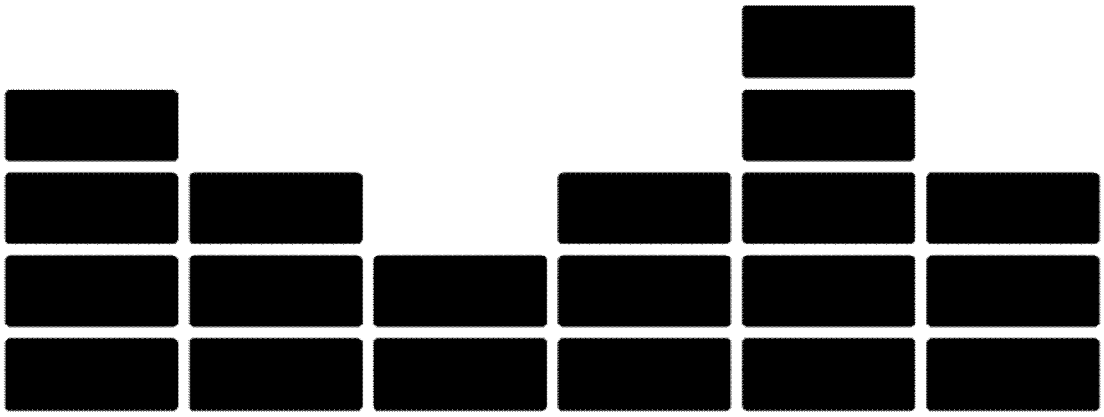
View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by DigitalCommons@CalPoly

10-Band Graphic Equalizer

By: Sean W. Michel



Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

2011

TABLE OF CONTENTS

Section *Page*

Acknowledgements iv

I. Introduction 1

II. Background 3

III. Requirements 4

IV. Design 5

Equalizer Circuitry 5

Display Circuitry 7

Microcontroller 10

Display 13

Power Supply 15

Microcontroller Code 16

V. Construction 21

Simulation 21

Printed Circuit Board 21

Assembly 23

VI. Testing 24

VII. Conclusion and Recommendations 26

VIII. Bibliography 30

Appendices

A. Schematics 31

B. Simulations 37

C. Parts List and Cost 39

D. Printed Circuit Board Artwork 42

E. Component Locations 44

F. Program Listing 46

LIST OF TABLES AND FIGURES

Tables *Page*

Table 1: Bill of Materials 39

Table 2: Project Costs 41

Figures

Figure 1: Equalizer Circuit 31

Figure 2: Inverting Summer 32

Figure 3: Bass and Treble Circuit 33

Figure 4: Display Band-pass Filter 34

Figure 5: Peak Detector Circuit 35

Figure 6: Full EQ Circuit 36

Figure 7: Equalizer Filter Response 37

Figure 8: EQ Filter Response at Single Frequency 37

Figure 9: Display Filter Response 38

Figure 10: LT Spice Simulation 38

Figure 11: Display PCB Layout 42

Figure 12: Equalizer PCB Layout 43

Figure 13: Display PCB Component Locations 44

Figure 14: Equalizer PCB Component Locations 45

Acknowledgements

I would like to thank my senior project advisor, Dr. Tina Smilkstein, for guiding me through the project and listening to my ideas. I would also like to thank my faculty advisor, Dr. John Oliver, for providing additional project support.

I. Introduction

This project consists of a 10-band, two channel graphic equalizer with a light emitting diode (LED) display. Left and right channels are operated on separately by ten sliders for each side. Moving the sliders up or down strengthens or weakens the energy in the corresponding frequency band. The sliders constitute part of an inverting band-pass filter that adds gain or attenuation over a certain range of frequencies, but has no effect outside the band. Each block of the equalizer circuitry acts on one of the ten frequency bands; an inverting summer with unity gain recombines the individual signals. The output of the inverting summer serves as both the speaker and display circuitry input so that the LED display will reflect any signal alterations. A repeated combination of a band-pass filter and peak detector for each frequency band comprises the display circuitry. The band-pass filter used here has a higher quality factor than those used in the equalizer, and removes any content outside the desired frequency band. Once it passes through the filter, the signal continues to the peak detector, where a capacitor follows the signal voltage until a maximum occurs. The capacitor holds the maximum until either a new maximum occurs or the transistor switches on and provides a discharge path to ground. An Atmega328P microcontroller samples the peak detector values of all ten bands and translates the recorded value to a number of LEDs to turn on. The microcontroller runs a real-time operating system (RTOS), using a separate task to sample each frequency band and write related data. After determining which LEDs to light up, the microcontroller transmits the corresponding data to a set of LED display drivers.

These drivers handle all display details, including pulse-width modulation (PWM). A 550 watt ATX computer power supply provides all power to the system.

Moving the equalizer sliders modifies whatever audio signal enters the circuit. With the ranges and frequency bands available, the system can negate audio distortion effects such as microphone response, instrument pick-ups, loudspeakers, and room acoustics. The system can also alter the signal to make it sound as if it were recorded or played in a particular environment or way, such as a large hall or over old-time radio.

Several design issues degraded the quality of the physical product. Poor trace routing during PCB layout resulted in constant interference or noise in the output signal. The display experienced an intermittent glitch that toggled it on and off, due to a poor quality push button. Misleading designations led to the purchase of logarithmic taper rotary potentiometers of limited use. Despite these problems, the final product worked more or less on the first try.

II. Background

Graphic equalizers (EQs) have been used since the 1920's to adjust audio recording sound levels. The term "graphic equalizer" comes from the graph of signal gain versus frequency. An equalized signal has a flat line with no gain. In the context of audio, "equalizer" does not actually mean flattening the signal as in other applications, but more generally the adjustment of frequency domain characteristics that result in a non-equalized signal. This type of signal modification may have practical or aesthetic purposes [1]. Each slider on an equalizer board either boosts (strengthens) or cuts (weakens) the intensity in a particular frequency band.

Audio recorded in a studio contains frequency distortions inherent to the recording process and the acoustics of the room. Large banks of equalizers restore the balance of the frequencies of the sound-reproduction to that of the original source [2]. During live performances, equalizers may be used to "correct the response of microphones, instrument pick-ups, loudspeakers, and hall acoustics" [3]. Graphic equalizers can also dramatically change how someone hears audio or music. Depending on the EQ settings, audio can distort to different acoustic settings. For example, boosting the midranges and cutting the low frequencies can alter music to sound like old-time radio. Typically, audio receivers in home sound systems have dials to adjust bass and treble (low and high frequencies) to a user's preference [1]. Typically, EQs control frequencies over a range of 20Hz to 20kHz, which is the range of human hearing [2].

III. Requirements

The project consists of a two channel (stereo) 10-band graphic equalizer with output display and bass and treble control. An audio signal from a computer or other multimedia device serves as the system input. Resistor potentiometer sliders either boost or cut the intensity in each corresponding band. Resistor potentiometer knobs do the same for signal bass (low frequencies) and treble (high frequencies). The equalizer circuitry sends the modified audio waveform to a set of speakers. An LED matrix display provides a visual representation of the signal amplitude in each frequency band examined. Using the equalizer circuit output as the display input ensures the display reflects any boost or cut alterations to the original signal. A microcontroller, running a real-time operating system, interprets the voltage levels from the peak detectors and translates the value to the correct display output. A push button allows a user to toggle the LED display on or off and a second button resets the microcontroller. Designing and manufacturing a printed circuit board utilizing the circuit design as a template presents a cleaner and more professional looking product.

IV. Design

Equalizer Circuitry

An audio signal from a computer or mobile device serves as the audio/voltage input to the equalizer. A series of 20 adjustable band-pass filters (ten per channel) and bass and treble filters (one each for both channels) perform all audio modifications separate from one another. Each frequency band filter (Figure 1) produces a modified, inverted version of the input signal. The filter outputs are recombined using an inverting summer with unity gain (Figure 2), which results in a non-inverted output waveform that sounds different from the original. As shown in Figure 7, these narrow band-pass filters boost or cut the amplitude in a specific frequency band while leaving all other content untouched. The capacitor placed across the terminals of the slide potentiometer acts as an open circuit, while the capacitor between the potentiometer wiper and the op-amp negative input acts as a short circuit [4]. This configurations allows a user to either boost audio content by moving a slider up (wiper moves to the left in Figure 1) or cut content by moving the slider down (wiper moves to the right in Figure 1). At frequencies lower than the center frequency, the capacitor between the slide potentiometer wiper and the op-amp inverting input acts like an open circuit. At frequencies higher than the center frequency, the capacitor between the potentiometer terminals acts as a short. As seen in Figure 8, the result is a bump or dip in the frequency versus gain graph, centered over a specified frequency band [4]. Each frequency band can attain a maximum gain of about ±12.5dB. Both the maximum op-

amp supply voltages and the resistance of the slide potentiometers limit the maximum attainable gain.

Bass and treble control (Figure 3) behaves much like the equalizer circuitry. The two capacitors behave as open circuits at low frequencies and shorts at high frequencies [4]. Any audio higher than the bass cutoff frequency and lower than the treble cutoff frequency has a gain of 0dB and remains unaltered. The op-amp behaves as an inverting amplifier, which makes it easy to integrate with the equalizer circuitry. Simply add the signal together with the equalizer band-pass outputs via the inverting summer circuit. For low frequency content, the virtual open circuits result in purely resistive amplifier feedback [4]. Moving the wiper of the bass resistor potentiometer to the left in Figure 3 decreases the resistance and results in gain. Moving the bass wiper to the right in Figure 3 increases amplifier resistance and attenuates the signal. At higher frequencies, the virtual short circuits provided by the capacitors results in shorting the bass potentiometer and connecting the treble potentiometer to the op-amp inverting input [4]. Moving the treble potentiometer wiper to the left in Figure 3 reduces the amplifier resistance and boosts the signal, while moving it to the right increases resistance and cuts the signal. In this design, the bass and treble cutoff frequencies are roughly 14Hz and 23kHz, respectively. Imperfect filter response causes the circuit to actually modify frequencies outside this range, but to a significantly lesser degree.

TL074 J-FET op-amps were selected for this project as they have high slew rate (rate of change), low noise and distortion, quad package availability and

relatively low cost. The high slew rate (13V/μs) means the chips operate at high speeds and keep up with any audio signal passed through them. Low noise and distortion (15nV/√Hz) means the chips will not introduce an appreciable amount of unwanted change into the audio signal. A quad package works well because each frequency band utilizes a total of four op-amps: one for the equalizer circuitry, one for the display band-pass filter and two for the peak detector.

Display Circuitry

Once the audio input signal passes through the equalizer circuitry, it continues to both the speakers and the equalizer display circuitry. This ensures the display reflects any alterations made by moving the slide potentiometers from their midpoints. In the display circuitry, a set of ten band-pass filters (Figure 4) split the signal into their respective frequency bands, invert it and add a gain of 20dB. The design of these filters comes from Texas Instruments Application Report SLOA093 [10]. Inverting the audio signal does not present a problem as this signal will not transmit to a speaker. Additionally, the audio signal positive and negative peaks are roughly equal in magnitude. Following the magnitude of the negative peaks produces peak detector voltage levels that are nearly identical to those if the positive peaks were followed. The filters in this section differ from those in the equalizer in that they allow the desired frequency to pass while removing all other spectral content. Comparing the graphs in Figure 7 and Figure 9 shows the differences between the equalizer and display filters. Each frequency band overlaps a small amount (about

3dB) with the adjacent bands. This seems undesirable at first glance, but actually provides better display results. Without overlap, only a very narrow frequency range will pass through to the peak detector and light up the LED display. Not all music will contain the specific required frequency for a long enough period to produce a visually interesting output at a certain band. Allowing the bands to overlap adds more content to the display and adds a feeling of transition between the bands.

A simple peak detector (Figure 5) follows the band-pass filter voltage output to the highest value and holds it until either the microcontroller samples it and toggles the reset pin, or a larger value occurs. A 1μF capacitor acts as the voltage storage element, holding the maximum voltage level seen between resets. Several capacitor sizes were tested during the prototyping phase, including 10μF, 1μF and 100nF. Circuit performance appeared unaffected by the capacitance used. Both voltage level and charge hold time were comparable between capacitor sizes. Ultimately, 1μF capacitors were chosen because the 10μF capacitors were more expensive and the 100nF capacitors felt wrong.

An nMOSFET serves as a switch between the charge capacitor and ground. Under normal operation, the microcontroller grounds the transistor gate, which breaks the capacitor discharge path. When the microcontroller sends a reset signal, the transistor turns on and provides a path for the capacitor to discharge. MOSFETs with low gate charge and low on-resistance were chosen to keep up with the relatively high-speed switching microcontroller pins. Changing these parameters would most

likely not affect circuit performance terribly much. The same holds true for the diode chosen; the 1N914 simply appeared in the original peak detector schematic.

Each peak detector contains a buffer stage that both isolates the 1µF charge capacitor and adds a gain of 2.64. A gain of 2.64 rests slightly higher than the midpoint of the ADC window used. It also produced the most visually appealing output when a .WAV file functioned as a voltage source in simulation. The peak detector used in this project tends to attenuate higher frequency content by a small amount. To combat this, each frequency band includes a different value resistor connected from the detector output to the buffer op-amp's inverting input. Prototyping the peak detector on a breadboard and passing through the circuit a sine wave with amplitude of one and a frequency equal to the frequency band (i.e. 32Hz, 64Hz, etc) determined the proper resistor values. Increasing the resistor magnitude until the output equaled 2.64 times the input provided resistor sizes.

Inserting a 5V Zener diode between the peak detector output and ground protects the microcontroller and the on-chip ADC. A 51Ω, ½ watt resistor between the buffer op-amp output and the peak detector output provides enough current to turn on the Zener diode when applying a minimum of 5V across the diode. The 51Ω resistor remains the only ½ watt resistor in the project; at a maximum peak detector output of 12V, seven of them dissipate across the resistor to lock the Zener diode at 5V. Using Ohm's Law, the maximum power through the 51Ω resistor equates to 0.96 watts. This discrepancy in power ratings was a design oversight. However, no resistor sustains current for more than a few dozen milliseconds before the peak detector

resets, so the components are unlikely to exceed maximum ratings. When the peak detector output falls below 5V, the Zener diode turns off and acts like an open circuit. The output can range anywhere from zero to five volts. Once the output reaches at least 5V, the Zener diode turns on and locks the output to five volts. This feature prevents any voltage higher than the ADC window from traveling to the microcontroller.

Close to a dozen peak detector circuits of varying complexities were simulated using LT Spice. Most either failed to hold maximum signal voltages on the charge capacitor or attenuated the high frequency signal content. The circuit chosen harbors neither of these drawbacks; it leaks current at a small pace and therefore holds a charge for a long time and attenuates the high signal frequency content by only a few hundredths of a volts. Additionally, 11 components, two of which are already included in the TL074 package, form the circuit. The advantages of this simple circuit make it more desirable than a complex one with several dozen components. Figure 6 shows the full equalizer and display circuitry.

Microcontroller

The Atmel Atmega328P microcontroller works for this project because it possesses a fairly large amount of Flash memory (32 Kbytes) and SRAM (2 Kbytes), comes in a convenient 28-pin dual-inline package and can easily use FreeRTOS. Of the 28 pins available, all but AREF and PC5 connect to devices, making the 328P a more efficient choice than the comparable 40-pin Atmega32. The internal RC

oscillator generates a microcontroller clock source of roughly 3.68MHz. Normally, a 16MHz external oscillator would generate a faster and more accurate clock source. However, the pins required to connect the extra circuitry control some of the peak detector reset pins. Since the project does not require precise timing and the pins are unavailable, the internal oscillator was deemed sufficient for this project.

A simple push button connected to pin PC6 resets the microcontroller when pressed. Under normal operation, PC6 connects to VCC through a 10kΩ pull-up resistor. When pressed, the reset button shorts the pin directly to ground. This resets the microcontroller and forces it to reload the program stored in Flash memory. As an added bonus, pressing the reset button gives the appearance of holding the current display output, almost like a screenshot.

An on-chip analog to digital converter (ADC) running at 115kHz samples peak detector voltage levels. The ADC uses its 5V supply as the reference voltage, and does not apply extra gain to the sampled waveform. Using an ADC reference voltage of five volts provides better resolution than one volt when following a band-pass filter with a gain of 20dB. Since the microcontroller only has five ADC channels and there are ten frequency bands, an analog multiplexor switches between the inputs to sample all bands. The multiplexor chosen for this project is an Analog Devices ADG406BNZ bi-directional 16:1 multiplexor that uses four select lines and one enable line. This particular chip provides enough input lines for all ten frequency bands and can operate at a wide range of single and dual supply voltages. Additionally, part samples were provided free of charge by Analog Devices. A low

pass filter (comprised of a 10μH inductor and a 100nF capacitor) quiets any noise or transients from the ADC supply voltage. This provides a steady input and reference voltage for the ADC to use, which in turn results in more accurate voltage conversions.

Once the ADC translates the sampled voltage into a digital representation, the microcontroller analyzes what range it falls in and writes the corresponding information to the display drivers. The range follows a linear distribution over the five volt reference level. There are nine possible display states, so the voltage between each state is approximately 0.56 volts. The digital representation of this voltage is a hex number between 0x000 and 0x3FF, where 0x000 corresponds to 0V and 0x3FF corresponds to 5V. The ranges used in the microcontroller code are predetermined hex numbers and not the voltages themselves. This method of calculating the number of LEDs to light up on the display saves time and power because the microcontroller does not need to convert the sampled value to a voltage first.

All large DIP packages sit in IC frames, which allows for easy removal should they cease to function or be needed elsewhere. The flexibility afforded by the frames justifies the extra few cents. Small DIP packages such as the TL074 are soldered directly to the printed circuit board.

A six pin ISP programming header provides access to on-the-fly microcontroller programming. This header connects to the MOSI, MISO, SCK, SS and reset microcontroller pins, as well as VCC and ground. When an external device

wishes to program the microcontroller, it pulls the reset pin low to disable the microcontroller and ready it for modification. The programmer can then transfer new program data over the serial communication port. The external device releases the rest pin when finished, allowing the microcontroller to resume operation. Atmel design document AVR042 [7] shows the correct pin connections for an ISP header.

Display

An 8x10 (eight rows, ten columns) common cathode LED matrix serves as the equalizer display. Two Maxim MAX7221CNG+ LED display drivers control which LEDs turn on based on the data received from the microcontroller. Each chip can drive a maximum of eight LED columns; the equalizer has ten. Cascading two drivers solves this issue; one chip controls the left half of the display while the other chip drives the right half. The drivers utilize pulse width modulation (PWM) to strobe the LED matrix columns at a pace faster than the eye can see. PWM limits the current drawn from the power supply and allows for easy adjustment of brightness levels. Assigning exactly half of the columns to each driver ensures that they strobe the columns at the same rate, which keeps the LEDs at the same brightness level. Should one driver control more columns than the other, the LEDs attached to it will appear dimmer because it must sweep more area in the same amount of time. For this project, the display drivers light up the LEDs at maximum intensity by setting the inter-digit blanking time to 31/32 of a cycle. The MAX7221 chips fully support SPI and handle all PWM.

The water clear LEDs were chosen over diffuse cover LEDs for several reasons. With a luminous intensity of 380mcd, the water clear package emits significantly brighter color than the 2mcd diffuse cover LEDs. Additionally, the water clear LEDs require a smaller voltage drop and less current to operate, which means they use less power. Also, they look more professional than similar packages with tinted coverings. Both types of LEDs have the same viewing angle. While they cost significantly more (6.5 cents versus 11 cents per LED), the benefits of water clear LEDs justify the extra price.

To set the current through each LED column, VCC connects to the ISET pin of the display driver through a single resistor. A combination of the LED forward voltage drop and required operating current determines the necessary size of this resistor. Plotting the common values provided in the MAX7221 datasheet allows for extrapolation of the resistance value. The LEDs used in this project require a forward drop of 2V at 20mA, which corresponds to an ISET resistance of about 37.4kΩ.

As recommended by the datasheet, a 10µF electrolytic and a 100nF ceramic capacitor decouple the positive voltage rail and ground. These decoupling capacitors sit directly next to the driver chips and reduce the effects of power supply ripple due to PWM, as well as any effects of wire inductance and electromagnetic interference. To further reduce these effects, separate 5V rails power the MAX7221 chips and the microcontroller.

A push button, connected to PD2 (external interrupt zero), disables the LED display. Pressing the button a second time re-enables the display. Much like the reset

button, the pin normally connects to VCC through a 10kΩ pull-up resistor. When pressed, the display on/off button shorts the microcontroller pin to ground. This generates an interrupt in the program code that sends a display on or off command to the display drivers. The MAX7221 chips still receive data while the LEDs are off, so the display will resume with current sound levels when re-enabled.

Power Supply

A 550 watt computer power supply (PSU) provides 5V, 12V and -12V rails at a current rating far above project requirements. An early project draft included a custom power supply, but this was ultimately too expensive to implement. A simply ATX computer supply costs only $22 and guarantees clean, constant voltage rails. In order to turn the supply on, the green wire (pin 16) must shorted to ground. Some PSUs require a 10Ω, 10 watt resister between the green wire and ground to supply any current, but the PSU used in this project does not. A 24-pin female Molex connector connects the PSU wires to the custom PCB. This allows for quick and clean wire disconnects, and avoids an unprofessional looking wire rats-nest. Unfortunately, the PSU includes a 60mm fan with distracting built in LEDs. Analog circuitry (±12V lines) has a dedicated ground wire, as does the microcontroller and display drivers (5V lines). Separating ground connections of analog and digital circuitry prevents high frequency noise from contaminating the output audio waveform.

Based on LT Spice simulations, the maximum current drawn by the ±12 volt rails equals about 800mA each. The power supply used in the project was chosen specifically because it can supply 1A at -12V. All other supplies examined were rated for 0.8A at -12V, which leaves little room for error. Should the system attempt to draw more than the maximum allowed current, system glitches may appear. The extra current rating leaves room to avoid any issues. Current ratings for 12V and 5V lines are large enough to not cause concern.

Microcontroller Code

The microcontroller runs a real-time operating system (RTOS), which allows for reusable, modular code and easy task and interrupt handling. This project uses a version of FreeRTOS that has been ported from the Atmega323 version to work on the Atmega328P. To achieve this, only a few lines of code need to change; these alterations were found on the AVR Freaks forum [5] and are listed below:

```
File: port.c
   portCLEAR_COUNTER_ON_MATCH changed from "0x08" to "(1<<WGM12)"
   portPRESCALE_64 changed from "0x03" to "((1<<CS11)|(1<<CS10))"
   portCOMPARE_MATCH_A_INTERRUPT_ENABLE changed from "0x10" to "(1<<OCIE1A)"
   references to "TIMSK" changed to "TIMSK1"
   references to "SIG_OUTPUT_COMPARE1A" changed to "TIMER1_COMPA_vect"
```

Additionally, FreeRTOS is configured to run in cooperative scheduler mode. This automatically eliminates the issue of preemptive task switching during data transmission. Cooperative mode also ensures that each task runs to completion before reentering the blocked state. When compiled, the application and FreeRTOS footprint size fills up 8042 bytes. This constitutes 24.5 percent of available Flash memory. Additionally, the estimated SRAM data contains 1843 bytes, which constitutes 90.0

percent of available space. Typically, 80 percent pushes the maximum allowed data size, as this estimate does not include any variables declared locally. However, each task uses only a single-byte variable, so the application for this project should not exceed volatile memory limits.

Applications written using C and FreeRTOS run a number of user created tasks, each with their own task definition. The code written for this project uses ten tasks—one for each frequency band—but only a single task definition. When created, each task receives a set of parameters that determines how it functions and what frequency band it belongs to. These parameters are passed to the task using a structure that contains the band number, the port letter and pin number that controls the peak detector reset, and the pin configuration to select an analog multiplexor channel. When the task runs, it disables interrupts using the taskENTER_CRITICAL API to avoid an external interrupt during SPI data transmission. Interrupts are re-enabled at the end of the task with the taskEXIT_CRITICAL API call. Once interrupts are disabled, pins C1, C2, C3 and C4 are configured to select the corresponding channel from the analog multiplexor. A dumb delay of ten milliseconds allows the multiplexor output to settle. The ADC then takes a sample of the channel by setting the ADSC bit of the ADC control register, delaying by the maximum conversion time of 260µs (as stated in the Atmega329P datasheet), then transfers the result to the corresponding slot in the voltage value array. Hardware clears the ADSC bit once the conversion completes, so there is no need to clear it manually. A series of IF/ELSE IF statements determine the LED display state

according to the sampled voltage level. A variable called *write_level* contains the hex code that determines which LEDs in a column turn on. This minimizes code redundancy by having only one SPI write function call instead of one in every IF statement. Once the microcontroller determines the number of LEDs in the column, it writes the correct data to the corresponding MAX7221 chip. Next, the microcontroller resets the band's peak detector by toggling a specific pin, which a SWITCH/CASE statement determines. Finally, a delay API places the task back in the blocked state for 2ms. This allows other tasks to run before the current task returns to the ready state. Increasing or decreasing the task block time changes how responsive the display appears. A smaller delay time may toggle the LEDs too quickly to see while a larger delay may cause the display to look slow and sluggish.

A second task and task definition exists for toggling the display on or off with a button press. This task only runs once the external interrupt vector *INT0_vect* gives the binary semaphore. Once the task takes the semaphore and runs once, it releases it and cannot run again until the interrupt gives the semaphore once more. As the display toggle button is non-ideal, pressing it oscillates the voltage applied to the microcontroller pin between VCC to ground before settling. To combat this, the task immediately enters the blocked state for 50ms as a button de-bounce. This prevents the task from running and resetting rapidly when pressing the button. Once the task continues running, it disables interrupts to prevent another button press while transmitting data to the display drivers. A SWITCH/CASE statement uses a binary variable *ext_int_data* to determine if the current display state. Once the task

determines the display status, it transmits the data required to toggle it to the opposite state. The task_EXITCRITICAL API re-enables interrupts at the end of the task.

All tasks definitions written for this project are reentrant. The requirements for reentrancy are:

- Code only uses data in automatic way
- Code uses static data in atomic way
- Code does not call any non-reentrant functions
- Code uses hardware in atomic way

By crudely disabling interrupts with the taskENTER_CRITICAL API call, nothing can interrupt the task and it therefore uses the static voltage array atomically. This also satisfies the hardware condition as the multiplexor and peak detector transistor switch are both controlled within the atomic block. Additionally, all other variables used either pass in from the parameter structure or exist locally. Therefore, the tasks only use data in an automatic way. Lastly, the code does not call any non-reentrant functions. The initialization functions are called before interrupts are enabled and can therefore be considered reentrant. Since all conditions are satisfied, the tasks are reentrant.

Writing data to the MAX7221 LED display drivers requires the use of the chip no-operation (no-op) codes. As the output of one chip connect to the input of the other, the no-op command serves to push data through one chip and into the next. Writing to the first diver in the chain requires sending two no-op codes (0x00) followed by the digit (column) number and segment code (number of LEDs). Writing to the second display chip follows a similar convention, except the column and segment data precedes the two no-op commands. Once the four commands have

finished transmitting, the microcontroller latches data in the display driver by disabling the chip select line.

V. Construction

Simulation

All circuit design and simulation work was completed using LT Spice IV. Components placed in the circuits used real world models of the devices actually used in the project. This gives a clearer picture of what to expect from circuit output when physically constructed. Both frequency and time domain circuit responses were examined. For the frequency domain, a signal with the parameter ".AC 1" as the input allows the program to generate a graph of frequency versus gain. This graph can help analyze filter design and demonstrate circuit response. For the time domain analysis, a .WAV file functions as the voltage input to the circuit. As audio signals are rarely pure waveforms, a realistic input to the system generates simulation results that are more useful when designing audio systems. See Figure 10 for an example simulation plot.

Printed Circuit Board

Printed circuit board design and layout was completed using CadSoft EAGLE. As the Spice design incorporated single op-amps and not the quad package TL074s, the circuit import function was not used. Instead, a schematic for a single band was generated and copied several times, with the appropriate values changed for each iteration. This new schematic was then used to create a printed circuit board layout, which appears in Figures 11 through 14. Again, the circuitry for a single band was laid out and copied for the remaining bands. The rest of the components were placed

according to nearest logical neighbor available space. The power connector and 3.5mm jacks needed to stay near the edge of the board, preferably on the left. The display drivers were placed above the microcontroller and analog multiplexor so they rest closer to the display connector pins. Since the peak detectors all feed into the analog multiplexor, it sits on the right of the microcontroller.

All power rails were routed by hand. This was done to avoid daisy-chaining the power and ground connections between components, which can cause line noise and interference. Each part connects to a power or ground rail that runs across the board. All other traces were routed with a second program called FreeRouting. This application works by importing a board layout from Eagle and simply routing all un-routed traces. After finding a possible way to route all traces, the program attempts to optimize the layout through an iterative process. This process will run until there are no remaining changes that reduce total trace distance. However, the amount of time to reach this point boarders on absurdly long. After a few hours, any optimizations made will result in such minor improvements that stopping the application constitutes a more efficient use of time. Once the process has halted, the layout can export back into Eagle for final review and processing.

Purchasing two boards with different patterns always costs more money than two boards with a single pattern. Therefore, two identical printed circuit boards, one for each channel, saves a fair amount of money. Additionally, a smaller microcontroller with less memory can sample peak detector values on each board. Some alterations were required to use a single board layout with one power and audio

input. Two three-pin connectors jump either the left or right audio signal to the equalizer input or output. The unconnected pins connect to the input and output of the second board. This way, each board can operate on the left or right audio signal. Power forwards to the second board through simple jumper wires. The push buttons also connect between boards; the microcontroller pins jump together so that button presses short them both to ground.

Assembly

Physical construction of the equalizer and display involved soldering every component to the custom boards by hand. Since this method of assembly was decided before PCB design, mostly through-hole components comprise the system. This allows for simpler construction with less chance of error. After each component was soldered to the board, the connections were checked with a multimeter continuity test. Touching one multimeter lead to the component leg and one to an electrically connected hole produces a beep noise if the connection is good. Otherwise, the multimeter will not beep. This ensures that every solder joint has a good connection, and eliminates tedious checking should something fail to work after all components are soldered.

VI. Testing

Each phase of the project underwent extensive testing before continuing to the next. Many hours of LT Spice simulations ensured the circuit would operate as intended. Using .WAV files as circuit voltage input allowed for simulation with a real audio waveform. The .WAV output feature provided tactile results, as it recorded the output voltage as a playable music file.

Eagle design rules and error checking features verified the printed circuit board design. An additional visual examination verified that each trace connected the proper component pins or leads. Consulting all part datasheets during the visual inspection ensured the connections matched the schematic. Every part soldered during project assembly was continuity tested using a multimeter. For this type of test, the multimeter beeps to signal an electrical connection exists. This verification methodology guarantees a working connection and prevents a testing nightmare.

After hardware construction, an audio signal was applied to the input while the sliders were left in the middle position. This tests that the audio makes it through the circuit. Next, the sliders were moved one by one to verify their functionality. Display and microcontroller code were first implemented and tested using an STK500 development board. This removes extra variables from the equation and allows for testing of code functionality only. Once the code worked on an external board, the microcontrollers were moved to the equalizer boards. A few code adjustments brought the equalizer display to full functionality.

During the microcontroller code testing phase, a glitch occurred that toggled the display on or off intermittently. Several code modifications were tried in an attempt to solve this issue, but it was ultimately determined that the button itself was generating the error. This was verified by disabling external interrupts in the code, but leaving the display toggle function intact. The display continued uninterrupted for several hours with no trace of the glitch.

VII. Conclusion and Recommendations

In the final product, an annoying buzzing and clicking noise emanates from the speakers. The constant, high switching rate of the digital circuitry causes this interference. The quick switching between high and low states generates both ripple in the ground levels and electromagnetic interference. The PCB design separates the analog, microcontroller and display driver ground connections in an effort to minimize this type of interference. However, all ground wires connect to the same place in the power supply (hence the term *common ground*), so audible high frequency digital interference still exists in the output waveform. The LED display driver connections pass near the positive and negative 12 volt rails, as well other analog device traces. The PWM of the drivers means the display connections are always switching high to low and vice versa, which causes inductance in nearby lines. This switching provides an additional source of interference. Leaving a large amount of space between analog and digital traces reduces switching-induced noise. Adding ground lines that parallel any digital traces could also reduce noise. Decoupling capacitors between positive and negative rails also reduces noise by smoothing supply ripple caused by digital circuitry. Two 10μF decoupling capacitors were placed near the MAX7221 display drivers in an attempt to minimize supply ripple, but the other glaring design issues make them a moot point. Further aggravating the noise problem, the analog multiplexor was powered from the 12V and analog ground rails. As these analog connections pass near several digital lines, the constant select line switching may cause additional interference. A final contributor to the

interference problem, the transistors in the peak detector constantly switch on and off and connect to the same analog ground as the input and output signals. Separating ground lines and more intelligent trace routing and component placement will significantly reduce the types of interference experienced in this project.

The six pin ISP header was added to the design after most of the PCB layout was completed. As a result, the only edge space available for it was on the opposite side of the board from the microcontroller. As nothing connects to the ISP head most of the time, this would not normally constitute an issue. However, when a programmer pushes data to the microcontroller, a high pitch static noise appears at the audio output. While not a huge problem, this source of potential interference is worth mentioning.

Incorrect assumptions lead to the use of resistors with ½ watt power ratings when one watt ratings were required. As the maximum voltage output of the peak detector roughly equals the 12V supply, the maximum voltage dissipated across the resistor in series with the Zener diode should be 7V, not 5V as originally thought. However, should this particular resistor exceed the ½ watt power rating, it will not sustain the current for long. Permanent damage is therefore unlikely.

Due to an incorrect datasheet listing, the audio input pins were connected backwards. The intended left channel board operated on the right channel while the right hand board operated on the left channel signal. This mix up rendered useless the two pins designated for jumping input and output signals to and from the right side board. Jumping the right side pins together on the three-pin channel connector of the

left board fixed the left channel input and output. Jumping the left side pins from the same header directly to the input pin of the header on the ride hand board fixed the right channel.

Occasionally, the display toggle button shorts itself to ground, triggering an interrupt that disables the display. Waiting a sufficient amount of time or pressing the button re-enables the display. Testing the buttons with a multimeter shows that they appear to work as intended. However, disabling external interrupts (but not interrupts or the task code) eliminates the glitch. Therefore, the button itself must cause the issue. This may stem from poor quality manufacturing since the push-buttons were the cheapest available. Incorporating a higher quality button should fix the problem.

Bass and treble controls work in a limited capacity. The potentiometers ordered have an audio taper which means they conform to a logarithmic function. Rotating them does not increase or decrease resistance linearly. Therefore, the middle of the potentiometer turn radius does not correspond to the “zero” position. Only a small fraction of the possible rotation produces a noticeable audio change. A similar issue was encountered with the slide potentiometers; the first sliders ordered had an audio taper as well. Once the error was recognized, the sliders were replaced with the correct linear taper components. The rotary potentiometers for bass and treble were not returned because they were ordered from a different supplier and shipping handling costs would have far exceeded the price of the parts themselves. Using a dual-gang linear rotary potentiometer would enhance the project by tying the bass and

treble controls of the channels together. Any changes made are applied to both channels via a single dial.

Aside from the complications outlined, the equalizer circuit performed admirably well. An audio signal enters the circuit and exits modified according to each slide potentiometer position. Moving the sliders up or down alters the intensity in the corresponding frequency bands, which changes how the music sounds. The LED display shows a visually interesting and accurate representation of the content in each band, and reflects changes made with the sliders. A user can modify both left and right channel independently.

The decision to use analog filters and discrete components was made with little thought to project cost or complexity. Tables I and II demonstrate the fiscal monstrosity this project formulates. Digital signal processing (DSP) would have proven a simpler and cheaper approach. A single, large microcontroller would operate directly on a sampled signal instead of peak detector values. Slide potentiometers could still provide tactile user input, but the controller would somehow sample the position and apply settings to the output waveform accordingly. As an added bonus, digital signal processing would allow for sharper filters with higher quality factors. Sharper filters blend with adjacent frequency bands less and therefore may produce better sounding audio. DSP could combine the entire audio recording suite into one package.

VIII. Bibliography

[1] "Equalization." Wikipedia, the Free Encyclopedia, 4 May 2011. Web. 05 June 2011. <http://en.wikipedia.org/wiki/Equalization>.

[2] "Equalisation." MediaWiki, 22 Feb. 2011. Web. 05 June 2011. <http://www.idc.ul.ie/idcwiki/index.php/Equalisation>.

[3] "Equalization (audio)." Wikipedia, the Free Encyclopedia, 4 June 2011. Web. 05 June 2011. <http://en.wikipedia.org/wiki/Equalization_(audio)>.

[4] Franco, Sergio. "Graphic Equalizers." *Design with Operational Amplifiers and Analog Integrated Circuits.* New York: McGraw-Hill, 2002. 125-26. Print.

[5] "FreeRTOS ATMega328-P Port." *AVR Freaks Forum*. Web. 26 May 2011. <http://www.avrfreaks.net/index.php?name=PNphpBB2>.

[6] "Atmega 16 and Maxim Max7219." AVR Freaks Forum. Web. 26 May 2011. <http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&p=717744>.

[7] Atmel. *AVR042: AVR Hardware Design Considerations*. 2010. PDF.

[8] "ATX (ATX12V) 24 Pin Power Supply Connector Pinout and Wiring." *Handbook of Hardware Pinouts, Cables Schemes and Connectors Layouts @ Pinouts.ru.* Ed. Dave Johnson, Ken Mazie, and Nick Dedman. 23 Mar. 2011. Web. 13 Apr. 2011. <http://pinouts.ru/Power/atx_v2_pinout.shtml>.

[9] Barry, Richard. *FreeRTOS Reference Manual: API Functions and Configuration Options*. [S.l.]: Real Time Engineers, 2009. Print.

[10] Carter, Bruce. "Filter Design in Thirty Seconds." *Application Report SLOA093*. Texas Instruments, Dec. 2001. PDF. <http://focus.ti.com/lit/an/sloa093/sloa093.pdf>.

[11] "Know Code Size and Data Size." *AVR Freaks Forum*. Web. 4 June 2011. <http://www.avrfreaks.net/index.php?name=PNphpBB2>.

[12] Maw, Carlyn. "Setting up an Arduino on a Breadboard." *ITP Physical Computing*. 23 Oct. 2008. Web. 29 Apr. 2011. <http://itp.nyu.edu/physcomp/Tutorials/ArduinoBreadboard>.

[13] "MAX7221 LED DRIVER TUTORIAL." *UMASS AMHERST M5*. The College of Engineering at UMass Amherst, 25 Jan. 2011. Web. 26 May 2011. <http://www.ecs.umass.edu/ece/m5/tutorials/7221_led_driver_tutorial.html>.

[14] "Peak Detector Circuit." *Electronics and Communications Engineering*. 07 Jan. 2011. Web. 10 Feb. 2011. <http://ecelab.com/circuit-peak-detector.htm>.

A. Schematics

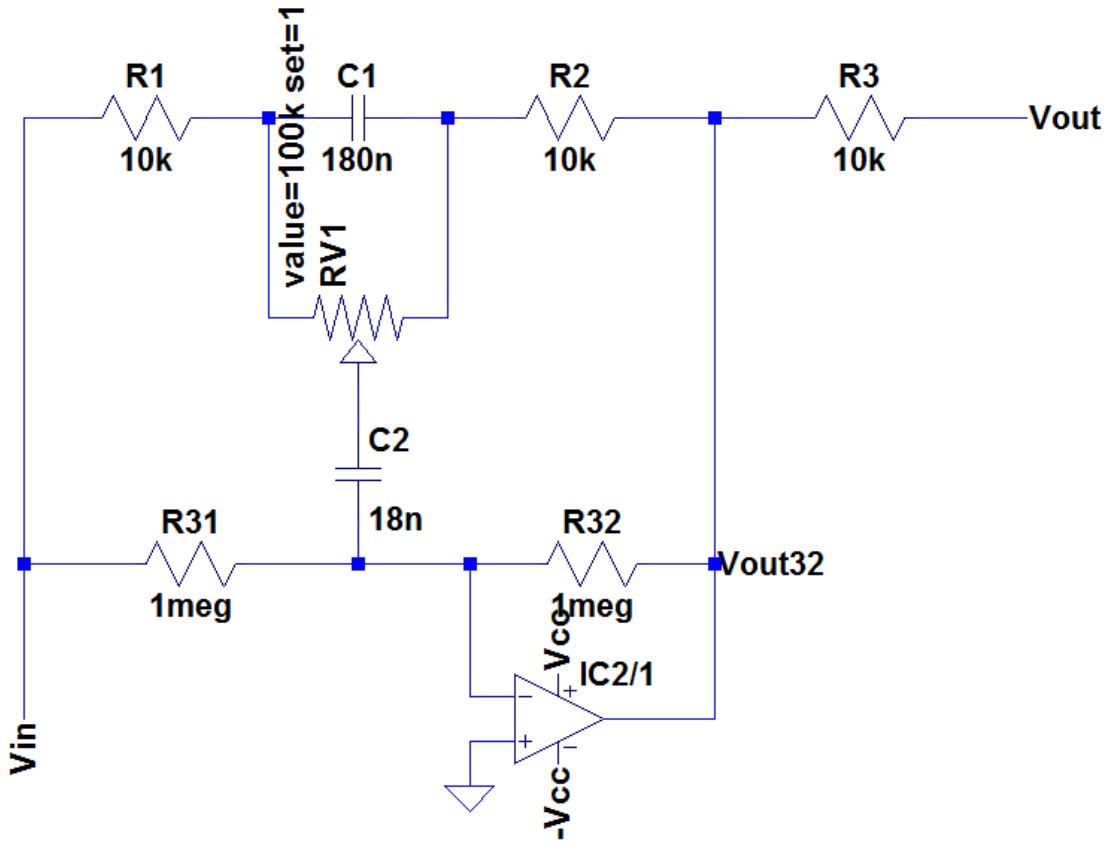


Figure 1: Equalizer Circuit. A block of the equalizer circuit operates on a single frequency band. Capacitors C1 and C2 determine which frequency band, and C1 will always be ten times greater than C2. Adjusting the resistor potentiometer changes the intensity in corresponding band. The design for this circuit comes from the textbook by Franco [4].

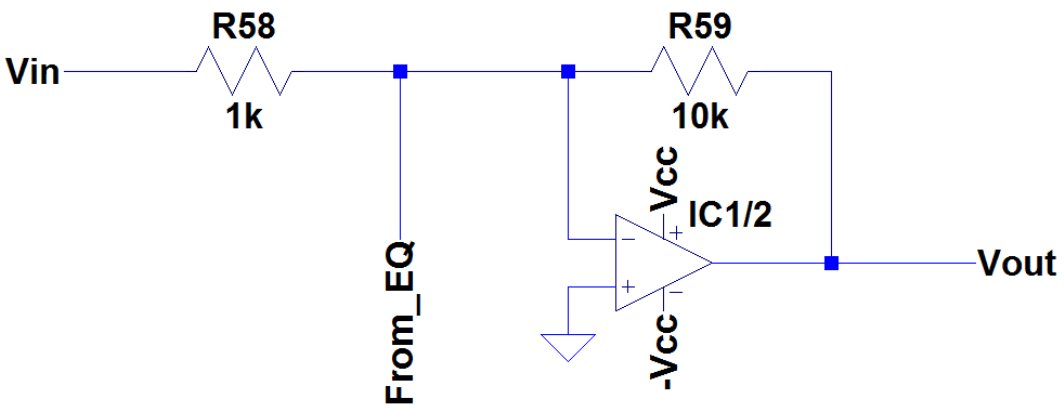


Figure 2: Inverting Summer. This block of circuitry adds the inverted and modified signals from all ten bands and bass/treble circuits together. The resulting waveform reflects any changes made to the input by the sliders and dials.

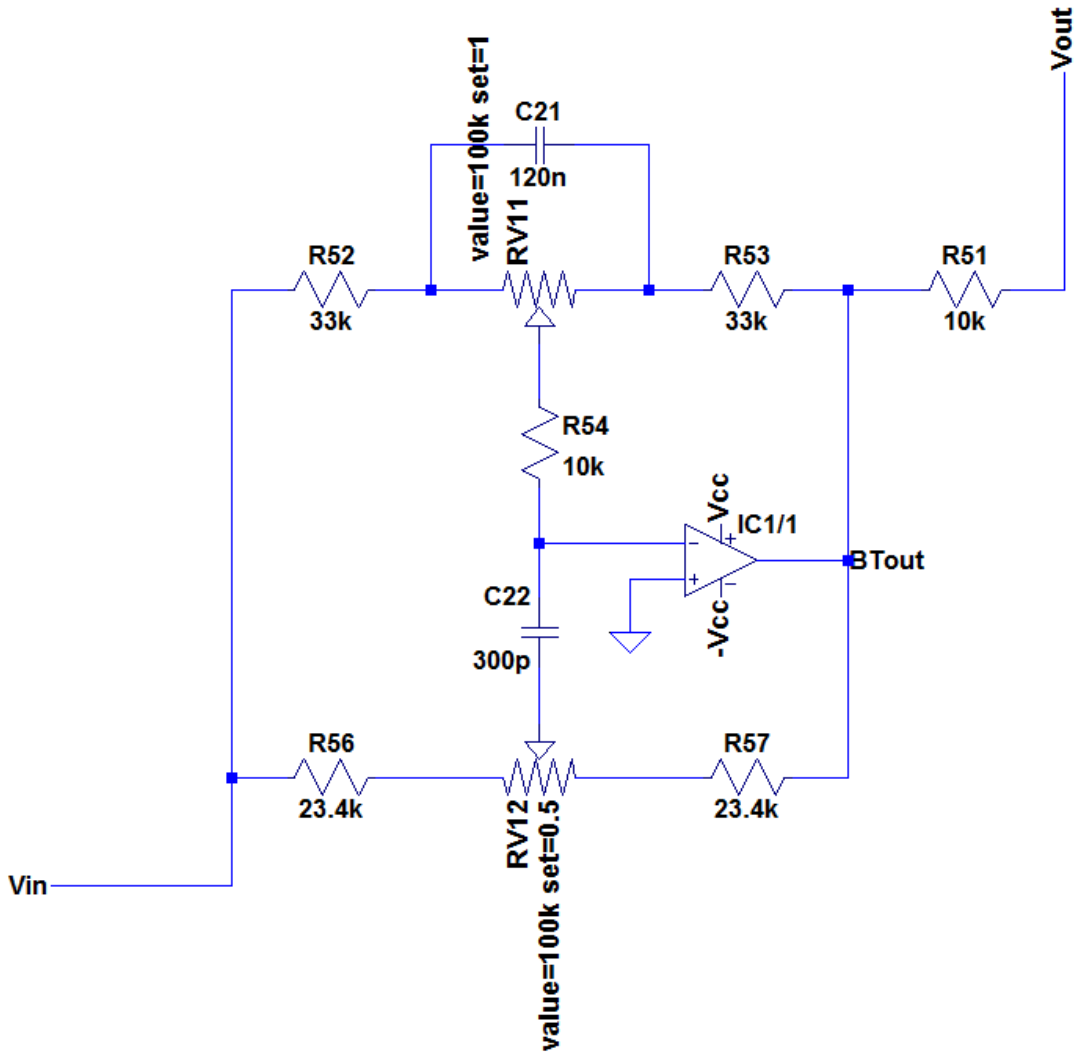


Figure 3: Bass and Treble Circuit. This block of circuitry modifies the low and high frequencies outside the 10 bands operated on by the sliders. The design for this circuit comes from the textbook by Franco [4].

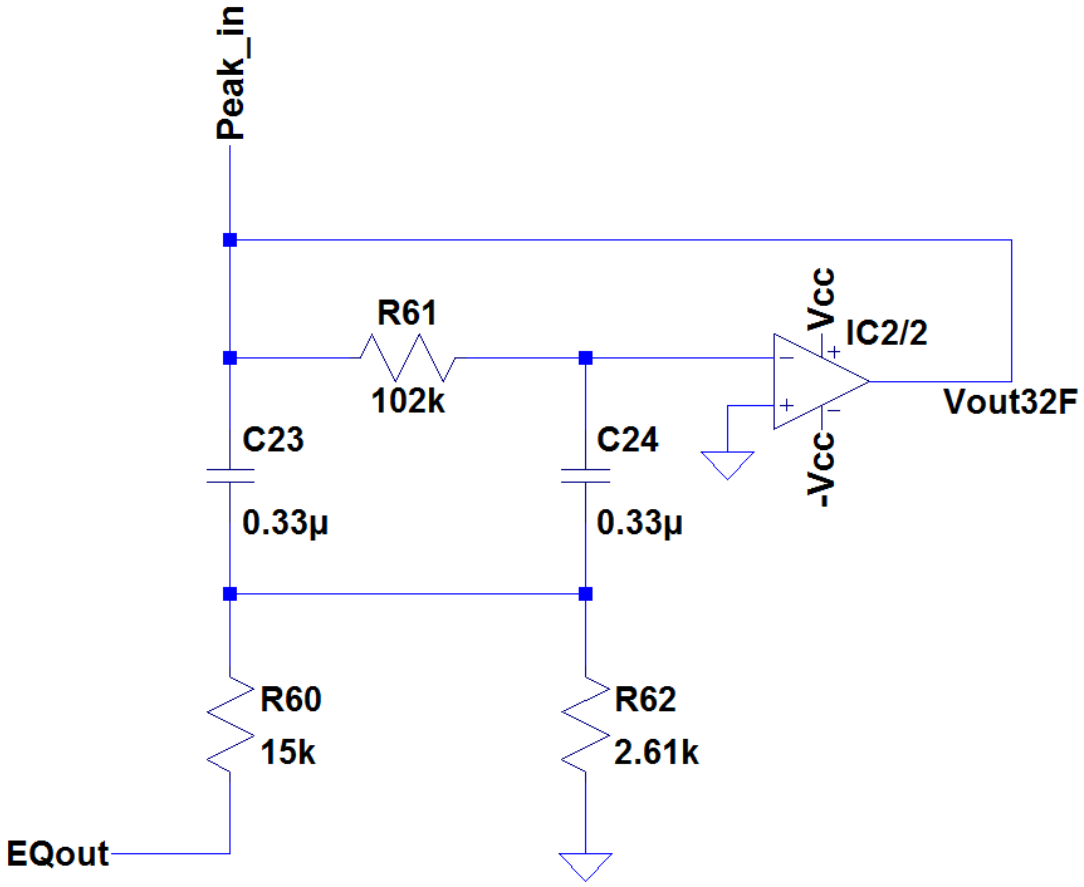


Figure 4: Display Band-pass Filter. This filter allows a specific frequency to pass, while removing any other content from the input. The combination of capacitors and resistors determine which frequency passes. The design for this circuit comes from Texas Instruments [10].

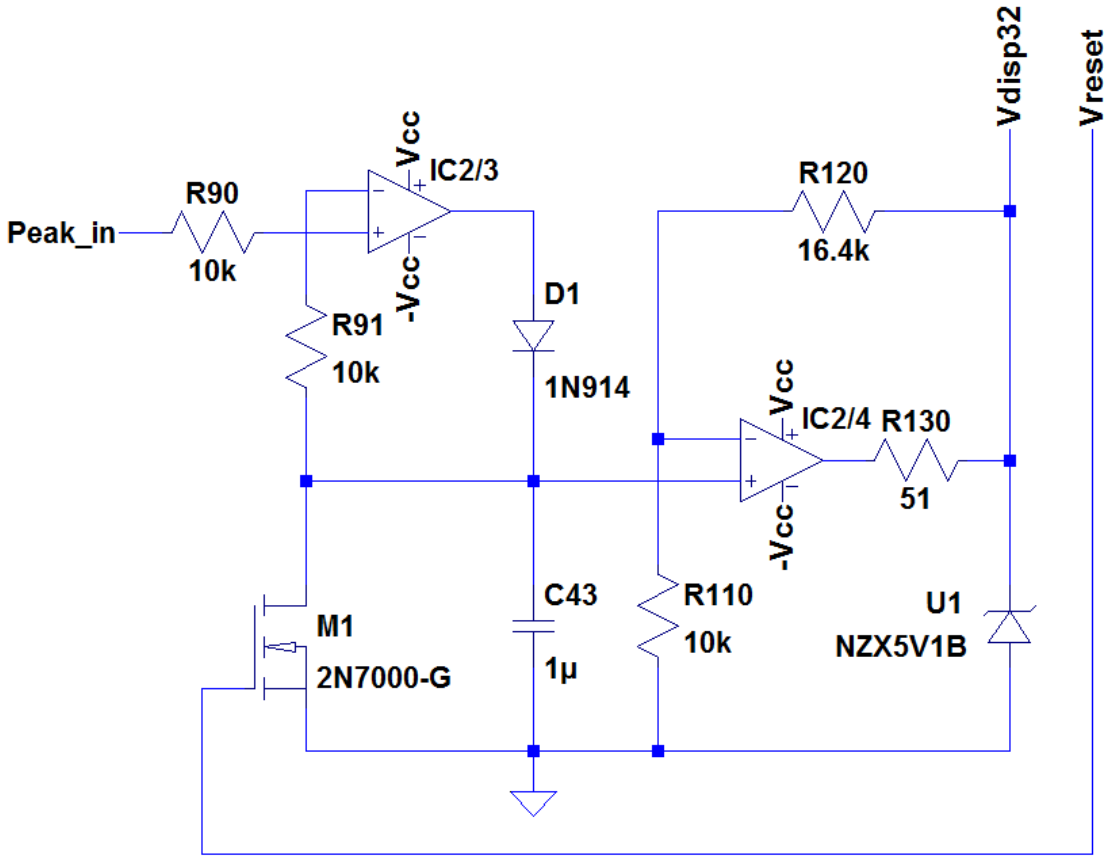


Figure 5: Peak Detector Circuit. The capacitor follows the input waveform to the maximum value and holds it until either a higher maximum occurs or the transistor switches on. The diode prevents any loss on the capacitor when the input signal falls below the capacitor voltage level. The resistors allow current to flow and/or provide output gain. The Zener diode prevents the output voltage from exceeding 5V, which could result in incorrect measurements or damage the microcontroller ADC. The basic design for this circuit comes from ECE Lab [14].

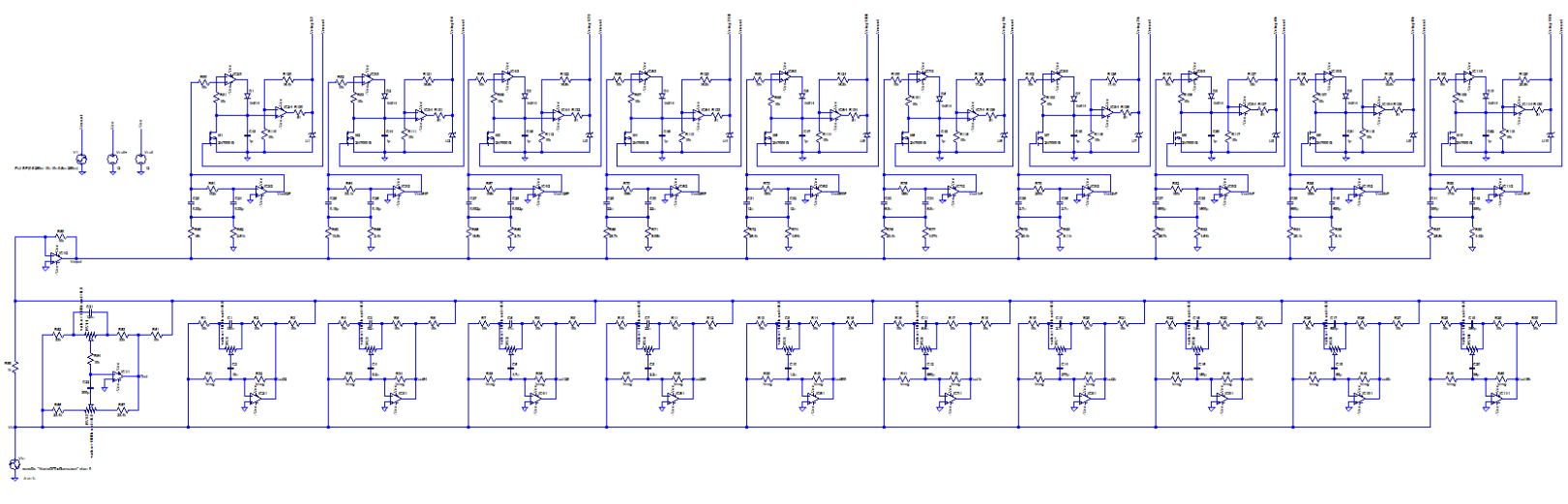


Figure 6: Full EQ Circuit. All circuit blocks combine to create a ten band graphic equalizer and display. The Quad J-FET op-amps, display drivers, analog multiplexor and microcontroller are not shown.

B. Simulations

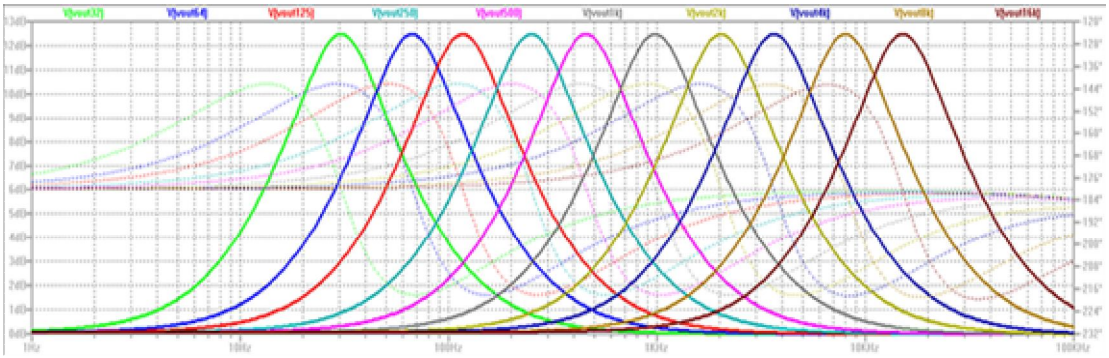


Figure 7: Equalizer Filter Response. Each frequency band can attain a maximum of about 12.5dB. The filters overlap and modify parts of adjacent bands, so the graph line of the output will look "bumpy". The EQ also alters signal phase but the delay equals nearly 1/30th of a second, a change small enough to go unnoticed by human ears.

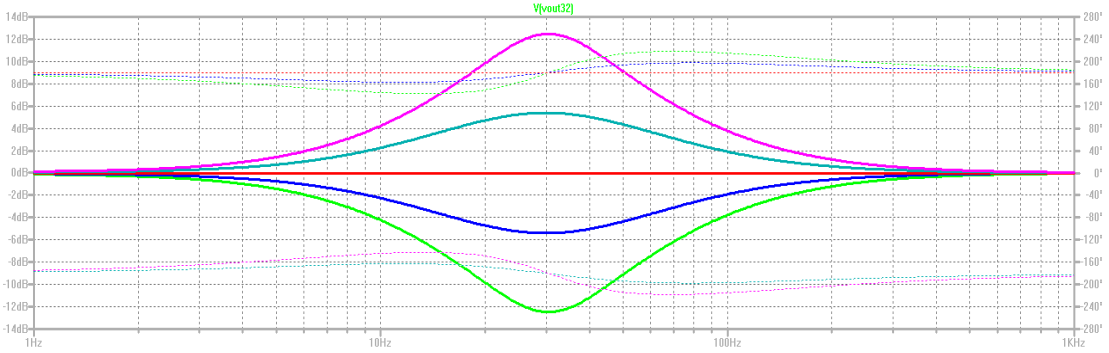


Figure 8: EQ Filter Response at Single Frequency. As the slider moves up and down, the intensity in the corresponding frequency band increases or decreases. The different lines represent a slider at varying track positions.

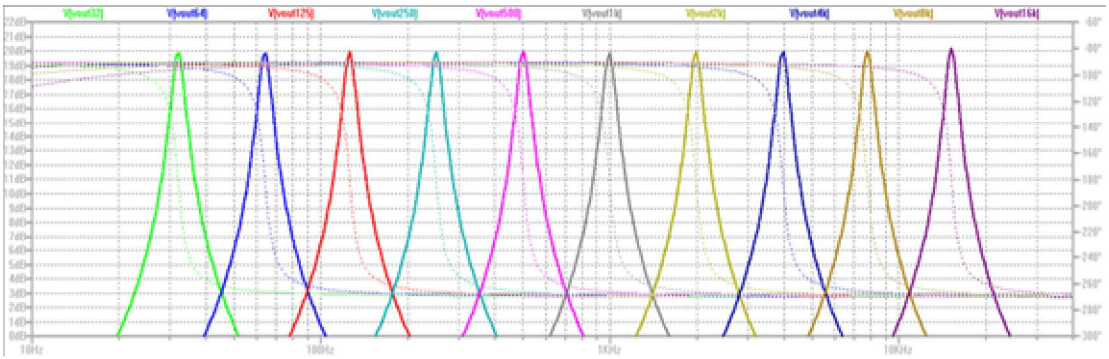


Figure 9: Display Filter Response. Each display block has a band-pass filter that only allows the specified frequency content to pass. Here, the bottom of the graph stops at 0dB, but each line continues below. Therefore, the filters remove any content outside the desired frequency band. Some overlap (~3dB) occurs between bands, but this can actually benefit the EQ. Additionally, each filter alters the signal phase, inverting the waveform.

Figure 10: LT Spice Simulation. The top waveform comes from an imported .WAV file, and follows an actual audio signal. The middle waveform results from the .WAV signal passing through the 32Hz display band-pass filter. Only content in the 32Hz band continues to the peak detector. The bottom waveform corresponds to the peak detector output. The capacitor charges to the maximum value seen in the 32Hz band, and the buffering op-amp increases it by a factor of 2.64. The maximum value of the bottom trace is 1.2V, while the maximum value of the middle trace is 0.48V.

C. Parts List and Cost

Device	Value	Manufacturer	Part Number	QTY	Price/ ea
Audio Jack	AUDIO-JACK	Switchcraft	35RAPC4BHN2	2	$0.86
Analog Multiplexor	-	Analog Devices	ADG406BNZ	2	Samp
Capacitor	180nF	Kemet	R82DC3180AA60J	6	$0.39
Capacitor	18nF	Kemet	MMK5183K100J01L16.5TA18	2	$0.18
Capacitor	82nF	Panasonic	ECQ-V1H823JL2	6	$0.13
Capacitor	8.2nF	Kemet	MMK5822K50J01L16.5TR16	2	$0.39
Capacitor	47nF	EPCOS	B32529C473J	2	$0.10
Capacitor	4.7nF	Kemet	R82EC1470Z350J	2	$0.38
Capacitor	22nF	Kemet	R82EC2220DQ50J	2	$0.25
Capacitor	2.2nF	Kemet	MMK5222K63J01L16.5TR18	2	$0.19
Capacitor	12nF	Xicon	140-PEI2A123J-RC	6	$0.14
Capacitor	1.2nF	Xicon	140-PEI2A122G-RC	2	$0.21
Capacitor	5.6nF	TDK	FK14C0G1H562J	2	$0.38
Capacitor	560pF	Vishay	K561J15C0GF5TL2	2	$0.10
Capacitor	2.7nF	Kemet	MMK5272J100J01L16.5TR16	6	$0.42
Capacitor	270pF	TDK	FK18C0G1H271J	2	$0.22
Capacitor	1.5nF	TDK	FK28C0G1H152J	6	$0.28
Capacitor	150pF	Vishay	K151J15C0GF5TH5	2	$0.07
Capacitor	680pF	Vishay	K681J15C0GF53L2	6	$0.11
Capacitor	68pF	Vishay	K680J15C0GF5TL2	2	$0.06
Capacitor	360pF	Murata	RPE5C1H361J2P1A03B	2	$0.30
Capacitor	36pF	AVX	SR151A360JAR	2	$0.23
Capacitor	120nF	Kemet	C320C124J5R5TA	2	$0.41
Capacitor	300pF	Murata	RPE5C1H301J2P1A03B	2	$0.30
Capacitor	0.33uF	Panasonic	ECQ-V1H334JL	4	$0.33
Capacitor	22nF	Kemet	C320C223J5R5CA7303	4	$0.51
Capacitor	6.8nF	TDK	FK14C0G1H682J	4	$0.38
Capacitor	390pF	TDK	FK28C0G2A391J	4	$0.22
Capacitor	1uF	TDK	FK28Y5V1C105Z	20	$0.23
Electrolytic Capacitor	10uF	Panasonic	ECE-A1CKA100	4	$0.23
Capacitor	0.1uF	Nichicon	QYS1H104JTP	4	$0.21
MOSFET	N-Channel	Supertex	2N7000-G	20	$0.27
Diode	-	Vishay	1N914TAP	20	$0.03
5V Zener Diode	NZX5V1B	NXP Semiconductors	NZX5V1B,133	20	$0.03
Quad J-FET Opamp	TL074IN	STMicroelectronics	TL074IN	22	$0.80
Microcontroller	ATMEGA328P	Atmel	ATMEGA328P-PU	2	$4.28

Device	Value	Manufacturer	Part Number	QTY	Price/ ea
Headers - Vertical	40-pin breakaway	Tyco Electronics	9-146274-0	4	$2.15
Headers - Right Angle	40-pin breakaway	Tyco Electronics	9-103323-0	7	$1.48
LED Display Driver	MAX7221	Maxim	MAX7221CNG+	4	Samp
Power Connector	MOLEX_24	Molex	39-28-8240	1	$3.66
Resistor	10k	KOA Speer	CMF1/41002FLFTR	128	$0.07
Resistor	1M	KOA Speer	MF1/4DCT52A1004F	40	$0.06
Resistor	33k	KOA Speer	MF1/4DCT52R3302F	4	$0.05
Resistor	23.7k	KOA Speer	MF1/4DCT52R2372F	6	$0.05
Resistor	1k	KOA Speer	MF1/4DCT52R1001F	2	$0.05
Resistor	15k	KOA Speer	MF1/4DCT52R1502F	2	$0.05
Resistor	102k	KOA Speer	MF1/4DCT52R1023F	2	$0.05
Resistor	2.61k	KOA Speer	MF1/4DCT52R2611F	2	$0.05
Resistor	13.7k	KOA Speer	MF1/4DCT52R1372F	2	$0.05
Resistor	93.1k	KOA Speer	MF1/4DCT52R9312F	2	$0.05
Resistor	2.4k	KOA Speer	MF1/4DCT52R2401F	2	$0.05
Resistor	15.8k	KOA Speer	MF1/4DCT52R1582F	2	$0.05
Resistor	105k	KOA Speer	MF1/4DCT52R1053F	2	$0.05
Resistor	2.7k	KOA Speer	MF1/4DCT52R2701F	2	$0.05
Resistor	28.7k	KOA Speer	MF1/4DC2872F	2	$0.06
Resistor	196k	KOA Speer	MF1/4DCT52R1963F	2	$0.05
Resistor	5.1k	KOA Speer	MF1/4DCT52R5101F	4	$0.05
Resistor	26.7k	KOA Speer	MF1/4DCT52R2672F	4	$0.05
Resistor	180k	KOA Speer	MF1/4DCT52R1803F	4	$0.05
Resistor	4.64k	KOA Speer	MF1/4DCT52R4641F2	4	$0.05
Resistor	158k	KOA Speer	MF1/4DCT52R1583F	2	$0.05
Resistor	4.02k	KOA Speer	MF1/4DCT52R4021F2	2	$0.05
Resistor	29.4k	KOA Speer	MF1/4DCT52R2942F	4	$0.05
Resistor	200k	KOA Speer	MFS1/4DCT52R2003F	4	$0.05
Resistor	5.11k	KOA Speer	MF1/4DCT52R5111F2	2	$0.05
Resistor	25.5k	KOA Speer	MF1/4DCT52R2552F	2	$0.05
Resistor	174k	KOA Speer	MF1/4DCT52R1743F	2	$0.05
Resistor	4.42k	KOA Speer	MF1/4DCT52R4421F	2	$0.05
Resistor	16.5k	KOA Speer	MF1/4DCT52R1652F	6	$0.05
Resistor	16.9k	KOA Speer	MF1/4DCT52R1692F	4	$0.05
Resistor	17.4k	KOA Speer	MF1/4DCT52R1742F	4	$0.05
Resistor	18k	KOA Speer	MF1/4DCT52R1802F	2	$0.05
Resistor	18.7k	KOA Speer	MF1/4DCT52R1872F	2	$0.05
Resistor	20.5k	KOA Speer	MF1/4DCT52R2052F	2	$0.05
Resistor	51	Xicon	293-51-RC	20	$0.09
Resistor	165	KOA Speer	MF1/4DCT52R1650F	2	$0.05
Resistor	37.4k	KOA Speer	MF1/4DCT52R3742F	4	$0.05

Device	Value	Manufacturer	Part Number	QTY	Price/ ea
100k Slide Pot	312-619AF-100K	Alpha (Taiwan)	RA6043F-20-10EB1-B15	20	$1.63
100k Rotary Pot	100k Pot	TT Electronics	P260T-D1BS3CA100K	2	$3.92
Tactile Switch	SW_TAC	Omron	B3F-1002	2	$0.24
IC Frame	IC_FRAME	3M	4828-3004-CP	2	$0.31
IC Frame	IC_FRAME	3M	4824-3004-CP	4	$0.23
IC Frame	IC_FRAME	3M	4828-6004-CP	2	$0.31
Green LED	LED_5MM	VCC	VAOL-5GDE4	101	$0.11
Yellow LED	LED_5MM	VCC	VAOL-5GCE4	40	$0.13
Red Led	LED_5MM	VCC	VAOL-5GAE4	20	$0.12
12" F/F Wires	-	Sparkfun	PRT-09389	1	$4.50
12" 6-pin Wires	-	Sparkfun	PRT-10376	4	$1.95
12" 5-pin Wires	-	Sparkfun	PRT-10375	4	$1.95
ATX PSU	550W	Broadway Com Corp	OKIA-BLACK-550	1	$21.99
EQ PCB	-	Advanced Circuits	GE v2.0.5 w/I	2	$33.00
Display PCB	-	Advanced Circuits	LED v1.2	2	$33.00
Washers	Nylon	Miner's	NYW4	24	$0.15
Stand-offs	Nylon	Miner's	SP-041	24	$0.20
Screws	Aluminum	Miner's	AWS-2K	24	$0.35

Table 1: Bill of Materials

Component Price	$362.64
Tax	$16.92
Shipping & Handling	$74.28
Project Total	**$453.84**

Table 2: Project Costs

D. Printed Circuit Board Artwork

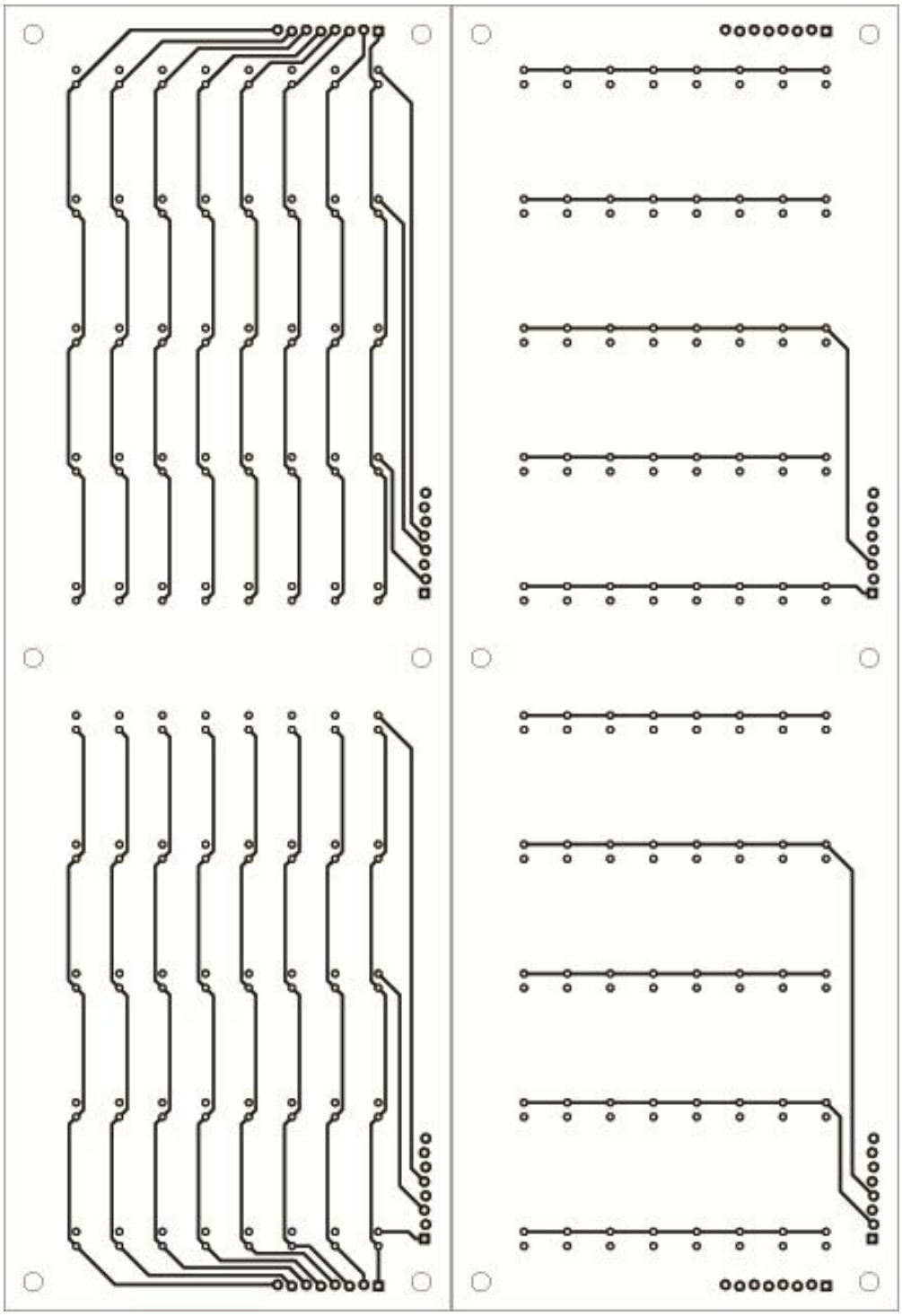


Figure 11: Display PCB Layout. Left is the top copper layer, right is the bottom copper layer. Two layers make the "criss-cross" style of the traces easy to realize.

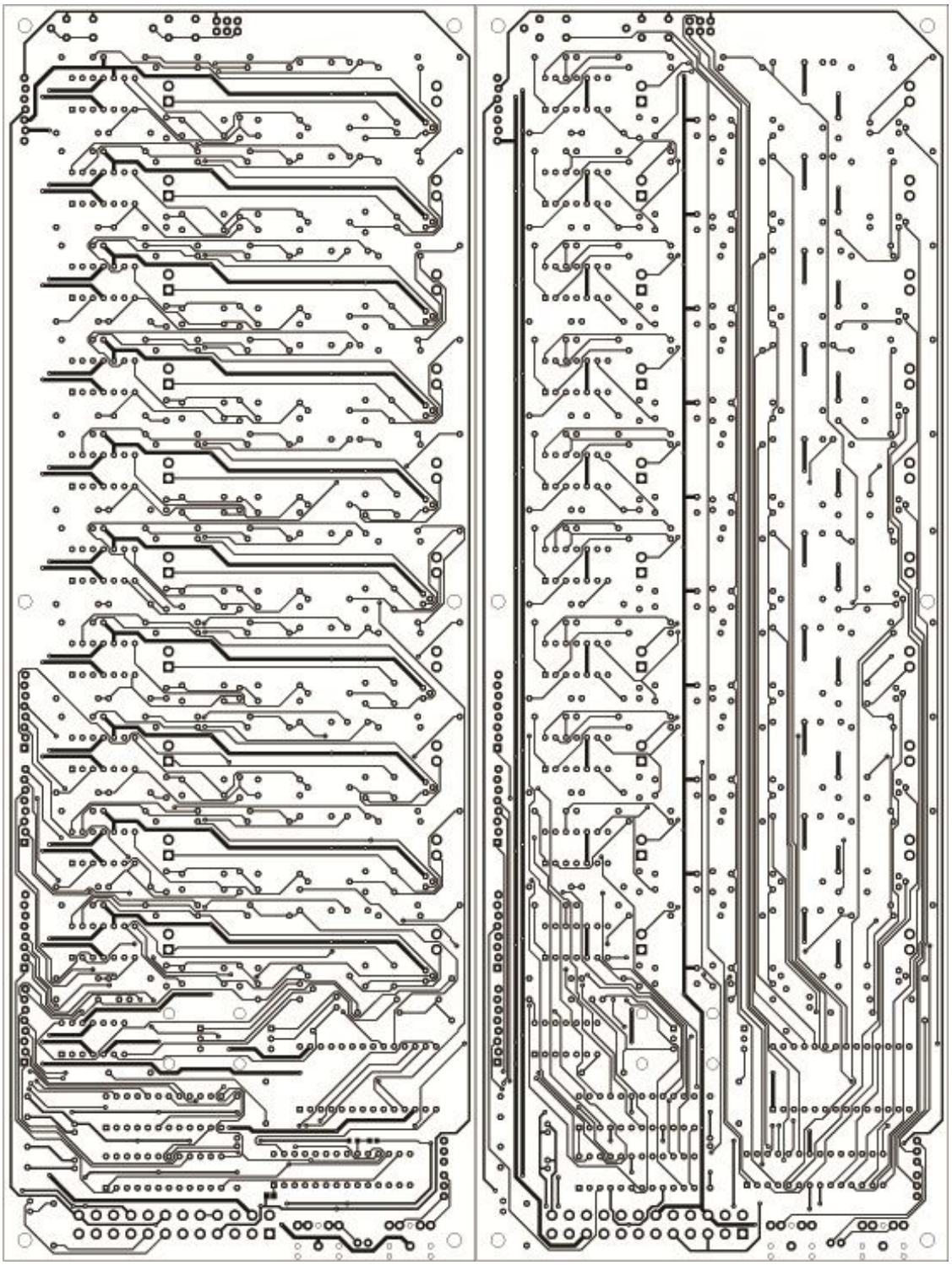


Figure 2: Equalizer PCB Layout. Left is top copper layer, right is bottom copper layer. Power traces are larger than others to allow for higher current magnitudes. All non-critical traces routed with FreeRouting (http://www.freerouting.net/).

E. Component Locations

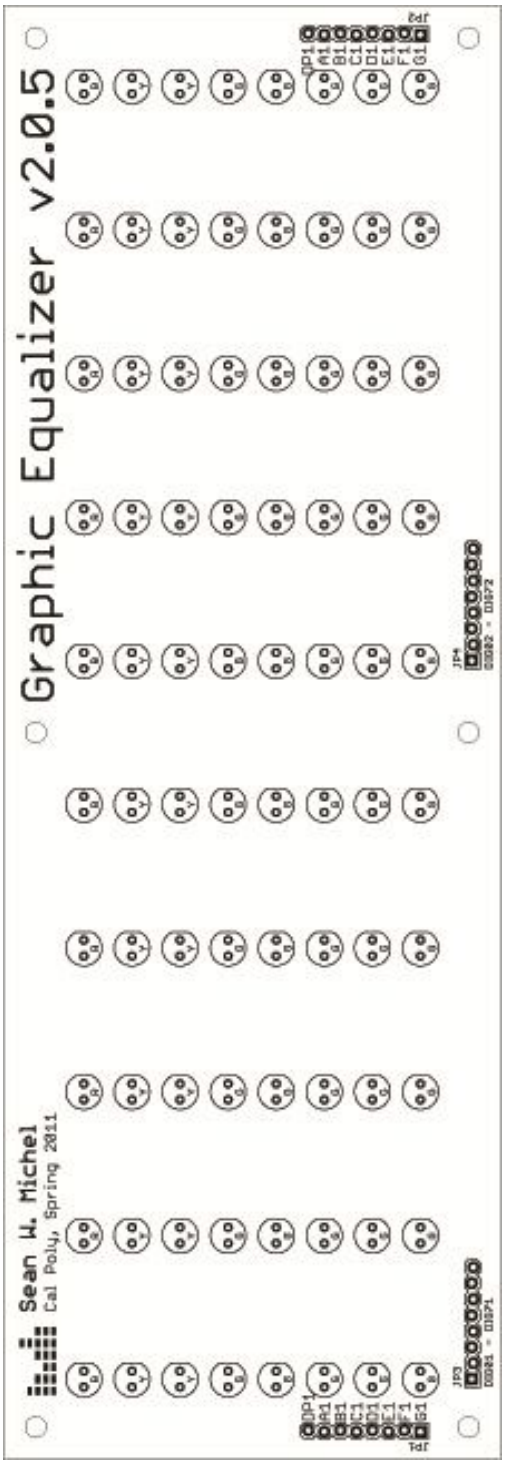


Figure 13: Display PCB Component Locations. Each LED has a ring with a "clipped" edge that represents the cathode. Color identifiers are covered by the component itself.

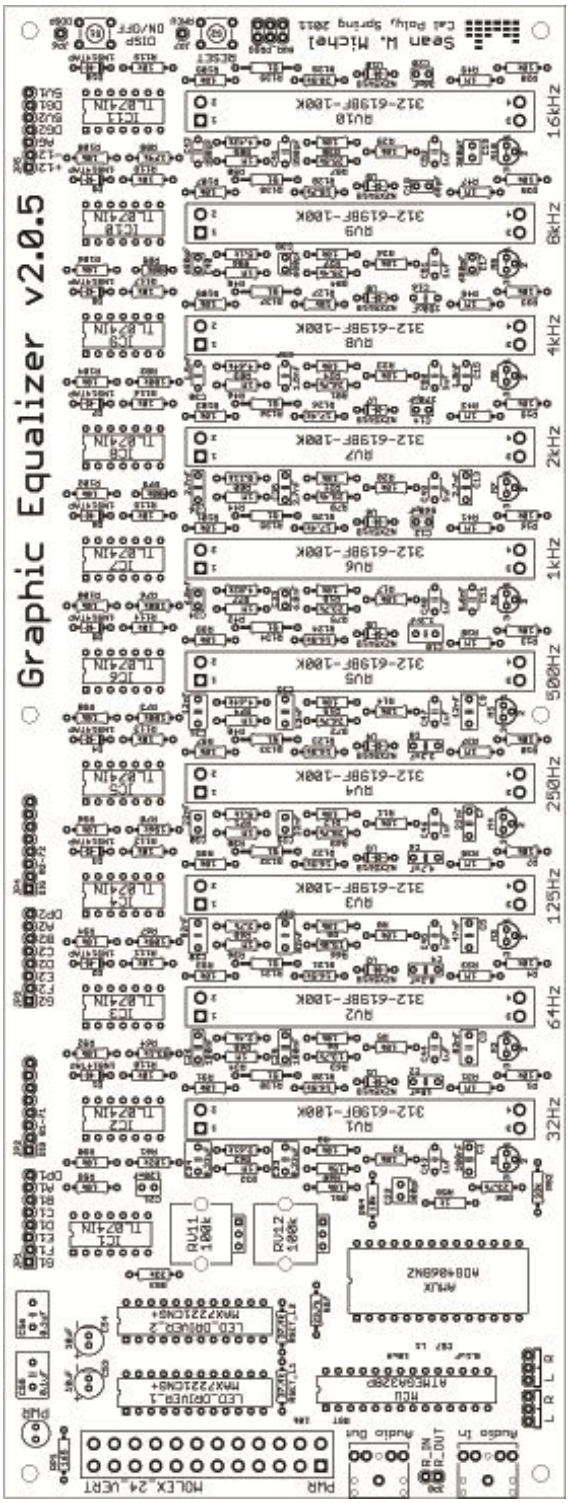


Figure 14: Equalizer PCB Component Locations. Circuit inputs are located on the left and power flows left to right. This corresponds to the way English text is read in books, so feels more natural.

F. Program Listing

```
// Sean W. Michel
// EE Senior Project
// Graphic Equalizer v2.0.5
// Cal Poly, SLO
// Spring 2011

// Header includes
#include <stdint.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"

// Define the pin numbers to make code more readable.
#define PR32    1   // PD0
#define PR64    2   // PD1
#define PR125   3   // PB7
#define PR250   4   // PD3
#define PR500   5   // PD4
#define PR1k    6   // PD5
#define PR2k    7   // PD6
#define PR4k    8   // PD7
#define PR8k    9   // PB0
#define PR16k   10  // PB1
#define A0      1   // PC1
#define A1      2   // PC2
#define A2      3   // PC3
#define A3      4   // PC4
#define MUXEN   6   // PB6
#define MOSI    3   // PB3
#define MISO    4   // PB4
#define SCK     5   // PB5
#define SS      2   // PB2

// RTDOS Function Prototypes
void vTaskGetLevels(void *pvParameters);
void vTaskDisplayShutdown(void);

// Function Prototypes
void vInitialize_IO(void);
void vInitialize_ADC0(void);
void vInitialize_SPI(void);
void vTransmitSPI(volatile int data);
void vTransmitLED_Right(int digAddress, int displayValue);
void vTransmitLED_Left(int digAddress, int displayValue);
void vClear_Disp(void);
void vInititialize_Disp(void);

// Semaphore definitions
xSemaphoreHandle xBinarySemaphore;

// Define the variables within the structure
typedef struct param_type {
    uint8_t band_number;
    uint8_t pin_number;
    uint8_t port_letter;
```

```
    uint8_t mux_select;
} param_type;

// Array that holds converted levels. Static variable because it exists
// outside the function it's used in
static uint16_t voltage[10];

// Variable structure definition, holds both frequency band number, reset
// port, pin reset number and analog multiplexor select lines
param_type val[10] =
{
    {PR32,  0, 1, 0x00},
    {PR64,  1, 1, 0x02},
    {PR125, 7, 0, 0x04},
    {PR250, 3, 1, 0x06},
    {PR500, 4, 1, 0x08},
    {PR1k,  5, 1, 0x0A},
    {PR2k,  6, 1, 0x0C},
    {PR4k,  7, 1, 0x0E},
    {PR8k,  0, 0, 0x10},
    {PR16k, 1, 0, 0x12}
};

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

int main(void)
{
    // Create and take binary semaphores to avoid errors
    vSemaphoreCreateBinary(xBinarySemaphore);
    xSemaphoreTake( xBinarySemaphore, portMAX_DELAY );

    // Initialize board IO, SPI, ADC0 and display
    // Interrupts are auto-enabled by FreeRTOS
    vInitialize_IO();
    vInitialize_ADC0();
    vInitialize_SPI();
    vInititialize_Disp();

    // Create a task for each frequency band. If the MCU runs out of memory,
    // just use a FOR loop to reduce the number of running tasks.
    // Each task has the same priority since they all do the same thing.
    // They take in a structure of information that contains four pieces
    // of data specific to the band the task belongs to. The first item is
    // the band number, used to determine which band is being operated on
    // and which column to write data to. The second piece of data is the
    // pin number that will reset the peak detector. The third item is an
    // identifier for which port the reset pin belongs to. 0 corresponds
    // to PORTB and 1 corresponds to PORTD. The last item is the mux select
    // lines for the specific band. They all have the same priority because
    // they all need to run, but none is more important than any other.
    xTaskCreate( (pdTASK_CODE) vTaskGetLevels, (signed char *) "32Hz Band",
                 configMINIMAL_STACK_SIZE, &val[0], 0, NULL );

    xTaskCreate( (pdTASK_CODE) vTaskGetLevels, (signed char *) "64Hz Band",
                 configMINIMAL_STACK_SIZE, &val[1], 0, NULL );

    xTaskCreate( (pdTASK_CODE) vTaskGetLevels, (signed char *) "125Hz Band",
                 configMINIMAL_STACK_SIZE, &val[2], 0, NULL );

    xTaskCreate( (pdTASK_CODE) vTaskGetLevels, (signed char *) "250Hz Band",
```

```
                configMINIMAL_STACK_SIZE, &val[3], 0, NULL );

    xTaskCreate( (pdTASK_CODE) vTaskGetLevels, (signed char *) "500Hz Band",
                configMINIMAL_STACK_SIZE, &val[4], 0, NULL );

    xTaskCreate( (pdTASK_CODE) vTaskGetLevels, (signed char *) "1kHz Band",
                configMINIMAL_STACK_SIZE, &val[5], 0, NULL );

    xTaskCreate( (pdTASK_CODE) vTaskGetLevels, (signed char *) "2kHz Band",
                configMINIMAL_STACK_SIZE, &val[6], 0, NULL );

    xTaskCreate( (pdTASK_CODE) vTaskGetLevels, (signed char *) "4kHz Band",
                configMINIMAL_STACK_SIZE, &val[7], 0, NULL );

    xTaskCreate( (pdTASK_CODE) vTaskGetLevels, (signed char *) "8kHz Band",
                configMINIMAL_STACK_SIZE, &val[8], 0, NULL );

    xTaskCreate( (pdTASK_CODE) vTaskGetLevels, (signed char *) "16kHz Band",
                configMINIMAL_STACK_SIZE, &val[9], 0, NULL );

    // Create a task for resetting the display. Only triggers on external
    // interrupt 0, PD2. Task has priority 1 because we want it to run as
    // quickly as possible after being unblocked
    xTaskCreate( (pdTASK_CODE) vTaskDisplayShutdown,
                (signed char *) "External Interrupt",
                 configMINIMAL_STACK_SIZE, (void *) NULL, 1, NULL );

    // Begin the scheduler and thus task execution.
    vTaskStartScheduler();

    return 0;
}

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

// A task for converting and storing frequency band voltage levels
void vTaskGetLevels(void *pvParameters)
{
    // Recieved struct hold band information
    param_type *xReceivedStructure = (struct param_type *) pvParameters;

    // This is a variable for the number of lights lit up in each column
    uint8_t write_level = 0x00;

    for (;;)
    {
        // Disable interrupts to avoid button press accidents
        taskENTER_CRITICAL();

        // Set the correct pins to get right level from analog mux
        PORTC = (xReceivedStructure->mux_select);

        // Delay 10 milliseconds to allow the mux output to settle
        _delay_ms(10);

        // Start ADC conversion by setting ADSC bit 6 high
        // ADSC is cleared by hardware on conversion complete
        ADCSRA |= (_BV(ADSC));

        // Maximum wait time for accurate conversion
```

```
_delay_us(260);

// Read voltage and store it in appropriate array space
voltage[(xReceivedStructure->band_number)-1] = ADC;

if( voltage[(xReceivedStructure->band_number)-1] < 0x072 )
{
    write_level = 0x00;
}

// One LED lit up
else if( voltage[(xReceivedStructure->band_number)-1] >= 0x072 &&
         voltage[(xReceivedStructure->band_number)-1] <  0x0E3)
{
    write_level = 0x01;
}

// Two LEDs lit up
else if( voltage[(xReceivedStructure->band_number)-1] >= 0x0E3 &&
         voltage[(xReceivedStructure->band_number)-1] < 0x155 )
{
    write_level = 0x03;
}

// Three LEDs lit up
else if( voltage[(xReceivedStructure->band_number)-1] >= 0x155 &&
         voltage[(xReceivedStructure->band_number)-1] < 0x1C7 )
{
    write_level = 0x07;
}

// Four LEDs lit up
else if( voltage[(xReceivedStructure->band_number)-1] >= 0x1C7 &&
         voltage[(xReceivedStructure->band_number)-1] < 0x238 )
{
    write_level = 0x0F;
}

// Five LEDs lit up
else if( voltage[(xReceivedStructure->band_number)-1] >= 0x238 &&
         voltage[(xReceivedStructure->band_number)-1] < 0x2AA )
{
    write_level = 0x1F;
}

// Six LEDs lit up
else if( voltage[(xReceivedStructure->band_number)-1] >= 0x2AA &&
         voltage[(xReceivedStructure->band_number)-1] < 0x31C )
{
    write_level = 0x3F;
}

// Seven LEDs lit up
else if( voltage[(xReceivedStructure->band_number)-1] >= 0x31C &&
         voltage[(xReceivedStructure->band_number)-1] < 0x38D )
{
    write_level = 0x7F;
}

// All LEDs lit up
```

```
        else if( voltage[(xReceivedStructure->band_number)-1] >= 0x38D &&
                 voltage[(xReceivedStructure->band_number)-1] <= 0x3FF )
        {
            write_level = 0xFF;
        }

        // Write to left side of the dispay if the band number is 0-5
        if(xReceivedStructure->band_number <= 5)
        {
            vTransmitLED_Left(xReceivedStructure->band_number, write_level);
        }

        // Write to the right side of the display if the band bumber is 6-10
        else
        {
            vTransmitLED_Right((xReceivedStructure->band_number)-5,
                               write_level);
        }

        // Reset the peak detector by sending the transistor base high then
        // toggle low. The switch/case statement chooses the correct port
        // to write to. In this code, only port B and D are used for this
        // purpose.
        switch ( xReceivedStructure->port_letter )
        {
            case 0:
                PORTB |= (_BV(xReceivedStructure->pin_number));
                _delay_ms(10);
                PORTB &= ~(_BV(xReceivedStructure->pin_number));
                break;
            case 1:
                PORTD |= (_BV(xReceivedStructure->pin_number));
                _delay_ms(10);
                PORTD &= ~(_BV(xReceivedStructure->pin_number));
                break;
            default:
                break;
        }

        // Exit the critical and allow interrupts again
        taskEXIT_CRITICAL();

        // This delay ensures all tasks have a chance to run without
        // conflicting with each other. Making it longer makes the
        // system look more "sluggish" and update slower. Can adjust
        // to liking.
        vTaskDelay(2 / portTICK_RATE_MS);
    }
}

// This task shutdowns the LED display once the external interrupt
// is triggered.
void vTaskDisplayShutdown(void)
{
    // Binary count to establish if the display is already on
    uint8_t ext_int_data = 0;

    // Since the switch/case function only runs once the task is
    // unblocked, it starts at zero and the zero case turns off
    // the display, then increases the count. The next time the
```

```
    // task is unblocked, it will run case one. This re-enables the
    // display and resets the count.

    for (;;)
    {
        // This task unblocks once it the binary semaphore has been
        // given. It won't run unless it gets it from the external
        // interrupt vector.
        xSemaphoreTake( xBinarySemaphore, portMAX_DELAY );

        // 50ms button debounce
        vTaskDelay(50 / portTICK_RATE_MS);

        // Disable interrupts while transmitting data
        taskENTER_CRITICAL();

        switch (ext_int_data)
        {
            case 0:
                // Enter shutdown, disable display
                vTransmitLED_Right(0x0C, 0x00);
                vTransmitLED_Left(0x0C, 0x00);
                ext_int_data = 1;
                break;
            case 1:
                // Leave shutdown, enable display
                vTransmitLED_Right(0x0C, 0x01);
                vTransmitLED_Left(0x0C, 0x01);
                ext_int_data = 0;
                break;
            default:
                break;
        }

        // Leave Critical mode.
        taskEXIT_CRITICAL();
    }
}

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

// This function initializes the Ports
// Enables interrupts, and configures them for BTNs
void vInitialize_IO(void)
{
    // Enable all of PORTB for SPI and such
    DDRB = 0xFF;

    // Enable most of PORTD for peak detector reset, except PD2
    // which is external interrupt 0
    DDRD |= (_BV(0)) |
            (_BV(1)) |
            (_BV(3)) |
            (_BV(4)) |
            (_BV(5)) |
            (_BV(6)) |
            (_BV(7));

    // Enable the analog mux select lines
    DDRC |= (_BV(A0)) |
```

```
            (_BV(A1)) |
            (_BV(A2)) |
            (_BV(A3));

    // Enable multiplexer and give it 10 milliseconds to get set up
    // Technically redundant after enabling all of PORTB, but makes
    // code easier to read and debug
    PORTB |= (_BV(MUXEN));
    _delay_ms(10);

    // Enable external interrupt 0
    EIMSK |= (_BV(INT0));

    // Falling edge of INT0 generates interrupt
    EICRA |= (_BV(ISC01));
}

// Function to initialize the AD converter
void vInitialize_ADC0(void)
{
    // Turn On ADC and set prescaler (CLK/32)
    // This gives an ADC frequency of 3686400/32=115200
    ADCSRA = 0x85;

    // Set ADC channel 0 with 1X gain, internal VREF
    ADMUX = 0x40;

    // Disable the digital input buffer of PA0 to reduce power consumption
    DIDR0 = (_BV(0));

    // 25 cycles or 260us (max for start-up)
    _delay_us(260);
}

void vInitialize_SPI(void)
{
    // Make sure SPI is not in low power mode
    PRR &= ~(_BV(PRSPI));

    // Make MOSI, SCK and SS outputs
    DDRB |= (_BV(MOSI)) |
            (_BV(SCK))  |
            (_BV(SS));

    // Enable SPI, Master, set clock rate (FCK/16)
    SPCR = (_BV(SPE))  |
           (_BV(MSTR)) |
           (_BV(SPR0));
}

// Transmit data to SPI device(s)
// May want to modify this code to incorporate the _BV function
void vTransmitSPI(volatile int data)
{
    // Start the transmission
    SPDR = data;

    // Wait the end of the transmission
    while (!(SPSR & (1<<SPIF)));
}
```

```
void vTransmitLED_Right(int digAddress, int displayValue)
{
    // Assert slave select line by setting it low
    PORTB &= ~(_BV(SS));

    // 4 byte data transfer
    vTransmitSPI(digAddress);
    vTransmitSPI(displayValue);
    vTransmitSPI(0x00);
    vTransmitSPI(0x00);

    // De-assert slave select by setting it high
    PORTB |= (_BV(SS));
}

void vTransmitLED_Left(int digAddress, int displayValue)
{
    // Assert slave select line by setting it low
    PORTB &= ~(_BV(SS));

    // 4 byte data transfer
    vTransmitSPI(0x00);
    vTransmitSPI(0x00);
    vTransmitSPI(digAddress);
    vTransmitSPI(displayValue);

    // De-assert slave select by setting it high
    PORTB |= (_BV(SS));
}

// Clear display by setting all registers equal to 0x00
void vClear_Disp(void)
{
    vTransmitLED_Right(0x01, 0x00);
    vTransmitLED_Right(0x02, 0x00);
    vTransmitLED_Right(0x03, 0x00);
    vTransmitLED_Right(0x04, 0x00);
    vTransmitLED_Right(0x05, 0x00);

    vTransmitLED_Left(0x01, 0x00);
    vTransmitLED_Left(0x02, 0x00);
    vTransmitLED_Left(0x03, 0x00);
    vTransmitLED_Left(0x04, 0x00);
    vTransmitLED_Left(0x05, 0x00);
}

// Initialize the LED displays
void vInititialize_Disp(void)
{
    vTransmitLED_Right(0x0A, 0x0F);   // max intensity
    vTransmitLED_Right(0x0B, 0x04);   // scan digits 0-4
    vTransmitLED_Right(0x09, 0x00);   // do not decode digits
    vTransmitLED_Right(0x0C, 0x01);   // leave shutdown
    vTransmitLED_Right(0x0F, 0x00);   // display test off

    vTransmitLED_Left(0x0A, 0x0F);   // max intensity
    vTransmitLED_Left(0x0B, 0x04);   // scan digits 0-4
    vTransmitLED_Left(0x09, 0x00);   // do not decode digits
    vTransmitLED_Left(0x0C, 0x01);   // leave shutdown
```

```
    vTransmitLED_Left(0x0F, 0x00);   // display test off

    // Clear the display of any data still help in the registers
    vClear_Disp();
}

// This interrupt is triggered by pressing the display on/off button
// It jumps to the task vTaskDisplayShutdown
ISR(INT0_vect)
{
   static portBASE_TYPE xHigherPriorityTaskWoken;
   xHigherPriorityTaskWoken = pdFALSE;
   xSemaphoreGiveFromISR( xBinarySemaphore, &xHigherPriorityTaskWoken );
}
```