Electric Violin Audio Processor

Final Report

ECE 445: Spring 2022

Wei Gao (weigao4)

Alex Seong (aseong2)

Scott Foster (scottbf2)

Date Written: May 4, 2022

TA: Jeff Chang

Team: 22

Abstract

We designed an audio processor for an electric violin. The user interface and battery power supply function as intended. The audio processing subsystem is non-functional since the code for the audio signal processor contained bugs. These were introduced by AKM's DSP code generation software, and is a known issue.

Contents

1 Introduction

Current electric violin pickups tend to fall in one of two categories. Inexpensive pickups are readily available for either acoustic or solid-body violins, but produce a sound quality which is sometimes described as ”tinny” or ”nasal”, and whose harmonic content is too limited for a significant amount of sound design to be carried out. These typically have one piezoelectric sensor for the entire bridge. High-quality pickups produce a ”rich” sound but are expensive and often hard to obtain due to low production volume. These typically have at least one sensor for each string.

Furthermore, the type of strings used can drastically affect the sound of the instrument; for example, steel strings are characteristically ”bright” and tinny, while Thomastik Dominant synthetic strings are known for having a ”thin”-sounding E string whose timbre contrasts with that of the other three strings. The sound of the electric violin can even be affected by the properties of the effects chain or sound system, such as the size of the amplifier speaker.

The sound quality of instruments is inherently a subjective assessment. An example of the ”nasal” sound of an acoustic violin piezo pickup is demonstrated in [1], and a discussion on the sound quality of common violin strings is given in [2].

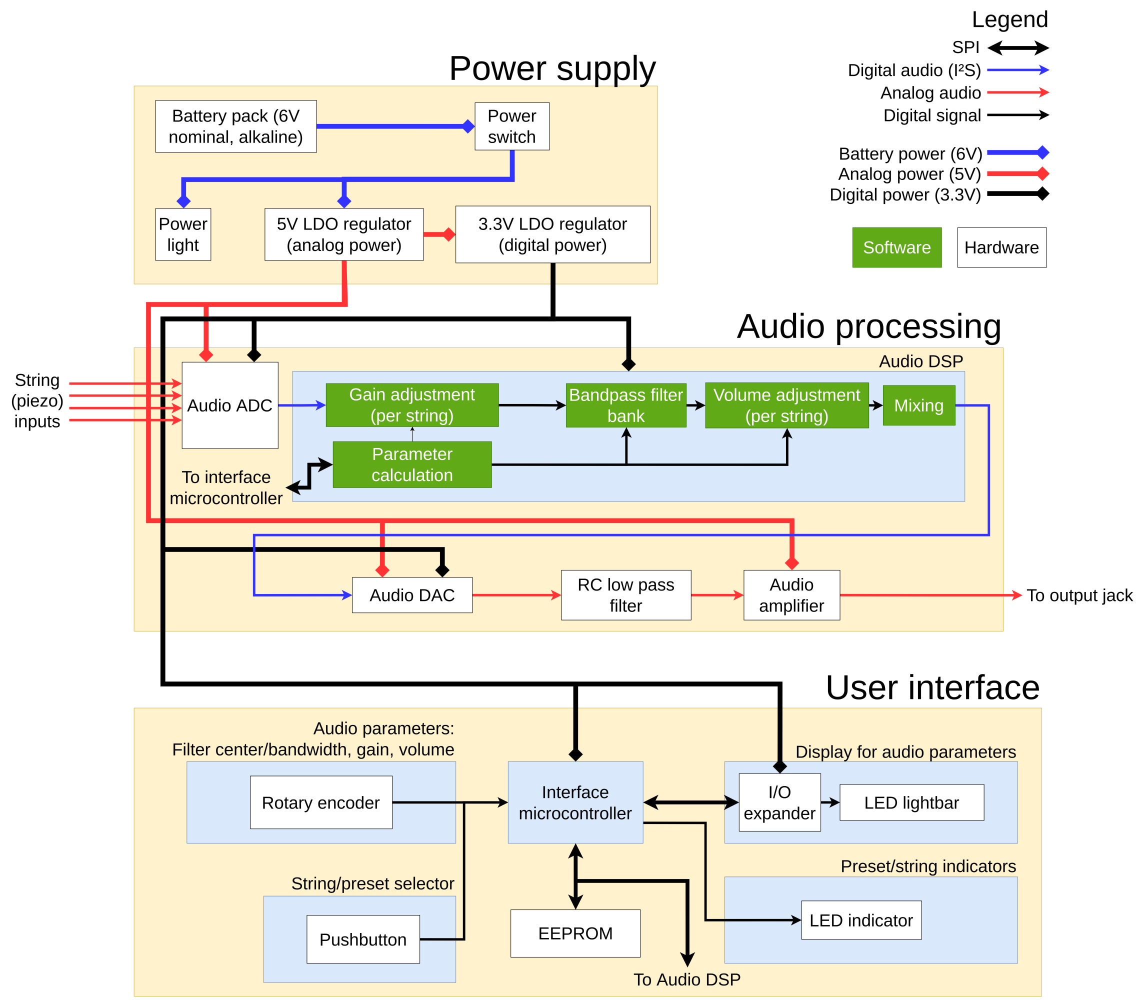


Figure 1: Block diagram of proposed solution

2 Design

We designed an audio processor which boosts/attenuates and filters each string of a four-string electric violin individually, then mixes the four string signals to the instrument output jack. The user interface of the processor allows the user to save and recall user-defined "presets" of audio parameters, to account for use with different effects chains or sound systems.

Figure 1 shows the block diagram for our design. We successfully integrated the power supply and user interface subsystems into our final product. Presently, the audio subsystem of the processor does not function due to a flaw in the software used to design the signal processing code. We will discuss this in depth later in the report.

2.1 High Level Requirements

1. The volume of each string should be adjustable with gain between $-\infty$ (mute) and $+3$ dB, and the string signal should be filtered using a bandpass filter with variable bandwidth and center frequency between $100 \pm 2\%$ and $8000\pm2\%$ Hz.

2. Two sets of audio parameters (i.e. gain, filter center frequency and bandwidth, and volume) must be able to be saved and recalled as presets using the user interface.

3. The processor must fit in a space of 150x100x50mm (±1 mm in each dimension), which is roughly the size of the decorative center bout on the Mina electric violin. The PCB should be housed in an enclosure attached to this part of the violin.

2.2 Subsystem Design

2.2.1 Power Supply

The power supply of the audio processor produces analog and digital supply voltages, to be used in the rest of the processor, from a battery pack mounted on the instrument. The analog power supply is 5 V nominal, and the digital power supply is 3.3 V nominal. The choice of separate voltages for the analog and digital sections of the design is motivated by the need to isolate the analog circuitry from noise produced, as well as design flexibility for the analog circuitry.

Originally, all analog components were intended to use the 5 V power rail. However, the DSP which we used requires 3.3 V analog power. Therefore, we included an additional 3.3 V regulator dedicated to the DSP.

To implement overcurrent protection, we included a resettable fuse with a rated trip current of 1 A in series with the positive battery connection. Given a fixed voltage, current flowing through the fuse resistively heats the fuse body, which has a positive temperature coefficient; higher temperatures give higher resistance. Therefore, if the initial current is large enough, the initial heating establishes a negative feedback loop, in which the current is reduced by the increased resistance, which then lowers the current, and so on. This effectively prevents large (steady-state) currents from flowing through the fuse.

To implement reverse polarity protection, we have opted for P-channel MOSFET protection instead of a diode. This is a fairly common design technique, but we referenced online materials such as [3] as a refresher during the design process. The MOSFET has a parasitic body diode which conducts when the battery polarity is correct. This switches on the MOSFET, which then provides a path for current to flow without the relatively large voltage drop of a forward-biased diode. The result is a more efficient protection circuit.

2.2.2 Audio Processing

This subsystem applies gain/volume adjustments and filtering to the input signals from each piezo pickup. It does this by converting the analog piezo sensor signals to digital signals, which are then mixed, filtered, and enhanced by the internal processing design implemented within our DSP, at which point the resulting signal is converted back to an analog signal and is output to an audio output jack.

For each string, the processor has a gain control prior to filtering which varies between $-\infty$ and $+3(\pm 0.5)$ dB, and the same type of gain control (called "volume") after filtering. The separation of these gain controls allows more flexibility in the sound design of each string. Piezoelectric sensors generally output a voltage signal around 200 mV_{p-p}. To boost that signal to a level such that it is usable by our DSP chip we make use of a simple preamp in the form of an op-amp charge amplifier.

After amplification, each string input, as well as any control inputs, would be passed through to the DSP IC. The signals would be converted from analog inputs to digital signals by the on-chip ADC, which would then allow the signals to be sent to the DSP for actual processing. Once this processing had been completed the processed signal would be converted through the on-chip DAC back into an analog signal to then be output through the output jack of the design to allow for play through whatever audio device the user may be using.

Internal Design

To accomplish the filtering and enhancement required by our project goals, we utilized the AKM AK7738VQ DSP chip. The AK7738VQ chip has 2 onboard 24-bit stereo ADCs for initial signal conversion. The processing of the signals is done by the chip's 2 DSPs that are capable of 28-bit floating point calculations at 2560 step/fs (when fs=48kHz) parallel processing. While the conversion for the processed signal will be done by the chip's 32-bit DAC.

The RAM-based, freely programmable nature of the AK7738VQ allowed for the DSP processing of our signals to be designed and implemented using AKM's GUI DSP design software, AKx. Our implementation of our processing design can be referenced in figure 17.

Our processing design utilized the parallel processing ability of the AK7738VQ chip to implement simultaneous processing, enhancement, and filtering of each of our 4 piezo signals simultaneously. The first component of our processing design is a limiter, acting as a redundancy for our pre-amp circuit, as well as a safe guard against any possible over-amplification. Each signal would then be equalized, filtered, and mixed based on the parameters based on any control inputs set by the user. After mixing, further user control would be possible in real-time from any user inputs to allow for an even more granular amount of user control over the sound characteristics and behavior. This signal would then again be filtered and equalized, resulting in a processed output that contains very little noise, has fully granular user control, and is quality enhanced to negate any drawbacks that may have been present due to bow noise, signal overlap, or inaccurate piezo pickup.

Design Alternatives

Our internal processing design is not possible to be adapted or changed due to the inherent need for us to process each string individually to be able to provide the granular user control and enhancement/filtering efficiency, meaning that if we were to redesign our internal processes, we would have to change our entire physical design and logic flow as well.

Therefore, the only real design alternatives for the audio subsystem were related to the use of a different DSP IC within our audio circuit. Our process for DSP selection was done by finding out the minimum range of values for our primary chip characteristics - such as sampling frequency, multiplication bit-length, and MMACs required for computation - and then looking for the most cost effective chip that could provide the required minimum performance at rated usages. We also had size constraints for our overall PCB design, so to find a DSP with integrated ADC and DAC chips was necessary to make sure that we could make our PCB design simple, and with sufficient spacing between components. These main focuses led us to most seriously consider the 3 chips compared in table 2.

ADAU1701	TIC6713	AK7738VQ
2 24-bit ADCs, SNR = 100	No integrated ADC or DAC	2 24-bit ADCs, SNR = 109
1 24-bit DAC	—	2 32-bit stereo DACs
Frequency up to 192 kHz	Frequency up to 192 kHz	Frequency up to 192 kHz
1 DSP	Best overall performance	2 DSPs and Sub DSP
<$15	>$40	<$15
Software not mentioned	Complete software suite	Standalone GUI programmer
Not found in stock	<10 in stock	In stock

Figure 2: DSP IC Options Considered

Other chips were considered but were consistently found to have stocking issues

As can be seen from the reference table, ADAU1701 and AK7738VQ were very similar in terms of performance with the only real differences being the number of internal converters, AK7738VQ having more, and the accompanying software to be used with the chip. The AK7738VQ was not only preferred over the ADAU1701 because of these extra components, but also because during the decision making process about which chip should be selected the ADAU1701 was sold out on Digikey, Mouser, and Octopart, with the restock projected to occur in October. Our chip selection was then down to two main chips, being the TIC6713 and the AK7738VQ. The actual DSP performance of the TI chip was much greater than that of the AKM chip. The TI chip also had much better computational processes and a more open design flow with much more documentation online from both TI and third party sources, however the TI chip fell short of the AKM chip in regards to its lack of internal signal converters, as well as its much higher price tag. Due to the fact that the performance of the TI chip would have been overkill, and both PCB space as well as budget considerations were at a premium, the choice between the two was easy considering the circumstances. While this means we had to make concessions in terms of documentation resources, as well as having to work with a more complex design structure, we were not in a position to be able to choose to use the better overall chip as we instead were in a position to prioritize the chip with the best performance per dollar.

We still hold that this was the correct decision given the circumstances, however, the chip set and accompanying software we tried to implement was not able to be made to function correctly. As mentioned earlier, the AK7738VQ chip has an accompanying GUI software for the programming of its DSP chips called AKx. AKx works in a drag-and-drop format to place various signal processing components that can then be modified or connected in any series or parallel configuration that the user may want to be able to accomplish the processing of their signals. The AKx software is then used to build the component design, at which point is automatically generates a set of output files that replicate the logic flow and signal processing done by the component design. These generated files are then converted into header files (OFREG, CRAM, and PRAM) that contain the instruction set and operations for the DSP chip to be able to apply the designed processes (this code generation and conversion process is visually represented by figure 18). These header files are uploaded to the DSP from the micro controller through an I2C connection, at which point the files are stored on the DSP within its RAM. This setup allows for the DSP to have persistent access to the instruction set and operations that it needs to complete. This design structure for programming and utilization of a DSP is used by multiple chip manufacturers, however the abstraction and layering of the process makes it difficult to debug if something were to go wrong.

We, in fact, did end up running into this issue. AKx has a bug somewhere in its code base, confirmed by AKM, that impacts the code generated from the AKx software between the component design step and the header file creation step. The specific cause of this particular bug has not yet been narrowed down to the usage of a specific component or a specific combination of components, but in the case of our design was, believed by AKM engineers, to be due to the nature of our design in regards to the simultaneous processing and enhancement of each of our signals separately. This was found not to be the case when our design was checked and verified by AKM engineers attempting to help us debug our design, but nevertheless we were unable to program our DSP chip due to the presence of the bug affecting our generated code and impeding the Ak7738VQ used in our design from being able to recognize and acknowledge the instruction set and operations that needed to be uploaded and implemented.

In the presence of such a blocker, the options for workarounds are few. There is the option to redesign the entire implemented processing algorithm; this was not feasible in our situation due to the need for us to process our signals in a specific way to allow for our user controls and associated processes. There is the option to work through the generated code and debug/hardcode in the correct instructions and operations; technically feasible but incredibly difficult and time consuming as we were working with assembly code and would have had to go through each instruction and operation by hand - on files of thousands of lines - to try to ensure that not only the logic flow of each step was correct but also that it was being initialized on the correct register and that it resulted in the correct operative values. And lastly, when faced with such a problem, there is the option to abandon working with the specific affected chip set, and design a new platform around a different DSP chip.

In our case, we would have exercised the third option of choosing a different DSP to design our platform around. If we were to reattempt this project, with the knowledge that we have now, we would have selected the TIC6713 (or another similar TI chip) as the base of our audio processing subsystem. While the TI chips considered at the beginning of the project did not have dedicated signal converters built in, and while all of the TI chips had a higher cost, we believe that the more open design framework would be well worth the cost, as our algorithm could utilize any design flow and could be implemented in many different configurations. We also believe that the massive amount of accessible documentation pertaining to the usage of, and development with, TI DSP chips would give us much more insight into how to optimize our processor design, as well as providing us with many more resources if we had to debug anything.

2.3 User Interface

This subsystem provides controls for the users to change the gain, filter center frequency, and volume for each string, and to save combinations of these parameters as presets. To do this, the interface has four rotary encoders that respectively controls the gain, filter center frequency, filter bandwidth, and volume of the active string. A initial sketch of the user interface is displayed in Fig 3.

The rotary encoders have a built in button that switches through the active string, and four LED indicators to show which string is active. In a similar way, one of the buttons is responsible for rotating between the active preset, and there are two LEDs that indicate which preset the user is working with. A set of four light bars with eight LEDs each is included to indicate the current intensity of each parameter. These light bars increase and decrease according to the turning of the rotary encoder.

The presets are saved to the memory each time interval and one preset is automatically loaded to the program once the user interface powers up. We chose EEPROM as our memory which has enough memory size to save two pairs of four parameter ranges.

3 Cost and Schedule

Table 3 shows the estimated cost of components for the current power supply and user interface design (not including the cost of the enclosure and circuit board). The audio processor design is still tentative, so the parts for it are not included.

The prices are from Digikey and are current as of May 4, 2022. Some of the unit prices are at higher price breaks since we purchased more than were needed for a single prototype. This prevented us from becoming bottlenecked due to lack of parts. We do not include the enclosure in the cost since it is likely to change if we continue developing the design.

The estimated cost of labor for us is given in Table 1, on the assumption that the total price of labor is 2.5 times the hourly wage times the number of hours worked to design and assemble the prototype. As the final layout of the user interface is not yet determined, we are currently unable to estimate the cost of labor for the machine shop.

The schedule we followed to design the audio processor is given in Table 2.

Contributor	Hourly wage ($)	Labor hours (estimated)	Total labor cost ($)
Wei	40	210	8400
Alex	40	220	8800
Scott	40	200	8000
Total	120	630	25,200

Table 1: Estimated labor cost for the three designers.

Task	Assignee	Date
Complete Audio DSP schematic	Scott	2022-02-24
Complete PCB layout	Scott, Wei	2022-02-28
Create interface mechanical layout	Wei	2022-03-05
Design user interface firmware	Alex	2022-03-12
Design audio DSP firmware	Scott	2022-03-12
First-run board assembly	Wei	2022-03-28
Debug user interface	Wei, Alex	2022-04-05
Second-run board assembly	Wei	2022-04-11
Validate audio processing	Scott	2022-04-13
Integrate prototype	Wei	2022-04-18

Table 2: Final project schedule.

Item	Part no.	Unit cost ($)	Qty	Extended cost ($)
0.1 μF capacitors	06035C104KAT2A	0.0239	27	0.6453
10 μF electrolytic capacitors	106BPS050M	0.293	2	0.586
4700 pF film capacitors	474MWR630K	1.05	5	5.25
0.47 μF film capacitors	474MWR100K	1.05	1	1.05
1500 pF capacitors	VJ0603Y152JXACW1BC	0.092	4	0.368
2.2 μF capacitors	CM105X5R225K16AT	0.156	2	0.312
8 pF capacitors (NP0)	CC0603CRNPO9BN8R0	0.107	2	0.214
Amber LEDs	IN-S63AT5A	0.186	39	8.254
1 A resettable polyfuse	0ZCJ0050AF2E	0.177	1	0.177
Multi DSP with ADC and DAC	AK7738VQ	12.40	1	12.40
2-pin keyed header (pins)	LHA-02-TS	0.074	2	0.148
2-pin MTA connector (sockets)	3-641535-2	2.05	2	4.10
5-pin keyed header (pins)	0022292051	1.06	1	1.06
5-pin MTA connector (sockets)	3-640441-5	0.47	1	0.47
10-pin JTAG header	3221-10-0100-00	0.615	1	0.615
P-channel MOSFET	DMP2123L-7	0.433	1	0.433
3.3V regulator	XC6227C331PR-G	0.833	2	1.666
5V regulator	NCP1117IDT50T4G	0.593	1	0.593
10 kΩ resistors	RMCF0603JG10K0	0.0061	14	0.085
1 kΩ resistors	AC0603JR-101KL	0.024	1	0.024
1 MΩ resistors	RMCF0603FG1M00	0.017	4	0.068
4.7 kΩ resistors	RMCF0603FT4K70	0.017	4	0.068
20 kΩ resistors	RNCP0603FTD20K0	0.061	4	0.244
270 Ω resistors	RMCF0603FT270R	0.0068	39	0.2652
EEPROM	BR25H010FVT-2CE2	0.482	1	4.82
Microcontroller	MKL43Z128VLH4	7.11	1	7.11
I/O expander	MCP23S08-E/SO	1.40	4	5.60
Audio amplifier	TPA6111A2DR	1.27	1	1.27
Op-amp	TL072HIDDFR	0.50	2	1.00
Tactile pushbutton	TL3315NF100Q	0.144	1	0.144
Rotary encoders	PEC11R-4120K-S0018	0.938	4	3.75
16 MHz crystal	NX5032GA-16MHZ-STD-CSK-8	0.581	1	0.581
4x AAA battery holder		1.87	1	1.87
Slide switch		1.13	1	1.13
2.5 mm audio plugs	MP-2511	0.42	4	1.68
2.5 mm audio jacks	MJ-2508	0.92	4	3.68
1/4 in. audio jack	PC12A	3.84	1	3.84
Total (to next highest cent)	-	-	-	75.58

Table 3: List of components with itemized costs.

4 Verification

4.1 Power Supply

Table 4 shows the requirements and verification steps for this subsystem.

Requirement	Verification steps
The power supply for the audio processor should regulate the 6V nominal battery voltage to a 5 ± 0.25V analog and 3.3 ± 0.17 digital power supply.	1. Connect a 6±0.3V power supply to the battery voltage input (this range simulates fully charged and almost-dead batteries). 2. Verify that the 5V regulator outputs 5 ± 0.25 V on its output voltage pin. 3. Verify that the 3.3V regulator outputs 3.3 ± 0.17 v on its output voltage pin.
The power supply should be able to provide 500±25 mA to all components (assuming 300 ± 15 mA allotted to digital components and 200 ± 10 mA to analog).	1. Turn on all 39 LEDs in the user interface, and program the microcontrollers to do something computationally intensive. 2. Verify that the current drawn from the battery input by the 5V regulator is 300 ± 15 mA.
The power supply should protect from reverse-polarity and overcurrent events. Overcurrent is defined as drawing 1±0.1 A or more from the battery input.	1. Connect a 6±0.3 V power supply to the battery input. 2. Connect a $6 \pm 1\%$ power resistor between the input to the 5V regulator and ground. 3. Verify that the power light turns off, indicating that power is lost to the rest of the audio processor.

Table 4: Requirements and verification for power supply.

Upon connection to a bench power supply set to 6 V, the 5V rail measures 4.99482 V, and the 3V3 measures 3.31737 V. These are within stated tolerances. By reversing the power connection from the bench power supply to create a reverse polarity condition, we measured -0.977 V at the 5V rail and -0.7061 V at the 3V3 rail. When the power supply polarity was "corrected", the rest of the circuitry powered on normally with no signs of damage.

With 35 of the 39 LEDs in the user interface turned on, the current consumption as measured by the bench power supply is 191 mA. This is the maximum number of LEDs which can be turned on. Each LED is current-limited to 5 mA, so if all 39 LEDs were turned on, then the total current consumption would be around 211 mA, still within acceptable limits as given in the requirements table.

When a 6 Ω (nominal) resistor was used to draw 1.037 A through the polyfuse, the rest of the circuit did not turn off. The 3V3 rail was measured to be 3.31356 V. However, when a wire jumper was used instead of the resistor, the 3V3 rail dropped to -0.16 V, with a steady-state current around 160 mA. This indicates that the overcurrent protection still works when a short circuit is present, but that the 1 A definition of overcurrent may be flawed.

4.2 User Interface

Table 5 shows the requirements and verification steps for the user interface.

Requirement	Verification steps
The rotary encoders must be able to control the gain, filter center frequency, and the volume of each string.	1. Slowly turn each knob while monitoring the output signal to verify that the output signal actually matches with the implemented range. 2. Verify with the status LED bars that will light up more on the light bars as each control parameters increase with the turning of each rotary encoder.
The button to switch between strings must output correct signal to activate the right string.	1. Connect the button to LED indicators and verify that the each LED light up in correct order on each push of the button
The button to switch between presets must load the correct preset that is saved in the SPI external memory.	1. Save the two presets with extreme parameters. For example, a full gauge of each parameters for the first preset and zero gauge for the second preset. 2. Connect the button to LED light bar indicators and verify that the light bars fully lights up and turns off on each push of the button. 3. Verify that the two LEDs for the preset notification lights up accordingly on each push of the button.

Table 5: Requirements and verification for user interface.

After loading the program to the user interface subsystem, we verified that the peripherals are working as expected. Starting with the rotary encoder, the pulse signals generated from the encoder are successfully recorded in the internal state within the program and is reflected the LED light bars.

After matching the pin addresses of each string with the buttons, the LEDs that indicate which string the user is working on correctly lights up upon the push of the built in buttons of rotary encoders.

After changing the parameters to extreme conditions for visibility, presets are correctly saved to and are loaded from the EEPROM. After scrambling up the current light bar conditions and pressing the preset button, the light bars light up according to the previous condition that is currently saved in the EEPROM.

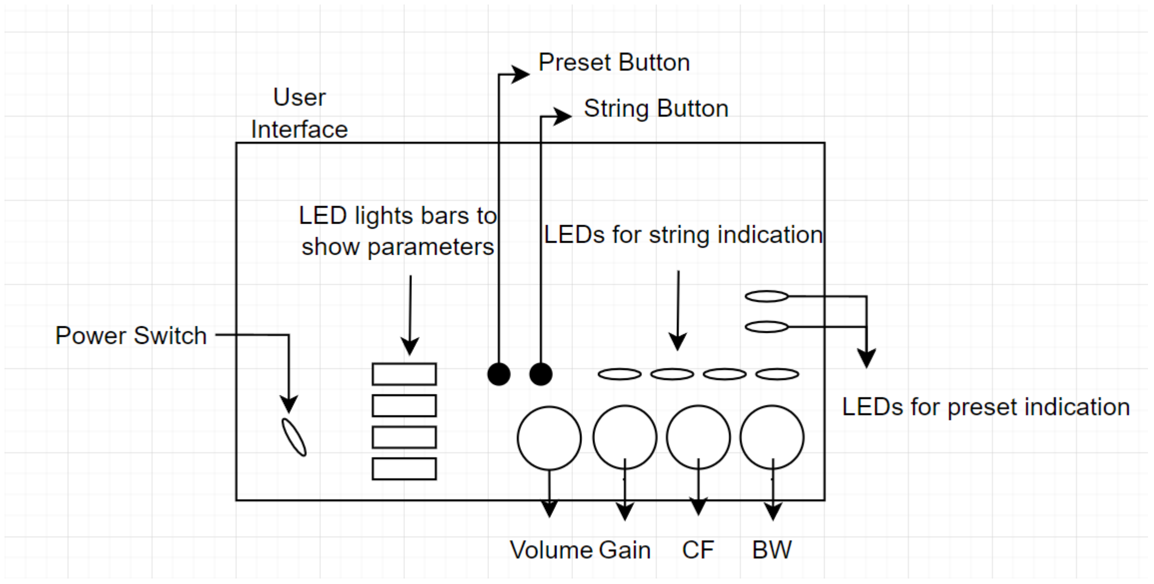

Preset Button
User Interface
String Button
LED lights bars to show parameters
LEDs for string indication
Power Switch
LEDs for preset indication
Volume
Gain
CF
BW

Figure 3: Sketch of proposed user interface

4.3 Audio Processing

Table 6 shows the requirements and verification steps for this subsystem.

Requirement	Verification steps
The audio processor must have a variable center frequency between $100 \pm 2\%$ and $8000 \pm 2\%$ Hz.	1. Play a white noise signal into one channel of the audio processor. 2. Configure the channel gain and volume to unity, and the filter to center frequency 100 ± 2 Hz and bandwidth of 1 ± 0.1 octaves. 3. Verify with oscilloscope FFT that the spectral peak of the output signal occurs at 440 ± 2 Hz and the half-power points are 1 ± 0.1 octaves apart. 4. Play a sinusoid signal which sweeps linearly from 100 to 8000 Hz over 5 seconds into the processor. 5. Verify in the time domain that the output amplitude is highest at time $\frac{7900 f_0}{5} \pm 10\%$, where $f_0 = 100 \pm 2$ is the center frequency. 6. Repeat these steps for 440 ± 9 and 8000 ± 160 Hz center frequency.
The audio processing must introduce latency of no more than 100 ± 10 milliseconds between the input and output audio streams.	1. Play an impulse (click) into a channel of the audio processor. 2. Verify on an oscilloscope that the impulse response at the output of the processor begins no later than 100 ± 10 ms after the impulse input begins. The beginning of a signal is defined as the first time when the signal amplitude reaches 5% of its peak value.

Table 6: Requirements and verification for audio processing.

Due to the bug present in the AKx code generation process our DSP was not able to be programmed as it was not able to recognize and acknowledge the necessary files that needed to be uploaded and programmed to the device to result in full operation. Due to the DSP being effectively bricked by the software bugs present we were unable to test and verify the correct function of the DSP and the audio processor subsystem. We were also unable to simulate the correct behaviors and responses of the designed audio processor algorithm as AKx does not have any built in simulation functionality, and to simulate our algorithm through other means would not be able to verify functionality as it would not be the implementation and processing methods that would be getting simulated, but it would be the algorithm itself, and the algorithm would always respond to the input signals and parameters as expected. We can however verify the meeting of our requirement criteria sans the DSP. Therefore we will verify the amount of granular control over the center frequency, as well as the latency introduced by the ADC and DAC. These verifications should not be referred to as exact, nor as a claim that our audio processing subsystem is infalible, as instead we are attempting to exemplify what a mathermatical verification of our declared sybsytem requirements would look like without taking into account the latency, or possibly frequency control error margin, that would be induced by the signal processing algorithm or by the DSP chip itself.

In this situation, the latency of the ADC and DAC would be the most impactful (latency induced by copper traces on PCB are negligable) when finding tolerances for the delay. The latencies for both the ADC and DAC are related directly to the sampling frequency used. In the case of the AK7738VQ, the latency induced by the ADC is Latency $= \frac{5}{F_s}s$ where $F_s = 48$ kHz, therefore

$$Latency = 0.104ms$$

While the same process can be done for the DAC which results in a latency of

$$Latency = \frac{6.6667}{48\,kHz} = .139ms$$

Using the results found, even if inducing a load on every converter available on the AK7738VQ chip, the maximum latency we find being introduced into our system is

$$Latency_{max} = \frac{10\,ms}{3_{ADC} + 1_{DAC}} = \frac{10}{4} = 2.5\,ms$$

With a latency this small, it can be assumed that, even with the heaviest load of signals possible being converted simultaneously over multiple ADCs and DACs, the latency is still unnoticeable. The idea could then be extended that, as it would be almost outside the realm of possibility for the DSP and the processing algorithm to introduce 7.5ms of latency themselves (based on the computational power and speed of the DSP), the latency of the entire system would be well within its required range, even when including the DSP.

We could also attempt to appease the verification of our sampling frequency requirement in a similar fashion. The sampling rate we require for our signal fidelity and integrity to meet our expectations is that of 44.8 kHz. Our usable sampling frequencies are much higher, but even just using the common sampling frequency of 48 kHz, would then allow for tolerances of

$$\frac{48 - 44.8}{48} = 0.06667$$

resulting in a tolerance range of $\pm 6.667\%$ toward respective frequencies, a tolerance range that well encompasses the magnitude of the needed tolerance range of $\pm 2\%$ to meet our stated requirement.

5 Ethics and Safety

This project presents few safety concerns. All voltages used are 6V or less, using alkaline battery chemistry. Once placed in an enclosure, there would be no meaningful contact between the user and the circuits inside. We have included a reverse polarity protection MOSFET in the power supply design to guard against the possibility of a user installing batteries backward. All capacitors are used are either electrolytic or ceramic and will create an open circuit in the event of a failure.

This project presents few significant ethical concerns. Following section 1.5 of the ACM Code of Ethics [4], we must acknowledge that similar prior work exists. Notably, the Strados electric violin, created by ZETA Violins [5], uses an internal active preamplifier system which allows the volume of each string and the overall gain to be adjusted manually [6]. ZETA calls this pickup system "patented" on their website, but does not list a patent number; additionally, we could not find a relevant patent upon searching for "ZETA violin" in Google Patents. Therefore, we cannot immediately confirm whether this patent exists, let alone if the product is still under patent protection. This may pose an obstacle if we were to commercialize the project in the future.

Our pickup design is inspired by Richard Barbera's design, "Resonant pick-up system" [7]. The patent expired over a decade ago, and since our bridge is not carved from wood, it is unlikely that our design would be infringing on this patent anyway.

6 Conclusion

We successfully designed and implemented a battery power supply and user interface for an electric violin-mounted effects processor. However, we were not able to make the audio subsystem work as intended due to issues with DSP code. Further research and work is needed to find a DSP chip which will support our intended design. Once this is done, the audio subsystem can be fully implemented to realize our vision for the effects processor using a multi-transducer bridge pickup.

References

[1] H. Reich, *I Had to Build a Custom Mute Switch for my Violin.* YouTube, Jun 2019. [Online]. Available: https://www.youtube.com/watch?v=oYsp7OIMFAs

[2] S. Tsuchiya, “What’s wrong with dominant e?” Jun 2008. [Online]. Available: https://www.violinist.com/discussion/archive/14090/

[3] “Reverse polarity voltage protection using p-channel mosfet.” [Online]. Available: https://electronics.stackexchange.com/questions/588808/reverse-polarity-voltage-protection-using-p-mosfet

[4] “Acm code of ethics and professional conduct,” Jun 2018. [Online]. Available: https://www.acm.org/code-of-ethics

[5] “Strados modern - zeta violins: Electric violins cello bass: Zeta mandolins: Pickups repairs,” Feb 2016. [Online]. Available: https://zetaviolins.com/strados-modern

[6] “Emg mxrp-5 internal preamp for zeta violins,” Jan 2021. [Online]. Available: https://www.electricviolinshop.com/emg-mxrp-5-preamp

[7] R. Barbera, “Resonant pick-up system,” Sep 1989, uS4867027A.

A Circuit Schematics

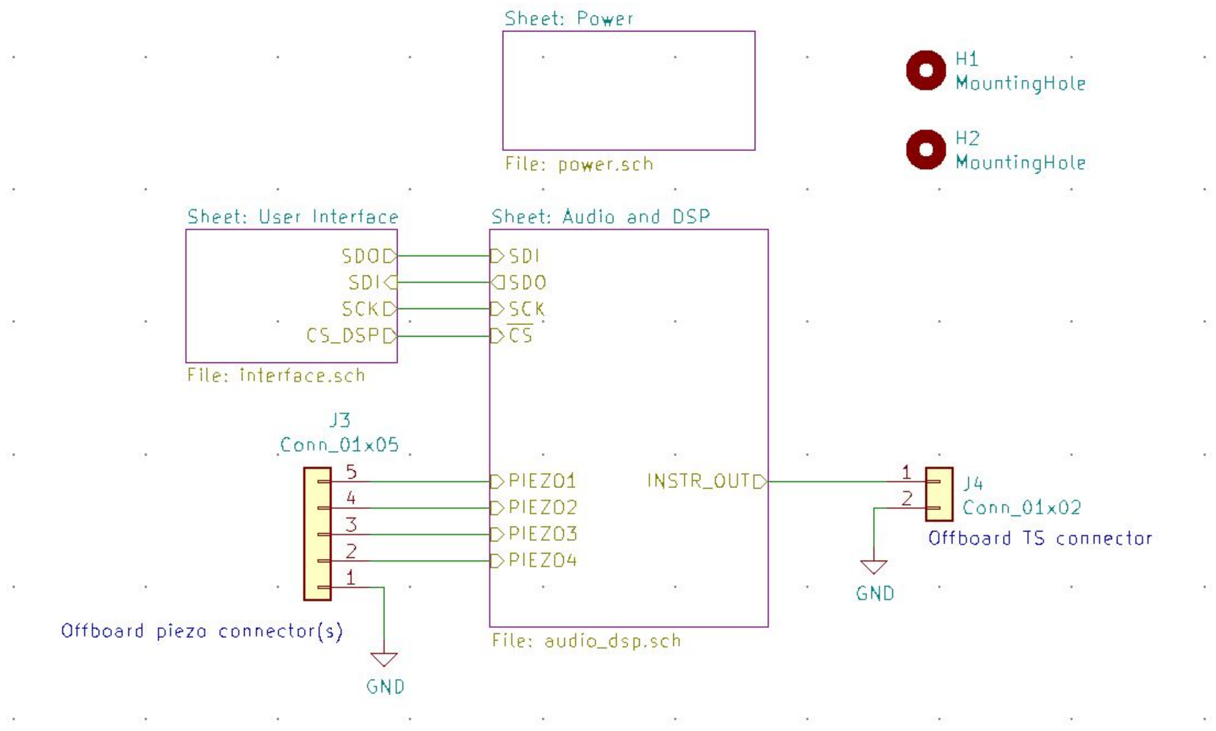


Figure 4: High-level subsystems.

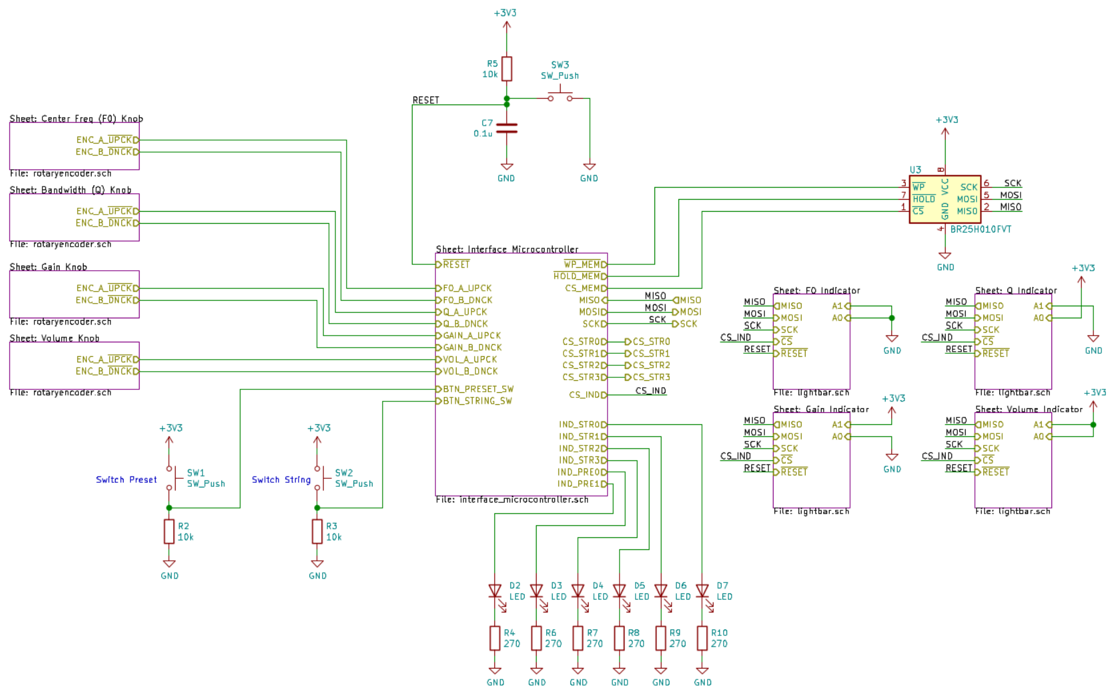


Figure 5: User interface overview.

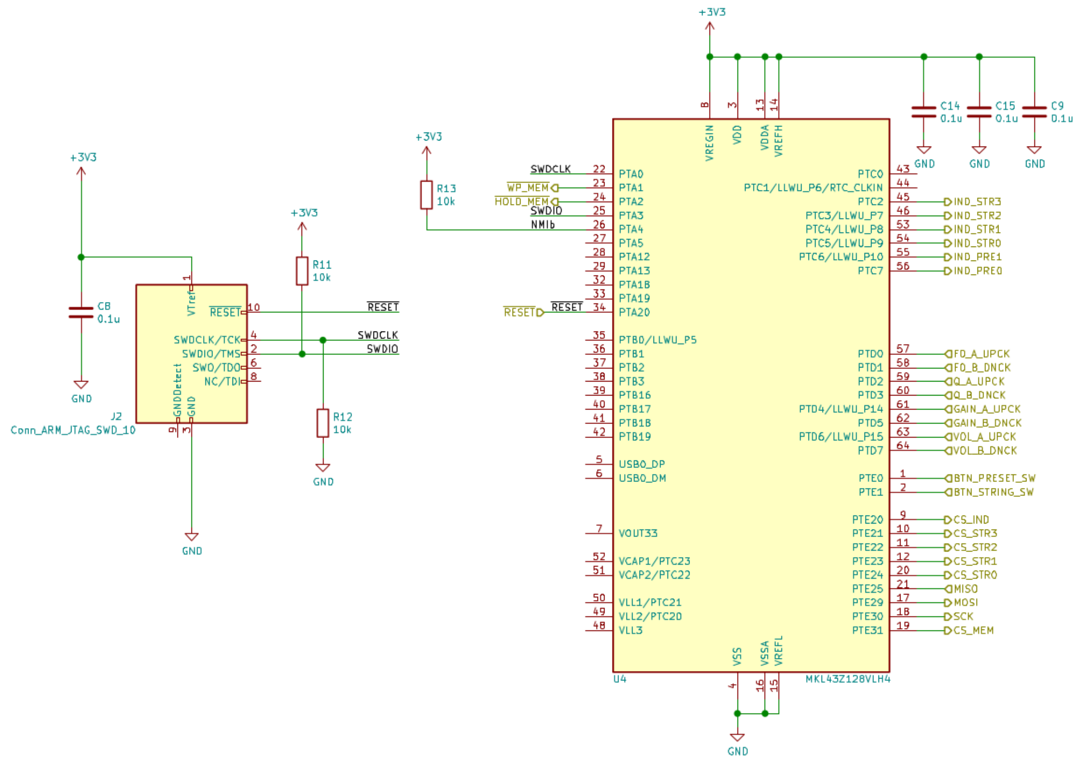


Figure 6: User interface microcontroller.

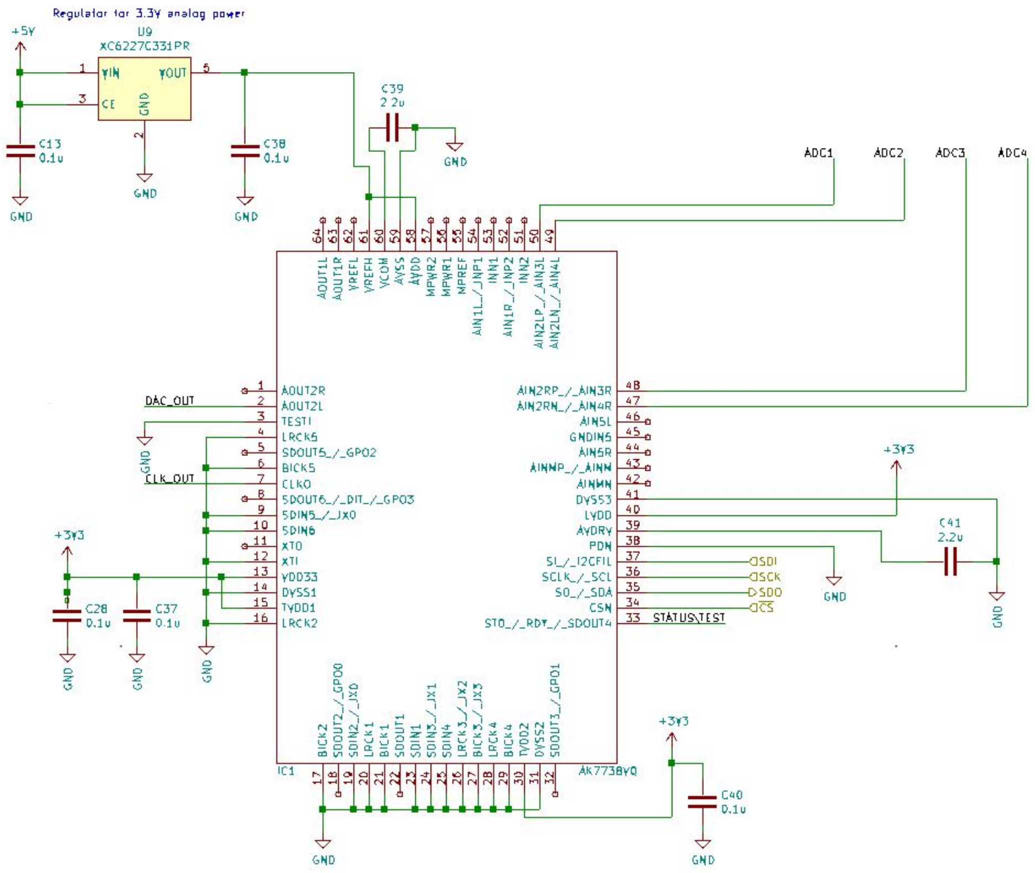


Figure 7: AK7738VQ DSP Circuit

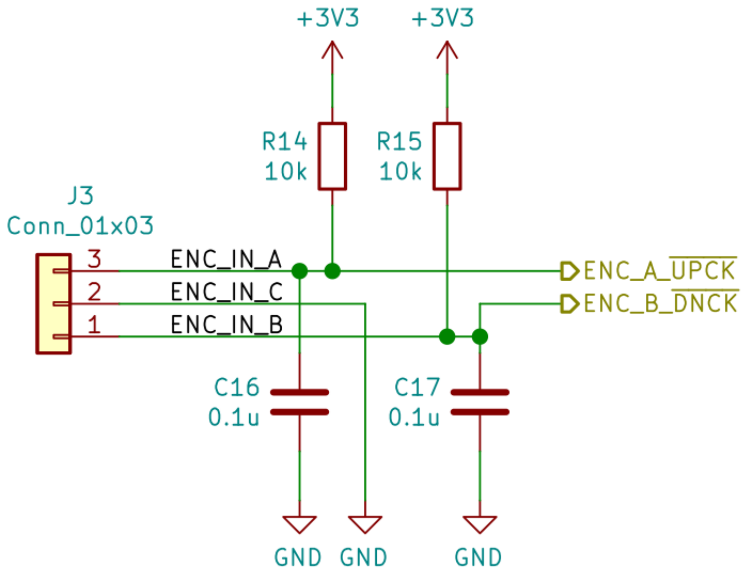


Figure 8: Rotary encoders and quadrature decoder.

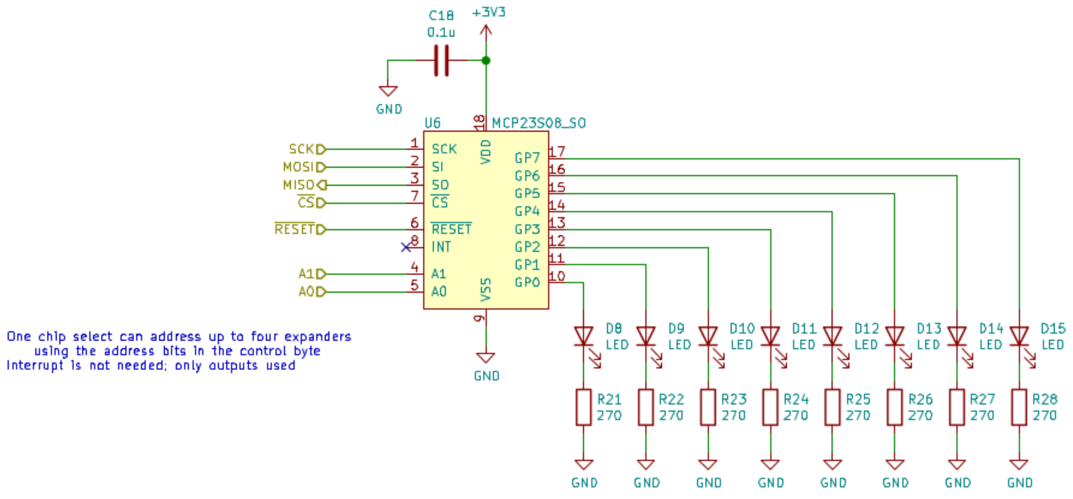


Figure 9: Lightbar indicator, used for audio parameters.

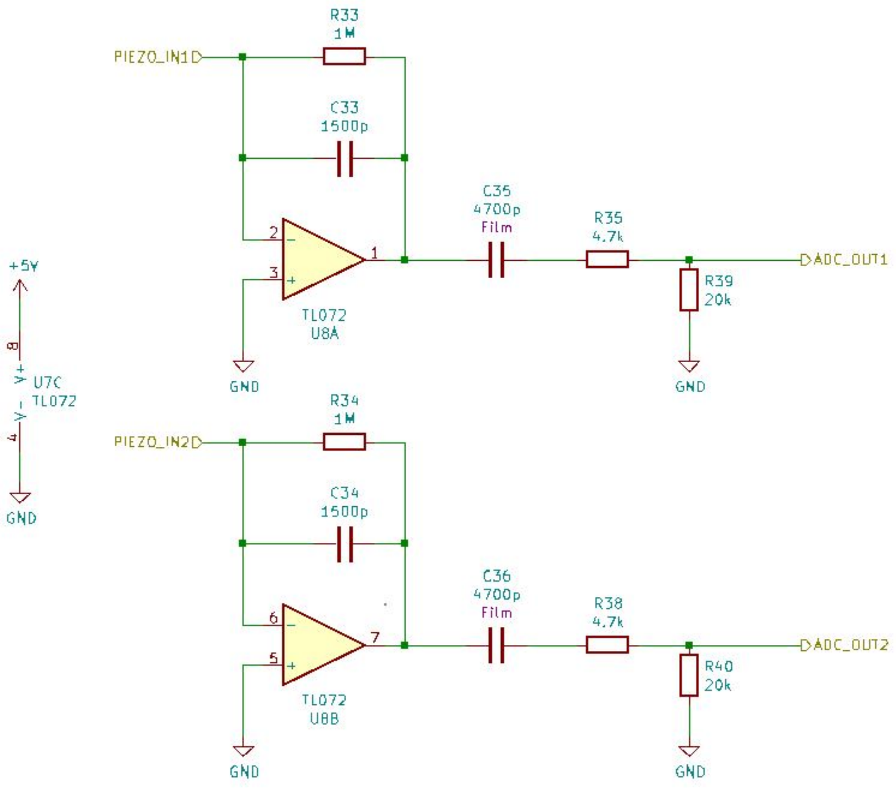


Figure 10: Preamp circuit schematic

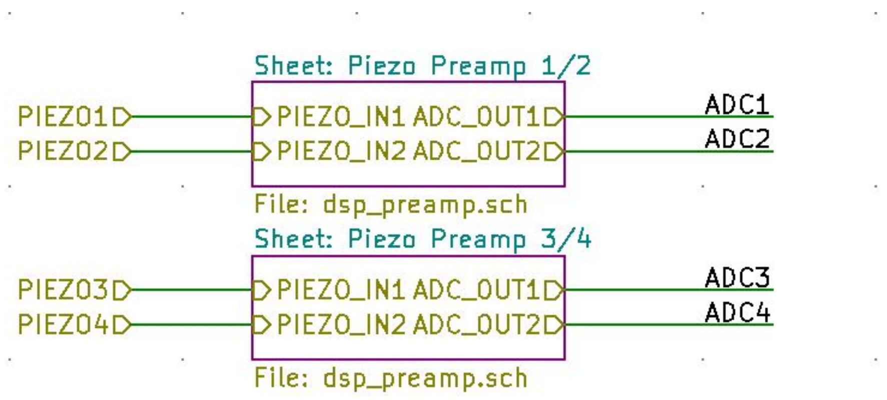


Figure 11: Modularized preamp circuits showing inputs and outputs

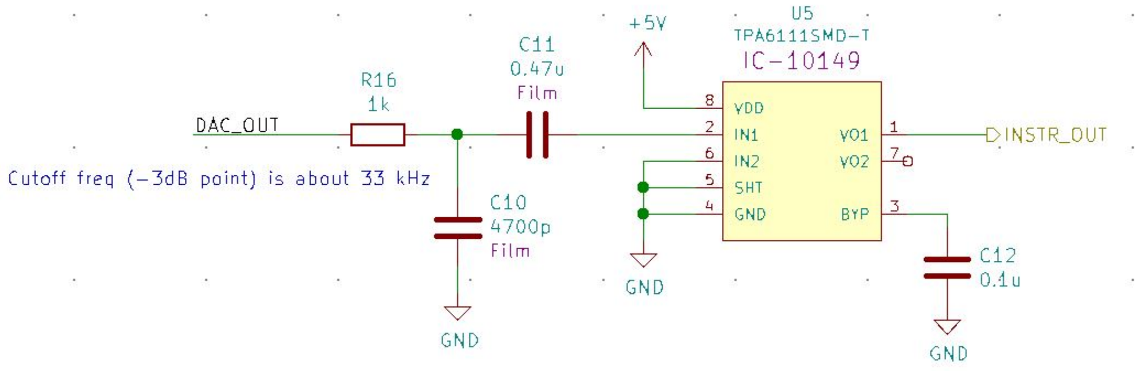


Figure 12: DAC output circuit

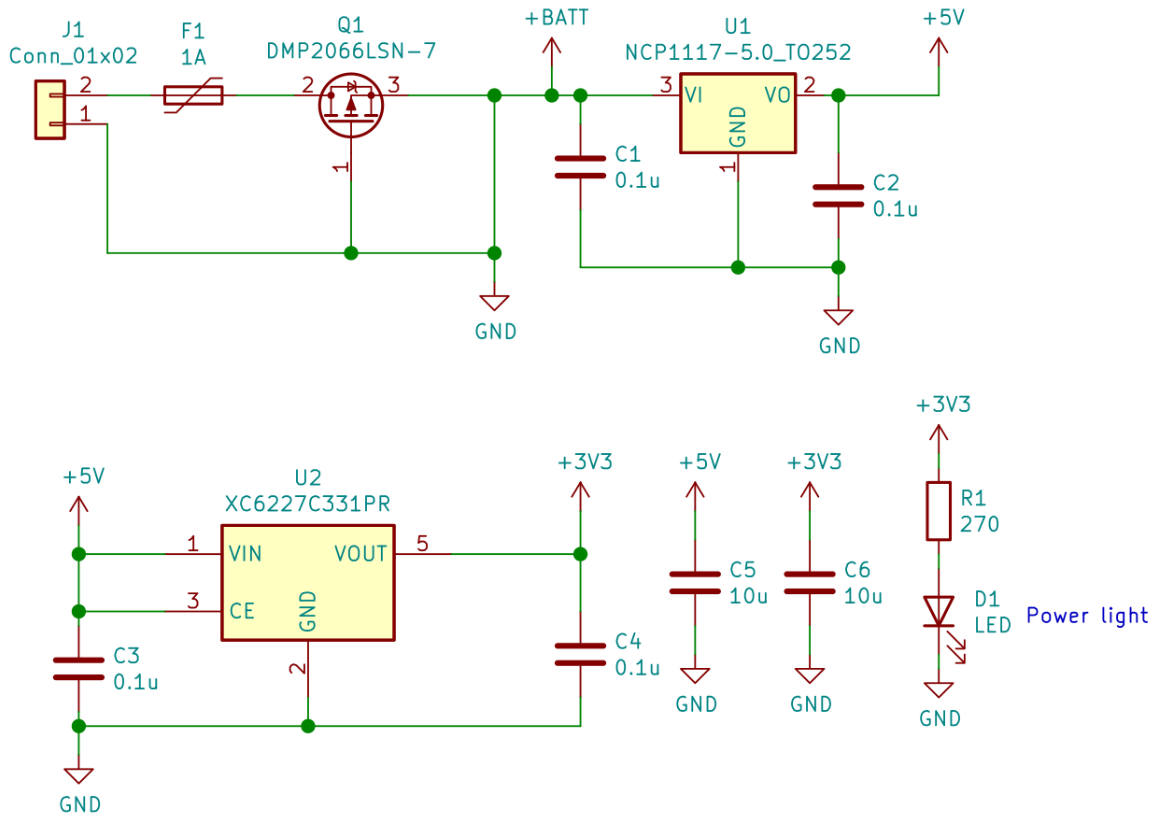


Figure 13: Power supply.

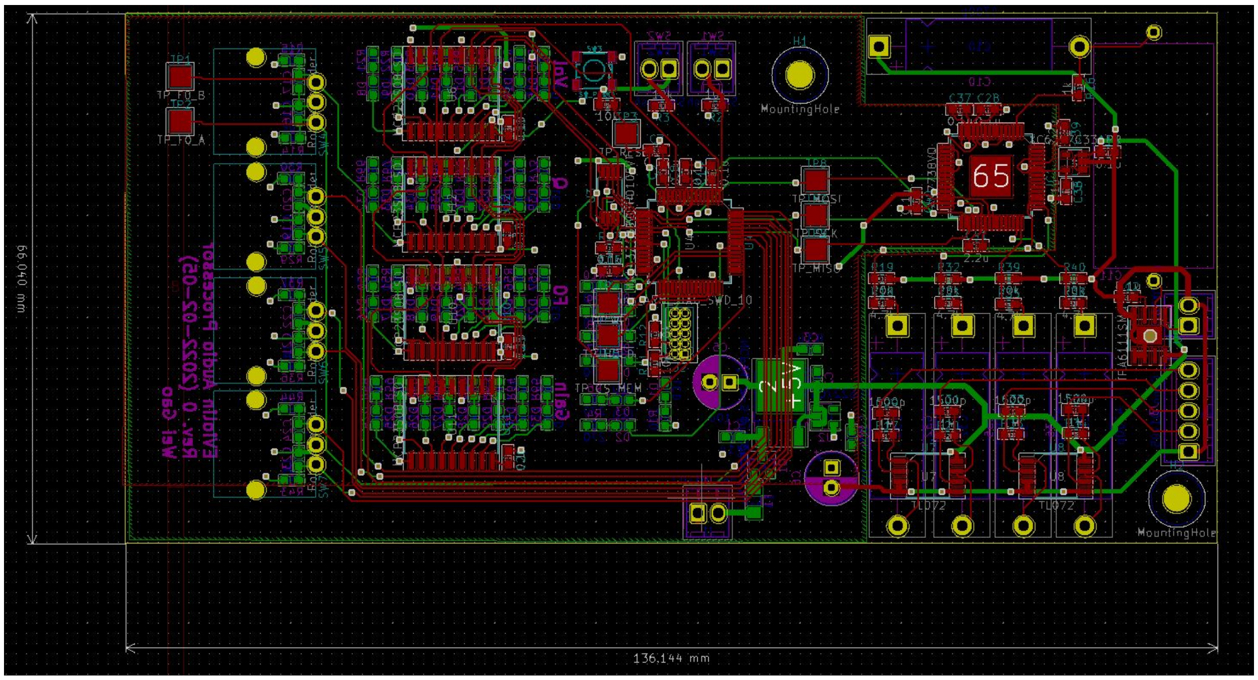


Figure 14: PCB including components and routing

B AK7738 and AKx

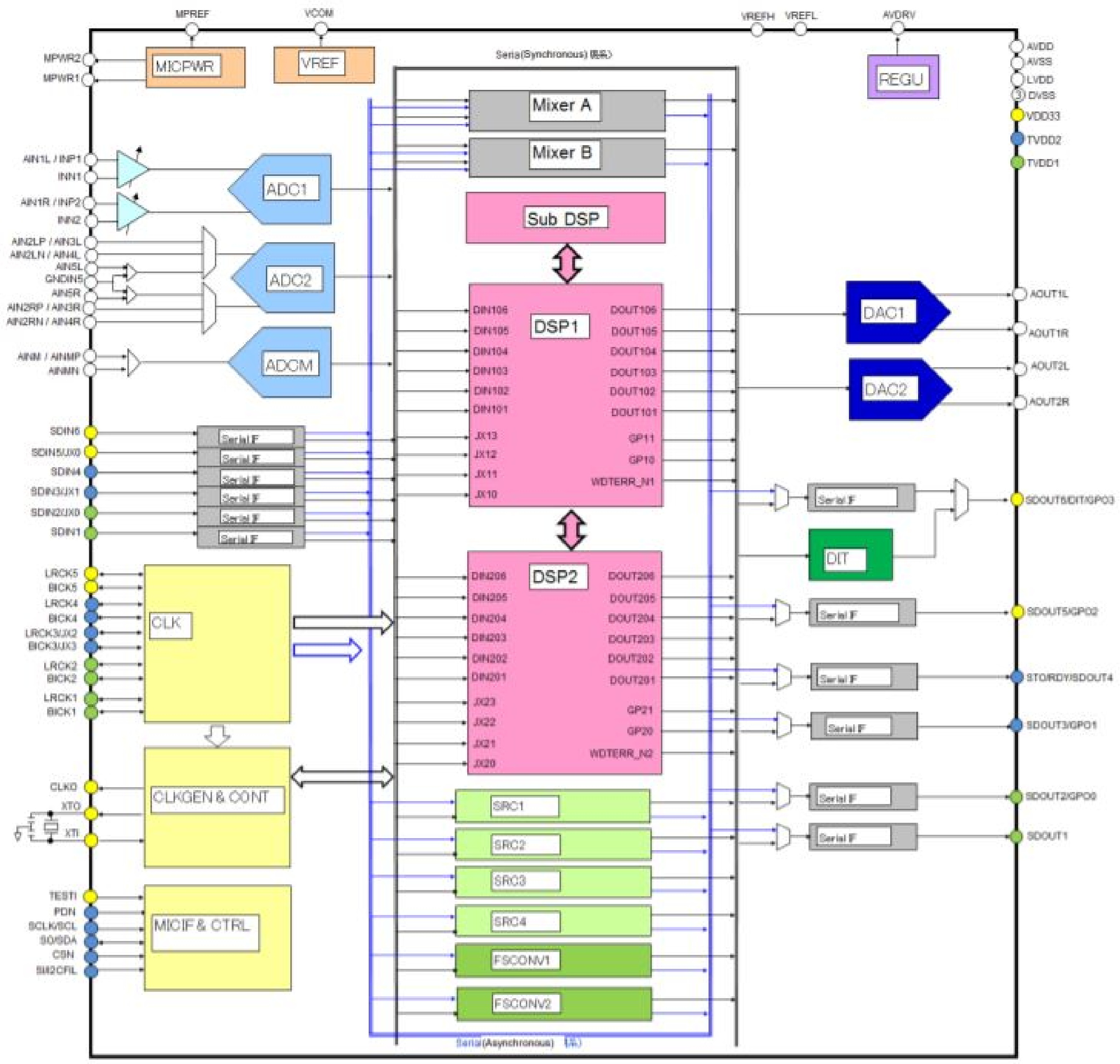


Figure 15: AK7738VQ Block Diagram

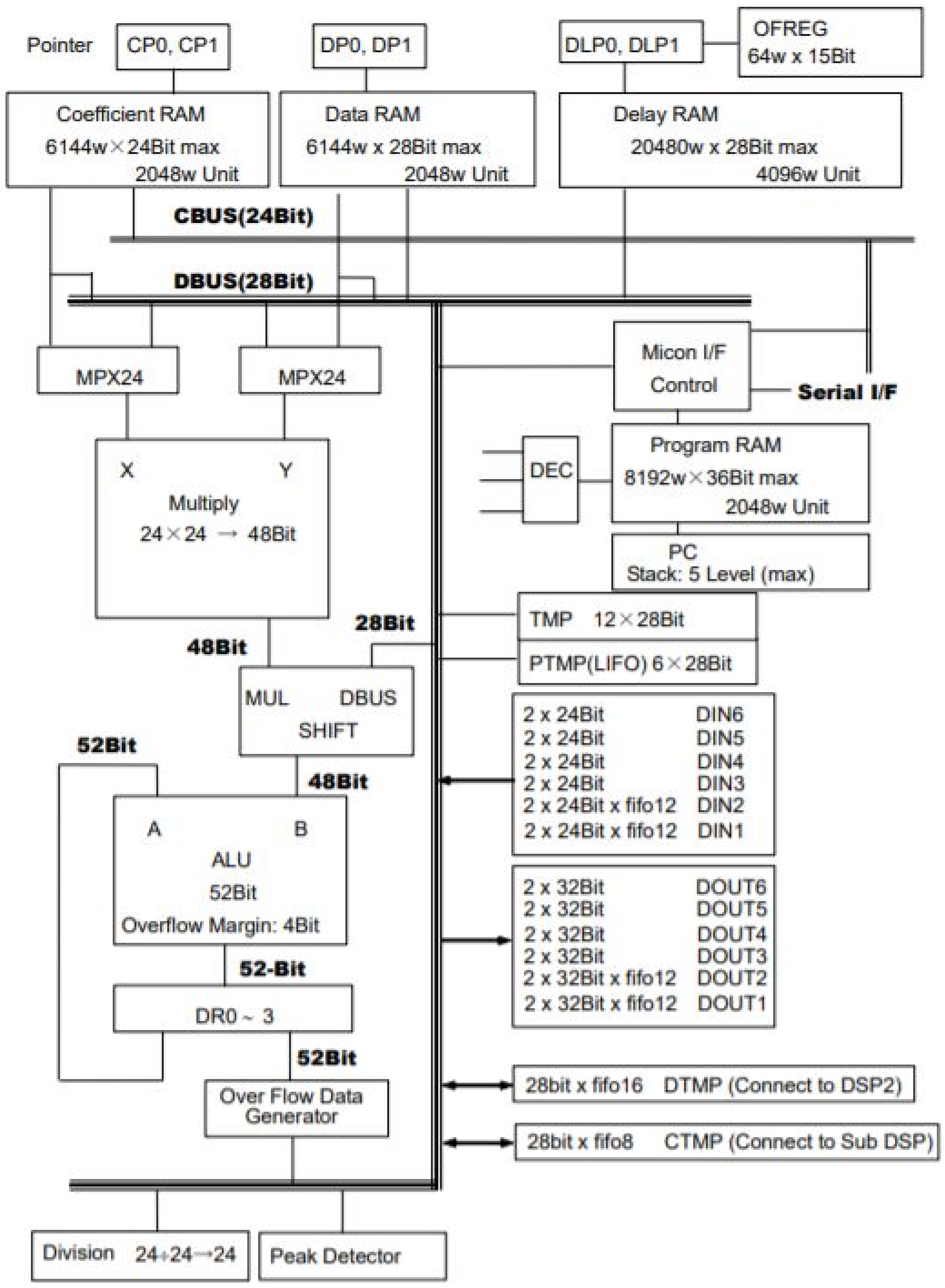


Figure 16: AK7738VQ DSP Block Diagram

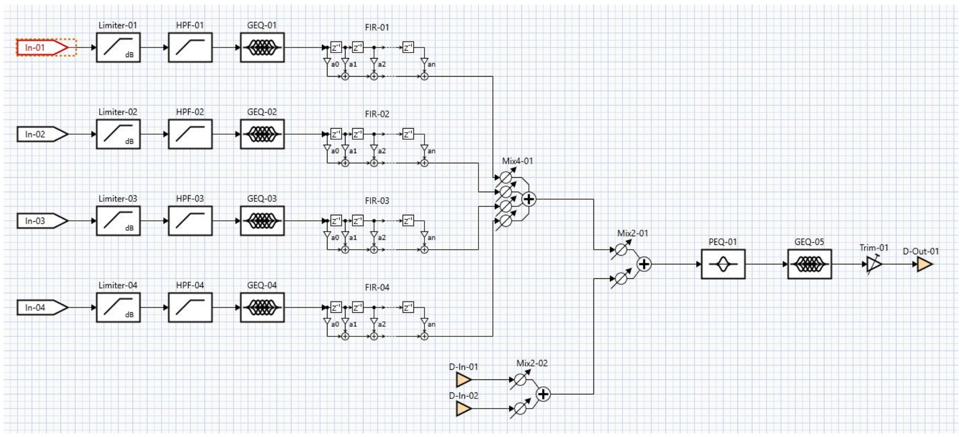
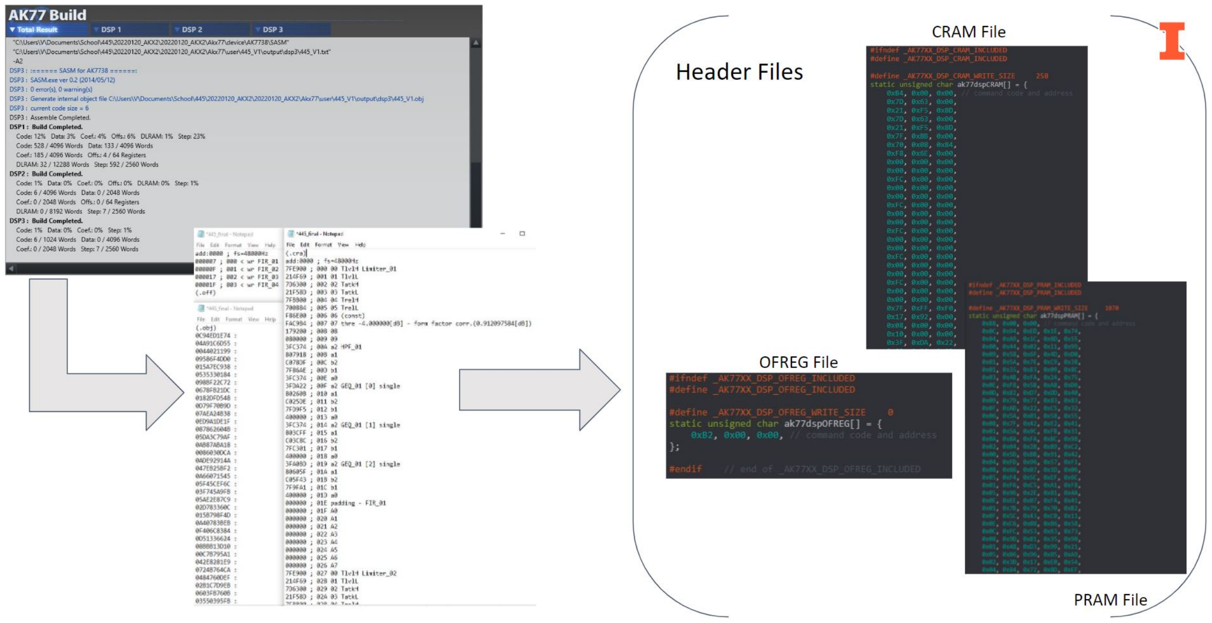


Figure 17: Internal DSP Processing Design



```
"C:\Users\V\Documents\School\445\20220120_AKX2\20220120_AKX2\Akx77\device\AK7738\SASM"
"C:\Users\V\Documents\School\445\20220120_AKX2\20220120_AKX2\Akx77\user\445_V1\output\dsp3\445_V1.txt"
-A2
DSP3 : :====== SASM for AK7738 ======:
DSP3 : SASM.exe ver 0.2 (2014/05/12)
DSP3 : 0 error(s), 0 warning(s)
DSP3 : Generate internal object file C:\Users\V\Documents\School\445\20220120_AKX2\20220120_AKX2\Akx77\user\445_V1\output\dsp3\445_V1.obj
DSP3 : current code size = 6
DSP3 : Assemble Completed.
DSP1 : Build Completed.
  Code: 12%  Data: 3%  Coef.: 4%  Offs.: 6%  DLRAM: 1%  Step: 23%
  Code: 528 / 4096 Words  Data: 133 / 4096 Words
  Coef.: 185 / 4096 Words  Offs.: 4 / 64 Registers
  DLRAM: 32 / 12288 Words  Step: 592 / 2560 Words
DSP2 : Build Completed.
  Code: 1%  Data: 0%  Coef.: 0%  Offs.: 0%  DLRAM: 0%  Step: 1%
  Code: 6 / 4096 Words  Data: 0 / 2048 Words
  Coef.: 0 / 2048 Words  Offs.: 0 / 64 Registers
  DLRAM: 0 / 8192 Words  Step: 7 / 2560 Words
DSP3 : Build Completed.
  Code: 1%  Data: 0%  Coef.: 0%  Step: 1%
  Code: 6 / 1024 Words  Data: 0 / 4096 Words
  Coef.: 0 / 2048 Words  Step: 7 / 2560 Words
```

*445_final - Notepad

```
add:0000 ; fs=48000Hz
000007 ; 000 < wr FIR_01
00000F ; 001 < wr FIR_02
000017 ; 002 < wr FIR_03
00001F ; 003 < wr FIR_04
(.off)
```

*445_final - Notepad

```
(.obj)
0C94ED1E74 :
04A91C6D55 :
0044021199 :
09586F4DD0 :
015A7EC938 :
0535330184 :
0988F22C72 :
0678FB21DC :
0182DFD548 :
0079F7089D :
07AEA24838 :
0ED9A1DE1F :
087862604B :
05DA3C79AF :
04B87ABA18 :
00860300CA :
0ADE92914A :
047EB258F2 :
0A66071545 :
05F45CEF6C :
03F74549FB :
05AE2E87C9 :
02D783360C :
0158798F4D :
044078380EB :
0F406C8384 :
0051336624 :
0888813D10 :
00C7B795A1 :
042E8281E9 :
07248764CA :
0484760DEF :
02B1C7D9EB :
0603F8760B :
03550395FB :
```

*445_final - Notepad

```
(.cra)
add:0000 ; fs=48000Hz
7FE900 ; 000 00 TlvlH Limiter_01
214F69 ; 001 01 TlvlL
7D6300 ; 002 02 TatkH
21F58D ; 003 03 TatkL
7FB800 ; 004 04 TrelH
700884 ; 005 05 TrelL
F86E00 ; 006 06 (const)
FAC984 ; 007 07 thre -4.000000[dB] - form factor corr.(0.912097584[dB])
179200 ; 008 08
080000 ; 009 09
3FC374 ; 00A a2 HPF_01
B07918 ; 00B a1
C07BDF ; 00C b2
7F864E ; 00D b1
3FC374 ; 00E a0
3FDA22 ; 00F a2 GEQ_01 [0] single
80260B ; 010 a1
C025DE ; 011 b2
7FD9F5 ; 012 b1
400000 ; 013 a0
3FC374 ; 014 a2 GEQ_01 [1] single
803CFF ; 015 a1
C03CBC ; 016 b2
7FC301 ; 017 b1
400000 ; 018 a0
3FA0BD ; 019 a2 GEQ_01 [2] single
80605F ; 01A a1
C05F43 ; 01B b2
7F9F41 ; 01C b1
400000 ; 01D a0
000000 ; 01E padding - FIR_01
000000 ; 01F A0
000000 ; 020 A1
000000 ; 021 A2
000000 ; 022 A3
000000 ; 023 A4
000000 ; 024 A5
000000 ; 025 A6
000000 ; 026 A7
7FE900 ; 027 00 TlvlH Limiter_02
214F69 ; 028 01 TlvlL
7D6300 ; 029 02 TatkH
21F58D ; 02A 03 TatkL
```

CRAM File

```
#ifndef _AK77XX_DSP_CRAM_INCLUDED
#define _AK77XX_DSP_CRAM_INCLUDED

#define _AK77XX_DSP_CRAM_WRITE_SIZE     258
static unsigned char ak77dspCRAM[] = {
    0xB4, 0x00, 0x00, // command code and address
    0x7D, 0x63, 0x00,
    0x21, 0xF5, 0x8D,
    0x7D, 0x63, 0x00,
    0x21, 0xF5, 0x8D,
    0x7F, 0xBB, 0x00,
    0x70, 0x08, 0x84,
    0xF8, 0x6E, 0x00,
    0x00, 0x00, 0x00,
    0x00, 0x00, 0x00,
    0xFC, 0x00, 0x00,
    0x00, 0x00, 0x00,
    0x00, 0x00, 0x00,
    0xFC, 0x00, 0x00,
    0x00, 0x00, 0x00,
    0x00, 0x00, 0x00,
    0xFC, 0x00, 0x00,
    0x00, 0x00, 0x00,
    0x00, 0x00, 0x00,
    0xFC, 0x00, 0x00,
    0x00, 0x00, 0x00,
    0x00, 0x00, 0x00,
    0xFC, 0x00, 0x00,
    0x00, 0x00, 0x00,
    0x00, 0x00, 0x00,
    0x7F, 0xFF, 0xF0,
    0x17, 0x92, 0x00,
    0x08, 0x00, 0x00,
    0x10, 0x00, 0x00,
    0x3F, 0xDA, 0x22,
```

OFREG File

```
#ifndef _AK77XX_DSP_OFREG_INCLUDED
#define _AK77XX_DSP_OFREG_INCLUDED

#define _AK77XX_DSP_OFREG_WRITE_SIZE    0
static unsigned char ak77dspOFREG[] = {
    0xB2, 0x00, 0x00, // command code and address
};

#endif    // end of _AK77XX_DSP_OFREG_INCLUDED
```

PRAM File

```
#ifndef _AK77XX_DSP_PRAM_INCLUDED
#define _AK77XX_DSP_PRAM_INCLUDED

#define _AK77XX_DSP_PRAM_WRITE_SIZE     1070
static unsigned char ak77dspPRAM[] = {
    [illegible]
```

Figure 18: Visual Representation of the Code Generation Process of AKx