FOURTH-ORDER Q-ENHANCED BAND-PASS FILTER TUNING ALGORITHM IMPLEMENTATION AND CONSIDERATIONS

by

JOEL RAYMOND SCHONBERGER

B.S., Kansas State University, 2008

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2010

Approved by:

Major Professor
Dr. William Kuhn

COPYRIGHT

ABSTRACT

Q-enhanced filtering technologies have been heavily researched, but have not yet been adopted into commercial designs due to tuning complexity, and performance issues such as noise figure and dynamic range. A multi-pole Q-enhanced band-pass filter operating at 450 MHz with tunable bandwidth is developed in this thesis. A noise figure of 14 dB and dynamic range of 140 dB/Hz have been measured, making the filter suitable for operating in the IF subsystem of a radio receiver.

The design utilizes off-chip resonators, created using surface mount components or embedded passives in LTCC processes, to have a reasonably high base-Q. The equivalent parallel loss resistance of the finite-Q inductor and connected circuitry at resonance is partially offset by negative resistances, implemented with tunable on-chip transconductors, as required to reach the needed Q for the targeted bandwidth. Each pole of the filter has binary weighted negative resistance cells for Q-enhancement and binary weighted capacitances for frequency tuning. Binary weighted capacitive coupling cells allow the filter to achieve the level of coupling appropriate to the targeted bandwidth.

To maintain the filter bandwidth, center frequency, and gain over environmental changes a real-time tuning algorithm is needed. A low complexity tuning algorithm has been implemented and found to accurately maintain the bandwidth, center frequency, and gain when operating at bandwidths of 10 or 20 MHz. Flatness of the pass-band is also maintained, to within 0.5 dB across a temperature range of 25-55 degrees C. In addition to the implementation of the tuning algorithm, the thesis provides a solution for pass-band asymmetries spawned from the use of finite-Q resonators and associated control circuitry.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my major professor, Dr. William Kuhn, who has continually proved to be an excellent mentor as well as my most valuable resource. Dr. Kuhn has, in my opinion, an unrivaled ability to present subject matter in a way that both provokes the student to think deeply and increases their desired to learn. Without his guidance and aid, this thesis would not have been possible.

I would to thank my committee members, Dr. Don Gruenbacher and Dr. Dwight Day, for their participation and guidance with my research. I would also like to recognize the roles my committee members have played in the classroom. Both Dr. Gruenbacher and Dr. Day have inspired the direction that I wish to pursue as a career through class room lectures and associated projects. Many aspects of the research performed in this thesis have been derived from such classes.

I would like to offer my appreciation to Sandia National Labs and Honeywell FM&T for the research opportunity as well as funding.

I would also like to extend my appreciation towards Renee Strouts for the design of the Q-Enhanced Filter IC. In addition to the current filter design, Renee compiled three different single-pole filter designs, from a digital radio IC design class room project, into a single IC. Renee and I first started working together on the single-pole filter project when the aspect of creating and implementing a tuning algorithm was addressed.

Finally, I would like to express my deepest appreciation to my parents, Rod and Darcy, and my girlfriend, Katie, for continually offering their support and expressing an interest in my work. My parents taught me at a young age the value of hard work and integrity. Finishing this research in the time frame that I did would have not been possible without the values my parents have instilled in me. There have been countless times that Katie has offered the support I needed while working on my research, even at wee hours of the night.

I am proud to have the support and influences of these people in my life. I hope that I might have as much of a positive influence on others as they have on me.

Section 1 - INTRODUCTION

1.1 OBJECTIVE

The objective of this research is to develop a tuning algorithm for a Fourth-Order (Two-Pole) Q-Enhanced Band-Pass Filter. The demonstration goal is to develop technology for super-heterodyne receivers with variable-bandwidth IF filtering, either on or off-chip. To demonstrate the technology, we target tuning a coupled-resonator LC filter to a 500 MHz center frequency with variable bandwidths ranging from 1 to 20 MHz over temperature variations from 25 to 75°C.

1.2 PRIMER

1.2.1 WHAT IS Q?

Non-ideal resonant circuits composed of capacitors and inductors can be characterized in terms of their Quality Factor (Q). The definition of Q in terms of energy stored and dissipated per cycle within a resonator is defined by:

$$Q = 2\pi \times \frac{Energy\ Stored}{Energy\ Dissipated\ Per\ Cycle} \qquad \textbf{Eqn. 1}$$

An ideal LC resonant circuit would not dissipate energy, thus having an infinite Q. Unfortunately, resistors, capacitors, and inductors do not behave ideally at radio frequencies (RF). Models of a resistor, capacitor, and an inductor at RF can be found within figures 1.1-3. Plots of the respective RF model's impedance can be found in figures 1.4-6.

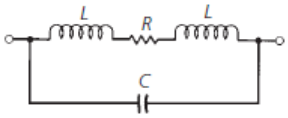


Figure 1.1 – RF Resistor Model from [1]

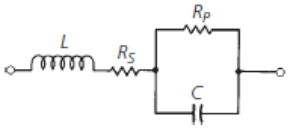


Figure 1.2 – RF Capacitor Model from [1]

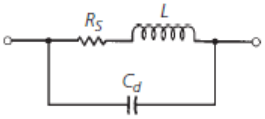


Figure 1.3 – RF Inductor Model from [1]

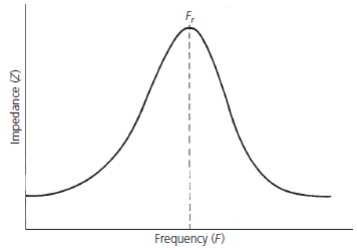


Figure 1.4 – Resistor Impedance vs. Frequency from [1]

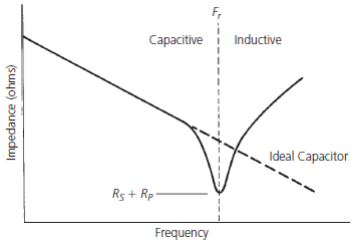


Figure 1.5 – Capacitor Impedance vs. Frequency from [1]

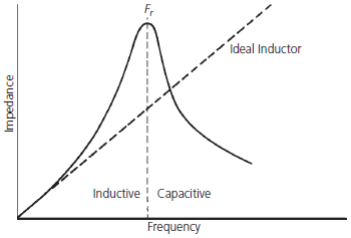


Figure 1.6 – Inductor Impedance vs. Frequency from [1]

As depicted in figures 1.3 and 1.6, an inductor at RF acts both as an inductor and capacitor. Similarly a capacitor at RF acts both as a capacitor and inductor. The frequency at which this transition occurs is defined as the self resonant frequency (F_r). More importantly, both capacitors and inductors exhibit losses modeled by the resistors shown in the equivalent circuits, leading to finite-Qs in the circuits in which they are used.

RF capacitor and inductor models can be simplified by assuming that the frequencies of interest are much lower than F_r. This assumption yields the RF capacitor and inductor models shown in figures 1.7-8.

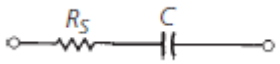


Figure 1.7 – RF Capacitor Model Below SRF from [1]

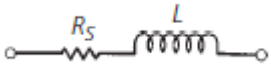


Figure 1.8 – RF Inductor Model Below SRF from [1]

The quality factor of these capacitor and inductor RF models determines the base-Q (Q_0) of a resonant circuit. Q_0 can be expressed in terms of series reactance and series resistances by:

$$Q_0 = \frac{X_s}{R_s} \qquad \textbf{Eqn. 2}$$

The importance of Q_0 will be explained in the following section.

1.2.2 The Need for Q-Enhancement

An additional definition of Q exists for a multi-pole band-pass filter. A band-pass filter's selectivity-Q can be expressed in terms of its center frequency and bandwidth (refer to figure 1.9) by:

$$Q = \frac{f_c}{f_2 - f_1} \qquad \textbf{Eqn. 3}$$

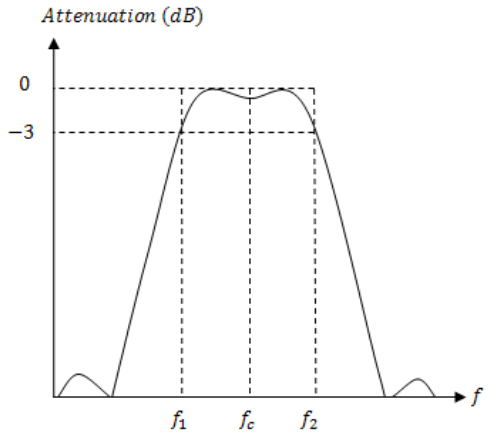


Figure 1.9 - Band-Pass Filter Attenuation vs. Frequency Plot

The filter discussed within this thesis has been designed to have an f_c of 500 MHz and variable bandwidths ranging from 20 MHz down to 1 MHz. These specifications imply that selectivity-Qs ranging from 25 to 500 are needed. If the inductors used within the filter resonators had an inductance of 3.2 nH and a series resistance of 2 Ω, equation 2 states that Q_0 at 500 MHz is 5. Because selectivity-Q cannot conventionally

exceed Q_0 of the components within its resonator [1], using a Q of 5 in equation 3 implies that at 500 MHz the narrowest bandwidth achievable is 100 MHz. Inductor Q_0 values vary due to construction style, but most are below 50 at 500 MHz. It is clear that to achieve specifications, Q-Enhancement is needed.

1.3 PRIOR ART

Many high-Q constringent applications exist within radio receiver architectures. For example, a typical off-chip preselect filter found prior to the LNA in super-heterodyne radio receiver architectures may have fractional bandwidths on the order of 1% of the operating frequency. Theoretically, the Q-Enhanced filter could act as an on-chip alternative, combining the preselect filter and LNA [2]. Traditionally, such applications have abandoned Q-Enhanced filters due to limited dynamic range and tuning difficulties. The limited dynamic range problem can be overcome if a sufficient Q_0 is used and the Q-Enhancement over Q_0 ratio is held under 20 or so [3 - 5]. With the limited dynamic range problem addressed, tuning becomes the main problem for many Q-Enhanced filter designs.

Similarly, Q-Enhanced filters could be used in the IF subsystem of a receiver [6]. Here, the performance requirements may be relaxed, but the tuning requirements remain.

Tuning of Q-Enhanced filters has been studied since their first introduction [2, 7 - 8], and many methods have been proposed. Master-Slave automatic tuning approaches using voltage controlled oscillators, over enhanced filter replicas, and frequency synthesizers were popular early on, but have since been less popular due to the limited frequency and Q-enhancement accuracy obtainable due to the need of matching and tracking of on-chip components, chip real-estate usage, power consumption, and cost [9].

An alternative to master-slave tuning is the concept of self-tuning, wherein the filter is tuned directly [10]. Many self-tuning methods, including the one discussed in this thesis, have been proposed. Although self-tuning methods offer high accuracy, they generally require the filter to be taken off-line while tuning, which could limit the applications in which such a filter might be used. To combat the need to take the filter off-line while tuning, some designers have opted to use a replica filter acting as the primary filter while the secondary undergoes the tuning process. This process was first introduced by [10]. Still, digital communication systems operating on time-division standards might allow the tuning process to be run during an idle slot.

Automatic self-tuning filter topologies rely on control circuitry to provide feedback on the state of the filter. Digital tuning algorithms, such as the one described in this thesis and [11, 12], offer a simple and

efficient method of achieving the desired frequency response over environmental changes and process tolerances, often with little additional hardware or power consumption.

1.4 K-State Two-Pole Q-Enhanced Filter Design

The Q-Enhanced Filter design utilized within this research is documented in [13]. The filter core and peripherals were designed by Ms. Strouts, while I designed the digital portions of the chip. The Q-Enhanced filter design allows real-time tuning of center frequency and bandwidth making it very well suited for applications such as Software Defined Radios (SDR) [14].

1.4.1 Block Diagram

A block diagram of the Q-Enhanced Filter chip can be found in figure 1.10. Differential inputs are connected to a cascoded differential amplifier core referred to as the Front-End. Similarly, an identical copy of the Front-End, with grounded inputs, is referred to as the Back-End, to help reduce frequency and Q offsets between the two sections. The filter utilizes two identical off-chip resonators, one connected to each amplifier core, created with either surface mount components or embedded passives in LTCC process [20, 21]. Each amplifier core has identical cells for frequency tuning and Q-Enhancement tuning. The frequency is tuned by varying binary weighted capacitances for coarse tuning and applying a DC voltage for fine analog tuning. Similarly, Q-Enhancement is adjusted by varying binary weighted negative resistance cells for coarse adjustment and applying a DC voltage for fine analog tuning. Each amplifier core also has an output buffer attached.

Each buffer drives an amplitude detector and a frequency divider. The amplitude detector outputs a voltage inversely proportional to the signal strength of the buffered output. Specifically, it is used to detect at what Q-Enhancement level first causes oscillation. The frequency divider scales the oscillating frequency by a factor of 64 so that it can be read by a frequency counter implemented on a microcontroller. The amplitude detector and frequency divider are the sole components used within the tuning algorithms discussed here. Since they are normally off when not in use, their circuitry does not significantly impact the overall power consumption of the filter.

The filtered signal is passed from the Front-End to the Back-End via quadrature-phase coupling in the form of on-chip binary weighted capacitances. Both lateral (resonator side's 1-to-1 and 2-to-2) and cross (resonator side's 1-to-2 and 2-to-1) capacitive coupling exist. Lateral coupling is designed to couple the resonators together establish bandwidth, while cross coupling is designed to minimize the effects of the inherent magnetic coupling between the inductors used within the off-chip resonators.

Section 3 of this thesis looks at these coupling mechanisms in detail and highlights an important issue when finite-Q components are involved.

All circuitry enable controls and binary-weighted digital tuning is controlled via a 64 bit serial-to-parallel register. The serial-to-parallel register is programmed using a three wire serial interface. A bonding diagram and pin descriptions for the complete chip are shown in Section 1.4.2.

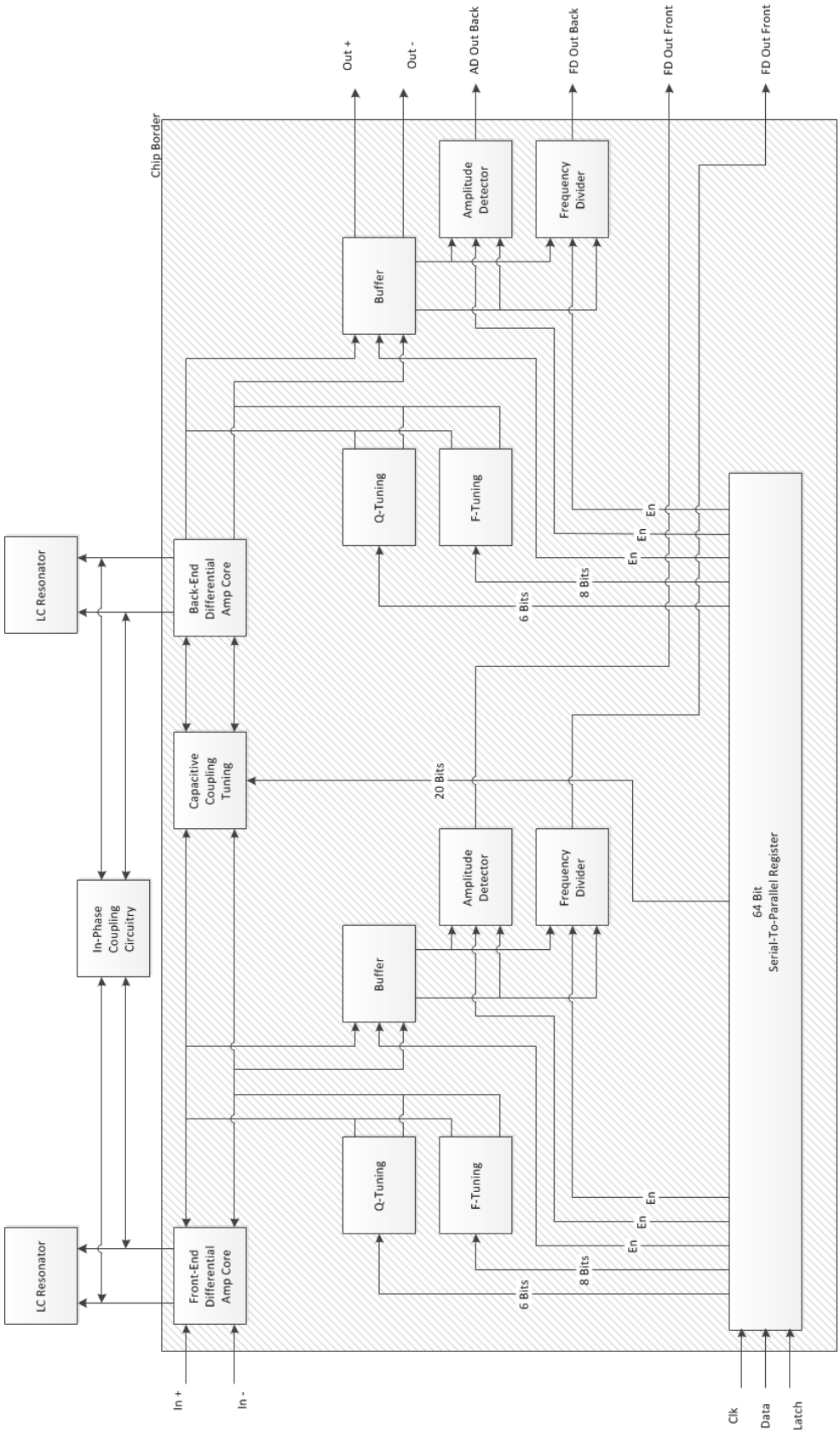


Figure 1.10 – Q-Enhanced Filter Block Diagram

1.4.2 Q-Enhanced Filter Pin-Out

A bonding diagram of the Q-Enhanced Filter is show in figure 1.11. Currently the filter design consumes half of a 3x3 mm die. The other half of the die is reserved for the synthesized digital tuning algorithm circuitry that will be developed based upon this research.

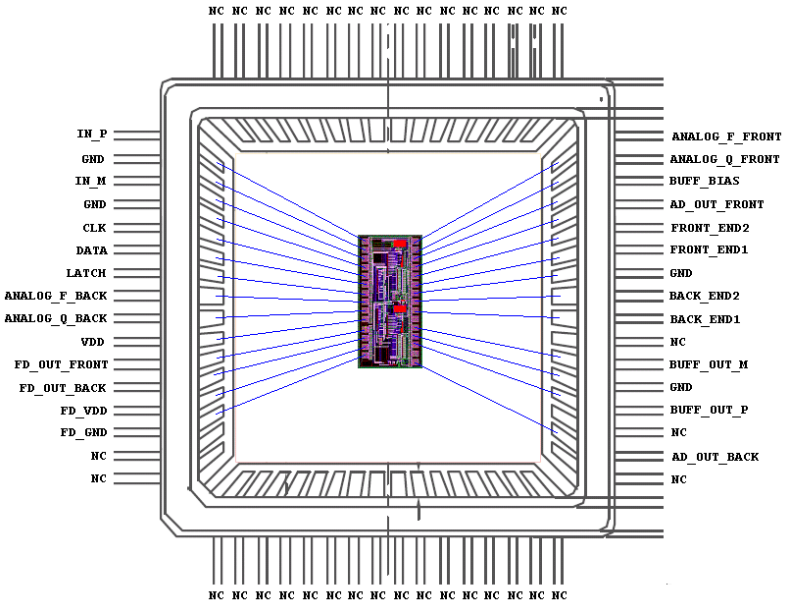


Figure 1.11 – Q-Enhanced Filter Bonding Diagram

Pin connection descriptions in table 1.1 are relevant to the printed circuit boards (PCBs) created to develop the Q-Enhanced Filter tuning algorithms. Refer to Appendices A-C for schematics, parts lists, layouts, photographs, and additional details about the PCBs.

Q-Enhanced Filter Pin Descriptions		
Pin	**Description**	**Connection**
IN_P & IN_M	Differential RF Inputs	SMA
CLK, DATA, LATCH	Serial Programming Lines	Microcontroller (µC)
ANALOG_F* & ANALOG_Q*	Analog Tuning Lines	Digital-to-Analog Converter
FD_OUT*	Frequency Divider Output	Multiplexor → µC
AD_OUT*	Amplitude Detector Output	Analog-to-Digital Converter (µC)
FRONT_END1 & FRONT_END2	Differential Core Amplifier Outputs	Front-End LC Resonator
BACK_END1 & BACK_END2	Differential Core Amplifier Outputs	Back-End LC Resonator
BUFF_OUT_P & BUFF_OUT_M	Differential Buffered RF Outputs	SMA
BUFF_BIAS	Buffer Bias	None (Left Floating)
VDD	Analog Supply Voltage (2.5V)	2.5 V Regulator
FD_VDD	Frequency Divider Supply (2.5V)	2.5 V Regulator
GND	Analog Ground	Ground
FD_GND	Frequency Divider Ground	Ground
* Denotes pin option for both Front-End and Back-End		

Table 1.1 – Q-Enhanced Filter Pin Descriptions

1.5 THESIS OVERVIEW

Prior to developing any of the proposed tuning algorithms in this thesis, the PCBs documented in Appendices A-B were created. Figure 1.12 shows the second version of the PCB documented in Appendix B. To verify that the filter design worked, the test application documented in Appendices D and F was created.

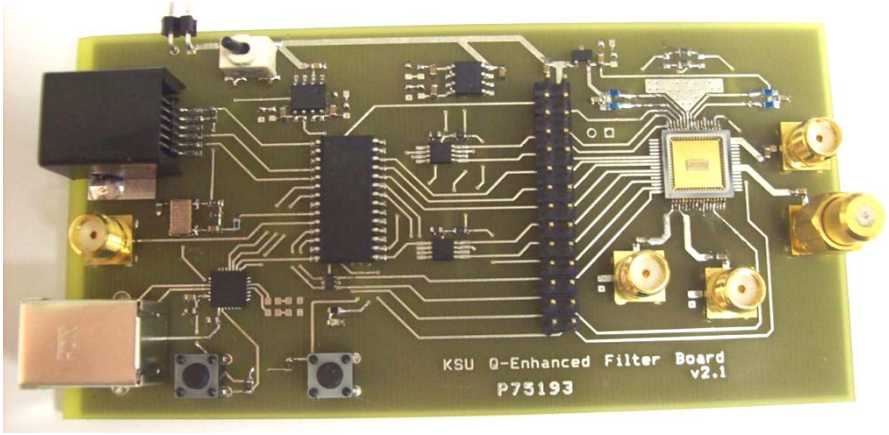


Figure 1.12 –Q-Enhanced Filter Board v 2.1

Filter testing was accomplished with the setup shown in figure 1.13. A spectrum analyzer was used both as a means of viewing the filter's output and providing the RF input. The spectrum analyzers tracking RF output was connected to an RF switch controlled by the Q-Enhanced Filter board. When the RF switch is turned on, the signal is passes through a RF splitter and into the differential input of the filter. The output of the filter is connected to the input of the spectrum analyzer to allow the filter response to be observed

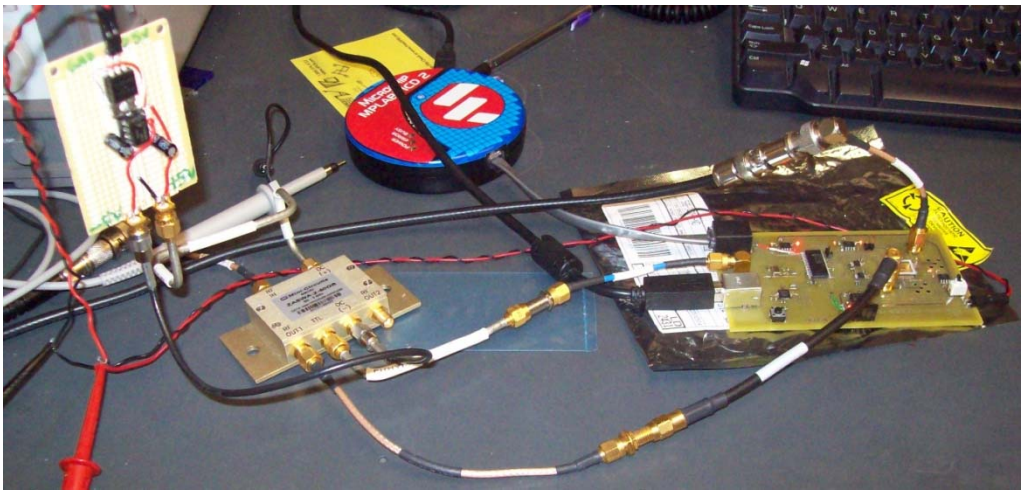


Figure 1.13 – Q-Enhanced Filter Testing Setup

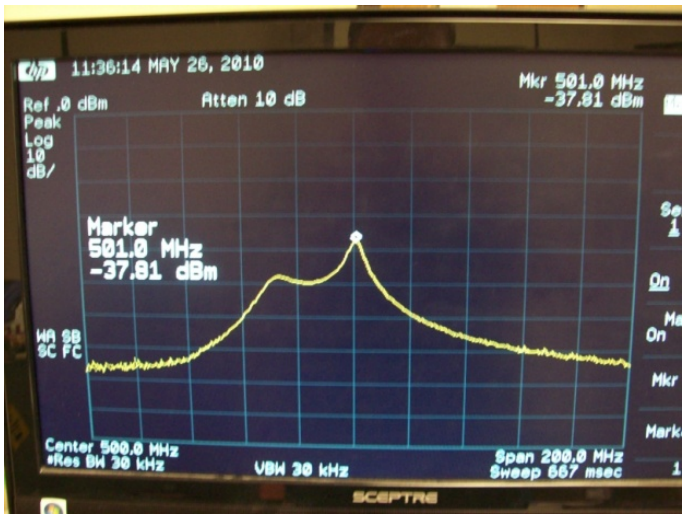


Figure 1.14 – Manually Tuned Response

Using the test application, I was able to manually tune the filter. Ideally, having identical Q-Enhancement and frequency controls on the Front-End and Back-End, with the appropriate amount of capacitive coupling, should yield a flat pass-band. Unfortunately, identical controls did not produce the desired response as figure 1.14 depicts. A significant amount of magnetic coupling between the Front-End and Back-End was initially believed to be causing the filter to behave differently than expected. However, the true causes of the response asymmetry are more involved and are investigated in depth in Section 3 of this thesis. Once these problems were solved, the tuning algorithm was developed and its performance assessed in Section 4. Finally, the thesis concludes with suggestions for future work in Section 5.

Section 2 - MAGNETIC COUPLING EFFECTS

The basic coupled-resonator filter topology on which this research is focused is the two-pole circuit shown in figure 2.1. Without the negative resistance elements shown, this topology is the foundation in which many filters have been implemented in practice. Additional resonators and coupling elements can be added to increase the amount of filter poles. The bandwidth of the filter is dependent upon resonator-Q and the amount of coupling between resonators [1]. The amount of coupling needed to obtain the targeted bandwidth adds an additional complexity to automatically tuned multi-pole filters. Without the correct amount of coupling, the bandwidth will be affected and the pass-band response will deviate from the desired flat shape.

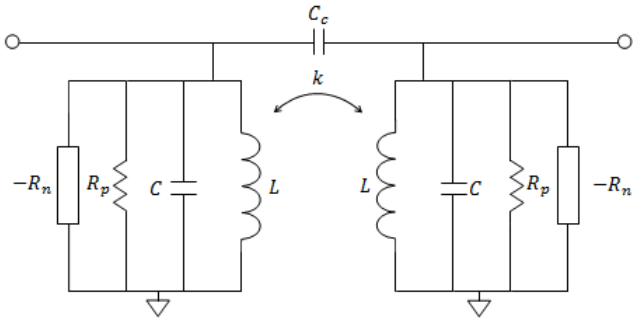


Figure 2.1 – Q-Enhanced Filter Model

While a filter can be built with only magnetic coupling, only capacitive coupling, or both, it is simpler to tune the filter if the coupling values are known in advance to a reasonable degree of accuracy. By effectively minimizing the magnetic coupling to the point where it becomes negligible, the tuning algorithm can solely use capacitive coupling to establish bandwidth. Thus, this section defines a method of measuring the magnetic coupling coefficient between inductors, and examines the magnetic coupling on two different PCB layouts.

2.1 ANALYSIS

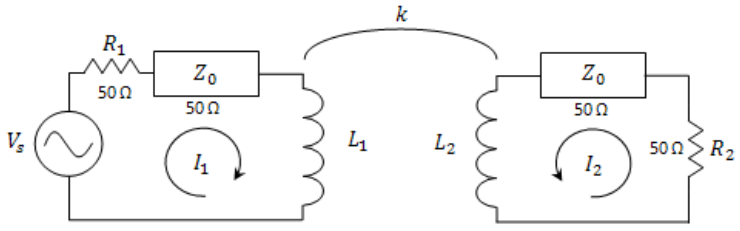


Figure 2.2 - Magnetic Coupling Analysis Setup

To solely depend on capacitive coupling as the means of transferring energy from the Front-End to the Back-End, magnetic coupling between the resonators must be quantified and minimized. To quantify this magnetic coupling, the circuit topology shown in figure 2.2 was used. Identical coupled inductors, L_1 and

L_2, were connected to 50 Ω characteristic impedance coax cables and fed into ports 1 and 2 of a network analyzer. Port 1 of the network analyzer is modeled as the AC voltage source with the 50 Ω source resistance R_1. Port 2 of the network analyzer is modeled as the 50 Ω input resistance R_2.

Using S-parameters as a means of two-port network analysis, figure 2.3, allows the magnetic coupling coefficient (k) to be analyzed and measured in terms of forward voltage gain (S_{21}). S-parameter definitions can be found in equations 4-7 below.

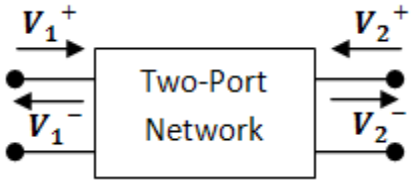


Figure 2.3 – S-Parameter Two-Port Network

Input Port Voltage Reflection Coefficient

$$S_{11} = \left.\frac{V_1^-}{V_1^+}\right|_{\Gamma_2=0} \qquad \textbf{Eqn. 4}$$

Reverse Voltage Gain

$$S_{12} = \left.\frac{V_1^-}{V_2^+}\right|_{\Gamma_2=0} \qquad \textbf{Eqn. 5}$$

Forward Voltage Gain

$$S_{21} = \left.\frac{V_2^-}{V_1^+}\right|_{\Gamma_1=0} \qquad \textbf{Eqn. 6}$$

Output Port Voltage Reflection Coefficient

$$S_{22} = \left.\frac{V_2^-}{V_2^+}\right|_{\Gamma_1=0} \qquad \textbf{Eqn. 7}$$

To see how the magnetic coupling coefficient k is related to S_{21}, equations for V_2^- and V_1^+ need to be defined. It can be shown that V_1^+ is related to the source voltage through equation 8, where Z_0 is the characteristic impedance of the transmission line:

$$V_1^+ = \left(\frac{Z_0}{R_1 + Z_0}\right) V_s = \left(\frac{50}{50 + 50}\right) V_s = \frac{V_s}{2} \qquad \textbf{Eqn. 8}$$

V_S can be expressed in terms of V_1^+ through the following equation:

$$V_s = 2V_1^+ \qquad \textbf{Eqn. 9}$$

The mutual inductance (M) between identical inductors can be expressed as:

$$M = k\sqrt{L_1 L_2}\Big|_{L_1=L_2=L} \approx kL \qquad \textbf{Eqn. 10}$$

Simplification of the M equation is valid because the inductors within the resonant LC tanks are identical.

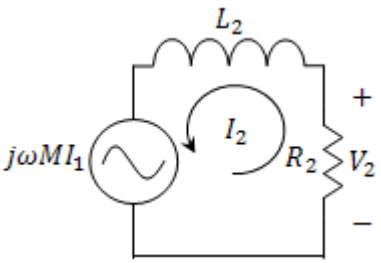


Figure 2.4 – Induced Voltage Model

To find V_2^- and hence S_{21} we must find the voltage across L_2 when it is loaded with 50 Ω. To quantify the voltage across L_2, the circuit model in figure 2.4 was used. Here, the voltage source represents the voltage induced in L_2 from current flowing in L_1. From Faraday's law, the voltage induced in L_2 is equal to $j\omega MI_1$. Using this circuit model, the total voltage across L_2 is identical to the known expression given by:

$$V_2 = j\omega LI_2 + j\omega MI_1 \qquad \textbf{Eqn. 11}$$

Using the circuit shown in figure 2.3 in place of equation 10 allows a single voltage divider to be examined. When L_2 is connected to port 2 of the network analyzer, V_2^- is equivalent to V_2 and given by:

$$V_2^- = \left(\frac{R_2}{j\omega L + R_2}\right) j\omega MI_1 = \left(\frac{50}{j\omega L + 50}\right) j\omega MI_1 \qquad \textbf{Eqn. 12}$$

Assuming that M is much less than the inductance of L_1 and L_2 ($k << 1$), the current flowing through L_1 can be found from figure 2.1 to be approximately equal to:

$$I_1 \cong \frac{V_s}{R_1 + j\omega L} = \frac{V_s}{50 + j\omega L} \qquad \textbf{Eqn. 13}$$

Combining equations 6, 9, and 11-12, the magnitude of S_{21} is equivalent to:

$$|S_{21}| \cong \left|\frac{V_2^-}{V_1^+}\right| = \left|\frac{100}{(j\omega L + 50)^2} j\omega kL\right|_{X_L = \omega L} = \left|\frac{100}{(jX_L + 50)^2} jkX_L\right| \qquad \textbf{Eqn. 14}$$

or, in terms of the inductive reactance X_L:

$$|S_{21}| \cong \left|\frac{100}{(jX_L + 50)^2} jkX_L\right| \qquad \textbf{Eqn. 15}$$

$|S_{21}|$ evaluated as a function of X_L and k in the following equation:

$$|S_{21}(X_L, k)| \cong \begin{cases} X_L \ll 50\Omega, & \frac{kX_L}{25} \\ X_L \gg 50\Omega, & \frac{100k}{{X_L}^2} \\ X_L = 50\Omega, & k \end{cases} \qquad \textbf{Eqn. 16}$$

When $|S_{21}|$ is evaluated at an X_L of 50 Ω, the $|S_{21}|$ is equivalent to k, which allows k to be easily measured on a network analyzer.

2.2 Verification Through Simulation

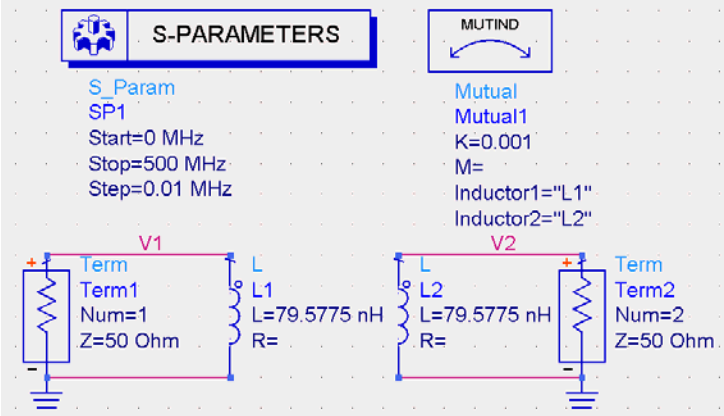


Figure 2.5 – Magnetic Coupling Measurement Model

Agilent Advanced Design Systems (ADS) simulations were used to verify the relationship between S_{21} and k. On the bench, L_1 and L_2 would be connected respectively to port 1 and port 2 of a network analyzer. In simulation, the termination resistors within the schematic shown in figure 2.4 represent the input and output impedances of network analyzer ports. The inductor sizes were chosen for a 50 Ω inductive reactance at 100 MHz.

Figures 2.5-9 show simulation results of $|S_{21}|$ and $|S_{11}|$ at coupling coefficient values of 0.001, 0.01, 0.1, 0.3, and 0.5 respectively to access the accuracy of the approximation in equation 16.

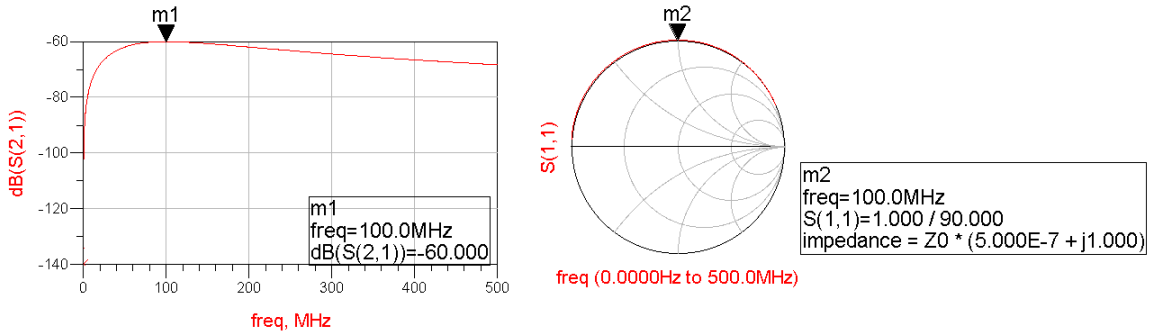


Figure 2.6 – Simulation with Coupling Coefficient of 0.001

$$k = 10^{db(|S_{21}|)/20} = 10^{-60/20} = 0.001 \qquad Z \approx j50\ \Omega$$

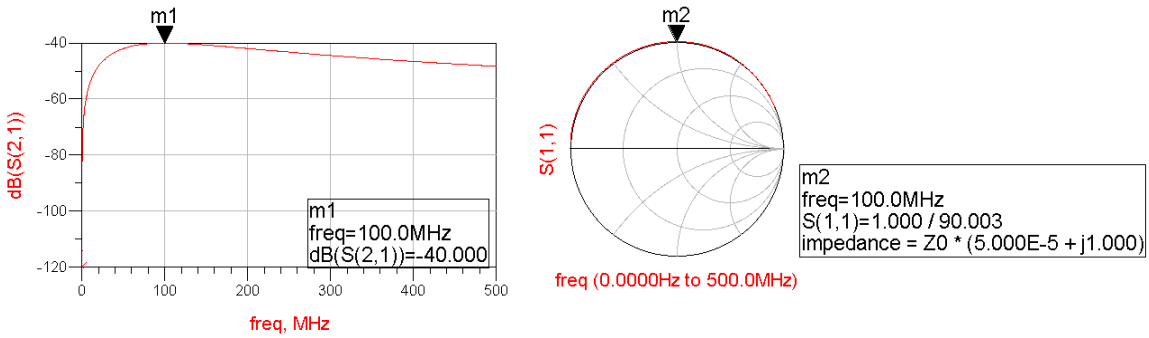


Figure 2.7 - Simulation with Coupling Coefficient of 0.01

$$k = 10^{db(|S_{21}|)/20} = 10^{-40/20} = 0.01 \qquad Z \approx j50\ \Omega$$

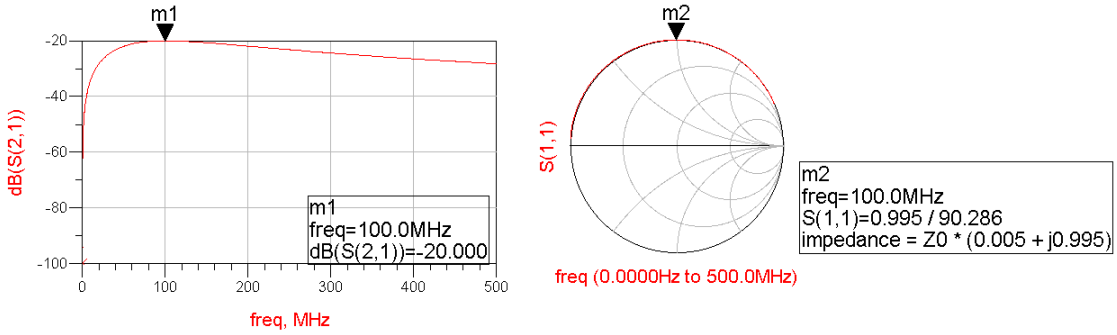


Figure 2.8 - Simulation with Coupling Coefficient of 0.1

$$k = 10^{db(|S_{21}|)/20} = 10^{-20/20} = 0.1 \qquad Z \approx 0.25 + j49.75\ \Omega$$

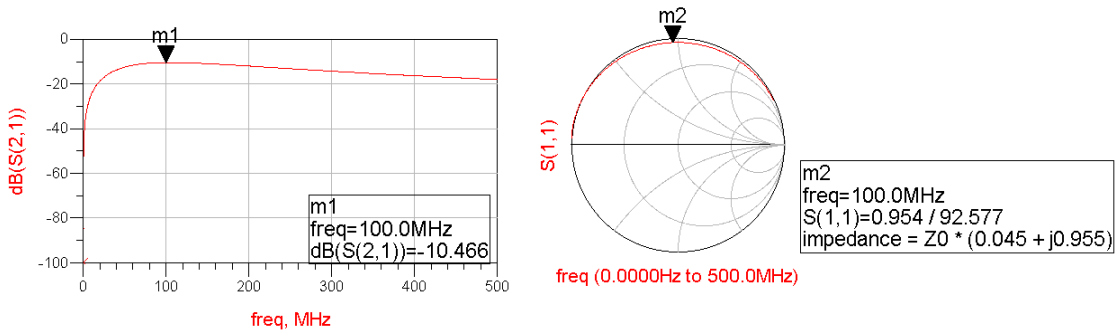


Figure 2.9 - Simulation with Coupling Coefficient of 0.3

$$k = 10^{db(|S_{21}|)/20} = 10^{-10.466/20} = 0.299 \qquad Z \approx 2.25 + j47.75\ \Omega$$

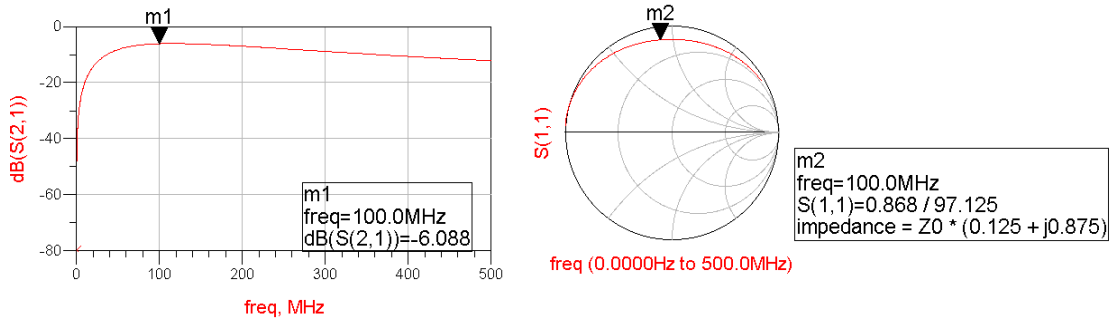


Figure 2.10 - Simulation with Coupling Coefficient of 0.5

$$k = 10^{db(S_{21})/20} = 10^{-6.088/20} = 0.496 \qquad Z \approx 6.25 + j43.75\ \Omega$$

Two observations can be made from these simulations results. First, $|S_{21}|$ is nearly equivalent to k at the frequency where X_L is equal to 50 Ω for even large values of k such as 0.5. Secondly, as k is increased, the inductor's impedance becomes more resistive and less inductive. The change in impedance is caused by the change in loading seen by L_1 as k is increased.

2.3 Measured Coupling Coefficients

With a means of measuring k between two inductors defined, scale models were used to quantify magnetic coupling as a function of separation. The first step in this measurement process was to create and adjust L_1 and L_2 to have 50 Ω reactances at the same frequency.

Figures 2.10-11 show S_{11} of L_1 and L_2 on a Smith Chart at roughly 51.214 MHz.

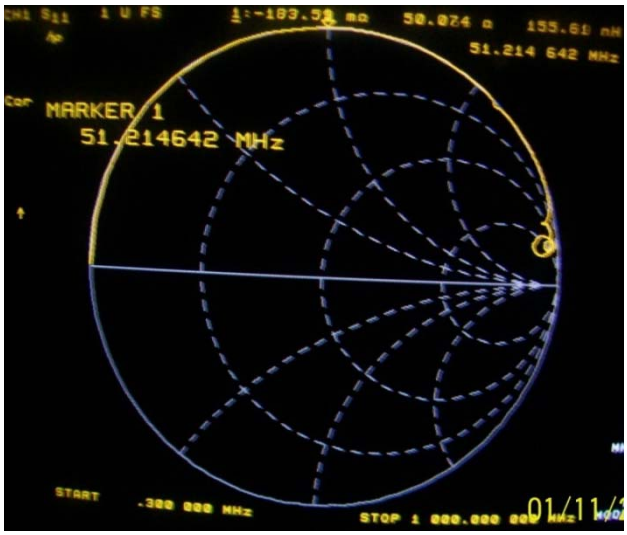


Figure 2.11 – S_{11} Measurement of L_1

Z_1 = -0.1 + j50 Ω L_1 = 155.6 nH

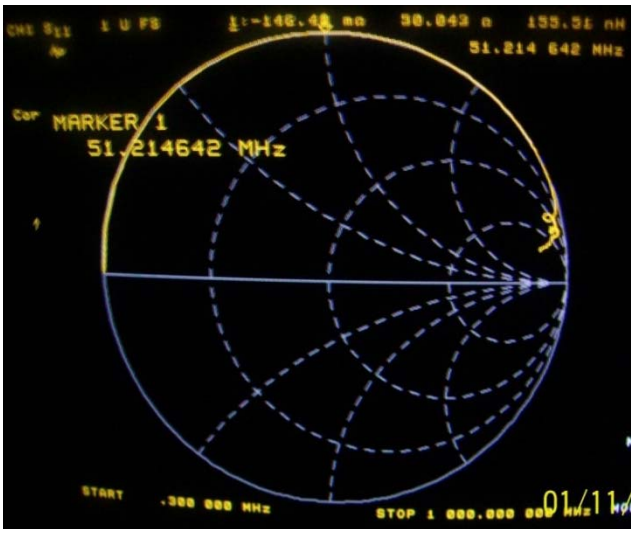


Figure 2.12 – S_{11} Measurement of L_2

Z_2 = -0.1 + j50 Ω L_2 = 155.5 nH

After the inductors were properly sized, the network analyzer was calibrated to measure S_{21}. To ensure the measurements were above the noise floor, the reference level was adjusted and the IF bandwidth was decreased from 3000 Hz to 30 Hz as shown in figures 2.12-13.

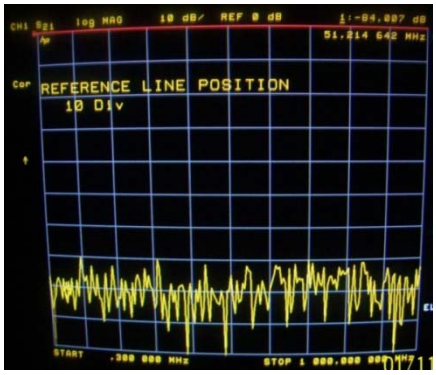


Figure 2.13 – Measurement Reference Adjustment

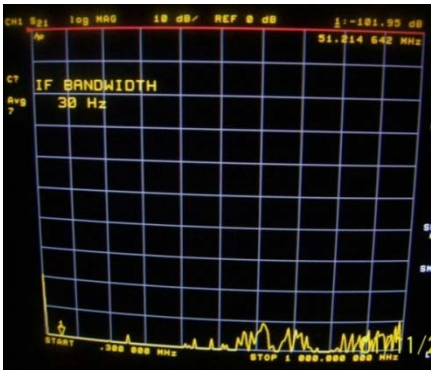


Figure 2.14 – IF Bandwidth Adjustment

S_{21} was measured at set intervals of separation over diameter (S/D) ratios. Figure 2.14 shows the experimental setup used for the S_{21} measurements. L_1 was held in a fixed position while the position of L_2 was varied.

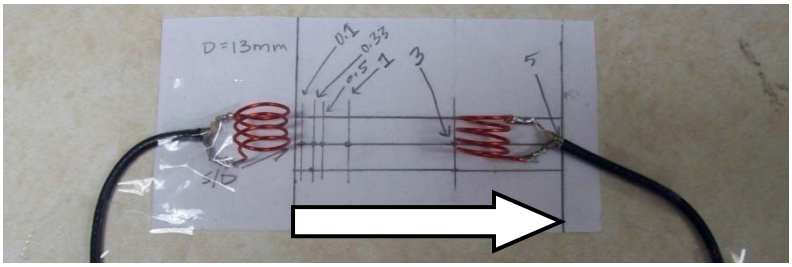


Figure 2.15 – Magnetic Coupling Measurement Setup

$|S_{21}|$ measurements of the inductors are found within table 2.1 and shown in figure 2.15.

Magnetic Coupling Coefficient Measurements of 13mm Diameter Inductors			
S/D	**S (mm)**	**|S21| (dB)**	**|K|**
0.1	1.3	-29.2	0.03467
0.33	4.29	-33.2	0.02188
0.5	6.5	-38.4	0.01202
1	13	-44.1	0.00624
3	39	-53.7	0.00207
5	65	-67.6	0.00042

Table 2.1 – Magnetic Coupling Coefficient Measurements

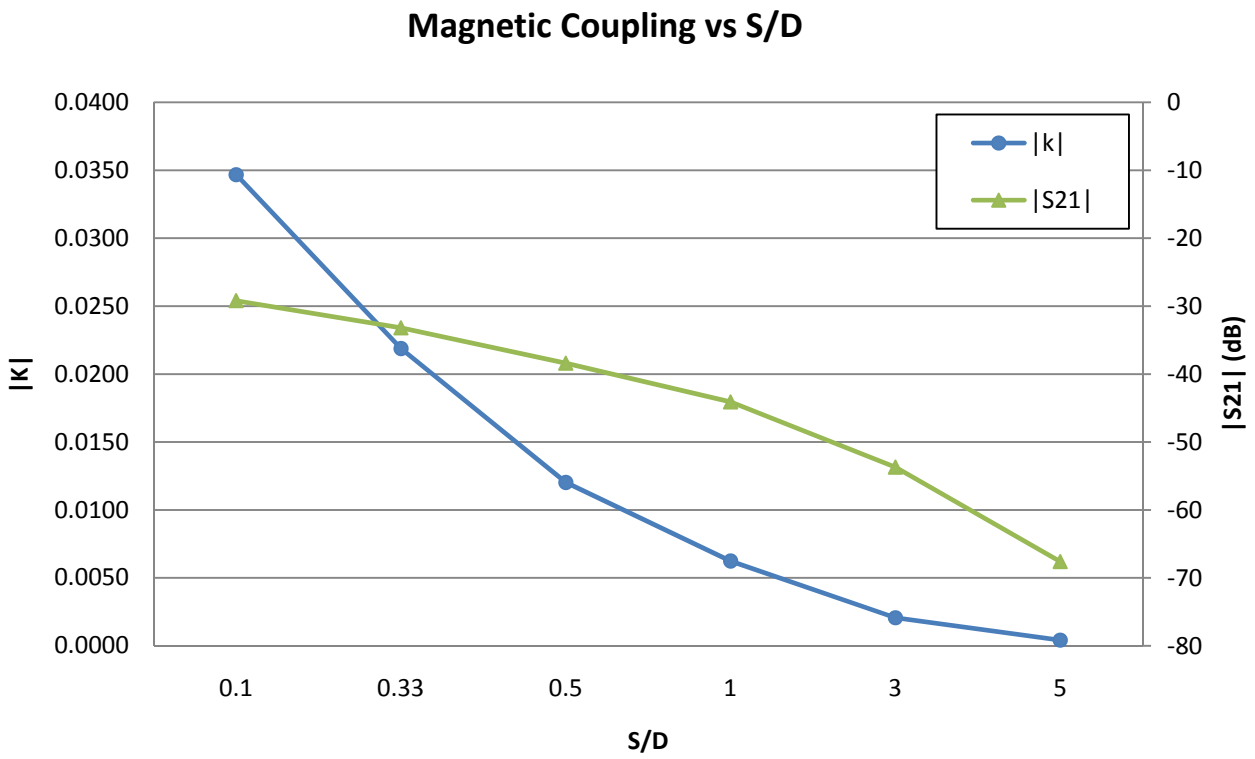
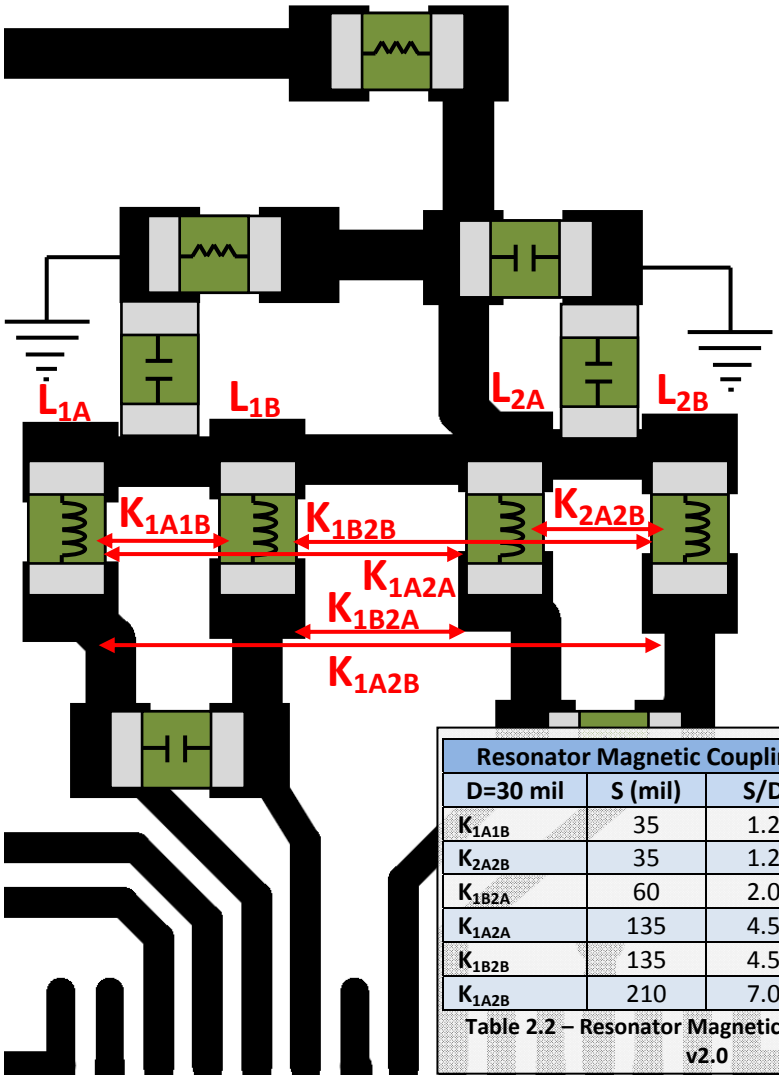


Figure 2.16 - Magnetic Coupling Coefficient Measurements

2.4 APPLICATION

With magnetic coupling measurements complete at various S/D ratios, interpolation of this data was used to estimate k between the resonator inductors on Version 2.0 of the Q-Enhanced Filter PCB (documented in Appendix A). Table 2.2, found in figure 2.16, shows data on each of the magnetic coupling possibilities. Since the required k value for a magnetic-only coupled filter is on the reciprocal of the selectivity Q, it is clear from the measurements that Version 2.0 PCB's resonator topology includes a reasonably large amount of magnetic coupling in the filter's response. To build a filter where magnetic coupling plays a negligible role, section 2.5 examines possible magnetic coupling minimization techniques.



Resonator Magnetic Coupling of PCB v2.0			
D=30 mil	S (mil)	S/D	k
K_{1A1B}	35	1.2	~ 0.005
K_{2A2B}	35	1.2	~ 0.005
K_{1B2A}	60	2.0	~ 0.004
K_{1A2A}	135	4.5	~ 0.0006
K_{1B2B}	135	4.5	~ 0.0006
K_{1A2B}	210	7.0	< 0.0003

Table 2.2 – Resonator Magnetic Coupling on PCB v2.0

Figure 2.17 – Magnetic Coupling Diagram of PCB v2.0

2.5 Minimization Techniques

Magnetic coupling of inductors can be minimized by increasing separation and adjusting their orientations. Figures 2.18 and 19 show inductors in perpendicular and inline configurations. It can be shown that the net magnetic fields of inductors in these configurations will be less than the net magnetic field of the inductors in parallel configurations, as shown in figure 2.20.

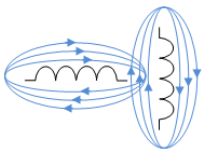


Figure 2.18 – Perpendicular Inductors

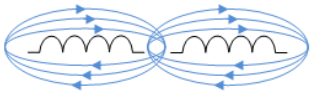


Figure 2.19 - Inline Inductors

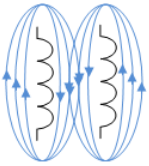


Figure 2.20 – Parallel Inductors

Version 2.0 of the Q-Enhanced Filter PCB had two magnetic coupling related flaws. The resonators were insufficiently spaced and were arranged in a parallel configuration. These two flaws made the magnetic coupling non-negligible. Version 2.1 of the Q-Enhanced Filter PCB, documented in Appendix B, separates the resonators by an S/D ratio of 23 and places the resonators in an inline configuration as shown in figure 2.20. To further minimize coupling, an additional ground plane was added between the resonators to terminate electrical fields created by the traces to the ground plane below the circuit layer on the board to mitigate magnetic fields around traces.

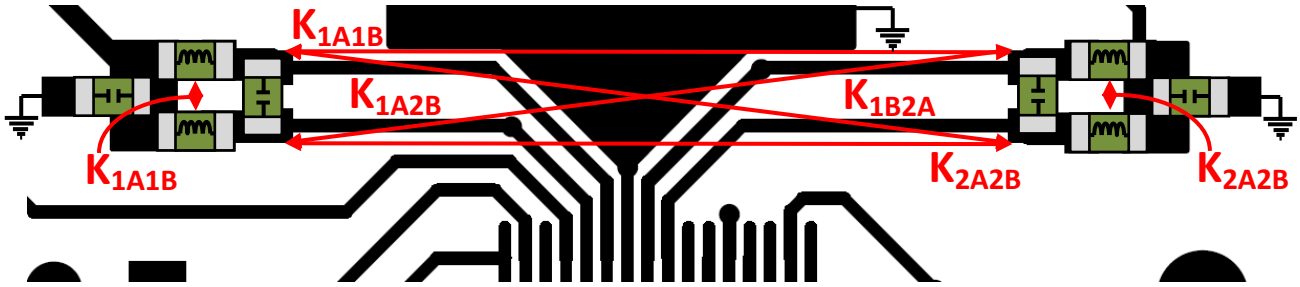


Figure 2.21 – Magnetic Coupling Diagram of PCB v2.1

2.6 Conclusions

Resonator Magnetic Coupling of PCB v2.1			
D=30 mil	S (mil)	S/D	k
K_{1A1B}	30	1	~ 0.006
K_{2A2B}	30	1	~ 0.006
K_{1B2A}	696	23.2	~ 0
K_{1A2A}	690	23.0	~ 0
K_{1B2B}	696	23.2	~ 0
K_{1A2B}	690	23.0	~ 0

Table 2.3 – Resonator Magnetic Coupling on PCB v2.1

Table 2.3 shows data on each magnetic coupling resulting from the layout of figure 2.21. The k between inductors of opposing resonators is negligible due to large S/D ratio. The k between inductors of the same resonator yields a higher total inductance affecting the center frequency of the resonator. This increase in total inductance can be countered by decreasing the size of capacitor used within the resonator. Clearly, the topology of the resonators in figure 2.20 makes the magnetic coupling of resonators negligible.

Section 3 - PASS-BAND ASYMMETRIES

Several different implementations of Q-Enhanced filters have lead to asymmetrical pass-bands [2, 15-19]. Such asymmetries have been referred to as pass-band: asymmetries, droop, distortion, and ripple. It has been generally accepted that the asymmetries occur due the inability of the negative resistance to cancel the parallel equivalent of the series loss of the resonator components at all frequencies. It has also been noted that using higher-Q resonant components reduces the presence of the asymmetries [18]. While the asymmetries have been observed by many designers, a solution to the problem has only been addressed in [2, 15]. This section takes a deeper look at the asymmetrical pass-band problem in terms of Q of resonator components and the coupling circuitry, and defines an off-chip solution.

3.1 OBSERVATIONS

Pass-band asymmetries have been observed within implementation and simulation. Figures 3.1-2 show the asymmetries found within implementation. Ideally, a symmetrical pass-band will occur when the Front-End and Back-End have identical frequency and Q-Enhancement settings, and a flat pass-band will be obtained when the coupling is at the correct value. In implementation, the asymmetries are exemplified by setting the lateral and cross coupling settings to opposing extremes. Although placing the coupling settings at their extremes is not required for the asymmetries to appear, doing so makes them easily identifiable. Figure 3.1 shows the asymmetry formed when the lateral coupling settings are set to their maximum, while the cross coupling settings are set to their minimum. Figure 3.2 shows the asymmetry formed when the cross coupling settings are set to their maximum, while the lateral coupling settings are set to their minimum. Figures 3.1-2 clearly demonstrate that the asymmetry within the pass-band can be moved by modifying the coupling settings.

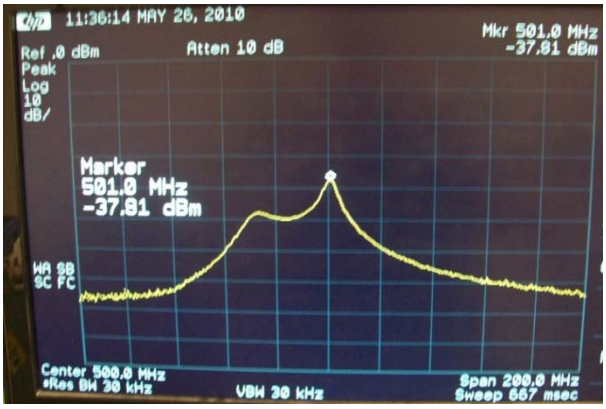


Figure 3.1 - Asymmetry with Maximum Lateral Coupling

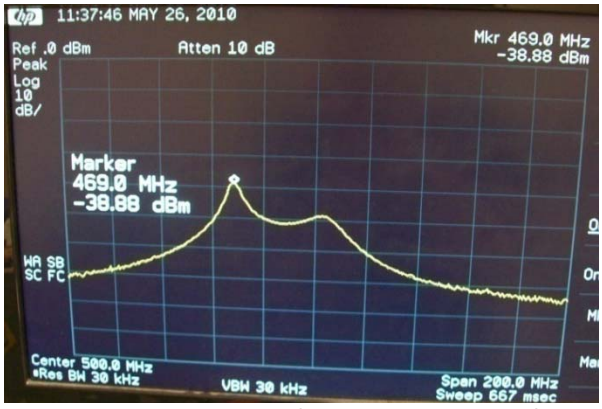


Figure 3.2 - Asymmetry with Maximum Cross Coupling

3.2 ASYMMETRY NEUTRALIZATION

Agilent ADS simulations were used to verify that asymmetries occur within a Q-Enhanced filter model and to investigate methods to neutralize them. In simulation, the Q_0 of the resonator inductors can be adjusted by varying the series resistance. Figure 3.3 shows the filter response when Q_0 of the inductors is reasonably high ($X_L >> R_s$). Figure 3.4 shows the response when the Q_0 of the inductors is reasonably low ($X_L > R_s$). Clearly, the asymmetries can be minimized by using inductors with reasonably high Q_0 as seen in [18]. However, this counter to the main idea of Q-Enhancement, thus a better method of neutralization is required.

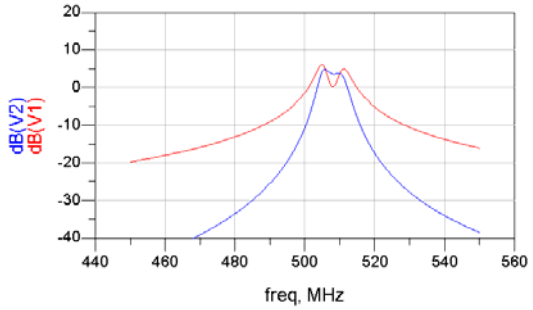


Figure 3.3 – Simulation with High Q

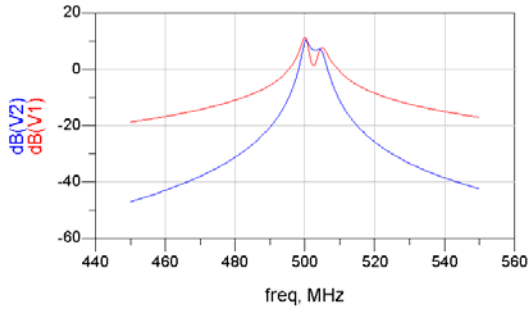


Figure 3.4 – Simulation with Low Q

The main origin of asymmetry is from non-quadrature coupling, as noted in [2]. When two ideal resonators are capacitively (or inductively) coupled, the current in the second resonator is in quadrature-phase with the current in the first resonator. When two *non-ideal* resonators are coupled, an undesired in-phase component appears in addition to the desired quadrature-phase component. After extensive investigations in the course of this research, it is believed that the asymmetries are caused by the in-phase component introduced by both the series resistance of the non-ideal resonator components [2, 15] and resistances found within the on-chip coupling capacitor circuitry. A detailed analysis of these two mechanisms is presented in the following section. Ultimately, resistors were added to the Q-Enhanced filter model, shown in figure 3.5, to neutralize the asymmetries.

Figure 3.6 shows the filter response when using this combined in-phase and quadrature coupling. By adjusting the amount of resistance in parallel with the coupling capacitance the asymmetries have been effectively neutralized for a flat pass-band.

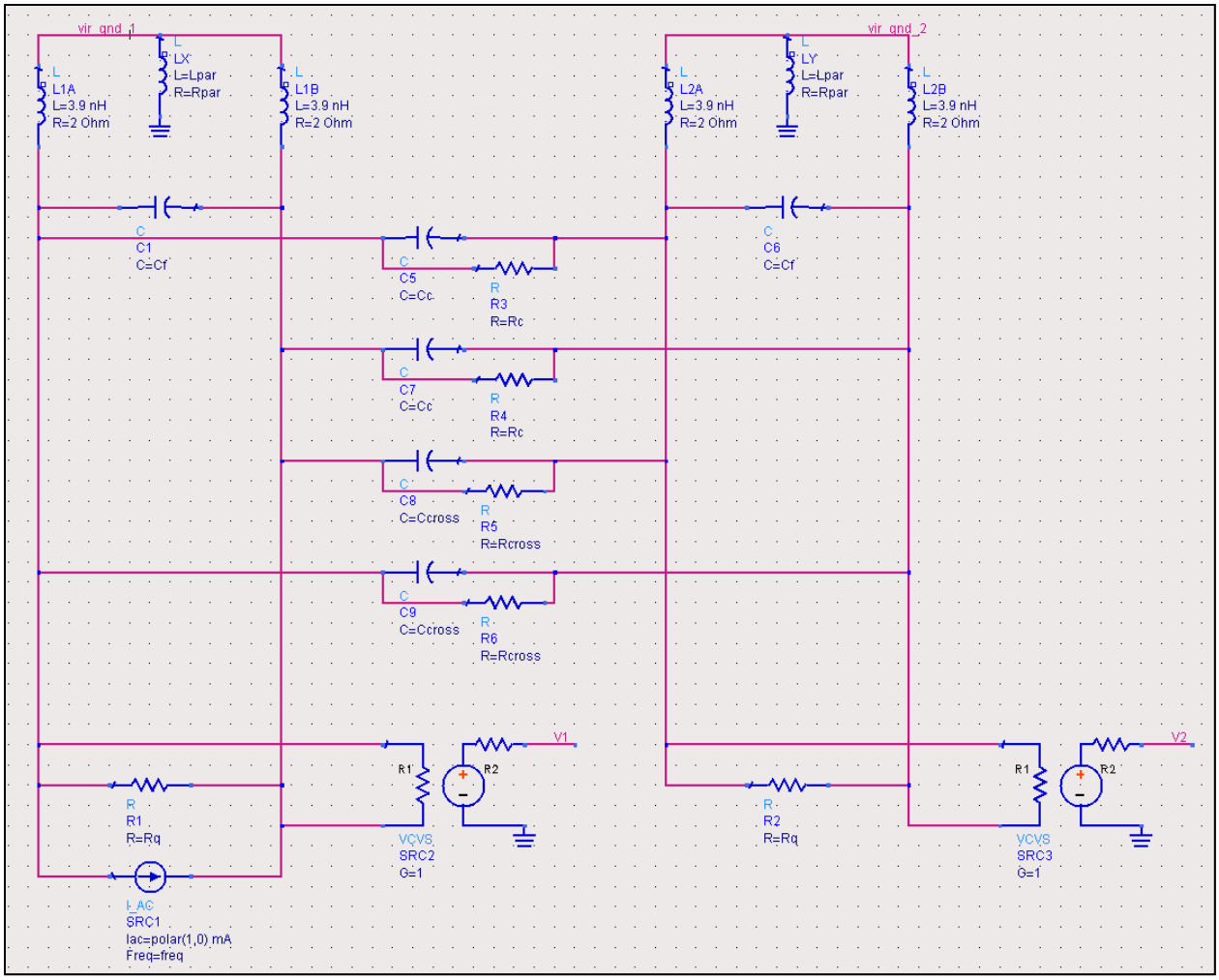


Figure 3.5 – Q-Enhanced Filter Model with In-Phase Coupling

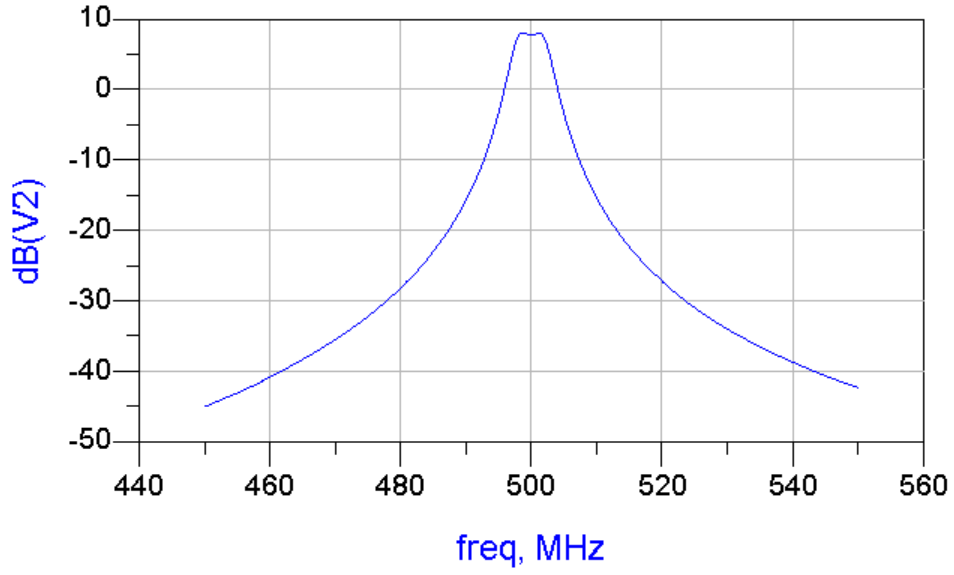


Figure 3.6 - Simulation with In-Phase Coupling

3.3 In-Phase Coupling

The amount of in-phase coupling needed to neutralize the asymmetries is a function of Q-Enhancement and frequency. Unfortunately, an additional fabrication of the filter was not possible during the course of this research to add the in-phase coupling controls implemented in [2, 15]. Off-chip resistors were therefore used between the resonator LC tanks to neutralize the asymmetries over a restricted tuning range as an alternative to on-chip neutralization. The resistor value required will depend on both the origin of the non-ideal in-phase coupling and its magnitude. For general design guidance we therefore examine both Inductor-Q based and Capacitive-Coupling based situations below.

3.3.1 In Terms of Resonator Inductor Q_0

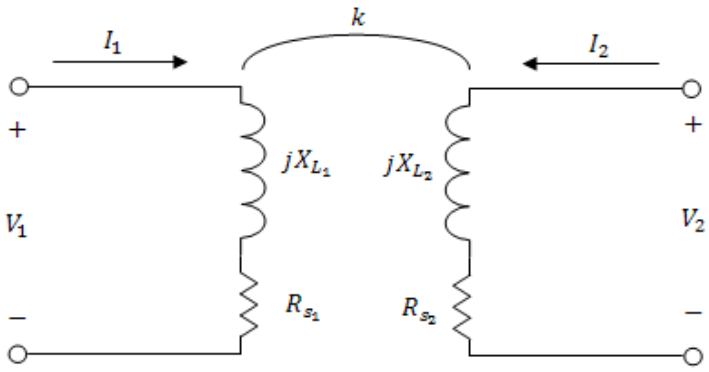


Figure 3.7 – Magnetic Coupling Between Non-Ideal Inductors

To determine the size of resistor needed when only magnetic-based in-phase coupling problems exist, the injected currents within the resonators needed to be examined. Figure 3.7 shows the inductor model used for the analysis. It can be assumed that inductors L_1 and L_2 have identical inductances and series resistances (same Q_0). Figures 3.8 and 3.9 show simplified models for the induced Faraday voltage in each side [2]. V_k and I_N are defined in equations 17 and 18 respectively.

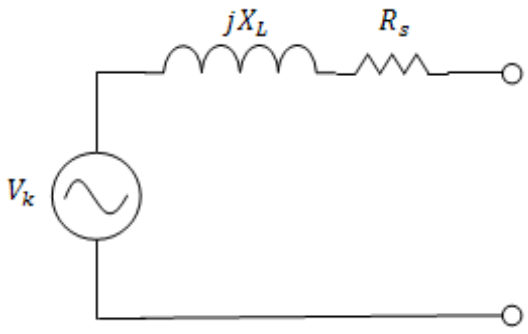


Figure 3.8 – Induced Voltage Model

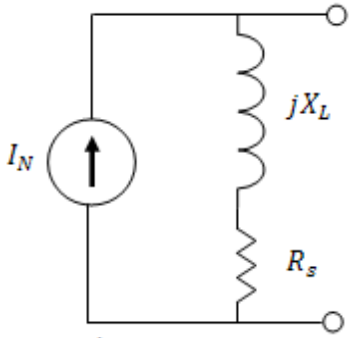


Figure 3.9 – Norton Equivalent of the Induced Voltage Model

Following [15], the voltage induced in each coil through magnetic coupling between L_1 and L_2 is expressed as:

$$V_k = kI_1 jX_L \quad \textbf{Eqn. 17}$$

The induced voltage, modeled as the voltage source in figure 3.8, can also be modeled as a current source. The Norton equivalent (I_N) is shown in figure 3.9 and expressed as:

$$I_N = \frac{V_k}{jX_L + R_s} = kI_1 \left(\frac{jX_L}{jX_L + R_s}\right) \quad \textbf{Eqn. 18}$$

Assuming the mutual inductance is much less than the inductance of L_1 and L_2 ($k << 1$), the current flowing through L_1 can be approximated by:

$$I_1 \cong \frac{V_1}{jX_L + R_s} \quad \textbf{Eqn. 19}$$

Combining equations 18 and 19, and splitting the equation into in-phase and quadrature components yields:

$$I_N \cong kV_1 \left[\frac{2R_s}{4{R_s}^2 + {X_L}^2} - j\frac{X_L}{4{R_s}^2 + {X_L}^2}\right] \quad \textbf{Eqn. 20}$$

The undesired in-phase component of I_N has been defined as α by the following expression:

$$\alpha = \left.\frac{2R_s}{4{R_s}^2 + {X_L}^2}\right|_{X_L \gg R_s} \approx \frac{2R_s}{{X_L}^2} = \frac{2}{Q_0 X_L} \quad \textbf{Eqn. 21}$$

The desired quadrature-phase component of I_N can be defined as β by the following expression:

$$\beta = \left.\frac{X_L}{4{R_s}^2 + {X_L}^2}\right|_{X_L \gg R_s} \approx \frac{1}{X_L} \quad \textbf{Eqn. 22}$$

Combining equations 20-22 yields:

$$I_N \cong kV_1[\alpha - j\beta] \cong kV_1\alpha\left[1 - j\frac{Q_0}{2}\right] \quad \textbf{Eqn. 23}$$

Equation 23 shows that the desired quadrature-phase component is $Q_0/2$ times larger than the in-phase component.

Adding a resistor in parallel with the coupling capacitor was shown to neutralize the asymmetry in simulation. The current through the in-phase coupling resistor R_k can be expressed as:

$$I_{R_k} = \frac{V_1 - V_2}{R_k} = \frac{V_1}{R_k} - \frac{V_2}{R_k} = I_{21} - I_{12} \quad \textbf{Eqn. 24}$$

For full neutralization, the current injected into the second resonator from the first resonator, defined as I_{21}, is found from:

$$I_{21} = \frac{V_1}{R_k} = \alpha k V_1|_{X_L \gg R_s} \approx \frac{2kV_1}{Q_0 X_L} \quad \textbf{Eqn. 25}$$

Assuming that the magnetic coupling coefficient is approximately equal to 1/Q (coupling needed for flat pass-band), the needed resistance to create I_{21} can be found from the following equation:

$$R_k = \frac{Q_0 X_L}{2k}\Big|_{k=1/Q} = \frac{Q(Q_0 X_L)}{2} \quad \textbf{Eqn. 26}$$

The resistor value shown above was placed into simulation in parallel with the coupling capacitors, and the calculated value was found to neutralize the asymmetries as predicted.

3.3.2 In Terms of Coupling Capacitors

Version 2.1 of the Q-Enhanced Filter PCB, documented in Appendix B, was created to reduce the magnetic coupling of the resonators and add pads for coupling resistors. In addition to adding the coupling resistors, high Q_0 inductors were tried to minimize the asymmetries as discussed in section 3.3.1. After the filter was tuned, it was discovered that the asymmetries were still prominent even with high Q_0 inductors. Because the high Q_0 had little effect on the asymmetries, it was believed that the implemented coupling capacitor's non-idealities might produce the asymmetries as well. This section discusses in-phase neutralization in terms of coupling capacitor issues.

The coupling capacitors of the Q-Enhanced filter were implemented with the circuit topology shown in figure 3.10 [13]. The currents I_{in} and I_{out} correspond to I_{12} and I_{21} of the previous section respectively. The coupling capacitors have 5 parallel, binary weighted bits. The capacitance of each bit is C_0*N and the resistance between the inverter and FETs is R_X /N, where C_0 is the capacitance and R_X is the resistance of the least significant bit. When a coupling capacitor is enabled, the circuitry can be modeled as figure 3.11.

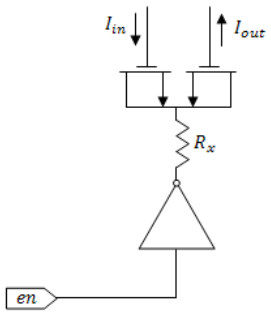


Figure 3.10 – Coupling Capacitor Circuitry

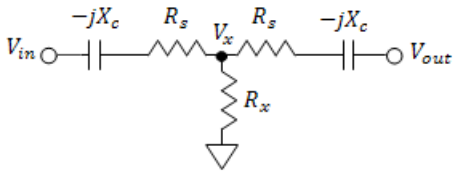


Figure 3.11 – Coupling Capacitor Model

Figure 3.12 shows the coupling capacitor model when R_X is assumed to be infinitely large, and superposition is used to find I_{21} (current injected into side 2 from side 1).

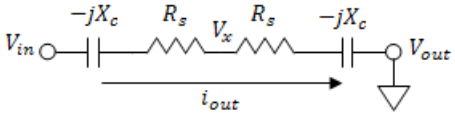


Figure 3.12 – Coupling Capacitor Model without Rx using Superposition to Find I_{21}

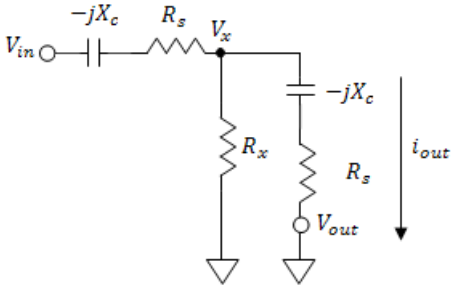


Figure 3.13 – Coupling Capacitor Model with Rx using Superposition to Find I_{21}

When the Q_0 of the capacitor is assumed to be large, the current flowing through the output is purely in quadrature-phase as desired:

$$I_{21} = I_{out} = \frac{V_1}{2(-jX_c + R_s)}\Big|_{X_c \gg R_s} \approx \frac{-V_1}{j2X_c} \qquad \textbf{Eqn. 27}$$

When Rx was considered finite and super position is used, the capacitor model shown in figure 3.13 results. To find I_{21} here, we need to solve first for V_X. The voltage V_X can be expressed as:

$$V_x = \frac{V_1[R_x \parallel (-jX_c + R_s)]}{(-jX_c + R_s) + [R_x \parallel (-jX_c + R_s)]}\Big|_{X_c \gg R_s} \approx \frac{V_1(R_x \parallel -jX_c)}{-jX_c + (R_x \parallel -jX_c)} \qquad \textbf{Eqn. 28}$$

When the Q_0 of the capacitor is assumed to be large, the current I_{21} flowing to the output can be expressed as:

$$I_{21} = \frac{V_x}{-jX_c + R_s}\Big|_{X_c \gg R_s} \approx \frac{V_x}{-jX_c} \qquad \textbf{Eqn. 29}$$

Combining equation 28 and 29, and then solving for admittance yields the following equation. The admittance y_k has been expressed in in-phase and quadrature components as:

$$y_k = \frac{I_{21}}{V_1} = \frac{-R_x}{4{R_x}^2 + {X_c}^2} + j\frac{2{R_x}^2}{X_c\left(4{R_x}^2 + {X_c}^2\right)} \quad \textbf{Eqn. 30}$$

The undesired in-phase component of y_k can be been defined and simplified as α by the following expression:

$$\alpha = \left.\frac{R_x}{4{R_x}^2 + {X_c}^2}\right|_{R_x \gg X_c} \approx \frac{1}{4R_x} \quad \textbf{Eqn. 31}$$

The desired quadrature-phase component of y_k can be defined and simplified as β by the following expression:

$$\beta = \left.\frac{2{R_x}^2}{X_c\left(4{R_x}^2 + {X_c}^2\right)}\right|_{R_x \gg X_c} \approx \frac{1}{2X_c} \quad \textbf{Eqn. 32}$$

The output current can now be expressed as:

$$i_{21} = V_1\alpha\left(-1 + j\frac{2R_x}{X_c}\right) \quad \textbf{Eqn. 33}$$

Equation 34 clearly demonstrates that the in-phase current component can be minimized by using large R_X values. After examination of the coupling capacitor circuitry, it was clear that insufficiently large R_X values were used, thus the current includes an in-phase term.

Adding a resistor in parallel with the coupling capacitor circuitry was found to neutralize the asymmetries caused by the non-idealities of the inductors used within the filter's resonators. The in-phase neutralization approach applies to the in-phase current component introduced by the insufficiently sized resistance R_X found within the coupling capacitor circuitry as discussed in section 3.3.3.

3.3.3 In-Phase Neutralization Analysis Using Y-Parameters

Using Y-parameters as a means of two-port network analysis, figure 3.14, allows the in-phase neutralization process to be formalized. Y-parameter definitions can be found in equations 34-37.

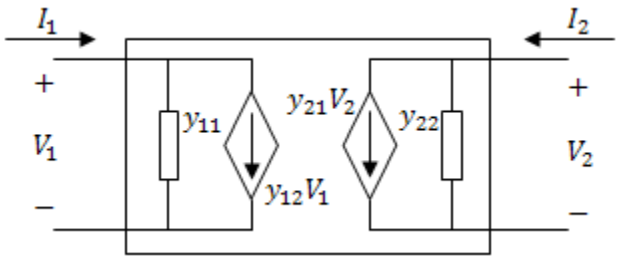


Figure 3.14 – Y-Parameter Two-Port Network

Input Port Admittance

$$y_{11} = \frac{I_1}{V_1}\Big|_{V_2=0} \qquad \textbf{Eqn. 34}$$

Reverse Transfer Admittance

$$y_{12} = \frac{I_1}{V_2}\Big|_{V_1=0} \qquad \textbf{Eqn. 35}$$

Forward Transfer Admittance

$$y_{21} = \frac{I_2}{V_1}\Big|_{V_2=0} \qquad \textbf{Eqn. 36}$$

Output Port Admittance

$$y_{22} = \frac{I_2}{V_2}\Big|_{V_1=0} \qquad \textbf{Eqn. 37}$$

The coupling capacitor circuitry shown in figure 3.11 can be split into the equivalent circuit model shown in figure 3.14. Y_{11} of the coupling capacitor circuitry is found by grounding V_2 and taking the division of I_1 and V_1 as shown by equation 38. The series resistance of the capacitors have been assumed negligible, thus y_{11} has been approximated.

$$y_{11} = \frac{I_1}{V_1}\Big|_{V_2=0} \approx [-jX_c + (R_x \parallel -jX_c)]^{-1} \qquad \textbf{Eqn. 38}$$

Y_{21} of the coupling capacitor circuitry is found by grounding V_2 and taking the division of I_2 and V_1, as performed in equation 39. Using the definitions of α and β found in equations 32 and 33, Y_{21} can be expressed as:

$$y_{21} = \frac{I_2}{V_1}\Big|_{V_2=0} \approx \alpha\left(-1 + j\frac{2R_x}{X_c}\right) \qquad \textbf{Eqn. 39}$$

By symmetry Y_{22} is equivalent to Y_{11}, and Y_{12} is equivalent to Y_{21}.

To have a symmetrical pass-band the real valued in-phase portion of y_{21} should be zero, but as equation 39 states, a real portion approximately equal to $-\alpha$ exists.

To cancel or neutralize this real part, we can add a second network between sides 1 and 2, find its Y-parameter representations and then add its Y_{21} to the Y_{21} in equation 39. Since we want the second network to have a real valued Y_{21}, a resistor was a good candidate.

Y_{11} of a resistor is found by grounding V_2 and taking the division of I_1 and V_1 as shown by:

$$y_{11} = \frac{I_1}{V_1}\Big|_{V_2=0} = \frac{1}{R} \qquad \textbf{Eqn. 40}$$

Y_{21} of a resistor is found by grounding V_2 and taking the division of I_2 and V_1 as shown by:

$$y_{21} = \frac{I_2}{V_1}\Big|_{V_2=0} = -\frac{1}{R} \qquad \textbf{Eqn. 41}$$

Again, by symmetry Y_{22} is equivalent to Y_{11}, and Y_{12} is equivalent to Y_{21}.

Placing a correctly sized resistor in parallel with the coupling capacitor neutralizes the asymmetry by canceling the in-phase component found in Y_{21} of the coupling capacitor circuitry. Kirchhoff's current law states that parallel currents add, thus the following equality must be satisfied in order for the in-phase component to be canceled.

$$\frac{1}{R} = |\alpha| \qquad \textbf{Eqn. 42}$$

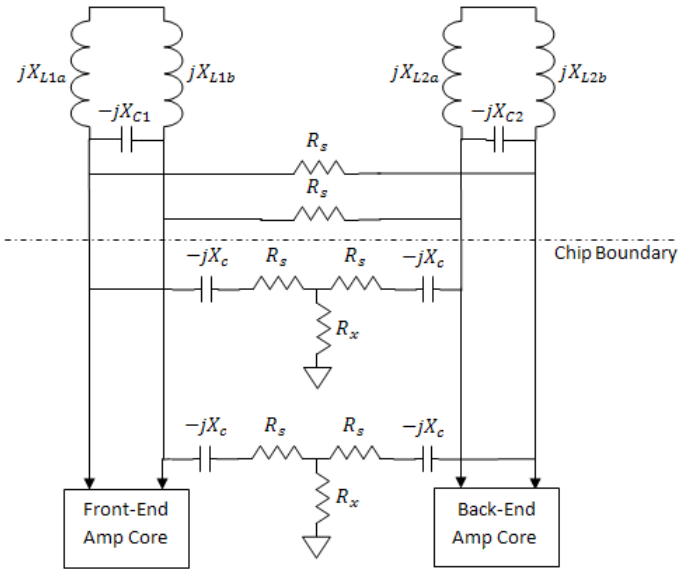


Figure 3.15 – Coupling Capacitor and Cross Connected Resistor Implementation Circuit Model

Because the in-phase components of Y_{21} in equations 39 and 41 are both negative, the polarity of Y_{21} in equation 41 needs to be changed by crossing the connections of the resistor with respect to the connections of the coupling capacitor circuitry as demonstrated in figure 3.15. This is easily achieved due to the differential circuits used in the resonator cores. With the polarity of Y_{21} of equation 41 now switched, summing the currents will cancel the undesired in-phase term.

Due to the presence of Y_{11} in equations 38 and 41, offsets in frequency and q-enhancement will occur, but will be corrected when the filter is tuned.

Section 4 - PROPOSED TUNING ALGORITHMS

Apart from solving the pass-band asymmetry problem discussed in the previous section, the main focus of this research has been to build and demonstrate a practical self-tuning approach for multi-pole filters. This section focuses on that issue, beginning with developing a single-pole algorithm followed by a two-pole algorithm.

4.1 SINGLE-POLE TUNING ALGORITHM

To gain a better understanding of the logic needed to tune the filter, a single-pole tuning algorithm was defined and implemented. A flow chart of the main tuning algorithm can be found in figure 4.2 (in two pages). The Algorithm Settings form of the Q-Enhanced Filter Test Application, documented in Appendix D.1.6, was used to provide inputs including Center Frequency, Frequency Tolerance, Amplitude Detector Threshold, Q-Back-Off, Q-Offset, and Frequency Offset.

4.1.1 DESCRIPTION

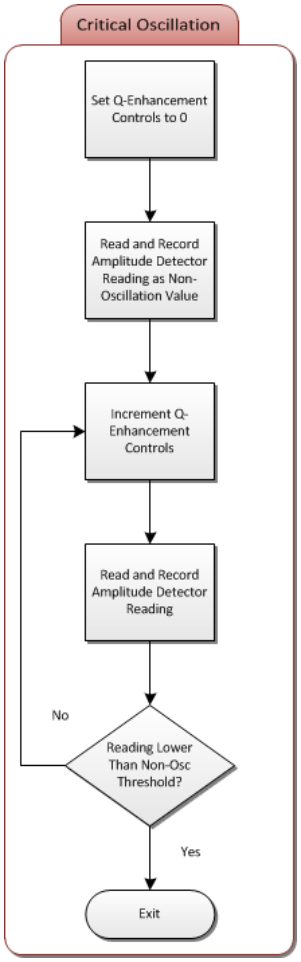


Figure 4.1 – Single-Pole Critical Oscillation Flowchart

The filter is initialized by setting all controls to their minima and disabling the Front-End, Back-End, amplitude detectors, and frequency dividers. The Front-End is used as the single-pole because the inputs are tied to the Front-End's differential amplifier core. After initialization, the frequency controls are set to their midpoint, which places the pole close to the desired center frequency at room temperature. The filter must be brought to oscillation so that the frequency of the resonator can be measured. Since this frequency will depend somewhat on the amount of Q-Enhancement over and above the critical value needed to oscillate, it is important to find this critical value.

Figure 4.1, left, shows a flow chart for finding critical oscillation. The Q-Enhancement controls are set to the minimum, and the amplitude detector is read and stored as the non-oscillation reading. Q-Enhancement controls are increased in a linear, single-step fashion until the amplitude detector reading has dropped below a threshold, which is based upon the non-oscillation reading.

Bringing the Q-Enhancement controls to the critical oscillation level provides an input signal to the frequency dividers. This signal allows the frequency at which the pole is located to be read as shown in figure 4.2 on the following page. The frequency divider output is connected to a frequency counter implemented on the microcontroller. If the frequency count is outside the frequency tolerance, the frequency controls are adjusted appropriately, critical oscillation is found, and the frequency count is again compared to the frequency tolerance. When the frequency count is within the frequency tolerance, the frequency controls are again adjusted and compared to the previous count. If the previous count was closer to the desired center frequency, the controls are set to their prior value.

The Q-Enhancement controls are backed-off according to the measured data shown in figure 4.3 to obtain the desired bandwidth. Finally, the frequency controls are offset to counter the frequency shift caused by the Q-Enhancement back-off. The tuning algorithm then waits until the Q-Enhanced Filter Test Application triggers it to run again.

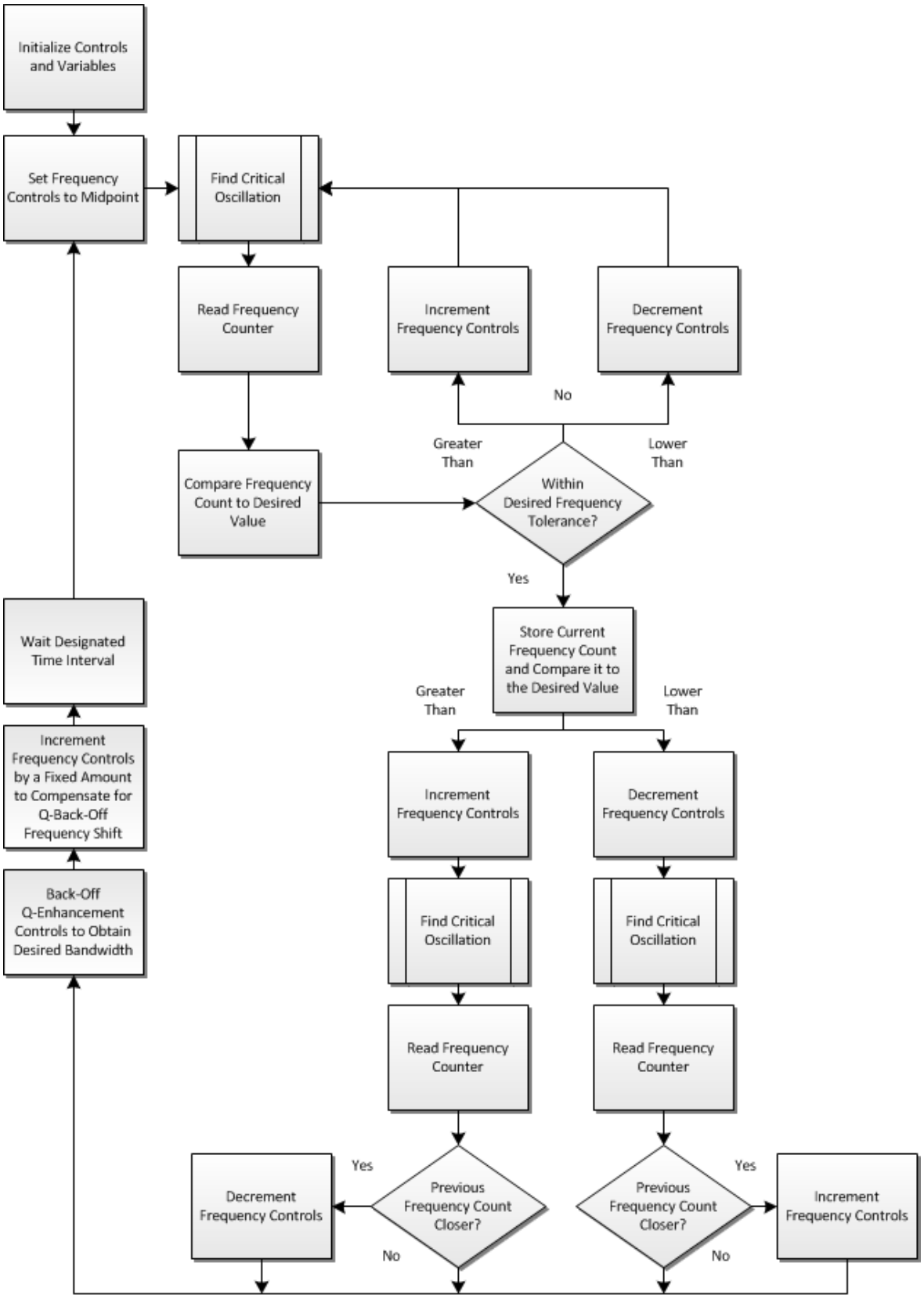


Figure 4.2 – Single-Pole Tuning Algorithm Flowchart

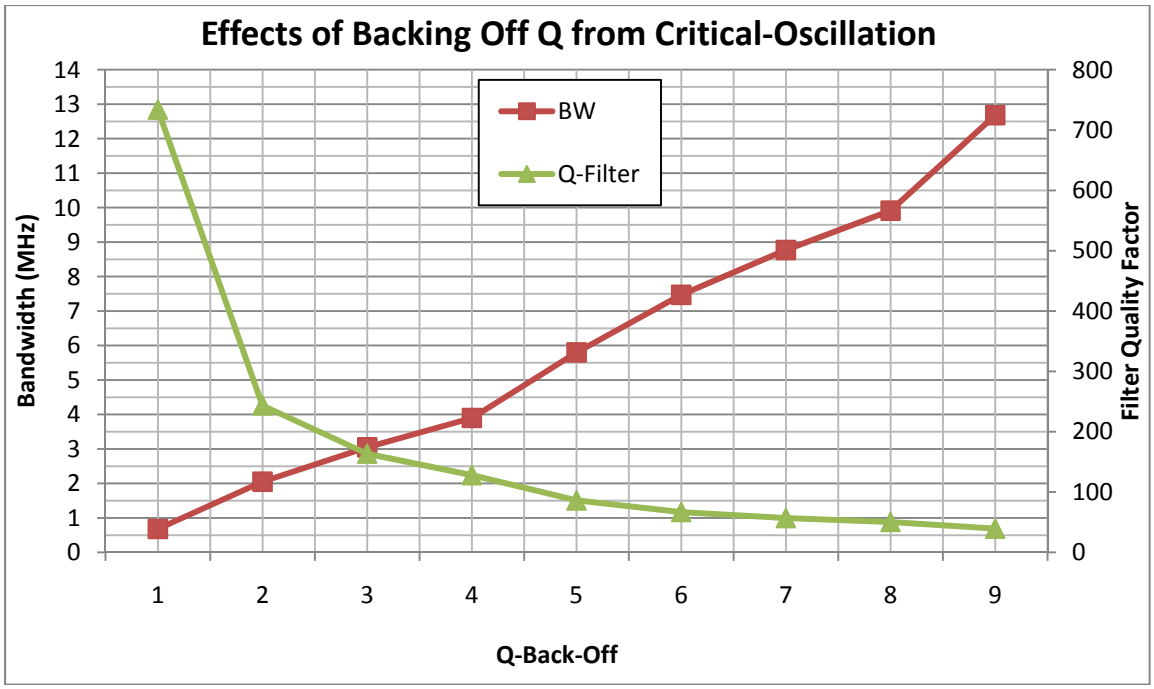


Figure 4.3 - Effects of Backing Off Q from Critical-Oscillation

4.1.2 Results

The single-pole version of the filter tuning algorithm provided much insight into the implementation of a two-pole version of the algorithm.

Figure 4.4 – Single-Pole Filter Response Video Clip (Vertical scale is 2 dB/division)

Click figure 4.4 to play an embedded video clip of the single-pole 500 MHz filter with 2 MHz bandwidth observed through the Back-End. The filter tuning algorithm is run once per second while a heat gun is blown on the filter chip. This video clip clearly demonstrates that the filter gain is held within 1 dB across temperature variations. The constant gain implies that bandwidth is also constant across temperature variations. Occasionally, small glitches in center frequency occur. The center frequency glitches are a product of frequency divider output instability as discussed in Section 5.2. The two-pole tuning algorithm, discussed in Section 4.2, uses much of the same tuning methodology developed in the single-pole tuning algorithm.

4.2 TWO-POLE TUNING ALGORITHM

The single-pole tuning algorithm was adapted into a two-pole tuning algorithm as shown in figure 4.5. Details of blocks with vertical lines in them are shown in companion charts as described below.

4.2.1 DESCRIPTION

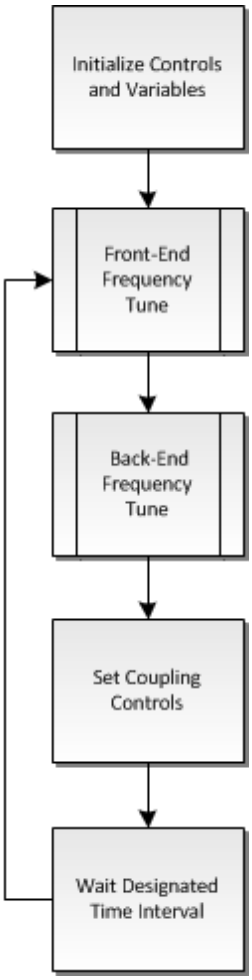


Figure 4.5 - Two-Pole Tuning Algorithm Flowchart

As noted in figure 4.5, the filter is initialized by setting all controls to their minima, enabling the Front-End and Back-End and disabling the amplitude detectors and frequency dividers. Once the filter is initialized, Front-End and then Back-End frequency tuning is performed using the sub-flowchart shown in figure 4.6. When both poles are set to the desired center frequency, the predetermined coupling settings are applied according to the desired bandwidth. For a multi-bandwidth filter, these values can be stored within a look-up table, since test results reported in Section 4.2.2 confirm that they are not strung functions of temperature.

Frequency tuning, detailed in figure 4.6, is similar to the single-pole tuning algorithm. The frequency controls are set to their midpoint, which places the pole close to the desired center frequency at room temperature.

The Get Frequency sub-flowchart is shown in figure 4.7. When reading the Front-End frequency divider, the Back-End Q-Enhancement level is first stored and then set to its minimum. This allows the loading of the Front-End to be minimized. The Front-End is then brought to critical oscillation and the Front-End frequency count is read and stored. The excess Q-Enhancement is removed from the Front-End, and the Back-End Q-Enhancement is restored to its prior level. A similar process is used for obtaining the Back-End frequency count.

The frequency divider output is connected to a frequency counter on the microcontroller. If the frequency count is outside the frequency tolerance, the frequency controls are adjusted appropriately, critical oscillation is found, and the frequency count is again compared to the frequency tolerance. When the frequency count is within the frequency tolerance, the frequency controls are again adjusted and compared to the previous count. If the previous count was closer to the desired center frequency, the controls are set to their prior value.

Figure 4.8 shows the sub-flowchart for finding critical oscillation. The Q-Enhancement controls are set to the minimum, and the amplitude detector is read and stored as the non-oscillation reading. Q-Enhancement controls are increased in a linear, single-step fashion until the amplitude detector reading has dropped below a predefined threshold, which is based upon the non-oscillation reading. At this point, first oscillation has been found, but an additional Q-Offset is needed to increase Q-Enhancement past first oscillation to stabilize the frequency divider reading. The Q-Enhancement setting which gives the stable frequency divider reading is known as critical oscillation.

Finally, the Q-Enhancement controls are backed-off a coupling dependent to achieve the desired resonator Q according to figure 4.3, and the frequency controls are offset to counter the frequency shift caused by the Q-Enhancement back-off. The tuning algorithm then waits until the Q-Enhanced Filter Test Application triggers it to run again.

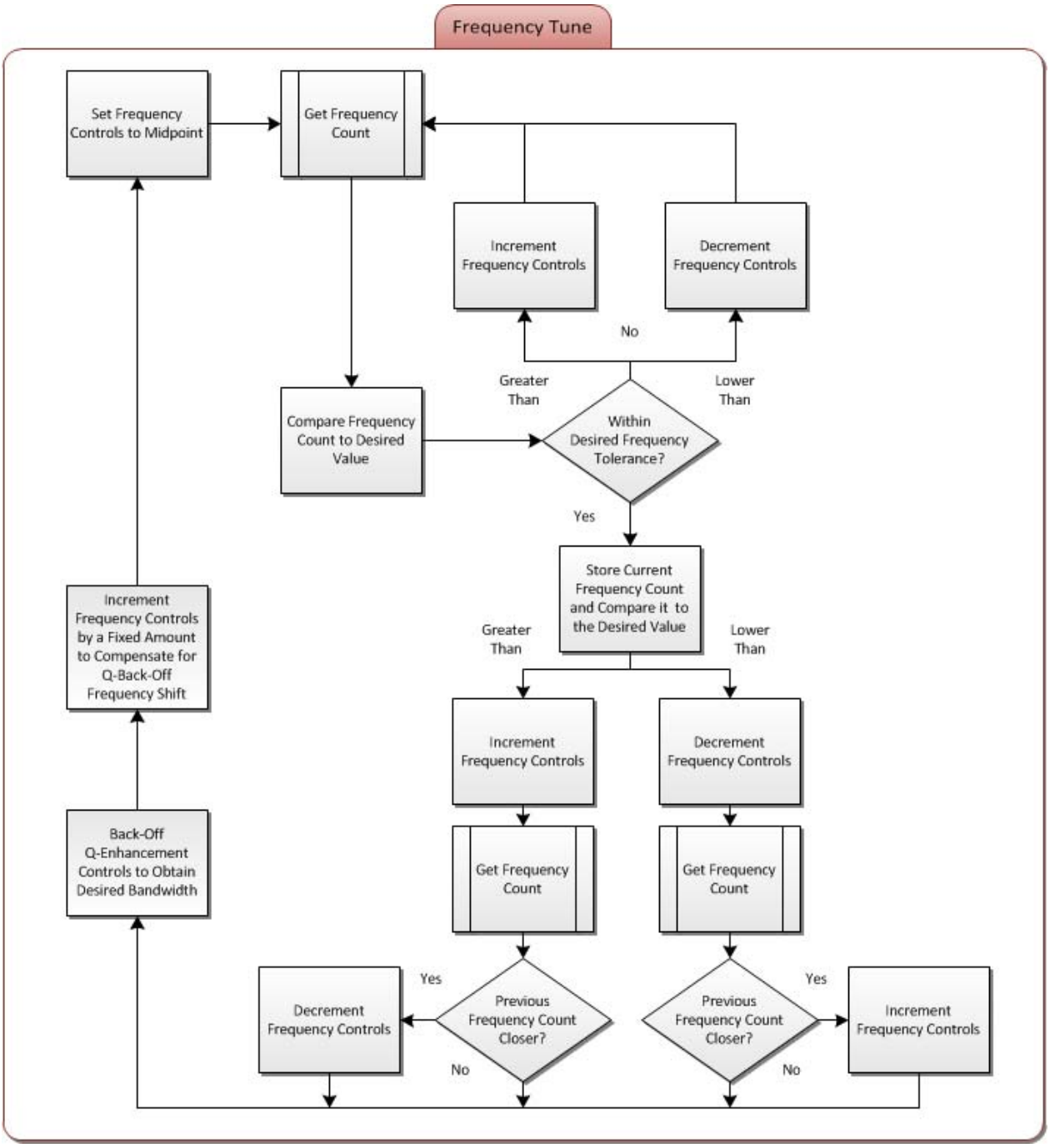


Figure 4.6 – Two-Pole Tuning Algorithm Frequency Tune Flowchart

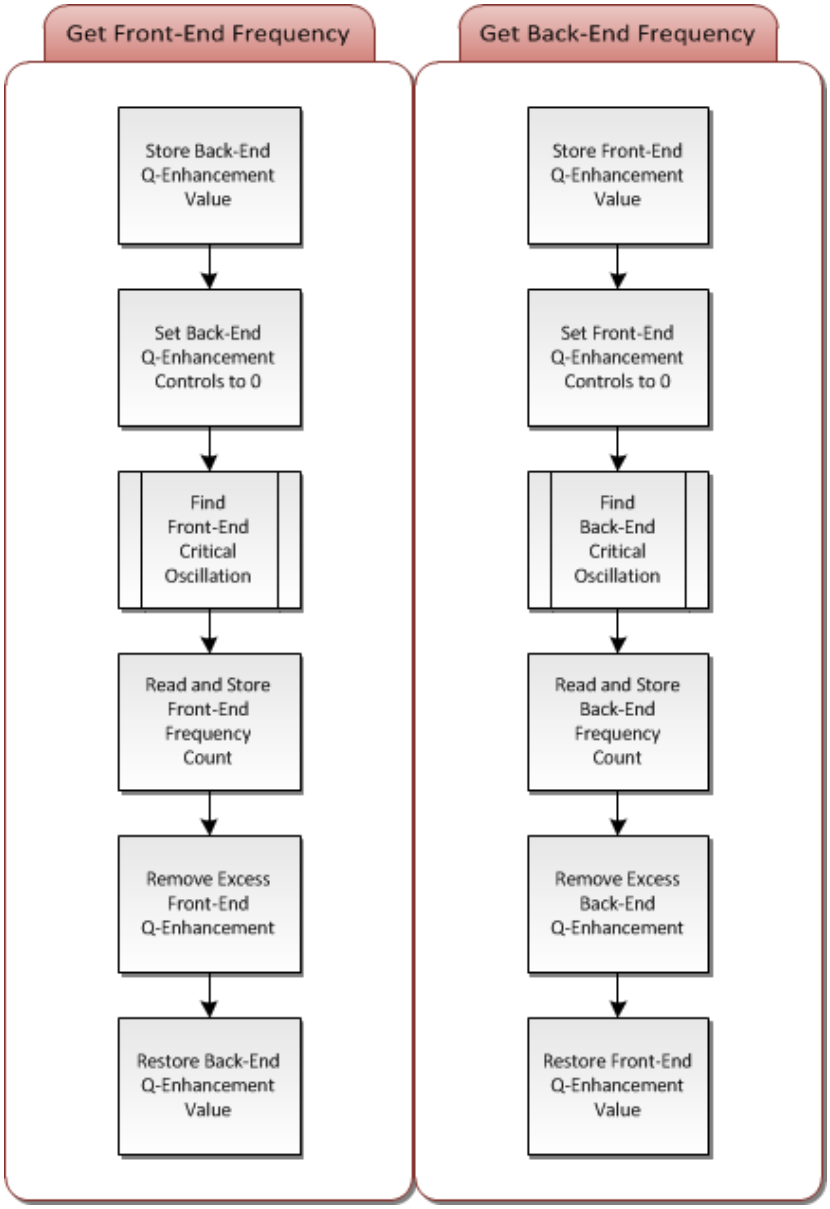


Figure 4.7 - Two-Pole Tuning Algorithm Get Frequency Count Flowchart

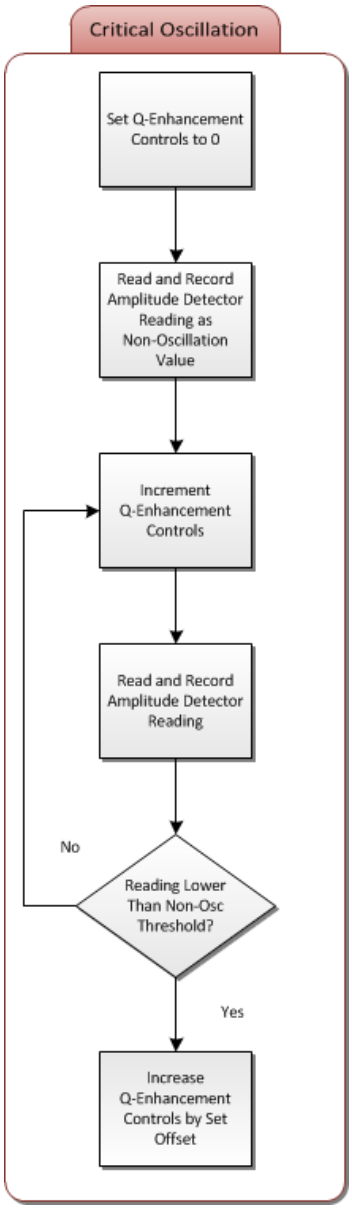


Figure 4.8 - Two-Pole Tuning Algorithm Critical Oscillation Flowchart

4.2.2 Results

The two-pole tuning algorithm was implemented on version 2.1 of the Q-Enhanced Filter PCB, documented in Appendix B. Due to unforeseen board level capacitances and problems with the Front-End frequency divider, a 500 MHz center frequency could not be achieved with the 3.9 nH high Q_0 inductors used within the filter resonators. Because of the limited working range of the Front-End frequency divider, a 450 MHz center frequency was chosen.

Table 4.1 shows the *single-tuned* response of a 10 MHz bandwidth filter heated across a temperature range of 25-55°C at various frequency spans without coupling neutralization. For these plots, automatic tuning is turned off and all controls are held at fix values obtained by running the algorithm once at 25°C.

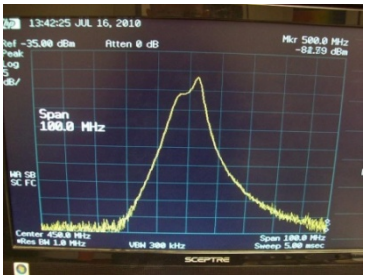
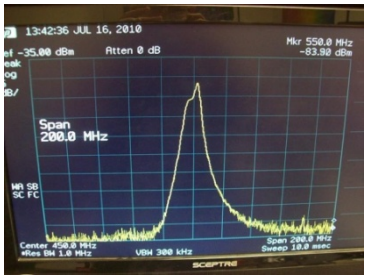
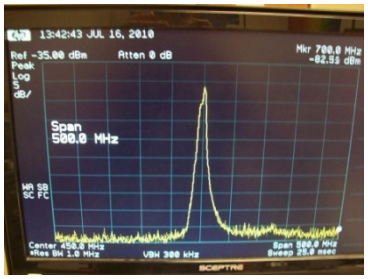
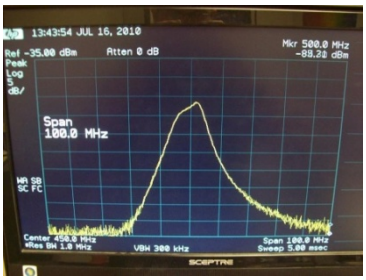
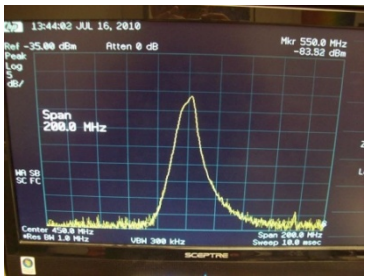
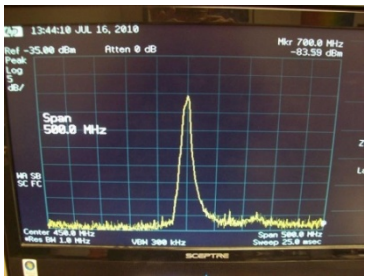
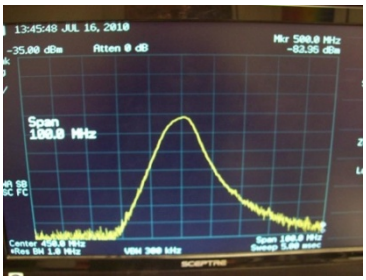
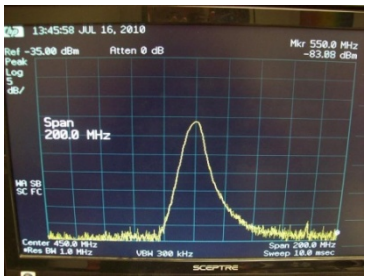
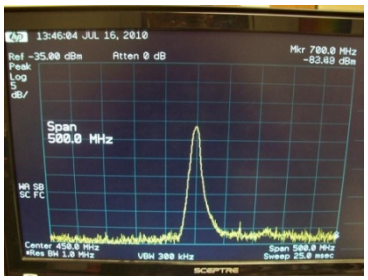
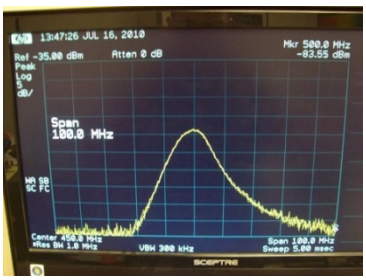
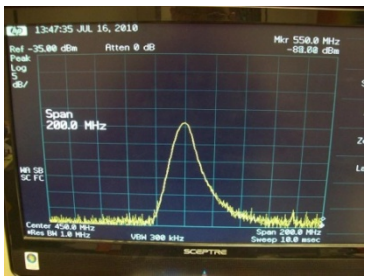
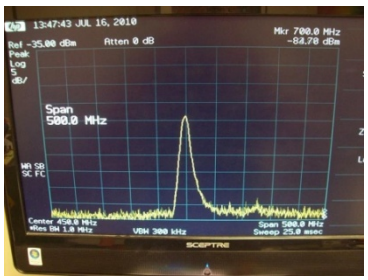
Single-Tuned 10 MHz Bandwidth Filter Responses Over Temperature Variations			
T (°C)	**100 MHz Span**	**200 MHz Span**	**500 MHz Span**
25			
35			
45			
55			

Table 4.1 – Single-Tuned 10 MHz Bandwidth Filter Responses Over Temperature Variations

Several observations can be made from the responses shown in table 4.1. The pass-band asymmetries, discussed in Section 3, are present due to the circuitry used to implement the coupling capacitors. The

asymmetries would be more prevalent if high Q_0 inductors had not been used. Filter-Q and pass-band flatness degrade as temperature increases clearly demonstrating the need for a real-time tuning algorithm.

Table 4.2 shows the continuously auto-tuned response of a 10 MHz bandwidth filter heated across a temperature range of 25-55°C at various frequency spans without coupling neutralization.

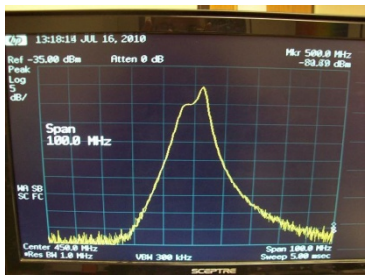
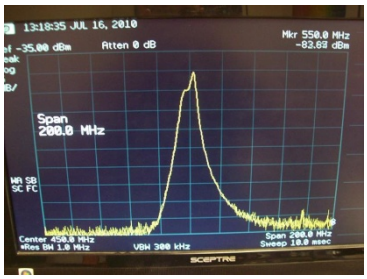
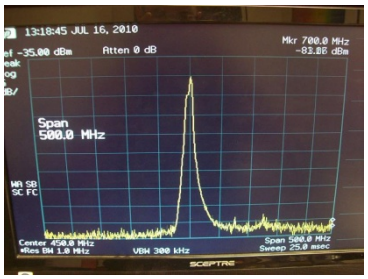
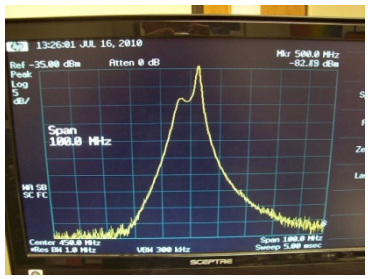
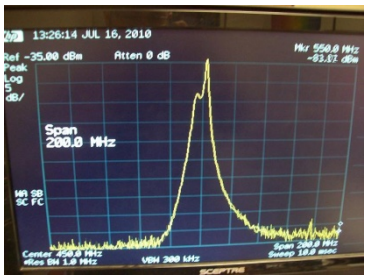
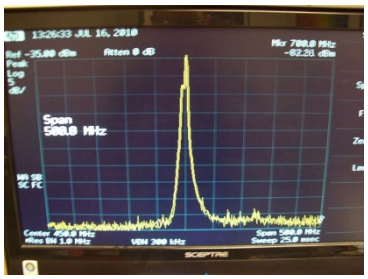
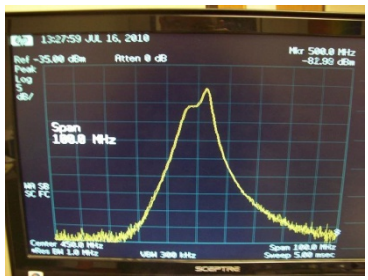
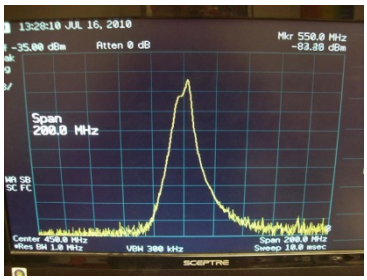
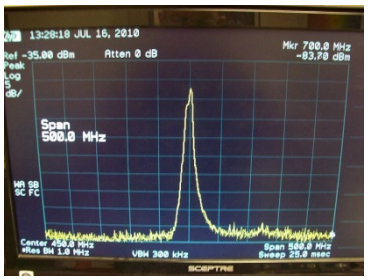
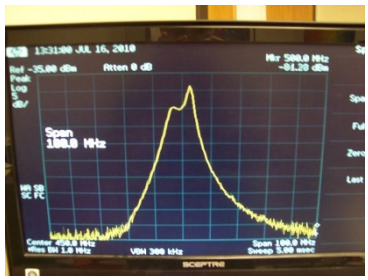
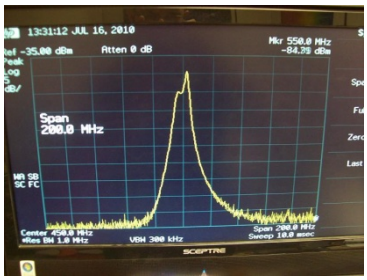
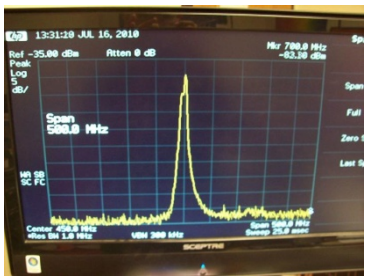
Auto-Tuned 10 MHz Bandwidth Filter Responses Over Temperature Variations			
T (°C)	**100 MHz Span**	**200 MHz Span**	**500 MHz Span**
25			
35			
45			
55			

Table 4.2 – Auto-Tuned 10 MHz Bandwidth Filter Responses Over Temperature Variations

The real-time algorithm implementation successfully maintains selectivity-Q and frequency across temperature variations. However, the pass-band varies by small amounts. The response in the pass-band is more sensitive to Q-Enhancement at narrower bandwidths and finer digital controls are needed.

Table 4.3 shows the single-tuned response of a 20 MHz bandwidth filter heated across a temperature range of 25-55°C at various frequency spans. The filter was tuned at 25°C and controls then held fixed. Two 32 kΩ resistors were found to neutralize the pass-band asymmetries for a 20 MHz bandwidth.

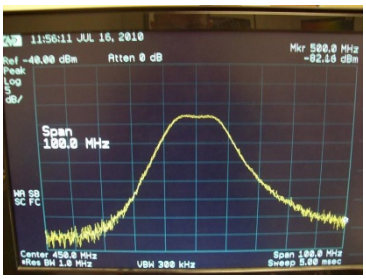
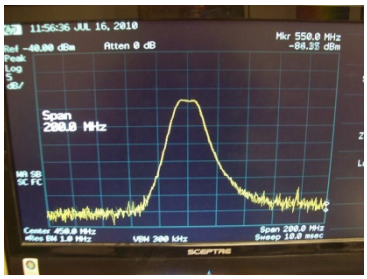
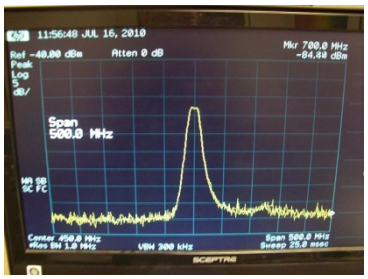
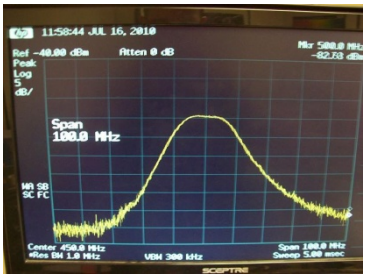
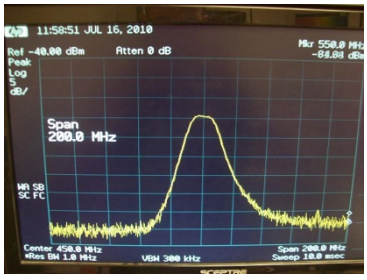
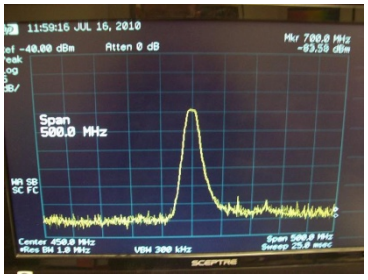
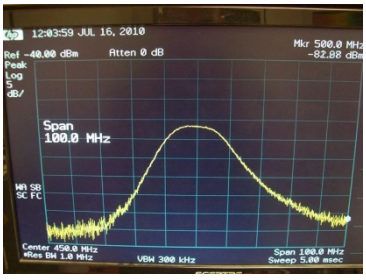
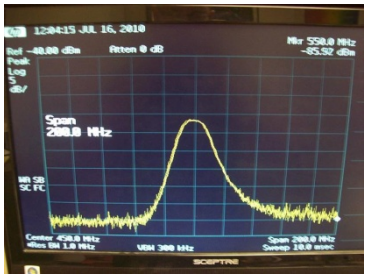
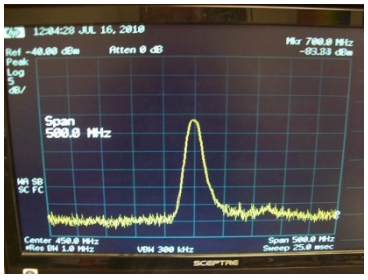
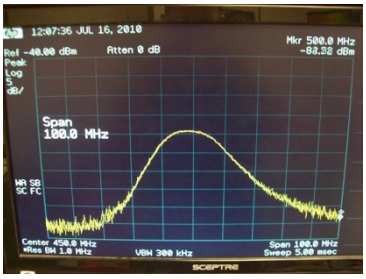
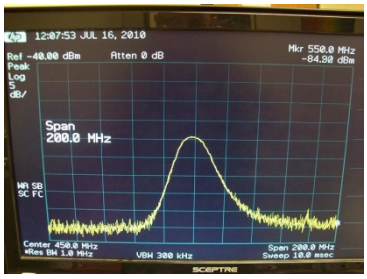
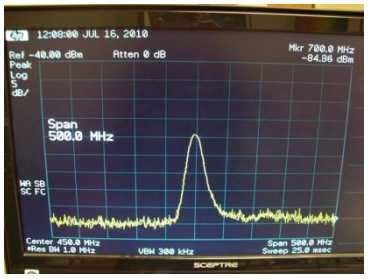
Single-Tuned 20 MHz Bandwidth Filter Responses Over Temperature Variations			
T (°C)	**100 MHz Span**	**200 MHz Span**	**500 MHz Span**
25			
35			
45			
55			

Table 4.3 – Single-Tuned 20 MHz Bandwidth Filter Responses Over Temperature Variations

Again, degradations of selectivity-Q and pass-band flatness as temperature increases clearly demonstrate the need for a real-time tuning algorithm.

Table 4.4 shows the continuously auto-tuned response of a 20 MHz bandwidth filter heated across a temperature range of 25-55°C at various frequency spans. Again, two 32 kΩ resistors were used to neutralize the pass-band asymmetries at a 20 MHz bandwidth.

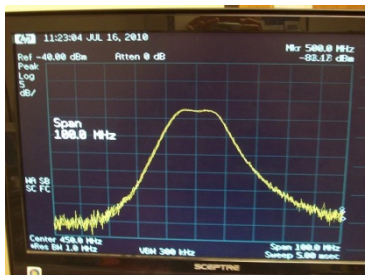
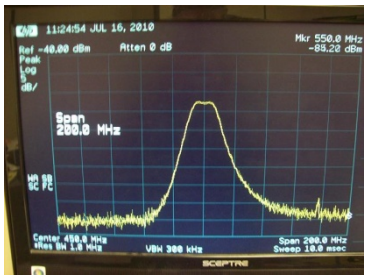
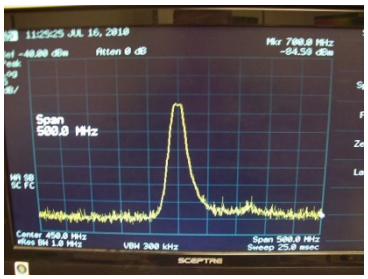
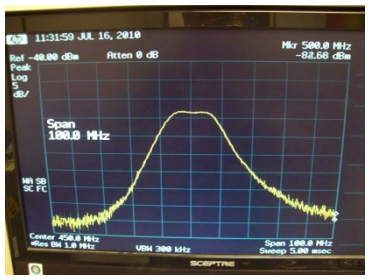
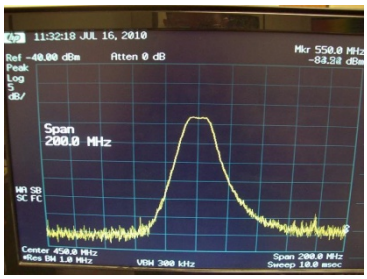
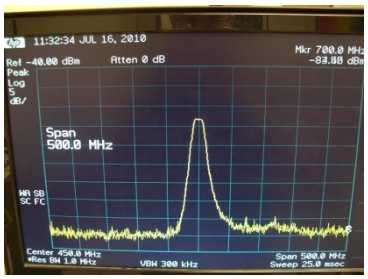
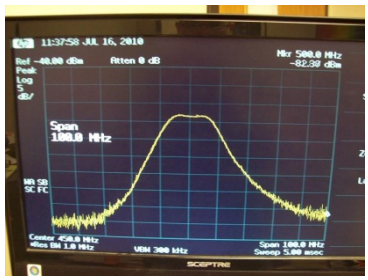
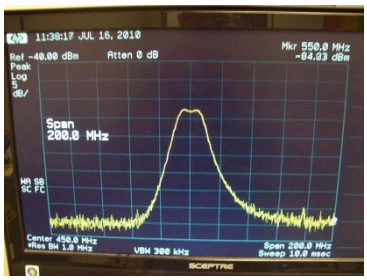
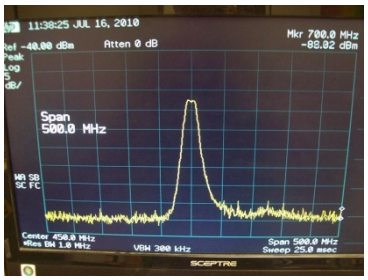
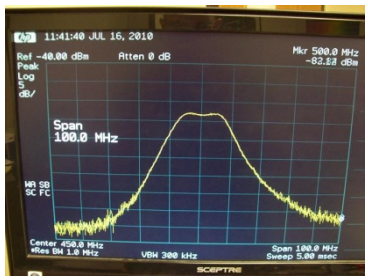
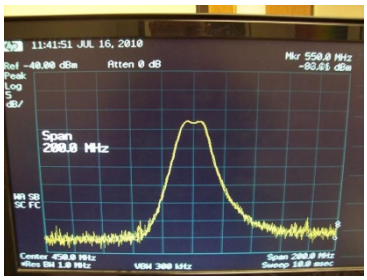
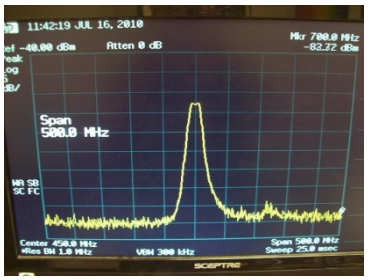
Auto-Tuned 20 MHz Bandwidth Filter Response Over Temperature Variations			
T (°C)	**100 MHz Span**	**200 MHz Span**	**500 MHz Span**
25			
35			
45			
55			

Table 4.4 – Auto-Tuned 20 MHz Bandwidth Filter Responses Over Temperature Variations

Clearly, the desired bandwidth and center frequency can be maintained over temperature variations with real-time implementations of the proposed tuning algorithm.

Table 4.5 shows the auto-tuned response of a 10 MHz bandwidth filter at various frequency spans and a 10 dB/division scale. Two 2 kΩ resistors were used to neutralize the pass-band asymmetries.

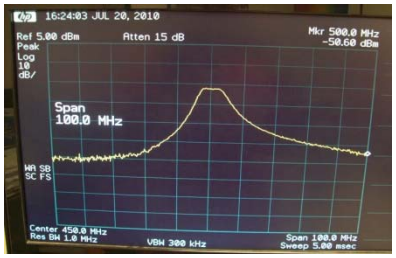
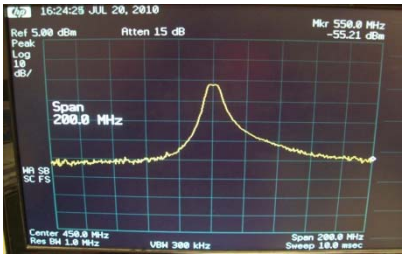
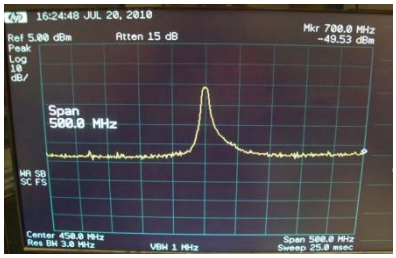
Auto-Tuned 10 MHz Bandwidth Filter Response with Asymmetry Neutralization		
100 MHz Span	**200 MHz Span**	**500 MHz Span**
		

Table 4.5 – Auto-Tuned 10 MHz Bandwidth Filter Responses with Asymmetry Neutralization

Click figure 4.11 to play an embedded video of the Auto-Tuned 10 MHz filter across temperature variations of 25-75°C. The video clip makes it clear the bandwidth, gain, and center frequency can be held extremely stable over temperature variations.

Figure 4.9 - Auto-Tuned 10 MHz Bandwidth Filter Response Over Temperature Variations from 25-75˚C Video Clip

Algorithm settings for 20 MHz and 10 MHz bandwidths have been documented in table 4.6. Note that the Q-Back-Off settings are consistent with the bandwidths given by figure 4.3 after taking into account a factor of approximately 1.5 due to the two-pole versus single-pole bandwidth increase expected for coupled-resonator designs.

Algorithm Settings		
Bandwidth	**20 MHz**	**10 MHz**
Center Frequency	450.0 MHz	
Frequency Tolerance	0.2 MHz	
AD Threshold 1	10	10
AD Threshold 2	10	10
Q-Offset	3	3
Q-Back-Off	9	5
F-Offset	3	3
Capacitive Upper Tuning	15	10
Capacitive Lower Tuning	15	10
Capacitive UFLB Tuning	0	0
Capacitive LFUB Tuning	0	0

Table 4.6 – Algorithm Settings for 20 MHz and 10 MHz Bandwidths

4.2.3 Optimization

As previously noted, the success of tuning depends on how narrow the bandwidth needs to be, or equivalently, on the level of Q-Enhancement used. For the plots shown above, only the digital tuning controls were used, so that there is a fixed resolution to the algorithms adjustment. To hold the responses tighter, especially if narrower bandwidths are desired, algorithm optimization could be implemented by separating frequency and Q-Enhancement tuning into coarse and fine tuning. Redundancy checks are needed to prevent the filter from entering infinite loops when invalid readings occur. The improvements discussed in Sections 5.2-3 will aid in eliminating the occurrence of invalid readings.

To improve the speed of tuning, coarse frequency tuning could be implemented by using binary searches to find a frequency setting within the defined frequency tolerance. Fine frequency tuning would be implemented by using the linear, stepping fashion. Similarly, coarse tuning of Q-Enhancement could be used to find critical oscillation. A binary search of the Q-Enhancement controls would yield an amplitude detector reading between AD Threshold 1 and AD Threshold 2. Fine Q-Enhancement tuning would use a linear, stepping fashion of Q-Enhancement controls to find the optimal setting for critical oscillation.

4.2.4 Dynamic Range and Power Consumption

Table 5.1 shows dynamic range and power-consumption measurements of 20 MHz and 10 MHz bandwidth filters centered at 450 MHz.

Filter Specifications verse Bandwidth		
Bandwidth	**20 MHz**	**10 MHz**
Gain	10 dB	17 dB
Input Referred 1 dB Comp. Pt.	-20 dBm	-25 dBm
Output Referred 1 dB Comp. Pt.	-10 dBm	-8 dBm
Noise Figure	14 dB	14 dB
Dynamic Range	140 dB/Hz	135 dB/Hz
Supply Voltage	2.5 V	
Supply Current of Tuned Circuit		
With 50 Ω Buffers	34 mA	
Without Buffers	10 mA	

Table 5.7 – Filter Specifications verse Bandwidth

The noise figure measurements, quoted in table 5.1, were obtained by the following process. The filter was tuned to 450 MHz, and the filter output was amplified by 40 dB without an input signal. Figures 5.2-3 show the noise floors of the filter responses. Once the noise floor was amplified above the spectrum analyzer's noise floor, the in-band noise floors power measurements were obtained at a 100 kHz resolution bandwidth. The measurements were converted to a 1 Hz bandwidth by subtracting 50 dB, making them in terms of dBm/Hz. The 40 dB amplifier gain and filter gain was subtracted to get the input referred noise floor from the output referred noise floor. The difference between the measured noise floor and the -174 dBm/Hz thermal noise floor was taken to obtain the noise figure. The dynamic range was then calculated by taken the difference of the measured noise floor and 1 dB compression point.

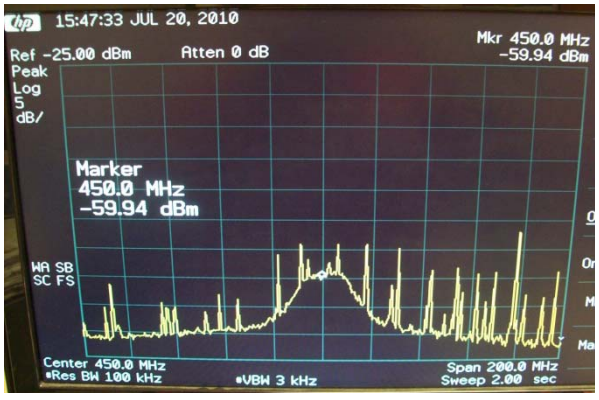


Figure 5.10 – Noise Floor of 20 MHz Bandwidth Filter

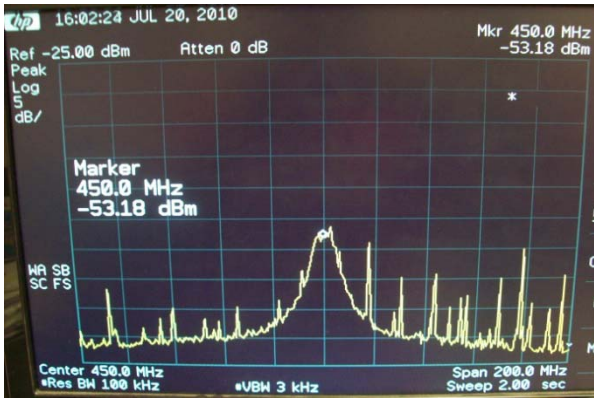


Figure 5.11 – Noise Floor of 10 MHz Bandwidth Filter

Section 5 - CONCLUSION

5.1 PERFORMANCE OVERVIEW

To illustrate the overall operation of the filter, figure 5.1 shows the filter response at various Q-Enhancement levels. The tuning algorithm was run once to tune the poles to 450 MHz. Q-Enhancement was manually adjusted to create the responses shown. The violet filter response shows the base-Q of the filter. The bandwidth is on the order of 45 MHz, indicating each pole has a bandwidth on the order of 30 MHz. The filter response in blue shows moderate levels of Q-Enhancement. The filter response in yellow shows the needed Q-Enhancement for a 10 MHz bandwidth, flat pass-band, and moderate gain.

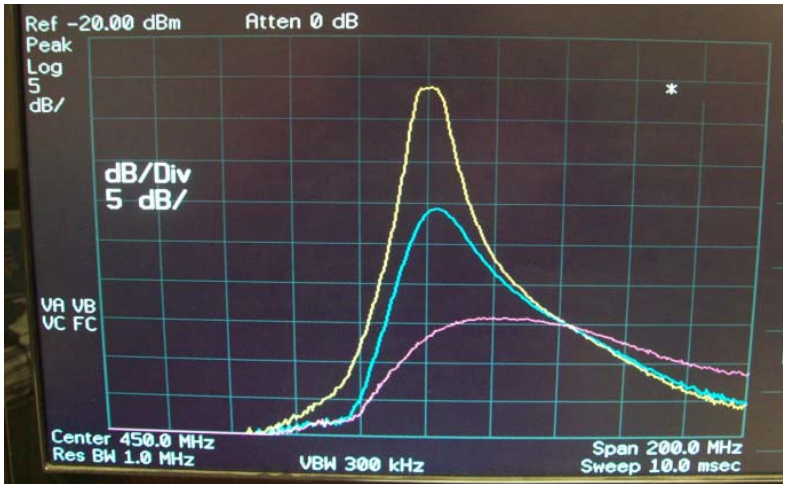


Figure 5.1 – Filter Responses at Various Q-Enhancement Levels

With real-time tuning enabled, the filter was able to control 20 MHz and 10 MHz bandwidth filters at 450 MHz with excellent performance over temperature. As mentioned earlier, due to the size of the inductors used and a problem with the Front-End frequency divider, 450 MHz was chosen as the desired center frequency instead of the nominal 500 MHz value planned. Nevertheless, the filter is able to tune over different frequencies in addition to being able to maintain responses over environmental changes. Bandwidths as narrow as 1 MHz should be achievable, but doing so in the current realization requires changing the resistors used in asymmetry neutralization. In a future realization with in-phase neutralization controls on-chip, the results obtained here suggest that a one-time trim of these setting should be sufficient.

5.2 Remaining Issues

The largest problem encountered by the filter tuning algorithm was related to the frequency dividers. The frequency dividers could only be read when the respective pole was brought to oscillation by sufficient Q-Enhancement. Ideally, the frequency divider output should be read when the filter first starts to oscillate to give the most accurate frequency reading. Unfortunately, to get a stable and accurate frequency reading, the pole needs to be enhanced past the first-oscillation Q-Enhancement level by a set Q-Offset.

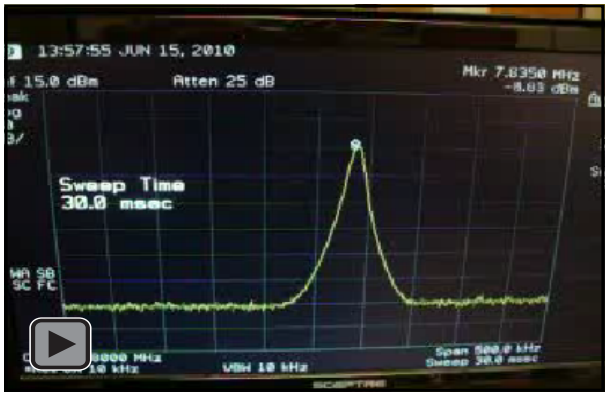


Figure 5.2 – Frequency Divider Output Spectrum Video Clip

Click figure 5.1 to play an embedded video of the frequency divider output spectrum. The frequency dividers offer a division of 64, so a divided 500 MHz signal appears at 7.8125 MHz. The video clip shows a linear back-off of Q-Enhancement from the maximum level to the point where the filter is no longer oscillating. This video clip makes it clear that applying a Q-Offset past first-oscillation will provide a clean spectrum allowing a stable and accurate frequency reading. Figures 5.2-5.3 show the difference in frequency divider spectrums when a Q-Offset has been applied. The Q-Offset provides a stable and accurate reading, but causes a larger frequency shift when Q-Enhancement is backed-off. To counter this frequency shift, a frequency offset was needed. The accuracy of the filter's center frequency is dependent on the resolution of the frequency counter used, 100 kHz in this implementation, and the amount of Q-Enhancement required for the divider's to have a clean output spectrum.

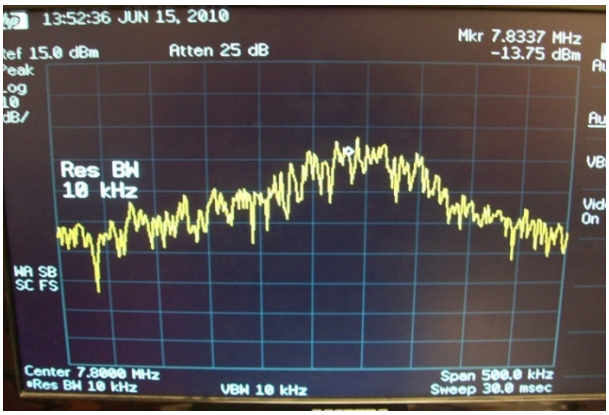


Figure 5.3 – Frequency Divider Output Spectrum at First Oscillation

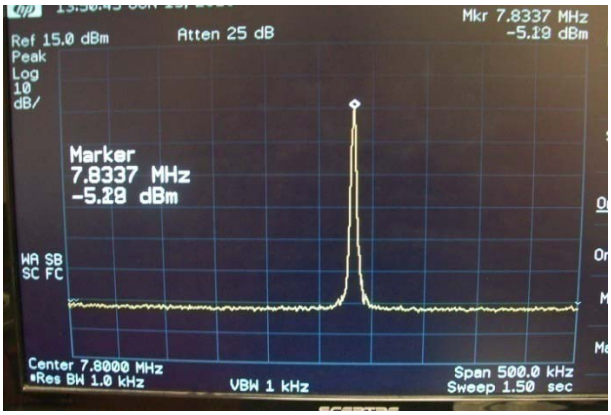


Figure 5.4 – Frequency Divider Output Spectrum with Q-Offset Past First Oscillation

5.3 FUTURE WORK

The future of the Q-Enhanced Filter two-pole tuning algorithm lies in more detailed investigations of the limits of narrow-bandwidth operation and in the synthesis of the algorithm into digital circuitry if desired. The following Sections describe the steps and considerations needed to implement the digital control circuit design.

5.3.1 FPGA IMPLEMENTATION

To implement the algorithm in firmware, the tuning algorithm, currently written in C, will need to be translated into a hardware description language such as Verilog. A FPGA version of the Q-Enhanced Filter board will needed to be created in order to run the Verilog version of the tuning algorithm. Rigorous simulation and timing characterization will need to be completed once the algorithm has been translated to address any potential race conditions between the algorithm and the filter ring-up and ring-down times or amplitude detector filtering time constants. Useful performance measures, such as the number of gates used, are available during the programming of the FPGA as well. The number of gates can be used to allude to the amount of chip real-estate needed for the tuning algorithm circuitry.

5.3.2 CHANGES TO THE FILTER DESIGN

As described in Section 3, pass-band asymmetries occur when using non-ideal inductors. Using high Q_0 inductors minimizes these asymmetries, but does not entirely eliminate them. In-phase coupling controls, such as the ones used in [2, 15], should be added to effectively neutralize the asymmetries. The capacitive coupling connections should be carefully examined, to prevent any undesired cross-couplings from being created. The off-capacitance of tuning circuits needs to be accounted for, as it can cause frequency shifts when the controls have been turned off. The resistor values (R_X in figure 3.10) used within the coupling capacitor circuitry should be on the order of 5 MΩ to negate the in-phase portions of the injected resonator currents from this area of circuitry.

Improvement of the frequency dividers is imperative. Q-Enhancement past first-oscillation was required to have a stable frequency divider output as described in Section 5.2. Frequency divider architectures such as the injection-locked frequency divider in [13] might offer a good alternative to the CML Latch comparator used in the current frequency divider architecture [22].

5.3.3 ALGORITHM SUPPORT CIRCUITRY

In addition to the algorithm logic currently implemented on the microcontroller, the algorithm depends on an Analog-to-Digital Converter, Digital-to-Analog Converter. The following sections describe design considerations of these support circuits.

5.3.3.1. ANALOG-TO-DIGITAL CONVERTERS

A Successive Approximation Register ADC is recommended as an on-chip replacement to the microcontroller's ADC. The number of bits needed is dependent on the voltage swing of the amplitude detectors and the reference voltage being used. A DC amplifier can be added to increase the voltage swing of the amplitude detector and relax required bit resolution of the ADC. The microcontroller uses an ADC channel for each amplitude detector output. The on-chip version can either use two different ADCs or multiplex the inputs (create channels).

5.3.3.2. DIGITAL-TO-ANALOG CONVERTERS

A DAC is needed for each analog tuning line to support more narrow bandwidths. Currently, the filter has analog Q-Enhancement and frequency tuning lines for each End. The bit resolution is dependent on the desired amount of steps between ground and the 2.5V reference voltage.

REFERENCES

[1] C. Bowick, C. Ajluni, J. Blyler, "RF Circuit Design," Newnes/Elsevier, 2007, pp. 2-62

[2] W.B. Kuhn, N.K. Yanduru, A.S. Wyszynski, "Q-Enhanced LC Bandpass Filters for Integrated Wireless Applications," *in IEEE Transactions on Microwave Theory and Techniques*, vol.46, no.12, pp.2577-2586, Dec. 1998

[3] W.B. Kuhn, D. Nobbe, D. Kelly, A.W. Orsborn, "Dynamic Range Performance of On-Chip RF Bandpass Filters," *in IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol.50, no.10, pp. 685- 694, Oct. 2003.

[4] X. He and W.B. Kuhn, "A 2.5-GHz Low-Power, High Dynamic Range, Self-Tuned Q-Enhanced LC Filter in SOI," *in IEEE Journal of Solid-State Circuits*, vol. 40, no. 8, 2005, pp. 1618-1628.

[5] W.B. Kuhn, "Fully Integrated Bandpass Filters for Wireless Transceivers – Problems and Promises," in *Proc. IEEE Midwest Symp. Circuits and Systems*, 2002, pp. 69-72.

[6] K. Van Hartingsveldt, P. Quinn, and A. Van Roermund , "A 10.7 MHz CMOS SC radio IF filter with variable gain and a Q of 55 ," *Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International* , vol., no., pp.152-153, 452, 2000

[7] R.A. Duncan, K.W. Martin, and A.S. Sedra, "A Q-Enhanced Active-RLC Bandpass Filter," IEEE Int. Symp. on Circuits and Systems, 1993 pp. 1416-1419.

[8] S. Pipolos, Y.P. Tsividis, J. Fenk, and Y. Papananos, "A Si 1.8 GHz RLC Filter with Tunable Center frequency and Quality Factor," IEEE J. Solid-State Circuits, pp. 1517 - 1525, Oct. 1996.

[9] Aparin, V.; Katzin, P., "Active GaAs MMIC Band-Pass Filters with Automatic Frequency Tuning and Insertion Loss Control," IEEE JSSC, pp. 1068-1073, Oct 1995.

[10] Y. Tsividis, "Self-Tuned Filters," Elect. Lettr., pp. 406-407, 11 June 1981.

[11] D. Li and Y. Tsividis, "Active LC Filters on Silicon," IEEE Proceedings - Circuits, Devices and Systems, pp. 49-56, 2000.

[12] J.K. Nakaska and J.W. Haslett, “2 GHz Automatically Tuned Q-Enhanced CMOS Bandpass Filter,” 2007 IEEE MTT-S International Microwave Symposium, Digest of Papers, Honolulu, Hawaii, United States, pp. 1599-1602, June 3-8 2007.

[13] R. Strouts “Automatic Tuning of Q-Enhanced Integrated Differential Bandpass Filters in a Silcon-On-Sapphire Process,” M.S. thesis, Kansas State University, Manhattan, KS, United States, 2009.

[14] J.R. MacLeod, M.A Beach, P.A. Warr, T. Nesimoglu, "Software Defined Radio Receiver Test-Bed," 54th IEEE Vehicular Technology Conference, pp. 1565 - 1569, 2001.

[15] W.B. Kuhn, F.W. Stephenson, A. Elshabini-Riad, “A 200 MHz CMOS Q-Enhanced LC Bandpass Filter,” *in IEEE Journal of Solid-State Circuits*, vol. 31, no. 8, 1996, pp. 1112-1122

[16] Shaorui Li; Stanic, N.; Tsividis, Y.; , "A VCF Loss-Control Tuning Loop for Q -Enhanced LC Filters," in IEEE Transcactions on Circuits and Systems II: Express Briefs, vol.53, no.9, pp.906-910, Sept. 2006

[17] Shaorui Li; Stanic, N.; Soumyanath, K.; Tsividis, Y.; , "An integrated 1.5 V 6 GHz Q-enhanced LC CMOS filter with automatic quality factor tuning using conductance reference," Radio Frequency integrated Circuits (RFIC) Symposium, 2005. Digest of Papers. 2005 IEEE , vol., no., pp. 621- 624, 12-14 June 2005

[18] Li, D.; Tsividis, Y.; , "A 1.9 GHz Si active LC filter with on-chip automatic tuning," Solid-State Circuits Conference, 2001. Digest of Technical Papers. ISSCC. 2001 IEEE International , vol., no., pp.368-369, 466, 2001

[19] Georgescu, B.; Finvers, I.G.; Ghannouchi, F.; , "2 GHz Q-Enhanced Active Filter With Low Passband Distortion and High Dynamic Range," Solid-State Circuits, IEEE Journal of , vol.41, no.9, pp.2029-2039, Sept. 2006

[20] A. Boutz “Inductors in LTCC Utilizing Full Tape Thickness Features,” M.S. thesis, Kansas State University, Manhattan, KS, United States, 2009.

[21] A. Boutz and W.B. Kuhn “Measurement and Performance of Embedded LTCC Inductors Utilizing Full Tape Thickness Feature Conductors,” IMAPS/CICMT January 2009.

[22] J.R. Hu, B.P. Otis, “A 3 µW, 400 MHz divide-by-5 injection-locked frequency divider with 56% lock range in 90nm CMOS,” *Radio Frequency Integrated Circuits Symposium, 2008. RFIC 2008. IEEE* , vol., no., pp.665-668, April 2008.

APPENDIX A - FILTER PRINTED CIRCUIT BOARD VERSION 2.0

Version 2.0 of the Q-Enhanced Filter PCB is documented in this appendix. The board is split up into RF, digital, and power schematics shown in figures A.1-3. The PCB layout and corresponding layers are shown in figures A.4-7. Photographs of the populated board top and bottom are shown in figures A.8-9.

A.1 PARTS

All parts and quantities can be found in table A.1 with the exception of resistors, capacitors, inductors, and headers. Refer to the schematics for more information regarding component values and quantities.

Q-Enhanced Filter PCB Version 2.0 Parts List				
Description	**Schematic Name**	**Digi-Key Part No.**	**Mouser Part No.**	**QTY**
3.3/2.5 V 1 A Dual Fixed Regulator	AP1120SL-13	AP1120SLDICT-ND	621-AP1120SL-13	3
USB-to-UART Bridge	CP2103	336-1164-ND	634-CP2103-GM	1
26 MHz TCXO	CSX-532T		695-CSX532T-26	1
dsPIC33	dsPIC33FJ64GP802	DSPIC33FJ64GP802-I/SO-ND	579-DSPIC364GP802ISO	1
SM RJ25 Jack	ICSP_CONN	WM5567CT-ND	538-44144-0005	1
Red LEDs	LED (RED)	404-1000-1-ND		6
10-Bit Dual DAC	MAX5522EUA+	MAX5522EUA+-ND	700-MAX5522EUA	2
2 Input Non-Inverting Multiplexor	NC7SZ157P6X	NC7SZ157P6XCT-ND	512-NC7SZ157P6X	1
Vertical SMA Connector	SMA	ACX1230-ND	523-132134	5
SM SPST Switch	Switch	CKN9050CT-ND		1
SM USB B-Connector	USB_B_CONN	WM17113-ND	538-67068-8000	1

Table A.1 – Parts List

A.2 MICROCONTROLLER CONFIGURATION

There are small differences between preprocessor definitions/macros on the Q-Enhanced Filter PCB versions. The BOARD_VERSION preprocessor definitions listed in table A.2 will need to be set to VERSION_2_0 to work correctly. Refer to Appendices G-I for the appropriate source code.

Preprocessor Definitions	
Preprocessor Definition	**Value**
BOARD_VERSION_2_0	0
BOARD_VERSION_2_1	1
BOARD_VERSION	BOARD_VERSION_2_0

Table A.2 – Preprocessor Definitions

A.3 NOTES

This version of the Q-Enhanced PCB needed a few revisions. Pin 2 of the ICSP_CONN should be tied to pin 5 of U7. The trace shown within the red circle in figure A.9 needs to be cut. The red wire in figure A.8 makes the connection previously described. The green wires shown in figure A.8 and the remaining red circle in figure A.9 can be ignored because the LCD is not used. Because the LCD is not used U8, J2, J3, and supporting components need not be populated.

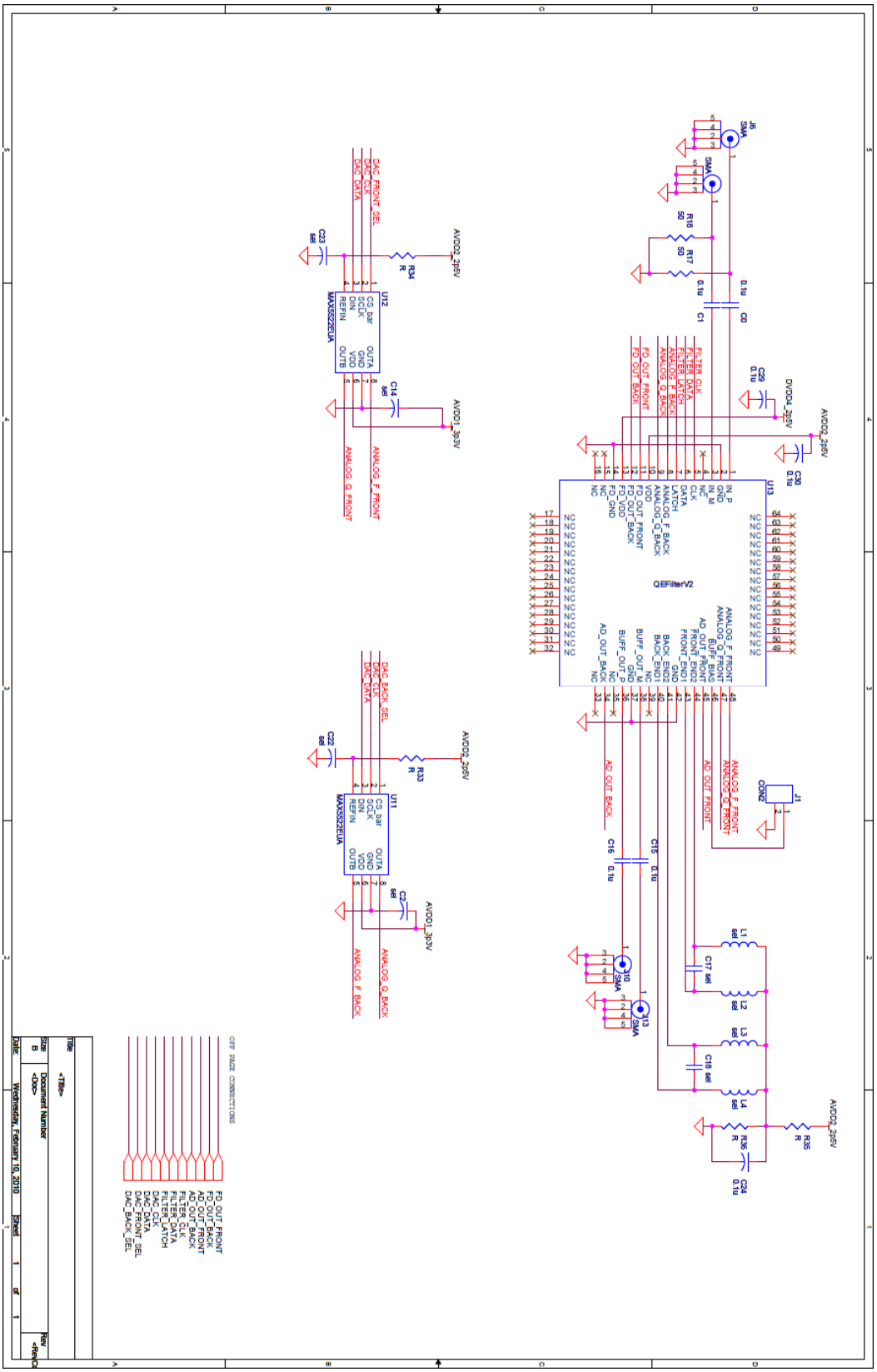


Figure A.1 – RF Schematic

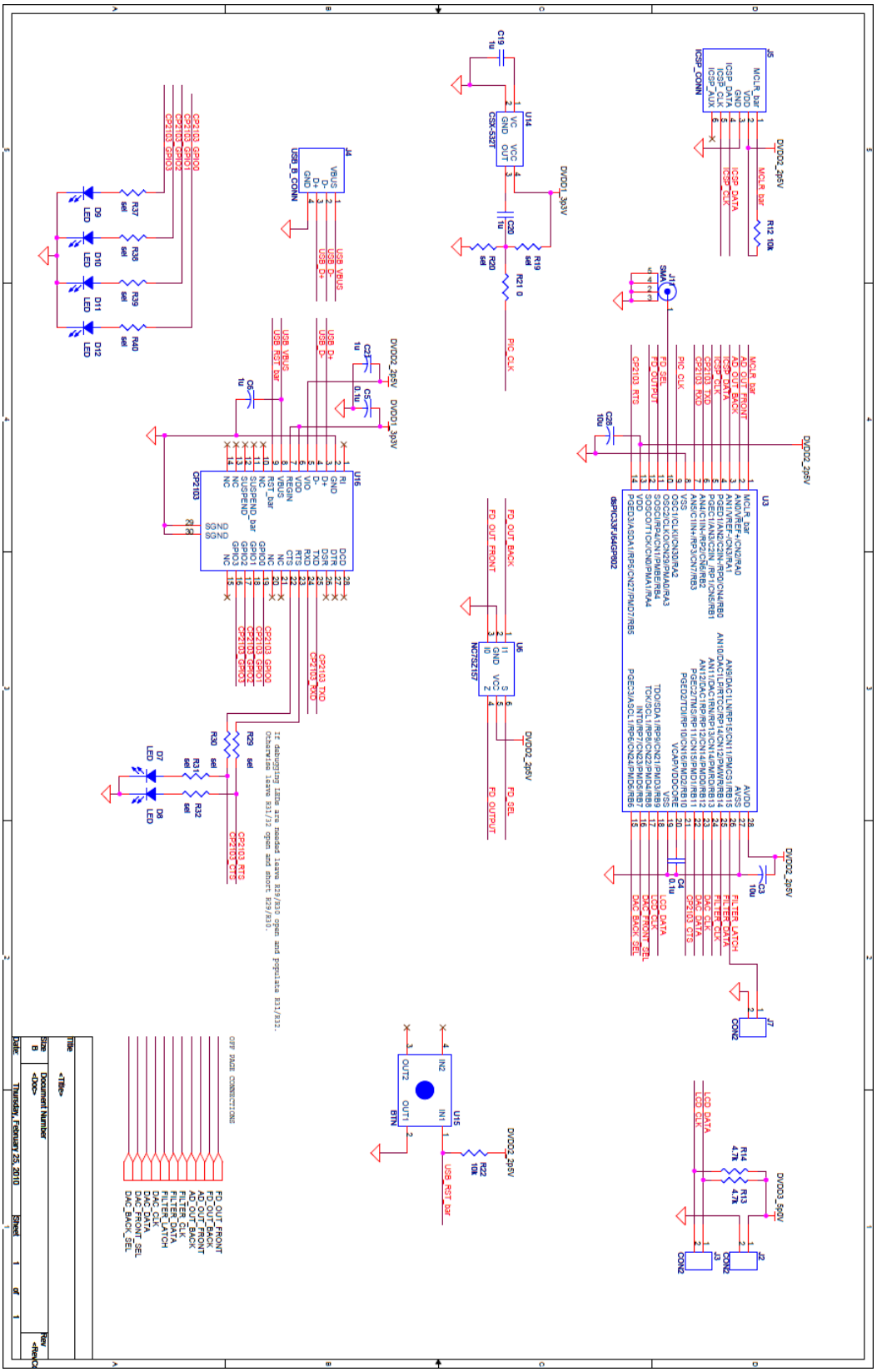


Figure A.2 – Digital Schematic

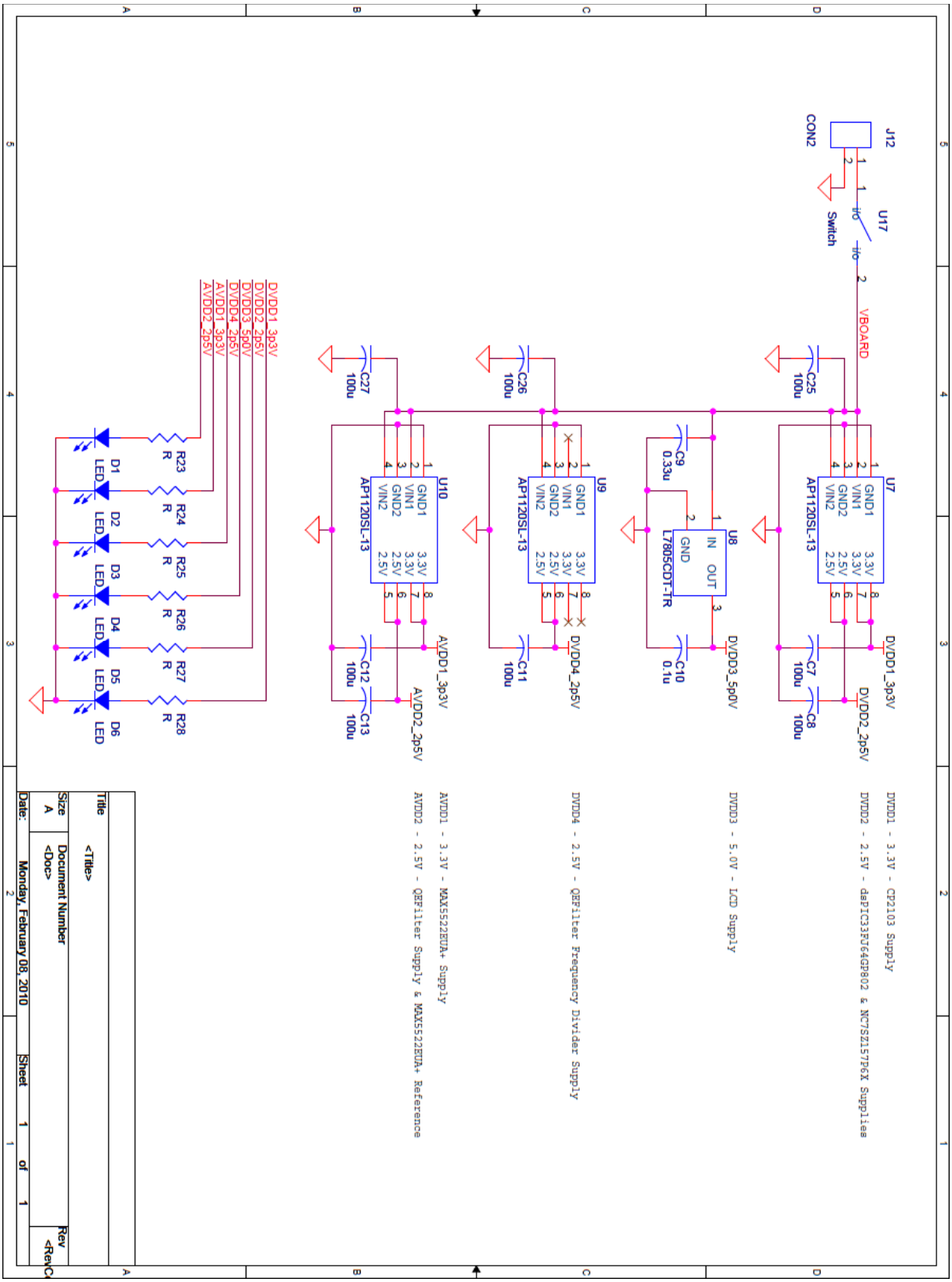


Figure A.3 – Power Schematic

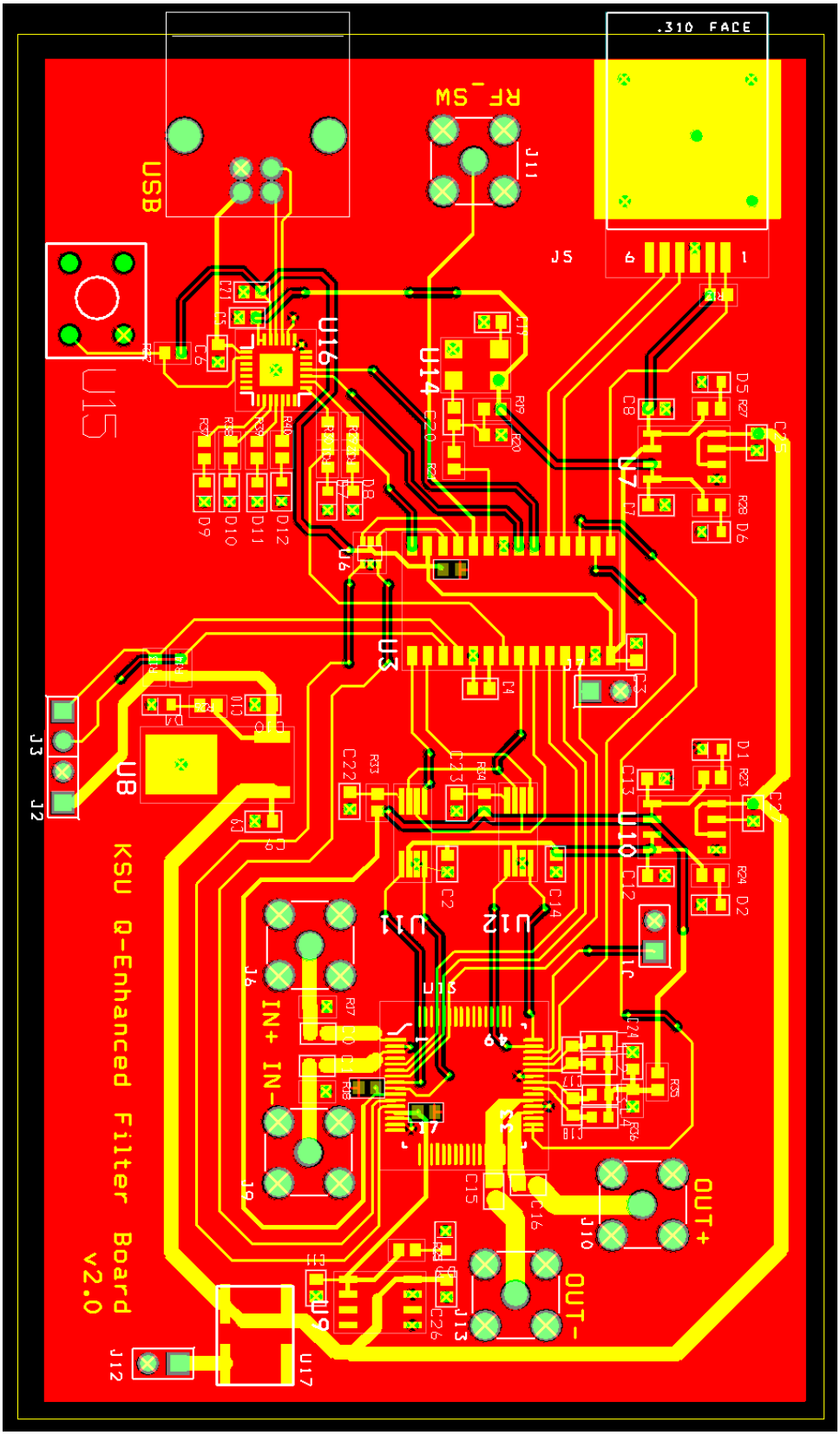


Figure A.4 – Printed Circuit Board Layout

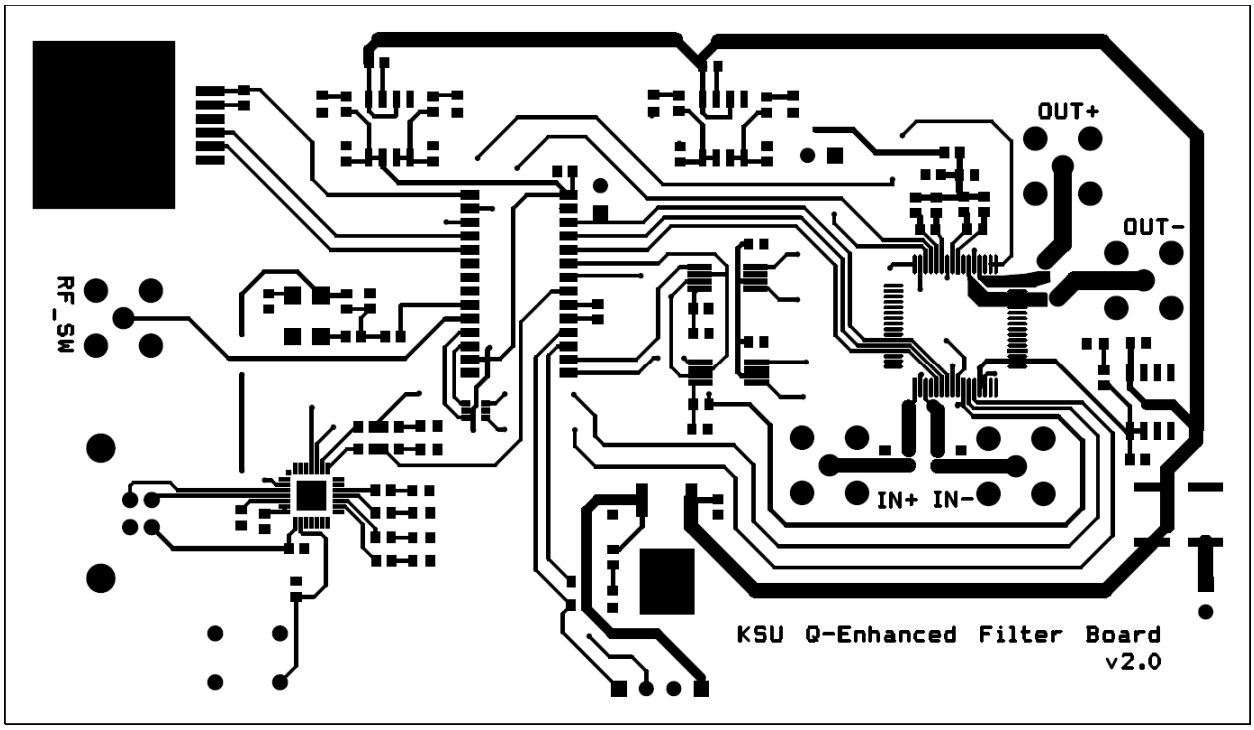


Figure A.5 – Printed Circuit Board Top Layer

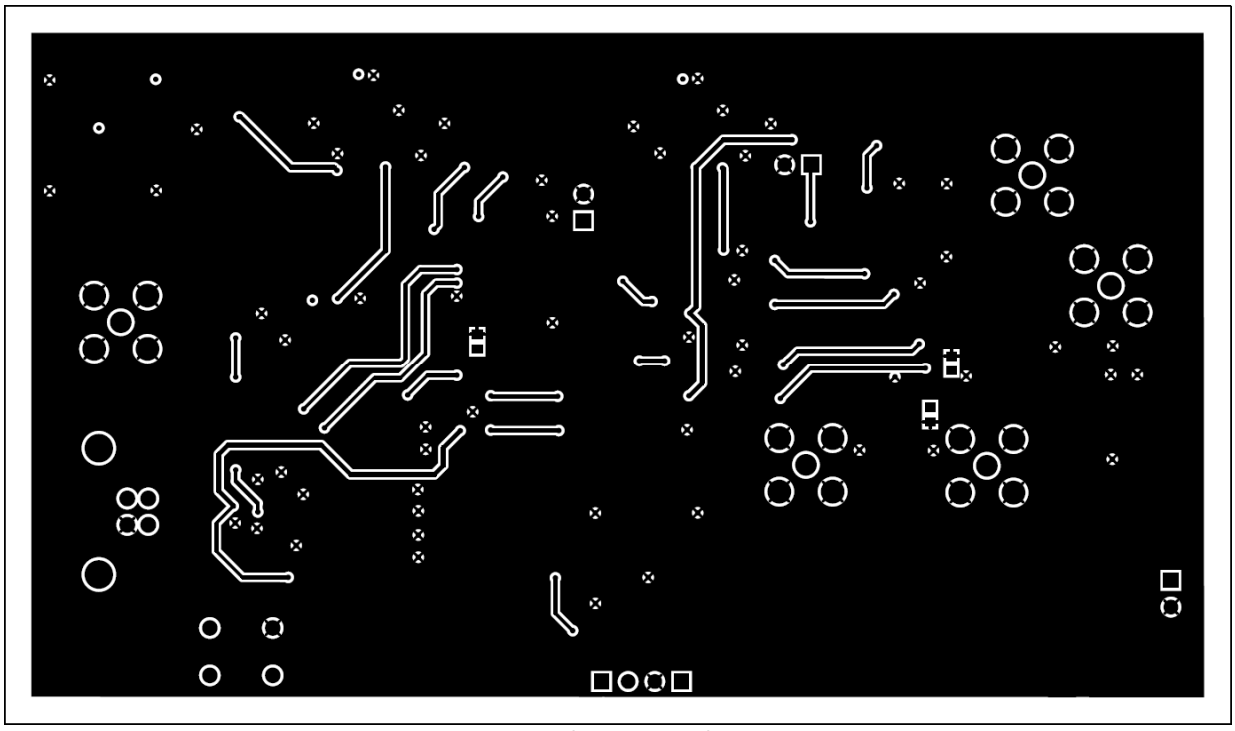


Figure A.6 – Printed Circuit Board Bottom Layer

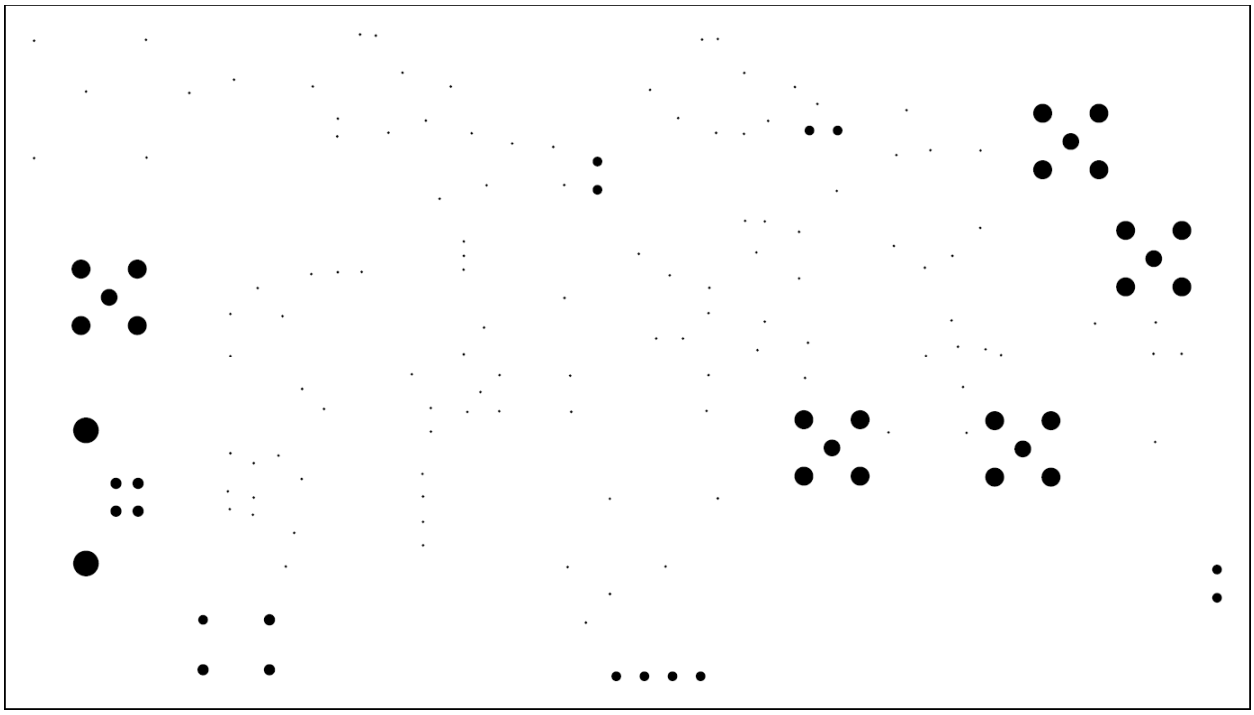


Figure A.7 – Printed Circuit Board Drill Taps

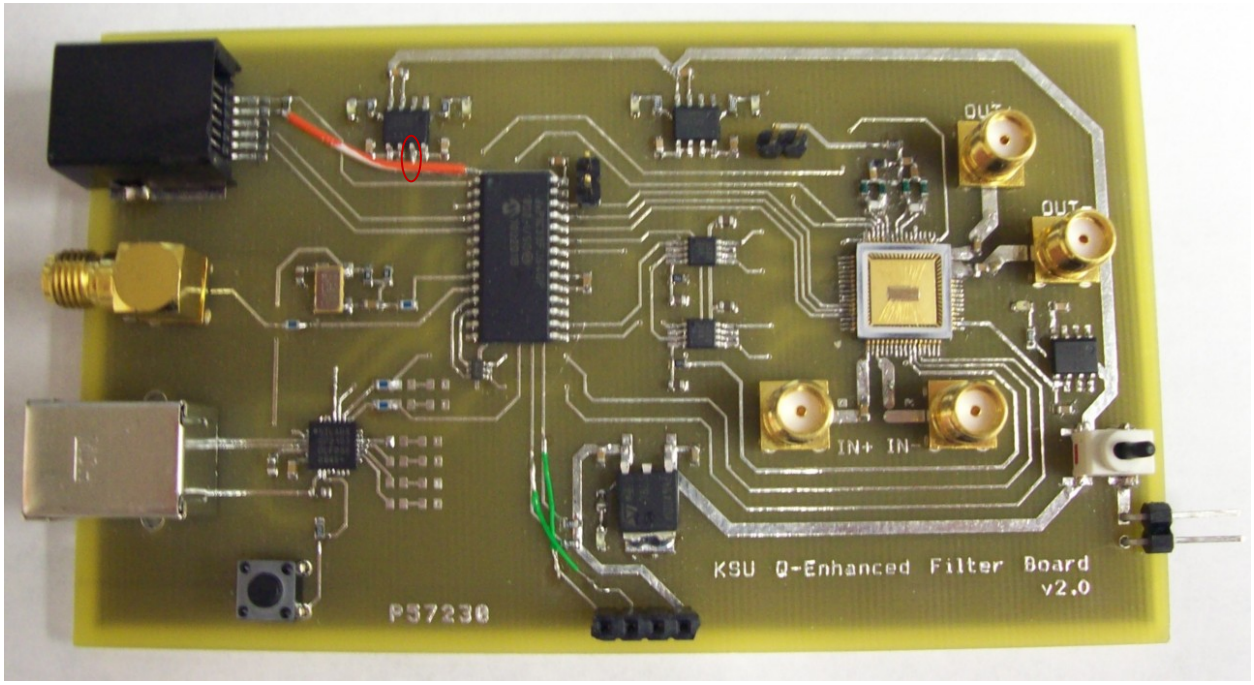


Figure A.8 – Printed Circuit Board Populated Top

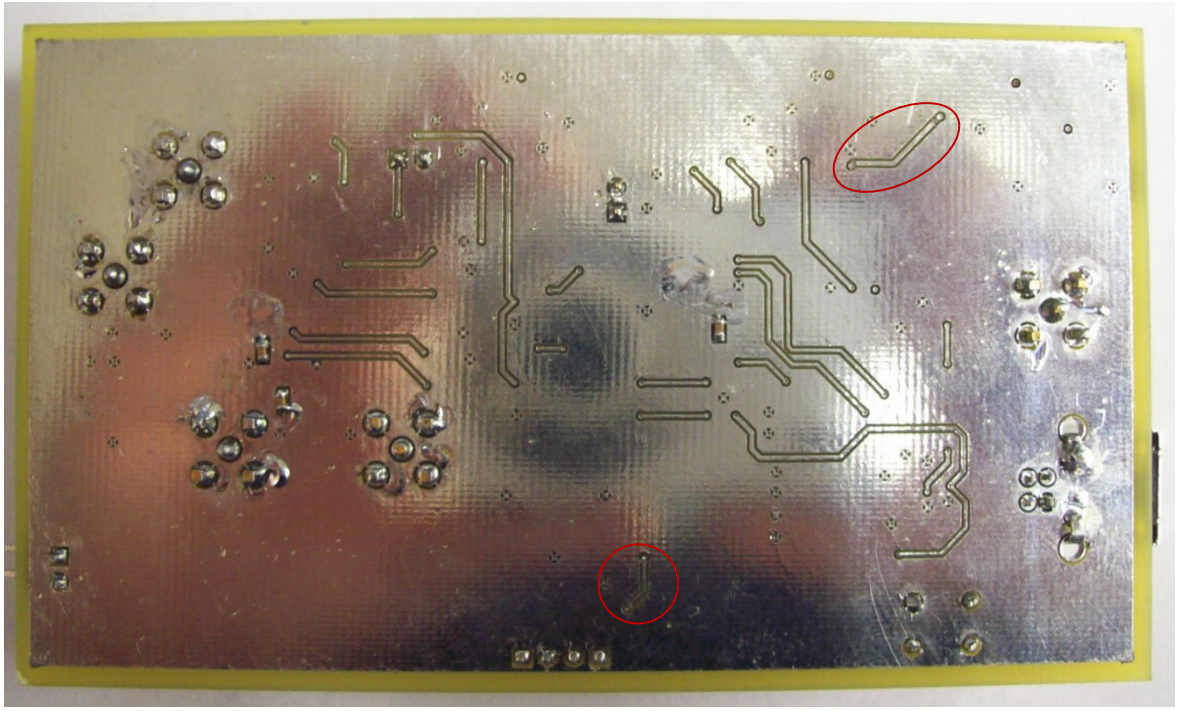


Figure A.9 – Printed Circuit Board Populated Bottom

APPENDIX B - FILTER PRINTED CIRCUIT BOARD VERSION 2.1

Version 2.1 of the Q-Enhanced Filter PCB is documented in this appendix. The board is split up into RF, digital, and power schematics shown in figures B.1-3. The PCB layout and corresponding layers are shown in figures B.4-7. Photographs of the populated board top and bottom are shown in figures B.8-9.

B.1 PARTS

All parts and quantities can be found in table B.1 with the exception of resistors, capacitors, inductors, and headers. Refer to the schematics for more information regarding component values and quantities.

Q-Enhanced Filter PCB Version 2.1 Parts List				
Description	**Schematic Name**	**Digi-Key Part No.**	**Mouser Part No.**	**QTY**
3.3/2.5 V 1 A Dual Fixed Regulator	AP1120SL-13	AP1120SLDICT-ND	621-AP1120SL-13	3
2.0 V 0.3 A Fixed Regulator	AP130		621-AP130-20WL-7	1
USB-to-UART Bridge	CP2103	336-1164-ND	634-CP2103-GM	1
26 MHz TCXO	CSX-532T		695-CSX532T-26	1
dsPIC33	dsPIC33FJ64GP802	DSPIC33FJ64GP802-I/SO-ND	579-DSPIC364GP802ISO	1
SM RJ25 Jack	ICSP_CONN	WM5567CT-ND	538-44144-0005	1
Red LEDs	LED (RED)	404-1000-1-ND		6
10-Bit Dual DAC	MAX5522EUA+	MAX5522EUA+-ND	700-MAX5522EUA	2
2 Input Non-Inverting Multiplexor	NC7SZ157P6X	NC7SZ157P6XCT-ND	512-NC7SZ157P6X	1
Vertical SMA Connector	SMA	ACX1230-ND	523-132134	5
SM SPST Switch	Switch	CKN9050CT-ND		1
SM USB B-Connector	USB_B_CONN	WM17113-ND	538-67068-8000	1

Table B.1 – Parts List

B.2 MICROCONTROLLER CONFIGURATION

There are small differences between preprocessor definitions/macros on the Q-Enhanced Filter PCB versions. The BOARD_VERSION preprocessor definitions listed in table A.2 will need to be set to VERSION_2_1 to work correctly. Refer to Appendices G-I for the appropriate source code.

Preprocessor Definitions	
Preprocessor Definition	**Value**
BOARD_VERSION_2_0	0
BOARD_VERSION_2_1	1
BOARD_VERSION	BOARD_VERSION_2_1

Table A.2 – Preprocessor Definitions

B.3 NOTES

This version of the Q-Enhanced PCB corrects the revisions needed in version 2.0. The LCD has been dropped; an additional button and debug LED have been added. The magnetic coupling issues discussed in Section 2.5 have also been addressed. A header has been added to allow a ribbon cable to be connected to the header on the LTCC Companion PCB version described in Appendix C.

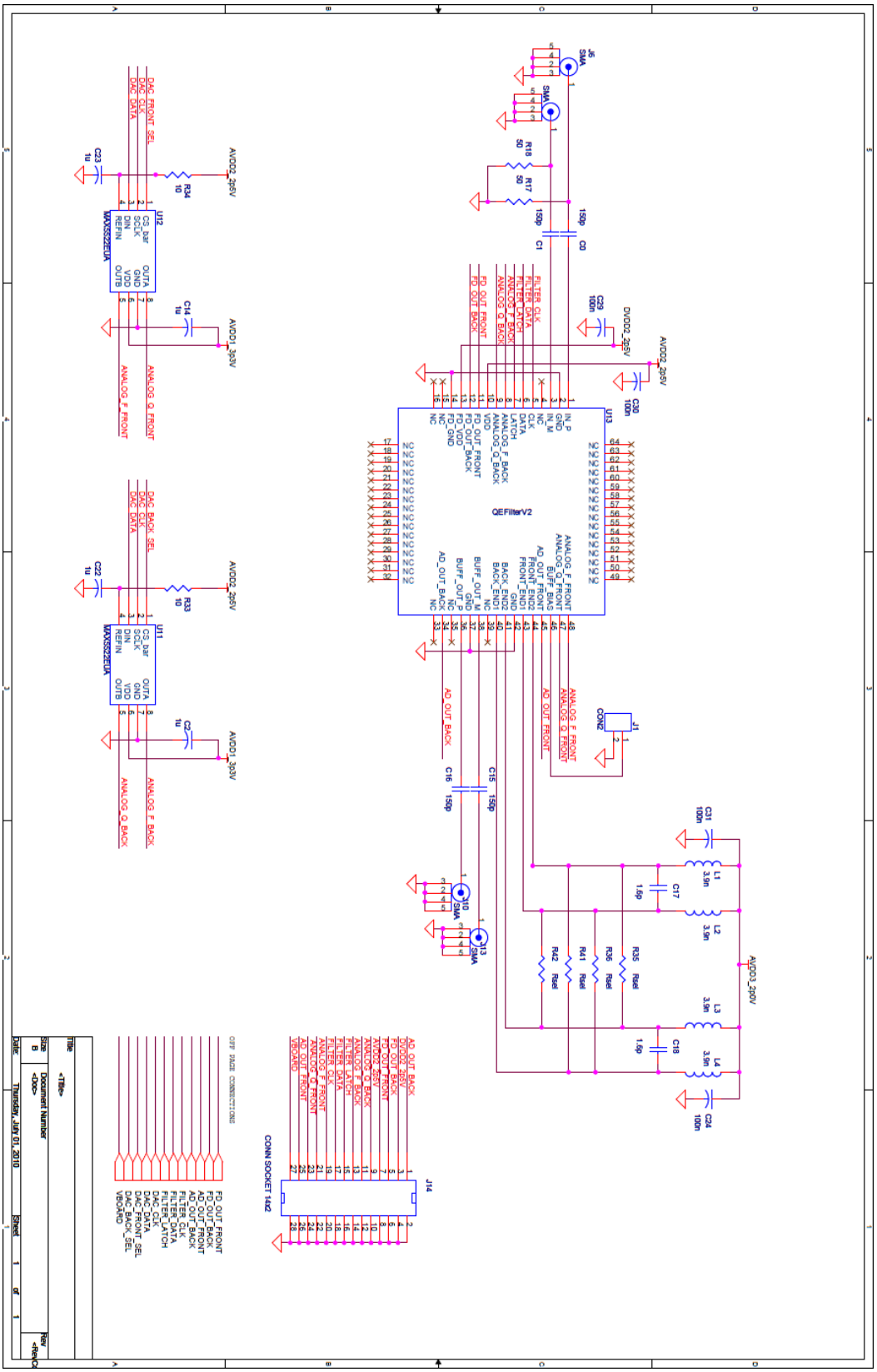


Figure B.1 – RF Schematic

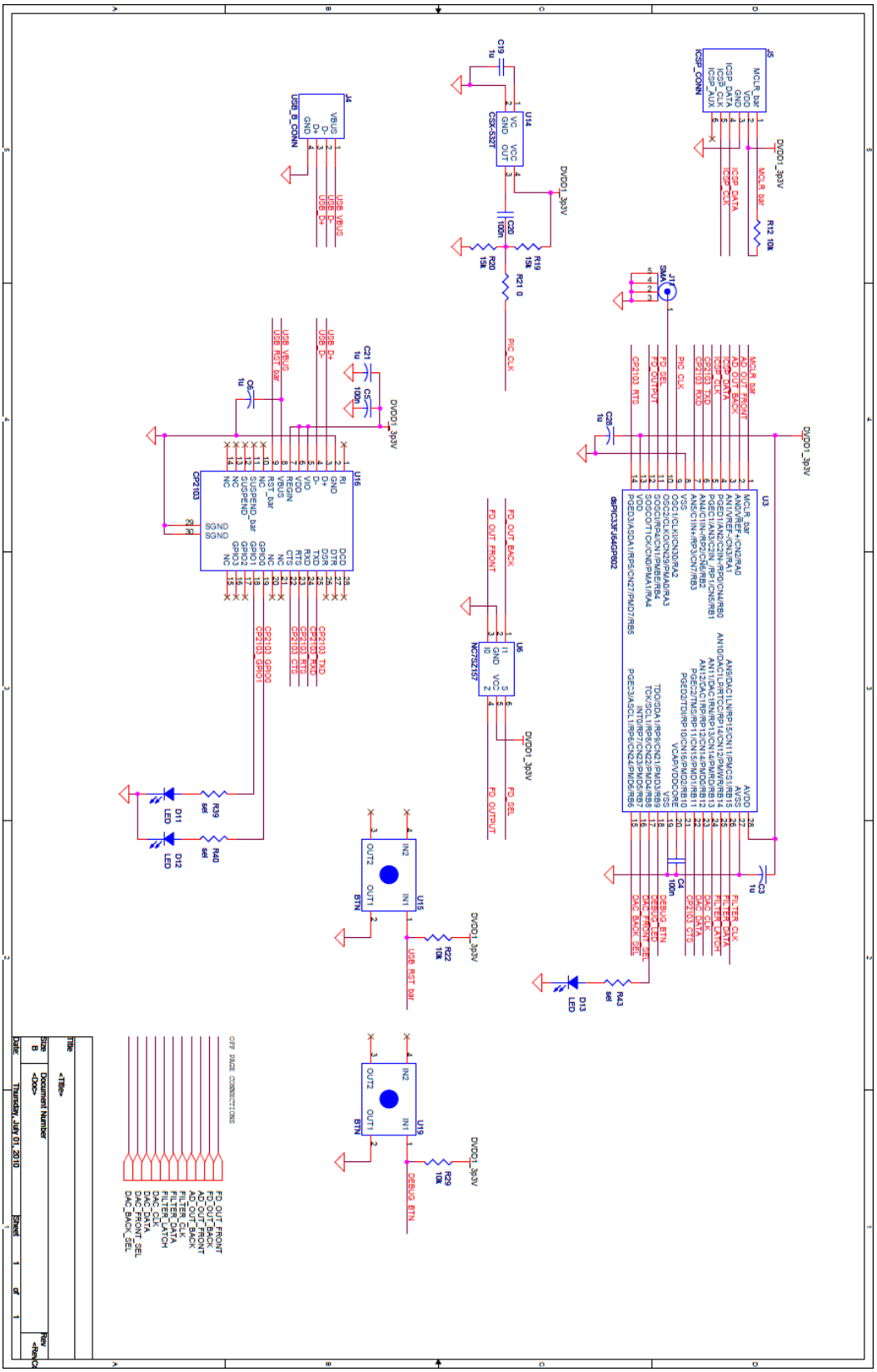


Figure B.2 – Digital Schematic

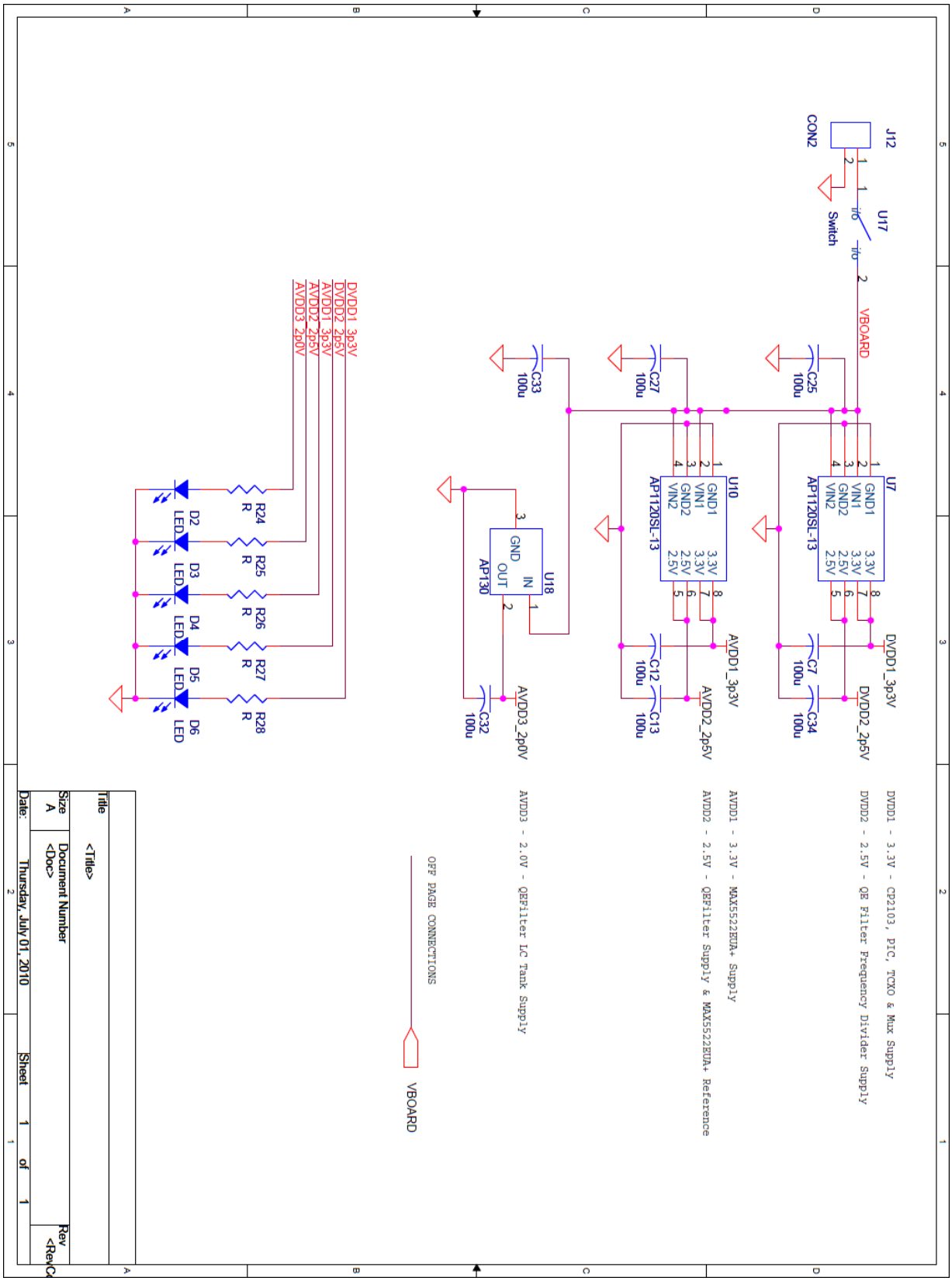


Figure B.3 – Power Schematic

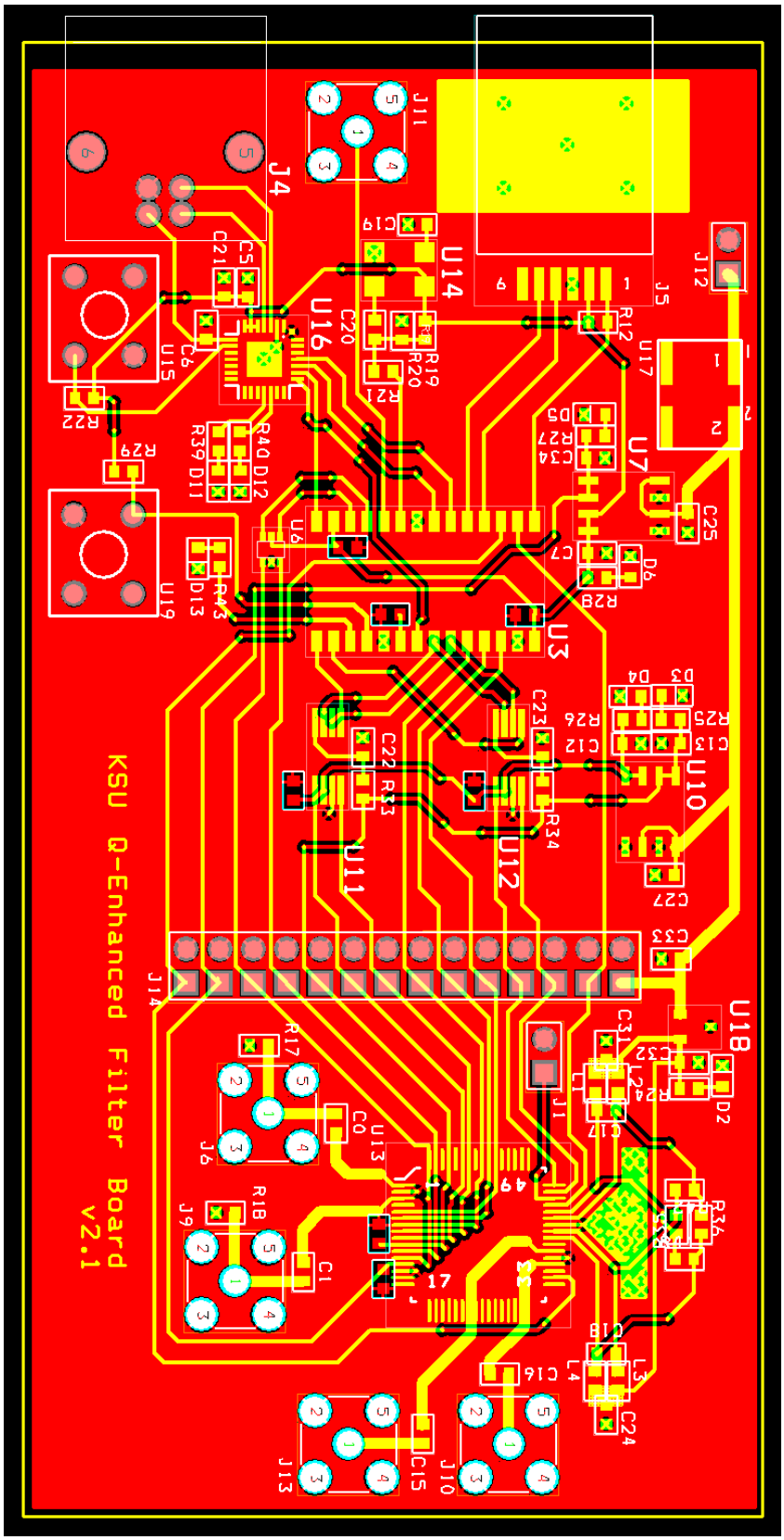


Figure B.4 – Printed Circuit Board Layout

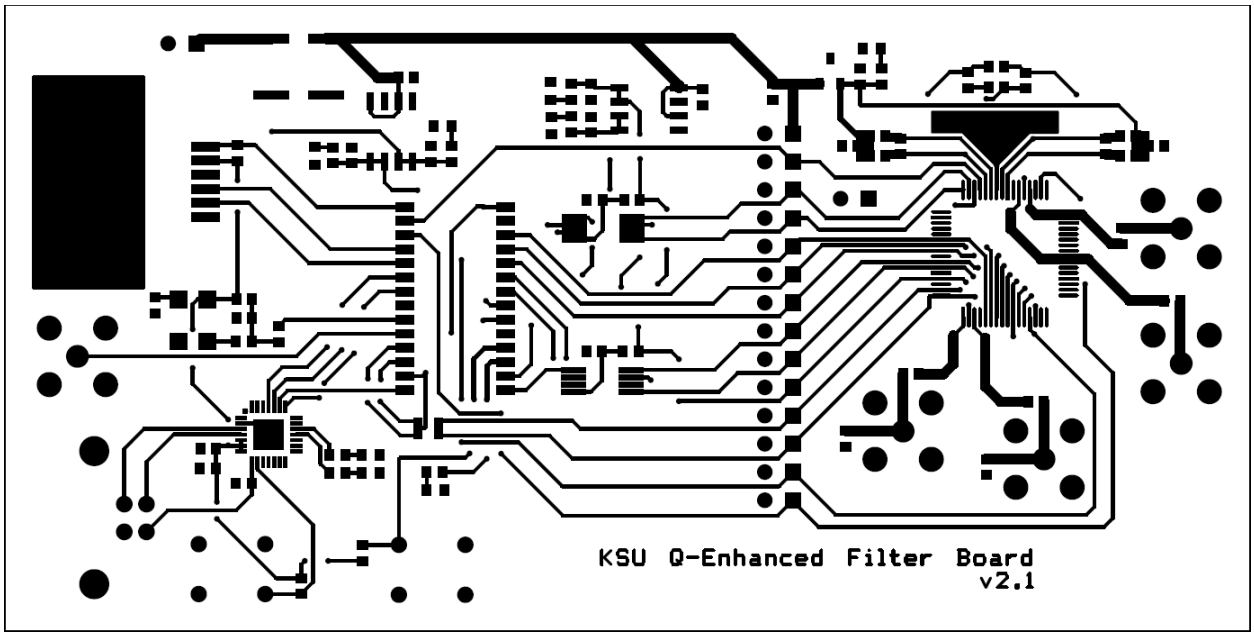


Figure B.5 – Printed Circuit Board Top Layer

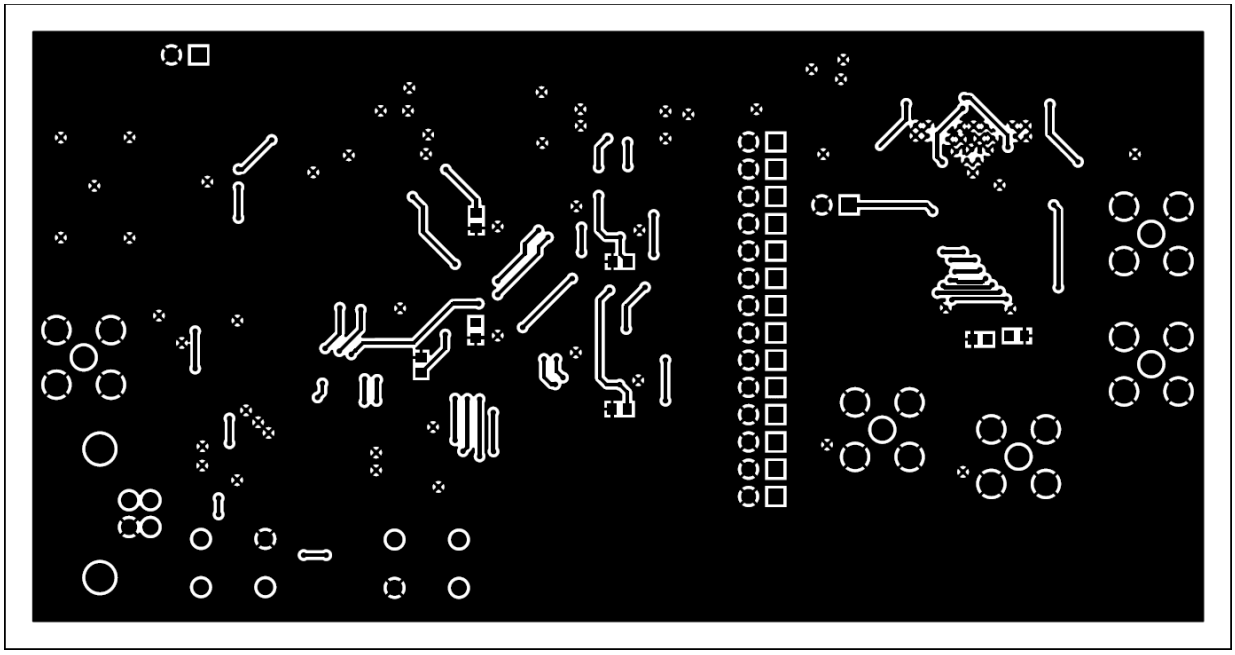


Figure B.6 – Printed Circuit Board Bottom Layer

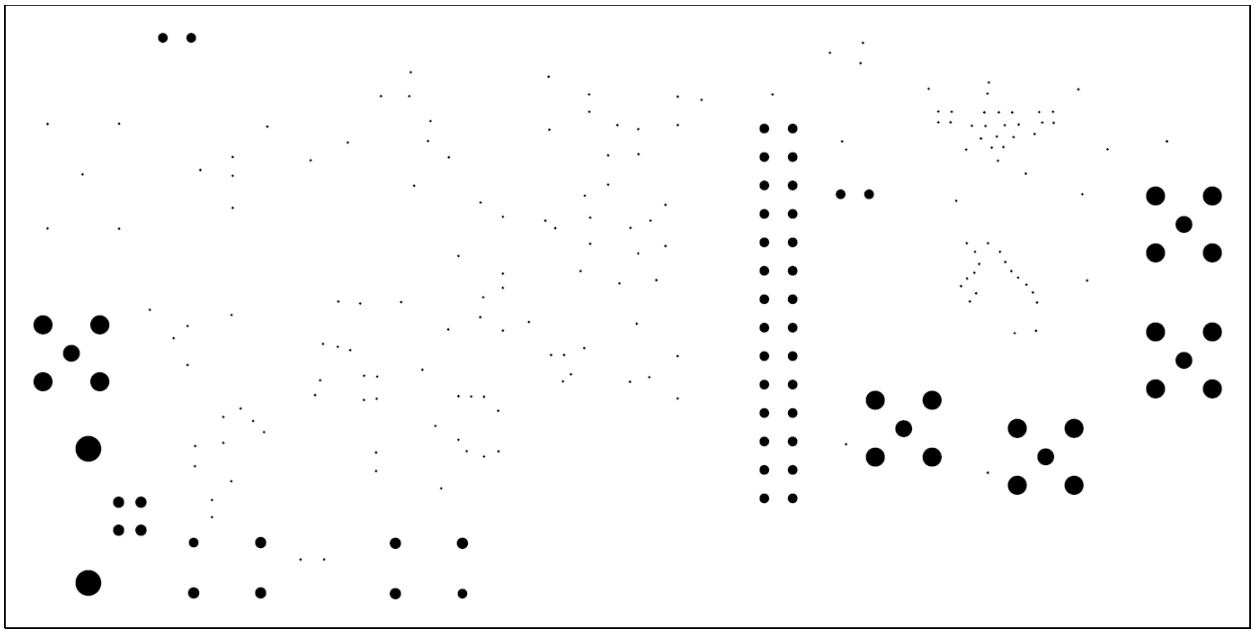


Figure B.7 – Printed Circuit Board Drill Taps

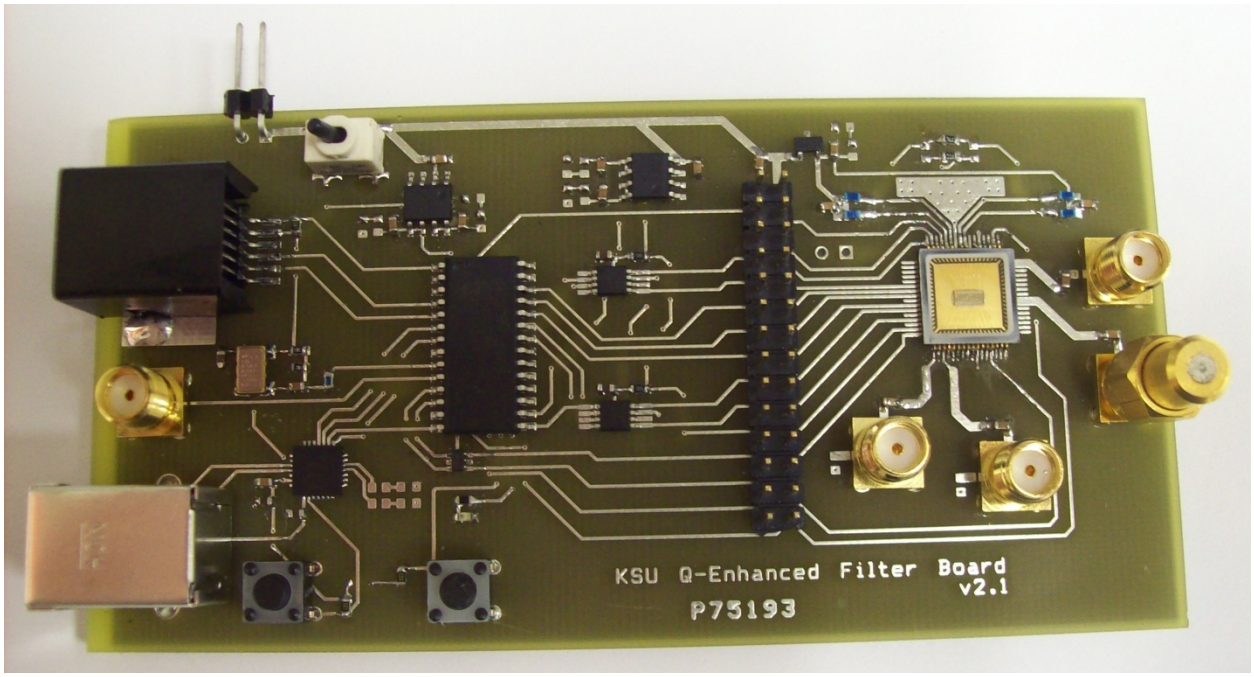


Figure B.8 – Printed Circuit Board Populated Top

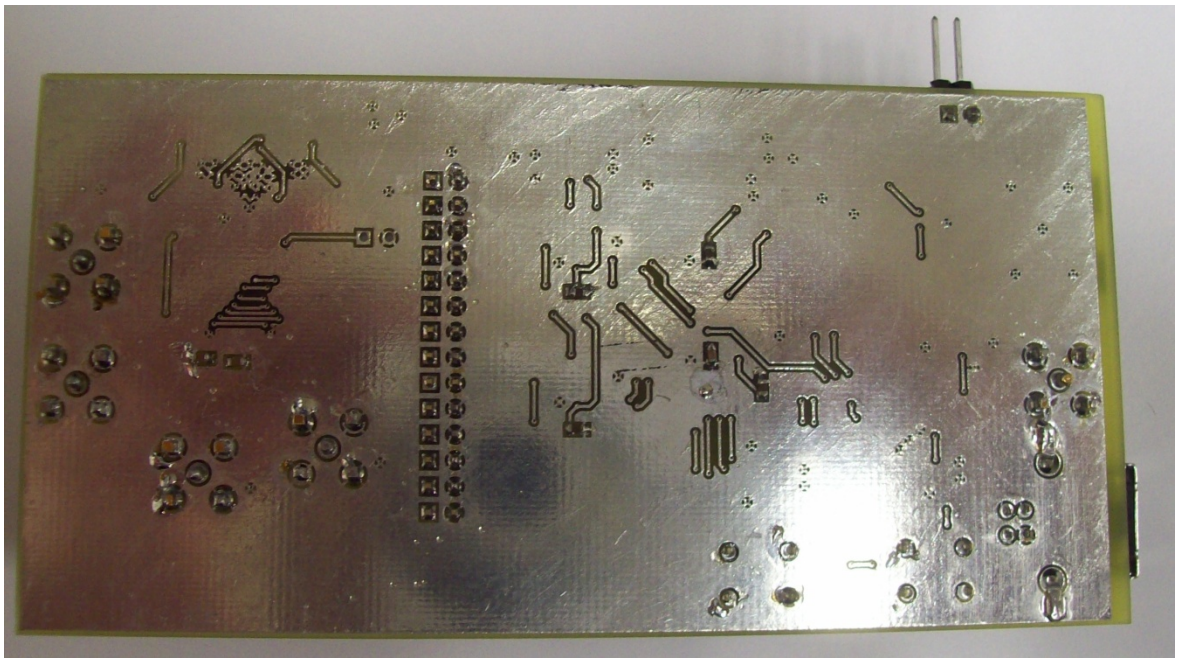


Figure B.9 – Printed Circuit Board Populated Bottom

APPENDIX C - LTCC COMPANION PRINTED CIRCUIT BOARD VERSION 1.0

Version 1.0 of the LTCC Companion PCB is documented in this appendix. This board is meant to act as a companion board to the Q-Enhanced Filter PCB version 2.1. A ribbon cable connected to the 28 pin header should also be connected to a version 2.1 PCB that does not contain a Q-Enhanced filter chip. The board was created in order to test Q-Enhanced filter with the LTCC substrate embedded high Q_0 inductors documented in [20 - 21]. A schematic of the board can be found in figure C.1. PCB layouts with and without ground planes can be found in figures C.2-3. For brevity layer documentation has been omitted.

C.1 PARTS

All parts and quantities can be found in table B.1 with the exception of resistors, capacitors, and inductors. Refer to the schematics for more information regarding component values and quantities.

Q-Enhanced Filter PCB Version 2.1 Parts List				
Description	**Schematic Name**	**Digi-Key Part No.**	**Mouser Part No.**	**QTY**
2.0 V 0.3 A Fixed Regulator	U2		621-AP130-20WL-7	1

Table C.1 – Parts List

Type B, 2.5 turn, 7 mil punch, 40 mil inner diameter inductors were chosen to have an inductance of roughly 3.9 nH.

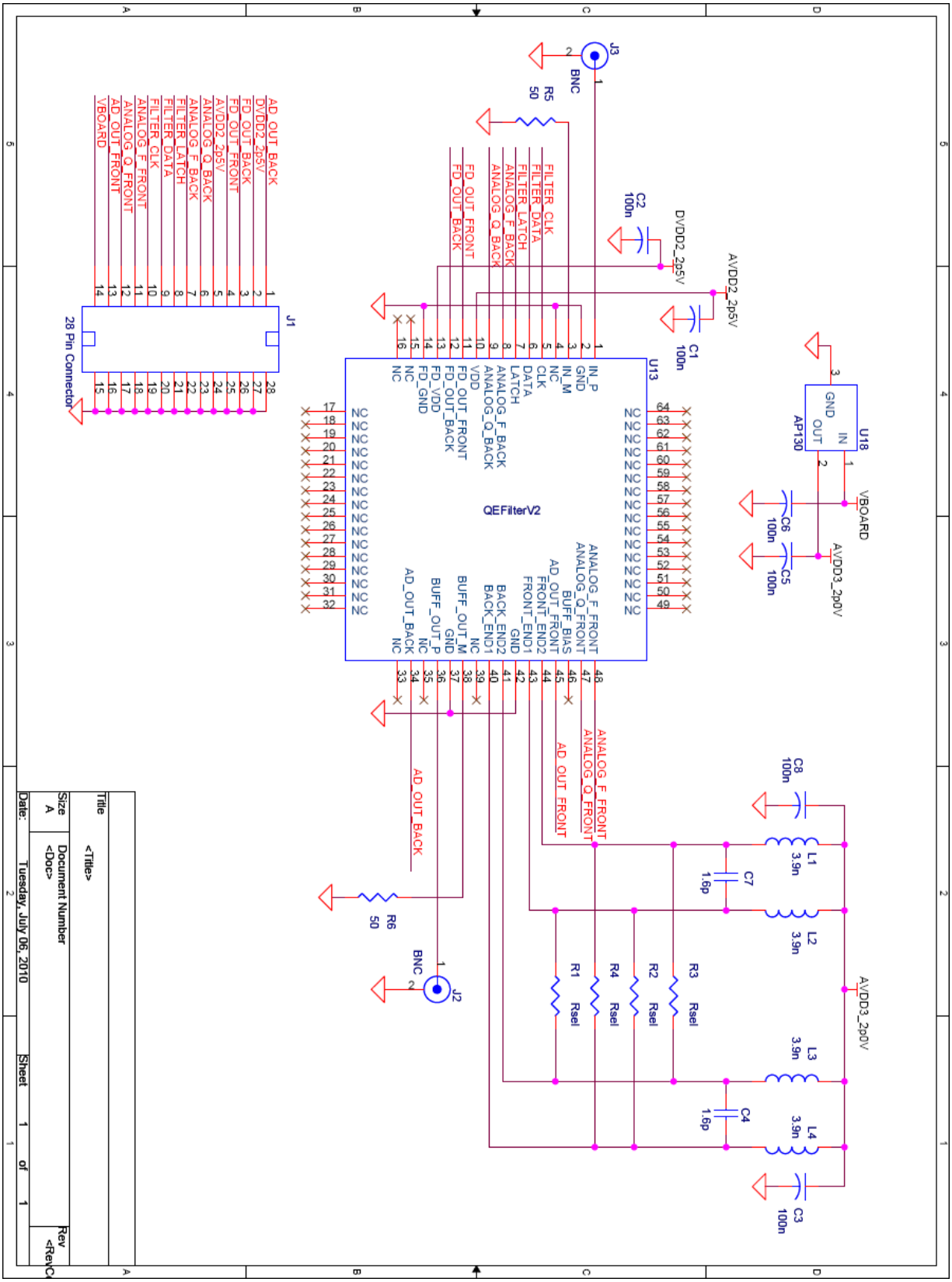


Figure C.1 – LTCC Printed Circuit Board Schematic

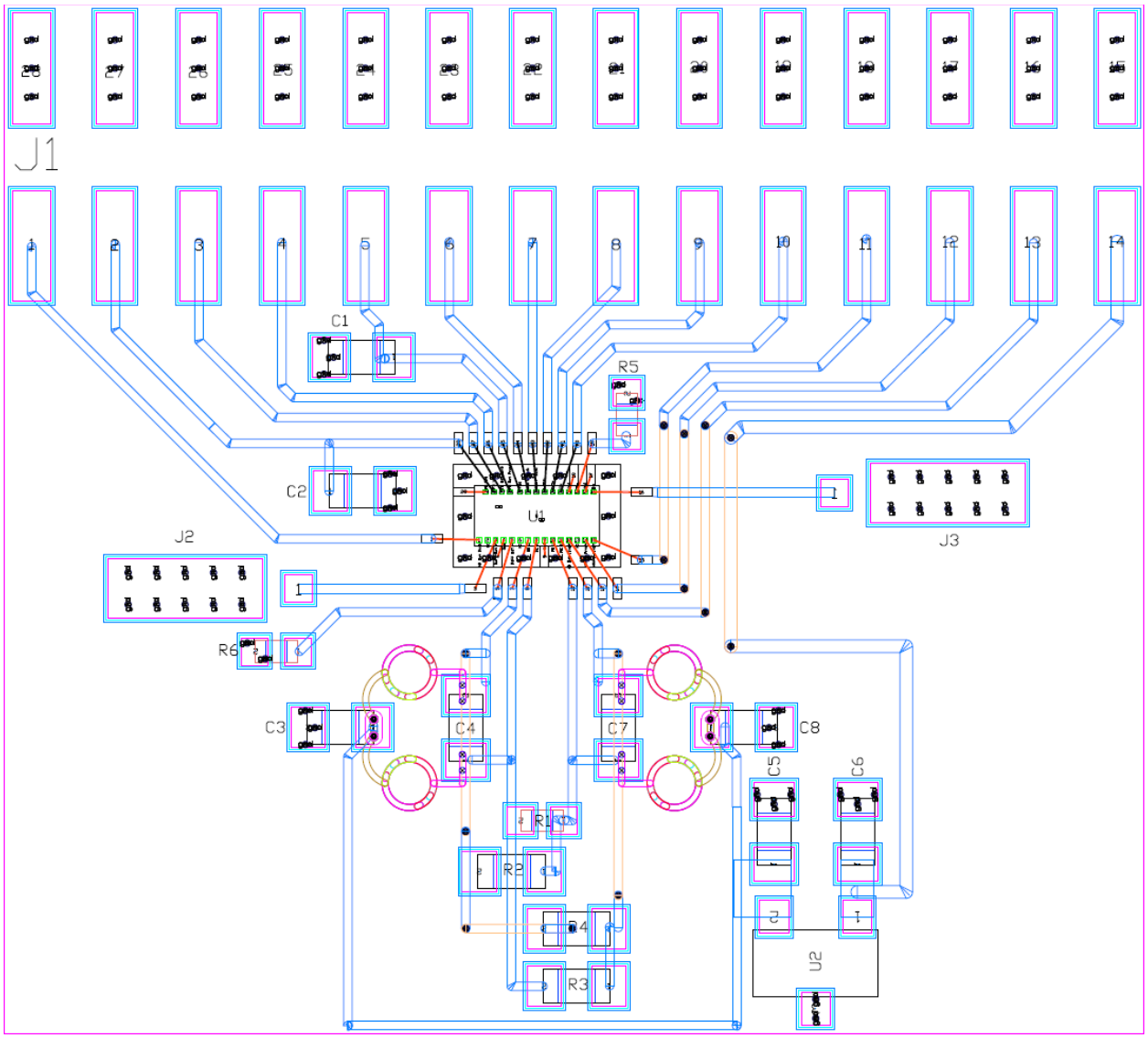


Figure C.2 – LTCC Printed Circuit Board Layout

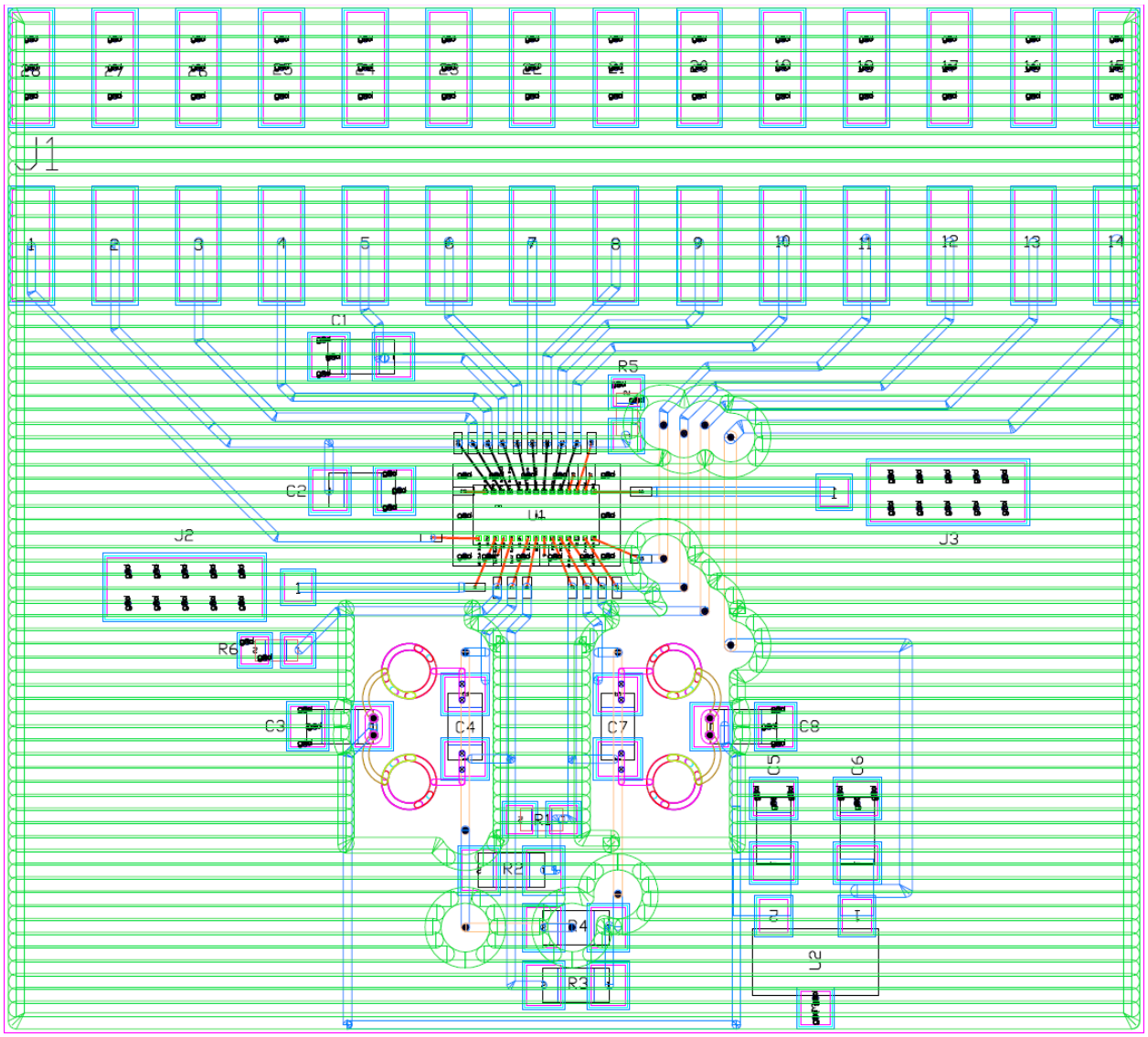


Figure C.3 – LTCC Printed Circuit Board Layout with Ground Plane

APPENDIX D – Q-ENHANCED FILTER TEST APPLICATION

The Q-Enhanced Filter Test Application is written for computers exclusively running Windows operating systems. It has been successfully run on Windows XP/Vista/7 machines. The test application was written in C# using Visual Studio 2008 with.NET Framework v3.5 Sp1. The test application is meant to operate as a means of communicating with the Q-Enhanced filter board. Communication is accomplished by using a Virtual COM Port driver provided by Silicon Labs. For more information regarding this driver, refer to Appendix E.

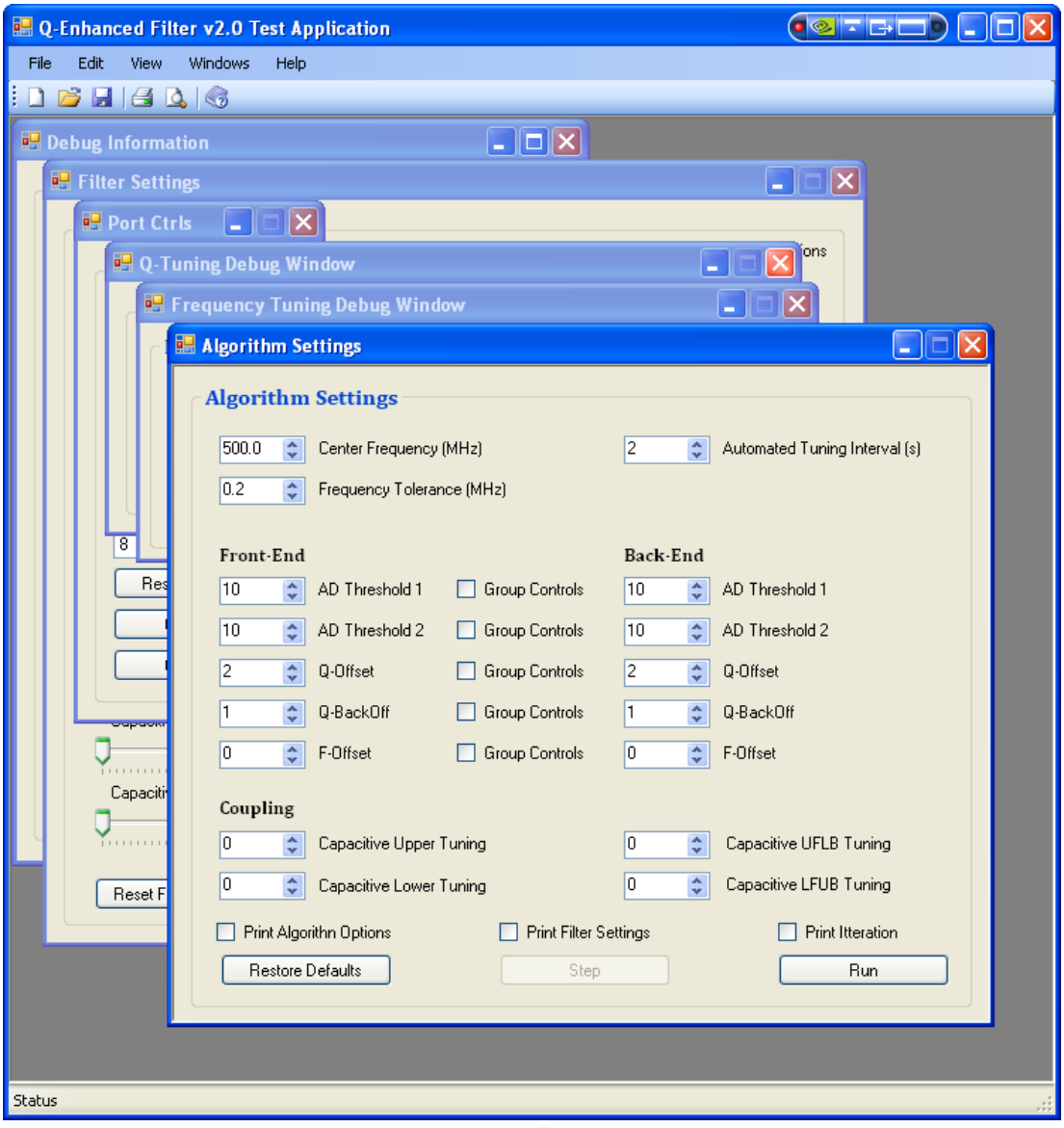


Figure D.1 – Test Application at Start Up

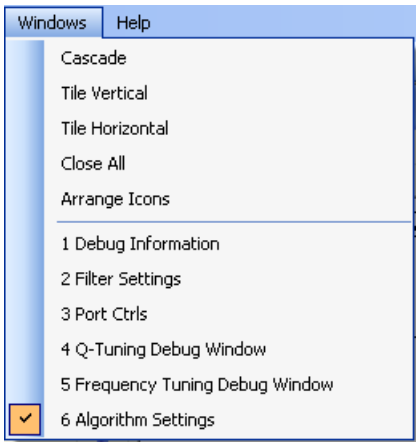


Figure D.2 – Windows Submenu

When the test application is first opened, the forms are cascaded as shown in figure D.1 on the previous page. Each form is held within the container titled 'Q-Enhanced Filter v2.0 Test Application'. The container has File, Edit, View, Window and Help menus. The container also has New, Open, Save, Print, Print Preview, and Help icons. The only valid option of the previously mentioned menus and icons is the Windows submenu. The Windows submenu, shown in figure D.2, contains options for form arrangement and selection.

D.1 Form Descriptions

D.1.1 Port Controls Form

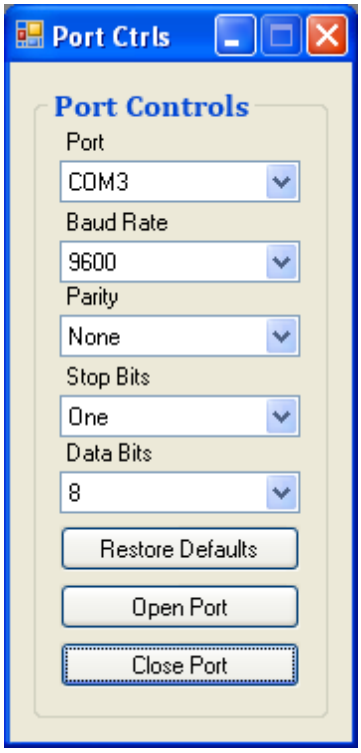


Figure D.3 – Port Controls Form

The Port Controls form is used to establish a connection to the dsPIC33 through the CP2103 USB-to-UART Bridge on the Q-Enhanced Filter Board. The Virtual COM port must be selected from the port drop down menu. Refer to Appendix E to learn how to verify which COM port is being used by the CP2103 USB-to-UART Bridge. Each time the port drop down menu is selected, the available ports list is repopulated. The microcontroller's UART has been configured to use the default Baud Rate, Parity, Stop Bits, and Data Bits shown in figure D.3.

A COM port must be open before any information can be sent to or received from the microcontroller on the Q-Enhanced Filter Board. Click the Open button to open the select COM port. Similarly, click the Close button to close the selected COM Port. When the port is successfully opened or closed, a message will be displayed in the Debug Information form. If the Open button is clicked while a port is currently open, the message in D.4 will appear. Selecting *yes* closes the currently open port and then opens the selected port.

The application will automatically close any open COM ports on exit, but it is still good practice to close the COM port prior to exit.

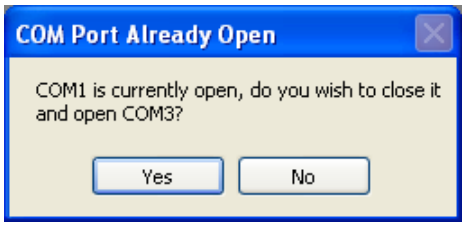


Figure D.4 – COM Port Already Open Message

D.1.2 Debug Information Form

The Debug Information form, shown in figure D.5, is used to display transmitted and received debug information. Text color indicates the type of information being displayed. Table D.1 gives a description for each text color. Click the Clear button to clear the debug text. Click the Copy button to copy the debug text to the Windows Clipboard. Clicking the save button opens a dialog box which allows the text to be saved as a Text File (*.txt) or a Rich Text File (*.rtf). The advantage of saving the text as a Rich Text File is that the text coloration is preserved. The Debug Information form is the only resizable form.

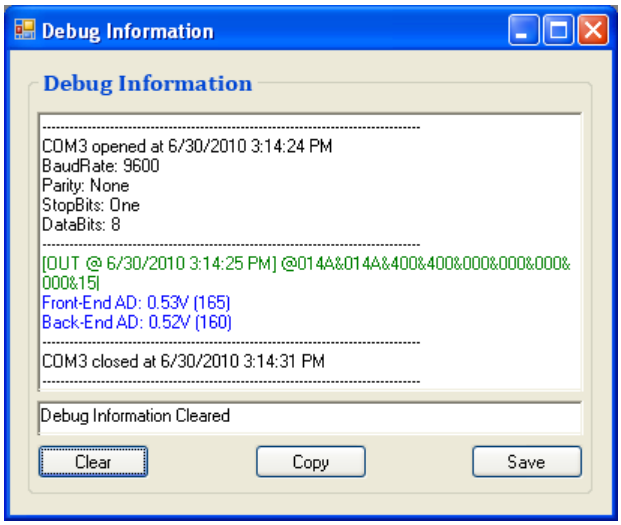


Figure D.5 – Debug Information Form

Debug Information Text Color Descriptions	
Text Color	**Description**
Black	General
Green	Outgoing ASCII Characters
Blue	Incoming ASCII Characters
Red	Error
Orange	Warning

Table D.1 – Debug Information Text Color Descriptions

D.1.3 Filter Settings Form

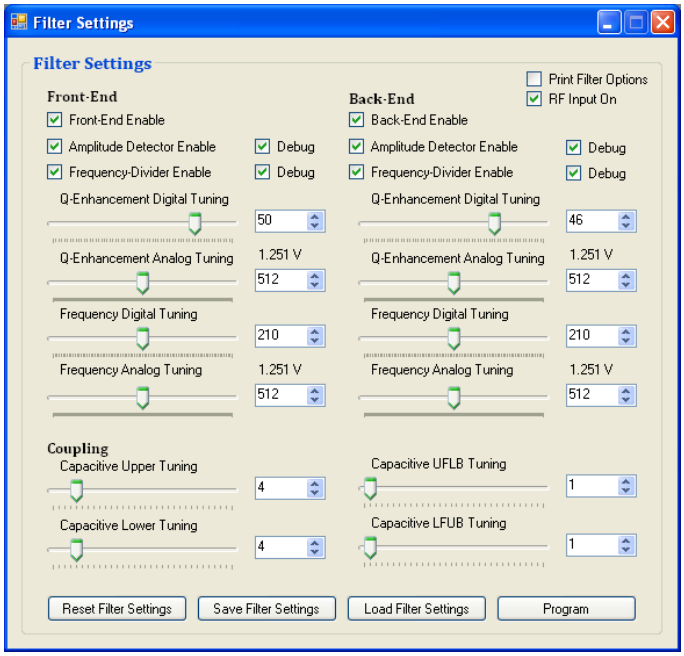


Figure D.6 – Filter Settings Form

The Filter Settings form, shown in figure D.6, is used to manually tune the Q-Enhanced Filter. Identical Q-Enhancement, frequency tuning, and enable controls exist for the Front-End and Back-End. Selecting the debug checkboxes causes the respective debug information to be displayed in the Debug Information form. Selecting the RF Input On checkbox allows the RF input to pass through the RF switch to the differential inputs of the filter. Selecting the Print Filter Settings checkbox displays the current filter settings in the Debug Information form.

Clicking the Reset Filter Settings button sets each control to its respective default, but does not apply the settings to the filter. Clicking the Save Filter Setting button allows the current filter settings to be saved as a text file. Clicking the Load Filter Settings button allows a previous set of saved filter settings to be loaded. Clicking the Program button creates and sends a control string composed of the current settings to the microcontroller via the CP2103 USB-to-UART Bridge. The control string is in the following format:

@ FENDCON & BENDCON & CAPCON1 & CAPCON2 & FANAF & FANAQ & BANAF & BANAQ & DEBUG |

The '@' character indicates the start of the control string. Similarly, the '|' character indicates the end of the control string. The '&' character acts as a separator between control words. Table D.2 describes each control word. The microcontroller must be running the **qefilt_usb_control** program for the filter settings to be applied. See Appendix G for the qefilt_usb_control source code.

Filter Settings Control String Format Information			
FENDCON: Front-End Control Word (4 Hex Digits)			
Front-End Amplitude Enable (1 Bit)	Front-End Q-Tuning (6 Bits)	Front-End F-Tuning (8 Bits)	Front-End Enable (1 Bit)
MSB			LSB
BENDCON: Back-End Control Word (4 Hex Digits)			
Back-End Amplitude Enable (1 Bit)	Back-End Q-Tuning (6 Bits)	Back-End F-Tuning (8 Bits)	Back-End Enable (1 Bit)
MSB			LSB
CAPCON1: Capacitive Tuning Control Word 1 (3 Hex Digits)			
Front-End Frequency Divider Enable (1 Bit)	Lower Coupling (5 Bits)	Upper Coupling (5 Bits)	
MSB			LSB
CAPCON2: Capacitive Tuning Control Word 2 (3 Hex Digits)			
Back-End Frequency Divider Enable (1 Bit)	UFLB Coupling (5 Bits)	LFUB Coupling (5 Bits)	
MSB			LSB
FANAF: Front-End Analog F-Tuning Control Word (3 Hex Digits)			
Front-End Analog F-Tuning (10 Bits)			
MSB			LSB
FANAQ: Front-End Analog Q-Tuning Control Word (3 Hex Digits)			
Front-End Analog Q-Tuning (10 Bits)			
MSB			LSB
BANAF: Back-End Analog F-Tuning Control Word (3 Hex Digits)			
Back -End Analog F-Tuning (10 Bits)			
MSB			LSB
BANAQ: Back -End Analog Q-Tuning Control Word (3 Hex Digits)			
Front-End Analog Q-Tuning (10 Bits)			
MSB			LSB

DEBUG: Debug Control Word (3 Hex Digits)				
RF Switch On	Back-End Freq. Divider Debug (1 Bit)	Back -End Amp. Detect Debug (1 Bit)	Front-End Freq. Divider Debug (1 Bit)	Front-End Amp. Detect Debug (1 Bit)
MSB				LSB

Table D.2 – Filter Settings Control String Format Information

When the microcontroller is prompted, it sends a similar string containing the debug information to the test application. The debug string is in the following format:

^ FRONTENDAD & FRONTENDFCNT & BACKENDAD & BACKENDFCNT |

The '^' character indicates the start of the debug string. Similarly, the '|' character indicates the end of the debug string. The '&' character acts as a separator between debug words. Table D.3 describes each debug word.

Debug String Format Information	
FRONTENDAD: Front-End Amplitude Detector Reading (3 Hex Digits)	
Front-End Amplitude Detector Reading (10 Bits)	
MSB	LSB
FRONTENDFCNT: Front-End Frequency Count (4 Hex Digits)	
Front-End Frequency Count (13 Bits)	
MSB	LSB
BACKENDAD: Back-End Amplitude Detector Reading (3 Hex Digits)	
Back-End Amplitude Detector Reading (10 Bits)	
MSB	LSB
FRONTENDFCNT: Back-End Frequency Count (4 Hex Digits)	
Back-End Frequency Count (13 Bits)	
MSB	LSB

Table D.3 – Debug String Format Information

D.1.4 Frequency Tuning Debug Form

The Frequency Tuning Debug form, shown in figure D.7, allows a sweep of digital frequency tuning settings to be performed. The sweep can be applied to the Front-End or Back-End by selecting the appropriate radio button. All controls are inherited from the Filter Settings form. The respective End's digital frequency tuning setting is replaced by the current sweep value, and the frequency divider/frequency divider debug is enabled. Because filter settings are inherited, it is imperative that the Program button on the Filter Settings form is clicked prior to running a sweep. The Start F and Stop F values indicate the first and last digital frequency tuning settings that will be applied during the sweep. The current digital frequency setting and corresponding frequency count (count of 1 is equivalent to 0.1 MHz) is displayed as the sweep progresses.

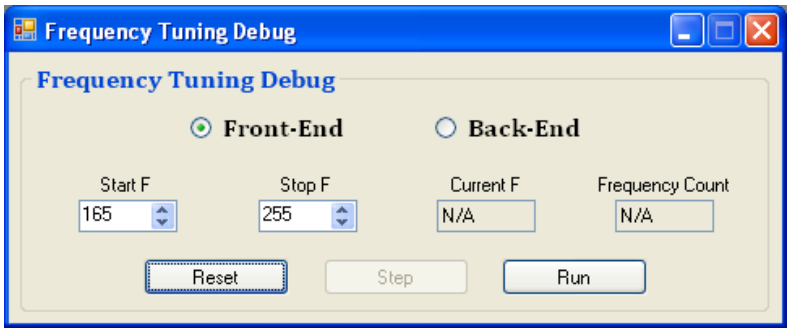


Figure D.7 – Frequency Tuning Debug Form Defaults

A sweep is started when the Run button is clicked. The Automate Sweep Dialog asks if the sweep should be automated. Selecting *yes* incrementally advances the sweep once per second. If *no* is selected, the Step button must be clicked to incrementally advance the sweep. The sweep cannot be advanced until the application receives the current frequency count from the microcontroller; if five seconds elapse without receipt, a dialog is displayed and sweep is stopped.

Figure D.8 shows the Frequency Tuning Debug form during a sweep of the Front-End. Clicking the Stop button (same button as Run) stops the sweep. The Reset button stops the sweep and restores the default sweep controls.

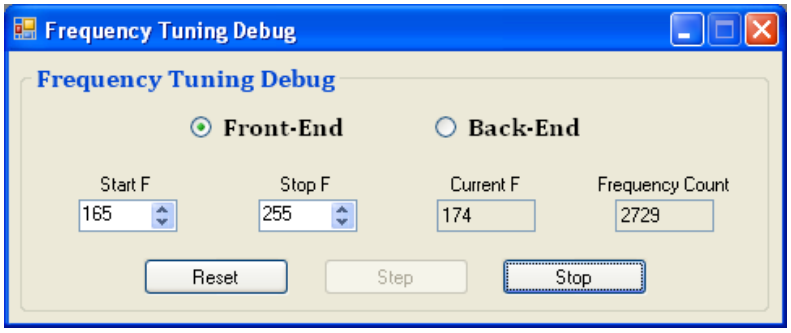


Figure D.8 – Frequency Tuning Debug Form Running

Once the sweep has finished, a dialog will ask if the sweep data should be saved to an excel spreadsheet. Selecting *yes* allows the excel file to be saved to a user selected location. The Save Debug Information dialog suggests a name based on the selected *End* and that *End's* digital Q-Enhancement setting. The resulting excel file, similar to figure D.9, displays the sweep information. Additionally, the final filter settings are displayed below the plot of the sweep.

Front-End Frequency Tuning Sweep (7/3/2010 1:02:35 PM)			
StartF	165		
StopF	255		
F-Tune	Count		
165	5238		
166	5233		
167	5227		
168	5222		
169	5216		
170	5211		
171	5206		
172	5200		
173	5195		
174	5190		
175	5184		
176	5180		
177	5174		
178	5168		
179	5164		
180	5158		
181	5152		
182	5147	Front-End Enable	Enabled
183	5143	Front-End Digital Q Tuning	63
184	5137	Front-End Analog Q Tuning	0
185	5132	Front-End Digital F Tuning	255
186	5128	Front-End Analog F Tuning	0
187	5123	Front-End Amplitude Detector Enable	Enabled
188	5117	Front-End Frequency Divider Enable	Enabled
189	5112		
190	5107	Back-End Enable	Disabled
191	5102	Back-End Digital Q Tuning	0
192	5100	Back-End Analog Q Tuning	0
193	5096	Back-End Digital F Tuning	165
194	5090	Back-End Analog F Tuning	0
195	5086	Back-End Amplitude Detector Enable	Disabled
196	5080	Back-End Frequency Divider Enable	Disabled
197	5075		
198	5071	Upper Capacitive Coupling Tuning	0
199	5066	Lower Capacitive Coupling Tuning	0
200	5060	UFLB Capacitive Coupling Tuning	0
201	5055	LFUB Capacitive Coupling Tuning	0
202	5051		
203	5046		
204	5040		
205	5036		

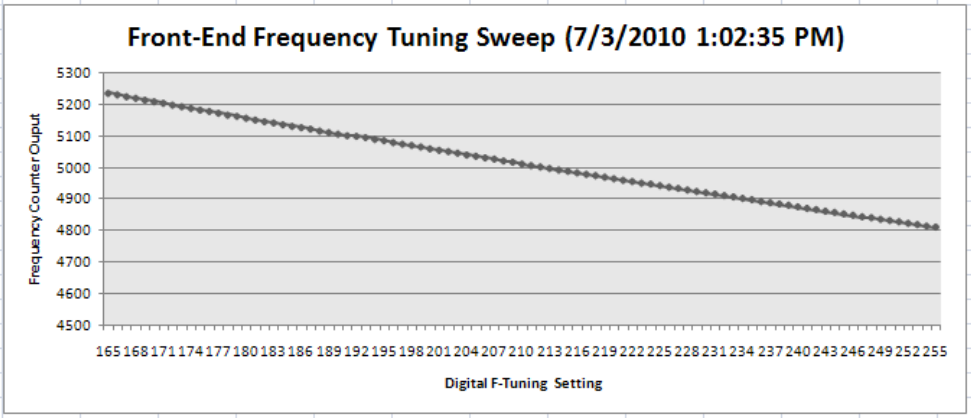


Figure D.9 – Frequency Tuning Debug Sweep Excel Form Example

D.1.5 Q-Tuning Debug Form

The Q-Tuning Debug form, shown in figure D.10, allows a sweep of digital Q-Enhancement settings to be performed. The sweep can be applied to the Front-End or Back-End by selecting the appropriate radio button. All controls are inherited from the Filter Settings form. The respective End's digital Q-Enhancement setting is replaced by the current sweep value, and the amplitude detector/amplitude detector debug is enabled. Because filter settings are inherited, it is imperative that the Program button on the Filter Settings form is clicked prior to running a sweep. The Start Q and Stop Q values indicate the first and last digital Q-Enhancement settings that will be applied during the sweep. The current Q-Enhancement setting and corresponding quantized amplitude detector reading is displayed as the sweep progresses.

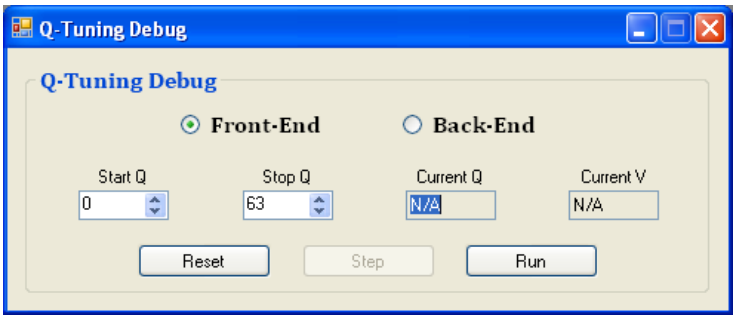


Figure D.10 – Q-Tuning Debug Form Defaults

A sweep is started when the Run button is clicked. The Automate Sweep Dialog asks if the sweep should be automated. Selecting *yes* incrementally advances the sweep once per second. If *no* is selected, the Step button must be clicked to incrementally advance the sweep. The sweep cannot be advanced until the application receives the current frequency count from the microcontroller; if five seconds elapse without receipt, a dialog is displayed and sweep is stopped.

Figure D.11 shows the Q-Tuning Debug form during a sweep of the Front-End. Clicking the Stop button (same button as Run) stops the sweep. The Reset button stops the sweep and restores the default sweep controls.

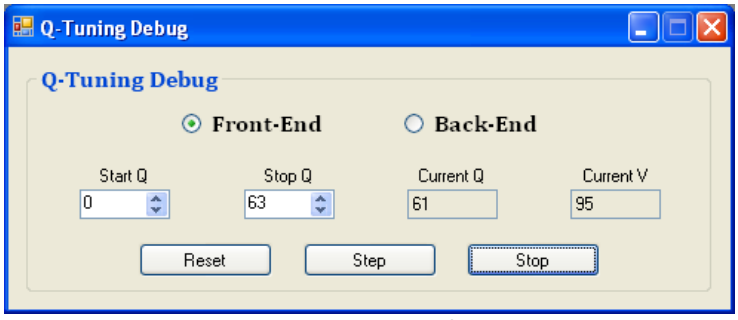


Figure D.11 – Q-Tuning Debug Form Running

Once the sweep has finished, a dialog will ask if the sweep data should be saved to an excel spreadsheet. Selecting *yes* allows the excel file to be saved to a user selected location. The Save Debug Information dialog suggests a name based on the selected *End* and that *End's* digital frequency setting. The resulting excel file, similar to figure D.12, displays the sweep information. Additionally, the final filter settings are displayed below the plot of the sweep.

Front-End Q-Tuning Sweep (7/3/2010 1:04:29 PM)			
StartQ	0		
StopQ	63		
Nq	Vq		
0	167		
1	167		
2	167		
3	167		
4	167		
5	167		
6	167		
7	167		
8	167		
9	167		
10	167		
11	167		
12	167		
13	167		
14	167		
15	167		
16	167		
17	167	Front-End Enable	Enabled
18	167	Front-End Digital Q Tuning	63
19	167	Front-End Analog Q Tuning	0
20	167	Front-End Digital F Tuning	210
21	168	Front-End Analog F Tuning	0
22	168	Front-End Amplitude Detector Enable	Enabled
23	168	Front-End Frequency Divider Enable	Enabled
24	167		
25	168	Back-End Enable	Disabled
26	168	Back-End Digital Q Tuning	0
27	168	Back-End Analog Q Tuning	0
28	168	Back-End Digital F Tuning	165
29	168	Back-End Analog F Tuning	0
30	168	Back-End Amplitude Detector Enable	Disabled
31	168	Back-End Frequency Divider Enable	Disabled
32	168		
33	168	Upper Capacitive Coupling Tuning	0
34	168	Lower Capacitive Coupling Tuning	0
35	168	UFLB Capacitive Coupling Tuning	0
36	168	LFUB Capacitive Coupling Tuning	0
37	168		
38	168		
39	168		
40	168		

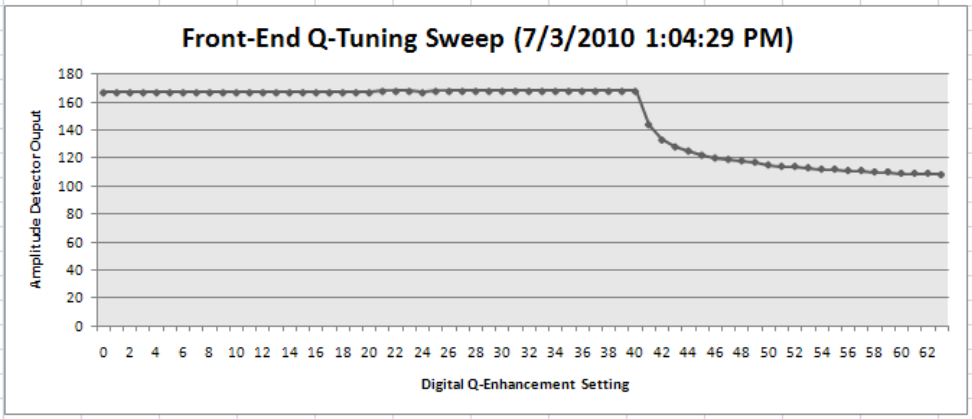


Figure D.12 – Q-Tuning Debug Sweep Excel Form Example

D.1.6 Algorithm Settings Form

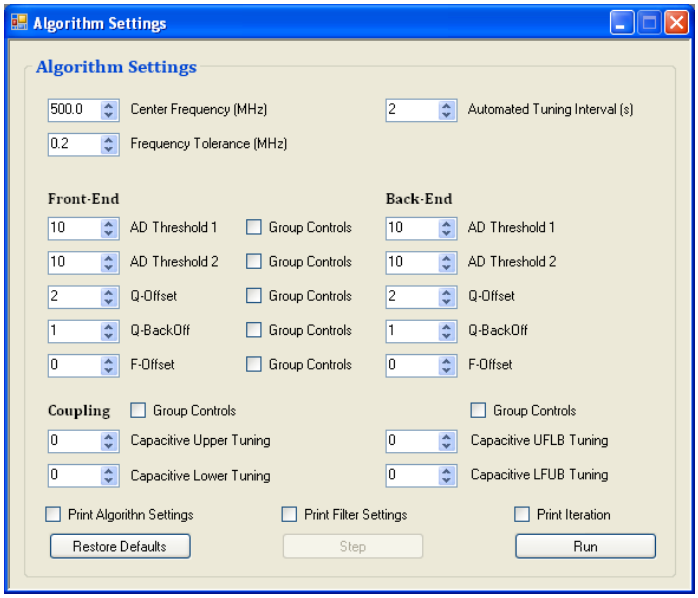


Figure D.14 – Algorithm Settings Form

The Algorithm Settings form, shown in figure D.13, provides the filter tuning algorithm's settings. The filter's Center Frequency is specified to a 100 kHz resolution. Similarly, a Frequency Tolerance used in the algorithms coarse tuning is also specified to a 100 kHz resolution. Identical controls exist for the Front-End and Back-End. Two amplitude detector thresholds are used to determine the Q-Enhancement level for critical oscillation. A Q-Enhancement offset is used to increase oscillation past critical oscillation ensuring the stability of the frequency divider outputs. A Q-Enhancement Back-Off is used to establish bandwidth in the single-pole tuning algorithm, while it is used as an adjustment factor in the two-pole tuning algorithm. A Frequency Offset is used to counter the frequency shift caused by Q-Enhancement back-off. Checking the Print Algorithm Settings checkbox causes the microcontroller to print the current Algorithm Options to the Debug Information form. Similarly, checking the Print Filter Settings checkbox causes the microcontroller to print the current Filter Settings to the Debug Information form. Checking the Print Iteration checkbox causes the microcontroller to print the number of times the algorithm has run to the Debug Information form.

Click the Run button to start the algorithm. The Automate Algorithm Dialog asks if the algorithm should be automated. Selecting *yes* applies the algorithm settings each time the Automated Tuning Interval elapses. The remaining time before the algorithm settings are next applied is displayed in red text as shown in figure D.15. If *no* is selected, the Step button becomes enabled and allows the algorithm settings to be applied when clicked.

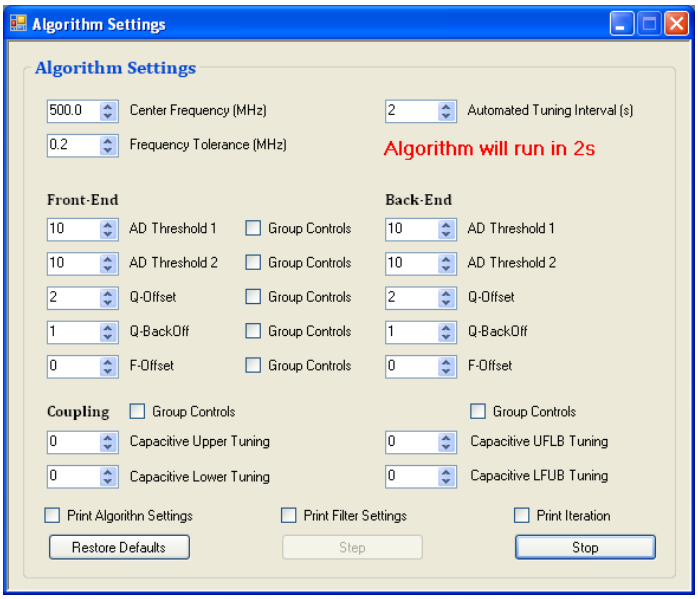


Figure D.14 – Algorithm Settings Form

Clicking the Stop button stops the algorithm settings from being applied. Clicking the Restore Defaults button stops the algorithm settings from being applied and restores default values of the controls. The algorithm settings are sent to the microcontroller via the CP2103 USB-to-UART Bridge in the following control string format:

@ CENTERFREQ & FREQTOL & FRONTENDAD & FRONTENDQF & BACKENDAD & BACKENDQF & COUPLAT & COUPCROSS & DEBUG |

The '@' character indicates the start of the control string. Similarly, the '|' character indicates the end of the control string. The '&' character acts as a separator between control words. Table D.4 describes each control word. The microcontroller must be running one of the **algorithm** programs for these settings to be applied. See Appendices H-I for the algorithm source codes.

Algorithm Settings Control String Format Information		
CENTERFREQ: Filter Center Frequency (4 Hex Digits)		
Filter Center Frequency (13 Bits)		
MSB		LSB
FREQTOL: Coarse Tuning Frequency Tolerance (2 Hex Digits)		
Coarse Tuning Frequency Tolerance (7 Bits)		
MSB		LSB
FRONTENDAD: Front-End Amplitude Detector Thresholds (4 Hex Digits)		
Front-End Amplitude Detector Threshold 1 (7 Bits)	Front-End Amplitude Detector Threshold 2 (7 Bits)	
MSB		LSB
FRONTENDQF: Front -End Q and F Settings (3 Hex Digits)		
Front-End Q-Offset (4 Bits)	Front-End Q-Back-Off (4 Bits)	Front-End F-Offset (4Bits)
MSB		LSB
BACKENDAD: Back-End Amplitude Detector Thresholds (4 Hex Digits)		
Back-End Amplitude Detector Threshold 1 (7 Bits)	Back-End Amplitude Detector Threshold 2 (7 Bits)	
MSB		LSB
BACKENDQF: Back-End Q and F Settings (3 Hex Digits)		
Back-End Q-Offset (4 Bits)	Back-End Q-Back-Off (4 Bits)	Back-End F-Offset (4Bits)
MSB		LSB
COUPLAT: Lateral Coupling (3 Hex Digits)		
Upper Coupling (6 Bits)	Lower Coupling (6 Bits)	
MSB		LSB
COUPCROSS: Cross Coupling (3 Hex Digits)		
UFLB Coupling (6 Bits)	LFUB Coupling (6 Bits)	
MSB		LSB
DEBUG: Back -End Analog Q-Tuning (1 Hex Digit)		
Print Algorithm Settings (1 Bit)	Print Filter Settings (1 Bit)	Print Iteration (1 Bit)
MSB		LSB

Table D.4 – Algorithm Settings Control String Format Information

D.2 TROUBLESHOOTING

D.2.1 APPLICATION LAUNCHING PROBLEMS

If the application does not launch or a Window's exception occurs on launch, then it is recommended to update the Windows .NET Framework. Go to Windows Update and apply any .NET Framework related updates.

D.2.2 COMMUNICATION PROBLEMS

If a communication problem such as not being able to receive data or garbage data is received from the filter board occurs, check the configuration bits on the microcontroller code. The UART baud rate of the microcontroller is dependent on the microcontroller's clock frequency. If there is a discrepancy the clock frequency, an incorrect baud rate will be used causing communication problems such as *garbage* text being displayed in the Debug Information Form. If the configuration bits are correct, refer to Appendix E for CP2103 configuration.

APPENDIX E – CP2103 SETUP GUIDE

When using the Q-Enhanced Filter Board with Q-Enhanced Filter Test Application on a Windows PC for the first time, the CP2103 drivers need to be installed. This appendix is meant to act as a guide in the driver installation and configuration of the CP2103 USB-to-UART Bridge.

E.1 DRIVER SETUP

1) Connect a USB cable from the Q-Enhanced Filter Board to a Windows PC. The computer recognizes that the CP2103 is connected when the **Found New Hardware** dialog is displayed as in figure E.1.

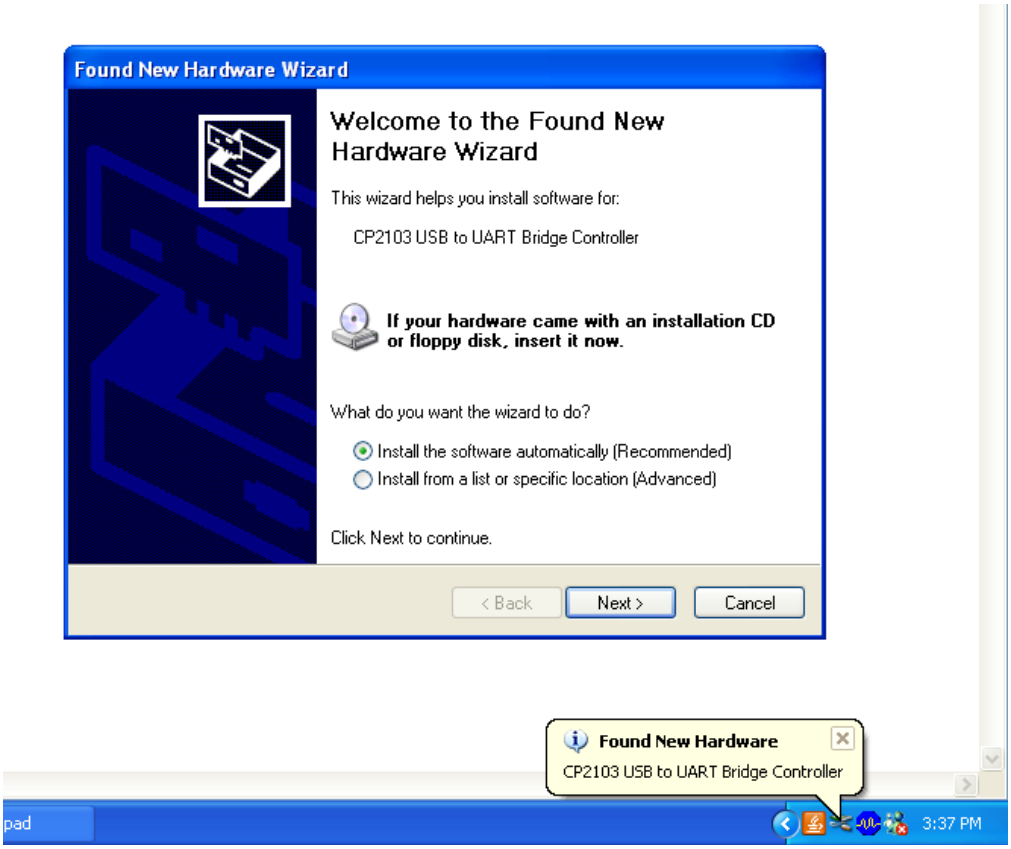


Figure E.1 – Found New Hardware

2) Navigate to the **USB to UART Bridge** page on the **Silicon Labs** website or use the following URL:

   http://www.silabs.com/products/interface/usbtouart/Pages/default.aspx

3) Click on the **Virtual Com Port (VCP) Download** link under the **Design Resources** heading or use the following URL:

   http://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx

4) Download the driver by clicking on the **VCP Driver Kit** link under the **Download for Windows XP/Server 2003/Vista/7** heading or use the following URL:

http://www.silabs.com/Support%20Documents/Software/CP210x_VCP_Win_XP_S2K3_Vista_7.exe

5) Once the **CP210x_VCP_Win_XP_S2K3_Vista_7.zip** download has completed, extract the file using WinZip or equivalent program. Then run the extracted file.
6) Click next on the **Welcome to the InstallSheild Wizard for Silicon Labratories CPI210x VCP Drivers for Windows XP/2003 Server/Vista/7** screen.
7) Check **I accept the terms of the license agreement** and then click next on the **License Agreement** screen.
8) Click next on the **Choose Destination Location** screen.
9) Click Download the driver by clicking on the **VCP Driver Kit** link under the **Download for Windows XP/Server 2003/Vista/7** heading or use the following URL:
10) Click install on the **Ready to Install the Program** screen.
11) Check **Launch the CP210x VCP Driver Installer** and click finish on the **InstallShield Wizard Complete** screen.
12) Click install on the **Silicon Laboratories CP210x USB to UART Bridge Driver Installer** window.
13) Once the installation is complete, click ok on the **Installation complete successfully** message.

E.1 CONFIGURATION

1) Navigate to the **USB to UART Bridge** page on the **Silicon Labs** website or use the following URL:

http://www.silabs.com/products/interface/usbtouart/Pages/default.aspx

2) Click on the drop down menu that says **Software Downloads** under the **Design Tools** heading and select **AN205SW** to download it.
3) Once the AN205SW.zip download has completed, extract the file into a directory and launch the **CP210xBaudRateAliasConfig** application.
4) If a device is not shown under **Connected Device** click refresh, if there still is not a device shown try the installation process again.
5) Click **Get Configuration** and wait for the dialog to be populated.

6) Select the line that has a **UART Baud Rate** of **9600**, as shown in figure E.2, and click **Set Configuration.**

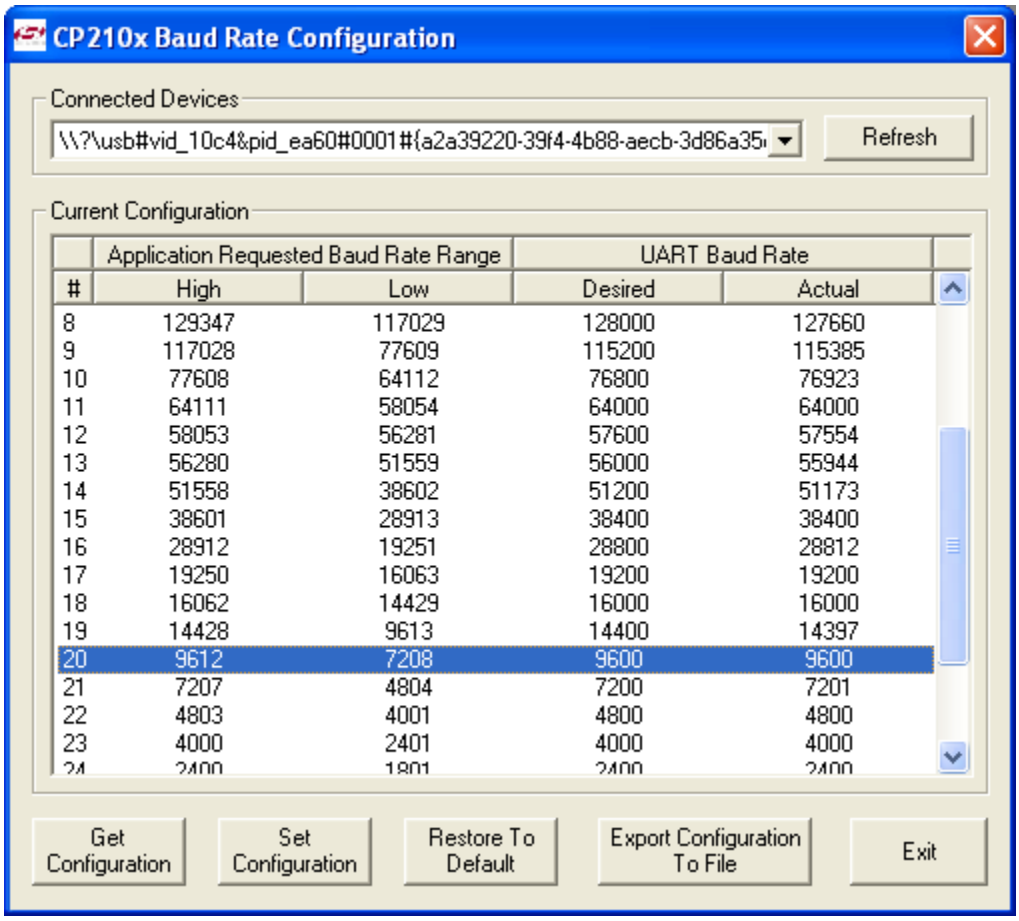


Figure E.2 - CP2103 Baud Rate Configuration

7) Click yes to set the current baud rage configuration to the part selected.

8) Navigate to the Windows Device Manger.

9) Expand the **Ports (COM & LPT)** list as shown in figure E.3.

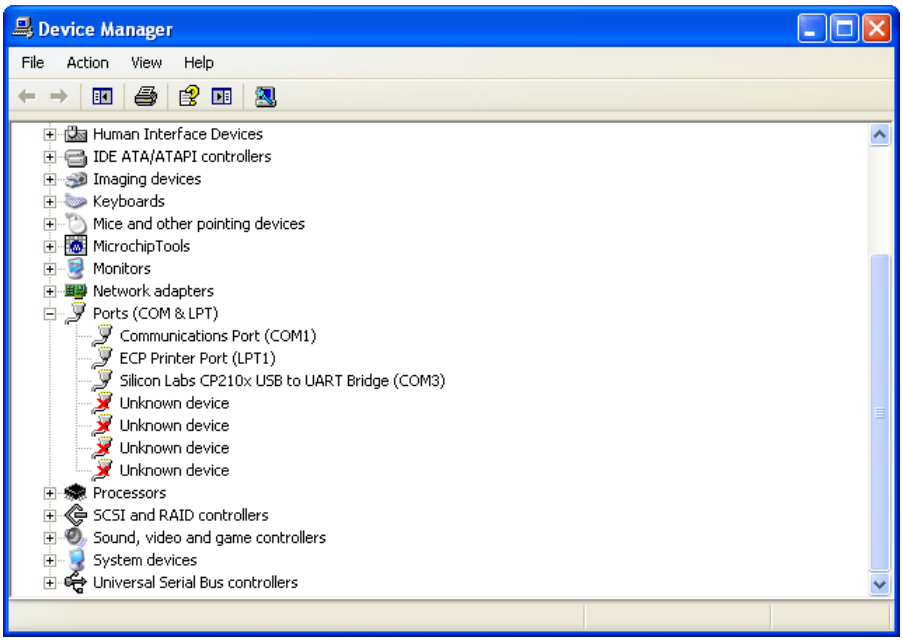


Figure E.3 – Device Manager

10) Double click on the **Silicon Labs CP210x USB to UART Bridge** entry and navigate to the **Port Settings** tab.

11) Configure all Port Settings to match those shown in figure E.4.

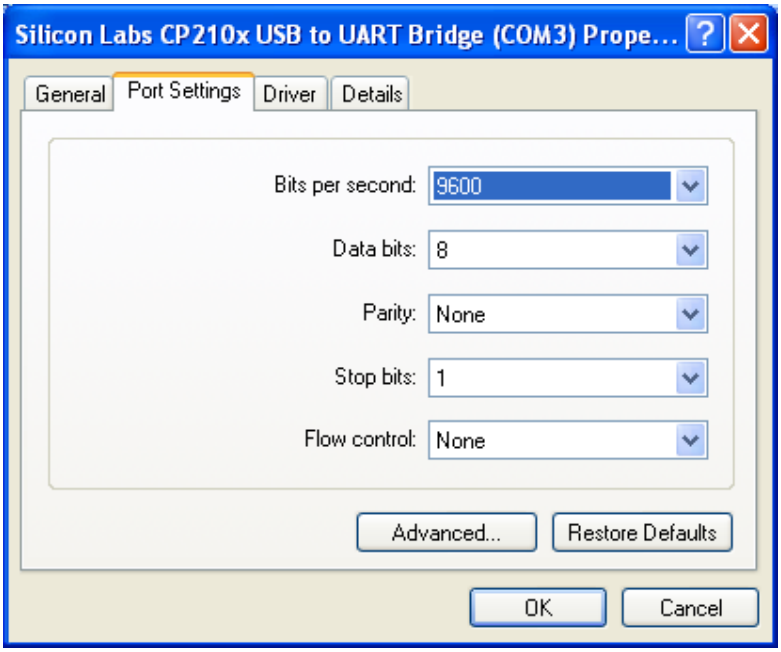


Figure E.4 – COM Port Settings

12) Click Ok to complete the configuration of the CP2103.

APPENDIX F – Q-ENHANCED FILTER TEST APPLICATION SOURCE CODE

PROGRAM.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace QEFilterV2TestAppV2
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new MDIParent1());
        }
    }
}
```

MDIPARENT1.CS

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace QEFilterV2TestAppV2
{
    public partial class MDIParent1 : Form
    {
        private int childFormNumber = 0;

        private AboutBox1 aboutBox;
        private Form frmDebugText;
        private Form frmFilterSettings;
        private Form frmPortControls;
        private Form frmQTuneDebug;
        private Form frmFTuneDebug;
        private Form frmAlgorithmSettings;

        private DebugText dt;
        private CommunicationManager cm;
        private FilterControls fs;
        private PortCtrls pc;
        private QTuneDebug qtd;
        private FTuneDebug ftd;
        private AlgorithmSettings alg;

        public MDIParent1()
        {
            InitializeComponent();
            this.Closing += new CancelEventHandler(MDI_Closing);
            this.WindowState = FormWindowState.Maximized;

            dt = new DebugText();
            cm = new CommunicationManager(dt);
            fs = new FilterControls(cm, dt);
            qtd = new QTuneDebug(fs, dt, cm);
            ftd = new FTuneDebug(fs, dt, cm);
            alg = new AlgorithmSettings(cm, dt);

            pc = new PortCtrls();
            cm.QTuneDebug = qtd;
            cm.FTuneDebug = ftd;

            frmDebugText = new frmDebugTxt(dt);
            frmDebugText.MdiParent = this;
            frmDebugText.Show();

            frmFilterSettings = new frmFilterControls(fs,dt);
            frmFilterSettings.MdiParent = this;
            frmFilterSettings.Show();

            frmPortControls = new frmPortCtrls(pc,dt,cm);
            frmPortControls.MdiParent = this;
            frmPortControls.Show();

            frmQTuneDebug = new frmQTuneDebug(qtd,dt,cm,fs);
            frmQTuneDebug.MdiParent = this;
            frmQTuneDebug.Show();

            frmFTuneDebug = new frmFTuneDebug(ftd, dt, cm, fs);
            frmFTuneDebug.MdiParent = this;
            frmFTuneDebug.Show();

            frmAlgorithmSettings = new frmAlgorithmSettings(alg, dt, cm);
            frmAlgorithmSettings.MdiParent = this;
            frmAlgorithmSettings.Show();

        }

        private void MDI_Closing(object sender, CancelEventArgs cArgs)
        {
            try
            {
                if (sender == this)
                {
```

```
                if (cm != null && cm.isOpen)
                    cm.ComPort.Close();

                dt.OkToClose = true;
                fs.OkToClose = true;
                pc.OkToClose = true;
                qtd.OkToClose = true;

                frmDebugText.Close();
                frmFilterSettings.Close();
                frmPortControls.Close();
                frmQTuneDebug.Close();
                frmFTuneDebug.Close();

                this.Dispose();
            }
        }
        catch (Exception ex)
        {
            string err = ex.Message + "\n" + ex.StackTrace;
            //MessageBox.Show(err);
        }
    }

    private void ShowNewForm(object sender, EventArgs e)
    {
        Form childForm = new Form();
        childForm.MdiParent = this;
        childForm.Text = "Window " + childFormNumber++;
        childForm.Show();
    }

    private void OpenFile(object sender, EventArgs e)
    {
        OpenFileDialog openFileDialog = new OpenFileDialog();
        openFileDialog.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
        openFileDialog.Filter = "Text Files (*.txt)|*.txt|All Files (*.*)|*.*";
        if (openFileDialog.ShowDialog(this) == DialogResult.OK)
        {
            string FileName = openFileDialog.FileName;
        }
    }

    private void SaveAsToolStripMenuItem_Click(object sender, EventArgs e)
    {
        SaveFileDialog saveFileDialog = new SaveFileDialog();
        saveFileDialog.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
        saveFileDialog.Filter = "Text Files (*.txt)|*.txt|All Files (*.*)|*.*";
        if (saveFileDialog.ShowDialog(this) == DialogResult.OK)
        {
            string FileName = saveFileDialog.FileName;
        }
    }

    private void ExitToolsStripMenuItem_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void CutToolStripMenuItem_Click(object sender, EventArgs e) {}

    private void CopyToolStripMenuItem_Click(object sender, EventArgs e) {}

    private void PasteToolStripMenuItem_Click(object sender, EventArgs e) {}

    private void ToolBarToolStripMenuItem_Click(object sender, EventArgs e)
    {
        toolStrip.Visible = toolBarToolStripMenuItem.Checked;
    }

    private void StatusBarToolStripMenuItem_Click(object sender, EventArgs e)
    {
        statusStrip.Visible = statusBarToolStripMenuItem.Checked;
    }

    private void CascadeToolStripMenuItem_Click(object sender, EventArgs e)
    {
        LayoutMdi(MdiLayout.Cascade);
    }

    private void TileVerticalToolStripMenuItem_Click(object sender, EventArgs e)
    {
        LayoutMdi(MdiLayout.TileVertical);
    }

    private void TileHorizontalToolStripMenuItem_Click(object sender, EventArgs e)
    {
        LayoutMdi(MdiLayout.TileHorizontal);
    }

    private void ArrangeIconsToolStripMenuItem_Click(object sender, EventArgs e)
    {
        LayoutMdi(MdiLayout.ArrangeIcons);
    }

    private void CloseAllToolStripMenuItem_Click(object sender, EventArgs e)
    {
        foreach (Form childForm in MdiChildren)
        {
            childForm.Close();
        }
    }

    private void cOMPortConfigToolStripMenuItem_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Open Com Port Configuration");
    }

    private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if (aboutBox == null || aboutBox.IsDisposed)
            aboutBox = new AboutBox1();
        aboutBox.Show();
        aboutBox.Focus();
    }
  }
}
```

MDIPARENT1.DESIGNER.CS

```
namespace QEFilterV2TestAppV2
{
    partial class MDIParent1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            System.ComponentModel.ComponentResourceManager resources = new System.ComponentModel.ComponentResourceManager(typeof(MDIParent1));
            this.menuStrip = new System.Windows.Forms.MenuStrip();
            this.fileMenu = new System.Windows.Forms.ToolStripMenuItem();
            this.newToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.openToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.toolStripSeparator3 = new System.Windows.Forms.ToolStripSeparator();
            this.saveToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.saveAsToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.toolStripSeparator4 = new System.Windows.Forms.ToolStripSeparator();
            this.printToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.printPreviewToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.printSetupToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.toolStripSeparator5 = new System.Windows.Forms.ToolStripSeparator();
            this.exitToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.editMenu = new System.Windows.Forms.ToolStripMenuItem();
            this.undoToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.redoToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.toolStripSeparator6 = new System.Windows.Forms.ToolStripSeparator();
            this.cutToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.copyToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.pasteToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.toolStripSeparator7 = new System.Windows.Forms.ToolStripSeparator();
            this.selectAllToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.viewMenu = new System.Windows.Forms.ToolStripMenuItem();
            this.toolBarToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.statusBarToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.windowsMenu = new System.Windows.Forms.ToolStripMenuItem();
            this.cascadeToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.tileVerticalToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.tileHorizontalToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.closeAllToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.arrangeIconsToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.helpMenu = new System.Windows.Forms.ToolStripMenuItem();
            this.contentsToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.indexToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.toolStripSeparator8 = new System.Windows.Forms.ToolStripSeparator();
            this.aboutToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.toolStrip = new System.Windows.Forms.ToolStrip();
            this.newToolStripButton = new System.Windows.Forms.ToolStripButton();
            this.openToolStripButton = new System.Windows.Forms.ToolStripButton();
            this.saveToolStripButton = new System.Windows.Forms.ToolStripButton();
            this.toolStripSeparator1 = new System.Windows.Forms.ToolStripSeparator();
            this.printToolStripButton = new System.Windows.Forms.ToolStripButton();
            this.printPreviewToolStripButton = new System.Windows.Forms.ToolStripButton();
            this.toolStripSeparator2 = new System.Windows.Forms.ToolStripSeparator();
            this.helpToolStripButton = new System.Windows.Forms.ToolStripButton();
            this.statusStrip = new System.Windows.Forms.StatusStrip();
            this.toolStripStatusLabel = new System.Windows.Forms.ToolStripStatusLabel();
            this.toolTip = new System.Windows.Forms.ToolTip(this.components);
            this.menuStrip.SuspendLayout();
            this.toolStrip.SuspendLayout();
            this.statusStrip.SuspendLayout();
            this.SuspendLayout();
            //
            // menuStrip
            //
            this.menuStrip.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
            this.fileMenu,
            this.editMenu,
            this.viewMenu,
            this.windowsMenu,
            this.helpMenu});
            this.menuStrip.Location = new System.Drawing.Point(0, 0);
            this.menuStrip.MdiWindowListItem = this.windowsMenu;
            this.menuStrip.Name = "menuStrip";
            this.menuStrip.Size = new System.Drawing.Size(632, 24);
            this.menuStrip.TabIndex = 0;
            this.menuStrip.Text = "MenuStrip";
            //
            // fileMenu
            //
            this.fileMenu.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {
            this.newToolStripMenuItem,
            this.openToolStripMenuItem,
            this.toolStripSeparator3,
            this.saveToolStripMenuItem,
            this.saveAsToolStripMenuItem,
            this.toolStripSeparator4,
            this.printToolStripMenuItem,
            this.printPreviewToolStripMenuItem,
            this.printSetupToolStripMenuItem,
            this.toolStripSeparator5,
```

```
this.exitToolStripMenuItem});
this.fileMenu.ImageTransparentColor = System.Drawing.SystemColors.ActiveBorder;
this.fileMenu.Name = "fileMenu";
this.fileMenu.Size = new System.Drawing.Size(35, 20);
this.fileMenu.Text = "&File";
//
// newToolStripMenuItem
//
this.newToolStripMenuItem.Image = ((System.Drawing.Image)(resources.GetObject("newToolStripMenuItem.Image")));
this.newToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Black;
this.newToolStripMenuItem.Name = "newToolStripMenuItem";
this.newToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.N)));
this.newToolStripMenuItem.Size = new System.Drawing.Size(151, 22);
this.newToolStripMenuItem.Text = "&New";
this.newToolStripMenuItem.Click += new System.EventHandler(this.ShowNewForm);
//
// openToolStripMenuItem
//
this.openToolStripMenuItem.Image = ((System.Drawing.Image)(resources.GetObject("openToolStripMenuItem.Image")));
this.openToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Black;
this.openToolStripMenuItem.Name = "openToolStripMenuItem";
this.openToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.O)));
this.openToolStripMenuItem.Size = new System.Drawing.Size(151, 22);
this.openToolStripMenuItem.Text = "&Open";
this.openToolStripMenuItem.Click += new System.EventHandler(this.OpenFile);
//
// toolStripSeparator3
//
this.toolStripSeparator3.Name = "toolStripSeparator3";
this.toolStripSeparator3.Size = new System.Drawing.Size(148, 6);
//
// saveToolStripMenuItem
//
this.saveToolStripMenuItem.Image = ((System.Drawing.Image)(resources.GetObject("saveToolStripMenuItem.Image")));
this.saveToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Black;
this.saveToolStripMenuItem.Name = "saveToolStripMenuItem";
this.saveToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.S)));
this.saveToolStripMenuItem.Size = new System.Drawing.Size(151, 22);
this.saveToolStripMenuItem.Text = "&Save";
//
// saveAsToolStripMenuItem
//
this.saveAsToolStripMenuItem.Name = "saveAsToolStripMenuItem";
this.saveAsToolStripMenuItem.Size = new System.Drawing.Size(151, 22);
this.saveAsToolStripMenuItem.Text = "Save &As";
this.saveAsToolStripMenuItem.Click += new System.EventHandler(this.SaveAsToolStripMenuItem_Click);
//
// toolStripSeparator4
//
this.toolStripSeparator4.Name = "toolStripSeparator4";
this.toolStripSeparator4.Size = new System.Drawing.Size(148, 6);
//
// printToolStripMenuItem
//
this.printToolStripMenuItem.Image = ((System.Drawing.Image)(resources.GetObject("printToolStripMenuItem.Image")));
this.printToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Black;
this.printToolStripMenuItem.Name = "printToolStripMenuItem";
this.printToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.P)));
this.printToolStripMenuItem.Size = new System.Drawing.Size(151, 22);
this.printToolStripMenuItem.Text = "&Print";
//
// printPreviewToolStripMenuItem
//
this.printPreviewToolStripMenuItem.Image = ((System.Drawing.Image)(resources.GetObject("printPreviewToolStripMenuItem.Image")));
this.printPreviewToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Black;
this.printPreviewToolStripMenuItem.Name = "printPreviewToolStripMenuItem";
this.printPreviewToolStripMenuItem.Size = new System.Drawing.Size(151, 22);
this.printPreviewToolStripMenuItem.Text = "Print Pre&view";
//
// printSetupToolStripMenuItem
//
this.printSetupToolStripMenuItem.Name = "printSetupToolStripMenuItem";
this.printSetupToolStripMenuItem.Size = new System.Drawing.Size(151, 22);
this.printSetupToolStripMenuItem.Text = "Print Setup";
//
// toolStripSeparator5
//
this.toolStripSeparator5.Name = "toolStripSeparator5";
this.toolStripSeparator5.Size = new System.Drawing.Size(148, 6);
//
// exitToolStripMenuItem
//
this.exitToolStripMenuItem.Name = "exitToolStripMenuItem";
this.exitToolStripMenuItem.Size = new System.Drawing.Size(151, 22);
this.exitToolStripMenuItem.Text = "E&xit";
this.exitToolStripMenuItem.Click += new System.EventHandler(this.ExitToolsStripMenuItem_Click);
//
// editMenu
//
this.editMenu.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {
this.undoToolStripMenuItem,
this.redoToolStripMenuItem,
this.toolStripSeparator6,
this.cutToolStripMenuItem,
this.copyToolStripMenuItem,
this.pasteToolStripMenuItem,
this.toolStripSeparator7,
this.selectAllToolStripMenuItem});
this.editMenu.Name = "editMenu";
this.editMenu.Size = new System.Drawing.Size(37, 20);
this.editMenu.Text = "&Edit";
//
// undoToolStripMenuItem
//
this.undoToolStripMenuItem.Image = ((System.Drawing.Image)(resources.GetObject("undoToolStripMenuItem.Image")));
this.undoToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Black;
this.undoToolStripMenuItem.Name = "undoToolStripMenuItem";
this.undoToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.Z)));
this.undoToolStripMenuItem.Size = new System.Drawing.Size(167, 22);
this.undoToolStripMenuItem.Text = "&Undo";
//
// redoToolStripMenuItem
//
this.redoToolStripMenuItem.Image = ((System.Drawing.Image)(resources.GetObject("redoToolStripMenuItem.Image")));
this.redoToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Black;
this.redoToolStripMenuItem.Name = "redoToolStripMenuItem";
this.redoToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.Y)));
this.redoToolStripMenuItem.Size = new System.Drawing.Size(167, 22);
this.redoToolStripMenuItem.Text = "&Redo";
```

```
            //
            // toolStripSeparator6
            //
            this.toolStripSeparator6.Name = "toolStripSeparator6";
            this.toolStripSeparator6.Size = new System.Drawing.Size(164, 6);
            //
            // cutToolStripMenuItem
            //
            this.cutToolStripMenuItem.Image = ((System.Drawing.Image)(resources.GetObject("cutToolStripMenuItem.Image")));
            this.cutToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Black;
            this.cutToolStripMenuItem.Name = "cutToolStripMenuItem";
            this.cutToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.X)));
            this.cutToolStripMenuItem.Size = new System.Drawing.Size(167, 22);
            this.cutToolStripMenuItem.Text = "Cu&t";
            this.cutToolStripMenuItem.Click += new System.EventHandler(this.CutToolStripMenuItem_Click);
            //
            // copyToolStripMenuItem
            //
            this.copyToolStripMenuItem.Image = ((System.Drawing.Image)(resources.GetObject("copyToolStripMenuItem.Image")));
            this.copyToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Black;
            this.copyToolStripMenuItem.Name = "copyToolStripMenuItem";
            this.copyToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.C)));
            this.copyToolStripMenuItem.Size = new System.Drawing.Size(167, 22);
            this.copyToolStripMenuItem.Text = "&Copy";
            this.copyToolStripMenuItem.Click += new System.EventHandler(this.CopyToolStripMenuItem_Click);
            //
            // pasteToolStripMenuItem
            //
            this.pasteToolStripMenuItem.Image = ((System.Drawing.Image)(resources.GetObject("pasteToolStripMenuItem.Image")));
            this.pasteToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Black;
            this.pasteToolStripMenuItem.Name = "pasteToolStripMenuItem";
            this.pasteToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.V)));
            this.pasteToolStripMenuItem.Size = new System.Drawing.Size(167, 22);
            this.pasteToolStripMenuItem.Text = "&Paste";
            this.pasteToolStripMenuItem.Click += new System.EventHandler(this.PasteToolStripMenuItem_Click);
            //
            // toolStripSeparator7
            //
            this.toolStripSeparator7.Name = "toolStripSeparator7";
            this.toolStripSeparator7.Size = new System.Drawing.Size(164, 6);
            //
            // selectAllToolStripMenuItem
            //
            this.selectAllToolStripMenuItem.Name = "selectAllToolStripMenuItem";
            this.selectAllToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)((System.Windows.Forms.Keys.Control |
System.Windows.Forms.Keys.A)));
            this.selectAllToolStripMenuItem.Size = new System.Drawing.Size(167, 22);
            this.selectAllToolStripMenuItem.Text = "Select &All";
            //
            // viewMenu
            //
            this.viewMenu.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {
            this.toolBarToolStripMenuItem,
            this.statusBarToolStripMenuItem});
            this.viewMenu.Name = "viewMenu";
            this.viewMenu.Size = new System.Drawing.Size(41, 20);
            this.viewMenu.Text = "&View";
            //
            // toolBarToolStripMenuItem
            //
            this.toolBarToolStripMenuItem.CheckOnClick = true;
            this.toolBarToolStripMenuItem.Name = "toolBarToolStripMenuItem";
            this.toolBarToolStripMenuItem.Size = new System.Drawing.Size(135, 22);
            this.toolBarToolStripMenuItem.Text = "&Toolbar";
            this.toolBarToolStripMenuItem.Click += new System.EventHandler(this.ToolBarToolStripMenuItem_Click);
            //
            // statusBarToolStripMenuItem
            //
            this.statusBarToolStripMenuItem.Checked = true;
            this.statusBarToolStripMenuItem.CheckOnClick = true;
            this.statusBarToolStripMenuItem.CheckState = System.Windows.Forms.CheckState.Checked;
            this.statusBarToolStripMenuItem.Name = "statusBarToolStripMenuItem";
            this.statusBarToolStripMenuItem.Size = new System.Drawing.Size(135, 22);
            this.statusBarToolStripMenuItem.Text = "&Status Bar";
            this.statusBarToolStripMenuItem.Click += new System.EventHandler(this.StatusBarToolStripMenuItem_Click);
            //
            // windowsMenu
            //
            this.windowsMenu.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {
            this.cascadeToolStripMenuItem,
            this.tileVerticalToolStripMenuItem,
            this.tileHorizontalToolStripMenuItem,
            this.closeAllToolStripMenuItem,
            this.arrangeIconsToolStripMenuItem});
            this.windowsMenu.Name = "windowsMenu";
            this.windowsMenu.Size = new System.Drawing.Size(62, 20);
            this.windowsMenu.Text = "&Windows";
            //
            // cascadeToolStripMenuItem
            //
            this.cascadeToolStripMenuItem.Name = "cascadeToolStripMenuItem";
            this.cascadeToolStripMenuItem.Size = new System.Drawing.Size(153, 22);
            this.cascadeToolStripMenuItem.Text = "&Cascade";
            this.cascadeToolStripMenuItem.Click += new System.EventHandler(this.CascadeToolStripMenuItem_Click);
            //
            // tileVerticalToolStripMenuItem
            //
            this.tileVerticalToolStripMenuItem.Name = "tileVerticalToolStripMenuItem";
            this.tileVerticalToolStripMenuItem.Size = new System.Drawing.Size(153, 22);
            this.tileVerticalToolStripMenuItem.Text = "Tile &Vertical";
            this.tileVerticalToolStripMenuItem.Click += new System.EventHandler(this.TileVerticalToolStripMenuItem_Click);
            //
            // tileHorizontalToolStripMenuItem
            //
            this.tileHorizontalToolStripMenuItem.Name = "tileHorizontalToolStripMenuItem";
            this.tileHorizontalToolStripMenuItem.Size = new System.Drawing.Size(153, 22);
            this.tileHorizontalToolStripMenuItem.Text = "Tile &Horizontal";
            this.tileHorizontalToolStripMenuItem.Click += new System.EventHandler(this.TileHorizontalToolStripMenuItem_Click);
            //
            // closeAllToolStripMenuItem
            //
            this.closeAllToolStripMenuItem.Name = "closeAllToolStripMenuItem";
            this.closeAllToolStripMenuItem.Size = new System.Drawing.Size(153, 22);
            this.closeAllToolStripMenuItem.Text = "C&lose All";
            this.closeAllToolStripMenuItem.Click += new System.EventHandler(this.CloseAllToolStripMenuItem_Click);
            //
            // arrangeIconsToolStripMenuItem
            //
            this.arrangeIconsToolStripMenuItem.Name = "arrangeIconsToolStripMenuItem";
```

```
            this.arrangeIconsToolStripMenuItem.Size = new System.Drawing.Size(153, 22);
            this.arrangeIconsToolStripMenuItem.Text = "&Arrange Icons";
            this.arrangeIconsToolStripMenuItem.Click += new System.EventHandler(this.ArrangeIconsToolStripMenuItem_Click);
            //
            // helpMenu
            //
            this.helpMenu.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {
            this.contentsToolStripMenuItem,
            this.indexToolStripMenuItem,
            this.toolStripSeparator8,
            this.aboutToolStripMenuItem});
            this.helpMenu.Name = "helpMenu";
            this.helpMenu.Size = new System.Drawing.Size(40, 20);
            this.helpMenu.Text = "&Help";
            //
            // contentsToolStripMenuItem
            //
            this.contentsToolStripMenuItem.Name = "contentsToolStripMenuItem";
            this.contentsToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)((System.Windows.Forms.Keys.Control |
System.Windows.Forms.Keys.F1)));
            this.contentsToolStripMenuItem.Size = new System.Drawing.Size(173, 22);
            this.contentsToolStripMenuItem.Text = "&Contents";
            //
            // indexToolStripMenuItem
            //
            this.indexToolStripMenuItem.Image = ((System.Drawing.Image)(resources.GetObject("indexToolStripMenuItem.Image")));
            this.indexToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Black;
            this.indexToolStripMenuItem.Name = "indexToolStripMenuItem";
            this.indexToolStripMenuItem.Size = new System.Drawing.Size(173, 22);
            this.indexToolStripMenuItem.Text = "&Index";
            //
            // toolStripSeparator8
            //
            this.toolStripSeparator8.Name = "toolStripSeparator8";
            this.toolStripSeparator8.Size = new System.Drawing.Size(170, 6);
            //
            // aboutToolStripMenuItem
            //
            this.aboutToolStripMenuItem.Name = "aboutToolStripMenuItem";
            this.aboutToolStripMenuItem.Size = new System.Drawing.Size(173, 22);
            this.aboutToolStripMenuItem.Text = "&About ... ...";
            this.aboutToolStripMenuItem.Click += new System.EventHandler(this.aboutToolStripMenuItem_Click);
            //
            // toolStrip
            //
            this.toolStrip.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
            this.newToolStripButton,
            this.openToolStripButton,
            this.saveToolStripButton,
            this.toolStripSeparator1,
            this.printToolStripButton,
            this.printPreviewToolStripButton,
            this.toolStripSeparator2,
            this.helpToolStripButton});
            this.toolStrip.Location = new System.Drawing.Point(0, 24);
            this.toolStrip.Name = "toolStrip";
            this.toolStrip.Size = new System.Drawing.Size(632, 25);
            this.toolStrip.TabIndex = 1;
            this.toolStrip.Text = "ToolStrip";
            //
            // newToolStripButton
            //
            this.newToolStripButton.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image;
            this.newToolStripButton.Image = ((System.Drawing.Image)(resources.GetObject("newToolStripButton.Image")));
            this.newToolStripButton.ImageTransparentColor = System.Drawing.Color.Black;
            this.newToolStripButton.Name = "newToolStripButton";
            this.newToolStripButton.Size = new System.Drawing.Size(23, 22);
            this.newToolStripButton.Text = "New";
            this.newToolStripButton.Click += new System.EventHandler(this.ShowNewForm);
            //
            // openToolStripButton
            //
            this.openToolStripButton.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image;
            this.openToolStripButton.Image = ((System.Drawing.Image)(resources.GetObject("openToolStripButton.Image")));
            this.openToolStripButton.ImageTransparentColor = System.Drawing.Color.Black;
            this.openToolStripButton.Name = "openToolStripButton";
            this.openToolStripButton.Size = new System.Drawing.Size(23, 22);
            this.openToolStripButton.Text = "Open";
            this.openToolStripButton.Click += new System.EventHandler(this.OpenFile);
            //
            // saveToolStripButton
            //
            this.saveToolStripButton.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image;
            this.saveToolStripButton.Image = ((System.Drawing.Image)(resources.GetObject("saveToolStripButton.Image")));
            this.saveToolStripButton.ImageTransparentColor = System.Drawing.Color.Black;
            this.saveToolStripButton.Name = "saveToolStripButton";
            this.saveToolStripButton.Size = new System.Drawing.Size(23, 22);
            this.saveToolStripButton.Text = "Save";
            //
            // toolStripSeparator1
            //
            this.toolStripSeparator1.Name = "toolStripSeparator1";
            this.toolStripSeparator1.Size = new System.Drawing.Size(6, 25);
            //
            // printToolStripButton
            //
            this.printToolStripButton.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image;
            this.printToolStripButton.Image = ((System.Drawing.Image)(resources.GetObject("printToolStripButton.Image")));
            this.printToolStripButton.ImageTransparentColor = System.Drawing.Color.Black;
            this.printToolStripButton.Name = "printToolStripButton";
            this.printToolStripButton.Size = new System.Drawing.Size(23, 22);
            this.printToolStripButton.Text = "Print";
            //
            // printPreviewToolStripButton
            //
            this.printPreviewToolStripButton.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image;
            this.printPreviewToolStripButton.Image = ((System.Drawing.Image)(resources.GetObject("printPreviewToolStripButton.Image")));
            this.printPreviewToolStripButton.ImageTransparentColor = System.Drawing.Color.Black;
            this.printPreviewToolStripButton.Name = "printPreviewToolStripButton";
            this.printPreviewToolStripButton.Size = new System.Drawing.Size(23, 22);
            this.printPreviewToolStripButton.Text = "Print Preview";
            //
            // toolStripSeparator2
            //
            this.toolStripSeparator2.Name = "toolStripSeparator2";
            this.toolStripSeparator2.Size = new System.Drawing.Size(6, 25);
            //
            // helpToolStripButton
            //
```

```
            this.helpToolStripButton.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image;
            this.helpToolStripButton.Image = ((System.Drawing.Image)(resources.GetObject("helpToolStripButton.Image")));
            this.helpToolStripButton.ImageTransparentColor = System.Drawing.Color.Black;
            this.helpToolStripButton.Name = "helpToolStripButton";
            this.helpToolStripButton.Size = new System.Drawing.Size(23, 22);
            this.helpToolStripButton.Text = "Help";
            //
            // statusStrip
            //
            this.statusStrip.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
            this.toolStripStatusLabel});
            this.statusStrip.Location = new System.Drawing.Point(0, 431);
            this.statusStrip.Name = "statusStrip";
            this.statusStrip.Size = new System.Drawing.Size(632, 22);
            this.statusStrip.TabIndex = 2;
            this.statusStrip.Text = "StatusStrip";
            //
            // toolStripStatusLabel
            //
            this.toolStripStatusLabel.Name = "toolStripStatusLabel";
            this.toolStripStatusLabel.Size = new System.Drawing.Size(38, 17);
            this.toolStripStatusLabel.Text = "Status";
            //
            // MDIParent1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(632, 453);
            this.Controls.Add(this.statusStrip);
            this.Controls.Add(this.toolStrip);
            this.Controls.Add(this.menuStrip);
            this.IsMdiContainer = true;
            this.MainMenuStrip = this.menuStrip;
            this.Name = "MDIParent1";
            this.Text = "Q-Enhanced Filter v2.0 Test Application";
            this.menuStrip.ResumeLayout(false);
            this.menuStrip.PerformLayout();
            this.toolStrip.ResumeLayout(false);
            this.toolStrip.PerformLayout();
            this.statusStrip.ResumeLayout(false);
            this.statusStrip.PerformLayout();
            this.ResumeLayout(false);
            this.PerformLayout();

        }
        #endregion


        private System.Windows.Forms.MenuStrip menuStrip;
        private System.Windows.Forms.ToolStrip toolStrip;
        private System.Windows.Forms.StatusStrip statusStrip;
        private System.Windows.Forms.ToolStripSeparator toolStripSeparator1;
        private System.Windows.Forms.ToolStripSeparator toolStripSeparator2;
        private System.Windows.Forms.ToolStripSeparator toolStripSeparator3;
        private System.Windows.Forms.ToolStripSeparator toolStripSeparator4;
        private System.Windows.Forms.ToolStripSeparator toolStripSeparator5;
        private System.Windows.Forms.ToolStripSeparator toolStripSeparator6;
        private System.Windows.Forms.ToolStripMenuItem printSetupToolStripMenuItem;
        private System.Windows.Forms.ToolStripSeparator toolStripSeparator7;
        private System.Windows.Forms.ToolStripSeparator toolStripSeparator8;
        private System.Windows.Forms.ToolStripStatusLabel toolStripStatusLabel;
        private System.Windows.Forms.ToolStripMenuItem aboutToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem tileHorizontalToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem fileMenu;
        private System.Windows.Forms.ToolStripMenuItem newToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem openToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem saveToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem saveAsToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem printToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem printPreviewToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem exitToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem editMenu;
        private System.Windows.Forms.ToolStripMenuItem undoToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem redoToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem cutToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem copyToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem pasteToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem selectAllToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem viewMenu;
        private System.Windows.Forms.ToolStripMenuItem toolBarToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem statusBarToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem windowsMenu;
        private System.Windows.Forms.ToolStripMenuItem cascadeToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem tileVerticalToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem closeAllToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem arrangeIconsToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem helpMenu;
        private System.Windows.Forms.ToolStripMenuItem contentsToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem indexToolStripMenuItem;
        private System.Windows.Forms.ToolStripButton newToolStripButton;
        private System.Windows.Forms.ToolStripButton openToolStripButton;
        private System.Windows.Forms.ToolStripButton saveToolStripButton;
        private System.Windows.Forms.ToolStripButton printToolStripButton;
        private System.Windows.Forms.ToolStripButton printPreviewToolStripButton;
        private System.Windows.Forms.ToolStripButton helpToolStripButton;
        private System.Windows.Forms.ToolTip toolTip;
    }
}
```

PORTCTRLS.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace QEFilterV2TestAppV2
{
    public class PortCtrls
    {
        private bool _OkToClose;

        public PortCtrls()
        {
            OkToClose = false;
        }

        public bool OkToClose
        {
            get { return _OkToClose; }
            set { _OkToClose = value; }
        }
    }
}
```

FRMPORTCTRLS.CS

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace QEFilterV2TestAppV2
{
    public partial class frmPortCtrls : Form
    {
        private PortCtrls _PortCtrls;
        private DebugText _DebugText;
        private CommunicationManager _ComManager;

        public frmPortCtrls(PortCtrls pc, DebugText dt, CommunicationManager cm)
        {
            InitializeComponent();
            this.Closing += new CancelEventHandler(frmPortCtrls_Closing);
            _PortCtrls = pc;
            _DebugText = dt;
            _ComManager = cm;

            InitalizePortControls();
        }

        private void frmPortCtrls_Closing(object sender, CancelEventArgs cArgs)
        {
            if (!_PortCtrls.OkToClose)
                cArgs.Cancel = true;
            else
                cArgs.Cancel = false;
        }

        private void InitalizePortControls()
        {
            cboPortName.Items.Clear();
            foreach (string port in _ComManager.PortList)
                cboPortName.Items.Add(port);
            if (cboPortName.Items.Count > 0)
                cboPortName.SelectedIndex = 0;

            cboPortParity.Items.Clear();
            foreach (string parity in _ComManager.ParityOptions)
                cboPortParity.Items.Add(parity);
            if (cboPortParity.Items.Count > 0)
                cboPortParity.SelectedIndex = 0;

            cboPortStop.Items.Clear();
            foreach (string stopBits in _ComManager.StopBitOptions)
                cboPortStop.Items.Add(stopBits);
            if (cboPortStop.Items.Count > 0)
                cboPortStop.SelectedIndex = 0;

            cboPortBaud.Items.Clear();
            cboPortBaud.Items.AddRange(_ComManager.BaudRateOptions);
            if (cboPortBaud.Items.Count > 0)
                cboPortBaud.SelectedIndex = cboPortBaud.FindString(CommunicationManager.BAUDRATE_DEFAULT, 0);

            cboPortData.Items.Clear();
            cboPortData.Items.AddRange(_ComManager.DataBitOptions);
            if (cboPortData.Items.Count > 0)
                cboPortData.SelectedIndex = cboPortData.FindString(CommunicationManager.DATABITS_DEFAULT, 0);
        }

        private void btnPortDefault_Click(object sender, EventArgs e)
        {
            InitalizePortControls();
        }

        private void btnPortOpen_Click(object sender, EventArgs e)
        {
            OpenSelectedPort();
        }

        private void btnPortClose_Click(object sender, EventArgs e)
        {
            CloseSelectedPort();
        }

        private void OpenSelectedPort()
        {
            bool err = false;
```

```
            string errMsg = string.Empty;

            if (cboPortName.Text.Equals(string.Empty))
                err = true;
            else if (cboPortBaud.Text.Equals(string.Empty))
                err = true;
            else if (cboPortParity.Text.Equals(string.Empty))
                err = true;
            else if (cboPortData.Text.Equals(string.Empty))
                err = true;
            else if (cboPortStop.Text.Equals(string.Empty))
                err = true;

            if (err)
            {
                errMsg = "Missing information please verify that all port configuration options have been selected";
                _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Error, errMsg);
                MessageBox.Show(errMsg);
                return;
            }

            if (!_ComManager.PortList.Contains(cboPortName.Text))
                err = true;
            else if (!_ComManager.BaudRateOptions.Contains(cboPortBaud.Text))
                err = true;
            else if (!_ComManager.ParityOptions.Contains(cboPortParity.Text))
                err = true;
            else if (!_ComManager.DataBitOptions.Contains(cboPortData.Text))
                err = true;
            else if (!_ComManager.StopBitOptions.Contains(cboPortStop.Text))
                err = true;

            if (err)
            {
                errMsg = "Invalid options please verify that all port configuration options are correct";
                _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Error, errMsg);
                MessageBox.Show(errMsg);
                return;
            }

            if (_ComManager.isOpen)
            {
                string msg = _ComManager.PortName + " is currently open, do you wish to close it\nand open " + cboPortName.Text + "?";
                DialogResult result = MessageBox.Show(msg, "COM Port Already Open", MessageBoxButtons.YesNo);

                if (result == System.Windows.Forms.DialogResult.Yes)
                    _ComManager.ClosePort();
                else
                    return;
            }

            _ComManager.PortName = cboPortName.Text;
            _ComManager.BaudRate = cboPortBaud.Text;
            _ComManager.Parity = cboPortParity.Text;
            _ComManager.DataBits = cboPortData.Text;
            _ComManager.StopBits = cboPortStop.Text;
            _ComManager.OpenPort();
        }

        private void CloseSelectedPort()
        {
            if (_ComManager.isOpen)
                _ComManager.ClosePort();
            else
                MessageBox.Show("COM Port is not currently open.");
        }

        private void cboPortName_Click(object sender, EventArgs e)
        {
            cboPortName.Items.Clear();
            foreach (string port in _ComManager.PortList)
                cboPortName.Items.Add(port);
            if (cboPortName.Items.Count > 0)
                cboPortName.SelectedIndex = 0;
        }
    }
}
```

FRMPORTCTRLS.DESIGNER.CS

```
namespace QEFilterV2TestAppV2
{
    partial class frmPortCtrls
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.groupBox2 = new System.Windows.Forms.GroupBox();
            this.btnPortDefault = new System.Windows.Forms.Button();
            this.btnPortClose = new System.Windows.Forms.Button();
            this.btnPortOpen = new System.Windows.Forms.Button();
            this.cboPortData = new System.Windows.Forms.ComboBox();
            this.label5 = new System.Windows.Forms.Label();
```

```
this.cboPortStop = new System.Windows.Forms.ComboBox();
this.label4 = new System.Windows.Forms.Label();
this.cboPortParity = new System.Windows.Forms.ComboBox();
this.label3 = new System.Windows.Forms.Label();
this.cboPortBaud = new System.Windows.Forms.ComboBox();
this.label2 = new System.Windows.Forms.Label();
this.cboPortName = new System.Windows.Forms.ComboBox();
this.label1 = new System.Windows.Forms.Label();
this.groupBox2.SuspendLayout();
this.SuspendLayout();
//
// groupBox2
//
this.groupBox2.Controls.Add(this.btnPortDefault);
this.groupBox2.Controls.Add(this.btnPortClose);
this.groupBox2.Controls.Add(this.btnPortOpen);
this.groupBox2.Controls.Add(this.cboPortData);
this.groupBox2.Controls.Add(this.label5);
this.groupBox2.Controls.Add(this.cboPortStop);
this.groupBox2.Controls.Add(this.label4);
this.groupBox2.Controls.Add(this.cboPortParity);
this.groupBox2.Controls.Add(this.label3);
this.groupBox2.Controls.Add(this.cboPortBaud);
this.groupBox2.Controls.Add(this.label2);
this.groupBox2.Controls.Add(this.cboPortName);
this.groupBox2.Controls.Add(this.label1);
this.groupBox2.Font = new System.Drawing.Font("Cambria", 12F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.groupBox2.Location = new System.Drawing.Point(12, 12);
this.groupBox2.Name = "groupBox2";
this.groupBox2.Size = new System.Drawing.Size(148, 315);
this.groupBox2.TabIndex = 2;
this.groupBox2.TabStop = false;
this.groupBox2.Text = "Port Controls";
//
// btnPortDefault
//
this.btnPortDefault.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.btnPortDefault.Location = new System.Drawing.Point(13, 219);
this.btnPortDefault.Name = "btnPortDefault";
this.btnPortDefault.Size = new System.Drawing.Size(121, 23);
this.btnPortDefault.TabIndex = 9;
this.btnPortDefault.Text = "Restore Defaults";
this.btnPortDefault.UseVisualStyleBackColor = true;
this.btnPortDefault.Click += new System.EventHandler(this.btnPortDefault_Click);
//
// btnPortClose
//
this.btnPortClose.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.btnPortClose.Location = new System.Drawing.Point(13, 277);
this.btnPortClose.Name = "btnPortClose";
this.btnPortClose.Size = new System.Drawing.Size(121, 23);
this.btnPortClose.TabIndex = 11;
this.btnPortClose.Text = "Close Port";
this.btnPortClose.UseVisualStyleBackColor = true;
this.btnPortClose.Click += new System.EventHandler(this.btnPortClose_Click);
//
// btnPortOpen
//
this.btnPortOpen.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.btnPortOpen.Location = new System.Drawing.Point(13, 248);
this.btnPortOpen.Name = "btnPortOpen";
this.btnPortOpen.Size = new System.Drawing.Size(121, 23);
this.btnPortOpen.TabIndex = 10;
this.btnPortOpen.Text = "Open Port";
this.btnPortOpen.UseVisualStyleBackColor = true;
this.btnPortOpen.Click += new System.EventHandler(this.btnPortOpen_Click);
//
// cboPortData
//
this.cboPortData.DropDownStyle = System.Windows.Forms.ComboBoxStyle.DropDownList;
this.cboPortData.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.cboPortData.FormattingEnabled = true;
this.cboPortData.Location = new System.Drawing.Point(13, 192);
this.cboPortData.Name = "cboPortData";
this.cboPortData.Size = new System.Drawing.Size(121, 21);
this.cboPortData.TabIndex = 8;
//
// label5
//
this.label5.AutoSize = true;
this.label5.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.label5.Location = new System.Drawing.Point(13, 176);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(50, 13);
this.label5.TabIndex = 0;
this.label5.Text = "Data Bits";
//
// cboPortStop
//
this.cboPortStop.DropDownStyle = System.Windows.Forms.ComboBoxStyle.DropDownList;
this.cboPortStop.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.cboPortStop.FormattingEnabled = true;
this.cboPortStop.Location = new System.Drawing.Point(13, 154);
this.cboPortStop.Name = "cboPortStop";
this.cboPortStop.Size = new System.Drawing.Size(121, 21);
this.cboPortStop.TabIndex = 7;
//
// label4
//
this.label4.AutoSize = true;
this.label4.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.label4.Location = new System.Drawing.Point(13, 138);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(49, 13);
this.label4.TabIndex = 0;
this.label4.Text = "Stop Bits";
//
// cboPortParity
//
this.cboPortParity.DropDownStyle = System.Windows.Forms.ComboBoxStyle.DropDownList;
this.cboPortParity.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.cboPortParity.FormattingEnabled = true;
this.cboPortParity.Location = new System.Drawing.Point(13, 114);
this.cboPortParity.Name = "cboPortParity";
this.cboPortParity.Size = new System.Drawing.Size(121, 21);
this.cboPortParity.TabIndex = 6;
//
```

```
            // label3
            //
            this.label3.AutoSize = true;
            this.label3.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.label3.Location = new System.Drawing.Point(13, 98);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(33, 13);
            this.label3.TabIndex = 0;
            this.label3.Text = "Parity";
            //
            // cboPortBaud
            //
            this.cboPortBaud.DropDownStyle = System.Windows.Forms.ComboBoxStyle.DropDownList;
            this.cboPortBaud.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.cboPortBaud.FormattingEnabled = true;
            this.cboPortBaud.Location = new System.Drawing.Point(13, 77);
            this.cboPortBaud.Name = "cboPortBaud";
            this.cboPortBaud.Size = new System.Drawing.Size(121, 21);
            this.cboPortBaud.TabIndex = 5;
            //
            // label2
            //
            this.label2.AutoSize = true;
            this.label2.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.label2.Location = new System.Drawing.Point(13, 61);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(58, 13);
            this.label2.TabIndex = 0;
            this.label2.Text = "Baud Rate";
            //
            // cboPortName
            //
            this.cboPortName.DropDownStyle = System.Windows.Forms.ComboBoxStyle.DropDownList;
            this.cboPortName.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.cboPortName.FormattingEnabled = true;
            this.cboPortName.Location = new System.Drawing.Point(13, 37);
            this.cboPortName.Name = "cboPortName";
            this.cboPortName.Size = new System.Drawing.Size(121, 21);
            this.cboPortName.TabIndex = 4;
            this.cboPortName.Click += new System.EventHandler(this.cboPortName_Click);
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.label1.Location = new System.Drawing.Point(13, 21);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(26, 13);
            this.label1.TabIndex = 0;
            this.label1.Text = "Port";
            //
            // frmPortCtrls
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(172, 339);
            this.Controls.Add(this.groupBox2);
            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
            this.MaximizeBox = false;
            this.Name = "frmPortCtrls";
            this.Text = "Port Ctrls";
            this.groupBox2.ResumeLayout(false);
            this.groupBox2.PerformLayout();
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.Button btnPortDefault;
        private System.Windows.Forms.Button btnPortClose;
        private System.Windows.Forms.Button btnPortOpen;
        private System.Windows.Forms.ComboBox cboPortData;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.ComboBox cboPortStop;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.ComboBox cboPortParity;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.ComboBox cboPortBaud;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.ComboBox cboPortName;
        private System.Windows.Forms.Label label1;
    }
}
```

COMMUNICATIONMANAGER.CS

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;

namespace QEFilterV2TestAppV2
{
    public class CommunicationManager
    {
        private DebugText _DebugText;
        private QTuneDebug _QTuneDebug;
        private FTuneDebug _FTuneDebug;
        private SerialPort _ComPort;

        private string[] _PortList;
        private string _PortName;
        private string _BaudRate;
        private string _Parity;
        private string _DataBits;
        private string _StopBits;
```

```
private string _RxDataFirstHalf;
private string _RxDataLastHalf;
private string[] BAUDRATEOPTIONS = { "300", "600", "1200", "2400", "4800", "9600", "14400", "28800", "36000", "115000" };
private string[] DATABITOPTIONS = { "5", "6", "7", "8" };

public const string SEPERATOR = "------------------------------------------------------------------------------------\r\n";
public const string BAUDRATE_DEFAULT   = "9600";
public const string DATABITS_DEFAULT   = "8";

public CommunicationManager(DebugText dt)
{
    _DebugText = dt;
    _ComPort = new SerialPort();
    _ComPort.DataReceived += new SerialDataReceivedEventHandler(_ComPort_DataReceived);
    _ComPort.Handshake = Handshake.RequestToSend;
    _PortList = null;
    _PortName = string.Empty;
    _BaudRate = string.Empty;
    _Parity = string.Empty;
    _DataBits = string.Empty;
    _StopBits = string.Empty;
    _RxDataFirstHalf = string.Empty;
    _RxDataLastHalf = string.Empty;
}

public QTuneDebug QTuneDebug
{
    get { return _QTuneDebug; }
    set { _QTuneDebug = value; }
}

public FTuneDebug FTuneDebug
{
    get { return _FTuneDebug; }
    set { _FTuneDebug = value; }
}

public SerialPort ComPort
{
    get { return _ComPort; }
}

public string[] PortList
{
    get
    {
        _PortList = SerialPort.GetPortNames();
        return _PortList;
    }
}

public string PortName
{
    get { return _PortName; }
    set { _PortName = value; }
}

public string BaudRate
{
    get { return _BaudRate; }
    set { _BaudRate = value; }
}

public string Parity
{
    get { return _Parity; }
    set { _Parity = value; }
}

public string DataBits
{
    get { return _DataBits; }
    set { _DataBits = value; }
}

public string StopBits
{
    get { return _StopBits; }
    set { _StopBits = value; }
}

public string[] ParityOptions
{
    get { return Enum.GetNames(typeof(Parity)); }
}

public string[] StopBitOptions
{
    get
    {
        string[] opts;
        List<string> tmp = (Enum.GetNames(typeof(StopBits))).ToList();
        tmp.Remove("None");
        tmp.Remove("OnePointFive");
        opts = tmp.ToArray();
        return opts;
    }
}

public string[] BaudRateOptions
{
    get { return BAUDRATEOPTIONS;  }
}

public string[] DataBitOptions
{
    get { return DATABITOPTIONS; }
}

public bool OpenPort()
{
    try
    {
        _ComPort.PortName = _PortName;
        _ComPort.BaudRate = int.Parse(_BaudRate);
        _ComPort.Parity = (Parity)(Enum.Parse(typeof(Parity), _Parity));
        _ComPort.DataBits = int.Parse(_DataBits);
        _ComPort.StopBits = (StopBits)(Enum.Parse(typeof(StopBits), _StopBits));
        _ComPort.Open();
```

```
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, SEPERATOR);
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, this.OpenMsg);
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, SEPERATOR);
            return true;
        }
        catch (Exception ex)
        {
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Error, ex.Message.ToString());
            return false;
        }
    }

    public bool isOpen
    {
        get { return _ComPort.IsOpen; }
    }

    public string OpenMsg
    {
        get
        {
            string msg = _PortName + " opened at " + DateTime.Now;
            msg += "\r\nBaudRate: " + _BaudRate;
            msg += "\r\nParity: " + _Parity;
            msg += "\r\nStopBits: " + _StopBits;
            msg += "\r\nDataBits: " + _DataBits + "\r\n";
            return msg;
        }
    }

    public bool ClosePort()
    {
        try
        {
            _ComPort.Close();
            _PortName = string.Empty;
            _BaudRate = string.Empty;
            _Parity = string.Empty;
            _DataBits = string.Empty;
            _StopBits = string.Empty;
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, SEPERATOR);
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, this.CloseMsg);
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, SEPERATOR);
            return true;
        }
        catch (Exception ex)
        {
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Error, ex.Message.ToString());
            return false;
        }
    }

    public string CloseMsg
    {
        get { return ("" + _ComPort.PortName + " closed at " + DateTime.Now + "\r\n"); }
    }

    public bool WriteData(string msg)
    {
        string errMsg = string.Empty;

        try
        {
            if (!_ComPort.IsOpen)
                errMsg += "Selected COM Port is not open.\n";
            if (msg == string.Empty)
                errMsg += "Nothing to be sent to the selected COM Port\n";

            if (errMsg != string.Empty)
            {
                MessageBox.Show(errMsg);
                return false;
            }
            _ComPort.Write(msg);
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Outgoing, "[OUT @ " + DateTime.Now + "] " + msg + "\r\n");
            return true;
        }
        catch (Exception ex)
        {
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Error, ex.Message.ToString());
            return false;
        }
    }

    private void _ComPort_DataReceived(object sender, SerialDataReceivedEventArgs e)
    {
        string msg = string.Empty;
        try
        {
            msg = CheckIfDebugMessage(_ComPort.ReadExisting().ToString());
            if (msg != string.Empty)
                _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Incoming, msg);
        }
        catch (Exception ex)
        {
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Error, ex.Message.ToString());
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Error, ex.StackTrace.ToString());
        }
    }

    private byte[] Hex2Byte(string msg)
    {
        byte[] comBuffer;
        msg = msg.Replace(" ", "");
        comBuffer = new byte[msg.Length / 2];
        for (int i = 0; i < msg.Length; i += 2)
            comBuffer[i / 2] = (byte)Convert.ToByte(msg.Substring(i, 2), 16);
        return comBuffer;
    }

    private string CheckIfDebugMessage(string RxStr)
    {
        string msg = RxStr;
        string MsgRemaining = string.Empty;

        if (msg.Contains('^') && msg.Contains('|'))
        {
            char startChar = '^', stopChar = '|';
            int startCharIndex, stopCharIndex, len;

            startCharIndex = msg.IndexOf(startChar)+1;
```

```
                stopCharIndex = msg.IndexOf(stopChar)-1;
                len = stopCharIndex - startCharIndex + 1;
                msg = msg.Substring(startCharIndex, len);

                stopCharIndex++;
                MsgRemaining = RxStr.Substring(0, startCharIndex - 1).ToString();
                if (stopCharIndex < RxStr.Length - 1)
                {
                    len = RxStr.Length - stopCharIndex - 1;
                    MsgRemaining += RxStr.Substring(stopCharIndex + 1, len);
                }

                HandleDebugData(msg);
                return MsgRemaining;
            }
            else if (msg.Contains('^') && !msg.Contains('|'))
            {
                int startCharIndex, len;
                startCharIndex = msg.IndexOf('^') + 1;
                len = msg.Length - startCharIndex;
                _RxDataFirstHalf = msg.Substring(startCharIndex, len);
                MsgRemaining = RxStr.Substring(0, startCharIndex - 1).ToString();
                return MsgRemaining;
            }
            else if (!msg.Contains('^') && msg.Contains('|'))
            {
                int stopCharIndex, len;
                stopCharIndex = msg.IndexOf('|');
                len = stopCharIndex;
                _RxDataLastHalf = msg.Substring(0, len);
                stopCharIndex++;
                MsgRemaining = RxStr.Substring(stopCharIndex, RxStr.Length - stopCharIndex).ToString();

                HandleDebugData(_RxDataFirstHalf + _RxDataLastHalf);
                return MsgRemaining;
            }
            else
                return RxStr;
        }

        private void HandleDebugData(string DebugStr)
        {
            try
            {
                string[] debugStrs = DebugStr.Split('&');
                _RxDataFirstHalf = string.Empty;
                _RxDataLastHalf = string.Empty;

                if (debugStrs.Count() == 4)
                {
                    int FrontEndAD = int.Parse(debugStrs.ElementAt(0));
                    int FrontEndFcnt = int.Parse(debugStrs.ElementAt(1));
                    int BackEndAD = int.Parse(debugStrs.ElementAt(2));
                    int BackEndFcnt = int.Parse(debugStrs.ElementAt(3));

                    if (_QTuneDebug.isRunning)
                    {
                        int VqCur = -1;
                        if (QTuneDebug.SelectedPole.Equals((int)QTuneDebug.PoleSelection.FrontEnd))
                            VqCur = FrontEndAD;
                        else if (QTuneDebug.SelectedPole.Equals((int)QTuneDebug.PoleSelection.BackEnd))
                            VqCur = BackEndAD;

                        if (VqCur != -1)
                            _QTuneDebug.UpdateSweepInfo(_QTuneDebug.SweepQTune.NqCur, VqCur);
                    }
                    if (_FTuneDebug.isRunning)
                    {
                        int FCnt = -1;
                        if (FTuneDebug.SelectedPole.Equals((int)FTuneDebug.PoleSelection.FrontEnd))
                            FCnt = FrontEndFcnt;
                        else if (FTuneDebug.SelectedPole.Equals((int)FTuneDebug.PoleSelection.BackEnd))
                            FCnt = BackEndFcnt;

                        if (FCnt != -1)
                            _FTuneDebug.UpdateSweepInfo(_FTuneDebug.SweepFTune.FCur, FCnt);


                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message + "\n" + ex.StackTrace + "\nDebugStr: " + DebugStr);
            }
        }
    }
}
```

DEBUGTEXT.CS

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace QEFilterV2TestAppV2
{
    public class DebugText
    {
        private bool _OkToClose;
        private RichTextBox _DebugRTB, _StatusRTB;

        private List<RichTextBox> MessageTarget = new List<RichTextBox>();
        private Color[] MessageColor = { Color.Blue, Color.Green, Color.Black, Color.Orange, Color.Red };
        private FontStyle[] MessageStyle = { FontStyle.Regular, FontStyle.Regular, FontStyle.Regular, FontStyle.Bold, FontStyle.Bold };

        public enum MessageType { Incoming, Outgoing, Normal, Warning, Error };
        public enum MessageClass { Debug, Status };

        public DebugText()
```

```
        {
            OkToClose = false;
            _DebugRTB = null;
        }

        public bool OkToClose
        {
            get { return _OkToClose; }
            set { _OkToClose = value; }
        }

        public void setDebugRTB(RichTextBox rtb)
        {
            _DebugRTB = rtb;
        }

        public void setStatusRTB(RichTextBox rtb)
        {
            _StatusRTB = rtb;
        }

        [STAThread]
        public void DisplayDebugData(MessageClass c, MessageType type, string msg)
        {
            MessageTarget.Clear();
            MessageTarget.Add(_DebugRTB);
            MessageTarget.Add(_StatusRTB);
            RichTextBox rtb = MessageTarget.ElementAt((int)c);

            if (c == MessageClass.Status)
                rtb.Clear();

            rtb.Invoke(new EventHandler(delegate
            {
                rtb.SelectedText = string.Empty;
                rtb.SelectionFont = new Font(rtb.SelectionFont, MessageStyle[(int)type]);
                rtb.SelectionColor = MessageColor[(int)type];
                rtb.AppendText(msg);
                rtb.ScrollToCaret();
            }));
        }

    }
}
```

FRMDEBUGTXT.CS

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace QEFilterV2TestAppV2
{
    public partial class frmDebugTxt : Form
    {
        private DebugText _DebugText;
        private SaveFileDialog _DialogSaveDebug;

        public frmDebugTxt(DebugText dt)
        {
            InitializeComponent();
            this.Closing += new CancelEventHandler(frmDebugText_Closing);
            _DebugText = dt;
            _DebugText.setDebugRTB(rtbDebug);
            _DebugText.setStatusRTB(rtbStatus);


            _DialogSaveDebug = new SaveFileDialog();
            _DialogSaveDebug.DefaultExt = "rtf";
            _DialogSaveDebug.Filter = "Rich Text File (*.rtf)|*.rtf|Text File (*.txt)|*.txt";
            _DialogSaveDebug.AddExtension = true;
            _DialogSaveDebug.RestoreDirectory = true;
            _DialogSaveDebug.Title = "Save Debug Information as...";
            _DialogSaveDebug.InitialDirectory = @"C:/";
            _DialogSaveDebug.FileName = "";
        }

        private void frmDebugText_Closing(object sender, CancelEventArgs cArgs)
        {
            if (!_DebugText.OkToClose)
                cArgs.Cancel = true;
            else
                cArgs.Cancel = false;
        }

        private void btnDebugSave_Click(object sender, EventArgs e)
        {
            saveDebug();
        }

        private void btnDebugClear_Click(object sender, EventArgs e)
        {
            DialogResult result = MessageBox.Show("Are you sure you want to clear the debug information?", "Clear Debug Information",
MessageBoxButtons.YesNo);

            if (result == System.Windows.Forms.DialogResult.Yes)
            {
                rtbDebug.Clear();
                _DebugText.DisplayDebugData(DebugText.MessageClass.Status, DebugText.MessageType.Normal, "Debug Information Cleared");
            }
        }

        private void saveDebug()
        {
            try
            {
                string fileName = string.Empty;
                int dirIndex = 0;
                _DialogSaveDebug.ShowDialog();
```

```
                if (_DialogSaveDebug.FileName != "")
                {
                    dirIndex = _DialogSaveDebug.FileName.LastIndexOf("\\") + 1;
                    fileName = _DialogSaveDebug.FileName.Substring(dirIndex, _DialogSaveDebug.FileName.Length - dirIndex);
                    switch (_DialogSaveDebug.FilterIndex)
                    {
                        case 1:
                            rtbDebug.SaveFile(_DialogSaveDebug.FileName);
                            break;
                        case 2:
                            File.WriteAllText(@_DialogSaveDebug.FileName, rtbDebug.Text);
                            break;
                        default:
                            rtbDebug.SaveFile(_DialogSaveDebug.FileName);
                            break;
                    }
                    _DebugText.DisplayDebugData(DebugText.MessageClass.Status, DebugText.MessageType.Normal, fileName + " saved successfully.");
                }
            }
            catch (Exception ex)
            {
                _DebugText.DisplayDebugData(DebugText.MessageClass.Status, DebugText.MessageType.Error, ex.Message.ToString());
            }
        }

        private void btnDebugCopy_Click(object sender, EventArgs e)
        {
            rtbDebug.SelectAll();
            rtbDebug.Copy();
        }

        private void frmDebugTxt_Resize(object sender, EventArgs e)
        {
            int btnY;

            if (this.Width < 407)
                this.Width = 407;
            if (this.Height < 528)
                this.Height = 528;

            //this.groupBox1.Location = new System.Drawing.Point(12, 12);
            this.groupBox1.Width = this.Width - 31;
            this.groupBox1.Height = this.Height - 58;
            this.rtbDebug.Width = this.Width - 43;
            this.rtbDebug.Height = this.Height - 155;
            this.rtbStatus.Location = new System.Drawing.Point(6, 35 + this.rtbDebug.Height);
            this.rtbStatus.Width = this.rtbDebug.Width;

            btnY = 41 + this.rtbDebug.Height + this.rtbStatus.Height;
            this.btnDebugClear.Location = new System.Drawing.Point(6, btnY);
            this.btnDebugCopy.Location = new System.Drawing.Point(12 + this.groupBox1.Width / 2 - this.btnDebugCopy.Width / 2, btnY);
            this.btnDebugSave.Location = new System.Drawing.Point(this.Width - 112, btnY);

        }
    }
}
```

FRMDEBUGTXT.DESIGNER.CS

```
namespace QEFilterV2TestAppV2
{
    partial class frmDebugTxt
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.groupBox1 = new System.Windows.Forms.GroupBox();
            this.btnDebugCopy = new System.Windows.Forms.Button();
            this.btnDebugClear = new System.Windows.Forms.Button();
            this.btnDebugSave = new System.Windows.Forms.Button();
            this.rtbStatus = new System.Windows.Forms.RichTextBox();
            this.rtbDebug = new System.Windows.Forms.RichTextBox();
            this.groupBox1.SuspendLayout();
            this.SuspendLayout();
            //
            // groupBox1
            //
            this.groupBox1.Controls.Add(this.btnDebugCopy);
            this.groupBox1.Controls.Add(this.btnDebugClear);
            this.groupBox1.Controls.Add(this.btnDebugSave);
            this.groupBox1.Controls.Add(this.rtbStatus);
            this.groupBox1.Controls.Add(this.rtbDebug);
            this.groupBox1.Font = new System.Drawing.Font("Cambria", 12F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.groupBox1.Location = new System.Drawing.Point(12, 12);
            this.groupBox1.Name = "groupBox1";
            this.groupBox1.Size = new System.Drawing.Size(376, 470);
            this.groupBox1.TabIndex = 2;
            this.groupBox1.TabStop = false;
            this.groupBox1.Text = "Debug Information";
            //
            // btnDebugCopy
            //
            this.btnDebugCopy.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
```

```
            this.btnDebugCopy.Location = new System.Drawing.Point(151, 437);
            this.btnDebugCopy.Name = "btnDebugCopy";
            this.btnDebugCopy.Size = new System.Drawing.Size(75, 23);
            this.btnDebugCopy.TabIndex = 4;
            this.btnDebugCopy.Text = "Copy";
            this.btnDebugCopy.UseVisualStyleBackColor = true;
            this.btnDebugCopy.Click += new System.EventHandler(this.btnDebugCopy_Click);
            //
            // btnDebugClear
            //
            this.btnDebugClear.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.btnDebugClear.Location = new System.Drawing.Point(6, 437);
            this.btnDebugClear.Name = "btnDebugClear";
            this.btnDebugClear.Size = new System.Drawing.Size(75, 23);
            this.btnDebugClear.TabIndex = 3;
            this.btnDebugClear.Text = "Clear";
            this.btnDebugClear.UseVisualStyleBackColor = true;
            this.btnDebugClear.Click += new System.EventHandler(this.btnDebugClear_Click);
            //
            // btnDebugSave
            //
            this.btnDebugSave.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.btnDebugSave.Location = new System.Drawing.Point(295, 437);
            this.btnDebugSave.Name = "btnDebugSave";
            this.btnDebugSave.Size = new System.Drawing.Size(75, 23);
            this.btnDebugSave.TabIndex = 2;
            this.btnDebugSave.Text = "Save";
            this.btnDebugSave.UseVisualStyleBackColor = true;
            this.btnDebugSave.Click += new System.EventHandler(this.btnDebugSave_Click);
            //
            // rtbStatus
            //
            this.rtbStatus.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.rtbStatus.Location = new System.Drawing.Point(6, 408);
            this.rtbStatus.Multiline = false;
            this.rtbStatus.Name = "rtbStatus";
            this.rtbStatus.Size = new System.Drawing.Size(364, 23);
            this.rtbStatus.TabIndex = 1;
            this.rtbStatus.Text = "";
            //
            // rtbDebug
            //
            this.rtbDebug.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.rtbDebug.Location = new System.Drawing.Point(6, 29);
            this.rtbDebug.Name = "rtbDebug";
            this.rtbDebug.Size = new System.Drawing.Size(364, 373);
            this.rtbDebug.TabIndex = 0;
            this.rtbDebug.Text = "";
            //
            // frmDebugTxt
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(401, 496);
            this.Controls.Add(this.groupBox1);
            this.Name = "frmDebugTxt";
            this.Text = "Debug Information";
            this.Resize += new System.EventHandler(this.frmDebugTxt_Resize);
            this.groupBox1.ResumeLayout(false);
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.Button btnDebugClear;
        private System.Windows.Forms.Button btnDebugSave;
        private System.Windows.Forms.RichTextBox rtbStatus;
        private System.Windows.Forms.RichTextBox rtbDebug;
        private System.Windows.Forms.Button btnDebugCopy;

    }
}
```

FILTERCONTROLS.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace QEFilterV2TestAppV2
{
    public class FilterControls
    {
        private CommunicationManager _ComManager;
        private DebugText _DebugText;
        private FrontEndCtrlReg _FrontEndCtrlReg;
        private BackEndCtrlReg _BackEndCtrlReg;
        private CapacitiveCouplingCtrlReg1 _CapacitiveCouplingCtrlReg1;
        private CapacitiveCouplingCtrlReg2 _CapacitiveCouplingCtrlReg2;
        private AnalogTuningCtrlReg _AnalogTuningCtrlReg;
        private DebugMainCtrlReg _DebugMainCtrlReg;
        private bool _OkToClose;
        private bool _PrintFilterOptions;
        private string _FilterCMDs;

        public const int DACMIN = 0;
        public const int DACMAX = 1023;        // 10-Bit DAC
        public const double DACREF = 2.5;      // DAC Reference Voltage
        public const int DIGQTUNEMIN = 0;
        public const int DIGQTUNEMAX = 63;     // 6-Bit Q-Enhancement Digital Tuning
        public const int DIGFTUNEMIN = 0;    // Adjusted for reliable readings
        public const int DIGFTUNEMAX = 255;    // 8-Bit Q-Enhancement Digital Tuning
        public const int COUPLEMIN = 0;
        public const int COUPLEMAX = 31;

        public FilterControls(CommunicationManager cm, DebugText dt)
        {
            _ComManager = cm;
            _DebugText = dt;
```

```
    _FrontEndCtrlReg = new FrontEndCtrlReg();
    _BackEndCtrlReg = new BackEndCtrlReg();
    _CapacitiveCouplingCtrlReg1 = new CapacitiveCouplingCtrlReg1();
    _CapacitiveCouplingCtrlReg2 = new CapacitiveCouplingCtrlReg2();
    _AnalogTuningCtrlReg = new AnalogTuningCtrlReg();
    _DebugMainCtrlReg = new DebugMainCtrlReg();
    _PrintFilterOptions = false;
    _OkToClose = false;
    _FilterCMDs = string.Empty;
}

public FrontEndCtrlReg FrontEndCtrlReg
{
    get { return _FrontEndCtrlReg; }
}

public BackEndCtrlReg BackEndCtrlReg
{
    get { return _BackEndCtrlReg; }
}

public CapacitiveCouplingCtrlReg1 CapacitiveCouplingCtrlReg1
{
    get { return _CapacitiveCouplingCtrlReg1; }
}

public CapacitiveCouplingCtrlReg2 CapacitiveCouplingCtrlReg2
{
    get { return _CapacitiveCouplingCtrlReg2; }
}

public AnalogTuningCtrlReg AnalogTuningCtrlReg
{
    get { return _AnalogTuningCtrlReg; }
}

public DebugMainCtrlReg DebugMainCtrlReg
{
    get { return _DebugMainCtrlReg; }
}

public bool OkToClose
{
    get { return _OkToClose; }
    set { _OkToClose = value; }
}

public bool PrintFilterOptions
{
    get { return _PrintFilterOptions; }
    set { _PrintFilterOptions = value; }
}

public string FilterCMDs
{
    get { return _FilterCMDs; }
    set { _FilterCMDs = value; }
}


public void updateAnalogTuneLabel(Label txt, object obj)
{
    try
    {
        string type = obj.GetType().ToString();
        int i = type.LastIndexOf(".") + 1;
        int val = -1;
        decimal dVal = -1;
        type = type.Substring(i, type.Length - i);

        switch (type)
        {
            case "TrackBar":
                val = ((TrackBar)obj).Value;
                break;
            case "NumericUpDown":
                val = int.Parse(((NumericUpDown)obj).Value.ToString());
                break;
        }

        if (val == -1)
            MessageBox.Show("Could Not Update DAC Voltage Display");
        else
        {
            dVal = (decimal)(val * (DACREF / DACMAX));
            if (dVal < 1)
                txt.Text = decimal.Round(1000 * dVal, 1).ToString() + " mV";
            else
                txt.Text = decimal.Round(dVal, 3).ToString() + " V";
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

public void updateNumFromTck(NumericUpDown num, TrackBar tck)
{
    try
    {
        int val = tck.Value;
        if (val > num.Maximum)
            val = int.Parse(num.Maximum.ToString());
        else if (val < num.Minimum)
            val = int.Parse(num.Minimum.ToString());
        num.Value = decimal.Round((decimal)val);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

public void updateTckFromNum(TrackBar tck, NumericUpDown num)
{
    try
    {
        int val = int.Parse(num.Value.ToString());
        if (val > tck.Maximum)
            val = tck.Maximum;
```

```
                else if (val < tck.Minimum)
                    val = tck.Minimum;
                tck.Value = val;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        public bool activeLow(bool x)
        {
            return (x ? false : true);
        }

        public int bool2int(bool x)
        {
            return (x ? 1 : 0);
        }

        public void UpdateFilterCMDs()
        {
            _FilterCMDs = "@";
            _FilterCMDs += _FrontEndCtrlReg.Value.ToString("X").PadLeft(4, '0') + "&";
            _FilterCMDs += _BackEndCtrlReg.Value.ToString("X").PadLeft(4, '0') + "&";
            _FilterCMDs += _CapacitiveCouplingCtrlReg1.Value.ToString("X").PadLeft(3, '0') + "&";
            _FilterCMDs += _CapacitiveCouplingCtrlReg2.Value.ToString("X").PadLeft(3, '0') + "&";
            _FilterCMDs += _AnalogTuningCtrlReg.FrontEndAnalogF.ToString("X").PadLeft(3, '0') + "&";
            _FilterCMDs += _AnalogTuningCtrlReg.FrontEndAnalogQ.ToString("X").PadLeft(3, '0') + "&";
            _FilterCMDs += _AnalogTuningCtrlReg.BackEndAnalogF.ToString("X").PadLeft(3, '0') + "&";
            _FilterCMDs += _AnalogTuningCtrlReg.BackEndAnalogQ.ToString("X").PadLeft(3, '0') + "&";
            _FilterCMDs += _DebugMainCtrlReg.Value.ToString("X").PadLeft(2, '0') + "|";
        }

        public void ProgramFilter()
        {
            UpdateFilterCMDs();

            if (_PrintFilterOptions)
                PrintFilterOptionsToDebugWindow();

            _ComManager.WriteData(_FilterCMDs);
        }

        public void PrintFilterOptionsToDebugWindow()
        {
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "RF Switch On: " + (_DebugMainCtrlReg.isRFSWOn == 0 ?
"true" : "false") + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Front-End Enabled: " +
(_FrontEndCtrlReg.FrontEndEnable == 0 ? "true" : "false") + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Front-End AD Enabled: " +
(_FrontEndCtrlReg.FrontADEnable == 0 ? "true" : "false") + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Front-End AD Debug Enabled: " +
(_DebugMainCtrlReg.FrontADDebug == 1 ? "true" : "false") + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Front-End FD Enabled: " +
(_CapacitiveCouplingCtrlReg1.FrontFDEnable == 0 ? "true" : "false") + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Front-End FD Debug Enabled: " +
(_DebugMainCtrlReg.FrontFDDebug == 1 ? "true" : "false") + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Front-End Digital Q-Tune: " +
_FrontEndCtrlReg.FrontQDigitalTune + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Front-End Analog Q-Tune: " +
_AnalogTuningCtrlReg.FrontEndAnalogQ + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Front-End Digital F-Tune: " +
_FrontEndCtrlReg.FrontFDigitalTune + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Front-End Analog F-Tune: " +
_AnalogTuningCtrlReg.FrontEndAnalogF + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Back-End Enabled: " + (_BackEndCtrlReg.BackEndEnable
== 0 ? "true" : "false") + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Back-End AD Enabled: " + (_BackEndCtrlReg.BackADEnable
== 0 ? "true" : "false") + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Back-End AD Debug Enabled: " +
(_DebugMainCtrlReg.BackADDebug == 1 ? "true" : "false") + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Back-End FD Enabled: " +
(_CapacitiveCouplingCtrlReg2.BackFDEnable == 0 ? "true" : "false") + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Back-End FD Debug Enabled: " +
(_DebugMainCtrlReg.BackFDDebug == 1 ? "true" : "false") + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Back-End Digital Q-Tune: " +
_BackEndCtrlReg.BackQDigitalTune + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Back-End Analog Q-Tune: " +
_AnalogTuningCtrlReg.BackEndAnalogQ + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Back-End Digital F-Tune: " +
_BackEndCtrlReg.BackFDigitalTune + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Back-End Analog F-Tune: " +
_AnalogTuningCtrlReg.BackEndAnalogF + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Couple Upper: " +
_CapacitiveCouplingCtrlReg1.CoupleUpper + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Couple Lower: " +
_CapacitiveCouplingCtrlReg1.CoupleLower + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Couple UFLB: " +
_CapacitiveCouplingCtrlReg2.CoupleUFLB + "\r\n");
            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Couple LFUB: " +
_CapacitiveCouplingCtrlReg2.CoupleLFUB + "\r\n");
        }
    }

    public class FrontEndCtrlReg
    {
        private int  _VALUE;
        private int _FrontEndEnable;
        private int  _FrontFDigitalTune;
        private int  _FrontQDigitalTune;
        private int _FrontADEnable;

        public FrontEndCtrlReg()
        {
            _FrontEndEnable = 1;
            _FrontFDigitalTune = FilterControls.DIGFTUNEMIN;
            _FrontQDigitalTune = FilterControls.DIGQTUNEMIN;
            _FrontADEnable = 1;
            Update();
        }

        public int Value
        {
            get { return _VALUE; }
        }

        public int FrontEndEnable
```

```
    {
        get { return _FrontEndEnable; }
        set { _FrontEndEnable = value; }
    }

    public int FrontFDigitalTune
    {
        get { return _FrontFDigitalTune; }
        set { _FrontFDigitalTune = value; }
    }

    public int FrontQDigitalTune
    {
        get { return _FrontQDigitalTune; }
        set { _FrontQDigitalTune = value; }
    }

    public int FrontADEnable
    {
        get { return _FrontADEnable; }
        set { _FrontADEnable = value; }
    }

    public void Update()
    {
        _VALUE = ((_FrontEndEnable & 1) +
                 ((_FrontFDigitalTune & FilterControls.DIGFTUNEMAX) << 1) +
                 ((_FrontQDigitalTune & FilterControls.DIGQTUNEMAX) << 9) +
                 ((_FrontADEnable & 1) << 15)) & 0xFFFF;
    }
}

public class BackEndCtrlReg
{
    private int _VALUE;
    private int _BackEndEnable;
    private int _BackFDigitalTune;
    private int _BackQDigitalTune;
    private int _BackADEnable;

    public BackEndCtrlReg()
    {
        _BackEndEnable = 1;
        _BackFDigitalTune = FilterControls.DIGFTUNEMIN;
        _BackQDigitalTune = FilterControls.DIGFTUNEMIN;
        _BackADEnable = 1;
        Update();
    }

    public int Value
    {
        get { return _VALUE; }
    }

    public int BackEndEnable
    {
        get { return _BackEndEnable; }
        set { _BackEndEnable = value; }
    }

    public int BackFDigitalTune
    {
        get { return _BackFDigitalTune; }
        set { _BackFDigitalTune = value; }
    }

    public int BackQDigitalTune
    {
        get { return _BackQDigitalTune; }
        set { _BackQDigitalTune = value; }
    }

    public int BackADEnable
    {
        get { return _BackADEnable; }
        set { _BackADEnable = value; }
    }

    public void Update()
    {
        _VALUE = ((_BackEndEnable & 1) +
                 ((_BackFDigitalTune & FilterControls.DIGFTUNEMAX) << 1) +
                 ((_BackQDigitalTune & FilterControls.DIGQTUNEMAX) << 9) +
                 ((_BackADEnable & 1) << 15)) & 0xFFFF;
    }
}

public class CapacitiveCouplingCtrlReg1
{
    private int _VALUE;
    private int _CoupleUpper;
    private int _CoupleLower;
    private int _FrontFDEnable;

    public CapacitiveCouplingCtrlReg1()
    {
        _CoupleUpper = FilterControls.COUPLEMIN;
        _CoupleLower = FilterControls.COUPLEMIN;
        _FrontFDEnable = 1;
        Update();
    }

    public int Value
    {
        get { return _VALUE; }
    }

    public int CoupleUpper
    {
        get { return _CoupleUpper; }
        set { _CoupleUpper = value; }
    }

    public int CoupleLower
    {
        get { return _CoupleLower; }
        set { _CoupleLower = value; }
    }

    public int FrontFDEnable
```

```
    {
        get { return _FrontFDEnable; }
        set { _FrontFDEnable = value; }
    }

    public void Update()
    {
        _VALUE = (((_CoupleUpper & FilterControls.COUPLEMAX)) +
                  ((_CoupleLower & FilterControls.COUPLEMAX) << 5) +
                  ((_FrontFDEnable & 1) << 10)) & 0x07FF;
    }
}

public class CapacitiveCouplingCtrlReg2
{
    private int _VALUE;
    private int _CoupleUFLB;
    private int _CoupleLFUB;
    private int _BackFDEnable;

    public CapacitiveCouplingCtrlReg2()
    {
        _CoupleUFLB = FilterControls.COUPLEMIN;
        _CoupleLFUB = FilterControls.COUPLEMIN;
        _BackFDEnable = 1;
        Update();
    }

    public int Value
    {
        get { return _VALUE; }
    }

    public int CoupleUFLB
    {
        get { return _CoupleUFLB; }
        set { _CoupleUFLB = value; }
    }

    public int CoupleLFUB
    {
        get { return _CoupleLFUB; }
        set { _CoupleLFUB = value; }
    }

    public int BackFDEnable
    {
        get { return _BackFDEnable; }
        set { _BackFDEnable = value; }
    }

    public void Update()
    {
        _VALUE = (((_CoupleUFLB & FilterControls.COUPLEMAX)) +
                  ((_CoupleLFUB & FilterControls.COUPLEMAX) << 5) +
                  ((_BackFDEnable & 1) << 10)) & 0x07FF;
    }
}

public class AnalogTuningCtrlReg
{
    private int _FrontEndAnalogF;
    private int _FrontEndAnalogQ;
    private int _BackEndAnalogF;
    private int _BackEndAnalogQ;

    public AnalogTuningCtrlReg()
    {
        _FrontEndAnalogF = FilterControls.DACMIN;
        _FrontEndAnalogQ = FilterControls.DACMIN;
        _BackEndAnalogF = FilterControls.DACMIN;
        _BackEndAnalogQ = FilterControls.DACMIN;
    }

    public int FrontEndAnalogF
    {
        get { return _FrontEndAnalogF; }
        set { _FrontEndAnalogF = value & FilterControls.DACMAX; }
    }

    public int FrontEndAnalogQ
    {
        get { return _FrontEndAnalogQ; }
        set { _FrontEndAnalogQ = value & FilterControls.DACMAX; }
    }

    public int BackEndAnalogF
    {
        get { return _BackEndAnalogF; }
        set { _BackEndAnalogF = value & FilterControls.DACMAX; }
    }

    public int BackEndAnalogQ
    {
        get { return _BackEndAnalogQ; }
        set { _BackEndAnalogQ = value & FilterControls.DACMAX; }
    }
}

public class DebugMainCtrlReg
{
    private int _VALUE;
    private int _FrontADDebug;
    private int _FrontFDDebug;
    private int _BackADDebug;
    private int _BackFDDebug;
    private int _RFSWOn;

    public DebugMainCtrlReg()
    {
        _FrontADDebug = 0;
        _FrontFDDebug = 0;
        _BackADDebug = 0;
        _BackFDDebug = 0;
        _RFSWOn = 1;              // Active-Low
        Update();
    }

    public int Value
    {
```

```
            get { return _VALUE; }
        }

        public int FrontADDebug
        {
            get { return _FrontADDebug; }
            set { _FrontADDebug = value; }
        }

        public int FrontFDDebug
        {
            get { return _FrontFDDebug; }
            set { _FrontFDDebug = value; }
        }

        public int BackADDebug
        {
            get { return _BackADDebug; }
            set { _BackADDebug = value; }
        }

        public int BackFDDebug
        {
            get { return _BackFDDebug; }
            set { _BackFDDebug = value; }
        }

        public int isRFSWOn
        {
            get { return _RFSWOn; }
            set { _RFSWOn = value; }
        }

        public void Update()
        {
             _VALUE = ((_FrontADDebug & 1) +
                       ((_FrontFDDebug & 1) << 1) +
                       ((_BackADDebug & 1) << 2) +
                       ((_BackFDDebug & 1) << 3) +
                       ((_RFSWOn & 1) << 4)) & 0x1F;
        }
    }
}
```

FRMFILTERCONTROLS.CS

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace QEFilterV2TestAppV2
{
    public partial class frmFilterControls : Form
    {
        private FilterControls _FilterControls;
        private DebugText _DebugText;
        private SaveFileDialog _DialogSaveFilterControls;
        private OpenFileDialog _DialogLoadFilterControls;
        private bool _RFSWOn;

        public frmFilterControls(FilterControls fs, DebugText dt)
        {
            InitializeComponent();
            this.Closing += new CancelEventHandler(frmFilterSettings_Closing);
            _FilterControls = fs;
            _DebugText = dt;
            _RFSWOn = false;

            _DialogSaveFilterControls = new SaveFileDialog();
            _DialogSaveFilterControls.DefaultExt = "txt";
            _DialogSaveFilterControls.Filter = "Text File (*.txt)|*.txt";
            _DialogSaveFilterControls.AddExtension = true;
            _DialogSaveFilterControls.RestoreDirectory = true;
            _DialogSaveFilterControls.Title = "Save Filter Controls as...";
            _DialogSaveFilterControls.InitialDirectory = @"C:/";
            _DialogSaveFilterControls.FileName = "";

            _DialogLoadFilterControls = new OpenFileDialog();
            _DialogLoadFilterControls.DefaultExt = "txt";
            _DialogLoadFilterControls.RestoreDirectory = true;
            _DialogLoadFilterControls.Title = "Load Filter Controls";
            _DialogLoadFilterControls.InitialDirectory = @"C:/";
            _DialogLoadFilterControls.FileName = "";

            InitalizeFilterControls();
        }

        private void frmFilterSettings_Closing(object sender, CancelEventArgs cArgs)
        {
            if (!_FilterControls.OkToClose)
                cArgs.Cancel = true;
            else
                cArgs.Cancel = false;
        }

        private void InitalizeFilterControls()
        {
            chkPrintOptions.Checked = false;
            chkFrontEnable.Checked = false;
            chkFrontADEnable.Checked = false;
            chkFrontADDebug.Checked = false;
            chkFrontADDebug.Enabled = false;
            chkFrontFDEnable.Checked = false;
            chkFrontFDDebug.Checked = false;
            chkFrontFDDebug.Enabled = false;
            chkBackEnable.Checked = false;
            chkBackADEnable.Checked = false;
            chkBackADDebug.Checked = false;
```

```
        chkBackADDebug.Enabled = false;
        chkBackFDEnable.Checked = false;
        chkBackFDDebug.Checked = false;
        chkBackFDDebug.Enabled = false;
        numFrontQDigitalTune.Minimum = FilterControls.DIGQTUNEMIN;
        numFrontQDigitalTune.Maximum = FilterControls.DIGQTUNEMAX;
        numFrontQDigitalTune.Value = numFrontQDigitalTune.Minimum;
        tckFrontQDigitalTune.Minimum = FilterControls.DIGQTUNEMIN;
        tckFrontQDigitalTune.Maximum = FilterControls.DIGQTUNEMAX;
        tckFrontQDigitalTune.Value = tckFrontQDigitalTune.Minimum;
        numFrontFDigitalTune.Minimum = FilterControls.DIGFTUNEMIN;
        numFrontFDigitalTune.Maximum = FilterControls.DIGFTUNEMAX;
        numFrontFDigitalTune.Value = numFrontFDigitalTune.Minimum;
        tckFrontFDigitalTune.Minimum = FilterControls.DIGFTUNEMIN;
        tckFrontFDigitalTune.Maximum = FilterControls.DIGFTUNEMAX;
        tckFrontFDigitalTune.Value = tckFrontFDigitalTune.Minimum;
        numFrontQAnalogTune.Minimum = FilterControls.DACMIN;
        numFrontQAnalogTune.Maximum = FilterControls.DACMAX;
        numFrontQAnalogTune.Value = numFrontQAnalogTune.Minimum;
        tckFrontQAnalogTune.Minimum = FilterControls.DACMIN;
        tckFrontQAnalogTune.Maximum = FilterControls.DACMAX;
        tckFrontQAnalogTune.Value = tckFrontQAnalogTune.Minimum;
        numFrontFAnalogTune.Minimum = FilterControls.DACMIN;
        numFrontFAnalogTune.Maximum = FilterControls.DACMAX;
        numFrontFAnalogTune.Value = numFrontFAnalogTune.Minimum;
        tckFrontFAnalogTune.Minimum = FilterControls.DACMIN;
        tckFrontFAnalogTune.Maximum = FilterControls.DACMAX;
        tckFrontFAnalogTune.Value = tckFrontFAnalogTune.Minimum;
        numBackQDigitalTune.Minimum = FilterControls.DIGQTUNEMIN;
        numBackQDigitalTune.Maximum = FilterControls.DIGQTUNEMAX;
        numBackQDigitalTune.Value = numBackQDigitalTune.Minimum;
        tckBackQDigitalTune.Minimum = FilterControls.DIGQTUNEMIN;
        tckBackQDigitalTune.Maximum = FilterControls.DIGQTUNEMAX;
        tckBackQDigitalTune.Value = tckBackQDigitalTune.Minimum;
        numBackFDigitalTune.Minimum = FilterControls.DIGFTUNEMIN;
        numBackFDigitalTune.Maximum = FilterControls.DIGFTUNEMAX;
        numBackFDigitalTune.Value = numBackFDigitalTune.Minimum;
        tckBackFDigitalTune.Minimum = FilterControls.DIGFTUNEMIN;
        tckBackFDigitalTune.Maximum = FilterControls.DIGFTUNEMAX;
        tckBackFDigitalTune.Value = tckBackFDigitalTune.Minimum;
        numBackQAnalogTune.Minimum = FilterControls.DACMIN;
        numBackQAnalogTune.Maximum = FilterControls.DACMAX;
        numBackQAnalogTune.Value = numBackQAnalogTune.Minimum;
        tckBackQAnalogTune.Minimum = FilterControls.DACMIN;
        tckBackQAnalogTune.Maximum = FilterControls.DACMAX;
        tckBackQAnalogTune.Value = tckBackQAnalogTune.Minimum;
        numBackFAnalogTune.Minimum = FilterControls.DACMIN;
        numBackFAnalogTune.Maximum = FilterControls.DACMAX;
        numBackFAnalogTune.Value = numBackFAnalogTune.Minimum;
        tckBackFAnalogTune.Minimum = FilterControls.DACMIN;
        tckBackFAnalogTune.Maximum = FilterControls.DACMAX;
        tckBackFAnalogTune.Value = tckBackFAnalogTune.Minimum;
        numCoupleUpper.Minimum = FilterControls.COUPLEMIN;
        numCoupleUpper.Maximum = FilterControls.COUPLEMAX;
        numCoupleUpper.Value = numCoupleUpper.Minimum;
        tckCoupleUpper.Minimum = FilterControls.COUPLEMIN;
        tckCoupleUpper.Maximum = FilterControls.COUPLEMAX;
        tckCoupleUpper.Value = tckCoupleUpper.Minimum;
        numCoupleLower.Minimum = FilterControls.COUPLEMIN;
        numCoupleLower.Maximum = FilterControls.COUPLEMAX;
        numCoupleLower.Value = numCoupleLower.Minimum;
        tckCoupleLower.Minimum = FilterControls.COUPLEMIN;
        tckCoupleLower.Maximum = FilterControls.COUPLEMAX;
        tckCoupleLower.Value = tckCoupleLower.Minimum;
        numCoupleUFLB.Minimum = FilterControls.COUPLEMIN;
        numCoupleUFLB.Maximum = FilterControls.COUPLEMAX;
        numCoupleUFLB.Value = numCoupleUFLB.Minimum;
        tckCoupleUFLB.Minimum = FilterControls.COUPLEMIN;
        tckCoupleUFLB.Maximum = FilterControls.COUPLEMAX;
        tckCoupleUFLB.Value = tckCoupleUFLB.Minimum;
        numCoupleLFUB.Minimum = FilterControls.COUPLEMIN;
        numCoupleLFUB.Maximum = FilterControls.COUPLEMAX;
        numCoupleLFUB.Value = numCoupleLFUB.Minimum;
        tckCoupleLFUB.Minimum = FilterControls.COUPLEMIN;
        tckCoupleLFUB.Maximum = FilterControls.COUPLEMAX;
        tckCoupleLFUB.Value = tckCoupleLFUB.Minimum;
    }

    private void tckFrontQDigitalTune_Scroll(object sender, EventArgs e)
    {
        _FilterControls.updateNumFromTck(numFrontQDigitalTune, tckFrontQDigitalTune);
        _FilterControls.updateTckFromNum(tckFrontQDigitalTune, numFrontQDigitalTune);
    }

    private void numFrontQDigitalTune_ValueChanged(object sender, EventArgs e)
    {
        _FilterControls.updateTckFromNum(tckFrontQDigitalTune, numFrontQDigitalTune);
        _FilterControls.updateNumFromTck(numFrontQDigitalTune, tckFrontQDigitalTune);
    }

    private void tckFrontFDigitalTune_Scroll(object sender, EventArgs e)
    {
        _FilterControls.updateNumFromTck(numFrontFDigitalTune, tckFrontFDigitalTune);
        _FilterControls.updateTckFromNum(tckFrontFDigitalTune, numFrontFDigitalTune);
    }

    private void numFrontFDigitalTune_ValueChanged(object sender, EventArgs e)
    {
        _FilterControls.updateTckFromNum(tckFrontFDigitalTune, numFrontFDigitalTune);
        _FilterControls.updateNumFromTck(numFrontFDigitalTune, tckFrontFDigitalTune);
    }

    private void tckFrontQAnalogTune_Scroll(object sender, EventArgs e)
    {
        _FilterControls.updateAnalogTuneLabel(lblFrontQAnalogTune, tckFrontQAnalogTune);
        _FilterControls.updateNumFromTck(numFrontQAnalogTune, tckFrontQAnalogTune);
        _FilterControls.updateTckFromNum(tckFrontQAnalogTune, numFrontQAnalogTune);
    }

    private void numFrontQAnalogTune_ValueChanged(object sender, EventArgs e)
    {
        _FilterControls.updateAnalogTuneLabel(lblFrontQAnalogTune, numFrontQAnalogTune);
        _FilterControls.updateTckFromNum(tckFrontQAnalogTune, numFrontQAnalogTune);
        _FilterControls.updateNumFromTck(numFrontQAnalogTune, tckFrontQAnalogTune);
    }
```

```
private void tckFrontFAnalogTune_Scroll(object sender, EventArgs e)
{
    _FilterControls.updateAnalogTuneLabel(lblFrontFAnalogTune, tckFrontFAnalogTune);
    _FilterControls.updateNumFromTck(numFrontFAnalogTune, tckFrontFAnalogTune);
    _FilterControls.updateTckFromNum(tckFrontFAnalogTune, numFrontFAnalogTune);
}

private void numFrontFAnalogTune_ValueChanged(object sender, EventArgs e)
{
    _FilterControls.updateAnalogTuneLabel(lblFrontFAnalogTune, numFrontFAnalogTune);
    _FilterControls.updateTckFromNum(tckFrontFAnalogTune, numFrontFAnalogTune);
    _FilterControls.updateNumFromTck(numFrontFAnalogTune, tckFrontFAnalogTune);
}

private void tckBackQDigitalTune_Scroll(object sender, EventArgs e)
{
    _FilterControls.updateNumFromTck(numBackQDigitalTune, tckBackQDigitalTune);
    _FilterControls.updateTckFromNum(tckBackQDigitalTune, numBackQDigitalTune);
}

private void numBackQDigitalTune_ValueChanged(object sender, EventArgs e)
{
    _FilterControls.updateTckFromNum(tckBackQDigitalTune, numBackQDigitalTune);
    _FilterControls.updateNumFromTck(numBackQDigitalTune, tckBackQDigitalTune);
}

private void tckBackFDigitalTune_Scroll(object sender, EventArgs e)
{
    _FilterControls.updateNumFromTck(numBackFDigitalTune, tckBackFDigitalTune);
    _FilterControls.updateTckFromNum(tckBackFDigitalTune, numBackFDigitalTune);
}

private void numBackFDigitalTune_ValueChanged(object sender, EventArgs e)
{
    _FilterControls.updateTckFromNum(tckBackFDigitalTune, numBackFDigitalTune);
    _FilterControls.updateNumFromTck(numBackFDigitalTune, tckBackFDigitalTune);
}

private void tckBackQAnalogTune_Scroll(object sender, EventArgs e)
{
    _FilterControls.updateAnalogTuneLabel(lblBackQAnalogTune, tckBackQAnalogTune);
    _FilterControls.updateNumFromTck(numBackQAnalogTune, tckBackQAnalogTune);
    _FilterControls.updateTckFromNum(tckBackQAnalogTune, numBackQAnalogTune);
}

private void numBackQAnalogTune_ValueChanged(object sender, EventArgs e)
{
    _FilterControls.updateAnalogTuneLabel(lblBackQAnalogTune, numBackQAnalogTune);
    _FilterControls.updateTckFromNum(tckBackQAnalogTune, numBackQAnalogTune);
    _FilterControls.updateNumFromTck(numBackQAnalogTune, tckBackQAnalogTune);
}

private void tckBackFAnalogTune_Scroll(object sender, EventArgs e)
{
    _FilterControls.updateAnalogTuneLabel(lblBackFAnalogTune, tckBackFAnalogTune);
    _FilterControls.updateNumFromTck(numBackFAnalogTune, tckBackFAnalogTune);
    _FilterControls.updateTckFromNum(tckBackFAnalogTune, numBackFAnalogTune);
}

private void numBackFAnalogTune_ValueChanged(object sender, EventArgs e)
{
    _FilterControls.updateAnalogTuneLabel(lblBackFAnalogTune, numBackFAnalogTune);
    _FilterControls.updateTckFromNum(tckBackFAnalogTune, numBackFAnalogTune);
    _FilterControls.updateNumFromTck(numBackFAnalogTune, tckBackFAnalogTune);
}

private void tckCoupleUpper_Scroll(object sender, EventArgs e)
{
    _FilterControls.updateNumFromTck(numCoupleUpper, tckCoupleUpper);
    _FilterControls.updateTckFromNum(tckCoupleUpper, numCoupleUpper);
}

private void numCoupleUpper_ValueChanged(object sender, EventArgs e)
{
    _FilterControls.updateTckFromNum(tckCoupleUpper, numCoupleUpper);
    _FilterControls.updateNumFromTck(numCoupleUpper, tckCoupleUpper);
}

private void tckCoupleLower_Scroll(object sender, EventArgs e)
{
    _FilterControls.updateNumFromTck(numCoupleLower, tckCoupleLower);
    _FilterControls.updateTckFromNum(tckCoupleLower, numCoupleLower);
}

private void numCoupleLower_ValueChanged(object sender, EventArgs e)
{
    _FilterControls.updateTckFromNum(tckCoupleLower, numCoupleLower);
    _FilterControls.updateNumFromTck(numCoupleLower, tckCoupleLower);
}

private void tckCoupleUFLB_Scroll(object sender, EventArgs e)
{
    _FilterControls.updateNumFromTck(numCoupleUFLB, tckCoupleUFLB);
    _FilterControls.updateTckFromNum(tckCoupleUFLB, numCoupleUFLB);
}

private void numCoupleUFLB_ValueChanged(object sender, EventArgs e)
{
    _FilterControls.updateTckFromNum(tckCoupleUFLB, numCoupleUFLB);
    _FilterControls.updateNumFromTck(numCoupleUFLB, tckCoupleUFLB);
}

private void tckCoupleLFUB_Scroll(object sender, EventArgs e)
{
    _FilterControls.updateNumFromTck(numCoupleLFUB, tckCoupleLFUB);
    _FilterControls.updateTckFromNum(tckCoupleLFUB, numCoupleLFUB);
}

private void numCoupleLFUB_ValueChanged(object sender, EventArgs e)
{
    _FilterControls.updateTckFromNum(tckCoupleLFUB, numCoupleLFUB);
    _FilterControls.updateNumFromTck(numCoupleLFUB, tckCoupleLFUB);
}

private void chkFrontEnable_CheckedChanged(object sender, EventArgs e)
{
}

private void chkBackEnable_CheckedChanged(object sender, EventArgs e)
{
}
```

```
private void chkFrontADEnable_CheckedChanged(object sender, EventArgs e)
{
    if (chkFrontADEnable.Checked)
        chkFrontADDebug.Enabled = true;
    else
    {
        chkFrontADDebug.Enabled = false;
        chkFrontADDebug.Checked = false;
    }
}

private void chkFrontFDEnable_CheckedChanged(object sender, EventArgs e)
{
    if (chkFrontFDEnable.Checked)
        chkFrontFDDebug.Enabled = true;
    else
    {
        chkFrontFDDebug.Enabled = false;
        chkFrontFDDebug.Checked = false;
    }
}

private void chkBackADEnable_CheckedChanged(object sender, EventArgs e)
{
    if (chkBackADEnable.Checked)
        chkBackADDebug.Enabled = true;
    else
    {
        chkBackADDebug.Enabled = false;
        chkBackADDebug.Checked = false;
    }
}

private void chkBackFDEnable_CheckedChanged(object sender, EventArgs e)
{
    if (chkBackFDEnable.Checked)
        chkBackFDDebug.Enabled = true;
    else
    {
        chkBackFDDebug.Enabled = false;
        chkBackFDDebug.Checked = false;
    }
}

private void chkFrontADDebug_CheckedChanged(object sender, EventArgs e)
{
}

private void chkFrontFDDebug_CheckedChanged(object sender, EventArgs e)
{
}

private void chkBackADDebug_CheckedChanged(object sender, EventArgs e)
{
}

private void chkBackFDDebug_CheckedChanged(object sender, EventArgs e)
{
}

private void btnFilterReset_Click(object sender, EventArgs e)
{
    InitalizeFilterControls();
}

private void btnFilterSave_Click(object sender, EventArgs e)
{
    SaveFilterControls();
}

private void btnFilterLoad_Click(object sender, EventArgs e)
{
    LoadFilterControls();
}

private void btnFilterProgram_Click(object sender, EventArgs e)
{
    if (chkPrintOptions.Checked)
        _FilterControls.PrintFilterOptions = true;
    else
        _FilterControls.PrintFilterOptions = false;

    _FilterControls.FrontEndCtrlReg.FrontEndEnable = _FilterControls.bool2int(_FilterControls.activeLow(chkFrontEnable.Checked));
    _FilterControls.FrontEndCtrlReg.FrontFDigitalTune = tckFrontFDigitalTune.Value;
    _FilterControls.FrontEndCtrlReg.FrontQDigitalTune = tckFrontQDigitalTune.Value;
    _FilterControls.FrontEndCtrlReg.FrontADEnable = _FilterControls.bool2int(_FilterControls.activeLow(chkFrontADEnable.Checked));
    _FilterControls.FrontEndCtrlReg.Update();

    _FilterControls.BackEndCtrlReg.BackEndEnable = _FilterControls.bool2int(_FilterControls.activeLow(chkBackEnable.Checked));
    _FilterControls.BackEndCtrlReg.BackFDigitalTune = tckBackFDigitalTune.Value;
    _FilterControls.BackEndCtrlReg.BackQDigitalTune = tckBackQDigitalTune.Value;
    _FilterControls.BackEndCtrlReg.BackADEnable = _FilterControls.bool2int(_FilterControls.activeLow(chkBackADEnable.Checked));
    _FilterControls.BackEndCtrlReg.Update();

    _FilterControls.CapacitiveCouplingCtrlReg1.CoupleUpper = tckCoupleUpper.Value;
    _FilterControls.CapacitiveCouplingCtrlReg1.CoupleLower = tckCoupleLower.Value;
    _FilterControls.CapacitiveCouplingCtrlReg1.FrontFDEnable = _FilterControls.bool2int(_FilterControls.activeLow(chkFrontFDEnable.Checked));
    _FilterControls.CapacitiveCouplingCtrlReg1.Update();

    _FilterControls.CapacitiveCouplingCtrlReg2.CoupleUFLB = tckCoupleUFLB.Value;
    _FilterControls.CapacitiveCouplingCtrlReg2.CoupleLFUB = tckCoupleLFUB.Value;
    _FilterControls.CapacitiveCouplingCtrlReg2.BackFDEnable = _FilterControls.bool2int(_FilterControls.activeLow(chkBackFDEnable.Checked));
    _FilterControls.CapacitiveCouplingCtrlReg2.Update();

    _FilterControls.AnalogTuningCtrlReg.FrontEndAnalogF = tckFrontFAnalogTune.Value;
    _FilterControls.AnalogTuningCtrlReg.FrontEndAnalogQ = tckFrontQAnalogTune.Value;
    _FilterControls.AnalogTuningCtrlReg.BackEndAnalogF = tckBackFAnalogTune.Value;
    _FilterControls.AnalogTuningCtrlReg.BackEndAnalogQ = tckBackQAnalogTune.Value;
    // No Explicit Update Function for AnalogTuningCtrlReg

    _FilterControls.DebugMainCtrlReg.FrontADDebug = _FilterControls.bool2int(chkFrontADDebug.Checked);
    _FilterControls.DebugMainCtrlReg.FrontFDDebug = _FilterControls.bool2int(chkFrontFDDebug.Checked);
    _FilterControls.DebugMainCtrlReg.BackADDebug = _FilterControls.bool2int(chkBackADDebug.Checked);
    _FilterControls.DebugMainCtrlReg.BackFDDebug = _FilterControls.bool2int(chkBackFDDebug.Checked);
    _FilterControls.DebugMainCtrlReg.isRFSWOn = _FilterControls.bool2int(_FilterControls.activeLow(_RFSWOn));
    _FilterControls.DebugMainCtrlReg.Update();

    _FilterControls.ProgramFilter();
}
```

```
private void SaveFilterControls()
{
    try
    {
        TextBox ctrls = new TextBox();
        string fileName = string.Empty;
        int dirIndex = 0;

        _DialogSaveFilterControls.ShowDialog();
        ctrls.AppendText("chkFrontEnable:" + (chkFrontEnable.Checked ? "true" : "false") + "\r\n");
        ctrls.AppendText("chkFrontADEnable:" + (chkFrontADEnable.Checked ? "true" : "false") + "\r\n");
        ctrls.AppendText("chkFrontADDebug:" + (chkFrontADDebug.Checked ? "true" : "false") + "\r\n");
        ctrls.AppendText("chkFrontFDEnable:" + (chkFrontFDEnable.Checked ? "true" : "false") + "\r\n");
        ctrls.AppendText("chkFrontFDDebug:" + (chkFrontFDDebug.Checked ? "true" : "false") + "\r\n");
        ctrls.AppendText("tckFrontQDigitalTune:" + tckFrontQDigitalTune.Value + "\r\n");
        ctrls.AppendText("tckFrontQAnalogTune:" + tckFrontQAnalogTune.Value + "\r\n");
        ctrls.AppendText("tckFrontFDigitalTune:" + tckFrontFDigitalTune.Value + "\r\n");
        ctrls.AppendText("tckFrontFAnalogTune:" + tckFrontFAnalogTune.Value + "\r\n");
        ctrls.AppendText("chkBackEnable:" + (chkBackEnable.Checked ? "true" : "false") + "\r\n");
        ctrls.AppendText("chkBackADEnable:" + (chkBackADEnable.Checked ? "true" : "false") + "\r\n");
        ctrls.AppendText("chkBackADDebug:" + (chkBackADDebug.Checked ? "true" : "false") + "\r\n");
        ctrls.AppendText("chkBackFDEnable:" + (chkBackFDEnable.Checked ? "true" : "false") + "\r\n");
        ctrls.AppendText("chkBackFDDebug:" + (chkBackFDDebug.Checked ? "true" : "false") + "\r\n");
        ctrls.AppendText("tckBackQDigitalTune:" + tckBackQDigitalTune.Value + "\r\n");
        ctrls.AppendText("tckBackQAnalogTune:" + tckBackQAnalogTune.Value + "\r\n");
        ctrls.AppendText("tckBackFDigitalTune:" + tckBackFDigitalTune.Value + "\r\n");
        ctrls.AppendText("tckBackFAnalogTune:" + tckBackFAnalogTune.Value + "\r\n");
        ctrls.AppendText("tckCoupleUpper:" + tckCoupleUpper.Value + "\r\n");
        ctrls.AppendText("tckCoupleLower:" + tckCoupleLower.Value + "\r\n");
        ctrls.AppendText("tckCoupleUFLB:" + tckCoupleUFLB.Value + "\r\n");
        ctrls.AppendText("tckCoupleLFUB:" + tckCoupleLFUB.Value + "\r\n");

        if (_DialogSaveFilterControls.FileName != "")
        {
            dirIndex = _DialogSaveFilterControls.FileName.LastIndexOf("\\") + 1;
            fileName = _DialogSaveFilterControls.FileName.Substring(dirIndex, _DialogSaveFilterControls.FileName.Length - dirIndex);
            File.WriteAllText(@_DialogSaveFilterControls.FileName, ctrls.Text);
            _DebugText.DisplayDebugData(DebugText.MessageClass.Status, DebugText.MessageType.Normal, fileName + " saved successfully.");
        }
        else
            throw new Exception("Save Dialog exited or canceled.");
    }
    catch (Exception ex)
    {
        _DebugText.DisplayDebugData(DebugText.MessageClass.Status, DebugText.MessageType.Error, ex.Message.ToString());
    }
}

private void LoadFilterControls()
{
    try
    {
        string fileName = string.Empty, line = string.Empty, param = string.Empty, paramName = string.Empty;
        int dirIndex = 0, delIndex = 0;
        TextReader tr = null;
        _DialogLoadFilterControls.ShowDialog();

        if (_DialogLoadFilterControls.FileName != "")
        {

            tr = new StreamReader(_DialogLoadFilterControls.FileName);
            dirIndex = _DialogLoadFilterControls.FileName.LastIndexOf("\\") + 1;
            fileName = _DialogLoadFilterControls.FileName.Substring(dirIndex, _DialogLoadFilterControls.FileName.Length - dirIndex);

            _DebugText.DisplayDebugData(DebugText.MessageClass.Status, DebugText.MessageType.Normal, fileName + " opened successfully.");
            while ((line = tr.ReadLine()) != null)
            {
                delIndex = line.IndexOf(':');
                _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, line + "\r\n");
                param = line.Substring(delIndex + 1, line.Length - delIndex - 1);
                paramName = line.Substring(0, delIndex);

                foreach (Control ctrl in groupFilterSettings.Controls)
                {
                    if (ctrl.Name.Equals(paramName) && paramName.Contains("chk"))
                        loadCheckBoxParam((CheckBox)ctrl, param);
                    else if (ctrl.Name.Equals(paramName) && paramName.Contains("tck"))
                    {
                        NumericUpDown num = null;
                        string numName = string.Empty;
                        foreach (Control ctrl2 in groupFilterSettings.Controls)
                        {
                            numName = "num" + paramName.Substring(3, paramName.Length - 3);
                            if (ctrl2.Name.Equals(numName))
                                num = (NumericUpDown)ctrl2;
                        }
                        loadTrackBarParam((TrackBar)ctrl, num, param);
                    }
                }
            }
            tr.Close();
        }
        else
            throw new Exception("Load Dialog exited or canceled.");

    }
    catch (Exception ex)
    {
        _DebugText.DisplayDebugData(DebugText.MessageClass.Status, DebugText.MessageType.Error, ex.Message.ToString());
    }
}

private void loadTrackBarParam(TrackBar tck, NumericUpDown num, string param)
{
    try
    {
        int val = int.Parse(param.Replace(" ", ""));
        tck.Value = val;
        num.Value = (decimal)val;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }

}

private void loadCheckBoxParam(CheckBox chk, string param)
{
```

```
            if (param.Replace(" ", "").Equals("true"))
            {
                chk.Checked = true;
                chk.Enabled = true;
            }
            else
                chk.Checked = false;
        }

        private void chkPrintOptions_CheckedChanged(object sender, EventArgs e)
        {
            if (chkPrintOptions.Checked)
                _FilterControls.PrintFilterOptions = true;
            else
                _FilterControls.PrintFilterOptions = false;
        }

        private void chkRFSW_CheckedChanged(object sender, EventArgs e)
        {
            if (chkRFSW.Checked)
                _RFSWOn = true;
            else
                _RFSWOn = false;

        }
    }
}
```

FRMFILTERCONTROLS.DESIGNER.CS

```
namespace QEFilterV2TestAppV2
{
    partial class frmFilterControls
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.groupFilterSettings = new System.Windows.Forms.GroupBox();
            this.chkRFSW = new System.Windows.Forms.CheckBox();
            this.chkPrintOptions = new System.Windows.Forms.CheckBox();
            this.btnFilterProgram = new System.Windows.Forms.Button();
            this.btnFilterLoad = new System.Windows.Forms.Button();
            this.btnFilterSave = new System.Windows.Forms.Button();
            this.btnFilterReset = new System.Windows.Forms.Button();
            this.numCoupleLFUB = new System.Windows.Forms.NumericUpDown();
            this.label23 = new System.Windows.Forms.Label();
            this.tckCoupleLFUB = new System.Windows.Forms.TrackBar();
            this.numCoupleUFLB = new System.Windows.Forms.NumericUpDown();
            this.label24 = new System.Windows.Forms.Label();
            this.tckCoupleUFLB = new System.Windows.Forms.TrackBar();
            this.numCoupleLower = new System.Windows.Forms.NumericUpDown();
            this.label22 = new System.Windows.Forms.Label();
            this.tckCoupleLower = new System.Windows.Forms.TrackBar();
            this.numCoupleUpper = new System.Windows.Forms.NumericUpDown();
            this.label21 = new System.Windows.Forms.Label();
            this.tckCoupleUpper = new System.Windows.Forms.TrackBar();
            this.label20 = new System.Windows.Forms.Label();
            this.chkBackFDDebug = new System.Windows.Forms.CheckBox();
            this.chkBackADDebug = new System.Windows.Forms.CheckBox();
            this.chkBackFDEnable = new System.Windows.Forms.CheckBox();
            this.chkBackADEnable = new System.Windows.Forms.CheckBox();
            this.chkBackEnable = new System.Windows.Forms.CheckBox();
            this.label13 = new System.Windows.Forms.Label();
            this.lblBackFAnalogTune = new System.Windows.Forms.Label();
            this.lblBackQAnalogTune = new System.Windows.Forms.Label();
            this.numBackFAnalogTune = new System.Windows.Forms.NumericUpDown();
            this.numBackFDigitalTune = new System.Windows.Forms.NumericUpDown();
            this.numBackQAnalogTune = new System.Windows.Forms.NumericUpDown();
            this.numBackQDigitalTune = new System.Windows.Forms.NumericUpDown();
            this.tckBackFAnalogTune = new System.Windows.Forms.TrackBar();
            this.label16 = new System.Windows.Forms.Label();
            this.label17 = new System.Windows.Forms.Label();
            this.label18 = new System.Windows.Forms.Label();
            this.label19 = new System.Windows.Forms.Label();
            this.tckBackQDigitalTune = new System.Windows.Forms.TrackBar();
            this.tckBackQAnalogTune = new System.Windows.Forms.TrackBar();
            this.tckBackFDigitalTune = new System.Windows.Forms.TrackBar();
            this.chkFrontFDDebug = new System.Windows.Forms.CheckBox();
            this.chkFrontADDebug = new System.Windows.Forms.CheckBox();
            this.chkFrontFDEnable = new System.Windows.Forms.CheckBox();
            this.chkFrontADEnable = new System.Windows.Forms.CheckBox();
            this.chkFrontEnable = new System.Windows.Forms.CheckBox();
            this.label12 = new System.Windows.Forms.Label();
            this.lblFrontFAnalogTune = new System.Windows.Forms.Label();
            this.lblFrontQAnalogTune = new System.Windows.Forms.Label();
            this.numFrontFAnalogTune = new System.Windows.Forms.NumericUpDown();
            this.numFrontFDigitalTune = new System.Windows.Forms.NumericUpDown();
            this.numFrontQAnalogTune = new System.Windows.Forms.NumericUpDown();
            this.numFrontQDigitalTune = new System.Windows.Forms.NumericUpDown();
            this.tckFrontFAnalogTune = new System.Windows.Forms.TrackBar();
```

```
            this.label9 = new System.Windows.Forms.Label();
            this.label8 = new System.Windows.Forms.Label();
            this.label7 = new System.Windows.Forms.Label();
            this.label6 = new System.Windows.Forms.Label();
            this.tckFrontQDigitalTune = new System.Windows.Forms.TrackBar();
            this.tckFrontQAnalogTune = new System.Windows.Forms.TrackBar();
            this.tckFrontFDigitalTune = new System.Windows.Forms.TrackBar();
            this.groupFilterSettings.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(this.numCoupleLFUB)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckCoupleLFUB)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numCoupleUFLB)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckCoupleUFLB)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numCoupleLower)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckCoupleLower)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numCoupleUpper)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckCoupleUpper)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackFAnalogTune)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackFDigitalTune)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackQAnalogTune)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackQDigitalTune)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckBackFAnalogTune)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckBackQDigitalTune)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckBackQAnalogTune)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckBackFDigitalTune)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontFAnalogTune)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontFDigitalTune)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontQAnalogTune)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontQDigitalTune)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckFrontFAnalogTune)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckFrontQDigitalTune)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckFrontQAnalogTune)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckFrontFDigitalTune)).BeginInit();
            this.SuspendLayout();
            //
            // groupFilterSettings
            //
            this.groupFilterSettings.Controls.Add(this.chkRFSW);
            this.groupFilterSettings.Controls.Add(this.chkPrintOptions);
            this.groupFilterSettings.Controls.Add(this.btnFilterProgram);
            this.groupFilterSettings.Controls.Add(this.btnFilterLoad);
            this.groupFilterSettings.Controls.Add(this.btnFilterSave);
            this.groupFilterSettings.Controls.Add(this.btnFilterReset);
            this.groupFilterSettings.Controls.Add(this.numCoupleLFUB);
            this.groupFilterSettings.Controls.Add(this.label23);
            this.groupFilterSettings.Controls.Add(this.tckCoupleLFUB);
            this.groupFilterSettings.Controls.Add(this.numCoupleUFLB);
            this.groupFilterSettings.Controls.Add(this.label24);
            this.groupFilterSettings.Controls.Add(this.tckCoupleUFLB);
            this.groupFilterSettings.Controls.Add(this.numCoupleLower);
            this.groupFilterSettings.Controls.Add(this.label22);
            this.groupFilterSettings.Controls.Add(this.tckCoupleLower);
            this.groupFilterSettings.Controls.Add(this.numCoupleUpper);
            this.groupFilterSettings.Controls.Add(this.label21);
            this.groupFilterSettings.Controls.Add(this.tckCoupleUpper);
            this.groupFilterSettings.Controls.Add(this.label20);
            this.groupFilterSettings.Controls.Add(this.chkBackFDDebug);
            this.groupFilterSettings.Controls.Add(this.chkBackADDebug);
            this.groupFilterSettings.Controls.Add(this.chkBackFDEnable);
            this.groupFilterSettings.Controls.Add(this.chkBackADEnable);
            this.groupFilterSettings.Controls.Add(this.chkBackEnable);
            this.groupFilterSettings.Controls.Add(this.label13);
            this.groupFilterSettings.Controls.Add(this.lblBackFAnalogTune);
            this.groupFilterSettings.Controls.Add(this.lblBackQAnalogTune);
            this.groupFilterSettings.Controls.Add(this.numBackFAnalogTune);
            this.groupFilterSettings.Controls.Add(this.numBackFDigitalTune);
            this.groupFilterSettings.Controls.Add(this.numBackQAnalogTune);
            this.groupFilterSettings.Controls.Add(this.numBackQDigitalTune);
            this.groupFilterSettings.Controls.Add(this.tckBackFAnalogTune);
            this.groupFilterSettings.Controls.Add(this.label16);
            this.groupFilterSettings.Controls.Add(this.label17);
            this.groupFilterSettings.Controls.Add(this.label18);
            this.groupFilterSettings.Controls.Add(this.label19);
            this.groupFilterSettings.Controls.Add(this.tckBackQDigitalTune);
            this.groupFilterSettings.Controls.Add(this.tckBackQAnalogTune);
            this.groupFilterSettings.Controls.Add(this.tckBackFDigitalTune);
            this.groupFilterSettings.Controls.Add(this.chkFrontFDDebug);
            this.groupFilterSettings.Controls.Add(this.chkFrontADDebug);
            this.groupFilterSettings.Controls.Add(this.chkFrontFDEnable);
            this.groupFilterSettings.Controls.Add(this.chkFrontADEnable);
            this.groupFilterSettings.Controls.Add(this.chkFrontEnable);
            this.groupFilterSettings.Controls.Add(this.label12);
            this.groupFilterSettings.Controls.Add(this.lblFrontFAnalogTune);
            this.groupFilterSettings.Controls.Add(this.lblFrontQAnalogTune);
            this.groupFilterSettings.Controls.Add(this.numFrontFAnalogTune);
            this.groupFilterSettings.Controls.Add(this.numFrontFDigitalTune);
            this.groupFilterSettings.Controls.Add(this.numFrontQAnalogTune);
            this.groupFilterSettings.Controls.Add(this.numFrontQDigitalTune);
            this.groupFilterSettings.Controls.Add(this.tckFrontFAnalogTune);
            this.groupFilterSettings.Controls.Add(this.label9);
            this.groupFilterSettings.Controls.Add(this.label8);
            this.groupFilterSettings.Controls.Add(this.label7);
            this.groupFilterSettings.Controls.Add(this.label6);
            this.groupFilterSettings.Controls.Add(this.tckFrontQDigitalTune);
            this.groupFilterSettings.Controls.Add(this.tckFrontQAnalogTune);
            this.groupFilterSettings.Controls.Add(this.tckFrontFDigitalTune);
            this.groupFilterSettings.Font = new System.Drawing.Font("Cambria", 12F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.groupFilterSettings.Location = new System.Drawing.Point(12, 12);
            this.groupFilterSettings.Name = "groupFilterSettings";
            this.groupFilterSettings.Size = new System.Drawing.Size(552, 502);
            this.groupFilterSettings.TabIndex = 4;
            this.groupFilterSettings.TabStop = false;
            this.groupFilterSettings.Text = "Filter Settings";
            //
            // chkRFSW
            //
            this.chkRFSW.AutoSize = true;
            this.chkRFSW.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.chkRFSW.Location = new System.Drawing.Point(435, 33);
            this.chkRFSW.Name = "chkRFSW";
            this.chkRFSW.Size = new System.Drawing.Size(84, 17);
            this.chkRFSW.TabIndex = 54;
            this.chkRFSW.Text = "RF Input On";
            this.chkRFSW.UseVisualStyleBackColor = true;
            this.chkRFSW.CheckedChanged += new System.EventHandler(this.chkRFSW_CheckedChanged);
```

```
//
// chkPrintOptions
//
this.chkPrintOptions.AutoSize = true;
this.chkPrintOptions.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.chkPrintOptions.Location = new System.Drawing.Point(435, 16);
this.chkPrintOptions.Name = "chkPrintOptions";
this.chkPrintOptions.Size = new System.Drawing.Size(113, 17);
this.chkPrintOptions.TabIndex = 32;
this.chkPrintOptions.Text = "Print Filter Settings";
this.chkPrintOptions.UseVisualStyleBackColor = true;
this.chkPrintOptions.CheckedChanged += new System.EventHandler(this.chkPrintOptions_CheckedChanged);
//
// btnFilterProgram
//
this.btnFilterProgram.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.btnFilterProgram.Location = new System.Drawing.Point(409, 468);
this.btnFilterProgram.Name = "btnFilterProgram";
this.btnFilterProgram.Size = new System.Drawing.Size(121, 23);
this.btnFilterProgram.TabIndex = 53;
this.btnFilterProgram.Text = "Program";
this.btnFilterProgram.UseVisualStyleBackColor = true;
this.btnFilterProgram.Click += new System.EventHandler(this.btnFilterProgram_Click);
//
// btnFilterLoad
//
this.btnFilterLoad.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.btnFilterLoad.Location = new System.Drawing.Point(280, 468);
this.btnFilterLoad.Name = "btnFilterLoad";
this.btnFilterLoad.Size = new System.Drawing.Size(121, 23);
this.btnFilterLoad.TabIndex = 52;
this.btnFilterLoad.Text = "Load Filter Settings";
this.btnFilterLoad.UseVisualStyleBackColor = true;
this.btnFilterLoad.Click += new System.EventHandler(this.btnFilterLoad_Click);
//
// btnFilterSave
//
this.btnFilterSave.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.btnFilterSave.Location = new System.Drawing.Point(151, 468);
this.btnFilterSave.Name = "btnFilterSave";
this.btnFilterSave.Size = new System.Drawing.Size(121, 23);
this.btnFilterSave.TabIndex = 51;
this.btnFilterSave.Text = "Save Filter Settings";
this.btnFilterSave.UseVisualStyleBackColor = true;
this.btnFilterSave.Click += new System.EventHandler(this.btnFilterSave_Click);
//
// btnFilterReset
//
this.btnFilterReset.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.btnFilterReset.Location = new System.Drawing.Point(22, 468);
this.btnFilterReset.Name = "btnFilterReset";
this.btnFilterReset.Size = new System.Drawing.Size(121, 23);
this.btnFilterReset.TabIndex = 50;
this.btnFilterReset.Text = "Reset Filter Settings";
this.btnFilterReset.UseVisualStyleBackColor = true;
this.btnFilterReset.Click += new System.EventHandler(this.btnFilterReset_Click);
//
// numCoupleLFUB
//
this.numCoupleLFUB.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numCoupleLFUB.Location = new System.Drawing.Point(469, 415);
this.numCoupleLFUB.Name = "numCoupleLFUB";
this.numCoupleLFUB.Size = new System.Drawing.Size(61, 20);
this.numCoupleLFUB.TabIndex = 49;
this.numCoupleLFUB.ValueChanged += new System.EventHandler(this.numCoupleLFUB_ValueChanged);
//
// label23
//
this.label23.AutoSize = true;
this.label23.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.label23.Location = new System.Drawing.Point(298, 399);
this.label23.Name = "label23";
this.label23.Size = new System.Drawing.Size(123, 13);
this.label23.TabIndex = 0;
this.label23.Text = "Capacitive LFUB Tuning";
//
// tckCoupleLFUB
//
this.tckCoupleLFUB.Location = new System.Drawing.Point(282, 415);
this.tckCoupleLFUB.Name = "tckCoupleLFUB";
this.tckCoupleLFUB.Size = new System.Drawing.Size(181, 45);
this.tckCoupleLFUB.TabIndex = 48;
this.tckCoupleLFUB.Scroll += new System.EventHandler(this.tckCoupleLFUB_Scroll);
//
// numCoupleUFLB
//
this.numCoupleUFLB.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numCoupleUFLB.Location = new System.Drawing.Point(469, 364);
this.numCoupleUFLB.Name = "numCoupleUFLB";
this.numCoupleUFLB.Size = new System.Drawing.Size(61, 20);
this.numCoupleUFLB.TabIndex = 47;
this.numCoupleUFLB.ValueChanged += new System.EventHandler(this.numCoupleUFLB_ValueChanged);
//
// label24
//
this.label24.AutoSize = true;
this.label24.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.label24.Location = new System.Drawing.Point(298, 348);
this.label24.Name = "label24";
this.label24.Size = new System.Drawing.Size(123, 13);
this.label24.TabIndex = 0;
this.label24.Text = "Capacitive UFLB Tuning";
//
// tckCoupleUFLB
//
this.tckCoupleUFLB.Location = new System.Drawing.Point(282, 364);
this.tckCoupleUFLB.Name = "tckCoupleUFLB";
this.tckCoupleUFLB.Size = new System.Drawing.Size(181, 45);
this.tckCoupleUFLB.TabIndex = 46;
this.tckCoupleUFLB.Scroll += new System.EventHandler(this.tckCoupleUFLB_Scroll);
//
// numCoupleLower
//
this.numCoupleLower.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numCoupleLower.Location = new System.Drawing.Point(201, 417);
this.numCoupleLower.Name = "numCoupleLower";
this.numCoupleLower.Size = new System.Drawing.Size(61, 20);
this.numCoupleLower.TabIndex = 45;
```

```
this.numCoupleLower.ValueChanged += new System.EventHandler(this.numCoupleLower_ValueChanged);
//
// label22
//
this.label22.AutoSize = true;
this.label22.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.label22.Location = new System.Drawing.Point(30, 401);
this.label22.Name = "label22";
this.label22.Size = new System.Drawing.Size(125, 13);
this.label22.TabIndex = 0;
this.label22.Text = "Capacitive Lower Tuning";
//
// tckCoupleLower
//
this.tckCoupleLower.Location = new System.Drawing.Point(14, 417);
this.tckCoupleLower.Name = "tckCoupleLower";
this.tckCoupleLower.Size = new System.Drawing.Size(181, 45);
this.tckCoupleLower.TabIndex = 44;
this.tckCoupleLower.Scroll += new System.EventHandler(this.tckCoupleLower_Scroll);
//
// numCoupleUpper
//
this.numCoupleUpper.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numCoupleUpper.Location = new System.Drawing.Point(201, 366);
this.numCoupleUpper.Name = "numCoupleUpper";
this.numCoupleUpper.Size = new System.Drawing.Size(61, 20);
this.numCoupleUpper.TabIndex = 43;
this.numCoupleUpper.ValueChanged += new System.EventHandler(this.numCoupleUpper_ValueChanged);
//
// label21
//
this.label21.AutoSize = true;
this.label21.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.label21.Location = new System.Drawing.Point(30, 350);
this.label21.Name = "label21";
this.label21.Size = new System.Drawing.Size(125, 13);
this.label21.TabIndex = 0;
this.label21.Text = "Capacitive Upper Tuning";
//
// tckCoupleUpper
//
this.tckCoupleUpper.Location = new System.Drawing.Point(14, 366);
this.tckCoupleUpper.Name = "tckCoupleUpper";
this.tckCoupleUpper.Size = new System.Drawing.Size(181, 45);
this.tckCoupleUpper.TabIndex = 42;
this.tckCoupleUpper.Scroll += new System.EventHandler(this.tckCoupleUpper_Scroll);
//
// label20
//
this.label20.AutoSize = true;
this.label20.Font = new System.Drawing.Font("Cambria", 9.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.label20.Location = new System.Drawing.Point(19, 334);
this.label20.Name = "label20";
this.label20.Size = new System.Drawing.Size(60, 15);
this.label20.TabIndex = 0;
this.label20.Text = "Coupling";
//
// chkBackFDDebug
//
this.chkBackFDDebug.AutoSize = true;
this.chkBackFDDebug.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.chkBackFDDebug.Location = new System.Drawing.Point(469, 97);
this.chkBackFDDebug.Name = "chkBackFDDebug";
this.chkBackFDDebug.Size = new System.Drawing.Size(58, 17);
this.chkBackFDDebug.TabIndex = 33;
this.chkBackFDDebug.Text = "Debug";
this.chkBackFDDebug.UseVisualStyleBackColor = true;
this.chkBackFDDebug.CheckedChanged += new System.EventHandler(this.chkBackFDDebug_CheckedChanged);
//
// chkBackADDebug
//
this.chkBackADDebug.AutoSize = true;
this.chkBackADDebug.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.chkBackADDebug.Location = new System.Drawing.Point(469, 75);
this.chkBackADDebug.Name = "chkBackADDebug";
this.chkBackADDebug.Size = new System.Drawing.Size(58, 17);
this.chkBackADDebug.TabIndex = 31;
this.chkBackADDebug.Text = "Debug";
this.chkBackADDebug.UseVisualStyleBackColor = true;
this.chkBackADDebug.CheckedChanged += new System.EventHandler(this.chkBackADDebug_CheckedChanged);
//
// chkBackFDEnable
//
this.chkBackFDEnable.AutoSize = true;
this.chkBackFDEnable.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.chkBackFDEnable.Location = new System.Drawing.Point(282, 96);
this.chkBackFDEnable.Name = "chkBackFDEnable";
this.chkBackFDEnable.Size = new System.Drawing.Size(148, 17);
this.chkBackFDEnable.TabIndex = 32;
this.chkBackFDEnable.Text = "Frequency-Divider Enable";
this.chkBackFDEnable.UseVisualStyleBackColor = true;
this.chkBackFDEnable.CheckedChanged += new System.EventHandler(this.chkBackFDEnable_CheckedChanged);
//
// chkBackADEnable
//
this.chkBackADEnable.AutoSize = true;
this.chkBackADEnable.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.chkBackADEnable.Location = new System.Drawing.Point(282, 74);
this.chkBackADEnable.Name = "chkBackADEnable";
this.chkBackADEnable.Size = new System.Drawing.Size(152, 17);
this.chkBackADEnable.TabIndex = 30;
this.chkBackADEnable.Text = "Amplitude Detector Enable";
this.chkBackADEnable.UseVisualStyleBackColor = true;
this.chkBackADEnable.CheckedChanged += new System.EventHandler(this.chkBackADEnable_CheckedChanged);
//
// chkBackEnable
//
this.chkBackEnable.AutoSize = true;
this.chkBackEnable.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.chkBackEnable.Location = new System.Drawing.Point(282, 51);
this.chkBackEnable.Name = "chkBackEnable";
this.chkBackEnable.Size = new System.Drawing.Size(109, 17);
this.chkBackEnable.TabIndex = 29;
this.chkBackEnable.Text = "Back-End Enable";
this.chkBackEnable.UseVisualStyleBackColor = true;
this.chkBackEnable.CheckedChanged += new System.EventHandler(this.chkBackEnable_CheckedChanged);
//
// label13
```

```
//
this.label13.AutoSize = true;
this.label13.Font = new System.Drawing.Font("Cambria", 9.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.label13.Location = new System.Drawing.Point(279, 33);
this.label13.Name = "label13";
this.label13.Size = new System.Drawing.Size(64, 15);
this.label13.TabIndex = 0;
this.label13.Text = "Back-End";
//
// lblBackFAnalogTune
//
this.lblBackFAnalogTune.AutoSize = true;
this.lblBackFAnalogTune.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.lblBackFAnalogTune.Location = new System.Drawing.Point(469, 267);
this.lblBackFAnalogTune.Name = "lblBackFAnalogTune";
this.lblBackFAnalogTune.Size = new System.Drawing.Size(31, 13);
this.lblBackFAnalogTune.TabIndex = 0;
this.lblBackFAnalogTune.Text = "0 mV";
//
// lblBackQAnalogTune
//
this.lblBackQAnalogTune.AutoSize = true;
this.lblBackQAnalogTune.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.lblBackQAnalogTune.Location = new System.Drawing.Point(469, 168);
this.lblBackQAnalogTune.Name = "lblBackQAnalogTune";
this.lblBackQAnalogTune.Size = new System.Drawing.Size(31, 13);
this.lblBackQAnalogTune.TabIndex = 0;
this.lblBackQAnalogTune.Text = "0 mV";
//
// numBackFAnalogTune
//
this.numBackFAnalogTune.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numBackFAnalogTune.Location = new System.Drawing.Point(469, 286);
this.numBackFAnalogTune.Name = "numBackFAnalogTune";
this.numBackFAnalogTune.Size = new System.Drawing.Size(61, 20);
this.numBackFAnalogTune.TabIndex = 41;
this.numBackFAnalogTune.ValueChanged += new System.EventHandler(this.numBackFAnalogTune_ValueChanged);
//
// numBackFDigitalTune
//
this.numBackFDigitalTune.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numBackFDigitalTune.Location = new System.Drawing.Point(469, 235);
this.numBackFDigitalTune.Name = "numBackFDigitalTune";
this.numBackFDigitalTune.Size = new System.Drawing.Size(61, 20);
this.numBackFDigitalTune.TabIndex = 39;
this.numBackFDigitalTune.ValueChanged += new System.EventHandler(this.numBackFDigitalTune_ValueChanged);
//
// numBackQAnalogTune
//
this.numBackQAnalogTune.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numBackQAnalogTune.Location = new System.Drawing.Point(469, 187);
this.numBackQAnalogTune.Name = "numBackQAnalogTune";
this.numBackQAnalogTune.Size = new System.Drawing.Size(61, 20);
this.numBackQAnalogTune.TabIndex = 37;
this.numBackQAnalogTune.ValueChanged += new System.EventHandler(this.numBackQAnalogTune_ValueChanged);
//
// numBackQDigitalTune
//
this.numBackQDigitalTune.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numBackQDigitalTune.Location = new System.Drawing.Point(469, 136);
this.numBackQDigitalTune.Name = "numBackQDigitalTune";
this.numBackQDigitalTune.Size = new System.Drawing.Size(61, 20);
this.numBackQDigitalTune.TabIndex = 35;
this.numBackQDigitalTune.ValueChanged += new System.EventHandler(this.numBackQDigitalTune_ValueChanged);
//
// tckBackFAnalogTune
//
this.tckBackFAnalogTune.Location = new System.Drawing.Point(282, 286);
this.tckBackFAnalogTune.Name = "tckBackFAnalogTune";
this.tckBackFAnalogTune.Size = new System.Drawing.Size(181, 45);
this.tckBackFAnalogTune.TabIndex = 40;
this.tckBackFAnalogTune.Scroll += new System.EventHandler(this.tckBackFAnalogTune_Scroll);
//
// label16
//
this.label16.AutoSize = true;
this.label16.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.label16.Location = new System.Drawing.Point(298, 267);
this.label16.Name = "label16";
this.label16.Size = new System.Drawing.Size(129, 13);
this.label16.TabIndex = 0;
this.label16.Text = "Frequency Analog Tuning";
//
// label17
//
this.label17.AutoSize = true;
this.label17.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.label17.Location = new System.Drawing.Point(298, 219);
this.label17.Name = "label17";
this.label17.Size = new System.Drawing.Size(125, 13);
this.label17.TabIndex = 0;
this.label17.Text = "Frequency Digital Tuning";
//
// label18
//
this.label18.AutoSize = true;
this.label18.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.label18.Location = new System.Drawing.Point(298, 171);
this.label18.Name = "label18";
this.label18.Size = new System.Drawing.Size(156, 13);
this.label18.TabIndex = 0;
this.label18.Text = "Q-Enhancement Analog Tuning";
//
// label19
//
this.label19.AutoSize = true;
this.label19.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.label19.Location = new System.Drawing.Point(298, 120);
this.label19.Name = "label19";
this.label19.Size = new System.Drawing.Size(152, 13);
this.label19.TabIndex = 0;
this.label19.Text = "Q-Enhancement Digital Tuning";
//
// tckBackQDigitalTune
//
this.tckBackQDigitalTune.Location = new System.Drawing.Point(282, 136);
this.tckBackQDigitalTune.Name = "tckBackQDigitalTune";
this.tckBackQDigitalTune.Size = new System.Drawing.Size(181, 45);
```

```
this.tckBackQDigitalTune.TabIndex = 34;
this.tckBackQDigitalTune.Scroll += new System.EventHandler(this.tckBackQDigitalTune_Scroll);
//
// tckBackQAnalogTune
//
this.tckBackQAnalogTune.Location = new System.Drawing.Point(282, 187);
this.tckBackQAnalogTune.Name = "tckBackQAnalogTune";
this.tckBackQAnalogTune.Size = new System.Drawing.Size(181, 45);
this.tckBackQAnalogTune.TabIndex = 36;
this.tckBackQAnalogTune.Scroll += new System.EventHandler(this.tckBackQAnalogTune_Scroll);
//
// tckBackFDigitalTune
//
this.tckBackFDigitalTune.Location = new System.Drawing.Point(282, 235);
this.tckBackFDigitalTune.Name = "tckBackFDigitalTune";
this.tckBackFDigitalTune.Size = new System.Drawing.Size(181, 45);
this.tckBackFDigitalTune.TabIndex = 38;
this.tckBackFDigitalTune.Scroll += new System.EventHandler(this.tckBackFDigitalTune_Scroll);
//
// chkFrontFDDebug
//
this.chkFrontFDDebug.AutoSize = true;
this.chkFrontFDDebug.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.chkFrontFDDebug.Location = new System.Drawing.Point(201, 96);
this.chkFrontFDDebug.Name = "chkFrontFDDebug";
this.chkFrontFDDebug.Size = new System.Drawing.Size(58, 17);
this.chkFrontFDDebug.TabIndex = 20;
this.chkFrontFDDebug.Text = "Debug";
this.chkFrontFDDebug.UseVisualStyleBackColor = true;
this.chkFrontFDDebug.CheckedChanged += new System.EventHandler(this.chkFrontFDDebug_CheckedChanged);
//
// chkFrontADDebug
//
this.chkFrontADDebug.AutoSize = true;
this.chkFrontADDebug.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.chkFrontADDebug.Location = new System.Drawing.Point(201, 74);
this.chkFrontADDebug.Name = "chkFrontADDebug";
this.chkFrontADDebug.Size = new System.Drawing.Size(58, 17);
this.chkFrontADDebug.TabIndex = 18;
this.chkFrontADDebug.Text = "Debug";
this.chkFrontADDebug.UseVisualStyleBackColor = true;
this.chkFrontADDebug.CheckedChanged += new System.EventHandler(this.chkFrontADDebug_CheckedChanged);
//
// chkFrontFDEnable
//
this.chkFrontFDEnable.AutoSize = true;
this.chkFrontFDEnable.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.chkFrontFDEnable.Location = new System.Drawing.Point(22, 96);
this.chkFrontFDEnable.Name = "chkFrontFDEnable";
this.chkFrontFDEnable.Size = new System.Drawing.Size(148, 17);
this.chkFrontFDEnable.TabIndex = 19;
this.chkFrontFDEnable.Text = "Frequency-Divider Enable";
this.chkFrontFDEnable.UseVisualStyleBackColor = true;
this.chkFrontFDEnable.CheckedChanged += new System.EventHandler(this.chkFrontFDEnable_CheckedChanged);
//
// chkFrontADEnable
//
this.chkFrontADEnable.AutoSize = true;
this.chkFrontADEnable.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.chkFrontADEnable.Location = new System.Drawing.Point(22, 74);
this.chkFrontADEnable.Name = "chkFrontADEnable";
this.chkFrontADEnable.Size = new System.Drawing.Size(152, 17);
this.chkFrontADEnable.TabIndex = 17;
this.chkFrontADEnable.Text = "Amplitude Detector Enable";
this.chkFrontADEnable.UseVisualStyleBackColor = true;
this.chkFrontADEnable.CheckedChanged += new System.EventHandler(this.chkFrontADEnable_CheckedChanged);
//
// chkFrontEnable
//
this.chkFrontEnable.AutoSize = true;
this.chkFrontEnable.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.chkFrontEnable.Location = new System.Drawing.Point(22, 51);
this.chkFrontEnable.Name = "chkFrontEnable";
this.chkFrontEnable.Size = new System.Drawing.Size(108, 17);
this.chkFrontEnable.TabIndex = 16;
this.chkFrontEnable.Text = "Front-End Enable";
this.chkFrontEnable.UseVisualStyleBackColor = true;
this.chkFrontEnable.CheckedChanged += new System.EventHandler(this.chkFrontEnable_CheckedChanged);
//
// label12
//
this.label12.AutoSize = true;
this.label12.Font = new System.Drawing.Font("Cambria", 9.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.label12.Location = new System.Drawing.Point(19, 31);
this.label12.Name = "label12";
this.label12.Size = new System.Drawing.Size(68, 15);
this.label12.TabIndex = 0;
this.label12.Text = "Front-End";
//
// lblFrontFAnalogTune
//
this.lblFrontFAnalogTune.AutoSize = true;
this.lblFrontFAnalogTune.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.lblFrontFAnalogTune.Location = new System.Drawing.Point(201, 267);
this.lblFrontFAnalogTune.Name = "lblFrontFAnalogTune";
this.lblFrontFAnalogTune.Size = new System.Drawing.Size(31, 13);
this.lblFrontFAnalogTune.TabIndex = 0;
this.lblFrontFAnalogTune.Text = "0 mV";
//
// lblFrontQAnalogTune
//
this.lblFrontQAnalogTune.AutoSize = true;
this.lblFrontQAnalogTune.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.lblFrontQAnalogTune.Location = new System.Drawing.Point(201, 168);
this.lblFrontQAnalogTune.Name = "lblFrontQAnalogTune";
this.lblFrontQAnalogTune.Size = new System.Drawing.Size(31, 13);
this.lblFrontQAnalogTune.TabIndex = 0;
this.lblFrontQAnalogTune.Text = "0 mV";
//
// numFrontFAnalogTune
//
this.numFrontFAnalogTune.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numFrontFAnalogTune.Location = new System.Drawing.Point(201, 286);
this.numFrontFAnalogTune.Name = "numFrontFAnalogTune";
this.numFrontFAnalogTune.Size = new System.Drawing.Size(61, 20);
this.numFrontFAnalogTune.TabIndex = 28;
this.numFrontFAnalogTune.ValueChanged += new System.EventHandler(this.numFrontFAnalogTune_ValueChanged);
//
```

```
// numFrontFDigitalTune
//
this.numFrontFDigitalTune.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numFrontFDigitalTune.Location = new System.Drawing.Point(201, 235);
this.numFrontFDigitalTune.Name = "numFrontFDigitalTune";
this.numFrontFDigitalTune.Size = new System.Drawing.Size(61, 20);
this.numFrontFDigitalTune.TabIndex = 26;
this.numFrontFDigitalTune.ValueChanged += new System.EventHandler(this.numFrontFDigitalTune_ValueChanged);
//
// numFrontQAnalogTune
//
this.numFrontQAnalogTune.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numFrontQAnalogTune.Location = new System.Drawing.Point(201, 187);
this.numFrontQAnalogTune.Name = "numFrontQAnalogTune";
this.numFrontQAnalogTune.Size = new System.Drawing.Size(61, 20);
this.numFrontQAnalogTune.TabIndex = 24;
this.numFrontQAnalogTune.ValueChanged += new System.EventHandler(this.numFrontQAnalogTune_ValueChanged);
//
// numFrontQDigitalTune
//
this.numFrontQDigitalTune.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numFrontQDigitalTune.Location = new System.Drawing.Point(201, 136);
this.numFrontQDigitalTune.Name = "numFrontQDigitalTune";
this.numFrontQDigitalTune.Size = new System.Drawing.Size(61, 20);
this.numFrontQDigitalTune.TabIndex = 22;
this.numFrontQDigitalTune.ValueChanged += new System.EventHandler(this.numFrontQDigitalTune_ValueChanged);
//
// tckFrontFAnalogTune
//
this.tckFrontFAnalogTune.Location = new System.Drawing.Point(14, 286);
this.tckFrontFAnalogTune.Name = "tckFrontFAnalogTune";
this.tckFrontFAnalogTune.Size = new System.Drawing.Size(181, 45);
this.tckFrontFAnalogTune.TabIndex = 27;
this.tckFrontFAnalogTune.Scroll += new System.EventHandler(this.tckFrontFAnalogTune_Scroll);
//
// label9
//
this.label9.AutoSize = true;
this.label9.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.label9.Location = new System.Drawing.Point(30, 267);
this.label9.Name = "label9";
this.label9.Size = new System.Drawing.Size(129, 13);
this.label9.TabIndex = 0;
this.label9.Text = "Frequency Analog Tuning";
//
// label8
//
this.label8.AutoSize = true;
this.label8.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.label8.Location = new System.Drawing.Point(30, 219);
this.label8.Name = "label8";
this.label8.Size = new System.Drawing.Size(125, 13);
this.label8.TabIndex = 0;
this.label8.Text = "Frequency Digital Tuning";
//
// label7
//
this.label7.AutoSize = true;
this.label7.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.label7.Location = new System.Drawing.Point(30, 171);
this.label7.Name = "label7";
this.label7.Size = new System.Drawing.Size(156, 13);
this.label7.TabIndex = 0;
this.label7.Text = "Q-Enhancement Analog Tuning";
//
// label6
//
this.label6.AutoSize = true;
this.label6.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.label6.Location = new System.Drawing.Point(30, 120);
this.label6.Name = "label6";
this.label6.Size = new System.Drawing.Size(152, 13);
this.label6.TabIndex = 0;
this.label6.Text = "Q-Enhancement Digital Tuning";
//
// tckFrontQDigitalTune
//
this.tckFrontQDigitalTune.Location = new System.Drawing.Point(14, 136);
this.tckFrontQDigitalTune.Name = "tckFrontQDigitalTune";
this.tckFrontQDigitalTune.Size = new System.Drawing.Size(181, 45);
this.tckFrontQDigitalTune.TabIndex = 21;
this.tckFrontQDigitalTune.Scroll += new System.EventHandler(this.tckFrontQDigitalTune_Scroll);
//
// tckFrontQAnalogTune
//
this.tckFrontQAnalogTune.Location = new System.Drawing.Point(14, 187);
this.tckFrontQAnalogTune.Name = "tckFrontQAnalogTune";
this.tckFrontQAnalogTune.Size = new System.Drawing.Size(181, 45);
this.tckFrontQAnalogTune.TabIndex = 23;
this.tckFrontQAnalogTune.Scroll += new System.EventHandler(this.tckFrontQAnalogTune_Scroll);
//
// tckFrontFDigitalTune
//
this.tckFrontFDigitalTune.Location = new System.Drawing.Point(14, 235);
this.tckFrontFDigitalTune.Name = "tckFrontFDigitalTune";
this.tckFrontFDigitalTune.Size = new System.Drawing.Size(181, 45);
this.tckFrontFDigitalTune.TabIndex = 25;
this.tckFrontFDigitalTune.Scroll += new System.EventHandler(this.tckFrontFDigitalTune_Scroll);
//
// frmFilterControls
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(577, 526);
this.Controls.Add(this.groupFilterSettings);
this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
this.MaximizeBox = false;
this.Name = "frmFilterControls";
this.Text = "Filter Settings";
this.groupFilterSettings.ResumeLayout(false);
this.groupFilterSettings.PerformLayout();
((System.ComponentModel.ISupportInitialize)(this.numCoupleLFUB)).EndInit();
((System.ComponentModel.ISupportInitialize)(this.tckCoupleLFUB)).EndInit();
((System.ComponentModel.ISupportInitialize)(this.numCoupleUFLB)).EndInit();
((System.ComponentModel.ISupportInitialize)(this.tckCoupleUFLB)).EndInit();
((System.ComponentModel.ISupportInitialize)(this.numCoupleLower)).EndInit();
((System.ComponentModel.ISupportInitialize)(this.tckCoupleLower)).EndInit();
((System.ComponentModel.ISupportInitialize)(this.numCoupleUpper)).EndInit();
```

```
            ((System.ComponentModel.ISupportInitialize)(this.tckCoupleUpper)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackFAnalogTune)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackFDigitalTune)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackQAnalogTune)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackQDigitalTune)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckBackFAnalogTune)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckBackQDigitalTune)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckBackQAnalogTune)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckBackFDigitalTune)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontFAnalogTune)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontFDigitalTune)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontQAnalogTune)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontQDigitalTune)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckFrontFAnalogTune)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckFrontQDigitalTune)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckFrontQAnalogTune)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.tckFrontFDigitalTune)).EndInit();
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.GroupBox groupFilterSettings;
        private System.Windows.Forms.NumericUpDown numCoupleLFUB;
        private System.Windows.Forms.Label label23;
        private System.Windows.Forms.TrackBar tckCoupleLFUB;
        private System.Windows.Forms.NumericUpDown numCoupleUFLB;
        private System.Windows.Forms.Label label24;
        private System.Windows.Forms.TrackBar tckCoupleUFLB;
        private System.Windows.Forms.NumericUpDown numCoupleLower;
        private System.Windows.Forms.Label label22;
        private System.Windows.Forms.TrackBar tckCoupleLower;
        private System.Windows.Forms.NumericUpDown numCoupleUpper;
        private System.Windows.Forms.Label label21;
        private System.Windows.Forms.TrackBar tckCoupleUpper;
        private System.Windows.Forms.Label label20;
        private System.Windows.Forms.CheckBox chkBackFDDebug;
        private System.Windows.Forms.CheckBox chkBackADDebug;
        private System.Windows.Forms.CheckBox chkBackFDEnable;
        private System.Windows.Forms.CheckBox chkBackADEnable;
        private System.Windows.Forms.CheckBox chkBackEnable;
        private System.Windows.Forms.Label label13;
        private System.Windows.Forms.Label lblBackFAnalogTune;
        private System.Windows.Forms.Label lblBackQAnalogTune;
        private System.Windows.Forms.NumericUpDown numBackFAnalogTune;
        private System.Windows.Forms.NumericUpDown numBackFDigitalTune;
        private System.Windows.Forms.NumericUpDown numBackQAnalogTune;
        private System.Windows.Forms.NumericUpDown numBackQDigitalTune;
        private System.Windows.Forms.TrackBar tckBackFAnalogTune;
        private System.Windows.Forms.Label label16;
        private System.Windows.Forms.Label label17;
        private System.Windows.Forms.Label label18;
        private System.Windows.Forms.Label label19;
        private System.Windows.Forms.TrackBar tckBackQDigitalTune;
        private System.Windows.Forms.TrackBar tckBackQAnalogTune;
        private System.Windows.Forms.TrackBar tckBackFDigitalTune;
        private System.Windows.Forms.CheckBox chkFrontFDDebug;
        private System.Windows.Forms.CheckBox chkFrontADDebug;
        private System.Windows.Forms.CheckBox chkFrontFDEnable;
        private System.Windows.Forms.CheckBox chkFrontADEnable;
        private System.Windows.Forms.CheckBox chkFrontEnable;
        private System.Windows.Forms.Label label12;
        private System.Windows.Forms.Label lblFrontFAnalogTune;
        private System.Windows.Forms.Label lblFrontQAnalogTune;
        private System.Windows.Forms.NumericUpDown numFrontFAnalogTune;
        private System.Windows.Forms.NumericUpDown numFrontFDigitalTune;
        private System.Windows.Forms.NumericUpDown numFrontQAnalogTune;
        private System.Windows.Forms.NumericUpDown numFrontQDigitalTune;
        private System.Windows.Forms.TrackBar tckFrontFAnalogTune;
        private System.Windows.Forms.Label label9;
        private System.Windows.Forms.Label label8;
        private System.Windows.Forms.Label label7;
        private System.Windows.Forms.Label label6;
        private System.Windows.Forms.TrackBar tckFrontQDigitalTune;
        private System.Windows.Forms.TrackBar tckFrontQAnalogTune;
        private System.Windows.Forms.TrackBar tckFrontFDigitalTune;
        private System.Windows.Forms.Button btnFilterProgram;
        private System.Windows.Forms.Button btnFilterLoad;
        private System.Windows.Forms.Button btnFilterSave;
        private System.Windows.Forms.Button btnFilterReset;
        private System.Windows.Forms.CheckBox chkPrintOptions;
        private System.Windows.Forms.CheckBox chkRFSW;
    }
}
```

FTUNEDEBUG.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace QEFilterV2TestAppV2
{
    public class FTuneDebug
    {
        private FilterControls _FilterControls;
        private DebugText _DebugText;
        private CommunicationManager _ComManager;
        private SweepFTune _SweepFTune;
        private bool _OkToClose;
        private int _SelectedPole;
        private bool _isRunning;
        private bool _RxDataAvailable;

        public const string VALUE_UNKNOWN = "N/A";
        public enum PoleSelection { FrontEnd, BackEnd }

        public FTuneDebug(FilterControls fs, DebugText dt, CommunicationManager cm)
        {
```

```
            _FilterControls = fs;
            _DebugText = dt;
            _ComManager = cm;
            _SweepFTune = new SweepFTune();
            _OkToClose = false;
            _SelectedPole = (int) FTuneDebug.PoleSelection.FrontEnd;
            _isRunning = false;
            _RxDataAvailable = false;
        }

        public SweepFTune SweepFTune
        {
            get { return _SweepFTune; }
        }

        public bool RxDataAvailable
        {
            get { return _RxDataAvailable; }
            set { _RxDataAvailable = value; }
        }

        public bool isRunning
        {
            get { return _isRunning; }
        }

        public int SelectedPole
        {
            get { return _SelectedPole; }
            set { _SelectedPole = value; }
        }

        public bool OkToClose
        {
            get { return _OkToClose; }
            set { _OkToClose = value; }
        }

        public void Run(bool run)
        {
            _isRunning = run;
            if (!_isRunning)
                SweepFTune.Reset();
        }

        public bool Step()
        {
            if (_isRunning)
            {
                string strPole = string.Empty;

                switch (_SelectedPole)
                {
                    case (int)PoleSelection.FrontEnd:
                        strPole = "Front-End";
                        _FilterControls.FrontEndCtrlReg.FrontEndEnable = _FilterControls.bool2int(_FilterControls.activeLow(true));             // Enable
Front-End
                        _FilterControls.CapacitiveCouplingCtrlReg1.FrontFDEnable = _FilterControls.bool2int(_FilterControls.activeLow(true));   // Enable
Front-End Frequency Divider
                        _FilterControls.DebugMainCtrlReg.FrontFDDebug = _FilterControls.bool2int(true);                                         // Enable
Front-End FD Debug
                        break;
                    case (int)PoleSelection.BackEnd:
                        strPole = "Back-End";
                        _FilterControls.BackEndCtrlReg.BackEndEnable = _FilterControls.bool2int(_FilterControls.activeLow(true));               // Enable
Back-End
                        _FilterControls.CapacitiveCouplingCtrlReg2.BackFDEnable = _FilterControls.bool2int(_FilterControls.activeLow(true));    // Enable
Back-End Frequency Divider
                        _FilterControls.DebugMainCtrlReg.BackFDDebug = _FilterControls.bool2int(true);                                          // Enable
Back-End FD Debug
                        break;
                    default:
                        return false;
                }
                return StepSweep(strPole);
            }
            return false;
        }

        private bool StepSweep(string strPole)
        {
            if (_SweepFTune.Itteration == -1) // Initial Run
            {
                _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, CommunicationManager.SEPERATOR);
                _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, strPole + " Frequency Tuning Sweep\r\n");
                _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, CommunicationManager.SEPERATOR);
                _FilterControls.PrintFilterOptionsToDebugWindow();
                _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, CommunicationManager.SEPERATOR);
                _SweepFTune.Itteration = 0;
            }
            else if (_SweepFTune.FCur == _SweepFTune.StopF)
                _SweepFTune.isDone = true;

            switch (_SelectedPole)
            {
                case (int)PoleSelection.FrontEnd:
                    _FilterControls.FrontEndCtrlReg.FrontFDigitalTune = _SweepFTune.FCur;
                    break;
                case (int)PoleSelection.BackEnd:
                    _FilterControls.BackEndCtrlReg.BackFDigitalTune = _SweepFTune.FCur;
                    break;
                default:
                    MessageBox.Show("Invalid Pole Selection");
                    return false;
            }

            _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "FCur: " + _SweepFTune.FCur + "\r\n");

            _FilterControls.FrontEndCtrlReg.Update();
            _FilterControls.BackEndCtrlReg.Update();
            _FilterControls.CapacitiveCouplingCtrlReg1.Update();
            _FilterControls.CapacitiveCouplingCtrlReg2.Update();
            _FilterControls.DebugMainCtrlReg.Update();
            _FilterControls.ProgramFilter();

            return (_SweepFTune.isDone ? true : false);
        }

        public void UpdateSweepInfo(int FCur, int FCnt)
        {
```

```
            _RxDataAvailable = true;
            _SweepFTune.FCur = FCur;
            _SweepFTune.FCnt = FCnt;
            _SweepFTune.FCurList.Add(FCur);
            _SweepFTune.FCntList.Add(FCnt);
            _SweepFTune.Itteration++;

            if (_SweepFTune.StartF < _SweepFTune.StopF)
                _SweepFTune.FCur++;
            else if (_SweepFTune.StartF > _SweepFTune.StopF)
                _SweepFTune.FCur--;
        }

        public void SaveData()
        {
            ExcelInterface ex;

            string chartTitle = "Frequency Tuning Sweep (" + DateTime.Now + ")";
            string cellStartF = "A6";
            string cellStopF = "A" + (6 + _SweepFTune.FCurList.Count - 1).ToString();
            string cellStartFCnt = "B6";
            string cellStopFCnt = "B" + (6 + _SweepFTune.FCntList.Count - 1).ToString();
            string fName = string.Empty;

            switch (_SelectedPole)
            {
                case (int)PoleSelection.FrontEnd:
                    chartTitle = "Front-End " + chartTitle;
                    fName = "Front-End F-Sweep at Q=" + _FilterControls.FrontEndCtrlReg.FrontQDigitalTune;
                    break;
                case (int)PoleSelection.BackEnd:
                    chartTitle = "Back-End " + chartTitle;
                    fName = "Back-End F-Sweep at Q=" + _FilterControls.BackEndCtrlReg.BackQDigitalTune;
                    break;
                default:
                    break;
            }
            ex = new ExcelInterface(chartTitle, "Digital F-Tuning Setting", "Frequency Counter Ouput", 8.24, 3.47);

            ex.xlWorkSheet.Cells[1, 1] = chartTitle;
            ex.xlWorkSheet.Cells[2, 1] = "StartF";
            ex.xlWorkSheet.Cells[3, 1] = "StopF";
            ex.xlWorkSheet.Cells[2, 2] = _SweepFTune.StartF;
            ex.xlWorkSheet.Cells[3, 2] = _SweepFTune.StopF;
            ex.xlWorkSheet.Cells[5, 1] = "F-Tune";
            ex.xlWorkSheet.Cells[5, 2] = "Count";

            ex.xlWorkSheet.get_Range("A1", "A3").Font.Bold = true;
            ex.xlWorkSheet.get_Range("B2", "B3").HorizontalAlignment = Microsoft.Office.Interop.Excel.XlHAlign.xlHAlignCenter;
            ex.xlWorkSheet.get_Range("A5", "B5").Font.Bold = true;
            ex.xlWorkSheet.get_Range("A5", cellStopFCnt).HorizontalAlignment = Microsoft.Office.Interop.Excel.XlHAlign.xlHAlignCenter;

            for (int i = 0; i < _SweepFTune.FCurList.Count; i++)
                ex.xlWorkSheet.Cells[6 + i, 1] = _SweepFTune.FCurList.ElementAt(i);

            for (int i = 0; i < _SweepFTune.FCntList.Count; i++)
                ex.xlWorkSheet.Cells[6 + i, 2] = _SweepFTune.FCntList.ElementAt(i);

            ex.chartRange = ex.xlWorkSheet.get_Range(cellStartFCnt, cellStopFCnt);
            ex.chartPage.SetSourceData(ex.chartRange, ex.misValue);
            ex.chartSeries = (Microsoft.Office.Interop.Excel.Series)ex.chartPage.SeriesCollection(1);
            ex.chartSeries.XValues = ex.xlWorkSheet.get_Range(cellStartF, cellStopF);
            ex.chartSeries.Format.Line.Weight = (float)2.0;
            ex.chartSeries.MarkerStyle = Microsoft.Office.Interop.Excel.XlMarkerStyle.xlMarkerStyleDiamond;
            ex.chartSeries.MarkerSize = 3;


            ex.xlWorkSheet.Cells[23, 4] = "Front-End Enable";
            ex.xlWorkSheet.Cells[23, 8] = printActiveLow(_FilterControls.FrontEndCtrlReg.FrontEndEnable);
            ex.xlWorkSheet.Cells[24, 4] = "Front-End Digital Q Tuning";
            ex.xlWorkSheet.Cells[24, 8] = _FilterControls.FrontEndCtrlReg.FrontQDigitalTune.ToString();
            ex.xlWorkSheet.Cells[25, 4] = "Front-End Analog Q Tuning";
            ex.xlWorkSheet.Cells[25, 8] = _FilterControls.AnalogTuningCtrlReg.FrontEndAnalogQ.ToString();
            ex.xlWorkSheet.Cells[26, 4] = "Front-End Digital F Tuning";
            ex.xlWorkSheet.Cells[26, 8] = _FilterControls.FrontEndCtrlReg.FrontFDigitalTune.ToString();
            ex.xlWorkSheet.Cells[27, 4] = "Front-End Analog F Tuning";
            ex.xlWorkSheet.Cells[27, 8] = _FilterControls.AnalogTuningCtrlReg.FrontEndAnalogF.ToString();
            ex.xlWorkSheet.Cells[28, 4] = "Front-End Amplitude Detector Enable";
            ex.xlWorkSheet.Cells[28, 8] = printActiveLow(_FilterControls.FrontEndCtrlReg.FrontADEnable);
            ex.xlWorkSheet.Cells[29, 4] = "Front-End Frequency Divider Enable";
            ex.xlWorkSheet.Cells[29, 8] = printActiveLow(_FilterControls.CapacitiveCouplingCtrlReg1.FrontFDEnable);
            ex.xlWorkSheet.Cells[31, 4] = "Back-End Enable";
            ex.xlWorkSheet.Cells[31, 8] = printActiveLow(_FilterControls.BackEndCtrlReg.BackEndEnable);
            ex.xlWorkSheet.Cells[32, 4] = "Back-End Digital Q Tuning";
            ex.xlWorkSheet.Cells[32, 8] = _FilterControls.BackEndCtrlReg.BackQDigitalTune.ToString();
            ex.xlWorkSheet.Cells[33, 4] = "Back-End Analog Q Tuning";
            ex.xlWorkSheet.Cells[33, 8] = _FilterControls.AnalogTuningCtrlReg.BackEndAnalogQ.ToString();
            ex.xlWorkSheet.Cells[34, 4] = "Back-End Digital F Tuning";
            ex.xlWorkSheet.Cells[34, 8] = _FilterControls.BackEndCtrlReg.BackFDigitalTune.ToString();
            ex.xlWorkSheet.Cells[35, 4] = "Back-End Analog F Tuning";
            ex.xlWorkSheet.Cells[35, 8] = _FilterControls.AnalogTuningCtrlReg.BackEndAnalogF.ToString();
            ex.xlWorkSheet.Cells[36, 4] = "Back-End Amplitude Detector Enable";
            ex.xlWorkSheet.Cells[36, 8] = printActiveLow(_FilterControls.BackEndCtrlReg.BackADEnable);
            ex.xlWorkSheet.Cells[37, 4] = "Back-End Frequency Divider Enable";
            ex.xlWorkSheet.Cells[37, 8] = printActiveLow(_FilterControls.CapacitiveCouplingCtrlReg2.BackFDEnable);
            ex.xlWorkSheet.Cells[39, 4] = "Upper Capacitive Coupling Tuning";
            ex.xlWorkSheet.Cells[39, 8] = _FilterControls.CapacitiveCouplingCtrlReg1.CoupleUpper.ToString();
            ex.xlWorkSheet.Cells[40, 4] = "Lower Capacitive Coupling Tuning";
            ex.xlWorkSheet.Cells[40, 8] = _FilterControls.CapacitiveCouplingCtrlReg1.CoupleLower.ToString();
            ex.xlWorkSheet.Cells[41, 4] = "UFLB Capacitive Coupling Tuning";
            ex.xlWorkSheet.Cells[41, 8] = _FilterControls.CapacitiveCouplingCtrlReg2.CoupleUFLB.ToString();
            ex.xlWorkSheet.Cells[42, 4] = "LFUB Capacitive Coupling Tuning";
            ex.xlWorkSheet.Cells[42, 8] = _FilterControls.CapacitiveCouplingCtrlReg2.CoupleLFUB.ToString();
            ex.xlWorkSheet.get_Range("H23", "H42").HorizontalAlignment = Microsoft.Office.Interop.Excel.XlHAlign.xlHAlignCenter;

            ex.SaveXLSFile(fName);
        }

        public string printActiveLow(int val)
        {
            return (val == 0 ? "Enabled" : "Disabled");
        }
    }

    public class SweepFTune
    {
        private int _Itteration;
```

```
        private int _StartF, _StopF;
        private int _FCur, _FCnt;
        private List<int> _FCurList;
        private List<int> _FCntList;
        private bool _isDone;

        public SweepFTune()
        {
            Reset();
        }

        public int Itteration
        {
            get { return _Itteration; }
            set { _Itteration = value; }
        }

        public int StartF
        {
            get { return _StartF; }
            set { _StartF = value; }
        }

        public int StopF
        {
            get { return _StopF; }
            set { _StopF = value; }
        }

        public int FCur
        {
            get { return _FCur; }
            set { _FCur = value; }
        }

        public int FCnt
        {
            get { return _FCnt; }
            set { _FCnt = value; }
        }

        public List<int> FCurList
        {
            get { return _FCurList; }
        }

        public List<int> FCntList
        {
            get { return _FCntList; }
        }

        public bool isDone
        {
            get { return _isDone; }
            set { _isDone = true; }
        }

        public void Reset()
        {
            _Itteration = -1;
            _StartF = -1;
            _StopF = -1;
            _FCurList = new List<int>();
            _FCntList = new List<int>();
            _isDone = false;
        }
    }
}
```

FRMFTUNEDEBUG.CS

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace QEFilterV2TestAppV2
{
    public partial class frmFTuneDebug : Form
    {
        private FTuneDebug _FTuneDebug;
        private DebugText _DebugText;
        private CommunicationManager _ComManager;
        private FilterControls _FilterControls;
        private Timer _ResponseTimer;
        private Timer _AutomaticStepTimer;
        private int _startTime;
        private bool _isRunning;
        private bool _isSweepDone;
        private bool _isSweepAutomated;

        public frmFTuneDebug(FTuneDebug ftd, DebugText dt, CommunicationManager cm, FilterControls fs)
        {
            InitializeComponent();
            this.Closing += new CancelEventHandler(frmFTuneDebug_Closing);
            _FTuneDebug = ftd;
            _DebugText = dt;
            _ComManager = cm;
            _FilterControls = fs;
            _ResponseTimer = new Timer();
            _ResponseTimer.Interval = 10; //10mS
            _ResponseTimer.Tick += new EventHandler(ResponseTimer_Tick);
            _AutomaticStepTimer = new Timer();
            _AutomaticStepTimer.Interval = 250; //250mS
            _AutomaticStepTimer.Tick += new EventHandler(AutomaticStepTimer_Tick);
            _isRunning = false;
            _isSweepDone = false;
            _isSweepAutomated = false;
            ResetAllControls();
        }
```

```
private void frmFTuneDebug_Closing(object sender, CancelEventArgs cArgs)
{
    if (!_FTuneDebug.OkToClose)
        cArgs.Cancel = true;
    else
        cArgs.Cancel = false;
}

private void ResetAllControls()
{
    radFrontEnd.Checked = true;
    radBackEnd.Checked = false;
    btnFTuningStep.Enabled = false;
    ResetSweepControls();
}

private void ResetSweepControls()
{
    numStartF.Minimum = FilterControls.DIGFTUNEMIN;
    numStartF.Maximum = FilterControls.DIGFTUNEMAX;
    numStopF.Minimum = FilterControls.DIGFTUNEMIN;
    numStopF.Maximum = FilterControls.DIGFTUNEMAX;
    numStartF.Value = numStartF.Minimum;
    numStopF.Value = numStopF.Maximum;
    txtCurF.Text = FTuneDebug.VALUE_UNKNOWN;
    txtFCnt.Text = FTuneDebug.VALUE_UNKNOWN;
}

private void radFrontEnd_CheckedChanged(object sender, EventArgs e)
{
    if (radFrontEnd.Checked)
    {
        radBackEnd.Checked = false;
        _FTuneDebug.SelectedPole = (int)FTuneDebug.PoleSelection.FrontEnd;
    }
    else if (radBackEnd.Checked)
    {
        radFrontEnd.Checked = false;
        _FTuneDebug.SelectedPole = (int)FTuneDebug.PoleSelection.BackEnd;
    }
    Reset();
}

private void radBackEnd_CheckedChanged(object sender, EventArgs e)
{
    if (radFrontEnd.Checked)
    {
        radBackEnd.Checked = false;
        _FTuneDebug.SelectedPole = (int)FTuneDebug.PoleSelection.FrontEnd;
    }
    else if (radBackEnd.Checked)
    {
        radFrontEnd.Checked = false;
        _FTuneDebug.SelectedPole = (int)FTuneDebug.PoleSelection.BackEnd;
    }
    Reset();
}

private void btnFTuningDebugReset_Click(object sender, EventArgs e)
{
    Reset();
}

private void Reset()
{
    _isRunning = false;
    _isSweepAutomated = false;
    _FTuneDebug.Run(false);
    ResetSweepControls();
    btnFTuningRun.Text = "Run";
    btnFTuningStep.Enabled = false;
}

private void btnFTuningStep_Click(object sender, EventArgs e)
{
    if (!_ComManager.isOpen)
    {
        MessageBox.Show("Selected COM Port is not open.");
        return;
    }
    step();
}

private void step()
{
    if (_FTuneDebug.SweepFTune.Itteration == -1)
    {
        _FTuneDebug.SweepFTune.StartF = int.Parse(numStartF.Value.ToString());
        _FTuneDebug.SweepFTune.StopF = int.Parse(numStopF.Value.ToString());
        _FTuneDebug.SweepFTune.FCur = _FTuneDebug.SweepFTune.StartF;
    }
    _isSweepDone = _FTuneDebug.Step();
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    //_FTuneDebug.UpdateSweepInfo(_FTuneDebug.SweepFTune.FCur, (new Random().Next(FilterControls.DIGFTUNEMIN, FilterControls.DIGFTUNEMAX)));
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    _startTime = DateTime.Now.Second;
    _ResponseTimer.Start();
}

private void btnFTuningRun_Click(object sender, EventArgs e)
{
    if (!_ComManager.isOpen)
    {
        MessageBox.Show("Selected COM Port is not open.");
        return;
    }
    if (numStartF.Value == numStopF.Value)
    {
        MessageBox.Show("Start and Stop values are the same, canceling sweep.");
        return;
    }

    _isRunning = !_isRunning;
    if (!_isRunning)
    {
        int tmpStartF = int.Parse(numStartF.Value.ToString());
        int tmpStopF = int.Parse(numStopF.Value.ToString());
        _FTuneDebug.Run(false);
        btnFTuningRun.Text = "Run";
        btnFTuningStep.Enabled = false;
```

```
                ResetSweepControls();
                numStartF.Value = tmpStartF;
                numStopF.Value = tmpStopF;
                _isSweepAutomated = false;
            }
            else
            {
                DialogResult SweepType = MessageBox.Show("Would you like to automate the sweep?", "Automate Sweep?", MessageBoxButtons.YesNo);
                txtCurF.Text = FTuneDebug.VALUE_UNKNOWN;
                txtFCnt.Text = FTuneDebug.VALUE_UNKNOWN;
                _FTuneDebug.Run(true);
                btnFTuningRun.Text = "Stop";
                btnFTuningStep.Enabled = true;
                switch (SweepType)
                {
                    case DialogResult.Yes:
                        _isSweepAutomated = true;
                        step();
                        break;
                    case DialogResult.No:
                        _isSweepAutomated = false;
                        break;
                    default:
                        return;
                }
            }
        }

        public void ResponseTimer_Tick(object sender, EventArgs eArgs)
        {
            if (sender == _ResponseTimer)
            {
                if (!_FTuneDebug.RxDataAvailable && _FTuneDebug.SweepFTune.Itteration != -1)
                {
                    btnFTuningStep.Enabled = false;
                    if (DateTime.Now.Second >= _startTime + 5) // 5 Seconds Have Elapsed w/o RxData
                    {
                        _ResponseTimer.Stop();
                        MessageBox.Show("No information has Been received for 5 seconds, exiting current sweep.");
                        _FTuneDebug.Run(false);
                        Reset();
                    }
                }
                else if (_FTuneDebug.RxDataAvailable && _FTuneDebug.SweepFTune.FCurList.Count > 0)
                {
                    _ResponseTimer.Stop();
                    int i = _FTuneDebug.SweepFTune.FCurList.Count - 1;
                    if (i > -1)
                    {
                        txtCurF.Text = _FTuneDebug.SweepFTune.FCurList.ElementAt(i).ToString();
                        txtFCnt.Text = _FTuneDebug.SweepFTune.FCntList.ElementAt(i).ToString();
                        _FTuneDebug.RxDataAvailable = false;
                        btnFTuningStep.Enabled = true;

                        if (_isSweepDone)
                        {
                            _isRunning = false;
                            _isSweepDone = false;
                            btnFTuningRun.Text = "Run";
                            btnFTuningStep.Enabled = false;
                            DialogResult result = MessageBox.Show("F-Tuning Sweep is Complete!\nWould you like to save it as an Excel Spreadsheet?", "Save
Debug Data", MessageBoxButtons.YesNo);
                            if (result == System.Windows.Forms.DialogResult.Yes)
                                _FTuneDebug.SaveData();

                            _FTuneDebug.Run(false);
                            return;
                        }
                        else if (_isSweepAutomated)
                            _AutomaticStepTimer.Start();
                    }
                }
            }
        }

        public void AutomaticStepTimer_Tick(object sender, EventArgs eArgs)
        {
            if (sender == _AutomaticStepTimer)
            {
                _AutomaticStepTimer.Stop();
                step();
            }
        }

    }
}
```

FRMFTUNEDEBUG.DESIGNER.CS

```
namespace QEFilterV2TestAppV2
{
    partial class frmFTuneDebug
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
```

```
private void InitializeComponent()
{
    this.panel1 = new System.Windows.Forms.Panel();
    this.radBackEnd = new System.Windows.Forms.RadioButton();
    this.radFrontEnd = new System.Windows.Forms.RadioButton();
    this.btnFTuningRun = new System.Windows.Forms.Button();
    this.btnFTuningStep = new System.Windows.Forms.Button();
    this.btnFTuningDebugReset = new System.Windows.Forms.Button();
    this.label12 = new System.Windows.Forms.Label();
    this.txtFCnt = new System.Windows.Forms.TextBox();
    this.label13 = new System.Windows.Forms.Label();
    this.txtCurF = new System.Windows.Forms.TextBox();
    this.groupFilterSettings = new System.Windows.Forms.GroupBox();
    this.label14 = new System.Windows.Forms.Label();
    this.numStopF = new System.Windows.Forms.NumericUpDown();
    this.numStartF = new System.Windows.Forms.NumericUpDown();
    this.label15 = new System.Windows.Forms.Label();
    this.panel1.SuspendLayout();
    this.groupFilterSettings.SuspendLayout();
    ((System.ComponentModel.ISupportInitialize)(this.numStopF)).BeginInit();
    ((System.ComponentModel.ISupportInitialize)(this.numStartF)).BeginInit();
    this.SuspendLayout();
    //
    // panel1
    //
    this.panel1.Controls.Add(this.radBackEnd);
    this.panel1.Controls.Add(this.radFrontEnd);
    this.panel1.Location = new System.Drawing.Point(74, 25);
    this.panel1.Name = "panel1";
    this.panel1.Size = new System.Drawing.Size(314, 36);
    this.panel1.TabIndex = 66;
    //
    // radBackEnd
    //
    this.radBackEnd.AutoSize = true;
    this.radBackEnd.Location = new System.Drawing.Point(182, 3);
    this.radBackEnd.Name = "radBackEnd";
    this.radBackEnd.Size = new System.Drawing.Size(97, 23);
    this.radBackEnd.TabIndex = 37;
    this.radBackEnd.TabStop = true;
    this.radBackEnd.Text = "Back-End";
    this.radBackEnd.UseVisualStyleBackColor = true;
    this.radBackEnd.CheckedChanged += new System.EventHandler(this.radBackEnd_CheckedChanged);
    //
    // radFrontEnd
    //
    this.radFrontEnd.AutoSize = true;
    this.radFrontEnd.Location = new System.Drawing.Point(31, 3);
    this.radFrontEnd.Name = "radFrontEnd";
    this.radFrontEnd.Size = new System.Drawing.Size(102, 23);
    this.radFrontEnd.TabIndex = 36;
    this.radFrontEnd.TabStop = true;
    this.radFrontEnd.Text = "Front-End";
    this.radFrontEnd.UseVisualStyleBackColor = true;
    this.radFrontEnd.CheckedChanged += new System.EventHandler(this.radFrontEnd_CheckedChanged);
    //
    // btnFTuningRun
    //
    this.btnFTuningRun.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
    this.btnFTuningRun.Location = new System.Drawing.Point(297, 120);
    this.btnFTuningRun.Name = "btnFTuningRun";
    this.btnFTuningRun.Size = new System.Drawing.Size(90, 23);
    this.btnFTuningRun.TabIndex = 64;
    this.btnFTuningRun.Text = "Run";
    this.btnFTuningRun.UseVisualStyleBackColor = true;
    this.btnFTuningRun.Click += new System.EventHandler(this.btnFTuningRun_Click);
    //
    // btnFTuningStep
    //
    this.btnFTuningStep.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
    this.btnFTuningStep.Location = new System.Drawing.Point(187, 120);
    this.btnFTuningStep.Name = "btnFTuningStep";
    this.btnFTuningStep.Size = new System.Drawing.Size(90, 23);
    this.btnFTuningStep.TabIndex = 62;
    this.btnFTuningStep.Text = "Step";
    this.btnFTuningStep.UseVisualStyleBackColor = true;
    this.btnFTuningStep.Click += new System.EventHandler(this.btnFTuningStep_Click);
    //
    // btnFTuningDebugReset
    //
    this.btnFTuningDebugReset.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
    this.btnFTuningDebugReset.Location = new System.Drawing.Point(76, 120);
    this.btnFTuningDebugReset.Name = "btnFTuningDebugReset";
    this.btnFTuningDebugReset.Size = new System.Drawing.Size(90, 23);
    this.btnFTuningDebugReset.TabIndex = 61;
    this.btnFTuningDebugReset.Text = "Reset";
    this.btnFTuningDebugReset.UseVisualStyleBackColor = true;
    this.btnFTuningDebugReset.Click += new System.EventHandler(this.btnFTuningDebugReset_Click);
    //
    // label12
    //
    this.label12.AutoSize = true;
    this.label12.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
    this.label12.Location = new System.Drawing.Point(353, 68);
    this.label12.Name = "label12";
    this.label12.Size = new System.Drawing.Size(88, 13);
    this.label12.TabIndex = 58;
    this.label12.Text = "Frequency Count";
    //
    // txtFCnt
    //
    this.txtFCnt.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
    this.txtFCnt.Location = new System.Drawing.Point(367, 84);
    this.txtFCnt.Name = "txtFCnt";
    this.txtFCnt.ReadOnly = true;
    this.txtFCnt.Size = new System.Drawing.Size(61, 20);
    this.txtFCnt.TabIndex = 57;
    //
    // label13
    //
    this.label13.AutoSize = true;
    this.label13.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
    this.label13.Location = new System.Drawing.Point(262, 68);
    this.label13.Name = "label13";
    this.label13.Size = new System.Drawing.Size(50, 13);
    this.label13.TabIndex = 56;
    this.label13.Text = "Current F";
```

```
            //
            // txtCurF
            //
            this.txtCurF.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.txtCurF.Location = new System.Drawing.Point(257, 84);
            this.txtCurF.Name = "txtCurF";
            this.txtCurF.ReadOnly = true;
            this.txtCurF.Size = new System.Drawing.Size(61, 20);
            this.txtCurF.TabIndex = 52;
            //
            // groupFilterSettings
            //
            this.groupFilterSettings.Controls.Add(this.panel1);
            this.groupFilterSettings.Controls.Add(this.btnFTuningRun);
            this.groupFilterSettings.Controls.Add(this.btnFTuningStep);
            this.groupFilterSettings.Controls.Add(this.btnFTuningDebugReset);
            this.groupFilterSettings.Controls.Add(this.label12);
            this.groupFilterSettings.Controls.Add(this.txtFCnt);
            this.groupFilterSettings.Controls.Add(this.label13);
            this.groupFilterSettings.Controls.Add(this.txtCurF);
            this.groupFilterSettings.Controls.Add(this.label14);
            this.groupFilterSettings.Controls.Add(this.numStopF);
            this.groupFilterSettings.Controls.Add(this.numStartF);
            this.groupFilterSettings.Controls.Add(this.label15);
            this.groupFilterSettings.Font = new System.Drawing.Font("Cambria", 12F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.groupFilterSettings.Location = new System.Drawing.Point(7, 7);
            this.groupFilterSettings.Name = "groupFilterSettings";
            this.groupFilterSettings.Size = new System.Drawing.Size(462, 153);
            this.groupFilterSettings.TabIndex = 6;
            this.groupFilterSettings.TabStop = false;
            this.groupFilterSettings.Text = "Frequency Tuning Debug";
            //
            // label14
            //
            this.label14.AutoSize = true;
            this.label14.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.label14.Location = new System.Drawing.Point(157, 68);
            this.label14.Name = "label14";
            this.label14.Size = new System.Drawing.Size(38, 13);
            this.label14.TabIndex = 55;
            this.label14.Text = "Stop F";
            //
            // numStopF
            //
            this.numStopF.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.numStopF.Location = new System.Drawing.Point(147, 84);
            this.numStopF.Name = "numStopF";
            this.numStopF.Size = new System.Drawing.Size(61, 20);
            this.numStopF.TabIndex = 54;
            //
            // numStartF
            //
            this.numStartF.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.numStartF.Location = new System.Drawing.Point(36, 84);
            this.numStartF.Name = "numStartF";
            this.numStartF.Size = new System.Drawing.Size(61, 20);
            this.numStartF.TabIndex = 53;
            //
            // label15
            //
            this.label15.AutoSize = true;
            this.label15.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.label15.Location = new System.Drawing.Point(46, 68);
            this.label15.Name = "label15";
            this.label15.Size = new System.Drawing.Size(38, 13);
            this.label15.TabIndex = 51;
            this.label15.Text = "Start F";
            //
            // frmFTuneDebug
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(477, 167);
            this.Controls.Add(this.groupFilterSettings);
            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
            this.MaximizeBox = false;
            this.Name = "frmFTuneDebug";
            this.Text = "Frequency Tuning Debug";
            this.panel1.ResumeLayout(false);
            this.panel1.PerformLayout();
            this.groupFilterSettings.ResumeLayout(false);
            this.groupFilterSettings.PerformLayout();
            ((System.ComponentModel.ISupportInitialize)(this.numStopF)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numStartF)).EndInit();
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.Panel panel1;
        private System.Windows.Forms.RadioButton radBackEnd;
        private System.Windows.Forms.RadioButton radFrontEnd;
        private System.Windows.Forms.Button btnFTuningRun;
        private System.Windows.Forms.Button btnFTuningStep;
        private System.Windows.Forms.Button btnFTuningDebugReset;
        private System.Windows.Forms.Label label12;
        private System.Windows.Forms.TextBox txtFCnt;
        private System.Windows.Forms.Label label13;
        private System.Windows.Forms.TextBox txtCurF;
        private System.Windows.Forms.GroupBox groupFilterSettings;
        private System.Windows.Forms.Label label14;
        private System.Windows.Forms.NumericUpDown numStopF;
        private System.Windows.Forms.NumericUpDown numStartF;
        private System.Windows.Forms.Label label15;
    }
}
```

QTUNEDEBUG.CS

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace QEFilterV2TestAppV2
{
    public class QTuneDebug
    {
        private FilterControls _FilterControls;
        private DebugText _DebugText;
        private CommunicationManager _ComManager;
        private SweepQTune _SweepQTune;
        private bool _OkToClose;
        private int _SelectedPole;
        private bool _isRunning;
        private bool _RxDataAvailable;

        public const string VALUE_UNKNOWN = "N/A";
        public enum PoleSelection { FrontEnd, BackEnd }

        public QTuneDebug(FilterControls fs, DebugText dt, CommunicationManager cm)
        {
            _FilterControls = fs;
            _DebugText = dt;
            _ComManager = cm;
            _SweepQTune = new SweepQTune();
            _OkToClose = false;
            _SelectedPole = (int) QTuneDebug.PoleSelection.FrontEnd;
            _isRunning = false;
            _RxDataAvailable = false;
        }

        public SweepQTune SweepQTune
        {
            get { return _SweepQTune; }
        }

        public bool RxDataAvailable
        {
            get { return _RxDataAvailable; }
            set { _RxDataAvailable = value; }
        }

        public bool isRunning
        {
            get { return _isRunning; }
        }

        public int SelectedPole
        {
            get { return _SelectedPole; }
            set { _SelectedPole = value; }
        }

        public bool OkToClose
        {
            get { return _OkToClose; }
            set { _OkToClose = value; }
        }

        public void Run(bool run)
        {
            _isRunning = run;
            if (!_isRunning)
                SweepQTune.Reset();
        }

        public bool Step()
        {
            if (_isRunning)
            {
                string strPole = string.Empty;

                switch (_SelectedPole)
                {
                    case (int)PoleSelection.FrontEnd:
                        strPole = "Front-End";
                        _FilterControls.FrontEndCtrlReg.FrontEndEnable = _FilterControls.bool2int(_FilterControls.activeLow(true));      // Enable Front-End
                        _FilterControls.FrontEndCtrlReg.FrontADEnable = _FilterControls.bool2int(_FilterControls.activeLow(true));      // Enable Front-End
Amplitude Detector
                        _FilterControls.DebugMainCtrlReg.FrontADDebug = _FilterControls.bool2int(true);                                  // Enable Front-End
AD Debug
                        break;
                    case (int)PoleSelection.BackEnd:
                        strPole = "Back-End";
                        _FilterControls.BackEndCtrlReg.BackEndEnable = _FilterControls.bool2int(_FilterControls.activeLow(true));       // Enable Back-End
                        _FilterControls.BackEndCtrlReg.BackADEnable = _FilterControls.bool2int(_FilterControls.activeLow(true));        // Enable Back-End
Amplitude Detector
                        _FilterControls.DebugMainCtrlReg.BackADDebug = _FilterControls.bool2int(true);                                  // Enable Back-End
AD Debug
                        break;
                    default:
                        return false;
                }

                return StepSweep(strPole);
            }
            return false;
        }


        private bool StepSweep(string strPole)
        {
            if (_SweepQTune.Itteration == -1) // Initial Run
            {
                _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, CommunicationManager.SEPERATOR);
                _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, strPole + " Q-Tuning Sweep\r\n");
                _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, CommunicationManager.SEPERATOR);
                _FilterControls.PrintFilterOptionsToDebugWindow();
                _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, CommunicationManager.SEPERATOR);
                _SweepQTune.Itteration = 0;
            }
            else if (_SweepQTune.NqCur == _SweepQTune.StopQ)
                _SweepQTune.isDone = true;

            switch (_SelectedPole)
            {
                case (int)PoleSelection.FrontEnd:
                    _FilterControls.FrontEndCtrlReg.FrontQDigitalTune = _SweepQTune.NqCur;
                    break;
```

```
            case (int)PoleSelection.BackEnd:
                _FilterControls.BackEndCtrlReg.BackQDigitalTune = _SweepQTune.NqCur;
                break;
            default:
                MessageBox.Show("Invalid Pole Selection");
                return false;
        }

        _DebugText.DisplayDebugData(DebugText.MessageClass.Debug, DebugText.MessageType.Normal, "Nq: " + _SweepQTune.NqCur + "\r\n");

        _FilterControls.FrontEndCtrlReg.Update();
        _FilterControls.BackEndCtrlReg.Update();
        _FilterControls.CapacitiveCouplingCtrlReg1.Update();
        _FilterControls.CapacitiveCouplingCtrlReg2.Update();
        _FilterControls.DebugMainCtrlReg.Update();
        _FilterControls.ProgramFilter();

        return (_SweepQTune.isDone ? true : false);
    }

    public void UpdateSweepInfo(int NqCur, int VqCur)
    {
        _RxDataAvailable = true;
        _SweepQTune.NqCur = NqCur;
        _SweepQTune.VqCur = VqCur;
        _SweepQTune.NqCurList.Add(NqCur);
        _SweepQTune.VqCurList.Add(VqCur);
        _SweepQTune.Itteration++;

        if (_SweepQTune.StartQ < _SweepQTune.StopQ)
            _SweepQTune.NqCur++;
        else if (_SweepQTune.StartQ > _SweepQTune.StopQ)
            _SweepQTune.NqCur--;
    }

    public void SaveData()
    {
        ExcelInterface ex;

        string chartTitle = "Q-Tuning Sweep (" + DateTime.Now + ")";
        string cellNqStart = "A6";
        string cellNqStop = "A" + (6 + _SweepQTune.NqCurList.Count - 1).ToString();
        string cellVqStart = "B6";
        string cellVqStop = "B" + (6 + _SweepQTune.VqCurList.Count - 1).ToString();
        string fName = string.Empty;

        switch (_SelectedPole)
        {
            case (int)PoleSelection.FrontEnd:
                chartTitle = "Front-End " + chartTitle;
                fName = "Front-End Q-Sweep at F=" + _FilterControls.FrontEndCtrlReg.FrontFDigitalTune;
                break;
            case (int)PoleSelection.BackEnd:
                chartTitle = "Back-End " + chartTitle;
                fName = "Back-End Q-Sweep at F=" + _FilterControls.BackEndCtrlReg.BackFDigitalTune;
                break;
            default:
                break;
        }
        ex = new ExcelInterface(chartTitle, "Digital Q-Enhancement Setting", "Amplitude Detector Ouput", 8.24, 3.47);

        ex.xlWorkSheet.Cells[1, 1] = chartTitle;
        ex.xlWorkSheet.Cells[2, 1] = "StartQ";
        ex.xlWorkSheet.Cells[3, 1] = "StopQ";
        ex.xlWorkSheet.Cells[2, 2] = _SweepQTune.StartQ;
        ex.xlWorkSheet.Cells[3, 2] = _SweepQTune.StopQ;
        ex.xlWorkSheet.Cells[5, 1] = "Nq";
        ex.xlWorkSheet.Cells[5, 2] = "Vq";

        ex.xlWorkSheet.get_Range("A1", "A3").Font.Bold = true;
        ex.xlWorkSheet.get_Range("B2", "B3").HorizontalAlignment = Microsoft.Office.Interop.Excel.XlHAlign.xlHAlignCenter;
        ex.xlWorkSheet.get_Range("A5", "B5").Font.Bold = true;
        ex.xlWorkSheet.get_Range("A5", cellVqStop).HorizontalAlignment = Microsoft.Office.Interop.Excel.XlHAlign.xlHAlignCenter;

        for (int i = 0; i < _SweepQTune.NqCurList.Count; i++)
            ex.xlWorkSheet.Cells[6 + i, 1] = _SweepQTune.NqCurList.ElementAt(i);

        for (int i = 0; i < _SweepQTune.VqCurList.Count; i++)
            ex.xlWorkSheet.Cells[6 + i, 2] = _SweepQTune.VqCurList.ElementAt(i);

        ex.chartRange = ex.xlWorkSheet.get_Range(cellVqStart, cellVqStop);
        ex.chartPage.SetSourceData(ex.chartRange, ex.misValue);
        ex.chartSeries = (Microsoft.Office.Interop.Excel.Series)ex.chartPage.SeriesCollection(1);
        ex.chartSeries.XValues = ex.xlWorkSheet.get_Range(cellNqStart, cellNqStop);
        ex.chartSeries.Format.Line.Weight = (float)2.0;
        ex.chartSeries.MarkerStyle = Microsoft.Office.Interop.Excel.XlMarkerStyle.xlMarkerStyleDiamond;
        ex.chartSeries.MarkerSize = 3;


        ex.xlWorkSheet.Cells[23, 4] = "Front-End Enable";
        ex.xlWorkSheet.Cells[23, 8] = printActiveLow(_FilterControls.FrontEndCtrlReg.FrontEndEnable);
        ex.xlWorkSheet.Cells[24, 4] = "Front-End Digital Q Tuning";
        ex.xlWorkSheet.Cells[24, 8] = _FilterControls.FrontEndCtrlReg.FrontQDigitalTune.ToString();
        ex.xlWorkSheet.Cells[25, 4] = "Front-End Analog Q Tuning";
        ex.xlWorkSheet.Cells[25, 8] = _FilterControls.AnalogTuningCtrlReg.FrontEndAnalogQ.ToString();
        ex.xlWorkSheet.Cells[26, 4] = "Front-End Digital F Tuning";
        ex.xlWorkSheet.Cells[26, 8] = _FilterControls.FrontEndCtrlReg.FrontFDigitalTune.ToString();
        ex.xlWorkSheet.Cells[27, 4] = "Front-End Analog F Tuning";
        ex.xlWorkSheet.Cells[27, 8] = _FilterControls.AnalogTuningCtrlReg.FrontEndAnalogF.ToString();
        ex.xlWorkSheet.Cells[28, 4] = "Front-End Amplitude Detector Enable";
        ex.xlWorkSheet.Cells[28, 8] = printActiveLow(_FilterControls.FrontEndCtrlReg.FrontADEnable);
        ex.xlWorkSheet.Cells[29, 4] = "Front-End Frequency Divider Enable";
        ex.xlWorkSheet.Cells[29, 8] = printActiveLow(_FilterControls.CapacitiveCouplingCtrlReg1.FrontFDEnable);
        ex.xlWorkSheet.Cells[31, 4] = "Back-End Enable";
        ex.xlWorkSheet.Cells[31, 8] = printActiveLow(_FilterControls.BackEndCtrlReg.BackEndEnable);
        ex.xlWorkSheet.Cells[32, 4] = "Back-End Digital Q Tuning";
        ex.xlWorkSheet.Cells[32, 8] = _FilterControls.BackEndCtrlReg.BackQDigitalTune.ToString();
        ex.xlWorkSheet.Cells[33, 4] = "Back-End Analog Q Tuning";
        ex.xlWorkSheet.Cells[33, 8] = _FilterControls.AnalogTuningCtrlReg.BackEndAnalogQ.ToString();
        ex.xlWorkSheet.Cells[34, 4] = "Back-End Digital F Tuning";
        ex.xlWorkSheet.Cells[34, 8] = _FilterControls.BackEndCtrlReg.BackFDigitalTune.ToString();
        ex.xlWorkSheet.Cells[35, 4] = "Back-End Analog F Tuning";
        ex.xlWorkSheet.Cells[35, 8] = _FilterControls.AnalogTuningCtrlReg.BackEndAnalogF.ToString();
        ex.xlWorkSheet.Cells[36, 4] = "Back-End Amplitude Detector Enable";
        ex.xlWorkSheet.Cells[36, 8] = printActiveLow(_FilterControls.BackEndCtrlReg.BackADEnable);
        ex.xlWorkSheet.Cells[37, 4] = "Back-End Frequency Divider Enable";
        ex.xlWorkSheet.Cells[37, 8] = printActiveLow(_FilterControls.CapacitiveCouplingCtrlReg2.BackFDEnable);
```

```
            ex.xlWorkSheet.Cells[39, 4] = "Upper Capacitive Coupling Tuning";
            ex.xlWorkSheet.Cells[39, 8] = _FilterControls.CapacitiveCouplingCtrlReg1.CoupleUpper.ToString();
            ex.xlWorkSheet.Cells[40, 4] = "Lower Capacitive Coupling Tuning";
            ex.xlWorkSheet.Cells[40, 8] = _FilterControls.CapacitiveCouplingCtrlReg1.CoupleLower.ToString();
            ex.xlWorkSheet.Cells[41, 4] = "UFLB Capacitive Coupling Tuning";
            ex.xlWorkSheet.Cells[41, 8] = _FilterControls.CapacitiveCouplingCtrlReg2.CoupleUFLB.ToString();
            ex.xlWorkSheet.Cells[42, 4] = "LFUB Capacitive Coupling Tuning";
            ex.xlWorkSheet.Cells[42, 8] = _FilterControls.CapacitiveCouplingCtrlReg2.CoupleLFUB.ToString();
            ex.xlWorkSheet.get_Range("H23", "H42").HorizontalAlignment = Microsoft.Office.Interop.Excel.XlHAlign.xlHAlignCenter;

            ex.SaveXLSFile(fName);
        }

        public string printActiveLow(int val)
        {
            return (val == 0 ? "Enabled" : "Disabled");
        }
    }

    public class SweepQTune
    {
        private int _Itteration;
        private int _StartQ, _StopQ;
        private int _NqCur, _VqCur;
        private List<int> _NqCurList;
        private List<int> _VqCurList;
        private bool _isDone;

        public SweepQTune()
        {
            Reset();
        }

        public int Itteration
        {
            get { return _Itteration; }
            set { _Itteration = value; }
        }

        public int StartQ
        {
            get { return _StartQ; }
            set { _StartQ = value; }
        }

        public int StopQ
        {
            get { return _StopQ; }
            set { _StopQ = value; }
        }

        public int NqCur
        {
            get { return _NqCur; }
            set { _NqCur = value; }
        }

        public int VqCur
        {
            get { return _VqCur; }
            set { _VqCur = value; }
        }

        public List<int> NqCurList
        {
            get { return _NqCurList; }
        }

        public List<int> VqCurList
        {
            get { return _VqCurList; }
        }

        public bool isDone
        {
            get { return _isDone; }
            set { _isDone = true; }
        }

        public void Reset()
        {
            _Itteration = -1;
            _StartQ = -1;
            _StopQ = -1;
            _NqCurList = new List<int>();
            _VqCurList = new List<int>();
            _isDone = false;
        }
    }
}
```

FRMQTUNEDEBUG.CS

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace QEFilterV2TestAppV2
{
    public partial class frmQTuneDebug : Form
    {
        private QTuneDebug _QTuneDebug;
        private DebugText _DebugText;
        private CommunicationManager _ComManager;
        private FilterControls _FilterControls;
        private Timer _ResponseTimer;
        private Timer _AutomaticStepTimer;
        private int _startTime;
        private bool _isRunning;
        private bool _isSweepDone;
        private bool _isSweepAutomated;
```

```
public frmQTuneDebug(QTuneDebug qtd, DebugText dt, CommunicationManager cm, FilterControls fs)
{
    InitializeComponent();
    this.Closing += new CancelEventHandler(frmQTuneDebug_Closing);
    _QTuneDebug = qtd;
    _DebugText = dt;
    _ComManager = cm;
    _FilterControls = fs;
    _ResponseTimer = new Timer();
    _ResponseTimer.Interval = 10;
    _ResponseTimer.Tick += new EventHandler(ResponseTimer_Tick);
    _AutomaticStepTimer = new Timer();
    _AutomaticStepTimer.Interval = 250; //250mS
    _AutomaticStepTimer.Tick += new EventHandler(AutomaticStepTimer_Tick);
    _isRunning = false;
    _isSweepDone = false;
    _isSweepAutomated = false;
    ResetAllControls();
}

private void frmQTuneDebug_Closing(object sender, CancelEventArgs cArgs)
{
    if (!_QTuneDebug.OkToClose)
        cArgs.Cancel = true;
    else
        cArgs.Cancel = false;
}

private void ResetAllControls()
{
    radFrontEnd.Checked = true;
    radBackEnd.Checked = false;
    btnQTuningStep.Enabled = false;
    ResetSweepControls();
}

private void ResetSweepControls()
{
    numStartQ.Minimum = FilterControls.DIGQTUNEMIN;
    numStartQ.Maximum = FilterControls.DIGQTUNEMAX;
    numStopQ.Minimum = FilterControls.DIGQTUNEMIN;
    numStopQ.Maximum = FilterControls.DIGQTUNEMAX;
    numStartQ.Value = numStartQ.Minimum;
    numStopQ.Value = numStopQ.Maximum;
    txtCurQ.Text = QTuneDebug.VALUE_UNKNOWN;
    txtCurV.Text = QTuneDebug.VALUE_UNKNOWN;
}

private void radFrontEnd_CheckedChanged(object sender, EventArgs e)
{
    if (radFrontEnd.Checked)
    {
        radBackEnd.Checked = false;
        _QTuneDebug.SelectedPole = (int)QTuneDebug.PoleSelection.FrontEnd;
    }
    else if (radBackEnd.Checked)
    {
        radFrontEnd.Checked = false;
        _QTuneDebug.SelectedPole = (int)QTuneDebug.PoleSelection.BackEnd;
    }
    Reset();
}

private void radBackEnd_CheckedChanged(object sender, EventArgs e)
{
    if (radFrontEnd.Checked)
    {
        radBackEnd.Checked = false;
        _QTuneDebug.SelectedPole = (int)QTuneDebug.PoleSelection.FrontEnd;
    }
    else if (radBackEnd.Checked)
    {
        radFrontEnd.Checked = false;
        _QTuneDebug.SelectedPole = (int)QTuneDebug.PoleSelection.BackEnd;
    }
    Reset();
}

private void btnQTuningDebugReset_Click(object sender, EventArgs e)
{
    Reset();
}

private void Reset()
{
    _isRunning = false;
    _isSweepAutomated = false;
    _QTuneDebug.Run(false);
    ResetSweepControls();
    btnQTuningRun.Text = "Run";
    btnQTuningStep.Enabled = false;
}

private void btnQTuningStep_Click(object sender, EventArgs e)
{
    if (!_ComManager.isOpen)
    {
        MessageBox.Show("Selected COM Port is not open.");
        return;
    }
    step();
}

private void step()
{
    if (_QTuneDebug.SweepQTune.Itteration == -1)
    {
        _QTuneDebug.SweepQTune.StartQ = int.Parse(numStartQ.Value.ToString());
        _QTuneDebug.SweepQTune.StopQ = int.Parse(numStopQ.Value.ToString());
        _QTuneDebug.SweepQTune.NqCur = _QTuneDebug.SweepQTune.StartQ;
    }
    _isSweepDone = _QTuneDebug.Step();
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    //_QTuneDebug.UpdateSweepInfo(_QTuneDebug.SweepQTune.NqCur,(new Random().Next(FilterControls.DIGQTUNEMIN,FilterControls.DIGQTUNEMAX)));
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    _startTime = DateTime.Now.Second;
    _ResponseTimer.Start();
}
```

```
        private void btnQTuningRun_Click(object sender, EventArgs e)
        {
            if (!_ComManager.isOpen)
            {
                MessageBox.Show("Selected COM Port is not open.");
                return;
            }
            if (numStartQ.Value == numStopQ.Value)
            {
                MessageBox.Show("Start and Stop values are the same, canceling sweep.");
                return;
            }

            _isRunning = !_isRunning;
            if (!_isRunning)
            {
                int tmpStartQ = int.Parse(numStartQ.Value.ToString());
                int tmpStopQ = int.Parse(numStopQ.Value.ToString());
                _QTuneDebug.Run(false);
                btnQTuningRun.Text = "Run";
                btnQTuningStep.Enabled = false;
                ResetSweepControls();
                numStartQ.Value = tmpStartQ;
                numStopQ.Value = tmpStopQ;
                _isSweepAutomated = false;
            }
            else
            {
                DialogResult SweepType = MessageBox.Show("Would you like to automate the sweep?","Automate Sweep?", MessageBoxButtons.YesNo);
                txtCurQ.Text = QTuneDebug.VALUE_UNKNOWN;
                txtCurV.Text = QTuneDebug.VALUE_UNKNOWN;
                _QTuneDebug.Run(true);
                btnQTuningRun.Text = "Stop";
                btnQTuningStep.Enabled = true;
                switch (SweepType)
                {
                    case DialogResult.Yes:
                        _isSweepAutomated = true;
                        step();
                        break;
                    case DialogResult.No:
                        _isSweepAutomated = false;
                        break;
                    default:
                        return;
                }
            }
        }

        public void ResponseTimer_Tick(object sender, EventArgs eArgs)
        {
            if (sender == _ResponseTimer)
            {
                if (!_QTuneDebug.RxDataAvailable && _QTuneDebug.SweepQTune.Itteration != -1)
                {
                    btnQTuningStep.Enabled = false;
                    if (DateTime.Now.Second >= _startTime + 5) // 5 Seconds Have Elapsed w/o RxData
                    {
                        _ResponseTimer.Stop();
                        MessageBox.Show("No information has Been received for 5 seconds, exiting current sweep.");
                        _QTuneDebug.Run(false);
                        Reset();
                    }
                }
                else if (_QTuneDebug.RxDataAvailable && _QTuneDebug.SweepQTune.NqCurList.Count > 0)
                {
                    _ResponseTimer.Stop();
                    int i = _QTuneDebug.SweepQTune.NqCurList.Count - 1;
                    if (i > -1)
                    {
                        txtCurQ.Text = _QTuneDebug.SweepQTune.NqCurList.ElementAt(i).ToString();
                        txtCurV.Text = _QTuneDebug.SweepQTune.VqCurList.ElementAt(i).ToString();
                        _QTuneDebug.RxDataAvailable = false;
                        btnQTuningStep.Enabled = true;

                        if (_isSweepDone)
                        {
                            _isRunning = false;
                            _isSweepDone = false;
                            btnQTuningRun.Text = "Run";
                            btnQTuningStep.Enabled = false;
                            DialogResult result = MessageBox.Show("Q-Tuning Sweep is Complete!\nWould you like to save it as an Excel Spreadsheet?", "Save
Debug Data", MessageBoxButtons.YesNo);
                            if (result == System.Windows.Forms.DialogResult.Yes)
                                _QTuneDebug.SaveData();

                            _QTuneDebug.Run(false);
                            return;
                        }
                        else if (_isSweepAutomated)
                            _AutomaticStepTimer.Start();
                    }
                }
            }
        }

        public void AutomaticStepTimer_Tick(object sender, EventArgs eArgs)
        {
            if (sender == _AutomaticStepTimer)
            {
                _AutomaticStepTimer.Stop();
                step();
            }
        }

    }
}
```

FRMQTUNEDEBUG.DESIGNER.CS

```
namespace QEFilterV2TestAppV2
{
    partial class frmQTuneDebug
    {
        /// <summary>
        /// Required designer variable.
```

```
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.groupFilterSettings = new System.Windows.Forms.GroupBox();
            this.panel1 = new System.Windows.Forms.Panel();
            this.radBackEnd = new System.Windows.Forms.RadioButton();
            this.radFrontEnd = new System.Windows.Forms.RadioButton();
            this.btnQTuningRun = new System.Windows.Forms.Button();
            this.btnQTuningStep = new System.Windows.Forms.Button();
            this.btnQTuningDebugReset = new System.Windows.Forms.Button();
            this.label12 = new System.Windows.Forms.Label();
            this.txtCurV = new System.Windows.Forms.TextBox();
            this.label13 = new System.Windows.Forms.Label();
            this.txtCurQ = new System.Windows.Forms.TextBox();
            this.label14 = new System.Windows.Forms.Label();
            this.numStopQ = new System.Windows.Forms.NumericUpDown();
            this.numStartQ = new System.Windows.Forms.NumericUpDown();
            this.label15 = new System.Windows.Forms.Label();
            this.groupFilterSettings.SuspendLayout();
            this.panel1.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(this.numStopQ)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numStartQ)).BeginInit();
            this.SuspendLayout();
            //
            // groupFilterSettings
            //
            this.groupFilterSettings.Controls.Add(this.panel1);
            this.groupFilterSettings.Controls.Add(this.btnQTuningRun);
            this.groupFilterSettings.Controls.Add(this.btnQTuningStep);
            this.groupFilterSettings.Controls.Add(this.btnQTuningDebugReset);
            this.groupFilterSettings.Controls.Add(this.label12);
            this.groupFilterSettings.Controls.Add(this.txtCurV);
            this.groupFilterSettings.Controls.Add(this.label13);
            this.groupFilterSettings.Controls.Add(this.txtCurQ);
            this.groupFilterSettings.Controls.Add(this.label14);
            this.groupFilterSettings.Controls.Add(this.numStopQ);
            this.groupFilterSettings.Controls.Add(this.numStartQ);
            this.groupFilterSettings.Controls.Add(this.label15);
            this.groupFilterSettings.Font = new System.Drawing.Font("Cambria", 12F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.groupFilterSettings.Location = new System.Drawing.Point(12, 12);
            this.groupFilterSettings.Name = "groupFilterSettings";
            this.groupFilterSettings.Size = new System.Drawing.Size(462, 153);
            this.groupFilterSettings.TabIndex = 5;
            this.groupFilterSettings.TabStop = false;
            this.groupFilterSettings.Text = "Q-Tuning Debug";
            //
            // panel1
            //
            this.panel1.Controls.Add(this.radBackEnd);
            this.panel1.Controls.Add(this.radFrontEnd);
            this.panel1.Location = new System.Drawing.Point(74, 25);
            this.panel1.Name = "panel1";
            this.panel1.Size = new System.Drawing.Size(314, 36);
            this.panel1.TabIndex = 66;
            //
            // radBackEnd
            //
            this.radBackEnd.AutoSize = true;
            this.radBackEnd.Location = new System.Drawing.Point(181, 3);
            this.radBackEnd.Name = "radBackEnd";
            this.radBackEnd.Size = new System.Drawing.Size(97, 23);
            this.radBackEnd.TabIndex = 37;
            this.radBackEnd.TabStop = true;
            this.radBackEnd.Text = "Back-End";
            this.radBackEnd.UseVisualStyleBackColor = true;
            this.radBackEnd.CheckedChanged += new System.EventHandler(this.radBackEnd_CheckedChanged);
            //
            // radFrontEnd
            //
            this.radFrontEnd.AutoSize = true;
            this.radFrontEnd.Location = new System.Drawing.Point(31, 3);
            this.radFrontEnd.Name = "radFrontEnd";
            this.radFrontEnd.Size = new System.Drawing.Size(102, 23);
            this.radFrontEnd.TabIndex = 36;
            this.radFrontEnd.TabStop = true;
            this.radFrontEnd.Text = "Front-End";
            this.radFrontEnd.UseVisualStyleBackColor = true;
            this.radFrontEnd.CheckedChanged += new System.EventHandler(this.radFrontEnd_CheckedChanged);
            //
            // btnQTuningRun
            //
            this.btnQTuningRun.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.btnQTuningRun.Location = new System.Drawing.Point(297, 120);
            this.btnQTuningRun.Name = "btnQTuningRun";
            this.btnQTuningRun.Size = new System.Drawing.Size(90, 23);
            this.btnQTuningRun.TabIndex = 64;
            this.btnQTuningRun.Text = "Run";
            this.btnQTuningRun.UseVisualStyleBackColor = true;
            this.btnQTuningRun.Click += new System.EventHandler(this.btnQTuningRun_Click);
            //
            // btnQTuningStep
            //
            this.btnQTuningStep.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.btnQTuningStep.Location = new System.Drawing.Point(187, 120);
            this.btnQTuningStep.Name = "btnQTuningStep";
            this.btnQTuningStep.Size = new System.Drawing.Size(90, 23);
```

```
        this.btnQTuningStep.TabIndex = 62;
        this.btnQTuningStep.Text = "Step";
        this.btnQTuningStep.UseVisualStyleBackColor = true;
        this.btnQTuningStep.Click += new System.EventHandler(this.btnQTuningStep_Click);
        //
        // btnQTuningDebugReset
        //
        this.btnQTuningDebugReset.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
        this.btnQTuningDebugReset.Location = new System.Drawing.Point(76, 120);
        this.btnQTuningDebugReset.Name = "btnQTuningDebugReset";
        this.btnQTuningDebugReset.Size = new System.Drawing.Size(90, 23);
        this.btnQTuningDebugReset.TabIndex = 61;
        this.btnQTuningDebugReset.Text = "Reset";
        this.btnQTuningDebugReset.UseVisualStyleBackColor = true;
        this.btnQTuningDebugReset.Click += new System.EventHandler(this.btnQTuningDebugReset_Click);
        //
        // label12
        //
        this.label12.AutoSize = true;
        this.label12.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
        this.label12.Location = new System.Drawing.Point(372, 68);
        this.label12.Name = "label12";
        this.label12.Size = new System.Drawing.Size(51, 13);
        this.label12.TabIndex = 58;
        this.label12.Text = "Current V";
        //
        // txtCurV
        //
        this.txtCurV.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
        this.txtCurV.Location = new System.Drawing.Point(367, 84);
        this.txtCurV.Name = "txtCurV";
        this.txtCurV.ReadOnly = true;
        this.txtCurV.Size = new System.Drawing.Size(61, 20);
        this.txtCurV.TabIndex = 57;
        //
        // label13
        //
        this.label13.AutoSize = true;
        this.label13.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
        this.label13.Location = new System.Drawing.Point(261, 68);
        this.label13.Name = "label13";
        this.label13.Size = new System.Drawing.Size(52, 13);
        this.label13.TabIndex = 56;
        this.label13.Text = "Current Q";
        //
        // txtCurQ
        //
        this.txtCurQ.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
        this.txtCurQ.Location = new System.Drawing.Point(257, 84);
        this.txtCurQ.Name = "txtCurQ";
        this.txtCurQ.ReadOnly = true;
        this.txtCurQ.Size = new System.Drawing.Size(61, 20);
        this.txtCurQ.TabIndex = 52;
        //
        // label14
        //
        this.label14.AutoSize = true;
        this.label14.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
        this.label14.Location = new System.Drawing.Point(157, 68);
        this.label14.Name = "label14";
        this.label14.Size = new System.Drawing.Size(40, 13);
        this.label14.TabIndex = 55;
        this.label14.Text = "Stop Q";
        //
        // numStopQ
        //
        this.numStopQ.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
        this.numStopQ.Location = new System.Drawing.Point(147, 84);
        this.numStopQ.Name = "numStopQ";
        this.numStopQ.Size = new System.Drawing.Size(61, 20);
        this.numStopQ.TabIndex = 54;
        //
        // numStartQ
        //
        this.numStartQ.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
        this.numStartQ.Location = new System.Drawing.Point(36, 84);
        this.numStartQ.Name = "numStartQ";
        this.numStartQ.Size = new System.Drawing.Size(61, 20);
        this.numStartQ.TabIndex = 53;
        //
        // label15
        //
        this.label15.AutoSize = true;
        this.label15.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
        this.label15.Location = new System.Drawing.Point(46, 68);
        this.label15.Name = "label15";
        this.label15.Size = new System.Drawing.Size(40, 13);
        this.label15.TabIndex = 51;
        this.label15.Text = "Start Q";
        //
        // frmQTuneDebug
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(487, 177);
        this.Controls.Add(this.groupFilterSettings);
        this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
        this.MaximizeBox = false;
        this.Name = "frmQTuneDebug";
        this.Text = "Q-Tuning Debug";
        this.groupFilterSettings.ResumeLayout(false);
        this.groupFilterSettings.PerformLayout();
        this.panel1.ResumeLayout(false);
        this.panel1.PerformLayout();
        ((System.ComponentModel.ISupportInitialize)(this.numStopQ)).EndInit();
        ((System.ComponentModel.ISupportInitialize)(this.numStartQ)).EndInit();
        this.ResumeLayout(false);

    }

    #endregion

    private System.Windows.Forms.GroupBox groupFilterSettings;
    private System.Windows.Forms.Label label12;
    private System.Windows.Forms.TextBox txtCurV;
    private System.Windows.Forms.Label label13;
    private System.Windows.Forms.TextBox txtCurQ;
    private System.Windows.Forms.Label label14;
```

```
        private System.Windows.Forms.NumericUpDown numStopQ;
        private System.Windows.Forms.NumericUpDown numStartQ;
        private System.Windows.Forms.Label label15;
        private System.Windows.Forms.Button btnQTuningRun;
        private System.Windows.Forms.Button btnQTuningStep;
        private System.Windows.Forms.Button btnQTuningDebugReset;
        private System.Windows.Forms.Panel panel1;
        private System.Windows.Forms.RadioButton radBackEnd;
        private System.Windows.Forms.RadioButton radFrontEnd;

    }
}
```

EXCELINTERFACE.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Excel = Microsoft.Office.Interop.Excel;
using System.Windows.Forms;

namespace QEFilterV2TestAppV2
{
    public class ExcelInterface
    {
        private Excel.Application _xlApp;
        private Excel.Workbook _xlWorkBook;
        private Excel.Worksheet _xlWorkSheet;
        private Excel.Range _chartRange;
        private Excel.Series _chartSeries;
        private Excel.ChartObjects _xlCharts;
        private Excel.ChartObject _myChart;
        private Excel.Chart _chartPage;
        private Excel.Axis _axis;

        private object _misValue;

        public ExcelInterface(string title, string xlabel, string ylabel, double widthInches, double heightInches)
        {
            try
            {
                widthInches *= 72;   // Convert to points (1 in = 72 pts)
                heightInches *= 72;  // Convert to points (1 in = 72 pts)

                _misValue = System.Reflection.Missing.Value;
                _xlApp = new Excel.ApplicationClass();
                //_xlApp.Visible = true;
                _xlWorkBook = _xlApp.Workbooks.Add(_misValue);
                _xlWorkSheet = (Excel.Worksheet)_xlWorkBook.Worksheets.get_Item(1);
                _xlCharts = (Excel.ChartObjects)_xlWorkSheet.ChartObjects(Type.Missing);
                _myChart = (Excel.ChartObject)_xlCharts.Add(145, 60, widthInches, heightInches);
                _chartPage = _myChart.Chart;

                _axis = (Excel.Axis)_chartPage.Axes(Excel.XlAxisType.xlCategory, Excel.XlAxisGroup.xlPrimary);
                _axis.HasTitle = true;
                _axis.AxisTitle.Text = xlabel;
                _axis = (Excel.Axis)_chartPage.Axes(Excel.XlAxisType.xlValue, Excel.XlAxisGroup.xlPrimary);
                _axis.HasTitle = true;
                _axis.AxisTitle.Text = ylabel;
                _chartPage.HasTitle = true;
                _chartPage.ChartTitle.Text = title;
                _chartPage.ChartType = Microsoft.Office.Interop.Excel.XlChartType.xlLineMarkers;
                _chartPage.HasLegend = false;
                _chartPage.ChartStyle = 33;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message.ToString() + "\n" + ex.StackTrace.ToString());
            }
        }

        public Excel.Worksheet xlWorkSheet
        {
            get { return _xlWorkSheet; }
            set { _xlWorkSheet = value; }
        }

        public Excel.Range chartRange
        {
            get { return _chartRange; }
            set { _chartRange = value; }
        }

        public Excel.Series chartSeries
        {
            get { return _chartSeries; }
            set { _chartSeries = value; }
        }

        public Excel.ChartObjects xlCharts
        {
            get { return _xlCharts; }
            set { _xlCharts = value; }
        }

        public Excel.ChartObject myChart
        {
            get { return _myChart; }
            set { _myChart = value; }
        }

        public Excel.Chart chartPage
        {
            get { return _chartPage; }
            set { _chartPage = value; }
        }

        public Excel.Axis axis
        {
            get { return _axis; }
            set { _axis = value; }
        }

        public object misValue
```

```
        {
            get { return _misValue; }
        }

        public void SaveXLSFile(string fName)
        {
            SaveFileDialog SaveXLS = new SaveFileDialog();
            try
            {
                SaveXLS.DefaultExt = "txt";
                SaveXLS.Filter = "Excel Spreadsheet (*.xls)|*.xls";
                SaveXLS.AddExtension = true;
                SaveXLS.RestoreDirectory = true;
                SaveXLS.Title = "Save Debug Information as...";
                SaveXLS.InitialDirectory = @"C:/";
                SaveXLS.FileName = fName;
                SaveXLS.ShowDialog();

                if (SaveXLS.FileName == "")
                    return;

                _xlWorkBook.SaveAs(SaveXLS.FileName, Excel.XlFileFormat.xlWorkbookNormal, _misValue, _misValue, _misValue, _misValue,
Excel.XlSaveAsAccessMode.xlExclusive, _misValue, _misValue, _misValue, _misValue, _misValue);
                _xlWorkBook.Close(true, _misValue, _misValue);
                _xlApp.Quit();

                releaseObject(_xlWorkSheet);
                releaseObject(_xlWorkBook);
                releaseObject(_xlApp);

                MessageBox.Show(SaveXLS.FileName + " has been created.");
            }
            catch (Exception ex)
            {
                if (ex.Message.Contains("Cannot access"))
                {
                    MessageBox.Show("Please close " + SaveXLS.FileName + "\nand try again or pick a new file name");
                    SaveXLSFile(fName);
                    return;
                }
                MessageBox.Show(ex.Message);
            }


        }
        private void releaseObject(object obj)
        {
            try
            {
                System.Runtime.InteropServices.Marshal.ReleaseComObject(obj);
                obj = null;
            }
            catch (Exception ex)
            {
                obj = null;
                MessageBox.Show("Exception Occured while releasing object " + ex.ToString());
            }
            finally
            {
                GC.Collect();
            }
        }
    }
}
```

ALGORITHMSETTINGS.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace QEFilterV2TestAppV2
{

    public class AlgorithmSettings
    {
        public const decimal DEFAULTCENTERFREQ = 500.0M;
        public const decimal MINIMUMCENTERFREQ = 400.0M;
        public const decimal MAXIMUMCENTERFREQ = 600.0M; // 600.0 -- 6000 -- 13 Bits (4 Hex Digits)
        public const decimal DEFAULTFREQTOL = 0.2M;
        public const decimal MINIMUMFREQTOL = 0.0M;
        public const decimal MAXIMUMFREQTOL = 10.0M;     // 10.0 -- 100 -- 7 Bits (2 Hex Digits)
        public const int DEFAULTADTHRESHOLD1 = 10;
        public const int MINIMUMADTHRESHOLD1 = 0;
        public const int MAXIMUMADTHRESHOLD1 = 127;      // 7 Bits (2 Hex Digits)
        public const int DEFAULTADTHRESHOLD2 = 10;
        public const int MINIMUMADTHRESHOLD2 = 0;
        public const int MAXIMUMADTHRESHOLD2 = 127;      // 7 Bits (2 Hex Digits)
        public const int DEFAULTFREQOFFSET = 0;
        public const int MINIMUMFREQOFFSET = 0;
        public const int MAXIMUMFREQOFFSET = 15;         // 4 Bits (1 Hex Digits)
        public const int DEFAULTQOFFSET = 2;
        public const int MINIMUMQOFFSET = 0;
        public const int MAXIMUMQOFFSET = 10;            // 4 Bits (1 Hex Digits)
        public const int DEFAULTQBACKOFF = 1;
        public const int MINIMUMQBACKOFF = 0;
        public const int MAXIMUMQBACKOFF = 15;           // 4 Bits (1 Hex Digits)
        public const int DEFAULTINTERVAL = 2;
        public const int MINIMUMINTERVAL = 1;
        public const int MAXIMUMINTERVAL = 300;

        private CommunicationManager _ComManager;
        private DebugText _DebugText;
        private AlgorithmCtrlReg _AlgorithmCtrlReg;
        private bool _OkToClose;
        private string _FilterCMDs;

        public AlgorithmSettings(CommunicationManager cm, DebugText dt)
        {
            _ComManager = cm;
            _DebugText = dt;
            _AlgorithmCtrlReg = new AlgorithmCtrlReg();
            _OkToClose = false;
```

```
            _FilterCMDs = string.Empty;
        }

        public AlgorithmCtrlReg AlgorithmCtrlReg
        {
            get { return _AlgorithmCtrlReg; }
        }

        public bool OkToClose
        {
            get { return _OkToClose; }
            set { _OkToClose = value; }
        }

        public void RunAlgorithm()
        {
            UpdateFilterCMDs();
            _ComManager.WriteData(_FilterCMDs);
        }

        private void UpdateFilterCMDs()
        {
            _FilterCMDs = "@";
            _FilterCMDs += _AlgorithmCtrlReg.CenterFreq.ToString("X").PadLeft(4, '0') + "&";
            _FilterCMDs += _AlgorithmCtrlReg.FreqTol.ToString("X").PadLeft(2, '0') + "&";
            _FilterCMDs += _AlgorithmCtrlReg.FrontEndAD.ToString("X").PadLeft(4, '0') + "&";
            _FilterCMDs += _AlgorithmCtrlReg.FrontEndQF.ToString("X").PadLeft(3, '0') + "&";
            _FilterCMDs += _AlgorithmCtrlReg.BackEndAD.ToString("X").PadLeft(4, '0') + "&";
            _FilterCMDs += _AlgorithmCtrlReg.BackEndQF.ToString("X").PadLeft(3, '0') + "&";
            _FilterCMDs += _AlgorithmCtrlReg.CoupLat.ToString("X").PadLeft(3, '0') + "&";
            _FilterCMDs += _AlgorithmCtrlReg.CoupCross.ToString("X").PadLeft(3, '0') + "&";
            _FilterCMDs += _AlgorithmCtrlReg.Debug.ToString("X").PadLeft(1, '0') + "|";
        }
    }

    public class AlgorithmCtrlReg
    {
        private int _CenterFreq;
        private int _FreqTol;
        private int _FrontEndADThreshold1;
        private int _FrontEndADThreshold2;
        private int _FrontEndFOffset;
        private int _FrontEndQOffset;
        private int _FrontEndQBackOff;
        private int _BackEndADThreshold1;
        private int _BackEndADThreshold2;
        private int _BackEndFOffset;
        private int _BackEndQOffset;
        private int _BackEndQBackOff;
        private int _CouplingUpper;
        private int _CouplingLower;
        private int _CouplingUFLB;
        private int _CouplingLFUB;
        private int _PrintAlgorithmSettings;
        private int _PrintFilterSettings;
        private int _PrintItterations;

        public AlgorithmCtrlReg()
        {
            _CenterFreq = int.Parse((10*double.Parse(AlgorithmSettings.DEFAULTCENTERFREQ.ToString())).ToString());
            _FreqTol = int.Parse((10 * double.Parse(AlgorithmSettings.DEFAULTFREQTOL.ToString())).ToString());
            _FrontEndADThreshold1 = AlgorithmSettings.DEFAULTADTHRESHOLD1;
            _FrontEndADThreshold2 = AlgorithmSettings.DEFAULTADTHRESHOLD2;
            _FrontEndFOffset = AlgorithmSettings.DEFAULTFREQOFFSET;
            _FrontEndQOffset = AlgorithmSettings.DEFAULTQOFFSET;
            _FrontEndQBackOff = AlgorithmSettings.DEFAULTQBACKOFF;
            _BackEndADThreshold1 = AlgorithmSettings.DEFAULTADTHRESHOLD1;
            _BackEndADThreshold2 = AlgorithmSettings.DEFAULTADTHRESHOLD2;
            _BackEndFOffset = AlgorithmSettings.DEFAULTFREQOFFSET;
            _BackEndQOffset = AlgorithmSettings.DEFAULTQOFFSET;
            _BackEndQBackOff = AlgorithmSettings.DEFAULTQBACKOFF;
            _CouplingUpper = FilterControls.COUPLEMIN;
            _CouplingLower = FilterControls.COUPLEMIN;
            _CouplingUFLB = FilterControls.COUPLEMIN;
            _CouplingLFUB = FilterControls.COUPLEMIN;
            _PrintAlgorithmSettings = 0;
            _PrintFilterSettings = 0;
            _PrintItterations = 0;
        }

        public int CenterFreq
        {
            get { return _CenterFreq; }
            set { _CenterFreq = value; }
        }

        public int FreqTol
        {
            get { return _FreqTol; }
            set { _FreqTol = value; }
        }

        public int FrontEndADThreshold1
        {
            get { return _FrontEndADThreshold1; }
            set { _FrontEndADThreshold1 = value; }
        }

        public int FrontEndADThreshold2
        {
            get { return _FrontEndADThreshold2; }
            set { _FrontEndADThreshold2 = value; }
        }

        public int FrontEndFOffset
        {
            get { return _FrontEndFOffset; }
            set { _FrontEndFOffset = value; }
        }

        public int FrontEndQOffset
        {
            get { return _FrontEndQOffset; }
            set { _FrontEndQOffset = value; }
        }

        public int FrontEndQBackOff
        {
```

```
    get { return _FrontEndQBackOff; }
    set { _FrontEndQBackOff = value; }
}

public int BackEndADThreshold1
{
    get { return _BackEndADThreshold1; }
    set { _BackEndADThreshold1 = value; }
}

public int BackEndADThreshold2
{
    get { return _BackEndADThreshold2; }
    set { _BackEndADThreshold2 = value; }
}

public int BackEndFOffset
{
    get { return _BackEndFOffset; }
    set { _BackEndFOffset = value; }
}

public int BackEndQOffset
{
    get { return _BackEndQOffset; }
    set { _BackEndQOffset = value; }
}

public int BackEndQBackOff
{
    get { return _BackEndQBackOff; }
    set { _BackEndQBackOff = value; }
}

public int CouplingUpper
{
    get { return _CouplingUpper; }
    set { _CouplingUpper = value; }
}

public int CouplingLower
{
    get { return _CouplingLower; }
    set { _CouplingLower = value; }
}

public int CouplingUFLB
{
    get { return _CouplingUFLB; }
    set { _CouplingUFLB = value; }
}

public int CouplingLFUB
{
    get { return _CouplingLFUB; }
    set { _CouplingLFUB = value; }
}

public int PrintAlgorithmSettings
{
    get { return _PrintAlgorithmSettings; }
    set { _PrintAlgorithmSettings = value; }
}

public int PrintFilterSettings
{
    get { return _PrintFilterSettings; }
    set { _PrintFilterSettings = value; }
}

public int PrintItterations
{
    get { return _PrintItterations; }
    set { _PrintItterations = value; }
}


public int FrontEndAD
{
    get
    {
        return (_FrontEndADThreshold1 & 0x7F) << 7 | (_FrontEndADThreshold2 & 0x7F);
    }
}

public int FrontEndQF
{
    get
    {
        return (_FrontEndQOffset & 0xF) << 8 | (_FrontEndQBackOff & 0xF) << 4 | (_FrontEndFOffset & 0xF);
    }
}

public int BackEndAD
{
    get
    {
        return (_BackEndADThreshold1 & 0x7F) << 7 | (_BackEndADThreshold2 & 0x7F);
    }
}

public int BackEndQF
{
    get
    {
        return (_BackEndQOffset & 0xF) << 8 | (_BackEndQBackOff & 0xF) << 4 | (_BackEndFOffset & 0xF);
    }
}

public int CoupLat
{
    get
    {
        return (_CouplingUpper & 0x3F) << 6 | (_CouplingLower & 0x3F);
    }
}

public int CoupCross
{
    get
    {
        return (_CouplingUFLB & 0x3F) << 6 | (_CouplingLFUB & 0x3F);
```

```
            }
        }

        public int Debug
        {
            get
            {
                return (_PrintAlgorithmSettings & 1) << 2 | (_PrintFilterSettings & 1) << 1 | (_PrintItterations & 1);
            }
        }
    }
}
```

FRMALGORITHMSETTINGS.CS

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace QEFilterV2TestAppV2
{
    public partial class frmAlgorithmSettings : Form
    {
        private AlgorithmSettings _AlgorithmSettings;
        private DebugText _DebugText;
        private CommunicationManager _ComManager;
        private Timer _AlgorithmTimer;
        private bool _isRunning, _isAutomated;
        private int _Tick;

        public frmAlgorithmSettings(AlgorithmSettings alg, DebugText dt, CommunicationManager cm)
        {
            InitializeComponent();
            this.Closing += new CancelEventHandler(frmAlgorithmSettings_Closing);
            _AlgorithmSettings = alg;
            _DebugText = dt;
            _ComManager = cm;

            _AlgorithmTimer = new Timer();
            _AlgorithmTimer.Tick += new EventHandler(AlgorithmTimer_Tick);
            _AlgorithmTimer.Interval = 1000; //1000 mS

            _isRunning = false;
            _isAutomated = false;
            _Tick = 0;

            InitalizeAlgorithmControls();
        }


        private void frmAlgorithmSettings_Closing(object sender, CancelEventArgs cArgs)
        {
            if (!_AlgorithmSettings.OkToClose)
                cArgs.Cancel = true;
            else
                cArgs.Cancel = false;
        }

        private void InitalizeAlgorithmControls()
        {
            numCenterFreq.Value = AlgorithmSettings.DEFAULTCENTERFREQ;
            numCenterFreq.Minimum = AlgorithmSettings.MINIMUMCENTERFREQ;
            numCenterFreq.Maximum = AlgorithmSettings.MAXIMUMCENTERFREQ;
            numFreqTol.Value = AlgorithmSettings.DEFAULTFREQTOL;
            numFreqTol.Minimum = AlgorithmSettings.MINIMUMFREQTOL;
            numFreqTol.Maximum = AlgorithmSettings.MAXIMUMFREQTOL;
            numAutoTime.Value = AlgorithmSettings.DEFAULTINTERVAL;
            numAutoTime.Minimum = AlgorithmSettings.MINIMUMINTERVAL;
            numAutoTime.Maximum = AlgorithmSettings.MAXIMUMINTERVAL;
            numFrontEndADThresh1.Value = AlgorithmSettings.DEFAULTADTHRESHOLD1;
            numFrontEndADThresh1.Minimum = AlgorithmSettings.MINIMUMADTHRESHOLD1;
            numFrontEndADThresh1.Maximum = AlgorithmSettings.MAXIMUMADTHRESHOLD1;
            numFrontEndADThresh2.Value = AlgorithmSettings.DEFAULTADTHRESHOLD2;
            numFrontEndADThresh2.Minimum = AlgorithmSettings.MINIMUMADTHRESHOLD2;
            numFrontEndADThresh2.Maximum = AlgorithmSettings.MAXIMUMADTHRESHOLD2;
            numFrontEndQOffset.Value = AlgorithmSettings.DEFAULTQOFFSET;
            numFrontEndQOffset.Minimum = AlgorithmSettings.MINIMUMQOFFSET;
            numFrontEndQOffset.Maximum = AlgorithmSettings.MAXIMUMQOFFSET;
            numFrontEndQBackOff.Value = AlgorithmSettings.DEFAULTQBACKOFF;
            numFrontEndQBackOff.Minimum = AlgorithmSettings.MINIMUMQBACKOFF;
            numFrontEndQBackOff.Maximum = AlgorithmSettings.MAXIMUMQBACKOFF;
            numFrontEndFOffset.Value = AlgorithmSettings.DEFAULTFREQOFFSET;
            numFrontEndFOffset.Minimum = AlgorithmSettings.MINIMUMFREQOFFSET;
            numFrontEndFOffset.Maximum = AlgorithmSettings.MAXIMUMFREQOFFSET;
            numBackEndADThresh1.Value = AlgorithmSettings.DEFAULTADTHRESHOLD1;
            numBackEndADThresh1.Minimum = AlgorithmSettings.MINIMUMADTHRESHOLD1;
            numBackEndADThresh1.Maximum = AlgorithmSettings.MAXIMUMADTHRESHOLD1;
            numBackEndADThresh2.Value = AlgorithmSettings.DEFAULTADTHRESHOLD2;
            numBackEndADThresh2.Minimum = AlgorithmSettings.MINIMUMADTHRESHOLD2;
            numBackEndADThresh2.Maximum = AlgorithmSettings.MAXIMUMADTHRESHOLD2;
            numBackEndQOffset.Value = AlgorithmSettings.DEFAULTQOFFSET;
            numBackEndQOffset.Minimum = AlgorithmSettings.MINIMUMQOFFSET;
            numBackEndQOffset.Maximum = AlgorithmSettings.MAXIMUMQOFFSET;
            numBackEndQBackOff.Value = AlgorithmSettings.DEFAULTQBACKOFF;
            numBackEndQBackOff.Minimum = AlgorithmSettings.MINIMUMQBACKOFF;
            numBackEndQBackOff.Maximum = AlgorithmSettings.MAXIMUMQBACKOFF;
            numBackEndFOffset.Value = AlgorithmSettings.DEFAULTFREQOFFSET;
            numBackEndFOffset.Minimum = AlgorithmSettings.MINIMUMFREQOFFSET;
            numBackEndFOffset.Maximum = AlgorithmSettings.MAXIMUMFREQOFFSET;
            numCoupleUpper.Value = FilterControls.COUPLEMIN;
            numCoupleUpper.Minimum = FilterControls.COUPLEMIN;
            numCoupleUpper.Maximum = FilterControls.COUPLEMAX;
            numCoupleLower.Value = FilterControls.COUPLEMIN;
            numCoupleLower.Minimum = FilterControls.COUPLEMIN;
            numCoupleLower.Maximum = FilterControls.COUPLEMAX;
            numCoupleUFLB.Value = FilterControls.COUPLEMIN;
            numCoupleUFLB.Minimum = FilterControls.COUPLEMIN;
```

```
        numCoupleUFLB.Maximum = FilterControls.COUPLEMAX;
        numCoupleLFUB.Value = FilterControls.COUPLEMIN;
        numCoupleLFUB.Minimum = FilterControls.COUPLEMIN;
        numCoupleLFUB.Maximum = FilterControls.COUPLEMAX;
        chkGroupADThresh1.Checked = false;
        chkGroupADThresh2.Checked = false;
        chkGroupQOffset.Checked = false;
        chkGroupQBackOff.Checked = false;
        chkGroupFOffset.Checked = false;
        chkGroupCoupLat.Checked = false;
        chkGroupCoupCross.Checked = false;
        chkPrintAlgorithmSettings.Checked = false;
        chkPrintFilterSettings.Checked = false;
        chkPrintItterations.Checked = false;

        _isRunning = false;
        btnAlgorithmRun.Text = "Run";
        btnAlgorithmStep.Enabled = false;
        _AlgorithmTimer.Stop();
        _Tick = 0;
        lblAutoTimeRemain.Text = "Algorithm will run in " + int.Parse(numAutoTime.Value.ToString()) + "s";
        lblAutoTimeRemain.Visible = false;
    }

    private void numCenterFreq_ValueChanged(object sender, EventArgs e)
    {

    }

    private void numFreqTol_ValueChanged(object sender, EventArgs e)
    {

    }

    private void numAutoTime_ValueChanged(object sender, EventArgs e)
    {
        if (_isRunning && _isAutomated)
        {
            lblAutoTimeRemain.Text = "Algorithm will run in " + int.Parse(numAutoTime.Value.ToString()) + "s";
            _AlgorithmTimer.Stop();
            _Tick = 0;
            _AlgorithmTimer.Start();
        }
    }

    private void numFrontEndADThresh1_ValueChanged(object sender, EventArgs e)
    {
        if (chkGroupADThresh1.Checked)
            numBackEndADThresh1.Value = numFrontEndADThresh1.Value;
    }

    private void numBackEndADThresh1_ValueChanged(object sender, EventArgs e)
    {
    }

    private void chkGroupADThresh1_CheckedChanged(object sender, EventArgs e)
    {
        if (chkGroupADThresh1.Checked)
        {
            numBackEndADThresh1.Enabled = false;
            numBackEndADThresh1.Value = numFrontEndADThresh1.Value;
        }
        else
            numBackEndADThresh1.Enabled = true;
    }

    private void numFrontEndADThresh2_ValueChanged(object sender, EventArgs e)
    {
        if (chkGroupADThresh2.Checked)
            numBackEndADThresh2.Value = numFrontEndADThresh2.Value;
    }

    private void numBackEndADThresh2_ValueChanged(object sender, EventArgs e)
    {
    }

    private void chkGroupADThresh2_CheckedChanged(object sender, EventArgs e)
    {
        if (chkGroupADThresh2.Checked)
        {
            numBackEndADThresh2.Enabled = false;
            numBackEndADThresh2.Value = numFrontEndADThresh2.Value;
        }
        else
            numBackEndADThresh2.Enabled = true;
    }

    private void numFrontEndQOffset_ValueChanged(object sender, EventArgs e)
    {
        if (chkGroupQOffset.Checked)
            numBackEndQOffset.Value = numFrontEndQOffset.Value;
    }

    private void numBackEndQOffset_ValueChanged(object sender, EventArgs e)
    {
    }

    private void chkGroupQOffset_CheckedChanged(object sender, EventArgs e)
    {
        if (chkGroupQOffset.Checked)
        {
            numBackEndQOffset.Enabled = false;
            numBackEndQOffset.Value = numFrontEndQOffset.Value;
        }
        else
            numBackEndQOffset.Enabled = true;
    }

    private void numFrontEndQBackOff_ValueChanged(object sender, EventArgs e)
    {
        if (chkGroupQBackOff.Checked)
            numBackEndQBackOff.Value = numFrontEndQBackOff.Value;
    }

    private void numBackEndQBackOff_ValueChanged(object sender, EventArgs e)
    {
    }

    private void chkGroupQBackOff_CheckedChanged(object sender, EventArgs e)
    {
        if (chkGroupQBackOff.Checked)
```

```
            {
                numBackEndQBackOff.Enabled = false;
                numBackEndQBackOff.Value = numFrontEndQBackOff.Value;
            }
            else
                numBackEndQBackOff.Enabled = true;
        }

        private void numFrontEndFOffset_ValueChanged(object sender, EventArgs e)
        {
            if (chkGroupFOffset.Checked)
                numBackEndFOffset.Value = numFrontEndFOffset.Value;
        }

        private void numBackEndFOffset_ValueChanged(object sender, EventArgs e)
        {
        }

        private void chkGroupFOffset_CheckedChanged(object sender, EventArgs e)
        {
            if (chkGroupFOffset.Checked)
            {
                numBackEndFOffset.Enabled = false;
                numBackEndFOffset.Value = numFrontEndFOffset.Value;
            }
            else
                numBackEndFOffset.Enabled = true;
        }

        private void numCoupleUpper_ValueChanged(object sender, EventArgs e)
        {
            if (chkGroupCoupLat.Checked)
                numCoupleLower.Value = numCoupleUpper.Value;
        }

        private void numCoupleLower_ValueChanged(object sender, EventArgs e)
        {

        }

        private void numCoupleUFLB_ValueChanged(object sender, EventArgs e)
        {
            if (chkGroupCoupCross.Checked)
                numCoupleLFUB.Value = numCoupleUFLB.Value;
        }

        private void numCoupleLFUB_ValueChanged(object sender, EventArgs e)
        {

        }


        private void chkGroupCoupLat_CheckedChanged(object sender, EventArgs e)
        {
            if (chkGroupCoupLat.Checked)
            {
                numCoupleLower.Enabled = false;
                numCoupleLower.Value = numCoupleUpper.Value;
            }
            else
                numCoupleLower.Enabled = true;
        }

        private void chkGroupCoupCross_CheckedChanged(object sender, EventArgs e)
        {
            if (chkGroupCoupCross.Checked)
            {
                numCoupleLFUB.Enabled = false;
                numCoupleLFUB.Value = numCoupleUFLB.Value;
            }
            else
                numCoupleLFUB.Enabled = true;
        }


        private void btnAlgorithmReset_Click(object sender, EventArgs e)
        {
            InitalizeAlgorithmControls();
        }

        private void btnAlgorithmStep_Click(object sender, EventArgs e)
        {
            UpdateAlgorithmSettings();
            _AlgorithmSettings.RunAlgorithm();
        }

        private void btnAlgorithmRun_Click(object sender, EventArgs e)
        {
            if (!_ComManager.isOpen)
            {
                 MessageBox.Show("Selected COM Port is not open.");
                 return;
            }

            _isRunning = !_isRunning;
            if (!_isRunning)
            {
                _isAutomated = false;
                btnAlgorithmRun.Text = "Run";
                btnAlgorithmStep.Enabled = false;
                _AlgorithmTimer.Stop();
                _Tick = 0;
                lblAutoTimeRemain.Text = "Algorithm will run in " + int.Parse(numAutoTime.Value.ToString()) + "s";
                lblAutoTimeRemain.Visible = false;
            }
            else
            {
                DialogResult SweepType = MessageBox.Show("Would you like to automate when the algorithm is run?", "Automate Algorithm?",
MessageBoxButtons.YesNo);

                btnAlgorithmRun.Text = "Stop";
                _Tick = 0;

                switch (SweepType)
                {
                    case DialogResult.Yes:
                        _isAutomated = true;
                        _AlgorithmTimer.Start();
                        lblAutoTimeRemain.Visible = true;
                        break;
                    case DialogResult.No:
```

```
                    _isAutomated = false;
                    btnAlgorithmStep.Enabled = true;
                    lblAutoTimeRemain.Visible = false;
                    break;
                default:
                    return;
            }
        }
    }

    public void AlgorithmTimer_Tick(object sender, EventArgs eArgs)
    {
        if (sender == _AlgorithmTimer)
        {
            lblAutoTimeRemain.Text = "Algorithm will run in " + (int.Parse(numAutoTime.Value.ToString()) - ++_Tick) + "s";
            if (_Tick == int.Parse(numAutoTime.Value.ToString()))
            {
                _AlgorithmTimer.Stop();
                _Tick = 0;
                lblAutoTimeRemain.Text = "Algorithm will run in " + (int.Parse(numAutoTime.Value.ToString()) - _Tick) + "s";

                UpdateAlgorithmSettings();
                _AlgorithmSettings.RunAlgorithm();
                _AlgorithmTimer.Start();
            }
        }
    }

    public void UpdateAlgorithmSettings()
    {
        _AlgorithmSettings.AlgorithmCtrlReg.CenterFreq = int.Parse((10*double.Parse(numCenterFreq.Value.ToString())).ToString());
        _AlgorithmSettings.AlgorithmCtrlReg.FreqTol = int.Parse((10 * double.Parse(numFreqTol.Value.ToString())).ToString()); ;
        _AlgorithmSettings.AlgorithmCtrlReg.FrontEndADThreshold1 = int.Parse(numFrontEndADThresh1.Value.ToString());
        _AlgorithmSettings.AlgorithmCtrlReg.FrontEndADThreshold2 = int.Parse(numFrontEndADThresh2.Value.ToString());
        _AlgorithmSettings.AlgorithmCtrlReg.FrontEndQOffset = int.Parse(numFrontEndQOffset.Value.ToString());
        _AlgorithmSettings.AlgorithmCtrlReg.FrontEndQBackOff = int.Parse(numFrontEndQBackOff.Value.ToString());
        _AlgorithmSettings.AlgorithmCtrlReg.FrontEndFOffset = int.Parse(numFrontEndFOffset.Value.ToString());
        _AlgorithmSettings.AlgorithmCtrlReg.BackEndADThreshold1 = int.Parse(numBackEndADThresh1.Value.ToString());
        _AlgorithmSettings.AlgorithmCtrlReg.BackEndADThreshold2 = int.Parse(numBackEndADThresh2.Value.ToString());
        _AlgorithmSettings.AlgorithmCtrlReg.BackEndQOffset = int.Parse(numBackEndQOffset.Value.ToString());
        _AlgorithmSettings.AlgorithmCtrlReg.BackEndQBackOff = int.Parse(numBackEndQBackOff.Value.ToString());
        _AlgorithmSettings.AlgorithmCtrlReg.BackEndFOffset = int.Parse(numBackEndFOffset.Value.ToString());
        _AlgorithmSettings.AlgorithmCtrlReg.CouplingUpper = int.Parse(numCoupleUpper.Value.ToString());
        _AlgorithmSettings.AlgorithmCtrlReg.CouplingLower = int.Parse(numCoupleLower.Value.ToString());
        _AlgorithmSettings.AlgorithmCtrlReg.CouplingUFLB = int.Parse(numCoupleUFLB.Value.ToString());
        _AlgorithmSettings.AlgorithmCtrlReg.CouplingLFUB = int.Parse(numCoupleLFUB.Value.ToString());
        _AlgorithmSettings.AlgorithmCtrlReg.PrintAlgorithmSettings = chkPrintAlgorithmSettings.Checked ? 1 : 0;
        _AlgorithmSettings.AlgorithmCtrlReg.PrintFilterSettings = chkPrintFilterSettings.Checked ? 1 : 0;
        _AlgorithmSettings.AlgorithmCtrlReg.PrintItterations = chkPrintItterations.Checked ? 1 : 0;
    }
  }
}
```

FRMALGORITHMSETTINGS.DESIGNER.CS

```
namespace QEFilterV2TestAppV2
{
    partial class frmAlgorithmSettings
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.groupFilterSettings = new System.Windows.Forms.GroupBox();
            this.chkPrintItterations = new System.Windows.Forms.CheckBox();
            this.chkPrintFilterSettings = new System.Windows.Forms.CheckBox();
            this.lblAutoTimeRemain = new System.Windows.Forms.Label();
            this.chkPrintAlgorithmSettings = new System.Windows.Forms.CheckBox();
            this.numAutoTime = new System.Windows.Forms.NumericUpDown();
            this.lblAutoTime = new System.Windows.Forms.Label();
            this.numBackEndFOffset = new System.Windows.Forms.NumericUpDown();
            this.lblBackEndFOffset = new System.Windows.Forms.Label();
            this.numBackEndQBackOff = new System.Windows.Forms.NumericUpDown();
            this.numBackEndQOffset = new System.Windows.Forms.NumericUpDown();
            this.lblBackEndQBackOff = new System.Windows.Forms.Label();
            this.lblBackEndQOffset = new System.Windows.Forms.Label();
            this.lblBackEnd = new System.Windows.Forms.Label();
            this.numBackEndADThresh2 = new System.Windows.Forms.NumericUpDown();
            this.numBackEndADThresh1 = new System.Windows.Forms.NumericUpDown();
            this.lblBackEndADThresh2 = new System.Windows.Forms.Label();
            this.lblBackEndADThresh1 = new System.Windows.Forms.Label();
            this.chkGroupFOffset = new System.Windows.Forms.CheckBox();
            this.chkGroupQBackOff = new System.Windows.Forms.CheckBox();
            this.chkGroupQOffset = new System.Windows.Forms.CheckBox();
            this.numFrontEndFOffset = new System.Windows.Forms.NumericUpDown();
            this.lblFrontEndFOffsset = new System.Windows.Forms.Label();
            this.numFrontEndQBackOff = new System.Windows.Forms.NumericUpDown();
            this.numFrontEndQOffset = new System.Windows.Forms.NumericUpDown();
            this.lblFrontEndQBackOff = new System.Windows.Forms.Label();
            this.lblFrontEndQOffset = new System.Windows.Forms.Label();
            this.btnAlgorithmRun = new System.Windows.Forms.Button();
            this.btnAlgorithmStep = new System.Windows.Forms.Button();
            this.btnAlgorithmReset = new System.Windows.Forms.Button();
```

```
            this.numCoupleLFUB = new System.Windows.Forms.NumericUpDown();
            this.lblCoupleLFUB = new System.Windows.Forms.Label();
            this.numCoupleUFLB = new System.Windows.Forms.NumericUpDown();
            this.lblCoupleUFLB = new System.Windows.Forms.Label();
            this.numCoupleLower = new System.Windows.Forms.NumericUpDown();
            this.lblCoupleLower = new System.Windows.Forms.Label();
            this.numCoupleUpper = new System.Windows.Forms.NumericUpDown();
            this.lblCoupleUpper = new System.Windows.Forms.Label();
            this.lblCoupling = new System.Windows.Forms.Label();
            this.chkGroupADThresh2 = new System.Windows.Forms.CheckBox();
            this.chkGroupADThresh1 = new System.Windows.Forms.CheckBox();
            this.lblFrontEnd = new System.Windows.Forms.Label();
            this.numFrontEndADThresh2 = new System.Windows.Forms.NumericUpDown();
            this.numFrontEndADThresh1 = new System.Windows.Forms.NumericUpDown();
            this.numFreqTol = new System.Windows.Forms.NumericUpDown();
            this.numCenterFreq = new System.Windows.Forms.NumericUpDown();
            this.lblFrontEndADThresh2 = new System.Windows.Forms.Label();
            this.lblFrontEndADThresh1 = new System.Windows.Forms.Label();
            this.lblFreqTol = new System.Windows.Forms.Label();
            this.lblCenterFreq = new System.Windows.Forms.Label();
            this.chkGroupCoupLat = new System.Windows.Forms.CheckBox();
            this.chkGroupCoupCross = new System.Windows.Forms.CheckBox();
            this.groupFilterSettings.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(this.numAutoTime)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackEndFOffset)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackEndQBackOff)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackEndQOffset)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackEndADThresh2)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackEndADThresh1)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontEndFOffset)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontEndQBackOff)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontEndQOffset)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numCoupleLFUB)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numCoupleUFLB)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numCoupleLower)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numCoupleUpper)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontEndADThresh2)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontEndADThresh1)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFreqTol)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numCenterFreq)).BeginInit();
            this.SuspendLayout();
            //
            // groupFilterSettings
            //
            this.groupFilterSettings.Controls.Add(this.chkGroupCoupCross);
            this.groupFilterSettings.Controls.Add(this.chkGroupCoupLat);
            this.groupFilterSettings.Controls.Add(this.chkPrintItterations);
            this.groupFilterSettings.Controls.Add(this.chkPrintFilterSettings);
            this.groupFilterSettings.Controls.Add(this.lblAutoTimeRemain);
            this.groupFilterSettings.Controls.Add(this.chkPrintAlgorithmSettings);
            this.groupFilterSettings.Controls.Add(this.numAutoTime);
            this.groupFilterSettings.Controls.Add(this.lblAutoTime);
            this.groupFilterSettings.Controls.Add(this.numBackEndFOffset);
            this.groupFilterSettings.Controls.Add(this.lblBackEndFOffset);
            this.groupFilterSettings.Controls.Add(this.numBackEndQBackOff);
            this.groupFilterSettings.Controls.Add(this.numBackEndQOffset);
            this.groupFilterSettings.Controls.Add(this.lblBackEndQBackOff);
            this.groupFilterSettings.Controls.Add(this.lblBackEndQOffset);
            this.groupFilterSettings.Controls.Add(this.lblBackEnd);
            this.groupFilterSettings.Controls.Add(this.numBackEndADThresh2);
            this.groupFilterSettings.Controls.Add(this.numBackEndADThresh1);
            this.groupFilterSettings.Controls.Add(this.lblBackEndADThresh2);
            this.groupFilterSettings.Controls.Add(this.lblBackEndADThresh1);
            this.groupFilterSettings.Controls.Add(this.chkGroupFOffset);
            this.groupFilterSettings.Controls.Add(this.chkGroupQBackOff);
            this.groupFilterSettings.Controls.Add(this.chkGroupQOffset);
            this.groupFilterSettings.Controls.Add(this.numFrontEndFOffset);
            this.groupFilterSettings.Controls.Add(this.lblFrontEndFOffsset);
            this.groupFilterSettings.Controls.Add(this.numFrontEndQBackOff);
            this.groupFilterSettings.Controls.Add(this.numFrontEndQOffset);
            this.groupFilterSettings.Controls.Add(this.lblFrontEndQBackOff);
            this.groupFilterSettings.Controls.Add(this.lblFrontEndQOffset);
            this.groupFilterSettings.Controls.Add(this.btnAlgorithmRun);
            this.groupFilterSettings.Controls.Add(this.btnAlgorithmStep);
            this.groupFilterSettings.Controls.Add(this.btnAlgorithmReset);
            this.groupFilterSettings.Controls.Add(this.numCoupleLFUB);
            this.groupFilterSettings.Controls.Add(this.lblCoupleLFUB);
            this.groupFilterSettings.Controls.Add(this.numCoupleUFLB);
            this.groupFilterSettings.Controls.Add(this.lblCoupleUFLB);
            this.groupFilterSettings.Controls.Add(this.numCoupleLower);
            this.groupFilterSettings.Controls.Add(this.lblCoupleLower);
            this.groupFilterSettings.Controls.Add(this.numCoupleUpper);
            this.groupFilterSettings.Controls.Add(this.lblCoupleUpper);
            this.groupFilterSettings.Controls.Add(this.lblCoupling);
            this.groupFilterSettings.Controls.Add(this.chkGroupADThresh2);
            this.groupFilterSettings.Controls.Add(this.chkGroupADThresh1);
            this.groupFilterSettings.Controls.Add(this.lblFrontEnd);
            this.groupFilterSettings.Controls.Add(this.numFrontEndADThresh2);
            this.groupFilterSettings.Controls.Add(this.numFrontEndADThresh1);
            this.groupFilterSettings.Controls.Add(this.numFreqTol);
            this.groupFilterSettings.Controls.Add(this.numCenterFreq);
            this.groupFilterSettings.Controls.Add(this.lblFrontEndADThresh2);
            this.groupFilterSettings.Controls.Add(this.lblFrontEndADThresh1);
            this.groupFilterSettings.Controls.Add(this.lblFreqTol);
            this.groupFilterSettings.Controls.Add(this.lblCenterFreq);
            this.groupFilterSettings.Font = new System.Drawing.Font("Cambria", 12F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.groupFilterSettings.Location = new System.Drawing.Point(12, 12);
            this.groupFilterSettings.Name = "groupFilterSettings";
            this.groupFilterSettings.Size = new System.Drawing.Size(552, 442);
            this.groupFilterSettings.TabIndex = 5;
            this.groupFilterSettings.TabStop = false;
            this.groupFilterSettings.Text = "Algorithm Settings";
            //
            // chkPrintItterations
            //
            this.chkPrintItterations.AutoSize = true;
            this.chkPrintItterations.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.chkPrintItterations.Location = new System.Drawing.Point(415, 381);
            this.chkPrintItterations.Name = "chkPrintItterations";
            this.chkPrintItterations.Size = new System.Drawing.Size(88, 17);
            this.chkPrintItterations.TabIndex = 78;
            this.chkPrintItterations.Text = "Print Iteration";
            this.chkPrintItterations.UseVisualStyleBackColor = true;
            //
```

```
            // chkPrintFilterSettings
            //
            this.chkPrintFilterSettings.AutoSize = true;
            this.chkPrintFilterSettings.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.chkPrintFilterSettings.Location = new System.Drawing.Point(218, 381);
            this.chkPrintFilterSettings.Name = "chkPrintFilterSettings";
            this.chkPrintFilterSettings.Size = new System.Drawing.Size(113, 17);
            this.chkPrintFilterSettings.TabIndex = 19;
            this.chkPrintFilterSettings.Text = "Print Filter Settings";
            this.chkPrintFilterSettings.UseVisualStyleBackColor = true;
            //
            // lblAutoTimeRemain
            //
            this.lblAutoTimeRemain.AutoSize = true;
            this.lblAutoTimeRemain.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.lblAutoTimeRemain.ForeColor = System.Drawing.Color.Red;
            this.lblAutoTimeRemain.Location = new System.Drawing.Point(301, 68);
            this.lblAutoTimeRemain.Name = "lblAutoTimeRemain";
            this.lblAutoTimeRemain.Size = new System.Drawing.Size(203, 20);
            this.lblAutoTimeRemain.TabIndex = 77;
            this.lblAutoTimeRemain.Text = "Algorithm will run in 10 s";
            //
            // chkPrintAlgorithmSettings
            //
            this.chkPrintAlgorithmSettings.AutoSize = true;
            this.chkPrintAlgorithmSettings.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.chkPrintAlgorithmSettings.Location = new System.Drawing.Point(18, 381);
            this.chkPrintAlgorithmSettings.Name = "chkPrintAlgorithmSettings";
            this.chkPrintAlgorithmSettings.Size = new System.Drawing.Size(132, 17);
            this.chkPrintAlgorithmSettings.TabIndex = 18;
            this.chkPrintAlgorithmSettings.Text = "Print Algorithn Settings";
            this.chkPrintAlgorithmSettings.UseVisualStyleBackColor = true;
            //
            // numAutoTime
            //
            this.numAutoTime.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.numAutoTime.Location = new System.Drawing.Point(306, 37);
            this.numAutoTime.Maximum = new decimal(new int[] {
            120,
            0,
            0,
            0});
            this.numAutoTime.Minimum = new decimal(new int[] {
            1,
            0,
            0,
            0});
            this.numAutoTime.Name = "numAutoTime";
            this.numAutoTime.Size = new System.Drawing.Size(61, 20);
            this.numAutoTime.TabIndex = 76;
            this.numAutoTime.Value = new decimal(new int[] {
            1,
            0,
            0,
            0});
            this.numAutoTime.ValueChanged += new System.EventHandler(this.numAutoTime_ValueChanged);
            //
            // lblAutoTime
            //
            this.lblAutoTime.AutoSize = true;
            this.lblAutoTime.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.lblAutoTime.Location = new System.Drawing.Point(373, 39);
            this.lblAutoTime.Name = "lblAutoTime";
            this.lblAutoTime.Size = new System.Drawing.Size(146, 13);
            this.lblAutoTime.TabIndex = 75;
            this.lblAutoTime.Text = "Automated Tuning Interval (s)";
            //
            // numBackEndFOffset
            //
            this.numBackEndFOffset.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.numBackEndFOffset.Location = new System.Drawing.Point(306, 253);
            this.numBackEndFOffset.Maximum = new decimal(new int[] {
            31,
            0,
            0,
            0});
            this.numBackEndFOffset.Name = "numBackEndFOffset";
            this.numBackEndFOffset.Size = new System.Drawing.Size(61, 20);
            this.numBackEndFOffset.TabIndex = 74;
            this.numBackEndFOffset.ValueChanged += new System.EventHandler(this.numBackEndFOffset_ValueChanged);
            //
            // lblBackEndFOffset
            //
            this.lblBackEndFOffset.AutoSize = true;
            this.lblBackEndFOffset.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.lblBackEndFOffset.Location = new System.Drawing.Point(373, 255);
            this.lblBackEndFOffset.Name = "lblBackEndFOffset";
            this.lblBackEndFOffset.Size = new System.Drawing.Size(44, 13);
            this.lblBackEndFOffset.TabIndex = 73;
            this.lblBackEndFOffset.Text = "F-Offset";
            //
            // numBackEndQBackOff
            //
            this.numBackEndQBackOff.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.numBackEndQBackOff.Location = new System.Drawing.Point(306, 224);
            this.numBackEndQBackOff.Maximum = new decimal(new int[] {
            15,
            0,
            0,
            0});
            this.numBackEndQBackOff.Name = "numBackEndQBackOff";
            this.numBackEndQBackOff.Size = new System.Drawing.Size(61, 20);
            this.numBackEndQBackOff.TabIndex = 72;
            this.numBackEndQBackOff.ValueChanged += new System.EventHandler(this.numBackEndQBackOff_ValueChanged);
            //
            // numBackEndQOffset
            //
            this.numBackEndQOffset.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.numBackEndQOffset.Location = new System.Drawing.Point(306, 195);
            this.numBackEndQOffset.Maximum = new decimal(new int[] {
            63,
            0,
            0,
            0});
            this.numBackEndQOffset.Name = "numBackEndQOffset";
            this.numBackEndQOffset.Size = new System.Drawing.Size(61, 20);
            this.numBackEndQOffset.TabIndex = 71;
            this.numBackEndQOffset.ValueChanged += new System.EventHandler(this.numBackEndQOffset_ValueChanged);
            //
```

```
            // lblBackEndQBackOff
            //
            this.lblBackEndQBackOff.AutoSize = true;
            this.lblBackEndQBackOff.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.lblBackEndQBackOff.Location = new System.Drawing.Point(373, 226);
            this.lblBackEndQBackOff.Name = "lblBackEndQBackOff";
            this.lblBackEndQBackOff.Size = new System.Drawing.Size(57, 13);
            this.lblBackEndQBackOff.TabIndex = 69;
            this.lblBackEndQBackOff.Text = "Q-BackOff";
            //
            // lblBackEndQOffset
            //
            this.lblBackEndQOffset.AutoSize = true;
            this.lblBackEndQOffset.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.lblBackEndQOffset.Location = new System.Drawing.Point(373, 197);
            this.lblBackEndQOffset.Name = "lblBackEndQOffset";
            this.lblBackEndQOffset.Size = new System.Drawing.Size(46, 13);
            this.lblBackEndQOffset.TabIndex = 70;
            this.lblBackEndQOffset.Text = "Q-Offset";
            //
            // lblBackEnd
            //
            this.lblBackEnd.AutoSize = true;
            this.lblBackEnd.Font = new System.Drawing.Font("Cambria", 9.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.lblBackEnd.Location = new System.Drawing.Point(303, 114);
            this.lblBackEnd.Name = "lblBackEnd";
            this.lblBackEnd.Size = new System.Drawing.Size(64, 15);
            this.lblBackEnd.TabIndex = 64;
            this.lblBackEnd.Text = "Back-End";
            //
            // numBackEndADThresh2
            //
            this.numBackEndADThresh2.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.numBackEndADThresh2.Location = new System.Drawing.Point(306, 166);
            this.numBackEndADThresh2.Maximum = new decimal(new int[] {
            1023,
            0,
            0,
            0});
            this.numBackEndADThresh2.Name = "numBackEndADThresh2";
            this.numBackEndADThresh2.Size = new System.Drawing.Size(61, 20);
            this.numBackEndADThresh2.TabIndex = 68;
            this.numBackEndADThresh2.ValueChanged += new System.EventHandler(this.numBackEndADThresh2_ValueChanged);
            //
            // numBackEndADThresh1
            //
            this.numBackEndADThresh1.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.numBackEndADThresh1.Location = new System.Drawing.Point(306, 137);
            this.numBackEndADThresh1.Maximum = new decimal(new int[] {
            1023,
            0,
            0,
            0});
            this.numBackEndADThresh1.Name = "numBackEndADThresh1";
            this.numBackEndADThresh1.Size = new System.Drawing.Size(61, 20);
            this.numBackEndADThresh1.TabIndex = 67;
            this.numBackEndADThresh1.ValueChanged += new System.EventHandler(this.numBackEndADThresh1_ValueChanged);
            //
            // lblBackEndADThresh2
            //
            this.lblBackEndADThresh2.AutoSize = true;
            this.lblBackEndADThresh2.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.lblBackEndADThresh2.Location = new System.Drawing.Point(373, 168);
            this.lblBackEndADThresh2.Name = "lblBackEndADThresh2";
            this.lblBackEndADThresh2.Size = new System.Drawing.Size(81, 13);
            this.lblBackEndADThresh2.TabIndex = 65;
            this.lblBackEndADThresh2.Text = "AD Threshold 2";
            //
            // lblBackEndADThresh1
            //
            this.lblBackEndADThresh1.AutoSize = true;
            this.lblBackEndADThresh1.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.lblBackEndADThresh1.Location = new System.Drawing.Point(373, 139);
            this.lblBackEndADThresh1.Name = "lblBackEndADThresh1";
            this.lblBackEndADThresh1.Size = new System.Drawing.Size(81, 13);
            this.lblBackEndADThresh1.TabIndex = 66;
            this.lblBackEndADThresh1.Text = "AD Threshold 1";
            //
            // chkGroupFOffset
            //
            this.chkGroupFOffset.AutoSize = true;
            this.chkGroupFOffset.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.chkGroupFOffset.Location = new System.Drawing.Point(188, 254);
            this.chkGroupFOffset.Name = "chkGroupFOffset";
            this.chkGroupFOffset.Size = new System.Drawing.Size(96, 17);
            this.chkGroupFOffset.TabIndex = 63;
            this.chkGroupFOffset.Text = "Group Controls";
            this.chkGroupFOffset.UseVisualStyleBackColor = true;
            this.chkGroupFOffset.CheckedChanged += new System.EventHandler(this.chkGroupFOffset_CheckedChanged);
            //
            // chkGroupQBackOff
            //
            this.chkGroupQBackOff.AutoSize = true;
            this.chkGroupQBackOff.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.chkGroupQBackOff.Location = new System.Drawing.Point(188, 225);
            this.chkGroupQBackOff.Name = "chkGroupQBackOff";
            this.chkGroupQBackOff.Size = new System.Drawing.Size(96, 17);
            this.chkGroupQBackOff.TabIndex = 62;
            this.chkGroupQBackOff.Text = "Group Controls";
            this.chkGroupQBackOff.UseVisualStyleBackColor = true;
            this.chkGroupQBackOff.CheckedChanged += new System.EventHandler(this.chkGroupQBackOff_CheckedChanged);
            //
            // chkGroupQOffset
            //
            this.chkGroupQOffset.AutoSize = true;
            this.chkGroupQOffset.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.chkGroupQOffset.Location = new System.Drawing.Point(188, 196);
            this.chkGroupQOffset.Name = "chkGroupQOffset";
            this.chkGroupQOffset.Size = new System.Drawing.Size(96, 17);
            this.chkGroupQOffset.TabIndex = 61;
            this.chkGroupQOffset.Text = "Group Controls";
            this.chkGroupQOffset.UseVisualStyleBackColor = true;
            this.chkGroupQOffset.CheckedChanged += new System.EventHandler(this.chkGroupQOffset_CheckedChanged);
            //
            // numFrontEndFOffset
            //
            this.numFrontEndFOffset.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.numFrontEndFOffset.Location = new System.Drawing.Point(20, 253);
```

```
this.numFrontEndFOffset.Maximum = new decimal(new int[] {
31,
0,
0,
0});
this.numFrontEndFOffset.Name = "numFrontEndFOffset";
this.numFrontEndFOffset.Size = new System.Drawing.Size(61, 20);
this.numFrontEndFOffset.TabIndex = 60;
this.numFrontEndFOffset.ValueChanged += new System.EventHandler(this.numFrontEndFOffset_ValueChanged);
//
// lblFrontEndFOffsset
//
this.lblFrontEndFOffsset.AutoSize = true;
this.lblFrontEndFOffsset.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.lblFrontEndFOffsset.Location = new System.Drawing.Point(87, 255);
this.lblFrontEndFOffsset.Name = "lblFrontEndFOffsset";
this.lblFrontEndFOffsset.Size = new System.Drawing.Size(44, 13);
this.lblFrontEndFOffsset.TabIndex = 59;
this.lblFrontEndFOffsset.Text = "F-Offset";
//
// numFrontEndQBackOff
//
this.numFrontEndQBackOff.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numFrontEndQBackOff.Location = new System.Drawing.Point(20, 224);
this.numFrontEndQBackOff.Maximum = new decimal(new int[] {
15,
0,
0,
0});
this.numFrontEndQBackOff.Name = "numFrontEndQBackOff";
this.numFrontEndQBackOff.Size = new System.Drawing.Size(61, 20);
this.numFrontEndQBackOff.TabIndex = 58;
this.numFrontEndQBackOff.ValueChanged += new System.EventHandler(this.numFrontEndQBackOff_ValueChanged);
//
// numFrontEndQOffset
//
this.numFrontEndQOffset.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numFrontEndQOffset.Location = new System.Drawing.Point(20, 195);
this.numFrontEndQOffset.Maximum = new decimal(new int[] {
63,
0,
0,
0});
this.numFrontEndQOffset.Name = "numFrontEndQOffset";
this.numFrontEndQOffset.Size = new System.Drawing.Size(61, 20);
this.numFrontEndQOffset.TabIndex = 57;
this.numFrontEndQOffset.ValueChanged += new System.EventHandler(this.numFrontEndQOffset_ValueChanged);
//
// lblFrontEndQBackOff
//
this.lblFrontEndQBackOff.AutoSize = true;
this.lblFrontEndQBackOff.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.lblFrontEndQBackOff.Location = new System.Drawing.Point(87, 226);
this.lblFrontEndQBackOff.Name = "lblFrontEndQBackOff";
this.lblFrontEndQBackOff.Size = new System.Drawing.Size(57, 13);
this.lblFrontEndQBackOff.TabIndex = 55;
this.lblFrontEndQBackOff.Text = "Q-BackOff";
//
// lblFrontEndQOffset
//
this.lblFrontEndQOffset.AutoSize = true;
this.lblFrontEndQOffset.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.lblFrontEndQOffset.Location = new System.Drawing.Point(87, 197);
this.lblFrontEndQOffset.Name = "lblFrontEndQOffset";
this.lblFrontEndQOffset.Size = new System.Drawing.Size(46, 13);
this.lblFrontEndQOffset.TabIndex = 56;
this.lblFrontEndQOffset.Text = "Q-Offset";
//
// btnAlgorithmRun
//
this.btnAlgorithmRun.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.btnAlgorithmRun.Location = new System.Drawing.Point(415, 404);
this.btnAlgorithmRun.Name = "btnAlgorithmRun";
this.btnAlgorithmRun.Size = new System.Drawing.Size(121, 23);
this.btnAlgorithmRun.TabIndex = 53;
this.btnAlgorithmRun.Text = "Run";
this.btnAlgorithmRun.UseVisualStyleBackColor = true;
this.btnAlgorithmRun.Click += new System.EventHandler(this.btnAlgorithmRun_Click);
//
// btnAlgorithmStep
//
this.btnAlgorithmStep.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.btnAlgorithmStep.Location = new System.Drawing.Point(218, 404);
this.btnAlgorithmStep.Name = "btnAlgorithmStep";
this.btnAlgorithmStep.Size = new System.Drawing.Size(121, 23);
this.btnAlgorithmStep.TabIndex = 52;
this.btnAlgorithmStep.Text = "Step";
this.btnAlgorithmStep.UseVisualStyleBackColor = true;
this.btnAlgorithmStep.Click += new System.EventHandler(this.btnAlgorithmStep_Click);
//
// btnAlgorithmReset
//
this.btnAlgorithmReset.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.btnAlgorithmReset.Location = new System.Drawing.Point(21, 404);
this.btnAlgorithmReset.Name = "btnAlgorithmReset";
this.btnAlgorithmReset.Size = new System.Drawing.Size(121, 23);
this.btnAlgorithmReset.TabIndex = 51;
this.btnAlgorithmReset.Text = "Restore Defaults";
this.btnAlgorithmReset.UseVisualStyleBackColor = true;
this.btnAlgorithmReset.Click += new System.EventHandler(this.btnAlgorithmReset_Click);
//
// numCoupleLFUB
//
this.numCoupleLFUB.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numCoupleLFUB.Location = new System.Drawing.Point(306, 346);
this.numCoupleLFUB.Maximum = new decimal(new int[] {
31,
0,
0,
0});
this.numCoupleLFUB.Name = "numCoupleLFUB";
this.numCoupleLFUB.Size = new System.Drawing.Size(61, 20);
this.numCoupleLFUB.TabIndex = 49;
this.numCoupleLFUB.ValueChanged += new System.EventHandler(this.numCoupleLFUB_ValueChanged);
//
// lblCoupleLFUB
//
this.lblCoupleLFUB.AutoSize = true;
this.lblCoupleLFUB.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
```

```
            this.lblCoupleLFUB.Location = new System.Drawing.Point(375, 348);
            this.lblCoupleLFUB.Name = "lblCoupleLFUB";
            this.lblCoupleLFUB.Size = new System.Drawing.Size(123, 13);
            this.lblCoupleLFUB.TabIndex = 0;
            this.lblCoupleLFUB.Text = "Capacitive LFUB Tuning";
            //
            // numCoupleUFLB
            //
            this.numCoupleUFLB.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.numCoupleUFLB.Location = new System.Drawing.Point(306, 317);
            this.numCoupleUFLB.Maximum = new decimal(new int[] {
            31,
            0,
            0,
            0});
            this.numCoupleUFLB.Name = "numCoupleUFLB";
            this.numCoupleUFLB.Size = new System.Drawing.Size(61, 20);
            this.numCoupleUFLB.TabIndex = 47;
            this.numCoupleUFLB.ValueChanged += new System.EventHandler(this.numCoupleUFLB_ValueChanged);
            //
            // lblCoupleUFLB
            //
            this.lblCoupleUFLB.AutoSize = true;
            this.lblCoupleUFLB.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.lblCoupleUFLB.Location = new System.Drawing.Point(375, 319);
            this.lblCoupleUFLB.Name = "lblCoupleUFLB";
            this.lblCoupleUFLB.Size = new System.Drawing.Size(123, 13);
            this.lblCoupleUFLB.TabIndex = 0;
            this.lblCoupleUFLB.Text = "Capacitive UFLB Tuning";
            //
            // numCoupleLower
            //
            this.numCoupleLower.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.numCoupleLower.Location = new System.Drawing.Point(20, 346);
            this.numCoupleLower.Maximum = new decimal(new int[] {
            31,
            0,
            0,
            0});
            this.numCoupleLower.Name = "numCoupleLower";
            this.numCoupleLower.Size = new System.Drawing.Size(61, 20);
            this.numCoupleLower.TabIndex = 45;
            this.numCoupleLower.ValueChanged += new System.EventHandler(this.numCoupleLower_ValueChanged);
            //
            // lblCoupleLower
            //
            this.lblCoupleLower.AutoSize = true;
            this.lblCoupleLower.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.lblCoupleLower.Location = new System.Drawing.Point(87, 349);
            this.lblCoupleLower.Name = "lblCoupleLower";
            this.lblCoupleLower.Size = new System.Drawing.Size(125, 13);
            this.lblCoupleLower.TabIndex = 0;
            this.lblCoupleLower.Text = "Capacitive Lower Tuning";
            //
            // numCoupleUpper
            //
            this.numCoupleUpper.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.numCoupleUpper.Location = new System.Drawing.Point(20, 317);
            this.numCoupleUpper.Maximum = new decimal(new int[] {
            31,
            0,
            0,
            0});
            this.numCoupleUpper.Name = "numCoupleUpper";
            this.numCoupleUpper.Size = new System.Drawing.Size(61, 20);
            this.numCoupleUpper.TabIndex = 43;
            this.numCoupleUpper.ValueChanged += new System.EventHandler(this.numCoupleUpper_ValueChanged);
            //
            // lblCoupleUpper
            //
            this.lblCoupleUpper.AutoSize = true;
            this.lblCoupleUpper.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.lblCoupleUpper.Location = new System.Drawing.Point(87, 319);
            this.lblCoupleUpper.Name = "lblCoupleUpper";
            this.lblCoupleUpper.Size = new System.Drawing.Size(125, 13);
            this.lblCoupleUpper.TabIndex = 0;
            this.lblCoupleUpper.Text = "Capacitive Upper Tuning";
            //
            // lblCoupling
            //
            this.lblCoupling.AutoSize = true;
            this.lblCoupling.Font = new System.Drawing.Font("Cambria", 9.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.lblCoupling.Location = new System.Drawing.Point(17, 293);
            this.lblCoupling.Name = "lblCoupling";
            this.lblCoupling.Size = new System.Drawing.Size(60, 15);
            this.lblCoupling.TabIndex = 0;
            this.lblCoupling.Text = "Coupling";
            //
            // chkGroupADThresh2
            //
            this.chkGroupADThresh2.AutoSize = true;
            this.chkGroupADThresh2.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.chkGroupADThresh2.Location = new System.Drawing.Point(188, 167);
            this.chkGroupADThresh2.Name = "chkGroupADThresh2";
            this.chkGroupADThresh2.Size = new System.Drawing.Size(96, 17);
            this.chkGroupADThresh2.TabIndex = 17;
            this.chkGroupADThresh2.Text = "Group Controls";
            this.chkGroupADThresh2.UseVisualStyleBackColor = true;
            this.chkGroupADThresh2.CheckedChanged += new System.EventHandler(this.chkGroupADThresh2_CheckedChanged);
            //
            // chkGroupADThresh1
            //
            this.chkGroupADThresh1.AutoSize = true;
            this.chkGroupADThresh1.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.chkGroupADThresh1.Location = new System.Drawing.Point(188, 138);
            this.chkGroupADThresh1.Name = "chkGroupADThresh1";
            this.chkGroupADThresh1.Size = new System.Drawing.Size(96, 17);
            this.chkGroupADThresh1.TabIndex = 16;
            this.chkGroupADThresh1.Text = "Group Controls";
            this.chkGroupADThresh1.UseVisualStyleBackColor = true;
            this.chkGroupADThresh1.CheckedChanged += new System.EventHandler(this.chkGroupADThresh1_CheckedChanged);
            //
            // lblFrontEnd
            //
            this.lblFrontEnd.AutoSize = true;
            this.lblFrontEnd.Font = new System.Drawing.Font("Cambria", 9.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.lblFrontEnd.Location = new System.Drawing.Point(17, 114);
```

```
this.lblFrontEnd.Name = "lblFrontEnd";
this.lblFrontEnd.Size = new System.Drawing.Size(68, 15);
this.lblFrontEnd.TabIndex = 0;
this.lblFrontEnd.Text = "Front-End";
//
// numFrontEndADThresh2
//
this.numFrontEndADThresh2.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numFrontEndADThresh2.Location = new System.Drawing.Point(20, 166);
this.numFrontEndADThresh2.Maximum = new decimal(new int[] {
1023,
0,
0,
0});
this.numFrontEndADThresh2.Name = "numFrontEndADThresh2";
this.numFrontEndADThresh2.Size = new System.Drawing.Size(61, 20);
this.numFrontEndADThresh2.TabIndex = 28;
this.numFrontEndADThresh2.ValueChanged += new System.EventHandler(this.numFrontEndADThresh2_ValueChanged);
//
// numFrontEndADThresh1
//
this.numFrontEndADThresh1.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numFrontEndADThresh1.Location = new System.Drawing.Point(20, 137);
this.numFrontEndADThresh1.Maximum = new decimal(new int[] {
1023,
0,
0,
0});
this.numFrontEndADThresh1.Name = "numFrontEndADThresh1";
this.numFrontEndADThresh1.Size = new System.Drawing.Size(61, 20);
this.numFrontEndADThresh1.TabIndex = 26;
this.numFrontEndADThresh1.ValueChanged += new System.EventHandler(this.numFrontEndADThresh1_ValueChanged);
//
// numFreqTol
//
this.numFreqTol.DecimalPlaces = 1;
this.numFreqTol.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numFreqTol.Increment = new decimal(new int[] {
1,
0,
0,
65536});
this.numFreqTol.Location = new System.Drawing.Point(20, 66);
this.numFreqTol.Name = "numFreqTol";
this.numFreqTol.Size = new System.Drawing.Size(61, 20);
this.numFreqTol.TabIndex = 24;
this.numFreqTol.ValueChanged += new System.EventHandler(this.numFreqTol_ValueChanged);
//
// numCenterFreq
//
this.numCenterFreq.DecimalPlaces = 1;
this.numCenterFreq.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.numCenterFreq.Increment = new decimal(new int[] {
1,
0,
0,
65536});
this.numCenterFreq.Location = new System.Drawing.Point(20, 37);
this.numCenterFreq.Maximum = new decimal(new int[] {
600,
0,
0,
0});
this.numCenterFreq.Minimum = new decimal(new int[] {
400,
0,
0,
0});
this.numCenterFreq.Name = "numCenterFreq";
this.numCenterFreq.Size = new System.Drawing.Size(61, 20);
this.numCenterFreq.TabIndex = 22;
this.numCenterFreq.Value = new decimal(new int[] {
400,
0,
0,
0});
this.numCenterFreq.ValueChanged += new System.EventHandler(this.numCenterFreq_ValueChanged);
//
// lblFrontEndADThresh2
//
this.lblFrontEndADThresh2.AutoSize = true;
this.lblFrontEndADThresh2.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.lblFrontEndADThresh2.Location = new System.Drawing.Point(87, 168);
this.lblFrontEndADThresh2.Name = "lblFrontEndADThresh2";
this.lblFrontEndADThresh2.Size = new System.Drawing.Size(81, 13);
this.lblFrontEndADThresh2.TabIndex = 0;
this.lblFrontEndADThresh2.Text = "AD Threshold 2";
//
// lblFrontEndADThresh1
//
this.lblFrontEndADThresh1.AutoSize = true;
this.lblFrontEndADThresh1.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.lblFrontEndADThresh1.Location = new System.Drawing.Point(87, 139);
this.lblFrontEndADThresh1.Name = "lblFrontEndADThresh1";
this.lblFrontEndADThresh1.Size = new System.Drawing.Size(81, 13);
this.lblFrontEndADThresh1.TabIndex = 0;
this.lblFrontEndADThresh1.Text = "AD Threshold 1";
//
// lblFreqTol
//
this.lblFreqTol.AutoSize = true;
this.lblFreqTol.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.lblFreqTol.Location = new System.Drawing.Point(87, 68);
this.lblFreqTol.Name = "lblFreqTol";
this.lblFreqTol.Size = new System.Drawing.Size(139, 13);
this.lblFreqTol.TabIndex = 0;
this.lblFreqTol.Text = "Frequency Tolerance (MHz)";
//
// lblCenterFreq
//
this.lblCenterFreq.AutoSize = true;
this.lblCenterFreq.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
this.lblCenterFreq.Location = new System.Drawing.Point(87, 39);
this.lblCenterFreq.Name = "lblCenterFreq";
this.lblCenterFreq.Size = new System.Drawing.Size(122, 13);
this.lblCenterFreq.TabIndex = 0;
this.lblCenterFreq.Text = "Center Frequency (MHz)";
//
// chkGroupCoupLat
//
```

```
            this.chkGroupCoupLat.AutoSize = true;
            this.chkGroupCoupLat.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.chkGroupCoupLat.Location = new System.Drawing.Point(90, 293);
            this.chkGroupCoupLat.Name = "chkGroupCoupLat";
            this.chkGroupCoupLat.Size = new System.Drawing.Size(96, 17);
            this.chkGroupCoupLat.TabIndex = 79;
            this.chkGroupCoupLat.Text = "Group Controls";
            this.chkGroupCoupLat.UseVisualStyleBackColor = true;
            this.chkGroupCoupLat.CheckedChanged += new System.EventHandler(this.chkGroupCoupLat_CheckedChanged);
            //
            // chkGroupCoupCross
            //
            this.chkGroupCoupCross.AutoSize = true;
            this.chkGroupCoupCross.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F);
            this.chkGroupCoupCross.Location = new System.Drawing.Point(378, 293);
            this.chkGroupCoupCross.Name = "chkGroupCoupCross";
            this.chkGroupCoupCross.Size = new System.Drawing.Size(96, 17);
            this.chkGroupCoupCross.TabIndex = 80;
            this.chkGroupCoupCross.Text = "Group Controls";
            this.chkGroupCoupCross.UseVisualStyleBackColor = true;
            this.chkGroupCoupCross.CheckedChanged += new System.EventHandler(this.chkGroupCoupCross_CheckedChanged);
            //
            // frmAlgorithmSettings
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(575, 463);
            this.Controls.Add(this.groupFilterSettings);
            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.Fixed3D;
            this.MaximizeBox = false;
            this.Name = "frmAlgorithmSettings";
            this.Text = "Algorithm Settings";
            this.groupFilterSettings.ResumeLayout(false);
            this.groupFilterSettings.PerformLayout();
            ((System.ComponentModel.ISupportInitialize)(this.numAutoTime)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackEndFOffset)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackEndQBackOff)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackEndQOffset)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackEndADThresh2)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numBackEndADThresh1)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontEndFOffset)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontEndQBackOff)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontEndQOffset)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numCoupleLFUB)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numCoupleUFLB)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numCoupleLower)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numCoupleUpper)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontEndADThresh2)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFrontEndADThresh1)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numFreqTol)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numCenterFreq)).EndInit();
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.GroupBox groupFilterSettings;
        private System.Windows.Forms.Button btnAlgorithmRun;
        private System.Windows.Forms.Button btnAlgorithmStep;
        private System.Windows.Forms.Button btnAlgorithmReset;
        private System.Windows.Forms.NumericUpDown numCoupleLFUB;
        private System.Windows.Forms.Label lblCoupleLFUB;
        private System.Windows.Forms.NumericUpDown numCoupleUFLB;
        private System.Windows.Forms.Label lblCoupleUFLB;
        private System.Windows.Forms.NumericUpDown numCoupleLower;
        private System.Windows.Forms.Label lblCoupleLower;
        private System.Windows.Forms.NumericUpDown numCoupleUpper;
        private System.Windows.Forms.Label lblCoupleUpper;
        private System.Windows.Forms.Label lblCoupling;
        private System.Windows.Forms.CheckBox chkGroupADThresh2;
        private System.Windows.Forms.CheckBox chkGroupADThresh1;
        private System.Windows.Forms.Label lblFrontEnd;
        private System.Windows.Forms.NumericUpDown numFrontEndADThresh2;
        private System.Windows.Forms.NumericUpDown numFrontEndADThresh1;
        private System.Windows.Forms.NumericUpDown numFreqTol;
        private System.Windows.Forms.NumericUpDown numCenterFreq;
        private System.Windows.Forms.Label lblFrontEndADThresh2;
        private System.Windows.Forms.Label lblFrontEndADThresh1;
        private System.Windows.Forms.Label lblFreqTol;
        private System.Windows.Forms.Label lblCenterFreq;
        private System.Windows.Forms.NumericUpDown numFrontEndFOffset;
        private System.Windows.Forms.Label lblFrontEndFOffsset;
        private System.Windows.Forms.NumericUpDown numFrontEndQBackOff;
        private System.Windows.Forms.NumericUpDown numFrontEndQOffset;
        private System.Windows.Forms.Label lblFrontEndQBackOff;
        private System.Windows.Forms.Label lblFrontEndQOffset;
        private System.Windows.Forms.NumericUpDown numBackEndFOffset;
        private System.Windows.Forms.Label lblBackEndFOffset;
        private System.Windows.Forms.NumericUpDown numBackEndQBackOff;
        private System.Windows.Forms.NumericUpDown numBackEndQOffset;
        private System.Windows.Forms.Label lblBackEndQBackOff;
        private System.Windows.Forms.Label lblBackEndQOffset;
        private System.Windows.Forms.Label lblBackEnd;
        private System.Windows.Forms.NumericUpDown numBackEndADThresh2;
        private System.Windows.Forms.NumericUpDown numBackEndADThresh1;
        private System.Windows.Forms.Label lblBackEndADThresh2;
        private System.Windows.Forms.Label lblBackEndADThresh1;
        private System.Windows.Forms.CheckBox chkGroupFOffset;
        private System.Windows.Forms.CheckBox chkGroupQBackOff;
        private System.Windows.Forms.CheckBox chkGroupQOffset;
        private System.Windows.Forms.NumericUpDown numAutoTime;
        private System.Windows.Forms.Label lblAutoTime;
        private System.Windows.Forms.Label lblAutoTimeRemain;
        private System.Windows.Forms.CheckBox chkPrintFilterSettings;
        private System.Windows.Forms.CheckBox chkPrintAlgorithmSettings;
        private System.Windows.Forms.CheckBox chkPrintItterations;
        private System.Windows.Forms.CheckBox chkGroupCoupCross;
        private System.Windows.Forms.CheckBox chkGroupCoupLat;

    }
}
```

APPENDIX G – QEFILT_USB_CONTROL SOURCE CODE

MAIN.H

```
/*******************************************************************************
 * Filename:    main.h
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Discription: This file houses the include files, preprocessor definitions and
 *              function prototypes needed by the QE Filtering System.
 ******************************************************************************/
#ifndef _MAIN_H
#define _MAIN_H

/* Include Files */
#include <p33fj64gp802.h>
#include "strfunc.h"
#include "ampdetect.h"
#include "cmdsm.h"
#include "dac.h"
#include "freqcount.h"
#include "pinconfig.h"
#include "qefilter.h"
#include "spi.h"
#include "uart1.h"

/* Preprocessor Definitions & Macros */
#define _DEBUG_RXSM_                    0
#define _DEBUG_ALGORITHM_               0
#define _DEBUG_CRITICALOSC_             0
#define TRUE                            1
#define FALSE                           0
#define INPUT                           1
#define OUTPUT                          0
#define ON                              1
#define OFF                             0
#define NUL                             0

#define absDiff(val1,val2)              ((val1-val2) > 0 ? (val1-val2) : -1*(val1-val2))
#define ASCII2HEX(val)                  (((val) >= 48 && (val) <= 57) ? ((val) - 48) : \
                                        (((val) >= 65 && (val) <= 70) ? ((val) - 55) : -1))
#define HEX2ASCII(val)                  (((val) >= 0 && (val) <= 9) ? ((val) + 48) : \
                                        (((val) >= 10 && (val) <= 15) ? ((val) + 55) : -1))
#define printSeperator()                txStrUART1(console_str_sep)
#define printBootStr()                  txStrUART1(console_str_boot)
#define printAlgorithmVer()             txStrUART1(algorithm_version);

/* Function Prototypes */
void init(void);

#endif

/* End of File */
```

MAIN.C

```
/*******************************************************************************
 * Filename:    main.c
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Description: This file houses the main loop and initialization functions.  Once
 *              initialized, the system waits for a CMD string to be received, from
 *              the QEFilter PC Application, that contains the algorithm's parameters.
 *              Once the command string has been analzed, the algorithm is run.  It
 *              then waits for the next CMD string.
 ******************************************************************************/
#include "main.h"

/* Device Configuration Bits */

// Boot segment may be written, no boot segement, no boot RAM
_FBS(RBS_NO_RAM & BSS_NO_FLASH  & BWRP_WRPROTECT_OFF);
// No Secure Ram, no secure segment, write protection disabled
_FSS(RSS_NO_RAM & SSS_NO_FLASH & SWRP_WRPROTECT_OFF);
// General Segment and Code protect off, General Segment may be written
_FGS(GSS_OFF & GCP_OFF & GWRP_OFF);
// Primary Oscillator (XT, HS, EC), start-up with user-selected oscillator
_FOSCSEL(FNOSC_PRI & IESO_OFF);
// Clock switching and monitoring disabled, one re-configuration for remappable I/O,
// Primary Oscillator is External Clock
_FOSC(FCKSM_CSDCMD & IOL1WAY_ON & OSCIOFNC_ON & POSCMD_EC);
// Watchdog Timer off
_FWDT(FWDTEN_OFF & WINDIS_OFF & WDTPRE_PR32 & WDTPOST_PS1);
// I2C mapped to SDA1/SCL1, 128ms Power on Reset
_FPOR(ALTI2C_OFF & FPWRT_PWR128);
// JTAG disabled, ICD communicate on PGC1/EMUC1 and PGD1/EMUD1
_FICD(JTAGEN_OFF & ICS_PGD1);


/* Global Variables */
int _UpdateFilter, _ReadFilterFeedBack;
const char * console_str_boot  = "dsPIC33fj64gp802 Acknowledges You\r\n";
const char * console_str_sep   = \
"--------------------------------------------------------------------------------\r\n";
int _UpdateFilter;

/*******************************************************************************
 * Function:    main
 * Parameters:  void
 * Return:      int - never reaches return value
 * Description: Initializes the system and filter variables.  The filter is then
 *              programmed to have all options at their minima or disabled.  It then
 *              waits reads the UART and process its data appropriately.  If the
 *              _UpdateFilte flag is set it will run the tuning Algorithm.
 ******************************************************************************/
int main(void)
{
    init();
```

```
    initFilter();
    programFilter();

            while ( TRUE )
    {
        processRxData();
        if ( _UpdateFilter )
        {
            _UpdateFilter = FALSE;
            updateAnalogTuning();
            if (isRFSwitchOn())
               turnRFSwitchOn();
            else
               turnRFSwitchOff();

            programFilter();
            readFilterFeedback();
        }
    }
}

/*****************************************************************************
 * Function:    init
 * Parameters:  void
 * Return:      void
 * Description: Initializeds the modules and sets up the interrupts.
 *****************************************************************************/
void init(void)
{
    RCONbits.SWDTEN = 0;                    // Software Disable of the Watchdog Timer

    _SPI1IE = 1;                            // Enable SPI1 Transfer Complete Interrupt
    _SPI1IF = 0;                            // Clear SPI1 Event Interrupt Flag
    _SPI2IE = 1;                            // Enable SPI1 Transfer Complete Interrupt
    _SPI2IF = 0;                            // Clear SPI1 Event Interrupt Flag
    _T2IE   = 1;                            // Enable Timer2 Overflow Flag
    _T2IF   = 0;                            // Clear Timer2 Overflow Flag

    initPins();
    initSPI1();
    initSPI2();
    initADC();
    initUART1(BAUDRATE, 0x06, 0x05);        // Configure UART Peripherial
    setState(CMD_STATE_STOP);
    printBootStr();
    printSeperator();

    _UpdateFilter = FALSE;
    _ReadFilterFeedBack = FALSE;
}

/*****************************************************************************
 * Function:    initPins
 * Parameters:  void
 * Return:      void
 * Description: Initializeds the systems pins (see pinconfig.h).
 *****************************************************************************/
void initPins(void)
{
    /* Configure Analog Pins */
    ADPCFG = 0xFFFC;                        // All Digital Pins Except AN0 & AN1

    /* UART1 Configuration */
    configU1RXDPin();                      // Setup U1RX Reprogrammable-Pin Register
    configU1CTSPin();                      // Setup U1CTS Reprogrammable-Pin Register
    configU1TXDPin();                      // Setup U1TX Reprogrammable-Pin Register
    configU1RTSPin();                      // Setup U1RTS Reprogrammable-Pin Register

    /* SPI2 Configuration */
    configSDO2Pin();                        // Setup SDO2 Reprogrammable-Pin Register
    configSCK2Pin();                        // Setup SCK2 Reprogrammable-Pin Register
    setFilterDataAsOutput();
    setFilterClkAsOutput();
    setFilterLatchAsOutput();
    setPinLow(FILTER_CLK);                  // Set FILTER_CLK Low
    setPinLow(FILTER_DATA);                 // Set FILTER_DATA Low
    setPinLow(FILTER_LATCH);                // Set FILTER_LATCH Low

    /* SPI1 Configuration */
    configSDO1Pin();                        // Setup SDO1 Reprogrammable-Pin Register
    configSCK1Pin();                        // Setup SCK1 Reprogrammable-Pin Register
    setBackDACSelAsOutput();                // Set DAC_BACK_SEL Low
    setFrontDACSelAsOutput();               // Set DAC_FRONT_SEL Low
    setPinLow(DAC_DATA);                    // Set DAC_DATA Low
    setPinLow(DAC_CLK);                     // Set DAC_CLK Low
    deselectBothDACs();                     // Set DAC_BACK_SEL & DAC_FRONT_SEL High

    /* Frequency Divider Configuration */
    setFreqDividerAsInput();                // Set FD_OUTPUT as Digital Input
    setFreqDividerSelAsOutput();            // Set FD_SEL as Digital Output
    setPinLow(FD_SEL);                      // Set FD_SEL Low

    /* RF Switch Configuration */
    setRFSwitchAsOutput();                  // Set RFSW as Digital Ouput

    /* Debug LED Configuration */
    setDebugLEDAsOutput();                  // Set Debug LED as Digital Output
    turnDebugLEDOff();                      // Turn Debug Low

    /* Debug BTN Configuration */
    setDebugBtnAsInput();
    _CNIE = 1;                              // Enable Change Notification Interrupts
    _CNIP = 2;                              // Set Interrupt priority
    _CN21IE = 1;                            // Enable Change Notification Interrupt on RB9
    _CNIF = 0;                              // Clear Change Notifitication Interupt Flag

}

/*****************************************************************************
 * Interrupt:   _CNInterrupt
 * Parameters:  void
 * Return:      void
 * Description: Interrupt is triggered when the debug button is clicked. The debug
 *              button simply toggles the debug LED in this implementation.
 *****************************************************************************/
void _ISR __attribute__((auto_psv)) _CNInterrupt(void)
{
    int i;
    for (i = 0; i < 1000; i++); // Debounce
```

```
    if ( isBtnPressed() )
        toggleDebugLED();
    _CNIF = 0; // Clear Change Notification Interrupt Flag
}

/* End of File */
```

QEFILTER.H

```
/*******************************************************************************
 * Filename:    qefilter.h
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Discription: This file houses the preprocessor definitions and function prototypes
 *              needed by the QE Filter Tuning Algorithm.
 *******************************************************************************/
#ifndef _QEFILTER_H
#define _QEFILTER_H

/* Preprocessor Definitions & Macros */
#define FTUNE_DIG_MIN                       165 // Limit the Frequency Range for Reliable Frequency Divider Ouputs
#define FTUNE_DIG_MAX                       255
#define QTUNE_DIG_MIN                       0
#define QTUNE_DIG_MAX                       63
#define COUPLING_MIN                        0
#define COUPLING_MAX                        31
#define ANALOG_MIN                          0
#define ANALOG_MAX                          1023
#define MAX_FREQTUNE_ITTS                   100
#define MAX_CRITOSC_ITTS                    64

#define isFrontEndEnabled()                 (!FENDCON.en ? 1 : 0)

#define enableFrontEnd()                    FENDCON.en    = 0 // Active-Low Enable

#define disableFrontEnd()                   FENDCON.en    = 1

#define isFrontEndADEnabled()               (!FENDCON.ADen ? 1 : 0)

#define enableFrontEndAD()                  FENDCON.ADen  = 0 // Active-Low Enable

#define disableFrontEndAD()                 FENDCON.ADen  = 1

#define isFrontEndFDEnabled()               (!CAPCON1.FDFen ? 1 : 0)

#define enableFrontEndFD()                  CAPCON1.FDFen = 0 // Active-Low Enable

#define disableFrontEndFD()                 CAPCON1.FDFen = 1

#define getFrontEndDigitalFTune()           FENDCON.fTune

#define setFrontEndDigitalFTune(val)        FENDCON.fTune = ((val) > FTUNE_DIG_MAX ? FTUNE_DIG_MAX : ((val) < FTUNE_DIG_MIN ? FTUNE_DIG_MIN : (val)))

#define incFrontEndDigitalFTune()           setFrontEndDigitalFTune(FENDCON.fTune + 1)

#define decFrontEndDigitalFTune()           setFrontEndDigitalFTune(FENDCON.fTune - 1)

#define getFrontEndDigitalQTune()           FENDCON.qTune

#define setFrontEndDigitalQTune(val)        FENDCON.qTune = ((val) > QTUNE_DIG_MAX ? QTUNE_DIG_MAX : ((val) < QTUNE_DIG_MIN ? QTUNE_DIG_MIN : (val)))

#define incFrontEndDigitalQTune()           setFrontEndDigitalQTune(FENDCON.qTune + 1)

#define decFrontEndDigitalQTune()           setFrontEndDigitalQTune(FENDCON.qTune - 1)

#define isBackEndEnabled()                  (!BENDCON.en ? 1 : 0)

#define enableBackEnd()                     BENDCON.en = 0 // Active-Low Enable

#define disableBackEnd()                    BENDCON.en = 1

#define isBackEndADEnabled()                (!BENDCON.ADen ? 1 : 0)

#define enableBackEndAD()                   BENDCON.ADen = 0 // Active-Low Enable

#define disableBackEndAD()                  BENDCON.ADen = 1

#define isBackEndFDEnabled()                (!CAPCON2.FDBen ? 1 : 0)

#define enableBackEndFD()                   CAPCON2.FDBen = 0 // Active-Low Enable

#define disableBackEndFD()                  CAPCON2.FDBen = 1

#define getBackEndDigitalFTune()            BENDCON.fTune

#define setBackEndDigitalFTune(val)         BENDCON.fTune = ((val) > FTUNE_DIG_MAX ? FTUNE_DIG_MAX : ((val) < FTUNE_DIG_MIN ? FTUNE_DIG_MIN : (val)))

#define incBackEndDigitalFTune()            setBackEndDigitalFTune(BENDCON.fTune + 1)

#define decBackEndDigitalFTune()            setBackEndDigitalFTune(BENDCON.fTune - 1)

#define getBackEndDigitalQTune()            BENDCON.qTune

#define setBackEndDigitalQTune(val)         BENDCON.qTune = ((val) > QTUNE_DIG_MAX ? QTUNE_DIG_MAX : ((val) < QTUNE_DIG_MIN ? QTUNE_DIG_MIN : (val)))

#define incBackEndDigitalQTune()            setBackEndDigitalQTune(BENDCON.qTune + 1)

#define decBackEndDigitalQTune()            setBackEndDigitalQTune(BENDCON.qTune - 1)

#define getCouplingUpper()                  CAPCON1.upper

#define setCouplingUpper(val)               CAPCON1.upper = ((val) > COUPLING_MAX ? COUPLING_MAX : ((val) < COUPLING_MIN ? COUPLING_MIN : (val)))

#define getCouplingLower()                  CAPCON1.lower

#define setCouplingLower(val)               CAPCON1.lower = ((val) > COUPLING_MAX ? COUPLING_MAX : ((val) < COUPLING_MIN ? COUPLING_MIN : (val)))

#define getCouplingUFLB()                   CAPCON2.UFLB

#define setCouplingUFLB(val)                CAPCON2.UFLB  = ((val) > COUPLING_MAX ? COUPLING_MAX : ((val) < COUPLING_MIN ? COUPLING_MIN : (val)))

#define getCouplingLFUB()                   CAPCON2.LFUB

#define setCouplingLFUB(val)                CAPCON2.LFUB  = ((val) > COUPLING_MAX ? COUPLING_MAX : ((val) < COUPLING_MIN ? COUPLING_MIN : (val)))

#define getFrontEndAnalogFTune()            ANALOG.FANAF

#define setFrontEndAnalogFTune(val)         ANALOG.FANAF = ((val) > ANALOG_MAX ? ANALOG_MAX : ((val) < ANALOG_MIN ? ANALOG_MIN : (val)))

#define getFrontEndAnalogQTune()            ANALOG.FANAQ
```

```
#define setFrontEndAnalogQTune(val)         ANALOG.FANAQ = ((val) > ANALOG_MAX ? ANALOG_MAX : ((val) < ANALOG_MIN ? ANALOG_MIN : (val)))
#define getBackEndAnalogFTune()             ANALOG.BANAF
#define setBackEndAnalogFTune(val)          ANALOG.BANAF = ((val) > ANALOG_MAX ? ANALOG_MAX : ((val) < ANALOG_MIN ? ANALOG_MIN : (val)))
#define getBackEndAnalogQTune()             ANALOG.BANAQ
#define setBackEndAnalogQTune(val)          ANALOG.BANAQ = ((val) > ANALOG_MAX ? ANALOG_MAX : ((val) < ANALOG_MIN ? ANALOG_MIN : (val)))
#define printFrontEndStatus()               (isFrontEndEnabled() ? txStrUART1("Front-End Enabled\r\n") : txStrUART1("Front-End Disabled\r\n"))
#define printFrontEndADStatus()             (isFrontEndADEnabled() ? txStrUART1("Front-End AD Enabled\r\n") : txStrUART1("Front-End AD Disabled\r\n"))
#define printFrontEndFDStatus()             (isFrontEndFDEnabled() ? txStrUART1("Front-End FD Enabled\r\n") : txStrUART1("Front-End FD Disabled\r\n"))
#define printFrontEndAD()                   strPopulate16Bit(strFrontEndAD,FrontEndAD,'#',4)
#define printFrontEndNonOsc()               strPopulate16Bit(strFrontEndNonOsc,FrontEndNonOsc,'#',4)
#define printFrontEndFCnt()                 strPopulate16Bit(strFrontEndFCnt,FrontEndFCnt,'#',4)
#define printFrontEndDigitalQTune()         strPopulate16Bit(strFrontEndDigitalQTune,getFrontEndDigitalQTune(),'#',2)
#define printFrontEndAnalogQTune()          strPopulate16Bit(strFrontEndAnalogQTune,getFrontEndAnalogQTune(),'#',4)
#define printFrontEndDigitalFTune()         strPopulate16Bit(strFrontEndDigitalFTune,getFrontEndDigitalFTune(),'#',3)
#define printFrontEndAnalogFTune()          strPopulate16Bit(strFrontEndAnalogFTune,getFrontEndAnalogFTune(),'#',4)
#define printBackEndStatus()               (isBackEndEnabled() ? txStrUART1("Back-End Enabled\r\n") : txStrUART1("Back-End Disabled\r\n"))
#define printBackEndADStatus()             (isBackEndADEnabled() ? txStrUART1("Back-End AD Enabled\r\n") : txStrUART1("Back-End AD Disabled\r\n"))
#define printBackEndFDStatus()             (isBackEndFDEnabled() ? txStrUART1("Back-End FD Enabled\r\n") : txStrUART1("Back-End FD Disabled\r\n"))
#define printBackEndAD()                    strPopulate16Bit(strBackEndAD,BackEndAD,'#',4)
#define printBackEndNonOsc()                strPopulate16Bit(strBackEndNonOsc,BackEndNonOsc,'#',4)
#define printBackEndFCnt()                  strPopulate16Bit(strBackEndFCnt,BackEndFCnt,'#',4)
#define printBackEndDigitalQTune()          strPopulate16Bit(strBackEndDigitalQTune,getBackEndDigitalQTune(),'#',2)
#define printBackEndAnalogQTune()           strPopulate16Bit(strBackEndAnalogQTune,getBackEndAnalogQTune(),'#',4)
#define printBackEndDigitalFTune()          strPopulate16Bit(strBackEndDigitalFTune,getBackEndDigitalFTune(),'#',3)
#define printBackEndAnalogFTune()           strPopulate16Bit(strBackEndAnalogFTune,getBackEndAnalogFTune(),'#',4)
#define printCouplingUpper()                strPopulate16Bit(strCouplingUpper,CAPCON1.upper,'#',2)
#define printCouplingLower()                strPopulate16Bit(strCouplingLower,CAPCON1.lower,'#',2)
#define printCouplingUFLB()                 strPopulate16Bit(strCouplingUFLB,CAPCON2.UFLB,'#',2)
#define printCouplingLFUB()                 strPopulate16Bit(strCouplingLFUB,CAPCON2.LFUB,'#',2)
#define printRFSwitchStatus()               (isRFSwitchOn() ? txStrUART1("RF Switch On\r\n") : txStrUART1("RF Switch Off\r\n"))
#define printCenterFreq()                   strPopulate16Bit(strCenterFreq,CenterFreq,'#',4)
#define printFreqTol()                      strPopulate16Bit(strFreqTol,FreqTol,'#',3)
#define printFrontEndADThresh1()            strPopulate16Bit(strFrontEndADThresh1,FrontEndADThresh1,'#',3)
#define printFrontEndADThresh2()            strPopulate16Bit(strFrontEndADThresh2,FrontEndADThresh2,'#',3)
#define printFrontEndQOffset()              strPopulate16Bit(strFrontEndQOffset,FrontEndQOffset,'#',2)
#define printFrontEndQBackOff()             strPopulate16Bit(strFrontEndQBackOff,FrontEndQBackOff,'#',2)
#define printFrontEndFOffset()              strPopulate16Bit(strFrontEndFOffset,FrontEndFOffset,'#',2)
#define printBackEndADThresh1()             strPopulate16Bit(strBackEndADThresh1,BackEndADThresh1,'#',3)
#define printBackEndADThresh2()             strPopulate16Bit(strBackEndADThresh2,BackEndADThresh2,'#',3)
#define printBackEndQOffset()               strPopulate16Bit(strBackEndQOffset,BackEndQOffset,'#',2)
#define printBackEndQBackOff()              strPopulate16Bit(strBackEndQBackOff,BackEndQBackOff,'#',2)
#define printBackEndFOffset()               strPopulate16Bit(strBackEndFOffset,BackEndFOffset,'#',2)

/* Function Prototypes */
void initFilter(void);
void printFilterOptions(void);
void updateFilterData(void);
void programFilter(void);
void prgmDelay(void);
void updateAnalogTuning(void);
void readFilterFeedback(void);
void updateFENDCON(int);
void updateBENDCON(int);
void updateCAPCON1(int);
void updateCAPCON2(int);
void updateFANAF(int);
void updateFANAQ(int);
void updateBANAF(int);
void updateBANAQ(int);
void updateDEBUG(int);
int isRFSwitchOn(void);
void turnRFSwitchOn(void);
void turnRFSwitchOff(void);

#endif

/* End of File */
```

QEFILTER.C

```
/*********************************************************************************
 * Filename:    qefilter.c
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Description: This file houses the functions needed to implement the QE Filter
 *              tuning algorithm.
 *********************************************************************************/
#include "main.h"

/* Global Variables */
char strCenterFreq[]           = "Center Frequency: ###.# MHz\r\n";
char strFreqTol[]              = "Frequency Tolerance: ##.# MHz\r\n";
char strFrontEndADThresh1[]    = "Front-End AD Threshold 1: ###\r\n";
char strFrontEndADThresh2[]    = "Front-End AD Threshold 2: ###\r\n";
char strFrontEndQOffset[]      = "Front-End Q-Offset: ##\r\n";
char strFrontEndQBackOff[]     = "Front-End Q-BackOff: ##\r\n";
char strFrontEndFOffset[]      = "Front-End F-Offset: ##\r\n";
char strBackEndADThresh1[]     = "Back-End AD Threshold 1: ###\r\n";
char strBackEndADThresh2[]     = "Back-End AD Threshold 2: ###\r\n";
char strBackEndQOffset[]       = "Back-End Q-Offset: ##\r\n";
char strBackEndQBackOff[]      = "Back-End Q-BackOff: ##\r\n";
char strBackEndFOffset[]       = "Back-End F-Offset: ##\r\n";
char strCouplingUpper[]        = "Coupling Upper: ##\r\n";
char strCouplingLower[]        = "Coupling Lower: ##\r\n";
char strCouplingUFLB[]         = "Coupling UFLB: ##\r\n";
char strCouplingLFUB[]         = "Coupling LFUB: ##\r\n";
char strFrontEndAD[]           = "Front-End Amp Detector: ####\r\n";
char strFrontEndNonOsc[]       = "Front-End Non-Osc: ####\r\n";
char strFrontEndFCnt[]         = "Front-End Freq Count: ###.# MHz\r\n";
char strFrontEndDigitalQTune[] = "Front-End Digital Q-Tune: ##\r\n";
char strFrontEndAnalogQTune[]  = "Front-End Analog Q-Tune: ####\r\n";
char strFrontEndDigitalFTune[] = "Front-End Digital F-Tune: ###\r\n";
char strFrontEndAnalogFTune[]  = "Front-End Analog F-Tune: ####\r\n";
char strBackEndAD[]            = "Back-End Amp Detector: ####\r\n";
char strBackEndNonOsc[]        = "Back-End Non-Osc: ####\r\n";
char strBackEndFCnt[]          = "Back-End Freq Count: ###.# MHz\r\n";
char strBackEndDigitalQTune[]  = "Back-End Digital Q-Tune: ##\r\n";
char strBackEndAnalogQTune[]   = "Back-End Analog Q-Tune: ####\r\n";
char strBackEndDigitalFTune[]  = "Back-End Digital F-Tune: ###\r\n";
char strBackEndAnalogFTune[]   = "Back-End Analog F-Tune: ####\r\n";
char strDebug1[]               = "^####";
char strDebug2[]               = "&####";
char strDebug3[]               = "&####|\r\n";

int FrontEndAD, BackEndAD, FrontEndFCnt, BackEndFCnt;
int filterData[4];

struct {
    unsigned int en:1;
    unsigned int fTune:8;
    unsigned int qTune:6;
    unsigned int ADen:1;
} FENDCON;

struct {
    unsigned int en:1;
    unsigned int fTune:8;
    unsigned int qTune:6;
    unsigned int ADen:1;
} BENDCON;

struct {
    unsigned int upper:5;
    unsigned int lower:5;
    unsigned int FDFen:1;
} CAPCON1;

struct {
    unsigned int UFLB:5;
    unsigned int LFUB:5;
    unsigned int FDBen:1;
} CAPCON2;

struct {
    unsigned int FANAF:10;
    unsigned int FANAQ:10;
    unsigned int BANAF:10;
    unsigned int BANAQ:10;
} ANALOG;

struct {
    unsigned int ADFen:1;
    unsigned int FDFen:1;
    unsigned int ADBen:1;
    unsigned int FDBen:1;
    unsigned int RFOn:1;
} DEBUG;

extern const char *console_str_sep;
extern int _PrintFilterSettings;


/*********************************************************************************
 * Function:    updateFENDCON
 * Parameters:  int val - appropriate portion of control string
 * Return:      void
 * Description: Sets the contents of the FENDCON programming register.
 *********************************************************************************/
void updateFENDCON(int val)
{
    FENDCON.en    = val & 1;
    FENDCON.fTune = (val >> 1) & 255;
    FENDCON.qTune = (val >> 9) & 63;
    FENDCON.ADen  = (val >> 15) & 1;
}
```

```
/*******************************************************************************
 * Function:    updateBENDCON
 * Parameters:  int val - appropriate portion of control string
 * Return:      void
 * Description: Sets the contents of the BENDCON programming register.
 *******************************************************************************/
void updateBENDCON(int val)
{
    BENDCON.en    = val & 1;
    BENDCON.fTune = (val >> 1) & 255;
    BENDCON.qTune = (val >> 9) & 63;
    BENDCON.ADen  = (val >> 15) & 1;
}

/*******************************************************************************
 * Function:    updateCAPCON1
 * Parameters:  int val - appropriate portion of control string
 * Return:      void
 * Description: Sets the contents of the CAPCON1 programming register.
 *******************************************************************************/
void updateCAPCON1(int val)
{
    CAPCON1.upper = val & 31;
    CAPCON1.lower = (val >> 5) & 31;
    CAPCON1.FDFen = (val >> 10) & 1;
}

/*******************************************************************************
 * Function:    updateCAPCON2
 * Parameters:  int val - appropriate portion of control string
 * Return:      void
 * Description: Sets the contents of the CAPCON2 programming register.
 *******************************************************************************/
void updateCAPCON2(int val)
{
    CAPCON2.UFLB = val & 31;
    CAPCON2.LFUB = (val >> 5) & 31;
    CAPCON2.FDBen = (val >> 10) & 1;
}

/*******************************************************************************
 * Function:    updateFANAF
 * Parameters:  int val - appropriate portion of control string
 * Return:      void
 * Description: Sets the Front-End Analog F-Tuning value of the ANALOG
 *              programming register.
 *******************************************************************************/
void updateFANAF(int val)
{
    ANALOG.FANAF = val & 1023;
}

/*******************************************************************************
 * Function:    updateFANAQ
 * Parameters:  int val - appropriate portion of control string
 * Return:      void
 * Description: Sets the Front-End Analog Q-Tuning value of the ANALOG
 *              programming register.
 *******************************************************************************/
void updateFANAQ(int val)
{
    ANALOG.FANAQ = val & 1023;
}

/*******************************************************************************
 * Function:    updateBANAF
 * Parameters:  int val - appropriate portion of control string
 * Return:      void
 * Description: Sets the Back-End Analog F-Tuning value of the ANALOG
 *              programming register.
 *******************************************************************************/
void updateBANAF(int val)
{
    ANALOG.BANAF = val & 1023;
}

/*******************************************************************************
 * Function:    updateBANAQ
 * Parameters:  int val - appropriate portion of control string
 * Return:      void
 * Description: Sets the Back-End Analog Q-Tuning value of the ANALOG
 *              programming register.
 *******************************************************************************/
void updateBANAQ(int val)
{
    ANALOG.BANAQ = val & 1023;
}

/*******************************************************************************
 * Function:    updateDEBUG
 * Parameters:  int val - appropriate portion of control string
 * Return:      void
 * Description: Sets the contents of the DEBUG register.
 *******************************************************************************/
void updateDEBUG(int val)
{
    DEBUG.ADFen = val & 1;
    DEBUG.FDFen = (val >> 1) & 1;
    DEBUG.ADBen = (val >> 2) & 1;
    DEBUG.FDBen = (val >> 3) & 1;
    DEBUG.RFOn = (val >> 4) & 1;
}

/*******************************************************************************
 * Function:    initFilter
 * Parameters:  void
 * Return:      void
 * Description: Initializes the Filter Controls to their minima and disables all
 *              of the enable variables.
 *******************************************************************************/
void initFilter(void)
{
    disableFrontEnd();
    setFrontEndDigitalFTune(FTUNE_DIG_MIN);
    setFrontEndDigitalQTune(QTUNE_DIG_MIN);
    disableFrontEndAD();
    disableFrontEndFD();

    disableBackEnd();
    setBackEndDigitalFTune(FTUNE_DIG_MIN);
    setBackEndDigitalQTune(QTUNE_DIG_MIN);
    disableBackEndAD();
    disableBackEndFD();
```

```
    setCouplingUpper(COUPLING_MIN);
    setCouplingLower(COUPLING_MIN);
    setCouplingUFLB(COUPLING_MIN);
    setCouplingLFUB(COUPLING_MIN);

    setFrontEndAnalogFTune(ANALOG_MIN);
    setFrontEndAnalogQTune(ANALOG_MIN);
    setBackEndAnalogFTune(ANALOG_MIN);
    setBackEndAnalogQTune(ANALOG_MIN);

    DEBUG.ADFen = 0;
    DEBUG.FDFen = 0;
    DEBUG.ADBen = 0;
    DEBUG.FDBen = 0;
    DEBUG.RFOn  = 1;
}

/*******************************************************************************
 * Function:    printFilterOptions
 * Parameters:  void
 * Return:      void
 * Description: Prints each of the the filter options to the Console.
 ******************************************************************************/
void printFilterOptions(void)
{
    printSeperator();
    printFrontEndStatus();
    printFrontEndADStatus();
    printFrontEndFDStatus();
    printFrontEndDigitalQTune();
    printFrontEndAnalogQTune();
    printFrontEndDigitalFTune();
    printFrontEndAnalogFTune();
    printBackEndStatus();
    printBackEndADStatus();
    printBackEndFDStatus();
    printBackEndDigitalQTune();
    printBackEndAnalogQTune();
    printBackEndDigitalFTune();
    printBackEndAnalogFTune();
    printCouplingUpper();
    printCouplingLower();
    printCouplingUFLB();
    printCouplingLFUB();
    printRFSwitchStatus();
    printSeperator();
}

/*******************************************************************************
 * Function:    updateFilterData
 * Parameters:  void
 * Return:      void
 * Description: Formats and stores the filter options to be programmed.
 ******************************************************************************/
void updateFilterData(void)
{
    filterData[0] = ((CAPCON2.FDBen & 1) << 10)             |
                    ((CAPCON2.UFLB & COUPLING_MAX) << 5 )   |
                    (CAPCON2.LFUB & COUPLING_MAX);
    filterData[1] = ((CAPCON1.FDFen & 1) << 10)             |
                    ((CAPCON1.lower & COUPLING_MAX) << 5)   |
                    (CAPCON1.upper & COUPLING_MAX);
    filterData[2] = ((BENDCON.ADen & 1) << 15)              |
                    ((BENDCON.qTune & QTUNE_DIG_MAX) << 9)  |
                    ((BENDCON.fTune & FTUNE_DIG_MAX) << 1)  |
                    (BENDCON.en & 1);
    filterData[3] = ((FENDCON.ADen & 1) << 15)              |
                    ((FENDCON.qTune & QTUNE_DIG_MAX) << 9)  |
                    ((FENDCON.fTune & FTUNE_DIG_MAX) << 1)  |
                    (FENDCON.en & 1);
}

/*******************************************************************************
 * Function:    programFilter
 * Parameters:  void
 * Return:      void
 * Description: The filter is programmed by transmitting the 64 programming bits to
 *              the serial-to-parallel register of the filter via SPI.  Once all 64
 *              bits are transmitted, the latch line is raised, a clock pulse is
 *              generated and the latch line is lowered storing the bits.
 ******************************************************************************/
void programFilter(void)
{
    updateFilterData();

    setPinLow(FILTER_CLK);
    setPinLow(FILTER_DATA);
    setPinLow(FILTER_LATCH);
    enableSPI2();
    writeSPI2(filterData[0]);
    writeSPI2(filterData[1]);
    writeSPI2(filterData[2]);
    writeSPI2(filterData[3]);
    disableSPI2();

    setFilterLatchAsOutput();
    setFilterDataAsOutput();
    setFilterClkAsOutput();
    setPinLow(FILTER_LATCH);
    setPinLow(FILTER_CLK);
    prgmDelay();
    setPinHigh(FILTER_LATCH);
    prgmDelay();
    setPinHigh(FILTER_CLK);
    prgmDelay();
    setPinLow(FILTER_DATA);
    setPinLow(FILTER_LATCH);
    prgmDelay();
    setPinLow(FILTER_CLK);
    prgmDelay();
}
```

```
/*******************************************************************************
 * Function:    prgmDelay
 * Parameters:  void
 * Return:      void
 * Description: A simplistic delay used to ensure timing requirements are met during
 *              the filter programming process.
 *******************************************************************************/
void prgmDelay(void)
{
    Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop();
    Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop();
}

/*******************************************************************************
 * Function:    updateAnalogTuning
 * Parameters:  void
 * Return:      void
 * Description: Programs the DACS with the appropriate current Analog F-Tune
 *              and Q-Tune values.
 *******************************************************************************/
void updateAnalogTuning(void)
{
    programFrontEndDAC(loadInputRegA(getFrontEndAnalogFTune()));
    programFrontEndDAC(loadInputRegB(getFrontEndAnalogQTune()));
    programFrontEndDAC(loadDACRegsABUpdateOutputsAB());
    programBackEndDAC(loadInputRegA(getBackEndAnalogQTune()));
    programBackEndDAC(loadInputRegB(getBackEndAnalogFTune()));
    programBackEndDAC(loadDACRegsABUpdateOutputsAB());
}


/*******************************************************************************
 * Function:    readFilterFeedback
 * Parameters:  void
 * Return:      void
 * Description: Reads and stores the selected Amplitude Detectors and Frequency
 *              Dividers.  The values are then transmitted in a control string format.
 *******************************************************************************/
void readFilterFeedback(void)
{
    int _SendDebugInfo = FALSE;

    FrontEndAD = FrontEndFCnt = BackEndAD = BackEndFCnt = 0;
    if (DEBUG.ADFen)
    {
        FrontEndAD = readFrontEndAD();
        printFrontEndAD();
        _SendDebugInfo = TRUE;
    }
    if (DEBUG.FDFen)
    {
        FrontEndFCnt = readFrontEndFD();
        printFrontEndFCnt();
        _SendDebugInfo = TRUE;
    }
    if (DEBUG.ADBen)
    {
        BackEndAD = readBackEndAD();
        printBackEndAD();
        _SendDebugInfo = TRUE;
    }
    if (DEBUG.FDBen)
    {
        BackEndFCnt = readBackEndFD();
        printBackEndFCnt();
        _SendDebugInfo = TRUE;
    }

    if (_SendDebugInfo)
    {
        txByteUART1(0); txByteUART1(0); txByteUART1(0); txByteUART1(0); txByteUART1(0); txByteUART1(0); txByteUART1(0);
        txByteUART1(0); txByteUART1(0); txByteUART1(0); txByteUART1(0); txByteUART1(0); txByteUART1(0); txByteUART1(0);
        txByteUART1(0); txByteUART1(0); txByteUART1(0); txByteUART1(0); txByteUART1(0); txByteUART1(0); txByteUART1(0);

        strPopulate16Bit(strDebug1,FrontEndAD,'#',4);
        strPopulate16Bit(strDebug2,FrontEndFCnt,'#',4);
        strPopulate16Bit(strDebug2,BackEndAD,'#',4);
        strPopulate16Bit(strDebug3,BackEndFCnt,'#',4);
    }
}

int isRFSwitchOn(void)
{
    return !DEBUG.RFOn;
}

void turnRFSwitchOn(void)
{
    setPinLow(RFSW);
    DEBUG.RFOn = 0;
}

void turnRFSwitchOff(void)
{
    setPinHigh(RFSW);
    DEBUG.RFOn = 1;
}

/* End of File */
```

CMDSM.H

```
/*******************************************************************************
 * Filename:    cmdsm.h
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Discription: This file houses the preprocessor definitions and function prototypes
 *              needed by the CMD State-Machine.
 *******************************************************************************/
#ifndef _CMDSS_H
#define _CMDSS_H
```

```
/* Preprocessor Definitions & Macros */
#define CMD_STATE_NAME_LEN                  7
#define CMD_NUMSTATES                       10
#define CMD_STATE_FENDCON                   0
#define CMD_STATE_BENDCON                   1
#define CMD_STATE_CAPCON1                   2
#define CMD_STATE_CAPCON2                   3
#define CMD_STATE_FANAF                     4
#define CMD_STATE_FANAQ                     5
#define CMD_STATE_BANAF                     6
#define CMD_STATE_BANAQ                     7
#define CMD_STATE_DEBUG                     8
#define CMD_STATE_STOP                      9

#define nextState()                         setState(RxCMD[curState].nextState)
#define printRxData()                       strPopulate16Bit(strRxData,tmpRegVal,'#',4)
#define printItteration()                   strPopulate16Bit(strItt,algItt,'#',4)

/* Function Prototypes */
void initStateMachine(void);
void setState(int);
void printCurState(void);
void processRxData(void);
void processTxData(void);
void fendconAction(void);
void bendconAction(void);
void capcon1Action(void);
void capcon2Action(void);
void fanafAction(void);
void fanaqAction(void);
void banafAction(void);
void banaqAction(void);
void debugAction(void);
void stopAction(void);

/* Type Definitions */
typedef struct {
   char stateName[CMD_STATE_NAME_LEN];
   int nextState;
   void (*action)();
} CommandStateMachine;

#endif

/* End of File */
```

CMDSM.C

```
/*******************************************************************************
 * Filename:    cmdsm.c
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Discription: This file houses the functions needed to read and interpret the CMD
 *              string being sent from the QEFilter PC application, via UART,
 *              containing the algorithms parameters.
 *******************************************************************************/
#include "main.h"

/* Global Variables */
unsigned char rxByte;
char strRxData[] = "tmpRegVal = ####\r\n";
char strItt[] = "~~~~~ Algorithm Itteration #### ~~~~~\r\n";
int curState, curRxByte, _PrintFilterOptions, tmpRegVal, algItt;
extern int _UpdateFilter;

CommandStateMachine RxCMD[CMD_NUMSTATES] =
{
    {"FENDCON", CMD_STATE_BENDCON, fendconAction},
    {"BENDCON", CMD_STATE_CAPCON1, bendconAction},
    {"CAPCON1", CMD_STATE_CAPCON2, capcon1Action},
    {"CAPCON2", CMD_STATE_FANAF,   capcon2Action},
    {"FANAF",   CMD_STATE_FANAQ,   fanafAction},
    {"FANAQ",   CMD_STATE_BANAF,   fanaqAction},
    {"BANAF",   CMD_STATE_BANAQ,   banafAction},
    {"BANAQ",   CMD_STATE_DEBUG,   banaqAction},
    {"DEBUG",   CMD_STATE_STOP,    debugAction},
    {"STOP",    CMD_STATE_FENDCON, stopAction}
};

/*******************************************************************************
 * Function:    initStateMachine
 * Parameters:  void
 * Return:      void
 * Description: Initializes the variables used by the CMD State Machine.
 *******************************************************************************/
void initStateMachine(void)
{
    rxByte    = NUL;
    curRxByte = 0;
    tmpRegVal = 0;
    _PrintFilterOptions = FALSE;
}

/*******************************************************************************
 * Function:    setState
 * Parameters:  int state - valid inputs defined in cdsms.h
 * Return:      void
 * Description: Sets the the current state variable based on the input variable state
 *              and clears the curRxByte (count of the number of received bytes).
 *******************************************************************************/
void setState(int state)
{
    curState = state;
    curRxByte = 0;

    #if _DEBUG_RXSM_ == 1
        txStrUART1("Entering ");
        txStrUART1(RxCMD[curState].stateName);
        txStrUART1("\r\n");
    #endif
}
```

```
/*******************************************************************************
 * Function:    processRxData
 * Parameters:  void
 * Return:      void
 * Description: Reads the current rxByte from UART1 and either sets the appropriate
 *              state or executes the current states action.
 *******************************************************************************/
void processRxData(void)
{
    rxByteUART1(&rxByte);

    switch ( rxByte )
    {
        case NUL:
            return;
        case '@': // Beginning of Command String
            setState(CMD_STATE_FENDCON);
            break;
        case '&': // Data Seperator within Command String
            nextState();
            break;
        case '|': // End of Command String
            setState(CMD_STATE_STOP);
            _PrintFilterOptions = TRUE;
            _UpdateFilter = TRUE;
            break;
        default:
            break;
    }

    if ( rxByte != '@' && rxByte != '&' )
        (*RxCMD[curState].action)();
    rxByte = NUL;
}

/*******************************************************************************
 * Function:    fendconAction
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 4 rxBytes from ASCII to Hex.  It then
 *              passes the Hex values to the updateFENDCON function.
 *******************************************************************************/
void fendconAction(void)
{
    switch ( curRxByte++ )
    {
        case 0:
            tmpRegVal = (int) (ASCII2HEX(rxByte) << 12);
            break;
        case 1:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 8);
            break;
        case 2:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 4);
            break;
        case 3:
            tmpRegVal += (int) (ASCII2HEX(rxByte));

            updateFENDCON(tmpRegVal);

            #if _DEBUG_RXSM_ == 1
                printRxData();
            #endif

            tmpRegVal = 0;
            break;
        default:
            break;
    }
}

/*******************************************************************************
 * Function:    bendconAction
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 4 rxBytes from ASCII to Hex.  It then
 *              passes the Hex values to the updateBENDCON function.
 *******************************************************************************/
void bendconAction(void)
{
    switch ( curRxByte++ )
    {
        case 0:
            tmpRegVal = (int) (ASCII2HEX(rxByte) << 12);
            break;
        case 1:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 8);
            break;
        case 2:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 4);
            break;
        case 3:
            tmpRegVal += (int) (ASCII2HEX(rxByte));

            updateBENDCON(tmpRegVal);

            #if _DEBUG_RXSM_ == 1
                printRxData();
            #endif

            tmpRegVal = 0;
            break;
        default:
            break;
    }
}

/*******************************************************************************
 * Function:    capcon1Action
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 3 rxBytes from ASCII to Hex.  It then
 *              passes the Hex calues to the updateCAPCON1 function.
 *******************************************************************************/
void capcon1Action(void)
{
    switch ( curRxByte++ )
    {
        case 0:
            tmpRegVal = (int) (ASCII2HEX(rxByte) << 8);
            break;
        case 1:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 4);
            break;
        case 2:
```

```
            tmpRegVal += (int) (ASCII2HEX(rxByte));

            updateCAPCON1(tmpRegVal);

            #if _DEBUG_RXSM_ == 1
                printRxData();
            #endif

            tmpRegVal = 0;
            break;
        default:
            break;
    }
}

/*******************************************************************************
 * Function:    capcon2Action
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 3 rxBytes from ASCII to Hex.  It then
 *              passes the Hex calues to the updateCAPCON2 function.
 *******************************************************************************/
void capcon2Action(void)
{
    switch ( curRxByte++ )
    {
        case 0:
            tmpRegVal = (int) (ASCII2HEX(rxByte) << 8);
            break;
        case 1:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 4);
            break;
        case 2:
            tmpRegVal += (int) (ASCII2HEX(rxByte));

            updateCAPCON2(tmpRegVal);

            #if _DEBUG_RXSM_ == 1
                printRxData();
            #endif

            tmpRegVal = 0;
            break;
        default:
            break;
    }
}

/*******************************************************************************
 * Function:    fanafAction
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 3 rxBytes from ASCII to Hex.  It then
 *              passes the Hex calues to the updateFANAF function.
 *******************************************************************************/
void fanafAction(void)
{
    switch ( curRxByte++ )
    {
        case 0:
            tmpRegVal = (int) (ASCII2HEX(rxByte) << 8);
            break;
        case 1:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 4);
            break;
        case 2:
            tmpRegVal += (int) (ASCII2HEX(rxByte));

            updateFANAF(tmpRegVal);

            #if _DEBUG_RXSM_ == 1
                printRxData();
            #endif

            tmpRegVal = 0;
            break;
        default:
            break;
    }
}

/*******************************************************************************
 * Function:    fanaqAction
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 3 rxBytes from ASCII to Hex.  It then
 *              passes the Hex calues to the updateFANAF function.
 *******************************************************************************/
void fanaqAction(void)
{
    switch ( curRxByte++ )
    {
        case 0:
            tmpRegVal = (int) (ASCII2HEX(rxByte) << 8);
            break;
        case 1:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 4);
            break;
        case 2:
            tmpRegVal += (int) (ASCII2HEX(rxByte));

            updateFANAQ(tmpRegVal);

            #if _DEBUG_RXSM_ == 1
                printRxData();
            #endif

            tmpRegVal = 0;
            break;
        default:
            break;
    }
}

/*******************************************************************************
 * Function:    banafAction
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 3 rxBytes from ASCII to Hex.  It then
 *              passes the Hex calues to the updateBANAF function.
 *******************************************************************************/
void banafAction(void)
{
    switch ( curRxByte++ )
    {
```

```
        case 0:
            tmpRegVal = (int) (ASCII2HEX(rxByte) << 8);
            break;
        case 1:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 4);
            break;
        case 2:
            tmpRegVal += (int) (ASCII2HEX(rxByte));

            updateBANAF(tmpRegVal);

            #if _DEBUG_RXSM_ == 1
                printRxData();
            #endif

            tmpRegVal = 0;
            break;
        default:
            break;
    }
}

/*******************************************************************************
 * Function:    banaqAction
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 3 rxBytes from ASCII to Hex.  It then
 *              passes the Hex calues to the updateBANAF function.
 ******************************************************************************/
void banaqAction(void)
{
    switch ( curRxByte++ )
    {
        case 0:
            tmpRegVal = (int) (ASCII2HEX(rxByte) << 8);
            break;
        case 1:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 4);
            break;
        case 2:
            tmpRegVal += (int) (ASCII2HEX(rxByte));

            updateBANAQ(tmpRegVal);

            #if _DEBUG_RXSM_ == 1
                printRxData();
            #endif

            tmpRegVal = 0;
            break;
        default:
            break;
    }
}

/*******************************************************************************
 * Function:    debugAction
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 2 rxByte from ASCII to Hex.  It then
 *              passes the Hex calues to the updateDEBUG function.
 ******************************************************************************/
void debugAction(void)
{
    switch ( curRxByte++ )
    {
        case 0:
            tmpRegVal = (int) (ASCII2HEX(rxByte) << 4);
            break;
        case 1:
            tmpRegVal += (int) (ASCII2HEX(rxByte));

            updateDEBUG(tmpRegVal);

            #if _DEBUG_RXSM_ == 1
                printRxData();
            #endif

            tmpRegVal = 0;
            break;
        default:
            break;
    }
}

/*******************************************************************************
 * Function:    stopAction
 * Parameters:  void
 * Return:      void
 * Description: Prints the filter options, when the flag is set.
 ******************************************************************************/
void stopAction(void)
{
    if (_PrintFilterOptions)
    {
        _PrintFilterOptions = FALSE;
        #if _DEBUG_ == 1
            printFilterOptions();
        #endif
    }
}

/* End of File */
```

PINCONFIG.H

```
/*******************************************************************************
 * Filename:    pinconfig.h
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Discription: This file houses the preprocessor definitions and function prototypes
 *              needed by the QE Filtering System.
 ******************************************************************************/
#ifndef _PINCONFIG_H
#define _PINCONFIG_H
```

```
/* Preprocessor Definitions & Macros */
#define CP2103_TXD                          _RB2
#define CP2103_RXD                          _RB3
#define CP2103_CTS                          _R10
#define CP2103_RTS                          _RB5
#define DAC_DATA                            _RB11
#define DAC_CLK                             _RB12
#define DAC_FRONT_SEL                       _RB7
#define DAC_BACK_SEL                        _RB6
#define FILTER_DATA                         _RB14
#define FD_SEL                              _RB4
#define FD_OUTPUT                           _RA4
#define RFSW                                _RA3
#define DEBUG_LED                           _RB8
#define DEBUG_BTN                           _RB9

#define BOARD_VERSION_2_0                   0
#define BOARD_VERSION_2_1                   1
#define BOARD_VERSION                       BOARD_VERSION_2_1

#if BOARD_VERSION == BOARD_VERSION_2_0
    #define FILTER_CLK                      _RB15
    #define FILTER_LATCH                    _RB13
    #define configSCK2Pin()                 RPOR6bits.RP13R = 0x0B     // SCK2 Output on RP13
    #define setFilterLatchAsOutput()        setOutputPin(TRISB,15)
    #define setFilterClkAsOutput()          setOutputPin(TRISB,13)
#elif BOARD_VERSION == BOARD_VERSION_2_1
    #define FILTER_CLK                      _RB15
    #define FILTER_LATCH                    _RB13
    #define configSCK2Pin()                 RPOR7bits.RP15R = 0x0B     // SCK2 Output on RP15
    #define setFilterLatchAsOutput()        setOutputPin(TRISB,13)
    #define setFilterClkAsOutput()          setOutputPin(TRISB,15)
#endif

#define configU1RXDPin()                    RPINR18bits.U1RXR  = 2     // U1RX Input on RP2
#define configU1CTSPin()                    RPINR18bits.U1CTSR = 5     // U1CTS Input on RP5
#define configU1TXDPin()                    RPOR1bits.RP3R     = 0x03  // U1TX Output on RP3
#define configU1RTSPin()                    RPOR5bits.RP10R    = 0x04  // U1RTS Output on RP10
#define configSDO1Pin(FUNC)                 RPOR5bits.RP11R    = 0x07  // SDO1 Output on RP11
#define configSCK1Pin(FUNC)                 RPOR6bits.RP12R    = 0x08  // SCK1 Output on RP12
#define configSDO2Pin(FUNC)                 RPOR7bits.RP14R    = 0x0A  // SDO2 Output on RP14
#define setOutputPin(PORT,PIN)              PORT &= ~(1 << PIN)
#define setInputPin(PORT,PIN)               PORT |= (1 << PIN)
#define configurePinOpenDrain(PORT,PIN)     PORT |= (1 << PIN)
#define togglePin(PIN)                      PIN = !PIN
#define setPinHigh(PIN)                     PIN = ON
#define setPinLow(PIN)                      PIN = OFF
#define setFilterDataAsOutput()             setOutputPin(TRISB,14)
#define setBackDACSelAsOutput()             setOutputPin(TRISB,6)
#define setFrontDACSelAsOutput()            setOutputPin(TRISB,7)
#define setFreqDividerAsInput()             setInputPin(TRISA,4)
#define setFreqDividerSelAsOutput()         setOutputPin(TRISB,4)
#define setRFSwitchAsOutput()               setOutputPin(TRISA,3)
#define setDebugLEDAsOutput()               setOutputPin(TRISB,8)
#define setDebugBtnAsInput()                setInputPin(TRISB,9)

#define turnDebugLEDOn()                    setPinHigh(DEBUG_LED)
#define turnDebugLEDOff()                   setPinLow(DEBUG_LED)
#define toggleDebugLED()                    togglePin(DEBUG_LED)
#define isBtnPressed()                     (!DEBUG_BTN ? TRUE : FALSE)

/* Function Prototypes */
void initPins(void);                       // Implemented in main.c

#endif

/* End of File */
```

TRAPS.C

```
/*******************************************************************************
 * Filename:    traps.c
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Description: This file houses the Interrupt Service Routines needed to catch
 *              Oscillator Failures, Address Errors, Stack Errors and Math Errors that
 *              would normally cause the PIC to reset.
 *******************************************************************************/
#include "main.h"

/*******************************************************************************
 * Interrupt:   _OscillatorFail
 * Parameters:  void
 * Return:      void
 * Description: Interrupt is triggered when an oscillator failure occurs.  It prints
 *              a notification to the console and infinitely loops.
 *******************************************************************************/
void _ISR __attribute__((auto_psv)) _OscillatorFail(void)
{
    txStrUART1("[[[ Trap -- Oscillator Failed ]]]\r\n");
    while ( TRUE );
}

/*******************************************************************************
 * Interrupt:   _AddressError
 * Parameters:  void
 * Return:      void
 * Description: Interrupt is triggered when an address error occurs.  It prints
 *              a notification to the console and infinitely loops.
 *******************************************************************************/
void _ISR __attribute__((auto_psv)) _AddressError(void)
{
    txStrUART1("[[[ Trap -- Address Error ]]]\r\n");
    while ( TRUE );
}

/*******************************************************************************
 * Interrupt:   _StackError
 * Parameters:  void
 * Return:      void
 * Description: Interrupt is triggered when an stack error occurs.  It prints
 *              a notification to the console and infinitely loops.
 *******************************************************************************/
void _ISR __attribute__((auto_psv)) _StackError(void)
{
    txStrUART1("[[[ Trap -- Stack Error ]]]\r\n");
    while ( TRUE );
}
```

```
/*****************************************************************************
 * Interrupt:   _MathError
 * Parameters:  void
 * Return:      void
 * Description: Interrupt is triggered when an math error occurs.  It prints
 *              a notification to the console and infinitely loops.
 *****************************************************************************/
void _ISR __attribute__((auto_psv)) _MathError(void)
{
    txStrUART1("[[[ Trap -- Math Error ]]]\r\n");
    while ( TRUE );
}

/* End of File */
```

FREQCOUNT.H

```
/*****************************************************************************
 * Filename:    freqcount.h
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Discription: This file houses the preprocessor definitions and function prototypes
 *              needed to count the rising edges of the Frequency Dividers' ouput.
 *****************************************************************************/
#ifndef _FREQCOUNT_H
#define _FREQCOUNT_H

/* Preprocessor Definitions & Macros */
#define TIMER1_CONFIG                       0x0002
#define TIMER2_CONFIG                       0x0000
#define TIMER2_PERIOD                       8312  //8320 <-- calculation yields this value

// Frequency Divider Outputs are Multiplexed
#define selectFrontEndFDOutput()            setPinLow(FD_SEL)
#define selectBackEndFDOutput()             setPinHigh(FD_SEL)
#define readFrontEndFD()                    updateFreqCount(1)
#define readBackEndFD()                     updateFreqCount(0)

#define disableTimer1()                     T1CONbits.TON = 0
#define enableTimer1()                      T1CONbits.TON = 1
#define disableTimer2()                     T2CONbits.TON = 0
#define enableTimer2()                      T2CONbits.TON = 1

/* Function Prototypes */
int updateFreqCount(int);
void _ISR __attribute__((auto_psv)) _T2Interrupt(void);

#endif

/* End of File */
```

FREQCOUNT.C

```
/*****************************************************************************
 * Filename:    freqcount.c
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Description: This file houses the functions needed to count the multiplexed
 *              Frequency Dividers output to determine the frequency of the Front-End
 *              and Back-End.
 *****************************************************************************/
#include "main.h"

/* Global Variables*/
int _FreqCntDone;
int freqCntTmp;

/*****************************************************************************
 * Function:    updateFreqCount
 * Parameters:  int select - 1 for FrontEnd or 0 for BackEnd Frequency Divider Ouput.
 * Return:      int - Selected frequency count.
 * Description: Sets Timer 1 up as a counter using the selected Frequency Divider
 *              output as an external clock and Timer 2 as a gate timer for the
 *              counter.  Once the gate time has elapsed the _T2Interrupt is triggered
 *              stopping Timer 1 from counting.
 *****************************************************************************/
int updateFreqCount(int select)
{
    if ( select )
       selectFrontEndFDOutput();
    else
       selectBackEndFDOutput();

    freqCntTmp = 0;
    _FreqCntDone = FALSE;
    T1CON = TIMER1_CONFIG;        // Counter
    T2CON = TIMER2_CONFIG;        // Gate Timer
    PR2 = TIMER2_PERIOD;
    TMR1 = 0;
    TMR2 = 0;
    enableTimer2();
    enableTimer1();
    while ( !_FreqCntDone );      // Count while flag is not set
    return freqCntTmp;
}

/*****************************************************************************
 * Interrupt:   _T2Interupt
 * Parameters:  void
 * Return:      void
 * Description: Interrupt stores the current count of Timer 1, disables both
 *              timers and sets a flag indicating that counting is done.
 *****************************************************************************/
void _ISR __attribute__((auto_psv)) _T2Interrupt(void)
{
    freqCntTmp  = TMR1;
    disableTimer1();
    disableTimer2();
```

```
    _FreqCntDone = TRUE;
    _T2IF  = 0;
}

/* End of File */
```

AMPDETECT.H

```
/*******************************************************************************
 * Filename:    ampdetect.h
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Discription: This file houses the preprocessor definitions and function prototypes
 *              needed by the Amplitude Detectors and Analog-to-Digital Converters.
 *******************************************************************************/
#ifndef _AMPDETECT_H
#define _AMPDETECT_H

/* Preprocessor Definitions */
#define AD_FRONT_CHANNEL                    0
#define AD_BACK_CHANNEL                     1

#define enableADC()                         AD1CON1bits.ADON = 1
#define disableADC()                        AD1CON1bits.ADON = 0
#define readFrontEndAD()                    readADC(AD_FRONT_CHANNEL)
#define readBackEndAD()                     readADC(AD_BACK_CHANNEL)
#define calcVoltage(val)                    ((double) ((2.5*(val))/1024.0))

/* Function Prototypes */
void initADC(void);
int readADC(int);

#endif

/* End of File */
```

AMPDETECT.C

```
/*******************************************************************************
 * Filename:    ampdetect.c
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Description: This file houses the functions needed to initialize and sample the
 *              Analog-to-Digital Converter (ADC).  The ADC Channels 0-1 are connected
 *              to the Front-End and Back-End Amplitude Detectors respectively.
 *******************************************************************************/
#include "main.h"

/*******************************************************************************
 * Function:    initADC
 * Parameters:  void
 * Return:      void
 * Description: Initializeds the ADC to sample at a rate of 100kHz.
 *******************************************************************************/
void initADC(void)
{
    AD1CON1   = 0x00E0;
    AD1CON2   = 0x0000;
    AD1CON3   = 0x0000;
    AD1CON4   = 0x0000;
    AD1CHS123 = 0x0000;
    AD1CHS0   = 0x0000;
    AD1CSSL   = 0x0000;
    AD1CON3bits.ADCS  = 12;              // Tad = (ADCS + 1)*Tcy = 13/(26MHz/2) = 1us
    AD1CON3bits.SAMC  = 10;              // Tsamp = SAMC*Tad = 10*Tad = 10us (Fs = 100kHz)
    AD1CHS0bits.CH0SA = AD_FRONT_CHANNEL; // AN0
}

/*******************************************************************************
 * Function:    readADC
 * Parameters:  int chan - 0 for the Front-End and 1 for the Back-End AD respectively.
 * Return:      int - 10 bit ADC Reading.
 * Description: Enables the ADC and samples the respective channel.  After the sample
 *              conversion is complete, it returns value and disables the ADC.
 *******************************************************************************/
int readADC(int chan)
{
    enableADC();
    ADC1BUF0 = 0;
    AD1CHS0bits.CH0SA = chan;
    AD1CON1bits.SAMP = 1;
    while ( !AD1CON1bits.DONE );          // Wait while converting
    AD1CON1bits.SAMP = 0;
    disableADC();
    return ADC1BUF0;
}

/* End of File */
```

DAC.H

```
/*******************************************************************************
 * Filename:    dac.h
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Discription: This file houses the preprocessor definitions and function prototypes
 *              needed by the Digital-to-Analog Converters for Analog Tuning.
 *******************************************************************************/
#ifndef _DAC_H
```

```
#define _DAC_H

/* Preprocessor Definitions & Macros */

// DAC Format 15-12: Ctrl, 11-2: Data, 1-2: Reserved
#define formatDACData(ctrl,data)            (((ctrl) << 12) | (((data) & 0x3FF) << 2))

// Load input register A from shift register; DAC registers and outputs unchanged.
#define loadInputRegA(data)                 formatDACData(0x1,(data))

// Load input register B from shift register; DAC registers and outputs unchanged.
#define loadInputRegB(data)                 formatDACData(0x2,(data))

// Load DAC registers A and B from respective input registers;
// DAC outputs A and B updated; Enters normal operation if in shutdown.
#define loadDACRegsABUpdateOutputsAB()      formatDACData(0x8,0x0000)

// Enters shutdown mode.
#define shutdownDAC()                       formatDACData(0xC,0x0000)

// Enters normal operation; DAC outputs reflect exsisting contents of DAC registers.
#define normalDACOperation()                formatDACData(0xD,0x0000)

// Enters shutdown mode; DAC outputs set to high impedance.
#define shutdownDACHighZ()                  formatDACData(0xE ,0x0000)

// Load input registers A and B and DAC registers A and B from shift register;
// DAC outputs A and B updated; Enters normal operation if in shutdown.
#define loadAllRegsUpdateAllOutputs(data)   formatDACData(0xF, (data))

// Set RB6 & RB7 High (Active Low Select) to deselect both DACs
#define deselectBothDACs()                  (PORTB |= 0x00C0)

// Set RB7 Low & RB6 High to select the Front-End DAC
#define selectFrontEndDAC()                 (PORTB = (PORTB & 0xFF3F) | 0x0040)

// Set RB6 Low & RB7 High to select the Back-End DAC
#define selectBackEndDAC()                  (PORTB = (PORTB & 0xFF3F) | 0x0080)

// Program Front-End DAC
#define programFrontEndDAC(data)             programDAC((data),1)

// Proram Back-End DAC
#define programBackEndDAC(data)              programDAC((data),0)

/* Function Prototypes */
void programDAC(int, int);

#endif

/* End of File */
```

DAC.C

```
/*******************************************************************************
 * Filename:    dac.c
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Description: This file houses the function needed to program the DACs used
 *              in the analog tuning of the QE Filter.
 *******************************************************************************/
#include "main.h"

/*******************************************************************************
 * Function:    programDAC
 * Parameters:  int data - The DAC specific CMDs outlined within cmdsm.h.
 *              int select - 1 for FrontEnd or 0 for BackEnd DAC.
 * Return:      void
 * Description: Selects the appropriate DAC and programs the DAC CMD via SPI.
 *******************************************************************************/
void programDAC(int data, int select)
{
    deselectBothDACs();
    enableSPI1();
    if ( select )
        selectFrontEndDAC();
    else
        selectBackEndDAC();
    writeSPI1(data);
    deselectBothDACs();
}

/* End of File */
```

SPI.H

```
/*******************************************************************************
 * Filename:    spi.h
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Discription: This file houses the preprocessor definitions and function prototypes
 *              needed by the SPI to program the QE Filter and DACs.
 *******************************************************************************/
#ifndef _SPI_H
#define _SPI_H

/* Preprocessor Definitions & Macros */
#define clearSPI1Stat()                     SPI1STAT &= 0x8000
#define enableSPI1()                        SPI1STATbits.SPIEN = TRUE
#define disableSPI1()                       SPI1STATbits.SPIEN = FALSE
#define configSPI1()                        SPI1CON1 = 0x05B3

#define clearSPI2Stat()                     SPI2STAT &= 0x8000
#define enableSPI2()                        SPI2STATbits.SPIEN = TRUE
#define disableSPI2()                       SPI2STATbits.SPIEN = FALSE
#define configSPI2()                        SPI2CON1 = 0x05B3 //04B3

/* Function Prototypes */
void initSPI1(void);
void initSPI2(void);
```

```
void writeSPI1(int);
void writeSPI2(int);

#endif

/* End of File */
```

SPI.C

```
/*******************************************************************************
 * Filename:    spi.c
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Description: This file houses the functions needed to program the QE Filter and
 *              DACS via SPI.
 *******************************************************************************/
#include "main.h"

/* Global Variables */
unsigned int _TransmitComplete1, _TransmitComplete2;

/*******************************************************************************
 * Function:    initSPI1
 * Parameters:  void
 * Return:      void
 * Description: Initializes SPI1 for Master Mode transmission.
 *******************************************************************************/
void initSPI1(void)
{
    configSPI1();
    disableSPI1();
    clearSPI1Stat();
    _TransmitComplete1 = FALSE;
}

/*******************************************************************************
 * Function:    initSPI2
 * Parameters:  void
 * Return:      void
 * Description: Initializes SPI1 for Master Mode transmission.
 *******************************************************************************/
void initSPI2(void)
{
    configSPI2();
    disableSPI2();
    clearSPI2Stat();
    _TransmitComplete2 = FALSE;
}

/*******************************************************************************
 * Function:    writeSPI1
 * Parameters:  int data - Data being transmitted to the slave.
 * Return:      void
 * Description: Transmits data to the DACs.
 *******************************************************************************/
void writeSPI1(int data)
{
    _TransmitComplete1 = FALSE;
    clearSPI1Stat();
    SPI1BUF = data;
    while( !_TransmitComplete1 );
}

/*******************************************************************************
 * Function:    writeSPI2
 * Parameters:  int data - Data being transmitted to the slave.
 * Return:      void
 * Description: Transmits data to the QE Filter Serial-to-Parallel Register.
 *******************************************************************************/
void writeSPI2(int data)
{
    _TransmitComplete2 = FALSE;
    clearSPI2Stat();
    SPI2BUF = data;
    while( !_TransmitComplete2 );
}

/*******************************************************************************
 * Interrupt:   _SPI1Interrupt
 * Parameters:  void
 * Return:      void
 * Description: Interrupt triggers when SPI1 transmit is completed.  It sets a
 *              transmit complete flag and returns.
 *******************************************************************************/
void _ISRFAST __attribute__((auto_psv)) _SPI1Interrupt(void)
{
     _TransmitComplete1 = TRUE;       // Set as transfer completed
     _SPI1IF = 0;                     // Clear Interrupt Flag
}

/*******************************************************************************
 * Interrupt:   _SPI2Interrupt
 * Parameters:  void
 * Return:      void
 * Description: Interrupt triggers when SPI2 transmit is completed.  It sets a
 *              transmit complete flag and returns.
 *******************************************************************************/
void _ISRFAST __attribute__((auto_psv)) _SPI2Interrupt(void)
{
     _TransmitComplete2 = TRUE;       // Set as transfer completed
     _SPI2IF = 0;                     // Clear Interrupt Flag
}

/* End of File */
```

STRFUNC.H

```
/*****************************************************************************
 * Filename:    strfunc.h
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Discription: This file houses the preprocessor definitions and function prototypes
 *              needed to write text to the console (Replacement for sprintf).
 *****************************************************************************/
#ifndef _STRFUNC_H
#define _STRFUNC_H

int numDigits16Bit(int);
char * digitSplit16Bit(int);
void strPopulate16Bit(char *, int, char, int);

#endif

/* End of File */
```

STRFUNC.C

```
/*****************************************************************************
 * Filename:    strfunc.c
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Description: This file houses the functions needed as an efficient alternative to
 *              the sprintf function included in stdio.h.
 *****************************************************************************/
#include "main.h"

/*****************************************************************************
 * Function:    numDigits16Bit
 * Parameters:  int num - The number that the digits will be counted from.
 * Return:      int - The number of digits contained in num.
 * Description: Returns the number of digits contained in the input integer num.
 *              It will return 1 if the number is 0.  It works for both negative and
 *              positive inputs.
 *****************************************************************************/
int numDigits16Bit(int num)
{
    num = num > 0 ? num : -1*num;
    if ( (num - 10000) >= 0 )
        return 5;
    if ( (num - 1000) >= 0 )
        return 4;
    if ( (num - 100) >= 0 )
        return 3;
    if ( (num - 10) >= 0 )
        return 2;
    return 1;
}

/*****************************************************************************
 * Function:    digitSplit16Bit
 * Parameters:  int num - The number that its digits will be split from.
 * Return:      char * - A character array containing each digit.
 * Description: Splits the input into digits and returns them in an array.
 *****************************************************************************/
char * digitSplit16Bit(int num)
{
    char digits[5] = {0};
    char *dig = digits;

    num = num < 0 ? -1*num : num;

    while ( num >= 10000 )
    {
        if ( (num -= 10000) >= 0 )
            digits[4]++;
    }
    while ( num >= 1000 )
    {
        if ( (num -= 1000) >= 0 )
            digits[3]++;
    }
    while ( num >= 100 )
    {
        if ( (num -= 100) >= 0 )
            digits[2]++;
    }
    while ( num >= 10 )
    {
        if ( (num -= 10) >= 0 )
            digits[1]++;
    }
    while ( num >= 1 )
    {
        if ( (num -= 1) >= 0 )
            digits[0]++;
    }

    digits[4] = HEX2ASCII(digits[4]);
    digits[3] = HEX2ASCII(digits[3]);
    digits[2] = HEX2ASCII(digits[2]);
    digits[1] = HEX2ASCII(digits[1]);
    digits[0] = HEX2ASCII(digits[0]);

    return dig;
}
```

```
/*****************************************************************************
 * Function:    strPopulate16Bit
 * Parameters:  char * str - the preallocated string of any length.
 *              int num - the number replacing the markers in the input string.
 *              char mark - the character that will be replaced by the digits of num.
 *              int numDigits - number of digits being displayed (same as the number
 *                              of markers).  If the number of digits in num is less
 *                              than the numDigits specififed, it will act like zero
 *                              padding.  numDigits should never be less than the
 *                              number of digits in num or it will give undersired
 *                              results.
 * Return:      void
 * Description: Replacement for the sprintf/printf function.  It replaces the markers
 *              of the input string with the digits of the input num and prints it to
 *              the console via UART.
 * Example:     strCenterFreq[] = "Center Frequency ###.# MHz";
 *              strPopulate16Bit(strCenterFreq,4326,'#',4);
 *              Output: Center Frequency 432.6 MHz
 *****************************************************************************/
void strPopulate16Bit(char * str, int num, char mark, int numDigits)
{
    int charIdx = 0, storeIdx = 0, N = numDigits;
    int store[5]  = {0};
    char digit[5] = {0};
    char *s = str;
    char *x = digitSplit16Bit(num);

    if ( numDigits < 0 )
        numDigits = numDigits16Bit(num);

    digit[4] = x[4];
    digit[3] = x[3];
    digit[2] = x[2];
    digit[1] = x[1];
    digit[0] = x[0];

    while ( *s++ != '\0' )
    {
        charIdx++;
        if ( *s == mark )
        {
            *s = digit[--numDigits];
            store[storeIdx++] = charIdx;
        }
    }
    txStrUART1(str);
    for ( charIdx = 0; charIdx < N; charIdx++ )
        str[store[charIdx]] = mark;
}

/* End of File */
```

UART1.H

```
/*****************************************************************************
 * Filename:    uart1.h
 * Date:        June 2010
 * Compiler:    C30
 * Modified by: Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Discription: This file houses the preprocessor definitions and function prototypes
 *              needed by the UART1 to communicate with the CP2103 USB-to-UART Bridge
 *              and PC.
 *****************************************************************************/
#ifndef _UART1_H
#define _UART1_H

/* Preprocessor Definitions & Macros */
#define UART_BUFFER_SIZE                    2048 //1024
#define BAUDRATE                            84 // (Fosc/2)/(2*9600) - 1 = 83.64 --> 84

/* Function Prototypes */
void initUART1(unsigned int, char, char);
unsigned char txByteUART1(unsigned char);
unsigned char txBytesUART1(unsigned char *, unsigned char);
unsigned char txStrUART1(const char *);
unsigned char txLengthUART1(void);
unsigned char rxByteUART1(unsigned char *);
unsigned char rxBytesUART1(unsigned char *, unsigned char);
unsigned char rxLengthUART1(void);
void _ISR __attribute__((auto_psv)) _U1RXInterrupt(void);
void _ISR __attribute__((auto_psv)) _U1TXInterrupt(void);

#endif

/* End of File */
```

UART1.C

```
/*****************************************************************************
 * Filename:    uart1.c
 * Date:        June 2010
 * Compiler:    C30
 * Modified by: Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Description: This file houses the functions needed by the UART1 to communicate with
 *              the CP2103 USB-to-UART Bridge and PC.
 *****************************************************************************/
#include "main.h"

/* Global Variables */
struct uart_buffer {
            unsigned char bytes[UART_BUFFER_SIZE];
            unsigned int length;
            unsigned int start;
};

volatile struct uart_buffer buf_rx;
volatile struct uart_buffer buf_tx;
```

```
/*****************************************************************************
 * Interrupt:   _U1RXInterrupt
 * Parameters:  void
 * Return:      void
 *****************************************************************************/
void _ISR __attribute__((auto_psv)) _U1RXInterrupt(void)
{
	unsigned char byte;

	IFS0bits.U1RXIF = 0;
	while (U1STAbits.URXDA)
	{
		byte = U1RXREG;
		if (buf_rx.length < UART_BUFFER_SIZE)
		{
			if (buf_rx.length + buf_rx.start >= UART_BUFFER_SIZE)
				buf_rx.bytes[(buf_rx.length + buf_rx.start) - UART_BUFFER_SIZE] = byte;
			else
				buf_rx.bytes[buf_rx.length + buf_rx.start] = byte;
			buf_rx.length++;
		}
	}
}

/*****************************************************************************
 * Interrupt:   _U1TXInterrupt
 * Parameters:  void
 * Return:      void
 *****************************************************************************/
void _ISR __attribute__((auto_psv)) _U1TXInterrupt(void)
{
	IFS0bits.U1TXIF = 0;
	while (!U1STAbits.UTXBF && buf_tx.length > 0)
	{
		U1TXREG = buf_tx.bytes[buf_tx.start];
		buf_tx.start++;
		buf_tx.length--;

		if (buf_tx.start >= UART_BUFFER_SIZE)
			buf_tx.start = 0;
	}
}

/*****************************************************************************
 * Function:    txBytesUART1
 * Parameters:  unsigned char * bytes
 *              unsigned char length
 * Return:      unsigned char
 *****************************************************************************/
unsigned char txBytesUART1(unsigned char * bytes, unsigned char length)
{
	unsigned char i;
	if (buf_tx.length + length >= UART_BUFFER_SIZE)
		return 0xFF;

	for(i = 0; i < length; ++i)
	{
		if (txByteUART1(bytes[i]))
			return (i + 1);
	}
	return 0;
}

/*****************************************************************************
 * Function:  txStrUART1
 * Parameters:  const char * str
 * Return:      unsigned char
 *****************************************************************************/
unsigned char txStrUART1(const char * str)
{
	unsigned char i = 0;
	while (str[i] != 0)
	{
		if (txByteUART1(str[i]))
			return (i + 1);
		i++;
	}
	return 0;
}

/*****************************************************************************
 * Function:    txByteUART1
 * Parameters:  unsigned char byte
 * Return:      unsigned char
 *****************************************************************************/
unsigned char txByteUART1(unsigned char byte)
{
	if (buf_tx.length >= UART_BUFFER_SIZE)
		return 1;

	if (buf_tx.length + buf_tx.start >= UART_BUFFER_SIZE)
		buf_tx.bytes[(buf_tx.length + buf_tx.start) - UART_BUFFER_SIZE] = byte;
	else
		buf_tx.bytes[buf_tx.length + buf_tx.start] = byte;
	buf_tx.length++;

	if (U1STAbits.TRMT)
	{
		U1TXREG = buf_tx.bytes[buf_tx.start];
		buf_tx.length--;
		buf_tx.start++;
		if (buf_tx.start >= UART_BUFFER_SIZE)
			buf_tx.start = 0;
	}
	return 0;
}

/*****************************************************************************
 * Function:    rxByteUART1
 * Parameters:  unsigned char * byte_ptr
 * Return:      unsigned char
 *****************************************************************************/
unsigned char rxByteUART1(unsigned char * byte_ptr)
{
	if (buf_rx.length == 0)
		return 1;

	(*byte_ptr) = buf_rx.bytes[buf_rx.start];
	buf_rx.start++;
	buf_rx.length--;
	if (buf_rx.start >= UART_BUFFER_SIZE)
		buf_rx.start = 0;
```

```
            return 0;
}

/*****************************************************************************
 * Function:    rxBytesUART1
 * Parameters:  unsigned char * bytes
 *              unsigned char length
 * Return:      unsigned char
 *****************************************************************************/
unsigned char rxBytesUART1(unsigned char * bytes, unsigned char length)
{
            unsigned char i, byte;
            if (buf_rx.length < length)
            {
                        return 0xFF;
            }
            for(i = 0; i < length; ++i)
            {
                        if (rxByteUART1(&byte))
                                    return (i + 1);
                        bytes[i] = byte;
            }
            return 0;
}

/*****************************************************************************
 * Function:    txLengthUART1
 * Parameters:  void
 * Return:      unsigned char
 *****************************************************************************/
unsigned char txLengthUART1(void)
{
            return buf_tx.length;
}

unsigned char rxLengthUART1(void)
{
            return buf_rx.length;
}

/*****************************************************************************
 * Function:    initUART1
 * Parameters:  unsigned int baud
 *              unsigned char rx_priority
 *              unsigned char tx_priority
 * Return:      void
 *****************************************************************************/
void initUART1(unsigned int baud, char rx_priority, char tx_priority)
{
            buf_rx.start       = 0;
            buf_rx.length      = 0;
            buf_tx.start       = 0;
            buf_tx.length      = 0;
            IPC2bits.U1RXIP    = rx_priority;
            IPC3bits.U1TXIP    = tx_priority;
            IEC0bits.U1RXIE    = 1;
            IEC0bits.U1TXIE    = 1;
            U1MODEbits.UEN     = 2;
            U1STA              = 0x0400;
            U1STAbits.UTXISEL0 = 1;
            U1BRG              = baud;
            U1MODEbits.BRGH    = 0;
            U1MODEbits.UARTEN  = 1;
            U1STAbits.UTXEN    = 1;
}

/* End of File */
```

APPENDIX H – SINGLE-POLE TUNING ALGORITHM SOURCE CODE

MAIN.C

```
/*****************************************************************************
 * Filename:    main.c
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Description: This file houses the main loop and initialization functions.  Once
 *              initialized, the system waits for a CMD string to be received, from
 *              the QEFilter PC Application, that contains the algorithm's parameters.
 *              Once the command string has been analzed, the algorithm is run.  It
 *              then waits for the next CMD string.
 *****************************************************************************/
#include "main.h"

/* Device Configuration Bits */
// Boot segment may be written, no boot segement, no boot RAM
_FBS(RBS_NO_RAM & BSS_NO_FLASH  & BWRP_WRPROTECT_OFF);
// No Secure Ram, no secure segment, write protection disabled
_FSS(RSS_NO_RAM & SSS_NO_FLASH & SWRP_WRPROTECT_OFF);
// General Segment and Code protect off, General Segment may be written
_FGS(GSS_OFF & GCP_OFF & GWRP_OFF);
// Primary Oscillator (XT, HS, EC), start-up with user-selected oscillator
_FOSCSEL(FNOSC_PRI & IESO_OFF);
// Clock switching and monitoring disabled, one re-configuration for remappable I/O,
// Primary Oscillator is External Clock
_FOSC(FCKSM_CSDCMD & IOL1WAY_ON & OSCIOFNC_ON & POSCMD_EC);
// Watchdog Timer off
_FWDT(FWDTEN_OFF & WINDIS_OFF & WDTPRE_PR32 & WDTPOST_PS1);
// I2C mapped to SDA1/SCL1, 128ms Power on Reset
_FPOR(ALTI2C_OFF & FPWRT_PWR128);
// JTAG disabled, ICD communicate on PGC1/EMUC1 and PGD1/EMUD1
_FICD(JTAGEN_OFF & ICS_PGD1);

/* Global Variables */
const char * console_str_boot  = "dsPIC33fj64gp802 Acknowledges You\r\n";
const char * algorithm_version = "Single-Pole Tuning Algorithm\r\n";
const char * console_str_sep   = \
"--------------------------------------------------------------------------------\r\n";
int _UpdateFilter;
extern int _PrintFilterSettings;
extern int CenterFreq, FreqTol;
extern int FrontEndADThresh1, FrontEndADThresh2, FrontEndQOffset, FrontEndQBackOff, FrontEndFOffset;
extern int BackEndADThresh1, BackEndADThresh2, BackEndQOffset, BackEndQBackOff, BackEndFOffset;
extern int CouplingUpper, CouplingLower, CouplingUFLB, CouplingLFUB;

/*****************************************************************************
 * Function:    main
 * Parameters:  void
 * Return:      int - never reaches return value
 * Description: Initializes the system and filter variables.  The filter is then
 *              programmed to have all options at their minima or disabled.  It then
 *              waits reads the UART and process its data appropriately.  If the
 *              _UpdateFilte flag is set it will run the tuning Algorithm.
 *****************************************************************************/
int main(void)
{
    init();
    initFilter();
    programFilter();

    _CN21IE = 1;                        // Enable Change Notification Interrupt on RB9
          while ( TRUE )
    {
        processRxData();
                    if ( _UpdateFilter )
        {
            _UpdateFilter = FALSE;
                        algorithm();
        }
    }
}

/*****************************************************************************
 * Function:    init
 * Parameters:  void
 * Return:      void
 * Description: Initializeds the modules and sets up the interrupts.
 *****************************************************************************/
void init(void)
{
    RCONbits.SWDTEN = 0;                // Software Disable of the Watchdog Timer

    _SPI1IE = 1;                        // Enable SPI1 Transfer Complete Interrupt
    _SPI1IF = 0;                        // Clear SPI1 Event Interrupt Flag
    _SPI2IE = 1;                        // Enable SPI1 Transfer Complete Interrupt
    _SPI2IF = 0;                        // Clear SPI1 Event Interrupt Flag
    _T2IE   = 1;                        // Enable Timer2 Overflow Flag
    _T2IF   = 0;                        // Clear Timer2 Overflow Flag

    initPins();
    initSPI1();
    initSPI2();
    initADC();
    initUART1(BAUDRATE, 0x06, 0x05);    // Configure UART Peripherial
    printBootStr();
    printSeperator();
    printAlgorithmVer();
    printSeperator();
}
```

```
/*****************************************************************************
 * Function:    initPins
 * Parameters:  void
 * Return:      void
 * Description: Initializeds the systems pins (see pinconfig.h).
 *****************************************************************************/
void initPins(void)
{
    /* Configure Analog Pins */
    ADPCFG = 0xFFFC;                        // All Digital Pins Except AN0 & AN1

    /* UART1 Configuration */
    configU1RXDPin();                      // Setup U1RX Reprogrammable-Pin Register
    configU1CTSPin();                      // Setup U1CTS Reprogrammable-Pin Register
    configU1TXDPin();                      // Setup U1TX Reprogrammable-Pin Register
    configU1RTSPin();                      // Setup U1RTS Reprogrammable-Pin Register

    /* SPI2 Configuration */
    configSDO2Pin();                        // Setup SDO2 Reprogrammable-Pin Register
    configSCK2Pin();                        // Setup SCK2 Reprogrammable-Pin Register
    setFilterDataAsOutput();
    setFilterClkAsOutput();
    setFilterLatchAsOutput();
    setPinLow(FILTER_CLK);                  // Set FILTER_CLK Low
    setPinLow(FILTER_DATA);                 // Set FILTER_DATA Low
    setPinLow(FILTER_LATCH);                // Set FILTER_LATCH Low

    /* SPI1 Configuration */
    configSDO1Pin();                        // Setup SDO1 Reprogrammable-Pin Register
    configSCK1Pin();                        // Setup SCK1 Reprogrammable-Pin Register
    setBackDACSelAsOutput();                // Set DAC_BACK_SEL Low
    setFrontDACSelAsOutput();               // Set DAC_FRONT_SEL Low
    setPinLow(DAC_DATA);                    // Set DAC_DATA Low
    setPinLow(DAC_CLK);                     // Set DAC_CLK Low
    deselectBothDACs();                     // Set DAC_BACK_SEL & DAC_FRONT_SEL High

    /* Frequency Divider Configuration */
    setFreqDividerAsInput();                // Set FD_OUTPUT as Digital Input
    setFreqDividerSelAsOutput();            // Set FD_SEL as Digital Output
    setPinLow(FD_SEL);                      // Set FD_SEL Low

    /* RF Switch Configuration */
    setRFSwitchAsOutput();                  // Set RF Switch as Digital Ouput

    /* Debug LED Configuration */
    setDebugLEDAsOutput();                  // Set Debug LED as Digital Output
    turnDebugLEDOff();                      // Turn Debug LED Off

    /* Debug BTN Configuration */
    setDebugBtnAsInput();
    _CNIE = 1;                              // Enable Change Notification Interrupts
    _CNIP = 2;                              // Set Interrupt priority
    _CN21IE = 0;                            // Disable Change Notification Interrupt on RB9
    _CNIF = 0;                              // Clear Change Notifitication Interupt Flag
}

/* End of File */

/*****************************************************************************
 * Interrupt:   _CNInterrupt
 * Parameters:  void
 * Return:      void
 * Description: Interrupt is triggered when the debug button is clicked. The debug
 *              button runs the algorithm.
 *****************************************************************************/
void _ISR __attribute__((auto_psv)) _CNInterrupt(void)
{
    int i;
    for (i = 0; i < 1000; i++); // Debounce

    if ( isBtnPressed() )
    {
        /* Default Algorithm Settings for 10 MHz Bandwidth centered at 450 MHz */
        CenterFreq = 4500;                          // 450.0 MHz
        FreqTol = 2;                                // 0.2 MHz
        FrontEndADThresh1 = FrontEndADThresh2 = 10;
        FrontEndQOffset = 2;
        FrontEndQBackOff = 8; // Sets Bandwidth of roughly 10 MHz
        FrontEndFOffset = 3;
        BackEndADThresh1 = BackEndADThresh2 = 0;
        BackEndQOffset = BackEndQBackOff = BackEndFOffset = 0;
        CouplingUpper = CouplingLower = CouplingUFLB = CouplingLFUB = 0;

        _PrintFilterSettings = TRUE;
        algorithm();
    }
    _CNIF = 0; // Clear Change Notification Interrupt Flag
}

/* End of File */
```

QEFILTER.H

```
/*****************************************************************************
 * Filename:    qefilter.h
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Discription: This file houses the preprocessor definitions and function prototypes
 *              needed by the QE Filter Tuning Algorithm.
 *****************************************************************************/
#ifndef _QEFILTER_H
#define _QEFILTER_H

/* Preprocessor Definitions & Macros */
#define FTUNE_DIG_MIN                       165 // Limit the Frequency Range for Reliable Frequency Divider Ouputs
#define FTUNE_DIG_MAX                       255
#define QTUNE_DIG_MIN                       0
#define QTUNE_DIG_MAX                       63
#define COUPLING_MIN                        0
#define COUPLING_MAX                        31
#define ANALOG_MIN                          0
#define ANALOG_MAX                          1023
#define MAX_FREQTUNE_ITTS                   100
#define MAX_CRITOSC_ITTS                    64
```

```
#define isFrontEndEnabled()               (!FENDCON.en ? 1 : 0)
#define enableFrontEnd()                  FENDCON.en    = 0 // Active-Low Enable
#define disableFrontEnd()                 FENDCON.en    = 1
#define isFrontEndADEnabled()             (!FENDCON.ADen ? 1 : 0)
#define enableFrontEndAD()                FENDCON.ADen  = 0 // Active-Low Enable
#define disableFrontEndAD()               FENDCON.ADen  = 1
#define isFrontEndFDEnabled()             (!CAPCON1.FDFen ? 1 : 0)
#define enableFrontEndFD()                CAPCON1.FDFen = 0 // Active-Low Enable
#define disableFrontEndFD()               CAPCON1.FDFen = 1
#define getFrontEndDigitalFTune()         FENDCON.fTune
#define setFrontEndDigitalFTune(val)      FENDCON.fTune = ((val) > FTUNE_DIG_MAX ? FTUNE_DIG_MAX : ((val) < FTUNE_DIG_MIN ? FTUNE_DIG_MIN : (val)))
#define incFrontEndDigitalFTune()         setFrontEndDigitalFTune(FENDCON.fTune + 1)
#define decFrontEndDigitalFTune()         setFrontEndDigitalFTune(FENDCON.fTune - 1)
#define getFrontEndDigitalQTune()         FENDCON.qTune
#define setFrontEndDigitalQTune(val)      FENDCON.qTune = ((val) > QTUNE_DIG_MAX ? QTUNE_DIG_MAX : ((val) < QTUNE_DIG_MIN ? QTUNE_DIG_MIN : (val)))
#define incFrontEndDigitalQTune()         setFrontEndDigitalQTune(FENDCON.qTune + 1)
#define decFrontEndDigitalQTune()         setFrontEndDigitalQTune(FENDCON.qTune - 1)
#define isBackEndEnabled()                (!BENDCON.en ? 1 : 0)
#define enableBackEnd()                   BENDCON.en = 0 // Active-Low Enable
#define disableBackEnd()                  BENDCON.en = 1
#define isBackEndADEnabled()              (!BENDCON.ADen ? 1 : 0)
#define enableBackEndAD()                 BENDCON.ADen = 0 // Active-Low Enable
#define disableBackEndAD()                BENDCON.ADen = 1
#define isBackEndFDEnabled()              (!CAPCON2.FDBen ? 1 : 0)
#define enableBackEndFD()                 CAPCON2.FDBen = 0 // Active-Low Enable
#define disableBackEndFD()                CAPCON2.FDBen = 1
#define getBackEndDigitalFTune()          BENDCON.fTune
#define setBackEndDigitalFTune(val)       BENDCON.fTune = ((val) > FTUNE_DIG_MAX ? FTUNE_DIG_MAX : ((val) < FTUNE_DIG_MIN ? FTUNE_DIG_MIN : (val)))
#define incBackEndDigitalFTune()          setBackEndDigitalFTune(BENDCON.fTune + 1)
#define decBackEndDigitalFTune()          setBackEndDigitalFTune(BENDCON.fTune - 1)
#define getBackEndDigitalQTune()          BENDCON.qTune
#define setBackEndDigitalQTune(val)       BENDCON.qTune = ((val) > QTUNE_DIG_MAX ? QTUNE_DIG_MAX : ((val) < QTUNE_DIG_MIN ? QTUNE_DIG_MIN : (val)))
#define incBackEndDigitalQTune()          setBackEndDigitalQTune(BENDCON.qTune + 1)
#define decBackEndDigitalQTune()          setBackEndDigitalQTune(BENDCON.qTune - 1)
#define getCouplingUpper()                CAPCON1.upper
#define setCouplingUpper(val)             CAPCON1.upper = ((val) > COUPLING_MAX ? COUPLING_MAX : ((val) < COUPLING_MIN ? COUPLING_MIN : (val)))
#define getCouplingLower()                CAPCON1.lower
#define setCouplingLower(val)             CAPCON1.lower = ((val) > COUPLING_MAX ? COUPLING_MAX : ((val) < COUPLING_MIN ? COUPLING_MIN : (val)))
#define getCouplingUFLB()                 CAPCON2.UFLB
#define setCouplingUFLB(val)              CAPCON2.UFLB  = ((val) > COUPLING_MAX ? COUPLING_MAX : ((val) < COUPLING_MIN ? COUPLING_MIN : (val)))
#define getCouplingLFUB()                 CAPCON2.LFUB
#define setCouplingLFUB(val)              CAPCON2.LFUB  = ((val) > COUPLING_MAX ? COUPLING_MAX : ((val) < COUPLING_MIN ? COUPLING_MIN : (val)))
#define getFrontEndAnalogFTune()          ANALOG.FANAF
#define setFrontEndAnalogFTune(val)       ANALOG.FANAF = ((val) > ANALOG_MAX ? ANALOG_MAX : ((val) < ANALOG_MIN ? ANALOG_MIN : (val)))
#define getFrontEndAnalogQTune()          ANALOG.FANAQ
#define setFrontEndAnalogQTune(val)       ANALOG.FANAQ = ((val) > ANALOG_MAX ? ANALOG_MAX : ((val) < ANALOG_MIN ? ANALOG_MIN : (val)))
#define getBackEndAnalogFTune()           ANALOG.BANAF
#define setBackEndAnalogFTune(val)        ANALOG.BANAF = ((val) > ANALOG_MAX ? ANALOG_MAX : ((val) < ANALOG_MIN ? ANALOG_MIN : (val)))
#define getBackEndAnalogQTune()           ANALOG.BANAQ
#define setBackEndAnalogQTune(val)        ANALOG.BANAQ = ((val) > ANALOG_MAX ? ANALOG_MAX : ((val) < ANALOG_MIN ? ANALOG_MIN : (val)))
#define turnRFSwitchOn()                  setPinLow(RFSW);  DEBUG.RFOn = 0
#define turnRFSwitchOff()                 setPinHigh(RFSW); DEBUG.RFOn = 1
#define isRFSwitchOn()                    !DEBUG.RFOn
#define printFrontEndStatus()             (isFrontEndEnabled() ? txStrUART1("Front-End Enabled\r\n") : txStrUART1("Front-End Disabled\r\n"))
#define printFrontEndADStatus()           (isFrontEndADEnabled() ? txStrUART1("Front-End AD Enabled\r\n") : txStrUART1("Front-End AD Disabled\r\n"))
#define printFrontEndFDStatus()           (isFrontEndFDEnabled() ? txStrUART1("Front-End FD Enabled\r\n") : txStrUART1("Front-End FD Disabled\r\n"))
#define printFrontEndAD()                 strPopulate16Bit(strFrontEndAD,FrontEndAD,'#',4)
#define printFrontEndNonOsc()             strPopulate16Bit(strFrontEndNonOsc,FrontEndNonOsc,'#',4)
#define printFrontEndFCnt()               strPopulate16Bit(strFrontEndFCnt,FrontEndFCnt,'#',4)
#define printFrontEndDigitalQTune()       strPopulate16Bit(strFrontEndDigitalQTune,getFrontEndDigitalQTune(),'#',2)
#define printFrontEndAnalogQTune()        strPopulate16Bit(strFrontEndAnalogQTune,getFrontEndAnalogQTune(),'#',4)
#define printFrontEndDigitalFTune()       strPopulate16Bit(strFrontEndDigitalFTune,getFrontEndDigitalFTune(),'#',3)
#define printFrontEndAnalogFTune()        strPopulate16Bit(strFrontEndAnalogFTune,getFrontEndAnalogFTune(),'#',4)
#define printBackEndStatus()              (isBackEndEnabled() ? txStrUART1("Back-End Enabled\r\n") : txStrUART1("Back-End Disabled\r\n"))
```

```
#define printBackEndADStatus()             (isBackEndADEnabled() ? txStrUART1("Back-End AD Enabled\r\n") : txStrUART1("Back-End AD Disabled\r\n"))
#define printBackEndFDStatus()             (isBackEndFDEnabled() ? txStrUART1("Back-End FD Enabled\r\n") : txStrUART1("Back-End FD Disabled\r\n"))
#define printBackEndAD()                    strPopulate16Bit(strBackEndAD,BackEndAD,'#',4)
#define printBackEndNonOsc()                strPopulate16Bit(strBackEndNonOsc,BackEndNonOsc,'#',4)
#define printBackEndFCnt()                  strPopulate16Bit(strBackEndFCnt,BackEndFCnt,'#',4)
#define printBackEndDigitalQTune()          strPopulate16Bit(strBackEndDigitalQTune,getBackEndDigitalQTune(),'#',2)
#define printBackEndAnalogQTune()           strPopulate16Bit(strBackEndAnalogQTune,getBackEndAnalogQTune(),'#',4)
#define printBackEndDigitalFTune()          strPopulate16Bit(strBackEndDigitalFTune,getBackEndDigitalFTune(),'#',3)
#define printBackEndAnalogFTune()           strPopulate16Bit(strBackEndAnalogFTune,getBackEndAnalogFTune(),'#',4)
#define printCouplingUpper()                strPopulate16Bit(strCouplingUpper,CouplingUpper,'#',2)
#define printCouplingLower()                strPopulate16Bit(strCouplingLower,CouplingLower,'#',2)
#define printCouplingUFLB()                 strPopulate16Bit(strCouplingUFLB,CouplingUFLB,'#',2)
#define printCouplingLFUB()                 strPopulate16Bit(strCouplingLFUB,CouplingLFUB,'#',2)
#define printRFSwitchStatus()               (isRFSwitchOn() ? txStrUART1("RF Switch On\r\n") : txStrUART1("RF Switch Off\r\n"))
#define printCenterFreq()                   strPopulate16Bit(strCenterFreq,CenterFreq,'#',4)
#define printFreqTol()                      strPopulate16Bit(strFreqTol,FreqTol,'#',3)
#define printFrontEndADThresh1()            strPopulate16Bit(strFrontEndADThresh1,FrontEndADThresh1,'#',3)
#define printFrontEndADThresh2()            strPopulate16Bit(strFrontEndADThresh2,FrontEndADThresh2,'#',3)
#define printFrontEndQOffset()              strPopulate16Bit(strFrontEndQOffset,FrontEndQOffset,'#',2)
#define printFrontEndQBackOff()             strPopulate16Bit(strFrontEndQBackOff,FrontEndQBackOff,'#',2)
#define printFrontEndFOffset()              strPopulate16Bit(strFrontEndFOffset,FrontEndFOffset,'#',2)
#define printBackEndADThresh1()             strPopulate16Bit(strBackEndADThresh1,BackEndADThresh1,'#',3)
#define printBackEndADThresh2()             strPopulate16Bit(strBackEndADThresh2,BackEndADThresh2,'#',3)
#define printBackEndQOffset()               strPopulate16Bit(strBackEndQOffset,BackEndQOffset,'#',2)
#define printBackEndQBackOff()              strPopulate16Bit(strBackEndQBackOff,BackEndQBackOff,'#',2)
#define printBackEndFOffset()               strPopulate16Bit(strBackEndFOffset,BackEndFOffset,'#',2)

/* Function Prototypes */
void initFilter();
void printFilterOptions(void);
void printAlgorithmOptions(void);
void updateFilterData(void);
void programFilter(void);
void prgmDelay(void);
void updateAnalogTuning(void);
void algorithm(void);
void findCriticalOsc(void);

#endif

/* End of File */
```

QEFILTER.C

```
/*******************************************************************************
 * Filename:    qefilter.c
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Single-Pole Q-Enhanced Filter Tuning Algorithm
 * Description: This file houses the functions needed to implement the QE Filter
 *              tuning algorithm.
 ******************************************************************************/
#include "main.h"

/* Global Variables */
char strCenterFreq[]           = "Center Frequency: ###.# MHz\r\n";
char strFreqTol[]              = "Frequency Tolerance: ##.# MHz\r\n";
char strFrontEndADThresh1[]    = "Front-End AD Threshold 1: ###\r\n";
char strFrontEndADThresh2[]    = "Front-End AD Threshold 2: ###\r\n";
char strFrontEndQOffset[]      = "Front-End Q-Offset: ##\r\n";
char strFrontEndQBackOff[]     = "Front-End Q-BackOff: ##\r\n";
char strFrontEndFOffset[]      = "Front-End F-Offset: ##\r\n";
char strBackEndADThresh1[]     = "Back-End AD Threshold 1: ###\r\n";
char strBackEndADThresh2[]     = "Back-End AD Threshold 2: ###\r\n";
char strBackEndQOffset[]       = "Back-End Q-Offset: ##\r\n";
char strBackEndQBackOff[]      = "Back-End Q-BackOff: ##\r\n";
char strBackEndFOffset[]       = "Back-End F-Offset: ##\r\n";
char strCouplingUpper[]        = "Coupling Upper: ##\r\n";
char strCouplingLower[]        = "Coupling Lower: ##\r\n";
char strCouplingUFLB[]         = "Coupling UFLB: ##\r\n";
char strCouplingLFUB[]         = "Coupling LFUB: ##\r\n";
char strFrontEndAD[]           = "Front-End Amp Detector: ####\r\n";
char strFrontEndNonOsc[]       = "Front-End Non-Osc: ####\r\n";
char strFrontEndFCnt[]         = "Front-End Freq Count: ###.# MHz\r\n";
char strFrontEndDigitalQTune[] = "Front-End Digital Q-Tune: ##\r\n";
char strFrontEndAnalogQTune[]  = "Front-End Analog Q-Tune: ####\r\n";
char strFrontEndDigitalFTune[] = "Front-End Digital F-Tune: ###\r\n";
char strFrontEndAnalogFTune[]  = "Front-End Analog F-Tune: ####\r\n";
char strBackEndAD[]            = "Back-End Amp Detector: ####\r\n";
char strBackEndNonOsc[]        = "Back-End Non-Osc: ####\r\n";
char strBackEndFCnt[]          = "Back-End Freq Count: ###.# MHz\r\n";
char strBackEndDigitalQTune[]  = "Back-End Digital Q-Tune: ##\r\n";
char strBackEndAnalogQTune[]   = "Back-End Analog Q-Tune: ####\r\n";
char strBackEndDigitalFTune[]  = "Back-End Digital F-Tune: ###\r\n";
char strBackEndAnalogFTune[]   = "Back-End Analog F-Tune: ####\r\n";

int FrontEndAD, BackEndAD, FrontEndFCnt, BackEndFCnt, FrontEndNonOsc, BackEndNonOsc;
int PrevFrontEndDigitalQTune, PrevBackEndDigitalQTune;
int filterData[4];
int CenterFreq, FreqTol;
```

```
int FrontEndADThresh1, FrontEndADThresh2, FrontEndQOffset, FrontEndQBackOff, FrontEndFOffset;
int BackEndADThresh1, BackEndADThresh2, BackEndQOffset, BackEndQBackOff, BackEndFOffset;
int CouplingUpper, CouplingLower, CouplingUFLB, CouplingLFUB;

struct {
    unsigned int en:1;
    unsigned int fTune:8;
    unsigned int qTune:6;
    unsigned int ADen:1;
} FENDCON;

struct {
    unsigned int en:1;
    unsigned int fTune:8;
    unsigned int qTune:6;
    unsigned int ADen:1;
} BENDCON;

struct {
    unsigned int upper:5;
    unsigned int lower:5;
    unsigned int FDFen:1;
} CAPCON1;

struct {
    unsigned int UFLB:5;
    unsigned int LFUB:5;
    unsigned int FDBen:1;
} CAPCON2;

struct {
    unsigned int FANAF:10;
    unsigned int FANAQ:10;
    unsigned int BANAF:10;
    unsigned int BANAQ:10;
} ANALOG;

struct {
    unsigned int ADFen:1;
    unsigned int FDFen:1;
    unsigned int ADBen:1;
    unsigned int FDBen:1;
    unsigned int RFOn:1;
} DEBUG;

extern const char *console_str_sep;
extern int _PrintFilterSettings;

/*****************************************************************************
 * Function:    initFilter
 * Parameters:  void
 * Return:      void
 * Description: Initializes the Filter Controls to their minima and disables all
 *              of the enable variables.
 *****************************************************************************/
void initFilter(void)
{
    disableFrontEnd();
    setFrontEndDigitalFTune(FTUNE_DIG_MIN);
    setFrontEndDigitalQTune(QTUNE_DIG_MIN);
    disableFrontEndAD();
    disableFrontEndFD();

    disableBackEnd();
    setBackEndDigitalFTune(FTUNE_DIG_MIN);
    setBackEndDigitalQTune(QTUNE_DIG_MIN);
    disableBackEndAD();
    disableBackEndFD();

    setCouplingUpper(COUPLING_MIN);
    setCouplingLower(COUPLING_MIN);
    setCouplingUFLB(COUPLING_MIN);
    setCouplingLFUB(COUPLING_MIN);

    setFrontEndAnalogFTune(ANALOG_MIN);
    setFrontEndAnalogQTune(ANALOG_MIN);
    setBackEndAnalogFTune(ANALOG_MIN);
    setBackEndAnalogQTune(ANALOG_MIN);

    DEBUG.ADFen = 0;
    DEBUG.FDFen = 0;
    DEBUG.ADBen = 0;
    DEBUG.FDBen = 0;
    DEBUG.RFOn = 1;
}

/*****************************************************************************
 * Function:    printFilterOptions
 * Parameters:  void
 * Return:      void
 * Description: Prints each of the the filter options to the Console.
 *****************************************************************************/
void printFilterOptions(void)
{
    printSeperator();
    printFrontEndStatus();
    printFrontEndADStatus();
    printFrontEndFDStatus();
    printFrontEndDigitalQTune();
    //printFrontEndAnalogQTune();
    printFrontEndDigitalFTune();
    //printFrontEndAnalogFTune();
    printBackEndStatus();
    printBackEndADStatus();
    printBackEndFDStatus();
    printBackEndDigitalQTune();
    //printBackEndAnalogQTune();
    printBackEndDigitalFTune();
    //printBackEndAnalogFTune();
    printCouplingUpper();
    printCouplingLower();
    printCouplingUFLB();
    printCouplingLFUB();
    printRFSwitchStatus();
    printSeperator();
}
```

```
/*******************************************************************************
 * Function:    printAlgorithmOptions
 * Parameters:  void
 * Return:      void
 * Description: Prints each of the the algorithm options to the console.
 *******************************************************************************/
void printAlgorithmOptions(void)
{
    printSeperator();
    printCenterFreq();
    printFreqTol();
    printFrontEndADThresh1();
    printFrontEndADThresh2();
    printFrontEndQOffset();
    printFrontEndQBackOff();
    printFrontEndFOffset();
    printBackEndADThresh1();
    printBackEndADThresh2();
    printBackEndQOffset();
    printBackEndQBackOff();
    printBackEndFOffset();
    printCouplingUpper();
    printCouplingLower();
    printCouplingUFLB();
    printCouplingLFUB();
    printSeperator();
}

/*******************************************************************************
 * Function:    updateFilterData
 * Parameters:  void
 * Return:      void
 * Description: Formats and stores the filter options to be programmed.
 *******************************************************************************/
void updateFilterData(void)
{
    filterData[0] = ((CAPCON2.FDBen & 1) << 10)             |
                    ((CAPCON2.UFLB & COUPLING_MAX) << 5 )   |
                    (CAPCON2.LFUB & COUPLING_MAX);
    filterData[1] = ((CAPCON1.FDFen & 1) << 10)             |
                    ((CAPCON1.lower & COUPLING_MAX) << 5)   |
                    (CAPCON1.upper & COUPLING_MAX);
    filterData[2] = ((BENDCON.ADen & 1) << 15)              |
                    ((BENDCON.qTune & QTUNE_DIG_MAX) << 9)  |
                    ((BENDCON.fTune & FTUNE_DIG_MAX) << 1)  |
                    (BENDCON.en & 1);
    filterData[3] = ((FENDCON.ADen & 1) << 15)              |
                    ((FENDCON.qTune & QTUNE_DIG_MAX) << 9)  |
                    ((FENDCON.fTune & FTUNE_DIG_MAX) << 1)  |
                    (FENDCON.en & 1);
}

/*******************************************************************************
 * Function:    programFilter
 * Parameters:  void
 * Return:      void
 * Description: The filter is programmed by transmitting the 64 programming bits to
 *              the serial-to-parallel register of the filter via SPI.  Once all 64
 *              bits are transmitted, the latch line is raised, a clock pulse is
 *              generated and the latch line is lowered storing the bits.
 *******************************************************************************/
void programFilter(void)
{
    updateFilterData();

    setPinLow(FILTER_CLK);
    setPinLow(FILTER_DATA);
    setPinLow(FILTER_LATCH);
    enableSPI2();
    writeSPI2(filterData[0]);
    writeSPI2(filterData[1]);
    writeSPI2(filterData[2]);
    writeSPI2(filterData[3]);
    disableSPI2();

    setFilterLatchAsOutput();
    setFilterDataAsOutput();
    setFilterClkAsOutput();
    setPinLow(FILTER_LATCH);
    setPinLow(FILTER_CLK);
    prgmDelay();
    setPinHigh(FILTER_LATCH);
    prgmDelay();
    setPinHigh(FILTER_CLK);
    prgmDelay();
    setPinLow(FILTER_DATA);
    setPinLow(FILTER_LATCH);
    prgmDelay();
    setPinLow(FILTER_CLK);
    prgmDelay();
}

/*******************************************************************************
 * Function:    prgmDelay
 * Parameters:  void
 * Return:      void
 * Description: A simplistic delay used to ensure timing requirements are met during
 *              the filter programming process.
 *******************************************************************************/
void prgmDelay(void)
{
    Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop();
    Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop();
}

/*******************************************************************************
 * Function:    updateAnalogTuning
 * Parameters:  void
 * Return:      void
 * Description: Programs the DACS with the appropriate current Analog F-Tune
 *              and Q-Tune values.
 *******************************************************************************/
void updateAnalogTuning(void)
{
    programFrontEndDAC(loadInputRegA(getFrontEndAnalogFTune()));
    programFrontEndDAC(loadInputRegB(getFrontEndAnalogQTune()));
    programFrontEndDAC(loadDACRegsABUpdateOutputsAB());
    programBackEndDAC(loadInputRegA(getBackEndAnalogQTune()));
    programBackEndDAC(loadInputRegB(getBackEndAnalogFTune()));
    programBackEndDAC(loadDACRegsABUpdateOutputsAB());
```

```
}

/*******************************************************************************
 * Function:    algorithm
 * Parameters:  void
 * Return:      void
 * Description: Implementation of the Single-Pole Tuning Algorithm
 *              using only Digital Controls.
 *******************************************************************************/
void algorithm(void)
{
    int itt = 0;
    int prevFrontEndFCnt = 0;
    int prevFrontEndFCntDiff = 0, curFrontEndFCntDiff = 0;

    turnRFSwitchOff();
    enableFrontEnd();   // Enable Front-End
    enableFrontEndAD(); // Enable Front-End Amplitude Detector
    enableFrontEndFD(); // Enable Front-End Frequency Divider
    disableBackEnd();   // Disable Back-End
    disableBackEndAD(); // Disable Back-End Amplitude Detector
    disableBackEndFD(); // Disable Back-End Frequency Divider

    // Set frequency setting to controls mid-point
    setFrontEndDigitalFTune((FTUNE_DIG_MIN + FTUNE_DIG_MAX) >> 1);
    findCriticalOsc();
    FrontEndFCnt = readFrontEndFD();
    while ( (FrontEndFCnt < (CenterFreq - FreqTol)) || (FrontEndFCnt > (CenterFreq + FreqTol)) )
    {
        if ( FrontEndFCnt < (CenterFreq - FreqTol) )
            decFrontEndDigitalFTune();
        else if ( FrontEndFCnt > (CenterFreq + FreqTol) )
            incFrontEndDigitalFTune();

        findCriticalOsc();
        FrontEndFCnt = readFrontEndFD();
        if ( ++itt > MAX_FREQTUNE_ITTS )
        {
            txStrUART1("Max Frequency Tune Itterations Exceeded...\r\n");
            return;
        }
    }
    // Frequency within Tolerance now find Closest Setting
    prevFrontEndFCnt = FrontEndFCnt;
    prevFrontEndFCntDiff = absDiff(CenterFreq,prevFrontEndFCnt);
    if ( FrontEndFCnt < CenterFreq )
    {
        decFrontEndDigitalFTune();
        findCriticalOsc();
        FrontEndFCnt = readFrontEndFD();
        curFrontEndFCntDiff = absDiff(CenterFreq,FrontEndFCnt);
        if ( curFrontEndFCntDiff > prevFrontEndFCntDiff )
        {
            incFrontEndDigitalFTune();
            programFilter();
        }
    }
    else if ( FrontEndFCnt > CenterFreq )
    {
        incFrontEndDigitalFTune();
        findCriticalOsc();
        FrontEndFCnt = readFrontEndFD();
        curFrontEndFCntDiff = absDiff(CenterFreq,FrontEndFCnt);
        if ( curFrontEndFCntDiff > prevFrontEndFCntDiff )
        {
            decFrontEndDigitalFTune();
            programFilter();
        }
    }
    disableFrontEndFD();

    #if _DEBUG_ALGORITHM_ == 1
        txStrUART1("----Freq Tune Essentially Done---\r\n");
        printFrontEndDigitalFTune();
        printFrontEndDigitalQTune();
    #endif

    // Remove Excess Q-Enhancement Needed for Dependable Frequency Divider Readings
    setFrontEndDigitalQTune(getFrontEndDigitalQTune() - FrontEndQOffset);
    #if _DEBUG_ALGORITHM_ == 1
        txStrUART1("----Need to decrement extra Q-Enhancement for Freq Readings---\r\n");
        printFrontEndDigitalQTune();
    #endif

    // Back-Off Q-Enhancement by Set Amount to Achieve Desired Bandwidth
    setFrontEndDigitalQTune(getFrontEndDigitalQTune() - FrontEndQBackOff);
    #if _DEBUG_ALGORITHM_ == 1
        txStrUART1("----Do Q Back-Off for BW---\r\n");
        printFrontEndDigitalQTune();
    #endif

    // Ensure Filter is Not Oscillating After Q-Enhancement Back-Off (Insufficient Back-Off)
    enableFrontEndAD();
    programFilter();
    FrontEndAD = readFrontEndAD();
    while ( FrontEndAD < (FrontEndNonOsc - FrontEndADThresh1) )
    {
        #if _DEBUG_ALGORITHM_ == 1
            txStrUART1("Still Oscillating after Back-Off...\r\n");
            printFrontEndDigitalQTune();
            printFrontEndAD();
        #endif

        decFrontEndDigitalQTune();
        programFilter();
        FrontEndAD = readFrontEndAD();

        #if _DEBUG_ALGORITHM_ == 1
            printFrontEndDigitalQTune();
            printFrontEndAD();
        #endif
    }
    disableFrontEndAD();

    // Counter the Frequency Shift Caused by Q-Enhancement Back-Off by Increasing Digital F-Tuning
    setFrontEndDigitalFTune(getFrontEndDigitalFTune() + FrontEndFOffset);

            // Set Coupling (Should be 0's for a Single-Pole Implementation);
            setCouplingUpper(CouplingUpper);
    setCouplingLower(CouplingLower);
```

```
    setCouplingUFLB(CouplingUFLB);
    setCouplingLFUB(CouplingLFUB);
    programFilter();

    // Algorithm is Complete, so Turn on RF Switch
    turnRFSwitchOn();

    if ( _PrintFilterSettings )
        printFilterOptions();
}

/*******************************************************************************
 * Function:    findCriticalOsc
 * Parameters:  void
 * Return:      void
 * Description: Critical Oscillation is found by comparing the Amplitude Detector
 *              readings of a Non-Oscillation Q-Level (Q=0) with a continually
 *              increasing Q-Level.  Once the Amplitude Detector reading drops below
 *              a the Non-Oscillation reading minus a threshold value, critical
 *              oscillation is found.   To ensure a stable Frequency Divider output
 *              the Q-Level is increased by a set offset.
 *
 *              The following variables are issued from the QEFilter PC Application:
 *              FrontEndADThresh1 and FrontEndQOffset.
 *******************************************************************************/
void findCriticalOsc(void)
{
    int itt = 0;
    FrontEndNonOsc = ANALOG_MIN;
    FrontEndAD = ANALOG_MAX;

    setFrontEndDigitalQTune(QTUNE_DIG_MIN);
    programFilter();
    FrontEndNonOsc = readFrontEndAD();

    #if _DEBUG_CRITICALOSC_ == 1
        printFrontEndNonOsc();
        printFrontEndDigitalQTune();
    #endif

    while ( FrontEndAD >= (FrontEndNonOsc - FrontEndADThresh1) )
    {
        incFrontEndDigitalQTune();
        programFilter();
        FrontEndAD = readFrontEndAD();

        #if _DEBUG_CRITICALOSC_ == 1
            printFrontEndAD();
            printFrontEndDigitalQTune();
        #endif

        if ( ++itt > MAX_CRITOSC_ITTS )
        {
            txStrUART1("Max Critical Oscillation Itterations Exceeded...\r\n");
            return;
        }
    }
    // Ensure Oscillation for Dependable Frequency Divider Readings
    setFrontEndDigitalQTune(getFrontEndDigitalQTune() + FrontEndQOffset);
    programFilter();
}

/* End of File */
```

CMDSM.H

```
/*******************************************************************************
 * Filename:    cmdsm.h
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Discription: This file houses the preprocessor definitions and function prototypes
 *              needed by the CMD State-Machine.
 *******************************************************************************/
#ifndef _CMDSS_H
#define _CMDSS_H

/* Preprocessor Definitions & Macros */
#define CMD_STATE_NAME_LEN                  10
#define CMD_NUMSTATES                       10
#define CMD_STATE_CENTERFREQ                0
#define CMD_STATE_FREQTOL                   1
#define CMD_STATE_FRONTENDAD                2
#define CMD_STATE_FRONTENDQF                3
#define CMD_STATE_BACKENDAD                 4
#define CMD_STATE_BACKENDQF                 5
#define CMD_STATE_COUPLAT                   6
#define CMD_STATE_COUPCROSS                 7
#define CMD_STATE_DEBUG                     8
#define CMD_STATE_STOP                      9

#define nextState()                         setState(RxCMD[curState].nextState)
#define printRxData()                       strPopulate16Bit(strRxData,tmpRegVal,'#',4)
#define printItteration()                   strPopulate16Bit(strItt,algItt,'#',4)

/* Function Prototypes */
void initStateMachine(void);
void setState(int);
void printCurState(void);
void processRxData(void);
void processTxData(void);
void centerFreqAction(void);
void freqTolAction(void);
void frontEndADAction(void);
void frontEndQFAction(void);
void backEndADAction(void);
void backEndQFAction(void);
void coupLatAction(void);
void coupCrossAction(void);
void debugAction(void);
void stopAction(void);

/* Type Definitions */
typedef struct {
   char stateName[CMD_STATE_NAME_LEN];
   int nextState;
```

```
   void (*action)();
} CommandStateMachine;

#endif

/* End of File */
```

CMDSM.C

```
/*******************************************************************************
 * Filename:    cmdsm.c
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Discription: This file houses the functions needed to read and interpret the CMD
 *              string being sent from the QEFilter PC application, via UART,
 *              containing the algorithms parameters.
 *******************************************************************************/
#include "main.h"

/* Global Variables */
unsigned char rxByte;
char strRxData[] = "tmpRegVal = ####\r\n";
char strItt[] = "~~~~~ Algorithm Itteration #### ~~~~~\r\n";

int curState, curRxByte, _PrintAlgorithmOptions, _PrintFilterSettings, _PrintItterations, tmpRegVal, algItt;

extern int _UpdateFilter;
extern int CenterFreq, FreqTol;
extern int FrontEndADThresh1, FrontEndADThresh2, FrontEndQOffset, FrontEndQBackOff, FrontEndFOffset;
extern int BackEndADThresh1, BackEndADThresh2, BackEndQOffset, BackEndQBackOff, BackEndFOffset;
extern int CouplingUpper, CouplingLower, CouplingUFLB, CouplingLFUB;

CommandStateMachine RxCMD[CMD_NUMSTATES] =
{
    { "CENTERFREQ",  CMD_STATE_FREQTOL,     centerFreqAction },
    { "FREQTOL   ",  CMD_STATE_FRONTENDAD,  freqTolAction },
    { "FRONTENDAD",  CMD_STATE_FRONTENDQF,  frontEndADAction },
    { "FRONTENDQF",  CMD_STATE_BACKENDAD,   frontEndQFAction },
    { "BACKENDAD ",  CMD_STATE_BACKENDQF,   backEndADAction },
    { "BACKENDQF ",  CMD_STATE_COUPLAT,     backEndQFAction },
    { "COUPLAT   ",  CMD_STATE_COUPCROSS,   coupLatAction },
    { "COUPCROSS ",  CMD_STATE_DEBUG,       coupCrossAction },
    { "DEBUG     ",  CMD_STATE_STOP,        debugAction },
    { "STOP      ",  CMD_STATE_CENTERFREQ,  stopAction }
};

/*******************************************************************************
 * Function:    initStateMachine
 * Parameters:  void
 * Return:      void
 * Description: Initializes the variables used by the CMD State Machine.
 *******************************************************************************/
void initStateMachine(void)
{
    rxByte    = NUL;
    curRxByte = 0;
    tmpRegVal = 0;
    algItt    = 0;
    _PrintAlgorithmOptions = FALSE;
    _PrintFilterSettings   = FALSE;
    _PrintItterations      = FALSE;
}

/*******************************************************************************
 * Function:    setState
 * Parameters:  int state - valid inputs defined in cdsms.h
 * Return:      void
 * Description: Sets the the current state variable based on the input variable state
 *              and clears the curRxByte (count of the number of received bytes).
 *******************************************************************************/
void setState(int state)
{
    curState = state;
    curRxByte = 0;

    #if _DEBUG_RXSM_ == 1
        txStrUART1("Entering ");
        txStrUART1(RxCMD[curState].stateName);
        txStrUART1("\r\n");
    #endif
}

/*******************************************************************************
 * Function:    processRxData
 * Parameters:  void
 * Return:      void
 * Description: Reads the current rxByte from UART1 and either sets the appropriate
 *              state or executes the current states action.
 *******************************************************************************/
void processRxData(void)
{
    rxByteUART1(&rxByte);

    switch ( rxByte )
    {
        case NUL:
            return;
        case '@': // Beginning of Command String
            setState(CMD_STATE_CENTERFREQ);
            break;
        case '&': // Data Seperator within Command String
            nextState();
            break;
        case '|': // End of Command String
            setState(CMD_STATE_STOP);
            break;
        default:
            break;
    }

    if ( rxByte != '@' && rxByte != '&' )
        (*RxCMD[curState].action)();
    rxByte = NUL;
}
```

```
/*******************************************************************************
 * Function:    centerFreqAction
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 4 rxBytes from ASCII to Hex.  It then
 *              stores this value into the CenterFreq variable.
 *******************************************************************************/
void centerFreqAction(void)
{
    switch ( curRxByte++ )
    {
        case 0:
            tmpRegVal = (int) (ASCII2HEX(rxByte) << 12);
            break;
        case 1:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 8);
            break;
        case 2:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 4);
            break;
        case 3:
            tmpRegVal += (int) (ASCII2HEX(rxByte));

            CenterFreq = tmpRegVal;

            #if _DEBUG_RXSM_ == 1
                printRxData();
            #endif

            tmpRegVal = 0;
            break;
        default:
            break;
    }
}

/*******************************************************************************
 * Function:    freqTolAction
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 2 rxBytes from ASCII to Hex.  It then
 *              stores this value into the FreqTol variable.
 *******************************************************************************/
void freqTolAction(void)
{
    switch ( curRxByte++ )
    {
        case 0:
            tmpRegVal = (int) (ASCII2HEX(rxByte) << 4);
            break;
        case 1:
            tmpRegVal += (int) (ASCII2HEX(rxByte));

            FreqTol = tmpRegVal;

            #if _DEBUG_RXSM_ == 1
                printRxData();
            #endif

            tmpRegVal = 0;
            break;
        default:
            break;
    }
}

/*******************************************************************************
 * Function:    frontEndADAction
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 4 rxBytes from ASCII to Hex.  It then
 *              stores this value into the FrontEndADThresh1 and
 *              FrontEndADThresh2 variables.
 *******************************************************************************/
void frontEndADAction(void)
{
    switch ( curRxByte++ )
    {
        case 0:
            tmpRegVal = (int) (ASCII2HEX(rxByte) << 12);
            break;
        case 1:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 8);
            break;
        case 2:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 4);
            break;
        case 3:
            tmpRegVal += (int) (ASCII2HEX(rxByte));

            FrontEndADThresh1 = tmpRegVal >> 7;
            FrontEndADThresh2 = tmpRegVal & 0x7F;

            #if _DEBUG_RXSM_ == 1
                printRxData();
            #endif

            tmpRegVal = 0;
            break;
        default:
            break;
    }
}

/*******************************************************************************
 * Function:    frontEndQFAction
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 3 rxBytes from ASCII to Hex.  It then
 *              stores this value into the FrontEndQOffset, FrontEndQBackOff and
 *              FrontEndFOffset variables.
 *******************************************************************************/
void frontEndQFAction(void)
{
    switch ( curRxByte++ )
    {
        case 0:
            tmpRegVal = (int) (ASCII2HEX(rxByte) << 8);
            break;
        case 1:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 4);
            break;
        case 2:
            tmpRegVal += (int) (ASCII2HEX(rxByte));
```

```
            FrontEndQOffset = tmpRegVal >> 8;
            FrontEndQBackOff = (tmpRegVal >> 4) & 0xF;;
            FrontEndFOffset = tmpRegVal & 0xF;

            #if _DEBUG_RXSM_ == 1
                printRxData();
            #endif

            tmpRegVal = 0;
            break;
        default:
            break;
    }
}

/*******************************************************************************
 * Function:    backEndADAction
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 4 rxBytes from ASCII to Hex.  It then
 *              stores this value into the BackEndADThresh1 and
 *              BackEndADThresh2 variables.
 *******************************************************************************/
void backEndADAction(void)
{
    switch ( curRxByte++ )
    {
        case 0:
            tmpRegVal = (int) (ASCII2HEX(rxByte) << 12);
            break;
        case 1:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 8);
            break;
        case 2:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 4);
            break;
        case 3:
            tmpRegVal += (int) (ASCII2HEX(rxByte));

            BackEndADThresh1 = tmpRegVal >> 7;
            BackEndADThresh2 = tmpRegVal & 0x7F;

            #if _DEBUG_RXSM_ == 1
                printRxData();
            #endif

            tmpRegVal = 0;
            break;
        default:
            break;
    }
}
/*******************************************************************************
 * Function:    backEndQFAction
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 3 rxBytes from ASCII to Hex.  It then
 *              stores this value into the BackEndQOffset, BackEndQBackOff and
 *              BackEndFOffset variables.
 *******************************************************************************/
void backEndQFAction(void)
{
    switch ( curRxByte++ )
    {
        case 0:
            tmpRegVal = (int) (ASCII2HEX(rxByte) << 8);
            break;
        case 1:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 4);
            break;
        case 2:
            tmpRegVal += (int) (ASCII2HEX(rxByte));

            BackEndQOffset = tmpRegVal >> 8;
            BackEndQBackOff = (tmpRegVal >> 4) & 0xF;;
            BackEndFOffset = tmpRegVal & 0xF;

            #if _DEBUG_RXSM_ == 1
                printRxData();
            #endif

            tmpRegVal = 0;
            break;
        default:
            break;
    }
}

/*******************************************************************************
 * Function:    coupLatAction
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 3 rxBytes from ASCII to Hex.  It then
 *              stores this value into the CouplingUpper and CouplingLower variables.
 *******************************************************************************/
void coupLatAction(void)
{
    switch ( curRxByte++ )
    {
        case 0:
            tmpRegVal = (int) (ASCII2HEX(rxByte) << 8);
            break;
        case 1:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 4);
            break;
        case 2:
            tmpRegVal += (int) (ASCII2HEX(rxByte));

            CouplingUpper = tmpRegVal >> 6;
            CouplingLower = tmpRegVal & 0x3F;

            #if _DEBUG_RXSM_ == 1
                printRxData();
            #endif

            tmpRegVal = 0;
            break;
        default:
            break;
    }
}
```

```
/*******************************************************************************
 * Function:    coupCrossAction
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 3 rxBytes from ASCII to Hex.  It then
 *              stores this value into the CouplingUFLB and CouplingLFUB variables.
 *******************************************************************************/
void coupCrossAction(void)
{
    switch ( curRxByte++ )
    {
        case 0:
            tmpRegVal = (int) (ASCII2HEX(rxByte) << 8);
            break;
        case 1:
            tmpRegVal += (int) (ASCII2HEX(rxByte) << 4);
            break;
        case 2:
            tmpRegVal += (int) (ASCII2HEX(rxByte));

            CouplingUFLB = tmpRegVal >> 6;
            CouplingLFUB = tmpRegVal & 0x3F;

            #if _DEBUG_RXSM_ == 1
                printRxData();
            #endif

            tmpRegVal = 0;
            break;
        default:
            break;
    }
}

/*******************************************************************************
 * Function:    debugAction
 * Parameters:  void
 * Return:      void
 * Description: Receives, arranges, and converts 1 rxByte from ASCII to Hex.  It then
 *              sets the appropriate print flags.
 *******************************************************************************/
void debugAction(void)
{
    tmpRegVal = (int) (ASCII2HEX(rxByte));

    _PrintAlgorithmOptions = (tmpRegVal >> 2) & 1;
    _PrintFilterSettings = (tmpRegVal >> 1) & 1;
    _PrintItterations = tmpRegVal & 1;

    #if _DEBUG_RXSM_ == 1
        printRxData();
    #endif

    tmpRegVal = 0;
}

/*******************************************************************************
 * Function:    stopAction
 * Parameters:  void
 * Return:      void
 * Description: Increments the algorithm's iteration and prints debug information if
 *              the appropriate flags are set.
 *******************************************************************************/
void stopAction(void)
{
    if (algItt++ > 1000)
        algItt = 1;

    if ( _PrintItterations )
    {
        _PrintItterations = FALSE;
        printItteration();
    }
    if ( _PrintAlgorithmOptions )
    {
        _PrintAlgorithmOptions = FALSE;
        printAlgorithmOptions();
    }
    _UpdateFilter = TRUE;
}

/* End of File */
```

Additional Files

All source code not documented in this appendix has been inherited from Appendix G.

APPENDIX I – TWO-POLE TUNING ALGORITHM SOURCE CODE

MAIN.C

```
/*****************************************************************************
 * Filename:    main.c
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Description: This file houses the main loop and initialization functions.  Once
 *              initialized, the system waits for a CMD string to be received, from
 *              the QEFilter PC Application, that contains the algorithm's parameters.
 *              Once the command string has been analzed, the algorithm is run.  It
 *              then waits for the next CMD string.
 *****************************************************************************/
#include "main.h"

/* Device Configuration Bits */
// Boot segment may be written, no boot segement, no boot RAM
_FBS(RBS_NO_RAM & BSS_NO_FLASH  & BWRP_WRPROTECT_OFF);
// No Secure Ram, no secure segment, write protection disabled
_FSS(RSS_NO_RAM & SSS_NO_FLASH & SWRP_WRPROTECT_OFF);
// General Segment and Code protect off, General Segment may be written
_FGS(GSS_OFF & GCP_OFF & GWRP_OFF);
// Primary Oscillator (XT, HS, EC), start-up with user-selected oscillator
_FOSCSEL(FNOSC_PRI & IESO_OFF);
// Clock switching and monitoring disabled, one re-configuration for remappable I/O,
// Primary Oscillator is External Clock
_FOSC(FCKSM_CSDCMD & IOL1WAY_ON & OSCIOFNC_ON & POSCMD_EC);
// Watchdog Timer off
_FWDT(FWDTEN_OFF & WINDIS_OFF & WDTPRE_PR32 & WDTPOST_PS1);
// I2C mapped to SDA1/SCL1, 128ms Power on Reset
_FPOR(ALTI2C_OFF & FPWRT_PWR128);
// JTAG disabled, ICD communicate on PGC1/EMUC1 and PGD1/EMUD1
_FICD(JTAGEN_OFF & ICS_PGD1);

/* Global Variables */
const char * console_str_boot  = "dsPIC33fj64gp802 Acknowledges You\r\n";
const char * algorithm_version = "Two-Pole Tuning Algorithm\r\n";
const char * console_str_sep   = \
"--------------------------------------------------------------------------------\r\n";
int _UpdateFilter;
extern int _PrintFilterSettings;
extern int CenterFreq, FreqTol;
extern int FrontEndADThresh1, FrontEndADThresh2, FrontEndQOffset, FrontEndQBackOff, FrontEndFOffset;
extern int BackEndADThresh1, BackEndADThresh2, BackEndQOffset, BackEndQBackOff, BackEndFOffset;
extern int CouplingUpper, CouplingLower, CouplingUFLB, CouplingLFUB;

/*****************************************************************************
 * Function:    main
 * Parameters:  void
 * Return:      int - never reaches return value
 * Description: Initializes the system and filter variables.  The filter is then
 *              programmed to have all options at their minima or disabled.  It then
 *              waits reads the UART and process its data appropriately.  If the
 *              _UpdateFilte flag is set it will run the tuning Algorithm.
 *****************************************************************************/
int main(void)
{
    init();
    initFilter();
    programFilter();

    _CN21IE = 1;                            // Enable Change Notification Interrupt on RB9
            while ( TRUE )
    {
        processRxData();
                    if ( _UpdateFilter )
        {
            _UpdateFilter = FALSE;
                        algorithm();
        }
    }
}

/*****************************************************************************
 * Function:    init
 * Parameters:  void
 * Return:      void
 * Description: Initializeds the modules and sets up the interrupts.
 *****************************************************************************/
void init(void)
{
    RCONbits.SWDTEN = 0;                    // Software Disable of the Watchdog Timer

    _SPI1IE = 1;                            // Enable SPI1 Transfer Complete Interrupt
    _SPI1IF = 0;                            // Clear SPI1 Event Interrupt Flag
    _SPI2IE = 1;                            // Enable SPI1 Transfer Complete Interrupt
    _SPI2IF = 0;                            // Clear SPI1 Event Interrupt Flag
    _T2IE   = 1;                            // Enable Timer2 Overflow Flag
    _T2IF   = 0;                            // Clear Timer2 Overflow Flag

    initPins();
    initSPI1();
    initSPI2();
    initADC();
    initUART1(BAUDRATE, 0x06, 0x05);        // Configure UART Peripherial
    printBootStr();
    printSeperator();
    printAlgorithmVer();
    printSeperator();
}
```

```
/*******************************************************************************
 * Function:    initPins
 * Parameters:  void
 * Return:      void
 * Description: Initializeds the systems pins (see pinconfig.h).
 *******************************************************************************/
void initPins(void)
{
    /* Configure Analog Pins */
    ADPCFG = 0xFFFC;                        // All Digital Pins Except AN0 & AN1

    /* UART1 Configuration */
    configU1RXDPin();                      // Setup U1RX Reprogrammable-Pin Register
    configU1CTSPin();                      // Setup U1CTS Reprogrammable-Pin Register
    configU1TXDPin();                      // Setup U1TX Reprogrammable-Pin Register
    configU1RTSPin();                      // Setup U1RTS Reprogrammable-Pin Register

    /* SPI2 Configuration */
    configSDO2Pin();                        // Setup SDO2 Reprogrammable-Pin Register
    configSCK2Pin();                        // Setup SCK2 Reprogrammable-Pin Register
    setFilterDataAsOutput();
    setFilterClkAsOutput();
    setFilterLatchAsOutput();
    setPinLow(FILTER_CLK);                  // Set FILTER_CLK Low
    setPinLow(FILTER_DATA);                 // Set FILTER_DATA Low
    setPinLow(FILTER_LATCH);                // Set FILTER_LATCH Low

    /* SPI1 Configuration */
    configSDO1Pin();                        // Setup SDO1 Reprogrammable-Pin Register
    configSCK1Pin();                        // Setup SCK1 Reprogrammable-Pin Register
    setBackDACSelAsOutput();                // Set DAC_BACK_SEL Low
    setFrontDACSelAsOutput();               // Set DAC_FRONT_SEL Low
    setPinLow(DAC_DATA);                    // Set DAC_DATA Low
    setPinLow(DAC_CLK);                     // Set DAC_CLK Low
    deselectBothDACs();                     // Set DAC_BACK_SEL & DAC_FRONT_SEL High

    /* Frequency Divider Configuration */
    setFreqDividerAsInput();                // Set FD_OUTPUT as Digital Input
    setFreqDividerSelAsOutput();            // Set FD_SEL as Digital Output
    setPinLow(FD_SEL);                      // Set FD_SEL Low

    /* RF Switch Configuration */
    setRFSwitchAsOutput();                  // Set RF Switch as Digital Ouput

    /* Debug LED Configuration */
    setDebugLEDAsOutput();                  // Set Debug LED as Digital Output
    turnDebugLEDOff();                      // Turn Debug LED Off

    /* Debug BTN Configuration */
    setDebugBtnAsInput();
    _CNIE = 1;                              // Enable Change Notification Interrupts
    _CNIP = 2;                              // Set Interrupt priority
    _CN21IE = 0;                            // Disable Change Notification Interrupt on RB9
    _CNIF = 0;                              // Clear Change Notifitication Interupt Flag
}

/* End of File */

/*******************************************************************************
 * Interrupt:   _CNInterrupt
 * Parameters:  void
 * Return:      void
 * Description: Interrupt is triggered when the debug button is clicked. The debug
 *              button runs the algorithm.
 *******************************************************************************/
void _ISR __attribute__((auto_psv)) _CNInterrupt(void)
{
    int i;
    for (i = 0; i < 1000; i++); // Debounce

    if ( isBtnPressed() )
    {
        /* Default Algorithm Settings for 10 MHz Bandwidth centered at 450 MHz */
        CenterFreq = 4500;                          // 450.0 MHz
        FreqTol = 2;                                // 0.2 MHz
        FrontEndADThresh1 = FrontEndADThresh2 = 10;
        FrontEndQOffset = 2;
        FrontEndQBackOff = 5;
        FrontEndFOffset = 3;
        BackEndADThresh1 = BackEndADThresh2 = 10;
        BackEndQOffset = 2;
        BackEndQBackOff = 5;
        BackEndFOffset = 3;
        CouplingUpper = CouplingLower = 10;
        CouplingUFLB = CouplingLFUB = 0;

        _PrintFilterSettings = TRUE;
        algorithm();
    }
    _CNIF = 0; // Clear Change Notification Interrupt Flag
}

/* End of File */
```

QEFILTER.H

```
/*******************************************************************************
 * Filename:    qefilter.h
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Discription: This file houses the preprocessor definitions and function prototypes
 *              needed by the QE Filter Tuning Algorithm.
 *******************************************************************************/
#ifndef _QEFILTER_H
#define _QEFILTER_H

/* Preprocessor Definitions & Macros */
#define FTUNE_DIG_MIN                       165 // Limit the Frequency Range for Reliable Frequency Divider Ouputs
#define FTUNE_DIG_MAX                       255
#define QTUNE_DIG_MIN                       0
#define QTUNE_DIG_MAX                       63
#define COUPLING_MIN                        0
#define COUPLING_MAX                        31
#define ANALOG_MIN                          0
```

```
#define ANALOG_MAX                      1023
#define MAX_FREQTUNE_ITTS               100
#define MAX_CRITOSC_ITTS                64
#define isFrontEndEnabled()             (!FENDCON.en ? 1 : 0)
#define enableFrontEnd()                FENDCON.en    = 0 // Active-Low Enable
#define disableFrontEnd()               FENDCON.en    = 1
#define isFrontEndADEnabled()           (!FENDCON.ADen ? 1 : 0)
#define enableFrontEndAD()              FENDCON.ADen  = 0 // Active-Low Enable
#define disableFrontEndAD()             FENDCON.ADen  = 1
#define isFrontEndFDEnabled()           (!CAPCON1.FDFen ? 1 : 0)
#define enableFrontEndFD()              CAPCON1.FDFen = 0 // Active-Low Enable
#define disableFrontEndFD()             CAPCON1.FDFen = 1
#define getFrontEndDigitalFTune()       FENDCON.fTune
#define setFrontEndDigitalFTune(val)    FENDCON.fTune = ((val) > FTUNE_DIG_MAX ? FTUNE_DIG_MAX : ((val) < FTUNE_DIG_MIN ? FTUNE_DIG_MIN : (val)))
#define incFrontEndDigitalFTune()       setFrontEndDigitalFTune(FENDCON.fTune + 1)
#define decFrontEndDigitalFTune()       setFrontEndDigitalFTune(FENDCON.fTune - 1)
#define getFrontEndDigitalQTune()       FENDCON.qTune
#define setFrontEndDigitalQTune(val)    FENDCON.qTune = ((val) > QTUNE_DIG_MAX ? QTUNE_DIG_MAX : ((val) < QTUNE_DIG_MIN ? QTUNE_DIG_MIN : (val)))
#define incFrontEndDigitalQTune()       setFrontEndDigitalQTune(FENDCON.qTune + 1)
#define decFrontEndDigitalQTune()       setFrontEndDigitalQTune(FENDCON.qTune - 1)
#define isBackEndEnabled()              (!BENDCON.en ? 1 : 0)
#define enableBackEnd()                 BENDCON.en = 0 // Active-Low Enable
#define disableBackEnd()                BENDCON.en = 1
#define isBackEndADEnabled()            (!BENDCON.ADen ? 1 : 0)
#define enableBackEndAD()               BENDCON.ADen = 0 // Active-Low Enable
#define disableBackEndAD()              BENDCON.ADen = 1
#define isBackEndFDEnabled()            (!CAPCON2.FDBen ? 1 : 0)
#define enableBackEndFD()               CAPCON2.FDBen = 0 // Active-Low Enable
#define disableBackEndFD()              CAPCON2.FDBen = 1
#define getBackEndDigitalFTune()        BENDCON.fTune
#define setBackEndDigitalFTune(val)     BENDCON.fTune = ((val) > FTUNE_DIG_MAX ? FTUNE_DIG_MAX : ((val) < FTUNE_DIG_MIN ? FTUNE_DIG_MIN : (val)))
#define incBackEndDigitalFTune()        setBackEndDigitalFTune(BENDCON.fTune + 1)
#define decBackEndDigitalFTune()        setBackEndDigitalFTune(BENDCON.fTune - 1)
#define getBackEndDigitalQTune()        BENDCON.qTune
#define setBackEndDigitalQTune(val)     BENDCON.qTune = ((val) > QTUNE_DIG_MAX ? QTUNE_DIG_MAX : ((val) < QTUNE_DIG_MIN ? QTUNE_DIG_MIN : (val)))
#define incBackEndDigitalQTune()        setBackEndDigitalQTune(BENDCON.qTune + 1)
#define decBackEndDigitalQTune()        setBackEndDigitalQTune(BENDCON.qTune - 1)
#define getCouplingUpper()              CAPCON1.upper
#define setCouplingUpper(val)           CAPCON1.upper = ((val) > COUPLING_MAX ? COUPLING_MAX : ((val) < COUPLING_MIN ? COUPLING_MIN : (val)))
#define getCouplingLower()              CAPCON1.lower
#define setCouplingLower(val)           CAPCON1.lower = ((val) > COUPLING_MAX ? COUPLING_MAX : ((val) < COUPLING_MIN ? COUPLING_MIN : (val)))
#define getCouplingUFLB()               CAPCON2.UFLB
#define setCouplingUFLB(val)            CAPCON2.UFLB  = ((val) > COUPLING_MAX ? COUPLING_MAX : ((val) < COUPLING_MIN ? COUPLING_MIN : (val)))
#define getCouplingLFUB()               CAPCON2.LFUB
#define setCouplingLFUB(val)            CAPCON2.LFUB  = ((val) > COUPLING_MAX ? COUPLING_MAX : ((val) < COUPLING_MIN ? COUPLING_MIN : (val)))
#define getFrontEndAnalogFTune()        ANALOG.FANAF
#define setFrontEndAnalogFTune(val)     ANALOG.FANAF = ((val) > ANALOG_MAX ? ANALOG_MAX : ((val) < ANALOG_MIN ? ANALOG_MIN : (val)))
#define getFrontEndAnalogQTune()        ANALOG.FANAQ
#define setFrontEndAnalogQTune(val)     ANALOG.FANAQ = ((val) > ANALOG_MAX ? ANALOG_MAX : ((val) < ANALOG_MIN ? ANALOG_MIN : (val)))
#define getBackEndAnalogFTune()         ANALOG.BANAF
#define setBackEndAnalogFTune(val)      ANALOG.BANAF = ((val) > ANALOG_MAX ? ANALOG_MAX : ((val) < ANALOG_MIN ? ANALOG_MIN : (val)))
#define getBackEndAnalogQTune()         ANALOG.BANAQ
#define setBackEndAnalogQTune(val)      ANALOG.BANAQ = ((val) > ANALOG_MAX ? ANALOG_MAX : ((val) < ANALOG_MIN ? ANALOG_MIN : (val)))
#define turnRFSwitchOn()                setPinLow(RFSW);  DEBUG.RFOn = 0
#define turnRFSwitchOff()               setPinHigh(RFSW); DEBUG.RFOn = 1
#define isRFSwitchOn()                  !DEBUG.RFOn
#define printFrontEndStatus()           (isFrontEndEnabled() ? txStrUART1("Front-End Enabled\r\n") : txStrUART1("Front-End Disabled\r\n"))
#define printFrontEndADStatus()         (isFrontEndADEnabled() ? txStrUART1("Front-End AD Enabled\r\n") : txStrUART1("Front-End AD Disabled\r\n"))
#define printFrontEndFDStatus()         (isFrontEndFDEnabled() ? txStrUART1("Front-End FD Enabled\r\n") : txStrUART1("Front-End FD Disabled\r\n"))
#define printFrontEndAD()               strPopulate16Bit(strFrontEndAD,FrontEndAD,'#',4)
#define printFrontEndNonOsc()           strPopulate16Bit(strFrontEndNonOsc,FrontEndNonOsc,'#',4)
#define printFrontEndFCnt()             strPopulate16Bit(strFrontEndFCnt,FrontEndFCnt,'#',4)
#define printFrontEndDigitalQTune()     strPopulate16Bit(strFrontEndDigitalQTune,getFrontEndDigitalQTune(),'#',2)
#define printFrontEndAnalogQTune()      strPopulate16Bit(strFrontEndAnalogQTune,getFrontEndAnalogQTune(),'#',4)
#define printFrontEndDigitalFTune()     strPopulate16Bit(strFrontEndDigitalFTune,getFrontEndDigitalFTune(),'#',3)
#define printFrontEndAnalogFTune()      strPopulate16Bit(strFrontEndAnalogFTune,getFrontEndAnalogFTune(),'#',4)
```

```
#define printBackEndStatus()              (isBackEndEnabled() ? txStrUART1("Back-End Enabled\r\n") : txStrUART1("Back-End Disabled\r\n"))
#define printBackEndADStatus()            (isBackEndADEnabled() ? txStrUART1("Back-End AD Enabled\r\n") : txStrUART1("Back-End AD Disabled\r\n"))
#define printBackEndFDStatus()            (isBackEndFDEnabled() ? txStrUART1("Back-End FD Enabled\r\n") : txStrUART1("Back-End FD Disabled\r\n"))
#define printBackEndAD()                   strPopulate16Bit(strBackEndAD,BackEndAD,'#',4)
#define printBackEndNonOsc()               strPopulate16Bit(strBackEndNonOsc,BackEndNonOsc,'#',4)
#define printBackEndFCnt()                 strPopulate16Bit(strBackEndFCnt,BackEndFCnt,'#',4)
#define printBackEndDigitalQTune()         strPopulate16Bit(strBackEndDigitalQTune,getBackEndDigitalQTune(),'#',2)
#define printBackEndAnalogQTune()          strPopulate16Bit(strBackEndAnalogQTune,getBackEndAnalogQTune(),'#',4)
#define printBackEndDigitalFTune()         strPopulate16Bit(strBackEndDigitalFTune,getBackEndDigitalFTune(),'#',3)
#define printBackEndAnalogFTune()          strPopulate16Bit(strBackEndAnalogFTune,getBackEndAnalogFTune(),'#',4)
#define printCouplingUpper()               strPopulate16Bit(strCouplingUpper,CouplingUpper,'#',2)
#define printCouplingLower()               strPopulate16Bit(strCouplingLower,CouplingLower,'#',2)
#define printCouplingUFLB()                strPopulate16Bit(strCouplingUFLB,CouplingUFLB,'#',2)
#define printCouplingLFUB()                strPopulate16Bit(strCouplingLFUB,CouplingLFUB,'#',2)
#define printRFSwitchStatus()              (isRFSwitchOn() ? txStrUART1("RF Switch On\r\n") : txStrUART1("RF Switch Off\r\n"))
#define printCenterFreq()                  strPopulate16Bit(strCenterFreq,CenterFreq,'#',4)
#define printFreqTol()                     strPopulate16Bit(strFreqTol,FreqTol,'#',3)
#define printFrontEndADThresh1()           strPopulate16Bit(strFrontEndADThresh1,FrontEndADThresh1,'#',3)
#define printFrontEndADThresh2()           strPopulate16Bit(strFrontEndADThresh2,FrontEndADThresh2,'#',3)
#define printFrontEndQOffset()             strPopulate16Bit(strFrontEndQOffset,FrontEndQOffset,'#',2)
#define printFrontEndQBackOff()            strPopulate16Bit(strFrontEndQBackOff,FrontEndQBackOff,'#',2)
#define printFrontEndFOffset()             strPopulate16Bit(strFrontEndFOffset,FrontEndFOffset,'#',2)
#define printBackEndADThresh1()            strPopulate16Bit(strBackEndADThresh1,BackEndADThresh1,'#',3)
#define printBackEndADThresh2()            strPopulate16Bit(strBackEndADThresh2,BackEndADThresh2,'#',3)
#define printBackEndQOffset()              strPopulate16Bit(strBackEndQOffset,BackEndQOffset,'#',2)
#define printBackEndQBackOff()             strPopulate16Bit(strBackEndQBackOff,BackEndQBackOff,'#',2)
#define printBackEndFOffset()              strPopulate16Bit(strBackEndFOffset,BackEndFOffset,'#',2)

/* Function Prototypes */
void initFilter();
void printFilterOptions(void);
void printAlgorithmOptions(void);
void updateFilterData(void);
void programFilter(void);
void prgmDelay(void);
void updateAnalogTuning(void);
void algorithm(void);
void coarseFrontEndFTune(void);
void coarseBackEndFTune(void);
int getFrontEndFrequency(void);
int getBackEndFrequency(void);
void findFrontEndCriticalOsc(void);
void findBackEndCriticalOsc(void);

#endif

/* End of File */
```

QEFILTER.C

```
/*******************************************************************************
 * Filename:    qefilter.c
 * Date:        June 2010
 * Compiler:    C30
 * Author:      Joel Schonberger
 * Company:     Kansas State University
 * Department:  Electrical & Computer Engineering
 * Research:    500 MHz Two-Pole Q-Enhanced Filter Tuning Algorithm
 * Description: This file houses the functions needed to implement the QE Filter
 *              tuning algorithm.
 *******************************************************************************/
#include "main.h"

/* Global Variables */
char strCenterFreq[]           = "Center Frequency: ###.# MHz\r\n";
char strFreqTol[]              = "Frequency Tolerance: ##.# MHz\r\n";
char strFrontEndADThresh1[]    = "Front-End AD Threshold 1: ###\r\n";
char strFrontEndADThresh2[]    = "Front-End AD Threshold 2: ###\r\n";
char strFrontEndQOffset[]      = "Front-End Q-Offset: ##\r\n";
char strFrontEndQBackOff[]     = "Front-End Q-BackOff: ##\r\n";
char strFrontEndFOffset[]      = "Front-End F-Offset: ##\r\n";
char strBackEndADThresh1[]     = "Back-End AD Threshold 1: ###\r\n";
char strBackEndADThresh2[]     = "Back-End AD Threshold 2: ###\r\n";
char strBackEndQOffset[]       = "Back-End Q-Offset: ##\r\n";
char strBackEndQBackOff[]      = "Back-End Q-BackOff: ##\r\n";
char strBackEndFOffset[]       = "Back-End F-Offset: ##\r\n";
char strCouplingUpper[]        = "Coupling Upper: ##\r\n";
char strCouplingLower[]        = "Coupling Lower: ##\r\n";
char strCouplingUFLB[]         = "Coupling UFLB: ##\r\n";
char strCouplingLFUB[]         = "Coupling LFUB: ##\r\n";
char strFrontEndAD[]           = "Front-End Amp Detector: ####\r\n";
char strFrontEndNonOsc[]       = "Front-End Non-Osc: ####\r\n";
char strFrontEndFCnt[]         = "Front-End Freq Count: ###.# MHz\r\n";
char strFrontEndDigitalQTune[] = "Front-End Digital Q-Tune: ##\r\n";
char strFrontEndAnalogQTune[]  = "Front-End Analog Q-Tune: ####\r\n";
char strFrontEndDigitalFTune[] = "Front-End Digital F-Tune: ###\r\n";
char strFrontEndAnalogFTune[]  = "Front-End Analog F-Tune: ####\r\n";
char strBackEndAD[]            = "Back-End Amp Detector: ####\r\n";
char strBackEndNonOsc[]        = "Back-End Non-Osc: ####\r\n";
char strBackEndFCnt[]          = "Back-End Freq Count: ###.# MHz\r\n";
char strBackEndDigitalQTune[]  = "Back-End Digital Q-Tune: ##\r\n";
char strBackEndAnalogQTune[]   = "Back-End Analog Q-Tune: ####\r\n";
```

```
char strBackEndDigitalFTune[]  = "Back-End Digital F-Tune: ###\r\n";
char strBackEndAnalogFTune[]   = "Back-End Analog F-Tune: ####\r\n";

int FrontEndAD, BackEndAD, FrontEndFCnt, BackEndFCnt, FrontEndNonOsc, BackEndNonOsc;
int PrevFrontEndDigitalQTune, PrevBackEndDigitalQTune;
int filterData[4];
int CenterFreq, FreqTol;
int FrontEndADThresh1, FrontEndADThresh2, FrontEndQOffset, FrontEndQBackOff, FrontEndFOffset;
int BackEndADThresh1, BackEndADThresh2, BackEndQOffset, BackEndQBackOff, BackEndFOffset;
int CouplingUpper, CouplingLower, CouplingUFLB, CouplingLFUB;

struct {
    unsigned int en:1;
    unsigned int fTune:8;
    unsigned int qTune:6;
    unsigned int ADen:1;
} FENDCON;

struct {
    unsigned int en:1;
    unsigned int fTune:8;
    unsigned int qTune:6;
    unsigned int ADen:1;
} BENDCON;

struct {
    unsigned int upper:5;
    unsigned int lower:5;
    unsigned int FDFen:1;
} CAPCON1;

struct {
    unsigned int UFLB:5;
    unsigned int LFUB:5;
    unsigned int FDBen:1;
} CAPCON2;

struct {
    unsigned int FANAF:10;
    unsigned int FANAQ:10;
    unsigned int BANAF:10;
    unsigned int BANAQ:10;
} ANALOG;

struct {
    unsigned int ADFen:1;
    unsigned int FDFen:1;
    unsigned int ADBen:1;
    unsigned int FDBen:1;
    unsigned int RFOn:1;
} DEBUG;

extern const char *console_str_sep;
extern int _PrintFilterSettings;

/*******************************************************************************
 * Function:    initFilter
 * Parameters:  void
 * Return:      void
 * Description: Initializes the Filter Controls to their minima and disables all
 *              of the enable variables.
 *******************************************************************************/
void initFilter(void)
{
    disableFrontEnd();
    setFrontEndDigitalFTune(FTUNE_DIG_MIN);
    setFrontEndDigitalQTune(QTUNE_DIG_MIN);
    disableFrontEndAD();
    disableFrontEndFD();

    disableBackEnd();
    setBackEndDigitalFTune(FTUNE_DIG_MIN);
    setBackEndDigitalQTune(QTUNE_DIG_MIN);
    disableBackEndAD();
    disableBackEndFD();

    setCouplingUpper(COUPLING_MIN);
    setCouplingLower(COUPLING_MIN);
    setCouplingUFLB(COUPLING_MIN);
    setCouplingLFUB(COUPLING_MIN);

    setFrontEndAnalogFTune(ANALOG_MIN);
    setFrontEndAnalogQTune(ANALOG_MIN);
    setBackEndAnalogFTune(ANALOG_MIN);
    setBackEndAnalogQTune(ANALOG_MIN);

    DEBUG.ADFen = 0;
    DEBUG.FDFen = 0;
    DEBUG.ADBen = 0;
    DEBUG.FDBen = 0;
    DEBUG.RFOn = 1;
}

/*******************************************************************************
 * Function:    printFilterOptions
 * Parameters:  void
 * Return:      void
 * Description: Prints each of the the filter options to the Console.
 *******************************************************************************/
void printFilterOptions(void)
{
    printSeperator();
    printFrontEndStatus();
    printFrontEndADStatus();
    printFrontEndFDStatus();
    printFrontEndDigitalQTune();
    //printFrontEndAnalogQTune();
    printFrontEndDigitalFTune();
    //printFrontEndAnalogFTune();
    printBackEndStatus();
    printBackEndADStatus();
    printBackEndFDStatus();
    printBackEndDigitalQTune();
    //printBackEndAnalogQTune();
    printBackEndDigitalFTune();
    //printBackEndAnalogFTune();
    printCouplingUpper();
    printCouplingLower();
    printCouplingUFLB();
```

```
    printCouplingLFUB();
    printRFSwitchStatus();
    printSeperator();
}

/*******************************************************************************
 * Function:    printAlgorithmOptions
 * Parameters:  void
 * Return:      void
 * Description: Prints each of the the algorithm options to the console.
 ******************************************************************************/
void printAlgorithmOptions(void)
{
    printSeperator();
    printCenterFreq();
    printFreqTol();
    printFrontEndADThresh1();
    printFrontEndADThresh2();
    printFrontEndQOffset();
    printFrontEndQBackOff();
    printFrontEndFOffset();
    printBackEndADThresh1();
    printBackEndADThresh2();
    printBackEndQOffset();
    printBackEndQBackOff();
    printBackEndFOffset();
    printCouplingUpper();
    printCouplingLower();
    printCouplingUFLB();
    printCouplingLFUB();
    printSeperator();
}

/*******************************************************************************
 * Function:    updateFilterData
 * Parameters:  void
 * Return:      void
 * Description: Formats and stores the filter options to be programmed.
 ******************************************************************************/
void updateFilterData(void)
{
    filterData[0] = ((CAPCON2.FDBen & 1) << 10)             |
                    ((CAPCON2.UFLB & COUPLING_MAX) << 5 )   |
                    (CAPCON2.LFUB & COUPLING_MAX);
    filterData[1] = ((CAPCON1.FDFen & 1) << 10)             |
                    ((CAPCON1.lower & COUPLING_MAX) << 5)   |
                    (CAPCON1.upper & COUPLING_MAX);
    filterData[2] = ((BENDCON.ADen & 1) << 15)              |
                    ((BENDCON.qTune & QTUNE_DIG_MAX) << 9)  |
                    ((BENDCON.fTune & FTUNE_DIG_MAX) << 1)  |
                    (BENDCON.en & 1);
    filterData[3] = ((FENDCON.ADen & 1) << 15)              |
                    ((FENDCON.qTune & QTUNE_DIG_MAX) << 9)  |
                    ((FENDCON.fTune & FTUNE_DIG_MAX) << 1)  |
                    (FENDCON.en & 1);
}

/*******************************************************************************
 * Function:    programFilter
 * Parameters:  void
 * Return:      void
 * Description: The filter is programmed by transmitting the 64 programming bits to
 *              the serial-to-parallel register of the filter via SPI.  Once all 64
 *              bits are transmitted, the latch line is raised, a clock pulse is
 *              generated and the latch line is lowered storing the bits.
 ******************************************************************************/
void programFilter(void)
{
    updateFilterData();

    setPinLow(FILTER_CLK);
    setPinLow(FILTER_DATA);
    setPinLow(FILTER_LATCH);
    enableSPI2();
    writeSPI2(filterData[0]);
    writeSPI2(filterData[1]);
    writeSPI2(filterData[2]);
    writeSPI2(filterData[3]);
    disableSPI2();

    setFilterLatchAsOutput();
    setFilterDataAsOutput();
    setFilterClkAsOutput();
    setPinLow(FILTER_LATCH);
    setPinLow(FILTER_CLK);
    prgmDelay();
    setPinHigh(FILTER_LATCH);
    prgmDelay();
    setPinHigh(FILTER_CLK);
    prgmDelay();
    setPinLow(FILTER_DATA);
    setPinLow(FILTER_LATCH);
    prgmDelay();
    setPinLow(FILTER_CLK);
    prgmDelay();
}

/*******************************************************************************
 * Function:    prgmDelay
 * Parameters:  void
 * Return:      void
 * Description: A simplistic delay used to ensure timing requirements are met during
 *              the filter programming process.
 ******************************************************************************/
void prgmDelay(void)
{
    Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop();
    Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop(); Nop();
}
```

```
/*******************************************************************************
 * Function:    updateAnalogTuning
 * Parameters:  void
 * Return:      void
 * Description: Programs the DACS with the appropriate current Analog F-Tune
 *              and Q-Tune values.
 *******************************************************************************/
void updateAnalogTuning(void)
{
    programFrontEndDAC(loadInputRegA(getFrontEndAnalogFTune()));
    programFrontEndDAC(loadInputRegB(getFrontEndAnalogQTune()));
    programFrontEndDAC(loadDACRegsABUpdateOutputsAB());
    programBackEndDAC(loadInputRegA(getBackEndAnalogQTune()));
    programBackEndDAC(loadInputRegB(getBackEndAnalogFTune()));
    programBackEndDAC(loadDACRegsABUpdateOutputsAB());
}

/*******************************************************************************
 * Function:    algorithm
 * Parameters:  void
 * Return:      void
 * Description: Implementation of the Two-Pole Tuning Algorithm.
 *******************************************************************************/
void algorithm(void)
{
    turnRFSwitchOff();
    enableFrontEnd();    // Enable Front-End
    disableFrontEndAD(); // Disble Front-End Amplitude Detector
    disableFrontEndFD(); // Disable Front-End Frequency Divider
    enableBackEnd();     // Enable Back-End
    disableBackEndAD();  // Disable Back-End Amplitude Detector
    disableBackEndFD();  // Disable Back-End Frequency Divider

    coarseFrontEndFTune();
    coarseBackEndFTune();

    setCouplingUpper(CouplingUpper);
    setCouplingLower(CouplingLower);
    setCouplingUFLB(CouplingUFLB);
    setCouplingLFUB(CouplingLFUB);
    programFilter();

    turnRFSwitchOn();    // Algorithm is Complete, so Turn on RF Switch

    if ( _PrintFilterSettings )
    {
        printFilterOptions();
        _PrintFilterSettings = FALSE;
    }
}

/*******************************************************************************
 * Function:    coarseFrontEndFTune
 * Parameters:  void
 * Return:      void
 * Description: Tunes the Front-End to the desired center frequency within a frequency
 *              tolerance.  Once within the frequency tolerance the frequency controls
 *              are ajusted and a comparison is done on which setting brought the pole
 *              closer to the desired center frequency.
 *******************************************************************************/
void coarseFrontEndFTune(void)
{
    int itt = 0;
    int prevFrontEndFCnt = 0;
    int prevFrontEndFCntDiff = 0, curFrontEndFCntDiff = 0;

    // Set frequency setting to controls mid-point
    setFrontEndDigitalFTune((FTUNE_DIG_MIN + FTUNE_DIG_MAX) >> 1);
    FrontEndFCnt = getFrontEndFrequency();

    while ( (FrontEndFCnt < (CenterFreq - FreqTol)) || (FrontEndFCnt > (CenterFreq + FreqTol)) )
    {
        if ( FrontEndFCnt < (CenterFreq - FreqTol) )
            decFrontEndDigitalFTune();
        else if ( FrontEndFCnt > (CenterFreq + FreqTol) )
            incFrontEndDigitalFTune();

        FrontEndFCnt = getFrontEndFrequency();
        if ( ++itt > MAX_FREQTUNE_ITTS )
        {
            txStrUART1("Max Front-End Frequency Tune Itterations Exceeded...\r\n");
            return;
        }
    }
    // Frequency within Tolerance now find Closest Setting
    prevFrontEndFCnt = FrontEndFCnt;
    prevFrontEndFCntDiff = absDiff(CenterFreq,prevFrontEndFCnt);
    if ( FrontEndFCnt < CenterFreq )
    {
        decFrontEndDigitalFTune();
        FrontEndFCnt = getFrontEndFrequency();
        curFrontEndFCntDiff = absDiff(CenterFreq,FrontEndFCnt);
        if ( curFrontEndFCntDiff > prevFrontEndFCntDiff )
        {
            incFrontEndDigitalFTune();
            programFilter();
        }
    }
    else if ( FrontEndFCnt > CenterFreq )
    {
        incFrontEndDigitalFTune();
        FrontEndFCnt = getFrontEndFrequency();
        curFrontEndFCntDiff = absDiff(CenterFreq,FrontEndFCnt);
        if ( curFrontEndFCntDiff > prevFrontEndFCntDiff )
        {
            decFrontEndDigitalFTune();
            programFilter();
        }
    }

    #if _DEBUG_ALGORITHM_ == 1
        txStrUART1("----Freq Tune Essentially Done---\r\n");
        printFrontEndDigitalFTune();
        printFrontEndDigitalQTune();
    #endif

    // Back-Off Q-Enhancement by Set Amount
    setFrontEndDigitalQTune(getFrontEndDigitalQTune() - FrontEndQBackOff);
    #if _DEBUG_ALGORITHM_ == 1
        txStrUART1("----Do Q Back-Off for BW---\r\n");
        printFrontEndDigitalQTune();
```

```
    #endif

    // Ensure Filter is Not Oscillating After Q-Enhancement Back-Off (Insufficient Back-Off)
    enableFrontEndAD();
    programFilter();
    FrontEndAD = readFrontEndAD();
    while ( FrontEndAD < (FrontEndNonOsc - FrontEndADThresh1) )
    {
        #if _DEBUG_ALGORITHM_ == 1
            txStrUART1("Still Oscillating after Back-Off...\r\n");
            printFrontEndDigitalQTune();
            printFrontEndAD();
        #endif

        decFrontEndDigitalQTune();
        programFilter();
        FrontEndAD = readFrontEndAD();

        #if _DEBUG_ALGORITHM_ == 1
            printFrontEndDigitalQTune();
            printFrontEndAD();
        #endif
    }
    disableFrontEndAD();

    // Counter the Frequency Shift Caused by Q-Enhancement Back-Off by Increasing Digital F-Tuning
    setFrontEndDigitalFTune(getFrontEndDigitalFTune() + FrontEndFOffset);
    programFilter();
}

/*******************************************************************************
 * Function:    coarseBackEndFTune
 * Parameters:  void
 * Return:      void
 * Description: Tunes the Back-End to the desired center frequency within a frequency
 *              tolerance.  Once within the frequency tolerance the frequency controls
 *              are ajusted and a comparison is done on which setting brought the pole
 *              closer to the desired center frequency.
 *******************************************************************************/
void coarseBackEndFTune(void)
{
    int itt = 0;
    int prevBackEndFCnt = 0;
    int prevBackEndFCntDiff = 0, curBackEndFCntDiff = 0;

    // Set frequency setting to controls mid-point
    setBackEndDigitalFTune((FTUNE_DIG_MIN + FTUNE_DIG_MAX) >> 1);
    BackEndFCnt = getBackEndFrequency();

    while ( (BackEndFCnt < (CenterFreq - FreqTol)) || (BackEndFCnt > (CenterFreq + FreqTol)) )
    {
        if ( BackEndFCnt < (CenterFreq - FreqTol) )
            decBackEndDigitalFTune();
        else if ( BackEndFCnt > (CenterFreq + FreqTol) )
            incBackEndDigitalFTune();

        BackEndFCnt = getBackEndFrequency();
        if ( ++itt > MAX_FREQTUNE_ITTS )
        {
            txStrUART1("Max Back-End Frequency Tune Itterations Exceeded...\r\n");
            return;
        }
    }
    // Frequency within Tolerance now find Closest Setting
    prevBackEndFCnt = BackEndFCnt;
    prevBackEndFCntDiff = absDiff(CenterFreq,prevBackEndFCnt);
    if ( BackEndFCnt < CenterFreq )
    {
        decBackEndDigitalFTune();
        BackEndFCnt = getBackEndFrequency();
        curBackEndFCntDiff = absDiff(CenterFreq,BackEndFCnt);
        if ( curBackEndFCntDiff > prevBackEndFCntDiff )
        {
            incBackEndDigitalFTune();
            programFilter();
        }
    }
    else if ( BackEndFCnt > CenterFreq )
    {
        incBackEndDigitalFTune();
        BackEndFCnt = getBackEndFrequency();
        curBackEndFCntDiff = absDiff(CenterFreq,BackEndFCnt);
        if ( curBackEndFCntDiff > prevBackEndFCntDiff )
        {
            decBackEndDigitalFTune();
            programFilter();
        }
    }

    #if _DEBUG_ALGORITHM_ == 1
        txStrUART1("----Freq Tune Essentially Done---\r\n");
        printBackEndDigitalFTune();
        printBackEndDigitalQTune();
    #endif

    // Back-Off Q-Enhancement by Set Amount
    setBackEndDigitalQTune(getBackEndDigitalQTune() - BackEndQBackOff);
    #if _DEBUG_ALGORITHM_ == 1
        txStrUART1("----Do Q Back-Off for BW---\r\n");
        printBackEndDigitalQTune();
    #endif

    // Ensure Filter is Not Oscillating After Q-Enhancement Back-Off (Insufficient Back-Off)
    enableBackEndAD();
    programFilter();
    BackEndAD = readBackEndAD();
    while ( BackEndAD < (BackEndNonOsc - BackEndADThresh1) )
    {
        #if _DEBUG_ALGORITHM_ == 1
            txStrUART1("Still Oscillating after Back-Off...\r\n");
            printBackEndDigitalQTune();
            printBackEndAD();
        #endif

        decBackEndDigitalQTune();
        programFilter();
        BackEndAD = readBackEndAD();

        #if _DEBUG_ALGORITHM_ == 1
            printBackEndDigitalQTune();
            printBackEndAD();
```

```
        #endif
    }
    disableBackEndAD();

    // Counter the Frequency Shift Caused by Q-Enhancement Back-Off by Increasing Digital F-Tuning
    setBackEndDigitalFTune(getBackEndDigitalFTune() + BackEndFOffset);
    programFilter();
}

/*****************************************************************************************
 * Function:    getFrontEndFrequency
 * Parameters:  void
 * Return:      int - Current Front-End Frequency Count
 * Description: Configures the filter so a reliable Front-End frequency count can
 *              be returned.
 *****************************************************************************************/
int getFrontEndFrequency(void)
{
    int fCnt;
    PrevBackEndDigitalQTune = getBackEndDigitalQTune();

    disableBackEndAD();  // Disable Back-End Amplitude Detector
    disableBackEndFD();  // Disable Back-End Frequency Divider
    setBackEndDigitalQTune(QTUNE_DIG_MIN); // Degrade Back-End Q-Enhancement

    enableFrontEndFD();  // Enable Front-End Frequency Divider
    findFrontEndCriticalOsc();
    fCnt = readFrontEndFD();
    disableFrontEndFD(); // Disable Front-End Frequency Divider

    // Remove Excess Q-Enhancement Needed for Dependable Frequency Divider Reading
    setFrontEndDigitalQTune(getFrontEndDigitalQTune() - FrontEndQOffset);

    // Restore Previous Back-End Digital Q-Tune Value
    setBackEndDigitalQTune(PrevBackEndDigitalQTune);
    programFilter();     // Apply Filter Settings

    return fCnt;
}

/*****************************************************************************************
 * Function:    getBackEndFrequency
 * Parameters:  void
 * Return:      int - Current Back-End Frequency Count
 * Description: Configures the filter so a reliable Back-End frequency count can
 *              be returned.
 *****************************************************************************************/
int getBackEndFrequency(void)
{
    int fCnt;
    PrevFrontEndDigitalQTune = getFrontEndDigitalQTune();

    disableFrontEndAD(); // Disable Front-End Amplitude Detector
    disableFrontEndFD(); // Disable Front-End Frequency Divider
    setFrontEndDigitalQTune(QTUNE_DIG_MIN); // Degrade Front-End Q-Enhancement

    enableBackEndFD();   // Enable Back-End Frequency Divider
    findBackEndCriticalOsc();
    fCnt = readBackEndFD();
    disableBackEndFD();  // Disable Back-End Frequency Divider

    // Remove Excess Q-Enhancement Needed for Dependable Frequency Divider Reading
    setBackEndDigitalQTune(getBackEndDigitalQTune() - BackEndQOffset);

    // Restore Previous Front-End Digital Q-Tune Value
    setFrontEndDigitalQTune(PrevFrontEndDigitalQTune);
    programFilter();     // Apply Filter Settings

    return fCnt;
}

/*****************************************************************************************
 * Function:    findFrontEndCriticalOsc
 * Parameters:  void
 * Return:      void
 * Description: Sets Front-End Q-Enhancement to 0 and reads the Amplitude Detector
 *              to determine the Non-Oscillation reading.   It then increases
 *              Q-Enhancement until the Amplitude Detector reading drops below
 *              the Non-Oscillation reading minus a set threshold value indicating that
 *              the Front-End is oscillating.  To ensure that a valid Frequency Divider
 *              reading is obtainable, the Q-Enhancement is increased by a set offset.
 *****************************************************************************************/
void findFrontEndCriticalOsc(void)
{
    int prevFrontEndDigitalQTune = -1;
    FrontEndNonOsc = ANALOG_MIN;
    FrontEndAD = ANALOG_MAX;

    setFrontEndDigitalQTune(QTUNE_DIG_MIN); // Set Front-End Q-Enhancment to Minimum
    enableFrontEndAD();                     // Enable Front-End Amplitude Detector
    programFilter();                        // Apply Filter Settings
    FrontEndNonOsc = readFrontEndAD();      // Store Front-End Amplitude Detector Reading

    #if _DEBUG_CRITICALOSC_ == 1
        printFrontEndNonOsc();
        printFrontEndDigitalQTune();
    #endif

    // Increase Q-Enhancement Until Front-End is Oscillating
    while ( FrontEndAD >= (FrontEndNonOsc - FrontEndADThresh1) )
    {
        if ( prevFrontEndDigitalQTune == getFrontEndDigitalQTune() )
        {
            txStrUART1("---> Could Not Obtain Front-End Critical Oscillation <---\r\n");
            return;
        }
        prevFrontEndDigitalQTune = getFrontEndDigitalQTune();

        incFrontEndDigitalQTune();          // Increment Front-End Digital Q-Tune
        programFilter();                    // Apply Filter Settings
        FrontEndAD = readFrontEndAD();      // Store Front-End Amplitude Detector Reading

        #if _DEBUG_CRITICALOSC_ == 1
            printFrontEndAD();
            printFrontEndDigitalQTune();
        #endif


    }
    disableFrontEndAD();                    // Disable Front-End Amplitude Detector

    // Ensure Oscillation for Dependable Frequency Divider Readings
```

```
    setFrontEndDigitalQTune(getFrontEndDigitalQTune() + FrontEndQOffset);
    programFilter();                        // Apply Filter Settings
}

/*******************************************************************************
 * Function:    findBackEndCriticalOsc
 * Parameters:  void
 * Return:      void
 * Description: Sets Back-End Q-Enhancement to 0 and reads the Amplitude Detector
 *              to determine the Non-Oscillation reading.   It then increases
 *              Q-Enhancement until the Amplitude Detector reading drops below
 *              the Non-Oscillation reading minus a set threshold value indicating that
 *              the Back-End is oscillating.  To ensure that a valid Frequency Divider
 *              reading is obtainable, the Q-Enhancement is increased by a set offset.
 *******************************************************************************/
void findBackEndCriticalOsc(void)
{
    int prevBackEndDigitalQTune = -1;
    BackEndNonOsc = ANALOG_MIN;
    BackEndAD = ANALOG_MAX;

    setBackEndDigitalQTune(QTUNE_DIG_MIN);
    enableBackEndAD();
    programFilter();
    BackEndNonOsc = readBackEndAD();

    #if _DEBUG_CRITICALOSC_ == 1
        printBackEndNonOsc();
        printBackEndDigitalQTune();
    #endif

    while ( BackEndAD >= (BackEndNonOsc - BackEndADThresh1) )
    {
        if ( prevBackEndDigitalQTune == getBackEndDigitalQTune() )
        {
            txStrUART1("---> Could Not Obtain Back-End Critical Oscillation <---\r\n");
            return;
        }
        prevBackEndDigitalQTune = getBackEndDigitalQTune();

        incBackEndDigitalQTune();
        programFilter();
        BackEndAD = readBackEndAD();

        #if _DEBUG_CRITICALOSC_ == 1
            printBackEndAD();
            printBackEndDigitalQTune();
        #endif
    }
    disableBackEndAD();

    // Ensure Oscillation for Dependable Frequency Divider Readings
    setBackEndDigitalQTune(getBackEndDigitalQTune() + BackEndQOffset);
    programFilter();
}

/* End of File */
```

Additional Files

All source code not documented in this appendix has been inherited from Appendix H.