 1 GHz to 2 GHz in increments of 1 kHz. On the receiving side, the amplified signal received from the antenna is down-converted with an In-Phase/Quadrature-Phase (I/Q) demodulator into its I and Q components. These signals are then passed through anti-aliasing filters with a cutoff frequency of 25 MHz. Once filtered, the signals are then sampled with analog to digital converters (ADCs) running at 50 Megasamples per second (MSPS). The samples from the two ADCs are passed into two digital down converters (DDCs), which can digitally filter and down-sample the output of the ADCs. Each of the DDCs outputs both I and Q data, which is then combined in a field programmable gate array (FPGA) to provide the receiver digital signal processor (DSP) with a single I/Q pair at a sampling rate of 25 MSPS, giving a maximum bandwidth of 25 MHz on the receiver side of the SDR.

On the transmitter side of the SDR, the message signal is created by two digital to analog converters (DACs) connected to the transmitter DSP. Each of the DACs is clocked by the same signal provided by the DSP, which allows for a variable sample rate up to 30 MSPS. The output of the DACs then goes through reconstruction filters with a cutoff of 25 MHz to reduce any higher frequencies induced from the digital to analog conversion. These I and Q signals are then passed into an I/Q modulator to up-convert them the desired carrier frequency. The final output power can then be adjusted by adjusting the gain on a variable gain amplifier (VGA).

The microcontroller in the SDR is responsible for relaying communication between the two DSPs and a computer connected via USB. The microcontroller can also connect to a 20 character by 4 line LCD display to output text messages to the user.

A detailed schematic for the software defined radio can be found in Appendix A.

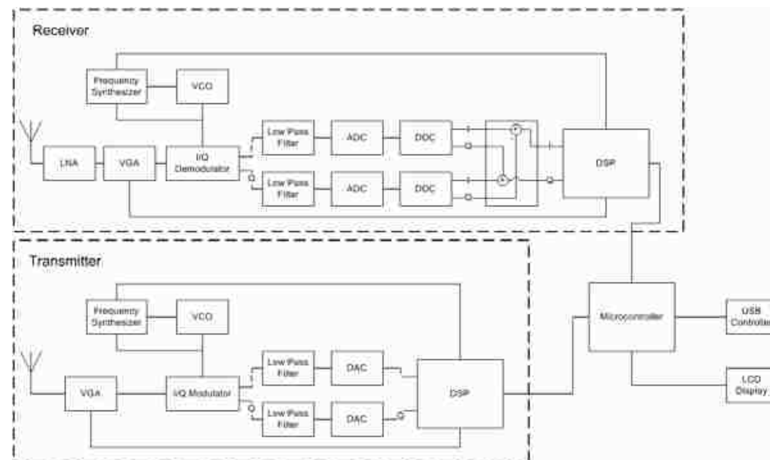


Figure 2.1: SDR System Overview, Copyright 2007, International Foundation for Telemetering, Used by Permission [1]

2.2. RECEIVER RF FRONT END

The RF front end of the receiver side of the SDR consists of low noise and variable gain amplifiers that take in the signal picked up by the antenna and amplify them to an appropriate level. The signal then goes through an I/Q demodulator which down-converts the signal to baseband. The local oscillator signal for the demodulator is provided by a frequency synthesizer and voltage controlled oscillator (VCO). The final stage of the RF front end is the anti-aliasing filters used to reject higher frequencies before the signal is sampled.

2.2.1. Antenna. The antenna chosen for use with this SDR is model number BB85 sold by Optoelectronics™. This antenna has a single-ended connection to the SDR via a BNC connector. Its operational frequency range is from 100 MHz to 2 GHz, which covers the range of the voltage controlled oscillator used in the radio, with has a range from 1 GHz to 2 GHz [10].

2.2.2. Low Noise and Variable Gain Amplifiers. The first component the received signal reaches is a low noise amplifier (LNA). The AD8354 produced by Analog Devices™. This device was chosen because it can operate over the useful frequency range of the SDR and it provides a single-ended input, which is internally matched to 50 Ohms. It is also capable of operating off of a single 5V supply, which is

used throughout the RF front end, and has a fixed gain of 20 dB and a noise figure of only 4.4 dB at 1 GHz and only 5.1 dB at 2 GHz [11].

Following the LNA is a variable gain amplifier produced by Analog Devices™, part number ADL5330. This component is capable of operating from 10 MHz to 3 GHz and has an adjustable gain from approximately -35 dB to 20 dB. This part utilizes differential inputs, so a balun transformer was used to convert the single-ended output of the LNA to differential to be used as the input for the VGA [12].

A possible improvement to the amplification section of the receiver RF front end would be to replace the VGA described above with one or more of the fixed gain LNAs also used in this portion of the system. This would allow for higher total gain in the system and the I/Q demodulator, described in Section 2.2.3 has a feature to adjust its gain to provide a given peak-to-peak voltage at its output.

2.2.3. I/Q Demodulator. The I/Q demodulator is used to down-convert the received signal and separate it into I and Q components. The device chosen to do this is the AD8347 produced by Analog Devices™. This I/Q demodulator is capable of using local oscillator frequencies from 800 MHz to 2.7 GHz and has a variable gain control capable of amplifying the signal from -30 dB to 39.5 dB with a noise figure of 11 dB at maximum gain. The gain on the I/Q demodulator can be controlled either with an external voltage or it can be connected to an on-chip automatic gain control which can provide a differential output of 3.8V. For the purposes of this system, the AD8347 was configured to utilize the automatic gain control functionality of the chip with an output of up to 3.8V differential [13].

2.2.4. Frequency Synthesizer and Voltage Controlled Oscillator. The local oscillator for the I/Q demodulator is provided by a fractional N frequency synthesizer produced by Analog Devices™, the ADF4153, which is capable of synthesizing frequencies up to 4 GHz . This component can be used with a PLL circuit to control a voltage controlled oscillator. Because it is a fractional N frequency synthesizer, it can control the output of a VCO to be a fractional multiple of a reference frequency. For this system, the frequency synthesizer uses reference clock of 20 MHz, which the frequency synthesizer can divide down to create its reference frequency (division factor is programmed by the DSP). The RF output of the VCO is then controlled by programming

the frequency synthesizer with three values: integer, fraction and modulus. The final output of the VCO is then given by equations (1) and (2). In these equations, F_I is the input reference frequency (20 MHz), D is the F_I doubler bit, R is the preset divide ratio of the reference counter, I is the preset divide ratio of the feedback counter, M is the preset fractional modulus, and F is the numerator of the fractional division. All values other than F_I are programmed by the DSP connected to the frequency synthesizer via a serial peripheral interface (SPI) connection [14].

$$F_O = F_P \times \left(I + F/M\right) \tag{1}$$

$$F_P = F_I \times (1 + D)/R \tag{2}$$

The Phase Locked Loop (PLL) filter for this system was designed using software provided by Analog Devices™, ADIsimPLL 3.0. With this software, the user can specify a given frequency synthesizer and VCO, along with the architecture of loop filter desired, and the program will output a schematic for the PLL loop. The filter used in this PLL is a low pass, active filter with a 15 kHz cutoff frequency. This is connected between the charge pump output of the frequency synthesizer and the voltage input of the VCO. The suggested schematic given by ADIsimPLL 3.0 is shown in Figure 2.2.

The VCO for this system was chosen to be the CVCO55BE-1000-2000 manufactured by Crystek Corporation™. This VCO was chosen because of its wide frequency range from 1 GHz to 2 GHz with a tuning sensitivity of 52 MHz/V and a typical output power of +5.5 dBm, which was sufficient to drive the local oscillator input of the frequency synthesizer. Other VCOs with the same footprint could be placed into this system to achieve a different range of operating frequencies, as long as the range stayed within the operational range of the I/Q demodulator, as it is the limiting factor in the system for carrier frequency [15].

2.2.5. Anti-Aliasing Filters. The filters used as anti-aliasing filters are the MEM2012T25R0 three-terminal filters for signal lines produced by TDK™. These filters provide a 20 dB insertion loss from 80 MHz to 1 GHz, effectively reducing the frequencies outside of the desired frequency band. However, these filters are single-

ended, so an operation amplifier circuits is used to convert the differential output of the I/Q demodulator to single-ended and another op amp circuit is used to convert the single-ended output of the filter back to differential for the analog to digital converters. Op amp circuits were used to preserve the baseband frequencies present and to provide a gain of 0.56 on the output of the I/Q demodulator to convert the 3.8V maximum output of the I/Q demodulator to be within the 2.2V input limit of the ADCs [16].

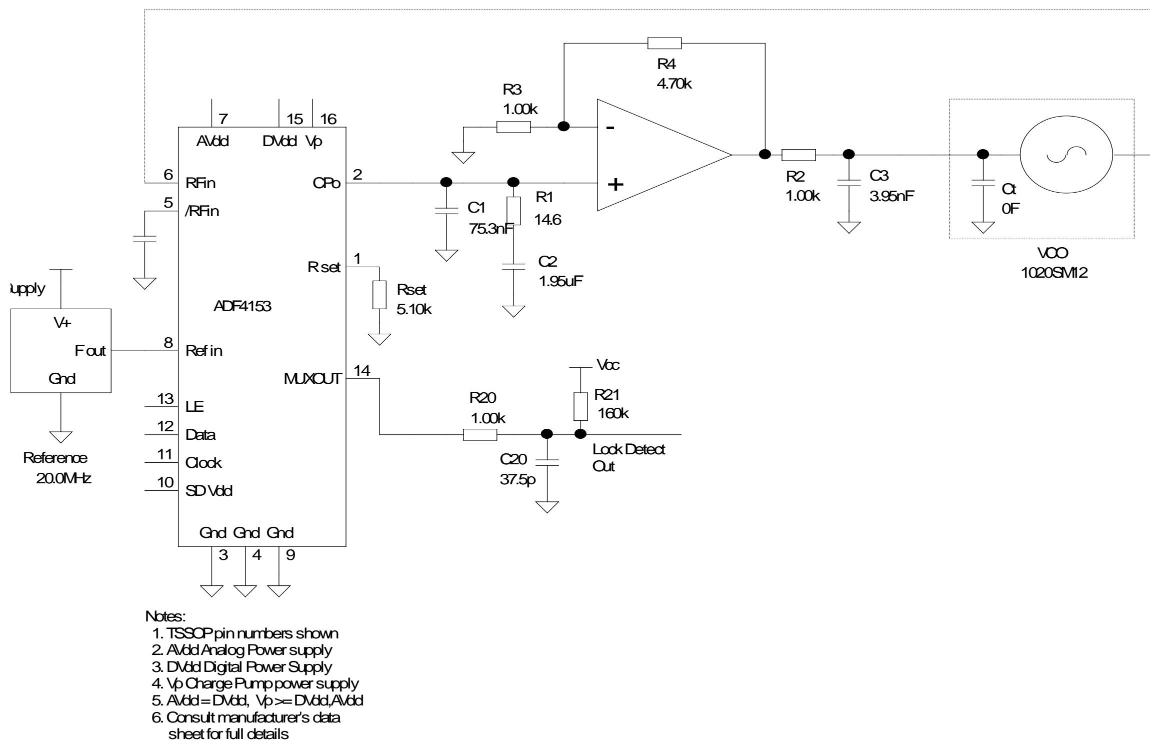


Figure 2.2: PLL Designed by ADIsimPLL 3.0

2.3. ANALOG TO DIGITAL CONVERSION

The analog to digital conversion section of the receiver consists of two high speed analog to digital converts to sample the I and Q signals outputted by the I/Q demodulator. The outputs of the ADCs then go into two digital down converters, where the signals are further filtered and down-sampled. The complex outputs of the two DDCs are then

passed through a FPGA used to add the two I/Q pairs to give a single I/Q pair, which is passed to the receiver DSP.

2.3.1. Analog to Digital Converters. The analog to digital converters chosen for this SDR are the AD6644 14-bit, 65 MSPS analog to digital converters produced by Analog Devices™. However, even though the ADCs have a 14-bit output, they only provide an 11-bit precision as the typical thermal noise for the components takes up the lower 2.5 bits. These ADCs share a 50 MHz clock signal to ensure the I and Q signals are sampled in phase with each other [17].

2.3.2. Digital Down Converters. The digital down converters chosen are also produced by Analog Devices™: part number AD6620. These DDCs are capable of down-converting up to 67 MSPS for a single channel real signal, as the ADCs output. This component has a numerically controlled oscillator capable of further up- or down-converting the digital signal if necessary before passing the signal through three decimation filters. The first decimation filter is a 2nd order cascaded integrator comb FIR filter capable of decimation rates from 2 to 16. The second filter is a 5th order cascaded integrator comb FIR filter that can further decimate the signal by a factor from 1 to 32. Finally, the DDC provides a programmable FIR decimation filter that can have up to 256 20-bit coefficients and can further decimate the signal by a factor from 1 to 32. As the DDCs are clocked by the 50 MHz data ready signal provided by the ADCs, the output of the DDCs can go up to 25 MSPS (complex), giving the receiver side of the SDR a usable bandwidth of 25 MHz. The I and Q outputs of the DDC are interleaved using the same 16-bit port and a I/Q output is used to determine the current byte available on the parallel output port. These DDCs also share synchronization signals to ensure that the down-sampling is done in phase between the two components [18].

2.3.3. Field Programmable Gate Array. An FPGA was used in this system to perform the complex addition required to combine the complex outputs of the DDCs. This is done on an FPGA to reduce the computational burden on the DSP.

The requirements for the FPGA in this system are fairly low, given the standards of modern FPGAs. This allowed the use of a Xilinx™ Spartan-3 FPGA with only 50,000 gates, the XC3S50. A 1 Megabit platform flash PROM was placed in the JTAG

boundary scan chain before the FPGA to store the FPGA's program and load it on power-up.

The program designed for the FPGA is shown in Figure 2.3. The program uses the "data valid", clock and I/Q signals from the DDCs to determine when to latch the output of the DDC as either an I or Q word. Once the data is latched, the imaginary component of the Q data is subtracted from the real component of the I data to give the real component of the combined data. Also, the imaginary component of the I data is added to the real component of the Q data to yield the imaginary component of the combined data. These values are latched three 50 MHz clock cycles after the I and Q data are first latched into the system. This allows plenty of time for the signals to propagate through the adders. One clock cycle after the outputs are latched, the data ready line is pulsed low for one clock cycle to trigger an interrupt in the receiver DSP [19].

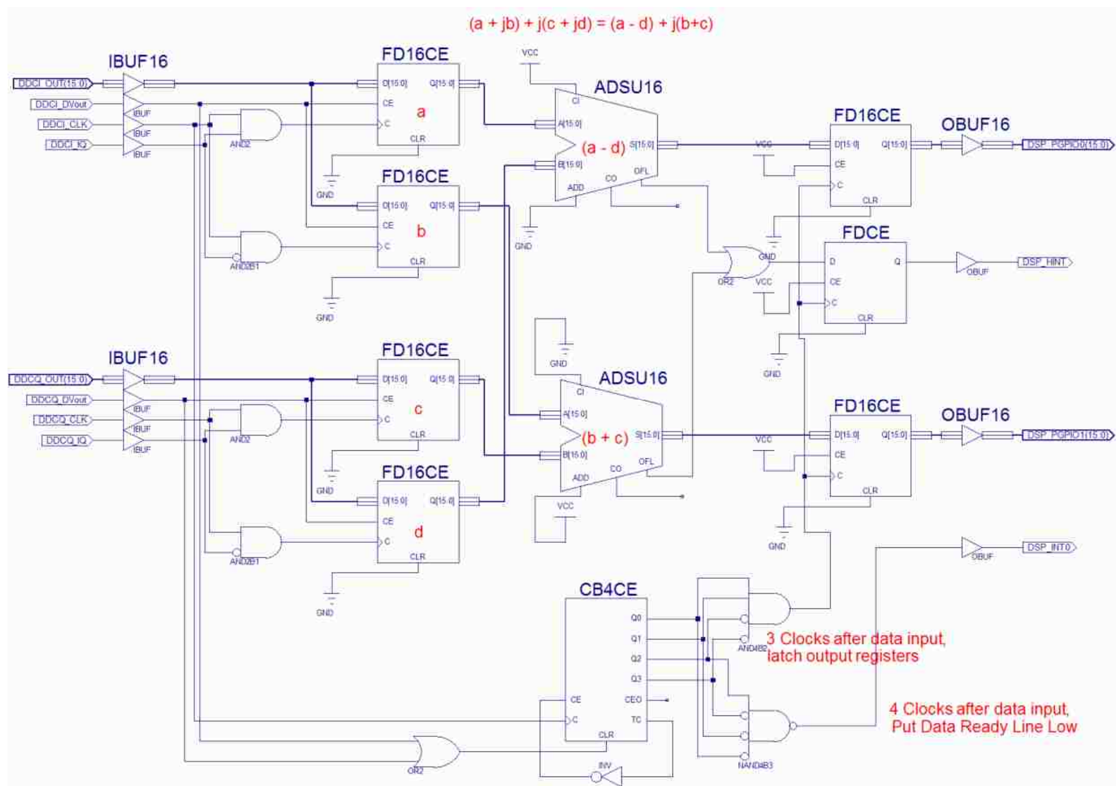


Figure 2.3: FPGA Program Design

2.4. TRANSMITTER RF FRONT END

The transmitter RF front end is similar in design to the receiver RF front end. After the digital to analog converters, separate I and Q signals go through reconstruction filters to remove any higher frequency harmonics in the signal and then go into an I/Q modulator. The I/Q modulator up-converts the signal with a local oscillator input from a PLL circuit with the same design as on the receiver side. Once the signal has been up-converted to the desired carrier frequency, it goes through a variable gain amplifier to adjust the output power of the signal. The signal is then passed to a BNC connector which is connected to the same antenna used on the receiver side.

2.4.1. Antenna. The transmitter side of the SDR uses the same type of antenna as the receiver side. For more information, see Section 2.2.1.

2.4.2. Variable Gain Amplifiers. The transmitter side of the radio utilizes the same VGA as the receiver side, the ADL5330. For more information, see Section 2.2.2.

2.4.3. I/Q Modulator. The I/Q modulator used was the AD8346, which has a frequency range from 800 MHz to 2.5 GHz and a modulation bandwidth from 0 to 70 MHz. The inputs of the modulator are connected to the outputs of the digital to analog converter through a reconstruction filter and op amp circuit used to convert the current outputs of the DACs to voltage signals (see Section 2.4.4). The outputs of the modulator are connected to a VGA for output power control. The local oscillator is drive by a VCO and PLL circuit [20].

2.4.4. Frequency Synthesizer and Voltage Controlled Oscillator. The VCO and frequency synthesizer used on the transmission side of the SDR are identical to the circuit used on the receiving side. See Section 2.2.4 for more details on the circuit.

2.4.5. Reconstruction Filters. The reconstruction filters used on the output of the DACs are the same filters used as anti-aliasing filters on the receiver side (MEM2012T25R0). However, to convert the differential current outputs of the DACs to single-ended, an op amp circuit was used that produces a peak-to-peak voltage of approximately 4 Vp-p centered at 2.5 V, as seen in the PSpice simulation in Figure 2.4 and Figure 2.5. Once the signals have gone through the reconstruction filters, they are converted back to differential using a differential op amp circuit.

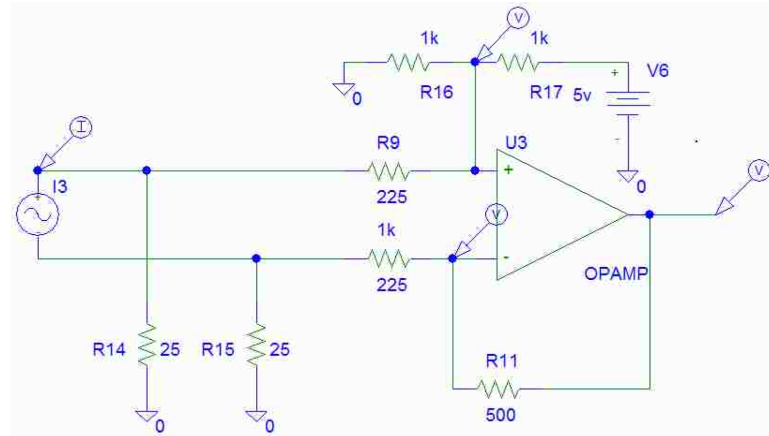


Figure 2.4: PSpice Differential Current to Single-Ended Voltage

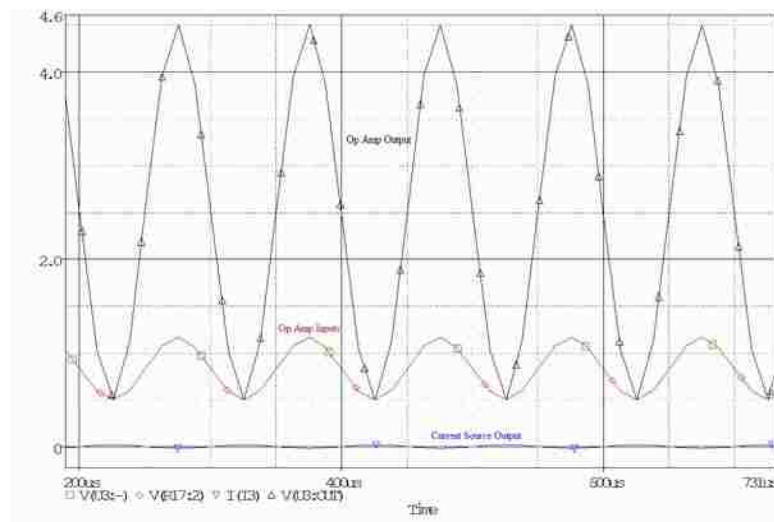


Figure 2.5: PSpice Simulation Results

2.5. DIGITAL TO ANALOG CONVERSION

The digital to analog converters used in this SDR are AD9774, 14-bit, 32 MSPS DACs. These DACs also have 4x interpolation filters on the outputs to reduce harmonic noise caused by stepping from one value to the next and operate from a single 3.3 V supply for the digital side and a 5 V supply for the analog side. The input of the DAC is through a parallel bus that is connected to two of the parallel general purpose I/O ports of

the transmitter DSP. The outputs of the DACs are current outputs and the full-scale current can be adjusted from 2 mA to 20 mA by connecting different resistance values to the full-scale output adjust pin of the device. For the purposes of this radio, the full-scale output was adjusted to 20 mA by placing a 2 kOhm resistor at the full-scale current adjust pin of the device [21].

2.6. DIGITAL SIGNAL PROCESSING

The digital signal processors are the workhorses of the software defined radio. They are responsible for determining the carrier frequency and sampling rate, as well as all the modulation and other signal processing that must take place. Because of this, the TMS320VC5502-300 by Texas Instruments™ (TI) was chosen for this SDR. This DSP is capable of running at processing speeds of up to 300 MHz and has the necessary I/O capabilities to communicate with all the necessary components on the radio. Another major factor in the choice of the TMS320VC5502-300 was that it was available in a thin quad flat pack (TQFP) package, which can be soldered by hand. The majority of the higher end DSPs that might be useful in this capacity are only available in ball grid array (BGA) packages, which have to be soldered on using solder reflow or other processes done in large scale manufacturing [22].

With the TMS320VC5502 DSPs, several configuration options on the DSP are determined at startup based on the status of the GPIO pins. These DSPs are configured to use an external clock by having GPIO4 high at reset. It is also configured to have the external memory interface (EMIF) disabled by having GPIO6 low at reset and the UART is enabled by having GPIO7 low at reset. The boot mode, which determines the source of the code, can be selected (via a switch) to either boot from the host port interface (HPI) or from an EPROM connected via I^2C. The HPI boot mode is required to download and run code from Code Composer Studio (TI's software to program and debug their DSPs) via the JTAG connection [22].

The DSPs have three multi-channel buffered serial ports (McBSP) used to communicate serially with other devices on the radio. On both the transmitter and receiver side, McBSP0 and McBSP1 are configured as SPI interfaces. McBSP0 is connected to the frequency synthesizer on the respective sides of the radio and McBSP1

is connected to a serial DAC that controls the gain on that side's variable gain amplifier. The third and final McBSP, McBSP2, is configured at startup to be a UART connection that is connected to the microcontroller.

The DSPs also have two 16-bit parallel GPIO ports (PGPIO0 and PGPIO1) and a 14-bit parallel GPIO port (PGPIO2). On the receiver side of the radio, the 16-bit PGPIO ports are connected to the FPGA to retrieve the I and Q data when it is ready at the FPGA's outputs. The 14-bit PGPIO is used to program the digital down converters. On the transmitter side, the two 16-bit PGPIO ports are used to output the 14-bit data to the DACs. For this purpose, the lower 14 bits of the 16-bit port are connected to the DACs' inputs.

2.7. MICROCONTROLLER

The microcontroller chosen for the SDR to relay information between the two DSPs and a computer is the C8051F120 produced by Silicon Laboratories™. This device is an 8051 based microcontroller with a maximum operating frequency of 100 MHz. It has two onboard UARTs (which are connected to the two DSPs) and a SPI bus, which is connected to a USB controller. There are also connections from the microcontroller to drive a 20 character by 4 line LCD display for displaying information to the user [23].

2.7.1. USB Controller. The USB controller used for the SDR is the MAX3421E produced by Maxim™. This device performs the low level handshaking in the USB protocol and has four endpoints. The USB controller has the standard bidirectional endpoint 0 (EP0) for control transfers, one OUT endpoint (EP1) for transfers from the SDR to the computer and two IN endpoints (EP2 and EP3) for transfers from the SDR to the computer. Endpoints 1 through 3 can be used for either interrupt or bulk transfers [24, 25]. The USB controller is connected to the microcontroller as a SPI slave.

To interface the computer to the SDR, a LabVIEW GUI was created to control the USB transfers between the computer and SDR. This GUI is shown in Figure 2.6. For interrupt transfers (sent over EP3), the user can choose to display the received data as ASCII characters, hex values or decimal values. For both control and bulk transfers, the communication can be bidirectional. Control transfers are sent over EP0 and bulk transfers are sent over EP1 and EP2. By the user selecting the desired direction of

transfer and the target of the communication (either of the DSPs or the microcontroller), the “Action” dropdown box is populated with the allowable actions for that combination of direction and target. Once these are set, the length of the transfer is set and any data to send is entered, the user can click the “Transfer” button and the program will send the USB packet with the control byte corresponding to the direction, target and action the user specified.

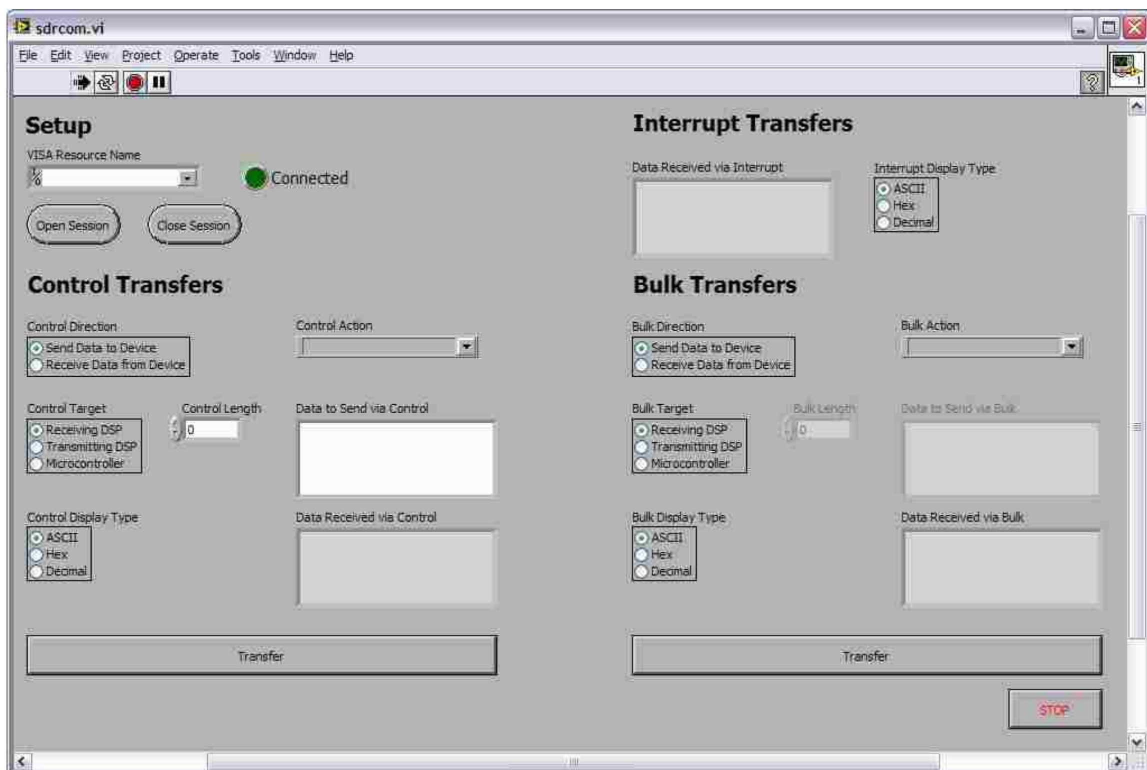


Figure 2.6: LabVIEW GUI for USB Communication

2.7.2. LCD Display. The microcontroller uses its port 2 and part of its port 3 to interface with an LCD display. The LCD display purchased for the SDR was model number LCM-S02004DSF produced by Lumex™, which is a 20 character by 4 line alphanumeric LCD display.

2.8. PBC LAYOUT

The PCB layout was done using EAGLE (Easily Applicable Graphical Layout Editor) made by CadSoft. The PCB was designed to be a 4 layer board with FR-4 acting as the dielectric. The signal traces were designed to have a 50 Ohm characteristic impedance with the trace width being 7.5 mils and the separation between signal and ground/power plane being 5.1 mils. A 50 Ohm characteristic impedance was used to match the input impedance of the RF components used on the SDR. The layer separation was designed for a 4 layer board with 1 oz copper and an overall thickness of 31 mils from Advanced Circuits. The trace width was determined using the trace impedance calculator provided by the University of Missouri – Rolla (UMR) Electromagnetic Compatibility (EMC) Laboratory [26].

Component placement was done to minimize the trace length of the high speed traces and maximize the number of components on the top layer for better signal integrity and ease of populating the board. Also, an effort was made to keep high speed signal lines from transferring between layers in order to better maintain signal integrity. Images of the PCB layout can be found in Appendix B.

3. SOFTWARE DEFINED RADIO TESTING

3.1. ANTENNA

Because the antennas purchased for this radio did not have a data sheet available, the S11 curve of the antenna was taken to get an idea of the reflection efficiency of the antenna. The S11 curve is an indicator of the input impedance of the antenna as a function of frequency, and so can be used to get an idea of the reflection losses from mismatched impedances between the antenna and the 50 Ohm trace on the PCB. The S11 curve of the BB85 antenna is shown in Figure 3.1. This is the worst case scenario for this antenna because this measurement was taken without a ground plane, which degrades the performance of a whip antenna. In this measurement, the input impedance varies from approximately 4 Ohms to 450 Ohms, both of which correspond to a refection efficiency of approximately 25%. A graph of the reflection efficiency of the BB85 antenna can be seen in Figure 3.2.

The antenna was also tested in the Lindgren 12' x 16' semi-anechoic chamber in the University of Missouri – Rolla Electromagnetic Compatibility Lab. A 1.6 GHz signal at approximately 32 dBm was transmitted through a Sunsol Sciences Corporation JB series broadband EMC test antenna to a BB85 antenna connected to a spectrum analyzer approximately 1.2 meters away. The received signal was -55 dBm in amplitude . Using this, the gain of the antenna at 1.6 GHz was determined using the Friis transmission equation (3) [27,28].

$$\frac{P_r}{P_t} = \left(1 - |\Gamma|^2\right)\left(\frac{\lambda}{4\pi R}\right)^2 G_{0t} G_{0r} \tag{3}$$

For this equation, P_r is the power received by the antenna (-55 dBm) converted to Watts, P_t is the power at the input of the transmitter antenna (32 dBm) converted to Watts, Γ is the reflection coefficient (S11 measurement) at 1.6 GHz, λ is the wavelength at 1.6 GHz, R is the distance between the transmitter and receiver antenna (approximately 1.2 meters), and G_{ot} and G_{or} are the gain of the transmitter and receiver antenna respectively. For the purposes of this calculation, conduction efficiency and dielectric

efficiency were assumed to be 1, the refection coefficient of the transmitter was assumed to be 0, and the antennas were assumed to be polarization matched and aligned for maximal directional radiation and reception.

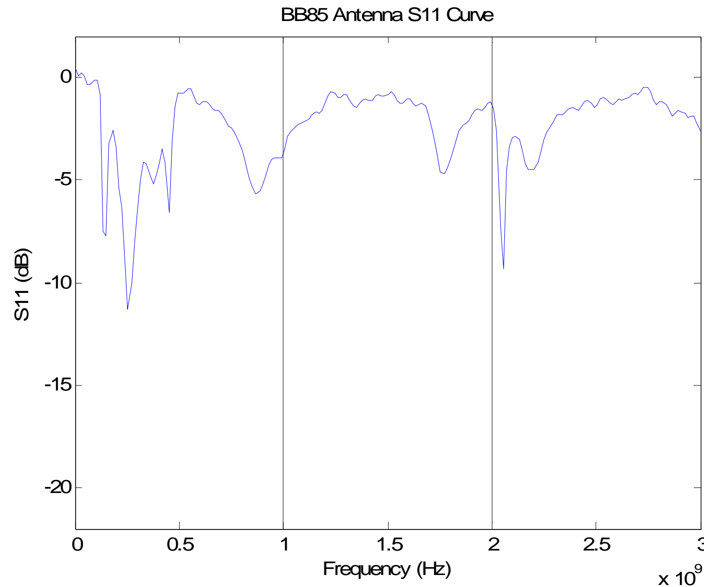


Figure 3.1: BB85 Antenna S11 Curve

To determine the gain of the transmitting antenna, equation (4) was used. In this equation, *AF* is the antenna factor, λ is the wavelength in meters, and *G* is the gain of the antenna. This equation assumes the transmission medium is free space with an impedance of 377 Ohms and the antenna has an input impedance of 50 Ohms, which are both true for the transmitting antenna. Performing this calculation, the gain of the antenna was determined to be approximately -49 dB at 1.6 GHz. By taking into account the reflection losses due to the mismatched impedance between the antenna and the traces on the PCB, the absolute gain of the antenna was determined to be approximately -55.7 dB at 1.6 GHz. Again, this is the worst case scenario for this antenna because the measurements were taken in the absence of a ground plane [28,29].

$$AF = \frac{9.73}{\lambda\sqrt{G}} \tag{4}$$

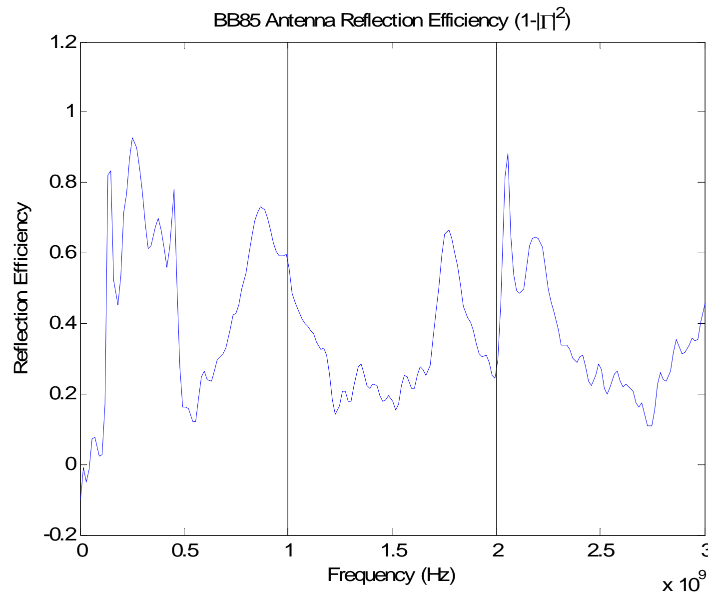


Figure 3.2: BB85 Antenna Reflection Efficiency

3.2. VOLTAGE CONTROLLED OSCILLATOR

The VCO circuit was tested on the receiver side of the SDR to test the accuracy and stability of the VCOs. The VCO was set at 1.600000 GHz in low noise mode and the output was measured with a spectrum analyzer. The results are shown in Figure 3.3. The center frequency of the VCO output remains stable within 1 kHz around its desired value. In this test, the VCO drifted a few hundred hertz, but it remained within 1 kHz of 1.600000 GHz set frequency. However, in low noise mode (where the greatest spurs are allowed), the spurs on the VCO output covered approximately 300 kHz on either side of the center frequency. VCO measurements at several other frequencies are shown in Appendix C.

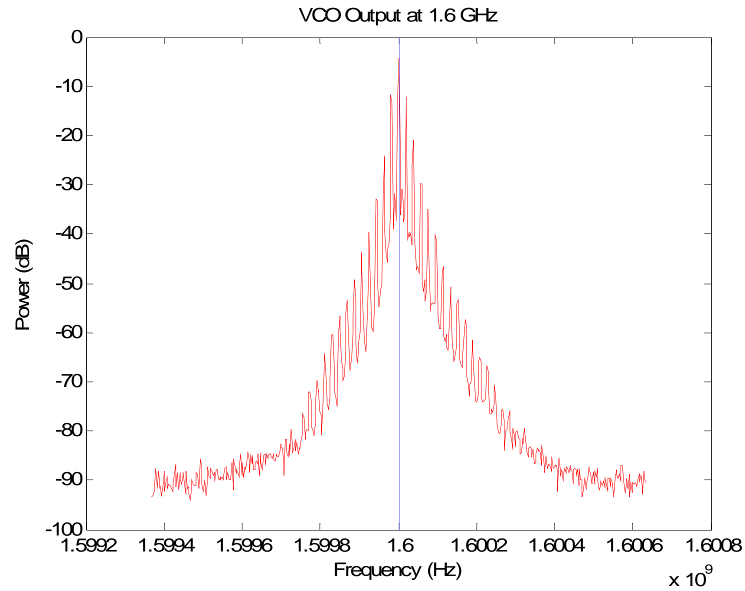


Figure 3.3: VCO Output at 1.6 GHz

3.3. RECEIVER

In order to test the receiver side of the SDR, the receiver input was connected directly to the output of a high frequency signal generator. This was done instead of transmitting a signal because the input power to the system could be controlled. The first test performed was to input a 20 MHz signal into the receiver and measure the outputs of the low noise amplifier and the variable gain amplifiers. For this test, the VGA was set to its maximum gain (20 dB). The output of the LNA is shown in Figure 3.4 and the output of the VGA is shown in Figure 3.5. As seen the LNA successfully amplified the signal by a factor of approximately 10, as expected. However, the VGA output is only slightly larger than the output of the LNA, though it was expected to again increase the signal by a factor of 10 as it was set to have approximately 20 dB of gain.

Once the LNA and VGA were verified to work, the signal generator was configured to output a 1.60025 GHz sine wave at -20 dBm and the VCO was configured to output 1.600000 GHz. The output of the I/Q demodulator was measured on an oscilloscope, but no output appeared above noise level. By varying the output frequency of the signal generator, it was determined that the spurs present in the VCO output tended

to overcome any signal within a few hundred kHz of the center frequency. This is beneficial in the sense that is removes the carrier from the signal, but it must be taken into account when determining the signal being transmitted.

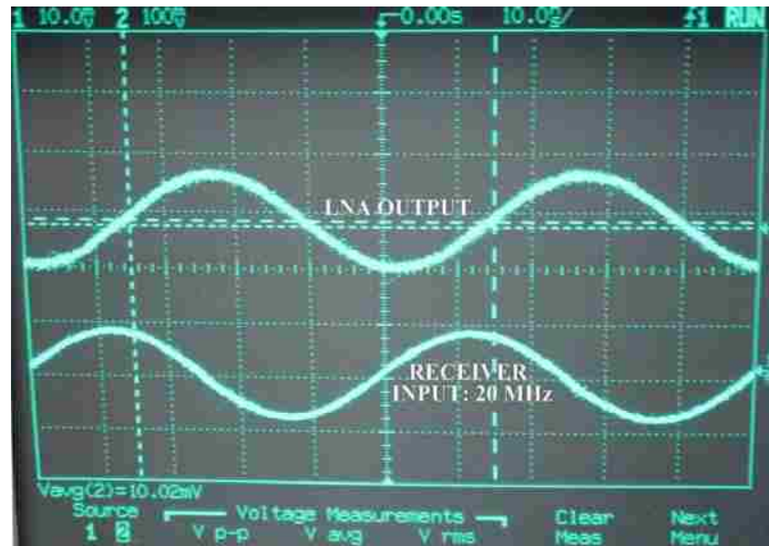


Figure 3.4: LNA Output 20MHz Input

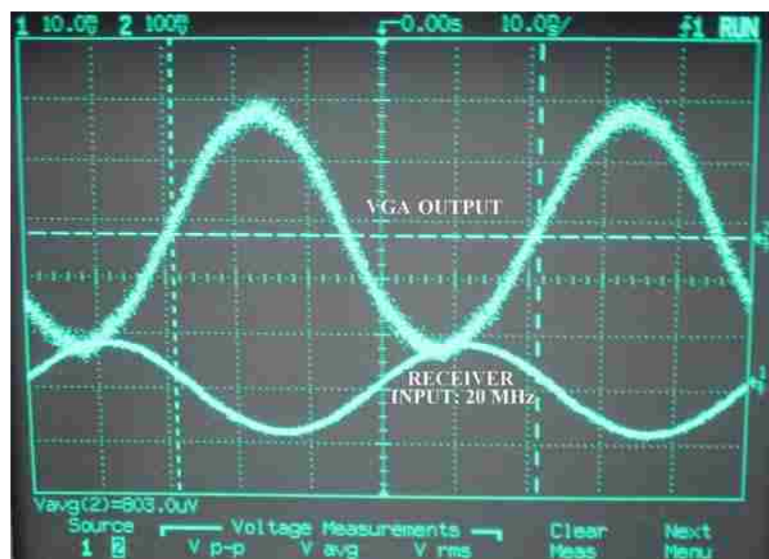


Figure 3.5: VGA Output

Because of this, the signal generator was adjusted to output 1.60525 GHz sine wave at -20 dBm. The receiver DSP was then programmed to set the VCO output to 1.600000 GHz and the DDCs were configured to further down-convert the signal by 5 MHz and then decimate the samples to a 1 MHz sampling frequency (code can be found in Appendix D). Since the samples are complex, this yielded a usable bandwidth of 1 MHz. Once the VCO and DDCs were configured, the DSP was set to collect 2048 samples from the FPGA (which is responsible for adding the complex outputs of the two DDCs). The FFT of the resulting data is shown in Figure 3.6, where the frequency scale has been adjusted to accommodate for the 1.605 GHz down-conversion. This figure also shows the effects of the spurs present on the VCO output.

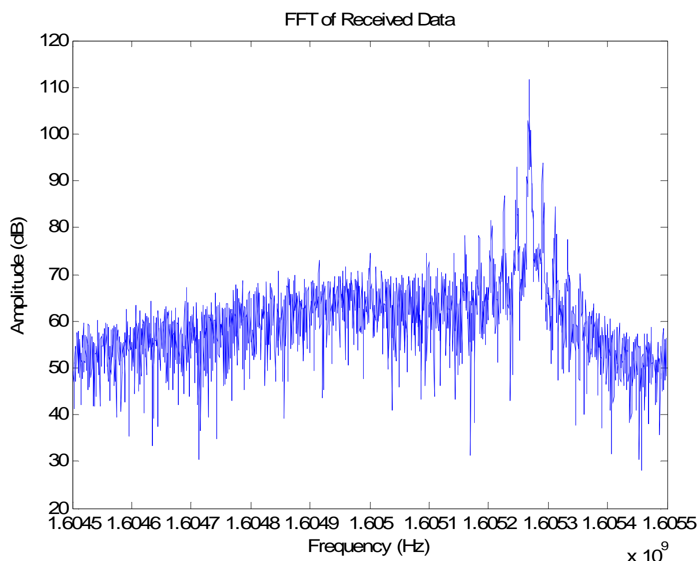


Figure 3.6: Data Received by Receiver DSP

3.4. TRANSMITTER

In order to test the transmitter side of the SDR, the transmitter DSP was programmed with the code in Appendix E. The DSP was configured to operate at its maximum frequency, 300 MHz. The GPIO pins were set as inputs (as they are not used),

the PGPIO ports were set as outputs to drive the DACs, the HPI pins were configured to be used as GPIO, McBSP0 and McGSP1 were set to be used as SPI, the UART was configured and the necessary interrupt masks were set. Timer 0 was also configured to be used as the clock for the DACs by setting the timer 0 output pin (TIM0) to change its value every time timer 0 overflows.

Testing of the transmitter was done in the semi-anechoic chamber in the UMR EMC Lab. Images of the test setup can be seen in Appendix F. The first test run was to test the output of the transmitter when no signals were present at the DAC outputs. For this, the carrier frequency was set to 1.600000 GHz and the gain on the VGA was set to approximately 0 dB. The SDR was placed in the chamber and the transmitted signal was measured with a spectrum analyzer. The results are shown in Figure 3.7. This shows that the accuracy of the frequency synthesizer and VCO is very good. The precision of the local oscillator circuit is 1 kHz and the carrier stays within 1 kHz of its set frequency, but it will vary within that 1 kHz tolerance.

To test the up-converting capabilities of the transmitter, two different tests were run. For both tests, the DAC frequency was set to 600 kHz and the carrier frequency was set to 1.600000 GHz. First, the DSP outputted a 512 sample sine wave to the I component DAC and outputted nothing to the Q component DAC. This yielded a sine wave at approximately 1.17 kHz entering the I/Q modulator on the real side. Again, the transmitted signal from the SDR was measured on a spectrum analyzer. The results, shown in Figure 3.8, show that the transmitter successfully up-converted the signal and transmitted it. However, the transmitter circuitry also transmitted the second harmonic of the message signal. This could be caused by a design error in the DAC section. The DAC was designed to output 20 mA causing a 4 V_{p-p} input to the I/Q modulator. However, the I/Q modulator is rated for a 2 V_{p-p} input. This can be fixed by changing the full-scale current to 10 mA by replacing the 2 kOhm full-scale adjust resistors with 3.8 kOhm resistors. To better determine the cause of these harmonics, buffered test points should be added to the output of the DACs.

The next test performed was to transmit $e^{\{j2\pi(1.17\ \text{kHz})t\}}$ to ensure the I/Q functionality of the transmitter worked correctly. To do this, two 512 sample arrays were used. Each array held a single cycle of a sine wave, with the array corresponding the Q

component being 90 degrees behind the array corresponding to the I component. This transmission was also measured in the semi-anechoic chamber and the results are in Figure 3.9. This shows that the transmission was successful because there is a spike approximately 1.17 kHz above the carrier, but no spike 1.17 kHz below the carrier. However, the second harmonic still appeared both above and below the carrier frequency.

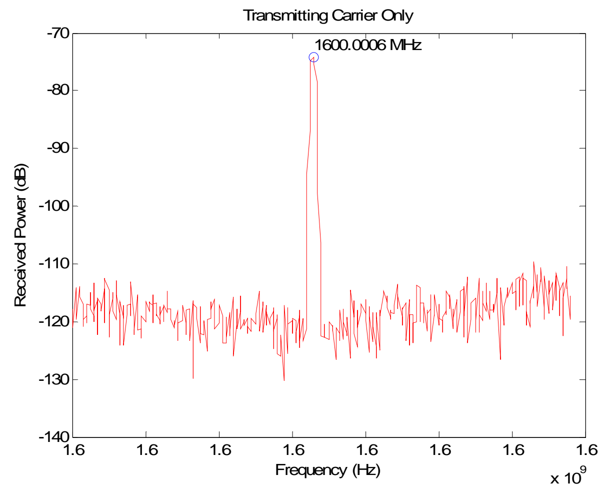


Figure 3.7: Transmitting only Carrier Frequency

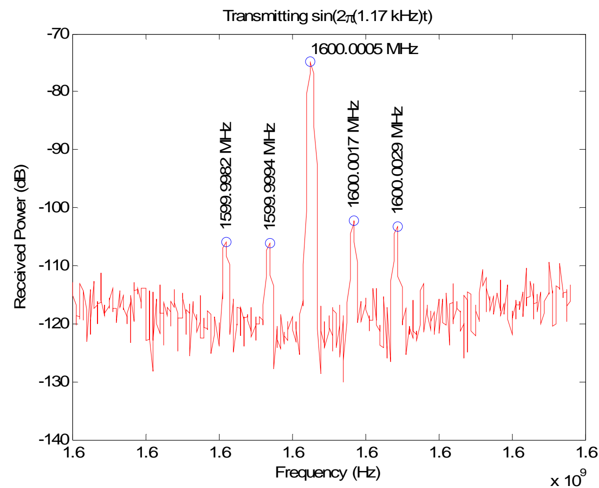


Figure 3.8: Transmitter DSP Transmitting 1.17 kHz Sine Wave with 1.6 GHz Carrier

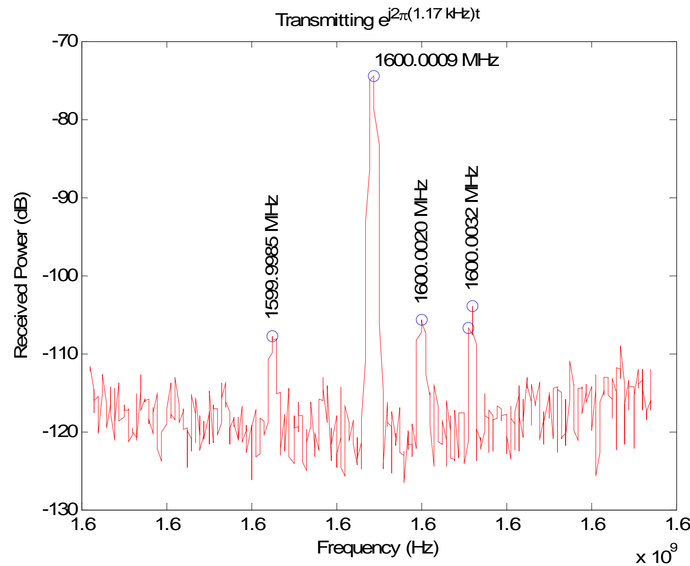


Figure 3.9: Transmitter DSP Transmitting $e^{\{j2\pi(1.17\ kHz)t\}}$ with a 1.6 GHz Carrier

These tests verified the operation of the transmitter side of the SDR. The only problem with the transmitter is the presence of the second harmonic of the transmitted signal.

3.5. MICROCONTROLLER

There were three main components to test with the microcontroller section of the SDR, the LCD display, UART communication with the DSPs and the USB functionality. The software used to test the microcontroller is in Appendix G. To test the LCD display, code was written to communicate with the display as outlined in the LCD General Information data sheet provided by Lumex™, the manufacturer of the display used for the SDR [30]. The microcontroller was programmed to display “Hello World” on the first line of the display once it was initialized and it either display “USB Success” or “USB Failure” based on if the USB controller successfully initializes or not. The test worked correctly, as seen by Figure 3.10.

The second portion of the microcontroller to test was the UART communication with the DSPs. To do this, a text message was sent from the receiver DSP to the

microcontroller once it initialized and the microcontroller displayed this message, “DSPR Initialized”, on the third line of the display. This test was also successful.

Figure 3.10: Successful LCD Display Test

Lastly, the USB portion of the microcontroller section of the board was tested. Unfortunately, this portion of the SDR never worked successfully and the problem was never located. Software has been written to handle the USB enumeration and other data transfers, but the USB controller fails to trigger an interrupt whenever it receives data from the computer. A USB monitoring program called USBTrace by SysNucleus was used to monitor any USB transfers between the computer and SDR, and the USB controller never responded to enumeration requests by the computer. A possible way to resolve this problem would be to switch to another microcontroller that has built in USB functionality, such as the C8051F343 made by Silicon Laboratories™. By using a microcontroller produced by Silicon Laboratories™, the code written for the C8051F120 can be easily ported over for use on a microcontroller with USB functionality.

4. ANALYSIS

4.1. LINK BUDGET ANALYSIS

In order to get a better idea of the performance of the SDR, a link budget analysis was performed. A link budget analysis estimates the signal to noise ratio (SNR) throughout the communication link to calculate the theoretical bit error rate (BER) of the system for a given modulation technique. For this link budget, calculations were done with a carrier frequency of 1.6 GHz because the gain of the antennas was known at that frequency. Also, a 5 m range was used and the data rate was set to 14.4 kilobits per second (kbps) with a bandwidth of 28.8 kHz. Using these parameters, the BER was calculated for PSK modulation.

The first step in that analog portion of the communications link is the I/Q modulator. The I/Q modulator outputs a signal with a power level of -10 dBm with a noise floor of -147 dBm/Hz. With the 28.8 kHz bandwidth, this gives a noise power of approximately -102.4 dBm. After the I/Q modulator, the signal is passed through a variable gain amplifier with a maximum gain of 20 dB and a noise figure of 10 at 20 dB gain. With the 20 dB gain, the output signal power was calculated to be 10 dBm. The noise figure of a device is the ratio of the input SNR to the output SNR as shown in equation (5) [31]. By using this, equation (6) was derived to calculate the noise power using only the noise figure, input noise power and gain. Using these, the output noise power was calculated to be at -72.4 dBm.

$$NF(dB) = 10\log\left(\frac{SNR_{IN}}{SNR_{OUT}}\right) \tag{5}$$

$$N_{OUT}(dBm) = N_{IN}(dBm) + NF(dB) + G(dB) \tag{6}$$

After the VGA, the signal goes out via the antenna, which has an absolute gain of approximately -55.7 dB at 1.6 GHz. This reduces both the signal power and the noise power by -55.7 dB to -45.7 dBm and -128 dBm respectively. Once in the channel, the signal is attenuated by -50.5 dB over a distance of 5 m (using the Friis Transmission Equation (3) without the antenna gains), giving a received signal power of approximately

-96.2 dBm. The noise power is also attenuated by -50.5 dB, but background noise is added with a power of approximately -174 dBm/Hz. These two figures are converted to watts and added, then converted back to a noise power level of -129.4 dBm. This signal and noise power is then received by the antenna with a gain of -55.7 dB, giving a received signal power of -151.9 dBm and a noise power of -185 dBm.

On the receiver side of the SDR, the signal first goes through a LNA with a fixed gain of 20 dB and a noise factor of 5. This yields an output signal power of -131.9 dBm and a noise power of -160 dBm. Then the signal is passed through the same type of VGA used on the transmitter side of the SDR, giving a signal power of -112 dBm and a noise power of -130 dBm. Finally, the signal goes through the I/Q demodulator, which has a maximum gain of 39.5 dB with a noise figure of 11 dB. This gives a final output signal power of -72.4 dBm and a noise power of -79.6 dBm, giving a SNR of 7.2 dB. Using equation (7), this SNR can be converted to an E_b/N_0 ratio of 10.5. Then, equation (8) can be used to determine the theoretical bit error rate for an E_b/N_0 of 10.5. With this configuration, the BER of this system is theoretically $2.3 * 10^{-6}$. However, the received signal power is 70 dB lower than the desired signal power input for the ADC, which would be approximately -2 dBm. Therefore, with this hardware configuration, the SDR would not be able to transmit over 5 m. The antennas would need to be replaced with antennas with a gain of approximately -20 dB or better.

To solve this problem, another link budget analysis was done with modified components. In this link budget, a LNA with a fixed gain of 20 dB and a noise figure of 2 dB was placed between the I/Q modulator and the VGA on the transmitter side, giving a maximum possible transmission power of 30 dBm, or 1 W. Also, 5 of the 20 dB gain LNAs with a noise figure of 2 were used to replace the LNA and VGA on the receiver side. This increased the possible gain in the system and allowed the I/Q demodulator to have a gain less than its maximum. With this configuration, the data rate was increased to 1 megabit per second (Mbps) and the resulting SNR was calculated to be 13.6 dB, yielding a theoretical BER of $2.3 * 10^{-22}$ for PSK modulation, which is a drastic increase in performance. Also, the output signal power of the I/Q demodulator was increased to the acceptable level of -2.9 dBm with the I/Q demodulator at a gain of only 29 dB.

4.2. DYNAMIC RANGE

The dynamic range of the receiver is defined as the ratio of the maximum power level detectable without clipping to the minimum detectable power above the noise level of the system. The ADCs used in this system claim a 100 dB dynamic range, but this is limited by the noise introduced in the system by the VGA. The VGA has a noise floor of -150 dBm/Hz. With the 25 MHz bandwidth that ADCs sample at, this gives a total noise power of -76 dBm. This noise is then passed through the I/Q demodulator, where, assuming maximum gain, it is amplified by 39.5 dB with a noise figure of 11 dB, yielding a total noise power at the ADCs of -25.5 dBm. The maximum input power to the ADCs is approximately -1.7 dBm, giving a dynamic range of 23.8 dB. This can be improved by removing the VGA from the system, as it is a source of a lot of noise. The LNA and VGA on the receiver side should be replaced with 5 LNAs with a fixed 20 dB gain an a noise figure of 2 dB or less.

4.3. SUGGESTED MODIFICATIONS

Due to the testing and analysis of this SDR, several modifications are suggested to improve the performance of the system. These modifications are:

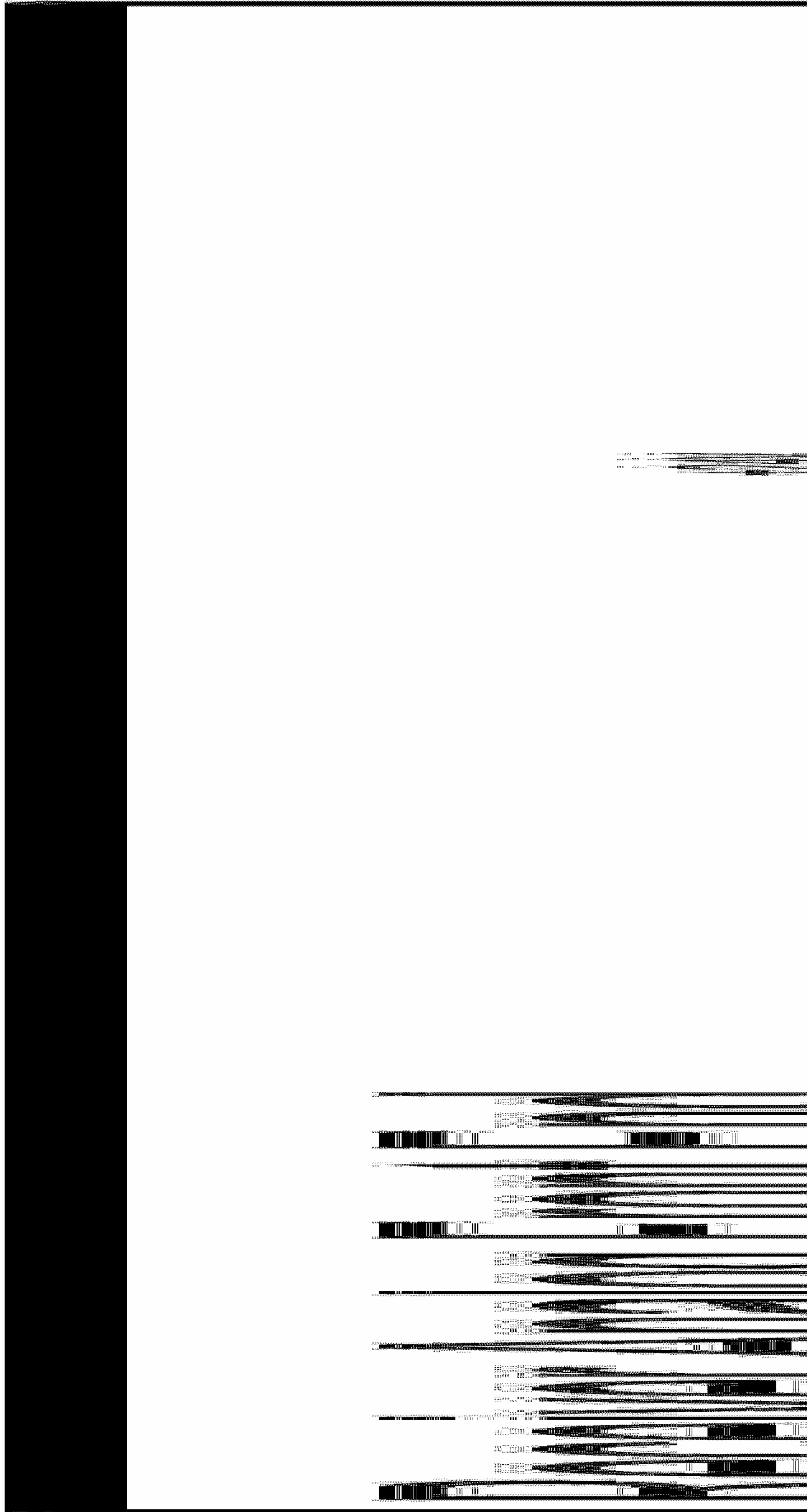
- Place buffered test points at DAC outputs and ADC inputs.
- Use vertical BNC connectors instead of 90º BNC connectors.
- Replace microcontroller and USB controller with a microcontroller that has built in USB capabilities.
- Change the resistor connected to the FSADJ pin of the DACs from 2 kOhm to 3.8 kOhm to adjust the full scale current output from 20 mA to 10 mA.
- Add a low noise amplifier with fixed 20 dB gain and a noise figure of 2 dB or less between the I/Q modulator and transmitter side variable gain amplifier.
- Replace the LNA and VGA on the receiver side with 5 LNAs with fixed 20 dB gain and a noise figure of 2 dB or less.
- Change the VCOs to either the 900 MHz ISM band or the 2.4 GHz ISM band to conform to FCC regulations.

5. CONCLUSION

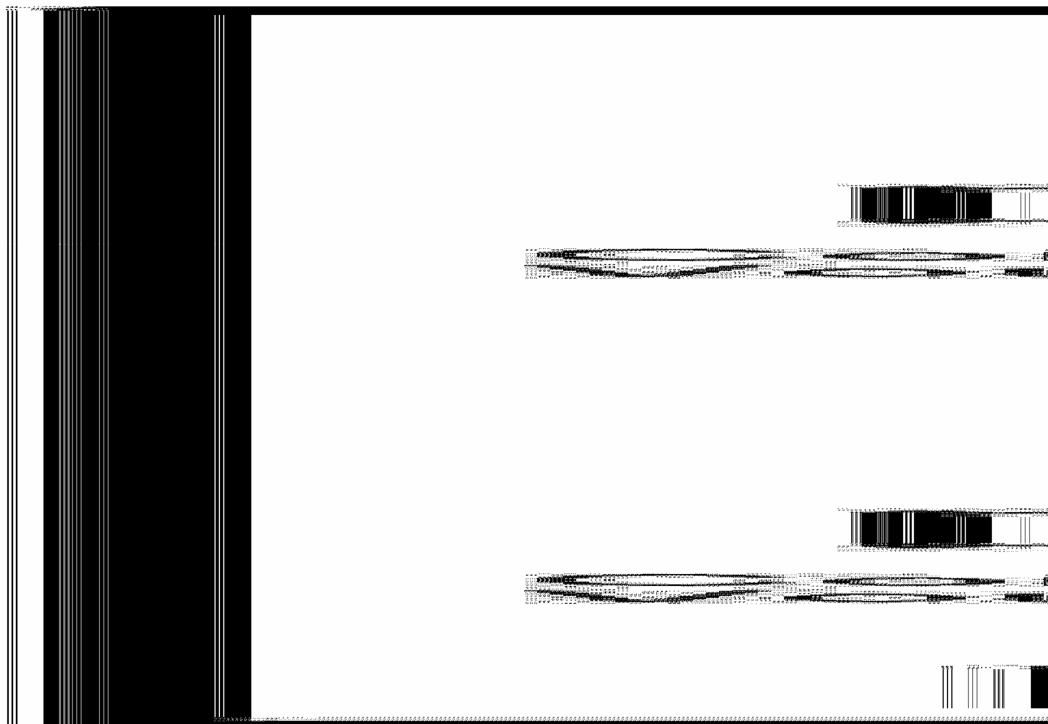
The software defined radio developed at the IFT/UMR Telemetry Learning Center is an ideal platform for future research into cognitive radio. This device can operate over a broad range of carrier frequencies that can be changed through software in increments of only 1 kHz. It can also change the modulation scheme used by performing necessary calculations at baseband using I/Q signals. Finally, the output power level can be changed via the variable gain amplifier on the transmit side of the SDR. These three features are the primary requirements for a cognitive radio.

Cognitive radio is a branch of research aimed at more fully utilizing the RF spectrum by allowing wireless communication systems to dynamically allocate unused portions of the RF spectrum to establish communication. To do this, the radio must sweep the spectrum to locate the unused bands, called white spaces, and determine the optimal band for communication. The cognitive radio can then determine the optimal modulation scheme and power level to use on that band without causing interference with any licensed users.

APPENDIX A.
SOFTWARE DEFINED RADIO SCHEMATICS

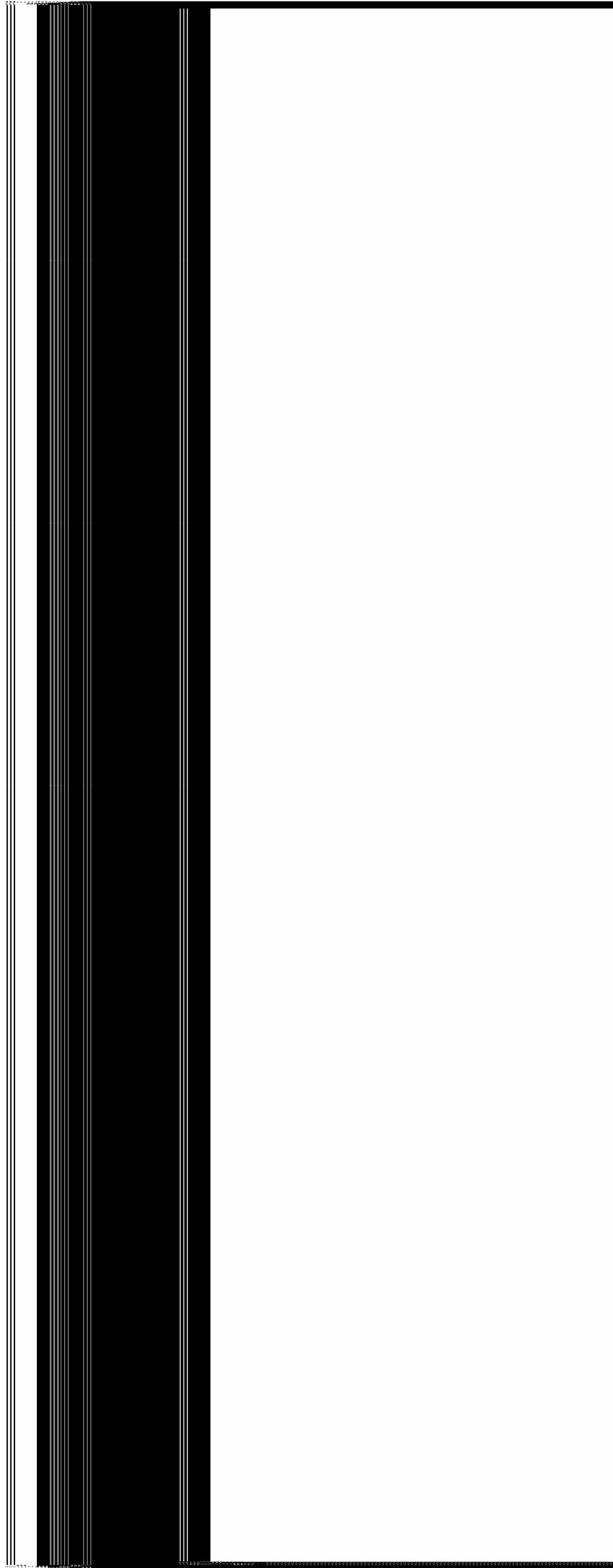


Receiver RF Front End

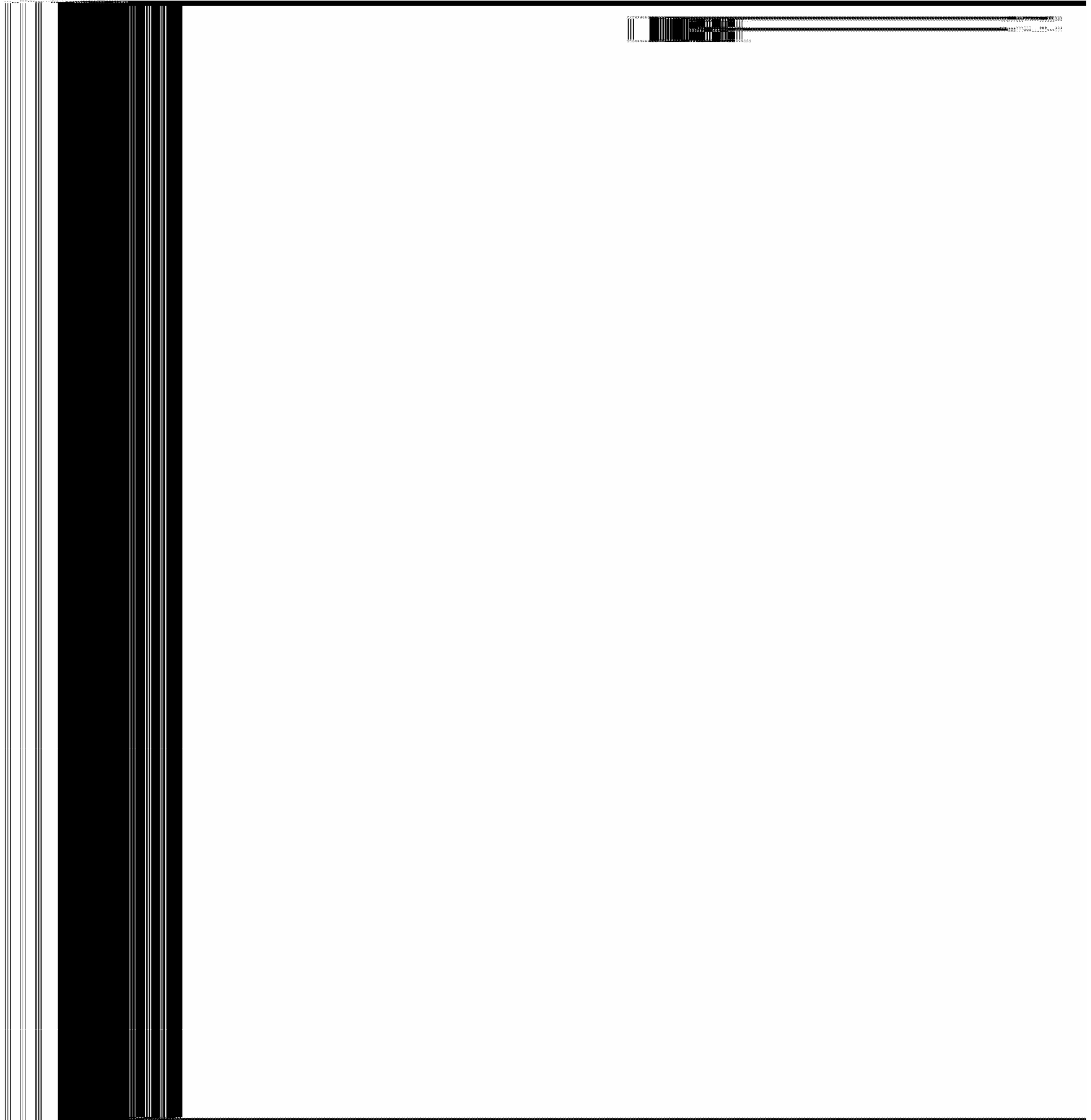


Receiver Anti-aliasing Filters

Analog to Digital Converters

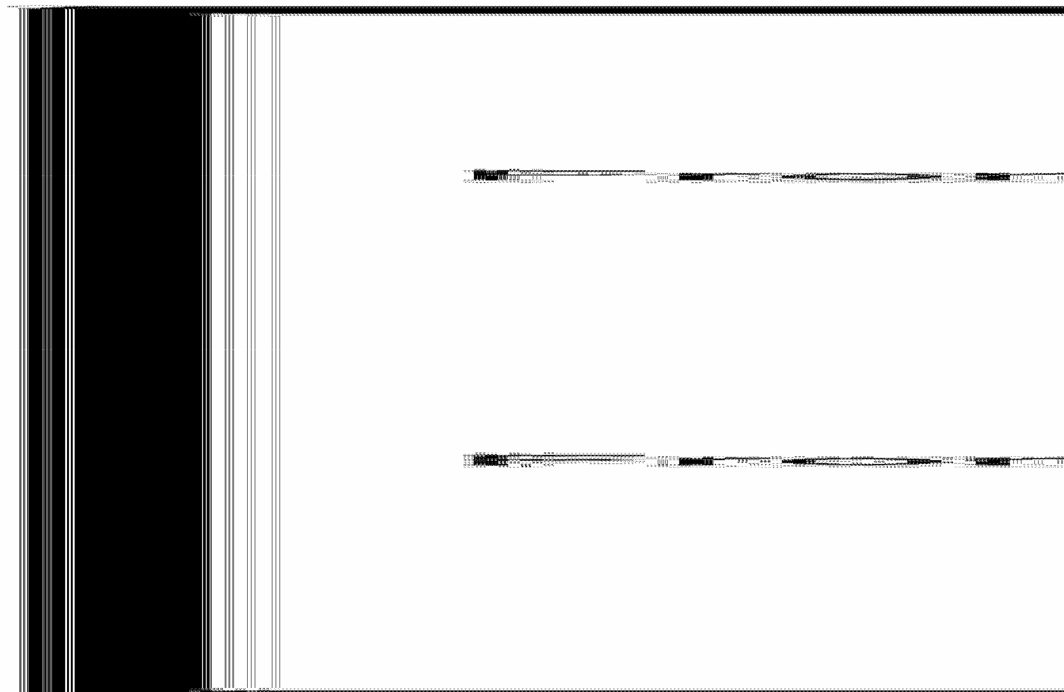


Digital Down Converters

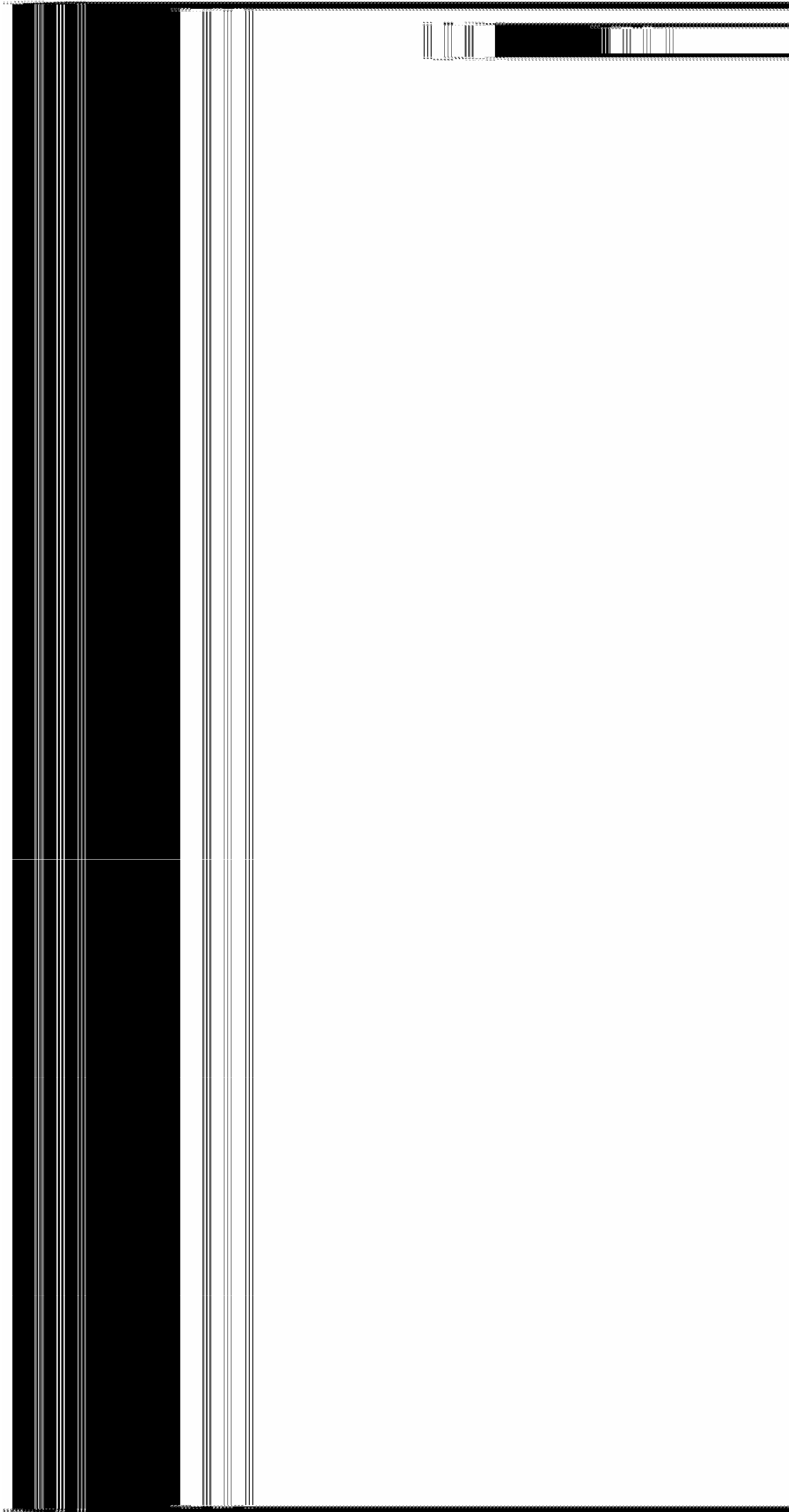


FPGA Design

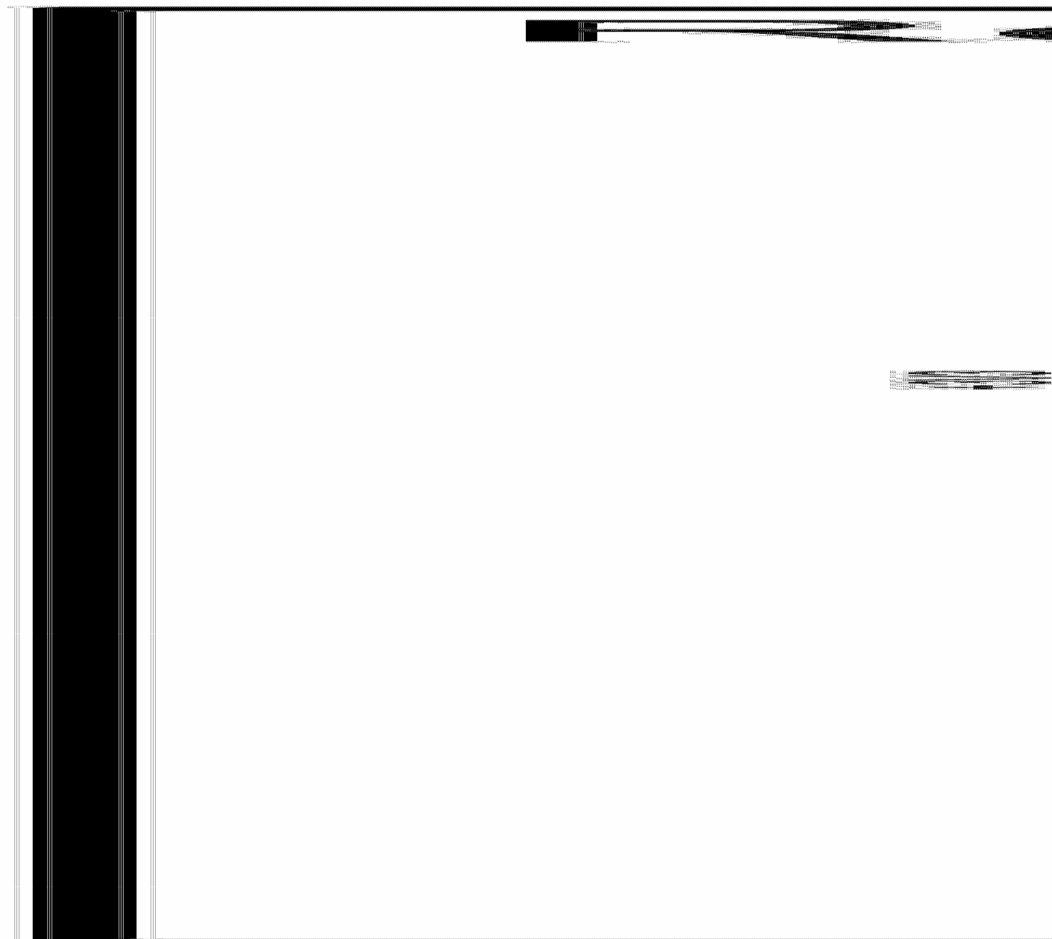
Transmitter RF Front End



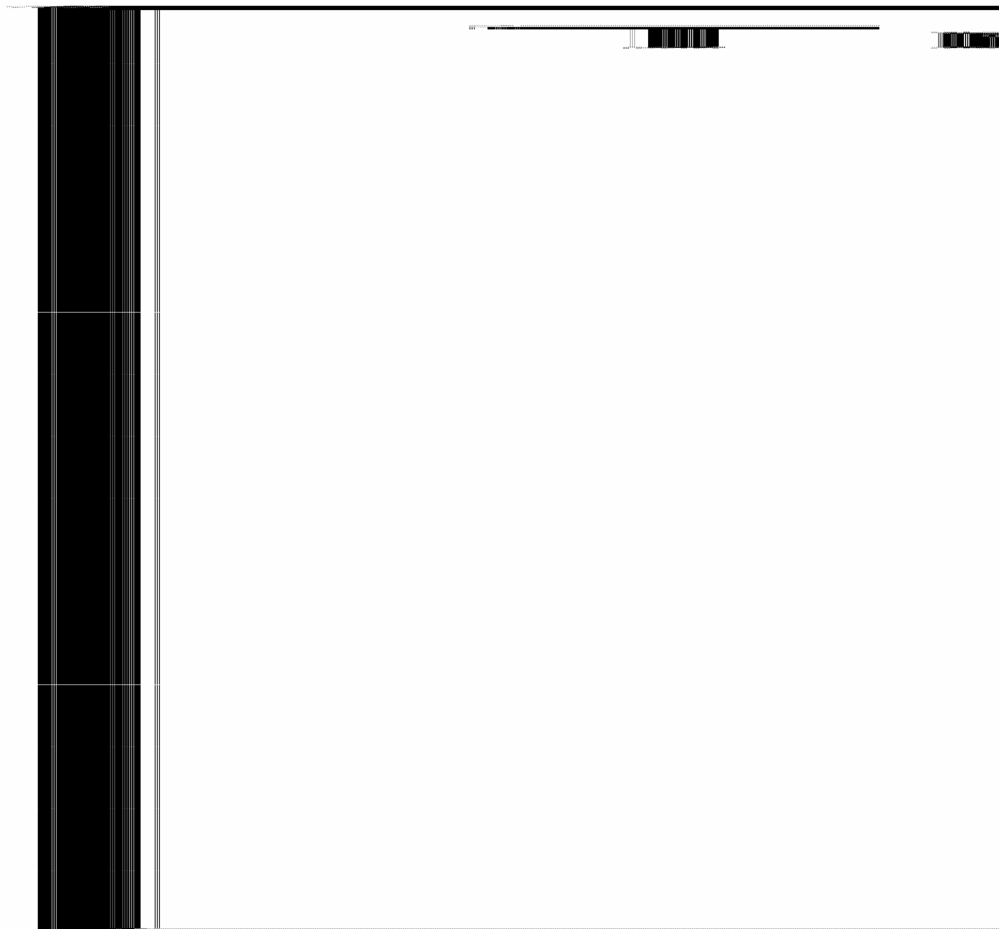
Transmitter Reconstruction Filters



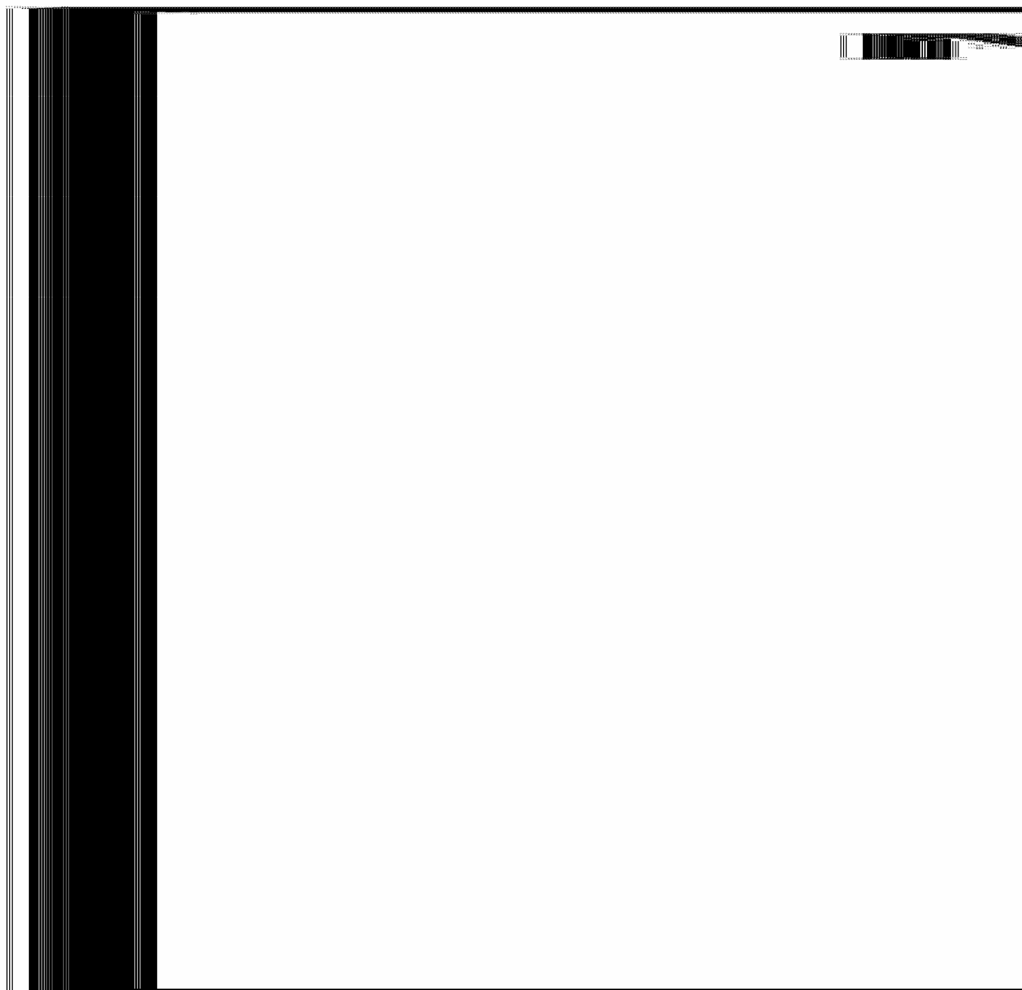
Digital to Analog Converters



Receiver DSP



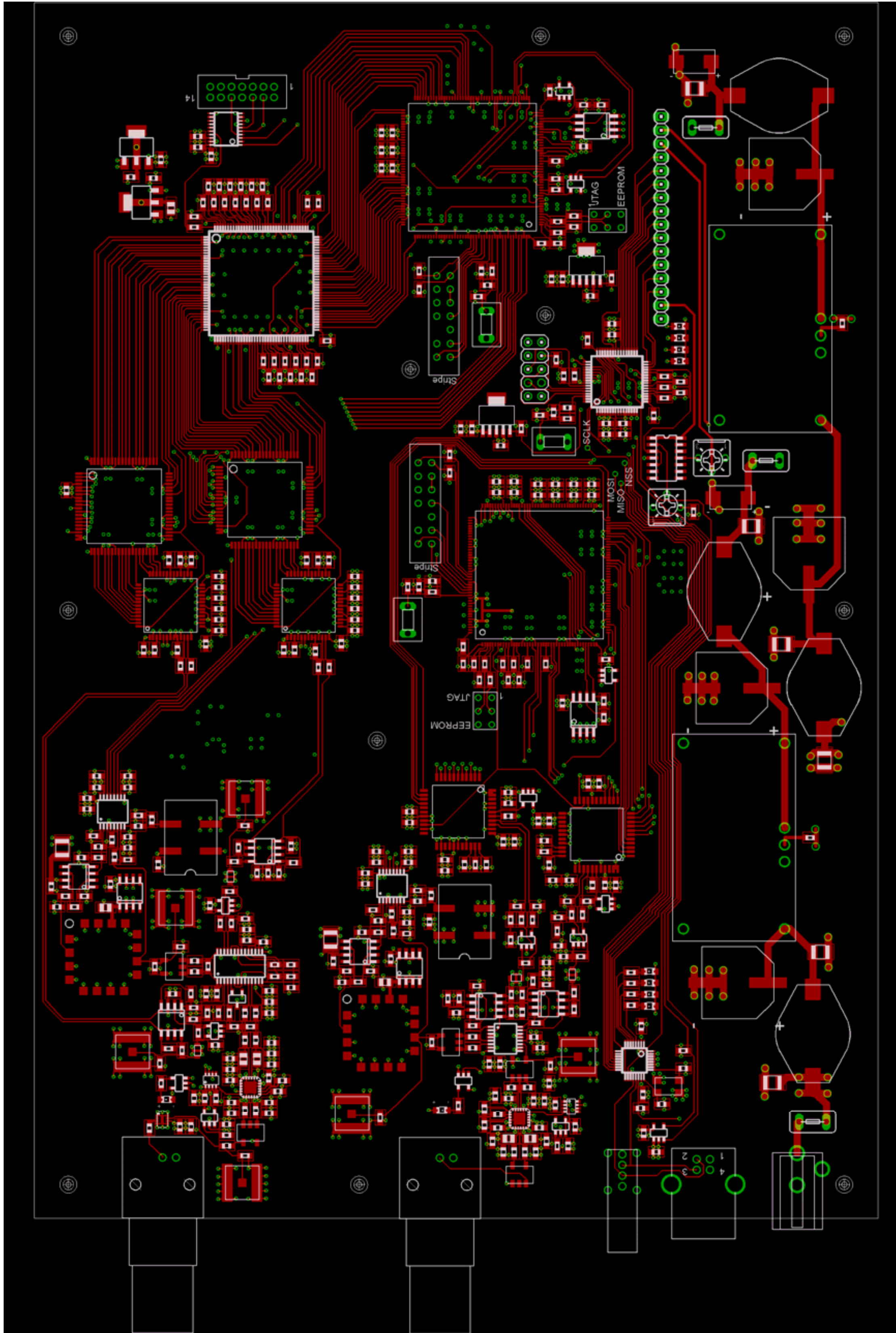
Transmitter DSP



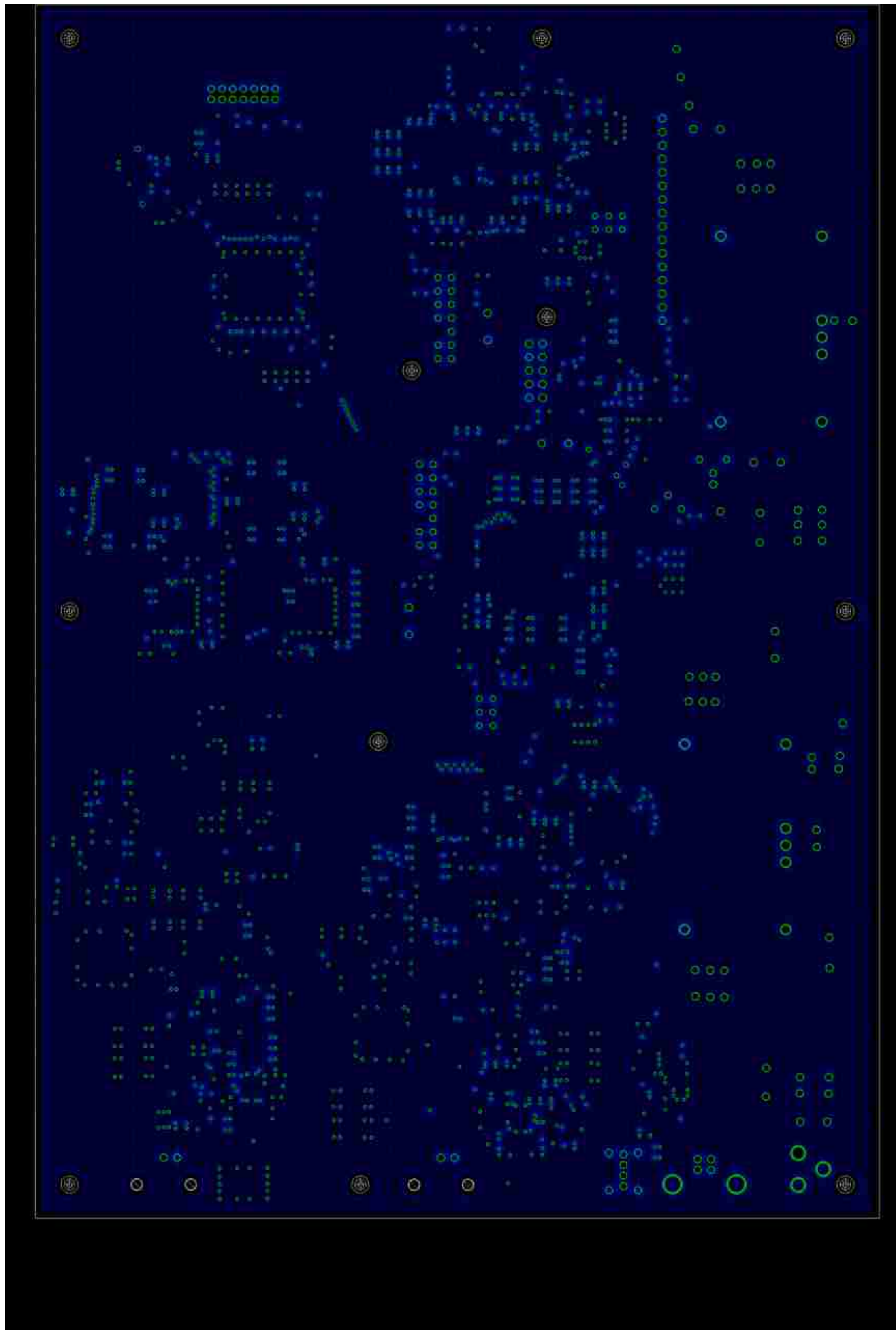
Microcontroller, USB Controller and LCD Display

Power Supplies

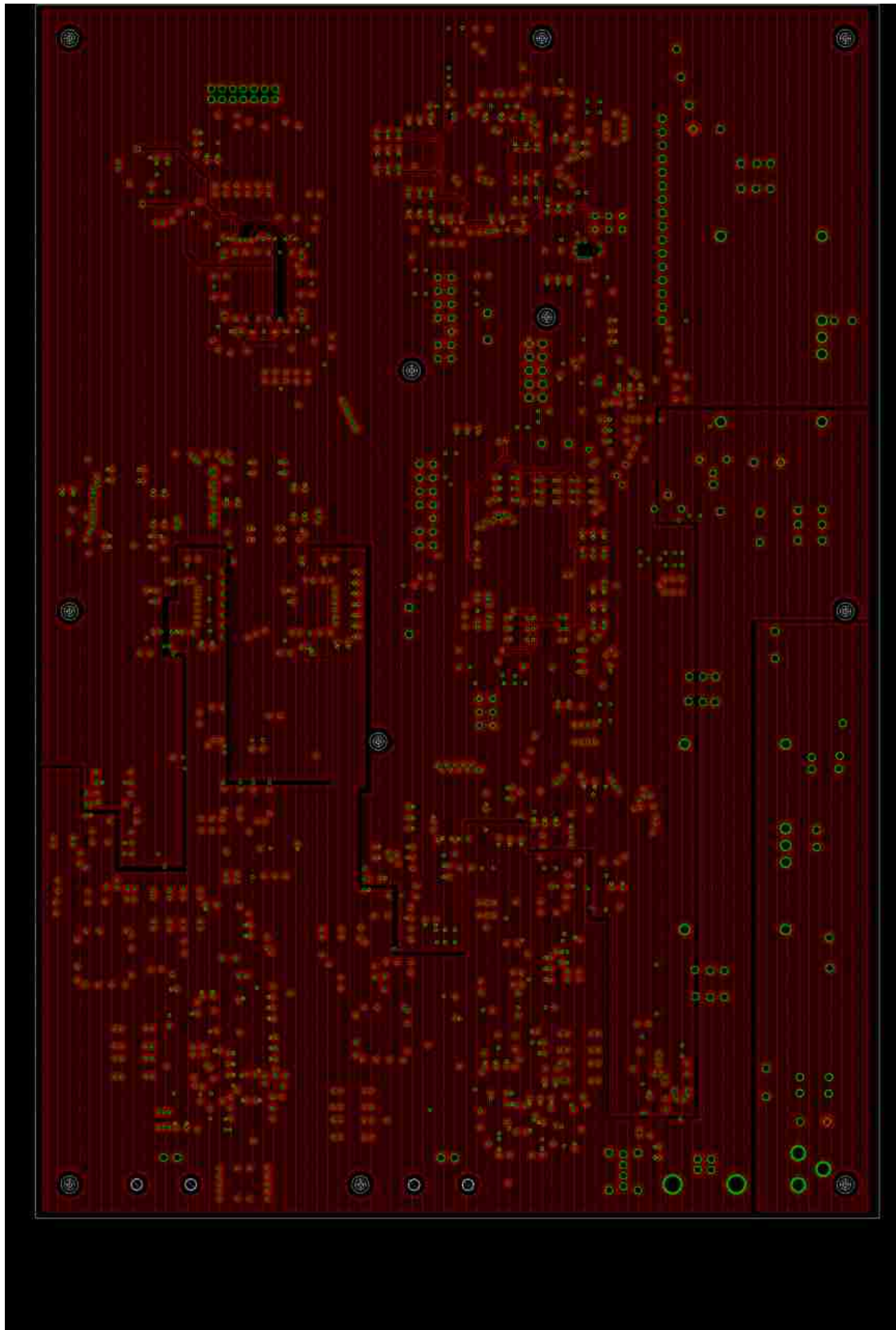
APPENDIX B.
SOFTWARE DEFINED RADIO PCB LAYOUT



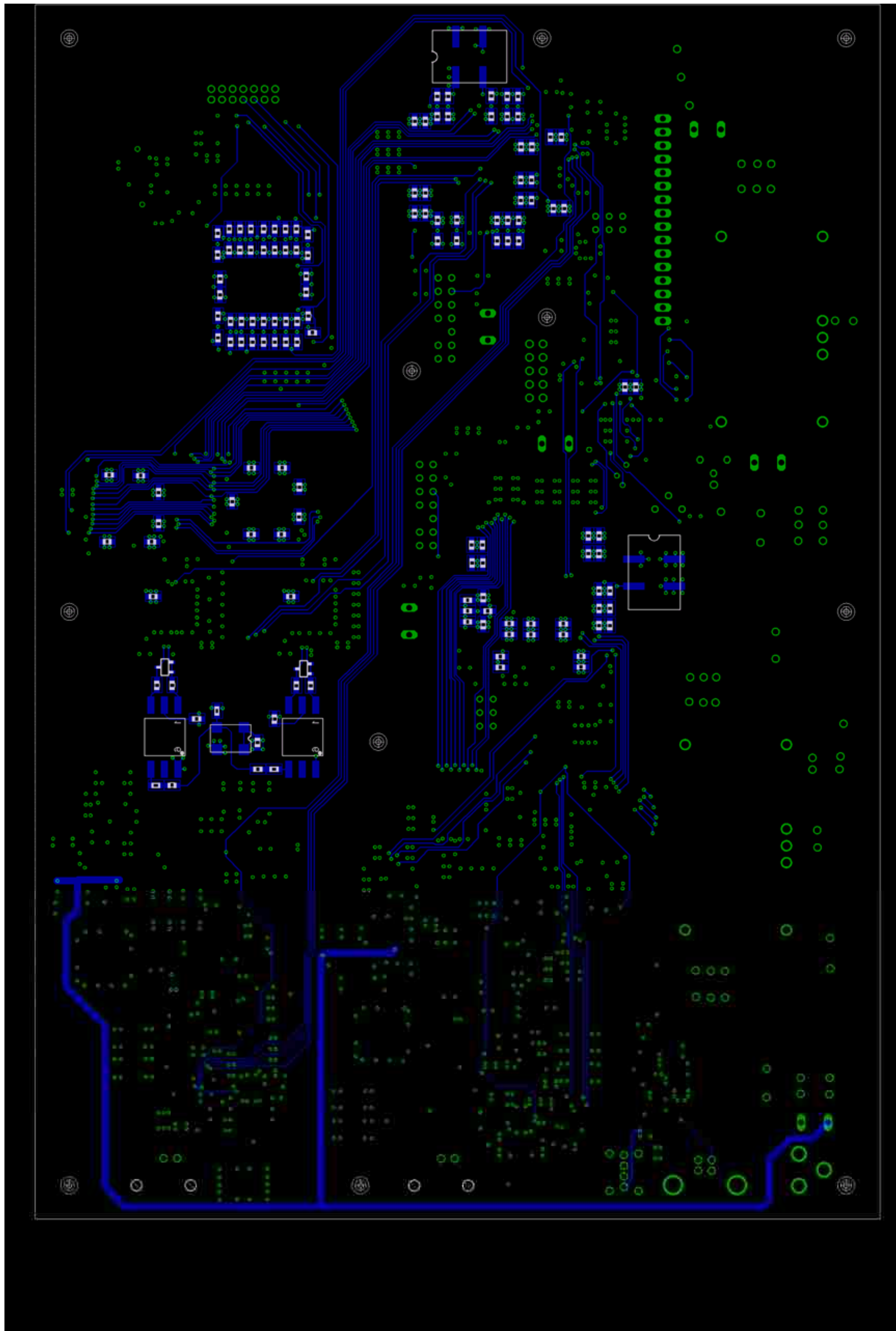
PCB Top Signal Layer (Layer 1)



PCB Ground Layer (Layer 2)

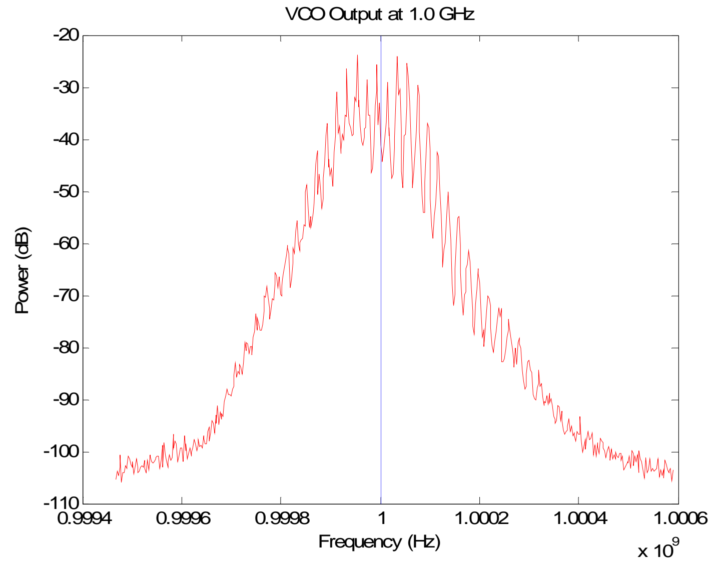


PCB Power Layer (Layer 3)

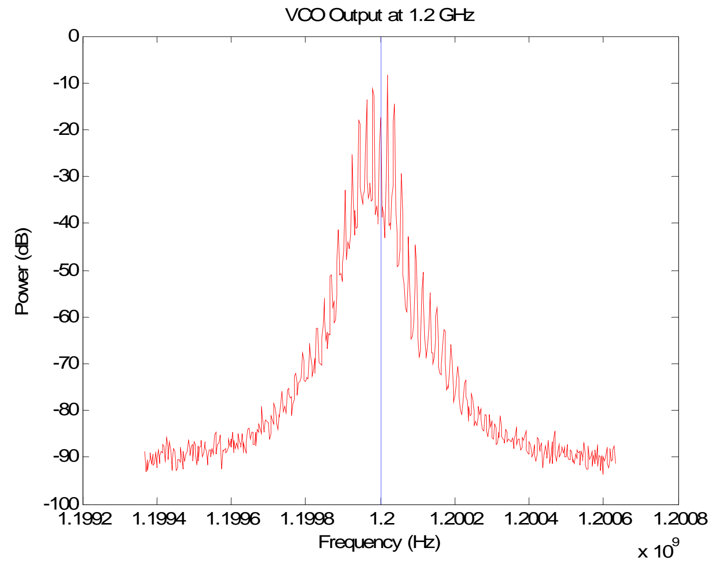


PCB Bottom Signal Layer (Layer 4)

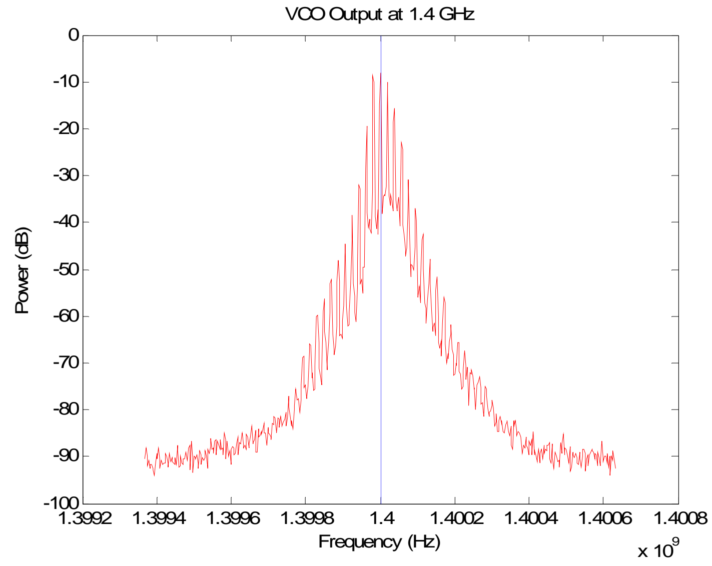
APPENDIX C.
VOLTAGE CONTROLLED OSCILLATOR TESTING RESULTS



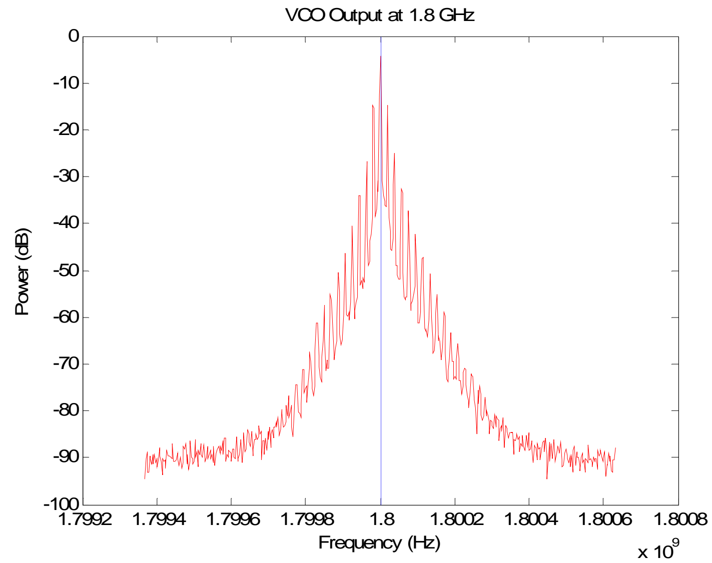
VCO Output at 1.0 GHz



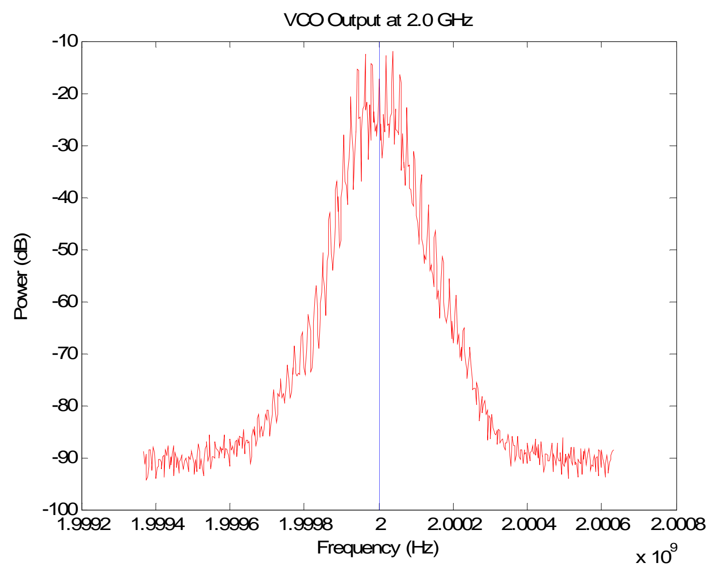
VCO Output at 1.2 GHz



VCO Output at 1.4 GHz



VCO Output at 1.8 GHz



VCO Output at 2.0 GHz

APPENDIX D.

RECEIVER DIGITAL SIGNAL PROCESSOR CODE

File: dspr_main.c

```
#include "dspr_init.h"
#include "data_types.h"
#include "dspr_main.h"
#include "c5502regs.h"


/*-------------- Global Variables -----------------------*/
/* For testing purposes, ibuf is initialized to a 512 length array of evenly
spaced points on a cosine curve and qbuf is initialized to a 512 length of
evenly spaced points on a sine curve.  Once outputed and upconverted, this should
output a sine wave at:

fo = fc + (fs / 512)

where:

fo = output frequency
fc = carrier frequency
fs = DAC (sampling) frequency
 */
uint16 ibuf[BUFSIZE];
uint16 qbuf[BUFSIZE];
uint16 start;
uint16 end;

uint16 rxbuf[UART_BUFSIZE];          // UART Receive Buffer
uint16 rxstart;
uint16 rxend;
uint16 rxbusy;                       // 1 if buffer empty or in process of filling
                                     // 0 if new data in buffer

uint16 txbuf[UART_BUFSIZE];          // UART Transmit Buffer
uint16 txstart;
uint16 txend;
uint16 txbusy;                       // 1 if new data ready to be transmitted or data still transmitting
                                     // 0 if no new data to send

uint32 fs;                           // Current sampling frequency expressed in Hz
uint32 fc;                           // Current Carrier Frequency expressed in kHz
uint16 mod;                          // Current modulation - Not currently used

uint16 ddc_ready;                    // Determines if DDC is ready for another instruction

uint16 code_loc;
uint16 check_ISR[NUM_ISRS];
/* ----------- END GLOBAL VARIABLES ----------------- */

void main(void)
{
        uint16 i,j;

        code_loc = MAIN;

        RX_DISABLE

        for (i=0; i<NUM_ISRS; i++)
        {
                check_ISR[i]=0;
        }

        start = 0;
        end = 0;

        rxstart = 0;
        rxend = 0;
        rxbusy = 0;
```

```
		txstart = 0;
		txend = 0;
		txbusy = 0;

		ddc_ready = 1;

		dspr_init();// Does not globally enable interrupts

		RX_ENABLE

		freqsyn_init();
		vga_gain_init();
		int_enable(INT_GLOBAL);
		ddc_init();


		fs = 1000000;						// in ddc_init, fs is set to 1 MHz
		i = set_carrier(1600000);			// Currently, due to the loop filter and un-ideal nature of the op amp, the
minimum frequency seems to be around 1.35 GHz
											// Can extend range by replacing 4.7 kOhm
resistor in PLL loop with 3.0 kOhm resistor to lower gain (will still meet upper voltage requirements)
		i = set_carrier(1000000);
		i = set_carrier(1200000);
		i = set_carrier(1400000);
		i = set_carrier(1600000);
		i = set_carrier(1800000);
		i = set_carrier(2000000);

		i = set_carrier(1600000);

		set_vga_gain(0x9000);
		set_vga_gain(0xC71B);

		send_lcd("DSPR Initialized", 16);

		i = 0;
		j = 0;

		for (i=0; i>64000; i++);

		int_enable(INT_DATA_RDY);

		while (1)
		{
/*			if ((ST1_55 & bmINT_GLOBAL) != 0x0000)
			{
				int_enable(INT_GLOBAL);
			}
*/			if (rxbusy)
			{
				handle_rxdata();
			}
			if (end == BUFSIZE-1)
			{
				int_disable(INT_DATA_RDY);
				end = 0;
				int_enable(INT_DATA_RDY);
			}

/*			if (URLSR & bmTHRE)		// Transmit Hold Register Empty
			{
				URTHR = SEND_LCD & 0x00FF;
				for (i=0; i<64000; i++);
				URTHR = 18 & 0x00FF;
				for (i=0; i<64000; i++);
				for (j=0; j<16; j++)
				{
					URTHR = message[j] & 0x00FF;
					for (i=0; i<64000; i++);
```

```
                }
            }
*/
            if (check_ISR[NOISR12] >= 18)
            {
                i++;
                if (i>1000)
                {
                    j++;
                    if (j>1000)
                    {
                        check_ISR[NOISR12] = 0;
                        i = 0;
                        j = 0;
                        send_lcd("DSPR Initialized", 16);
                    }
                    else
                    {
                        i = 0;
                    }
                }
            }

        }

}

/****************************************************
void freqsyn_init(void)

This function initializes the frequency synthesizer based
on the initialization perameters defiened in dspr_main.h
****************************************************/
void freqsyn_init(void)
{
        uint16 prev_cl;
        uint32 value;
        prev_cl = code_loc;
        code_loc = INIT_FQSYN;
        /*
        Configures Frequency Synthesizer to start with a 1.000000 GHz output

        F_PFD = RF_IN * (1+D)/R

        RF_OUT = F_PFD * (INT + (FRAC/MOD))

        RF_IN = 20 MHz              (Input Reference Clock)
        D = 0                               (Reference Doubler)
        R = 5                               (R Counter)
        gives F_PFD = 4 MHz

        INT = 250
        FRAC = 0
        MOD = 4000                          (gives resolution of 1 kHz)
        gives RF_OUT = 1.000000 GHz

        */
        RX_DISABLE
        RX_ENABLE

        value = (REG_C_VAL) | 0x10;   // Power-Down mode
        send_freqsyn(value);
        for (value = 0; value<30000000; value++);   // wait a second to let the freq syn reset
        send_freqsyn(0x0003);              // Write all zeros to noise and spur register
        value = REG_NS_VAL;
        send_freqsyn(value);  // Configure Noise and Spur Register
        value = (REG_C_VAL) | 0x0014;               // Keep in Power-Down mode
        send_freqsyn(value);  // Put Counters in reset and congifure control register
        value = REG_R_VAL;
        send_freqsyn(value);  // Configure R Divider Register
```

```
		value = REG_N_VAL;
		send_freqsyn(value);  // Configure N Divider Register
		value = (REG_C_VAL) & ~0x0004;
		send_freqsyn(value);  // Enable Counters and Power Up

		code_loc = prev_cl;
}


/*****************************************************
uint16 set_carrier(uint32 freq)

Sets the carrier frequency of the receiver

Input: desired frequency in kHz
Return: 0 if unsuccessful
			1 if successful

*****************************************************/
uint16 set_carrier(uint32 freq)
{
		// vary the carrier by changind the INT and FRAC values in the N Divider Register
		// 250 <= INT <= 500 for 1.0 GHz - 2.0 GHz operation
		// each increment of INT changes output by 4 MHz
		// each increment of FRAC changes output by 1 kHz

		uint32 n,frac,reg_val;
		uint16 prev_cl;
		uint32 stuck = 0;
		uint16 success = 1;
		prev_cl = code_loc;

		code_loc = SET_CARR;

		if ((freq < FS_FREQ_MIN) || (freq > FS_FREQ_MAX))		// if outside 1-2GHz range, return without changes
		{
				code_loc = prev_cl;
				return 0;
		}
		frac = freq % FS_FPFD;					// calculate frac value
		n = (freq - frac) / FS_FPFD;		// calculate int value
		reg_val = (((uint32)FASTLOCK & 0x1)<<23)+(n << 14) + (frac << 2);		// build value to write to N Divider Register
		send_freqsyn(reg_val);					// send value to frequency synthesizer
		while ((SPCR2_0 & 0x0004) == 0x0004);	// wait for transmission to complete
		for (stuck=0; stuck<3000; stuck++);			// wait for 10 usec (min time to lock detect)
		stuck = 0;
		while (FS_MUXOUT == 0)					// Wait for PLL Lock
		{
				stuck++;
				if (stuck > 500000)
				{
						success = freqsyn_reinit(reg_val);
				}
		}

		fc = freq;
		code_loc = prev_cl;
		return success;
}

/*****************************************************
uint16 freqsyn_reinit(uint32 reg_val)

Reinitialize the frequency synthesizer if it won't lock

Input: register value for the N divider Register
Ouptut: 0 if unsuccessful
			1 if successful

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

```
uint16 freqsyn_reinit(uint32 reg_val)
{
        uint16 prev_cl;
        uint32 value;
        uint16 success = 1;
        uint32 stuck;

        prev_cl = code_loc;
        code_loc = INIT_FQSYN;

        value = (REG_C_VAL) | 0x10;    // Power-Down mode
        send_freqsyn(value);
        for (value = 0; value<30000000; value++);    // wait a second to let the freq syn and VCO react
        send_freqsyn(0x0003);                 // Write all zeros to noise and spur register
        value = REG_NS_VAL;
        send_freqsyn(value);  // Configure Noise and Spur Register
        value = (REG_C_VAL) | 0x0014;                // Keep in Power-Down mode
        send_freqsyn(value);  // Put Counters in reset and congifure control register
        value = REG_R_VAL;
        send_freqsyn(value);  // Configure R Divider Register

        send_freqsyn(reg_val);                // Configure N Divider Register
        value = (REG_C_VAL) & ~0x0014;
        send_freqsyn(value);  // Enable Counters and Power Up

        while (FS_MUXOUT == 0)                                    // Wait for PLL Lock
        {
                stuck++;
                if (stuck > 300000)
                {
                        success = 0;
                }
        }

        code_loc = prev_cl;
        return success;
}


/****************************************************
void send_freqsyn(uint32 data)

Send data to frequency synthesizer via SPI0

Input: 24-bit data to be sent

****************************************************/
void send_freqsyn(uint32 data)
{
        uint16 prev_cl;
        prev_cl = code_loc;
        code_loc = SEND_FQSYN;

        data = data & 0x00FFFFFF;        // Clear upper 8 bits
        while ((SPCR2_0 & bmXEMPTY) == bmXEMPTY);    // wait for transmit buffer to empty
        DXR2_0 = (uint16) ((data & 0x00FF0000) >> 16);       // write upper word to upper transmit register
        DXR1_0 = (uint16) (data & 0x0000FFFF);    // write lower word to transmit register
                                                                                          // this
initiates SPI transfer
        while ((SPCR2_0 & bmXEMPTY) == bmXEMPTY);    // wait for transmit buffer to empty

        code_loc = prev_cl;
}


/****************************************************
void vga_gain_init(void)

Initializes the gain on the VGA to approx. 0 dB
****************************************************/
```

```
void vga_gain_init(void)
{
		uint16 prev_cl;
		prev_cl = code_loc;
		code_loc = INIT_VGA;

		// initializes gain to approximately 0 dB
		set_vga_gain(0x638D);		// corresponds to a voltage of about 0.7 V
		code_loc = prev_cl;
}


/****************************************************
void set_vga_gain(uint16 voltage)

Set the gain on the VGA to the desired value

Input: 16-bit word.  0x0000 = 0 V
				0xFFFF = 1.8 V

****************************************************/
void set_vga_gain(uint16 voltage)
{
		uint16 prev_cl;
		prev_cl = code_loc;
		code_loc = SET_VGA;

		if ((voltage < VGA_GAIN_MIN) || (voltage > VGA_GAIN_MAX))		// if outside valid range, do nothing
		{
				code_loc = prev_cl;
				return;
		}

		while ((SPCR2_1 & bmXEMPTY) == bmXEMPTY);	// wait for transmission to complete
		DXR2_1 = 0x0000;				// bits 17:16 set mode
								// 00: normal operation
								// 01: 3-state
								// 10: 100 kOhm to ground
								// 11: 1 kOhm to ground
		DXR1_1 = voltage;				// write lower word to transmit register
								// this initiates SPI transfer
		while ((SPCR2_1 & bmXEMPTY) == bmXEMPTY);	// wait for transmission to complete
		code_loc = prev_cl;
}



/****************************************************
void send_lcd(char * string, uint16 length)

Send a character string to the microcontroller to be
displayed on the LCD display

Input: character array pointer - can be passed as: "message"
	length: length of the character array

****************************************************/
void send_lcd(char * string, uint16 length)	// length must be <= (UART_BUFSIZE-2)
{
		uint16 prev_cl;
		prev_cl = code_loc;
		code_loc = SEND_MSG;

		while (txbusy);			// Wait for transmitter to become available
		txstart = 0;
		txbuf[0] = SEND_LCD;
		txbuf[1] = length + 2; // add for the control byte and the length byte
		if (length > UART_BUFSIZE - 2)		// if length too long, cut off end of string
		{
```

```
			length = UART_BUFSIZE - 2;
		}
		for (txend=2; txend<length+2; txend++)
		{
			txbuf[txend] = (uint16) string[txend-2];	// Copy string to txbuf
		}
		txend = length+2;
		txbusy = 1;
		if (URLSR & bmTHRE)		// Transmit Hold Register Empty
		{
			URTHR = txbuf[0] & 0x00FF;			// Initiate UART Transfer - For some reason, cannot write to IFR0
		}
		code_loc = prev_cl;
}


/*****************************************************
void int_enable(uint16 int_num)

Enable the given interrupt.  Interrupts are defined in dspr_main.h

*****************************************************/
void int_enable(uint16 int_num)
{
		uint16 prev_cl;
		prev_cl = code_loc;
		code_loc = INT_ENBL;

		switch (int_num)
		{
			case INT_INT0:
				IER0 |= bmINT_INT0;
				break;
			case INT_INT2:
				IER0 |= bmINT_INT2;
				break;
			case INT_TINT0:
				IER0 |= bmINT_TINT0;
				break;
			case INT_RINT0:
				IER0 |= bmINT_RINT0;
				break;
			case INT_RINT1:
				IER0 |= bmINT_RINT1;
				break;
			case INT_XINT1:
				IER0 |= bmINT_XINT1;
				break;
			case INT_DMAC1:
				IER0 |= bmINT_DMAC1;
				break;
			case INT_DSPINT:
				IER0 |= bmINT_DSPINT;
				break;
			case INT_INT3_WDTINT:
				IER0 |= bmINT_INT3_WDTINT;
				break;
			case INT_RINT2_UART:
				IER0 |= bmINT_RINT2_UART;
				break;
			case INT_XINT2:
				IER0 |= bmINT_XINT2;
				break;
			case INT_DMAC4:
				IER0 |= bmINT_DMAC4;
				break;
			case INT_DMAC5:
				IER0 |= bmINT_DMAC5;
				break;
```

```
                case INT_INT1:
                        IER1 |= bmINT_INT1;
                        break;
                case INT_XINT0:
                        IER1 |= bmINT_XINT0;
                        break;
                case INT_DMAC0:
                        IER1 |= bmINT_DMAC0;
                        break;
                case INT_DMAC2:
                        IER1 |= bmINT_DMAC2;
                        break;
                case INT_DMAC3:
                        IER1 |= bmINT_DMAC3;
                        break;
                case INT_TINT1:
                        IER1 |= bmINT_TINT1;
                        break;
                case INT_I2C:
                        IER1 |= bmINT_I2C;
                        break;
                case INT_BERR:
                        IER1 |= bmINT_BERR;
                        break;
                case INT_DLOG:
                        IER1 |= bmINT_DLOG;
                        break;
                case INT_RTOS:
                        IER1 |= bmINT_RTOS;
                        break;
                case INT_GLOBAL:
                        // dummy = ST1_55;
                        // dummy &= ~bmINT_GLOBAL;
                        // ST1 = dummy;
                        asm(" BCLR INTM ");
                        break;
        }
        code_loc = prev_cl;
}


/****************************************************
void int_disable(uint16 int_num)

Disable the given interrupt.  Interrupts are defined in dspr_main.h

****************************************************/
void int_disable(uint16 int_num)
{
        uint16 prev_cl;
        prev_cl = code_loc;
        code_loc = INT_DABL;

        switch (int_num)
        {
                case INT_INT0:
                        IER0 &= ~bmINT_INT0;
                        break;
                case INT_INT2:
                        IER0 &= ~bmINT_INT2;
                        break;
                case INT_TINT0:
                        IER0 &= ~bmINT_TINT0;
                        break;
                case INT_RINT0:
                        IER0 &= ~bmINT_RINT0;
                        break;
                case INT_RINT1:
                        IER0 &= ~bmINT_RINT1;
```

```
                        break;
                case INT_XINT1:
                        IER0 &= ~bmINT_XINT1;
                        break;
                case INT_DMAC1:
                        IER0 &= ~bmINT_DMAC1;
                        break;
                case INT_DSPINT:
                        IER0 &= ~bmINT_DSPINT;
                        break;
                case INT_INT3_WDTINT:
                        IER0 &= ~bmINT_INT3_WDTINT;
                        break;
                case INT_RINT2_UART:
                        IER0 &= ~bmINT_RINT2_UART;
                        break;
                case INT_XINT2:
                        IER0 &= ~bmINT_XINT2;
                        break;
                case INT_DMAC4:
                        IER0 &= ~bmINT_DMAC4;
                        break;
                case INT_DMAC5:
                        IER0 &= ~bmINT_DMAC5;
                        break;
                case INT_INT1:
                        IER1 &= ~bmINT_INT1;
                        break;
                case INT_XINT0:
                        IER1 &= ~bmINT_XINT0;
                        break;
                case INT_DMAC0:
                        IER1 &= ~bmINT_DMAC0;
                        break;
                case INT_DMAC2:
                        IER1 &= ~bmINT_DMAC2;
                        break;
                case INT_DMAC3:
                        IER1 &= ~bmINT_DMAC3;
                        break;
                case INT_TINT1:
                        IER1 &= ~bmINT_TINT1;
                        break;
                case INT_I2C:
                        IER1 &= ~bmINT_I2C;
                        break;
                case INT_BERR:
                        IER1 &= ~bmINT_BERR;
                        break;
                case INT_DLOG:
                        IER1 &= ~bmINT_DLOG;
                        break;
                case INT_RTOS:
                        IER1 &= ~bmINT_RTOS;
                        break;
                case INT_GLOBAL:
                        asm(" BSET INTM ");
                        //ST1_55 |= bmINT_GLOBAL;
                        break;
        }
        code_loc = prev_cl;
}


/*****************************************************
interrupt void int0_ISR(void)

Interrupt Service Routine for external interrupt 0,
controlled by the output of timer 0 (TIM0).  Interrupt
```

```
signals new data available from the FPGA

****************************************************/
interrupt void int0_ISR(void)
{
	uint16 prev_cl;
	prev_cl = code_loc;
	code_loc = INT0ISR;
	check_ISR[NOISR2] += 1;

	if (end != (start-1)%BUFSIZE)
	{
		ibuf[end] = FPGAI;				// Read in the data from the FPGAI output
		qbuf[end] = FPGAQ;				// Read in the data from the FPGAQ output
		end = (++end) % BUFSIZE;	// increment end
	}
	else
	{
		int_disable(INT_DATA_RDY);
	}
	code_loc = prev_cl;
}

/****************************************************
interrupt void int1_ISR(void)

Interrupt Service Routine for external interrupt 1,
controlled by the data acknoledgment from the I signal DDC

****************************************************/
interrupt void int1_ISR(void)
{
	uint16 i;
	uint16 prev_cl;

	DDCI_DESEL

	prev_cl = code_loc;
	code_loc = INT1ISR;
	check_ISR[NOISR16] += 1;

	for (i=0; i<6; i++);	// wait for enough time to pass for the next clock of
								// the DDC clock (50 MHz)
	ddc_ready = 1;
	code_loc = prev_cl;
}


/****************************************************
interrupt void int2_ISR(void)

Interrupt Service Routine for external interrupt 2,
controlled by the data acknoledgment from the Q signal DDC

****************************************************/
interrupt void int2_ISR(void)
{
	uint16 i;
	uint16 prev_cl;

	DDCQ_DESEL

	prev_cl = code_loc;
	code_loc = INT2ISR;
	check_ISR[NOISR3] += 1;

	for (i=0; i<6; i++);	// wait for enough time to pass for the next clock of
								// the DDC clock (50 MHz)
	ddc_ready = 1;
```

```
		code_loc = prev_cl;
}


/****************************************************
interrupt void int3_ISR(void)

ISR for external interrupt 3, controlled by the ADC overflow
signal compiled by the FPGA

****************************************************/
interrupt void int3_ISR(void)
{
		uint16 prev_cl;
		prev_cl = code_loc;
		code_loc = INT3ISR;
		check_ISR[NOISR11] += 1;

		code_loc = prev_cl;

}


/****************************************************

void uart_ISR(void)

Interrupt Service Routine for UART communication

Communication is done 8 bits at a time so buffers need to be
filled and read accordingly.

Format of Packet:
Byte 0: Control Byte
Byte 1: Total Packet Length (max length: 64)
Bytes 2...63: Data Bytes

****************************************************/
interrupt void uart_ISR(void)
{
		uint16 urlsr_holder;

		uint16 prev_cl;
		prev_cl = code_loc;
		code_loc = UARTISR;
		check_ISR[NOISR12] += 1;

		urlsr_holder = URLSR;

		if (urlsr_holder & bmTHRE)	// Transmit Hold Register Empty
		{
			if (txbusy) // if data still ready to be transmitted
			{
				URTHR = txbuf[txstart] & 0x00FF;		// Can only transmit lower 8 bits
				txstart = (++txstart) % UART_BUFSIZE;
				if (txstart == txend)	// All data has been transmitted
				{
					txstart = 0;
					txend = 0;
					txbusy = 0;
				}
			}
		}
		if (urlsr_holder & bmDR)		// Receive Data Ready
		{
			if (rxbusy)				// If buffer available
			{
				if (rxend < UART_BUFSIZE)	// Make sure of no buffer overflow
				{
```

```
                    rxbuf[rxend] = URRBR;           // Read data
                    rxend++;                        // increment end
                }
                if ((rxend > 1) && (rxend >= rxbuf[1]))     // if received length byte and have received that many
bytes
                {
                    rxstart = 0;
                    rxbusy = 0;
                }
            }
    }
    code_loc = prev_cl;
}

/******************************************************
Interrupt Service Routines for all interrupts by number to
track if ISRs are called

******************************************************/
interrupt void no_ISR1(void) {check_ISR[NOISR1] += 1;}
interrupt void no_ISR2(void) {check_ISR[NOISR2] += 1;}
interrupt void no_ISR3(void) {check_ISR[NOISR3] += 1;}
interrupt void no_ISR4(void) {check_ISR[NOISR4] += 1;}
interrupt void no_ISR5(void) {check_ISR[NOISR5] += 1;}
interrupt void no_ISR6(void) {check_ISR[NOISR6] += 1;}
interrupt void no_ISR7(void) {check_ISR[NOISR7] += 1;}
interrupt void no_ISR8(void) {check_ISR[NOISR8] += 1;}
interrupt void no_ISR9(void) {check_ISR[NOISR9] += 1;}
interrupt void no_ISR10(void) {check_ISR[NOISR10] += 1;}
interrupt void no_ISR11(void) {check_ISR[NOISR11] += 1;}
interrupt void no_ISR12(void) {check_ISR[NOISR12] += 1;}
interrupt void no_ISR13(void) {check_ISR[NOISR13] += 1;}
interrupt void no_ISR14(void) {check_ISR[NOISR14] += 1;}
interrupt void no_ISR15(void) {check_ISR[NOISR15] += 1;}
interrupt void no_ISR16(void) {check_ISR[NOISR16] += 1;}
interrupt void no_ISR17(void) {check_ISR[NOISR17] += 1;}
interrupt void no_ISR18(void) {check_ISR[NOISR18] += 1;}
interrupt void no_ISR19(void) {check_ISR[NOISR19] += 1;}
interrupt void no_ISR20(void) {check_ISR[NOISR20] += 1;}
interrupt void no_ISR21(void) {check_ISR[NOISR21] += 1;}
interrupt void no_ISR22(void) {check_ISR[NOISR22] += 1;}
interrupt void no_ISR23(void) {check_ISR[NOISR23] += 1;}
interrupt void no_ISR24(void) {check_ISR[NOISR24] += 1;}
interrupt void no_ISR25(void) {check_ISR[NOISR25] += 1;}
interrupt void no_ISR26(void) {check_ISR[NOISR26] += 1;}
interrupt void no_ISR27(void) {check_ISR[NOISR27] += 1;}
interrupt void no_ISR28(void) {check_ISR[NOISR28] += 1;}
interrupt void no_ISR29(void) {check_ISR[NOISR29] += 1;}
interrupt void no_ISR30(void) {check_ISR[NOISR30] += 1;}
interrupt void no_ISR31(void) {check_ISR[NOISR31] += 1;}



/******************************************************
uint16 push_txbuf16(uint16 data)

Splits a 16-bit word into two 8-bit pieces and a places them in txbuf
Word is transmitted Most Significant Byte first (Big Endian)

Returns:
0: Buffer full and push failed
1: Push succeeded
******************************************************/
uint16 push_txbuf16(uint16 data)
{
    uint16 prev_cl;
    prev_cl = code_loc;
    code_loc = PUSH_TX16;
```

```
		if (txend < UART_BUFSIZE - 1)// if there is enough room in the buffer
		{
			txbuf[txend] = (data & 0xFF00) >> 8;
			txbuf[txend+1] = data & 0x00FF;
			txend += 2;
			code_loc = prev_cl;
			return 1;
		}
		else
		{
			code_loc = prev_cl;
			return 0;
		}

}



/****************************************************
uint16 push_txbuf32(uint32 data)

Splits a 32-bit word into four 8-bit pieces and a places them in txbuf
Word is transmitted Most Significant Byte first (Big Endian)

Returns:
0: Buffer full and push failed
1: Push succeeded
****************************************************/
uint16 push_txbuf32(uint32 data)
{
		uint16 prev_cl;
		prev_cl = code_loc;
		code_loc = PUSH_TX32;

		if (txend < UART_BUFSIZE - 3)// if there is enough room in the buffer
		{
			txbuf[txend] = (data & 0xFF000000) >> 24;
			txbuf[txend+1] = (data & 0x00FF0000) >> 16;
			txbuf[txend+2] = (data & 0x0000FF00) >> 8;
			txbuf[txend+3] = (data & 0x000000FF);
			txend += 4;
			code_loc = prev_cl;
			return 1;
		}
		else
		{
			code_loc = prev_cl;
			return 0;
		}

}


/****************************************************
void rxbuf_reset(void)

Receive buffer reset.  Called when data received by the UART
has been handled.

****************************************************/
void rxbuf_reset(void)
{
		uint16 prev_cl;
		prev_cl = code_loc;
		code_loc = RX_RESET;

		rxstart = 0;
		rxend = 0;
		rxbusy = 1;
```

```
		code_loc = prev_cl;
}


/****************************************************
void handle_rxdata(void)

Handles data received via the UART based on the control byte
(first byte int the transmission)
****************************************************/
void handle_rxdata(void)
{
		uint16 control;

		uint16 prev_cl;
		prev_cl = code_loc;
		code_loc = HDL_RX;

		control = rxbuf[0];
		switch (control & 0x009F)
		{
			case SET_FC:
				fc = (rxbuf[2] << 8) + rxbuf[3];
				set_carrier(fc);
				break;
			case SET_BW:
				// Do nothing here
				break;
			case SET_FS:
				// Not implemented yet - will need to change DDC configuration
				break;
			case SET_MOD:					// Not really anthing to do with this one yet
				mod = rxbuf[2];
				break;
			case GET_FC:
				if (txbusy == 0)		// if no data already being sent
				{
					txbuf[0] = control;
					txbuf[1] = 6;
					push_txbuf32(fc);
					txbusy = 1;
				}
				else
				{
					code_loc = prev_cl;
					return;
				}
				break;
			case GET_BW:
				if (txbusy == 0)		// if no data already being sent
				{
					txbuf[0] = control;
					txbuf[1] = 6;
					push_txbuf32(0x0000);
					txbusy = 1;
				}
				else
				{
					code_loc = prev_cl;
					return;
				}
				break;
			case GET_FS:
				if (txbusy == 0)		// if no data already being sent
				{
					txbuf[0] = control;
					txbuf[1] = 6;
					push_txbuf32(fs);
					txbusy = 1;
				}
```

```
                        else
                        {
                                code_loc = prev_cl;
                                return;
                        }
                        break;
                case GET_MOD:
                        if (txbusy == 0)        // if no data already being sent
                        {
                                txbuf[0] = control;
                                txbuf[1] = 3;
                                txbuf[3] = mod;
                                txbusy = 1;
                        }
                        else
                        {
                                code_loc = prev_cl;
                                return;
                        }
                        break;
                case GET_DATA:
                        // This will be used in the future - do nothing yet
                        break;
                case GET_SMPL:
                        // This will be used in the future - do nothing yet
                        break;

        }
        rxbuf_reset();          // Release rxbuf
        code_loc = prev_cl;
}


/****************************************************
void ddc_init(void)

Initialize the digital down converters.  This function currenty
sets the NCO to 5 MHz, the sampling frequency to 1 MHz, and the
noise/spur register to low noise mode.

****************************************************/
void ddc_init(void)
{
        uint16 i,chip;
        uint16 prev_cl;
        prev_cl = code_loc;

        code_loc = INIT_DDC;


        for (chip=0; chip<2; chip++)                    // initialize both DDCs the same (except Sync Master/Slave: DDCI - Master,
DDCQ - Slave
        {
                if (chip == DDCI)
                {
                        DDCI_RST
                        for (i=0; i<20; i++);   // Wait at least 30 nsec
                        DDCI_RUN
                }
                else
                {
                        DDCQ_RST
                        for (i=0; i<20; i++);   // Wait at least 30 nsec
                        DDCQ_RUN
                }

                write_ddc(chip,0x000,0x7FFFF); // Make first tap of RCF filter = full scale - this will be the only tap
                write_ddc(chip,0x100,0x00000); // Make first RCF Data RAM = 0
                if (chip == DDCI)
                {
```

```
				write_ddc(DDCI,0x300,0x09);	// Set as Sync Master
			}
			else
			{
				write_ddc(DDCQ,0x300,0x01);	// Set as Sync Slave
			}
			write_ddc(chip,0x301,0x00);				// Enable NCO
			write_ddc(chip,0x302,0xFFFFFFFF);		// Set sync mask to full value
			write_ddc(chip,0x303,0x1999999A);		// Set NCO_FREQ to 5 MHz
			write_ddc(chip,0x304,0x0000);			// Set NCO Phase Offset to 0
			write_ddc(chip,0x305,0x01);				// Set ExpOff = 0, ExpInv = 0, Scic2 = 1
			write_ddc(chip,0x306,0x01);				// Set Mcic2 = 2 (Decimate by 2)
			write_ddc(chip,0x307,0x0F);				// Set Scic5 = 15
			write_ddc(chip,0x308,0x19);				// Set Mcic5 = 25
			write_ddc(chip,0x309,0x06);				// Set Output Gain to 1/4: value of 6
			write_ddc(chip,0x30A,0x00);				// Set RCF decimation to 1 - no filter
			write_ddc(chip,0x30B,0x00);				// set RCF filter coefficient offset to 0
			write_ddc(chip,0x30C,0x00);				// Set number of taps to 1
			write_ddc(chip,0x30D,0x00);				// Reserved (should be written 0)
			if (chip == DDCI)
			{
				write_ddc(DDCI,0x300,0x08);	// Set as Sync Master and bring out of reset
			}
			else
			{
				write_ddc(DDCQ,0x300,0x00);	// Set as Sync Slave and bring out of reset
			}
		}


		code_loc = prev_cl;
}


/*****************************************************
void write_micro(uint16 chip, uint16 addr, uint16 data)

Write data to the microport of the DDCs

Input: chip - 0 for DDCI, 1 for DDCQ
	   addr - desired external address
	   data - desired 8-bit data

*****************************************************/
void write_micro(uint16 chip, uint16 addr, uint16 data)
{
		uint16 stuck = 0;
		uint16 prev_cl;
		prev_cl = code_loc;

		code_loc = WRT_MICRO;

		while (ddc_ready == 0)			// wait for DDC to become available
		{
			stuck++;
			if (stuck > 500)
			{
				stuck = 0;
				break;
			}
		}
		DDC_PORT = 0x0001 + ((addr & 0x0007) << 1) + ((data & 0x00FF) << 4);
		ddc_ready = 0;
		switch (chip)
		{
			case DDCI:
				DDCI_SEL
				break;
			case DDCQ:
				DDCQ_SEL
```

```
                        break;
        }
        while (ddc_ready == 0)                  // wait for DDC to become available
        {
                stuck++;
                if (stuck > 1000)
                {
                        switch (chip)
                        {
                                case DDCI:
                                        DDCI_DESEL
                                        break;
                                case DDCQ:
                                        DDCQ_DESEL
                                        break;
                        }
                        for (stuck=0; stuck<6; stuck++);  // wait for enough time to pass for the next clock of

        // the DDC clock (50 MHz)
                        ddc_ready = 1;
                        stuck = 1;
                }
        }

        code_loc = prev_cl;
}


/****************************************************
void write_ddc(uint16 chip, uint16 addr, uint40 data)

Write data to the internal memory of the DDCs.

Inputs: chip - 0 for DDCI, 1 for DDCQ
                addr - internal address of DDCI
                data - up to 36-bit data word

****************************************************/
void write_ddc(uint16 chip, uint16 addr, uint40 data)
{
        uint16 prev_cl;
        prev_cl = code_loc;
        code_loc = WRT_DDC;

        /* Need to write to register DR0 as this causes the DDC to latch all the data */
        if ((addr <= 0x1FF) || ((addr >= 0x300) && (addr <= 0x30D)))     // Ensure valid internal address
        {
                write_micro(chip,DDC_AMR,(addr >> 8) & 0x0003);
                write_micro(chip,DDC_LAR,addr & 0x00FF);
                write_micro(chip,DDC_DR4,(data >> 32) & 0x000F);
                write_micro(chip,DDC_DR3,(data >> 24) & 0x00FF);
                write_micro(chip,DDC_DR2,(data >> 16) & 0x00FF);
                write_micro(chip,DDC_DR1,(data >> 8) & 0x00FF);
                write_micro(chip,DDC_DR0,data & 0x00FF);
        }

        code_loc = prev_cl;
}
```

File: dspr_main.h

```
#ifndef DSPT_MAIN_H
#define DSPT_MAIN_H

#include "dspr_init.h"
#include "data_types.h"

/* -------------- Changeable Parameters ---------------------- */
#define SPI0_SCLK_DIV       75          // Divedes the Slow Peripheral Clock (75 MHz) to get SCLK for SPI0
                                        // Acceptable Values: 1 - 256
#define SPI1_SCLK_DIV       75          // Divedes the Slow Peripheral Clock (75 MHz) to get SCLK for SPI1
                                        // Acceptable Values: 1 - 256
#define UART_BAUD_DIV       82          // Divisor = 82 to make baud rate 57165 (75 MHz/(16*82))


#define FS_FPFD     (uint32)4000        // Frequency Synthesizer Phase Frequency Detector (in kHz)
                                        // Set by varying D and R (see freqsyn_init(void) comments)
#define FS_FREQ_MIN         1000000     // Minimum Frequency for frequency synthesizer
#define FS_FREQ_MAX         2000000     // Maximum Frequency for frequency synthesizer

#define FS_INT_MIN 250      // Minimum value for INT in freq synthesizer (corresponds to 1 GHz)
#define FS_INT_MAX 500      // Maximum value for INT in freq synthesizer (corresponds to 2 GHz)

#define VGA_GAIN_MIN        0x0000  //0x471C   // Minimum gain for the VGA (corresponds to 0.5V: GAIN Vref = 1.8V)
#define VGA_GAIN_MAX        0xC71B  // Maximum gain for the VGA (corresponds to 1.4V: GAIN Vref = 1.8V)

#define BUFSIZE                     2048        // Buffer size for ibuf and qbuf
#define UART_BUFSIZE        64          // Buffer size for rxbuf and txbuf

#define SYSCLK1     150000000           // Fast Peripherals Clock Group
#define SYSCLK2     75000000  // Slow Peripherals Clock Group
#define SYSCLK3     150000000           // External Memroy Interface Clock Group

/* Control Bytes for UART Communication */
/* Possible OUT Control Bytes */
#define SET_FC              0x0000      // Set Carrier Frequency
#define SET_BW              0x0001      // Set Bandwidth - This is not really applicable on the transmitter
#define SET_FS              0x0002      // Set Sampling Frequency
#define SET_MOD             0x0003      // Set Modulation Technique
/* Possible IN Control Bytes - much change bits 6..5 depending on EP used */
#define     GET_FC          0x0080      // Get Current Carrier Frequency
#define GET_BW              0x0081      // Get Current Bandwidth
#define GET_FS              0x0082      // Get Current Sampling Frequency
#define GET_MOD             0x0083      // Get Current Modulation
#define GET_DATA  0x0084    // Get data - Not currently used
#define GET_SMPL  0x0085    // Get samples
/* Possible Control Bytes to Microcontroller */
#define SEND_LCD  0x0001    // Send characters to LCD

/* Frequency Synthesizer Initialization Values */
/* N Divider Register */
#define FASTLOCK  0                     // 0: Disabled
                                        // 1: Enabled
#define INTEGER             500         // Valid Range: 31 - 511 if PRESCALER = 0
                                        //                          91 - 511 if
PRESCALER = 1
#define FRAC                0           // Valid Range: 0 - (MOD-1)
#define REG_N               0           // Control Bits for N divider register
/* R Divider Register */
#define LD_CNTL             0           // Load Control: 0: Normal Operation
                                        //                          1: Load
Resync
#define MUXOUT              1           // 0: Three-State
                                        // 1: Digital Lock Detect
                                        // 2: N Divider Output
                                        // 3: Logic High
                                        // 4: R Divider Output
                                        // 5: Analog Lock Detect
                                        // 6: Fastlock Switch
```

```
                                                // 7: Logic Low
#define PRESCALER 0                     // 0: 4/5 - Can only go up to 2 GHz
                                                // 1: 8/9
#define R_COUNTER            5                  // Valid Range: 1 - 15
#define MOD                         4000        // Valid Range: 2 - 4095
#define REG_R                1                  // Control Bits for R divider register
/* Control Register */
#define RESYNC               1                  // Valid Range: 1 - 15
#define REF_DBL              0                  // Reference Doubler: 0: Disabled
                                                //                        1:
Enabled
#define CP_CRNT              0x8                // CP Current Setting - See ADF4153 Data sheet pg 13
#define PD_POL               1                  // PD Polarity: 0: Negative
                                                //                        1: Positive
#define LDP                         1                  // Lock Detect Precision: 0: 24 PFD Cycles
                                                //
        1: 40 PFD Cycles
#define PWR_DWN              0                  // Powerdown: 0: Normal Operation
                                                //                   1: Power Down
#define CP_TRI               0                  // CP Three-State: 0: Disabled
                                                //                        1: Three-
State
#define COUNT_RST            0                  // Counter Reset: 0: Normal Operation
                                                //                        1:
Counters in Reset
#define REG_CNTL  2                     // // Control Bits for control register
/* Noise and Spur Register */
#define REG_NS_VAL           0x03C7     // Noise and Spur Mode - includes control bits
                                                // 0x0003: Lowest Spur Mode
                                                // 0x0383: Low Noise and Low Spur Mode
                                                // 0x03C7: Lowest Noise Mode
/* Register Value Computations */
#define REG_N_VAL            (((uint32)FASTLOCK & 0x1)<<23)+(((uint32)INTEGER & 0x1FF)<<14)+(((uint32)FRAC &
0xFFF)<<2)+(REG_N & 0x3)
#define REG_R_VAL((uint32)(LD_CNTL & 0x1)<<23)+((uint32)(MUXOUT & 0x7)<<20)+((uint32)(PRESCALER &
0x1)<<18)+((uint32)(R_COUNTER & 0xF)<<14)+((uint32)(MOD & 0xFFF)<<2)+(REG_R & 0x3)
#define REG_C_VAL((uint32)(RESYNC & 0xF)<<12)+((REF_DBL & 0x1)<<11)+((CP_CRNT & 0xF)<<7)+((PD_POL &
0x1)<<6)+((LDP & 0x1)<<5)+((PWR_DWN & 0x1)<<4)+((CP_TRI & 0x1)<<3)+((COUNT_RST & 0x1)<<2)+(REG_CNTL &
0x3)

/* --------- Bit Masks and Pin Function Definitions ------------ */
/* Bit Mask for SPCR2 regiter for McBSPs */
#define bmXRDY      0x0002    // Bit mask for the transmit buffer ready bit of SPCR2
#define bmXEMPTY  0x0004    // Bit mask for all transmissions
#define bmRRDY      0x0002    // bit mask for data recieved bit of SPCR1

/* Bit Masks for UART Line Status Register (URLSR) */
#define bmTEMT      0x0040    // Transmitter Empty in URLSR register (UART Line Status Register)
#define bmTHRE      0x0020    // Transmitter Holding Register Empty in URLSR
#define bmDR        0x0001    // Receiver Data Ready indicator in URLSR

/* UART Trigger Interrupt command */
#define TRIGGER_UART         IFR0 |= 0x1000;                // Triggers a UART interrupt to go to uart_ISR


/* HGPIO pins */
#define RX_ENABLE            HGPIODAT2 |= 0x0002;           // HCS
#define RX_DISABLE           HGPIODAT2 &= ~0x0002;          // HCS
#define ADCI_OVR   (HGPIODAT2 & 0x0004)          // HDS1
#define ADCQ_OVR (HGPIODAT2 & 0x0008)          // HDS2
#define DDCI_DESEL           HGPIODAT2 |= 0x0010;           // HRW
#define DDCI_SEL    HGPIODAT2 &= ~0x0010;        // HRW
#define FS_MUXOUT            (HGPIODAT2 & 0x0040)           // HCNTL1
#define DDCI_RST    HGPIODAT2 &= ~0x0080;        // HCNTL2
#define DDCI_RUN   HGPIODAT2 |= 0x0080;          // HCNTL2
#define FPGA_OVR   (HGPIODAT2 & 0x0100)          // HINT
/* GPIO pins */
#define DDCQ_RST  IODATA &= ~0x0020;                        // GPIO5
#define DDCQ_RUN IODATA |= 0x0020;               // GPIO5
#define DDCQ_DESEL           IODATA |= 0x0008;              // GPIO3
```

```
#define DDCQ_SEL   IODATA &= ~0x0008;                    // GPIO3
/* PGPIO Pins */
#define FPGAI             PGPIODAT0
#define FPGAQ             PGPIODAT1
#define DDC_PORT   PGPIODAT2
/* DDC constants */
#define DDCI       0
#define DDCQ       1
// External Interface Registers
#define DDC_DR0    0                 // D[7:0]
#define DDC_DR1    1                 // D[15:8]
#define DDC_DR2    2                 // D[23:16]
#define DDC_DR3    3                 // D[31:24]
#define DDC_DR4    4                 // D[35:32]
#define DDC_LAR 6         // A[7:0]
#define DDC_AMR 7         // Bits 1:0 - A[9:8]
                                                         // Bit 5:2   - Reserved
                                                         // Bit 6     - Read Increment
                                                         // Bit 7     - Write Increment
/* Interrupts: for use with int_enable and int_disable */
#define INT_DATA_RDY        INT_INT0
#define INT_DDCI_DTACK      INT_INT1
#define INT_DDCQ_DTACK      INT_INT2
#define INT_ADC_OVR                   INT_INT3_WDTINT
#define INT_UART            INT_RINT2_UART

#define INT_GLOBAL                    32                 // ST1_55
#define INT_INT0            2                  // IER0
#define INT_INT2            3                  // IER0
#define INT_TINT0           4                  // IER0
#define INT_RINT0           5                  // IER0
#define INT_RINT1           6                  // IER0
#define INT_XINT1           7                  // IER0
#define INT_DMAC1           9                  // IER0
#define INT_DSPINT          10                 // IER0
#define INT_INT3_WDTINT     11                 // IER0
#define INT_RINT2_UART      12                 // IER0
#define INT_XINT2           13                 // IER0
#define INT_DMAC4           14                 // IER0
#define INT_DMAC5           15                 // IER0

#define INT_INT1            16                 // IER1
#define INT_XINT0           17                 // IER1
#define INT_DMAC0           18                 // IER1
#define INT_DMAC2           20                 // IER1
#define INT_DMAC3           21                 // IER1
#define INT_TINT1           22                 // IER1
#define INT_I2C                       23                 // IER1
#define INT_BERR            24                 // IER1
#define INT_DLOG            25                 // IER1
#define INT_RTOS            26                 // IER1

/* Bit Mask for INTM in ST1_55 */
#define bmINT_GLOBAL                  0x0800             // set to diable interrupts, clear to enable interrupts
/* Bit Masks for IER0, IFR0 */
#define bmINT_DMAC5                            0x8000
#define bmINT_DMAC4                            0x4000
#define bmINT_XINT2                            0x2000
#define bmINT_RINT2_UART    0x1000
#define bmINT_INT3_WDTINT 0x0800
#define bmINT_DSPINT                  0x0400
#define bmINT_DMAC1                            0x0200
#define bmINT_XINT1                            0x0080
#define bmINT_RINT1                            0x0040
#define bmINT_RINT0                            0x0020
#define bmINT_TINT0                            0x0010
#define bmINT_INT2                    0x0008
#define bmINT_INT0                    0x0004
/* Bit Masks for IER1, IFR1 */
#define bmINT_RTOS                             0x0400
```

```
#define bmINT_DLOG		0x0200
#define bmINT_BERR		0x0100
#define bmINT_I2C	0x0080
#define bmINT_TINT1		0x0040
#define bmINT_DMAC3		0x0020
#define bmINT_DMAC2		0x0010
#define bmINT_INT4	0x0008
#define bmINT_DMAC0		0x0004
#define bmINT_XINT0		0x0002
#define bmINT_INT1	0x0001


/* Code Locations */
#define NUM_LOCS 32

#define	INIT_CLK0
#define INIT_TMR	1
#define INIT_GPIO	2
#define INIT_HPI	3
#define INIT_SPI0	4
#define INIT_SPI1	5
#define INIT_UART	6
#define INIT_INT	7
#define INIT_FQSYN		8
#define SET_CARR	9
#define SEND_FQSYN		10
#define INIT_VGA	11
#define SET_VGA		12
#define SEND_MSG	13
#define INT_ENBL	14
#define INT_DABL	15
#define INT0ISR		16
#define INT1ISR		17
#define INT2ISR		18
#define INT3ISR		19
#define NOINTISR	20
#define UARTISR		21
#define PUSH_TX16	22
#define PUSH_TX32	23
#define RX_RESET	24
#define HDL_RX		25
#define	INIT_DDC	26
#define WRT_MICRO		27
#define WRT_DDC		28
#define INIT_I2C	29
#define INIT_DSPR	30
#define MAIN		31

/* ISR NUMBERS */
#define NUM_ISRS	32

#define NOISR0		0		// RESET
#define NOISR1		1		// NMI
#define NOISR2		2		// INT0
#define NOISR3		3		// INT2
#define NOISR4		4		// TINT0
#define NOISR5		5		// RINT0
#define NOISR6		6		// RINT1
#define NOISR7		7		// XINT1
#define NOISR8		8		// LCKINT1
#define NOISR9		9		// DMAC1
#define NOISR10		10		// DSPINT
#define NOISR11		11		// INT3/WDTINT
#define NOISR12		12		// RINT2/UART
#define NOISR13		13		// XINT2
#define NOISR14		14		// DMAC4
#define NOISR15		15		// DMAC5
#define NOISR16		16		// INT1
#define NOISR17		17		// XINT0
#define NOISR18		18		// DMAC0
```

```
#define NOISR19            19          //
#define NOISR20            20          // DMAC2
#define NOISR21            21          // DMAC3
#define NOISR22            22          // TINT1
#define NOISR23            23          // IIC
#define NOISR24            24          // BERR
#define NOISR25            25          // DLOG
#define NOISR26            26          // RTOS
#define NOISR27            27          //
#define NOISR28            28          //
#define NOISR29            29          //
#define NOISR30            30          //
#define NOISR31            31          //


extern uint16 check_ISR[NUM_ISRS];
extern uint16 code_loc;


void freqsyn_init(void);
uint16 set_carrier(uint32 freq);    // in kHz
uint16 freqsyn_reinit(uint32 reg_val);        // Frequency Synthesizer reinitialize when it gets hopelessly unlocked
void send_freqsyn(uint32 data);   // will only use right-most 24 bits

void vga_gain_init(void);                   // initialize gain to approx 0 dB
void set_vga_gain(uint16 voltage);          // valid inputs correspond to 0.5V to 1.4V
                                                                                  // 0x1999 - 0x47AE

void send_lcd(char * string, uint16 length);   // Send a character string to the LCD connected to the microcontroller

void int_enable(uint16 int_num); // enable interrupt in reg with bitmask bm
void int_disable(uint16 int_num);// disable interrupt in reg with bitmask bm
interrupt void int0_ISR(void);              // External Interrupt 0 ISR - FPGA Data Ready
interrupt void int1_ISR(void);              // External Interrupt 1 ISR - DDCI DTACK
interrupt void int2_ISR(void);              // External Interrupt 2 ISR - DDCQ DTACK
interrupt void int3_ISR(void);              // External Interrupt 3 ISR - ADC Overflow
interrupt void uart_ISR(void);              // UART ISR

interrupt void no_ISR1(void);
interrupt void no_ISR2(void);
interrupt void no_ISR3(void);
interrupt void no_ISR4(void);
interrupt void no_ISR5(void);
interrupt void no_ISR6(void);
interrupt void no_ISR7(void);
interrupt void no_ISR8(void);
interrupt void no_ISR9(void);
interrupt void no_ISR10(void);
interrupt void no_ISR11(void);
interrupt void no_ISR12(void);
interrupt void no_ISR13(void);
interrupt void no_ISR14(void);
interrupt void no_ISR15(void);
interrupt void no_ISR16(void);
interrupt void no_ISR17(void);
interrupt void no_ISR18(void);
interrupt void no_ISR19(void);
interrupt void no_ISR20(void);
interrupt void no_ISR21(void);
interrupt void no_ISR22(void);
interrupt void no_ISR23(void);
interrupt void no_ISR24(void);
interrupt void no_ISR25(void);
interrupt void no_ISR26(void);
interrupt void no_ISR27(void);
interrupt void no_ISR28(void);
interrupt void no_ISR29(void);
interrupt void no_ISR30(void);
interrupt void no_ISR31(void);
```

```
uint16 push_txbuf16(uint16 data);           // Used to split a 16-bit word into two and place in txbuf
uint16 push_txbuf32(uint32 data);           // Used to split a 32-bit word into four and place in txbuf
void rxbuf_reset(void);                               // Resets rxbuf when data used
void handle_rxdata(void);                   // Handle received data

void ddc_init(void);                        // Initialize both I and Q digital down converters
void write_micro(uint16 chip, uint16 addr, uint16 data);// Write the given data to the given external address of the given DDC
void write_ddc(uint16 chip, uint16 addr, uint40 data);  // Write the given data to the given internal address of the given DDC


#endif
```

File: dspr_init.c

```
#include "dspr_init.h"
#include "dspr_main.h"
#include "data_types.h"
#include "c5502regs.h"


/****************************************************
void dspr_init(void)

This function simply calls all the other initialization
functions.

****************************************************/
void dspr_init(void)
{
	uint16 prev_cl;			// Holds the previous code location and will return code_loc to it at the end of the
function

/* External Bus Control */
	prev_cl = code_loc;
	code_loc = INIT_DSPR;
	XBCR = 0x0000;			// Enable pull-up, pull-down, and bus holders on all pins (p.97)

	i2c_init();
	clock_init();
	timer_init();
	gpio_init();
	hpi_init();
	spi0_init();
	spi1_init();
	uart_init();
	int_init();

	RX_DISABLE			// Disable Receiver
	DDCI_DESEL			// Deselect DDCI
	DDCQ_DESEL			// Deselect DDCQ
	DDCI_RST			// Put DDCI		in hard reset
	DDCQ_RST			// Put DDCQ		in hard reset

	code_loc = prev_cl;
}


/****************************************************
void i2c_init(void)

This function will need to initialize the I2C module on
the DSP when code is loaded onto the EPROM.  This
function is not yet written.

****************************************************/
void i2c_init(void)
{
	uint16 prev_cl;			// Holds the previous code location and will return code_loc to it at the end of the
function
/* I2C module Configuration --- This must be done before PLL Configuration for running from I2C EEPROM */
	prev_cl = code_loc;
	code_loc = INIT_I2C;



	code_loc = prev_cl;
}


/****************************************************
void timer_init(void)
```

```
Initialize timer 0, but don't enable it.  Timer 0 is not
used yet on this DSP

****************************************************/
void timer_init(void)
{
	uint16 prev_cl;			// Holds the previous code location and will return code_loc to it at the end of the
function
/* Timer Configuration */
	prev_cl = code_loc;
	code_loc = INIT_TMR;

	TSSR = 0x0038;		// Timer Signal Selection Register (p.45)
						// Timer Pins as outputs
						// Disable Watchdog Timer
	GPTEMU_0 = 0x0000;		// Timer 0 stops immediately when emulation suspend event occurs
	GPTCLK_0 = 0x0000;		// Use Fast Peripherals Clock as clock source for timer
	GPTGPINT_0 = 0x0000;		// Don't use the Timer input pin as interrupt source
	GPTGPEN_0 = 0x0000;		// Don't use TIN1 and TOUT1 as GPIO

	GPTPRD1_0 = 0x0002;		// Set the timer output to oscillate at 25 MHz
	GPTPRD2_0 = 0x0000;
	GPTPRD3_0 = 0x0000;
	GPTPRD4_0 = 0x0000;

	GPTGCTL1_0 = 0x0003;		// Timer 0 in 64-bit mode, both 32-bit timers not in reset
	GPTGCTL1_1 = 0x0003;		// Timer 1 in 64-bit mode, both 32-bit timers not in reset
	WDTGCTL1 = 0x0003;		// WDT in 64-bit mode, both 32-bit timers not in reset

	WDTWCTL1 = 0x8000;		// Disable watchdog timer

	GPTCTL1_0 = 0x0008;		// Timer 0 configured for clock mode, but not enabled yet (0x0088 to enable)
	GPTCTL1_1 = 0x0000; // Timer 1 disabled
	WDTGCTL1 = 0x0000;		// WDT disabled

	code_loc = prev_cl;
}


/****************************************************
void clock_init(void)

Initialize the internal clocks of the DSP.  CPU clock
set to 300 MHz, SYSCLK1 to 150 MHz, SYSCLK2 to 75 MHz,
and SYSCLK3 to 150 MHz

****************************************************/
void clock_init(void)
{
	uint16 i;
	uint16 prev_cl;			// Holds the previous code location and will return code_loc to it at the end of the
function

	prev_cl = code_loc;
	code_loc = INIT_CLK;

/* Clock configuration */
	/* Configure PLL Control Register (p.57)
	1. Switch to bypass mode by setting the PLLEN bit to 0.
	2. Set the PLL to its reset state by setting the PLLRST bit to 1.
	3. Change the PLL setting through the PLLM and PLLDIV0 bits.
	4. Wait for 1 microsec.
	5. Release the PLL from its reset state by setting PLLRST to 0.
	6. Wait for the PLL to relock by polling the LOCK bit or by setting up a LOCK interrupt
	7. Switch back to PLL mode by setting the PLLEN bit to 1.
	*/

	PLLCSR &= ~bmPLLEN;		// 1. Switch to bypass mode by setting the PLLEN bit to 0 (p.59)
	PLLCSR |= bmPLLRST;		// 2. Set the PLL to its reset state by setting the PLLRST bit to 1 (p.59)
```

```
        PLLDIV0 = 0x8000;  // 3. Set to divide by 1 (20 MHz -> 20 MHz) (p.61)
        PLLM = 0x000F;              // 3. Set to multiply by 15 (20 MHz -> 300 MHz) (p.60)
        for (i=0; i<10; i++);   // 4. Wait for 1 microsec (Clock at 20 MHz, period of 0.1 usec)
        PLLCSR &= ~bmPLLRST;        // 5. Release the PLL from its reset state by setting PLLRST to 0 (p.59)
        while (PLLCSR & bmLOCK);    // 6. Wait for the PLL to relock by polling the LOCK bit (p.59)
        PLLCSR |= bmPLLEN;          // 7. Switch back to PLL mode by setting the PLLEN bit to 1 (p.59)

        CK3SEL = 0x000B;   // Select the PLL output as the source for CLKOUT3 (System Clock) (p.67)
        PLLDIV1 = 0x8001;  // Set SYSCLK1 (Fast Peripherals Clock Group) to divide by 2 (300 MHz -> 150 MHz) (p.62)
        PLLDIV2 = 0x8003;  // Set SYSCLK2 (Slow Peripherals Clock Group) to divide by 4 (300 MHz -> 75 MHz) (p.63)
        PLLDIV3 = 0x8001;  // Set SYSCLK3 (External Memory Interface Group) to divide by 2 (300 MHz -> 150 MHz) (p.64)
        OSCDIV1 = 0x0000;  // Disable OD1 divider for CLKOUT3.  This is not part of PLL path (p.64)

        WKEN = 0x001F;     // Disable wakeup from all external interrupts (p.66)
        CLKOUTSR = 0x0003;          // Disable the CLKOUT pin (goes to logic 0) (p.68)


        code_loc = prev_cl;
}


/****************************************************
void int_init(void)

Initialize the interrupt masks as needed.  This function
does not globally enable interrupts

****************************************************/
void int_init(void)
{
        uint16 prev_cl;                 // Holds the previous code location and will return code_loc to it at the end of the
function
/* Interrupts --------- Different between Transmitter and Receiver DSP (need INT1,INT2,INT3 in Receiver)*/
        prev_cl = code_loc;
        code_loc = INIT_INT;

        int_disable(INT_GLOBAL);

        IVPD = 0x00FF;
        IVPH = 0x00FF;

        IER0 = 0x1808;
                /* Enabled:
                        RINT2/UART - McBSP Recieve or UART Interrupt     Bit 12
                        INT3/WDTINT - External Interrupt #3                   Bit 11
                        INT2 - External Interrupt #2                          Bit 3
                -- Not Enabled, but maybe later
                        XINT2 - McBSP 2 Transmit Interrupt                    Bit 13
                        XINT1 - McBSP 1 Transmit Interrupt                    Bit 7
                        RINT1 - McBSP 0 Receive Interrupt                     Bit 6
                        RINT0 - McBSP 0 Receive Interrupt                     Bit 5
                        INT0 - External Interrupt #0                          Bit 2
                                INT0 is disabled until after DDCs are initialized
                */
        IER1 = 0x0001;
                /* Enabled:
                        INT1 - External Interrupt #1                          Bit 0
                -- Not Enabled, but maybe later
                        XINT0 - McBSP 0 Transmit Interrupt                    Bit 1
                        I2C - I2C Interrupt
        Bit 7
                */
        /* Clear all pending interrupts */
        IFR0 = 0xFFFF;
        IFR1 = 0xFFFF;

        code_loc = prev_cl;
}
```

```
/*****************************************************
void spi0_init(void)

SPI0 Configuration (connected to ADF4153 Frequency Synthesizer)
                    24-bit words
                    clocked on raising edge of SCLK
                    SCLK is 20 MHz maximum

*****************************************************/
void spi0_init(void)
{
        uint16 i;
        uint16 prev_cl;                 // Holds the previous code location and will return code_loc to it at the end of the
function

        prev_cl = code_loc;
        code_loc = INIT_SPI0;

        SPCR1_0 &= ~0x0001;             // Reset the Receiver
        SPCR2_0 &= ~0x00C1;             // Reset the Transmitter, Sample Rate Generator and frame sync logic


        SPCR1_0 = 0x1800;   // Set Clock Stop Mode to 1/2 cycle delay to center data on raising edge of SCLK
//      SPCR1_0 = 0x9800;   // Digital Loopback Enabled; Set Clock Stop Mode to 1/2 cycle delay to center data on raising edge
of SCLK

        SPCR2_0 = 0x0100;   // Set up McBSP0 for soft stop on emulator halt
        RCR1_0 = 0x0080;    // Set up to receive words of 24 bits long
        RCR2_0 = 0x0081;    // Set up a 1 bit delay on the receive data
        XCR1_0 = 0x0080;    // Set up to transmit words of 24 bits long
        XCR2_0 = 0x0081;    // Set up a 1 bit delay on the transmit data
        SRGR1_0 = 0x1700+SPI0_SCLK_DIV-1;
                                        // Set Frame Sync to 24 cycles wide (value=23) and diveder for
CLKG to 38 (value=37)
                                        // to bring SCLK down to approx 1 MHz (37.5 MHz / 38 = 1 MHz)
        SRGR2_0 = 0x2000;   // McBSP uses internal input clock (used in conjunction with SCLKME bit)
                                        // Sets up transmitter to drive FSX pin every time data is transfered
        MCR1_0 = 0x0000;
        MCR2_0 = 0x0000;
        PCR0 = 0x0F0C;                  // Configure Port Control Register (p.12-38 of SPRU592e data sheet)
        RCERA_0 = 0x0000;
        RCERB_0 = 0x0000;
        XCERA_0 = 0x0000;
        XCERB_0 = 0x0000;
        RCERC_0 = 0x0000;
        RCERD_0 = 0x0000;
        XCERC_0 = 0x0000;
        XCERD_0 = 0x0000;
        RCERE_0 = 0x0000;
        RCERF_0 = 0x0000;
        XCERE_0 = 0x0000;
        XCERF_0 = 0x0000;
        RCERG_0 = 0x0000;
        RCERH_0 = 0x0000;
        XCERG_0 = 0x0000;
        XCERH_0 = 0x0000;

        SPCR2_0 |= 0x0040;  // Enable the Sample Rate Generator
        SPCR2_0 |= 0x0080;  // Enable Frame Sync Logic
        for (i=0; i<3*SPI0_SCLK_DIV; i++);      // Wait at least 2 sample rate generator clock periods for logic to stabalize

        SPCR1_0 |= 0x0001;  // Enable the Receiver
        SPCR2_0 |= 0x0001;  // Enable the Transmitter

        code_loc = prev_cl;
}


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

```
void spi1_init(void)

SPI1 Configuration (connected to AD5060 DAC controlling the VGA)
                        24-bit words
                        clocked on falling edge of SCLK
                        SCLK is 30 MHz maximum

****************************************************/
void spi1_init(void)
{
            uint16 i;
            uint16 prev_cl;                     // Holds the previous code location and will return code_loc to it at the end of the
function

            prev_cl = code_loc;
            code_loc = INIT_SPI1;

            SPCR1_1 &= ~0x0001;             // Reset the Receiver
            SPCR2_1 &= ~0x00C1;             // Reset the Transmitter, Sample Rate Generator and frame sync logic


            SPCR1_1 = 0x1000;   // Set Clock Stop Mode to 1/2 cycle delay to center data on raising edge of SCLK
//          SPCR1_1 = 0x9000;   // Digital Loopback Enabled; Set Clock Stop Mode to 1/2 cycle delay to center data on raising edge
of SCLK

            SPCR2_1 = 0x0100;   // Set up McBSP1 for soft stop on emulator halt
            RCR1_1 = 0x0080;    // Set up to receive words of 24 bits long
            RCR2_1 = 0x0081;    // Set up a 1 bit delay on the receive data
            XCR1_1 = 0x0080;    // Set up to transmit words of 24 bits long
            XCR2_1 = 0x0081;    // Set up a 1 bit delay on the transmit data
            SRGR1_1 = 0x1700+SPI1_SCLK_DIV-1;
                                                        // Set Frame Sync to 24 cycles wide (value=23) and diveder for
CLKG to 38 (value=37)
                                                        // to bring SCLK down to approx 1 MHz (37.5 MHz / 38 = 1 MHz)
            SRGR2_1 = 0x2000;   // McBSP uses internal input clock (used in conjunction with SCLKME bit)
                                                        // Sets up transmitter to drive FSX pin every time data is transfered
            MCR1_1 = 0x0000;
            MCR2_1 = 0x0000;
            PCR1 = 0x0F0C;                      // Configure Port Control Register (p.12-38 of SPRU592e data sheet)
            RCERA_1 = 0x0000;
            RCERB_1 = 0x0000;
            XCERA_1 = 0x0000;
            XCERB_1 = 0x0000;
            RCERC_1 = 0x0000;
            RCERD_1 = 0x0000;
            XCERC_1 = 0x0000;
            XCERD_1 = 0x0000;
            RCERE_1 = 0x0000;
            RCERF_1 = 0x0000;
            XCERE_1 = 0x0000;
            XCERF_1 = 0x0000;
            RCERG_1 = 0x0000;
            RCERH_1 = 0x0000;
            XCERG_1 = 0x0000;
            XCERH_1 = 0x0000;

            SPCR2_1 |= 0x0040;  // Enable the Sample Rate Generator
            SPCR2_1 |= 0x0080;  // Enable Frame Sync Logic
            for (i=0; i<3*SPI1_SCLK_DIV; i++);          // Wait at least 2 sample rate generator clock periods for logic to stabalize

            SPCR1_1 |= 0x0001;  // Enable the Receiver
            SPCR2_1 |= 0x0001;  // Enable the Transmitter

            code_loc = prev_cl;
}


/****************************************************
void hpi_init(void)
```

```
Initialize the HPI ports to be used as GPIO

****************************************************/
void hpi_init(void)
{
        uint16 prev_cl;                 // Holds the previous code location and will return code_loc to it at the end of the
function
/* HPI Configuration ---------Different between transmitter and receiver */
        prev_cl = code_loc;
        code_loc = INIT_HPI;

        HGPIOEN = 0x19D7;// Enable all HPI pins and GPIO
        HGPIODIR1 = 0xFFFF;             // Enable HD15:0 as Outputs
        HGPIODAT1 = 0x0000;             // Push HD15:0 to low (floating connections)
        HGPIODIR2 = 0x00B3;             // Enable HPI pins as Inputs or Outputs as needed
        HGPIODAT2 = 0x0000;             // Push floating connections low and disable receiver
        HGPIODIR3 = 0xFFFF;             // Enable HA15:0 as GPIO
        HGPIODAT3 = 0x0000;             // Push HA15:0 to low (floating connections)

        code_loc = prev_cl;
}


/****************************************************
void gpio_init(void)

Initialize the general purpose inputs as inputs (they are
not used) and the parallel GPIO as inputs for PGPIO0/1 as
the data inputs from the FPGA and PGPIO2 as outputs to the
DDCs

****************************************************/
void gpio_init(void)
{
        uint16 prev_cl;                 // Holds the previous code location and will return code_loc to it at the end of the
function
/* GPIO Configuration ---------Different between transmitter and receiver */
        prev_cl = code_loc;
        code_loc = INIT_GPIO;

        IODIR = 0x0028;     // Enable GPIO as inputs, except bits 3 and 5
        IODATA |= 0x0028;   // Set GPIO3 and GPIO5 to high

        /*
        Use PGPIO as Outputs: PGPIOENx = PGPIODIRx = 1
        Use PGPIO as Inputs: PGPIOENx = PGPIODIRx = 0
        */
        PGPIOEN0 = 0x0000;              // Enable PGPIO0
        PGPIODIR0 = 0x0000;             // Set PGPIO0 as Inputs

        PGPIOEN1 = 0x0000;              // Enable PGPIO1
        PGPIODIR1 = 0x0000;             // Set PGPIO1 as Inputs

        PGPIOEN2 = 0x3FFF;              // Enable PGPIO2
        PGPIODIR2 = 0x3FFF;             // Set PGPIO2 as Outputs
        PGPIODAT2 = 0x0000;             // Set PGPIO2 low


        code_loc = prev_cl;
}


/****************************************************
void uart_init(void)

Initialize the UART using the baud rate divisor defined
in dspr_init.h

****************************************************/
void uart_init(void)
```

```
{
        uint16 prev_cl;                 // Holds the previous code location and will return code_loc to it at the end of the
function
/* UART Configuration -------Different Between transmitter and Receiver*/
        prev_cl = code_loc;
        code_loc = INIT_UART;

        URDLM = (UART_BAUD_DIV & 0xFF00) >> 8;
        URDLL = (UART_BAUD_DIV & 0x00FF);           // Divisor = 82 to make baud rate 57165 (75 MHz/(16*41))
        URLCR = 0x0003;     // Configure the Parity (no parity), stop bit (1 stop bit) and word length (8 bits)
        URMCR = 0x0000;                 // Disable Loop back for UART debugging
        URIER = 0x0003;                 // Enable interrupts for transmit buffer empty and recieve data ready
        URPECR = 0x8000;   // Enable the UART trasnmitter and receiver

        code_loc = prev_cl;
}
```

File: dspr_init.h

```
#ifndef DSPT_INIT_H
#define DSPT_INIT_H

// PLLCSR Register bit masks
#define bmSTABLE   0x0040
#define bmLOCK           0x0020
#define bmPLLRST   0x0008
#define bmOSCPWRDN       0x0004
#define bmPLLPWRDN       0x0002
#define bmPLLEN          0x0001

void dspr_init(void);
void timer_init(void);
void clock_init(void);
void int_init(void);
void spi0_init(void);
void spi1_init(void);
void hpi_init(void);
void gpio_init(void);
void uart_init(void);
void i2c_init(void);

#endif
```

APPENDIX E.
TRANSMITTER DIGITAL SIGNAL PROCESSOR CODE

File: dspt_main.c

```
#include "dspt_init.h"
#include "data_types.h"
#include "dspt_main.h"
#include "c5502regs.h"


/*-------------- Global Variables -----------------------*/
/* For testing purposes, ibuf is initialized to a 512 length array of evenly
spaced points on a cosine curve and qbuf is initialized to a 512 length of
evenly spaced points on a sine curve.  Once outputed and upconverted, this should
output a sine wave at:

fo = fc + (fs / 512)

where:

fo = output frequency
fc = carrier frequency
fs = DAC (sampling) frequency
 */
uint16 ibuf[BUFSIZE] =
{65535,65534,65526,65514,65497,65474,65447,65415,65378,65336,65290,65238,65181,65120,65054,64982,64906,64826,64740,64
649,64554,64454,64349,64239,64125,64006,63882,63754,63621,63483,63340,63193,63042,62886,62725,62560,62390,62216,62037,
61854,61667,61475,61279,61079,60874,60665,60452,60235,60014,59788,59559,59325,59088,58846,58601,58351,58098,57841,575
80,57316,57047,56776,56500,56221,55938,55652,55363,55070,54774,54474,54171,53865,53556,53243,52928,52609,52288,51963,5
1636,51306,50973,50637,50299,49958,49614,49268,48919,48568,48215,47859,47501,47141,46778,46414,46047,45678,45308,4493
5,44561,44185,43807,43428,43047,42664,42280,41895,41508,41119,40730,40339,39948,39555,39161,38766,38370,37973,37576,37
178,36779,36380,35980,35579,35179,34777,34376,33974,33572,33170,32768,32366,31964,31562,31160,30759,30357,29957,29556,
29156,28757,28358,27960,27563,27166,26770,26375,25981,25588,25197,24806,24417,24028,23641,23256,22872,22489,22108,217
29,21351,20975,20601,20228,19858,19489,19122,18758,18395,18035,17677,17321,16968,16617,16268,15922,15578,15237,14899,1
4563,14230,13900,13573,13248,12927,12608,12293,11980,11671,11365,11062,10762,10466,10173,9883.6,9597.5,9314.9,9035.9,87
60.4,8488.5,8220.3,7955.8,7695,7438,7184.8,6935.5,6690,6448.5,6210.9,5977.4,5747.9,5522.4,5301.1,5083.8,4870.8,4661.9,4457.3,
4257,4060.9,3869.2,3681.8,3498.8,3320.2,3146.1,2976.4,2811.2,2650.5,2494.3,2342.7,2195.7,2053.3,1915.5,1782.3,1653.9,1530.1,1
411,1296.6,1187,1082.1,982.02,886.7,796.19,710.5,629.63,553.6,482.42,416.11,354.66,298.1,246.43,199.66,157.79,120.83,88.786,61
.666,39.47,22.204,9.8691,2.4674,0,2.4674,9.8691,22.204,39.47,61.666,88.786,120.83,157.79,199.66,246.43,298.1,354.66,416.11,482.
42,553.6,629.63,710.5,796.19,886.7,982.02,1082.1,1187,1296.6,1411,1530.1,1653.9,1782.3,1915.5,2053.3,2195.7,2342.7,2494.3,265
0.5,2811.2,2976.4,3146.1,3320.2,3498.8,3681.8,3869.2,4060.9,4257,4457.3,4661.9,4870.8,5083.8,5301.1,5522.4,5747.9,5977.4,6210.
9,6448.5,6690,6935.5,7184.8,7438,7695,7955.8,8220.3,8488.5,8760.4,9035.9,9314.9,9597.5,9883.6,10173,10466,10762,11062,11365
,11671,11980,12293,12608,12927,13248,13573,13900,14230,14563,14899,15237,15578,15922,16268,16617,16968,17321,17677,180
35,18395,18758,19122,19489,19858,20228,20601,20975,21351,21729,22108,22489,22872,23256,23641,24028,24417,24806,25197,2
5588,25981,26375,26770,27166,27563,27960,28358,28757,29156,29556,29957,30357,30759,31160,31562,31964,32366,32768,3317
0,33572,33974,34376,34777,35179,35579,35980,36380,36779,37178,37576,37973,38370,38766,39161,39555,39948,40339,40730,41
119,41508,41895,42280,42664,43047,43428,43807,44185,44561,44935,45308,45678,46047,46414,46778,47141,47501,47859,48215,
48568,48919,49268,49614,49958,50299,50637,50973,51306,51636,51963,52288,52609,52928,53243,53556,53865,54171,54474,547
74,55070,55363,55652,55938,56221,56500,56776,57047,57316,57580,57841,58098,58351,58601,58846,59088,59325,59559,59788,6
0014,60235,60452,60665,60874,61079,61279,61475,61667,61854,62037,62216,62390,62560,62725,62886,63042,63193,63340,6348
3,63621,63754,63882,64006,64125,64239,64349,64454,64554,64649,64740,64826,64906,64982,65054,65120,65181,65238,65290,65
336,65378,65415,65447,65474,65497,65514,65526,65534};
uint16 qbuf[BUFSIZE] =
{32768,33170,33572,33974,34376,34777,35179,35579,35980,36380,36779,37178,37576,37973,38370,38766,39161,39555,39948,40
339,40730,41119,41508,41895,42280,42664,43047,43428,43807,44185,44561,44935,45308,45678,46047,46414,46778,47141,47501,
47859,48215,48568,48919,49268,49614,49958,50299,50637,50973,51306,51636,51963,52288,52609,52928,53243,53556,53865,541
71,54474,54774,55070,55363,55652,55938,56221,56500,56776,57047,57316,57580,57841,58098,58351,58601,58846,59088,59325,5
9559,59788,60014,60235,60452,60665,60874,61079,61279,61475,61667,61854,62037,62216,62390,62560,62725,62886,63042,6319
3,63340,63483,63621,63754,63882,64006,64125,64239,64349,64454,64554,64649,64740,64826,64906,64982,65054,65120,65181,65
238,65290,65336,65378,65415,65447,65474,65497,65514,65526,65534,65535,65534,65526,65514,65497,65474,65447,65415,65378,
65336,65290,65238,65181,65120,65054,64982,64906,64826,64740,64649,64554,64454,64349,64239,64125,64006,63882,63754,636
21,63483,63340,63193,63042,62886,62725,62560,62390,62216,62037,61854,61667,61475,61279,61079,60874,60665,60452,60235,6
0014,59788,59559,59325,59088,58846,58601,58351,58098,57841,57580,57316,57047,56776,56500,56221,55938,55652,55363,5507
0,54774,54474,54171,53865,53556,53243,52928,52609,52288,51963,51636,51306,50973,50637,50299,49958,49614,49268,48919,48
568,48215,47859,47501,47141,46778,46414,46047,45678,45308,44935,44561,44185,43807,43428,43047,42664,42280,41895,41508,
41119,40730,40339,39948,39555,39161,38766,38370,37973,37576,37178,36779,36380,35980,35579,35179,34777,34376,33974,335
72,33170,32768,32366,31964,31562,31160,30759,30357,29957,29556,29156,28757,28358,27960,27563,27166,26770,26375,25981,2
5588,25197,24806,24417,24028,23641,23256,22872,22489,22108,21729,21351,20975,20601,20228,19858,19489,19122,18758,1839
5,18035,17677,17321,16968,16617,16268,15922,15578,15237,14899,14563,14230,13900,13573,13248,12927,12608,12293,11980,11
671,11365,11062,10762,10466,10173,9883.6,9597.5,9314.9,9035.9,8760.4,8488.5,8220.3,7955.8,7695,7438,7184.8,6935.5,6690,644
8.5,6210.9,5977.4,5747.9,5522.4,5301.1,5083.8,4870.8,4661.9,4457.3,4257,4060.9,3869.2,3681.8,3498.8,3320.2,3146.1,2976.4,2811.
2,2650.5,2494.3,2342.7,2195.7,2053.3,1915.5,1782.3,1653.9,1530.1,1411,1296.6,1187,1082.1,982.02,886.7,796.19,710.5,629.63,553.
```

```
6,482.42,416.11,354.66,298.1,246.43,199.66,157.79,120.83,88.786,61.666,39.47,22.204,9.8691,2.4674,0,2.4674,9.8691,22.204,39.47,
61.666,88.786,120.83,157.79,199.66,246.43,298.1,354.66,416.11,482.42,553.6,629.63,710.5,796.19,886.7,982.02,1082.1,1187,1296.6
,1411,1530.1,1653.9,1782.3,1915.5,2053.3,2195.7,2342.7,2494.3,2650.5,2811.2,2976.4,3146.1,3320.2,3498.8,3681.8,3869.2,4060.9,4
257,4457.3,4661.9,4870.8,5083.8,5301.1,5522.4,5747.9,5977.4,6210.9,6448.5,6690,6935.5,7184.8,7438,7695,7955.8,8220.3,8488.5,8
760.4,9035.9,9314.9,9597.5,9883.6,10173,10466,10762,11062,11365,11671,11980,12293,12608,12927,13248,13573,13900,14230,14
563,14899,15237,15578,15922,16268,16617,16968,17321,17677,18035,18395,18758,19122,19489,19858,20228,20601,20975,21351,
21729,22108,22489,22872,23256,23641,24028,24417,24806,25197,25588,25981,26375,26770,27166,27563,27960,28358,28757,291
56,29556,29957,30357,30759,31160,31562,31964,32366};
uint16 start;
uint16 end;
uint16 current_i;
uint16 current_q;

uint16 rxbuf[UART_BUFSIZE];          // UART Receive Buffer
uint16 rxstart;
uint16 rxend;
uint16 rxbusy;                               // 1 if buffer empty or in process of filling
                                                   // 0 if new data in buffer

uint16 txbuf[UART_BUFSIZE];          // UART Transmit Buffer
uint16 txstart;
uint16 txend;
uint16 txbusy;                               // 1 if new data ready to be transmitted or data still transmitting
                                                   // 0 if no new data to send

uint32 fs;                                   // Current sampling frequency expressed in Hz
uint32 fc;                                   // Current Carrier Frequency expressed in kHz
uint16 mod;                                       // Current modulation - Not currently used

uint16 code_loc;
uint16 check_ISR[NUM_ISRS];
/* ----------- END GLOBAL VARIABLES ----------------- */

void main(void)
{
        uint16 i,j;

        code_loc = MAIN;

        for (i=0; i<NUM_ISRS; i++)
        {
                check_ISR[i]=0;
        }

        start = 0;
        end = 512;

        rxstart = 0;
        rxend = 0;
        rxbusy = 0;

        txstart = 0;
        txend = 0;
        txbusy = 0;


        dspt_init();
        int_enable(INT_GLOBAL);
        freqsyn_init();
        vga_gain_init();

        DAC_clock_disable();

        fs = set_DAC_clock(600000);     // sets DAC clock to 600 kHz

        j = set_carrier(1000000);               // set carrier frequency to 1.0 GHz
        j = set_carrier(1200000);               // set carrier frequency to 1.2 GHz
        j = set_carrier(1400000);               // set carrier frequency to 1.4 GHz
        j = set_carrier(1600000);               // set carrier frequency to 1.6 GHz
        j = set_carrier(1800000);               // set carrier frequency to 1.8 GHz
```

```
	j = set_carrier(2000000);				// set carrier frequency to 2.0 GHz


	j = set_carrier(1600000);

//	send_lcd("DSPT Initialized", 16);
	i = 0;
	j = 0;
	DAC_clock_enable();
	while (1)
	{
		if (rxbusy)
		{
			handle_rxdata();
		}
		if (end != (start-1) % BUFSIZE)
		{
			end = (++end) % BUFSIZE;
		}

/*		if (check_ISR[NOISR12] >= 18)
		{
			i++;
			if (i>1000)
			{
				j++;
				if (j>1000)
				{
					check_ISR[NOISR12] = 0;
					i = 0;
					j = 0;
					send_lcd("DSPT Initialized", 16);
				}
				else
				{
					i = 0;
				}
			}
		}
*/
	}

}


/***************************************************
void freqsyn_init(void)

This function initializes the frequency synthesizer based
on the initialization perameters defiened in dspr_main.h
***************************************************/
void freqsyn_init(void)
{
	uint16 prev_cl;
	uint32 value;
	prev_cl = code_loc;
	code_loc = INIT_FQSYN;
	/*
	Configures Frequency Synthesizer to start with a 1.000000 GHz output

	F_PFD = RF_IN * (1+D)/R

	RF_OUT = F_PFD * (INT + (FRAC/MOD))

	RF_IN = 20 MHz			(Input Reference Clock)
	D = 0				(Reference Doubler)
	R = 5				(R Counter)
	gives F_PFD = 4 MHz

	INT = 250
```

```
		FRAC = 0
		MOD = 4000					(gives resolution of 1 kHz)
		gives RF_OUT = 1.000000 GHz

		*/
		TX_DISABLE
		TX_ENABLE

		value = (REG_C_VAL) | 0x18;	// Power-Down mode, and three-state CP
		send_freqsyn(value);
		for (value = 0; value<30000000; value++);	// wait a second to let the freq syn reset
		send_freqsyn(0x0003);			// Write all zeros to noise and spur register
		value = REG_NS_VAL;
		send_freqsyn(value);	// Configure Noise and Spur Register
		value = (REG_C_VAL) | 0x0014;			// Keep in Power-Down mode
		send_freqsyn(value);	// Put Counters in reset and congifure control register
		value = REG_R_VAL;
		send_freqsyn(value);	// Configure R Divider Register
		value = REG_N_VAL;
		send_freqsyn(value);	// Configure N Divider Register
		value = (REG_C_VAL) & ~0x0004;
		send_freqsyn(value);	// Enable Counters and Power Up

		code_loc = prev_cl;
}


/****************************************************
uint16 set_carrier(uint32 freq)

Sets the carrier frequency of the receiver

Input: desired frequency in kHz
Return: 0 if unsuccessful
			1 if successful

****************************************************/
uint16 set_carrier(uint32 freq)
{
		// vary the carrier by changind the INT and FRAC values in the N Divider Register
		// 250 <= INT <= 500 for 1.0 GHz - 2.0 GHz operation
		// each increment of INT changes output by 4 MHz
		// each increment of FRAC changes output by 1 kHz

		uint32 n,frac,reg_val;
		uint16 prev_cl;
		uint32 stuck = 0;
		uint16 success = 1;
		prev_cl = code_loc;

		code_loc = SET_CARR;

		if ((freq < FS_FREQ_MIN) || (freq > FS_FREQ_MAX))		// if outside 1-2GHz range, return without changes
		{
			code_loc = prev_cl;
			return 0;
		}
		frac = freq % FS_FPFD;				// calculate frac value
		n = (freq - frac) / FS_FPFD;		// calculate int value
		reg_val = (((uint32)FASTLOCK & 0x1)<<23)+(n << 14) + (frac << 2);		// build value to write to N Divider Register
		send_freqsyn(reg_val);				// send value to frequency synthesizer
		while ((SPCR2_0 & 0x0004) == 0x0004);	// wait for transmission to complete
		for (stuck=0; stuck<3000; stuck++);			// wait for 10 usec (min time to lock detect)
		stuck = 0;
		while (FS_MUXOUT == 0)				// Wait for PLL Lock
		{
			stuck++;
			if (stuck > 500000)
			{
				success = freqsyn_reinit(reg_val);
```

```
                    code_loc = prev_cl;
                    return success;
                }
        }

        fc = freq;
        code_loc = prev_cl;
        return success;
}


/****************************************************
uint16 freqsyn_reinit(uint32 reg_val)

Reinitialize the frequency synthesizer if it won't lock

Input: register value for the N divider Register
Ouptut: 0 if unsuccessful
                1 if successful

****************************************************/
uint16 freqsyn_reinit(uint32 reg_val)
{
        uint16 prev_cl;
        uint32 value;
        uint16 success = 1;
        uint32 stuck=0;

        prev_cl = code_loc;
        code_loc = INIT_FQSYN;

        value = (REG_C_VAL) | 0x18;    // Power-Down mode, and three-state CP
        send_freqsyn(value);
        for (value = 0; value<3000000; value++);     // wait a second to let the freq syn and VCO react
        send_freqsyn(0x0003);            // Write all zeros to noise and spur register
        value = REG_NS_VAL;
        send_freqsyn(value);  // Configure Noise and Spur Register
        value = (REG_C_VAL) | 0x0014;            // Keep in Power-Down mode
        send_freqsyn(value);  // Put Counters in reset and congifure control register
        value = REG_R_VAL;
        send_freqsyn(value);  // Configure R Divider Register

        send_freqsyn(reg_val);           // Configure N Divider Register
        value = (REG_C_VAL) & ~0x0014;
        send_freqsyn(value);  // Enable Counters and Power Up

        while (FS_MUXOUT == 0)                          // Wait for PLL Lock
        {
                stuck++;
                if (stuck > 30000)
                {
                    success = 0;
                    break;
                }
        }

        code_loc = prev_cl;
        return success;
}


/****************************************************
void send_freqsyn(uint32 data)

Send data to frequency synthesizer via SPI0

Input: 24-bit data to be sent

****************************************************/
void send_freqsyn(uint32 data)
```

```
{
        uint16 prev_cl;
        prev_cl = code_loc;
        code_loc = SEND_FQSYN;

        data = data & 0x00FFFFFF;       // Clear upper 8 bits
        while ((SPCR2_0 & bmXEMPTY) == bmXEMPTY);   // wait for transmit buffer to empty
        DXR2_0 = (uint16) ((data & 0x00FF0000) >> 16);      // write upper word to upper transmit register
        DXR1_0 = (uint16) (data & 0x0000FFFF);   // write lower word to transmit register
                                                                                        // this
initiates SPI transfer
        while ((SPCR2_0 & bmXEMPTY) == bmXEMPTY);   // wait for transmit buffer to empty

        code_loc = prev_cl;
}


/****************************************************
void vga_gain_init(void)

Initializes the gain on the VGA to approx. 0 dB

****************************************************/
void vga_gain_init(void)
{
        uint16 prev_cl;
        prev_cl = code_loc;
        code_loc = INIT_VGA;

        // initializes gain to approximately 0 dB
        set_vga_gain(0x638D);           // corresponds to a voltage of about 0.7 V
        code_loc = prev_cl;
}


/****************************************************
void set_vga_gain(uint16 voltage)

Set the gain on the VGA to the desired value

Input: 16-bit word.  0x0000 = 0 V
                                0xFFFF = 1.8 V

****************************************************/
void set_vga_gain(uint16 voltage)
{
        uint16 prev_cl;
        prev_cl = code_loc;
        code_loc = SET_VGA;

        if ((voltage < VGA_GAIN_MIN) || (voltage > VGA_GAIN_MAX))          // if outside valid range, do nothing
        {
                code_loc = prev_cl;
                return;
        }

        while ((SPCR2_1 & bmXEMPTY) == bmXEMPTY);   // wait for transmission to complete
        DXR2_1 = 0x0000;                            // bits 17:16 set mode
                                                        // 00: normal operation
                                                        // 01: 3-state
                                                        // 10: 100 kOhm to ground
                                                        // 11: 1 kOhm to ground
        DXR1_1 = voltage;                           // write lower word to transmit register
                                                        // this initiates SPI transfer
        while ((SPCR2_1 & bmXEMPTY) == bmXEMPTY);   // wait for transmission to complete
        code_loc = prev_cl;
}


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

```
void send_lcd(char * string, uint16 length)

Send a character string to the microcontroller to be
displayed on the LCD display

Input: character array pointer - can be passed as: "message"
        length: length of the character array

****************************************************/
void send_lcd(char * string, uint16 length)    // length must be <= (UART_BUFSIZE-2)
{
        uint16 prev_cl;
        prev_cl = code_loc;
        code_loc = SEND_MSG;

        while (txbusy);                 // Wait for transmitter to become available
        txstart = 0;
        txbuf[0] = SEND_LCD;
        txbuf[1] = length + 2; // add for the control byte and the length byte
        if (length > UART_BUFSIZE - 2)          // if length too long, cut off end of string
        {
                length = UART_BUFSIZE - 2;
        }
        for (txend=2; txend<length+2; txend++)
        {
                txbuf[txend] = (uint16) string[txend-2];        // Copy string to txbuf
        }
        txend = length+2;
        txbusy = 1;
        if (URLSR & bmTHRE)             // Transmit Hold Register Empty
        {
                URTHR = txbuf[0] & 0x00FF;              // Initiate UART Transfer - For some reason, cannot write to IFR0
        }
        code_loc = prev_cl;
}


/****************************************************
void int_enable(uint16 int_num)

Enable the given interrupt.  Interrupts are defined in dspr_main.h

****************************************************/
void int_enable(uint16 int_num)
{
        uint16 prev_cl;
        prev_cl = code_loc;
        code_loc = INT_ENBL;

        switch (int_num)
        {
                case INT_INT0:
                        IER0 |= bmINT_INT0;
                        break;
                case INT_INT2:
                        IER0 |= bmINT_INT2;
                        break;
                case INT_TINT0:
                        IER0 |= bmINT_TINT0;
                        break;
                case INT_RINT0:
                        IER0 |= bmINT_RINT0;
                        break;
                case INT_RINT1:
                        IER0 |= bmINT_RINT1;
                        break;
                case INT_XINT1:
                        IER0 |= bmINT_XINT1;
                        break;
```

```
			case INT_DMAC1:
				IER0 |= bmINT_DMAC1;
				break;
			case INT_DSPINT:
				IER0 |= bmINT_DSPINT;
				break;
			case INT_INT3_WDTINT:
				IER0 |= bmINT_INT3_WDTINT;
				break;
			case INT_RINT2_UART:
				IER0 |= bmINT_RINT2_UART;
				break;
			case INT_XINT2:
				IER0 |= bmINT_XINT2;
				break;
			case INT_DMAC4:
				IER0 |= bmINT_DMAC4;
				break;
			case INT_DMAC5:
				IER0 |= bmINT_DMAC5;
				break;
			case INT_INT1:
				IER1 |= bmINT_INT1;
				break;
			case INT_XINT0:
				IER1 |= bmINT_XINT0;
				break;
			case INT_DMAC0:
				IER1 |= bmINT_DMAC0;
				break;
			case INT_DMAC2:
				IER1 |= bmINT_DMAC2;
				break;
			case INT_DMAC3:
				IER1 |= bmINT_DMAC3;
				break;
			case INT_TINT1:
				IER1 |= bmINT_TINT1;
				break;
			case INT_I2C:
				IER1 |= bmINT_I2C;
				break;
			case INT_BERR:
				IER1 |= bmINT_BERR;
				break;
			case INT_DLOG:
				IER1 |= bmINT_DLOG;
				break;
			case INT_RTOS:
				IER1 |= bmINT_RTOS;
				break;
			case INT_GLOBAL:
				// dummy = ST1_55;
				// dummy &= ~bmINT_GLOBAL;
				// ST1 = dummy;
				asm(" BCLR INTM ");
				break;
		}
		code_loc = prev_cl;
}


/*****************************************************
void int_disable(uint16 int_num)

Disable the given interrupt.  Interrupts are defined in dspr_main.h

*****************************************************/
void int_disable(uint16 int_num)
```

```
{
	uint16 prev_cl;
	prev_cl = code_loc;
	code_loc = INT_DABL;

	switch (int_num)
	{
		case INT_INT0:
			IER0 &= ~bmINT_INT0;
			break;
		case INT_INT2:
			IER0 &= ~bmINT_INT2;
			break;
		case INT_TINT0:
			IER0 &= ~bmINT_TINT0;
			break;
		case INT_RINT0:
			IER0 &= ~bmINT_RINT0;
			break;
		case INT_RINT1:
			IER0 &= ~bmINT_RINT1;
			break;
		case INT_XINT1:
			IER0 &= ~bmINT_XINT1;
			break;
		case INT_DMAC1:
			IER0 &= ~bmINT_DMAC1;
			break;
		case INT_DSPINT:
			IER0 &= ~bmINT_DSPINT;
			break;
		case INT_INT3_WDTINT:
			IER0 &= ~bmINT_INT3_WDTINT;
			break;
		case INT_RINT2_UART:
			IER0 &= ~bmINT_RINT2_UART;
			break;
		case INT_XINT2:
			IER0 &= ~bmINT_XINT2;
			break;
		case INT_DMAC4:
			IER0 &= ~bmINT_DMAC4;
			break;
		case INT_DMAC5:
			IER0 &= ~bmINT_DMAC5;
			break;
		case INT_INT1:
			IER1 &= ~bmINT_INT1;
			break;
		case INT_XINT0:
			IER1 &= ~bmINT_XINT0;
			break;
		case INT_DMAC0:
			IER1 &= ~bmINT_DMAC0;
			break;
		case INT_DMAC2:
			IER1 &= ~bmINT_DMAC2;
			break;
		case INT_DMAC3:
			IER1 &= ~bmINT_DMAC3;
			break;
		case INT_TINT1:
			IER1 &= ~bmINT_TINT1;
			break;
		case INT_I2C:
			IER1 &= ~bmINT_I2C;
			break;
		case INT_BERR:
			IER1 &= ~bmINT_BERR;
			break;
```

```
			case INT_DLOG:
				IER1 &= ~bmINT_DLOG;
				break;
			case INT_RTOS:
				IER1 &= ~bmINT_RTOS;
				break;
			case INT_GLOBAL:
				asm(" BSET INTM ");
				//ST1_55 |= bmINT_GLOBAL;
				break;
		}
		code_loc = prev_cl;
}


/****************************************************
interrupt void int0_ISR(void)

Interrupt Service Routine for external interrupt 0,
controlled by the output of timer 0 (TIM0).  Interrupt
signals when to output new data to the DACs

****************************************************/
interrupt void int0_ISR(void)
{
		uint16 prev_cl;
		prev_cl = code_loc;
		code_loc = INT0ISR;
		check_ISR[NOISR2] += 1;

		if (start != end)
		{
			DACI = ibuf[start] >> 2;		// 14-bit DAC connected to lower 14 bits of port
//			DACQ = 0x0000;
			DACQ = qbuf[start] >> 2;		// 14-bit DAC connected to lower 14 bits of port
			start = (++start) % BUFSIZE;	// increment start
		}
		code_loc = prev_cl;
}


/****************************************************
void uart_ISR(void)

Interrupt Service Routine for UART communication

Communication is done 8 bits at a time so buffers need to be
filled and read accordingly.

Format of Packet:
Byte 0: Control Byte
Byte 1: Total Packet Length (max length: 64)
Bytes 2...63: Data Bytes

****************************************************/
interrupt void uart_ISR(void)
{
		uint16 urlsr_holder;

		uint16 prev_cl;
		prev_cl = code_loc;
		code_loc = UARTISR;
		check_ISR[NOISR12] += 1;

		urlsr_holder = URLSR;

		if (urlsr_holder & bmTHRE)	// Transmit Hold Register Empty
		{
			if (txbusy) // if data still ready to be transmitted
```

```
			{
				URTHR = txbuf[txstart] & 0x00FF;		// Can only transmit lower 8 bits
				txstart = (++txstart) % UART_BUFSIZE;
				if (txstart == txend)	// All data has been transmitted
				{
					txstart = 0;
					txend = 0;
					txbusy = 0;
				}
			}
		}
		if (urlsr_holder & bmDR)		// Receive Data Ready
		{
			if (rxbusy)				// If buffer available
			{
				if (rxend < UART_BUFSIZE)	// Make sure of no buffer overflow
				{
					rxbuf[rxend] = URRBR;		// Read data
					rxend++;					// increment end
				}
				if ((rxend > 1) && (rxend >= rxbuf[1]))	// if received length byte and have received that many
bytes
				{
					rxstart = 0;
					rxbusy = 0;
				}
			}
		}
		code_loc = prev_cl;
}

/******************************************************
Interrupt Service Routines for all interrupts by number to
track if ISRs are called

******************************************************/
interrupt void no_ISR1(void) {check_ISR[NOISR1] += 1;}
interrupt void no_ISR2(void) {check_ISR[NOISR2] += 1;}
interrupt void no_ISR3(void) {check_ISR[NOISR3] += 1;}
interrupt void no_ISR4(void) {check_ISR[NOISR4] += 1;}
interrupt void no_ISR5(void) {check_ISR[NOISR5] += 1;}
interrupt void no_ISR6(void) {check_ISR[NOISR6] += 1;}
interrupt void no_ISR7(void) {check_ISR[NOISR7] += 1;}
interrupt void no_ISR8(void) {check_ISR[NOISR8] += 1;}
interrupt void no_ISR9(void) {check_ISR[NOISR9] += 1;}
interrupt void no_ISR10(void) {check_ISR[NOISR10] += 1;}
interrupt void no_ISR11(void) {check_ISR[NOISR11] += 1;}
interrupt void no_ISR12(void) {check_ISR[NOISR12] += 1;}
interrupt void no_ISR13(void) {check_ISR[NOISR13] += 1;}
interrupt void no_ISR14(void) {check_ISR[NOISR14] += 1;}
interrupt void no_ISR15(void) {check_ISR[NOISR15] += 1;}
interrupt void no_ISR16(void) {check_ISR[NOISR16] += 1;}
interrupt void no_ISR17(void) {check_ISR[NOISR17] += 1;}
interrupt void no_ISR18(void) {check_ISR[NOISR18] += 1;}
interrupt void no_ISR19(void) {check_ISR[NOISR19] += 1;}
interrupt void no_ISR20(void) {check_ISR[NOISR20] += 1;}
interrupt void no_ISR21(void) {check_ISR[NOISR21] += 1;}
interrupt void no_ISR22(void) {check_ISR[NOISR22] += 1;}
interrupt void no_ISR23(void) {check_ISR[NOISR23] += 1;}
interrupt void no_ISR24(void) {check_ISR[NOISR24] += 1;}
interrupt void no_ISR25(void) {check_ISR[NOISR25] += 1;}
interrupt void no_ISR26(void) {check_ISR[NOISR26] += 1;}
interrupt void no_ISR27(void) {check_ISR[NOISR27] += 1;}
interrupt void no_ISR28(void) {check_ISR[NOISR28] += 1;}
interrupt void no_ISR29(void) {check_ISR[NOISR29] += 1;}
interrupt void no_ISR30(void) {check_ISR[NOISR30] += 1;}
interrupt void no_ISR31(void) {check_ISR[NOISR31] += 1;}
```

```
/****************************************************
void DAC_clock_enable()

Enables Timer 0 as DAC Clock

****************************************************/
void DAC_clock_enable()
{
	GPTCTL1_0 |= 0x0080;		// Timer 0 enabled continuously
}


/****************************************************
void DAC_clock_disable()

Disables Timer 0 as DAC clock

****************************************************/
void DAC_clock_disable()
{
	GPTCTL1_0 &= ~0x0080;		// Timer 0 disabled
}


/****************************************************
uint32 set_DAC_clock(uint32 dfreq)

sets DAC clock output at the nearest possbile frequency
to the desired frequency and returns the actaul
frequency the clock is set at.  Frequencies are given in Hz.

****************************************************/
uint32 set_DAC_clock(uint32 dfreq)
{
	uint32 afreq,divisor,remainder;

	dfreq *= 2;		// Timer frequency must be twice desired sample frequency

	remainder = SYSCLK1 % dfreq;
	divisor = (SYSCLK1-remainder)/dfreq;
	if (remainder > dfreq/2)
	{
		divisor++;
	}
	remainder = SYSCLK1 % (divisor * 2);
	afreq = SYSCLK1 / (divisor * 2);
	if (remainder > divisor)
	{
		afreq++;
	}

	GPTPRD2_0 = (uint16)((divisor & 0xFFFF0000) >> 16);		// Set upper 16 bits of Period
	GPTPRD1_0 = (uint16)(divisor & 0x0000FFFF);			// Set lower 16 bits of period

	return afreq;

}

/****************************************************
uint16 push_txbuf16(uint16 data)

Splits a 16-bit word into two 8-bit pieces and a places them in txbuf
Word is transmitted Most Significant Byte first (Big Endian)

Returns:
0: Buffer full and push failed
1: Push succeeded
****************************************************/
uint16 push_txbuf16(uint16 data)
{
```

```
	uint16 prev_cl;
	prev_cl = code_loc;
	code_loc = PUSH_TX16;

	if (txend < UART_BUFSIZE - 1)// if there is enough room in the buffer
	{
		txbuf[txend] = (data & 0xFF00) >> 8;
		txbuf[txend+1] = data & 0x00FF;
		txend += 2;
		code_loc = prev_cl;
		return 1;
	}
	else
	{
		code_loc = prev_cl;
		return 0;
	}

}



/****************************************************
uint16 push_txbuf32(uint32 data)

Splits a 32-bit word into four 8-bit pieces and a places them in txbuf
Word is transmitted Most Significant Byte first (Big Endian)

Returns:
0: Buffer full and push failed
1: Push succeeded
****************************************************/
uint16 push_txbuf32(uint32 data)
{
	uint16 prev_cl;
	prev_cl = code_loc;
	code_loc = PUSH_TX32;

	if (txend < UART_BUFSIZE - 3)// if there is enough room in the buffer
	{
		txbuf[txend] = (data & 0xFF000000) >> 24;
		txbuf[txend+1] = (data & 0x00FF0000) >> 16;
		txbuf[txend+2] = (data & 0x0000FF00) >> 8;
		txbuf[txend+3] = (data & 0x000000FF);
		txend += 4;
		code_loc = prev_cl;
		return 1;
	}
	else
	{
		code_loc = prev_cl;
		return 0;
	}

}


/****************************************************
void rxbuf_reset(void)

Receive buffer reset.  Called when data received by the UART
has been handled.

****************************************************/
void rxbuf_reset(void)
{
	uint16 prev_cl;
	prev_cl = code_loc;
	code_loc = RX_RESET;
```

```
		rxstart = 0;
		rxend = 0;
		rxbusy = 1;
		code_loc = prev_cl;
}


/***************************************************
void handle_rxdata(void)

Handles data received via the UART based on the control byte
(first byte int the transmission)
***************************************************/
void handle_rxdata(void)
{
		uint16 control;
		uint32 data;

		uint16 prev_cl;
		prev_cl = code_loc;
		code_loc = HDL_RX;

		control = rxbuf[0];
		switch (control & 0x009F)
		{
				case SET_FC:
						fc = (rxbuf[2] << 8) + rxbuf[3];
						set_carrier(fc);
						break;
				case SET_BW:
						// Do nothing here
						break;
				case SET_FS:
						data = (rxbuf[2] << 8) + rxbuf[3];
						fs = set_DAC_clock(data);
						break;
				case SET_MOD:				// Not really anthing to do with this one yet
						mod = rxbuf[2];
						break;
				case SEND_DATA:
						// This will be used in the future - do nothing yet
						break;
				case SEND_SMPL:
						// This will be used in the future - do nothing yet
						break;
				case GET_FC:
						if (txbusy == 0)	// if no data already being sent
						{
								txbuf[0] = control;
								txbuf[1] = 6;
								push_txbuf32(fc);
								txbusy = 1;
						}
						else
						{
								code_loc = prev_cl;
								return;
						}
						break;
				case GET_BW:
						if (txbusy == 0)	// if no data already being sent
						{
								txbuf[0] = control;
								txbuf[1] = 6;
								push_txbuf32(0x0000);
								txbusy = 1;
						}
						else
						{
```

```
                    code_loc = prev_cl;
                    return;
                }
                break;
            case GET_FS:
                if (txbusy == 0)        // if no data already being sent
                {
                    txbuf[0] = control;
                    txbuf[1] = 6;
                    push_txbuf32(fs);
                    txbusy = 1;
                }
                else
                {
                    code_loc = prev_cl;
                    return;
                }
                break;
            case GET_MOD:
                if (txbusy == 0)        // if no data already being sent
                {
                    txbuf[0] = control;
                    txbuf[1] = 3;
                    txbuf[3] = mod;
                    txbusy = 1;
                }
                else
                {
                    code_loc = prev_cl;
                    return;
                }
                break;

        }
        rxbuf_reset();          // Release rxbuf
        code_loc = prev_cl;
}
```

File: dspt_main.h

```
#ifndef DSPT_MAIN_H
#define DSPT_MAIN_H

#include "dspt_init.h"
#include "data_types.h"

/* -------------- Changeable Parameters ---------------------- */
#define SPI0_SCLK_DIV        75          // Divedes the Slow Peripheral Clock (75 MHz) to get SCLK for SPI0
                                         // Acceptable Values: 1 - 256
#define SPI1_SCLK_DIV        75          // Divedes the Slow Peripheral Clock (75 MHz) to get SCLK for SPI1
                                         // Acceptable Values: 1 - 256
#define UART_BAUD_DIV        80          // Divisor = 82 to make baud rate 58594 (75 MHz/(16*80))

#define FS_FPFD     (uint32)4000     // Frequency Synthesizer Phase Frequency Detector (in kHz)
                                     // Set by varying D and R (see freqsyn_init(void) comments)
#define FS_FREQ_MIN          1000000     // Minimum Frequency for frequency synthesizer
#define FS_FREQ_MAX          2000000     // Maximum Frequency for frequency synthesizer

#define FS_INT_MIN           250     // Minimum value for INT in freq synthesizer (corresponds to 1 GHz)
#define FS_INT_MAX           500     // Maximum value for INT in freq synthesizer (corresponds to 2 GHz)

#define VGA_GAIN_MIN         0x471C  // Minimum gain for the VGA (corresponds to 0.5V: GAIN Vref = 1.8V)
#define VGA_GAIN_MAX         0xC71B  // Maximum gain for the VGA (corresponds to 1.4V: GAIN Vref = 1.8V)

#define BUFSIZE              512     // Buffer size for ibuf and qbuf
#define UART_BUFSIZE         64      // Buffer size for rxbuf and txbuf

#define SYSCLK1              150000000   // Fast Peripherals Clock Group
#define SYSCLK2              75000000  // Slow Peripherals Clock Group
#define SYSCLK3              150000000   // External Memroy Interface Clock Group

/* Control Bytes for UART Communication */
/* Possible OUT Control Bytes */
#define SET_FC               0x0000  // Set Carrier Frequency
#define SET_BW               0x0001  // Set Bandwidth - This is not really applicable on the transmitter
#define SET_FS               0x0002  // Set Sampling Frequency
#define SET_MOD              0x0003  // Set Modulation Technique
#define SEND_DATA            0x0004  // Send given data
#define SEND_SMPL0x0005      // Send given samples
/* Possible IN Control Bytes - Must change bits 6..5 depending on EP used */
#define    GET_FC            0x0080  // Get Current Carrier Frequency
#define GET_BW               0x0081  // Get Current Bandwidth
#define GET_FS               0x0082  // Get Current Sampling Frequency
#define GET_MOD              0x0083  // Get Current Modulation
/* Possible Control Bytes to Microcontroller */
#define SEND_LCD   0x0001    // Send characters to LCD



/* Frequency Synthesizer Initialization Values */
/* N Divider Register */
#define FASTLOCK  0                  // 0: Disabled
                                     // 1: Enabled
#define INTEGER              500         // Valid Range: 31 - 511 if PRESCALER = 0
                                     //                  91 - 511 if
PRESCALER = 1
#define FRAC                 0           // Valid Range: 0 - (MOD-1)
#define REG_N                0           // Control Bits for N divider register
/* R Divider Register */
#define LD_CNTL              0           // Load Control: 0: Normal Operation
                                     //                  1: Load
Resync
#define MUXOUT               1           // 0: Three-State
                                     // 1: Digital Lock Detect
                                     // 2: N Divider Output
                                     // 3: Logic High
                                     // 4: R Divider Output
                                     // 5: Analog Lock Detect
```

```
                                                // 6: Fastlock Switch
                                                // 7: Logic Low
#define PRESCALER 0                 // 0: 4/5 - Can only go up to 2 GHz
                                                // 1: 8/9
#define R_COUNTER       5           // Valid Range: 1 - 15
#define MOD                   4000  // Valid Range: 2 - 4095
#define REG_R           1           // Control Bits for R divider register
/* Control Register */
#define RESYNC          1           // Valid Range: 1 - 15
#define REF_DBL         0           // Reference Doubler: 0: Disabled
                                                //                  1:
Enabled
#define CP_CRNT         0x4         // CP Current Setting - See ADF4153 Data sheet pg 13
#define PD_POL          1           // PD Polarity: 0: Negative
                                                //                  1: Positive
#define LDP                   1           // Lock Detect Precision: 0: 24 PFD Cycles
                                                //
        1: 40 PFD Cycles
#define PWR_DWN         0           // Powerdown: 0: Normal Operation
                                                //             1: Power Down
#define CP_TRI          0           // CP Three-State: 0: Disabled
                                                //                  1: Three-
State
#define COUNT_RST       0           // Counter Reset: 0: Normal Operation
                                                //                  1:
Counters in Reset
#define REG_CNTL  2           // // Control Bits for control register
/* Noise and Spur Register */
#define REG_NS_VAL      0x03C7      // Noise and Spur Mode - includes control bits
                                                // 0x0003: Lowest Spur Mode
                                                // 0x0383: Low Noise and Low Spur Mode
                                                // 0x03C7: Lowest Noise Mode
/* Register Value Computations */
#define REG_N_VAL       (((uint32)FASTLOCK & 0x1)<<23)+(((uint32)INTEGER & 0x1FF)<<14)+(((uint32)FRAC &
0xFFF)<<2)+(REG_N & 0x3)
#define REG_R_VAL((uint32)(LD_CNTL & 0x1)<<23)+((uint32)(MUXOUT & 0x7)<<20)+((uint32)(PRESCALER &
0x1)<<18)+((uint32)(R_COUNTER & 0xF)<<14)+((uint32)(MOD & 0xFFF)<<2)+(REG_R & 0x3)
#define REG_C_VAL((uint32)(RESYNC & 0xF)<<12)+((REF_DBL & 0x1)<<11)+((CP_CRNT & 0xF)<<7)+((PD_POL &
0x1)<<6)+((LDP & 0x1)<<5)+((PWR_DWN & 0x1)<<4)+((CP_TRI & 0x1)<<3)+((COUNT_RST & 0x1)<<2)+(REG_CNTL &
0x3)

/* --------- Bit Masks and Pin Function Definitions ------------ */
/* Bit Mask for SPCR2 regiter for McBSPs */
#define bmXRDY     0x0002   // Bit mask for the transmit buffer ready bit of SPCR2
#define bmXEMPTY   0x0004   // Bit mask for all transmissions
#define bmRRDY     0x0002   // bit mask for data recieved bit of SPCR1

/* Bit Masks for UART Line Status Register (URLSR) */
#define bmTEMT     0x0040   // Transmitter Empty in URLSR register (UART Line Status Register)
#define bmTHRE     0x0020   // Transmitter Holding Register Empty in URLSR
#define bmDR       0x0001   // Receiver Data Ready indicator in URLSR

/* UART Trigger Interrupt command */
#define TRIGGER_UART        IFR0 |= 0x1000;           // Triggers a UART interrupt to go to uart_ISR

/* HGPIO pins */
#define FS_MUXOUT                   (HGPIODAT2 & 0x0040)
#define TX_ENABLE                   HGPIODAT2 |= 0x0002;
#define TX_DISABLE                  HGPIODAT2 &= ~0x0002;
#define DACI_PLLLOCK        (HGPIODAT2 & 0x0080)
#define DACQ_PLLLOCK        (HGPIODAT2 & 0x0010)
/* PGPIO Pins */
#define DACI        PGPIODAT0
#define DACQ        PGPIODAT1

/* Interrupts: for use with int_enable and int_disable */
#define INT_DATA_RDY        INT_INT0
#define INT_UART            INT_RINT2_UART

#define INT_GLOBAL                  32                // ST1_55
```

```
#define INT_INT0            2           // IER0
#define INT_INT2            3           // IER0
#define INT_TINT0           4           // IER0
#define INT_RINT0           5           // IER0
#define INT_RINT1           6           // IER0
#define INT_XINT1           7           // IER0
#define INT_DMAC1           9           // IER0
#define INT_DSPINT          10          // IER0
#define INT_INT3_WDTINT     11          // IER0
#define INT_RINT2_UART      12          // IER0
#define INT_XINT2           13          // IER0
#define INT_DMAC4           14          // IER0
#define INT_DMAC5           15          // IER0

#define INT_INT1            16          // IER1
#define INT_XINT0           17          // IER1
#define INT_DMAC0           18          // IER1
#define INT_DMAC2           20          // IER1
#define INT_DMAC3           21          // IER1
#define INT_TINT1           22          // IER1
#define INT_I2C                  23          // IER1
#define INT_BERR            24          // IER1
#define INT_DLOG            25          // IER1
#define INT_RTOS            26          // IER1

/* Bit Mask for INTM in ST1_55 */
#define bmINT_GLOBAL             0x0800      // set to diable interrupts, clear to enable interrupts
/* Bit Masks for IER0, IFR0 */
#define bmINT_DMAC5                  0x8000
#define bmINT_DMAC4                  0x4000
#define bmINT_XINT2                  0x2000
#define bmINT_RINT2_UART   0x1000
#define bmINT_INT3_WDTINT 0x0800
#define bmINT_DSPINT             0x0400
#define bmINT_DMAC1                  0x0200
#define bmINT_XINT1                  0x0080
#define bmINT_RINT1                  0x0040
#define bmINT_RINT0                  0x0020
#define bmINT_TINT0                  0x0010
#define bmINT_INT2               0x0008
#define bmINT_INT0               0x0004
/* Bit Masks for IER1, IFR1 */
#define bmINT_RTOS                   0x0400
#define bmINT_DLOG                   0x0200
#define bmINT_BERR                   0x0100
#define bmINT_I2C                0x0080
#define bmINT_TINT1                  0x0040
#define bmINT_DMAC3                  0x0020
#define bmINT_DMAC2                  0x0010
#define bmINT_INT4               0x0008
#define bmINT_DMAC0                  0x0004
#define bmINT_XINT0                  0x0002
#define bmINT_INT1               0x0001

/* Code Locations */
#define NUM_LOCS 32

#define     INIT_CLK0
#define INIT_TMR    1
#define INIT_GPIO   2
#define INIT_HPI    3
#define INIT_SPI0   4
#define INIT_SPI1   5
#define INIT_UART   6
#define INIT_INT    7
#define INIT_FQSYN          8
#define SET_CARR    9
#define SEND_FQSYN          10
#define INIT_VGA    11
#define SET_VGA             12
```

```
#define SEND_MSG  13
#define INT_ENBL    14
#define INT_DABL    15
#define INT0ISR                 16
#define INT1ISR                 17
#define INT2ISR                 18
#define INT3ISR                 19
#define NOINTISR    20
#define UARTISR                 21
#define PUSH_TX16 22
#define PUSH_TX32 23
#define RX_RESET   24
#define HDL_RX                  25
#define     INIT_DDC            26
#define WRT_MICRO               27
#define WRT_DDC                 28
#define INIT_I2C      29
#define INIT_DSPR   30
#define MAIN                    31

/* ISR NUMBERS */
#define NUM_ISRS   32

#define NOISR0                  0                       // RESET
#define NOISR1                  1                       // NMI
#define NOISR2                  2                       // INT0
#define NOISR3                  3                       // INT2
#define NOISR4                  4                       // TINT0
#define NOISR5                  5                       // RINT0
#define NOISR6                  6                       // RINT1
#define NOISR7                  7                       // XINT1
#define NOISR8                  8                       // LCKINT1
#define NOISR9                  9                       // DMAC1
#define NOISR10                 10                      // DSPINT
#define NOISR11                 11                      // INT3/WDTINT
#define NOISR12                 12                      // RINT2/UART
#define NOISR13                 13                      // XINT2
#define NOISR14                 14                      // DMAC4
#define NOISR15                 15                      // DMAC5
#define NOISR16                 16                      // INT1
#define NOISR17                 17                      // XINT0
#define NOISR18                 18                      // DMAC0
#define NOISR19                 19                      //
#define NOISR20                 20                      // DMAC2
#define NOISR21                 21                      // DMAC3
#define NOISR22                 22                      // TINT1
#define NOISR23                 23                      // IIC
#define NOISR24                 24                      // BERR
#define NOISR25                 25                      // DLOG
#define NOISR26                 26                      // RTOS
#define NOISR27                 27                      //
#define NOISR28                 28                      //
#define NOISR29                 29                      //
#define NOISR30                 30                      //
#define NOISR31                 31                      //

extern uint16 check_ISR[NUM_ISRS];
extern uint16 code_loc;


void freqsyn_init(void);
uint16 set_carrier(uint32 freq);    // in kHz
uint16 freqsyn_reinit(uint32 reg_val);          // Frequency Synthesizer reinitialize when it gets hopelessly unlocked
void send_freqsyn(uint32 data);   // will only use right-most 24 bits

void vga_gain_init(void);                       // initialize gain to approx 0 dB
void set_vga_gain(uint16 voltage);              // valid inputs correspond to 0.5V to 1.4V
                                                                        // 0x1999 - 0x47AE

void send_lcd(char * string, uint16 length);   // Send a character string to the LCD connected to the microcontroller
```

```
void int_enable(uint16 int_num); // enable interrupt in reg with bitmask bm
void int_disable(uint16 int_num);// disable interrupt in reg with bitmask bm
interrupt void int0_ISR(void);                    // External Interrupt 0 ISR
interrupt void uart_ISR(void);                    // UART ISR

interrupt void no_ISR1(void);
interrupt void no_ISR2(void);
interrupt void no_ISR3(void);
interrupt void no_ISR4(void);
interrupt void no_ISR5(void);
interrupt void no_ISR6(void);
interrupt void no_ISR7(void);
interrupt void no_ISR8(void);
interrupt void no_ISR9(void);
interrupt void no_ISR10(void);
interrupt void no_ISR11(void);
interrupt void no_ISR12(void);
interrupt void no_ISR13(void);
interrupt void no_ISR14(void);
interrupt void no_ISR15(void);
interrupt void no_ISR16(void);
interrupt void no_ISR17(void);
interrupt void no_ISR18(void);
interrupt void no_ISR19(void);
interrupt void no_ISR20(void);
interrupt void no_ISR21(void);
interrupt void no_ISR22(void);
interrupt void no_ISR23(void);
interrupt void no_ISR24(void);
interrupt void no_ISR25(void);
interrupt void no_ISR26(void);
interrupt void no_ISR27(void);
interrupt void no_ISR28(void);
interrupt void no_ISR29(void);
interrupt void no_ISR30(void);
interrupt void no_ISR31(void);

void DAC_clock_enable();                // Enable Timer 0 interrupts to clock DAC
void DAC_clock_disable();               // Disable Timer 0 interrupts to clock DAC
uint32 set_DAC_clock(uint32 freq);      // Set frequency of DAC clock (in Hz)
                                                        // returns actual value nearest to
desired value
uint16 push_txbuf16(uint16 data);       // Used to split a 16-bit word into two and place in txbuf
uint16 push_txbuf32(uint32 data);       // Used to split a 32-bit word into four and place in txbuf
void rxbuf_reset(void);                           // Resets rxbuf when data used
void handle_rxdata(void);               // Handle received data
#endif
```

File: dspt_init.c

```
#include "dspt_init.h"
#include "dspt_main.h"
#include "data_types.h"
#include "c5502regs.h"



/****************************************************
void dspr_init(void)

This function simply calls all the other initialization
functions.

****************************************************/
void dspt_init(void)
{
	uint16 prev_cl;			// Holds the previous code location and will return code_loc to it at the end of the
function

/* External Bus Control */
	prev_cl = code_loc;
	code_loc = INIT_DSPR;
	XBCR = 0x0000;			// Enable pull-up, pull-down, and bus holders on all pins (p.97)

	i2c_init();
	clock_init();
	timer_init();
	gpio_init();
	hpi_init();
	spi0_init();
	spi1_init();
	uart_init();
	int_init();

	TX_DISABLE			// Disable Receiver

	code_loc = prev_cl;
}


/****************************************************
void i2c_init(void)

This function will need to initialize the I2C module on
the DSP when code is loaded onto the EPROM.  This
function is not yet written.

****************************************************/
void i2c_init(void)
{
	uint16 prev_cl;			// Holds the previous code location and will return code_loc to it at the end of the
function
/* I2C module Configuration --- This must be done before PLL Configuration for running from I2C EEPROM */
	prev_cl = code_loc;
	code_loc = INIT_I2C;



	code_loc = prev_cl;
}


/****************************************************
void timer_init(void)

Initialize timer 0, but don't enable it.  Timer 0 is not
used yet on this DSP
```

```
*****************************************************/
void timer_init(void)
{
	uint16 prev_cl;			// Holds the previous code location and will return code_loc to it at the end of the
function
/* Timer Configuration */
	prev_cl = code_loc;
	code_loc = INIT_TMR;

	TSSR = 0x0038;		// Timer Signal Selection Register (p.45)
						// Timer Pins as outputs
						// Disable Watchdog Timer
	GPTEMU_0 = 0x0000;		// Timer 0 stops immediately when emulation suspend event occurs
	GPTCLK_0 = 0x0000;		// Use Fast Peripherals Clock as clock source for timer
	GPTGPINT_0 = 0x0000;		// Don't use the Timer input pin as interrupt source
	GPTGPEN_0 = 0x0000;		// Don't use TIN1 and TOUT1 as GPIO

	GPTPRD1_0 = 0x0002;		// Set the timer output to oscillate at 25 MHz
	GPTPRD2_0 = 0x0000;
	GPTPRD3_0 = 0x0000;
	GPTPRD4_0 = 0x0000;

	GPTGCTL1_0 = 0x0003;		// Timer 0 in 64-bit mode, both 32-bit timers not in reset
	GPTGCTL1_1 = 0x0003;		// Timer 1 in 64-bit mode, both 32-bit timers not in reset
	WDTGCTL1 = 0x0003;		// WDT in 64-bit mode, both 32-bit timers not in reset

	WDTWCTL1 = 0x8000;		// Disable watchdog timer

	GPTCTL1_0 = 0x0008;		// Timer 0 configured for clock mode, but not enabled yet (0x0088 to enable)
	GPTCTL1_1 = 0x0000; // Timer 1 disabled
	WDTGCTL1 = 0x0000;		// WDT disabled

	code_loc = prev_cl;
}


/*****************************************************
void clock_init(void)

Initialize the internal clocks of the DSP.  CPU clock
set to 300 MHz, SYSCLK1 to 150 MHz, SYSCLK2 to 75 MHz,
and SYSCLK3 to 150 MHz

*****************************************************/
void clock_init(void)
{
	uint16 i;
	uint16 prev_cl;			// Holds the previous code location and will return code_loc to it at the end of the
function

	prev_cl = code_loc;
	code_loc = INIT_CLK;

/* Clock configuration */
	/* Configure PLL Control Register (p.57)
	1. Switch to bypass mode by setting the PLLEN bit to 0.
	2. Set the PLL to its reset state by setting the PLLRST bit to 1.
	3. Change the PLL setting through the PLLM and PLLDIV0 bits.
	4. Wait for 1 microsec.
	5. Release the PLL from its reset state by setting PLLRST to 0.
	6. Wait for the PLL to relock by polling the LOCK bit or by setting up a LOCK interrupt
	7. Switch back to PLL mode by setting the PLLEN bit to 1.
	*/

	PLLCSR &= ~bmPLLEN;		// 1. Switch to bypass mode by setting the PLLEN bit to 0 (p.59)
	PLLCSR |= bmPLLRST;		// 2. Set the PLL to its reset state by setting the PLLRST bit to 1 (p.59)
	PLLDIV0 = 0x8000;	// 3. Set to divide by 1 (20 MHz -> 20 MHz) (p.61)
	PLLM = 0x000F;			// 3. Set to multiply by 15 (20 MHz -> 300 MHz) (p.60)
	for (i=0; i<10; i++);	// 4. Wait for 1 microsec (Clock at 20 MHz, period of 0.1 usec)
```

```
        PLLCSR &= ~bmPLLRST;        // 5. Release the PLL from its reset state by setting PLLRST to 0 (p.59)
        while (PLLCSR & bmLOCK);    // 6. Wait for the PLL to relock by polling the LOCK bit (p.59)
        PLLCSR |= bmPLLEN;          // 7. Switch back to PLL mode by setting the PLLEN bit to 1 (p.59)

        CK3SEL = 0x000B;   // Select the PLL output as the source for CLKOUT3 (System Clock) (p.67)
        PLLDIV1 = 0x8001;  // Set SYSCLK1 (Fast Peripherals Clock Group) to divide by 2 (300 MHz -> 150 MHz) (p.62)
        PLLDIV2 = 0x8003;  // Set SYSCLK2 (Slow Peripherals Clock Group) to divide by 4 (300 MHz -> 75 MHz) (p.63)
        PLLDIV3 = 0x8001;  // Set SYSCLK3 (External Memory Interface Group) to divide by 2 (300 MHz -> 150 MHz) (p.64)
        OSCDIV1 = 0x0000;  // Disable OD1 divider for CLKOUT3.  This is not part of PLL path (p.64)

        WKEN = 0x001F;     // Disable wakeup from all external interrupts (p.66)
        CLKOUTSR = 0x0003;          // Disable the CLKOUT pin (goes to logic 0) (p.68)


        code_loc = prev_cl;
}

/****************************************************
void int_init(void)

Initialize the interrupt masks as needed.  This function
does not globally enable interrupts

****************************************************/
void int_init(void)
{
        uint16 prev_cl;             // Holds the previous code location and will return code_loc to it at the end of the
function
/* Interrupts --------- Different between Transmitter and Receiver DSP (need INT1,INT2,INT3 in Receiver)*/
        prev_cl = code_loc;
        code_loc = INIT_INT;

        int_disable(INT_GLOBAL);

        IVPD = 0x00FF;
        IVPH = 0x00FF;

        IER0 = 0x1004;
                /* Enabled:
                        RINT2/UART - McBSP Recieve or UART Interrupt    Bit 12
                        INT0 - External Interrupt 0                                     Bit 2
                -- Not Enabled, but maybe later
                        XINT2 - McBSP 2 Transmit Interrupt                    Bit 13
                        XINT1 - McBSP 1 Transmit Interrupt                    Bit 7
                        RINT1 - McBSP 1 Receive Interrupt                     Bit 6
                        RINT0 - McBSP 0 Receive Interrupt                     Bit 5
                */
        IER1 = 0x0000;
                /* Enabled:
                -- Not Enabled, but maybe later
                        I2C - I2C Interrupt
        Bit 7
                        XINT0 - McBSP 0 Transmit Interrupt                    Bit 1
                */
        /* Clear all pending interrupts */
        IFR0 = 0xFFFF;
        IFR1 = 0xFFFF;

        code_loc = prev_cl;
}


/****************************************************
void spi0_init(void)

SPI0 Configuration (connected to ADF4153 Frequency Synthesizer)
                24-bit words
                clocked on raising edge of SCLK
                SCLK is 20 MHz maximum
```

```
*****************************************************/
void spi0_init(void)
{
	uint16 i;
	uint16 prev_cl;			// Holds the previous code location and will return code_loc to it at the end of the
function

	prev_cl = code_loc;
	code_loc = INIT_SPI0;

	SPCR1_0 &= ~0x0001;		// Reset the Receiver
	SPCR2_0 &= ~0x00C1;		// Reset the Transmitter, Sample Rate Generator and frame sync logic


	SPCR1_0 = 0x1800;	// Set Clock Stop Mode to 1/2 cycle delay to center data on raising edge of SCLK
//	SPCR1_0 = 0x9800;	// Digital Loopback Enabled; Set Clock Stop Mode to 1/2 cycle delay to center data on raising edge
of SCLK

	SPCR2_0 = 0x0100;	// Set up McBSP0 for soft stop on emulator halt
	RCR1_0 = 0x0080;	// Set up to receive words of 24 bits long
	RCR2_0 = 0x0081;	// Set up a 1 bit delay on the receive data
	XCR1_0 = 0x0080;	// Set up to transmit words of 24 bits long
	XCR2_0 = 0x0081;	// Set up a 1 bit delay on the transmit data
	SRGR1_0 = 0x1700+SPI0_SCLK_DIV-1;
						// Set Frame Sync to 24 cycles wide (value=23) and diveder for
CLKG to 38 (value=37)
						// to bring SCLK down to approx 1 MHz (37.5 MHz / 38 = 1 MHz)
	SRGR2_0 = 0x2000;	// McBSP uses internal input clock (used in conjunction with SCLKME bit)
						// Sets up transmitter to drive FSX pin every time data is transfered
	MCR1_0 = 0x0000;
	MCR2_0 = 0x0000;
	PCR0 = 0x0F0C;			// Configure Port Control Register (p.12-38 of SPRU592e data sheet)
	RCERA_0 = 0x0000;
	RCERB_0 = 0x0000;
	XCERA_0 = 0x0000;
	XCERB_0 = 0x0000;
	RCERC_0 = 0x0000;
	RCERD_0 = 0x0000;
	XCERC_0 = 0x0000;
	XCERD_0 = 0x0000;
	RCERE_0 = 0x0000;
	RCERF_0 = 0x0000;
	XCERE_0 = 0x0000;
	XCERF_0 = 0x0000;
	RCERG_0 = 0x0000;
	RCERH_0 = 0x0000;
	XCERG_0 = 0x0000;
	XCERH_0 = 0x0000;

	SPCR2_0 |= 0x0040;	// Enable the Sample Rate Generator
	SPCR2_0 |= 0x0080;	// Enable Frame Sync Logic
	for (i=0; i<3*SPI0_SCLK_DIV; i++);		// Wait at least 2 sample rate generator clock periods for logic to stabalize

	SPCR1_0 |= 0x0001;	// Enable the Receiver
	SPCR2_0 |= 0x0001;	// Enable the Transmitter

	code_loc = prev_cl;
}


/*****************************************************
void spi1_init(void)

SPI1 Configuration (connected to AD5060 DAC controlling the VGA)
			24-bit words
			clocked on falling edge of SCLK
			SCLK is 30 MHz maximum

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

```
void spi1_init(void)
{
	uint16 i;
	uint16 prev_cl;			// Holds the previous code location and will return code_loc to it at the end of the
function

	prev_cl = code_loc;
	code_loc = INIT_SPI1;

	SPCR1_1 &= ~0x0001;		// Reset the Receiver
	SPCR2_1 &= ~0x00C1;		// Reset the Transmitter, Sample Rate Generator and frame sync logic


	SPCR1_1 = 0x1000;	// Set Clock Stop Mode to 1/2 cycle delay to center data on raising edge of SCLK
//	SPCR1_1 = 0x9000;	// Digital Loopback Enabled; Set Clock Stop Mode to 1/2 cycle delay to center data on raising edge
of SCLK

	SPCR2_1 = 0x0100;	// Set up McBSP1 for soft stop on emulator halt
	RCR1_1 = 0x0080;	// Set up to receive words of 24 bits long
	RCR2_1 = 0x0081;	// Set up a 1 bit delay on the receive data
	XCR1_1 = 0x0080;	// Set up to transmit words of 24 bits long
	XCR2_1 = 0x0081;	// Set up a 1 bit delay on the transmit data
	SRGR1_1 = 0x1700+SPI1_SCLK_DIV-1;
						// Set Frame Sync to 24 cycles wide (value=23) and diveder for
CLKG to 38 (value=37)
						// to bring SCLK down to approx 1 MHz (37.5 MHz / 38 = 1 MHz)
	SRGR2_1 = 0x2000;	// McBSP uses internal input clock (used in conjunction with SCLKME bit)
						// Sets up transmitter to drive FSX pin every time data is transfered
	MCR1_1 = 0x0000;
	MCR2_1 = 0x0000;
	PCR1 = 0x0F0C;			// Configure Port Control Register (p.12-38 of SPRU592e data sheet)
	RCERA_1 = 0x0000;
	RCERB_1 = 0x0000;
	XCERA_1 = 0x0000;
	XCERB_1 = 0x0000;
	RCERC_1 = 0x0000;
	RCERD_1 = 0x0000;
	XCERC_1 = 0x0000;
	XCERD_1 = 0x0000;
	RCERE_1 = 0x0000;
	RCERF_1 = 0x0000;
	XCERE_1 = 0x0000;
	XCERF_1 = 0x0000;
	RCERG_1 = 0x0000;
	RCERH_1 = 0x0000;
	XCERG_1 = 0x0000;
	XCERH_1 = 0x0000;

	SPCR2_1 |= 0x0040;	// Enable the Sample Rate Generator
	SPCR2_1 |= 0x0080;	// Enable Frame Sync Logic
	for (i=0; i<3*SPI1_SCLK_DIV; i++);		// Wait at least 2 sample rate generator clock periods for logic to stabalize

	SPCR1_1 |= 0x0001;	// Enable the Receiver
	SPCR2_1 |= 0x0001;	// Enable the Transmitter

	code_loc = prev_cl;
}



/****************************************************
void hpi_init(void)

Initialize the HPI ports to be used as GPIO

****************************************************/
void hpi_init(void)
{
```

```
	uint16 prev_cl;			// Holds the previous code location and will return code_loc to it at the end of the
function
/* HPI Configuration ---------Different between transmitter and receiver */
	prev_cl = code_loc;
	code_loc = INIT_HPI;

	HGPIOEN = 0x19D7;// Enable all HPI pins and GPIO
	HGPIODIR1 = 0xFFFF;		// Enable HD15:0 as Outputs
	HGPIODAT1 = 0x0000;		// Push HD15:0 to low (floating connections)
	HGPIODIR2 = 0x012F;		// Enable HPI pins as Inputs or Outputs as needed
	HGPIODAT2 = 0x0000;		// Push floating connections low and disable transmit
	HGPIODIR3 = 0xFFFF;		// Enable HA15:0 as GPIO
	HGPIODAT3 = 0x0000;		// Push HA15:0 to low (floating connections)

	code_loc = prev_cl;
}


/****************************************************
void gpio_init(void)

Initialize the general purpose inputs as inputs (they are
not used) and the parallel GPIO as inputs for PGPIO0/1 as
the data inputs from the FPGA and PGPIO2 as outputs to the
DDCs

****************************************************/
void gpio_init(void)
{
	uint16 prev_cl;			// Holds the previous code location and will return code_loc to it at the end of the
function
/* GPIO Configuration ---------Different between transmitter and receiver */
	prev_cl = code_loc;
	code_loc = INIT_GPIO;

	IODIR = 0x0000;	// Enable GPIO as inputs

	/*
	Use PGPIO as Outputs: PGPIOENx = PGPIODIRx = 1
	Use PGPIO as Inputs: PGPIOENx = PGPIODIRx = 0
	*/
	PGPIOEN0 = 0xFFFF;		// Enable PGPIO0
	PGPIODIR0 = 0xFFFF;		// Set PGPIO0 as Outputs
	PGPIODAT0 = 0x0000;		// Set PGPIO0 low

	PGPIOEN1 = 0xFFFF;		// Enable PGPIO1
	PGPIODIR1 = 0xFFFF;		// Set PGPIO1 as Outputs
	PGPIODAT1 = 0x0000;		// Set PGPIO1 low

	PGPIOEN2 = 0x3FFF;		// Enable PGPIO2
	PGPIODIR2 = 0x3FFF;		// Set PGPIO2 as Outputs
	PGPIODAT2 = 0x0000;		// Set PGPIO2 low

	code_loc = prev_cl;
}


/****************************************************
void uart_init(void)

Initialize the UART using the baud rate divisor defined
in dspr_init.h

****************************************************/
void uart_init(void)
{
	uint16 prev_cl;			// Holds the previous code location and will return code_loc to it at the end of the
function
/* UART Configuration */
	prev_cl = code_loc;
```

```
        code_loc = INIT_UART;

        URDLM = (UART_BAUD_DIV & 0xFF00) >> 8;
        URDLL = (UART_BAUD_DIV & 0x00FF);           // Divisor = 82 to make baud rate 57165 (75 MHz/(16*41))
        URLCR = 0x0003;    // Configure the Parity (no parity), stop bit (1 stop bit) and word length (8 bits)
        URMCR = 0x0000;             // Disable Loop back for UART debugging
        URIER = 0x0003;             // Enable interrupts for transmit buffer empty and recieve data ready
        URPECR = 0x8000;   // Enable the UART trasnmitter and receiver

        code_loc = prev_cl;
}
```

File: dspt_init.h

```
#ifndef DSPT_INIT_H
#define DSPT_INIT_H

// PLLCSR Register bit masks
#define bmSTABLE   0x0040
#define bmLOCK          0x0020
#define bmPLLRST   0x0008
#define bmOSCPWRDN      0x0004
#define bmPLLPWRDN      0x0002
#define bmPLLEN         0x0001

void dspt_init(void);
void timer_init(void);
void clock_init(void);
void int_init(void);
void spi0_init(void);
void spi1_init(void);
void hpi_init(void);
void gpio_init(void);
void uart_init(void);
void i2c_init(void);

#endif
```

APPENDIX F.

TRANSMITTER TESTING SETUP PICTURES

Transmitter Test Setup: Semi-Anechoic Chamber

Transmitter Test Setup: Semi-Anechoic Chamber

Transmitter Test Setup: SDR Setup

Transmitter Test Setup: Spectrum analyzer

APPENDIX G.

MICROCONTROLLER CODE

File: main.c

```
#include "c8051F120.h"
#include "lcd.h"
#include "usb.h"
#include "uart.h"
#include "max.h"
#include "main.h"

extern void Init_Device(void);

extern unsigned char in3buf[64];
extern unsigned char in3len;

uchar num_interrupts = 0;

void main(void)
{
		uchar mesnumten = 0;
		uchar mesnumone = 1;
		uchar i, offset, line;
		uchar usb_init;
		uchar csfrpage;

		SS_HI
		USB_RST = 0;		// Place USB controller in reset


		Init_Device();

		usb_init = init_usb();

		delay_us(40000);

		i = rreg(prEPIEN);
		i = rreg(prUSBIEN);
		i = rreg(prUSBCTL);
		i = rreg(prPINCTL);

		for (i=0; i<64; i++)
		{
			uart0r_buf[i] = 0;
		}

		LCD_init();
		LCD_write(LCD_LINE1,"Hello World",11);

		switch (usb_init)
		{
			case 0:
				LCD_write(LCD_LINE2,"USB Failure",11);
				break;
			case 1:
				LCD_write(LCD_LINE2,"USB Success",11);
				break;
		}

		offset = 1;
		line=1;

		while (1)
		{
			if (uart0_data_r_len)
			{
				Transfer_DSP(0,uart0_data_r_len);
			}
			if (uart1_data_r_len)
			{
				Transfer_DSP(1,uart1_data_r_len);
			}
```

```
/*                      mesnumone = rreg(prEPIRQ);
                        mesnumten = rreg(prFNADDR);

                        wreg(prIOPINS1, 0x0F);          // Set GPO bits 3..0
                        wreg(prIOPINS2, 0x00);          // Clear GPO bits 7..4

                        mesnumone = rreg(prIOPINS1);
                        mesnumten = rreg(prIOPINS2);

                        wreg(prIOPINS1, 0x00);          // Clear GPO bits 3..0
                        wreg(prIOPINS2, 0x0A);          // Set GPO bits 7..4

                        mesnumone = rreg(prIOPINS1);
                        mesnumten = rreg(prIOPINS2);
*/


                        if (uart1_busy == 0)
                        {
                                uart1t_buf[0] = 0x84;           // Get Data -- Currently does nothing on DSPR
                                uart1t_buf[1] = 0x04;           // Total length of 4
                                uart1t_buf[2] = 0x55;
                                uart1t_buf[3] = 0xAA;
                                uart1t_len = 4;
                                uart1_busy = 1;
                                csfrpage = SFRPAGE;
                                SFRPAGE = UART1_PAGE;
                                TI1 = 1;
                                SFRPAGE = csfrpage;
                        }


                }

}

/******************************************************
void int_enable(uchar intnum)

Enable the given interrupt

******************************************************/
void int_enable(uchar intnum)
{
        switch (intnum)
        {
                case INT_GLOBAL:
                        EA = 1;
                        break;
                case INT_INT0:
                        EX0 = 1;
                        break;
                case INT_TIMER0:
                        ET0 = 1;
                        break;
                case INT_INT1:
                        EX1 = 1;
                        break;
                case INT_TIMER1:
                        ET1 = 1;
                        break;
                case INT_UART0:
                        ES0 = 1;
                        break;
                case INT_TIMER2:
                        ET2 = 1;
                        break;
                case INT_SPI:
                        EIE1 |= 0x01;
                        break;
```

```
                case INT_SMB:
                        EIE1 |= 0x02;
                        break;
                case INT_ADC0WC:
                        EIE1 |= 0x04;
                        break;
                case INT_PCA0:
                        EIE1 |= 0x08;
                        break;
                case INT_CP0F:
                        EIE1 |= 0x10;
                        break;
                case INT_CP0R:
                        EIE1 |= 0x20;
                        break;
                case INT_CP1F:
                        EIE1 |= 0x40;
                        break;
                case INT_CP1R:
                        EIE1 |= 0x80;
                        break;
                case INT_TIMER3:
                        EIE2 |= 0x01;
                        break;
                case INT_ADC0EOC:
                        EIE2 |= 0x02;
                        break;
                case INT_TIMER4:
                        EIE2 |= 0x04;
                        break;
                case INT_ADC2WC:
                        EIE2 |= 0x08;
                        break;
                case INT_ADC2EOC:
                        EIE2 |= 0x10;
                        break;
                case INT_UART1:
                        EIE2 |= 0x40;
                        break;
        }
}

/*****************************************************
void int_disable(uchar intnum)

Disable the given interrupt

*****************************************************/
void int_disable(uchar intnum)
{
        switch (intnum)
        {
                case INT_GLOBAL:
                        EA = 0;
                        break;
                case INT_INT0:
                        EX0 = 0;
                        break;
                case INT_TIMER0:
                        ET0 = 0;
                        break;
                case INT_INT1:
                        EX1 = 0;
                        break;
                case INT_TIMER1:
                        ET1 = 0;
                        break;
                case INT_UART0:
                        ES0 = 0;
```

```
                        break;
                case INT_TIMER2:
                        ET2 = 0;
                        break;
                case INT_SPI:
                        EIE1 &= ~0x01;
                        break;
                case INT_SMB:
                        EIE1 &= ~0x02;
                        break;
                case INT_ADC0WC:
                        EIE1 &= ~0x04;
                        break;
                case INT_PCA0:
                        EIE1 &= ~0x08;
                        break;
                case INT_CP0F:
                        EIE1 &= ~0x10;
                        break;
                case INT_CP0R:
                        EIE1 &= ~0x20;
                        break;
                case INT_CP1F:
                        EIE1 &= ~0x40;
                        break;
                case INT_CP1R:
                        EIE1 &= ~0x80;
                        break;
                case INT_TIMER3:
                        EIE2 &= ~0x01;
                        break;
                case INT_ADC0EOC:
                        EIE2 &= ~0x02;
                        break;
                case INT_TIMER4:
                        EIE2 &= ~0x04;
                        break;
                case INT_ADC2WC:
                        EIE2 &= ~0x08;
                        break;
                case INT_ADC2EOC:
                        EIE2 &= ~0x10;
                        break;
                case INT_UART1:
                        EIE2 &= ~0x40;
                        break;
        }
}
```

File: main.h

```
#ifndef MAIN_H
#define MAIN_H

#define INT_INT0	0
#define INT_TIMER0		1
#define INT_INT1	2
#define INT_TIMER1		3
#define INT_UART0	4
#define INT_TIMER2		5
#define INT_SPI		6
#define INT_SMB		7
#define INT_ADC0WC		8
#define INT_PCA0	9
#define INT_CP0F	10
#define INT_CP0R	11
#define INT_CP1F	12
#define INT_CP1R	13
#define INT_TIMER3		14
#define INT_ADC0EOC		15
#define INT_TIMER4		16
#define INT_ADC2WC		17
#define INT_ADC2EOC		18
#define INT_UART1	20
#define INT_GLOBAL		21


void int_enable(uchar intnum);
void int_disable(uchar intnum);




#endif
```

File: lcd.c

```
#include "lcd.h"
#include "pindef.h"

#define uchar unsigned char
#define uint unsigned int


/*****************************************************
void delay_us(unsigned long int delay)

Delay the given number of microseconds.

*****************************************************/
void delay_us(unsigned long int delay)
{
	unsigned long int i,j;
	for(i=0;i<delay;i++)
		for(j=0;j<USEC_CYCLES;j++);
}


/*****************************************************
void LCD_init(void)

Initialize the LCD.

*****************************************************/
void LCD_init(void)
{
	LCD_POWER = 1;
	/* Function Set*/
	LCD_inst(LCD_FUNC);
	/* Display ON/OFF Control */
	LCD_inst(LCD_DISPLAY);
	/* Clear Display */
	LCD_inst(LCD_CLEAR);
	/* Entry Mode Set */
	LCD_inst(LCD_ENTRY);
	/* Return to Home Position */
	LCD_inst(LCD_HOME);
}

/*****************************************************
void LCD_inst(uchar instruction)

Send the given byte to the LCD as an instruction

*****************************************************/
void LCD_inst(uchar instruction)
{
	uint stuck = 0;
	uint timeout = 0;
```

```
	while (~LCD_ready())
	{
		stuck++;
		switch (instruction)
		{
			case LCD_CLEAR:
				if (stuck > LCD_TO_1530)
				{
					timeout = 1;
				}
				break;
			case LCD_HOME:
				if (stuck > LCD_TO_1530)
				{
					timeout = 1;
				}
				break;
			default:
				if (stuck > LCD_TO_39)
				{
					timeout = 1;
				}
				break;
		}
		if (timeout)
		{
			break;
		}
	}
	LCD_RS = LCD_RS_INS;
	LCD_RW = LCD_RW_WRTE;
	LCD_DATA = instruction;
	LCD_E = 1;
	delay_us(1);
	LCD_E = 0;
	delay_us(2);
}

/*****************************************************
void LCD_data(uchar character)

Send the given byte to the LCD as data

*****************************************************/
void LCD_data(uchar character)
{
	unsigned int stuck=0;

	while (~LCD_ready())
	{
		stuck++;
		if (stuck>LCD_TO_43)
		{
			break;
		}
	}
```

```
        LCD_RS = LCD_RS_DATA;
        LCD_RW = LCD_RW_WRTE;
        LCD_DATA = character;
        LCD_E = 1;
        delay_us(1);
        LCD_E = 0;
        delay_us(2);
}

/*****************************************************
int LCD_ready(void)

Check to see if the LCD is ready for the next byte.

Return: 0 if not ready
                1 if ready

*****************************************************/
int LCD_ready(void)
{
        uchar csfrpage;
        uchar status;

        csfrpage = SFRPAGE;
        /* set up lcd port as input */
        SFRPAGE = CONFIG_PAGE;
        LCD_DATA_CONFIG = 0xFF;
        LCD_DATA = 0xFF;
        /* get LCD status */
        LCD_RS = LCD_RS_INS;
        LCD_RW = LCD_RW_READ;
        LCD_E = 1;
        delay_us(1);
        status = LCD_DATA;
        LCD_E = 0;
        delay_us(1);

        /* return lcd port to it's previous state */
        LCD_DATA_CONFIG = 0x00;
        SFRPAGE = csfrpage;

        return ~((status & 0x80)>0);
}


/*****************************************************
void LCD_write(uchar line, char * string, uchar length)

Write the given character string to the LCD

Inputs: line - the desired line to write to.  Pass LCD_LINE1,
                                LCD_LINE2, etc.
                string - character array with the desired message
                length - length of the string (max 20)

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

```
void LCD_write(uchar line, char * string, uchar length)
{
	uchar i;

	LCD_inst(line);			// go to specified line

	if (length > 20)
	{
		length = 20;
	}

	for (i=0; i<length; i++)
	{
		LCD_data(string[i]);
	}
}
```

File: lcd.h

```
#ifndef LCD_H
#define LCD_H

#include "c8051F120.h"
#include "data_types.h"

/* An empty 'for' loop in C takes 3 instructions per loop.
One microsecond takes 98 cycles at 98 MHz,
so 32 instances of a 'for' loop is approx 1 usec. */
#define USEC_CYCLES 32

#define LCD_DATA_CONFIG   P2MDOUT
#define LCD_DATA          P2              /* Bidirectional, Output default */
#define LCD_RS                    P3_5    /* Output */
#define LCD_RW                    P3_6    /* Output */
#define LCD_E                     P3_7    /* Output */

#define LCD_RS_INS  0
#define LCD_RS_DATA       1
#define LCD_RW_WRTE       0
#define LCD_RW_READ 1

#define LCD_FUNC  0x3C
#define LCD_DISPLAY       0x0C
#define LCD_ENTRY         0x06

#define LCD_CLEAR         0x01
#define LCD_LINE1  0x80
#define LCD_LINE2  0xC0
#define LCD_LINE3  0x94
#define LCD_LINE4  0xD4
#define LCD_HOME 0x02

#define LCD_TO_1530       510             // LCD timeout for 1.53 ms
#define LCD_TO_39  13             // LCD timeout for 39 us
#define LCD_TO_43  14             // LCD timeout for 43 us


/*
Delays the program by 'delay' microseconds.
*/
void delay_us(unsigned long int delay);

/*

*/
void LCD_init(void);

/*

*/
void LCD_inst(unsigned char instruction);

/*

*/
void LCD_data(unsigned char character);

/*

*/
int LCD_ready(void);


/*
Send a string of characters to the given line.  The character string must be 20 characters or less.
*/
```

```
void LCD_write(uchar line, char * string, uchar length);

#endif
```

File: device_init.c

```
////////////////////////////////////
//  Generated Initialization File  //
////////////////////////////////////

#include "c8051F120.h"

// Peripheral specific initialization functions,
// Called from the Init_Device() function
void Reset_Sources_Init()
{
    WDTCN     = 0xDE;
    WDTCN     = 0xAD;
}

void Timer_Init()
{
    SFRPAGE   = TIMER01_PAGE;
    TCON      = 0x40;
    TMOD      = 0x20;
    CKCON     = 0x01;
    TH1       = 0x2F;
    SFRPAGE   = TMR2_PAGE;
    TMR2CN    = 0x04;
    TMR2CF    = 0x08;
    RCAP2L    = 0x95;
    RCAP2H    = 0xFF;
    SFRPAGE   = TMR3_PAGE;
    TMR3CF    = 0x18;
    RCAP3L    = 0x97;
    RCAP3H    = 0x40;
    TMR3L     = 0x97;
    TMR3H     = 0x40;
}

void UART_Init()
{
    SFRPAGE   = UART0_PAGE;
    SCON0     = 0x50;
    SSTA0     = 0x15;
    SFRPAGE   = UART1_PAGE;
    SCON1     = 0x50;
}

void SPI_Init()
{
    SFRPAGE   = SPI0_PAGE;
    SPI0CFG   = 0x40;
    SPI0CN    = 0x01;
    SPI0CKR   = 0xFF;
}

void Voltage_Reference_Init()
{
    SFRPAGE   = ADC0_PAGE;
    REF0CN    = 0x01;
}

void Port_IO_Init()
{
    // P0.0  -  TX0 (UART0), Push-Pull,  Digital
    // P0.1  -  RX0 (UART0), Open-Drain, Digital
    // P0.2  -  SCK  (SPI0), Push-Pull,  Digital
    // P0.3  -  MISO (SPI0), Open-Drain, Digital
    // P0.4  -  MOSI (SPI0), Push-Pull,  Digital
    // P0.5  -  TX1 (UART1), Push-Pull,  Digital
    // P0.6  -  RX1 (UART1), Open-Drain, Digital
    // P0.7  -  INT0 (Tmr0), Open-Drain, Digital
```

```
    // P1.0  -  Unassigned,  Push-Pull,  Digital
    // P1.1  -  Unassigned,  Open-Drain, Digital
    // P1.2  -  Unassigned,  Push-Pull,  Digital
    // P1.3  -  Unassigned,  Push-Pull,  Digital
    // P1.4  -  Unassigned,  Open-Drain, Digital
    // P1.5  -  Unassigned,  Open-Drain, Digital
    // P1.6  -  Unassigned,  Open-Drain, Digital
    // P1.7  -  Unassigned,  Push-Pull,  Digital

    // P2.0  -  Unassigned,  Push-Pull,  Digital
    // P2.1  -  Unassigned,  Push-Pull,  Digital
    // P2.2  -  Unassigned,  Push-Pull,  Digital
    // P2.3  -  Unassigned,  Push-Pull,  Digital
    // P2.4  -  Unassigned,  Push-Pull,  Digital
    // P2.5  -  Unassigned,  Push-Pull,  Digital
    // P2.6  -  Unassigned,  Push-Pull,  Digital
    // P2.7  -  Unassigned,  Push-Pull,  Digital

    // P3.0  -  Unassigned,  Open-Drain, Digital
    // P3.1  -  Unassigned,  Push-Pull,  Digital
    // P3.2  -  Unassigned,  Push-Pull,  Digital
    // P3.3  -  Unassigned,  Open-Drain, Digital
    // P3.4  -  Unassigned,  Push-Pull,  Digital
    // P3.5  -  Unassigned,  Push-Pull,  Digital
    // P3.6  -  Unassigned,  Push-Pull,  Digital
    // P3.7  -  Unassigned,  Push-Pull,  Digital

    SFRPAGE   = CONFIG_PAGE;
    P0MDOUT   = 0x35;
    P1MDOUT   = 0x8D;
    P2MDOUT   = 0xFF;
    P3MDOUT   = 0xF6;
    XBR0      = 0x06;
    XBR1      = 0x04;
    XBR2      = 0x44;
}

void Oscillator_Init()
{
    int i = 0;
    SFRPAGE   = CONFIG_PAGE;
    CCH0CN    &= ~0x20;
    SFRPAGE   = LEGACY_PAGE;
    FLSCL     = 0xB0;
    SFRPAGE   = CONFIG_PAGE;
    CCH0CN    |= 0x20;
    PLL0CN    |= 0x01;
    PLL0DIV   = 0x01;
    PLL0FLT   = 0x01;
    PLL0MUL   = 0x04;
    for (i = 0; i < 15; i++);  // Wait 5us for initialization
    PLL0CN    |= 0x02;
    while ((PLL0CN & 0x10) == 0);
    CLKSEL    = 0x02;
    OSCICN    = 0x83;
}

void Interrupts_Init()
{
    IE        = 0x91;
    IP        = 0x10;
    EIE1      = 0x01;
    EIE2      = 0x40;
    EIP1      = 0x01;
    EIP2      = 0x40;
}

// Initialization function for device,
// Call Init_Device() from your main program
void Init_Device(void)
```

```
{
   Reset_Sources_Init();
   Timer_Init();
   UART_Init();
   SPI_Init();
   Voltage_Reference_Init();
   Port_IO_Init();
   Oscillator_Init();
   Interrupts_Init();
}
```

File: usb.c

```
#include "c8051F120.h"
#include "pindef.h"
#include "usb.h"
#include "max.h"
#include "lcd.h"
#include "uart.h"



// Global Variables
/*-----------------------------------------------*/
uchar usb_status;	// Status byte read in as first byte of SPI transfer
		/*	Status byte composition
			Bit0: IN0BAVIRQ
			Bit1: OUT0DAVIRQ
			Bit2: OUT1DAVIRQ
			Bit3: IN2BAVIRQ
			Bit4: IN3BAVIRQ
			Bit5: SUDAVIRQ
			Bit6: URESIRQ
			Bit7: SUSPIRQ
		*/

uchar SUD[8];			// Buffer for Set-Up Data

uchar in0buf[64];	// Buffer for EP0-IN
uchar in0len;			// Length of data for EP0-IN

uchar out0buf[64];	// Buffer for EP0-OUT
uchar out0len;			// Length of data available from EP0-OUT

uchar out1buf[64];	// Buffer for EP1-OUT
uchar out1len;			// Length of data available from EP1-OUT

uchar in2buf[64];	// Buffer for EP2-IN
uchar in2len;			// Length of data for EP2-IN

uchar in3buf[64];	// Buffer for EP3-IN
uchar in3len;			// Length of data for EP3-IN

uchar configval;	// Current Configuration Value being used
uchar suspend;			// Whether the device is in SUSPEND mode or not
uchar RWU_enabled; // Whether Remote Wake-Up is enabled or not
uchar ep1stall;			// Whether EP1 is stalled or not
uchar ep2stall;			// Whether EP2 is stalled or not
uchar ep3stall;			// Whether EP3 is stalled or not
uchar osc_ready;	// Whether the MAX on-chip oscillator is stable

uchar int0_num = 0;

/*-----------------------------------------------*/


/*-------------------- Enumeration Data -----------------------*/
const uchar DD[]=	// DEVICE Descriptor
	{0x12,				// bLength = 18d
	0x01,			// bDescriptorType = Device (1)
	0x01,0x01,		// bcdUSB(L/H) USB spec rev (BCD) - Runs at USB 1.1
		0x00,0x00,0x00,	// bDeviceClass, bDeviceSubClass, bDeviceProtocol
		0x40,			// bMaxPacketSize0 EP0 is 64 bytes
		0x42,0x00,		// idVendor(L/H)--Set as unused Value: 0x0042
		0x01,0x00,		// idProduct(L/H)--Set to be 1: 0x0001
		0x01,0x00,		// bcdDevice--Device release number: 0x1234
		1,2,3,			// iManufacturer, iProduct, iSerialNumber
		1};			// bNumConfigurations

const uchar CD[]=	// CONFIGURATION Descriptor
```

```
        {0x09,                  // bLength
        0x02,                   // bDescriptorType = Config
        0x27,0x00,              // wTotalLength(L/H) = 39 bytes
        0x01,                   // bNumInterfaces - 1 interface: EP1 & EP2: Bulk, EP3: Interrupt
        0x01,                   // bConfigValue
        0x00,                   // iConfiguration
        0xE0,                   // bmAttributes. b7=1 b6=self-powered b5=RWU supported
        0x01,                   // MaxPower is 2 ma
// INTERFACE Descriptor for Bulk Transfers
        0x09,                   // length = 9
        0x04,                   // type = IF
        0x00,                   // IF #0
        0x00,                   // bAlternate Setting
        0x03,                   // bNum Endpoints - EP1 and EP2
        0xFF,                   // bInterfaceClass = Vendor Specific
        0x00,0x00,              // bInterfaceSubClass, bInterfaceProtocol
        0x00,                   // iInterface
// Endpoint Descriptor
        0x07,                   // bLength
        0x05,                   // bDescriptorType (Endpoint)
        0x81,                   // bEndpointAddress (EP1-OUT)
        0x02,                   // bmAttributes     (Bulk)
        64,0,       // wMaxPacketSize (64)
        0,
// Endpoint Descriptor
        0x07,                   // bLength
        0x05,                   // bDescriptorType (Endpoint)
        0x82,                   // bEndpointAddress (EP2-IN)
        0x02,                   // bmAttributes     (Bulk)
        64,0,       // wMaxPacketSize (64)
        0,
// Endpoint Descriptor
        0x07,                   // bLength
        0x05,                   // bDescriptorType (Endpoint)
        0x83,                   // bEndpointAddress (EP3-IN)
        0x03,                   // bmAttributes     (interrupt)
        64,0,       // wMaxPacketSize (64)
        10};                    // bInterval (poll every 10 msec)


// STRING descriptors. An array of string arrays

const uchar strDesc[][64]= {
// STRING descriptor 0--Language string
{
    0x04,               // bLength
        0x03,                   // bDescriptorType = string
        0x09,0x04       // wLANGID(L/H) = English-United Sates
},
// STRING descriptor 1--Manufacturer ID
{
    60,                 // bLength
        0x03,                   // bDescriptorType = string
        'U',0,'M',0,'R',0,' ',0,
                'T',0,'e',0,'l',0,'e',0,'m',0,'e',0,'t',0,'r',0,'y',0,
                ' ',0,'L',0,'e',0,'a',0,'r',0,'n',0,'i',0,'n',0,'g',0,
                ' ',0,'C',0,'e',0,'n',0,'t',0,'e',0,'r',0 // text in Unicode
},
// STRING descriptor 2 - Product ID
{       28,                     // bLength
        0x03,                   // bDescriptorType = string
        'L',0,'o',0,'w',0,' ',0,'R',0,'a',0,'n',0,'g',0,'e',0,
    ' ',0,'S',0,'D',0,'R',0,
},

// STRING descriptor 3 - Serial Number ID
{    18,                // bLength
        0x03,                   // bDescriptorType = string
        'S',0,
        '/',0,
```

```
            'N',0,
            ' ',0,
            '0',0,
            '0',0,
            '0',0,
            '1',0
}};
/*-----------------------------------------------------------------*/


/*****************************************************
void INT0_ISR(void) interrupt 0

Interrupt Service Routine for interrupt 0 triggered by the
USB controller.  Examines which interrupt flags triggered
the interrupt and handles them

*****************************************************/
void INT0_ISR(void) interrupt 0
{
            uchar epirq, usbirq, epien, usbien;
            uchar clear;

            int0_num++;

            usbirq = rreg(prUSBIRQ);
            epirq = usb_status & 0x3F;
            usbien = rreg(prUSBIEN);
            epien = rreg(prEPIEN);

            clear = epirq - (epirq & epien);                    // clear interrrupts not enabled
            wreg(prEPIRQ,clear);
            clear = usbirq - (usbirq & usbien);                              // clear interrrupts not enabled
            wreg(prUSBIRQ,clear);


            usbirq &= usbien;       // only look at interrupts if they are enabled
            epirq &= epien;                         // only look at interrupts if they are enabled


            // Check EPIRQ buffer for Interrupt Requests
            if (epirq & bmSUDAVIRQ)                      // Setup Data received
            {
                        wreg(prEPIRQ,bmSUDAVIRQ); // clear the IRQ
                        do_SUD();
            }
            if (epirq & bmIN3BAVIRQ)                     // EP3-IN empty and data ready for it
            {
                        if (in3len>0)
                        {
                                    do_IN3();                                     // Send data
                        }
                        else
                        {
                                    wreg(prEPIRQ,bmIN3BAVIRQ);              // Clear interrupt request if no data to send
                        }
            }
            if (epirq & bmIN2BAVIRQ)                     // EP2-IN empty and data ready for it
            {
                        if (in2len>0)
                        {
                                    do_IN2();                                     // Send data
                        }
                        else
                        {
                                    wreg(prEPIRQ,bmIN2BAVIRQ);              // Clear interrupt request if no data to send
                        }
            }
            if (epirq & bmOUT1DAVIRQ)    // Data received to EP1-OUT
            {
```

```
		wreg(prEPIRQ,bmOUT1DAVIRQ);		// clear the IRQ
		do_OUT1();
	}
	if (epirq & bmOUT0DAVIRQ)	// Data received to EP0-OUT
	{
		do_OUT0();
	}
	if (epirq & bmIN0BAVIRQ)			// EP0-IN empty and data ready for it
	{
		if (in0len>0)
		{
			do_IN0();				// Send data
		}
		else
		{
			wreg(prEPIRQ,bmIN0BAVIRQ);		// Clear interrupt request if no data to send
		}
	}

	// Check the USBIRQ buffer for interrupt requests
	if ((configval != 0) && (usbirq & bmSUSPIRQ))
	{
		wreg(prUSBIRQ,(bmSUSPIRQ + bmBUSACTIRQ));	// clear the IRQ and bus activity IRQ
		suspend = 1;
		usbirq = rreg(prUSBIRQ);
	}
	if (usbirq & bmURESIRQ)
	{
		wreg(prUSBIRQ,bmURESIRQ); // Clear the URES IRQ
		usbirq = rreg(prUSBIRQ);
	}
	if (usbirq & bmURESDNIRQ)
	{
		wreg(prUSBIRQ,bmURESDNIRQ);		// Clear the URESDN IRQ
		suspend = 0;				// In case in suspend mode
		enable_irqs();				// Re-enable IRQs
		usbirq = rreg(prUSBIRQ);
	}
	if ((suspend==1)&&(usbirq & bmBUSACTIRQ))	// if bus activity resumes while in suspend
	{
		wreg(prUSBIRQ,bmBUSACTIRQ);
		suspend = 0;
		usbirq = rreg(prUSBIRQ);
	}
	if (usbirq & bmRWUDNIRQ)
	{
		CLRBIT(prUSBCTL,bmSIGRWU)		// remove the RESUME signal
		wreg(prUSBIRQ,bmRWUDNIRQ);		// Clear the RWUDN IRQ
		usbirq = rreg(prUSBIRQ);
	}
	if (usbirq & bmOSCOKIRQ)
	{
		osc_ready = 1;
		wreg(prUSBIRQ,bmOSCOKIRQ);		// Clear the IRQ
	}
}


/*****************************************************
void SPI0_ISR(void) interrupt 6

ISR for SPI bus.

*****************************************************/
void SPI0_ISR(void) interrupt 6
{
	static uchar wcol_count=0;

	if (SPIF)
	{
```

```
                SPIF=0;
        }
        if (WCOL)
        {
                WCOL=0;
                LCD_inst(LCD_LINE2);
                LCD_data('W');
                LCD_data('C');
                LCD_data('O');
                LCD_data('L');
                LCD_data(' ');
                LCD_data('0'+wcol_count);
                LCD_data(' ');
                LCD_data(' ');
                wcol_count = (++wcol_count)%10;
        }
        if (MODF)
        {
                MODF=0;
                LCD_inst(LCD_LINE2);
                LCD_data('M');
                LCD_data('O');
                LCD_data('D');
                LCD_data('F');
                LCD_data(' ');
                LCD_data(' ');
                LCD_data(' ');
        }
        if (RXOVRN)
        {
                RXOVRN=0;
                LCD_inst(LCD_LINE2);
                LCD_data('R');
                LCD_data('X');
                LCD_data('O');
                LCD_data('V');
                LCD_data('R');
                LCD_data('N');
                LCD_data(' ');
        }

}



/* ------------------------ MAX Chip Configuration ----------------------*/
/*******************************************************
uchar init_usb(void)

Initializes the MAX3421E USB controller.

*******************************************************/
uchar init_usb(void)
{
        uchar value;
        uint stuck = 0;

        configval = 0;
        suspend = 0;
        RWU_enabled = 0;
        ep1stall = 0;
        ep2stall = 0;
        ep3stall = 0;
        in0len = 0;
        in2len = 0;
        in3len = 0;
        osc_ready = 0;

        USB_RST = 0;        // Place USB controller in reset
```

```
        delay_us(1);            // delay 1 msec

        USB_RST = 1;            // Pull USB controller out of reset

        value = bmFDUPSPI + bmINTLEVEL;                 // 0x18
        wreg(prPINCTL,value);
        value = rreg(prPINCTL);

        value = reset_usb();

        enable_irqs();

        value = rreg(prEPIEN);
        value = rreg(prUSBIEN);

        value = bmHOSCSTEN + bmVBGATE;                          // 0xC0
        wreg(prUSBCTL,value);
        value = rreg(prUSBCTL);

        value = rreg(prEPIEN);
        value = rreg(prUSBIEN);

        wreg(prCPUCTL,bmIE);            // Enable interrupt pin

        while ((USB_GPX==0) && (osc_ready==0))          // wait for on-chip oscillator to stablize or get the OPERATE signal
        {
                value = rreg(prUSBIRQ);
                if ((value & bmOSCOKIRQ) == bmOSCOKIRQ)
                {
                        osc_ready = 1;
                        wreg(prUSBIRQ,bmOSCOKIRQ);              // Clear the IRQ
                }
                if ((++stuck) > OSC_TIMEOUT)
                {
                        return 0;
                }
        }

        value = bmHOSCSTEN + bmVBGATE+bmCONNECT;                // 0xC8
        wreg(prUSBCTL,value);

        value = rreg(prUSBCTL);

        value = rreg(prEPIEN);
        value = rreg(prUSBIEN);

        return 1;
}


/*****************************************************
void enable_irqs(void)

Enable the interrupt request masks needed.

*****************************************************/
void enable_irqs(void)
{
        uchar temp;

//      value = bmSUDAVIE | bmOUT1DAVIE | bmOUT0DAVIE;
        wreg(prEPIEN,0x26);

//      value = bmURESDNIE | bmSUSPIE | bmURESIE | bmRWUDNIE | bmOSCOKIE;
        wreg(prUSBIEN,0x9B);
        temp = rreg(prEPIEN);
        temp = rreg(prUSBIEN);
        temp = rreg(prEPIEN);
```

```
}

/*******************************************************
void rwu_usb(void)

Initiate a Remote Wake-up when in Shutdown Mode

*******************************************************/
void rwu_usb(void)
{
	if (suspend && RWU_enabled)
	{
		SETBIT(prUSBCTL,bmSIGRWU);		// signal a RWU
	}
}

/*******************************************************
uchar reset_usb(void)

Perform a chip reset on the MAX3421E.

*******************************************************/
uchar reset_usb(void)
{
	uchar value=0;
	uint stuck;

	wreg(prUSBCTL,bmCHIPRES);		// set chip reset
	wreg(prUSBCTL,0x00);				// clear chip reset
	while ((USB_GPX==0) && (osc_ready==0))		// wait for on-chip oscillator to stablize or get the OPERATE signal
	{
		value = rreg(prUSBIRQ);
		if ((value & bmOSCOKIRQ) == bmOSCOKIRQ)
		{
			osc_ready = 1;
			wreg(prUSBIRQ,bmOSCOKIRQ);		// Clear the IRQ
		}
		if ((++stuck) > OSC_TIMEOUT)
		{
			return 0;
		}
	}
	wreg(prUSBIRQ,bmOSCOKIRQ);
	return 1;
}



/*-------------------- Endpoint Transfers -------------------*/

/*******************************************************
void do_SUD(void)

Handle a Setup Data Transfer.

*******************************************************/
void do_SUD(void)
{
	readbytes(prSUDFIFO,8,SUD);		// Read the 8 SETUP bytes
	switch (SUD[bmRequestType]&0x60)	// Determine request type.  Bits 5&6.
	{
	case 0x00:				// Standard Request
		std_request();
		break;
	case 0x20:				// Class Request - Issues STALL in this program
		class_request();
		break;
```

```
		case 0x40:				// Vendor Request - Issues STALL in this program
			vendor_request();
			break;
		default:
			STALL_EP0
		}
}

/*******************************************************
void do_OUT0(void)

Handle an OUT transfer (from computer to SDR) over Endpoint 0
(Control Transfer)

*******************************************************/
void do_OUT0(void)  // Control Transfer
{
		if (out0len==0)			// If the previous data has been handled
		{
			out0len = rreg(prEP0BC);						// Determine number of bytes to
read
			readbytes(prEP0FIFO,out0len,out0buf);	// Read data from FIFO
			wreg(prEPIRQ,bmOUT0DAVIRQ);						// clear the IRQ
		}
}

/*******************************************************
void do_IN0(void)

Handle an IN transfer request over Endpoint 0 (Control Transfer)

*******************************************************/
void do_IN0(void)	// Control Transfer
{
		if (in0len>0)		// If there is data ready for EP3-IN
		{
			if (rreg(prEPIRQ)&bmIN0BAVIRQ)		// If buffer available
			{
				writebytes(prEP0FIFO,in0len,in0buf);	// Write the message to the FIFO
				wreg(prEP0BC,in0len);						// Load the Byte
Count Register to trigger the MAX chip to send the data
				in0len = 0;
			}
			SETBIT(prEPIEN,bmIN0BAVIE)		// Turn on the Interrupt for EP3-IN Buffer available
		}
		else
		{
			CLRBIT(prEPIEN,bmIN0BAVIE)		// Turn off the Interrupt for EP3-IN Buffer available
		}
}

/*******************************************************
void do_OUT1(void)

Handle an OUT transfer over EP1 (Bulk Transfer)

*******************************************************/
void do_OUT1(void)  // Bulk Transfer
{
		if (out1len==0)			// If the previous data has been handled
		{
			out1len = rreg(prEP1OUTBC);						// Determine number of bytes to
read
			readbytes(prEP1OUTFIFO,out1len,out1buf);		// Read data from FIFO
			wreg(prEPIRQ,bmOUT1DAVIRQ);						// clear the IRQ
			Transfer_USB_Bulk();
		}
```

```
}

/*****************************************************
void do_IN2(void)

Handle an IN tranfer request over EP2 (Bulk Transfer)

*****************************************************/
void do_IN2(void)     // Bulk Transfer
{
        if (in2len>0)           // If there is data ready for EP3-IN
        {
                if (rreg(prEPIRQ)&bmIN2BAVIRQ)          // If buffer available
                {
                        writebytes(prEP2INFIFO,in2len,in2buf);    // Write the message to the FIFO
                        wreg(prEP2INBC,in2len);                                       // Load the Byte
Count Register to trigger the MAX chip to send the data
                        in2len = 0;
                }
                SETBIT(prEPIEN,bmIN2BAVIE)            // Turn on the Interrupt for EP3-IN Buffer available
        }
        else
        {
                CLRBIT(prEPIEN,bmIN2BAVIE)            // Turn off the Interrupt for EP3-IN Buffer available
        }
}

/*****************************************************
void do_IN3(void)

Send data to the MAX3421E EP3 for transmission at the next
interrupt signaled by the computer (Interrupt Transfer)

*****************************************************/
void do_IN3(void)     // Interrupt Transfer
{
        if (in3len>0)           // If there is data ready for EP3-IN
        {
                if (rreg(prEPIRQ)&bmIN3BAVIRQ)          // If buffer available
                {
                        writebytes(prEP3INFIFO,in3len,in3buf);    // Write the message to the FIFO
                        wreg(prEP3INBC,in3len);                                       // Load the Byte
Count Register to trigger the MAX chip to send the data
                        in3len = 0;
                }
                SETBIT(prEPIEN,bmIN3BAVIE)            // Turn on the Interrupt for EP3-IN Buffer available
        }
        else
        {
                CLRBIT(prEPIEN,bmIN3BAVIE)            // Turn off the Interrupt for EP3-IN Buffer available
        }
}


/*-------------------- Higher Level Transfer Functions -------------------*/


/*****************************************************
void Transfer_USB_Bulk(void)

Perform an USB Bulk Transfer (Uses EP1 and EP2)
```

```
*****************************************************/
void Transfer_USB_Bulk(void)
{
	uchar control, target;
	uchar i;

	control = out1buf[0];
	target = control & 0x60;
	if (target == 0x00)			// Target Receiving DSP
	{
		while (uart0_busy);	// Wait for the UART to become available
		uart0t_buf[0] = (control & 0x9F) + 0x40;	// Change target to EP2 for return transmission
		for (i=1; i<64; i++)
		{
			uart0t_buf[i] = out1buf[i];
		}
		uart0t_len = out1len;
		out1len = 0;
		TI0 = 1;				// Trigger a UART0 Interrupt to start transfer
	}
	if (target == 0x20)			// Target Transmitting DSP
	{
		while (uart1_busy);	// Wait for the UART to become available
		uart1t_buf[0] = (control & 0x9F) + 0x40;	// Change target to EP2 for return transmission
		for (i=1; i<64; i++)
		{
			uart1t_buf[i] = out1buf[i];
		}
		uart1t_len = out1len;
		out1len = 0;
		TI1 = 1;				// Trigger a UART1 Interrupt to start transfer
	}
	if (target == 0x40)			// Target Microcontroller
	{
		switch (control)
		{
		case SEND_LCD:
			LCD_inst(LCD_LINE1);
			for (i=1; i<out1len; i++)
			{
				switch (i)
				{
				case 21: LCD_inst(LCD_LINE2);
					break;
				case 41: LCD_inst(LCD_LINE3);
					break;
				case 61: LCD_inst(LCD_LINE4);
					break;
				default : break;
				}
				LCD_data(out1buf[i]);
			}
			break;
		case GET_JUNK:
			while (in2len != 0);	// Wait for IN2 to be ready
			in2buf[0] = 'J';
			in2buf[1] = 'u';
			in2buf[2] = 'n';
			in2buf[3] = 'k';
			in2len = 4;
			do_IN2();
			break;
		}
		out1len = 0;
	}
}

/*****************************************************
void control_in(uchar control, uchar len)
```

```
Handle a Control IN transfer:
Inputs: control - control byte for the tranfer indicates
                                what data is requested
        len - indicates the length of the data requested

*****************************************************/
void control_in(uchar control, uchar len)
{
        uchar target;
        uchar i;
        uchar junk[5];

        junk[0] = 'J';
        junk[1] = 'u';
        junk[2] = 'n';
        junk[3] = 'k';
        junk[4] = ' ';


        target = control & 0x60;
        if (target == 0x00)             // Target Receiving DSP
        {
                while (uart0_busy);     // Wait for the UART to become available
                uart0t_buf[0] = (control & 0x9F) + 0x00;     // Change target to EP0 for return transmission
                uart0t_buf[1] = len;
                uart0t_len = 2;
        }
        if (target == 0x20)             // Target Transmitting DSP
        {
                while (uart1_busy);     // Wait for the UART to become available
                uart1t_buf[0] = (control & 0x9F) + 0x00;     // Change target to EP0 for return transmission
                uart1t_buf[1] = len;
                uart1t_len = 2;
        }
        if (target == 0x40)             // Target Microcontroller
        {
                switch (control)
                {
                case GET_JUNK:
                        while (in0len != 0);    // Wait for IN0 to be ready
                        for (i=0; i<len; i++)
                        {
                                in0buf[i] = junk[i%5];
                        }
                        in0len = len;
                        do_IN0();
                        break;
                }
        }
}


/*****************************************************
void control_out(uchar control)

Handles a Control OUT tranfer

Inputs: control - control byte for the transfer designates
                                destination and purpose of transfer

*****************************************************/
void control_out(uchar control)
{
        uchar target;
        uchar i;

        while (out0len == 0);  // Wait for the data to transfer in over OUT0

        target = control & 0x60;
```

```
        if (target == 0x00)                 // Target Receiving DSP
        {
                uart0t_buf[0] = out0len;            // Send the length of the packet first
                for (i=1; i<out0len+1; i++)
                {
                        uart0t_buf[i] = out0buf[i-1];
                }
                uart0t_len = out0len+1;
                out0len = 0;
        }
        if (target == 0x20)                 // Target Transmitting DSP
        {
                uart1t_buf[0] = out0len;            // Send the length of the packet first
                for (i=1; i<out0len+1; i++)
                {
                        uart1t_buf[i] = out0buf[i-1];
                }
                uart1t_len = out0len+1;
                out0len = 0;
        }
        if (target == 0x40)                 // Target Microcontroller
        {
                switch (control)
                {
                case SEND_LCD:
                        LCD_inst(LCD_LINE1);
                        for (i=0; i<out0len; i++)
                        {
                                switch (i)
                                {
                                case 21: LCD_inst(LCD_LINE2);
                                        break;
                                case 41: LCD_inst(LCD_LINE3);
                                        break;
                                case 61: LCD_inst(LCD_LINE4);
                                        break;
                                default : break;
                                }
                                LCD_data(out1buf[i]);
                        }
                        break;
                }
        }

}




/*-------------------- USB Functions -------------------*/


/*****************************************************
void std_request(void)

Function to handle a standard request by the computer.

*****************************************************/
void std_request(void)
{
        switch (SUD[bRequest])
        {
        case SR_GET_DESCRIPTOR:
                send_descriptor();
                break;
        case SR_SET_FEATURE:
                feature(1);
```

```
			break;
		case SR_CLEAR_FEATURE:
			feature(0);
			break;
		case SR_GET_STATUS:
			get_status();
			break;
		case SR_SET_INTERFACE:
			set_interface();
			break;
		case SR_GET_INTERFACE:
			get_interface();
			break;
		case SR_GET_CONFIGURATION:
			get_configuration();
			break;
		case SR_SET_CONFIGURATION:
			set_configuration();
			break;
		case SR_SET_ADDRESS:
			rregAS(prFNADDR);	// discard return value
			break;
		case SR_CONTROL_IN:			// Device Specific request - I use this Request for requesting the Micro to send
data via CONTROL Transfer
			control_in(SUD[wValueL], SUD[wLengthL]);		// for now, this program can only handle lengths of up
to 64, so only need the low byte
			break;
		case SR_CONTROL_OUT:		// Device Specific request - I use this Request for sending the Micro data via CONTROL
Transfer
			control_out(SUD[wValueL]);
			break;
		default:
			STALL_EP0
		}
}

/*******************************************************
void class_request(void)

Function to handle a class request by the computer.  Just
stalls EP0

*******************************************************/
void class_request(void)
{
		STALL_EP0
}


/*******************************************************
void vendor_request(void)

Function to handle a vendor request by the computer.  Just
stalls EP0

*******************************************************/
void vendor_request(void)
{
		STALL_EP0
}


/*******************************************************
void send_descriptor(void)

Function to send the SDRs descriptor in response to a standard
request of GET DESCRIPTOR

*******************************************************/
void send_descriptor(void)
```

```
{
	uint reqlen,sendlen,desclen;
	uchar *pDdata;						// pointer to ROM Descriptor data to send
	//
	// NOTE This function assumes all descriptors are 64 or fewer bytes and can be sent in a single packet
	//
	desclen = 0;						// check for zero as error condition (no case statements
satisfied)
	reqlen = (((uint)SUD[wLengthH]) << 8) + SUD[wLengthL];	// 16-bit
	switch (SUD[wValueH])				// wValueH is descriptor type
	{
		case  GD_DEVICE:
			desclen = DD[0];	// descriptor length
			pDdata = DD;
			break;
		case  GD_CONFIGURATION:
			desclen = CD[2];	// Config descriptor includes interface, HID, report and ep descriptors
			pDdata = CD;
			break;
		case  GD_STRING:
			desclen = strDesc[SUD[wValueL]][0];   // wValueL=string index, array[0] is the length
			pDdata = strDesc[SUD[wValueL]];	// point to first array element
			break;
	}		// end switch on descriptor type
	if (desclen!=0)		// one of the case statements above filled in a value
	{
		sendlen = (reqlen <= desclen) ? reqlen : desclen; // send the smaller of requested and available
		writebytes(prEP0FIFO,sendlen,pDdata);
		wregAS(prEP0BC,sendlen);   // load EP0BC to arm the EP0-IN transfer & ACKSTAT
	}
	else STALL_EP0  // none of the descriptor types match
}


/*****************************************************
void feature(uchar sc)

Function to set the SDRs feature in response to a SET FEATURE
or CLEAR FEATURE standard request

Inputs: sc - 1 for SET FEATURE, 0 for CLEAR FEATURE

*****************************************************/
void feature(uchar sc)
{
	uchar mask;
	if((SUD[bmRequestType]==0x02)		// dir=h->p, recipient = ENDPOINT
	&&  (SUD[wValueL]==0x00))   // wValueL is feature selector, 00 is EP Halt
	{
		if (SUD[wIndexL]==0x81)		// Endpoint 1
		{
			mask=rreg(prEPSTALLS);   // read existing bits
			if (sc==1)
			{
				mask += bmSTLEP1OUT;			// Halt EP1OUT
				ep1stall = 1;
			}
			else
			{
				mask &= ~bmSTLEP1OUT;			// Unhalt EP1OUT
				ep1stall = 0;
				wreg(prCLRTOGS,bmCTGEP1OUT);		// clear the EP1 data toggle
			}
			wreg(prEPSTALLS,(mask | bmACKSTAT));		// Don't use wregAS for this--directly
writing the ACKSTAT bit
		}
		else if (SUD[wIndexL]==0x82)   // Endpoint 2
		{
			mask=rreg(prEPSTALLS);   // read existing bits
			if (sc==1)
```

```
                {
                        mask += bmSTLEP2IN;                     // Halt EP2IN
                        ep2stall = 1;
                }
                else
                {
                        mask &= ~bmSTLEP2IN;                    // Unhalt EP2IN
                        ep2stall = 0;
                        wreg(prCLRTOGS,bmCTGEP2IN);             // clear the EP2 data toggle
                }
                wreg(prEPSTALLS,(mask | bmACKSTAT));            // Don't use wregAS for this--directly
writing the ACKSTAT bit
        }
        else if (SUD[wIndexL]==0x83)   // Endpoint 3
        {
                mask=rreg(prEPSTALLS);   // read existing bits
                if (sc==1)
                {
                        mask += bmSTLEP3IN;                     // Halt EP3IN
                        ep3stall = 1;
                }
                else
                {
                        mask &= ~bmSTLEP3IN;                    // Unhalt EP3IN
                        ep3stall = 0;
                        wreg(prCLRTOGS,bmCTGEP3IN);             // clear the EP3 data toggle
                }
                wreg(prEPSTALLS,(mask | bmACKSTAT));            // Don't use wregAS for this--directly
writing the ACKSTAT bit
        }
        else STALL_EP0
    }
    else if ((SUD[bmRequestType]==0x00)       // dir=h->p, recipient = DEVICE
  &&  (SUD[wValueL]==0x01))          // wValueL is feature selector, 01 is Device_Remote_Wakeup
    {
        RWU_enabled = sc<<1;            // =2 for set, =0 for clear feature. The shift puts it in the get_status bit position.

        rregAS(prFNADDR);               // dummy read to set ACKSTAT
    }
    else STALL_EP0
}


/*******************************************************
void get_status(void)

Function to send the status of the device, an interface or
an endpoint in response to a GET STATUS standard request by
the computer.

*******************************************************/
void get_status(void)
{
    uchar testbyte;
    testbyte=SUD[bmRequestType];
    switch(testbyte)
    {
        case 0x80:                      // directed to DEVICE
                wreg(prEP0FIFO,(RWU_enabled+1));        // first byte is 000000rs where r=enabled for RWU and
s=self-powered.
                wreg(prEP0FIFO,0x00);                   // second byte is always 0
                wregAS(prEP0BC,2);                              // load byte count, arm the IN transfer, ACK
the status stage of the CTL transfer
                break;
        case 0x81:                      // directed to INTERFACE
                wreg(prEP0FIFO,0x00);                   // this one is easy--two zero bytes
                wreg(prEP0FIFO,0x00);
                wregAS(prEP0BC,2);                              // load byte count, arm the IN transfer, ACK
the status stage of the CTL transfer
                break;
```

```
			case 0x82:				// directed to ENDPOINT
				if(SUD[wIndexL]==0x81)			// We only reported ep3, so it's the only one the host can
stall IN3=83
				{
					wreg(prEP0FIFO,ep1stall);	// first byte is 0000000h where h is the halt (stall) bit
					wreg(prEP0FIFO,0x00);			// second byte is always 0
					wregAS(prEP0BC,2);				// load byte count, arm the IN
transfer, ACK the status stage of the CTL transfer
					break;
				}
				else if(SUD[wIndexL]==0x82)		// We only reported ep3, so it's the only one the host can
stall IN3=83
				{
					wreg(prEP0FIFO,ep2stall);	// first byte is 0000000h where h is the halt (stall) bit
					wreg(prEP0FIFO,0x00);			// second byte is always 0
					wregAS(prEP0BC,2);				// load byte count, arm the IN
transfer, ACK the status stage of the CTL transfer
					break;
				}
				else if(SUD[wIndexL]==0x83)		// We only reported ep3, so it's the only one the host can
stall IN3=83
				{
					wreg(prEP0FIFO,ep3stall);	// first byte is 0000000h where h is the halt (stall) bit
					wreg(prEP0FIFO,0x00);			// second byte is always 0
					wregAS(prEP0BC,2);			// load byte count, arm the IN transfer, ACK
the status stage of the CTL transfer
					break;
				}
				else STALL_EP0				// Host tried to stall an invalid endpoint (not 3)

			default:	STALL_EP0			// don't recognize the request
		}
}


/*******************************************************
void set_interface(void)

Function to set the interface designated by the computer
by a SET INTERFACE standard request

*******************************************************/
void set_interface(void)
{
		uchar dummy;
		if((SUD[wValueL]==0)			// wValueL=Alternate Setting index
		 &&(SUD[wIndexL]==0))			// wIndexL=Interface index
			dummy=rregAS(prFNADDR);	// dummy read to set the ACKSTAT bit
		else STALL_EP0
}


/*******************************************************
void get_interface(void)

Function to send the computer the current interface in
response to a GET INTERFACE standard request

*******************************************************/
void get_interface(void)
{
		if(SUD[wIndexL]==0)			// wIndexL=Interface index
		{
			wreg(prEP0FIFO,0);		// AS=0
			wregAS(prEP0BC,1);		// send one byte, ACKSTAT
		}
		else STALL_EP0
}
```

```
/******************************************************
void set_configuration(void)

Function to set the configuration designated by the computer
by a SET CONFIGURATION standard request

******************************************************/
void set_configuration(void)
{
	configval=SUD[wValueL];				// Store the config value
	if(configval != 0)					// If we are configured,
		SETBIT(prUSBIEN,bmSUSPIE);		// start looking for SUSPEND interrupts
	rregAS(prFNADDR);					// dummy read to set the ACKSTAT bit
}


/******************************************************
void get_configuration(void)

Function to send the computer the current configuration in
response to a GET CONFIGURATION standard request

******************************************************/
void get_configuration(void)
{
	wreg(prEP0FIFO,configval);			// Send the config value
	wregAS(prEP0BC,1);					// set the bytecount
}




/*------------- Low Level SPI Operations -----------------*/


/******************************************************
void wreg(uchar reg, uchar dat)

Function to write to a register on the MAX3421E without
setting the Acknowledgment bit

Inputs: reg - register number
		dat - data to be send

******************************************************/
void wreg(uchar reg, uchar dat)
{
	uchar csfrpage;
	uchar dummy;
	uchar prev_ea;

	prev_ea = EA;
	EA = 0;	// Disable all interrupts

	csfrpage = SFRPAGE;
	SFRPAGE = SPI0_PAGE;

	while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
	SS_LO
	SPI0DAT = reg | 0x02;
	while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
	usb_status = SPI0DAT;
	SPI0DAT = dat;
	while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
	SS_HI
	dummy = SPI0DAT;

	SFRPAGE = csfrpage;
```

```
		EA = prev_ea;		// Enable all Interrupts
}


/*****************************************************
void wregAS(uchar reg, uchar dat)

Function to write to a register on the MAX3421E with
setting the Acknowledgment bit

Inputs: reg - register number
			dat - data to be send

*****************************************************/
void wregAS(uchar reg, uchar dat)
{
		uchar csfrpage;
		uchar dummy;

		csfrpage = SFRPAGE;
		SFRPAGE = SPI0_PAGE;

		while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
		SS_LO
		SPI0DAT = reg | 0x03;
		while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
		usb_status = SPI0DAT;
		SPI0DAT = dat;
		while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
		SS_HI
		dummy = SPI0DAT;

		SFRPAGE = csfrpage;
}


/*****************************************************
uchar rreg(uchar reg)

Function to read a register on the MAX3421E without setting
the Acknowledgment bit

Inputs: reg - register value to be read

Output: value read from the register

*****************************************************/
uchar rreg(uchar reg)
{
		uchar csfrpage;
		uchar dat=0;
		uchar prev_ea;

		prev_ea = EA;
		EA = 0;	// Disable all interrupts

		csfrpage = SFRPAGE;
		SFRPAGE = SPI0_PAGE;

		while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
		SS_LO
		SPI0DAT = reg;
		while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
		usb_status = SPI0DAT;
		SPI0DAT = 0x00;					// Just need SCLK to continue clocking
		while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
		dat = SPI0DAT;
		SS_HI

		SFRPAGE = csfrpage;
```

```
		EA = prev_ea;		// Enable all Interrupts
		return (dat);
}


/*****************************************************
uchar rregAS(uchar reg)

Function to read a register on the MAX3421E with setting
the Acknowledgment bit

Inputs: reg - register value to be read

Output: value read from the register

*****************************************************/
uchar rregAS(uchar reg)
{
		uchar csfrpage;
		uchar dat;

		csfrpage = SFRPAGE;
		SFRPAGE = SPI0_PAGE;

		while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
		SS_LO
		SPI0DAT = reg | 0x01;
		while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
		usb_status = SPI0DAT;
		SPI0DAT = 0x00;				// Dummy value to keep SCLK clocking
		while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
		dat = SPI0DAT;
		SS_HI

		SFRPAGE = csfrpage;
		return (dat);
}


/*****************************************************
void readbytes(uchar reg, uchar N, uchar *p)

Function to read several bytes from a register on the
MAX3421E.  Used to read data from Endpoint buffers

Inputs: reg - register to be read
			N - number of bytes to be read
			*p - pointer to an array to be filled with the values read

*****************************************************/
void readbytes(uchar reg, uchar N, uchar *p)
{
		uchar csfrpage;
		uchar i;

		csfrpage = SFRPAGE;
		SFRPAGE = SPI0_PAGE;

		while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
		SS_LO
		SPI0DAT = reg;
		while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
		usb_status = SPI0DAT;
		for (i=0; i<N; i++)
		{
			SPI0DAT = 0x00;			// Dummy value to keep SCLK clocking
			while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
			*p = SPI0DAT;
			p++;
		}
```

```
        SS_HI

        SFRPAGE = csfrpage;
}


/*****************************************************
void writebytes(uchar reg, uchar N, uchar *p)

Function to write several bytes to a register on the
MAX3421E.  Used to write data to Endpoint buffers

Inputs: reg - register to be written
                N - number of bytes to be written
                *p - pointer to an array with the data to be written

*****************************************************/
void writebytes(uchar reg, uchar N, uchar *p)
{
        uchar csfrpage;
        uchar i,dat;

        csfrpage = SFRPAGE;
        SFRPAGE = SPI0_PAGE;

        while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
        SS_LO
        SPI0DAT = reg | 0x02;
        while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
        usb_status = SPI0DAT;
        for (i=0; i<N; i++)
        {
                dat = *p;
                SPI0DAT = dat;
                while (((SPI0CN & bmTXBMT) == 0x00) || ((SPI0CFG & 0x80) == 0x80)); // Wait for SPI available
                usb_status = SPI0DAT;
                p++;
        }
        SS_HI

        SFRPAGE = csfrpage;
}
```

File: usb.h

```
#ifndef USB_H
#define USB_H

#include "c8051F120.h"
#include "max.h"
#include "pindef.h"
#include "data_types.h"



#define SS_LO USB_NSS=0;
#define SS_HI USB_NSS=1;

#define bmSPIBSY 0x80
#define bmTXBMT     0x02

#define SR_CONTROL_IN        0xC1
#define SR_CONTROL_OUT     0xC0

#define OSC_TIMEOUT            1000

// Control Values for the Microcontroller
#define CV_MC_LCD


/******************************
Some of this code was adapted from code written by Maxim for the MAX3420E
******************************/

// Possible Control Values for USB requests directed at Microcontroller
#define SEND_LCD   0x40
#define GET_JUNK   0xC0

// MAX chip setup/configuration
uchar init_usb(void);                          // Initialize MAX chip
void enable_irqs(void);                                // Enable Interrupt Requests
void rwu_usb(void);                                    // Remote Wake-Up the host if possible
uchar reset_usb(void);                         // Reset the MAX chip

// Endpoint transfer functions
void do_SUD(void);
void do_OUT0(void);
void do_IN0(void);
void do_OUT1(void);
void do_IN2(void);
void do_IN3(void);

// Higher Level Transfer Functions
void Transfer_USB_Bulk(void);
void control_in(uchar control, uchar len);
void control_out(uchar control);

// USB functions
void std_request(void);
void class_request(void);
void vendor_request(void);
void send_descriptor(void);
void feature(uchar sc);
void get_status(void);
void set_interface(void);
void get_interface(void);
void set_configuration(void);
void get_configuration(void);

void wreg(uchar reg, uchar dat);
void wregAS(uchar reg, uchar dat);
uchar rreg(uchar reg);
uchar rregAS(uchar reg);
```

```
void readbytes(uchar reg, uchar N, uchar *p);
void writebytes(uchar reg, uchar N, uchar *p);


extern uchar usb_status;            // Status byte read in as first byte of SPI transfer
        /*      Status byte composition
                Bit0: IN0BAVIRQ
                Bit1: OUT0DAVIRQ
                Bit2: OUT1DAVIRQ
                Bit3: IN2BAVIRQ
                Bit4: IN3BAVIRQ
                Bit5: SUDAVIRQ
                Bit6: URESIRQ
                Bit7: SUSPIRQ
        */

extern uchar SUD[8];                // Buffer for Set-Up Data

extern uchar in0buf[64];            // Buffer for EP0-IN
extern uchar in0len;                // Length of data for EP0-IN

extern uchar out0buf[64];           // Buffer for EP0-OUT
extern uchar out0len;               // Length of data available from EP0-OUT

extern uchar out1buf[64];           // Buffer for EP1-OUT
extern uchar out1len;               // Length of data available from EP1-OUT

extern uchar in2buf[64];            // Buffer for EP2-IN
extern uchar in2len;                // Length of data for EP2-IN

extern uchar in3buf[64];            // Buffer for EP3-IN
extern uchar in3len;                // Length of data for EP3-IN

extern uchar configval;             // Current Configuration Value being used
extern uchar suspend;               // Whether the device is in SUSPEND mode or not
extern uchar RWU_enabled;           // Whether Remote Wake-Up is enabled or not
extern uchar ep1stall;              // Whether EP1 is stalled or not
extern uchar ep2stall;              // Whether EP2 is stalled or not
extern uchar ep3stall;              // Whether EP3 is stalled or not
extern uchar osc_ready;             // Whether the MAX on-chip oscillator is stable

#endif
```

File: max.h

```
#ifndef MAX_H
#define MAX_H
// MAX3421E.h



// Macros
#define SETBIT(reg,val) wreg(reg,(rreg(reg)|val));
#define CLRBIT(reg,val) wreg(reg,(rreg(reg)&~val));
#define STALL_EP0 wreg(prEPSTALLS,0x23);          // Set all three EP0 stall bits--data stage IN/OUT and status stage

// ******** END OF BUG FIX**********


// MAX3421E Registers - Peripheral Mode
#define prEP0FIFO    0<<3
#define prEP1OUTFIFO 1<<3
#define prEP2INFIFO  2<<3
#define prEP3INFIFO  3<<3
#define prSUDFIFO    4<<3
#define prEP0BC      5<<3
#define prEP1OUTBC   6<<3
#define prEP2INBC    7<<3
#define prEP3INBC    8<<3
#define prEPSTALLS   9<<3
#define prCLRTOGS    10<<3
#define prEPIRQ      11<<3
#define prEPIEN      12<<3
#define prUSBIRQ     13<<3
#define prUSBIEN     14<<3
#define prUSBCTL     15<<3
#define prCPUCTL     16<<3
#define prPINCTL     17<<3
#define prREVISION   18<<3
#define prFNADDR     19<<3
#define prIOPINS1    20<<3
#define prIOPINS2    21<<3
#define prGPINIRQ    22<<3
#define prGPINIEN    23<<3
#define prGPINPOL    24<<3
#define prMODE                   27<<3

// MAX3421E Registers - Host Mode
#define hrRCVFIFO    1<<3
#define hrSNDFIFO    2<<3
#define hrSUDFIFO    4<<3
#define hrRCVBC                  6<<3
#define hrSNDBC                  7<<3
#define hrUSBIRQ     13<<3
#define hrUSBIEN     14<<3
#define hrUSBCTL     15<<3
#define hrCPUCTL     16<<3
#define hrPINCTL     17<<3
#define hrREVISION   18<<3
#define hrIOPINS1    20<<3
#define hrIOPINS2    21<<3
#define hrGPINIRQ    22<<3
#define hrGPINIEN    23<<3
#define hrGPINPOL    24<<3
#define hrHIRQ                   25<<3
#define hrHIEN                   26<<3
#define hrMODE                   27<<3
#define hrPERADDR  28<<3
#define hrHCTL                   29<<3
#define hrHXFR                   30<<3
#define hrHRSL                   31<<3

// MAX3421E bit masks for status byte
```

```
#define bmSTATUS_SUSPIRQ  0x80
#define bmSTATUS_URESIRQ  0x40
#define bmSTATUS_SUDAVIRQ           0x20
#define bmSTATUS_IN3BAVIRQ          0x10
#define bmSTATUS_IN2BAVIRQ          0x08
#define bmSTATUS_OUT1DAVIRQ         0x04
#define bmSTATUS_OUT0DAVIRQ         0x02
#define bmSTATUS_IN0BAVIRQ          0x01

// MAX3420E bit masks for register bits
// R9 EPSTALLS Register
#define bmACKSTAT   0x40
#define bmSTLSTAT   0x20
#define bmSTLEP3IN  0x10
#define bmSTLEP2IN  0x08
#define bmSTLEP1OUT 0x04
#define bmSTLEP0OUT 0x02
#define bmSTLEP0IN  0x01

// R10 CLRTOGS Register
#define bmEP3DISAB  0x80
#define bmEP2DISAB  0x40
#define bmEP1DISAB  0x20
#define bmCTGEP3IN  0x10
#define bmCTGEP2IN  0x08
#define bmCTGEP1OUT 0x04

// R11 EPIRQ register bits
#define bmSUDAVIRQ  0x20
#define bmIN3BAVIRQ 0x10
#define bmIN2BAVIRQ 0x08
#define bmOUT1DAVIRQ 0x04
#define bmOUT0DAVIRQ 0x02
#define bmIN0BAVIRQ 0x01

// R12 EPIEN register bits
#define bmSUDAVIE   0x20
#define bmIN3BAVIE  0x10
#define bmIN2BAVIE  0x08
#define bmOUT1DAVIE 0x04
#define bmOUT0DAVIE 0x02
#define bmIN0BAVIE  0x01

// R13 USBIRQ register bits
#define bmURESDNIRQ 0x80
#define bmVBUSIRQ   0x40
#define bmNOVBUSIRQ 0x20
#define bmSUSPIRQ   0x10
#define bmURESIRQ   0x08
#define bmBUSACTIRQ 0x04
#define bmRWUDNIRQ  0x02
#define bmOSCOKIRQ  0x01

// R14 USBIEN register bits
#define bmURESDNIE  0x80
#define bmVBUSIE    0x40
#define bmNOVBUSIE  0x20
#define bmSUSPIE    0x10
#define bmURESIE    0x08
#define bmBUSACTIE  0x04
#define bmRWUDNIE   0x02
#define bmOSCOKIE   0x01

// R15 USBCTL Register
#define bmHOSCSTEN  0x80
#define bmVBGATE    0x40
#define bmCHIPRES   0x20
#define bmPWRDOWN   0x10
#define bmCONNECT   0x08
#define bmSIGRWU    0x04
```

```
// R16 CPUCTL Register
#define bmIE        0x01

// R17 PINCTL Register
#define bmFDUPSPI   0x10
#define bmINTLEVEL  0x08
#define bmPOSINT    0x04
#define bmGPOB      0x02
#define     bmGPOA      0x01

//
// GPX[B:A] settings (PINCTL register)
#define gpxOPERATE  0x00
#define gpxVBDETECT 0x01
#define gpxBUSACT   0x02
#define gpxSOF      0x03

// ************************
// Standard USB Requests
#define SR_GET_STATUS                   0x00        // Get Status
#define SR_CLEAR_FEATURE 0x01         // Clear Feature
#define SR_RESERVED                                 0x02        // Reserved
#define SR_SET_FEATURE                  0x03        // Set Feature
#define SR_SET_ADDRESS                  0x05        // Set Address
#define SR_GET_DESCRIPTOR               0x06        // Get Descriptor
#define SR_SET_DESCRIPTOR 0x07        // Set Descriptor
#define SR_GET_CONFIGURATION            0x08        // Get Configuration
#define SR_SET_CONFIGURATION            0x09        // Set Configuration
#define SR_GET_INTERFACE  0x0a        // Get Interface
#define SR_SET_INTERFACE  0x0b        // Set Interface

// Get Descriptor codes
#define GD_DEVICE           0x01        // Get device descriptor: Device
#define GD_CONFIGURATION 0x02        // Get device descriptor: Configuration
#define GD_STRING           0x03        // Get device descriptor: String
#define GD_HID              0x21        // Get descriptor: HID
#define GD_REPORT                0x22   // Get descriptor: Report

// SETUP packet offsets
#define bmRequestType   0
#define     bRequest            1
#define wValueL                         2
#define wValueH                         3
#define wIndexL                         4
#define wIndexH                         5
#define wLengthL            6
#define wLengthH            7

// HID bRequest values
#define GET_REPORT                      1
#define GET_IDLE            2
#define GET_PROTOCOL         3
#define SET_REPORT                      9
#define SET_IDLE            0x0A
#define SET_PROTOCOL         0x0B
#define INPUT_REPORT        1


#endif
```

File: uart.c

```
#include "c8051F120.h"
#include "uart.h"
#include "usb.h"
#include "data_types.h"
#include "lcd.h"

#define UART_LEN  64

uchar uart0r_buf[UART_LEN];  // Used for data received from the Receiver DSP
uchar uart0t_buf[UART_LEN];  // Used for data to send to the Receiver DSP
uchar uart0t_len=0;
uchar uart0_busy=0;  // Used to designate when the UART is transferring data
uchar uart0_data_r_len = 0;        // Designates when Transfer_DSP needs to be run

uchar uart1r_buf[UART_LEN];  // Used for data received from the Transmitter DSP
uchar uart1t_buf[UART_LEN];  // Used for data to send to the Transmitter DSP
uchar uart1t_len=0;
uchar uart1_busy=0;  // Used to designate when the UART is transferring data
uchar uart1_data_r_len = 0;        // Designates when Transfer_DSP needs to be run

extern uchar in0buf[64];           // Buffer for EP0-IN
extern uchar in0len;               // Length of data for EP0-IN
extern uchar in2buf[64];           // Buffer for EP2-IN
extern uchar in2len;               // Length of data for EP2-IN
extern uchar in3buf[64];           // Buffer for EP3-IN
extern uchar in3len;               // Length of data for EP3-IN
extern uchar num_interrupts;


/*------------------------------------------------------------------------
Format of Data transmission between Microcontroller and DSP is:

Byte 0: Control byte designating destination and purpose of transmission
Byte 1: Total length of transmission.  The maximum packet length is 64.
Bytes 2-63: Data bytes
------------------------------------------------------------------------*/


/*****************************************************
void UART0_ISR(void) interrupt 4

ISR for UART communication with the Receiver DSP

*****************************************************/
void UART0_ISR(void) interrupt 4            // Receiver DSP
{
        static uchar uart0r_end = 0;
        static uchar uart0r_len = 0;
        static uchar uart0t_end = 0;
        static uchar dummy = 0;
        static uchar ignore_rec = 0;
        static uchar ignore_len = 0;
        uchar csfrpage;

        num_interrupts++;


        csfrpage = SFRPAGE;
        SFRPAGE = UART0_PAGE;

        if (RI0)
        {
                RI0 = 0;
                if ((uart0_data_r_len == 0) && (ignore_rec == 0))        // if Micro ready to receive more data
                {
                        uart0r_buf[uart0r_end] = SBUF0;
                        if (uart0r_end == UART_LEN_BYTE)
```

```
                        {
                                uart0r_len = uart0r_buf[uart0r_end];
                        }
                        uart0r_end = (++uart0r_end) % UART_LEN;
                        if ((uart0r_len > 0) && (uart0r_end >= uart0r_len))
                        {
                                uart0_data_r_len = uart0r_len;     //
                                uart0r_end = 0;
                                uart0r_len = 0;
                        }
                }
                else
                {
                        dummy = SBUF0;
                        if (ignore_rec == UART_LEN_BYTE)
                        {
                                ignore_len = dummy;
                        }
                        if (ignore_rec >= ignore_len)
                        {
                                ignore_rec = 0;
                                ignore_len = 0;
                        }
                        ignore_rec++;

                }
        }
        if (TI0)
        {
                TI0 = 0;
                uart0_busy = 0;
                if (uart0t_end < uart0t_len)         // Still data in the buffer to send
                {
                        SBUF0 = uart0t_buf[uart0t_end];
                        uart0t_end++;
                        uart0_busy = 1;
                }
                else                        // finished transmitting data in buffer
                {
                        uart0t_end = 0;
                        uart0t_len = 0;
                }
        }

        SFRPAGE = csfrpage;

}


/*****************************************************
void UART1_ISR(void) interrupt 20

ISR for UART communication with the Transmitter DSP

*****************************************************/
void UART1_ISR(void) interrupt 20          // Transmitter DSP
{
        static uchar uart1r_end = 0;
        static uchar uart1r_len = 0;
        static uchar uart1t_end = 0;
        static uchar dummy,ignore_rec,ignore_len;
        uchar csfrpage;

        num_interrupts++;


        csfrpage = SFRPAGE;
        SFRPAGE = UART1_PAGE;

        if (RI1)
```

```
	{
		RI1 = 0;
		if ((uart1_data_r_len == 0) && (ignore_rec == 0))	// if Micro ready to receive more data
		{
			uart1r_buf[uart1r_end] = SBUF1;
			if (uart1r_end == UART_LEN_BYTE)
			{
				uart1r_len = uart1r_buf[uart1r_end];
			}
			uart1r_end = (++uart1r_end) % UART_LEN;
			if (uart1r_end >= uart1r_len)
			{
				uart1_data_r_len = uart1r_len;	//
				uart1r_end = 0;
				uart1r_len = 0;
			}
		}
		else
		{
			dummy = SBUF1;
			if (ignore_rec == UART_LEN_BYTE)
			{
				ignore_len = dummy;
			}
			if (ignore_rec >= ignore_len)
			{
				ignore_rec = 0;
				ignore_len = 0;
			}
			ignore_rec++;

		}
	}
	if (TI1)
	{
		TI1 = 0;
		uart1_busy = 0;
		if (uart1t_end < uart1t_len)		// Still data in the buffer to send
		{
			SBUF1 = uart1t_buf[uart1t_end];
			uart1t_end++;
			uart1_busy = 1;
		}
		else				// finished transmitting data in buffer
		{
			uart1t_end = 0;
			uart1t_len = 0;
		}
	}

	SFRPAGE = csfrpage;
}

/*****************************************************
void Transfer_DSP(uchar target, uchar len)

Transfers data received from a DSP to the computer.

Inputs: target - determines which DSP was the source of
				the data to be transfered
		len - length of the data to be transfered

*****************************************************/
void Transfer_DSP(uchar target, uchar len)
{
	uchar control;
	uchar i;
	uchar * source;
	uchar string[20];
```

```
switch (target)
{
	case 0:
		source = uart0r_buf;
		break;
	case 1:
		source = uart1r_buf;
		break;
}
control = source[0];
switch (control & 0x80)
{
	case 0x00:		// Directed at Microcontroller
		switch (control & 0x9F)			// Clear EP bits because they are don't cares
		{
			case DSP_SEND_LCD:
				for (i=2; (i<len) && (i<22); i++)
				{
					string[i-2] = source[i];
				}
				switch (target)
				{
					case 0:
						i = LCD_LINE3;
						uart0_data_r_len = 0;
						break;
					case 1:
						i = LCD_LINE4;
						uart1_data_r_len = 0;
						break;
				}
				len = source[1] - 2;
				LCD_write(i,string,len);
				break;
		}

	case 0x80:		// Directed at Computer
		control = (source[0] & 0xC0)>>6;		// Isolate bits 7 and 6 which determine the endpoint to
send on
		switch (control)
		{
		case 0:	// IN0
			while (in0len!=0);	// wait for IN0 to be available
			for (i=0; i<len; i++)
			{
				in0buf[i] = source[i];
			}
			in0len = len;
			do_IN0();
			break;
		case 2:	// IN2
			while (in2len!=0);	// wait for IN2 to be available
			for (i=0; i<len; i++)
			{
				in2buf[i] = source[i];
			}
			in2len = len;
			do_IN2();
			break;
		case 3:	// IN3
			while (in3len!=0);	// wait for IN3 to be available
			for (i=0; i<len; i++)
			{
				in3buf[i] = source[i];
			}
			in3len = len;
			do_IN3();
			break;
		}
		switch (target)
```

```
                {
                        case 0:
                                uart0_data_r_len = 0;
                                break;
                        case 1:
                                uart1_data_r_len = 0;
                                break;
                }

                break;
        }

}
```

File: uart.h

```
#ifndef UART_H
#define UART_H

#include "data_types.h"


/* -------- Possible control words from DSP directed at micro -------- */
#define DSP_SEND_LCD          0x01

#define UART_CNTL_BYTE     0
#define UART_LEN_BYTE       1


void Transfer_DSP(uchar target, uchar len);


extern uchar uart0r_buf[64];        // Used for data received from the Receiver DSP
extern uchar uart0t_buf[64];        // Used for data to send to the Receiver DSP
extern uchar uart0t_len;
extern uchar uart0_busy;             // Used to designate when the UART is transferring data
extern uchar uart0_data_r_len;    // Designates when Transfer_DSP needs to be run

extern uchar uart1r_buf[64];        // Used for data received from the Transmitter DSP
extern uchar uart1t_buf[64];        // Used for data to send to the Transmitter DSP
extern uchar uart1t_len;
extern uchar uart1_busy;             // Used to designate when the UART is transferring data
extern uchar uart1_data_r_len;    // Designates when Transfer_DSP needs to be run




#endif
```

File: data_types.h

```
#ifndef DATA_TYPES_H
#define DATA_TYPES_H

#define uchar unsigned char
#define uint unsigned int


#endif
```

File: pindef.h

```
#ifndef PINDEF_H
#define PINDEF_H

#define TX0             P0_0    /* Output */
#define RX0             P0_1    /* Input  */
#define SCK             P0_2    /* Output */
#define MISO        P0_3    /* Input  */
#define MOSI        P0_4    /* Output */
#define TX1             P0_5    /* Output */
#define RX1             P0_6    /* Input  */
#define INT0        P0_7    /* Input  */

#define LCD_POWER               P1_0    /* Output */
#define DSPT_PWR_RST    P1_2    /* Input  */
#define DSPR_PWR_RST    P1_3    /* Input  */

/*
Port 2 is defined at LCD_DATA in lcd.h
*/

#define USB_GPX         P3_0    /* Input  */
#define USB_NSS         P1_7    /* Output */
#define USB_RST         P3_2    /* Output */
#define USB_PWR_FLG     P3_3    /* Input  */
#define USB_PWR_ON      P3_4    /* Output */

/*
P3.5 - P3.7 are defined in lcd.h
*/


#endif
```

APPENDIX H.
SOFTWARE DEFINED RADIO PARTS LIST

Software Defined Radio Parts List

Manufacturer	Manufacturer Part Number	Description	Qty. to Buy	Total Price
Abracon Corporation	ABM3B-12.000MHZ-10-1-U-T	CRYSTAL 12.0000MHZ 10PF SMD	1	$1.93
Analog Devices	AD6644ASTZ-65	IC ADC 14BIT 65MSPS CMOS 52-LQFP	2	$118.42
Texas Instruments	SN74AHC1G08DBVR	IC SGL 2IN POS-AND GATE SOT23-5	2	$0.78
Mini-Circuits	TC1-1-13M+	Transformer, RF, RoHS	5	$49.75
Mini-Circuits	T4-1-KK81+	Transformer, RF, RoHS	2	$6.50
Tyco Electronics / AMP	5227161-6	BNC PC Board/Panel Mount Connectors R.A. JACK 50 OHM	2	$3.70
Tyco Electronics / Alcoswitch	TP11CGPC004	SWITCH PB SPST MOM NO .4VA BLK	3	$7.62
Panasonic - ECG	EEV-FK1H331Q	CAP 330UF 50V ELECT FK SMD	5	$7.08
Kemet	C0603C103K5RACTU	CAP 10000PF 50V CERAMIC X7R 0603	80	$1.76
Kemet	C0603C104K3RACTU	CAP .1UF 25V CERAMIC X7R 0603	130	$9.05
Kemet	C0603C474K4RACTU	CAP CERM .47UF 16V X7R 0603	10	$5.50
Kemet	C0603C225K9PACTU	CAP CERAMIC 2.2UF 6.3V X5R 0603	10	$4.40
Kemet	C0603C102K5RACTU	CAP 1000PF 50V CERAMICX7R 0603	20	$0.44
Kemet	C0603C101J5GACTU	CAP CERAMIC 100PF 50V NP0 0603	40	$10.40
Taiyo Yuden	JMK107BJ106MA-T	CAP CER 10UF 6.3V X5R 0603	30	$39.60
Taiyo Yuden	EMK212BJ106KG-T	CAP CER 10UF 16V X5R 0805	10	$6.60
Kemet	C0603C105K8PACTU	CAP CERAMIC 1.0UF 10V X5R 0603	100	$11.70
Kemet	C0603C180J5GACTU	CAP CERAMIC 18PF 50V NP0 0603	10	$0.36
Kemet	C0603C472K5RACTU	CAP 4700PF 50V CERAMIC X7R 0603	10	$0.28
Kemet	C0603C475K9PACTU	CAP CER 4.7UF 6.3V X5R 0603	10	$6.05
Kemet	C0603C470J5GACTU	CAP CERAMIC 47PF 50V NP0 0603	10	$0.39
Kemet	C0805C226M9PACTU	CAP CERAMIC 22UF 6.3V X5R 0805	10	$27.50
Kemet	B45197A3107K509	CAP TANT 100UF 16V 10% LOESR SMD	3	$4.92
Kemet	C1210C105K5RACTU	CAP 1.0UF 50V CERAMIC X7R 1210	10	$6.88
Kemet	C1210C225K3RACTU	CAP 2.2UF 25V CERAMIC X7R 1210	10	$16.50
CTS-Frequency Controls	CB3LV-3C-50M0000-T	OSC CLOCK 50.000 MHZ 3.3V SMD	1	$3.75
ECS Inc	ECS-8FMX-200-TR	OSCILLATOR 20.000MHZ SMD	4	$11.88
Analog Devices	AD5060BRJZ-2REEL7	IC DAC 16BIT 2.7-5.5V SOT23-8	2	$25.00
Analog Devices	AD9774AS	IC DAC 14BIT 32MSPS 44-MQFP	2	$75.66
CUI Inc	PJ-102AH	CONN PWR JACK 2.1X5.5MM HIGH CUR	1	$0.42
Analog Devices	AD6620AS	IC DGTL RCVR SIGNAL PROC 80-PQFP	2	$51.60
Texas Instruments	TMS320VC5502PGF300	IC FXD-PNT DSP 600 MIPS 176-LQFP	2	$29.34
TDK Corporation	MEM2012T25R0	FILTER 3-TERM 25MHZ 150MA SMD	4	$1.04
Xilinx	XC3S50-4TQG144C	50000 SYSTEM GATE 1.2 VOLT FPGA	1	$13.48

Xilinx	XCF01SVOG20C	IC PROM SRL FOR 1M GATE 20-TSSOP	1	$4.28
Analog Devices	ADF4153BRUZ	IC SYNTH PLL RF F-N FREQ 16TSSOP	2	$10.26
Wickmann USA	3951500044	FUSE FAST 5.00A SHORT UL TE5	2	$1.82
Wickmann USA	3951100044	FUSE FAST 1.00A SHORT UL TE5	1	$0.91
Wickmann USA	5620000100	FUSEHOLDER TE5 MACHINE CONTACTS	3	$3.48
Texas Instruments	SN74LVC1G14DBVR	IC SCHMITT TRIG INV SOT-23-5	2	$0.80
Analog Devices	AD8347ARUZ	IC QUADRATURE DEMOD 28-TSSOP	1	$8.96
Analog Devices	AD8346ARUZ	IC QUADRATURE MOD .8GHZ 16-TSSOP	1	$7.94
Microchip	24LC512-I/SMG	Serial EEPROM - 512k 64kx8 - 2.5V Lead Free Package	2	$6.10
TDK Corporation	MLF2012DR12K	INDUCTOR MULTILAYER 0.12UH 2012	4	$1.64
Bourns	SDR1806-2R2ML	Inductors 2.2uH	6	$5.10
Lumex	SML-LX0805SIC-TR	LED 636NM SUPER RED 0805 SMD	10	$3.00
Analog Devices	AD8354ACPZ-REEL7	IC RF GAIN BLOCK 8-LFCSP	1	$1.44
Silicon Laboratories Inc	C8051F123-GQ	IC 8051 MCU FLASH 128K 64TQFP	1	$21.77
Molex / Waldom Electronics Corp	87831-1420	CONN HEADER 14POS 2MM VERT GOLD	1	$1.70
Texas Instruments	SN74AHC1G00DBVR	IC GATE NAND DUAL W/SCHMTR 8SSOP	2	$0.78
Texas Instruments	SN74AHC1G02DBVR	IC SGL 2IN POS-NOR GATE SOT23-5	1	$0.39
Analog Devices	OP184FSZ	IC OPAMP PREC SGL-SUP R-R 8SOIC	2	$6.40
Analog Devices	AD8031ARTZ-REEL7	IC OPAMP RRIO VFB 80MHZ SOT-23-5	4	$10.08
Analog Devices	AD8138ARZ	IC DIFF ADC DVR 8-SOIC	4	$29.28
Texas Instruments	OPA2674I-14D	Amplifiers - High Speed Greater than Equal to 50MHz Dual Wideband, High Output Current	1	$3.85
Mill-Max Manufacturing Corporation	800-10-064-10-001000	SIP HEADER 64 POS STRAIGHT PCB	1	$7.35
Mill-Max Manufacturing Corporation	802-10-064-10-001000	CONN HDR 64 PIN DUAL ROW	1	$8.48
Bourns	3362R-1-203LF	POT 20K OHM 1/4" SQ CERM SL ST	2	$1.66
Yageo Corporation	9C06031A14R7FKHFT	RES 14.7 OHM 1/10W 1% 0603 SMD	10	$0.77
Yageo Corporation	RC0603FR-0724R9L	RES 24.9 OHM 1/10W 1% 0603 SMD	10	$0.77
Yageo Corporation	RC0603FR-0751RL	RES 51.0 OHM 1/10W 1% 0603 SMD	20	$1.54
Yageo Corporation	RC0603FR-0768RL	RES 68.0 OHM 1/10W 1% 0603 SMD	20	$1.54
Yageo Corporation	RC0603FR-07100RL	RES 100 OHM 1/10W 1% 0603 SMD	10	$0.77
Yageo Corporation	RC0603FR-07150RL	RES 150 OHM 1/10W 1% 0603 SMD	10	$0.77
Yageo Corporation	RC0603FR-07200RL	RES 200 OHM 1/10W 1% 0603 SMD	10	$0.77
Yageo Corporation	RC0603FR-07226RL	RES 226 OHM 1/10W 1% 0603 SMD	10	$0.77
Yageo Corporation	RC0603FR-07330RL	RES 330 OHM 1/10W 1% 0603 SMD	10	$0.77
Yageo Corporation	RC0603FR-07499RL	RES 499 OHM 1/10W 1% 0603 SMD	20	$1.54
Yageo Corporation	RC0603FR-07100KL	RES 100K OHM 1/10W 1% 0603 SMD	10	$0.77

Yageo Corporation	RC0603FR-0710KL	RES 10.0K OHM 1/10W 1% 0603 SMD	60	$2.48
Yageo Corporation	RC0603FR-071KL	RES 1.00K OHM 1/10W 1% 0603 SMD	20	$1.54
Yageo Corporation	RC0603FR-0721KL	RES 21.0K OHM 1/10W 1% 0603 SMD	10	$0.77
Yageo Corporation	RC0603FR-072KL	RES 2.00K OHM 1/10W 1% 0603 SMD	10	$0.77
Yageo Corporation	RC0603FR-073K83L	RES 3.83K OHM 1/10W 1% 0603 SMD	10	$0.77
Yageo Corporation	RC0603FR-074K75L	RES 4.75K OHM 1/10W 1% 0603 SMD	10	$0.77
Yageo Corporation	RC0603FR-074K7L	RES 4.70K OHM 1/10W 1% 0603 SMD	10	$0.77
Yageo Corporation	RC0603FR-074K02L	RES 4.02K OHM 1/10W 1% 0603 SMD	10	$0.77
Yageo Corporation	RC0603FR-075K1L	RES 5.10K OHM 1/10W 1% 0603 SMD	10	$0.77
Yageo Corporation	RC0603FR-075K6L	RES 5.60K OHM 1/10W 1% 0603 SMD	10	$0.77
Yageo Corporation	RC0603FR-0749K9L	RES 49.9K OHM 1/10W 1% 0603 SMD	10	$0.77
Yageo Corporation	RC0603FR-0778K7L	RES 78.7K OHM 1/10W 1% 0603 SMD	10	$0.77
Fairchild Semiconductor	BAT54S	Diodes - Schottky 30V 200mA	2	$0.14
Johnson / Emerson	142-0711-201	SMA Connectors SMT VERTICAL JACK 50ohm, 18GHz, GOLD	6	$30.96
NKK	GW22RBP	Rocker/Paddle Switches DPDT ON-ON WHT ROCKER ACTUATOR	2	$6.50
FCI	73725-0110BLF	CONN RCPT USB TYPE A FLAG PCB	1	$0.99
Molex / Waldom Electronics Corp	67068-8000	CONN USB RT ANG RECPT TYPE B BLK	1	$1.92
Maxim	MAX3421EEHJ+	USB Peripheral/Host Controller with SPI™ Interface	1	$5.62
Maxim	MAX4789EUK+	200mA Current Limit Switch	1	$0.00
Crystek Crystals	CVCO55BE-1000-2000	Crystals High Performance VCOs 1000 - 2000 MHz	2	$59.00
Analog Devices	ADL5330ACPZ-R2	IC AMP 10MHZ-3GHZ VGA RF 24LFCSP	2	$18.30
Analog Devices	ADR318ARJ-R2	IC REF PREC LOW DRIFT SOT23-5	2	$5.62
Texas Instruments	REF2925AIDBZT	Voltage References 2.5V 100ppm/DegC, 50uA, SOT23-3 Series	1	$1.65
Texas Instruments	TPS726126DCQ	IC LDO REG LINEAR 1.26V 1A 6-SOT	2	$4.96
Fairchild Semiconductor	FAN1112SX	Linear Regulators Low Dropout Linear	1	$0.99
Texas Instruments	TL750L10CD	IC 10V 150MA LDO REG 8-SOIC	2	$1.36
Analog Devices	ADP3339AKCZ-2.5-R7	IC REG LDO 1.5A 2.5V SOT223-3	1	$3.94
Texas Instruments	PTN78020WAH	REG SW 6A HORZ T/H	2	$45.00
Cincon	TR45A24-01A03	AC Adapters 24VDC 45W DESKTOP	1	$38.62
Kobiconn	173-63101-E	AC Power Cords 7.5' 3WIRE 18AWG BLK	1	$3.47
?	?	Broadband Antenna Frequency Range 100MHZ - 2GHz	2	$98.00
CUI Inc	PP3-002A	CONN 2.1MM FEMALE PLUG 5.5MM OUT	1	$0.93

CUI Inc	PJ-005A	CONN 2.1MM JACK W/HARD PNL MNT	1	$1.83
Lumex Opto/Components Inc	LCM-S02004DSF	LCD MODULE 20X4 CHARACTER W/LED	1	$31.50
Lumex Opto/Components Inc	CLS-TC11A12252G	SWITCH TOGGLE ILLUM SPST 20A GRN	1	$3.21
		Total:		$1,094.76

APPENDIX I.

SOFTWARE DEFINED RADIO PICTURES

Software Defined Radio with Antennas

Software Defined Radio with LCD Display

BIBLIOGRAPHY

[1] Pratt, Jason and Kosbar, Kurt, "A Hardware Platform for Cognitive Radio," *Proceedings of the 43rd Annual International Telemetering Conference (ITC 2007)*, Las Vegas, 2007.

[2] Rubenstein, Roy, "Radios Get Smart," *IEEE Spectrum*, IEEE, Volume 44, Number 2, February, 2007, pp. 47 – 50.

[3] Panagos, Adam and Kosbar, Kurt, "Cognitive Ad-hoc Wireless Networks," *Proceedings of the 42nd Annual International Telemetering Conference (ITC 2006)*, San Diego, CA, 2006.

[4] Sahin, Mastafa E. and Arsian, Huseyin, "System Design for Cognitive Radio Communications," *Proceedings of the 1st International Conference on Cognitive Radio Oriented Wireless Networks and Communications, 2006*, Mykonos Island, Greece, 2006.

[5] Jovicic, Aleksandar and Viswanath, Pramod, "Cognitive Radio: An Information-Theoretic Perspective," *Proceedings of 2006 IEEE International Symposium on Information Theory*, Seattle, Washington, 2006.

[6] Godard, Loig, Moy, Christophe and Palicot, Jacques, "From a Configuration Management to a Cognitive Radio Management of SDR Systems," *Proceedings of the 1st International Conference on Cognitive Radio Oriented Wireless Networks and Communications, 2006*, Mykonos Island, Greece, 2006.

[7] http://enterprise.spawar.navy.mil/getfile.cfm?contentId=1488&type=R. JTRS Overview for CCEB Spectrum Task Force, June 2007.

[8] http://www.boeing.com/defense-space/ic/jtrs/docs/JTRS_GMR_overview.pdf. Joint Tactical Radio Systems for Ground Mobile Radios, June 2007.

[9] http://www.ettus.com/downloads/usrp_v4.pdf. Universal Software Radio Peripheral, June 2007.

[10] http://www.optoelectronics.com/accessories.htm. Optoelectronics™ Accessories, July 2007.

[11] "AD8354 RF Gain Block Datasheet: Rev. C," Analog Devices™: http://www.analog.com/en/, July 2007.

[12] "ADL5530 Variable Gain Amplifier Datasheet: Rev. A," Analog Devices™: http://www.analog.com/en/, July 2007.

[13] "AD8347 Direct Conversion Quadrature Demodulator Datasheet: Rev. A," Analog Devices™: http://www.analog.com/en/, July 2007.

[14] "ADF4153 Fractional-N Frequency Synthesizer Datasheet: Rev. B," Analog Devices™: http://www.analog.com/en/, July 2007.

[15] "CVCO55BE-1000-2000 Voltage Controlled Oscillator Datasheet," Crystek Crystals Corporation™: http://www.crystek.com/default.htm, July 2007.

[16] "MEM2012T25R0 Three-Terminal Filters Datasheet," TDK Components™: http://www.tdk-components.de/en/home/index.php, July 2007.

[17] "AD6644 Analog to Digital Converter Datasheet: Rev. C," Analog Devices™: http://www.analog.com/en/, July 2007.

[18] "AD6620 Digital Receive Signal Processor Datasheet: Rev. A," Analog Devices™: http://www.analog.com/en/, July 2007.

[19] "XC3S50 FPGA Datasheet: DS099-1 v2.1" Xilinx™: http://www.xilinx.com/, July 2007.

[20] "AD8346 Quadrature Modulator Datasheet: Rev. A," Analog Devices™: http://www.analog.com/en/, July 2007.

[21] "AD9774 Digital to Analog Converter Datasheet: Rev. B," Analog Devices™: http://www.analog.com/en/, July 2007.

[22] "TMS320VC5502 Fixed-Point Digital Signal Processor Data Manual: Revised August 2006," Texas Instruments™: http://www.ti.com/, July 2007.

[23] "C8051F12x-13x Mixed Signal ISP Flash MCU Family Datasheet: Rev. 1.3," Silicon Laboratories™: http://www.silabs.com/tgwWebApp/public/index.htm, July 2007.

[24] "MAX3421E USB Peripheral/Host Controller with SPI Interface," Maxim™: http://www.maxim-ic.com/, July 2007.

[25] http://www.beyondlogic.org/usbnutshell/usb1.htm. USB in a Nutshell, August 2007.

[26] http://www.emclab.umr.edu/pcbtlc2/index.html. University of Missouri - Rolla Electromagnetic Compatibility Laboratory PCB Trace Impedance Calculator, August 2007.

[27] Pozar, David, *Microwave Engineering: Second Edition*, John Wiley & Sons, Inc., pp. 196 – 198, 1998.

[28] Balanis, Constantine A., *Antenna Theory: Analysis and Design*, John Wiley & Sons, Inc., pp. 65 – 67, 94 – 95, 983 - 984, 2005

[29] Schantz, Hans, *The Art and Science of Ultrawideband Antennas*, Artech House, pp. 53-55, 2005.

[30] "LCD General Information," Lumex™: http://www.lumex.com/, July 2007.

[31] "Noise Figure," Wikipedia: The Free Encyclopedia, http://en.wikipedia.org/wiki/Noise_figure, August 2007.

[32] Schwartz, Mischa, *Mobile Wireless Communications*, Cambridge University Press, pp. 147-148, 2005.

VITA

Jason Michael Pratt was born June 20, 1984 in St. Louis, Missouri. In May 2006, he received Bachelor of Science degrees in Electrical Engineering and Computer Engineering from the University of Missouri - Rolla. Upon completion of his bachelor degrees, Jason was awarded the Chancellor's Fellowship to continue his education at the University of Missouri – Rolla. Jason graduated with a Master of Science degree in Electrical Engineering in December 2007.