Danalog: Digital Musical Synthesizer

Spring 2017

Evan Lew

Bryan Bellin

Jordan Wong

Vikrant Marathe

Table of Contents

List of Tables and Figures

Tables

Figures

Acknowledgements

Without the help of Cathy Wicks at TI who generously donated 4 ezDSP c5535 development boards to our team we would not have been able to make as much progress as we did.

Without the help of Dr. Wayne Pilkington, our advisor, our project would have never made it off the ground.

Acknowledgements

Without the help of Cathy Wicks at TI who generously donated 4 ezDSP c5535 development boards to our team we would not have been able to make as much progress as we did.

Without the help of Dr. Wayne Pilkington, our advisor, our project would have never made it off the ground.

Abstract

The Danalog is a 25 key portable digital music synthesizer that uses multiple synthesis methods and effects to generate sounds. Sound varieties included three synthesis methods including FM, subtractive, and sample-based, with up to eight adjustable parameters, at least four effects, including reverb, chorus, and flange, with five adjustable parameters, and at least two note polyphony, and a five band equalizer. The user would be able to adjust these effects using digital encoders and potentiometers and view the settings on two LCD screens.

The final project was unable to meet the original design requirements. The FM synthesis method was primarily working in the end product. The synthesizer was built to produce three note polyphony. The LCD screen displayed the information about the synthesis method as the user plays.

I. Introduction

The purpose of this project was to create a portable, inexpensive digital music synthesizer for amateur musicians. The intended customer base consists of young, amateur musicians who don’t have a big budget for a more expensive music synthesizer.

The market requirements for this product are as follows:

- The Danalog Synthesizer will be inexpensive at less than $200
- The design will be sleek and lightweight to promote portability
- There will be up to eight adjustable synthesis parameters
- There will be up to five adjustable effects parameters
- Five band equalizer

Our intended customer is an amateur musician seeking an inexpensive digital synthesizer to create a wide array of user-defined sounds.

Several other companies have their own digital synthesizers equipped with numerous features. The Danalog’s main competitors would be the Yamaha Reface, Korg Minilogue, Roland Boutique, and Arturia MicroBrute. The lowest price of these is the Arturia MicroBrute at $299 - which the Danalog has beat by $100. The Danalog digital synthesizer is also smaller than the other competitor’s options.

II. Product Design Engineering Requirements

Functional and Feature Requirements

- The Danalog Synthesizer will produce notes over a 2 octave range via Frequency Modulation Synthesis, with two note polyphony.
- The chassis will be made from lightweight plastic that is easy to carry and hold.
- All components, peripherals, and circuit boards are industry standard and well supported.
- The encoders, potentiometers, and switches will be strategically placed in a manner that follows the logical path of the signal from generation, to equalizing, to modulating.
- The processing will be split among two IC's: The ATmega2560 for peripheral information, and the TMS320C5535 for Digital Signal Processing.

Performance Specifications

- Low latency (<3ms delay) production of notes
- Instant visual feedback (<3ms) on pressed note
- Internal rechargeable battery of 5 hour life
- Able to run on 5V 500mA USB power
- Low noise audio outputs. 90dBc S/N with +4dBu max output

Level 0 Blackbox Diagram

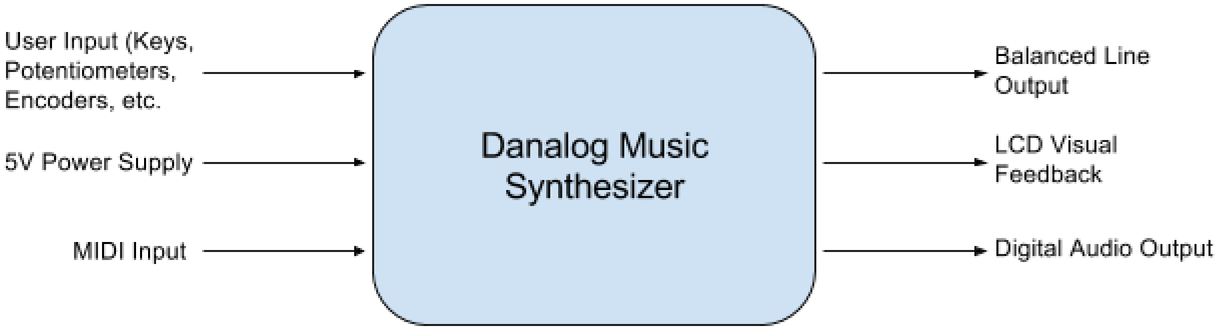


Figure 2.1: Level 0 Blackbox Diagram

User Input

All of the user inputs are translated into 8-bit signals, handled by the ATMega2560 microcontroller. This includes MIDI protocol, potentiometer positions, encoder rotation direction, switch and button positions (via mux), for a total of 30 bytes. All of them are traced to an Arduino Mega development board on a PCB where the ATMega chip resides.

5V Power Supply

The synthesizer is powered via 5V 500mA USB power or a 5 volt battery. There is a level shifter as well, because the Arduino Mega board runs on 5V while the TMS320C5535 runs on 3.3V.

MIDI Input

An optional external MIDI input is available too, which will override the bytes sent by the in-built keyboard.

User Interface

The user can control the type of synthesis (FM, Subtractive, Sample), and shift octaves on the keys using the rotary switches. With the encoders, the user can control the ADSR envelope of the audio wave, select the digital effects that will be utilized, and control their parameters. For example, for the reverb effect, the user may be able to control the delay time (10-200 samples) between each reverberation as well as the attenuation constant (0.1-0.99). The potentiometers are used to control the audio equalizer, by setting the gains (-12dB - +12dB) at specific frequencies to attenuate and boost certain frequency ranges. There are also potentiometers used to modulate a user-defined parameter, bend the master pitch, and control the master volume.

The two LCD screens provide visual feedback for the user. The left one lets the user know the type of synthesis and the ADSR envelope settings, and the gains of the equalizer. The right screen displays the type of audio effect in use along with its respective parameters and their settings.

IV. System Design - Functional Decomposition (Level 1)

The system can be broken down as shown in figure 4.1. The operation of the system can be generalized as:

1. The user interacts with the device
   a. Presses a key on keyboard
   b. Sends a MIDI event
   c. Changes a parameter on the front panel
2. The ATmega reads in information from the user
3. The C5535 requests an update on the status of the system
4. The ATmega responds with the latest information on key presses, parameter changes and MIDI information
5. The C5535 generates a waveform based on the state of the system
6. The sound is enjoyed by the listener

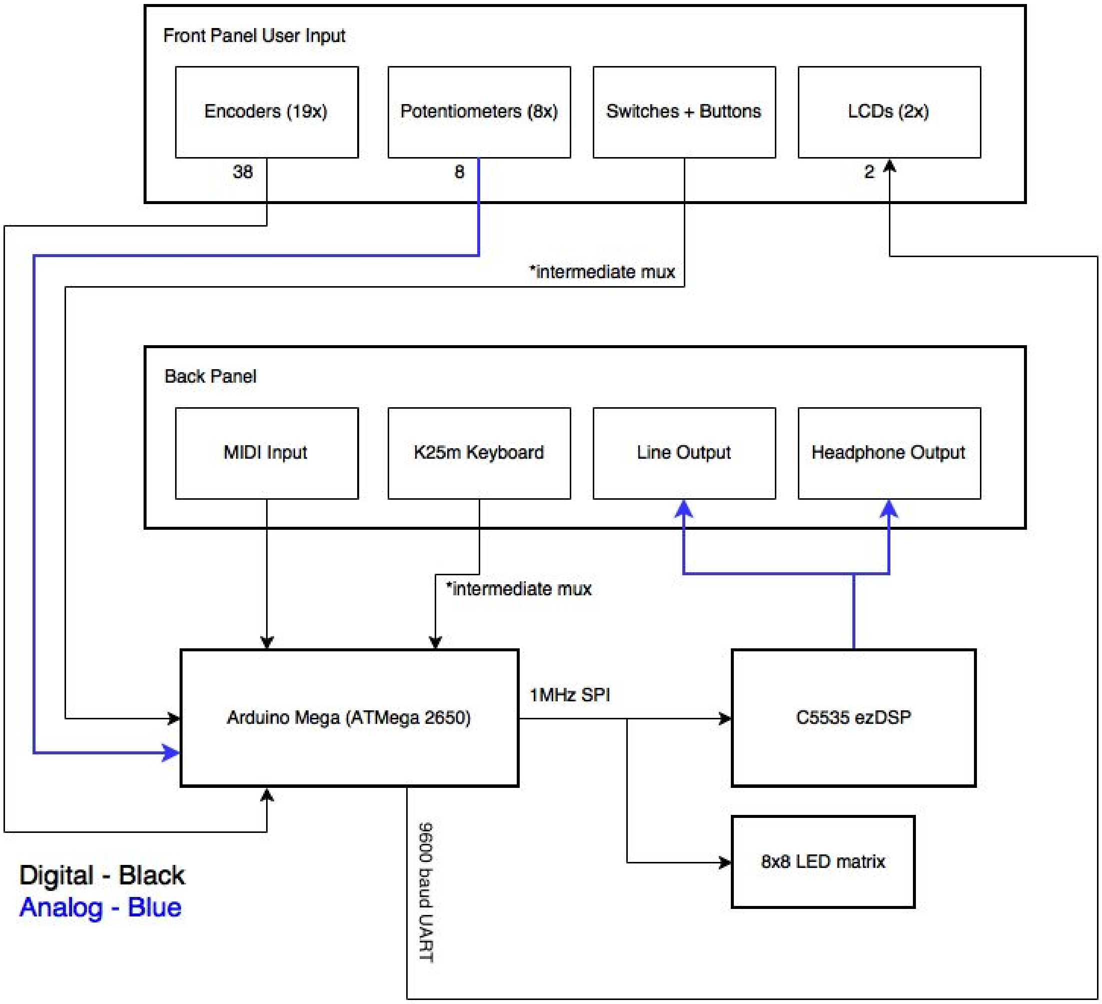


Figure 4.1: Level 1 block diagram of system

V. Technology Choices and Design Approach Alternatives Considered

DSP Selection

To build our music synthesizer we needed to select a proper digital signal processor. We ruled out a standard microcontroller early on because of the intensive math we would need to be doing. To figure out the computational requirements of our synthesizer we modeled our code in MATLAB and counted the number to mathematical operations need to generate one sample of audio. We made a couple of assumptions to simplify our estimation:

1. The sampling rate of our system would be 48kHz
2. The DSP would be able to complete a mathematical operation in one cycle

Using the following rough estimation we could guess the amount of processing speed needed to generate sound:

$$ClockFrequency = F_s \times \frac{operations}{sample}$$

Using my FM synthesis algorithm as a base case we estimated we would need a min clock rate of 5MHz based on 48kHz sampling rate and 100 mathematical operations per sample. *In retrospect this was not a good assumption because the processor has to do other operations to manage system resources*.

Since Texas Instruments (TI) is the largest DSP vendor I browsed their selection to see if they had any devices for our application. My criteria were:

1. The device must be able to fit in the form factor of our chassis
2. The device should be part of a development kit, for easy programming and debugging
3. The device must fit into our budget
4. The device must satisfy our computational requirements

Luckily TI has a category of evaluation modules called “ezDSP” that satisfy most of our project requirements. We settled on the C5535 ezDSP because of it’s price to performance ratio.

Arduino Mega (ATMega 2560)

Since the DSP was going to be solely focused on generating audio, it was decided that a separate microcontroller be responsible for interfacing with the user. The jobs the microcontroller would be responsible for are:

1. Checking encoders to see if a tick has occurred
2. Checking the values of the potentiometers
3. Updating the 2 LCD displays to show the current state of the system to the user
4. Reading the K25m external keyboard
5. Reading MIDI
6. Sending the data to the DSP via SPI

Since Jordan and I had positive experiences with the AVR toolset and the ATMega chip series we decided that we would use an AVR to perform the aforementioned set of tasks. We had the option to design the AVR right into our PCB which would have given us a smaller footprint and access to all the pins on the device however we opted to use an Arduino Mega which has an ATMega2560 onboard. The advantage of using a development board over an in house system was that we wouldn’t have to design a power system, a programmer or a clock setup. Since no one on the team had much experience designing any of those system we opted for the ready-to-go package. Another added benefit was that if we had trouble writing our code in C we could always fall back on the safety net of the Arduino ecosystem which has tons of examples and a huge user ecosystem.

Circuit Integration

We had a couple of options for integrating all the circuit into a final product. We could have used a breadboard to connect all the components together however this method likely be very messy and hard to debug nor would it be space efficient nor would it be a realistic packaging method for an off the shelf product. Furthermore we would always run the risk of having something coming loose. The only advantage to using a breadboard would be last minute changes would be possible. Instead of using a breadboard we also considered using a proto-board which would be similar to the breadboard but less configurable but slightly more robust (physically and in terms of electrical signal performance). Using a breadboard would allow us to have a custom solution that would perform well once everything was connected. The disadvantage of using protoboard would be the final product would be messy. A printed circuit board (PCB) would be a clean solution however it would require detailed planning because manufacturing is typically done in bulk and is expensive. Once the board is fabricated rework is difficult so care must be taken to make sure the board works on delivery.

Displays

We decided that we wanted to provide visual feedback to the user about the status of the system using character displays. Most LCD character displays require a 8-bit or 4-bit parallel interface to communicate with the LCD. Since we anticipated being IO limited on the ATMega2560 we decided that it would be a worthwhile investment to use serial displays instead of parallel displays. Sparkfun (sparkfun.com) sells a serial display which is simply adds an intermediate processor to convert the incoming serial data to parallel data to send to the displays. An added benefit of using serial displays is we can use the onboard UART peripheral on the ATMega2560 to send the data to the display which frees up the processor to do other tasks.

User interface controls

When planning out the user interface of the synthesizer we decided that knobs were the best way for users to input commands and parameters into the device because most of the controls on a typical synthesizer are variable in nature and humans are accustomed to using knobs to adjust variable parameters. Some of the controls control discrete parameters such as synthesis type and octave shift so we decided to use rotary switches which give a noticeable click to signify a change in parameters. Others are more continuous in nature for example envelope attack, reverb decay, and modulation ratio. For those knobs we have the choice of using rotary encoders (which have a digital clicky feeling) and potentiometers (which have a smooth continuous feeling). Figure 5.1 shows how we partitioned the potentiometer controls and encoder controls.

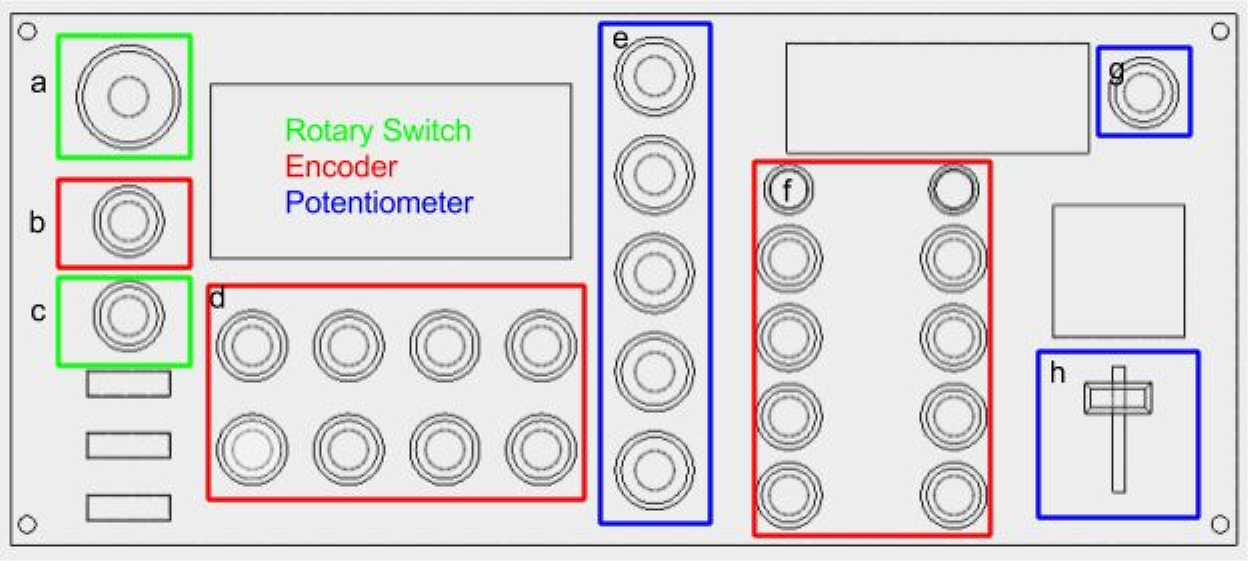


Figure 5.1: Allocation of rotary switches, encoders, potentiometers

Figure 5.1 shows how we allotted knob device type on the synthesizer front panel. Knobs a and c represent synthesis method and octave shift controls both of which are discrete in nature (only 3 synthesis methods proposed on the device so we elected to use rotary switches with predefined detents. Sections b and d are synthesis preset and synthesis parameter controls respectively. Synthesis preset is a discrete control in that only one preset can be selected at a time, which lends itself well to encoders. Synthesis parameters could be controlled by a potentiometer or encoder but since we were limited in analog channels we opts to use encoders. Same logic applies to the effects controls (section f).

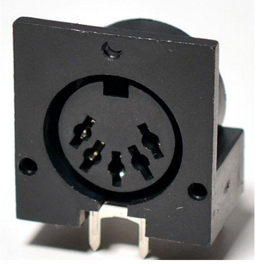
Keyboard

We wanted the synth to be as interactive as possible. Some hardware synthesizer are only able to receive MIDI information to create sounds and while this was an option for our synthesizer we wanted the user to have a keyboard at the ready to make the experience as intimate as possible. Our team didn’t have the design or manufacturing resources to build our own keyboard so we decided to ride on the coattails of others and use a Roland k25m keyboard that was designed to interface with Roland’s boutique line of hardware synthesizers.

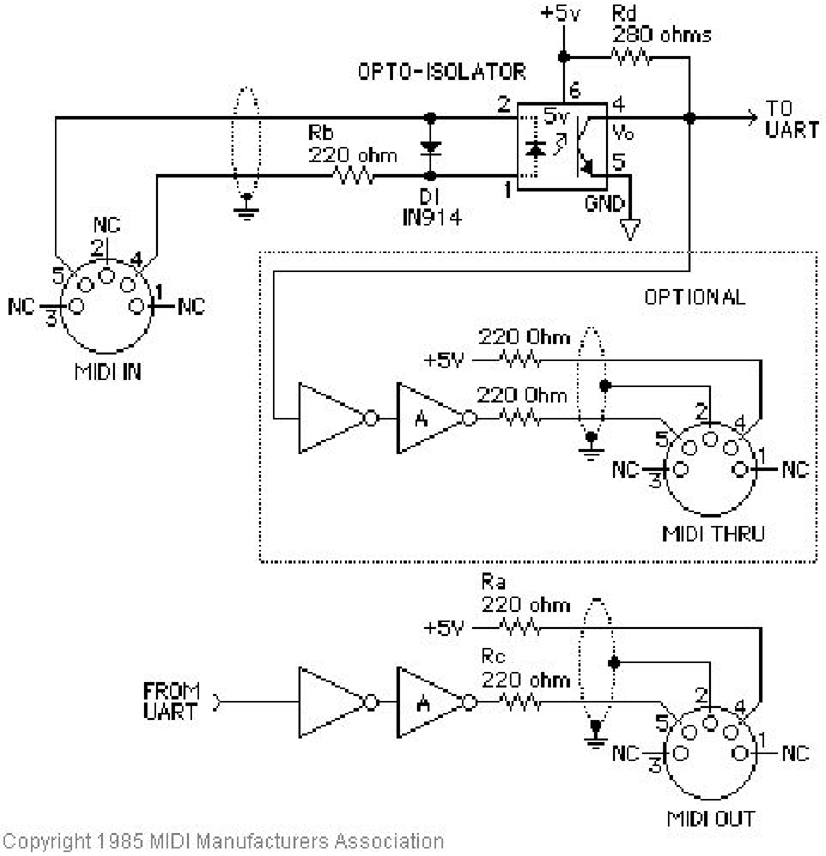
VI. Project Design Description

Hardware

MIDI



We wanted our synthesizer to be a flexible hardware synth which meant not only being able to receive input from an onboard keyboard but also from a MIDI source. The first step to being able to process MIDI input is to have a MIDI port on the device. This port is a 5 pin DIN connector. MIDI is a serial protocol that operates with 5V signaling. Figure 6.1 shows the standard MIDI reference design.



Copyright 1985 MIDI Manufacturers Association

Figure 6.1: MIDI specification circuit reference

For simplicities sake we elected to only have a MIDI input. Using the reference design as a guide we designed the circuit and layout shown figure 6.2

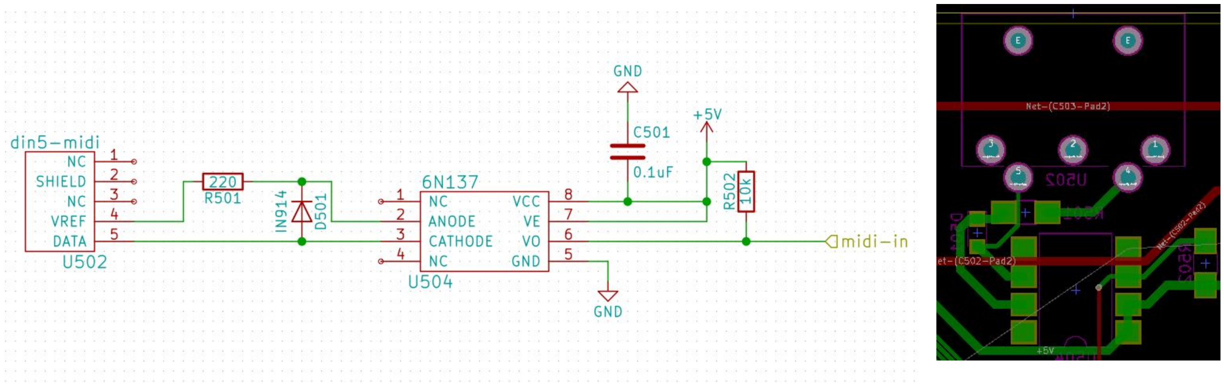


Figure 6.2: MIDI circuit schematic (left) and layout (right)

The circuit shown in figure 6.2 connects to the UART peripheral on the ATMega2560.

Diode connected keyboard

The Roland k25m has a 16 pin connection. When we first received the keyboard there was no documentation about the communication protocol. Based on our knowledge of how keyboards are traditionally connected we assumed that the k25m was connected in the typical diode connected fashion. Figure 6.3 shows how the topology of a standard diode connected keyboard matrix.

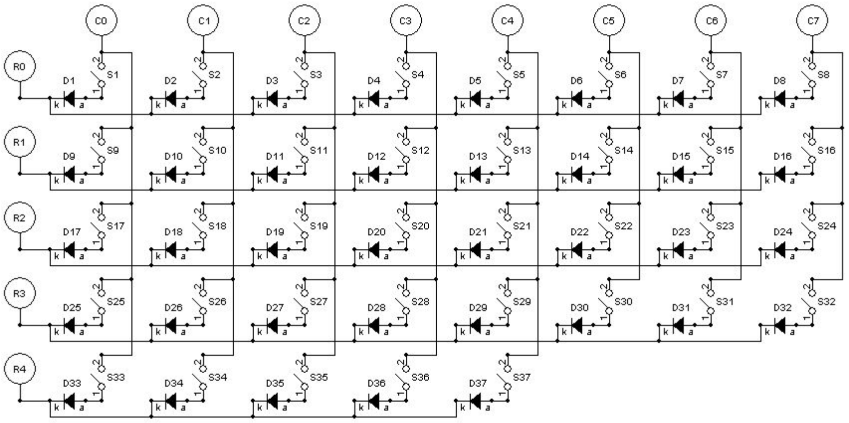
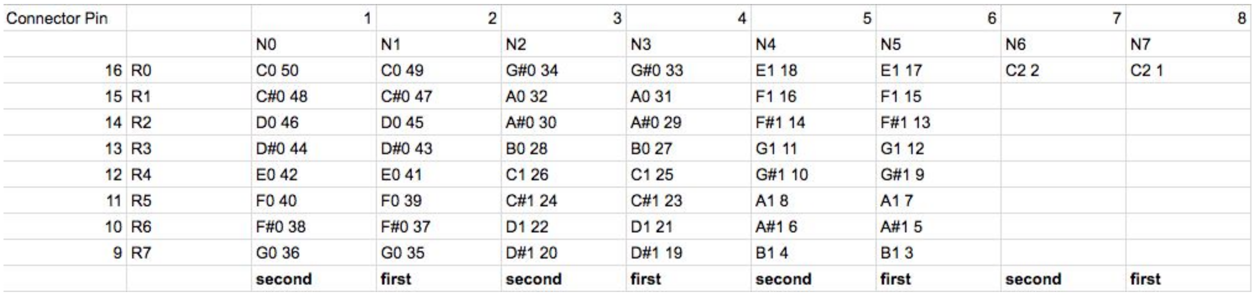


Figure 6.3: Typical diode connected keyboard matrix circuit topology

Knowing that we had the 16 pin connector most likely contained the pinout for the the rows and the columns we began to test which key corresponded to what row-column intersection. We created the spreadsheet in figure 6.4 to track all the combinations.



Connector Pin		1	2	3	4	5	6	7	8
		N0	N1	N2	N3	N4	N5	N6	N7
16	R0	C0 50	C0 49	G#0 34	G#0 33	E1 18	E1 17	C2 2	C2 1
15	R1	C#0 48	C#0 47	A0 32	A0 31	F1 16	F1 15		
14	R2	D0 46	D0 45	A#0 30	A#0 29	F#1 14	F#1 13		
13	R3	D#0 44	D#0 43	B0 28	B0 27	G1 11	G1 12		
12	R4	E0 42	E0 41	C1 26	C1 25	G#1 10	G#1 9		
11	R5	F0 40	F0 39	C#1 24	C#1 23	A1 8	A1 7		
10	R6	F#0 38	F#0 37	D1 22	D1 21	A#1 6	A#1 5		
9	R7	G0 36	G0 35	D#1 20	D#1 19	B1 4	B1 3		
		second	**first**	**second**	**first**	**second**	**first**	**second**	**first**

Figure 6.4: Spreadsheet of k25m pinout

Since the ATMega has limited IO we decided to use multiplexers to driver the rows and columns of the keyboard matrix. Figure 6.5 is the circuit I designed to drive and read the keyboard matrix.

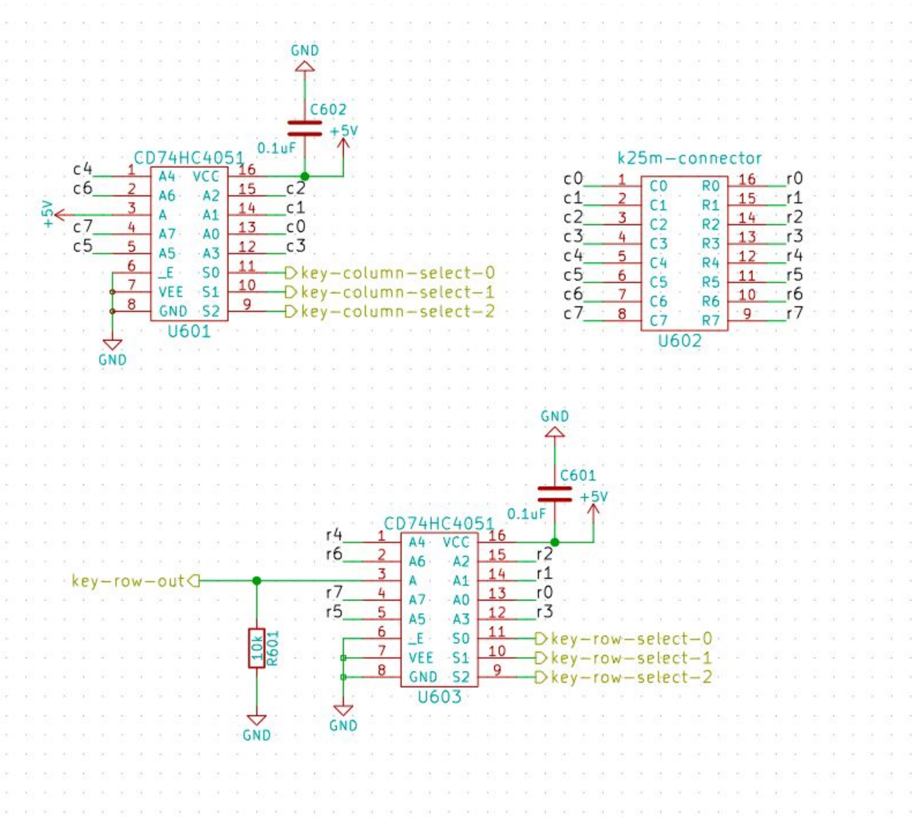


Figure 6.5a: Schematic of keyboard multiplexer circuit

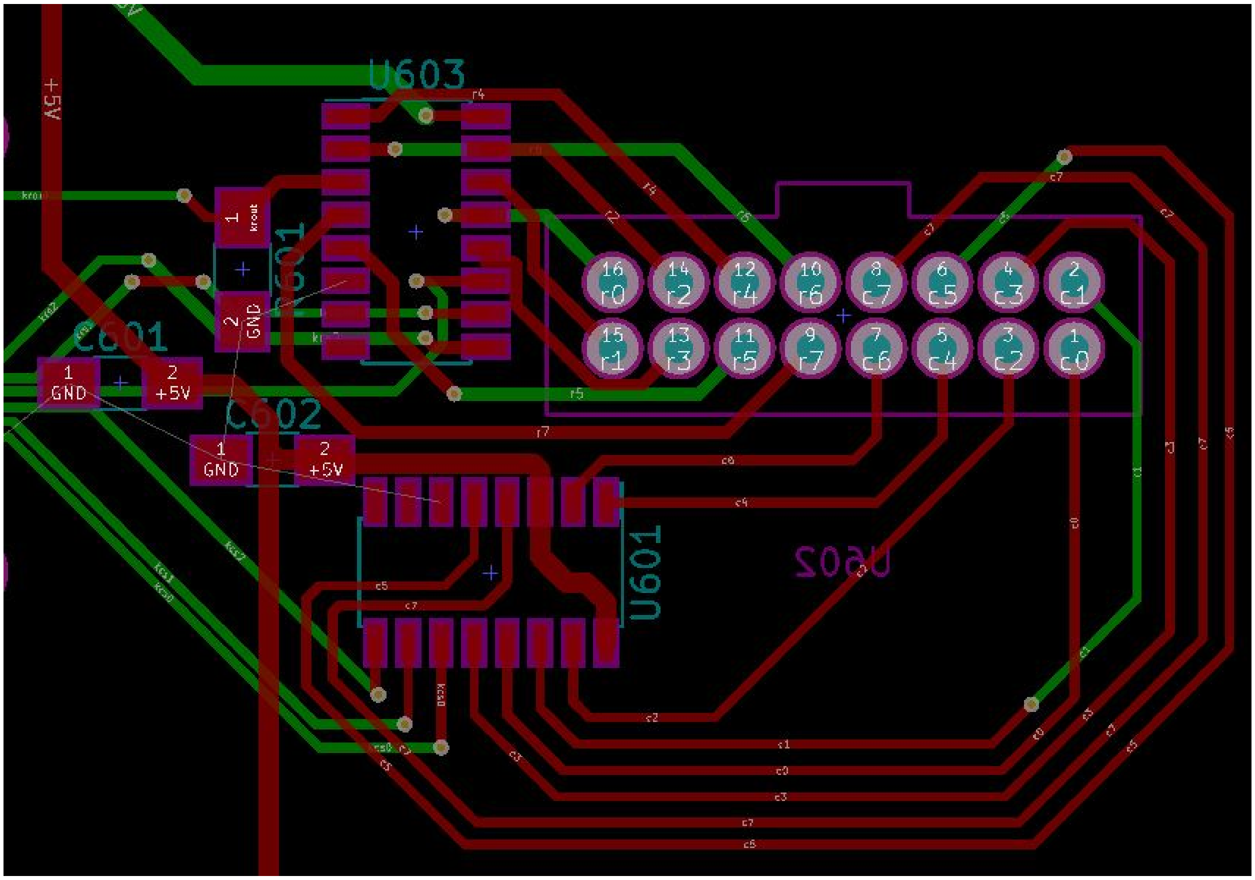


Figure 6.5b: Layout of keyboard multiplexer circuit

Encoders

Quadrature encoders operator on the principle of leading and lagging pulse phase. Each quadrature encoder has to output signal lines. In our circuit topology each line is pulled high by the internal pullup on the ATMega. When the encoder is turned it momentarily pulls the line low, this corresponds to on “tick” or detent in the encoder. To determine the direction of rotation of the encoder the microprocessor has to determine which pulse arrived first. Knowing the order in which the pulses occurred corresponds to the direction of rotation. Figure 6.6 illustrates how encoder pulse order relates to the direction of encoder motion.

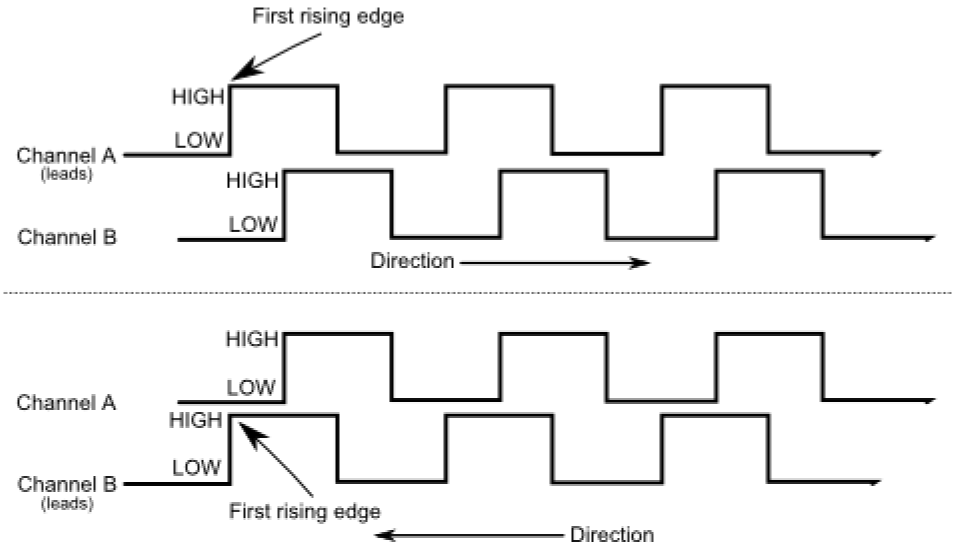


Figure 6.6: Encoder pulse relationships

The circuit I designed to capture the encoder pulse is shown in figure 6.7. The circuit was designed to be as simple as possible to minimize the amount of miniature components for hand soldering. The capacitor size was selected to minimize the contact bounce (when the mechanical components inside the encoder physically bounce creating multiple closely spaced phantom edges).

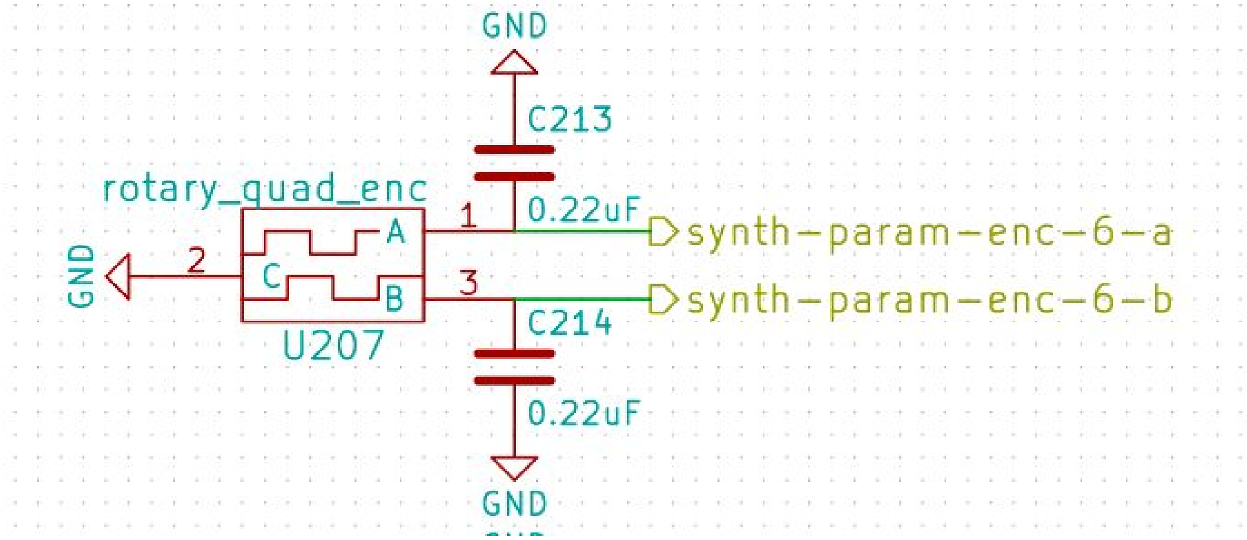


Figure 6.7: Encoder circuit

Since there are 19 encoders on the board, this subcircuit is repeated 19 times connected to different IO ports on the ATMega.

Potentiometer

The 5-band equalizer uses potentiometers to give the user a smooth continuous feeling as they turn the knob. The potentiometers also have a detent in the center to tactilely alert the user that they are at 0dB. The circuit for monitoring the position of the potentiometer (figure 6.8) is very simple.

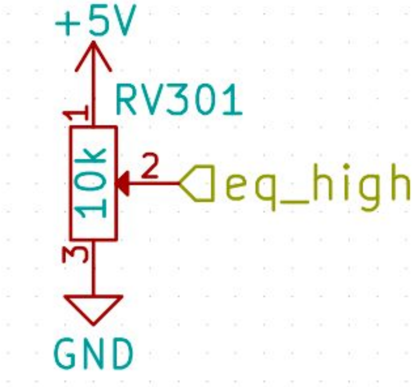


Figure 6.8: Potentiometer circuit

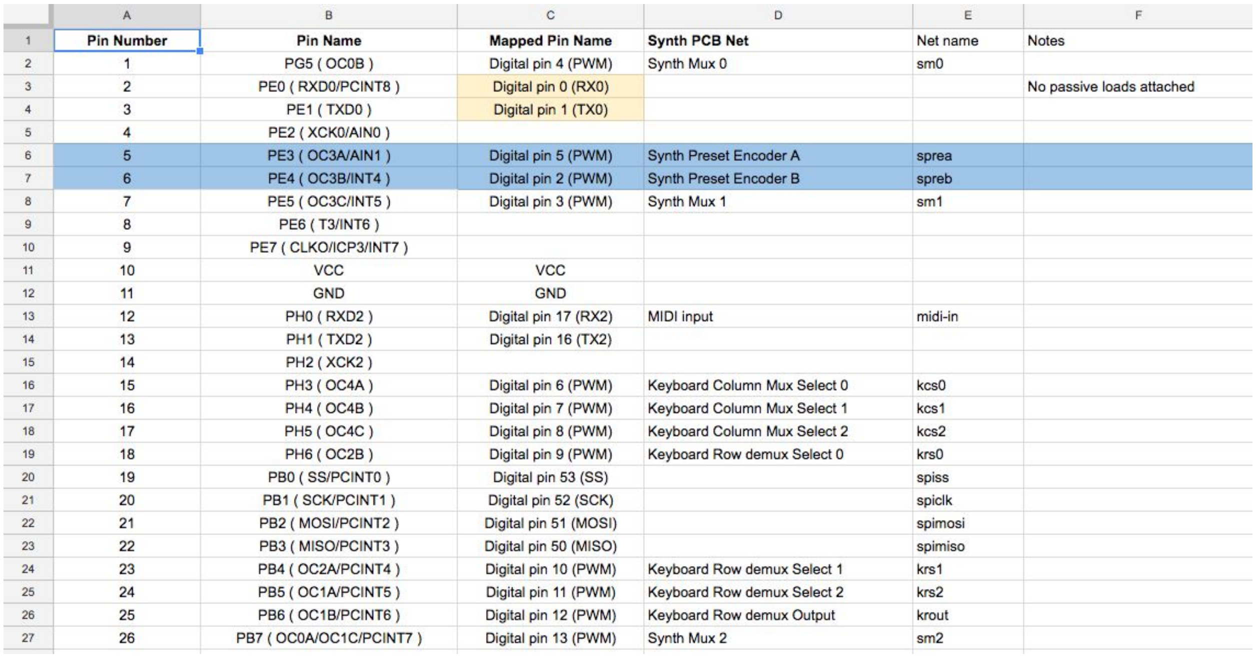
In figure 6.8 the signal labeled *eq_high* goes directly to the ADC peripheral on the ATMega.

Serial displays

The serial displays were simple to configure, just one signal going from the ATMega UART peripheral to the display via a cable connector.

Schematic Design

The first step to integrating all of the subcircuits into a single design was figuring out how all the pins would connect to ATMega, since the ATMega would be functioning as the central controller. To assign pins I created a spreadsheet that contained all the pins exposed by the Arduino Mega breakout board. I began to assign pins based on what devices we had decided we wanted on the device. Since I had a general idea what pin requirements those devices would need I was able to allocate pins for them. Using a spreadsheet to keep track of pin assignments allowed me to make sure that I was over assigning pins and to keep an eye out for how much free IO we would have. Figure 6.9 shows a section of the Arudino Mega pin assignment spreadsheet.



	A	B	C	D	E	F
1	**Pin Number**	**Pin Name**	**Mapped Pin Name**	**Synth PCB Net**	Net name	Notes
2	1	PG5 (OC0B)	Digital pin 4 (PWM)	Synth Mux 0	sm0	
3	2	PE0 (RXD0/PCINT8)	Digital pin 0 (RX0)			No passive loads attached
4	3	PE1 (TXD0)	Digital pin 1 (TX0)			
5	4	PE2 (XCK0/AIN0)				
6	5	PE3 (OC3A/AIN1)	Digital pin 5 (PWM)	Synth Preset Encoder A	sprea	
7	6	PE4 (OC3B/INT4)	Digital pin 2 (PWM)	Synth Preset Encoder B	spreb	
8	7	PE5 (OC3C/INT5)	Digital pin 3 (PWM)	Synth Mux 1	sm1	
9	8	PE6 (T3/INT6)				
10	9	PE7 (CLKO/ICP3/INT7)				
11	10	VCC	VCC			
12	11	GND	GND			
13	12	PH0 (RXD2)	Digital pin 17 (RX2)	MIDI input	midi-in	
14	13	PH1 (TXD2)	Digital pin 16 (TX2)			
15	14	PH2 (XCK2)				
16	15	PH3 (OC4A)	Digital pin 6 (PWM)	Keyboard Column Mux Select 0	kcs0	
17	16	PH4 (OC4B)	Digital pin 7 (PWM)	Keyboard Column Mux Select 1	kcs1	
18	17	PH5 (OC4C)	Digital pin 8 (PWM)	Keyboard Column Mux Select 2	kcs2	
19	18	PH6 (OC2B)	Digital pin 9 (PWM)	Keyboard Row demux Select 0	krs0	
20	19	PB0 (SS/PCINT0)	Digital pin 53 (SS)		spiss	
21	20	PB1 (SCK/PCINT1)	Digital pin 52 (SCK)		spiclk	
22	21	PB2 (MOSI/PCINT2)	Digital pin 51 (MOSI)		spimosi	
23	22	PB3 (MISO/PCINT3)	Digital pin 50 (MISO)		spimiso	
24	23	PB4 (OC2A/PCINT4)	Digital pin 10 (PWM)	Keyboard Row demux Select 1	krs1	
25	24	PB5 (OC1A/PCINT5)	Digital pin 11 (PWM)	Keyboard Row demux Select 2	krs2	
26	25	PB6 (OC1B/PCINT6)	Digital pin 12 (PWM)	Keyboard Row demux Output	krout	
27	26	PB7 (OC0A/OC1C/PCINT7)	Digital pin 13 (PWM)	Synth Mux 2	sm2	

Figure 6.9: Pin assignment spreadsheet for the Arduino mega

To integrate each of the subcircuits listed above into a single design I used a circuit capture and layout tool called KiCad. KiCad allows the circuit designer to build schematics using hierarchical blocks which allow for clean separation of subcircuits and block level elements. Figure 6.10 shows the top level schematic for the design. Note that there are no actual electrical components or packages exposed at this level of the schematic, these are all tucked away inside the sub-blocks.

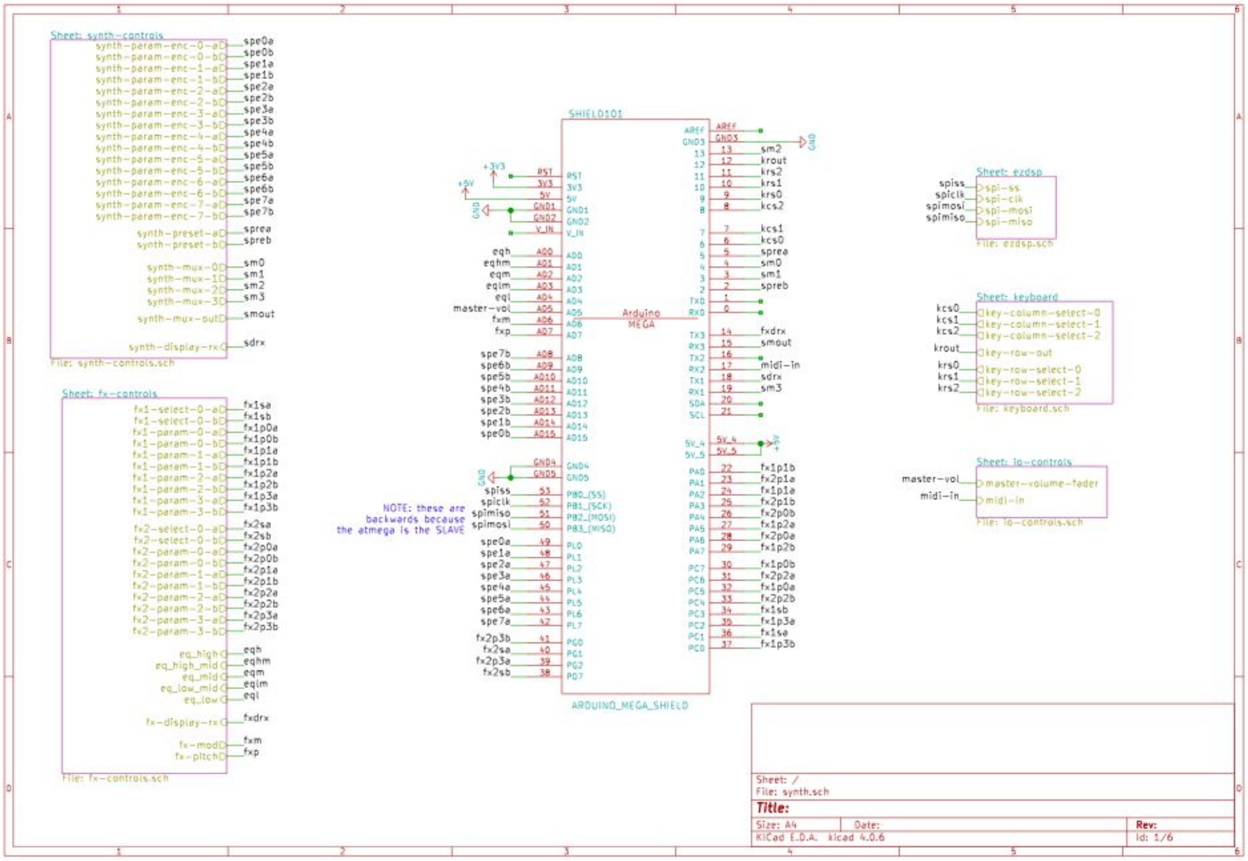


Figure 6.10: Top level schematic showing circuit sub-blocks

Filling out the sub-blocks of the schematics was as simple as connecting up the circuits shown in the subcircuits above, making sure that all the components corresponded to a front panel device as designed, and then final making sure all the nets were connected correctly. A large portion of the time I spent working on designing the schematic was dedicated to learning the KiCad EDA toolset, as I had never used any large schematic capture and board layout package before.

PCB Layout

Once the schematic was finalized (or at least mostly solidified). I began to layout the printed circuit board. The first thing to do was figure out what the board outline would be since we knew the form factor that chassis and board would have to fit inside of I created a rough outline of the size of the PCB. It was important to have a rough outline because it allowed me to begin laying out circuit components and get a price estimate for manufacturing a small batch of boards.

Since the board outline was not finalized I avoided placing components to close to the edge of the board.

The team had already designed how the front panel should look, so I created 2D design in Autodesk Fusion 360 (see *Mechanical Design* for more info) of the front panel. Autodesk Fusion 360 is able to export a DXF which conveniently KiCad can import into a PCB layout onto a user defined layer. Thus, I exported the 2D drawing of the front panel interface and imported it into KiCad so I could place the components in the correct spot. Figure 6.11 shows the board outline (yellow), the components and the DXF guide (white).

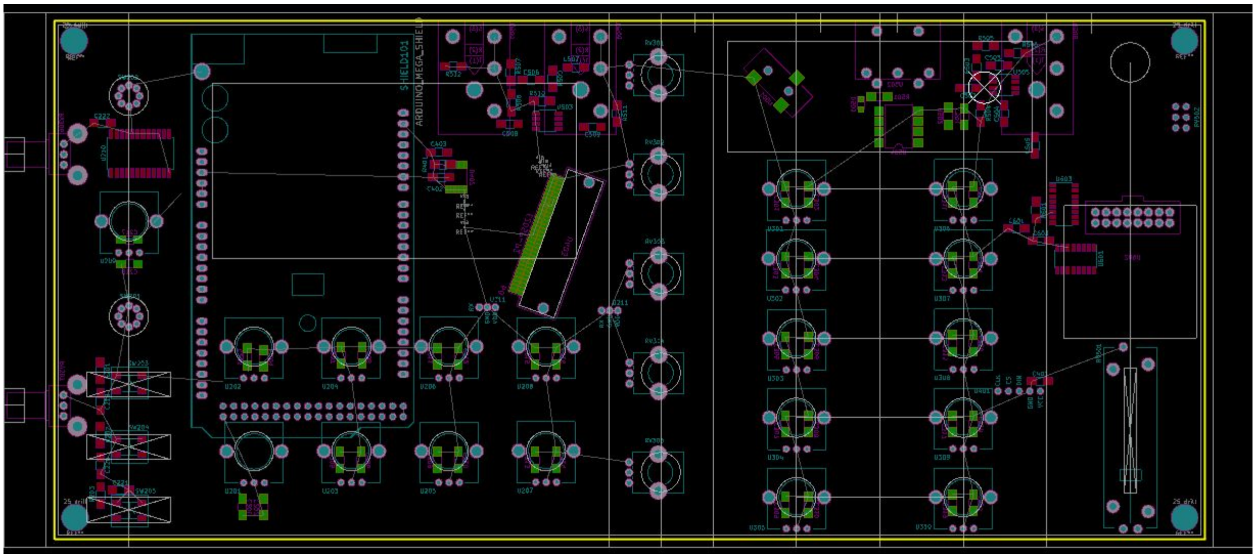
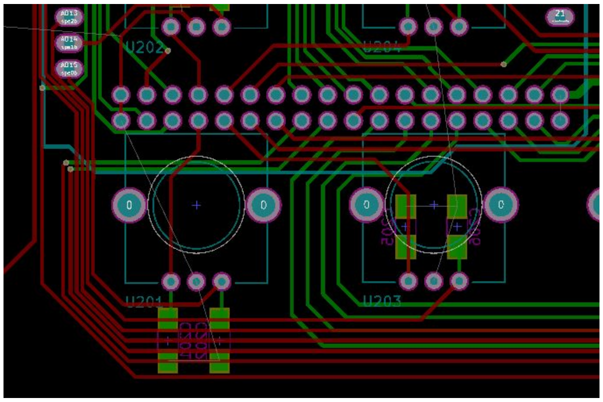
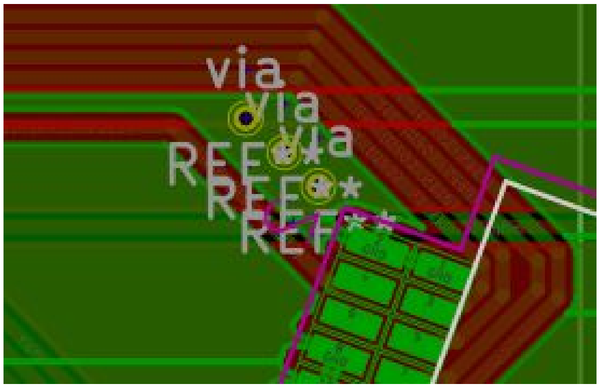
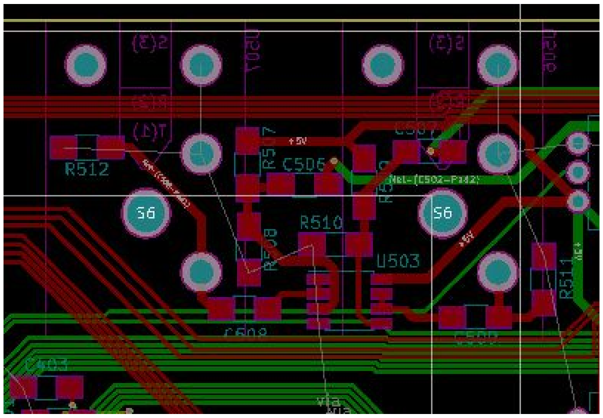
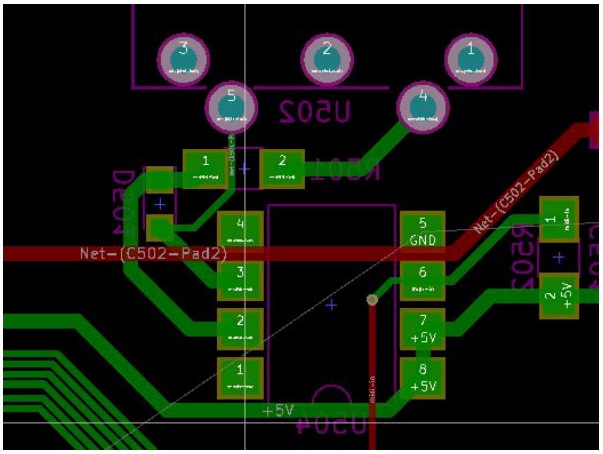
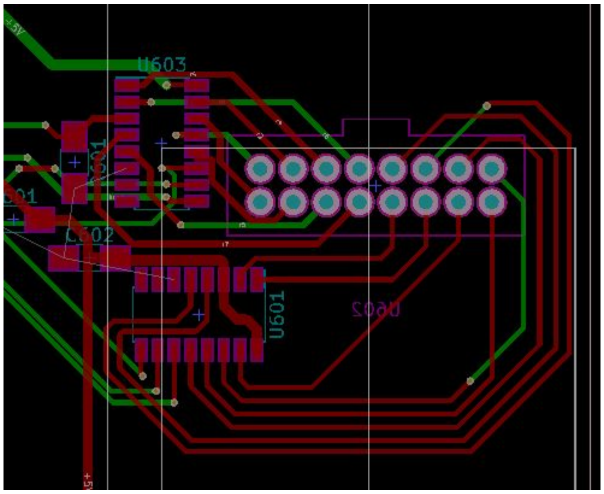


Figure 6.11: PCB layout without traces or copper fill

Next was to start adding traces to connect components (mainly back to the Arduino Mega). Since KiCad doesn't have a built-in auto router I had to manually route all the traces. I became stuck when trying to route all the signal lines for the 19 encoders, so unfortunately I had to go back to the pin assignment spreadsheet and rearrange some of the encoders to make them easier to route. This is generally not great practice to let layout constraints drive schematic design however in this case it was the easiest way for me to resolve this issue. I focused only on connected signal lines and then leaving grounding for a ground fill. Since this board is only 2 layers and I need both layer for routing I won't be able to take advantage of power ground planes so I'll be left with using a ground fill-in to connect nets to ground. Using a fill-in (or a ground plane) is advantageous for many reasons namely: it proves a low impedance path to ground, it provides some shielding and noise reduction and it saves the designer the time of have manually route each ground connection by hand. Table 6 shows close-ups of interesting sections of the final PCB layout. Note that some of the images in Table 6 are shown without the ground fill-in just for ease of viewing the traces

Table 6.1: PCB layout points of interest

	High density traces required me to the thin the trace size to 15 mil (0.015 inches). Since these traces aren't carrying power or high frequency signals this was ok.
	Even though the ground fill-in was able to connect most ground net on the board together, there was a couple of sections that became "islands" where they were completely walled off from the rest of the ground fill-in. The solution was to use "via stitching" to connect the island ground section to the ground fill-in on the layer opposite side.
	This is one of the two audio buffers designed by Bryan Bellin and laid out by me. Note the use of thick traces to have as clean signals as possible.

	MIDI optocoupler circuit layout
	K25m keyboard receiver circuit. 2 multiplexers were used to minimize the IO needed from the ATMega. Note that there was a layout mistake here that stemmed from a problem made in the schematic capture. Pins 6, 7, 8 should all be connected to ground however only pin 6 is connected (via the ratsnest line).

Chassis Design

The chassis was designed in Autodesk Fusion 360. The first step as noted earlier was to take the team designed front panel and turn it into a dimensioned sketch, figure 6.12.

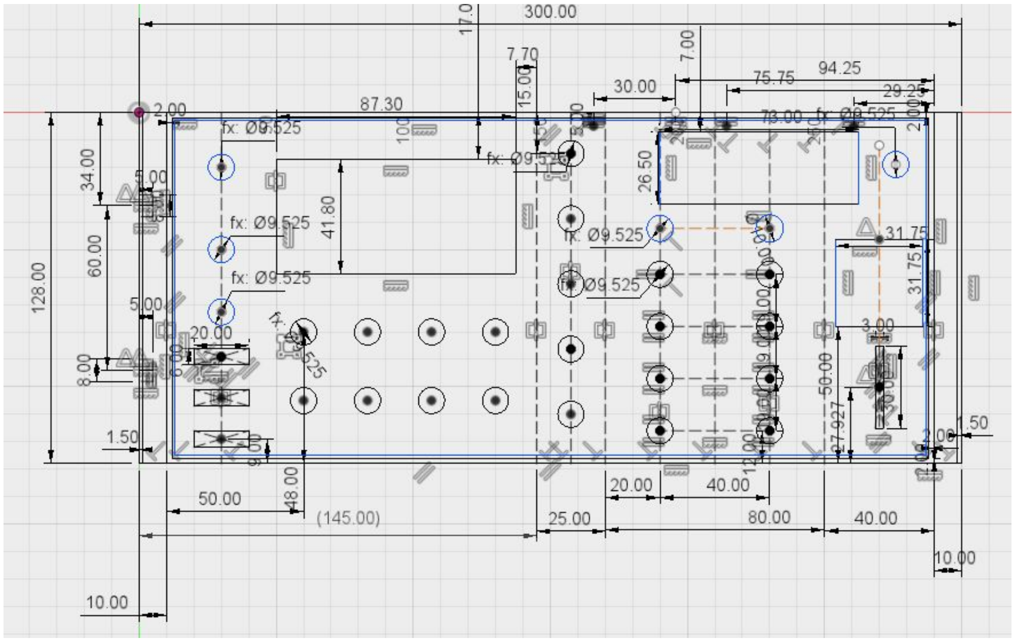


Figure 6.12: Front panel design

This front panel was the driving component for the rest of the design. Once the front panel was designed I was able to design the rest of the chassis around. The modules that are designed to find inside of the keyboard have the dimensions 308mm x 130mm x 51mm. Using those dimensions as a guideline I designed the bottom piece of the chassis, figure 6.13.

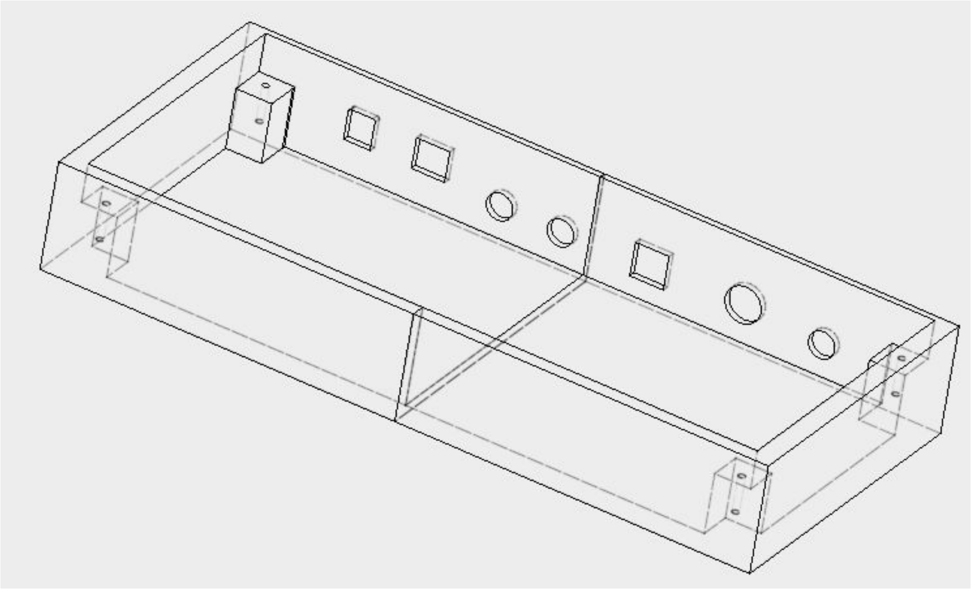


Figure 6.13: Chassis design

The 3D model generated by KiCad was imported into Autodesk Fusion 360 to check that the model's dimensions we correct most importantly to verify that clearances and cutouts were correctly spaced. The model with the 3D model of the PCB is shown in figure 6.14.

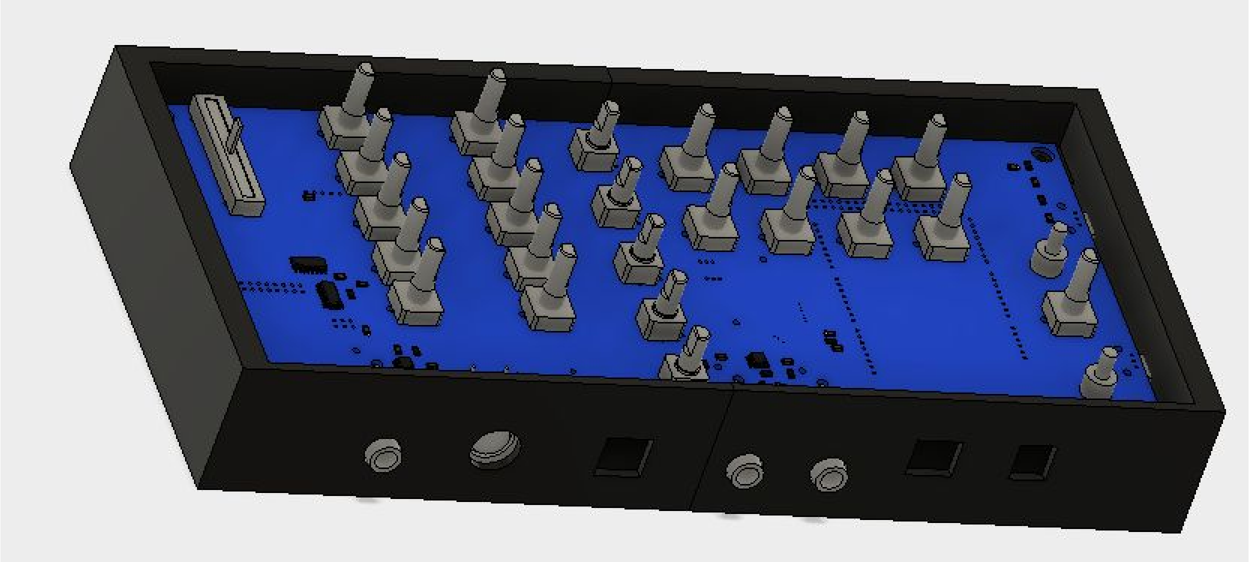


Figure 6.14: 3D model with KiCad PCB included for dimensions checking

Once the board and chassis were checked to make sure they fit together, the chassis was ready to print. 3D printing was done on my home printer, the MakerFarm 8" Prusa i3v, figure 6.15. Black PLA (Polylactic Acid) was used to print the chassis.

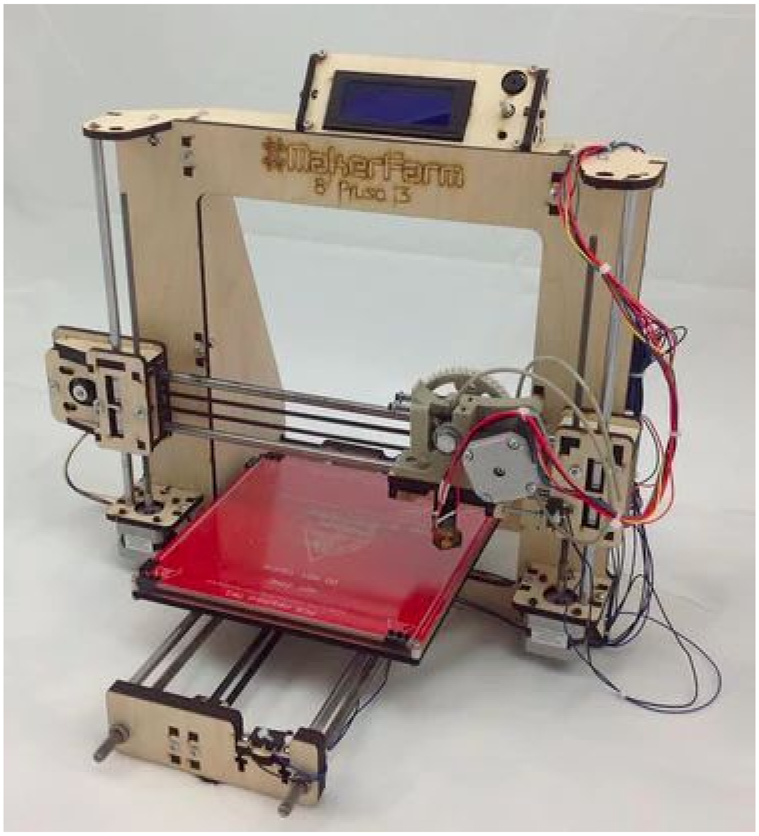


Figure 6.15: Prusa i3v 3D printer

Software

ATMega 2560

The ATMega is a low power RISC processor. It has limited hardware capabilities and therefore all the code written for it is essentially custom for that application since no Real Time Operating Systems (RTOS) exist for low performance chips.

To review, the task that the ATMega needs to perform are:

1. Checking encoders to see if a tick has occurred
2. Checking the values of the potentiometers
3. Updating the 2 LCD displays to show the current state of the system to the user
4. Reading the K25m external keyboard
5. Reading MIDI
6. Sending the data to the DSP via SPI

Timing for these tasks determines how often they need to be run. The time between encoder pulses is on the order 10ms, that means we need to check each encoder at least every 10ms or service the routine that does every 10ms. The potentiometers are absolute references that do not change very quickly therefore they can checked every 100ms. Updating the LCD's needs to reflect the state of the system without too much noticeable latency, conservatively this should be around 10ms. The keyboard and MIDI should be read as fast as possible to keep the audio latency as low as possible. Musicians are sensitive to latency upwards of 10ms. Everything in the chain between the key press and the output sound will add latency so it is important to keep everything as fast as possible. The data send over SPI contains all the information that the ATMega has gathered, including note information. This should be as fast as possible therefore it will be interrupt driven to reduce latency. All other operations (other than SPI) will run in the main loop.

C5535

The DSP will have two main jobs: most importantly generating audio and secondly getting the latest system information from the ATMega over SPI. Fortunately the C5535 comes with a chip support library (CSL) developed by Texas Instruments and a real time operating system (RTOS) called DSP/BIOS 5.42. The combination of the CSL and the RTOS allows us to write code at a higher level of abstraction without having to worry about extremely low level chip details. The high level software architecture is shown in figure 6.16.

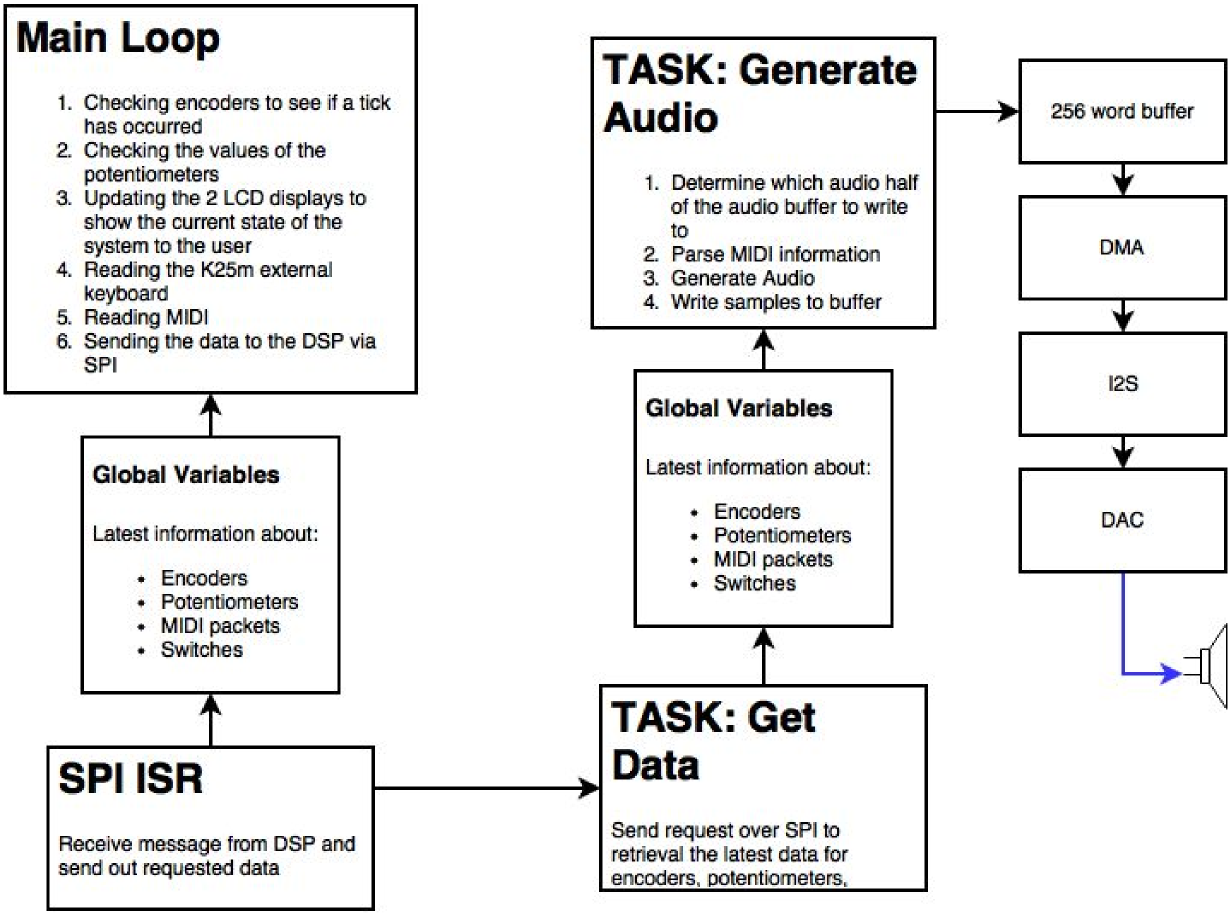

Main Loop
1. Checking encoders to see if a tick has occurred
2. Checking the values of the potentiometers
3. Updating the 2 LCD displays to show the current state of the system to the user
4. Reading the K25m external keyboard
5. Reading MIDI
6. Sending the data to the DSP via SPI
Global Variables
Latest information about:
• Encoders
• Potentiometers
• MIDI packets
• Switches
SPI ISR
Receive message from DSP and send out requested data
TASK: Generate Audio
1. Determine which audio half of the audio buffer to write to
2. Parse MIDI information
3. Generate Audio
4. Write samples to buffer
Global Variables
Latest information about:
• Encoders
• Potentiometers
• MIDI packets
• Switches
TASK: Get Data
Send request over SPI to retrieval the latest data for encoders, potentiometers,
256 word buffer
DMA
I2S
DAC

Figure 6.16: General software architecture

DSP/BIOS takes care of balancing the time spent on servicing the *Generate Audio* task versus servicing the *Get Data*. It is imperative that we do not drop samples so the *Generate Audio* task has a higher priority than the *Get Audio* task.

To make audio generation efficient many steps were taken to minimize the processing power need to generate a sound.

The example code included with the TI C5535 ezDSP starter kit used the simplest method to generate audio: a lookup table of sinusoid points and a main loop that polled the I2S peripheral to see if it was ready to accept a new sample. This method contains few lines of code but it doesn’t leave the CPU free to do anything else.

The logical fix to this issue is to use the built-in DMA peripheral. The DMA, short for direct memory access controller, is designed to solve problems like this. It works by offloading the task of transferring data from the CPU to the DMA controller. The DMA has access to the same addressing space as the CPU so in our case we can configure the DMA to look at a specific block of memory and transfer that block of memory to the I2S. The DMA is also configured to generate an interrupt once it is half way through the block of memory, this way the CPU can refill the other half of the block of memory with new samples so the DMA always has new data to send to the I2S. In this implementation we were generating audio inside the ISR.

Generating large amount of samples or doing any large amount of processing inside a hardware interrupt (HWI) is generally not a good idea because when the system is inside a HWI the scheduler cannot interrupt to service another task. The solution is semaphores. A semaphore allows different parts a system to communicate about the status of a resource by posting to and pending on a semaphore. This allows us to generate all the audio samples inside a task (rather than a HWI) which is nicer from a programming paradigm perspective. This also gave us a performance boost.

The compiler also has functionality to further optimize code by performing numerous optimization techniques that are beyond the scope of this project. However we can take advantage of them! Compiling with the *-03* flag allows the compiler to make performance optimizations at the expense of code size. Since our program wasn't pushing the memory footprint of the device this was a worthwhile tradeoff. Furthermore, the *TI C5000 DSP Programming Guide* give advice on how to further optimize C code for speed. For our application we can take advantage of a compiler directive called *MUST_ITERATE*

```
#pragma MUST_ITERATE(I2C_BUFFER_SIZE, I2C_BUFFER_SIZE)
```

This allows the compiler to perform more aggressive optimizations knowing that the loop will run exactly I2C_BUFFER_SIZE times. Under the hood the compiler maybe unwrapping the loop to get rid of unnecessary branch statements or using efficient hardware loop routines.

We could achieve another performance boost if we were able to use exclusively 16bit integer math, however due to overflow from arithmetic operations 32bit integer math is necessary for our algorithms.

Other simple things can make substantial performance improvements as well. Declaring variables outside the scope that they are going to be cuts of the compiler from performing some optimizations. Using arithmetic shifts can also be faster than regular division in some cases.

FM Synthesis

The sound generation technique that I implemented is known as frequency modulation synthesis. In its most basic form frequency modulation consists of two oscillators: a carrier and a modulator. The carrier frequency is modulated by the modulating wave, resulting in a waveform that changes frequency very rapidly. Depending the modulation ratio (the ratio of the modulator frequency to the carrier frequency) and the modulation depth (how much the modulator affects the carrier wave) the resulting waveform can contain many rich harmonics. Since we are using a fixed point processor with limited computational bandwidth we pregenerated all the sinusoid math so during runtime the sine calculation was reduced to a simple lookup.

To make the sounds generated by the FM synthesis engine sound dynamic and organic I added ADSR envelopes to both oscillators. An ADSR (Attack Decay Sustain Release) envelope gives the user the ability control to amplitude of the oscillator output with respect to time . This is especially powerful with FM synthesis because by changing the amplitude of the modulating waveform you can change the harmonic content of the final waveform with respect to time. Many cool effects can be created by experimenting the the envelopes.

VII. Physical Construction and Integration

Physically the Danalog synthesizer consists of

1. Main PCB: The PCB connects all the devices together and functions mechanically to hold all the components neatly in place inside the enclosure.
2. Arduino Mega: The Arudino Mega functions to interface with all the user input controls. It communicates all the fundamental information to the C5535 via a SPI communication bus
3. TI ezDSP C5535: This device is responsible for interpreting the information sent by the Arduino and generating sound.
4. 3D printed chassis: Encloses all components

The PCB functions as the harness for all the front panel interface controls, which consist of rotary quadrature encoders, rotary potentiometers, linear potentiometers, and rotary switches. The organization of the user interface was decided by the team during the initial planning phase. All interconnections on the PCB were made to accommodate the initial user interface design.

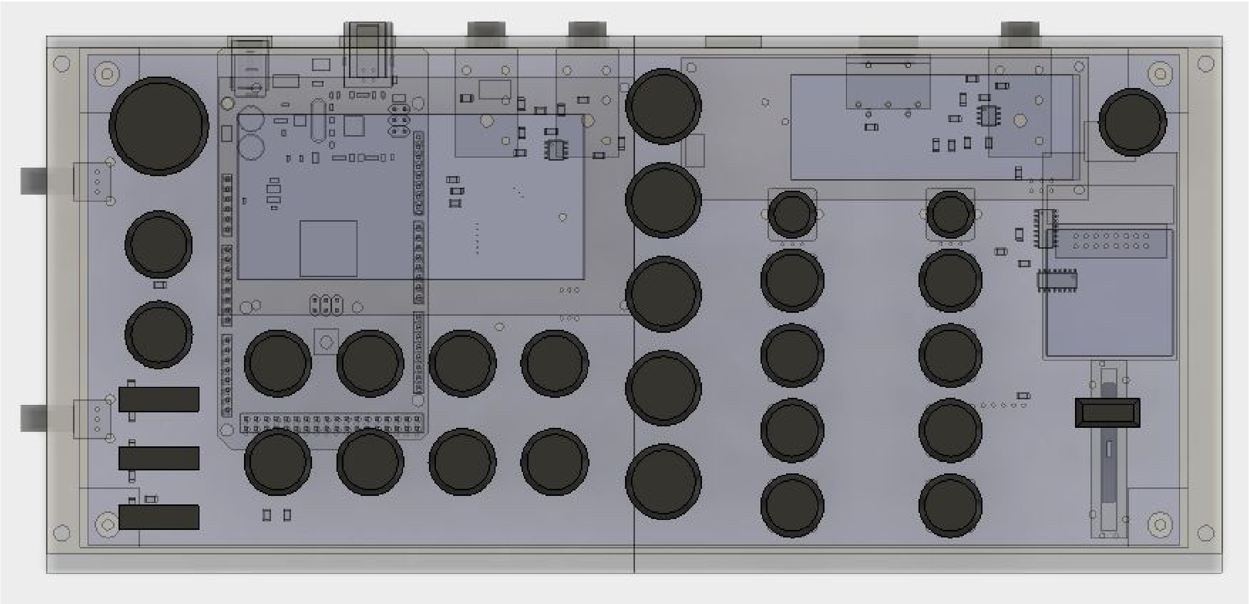


Figure 7.1: Overview of internal device construction and organization

Each device is routed to pins on the Arduino Mega board. Since the amount of IO needed was slightly more than the Arduino Mega provided we used multiplexers between the diode connected matrix keyboard (figure 7.2a) and between the rotary switches and front panel buttons (figure 7.2b)

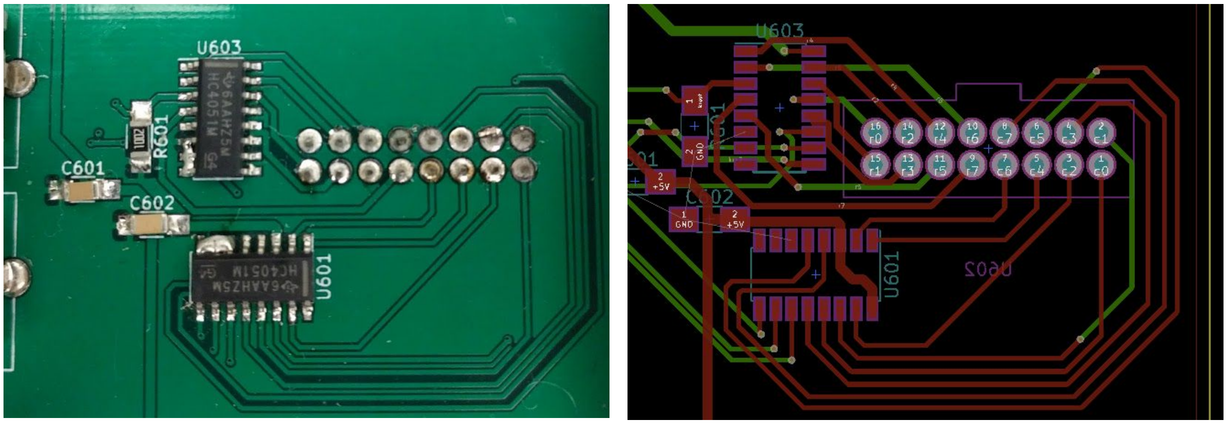


Figure 7.2a: Diode connected keyboard multiplexer layout

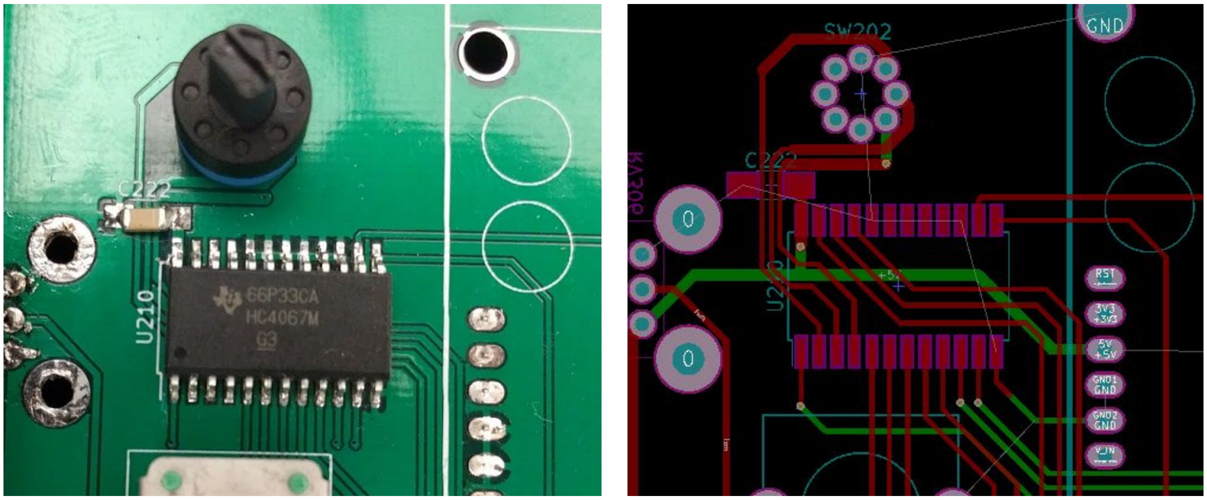


Figure 7.2b: Rotary switches and buttons multiplexer layout

The device also has two displays for outputting information about the state of the synthesis engine and the state of the effects processor. The displays were purchased from sparkfun as separate units not soldered to the the main circuit board. These displays were used because of their simple serial interface which allowed us to use a hardware UART to communicate with the display

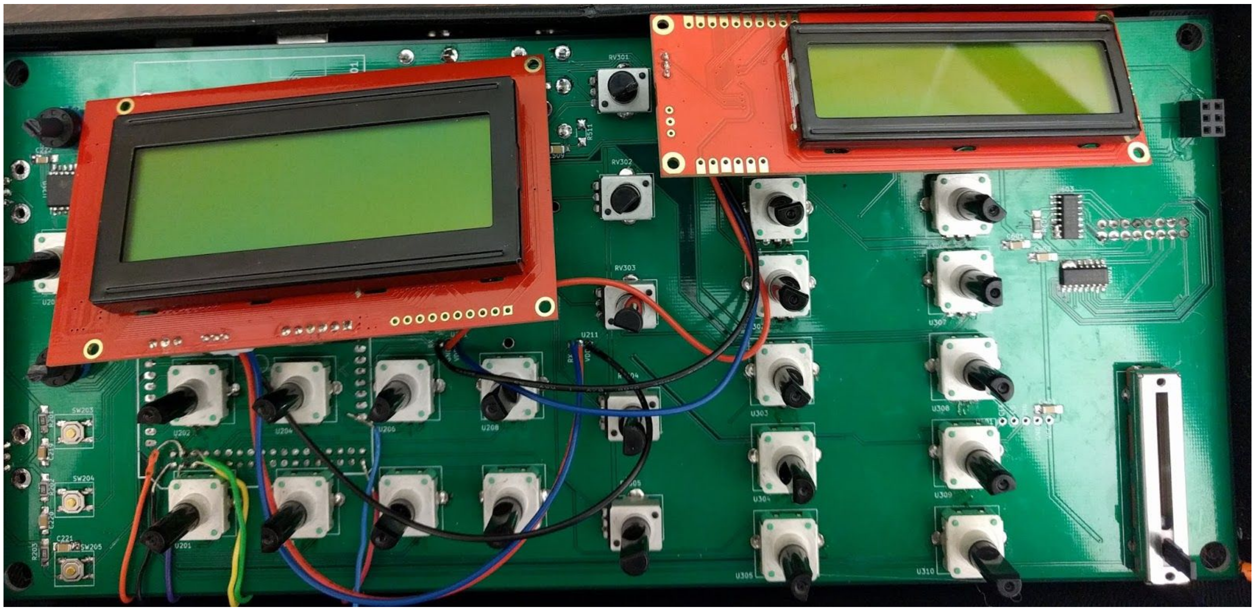


Figure 7.3: Both LCD displays connected to the main PCB via wires

Since the both the Arduino Mega and TI ezDSP both can be driven by 5 volts USB power there was no need to design any sort of power system. Additionally since the devices are low power, as USB devices usually are, there is no need for any form of heat sinking inside of the enclosure.

The chassis was 3D printed on Evan Lew's home 3D printer. Due to sizing constraints the chassis was printed in two halves and then glued together to form the final chassis. Figure 7.4 shows the 3D model of the chassis and figure 7.5 shows the real life chassis supported by the keyboard.

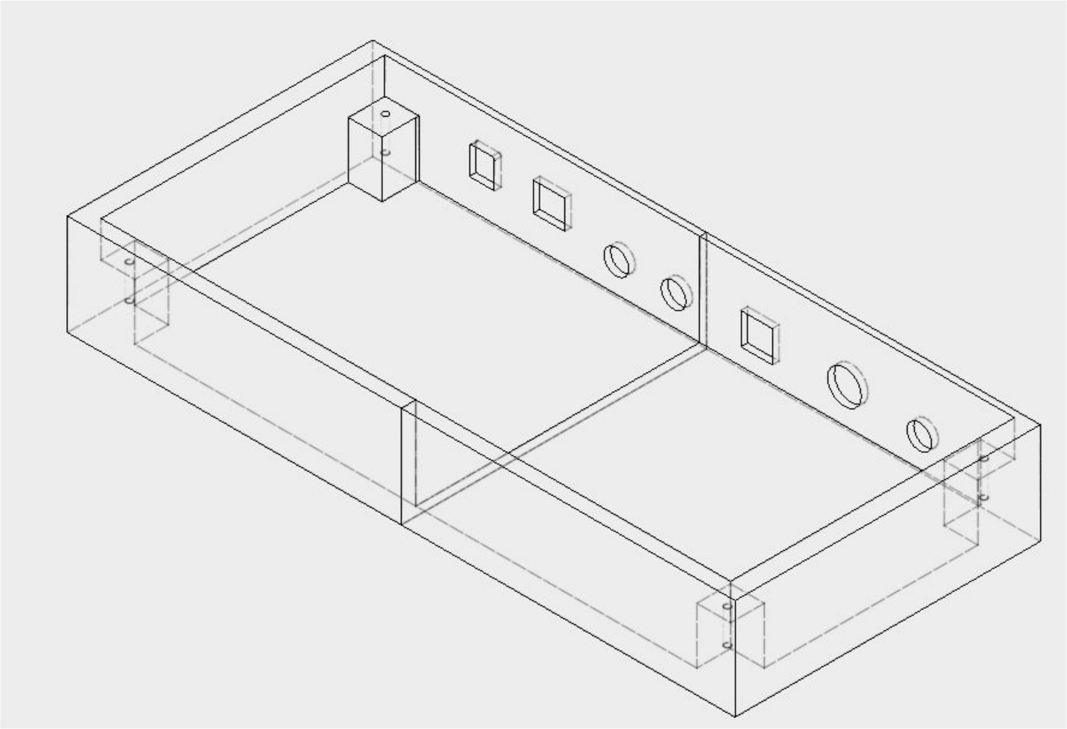


Figure 7.4: 3D model of the chassis

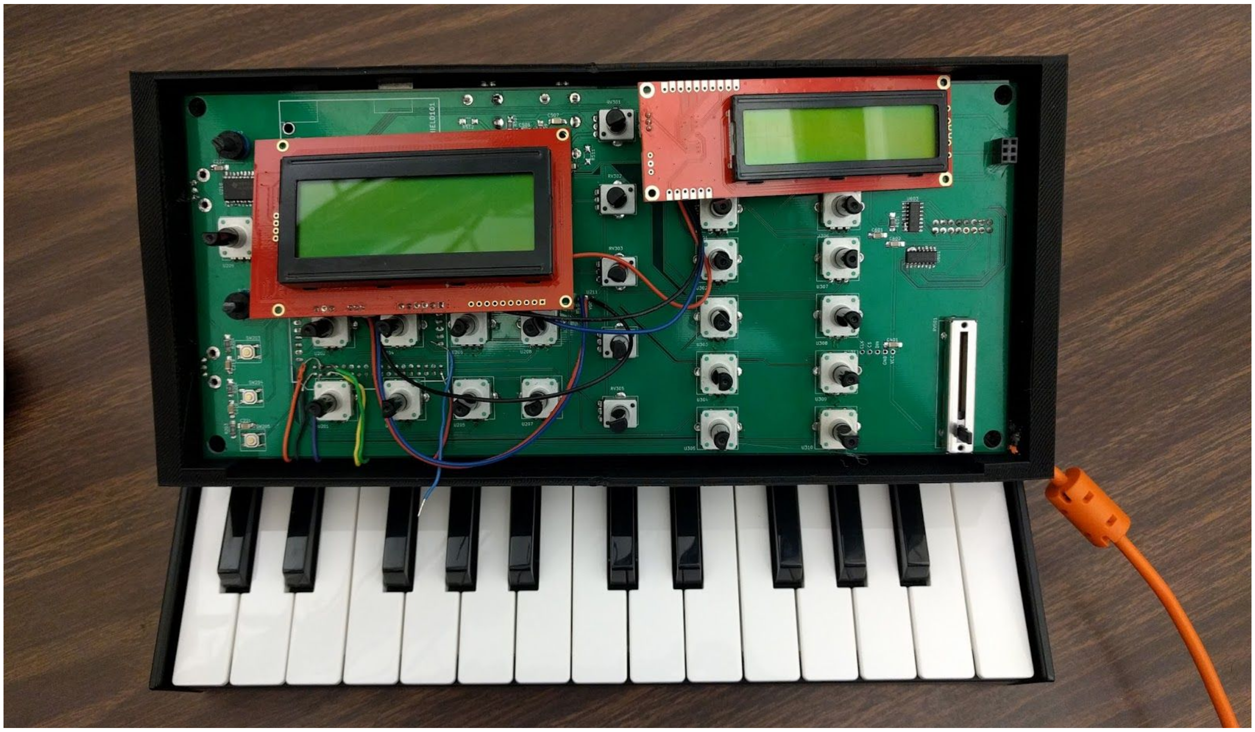


Figure 7.5: Chassis with internal hardware

VIIIa. Individual Subsystem Tests and Results

MIDI

This test was conducted to verify the operation of the MIDI receiving circuitry for the danalog music synthesizer. Figure 1.1 shows the circuit under test. Note that the external connection labeled “midi-in” is connected to UART2 (labeled pin 17) on the Arudino Mega development board. Goals for this test are as follows.

1. Verify the MIDI receiver circuit works as designed
2. Create proof of concept code to interface with the MIDI receiver
3. Obtain metrics for software timing constraints

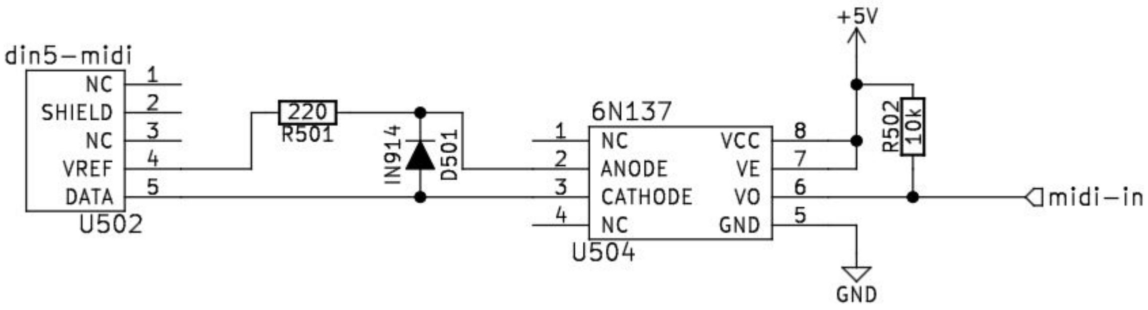


Figure 8.1: MIDI receiver circuit

Note that “din5-midi” is simply the MIDI connector which only uses two pins. MIDI is isolated so the information is transmitted through the optocoupler (6N137). The circuit was created in a breadboard shown in figure 1.2. D501 was not included as it is a protection diode that is not active during normal operation.

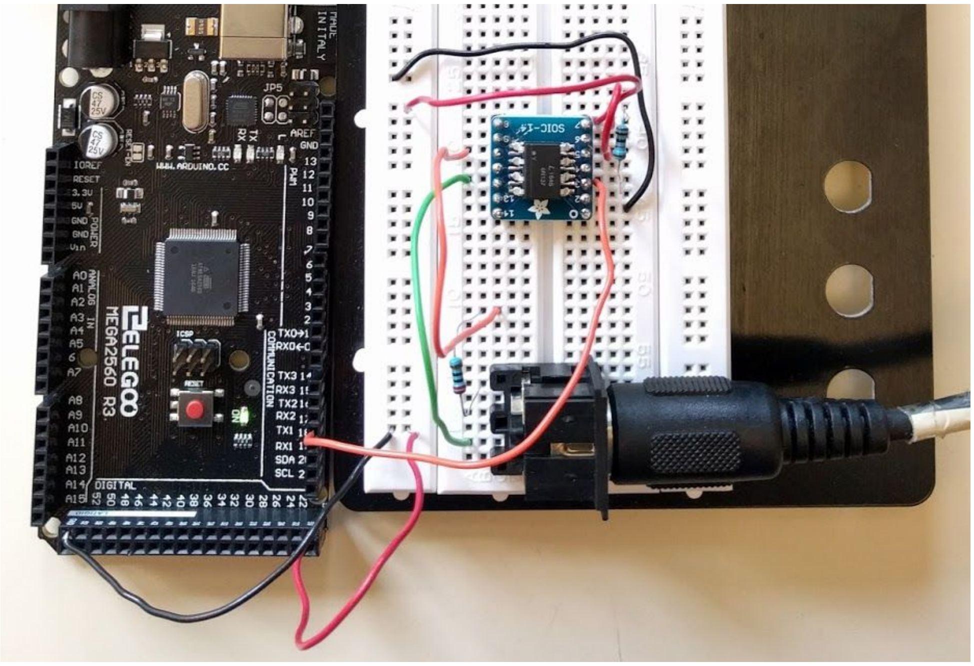


Figure 8.2: Breadboard realization of the figure 1.1

To simulate the circuit a midi interface was connected to a laptop with Logic Pro X. Logic was configured to send a repeating sequence of sequential notes as shown in figure 1.3. An M-AUDIO 1x1 midi interface was used to translate the data into midi format.

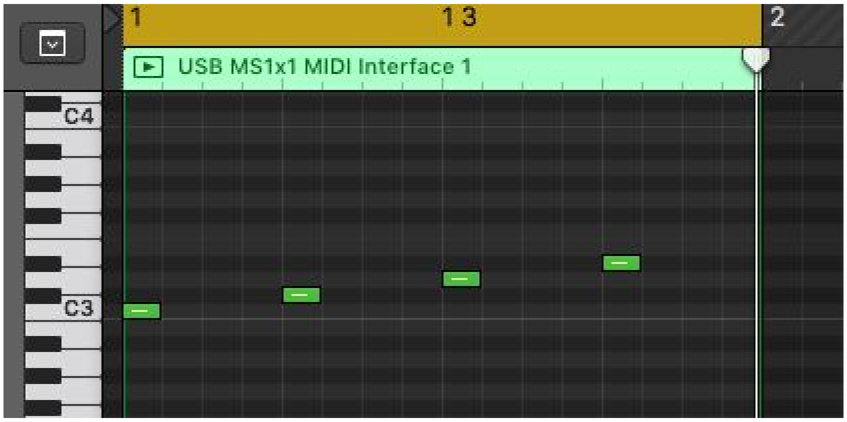


Figure 8.3: MIDI pattern inside of Logic Pro X

To verify that the midi sequence was being generated properly, a Saleae Logic 8 to view and interpret the midi signal on the microcontroller side (not the isolated side). Figure 2.4 shows the signal in the Saleae Logic software decoded. The signal was correct.

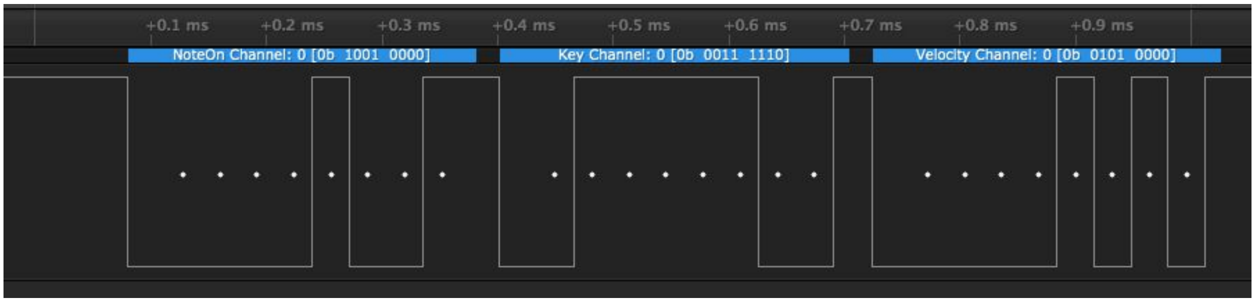

+0.1 ms
+0.2 ms
+0.3 ms
+0.4 ms
+0.5 ms
+0.6 ms
+0.7 ms
+0.8 ms
+0.9 ms
NoteOn Channel: 0 [0b 1001 0000]
Key Channel: 0 [0b 0011 1110]
Velocity Channel: 0 [0b 0101 0000]

Figure 8.4: MIDI signal interpreted by the Saleae Logic analysis software

To receive the midi messages on the Arudino Mega, UART2 was configured with the following parameters:

- 31250 baud
- No Parity
- 8 bit word
- 1 stop bit

Once the UART peripheral detects new data an interrupt service routine is called putting the new data into a circular buffer which is read by the main program loop. To view the midi data, the main loop formats and sends a string out of UART0 which is connected to the USB over serial chip so it can be view on a PC. Listed below is the main loop code and the midi reception code.

Roland k25m Keyboard

To test the Roland k25m keyboard the circuit shown in figure 8.5 was prototyped in a breadboard. Leads were connected from the k25m connector to the breadboard to simulate plugging the keyboard into a physical connector and the Arduino Mega was used to interface with the two multiplexers . Unfortunately no photos were taken of the test setup, but the big picture was very similar to the other test setups.

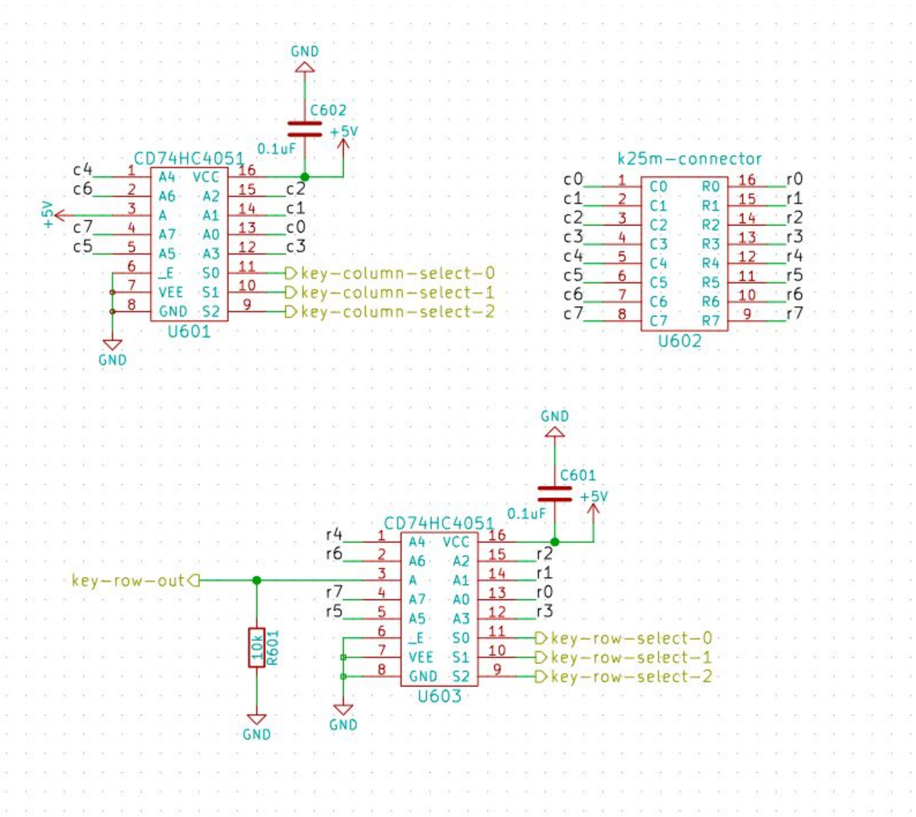


Figure 8.5: Keyboard multiplexer circuit

The Arudino Mega was programmed to drive the column multiplexer in a step sequence rotating through all possible output pins. For every column that was asserted high each row was checked to see it a voltage was present, if a voltage did in fact exist that would indicate that the key corresponding to that column-row location. To be able detect velocity (the force with which the key was pressed) two switches slightly offset in depth are assigned to each key. This way they get triggered at slightly different times. The less time between switch presses, the harder the key was pressed.

The code must keep track of what state the note is in: off, on or contact (when the first switch has been pressed but the second switch hasn't been pressed yet). A simple struct keeps track of what state the note is in, what the velocity of the note is (if it has been pressed). An 2-dimensional array holds all the structs for safe keeping, the array indices correspond to the column-row position.

During testing there was some trouble with false triggered notes however after some debugging I found that the power and ground connections on row multiplexer were not correctly attached.

Level Shifter

The level shifter is responsible for bridging the 5V main system with the 3.3V DSP domain. The only communication between the two domain is the SPI bus which should be able to run in the low MHz range. If the level shifter is capable of converting signals in the low MHz range than the systems should be able to reliably communicate. Figure 8.6 shows the breadboard setup with the TXB0106 6 channel bi-directional level shifter in a breakout board.

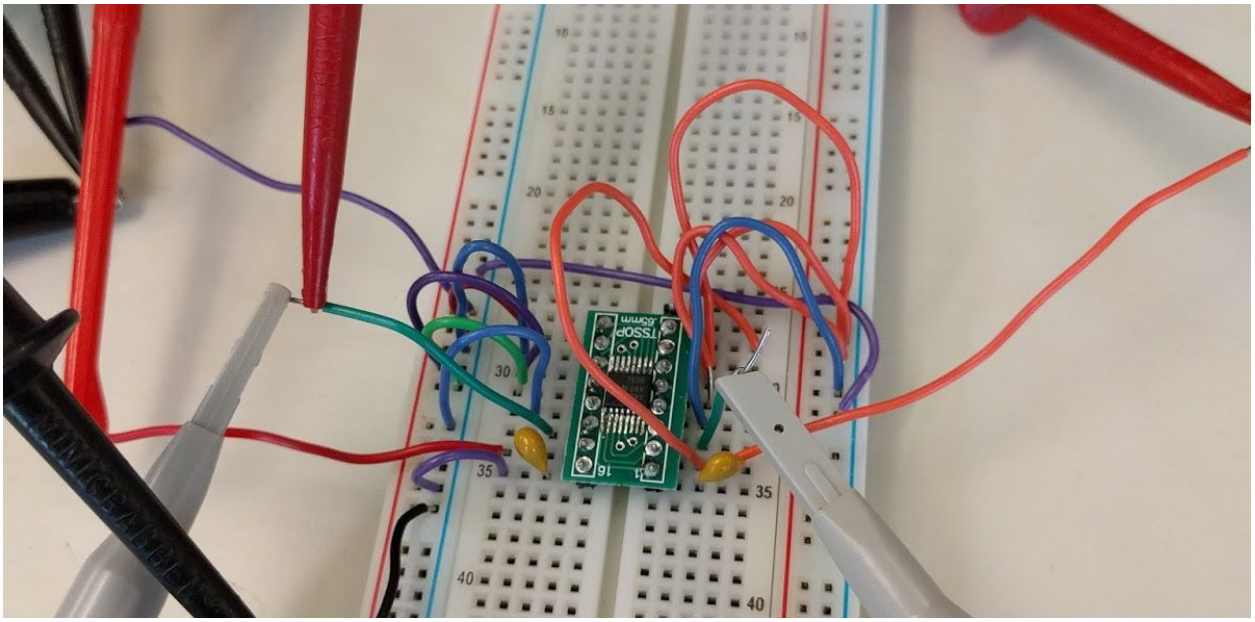


Figure 8.6: SPI level shifter test setup

The original TXB0106 on the breakout board was soldered poorly and likely was exposed to excessive heat thus it did not perform at all. This was an important lesson for the actual PCB. After soldering another device to a breakout board taking care to not put too much heat into the device the same circuit was configured and ready to test. The circuit was setup in a similar fashion to the way it was outlined in the schematic in figure 8.7.

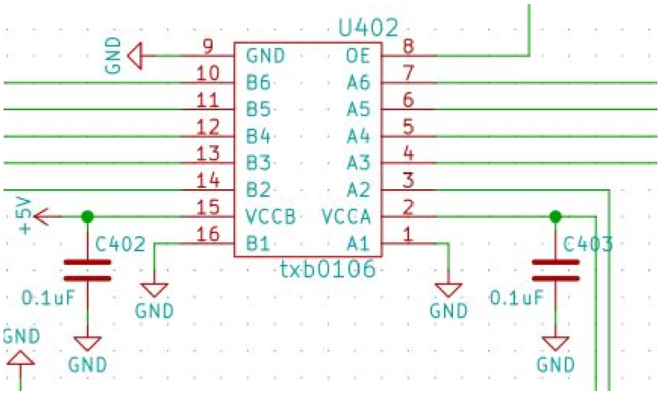


Figure 8.7: SPI level shifter circuit schematic

Only channel 2 was tested (I assumed that they would all perform the same). The rest were all tied to ground to avoid bus contention issues. To verify that the device was performing to specifications a square wave was applied to the one of the voltage domains to simulate a clock input or a data signal. Then we checked the corresponding pin on the opposite voltage domain side. Figure 8.8 shows an oscilloscope capture showing the signal propagation from one voltage domain to the other.

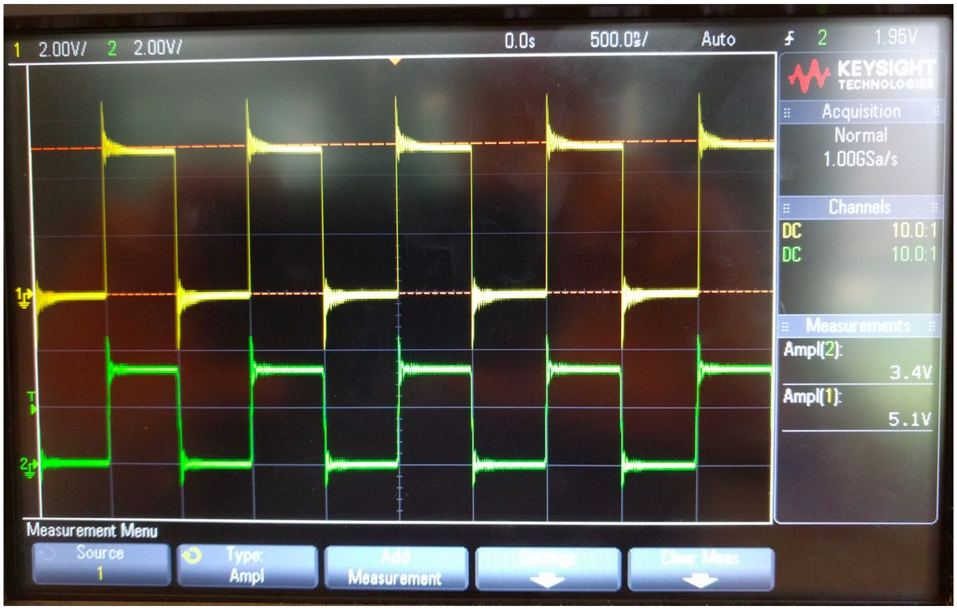
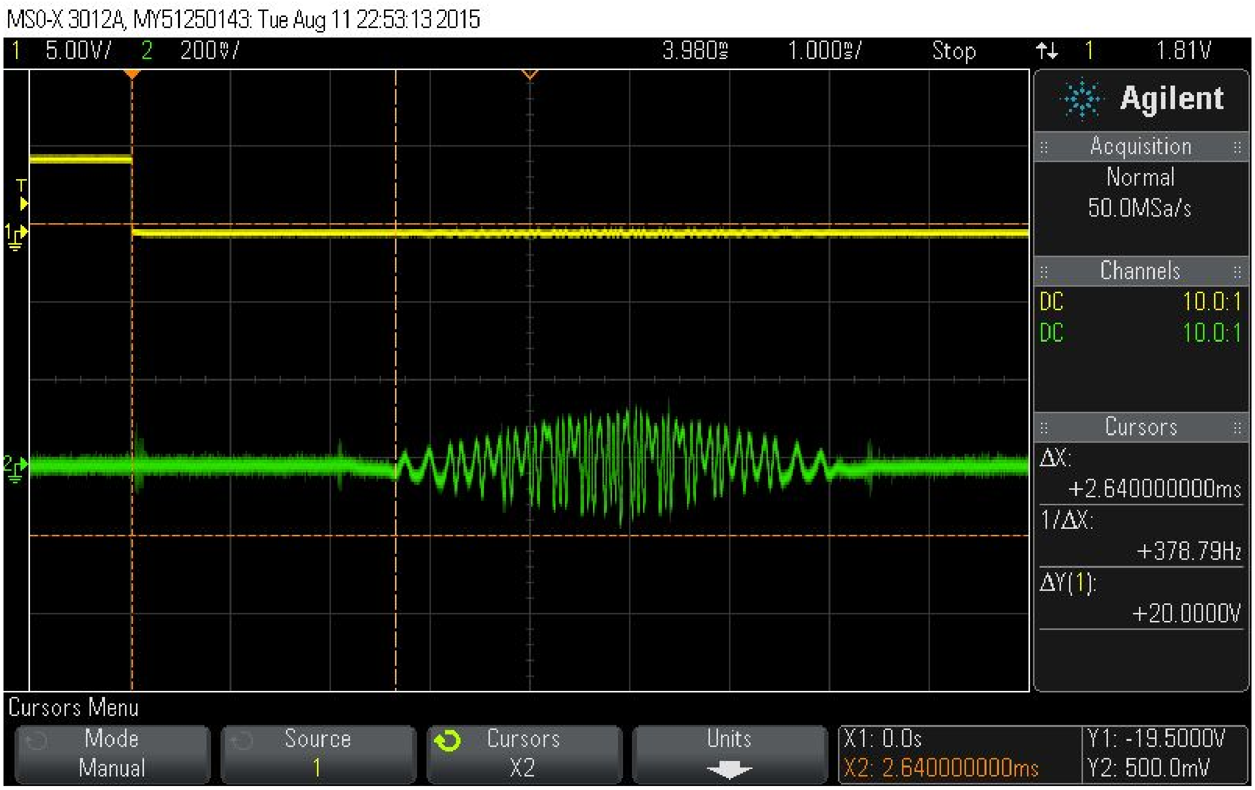


Figure 8.8: TXB0106 level shifter propagating signals from one voltage domain to the other

The measurement on the right side of figure 8.8 verify that the device is functioning correctly at 1MHz.

VIIIb. Integrated System Tests and Results

The FM synthesis works. The latency is tested by having the arduino send a pin high when a key is pressed and measuring the delay between that transition and the start of the note being played.



MSO-X 3012A, MY51250143: Tue Aug 11 22:53:13 2015

1 5.00V/ 2 200mV/ 3.980ms 1.000ms/ Stop 1 1.81V

Acquisition: Normal, 50.0MSa/s

Channels: DC 10.0:1; DC 10.0:1

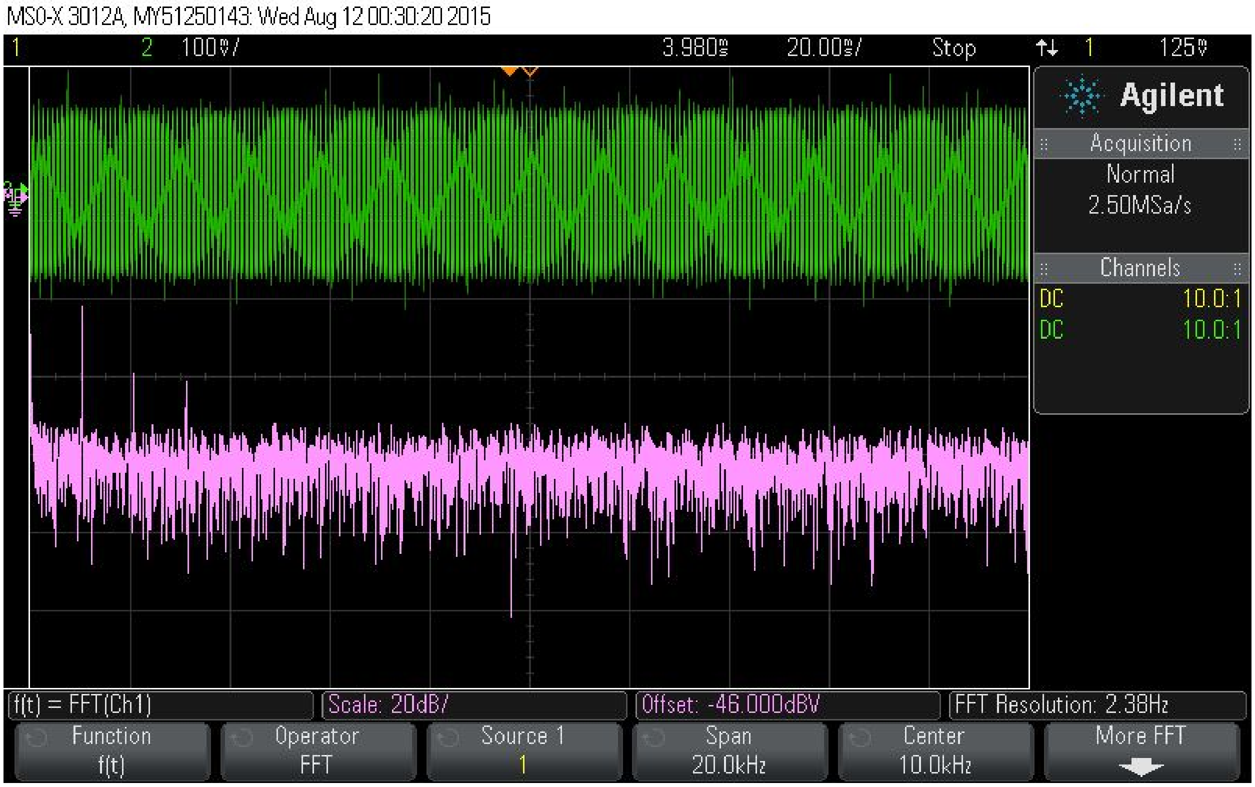
Cursors: ΔX: +2.640000000ms; 1/ΔX: +378.79Hz; ΔY(1): +20.0000V

Mode: Manual; Source: 1; Cursors: X2

X1: 0.0s X2: 2.640000000ms Y1: -19.5000V Y2: 500.0mV

Figure 8.9. FM synthesis minimum latency test.

It should be noted that latency varied. We found the minimum latency to be 2.64 milliseconds while maximum extended to 5.36 milliseconds. This meets our specifications as it is not noticed by the human ear.


MSO-X 3012A, MY51250143: Wed Aug 12 00:30:20 2015

f(t) = FFT(Ch1)

Scale: 20dB/

Offset: -46.000dBV

FFT Resolution: 2.38Hz

Figure 8.10. Signal to Noise Ratio Test.

The signal to noise ratio can be determined by having a singular tone play and performing an FFT on the signal. As shown above a tone peak appears but along with unintended harmonics. Using cursors the difference between the tone's peak and the noise level is around 45 dBV. This did not meet our original specification as we aimed to have 90 dBV. The output is admittedly a little noisy to the human ear but this could be due to the probes. While connecting the probes the noise became much more apparent with increased volume.

Table 8.1: Design versus Product Performance

	Minimum	**Maximum**
Signal to Noise Ratio	44 dBV	44 dBV
Latency	2.64 milliseconds	5.36 milliseconds

Summary: The FM synthesis has relatively met our predefined specifications. The output is a bit noisier when probing but sounds fine without. Latency is low enough for the synthesis to be considered in real time.

IX. Conclusions

Our final product wasn't what we pictured it would be at the beginning or our senior project journey. In retrospect, the original design was very ambitious given our time constraints and collective design experience. However, we did end up with a functioning product that with continued effort could reach our initial design specifications in many aspects.

Our design was successful in the following areas:

- Designed functioning PCB that was fabricated by a 3rd party vendor
- Designed and manufactured a chassis
- Created a FM synthesis engine that responds to user input on a musical keyboard

Our design fell short in the following areas:

- We were not able to create a 5 band equalizer
- We were not able to create an effects section
- We did not completely finish the chassis of the device, namely the front panel and knobs

From a technical standpoint we had two main specifications achieve 90dB of signal to noise and have a key-press latency of less than 3ms.

We failed to meet the 90dB SNR performance specification because we decided during the design phase that we didn't have the expertise to design a full or audiophile DAC so we resorted to using the DAC on the C5535 ezDSP board which saved us engineering effort but caused us to miss our audio output SNR specification. In the end our output SNR was around 40dB which isn't close to our original spec. The resulting audio was not noticeably noisy but certainly not dead quiet. In retrospect the 90dB spec may have been overly aggressive.

The key-press specification was a much more reasonable spec (and arguable more important). We were able to meet this specification with a key-press latency or around 2.5ms.

If we had to start this project over again there would a couple main items that would help us achieve our design specifications.

- Design our own onboard DAC, this would allow us to get a performance boost and be easier from an integration standpoint. Additionally it would be good design experience for the team

- Start programming the DSP much earlier in the design phase. DSP programming happened too late in our project cycle for us to be able to consider alternative software architectures and programming paradigms. If we started DSP programming from the get go we would have hit critical issues sooner and had more time to recover. Since the team was relatively inexperienced in DSP programming this hit us hard.

Lessons learned

1. *Double check pad sizes.* I had based the pad sizing for the encoders based on the datasheet drawing however the holes I had drilled in the PCB were not wide enough to fit the through hole pins of the encoders. This is an annoying problem with an easy fix.
2. *Be careful with connectors and clearances.* The right angle connector for the ezDSP is very poorly designed because it doesn't allow the ezDSP board to sit against the carrier board without wedging the connector. In our case, partially due to the fact that our surface mount soldering abilities were limited, the connector just broke of the board completely leaving us with an extremely messy rework situation. If we had know that the connector specced for the device had this limitation we could have designed a breadboard to work around the issue.
3. *Make sure 3D models are accurate.* The initial 3D print had some clearance issues that were not caught in the initial inspection because the 3D models for the ¼ inch jacks were slightly undersized.
4. *Don't underestimate software complexity.* Self explanatory.
5. *Get the team onboard early.* Each team member was writing DSP code independently linking against different libraries, building in different environments, and working in different projects. When it came time to integrate code from different team members code bases the process was messy and at times prevented us from making progress entirely.
6. *Pay attention to ERC.* I made the mistake of glossing over the issues raised by the ERC in the schematic capture tool. When the PCB came back I had forgotten to connect three adjacent pins to ground. The mistake was although the wires crossed in the schematic

no junction was present resulting in no ratsnest in the layout tool. The fix was easy (a simple solder bridge) but the potential for problems could have been large.

7. *Verify all your assumptions.* I mistakenly thought the SPI peripheral pinout on the was flipped in slave mode. This is not true. We didn't encounter this issue because we had to do a heavy work-around for the connector but a the potential for disaster was there. A simple check of the datasheet would have avoided this.

A. Analysis of Senior Project Design

Project Title: Danalog

Student's Names: Evan Lew, Vikrant Marathe, Bryan Bellin, Jordan Wong

Advisor's Name: Wayne Pilkington **Advisor's Initials:** **Date:** 6/16/17

Summary of Functional Requirements:

The Danalog produces audio via FM synthesis with two note polyphony. It has a controllable ADSR envelope and phase between the carrier and modulating wave. There is also a digital equalizer to boost/attenuate certain frequency ranges, a master volume/pitch fader, and a modulation wheel that can affect a user defined parameter with ease. Finally, up to two digital effects (reverb, flange, chorus, etc) with adjustable parameters can be applied to the audio signal. All settings are displayed between two LCD screens.

Primary Constraints:

- Given a fixed point DSP chip, we were restricted to fixed point computations, greatly preventing accuracy in calculations which could have been achieved with a floating point processor.
- Using TI's dsplib for optimized fixed point processing created a large detour that unfortunately led to no results. The FFT function for our equalizer required a twiddle factor table to multiply the signals with the factors, but we could not get the the table to be read properly in our program.

Economic:

Several hundred man-hours were put into product design, subcircuit building/testing, subcircuit integration, and programming the Danalog. A total of $792.70 was needed to make the project a reality. Several components and peripherals were needed, and a PCB had to be built and printed to connect the peripherals together. The chassis and keys were made from plastic, the PCB was made from fiberglass and copper, and many of the components as well as the development board were made of plastic, fiberglass, and various metals.

The vast majority of costs accrue in prototyping the product, researching and developing, and ordering all the necessary components. With an optimum design established, the cost to build a single Danalog will be significantly reduced, and we are confident we will be able to establish a strong customer base that will buy the product, which will compensate for the costs and eventually lead to a profit at the peak of its sales.

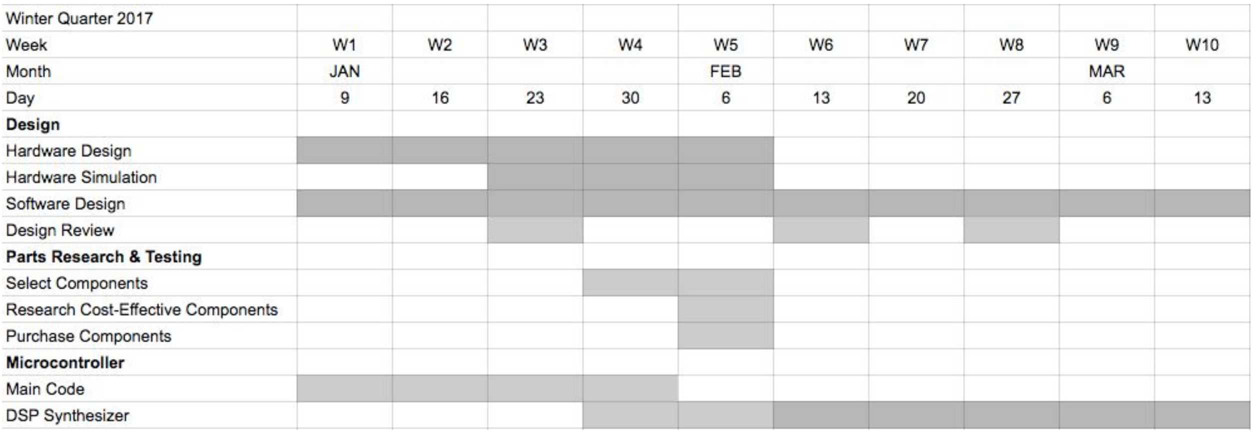
Originally, the project was estimated to cost $300. At the end, all the materials ended up costing $469.19. The bill of materials is shown as follows:

Price	Order
$106.67	K25M Keyboard from Amazon
$98.29	Sparkfun order (Buttons, LCD Screens, MIDI Connector, Jumper Adapter, Header
$119.26	Digikey order (Pots, Encoders, Rotary switches, Multiplexers, Diodes, Audio Jacks, Amplifiers)
$23.98	2 Arduino Megas
$12.96	DSP connector
$96.00	PCB
$3.61	USB adapter
$8.42	USB cable

The prototype included the purchase of many extra components in case of part failure or damage, therefore the cost would most likely be around $400, therefore selling at a price of around $450 would easily create a large profit for our company.

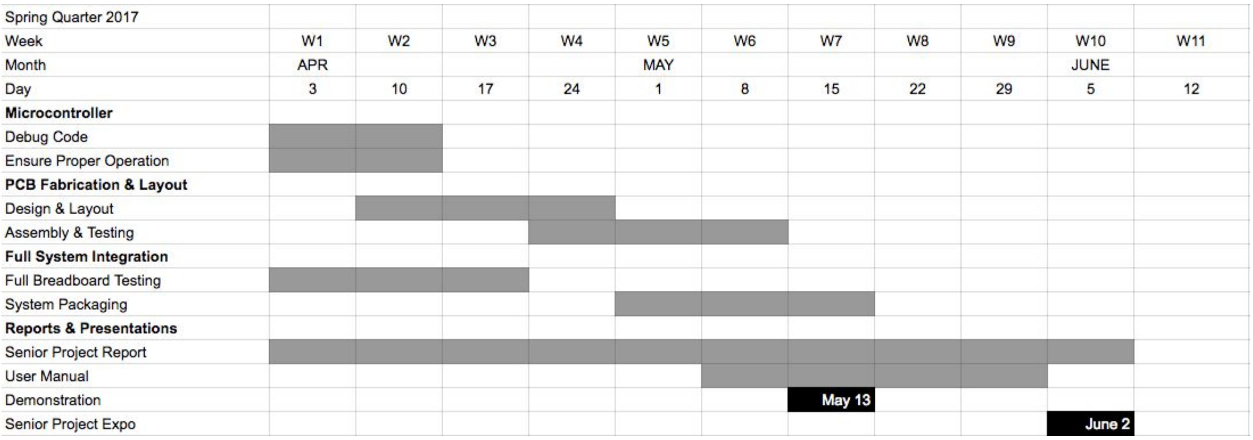
The products would emerge as soon as the first mass shipment of Danalog is complete, which would most likely occur about a year from the completion of the prototype. We expect a long shelf life of about 10 years with no maintenance costs.

Our original estimated development time is as follows:



Winter Quarter 2017										
Week	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
Month	JAN				FEB				MAR	
Day	9	16	23	30	6	13	20	27	6	13
Design										
Hardware Design	▓	▓	▓	▓	▓					
Hardware Simulation			▓	▓	▓					
Software Design	▓	▓	▓	▓	▓	▓	▓	▓	▓	▓
Design Review			░			░		░		
Parts Research & Testing										
Select Components				░	░					
Research Cost-Effective Components					░					
Purchase Components					░					
Microcontroller										
Main Code	░	░	░	░						
DSP Synthesizer				░	░	▓	▓	▓	▓	▓

Winter project timeline



Spring Quarter 2017											
Week	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11
Month	APR				MAY					JUNE	
Day	3	10	17	24	1	8	15	22	29	5	12
Microcontroller											
Debug Code	▓	▓									
Ensure Proper Operation	▓	▓									
PCB Fabrication & Layout											
Design & Layout		▓	▓	▓							
Assembly & Testing				▓	▓	▓					
Full System Integration											
Full Breadboard Testing	▓	▓	▓								
System Packaging					▓	▓	▓				
Reports & Presentations											
Senior Project Report	▓	▓	▓	▓	▓	▓	▓	▓	▓	▓	
User Manual						▓	▓	▓	▓		
Demonstration							May 13				
Senior Project Expo										June 2	

Spring project timeline (estimated)

Our actual development time:

Spring Quarter 2017											
Week	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11
Month	APR				MAY					JUNE	
Day	3	10	17	24	1	8	15	22	29	5	12
Microcontroller											
Debug Code	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Ensure Proper Operation	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
PCB Fabrication & Layout											
Design & Layout		▨	▨	▨							
Assembly & Testing				▨	▨	▨					
Full System Integration											
Full Breadboard Testing	▨	▨	▨								
System Packaging					▨	▨	▨	▨	▨	▨	
Reports & Presentations											
Senior Project Report						▨	▨	▨	▨	▨	▨
User Manual						▨	▨	▨	▨		
Demonstration											June 16
Senior Project Expo										June 2	

Once project ends, perhaps we will work to improve on the shortcomings of the prototype in order to meet the expectations of the beginning of the project.

Environmental:

Aside from the raw metal ore and plastics being manufactured to produce this product, there is no significant environmental impact from this product.

Manufacturability:

Since our PCB was printed by a third party company, it was important to verify the design is correct before sending out an order for the print. Also, the chassis had to be created one half at a time due to fact that we were using a group member's 3D printer.

Sustainability:

There are not really any issues associated with maintaining the synthesizer. One upgrade that could possibly help is using a floating point digital signal processor in order to use decimal numbers in the C code for the DSP chip, making it easier to program accurate filters for signals.

Ethical:

None.

Health and Safety:

One potential concern with safety is the possibility of ear damage due to long exposure to audio by our users, or from accidentally setting the volume too high.

Social and Political:

This product is intended to mainly impact the amateur music industry, providing music enthusiasts an opportunity to toy with different sounds and experience the Danalog synthesizer.

Development:

One important technique used for this project is the ping-pong buffer. This was necessary for real time signal generation. Essentially, while the ping buffer was being written to by the audio generator, the pong buffer was being read by the DMA, and vice versa. This prevented any loss of time in outputting the audio signals without losing samples.

B. Parts List and Costs

Yellow means that accurate price information for volume parts was unavailable

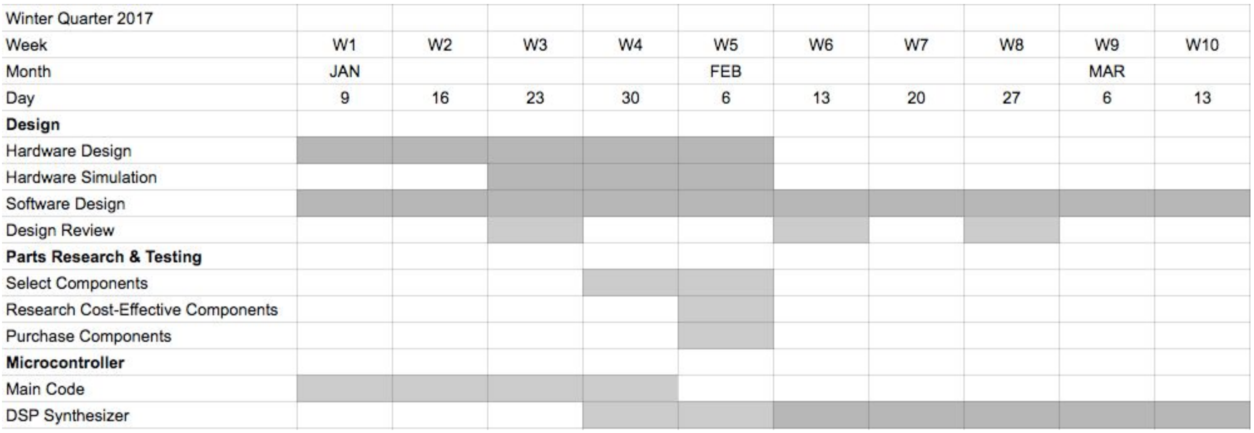
Package	Quantity	Designation	Unit cost (@ 1000)	Total
ra49c	3	1-4in_jack	$2.05	6.15
C_1206_HandSoldering	41	0.22uF	$0.02	0.74415
R_1206_HandSoldering	6	10k	$0.01	0.0366
R_1206_HandSoldering	1	220	$0.01	0.0061
SPST_SW	3	SW_PUSH	$0.86	2.58
SOIC-24W_7.5x15.4mm_Pitch1.27mm	1	CD74HC4067	$0.33	0.3296
LCD	1	20x4-lcd	$26.96	26.96
LCD	1	16x2_lcd	$22.46	22.46
LED-MATRIX-CONNECTOR	1	led-matrix-kit	$8.96	8.96
TSSOP-16_4.4x5mm_Pitch0.65mm	1	txb0106	$0.72	0.72306
MEC1-130-XX-XX-D-RAX-NP-SL	1	EZDSP-P2	$4.31	4.31
midi	1	din5-midi	$1.76	1.76
DIP-8_W9.53mm_SMD	1	6N137	$0.40	0.39502
SOIC-16_3.9x9.9mm_Pitch1.27mm	2	CD74HC4051	$0.21	0.42024
2X8-SHROUD-CON	1	k25m-connector	$2.98	2.98
D_SOD-323_HandSoldering	1	IN914	$0.03	0.02952
BOURNS-PTA3043	1	10k	$0.74	0.74
C_1206_HandSoldering	7	0.1uF	$0.02	0.12705
C_1206_HandSoldering	8	220uF	$0.55	4.38256
R_1206_HandSoldering	8	150k	$0.01	0.0488
R_1206_HandSoldering	2	100k	$0.01	0.0122
LOG-PANEL-POT	1	POT_Dual	$1.00	1
35RAMT2BHNTRX	1	35rasmt2bhntrx	$0.72	0.715
SOIC-8_3.9x4.9mm_Pitch1.27mm	2	LM833	$0.17	0.3467
Potentiometer_Bourns_PTV09A-4_Horizontal	5	10k	$0.46	2.28
C&K-RM1XX	1	rotary_switch	$1.47	1.4663

C&K-RM1XX	1	rotary_switch5	$1.47	1.4663
Potentiometer_Bourns_PTV09A-2_Vertical	2	10k	$0.46	0.912
TT-EN12-HN	19	rotary_quad_enc	$0.43	8.208
exDSP c5535	1		$100.00	100
Arduino Mega	1		$12.00	12
PCB	1		$2.76	2.762
Chassis	1		$20.00	$20.00

Grand Total **235.3112**

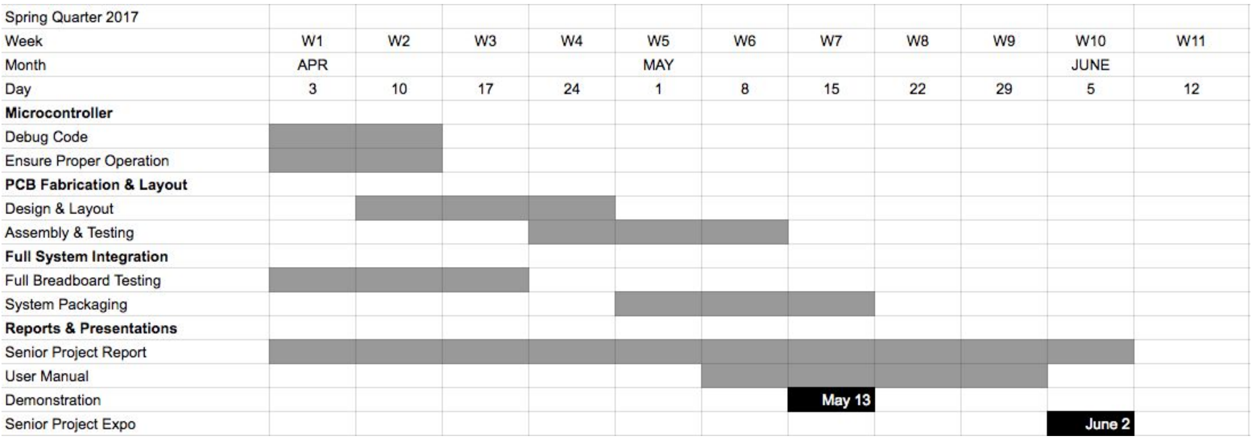
C. Project Schedule - Time Estimates & Actuals

Our original estimated development time is as follows:



Winter Quarter 2017										
Week	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
Month	JAN				FEB				MAR	
Day	9	16	23	30	6	13	20	27	6	13
Design										
Hardware Design	▨	▨	▨	▨	▨					
Hardware Simulation			▨	▨	▨					
Software Design	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Design Review			░			░		░		
Parts Research & Testing										
Select Components				░	░					
Research Cost-Effective Components					░					
Purchase Components					░					
Microcontroller										
Main Code	░	░	░	░						
DSP Synthesizer				░	░	▨	▨	▨	▨	▨

Winter project timeline



Spring Quarter 2017											
Week	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11
Month	APR				MAY					JUNE	
Day	3	10	17	24	1	8	15	22	29	5	12
Microcontroller											
Debug Code	▨	▨									
Ensure Proper Operation	▨	▨									
PCB Fabrication & Layout											
Design & Layout		▨	▨	▨							
Assembly & Testing				▨	▨	▨					
Full System Integration											
Full Breadboard Testing	▨	▨	▨								
System Packaging					▨	▨	▨				
Reports & Presentations											
Senior Project Report	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	
User Manual						▨	▨	▨	▨		
Demonstration							May 13				
Senior Project Expo										June 2	

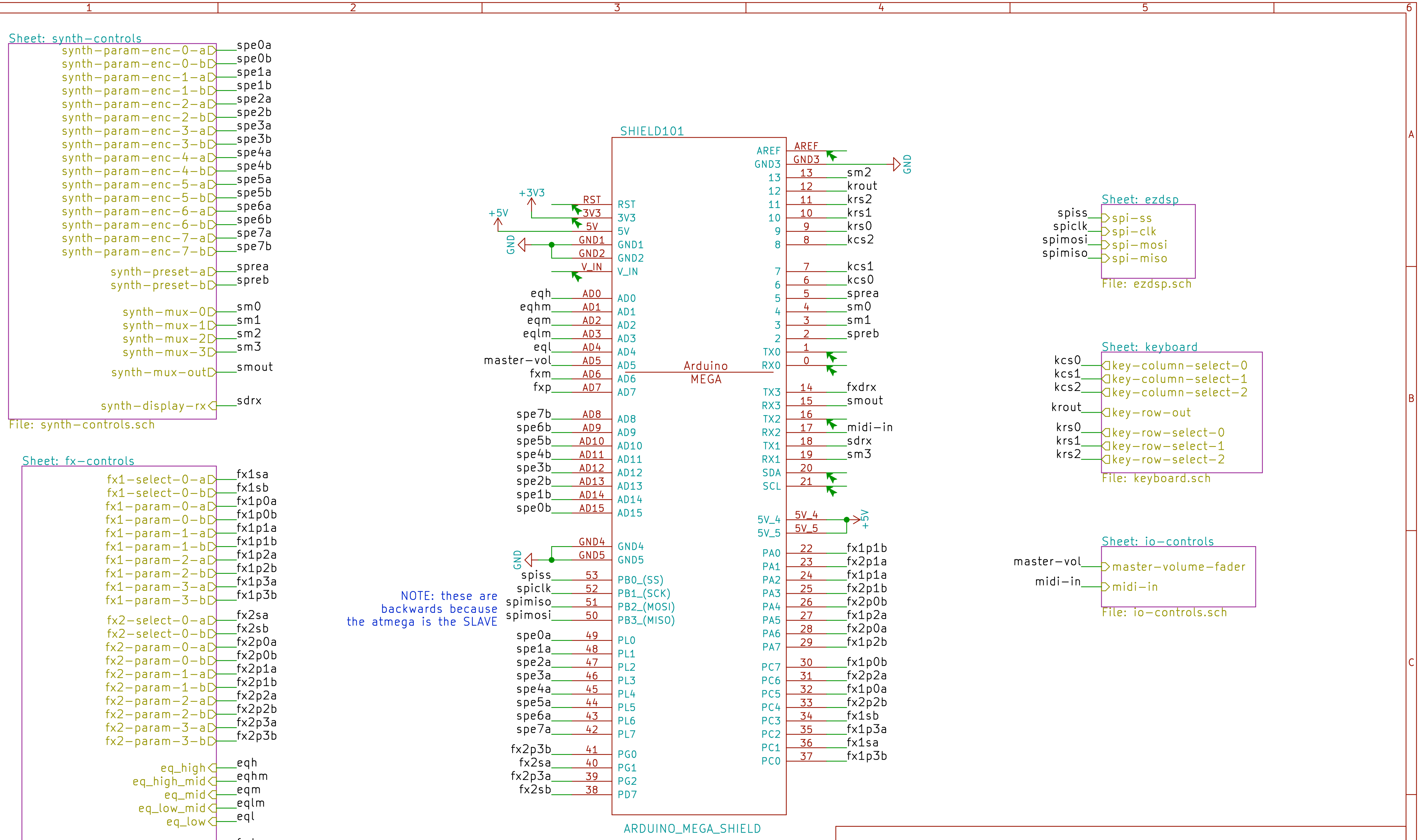
Spring project timeline (estimated)

Our actual development time is on the next page

Actual Development Time

Spring Quarter 2017											
Week	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11
Month	APR				MAY					JUNE	
Day	3	10	17	24	1	8	15	22	29	5	12
Microcontroller											
Debug Code	■	■	■	■	■	■	■	■	■	■	■
Ensure Proper Operation	■	■	■	■	■	■	■	■	■	■	■
PCB Fabrication & Layout											
Design & Layout		■	■	■							
Assembly & Testing				■	■	■					
Full System Integration											
Full Breadboard Testing	■	■	■								
System Packaging					■	■	■	■	■	■	
Reports & Presentations											
Senior Project Report						■	■	■	■	■	■
User Manual						■	■	■	■		
Demonstration											June 16
Senior Project Expo										June 2	

D. PC Board Layout

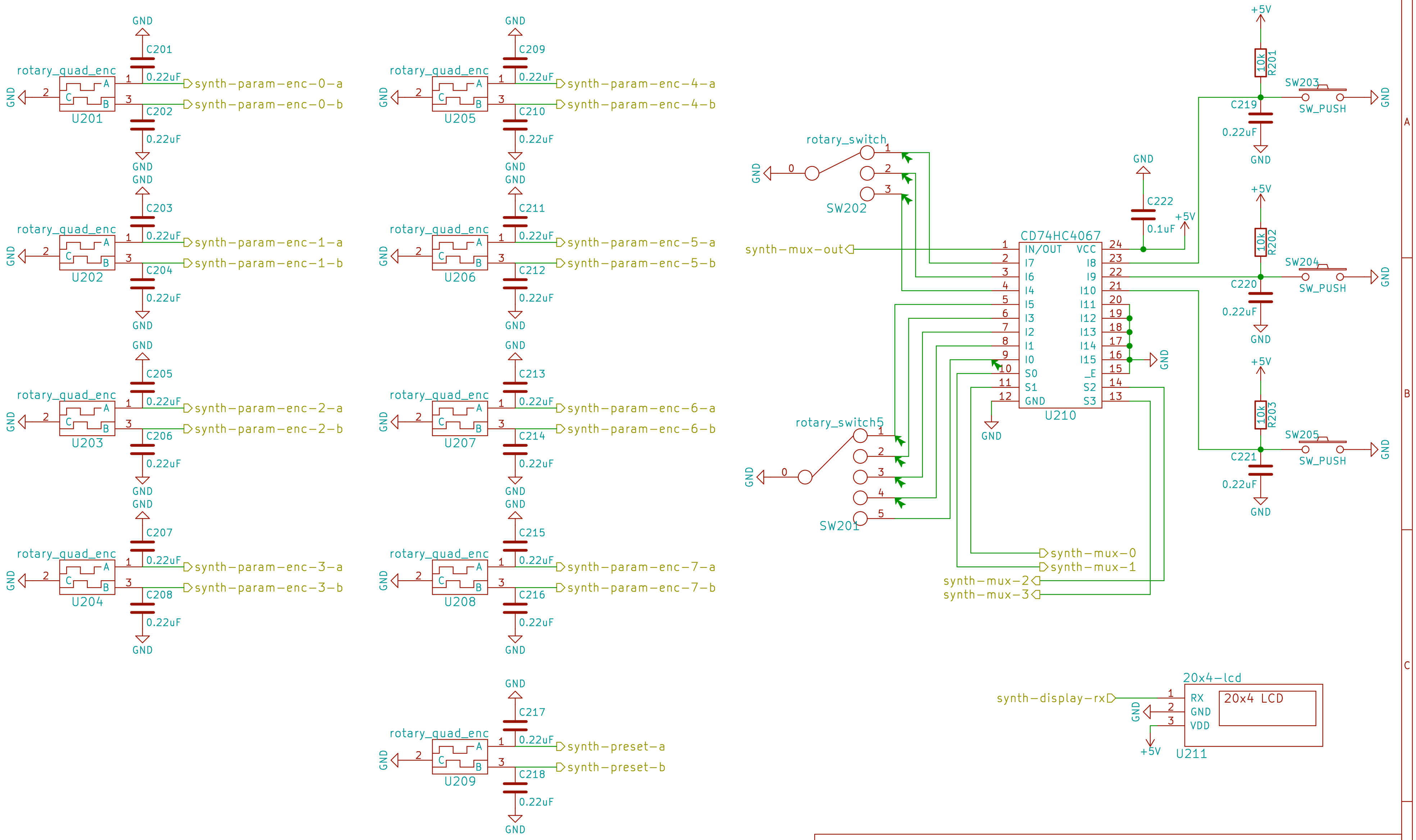


NOTE: these are
backwards because
the atmega is the SLAVE

Sheet: /
File: synth.sch

Title:

Size: A4	Date:	**Rev:**
KiCad E.D.A. kicad 4.0.6		Id: 1/6



Sheet: /synth−controls/
File: synth−controls.sch

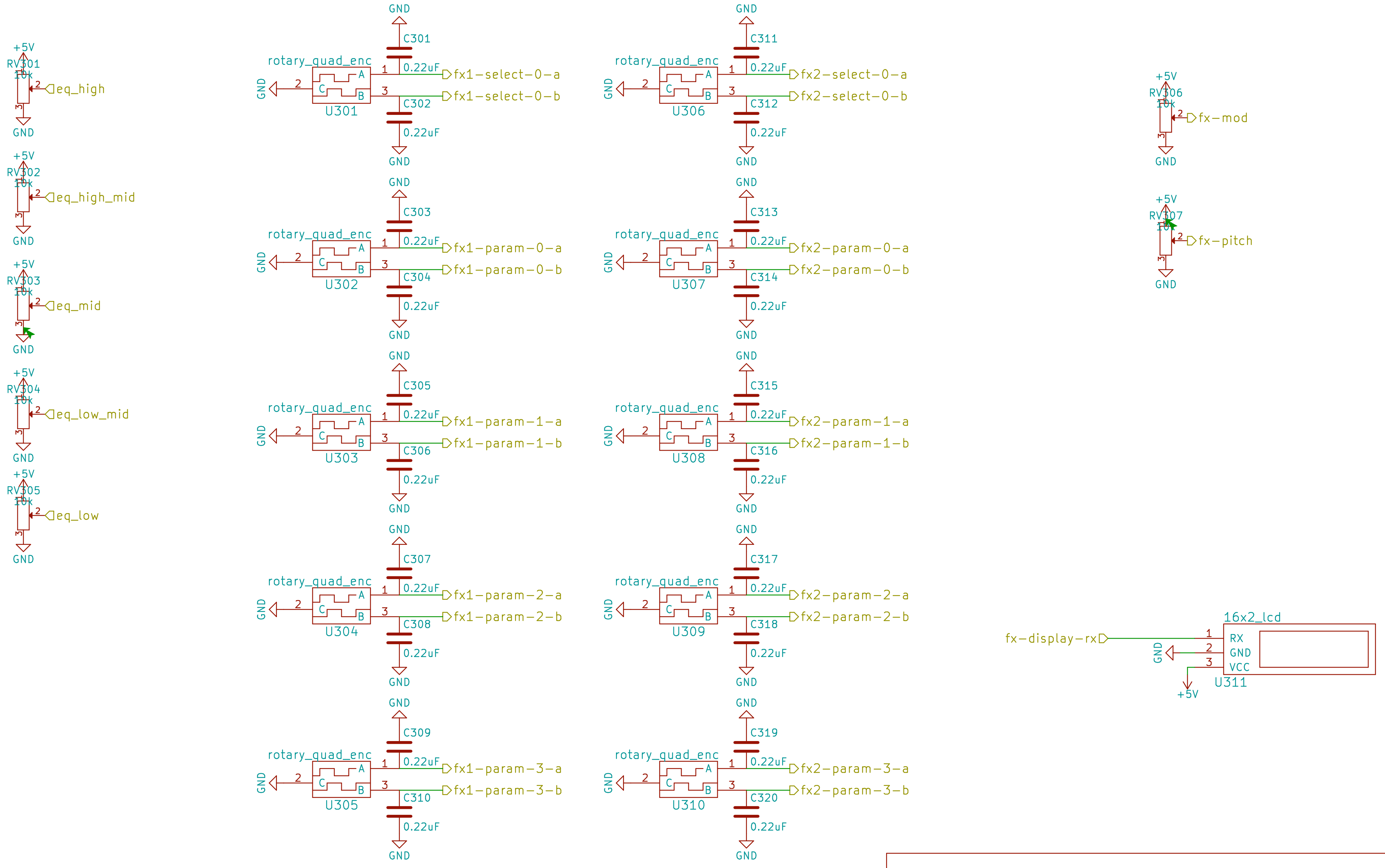
Title:

Size: A4 Date:

KiCad E.D.A. kicad 4.0.6

Rev:

Id: 2/6

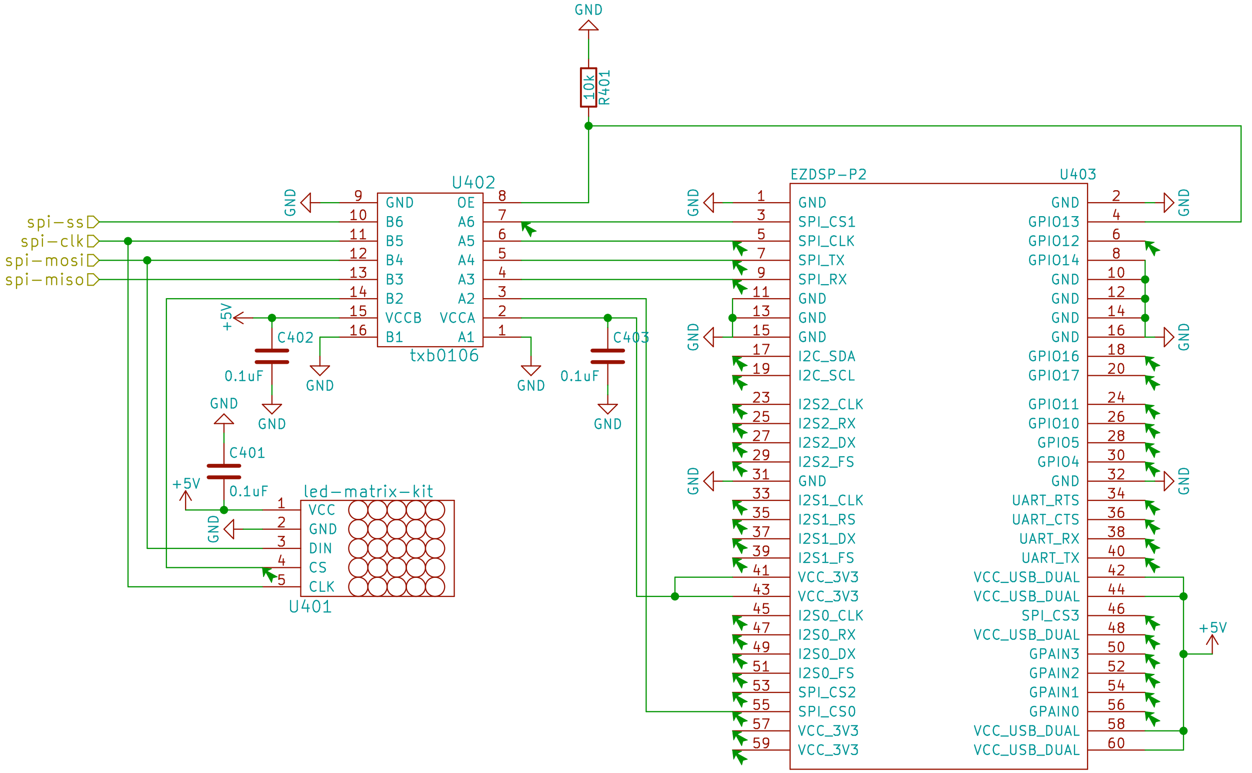


Sheet: /fx-controls/
File: fx-controls.sch

Title:

Size: A4 Date: **Rev:**

KiCad E.D.A. kicad 4.0.6 Id: 3/6



Sheet: /ezdsp/
File: ezdsp.sch

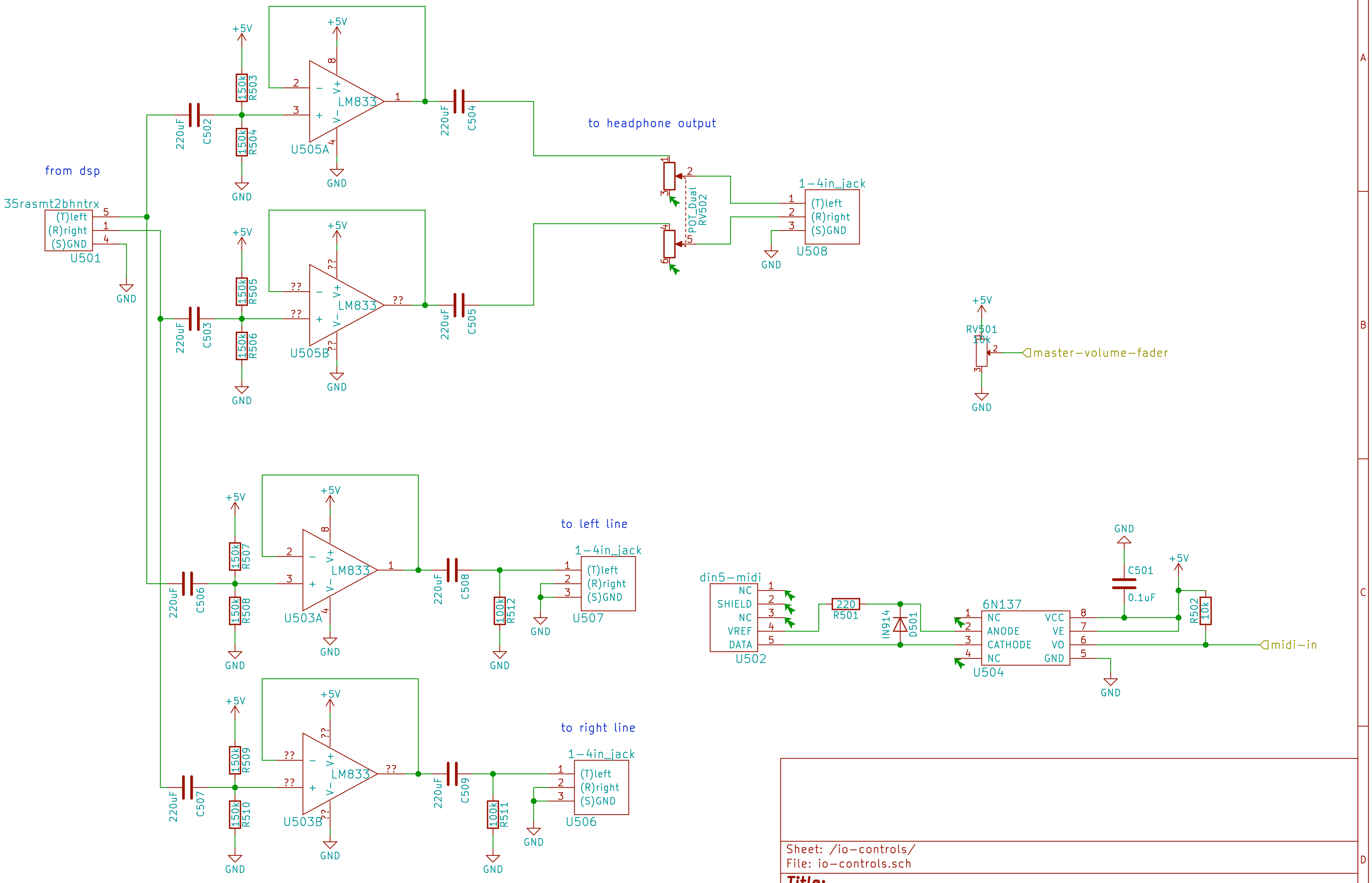
Title:

Size: A4 Date:

KiCad E.D.A. kicad 4.0.6

Rev:

Id: 4/6



Sheet: /io-controls/
File: io-controls.sch

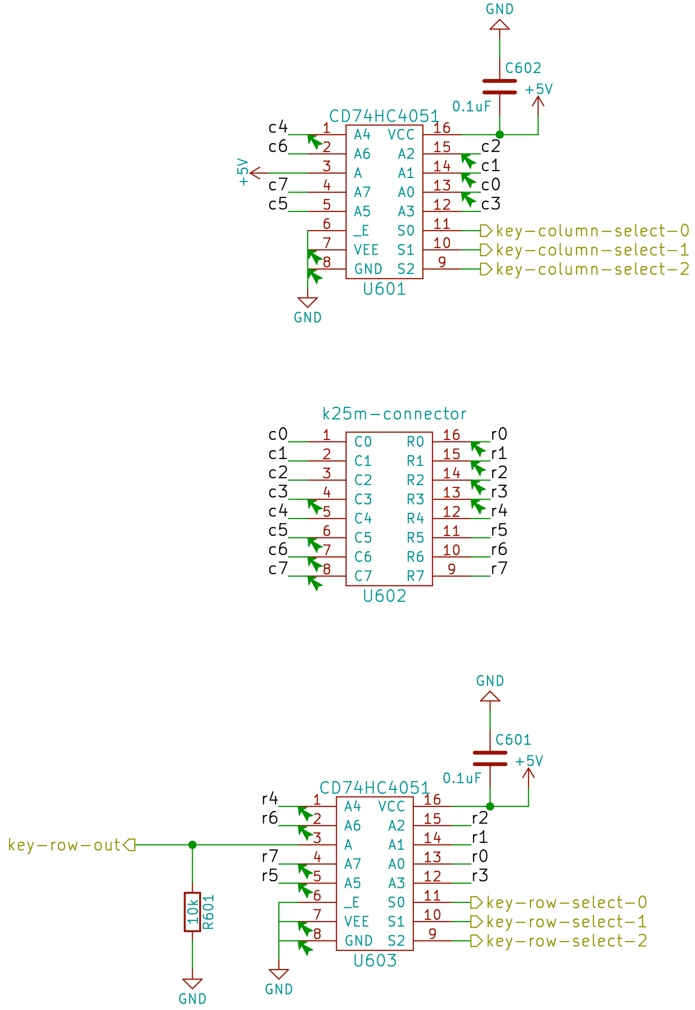
Title:

Size: A4 Date:

KiCad E.D.A. kicad 4.0.6

Rev:

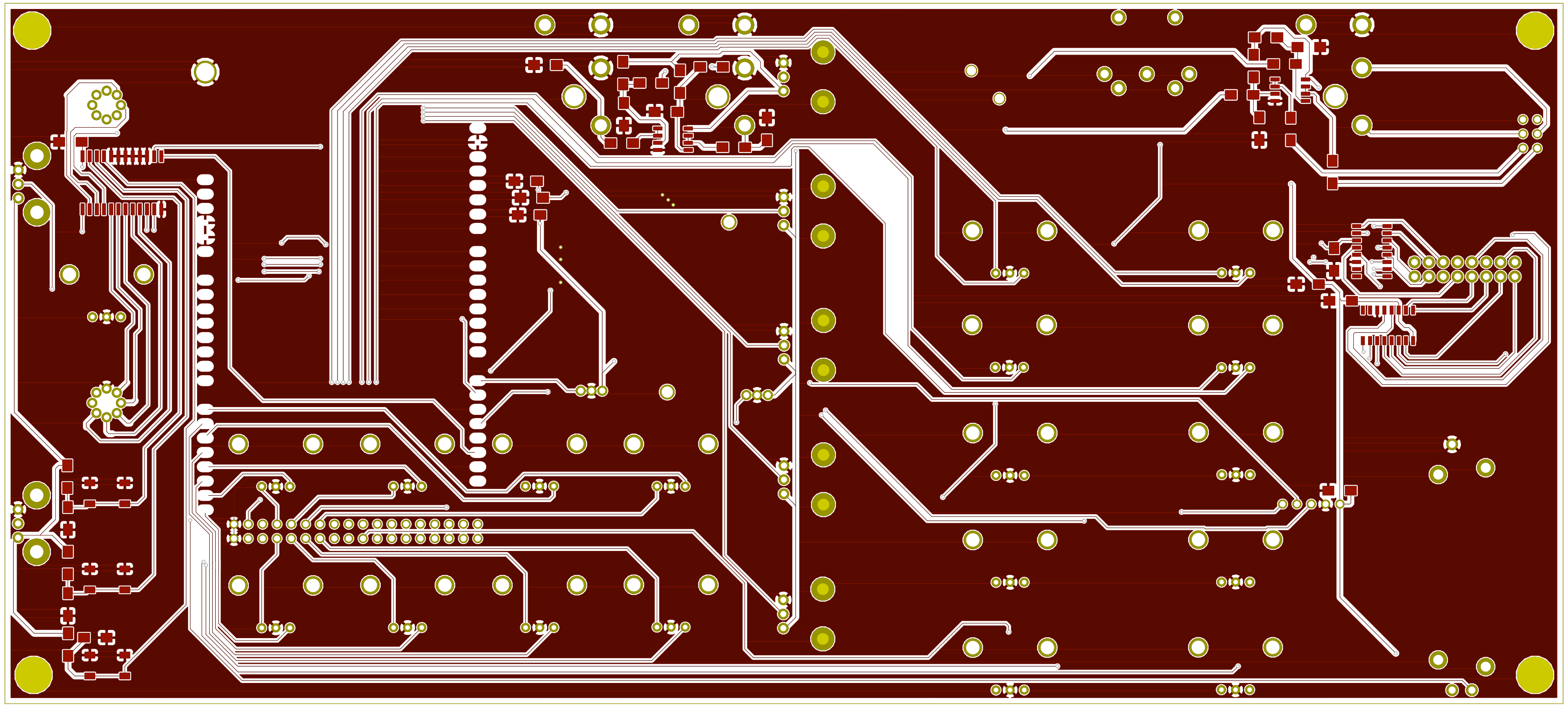
Id: 5/6



Sheet: /keyboard/
File: keyboard.sch

Title:

Size: A4	Date:	**Rev:**
KiCad E.D.A. kicad 4.0.6		Id: 6/6

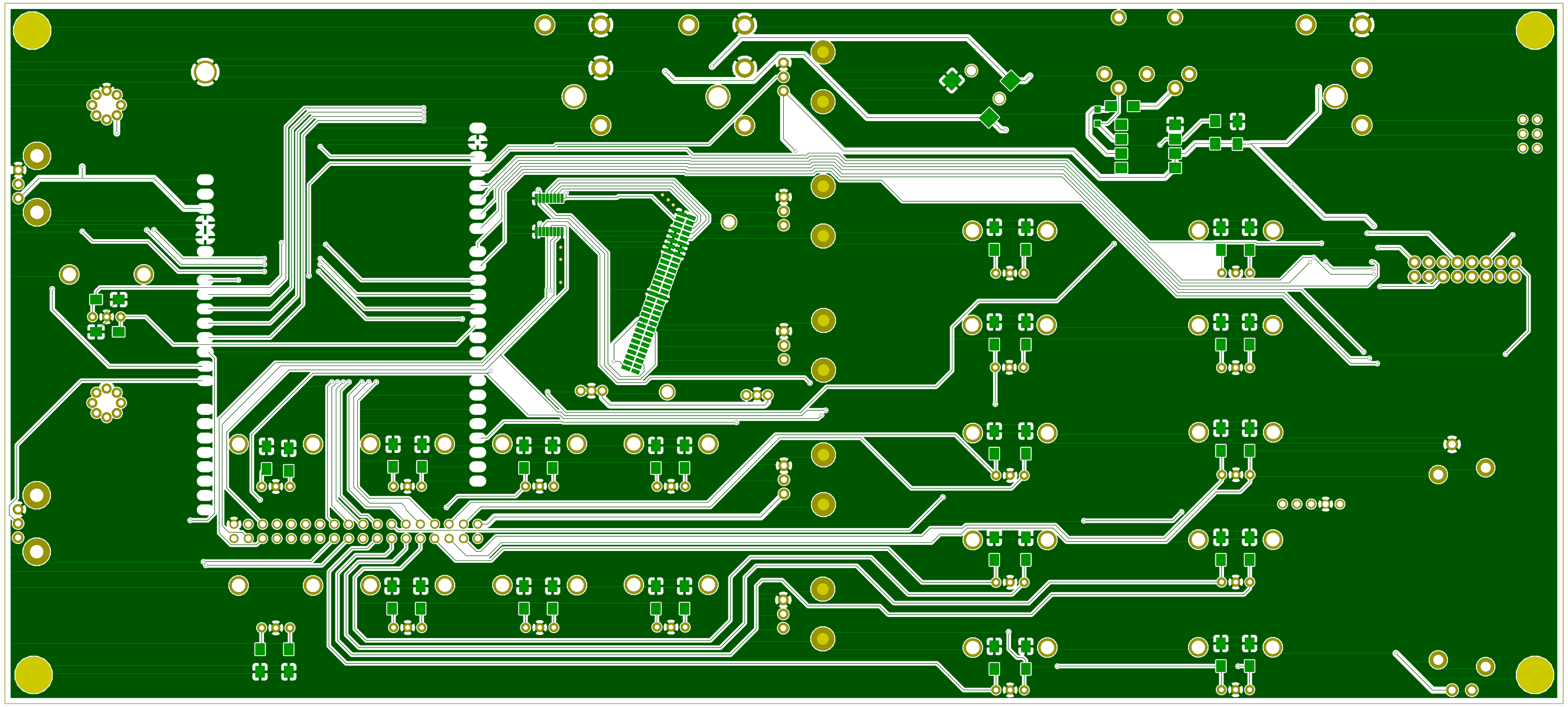


Sheet:

File: synth.kicad_pcb

Title:

Size: USLegal	Date:		Rev:
KiCad E.D.A.	kicad 4.0.6		Id: 1/1

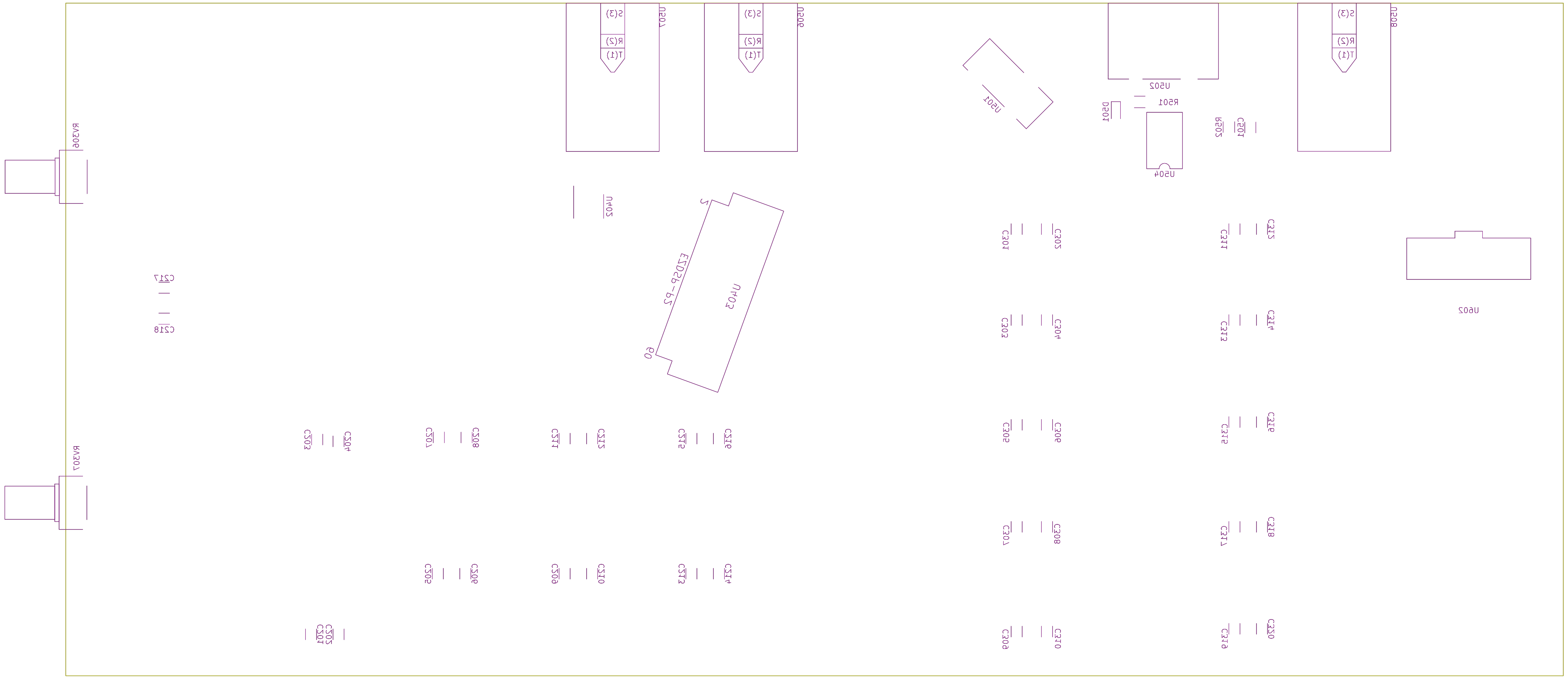


Sheet:

File: synth.kicad_pcb

Title:

Size: USLegal	Date:	Rev:
KiCad E.D.A. kicad 4.0.6		Id: 1/1

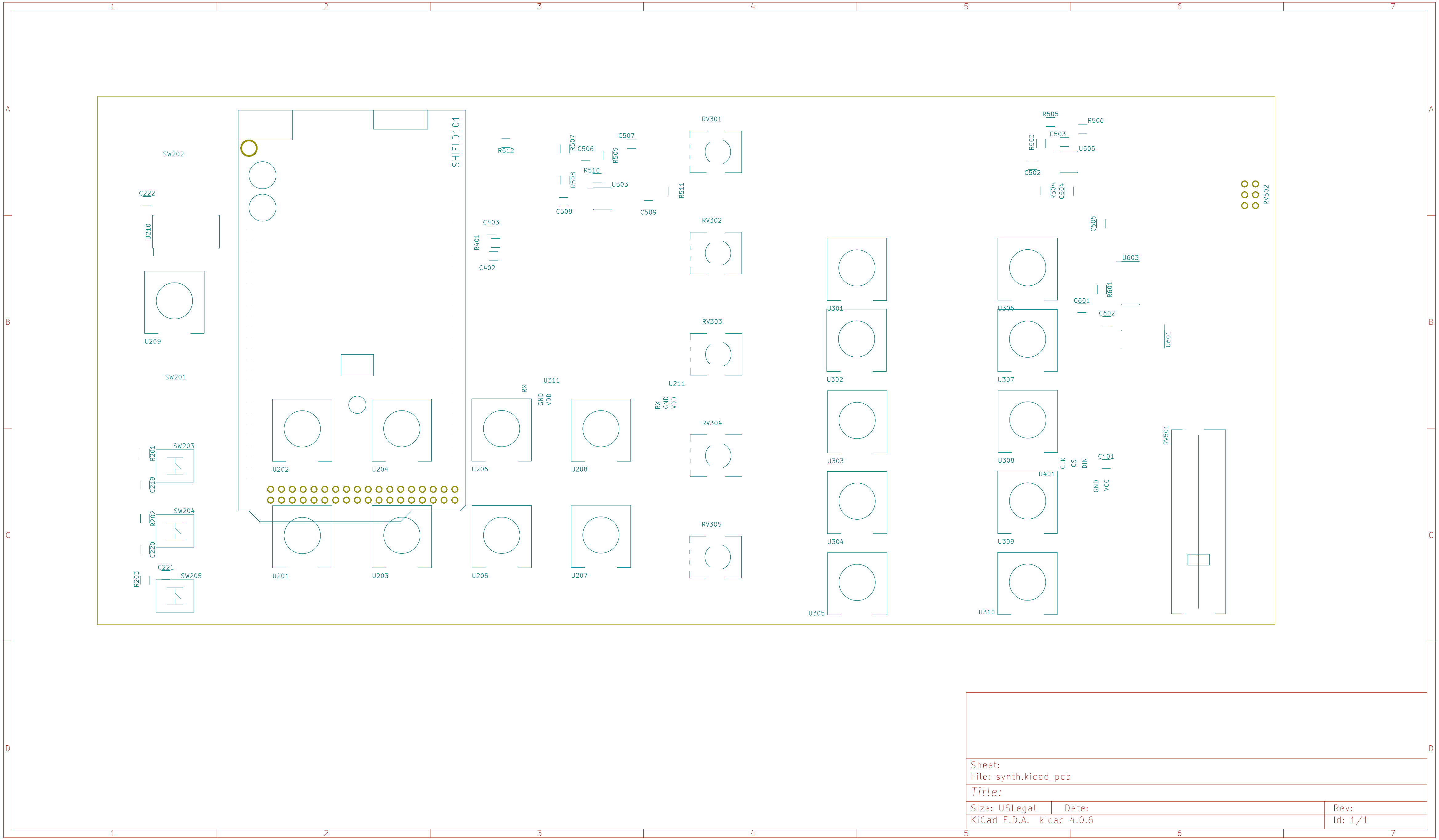


Sheet:
File: synth.kicad_pcb

Title:

Size: USLegal Date: Rev:

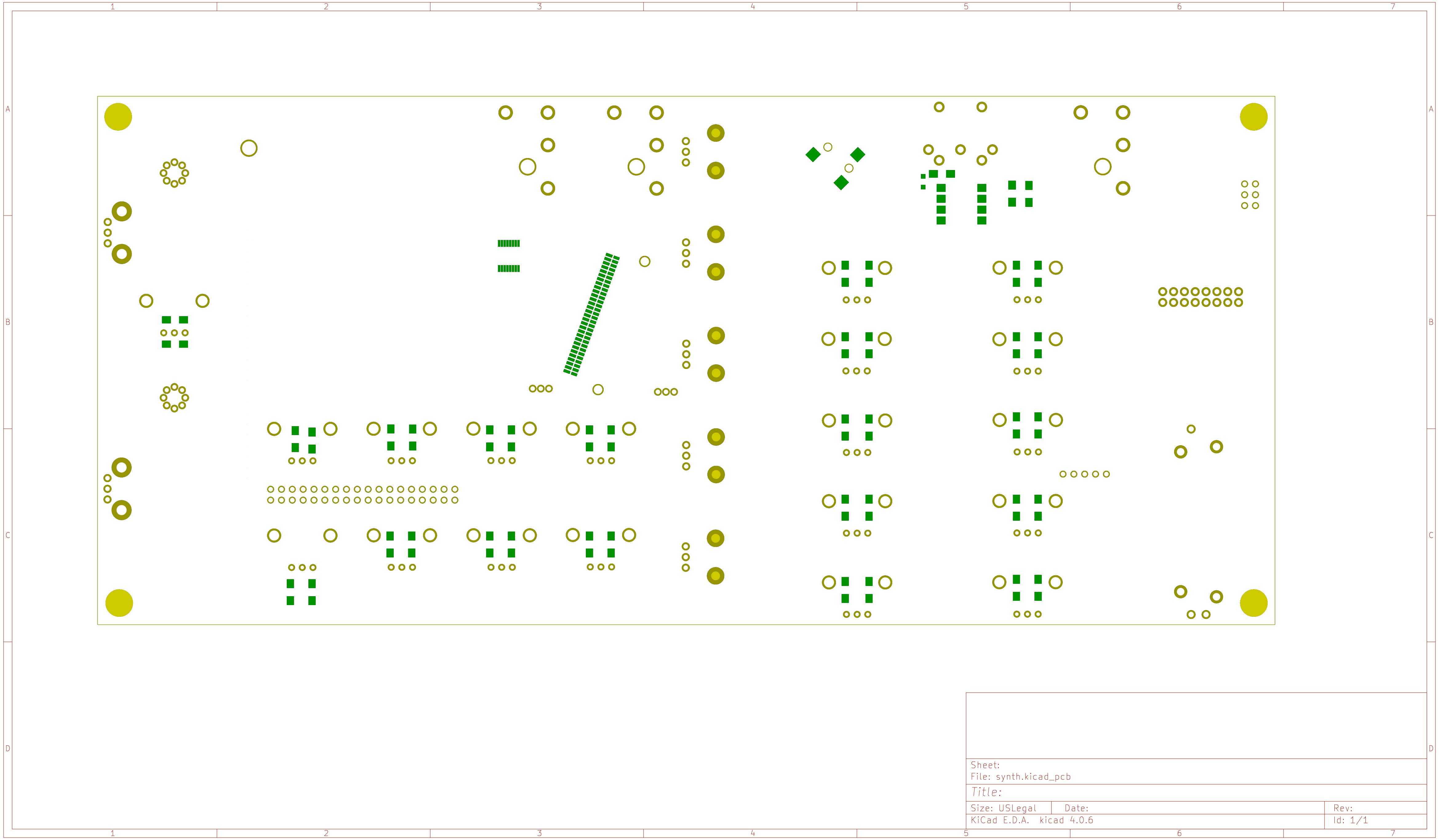
KiCad E.D.A. kicad 4.0.6 Id: 1/1



Sheet:
File: synth.kicad_pcb

Title:

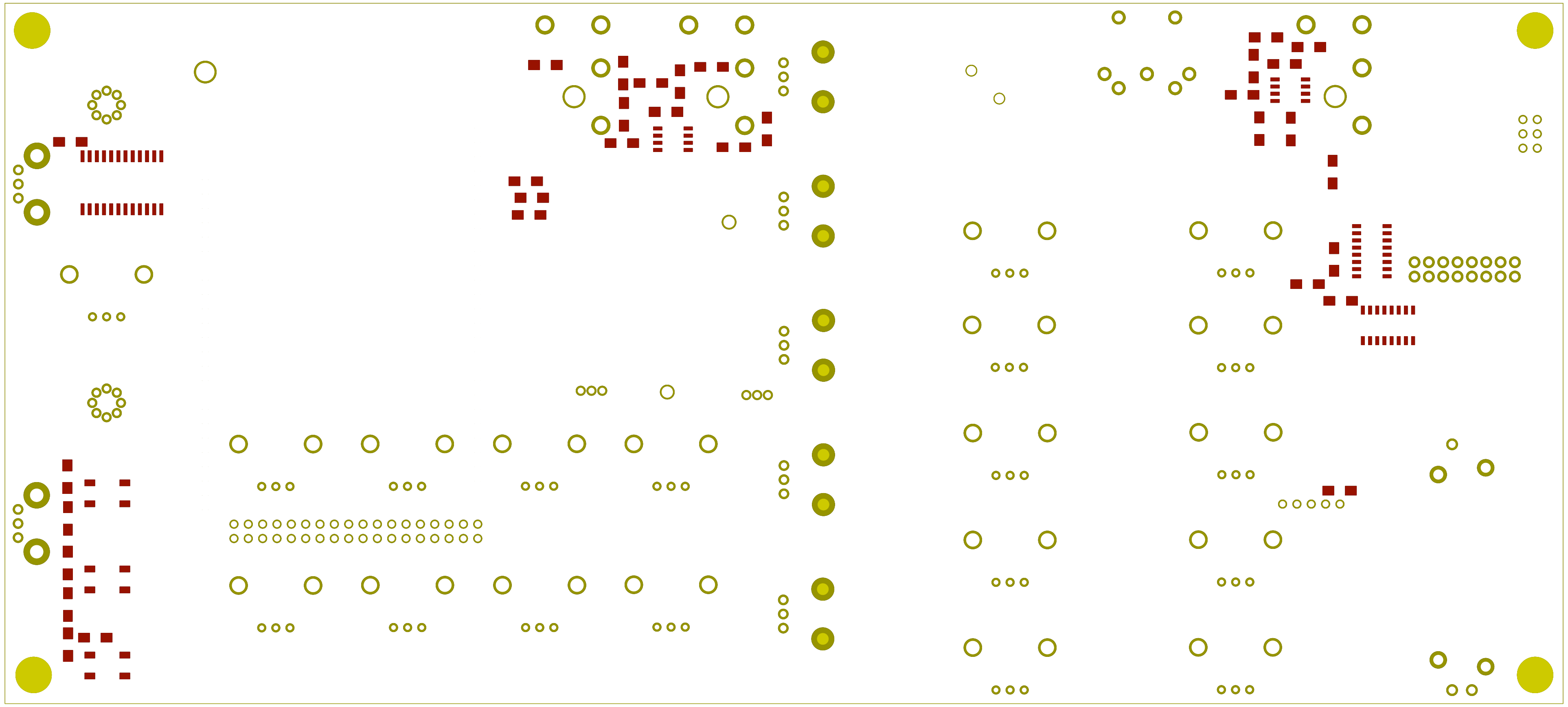
Size: USLegal	Date:	Rev:
KiCad E.D.A. kicad 4.0.6		Id: 1/1



Sheet:
File: synth.kicad_pcb

Title:

Size: USLegal	Date:	Rev:
KiCad E.D.A. kicad 4.0.6		Id: 1/1



Sheet:
File: synth.kicad_pcb

Title:

Size: USLegal | Date: | Rev:
KiCad E.D.A. kicad 4.0.6 | Id: 1/1

Sheet:
File: synth.kicad_pcb

Title:

Size: USLegal Date: Rev:
KiCad E.D.A. kicad 4.0.6 Id: 1/1

E. Program Listings (for software/firmware)

DSP directory structure

```
├── audio
│   ├── Icon\r
│   ├── singen.c
│   ├── singen.h
│   ├── sintable.c
│   └── sintable.h
├── fm
│   ├── Icon\r
│   ├── envelope.c
│   ├── envelope.h
│   ├── fm.c
│   ├── fm.h
│   ├── ringbuf.c
│   └── ringbuf.h
├── global_vars.h
├── hello.tcf
├── io
│   ├── Icon\r
│   ├── fetch_data.c
│   ├── midi.c
│   ├── midi.h
│   ├── midi_queue.c
│   └── midi_queue.h
├── main.c
└── pconfig
    ├── Icon\r
    ├── aic3204.c
    ├── aic3204.h
    ├── i2s_dma.c
    ├── i2s_dma.h
    ├── spi_config.c
    └── spi_config.h
```

```
#include "aic3204.h"

#include "ezdsp5535.h"
#include "ezdsp5535_i2c.h"

#define AIC3204_I2C_ADDR 0x18


Int16 aic3204_init() {
    /* Configure AIC3204 */
        AIC3204_rset( 0,  0x00 );  // Select page 0
        AIC3204_rset( 1,  0x01 );  // Reset codec
        EZDSP5535_waitusec(1000);  // Wait 1ms after reset
        AIC3204_rset( 0,  0x01 );  // Select page 1
        AIC3204_rset( 1,  0x08 );  // Disable crude AVDD generation from DVDD
        AIC3204_rset( 2,  0x01 );  // Enable Analog Blocks, use LDO power
        AIC3204_rset( 123,0x05 );  // Force reference to power up in 40ms
        EZDSP5535_waitusec(50000); // Wait at least 40ms
        AIC3204_rset( 0,  0x00 );  // Select page 0

        /* PLL and Clocks config and Power Up  */
        AIC3204_rset( 27, 0x0d );  // BCLK and WCLK are set as o/p;
            AIC3204(Master)
        AIC3204_rset( 28, 0x00 );  // Data ofset = 0
        AIC3204_rset( 4,  0x03 );  // PLL setting: PLLCLK <- MCLK, CODEC_CLKIN
            <-PLL CLK
        AIC3204_rset( 6,  0x07 );  // PLL setting: J=7
        AIC3204_rset( 7,  0x06 );  // PLL setting: HI_BYTE(D=1680)
        AIC3204_rset( 8,  0x90 );  // PLL setting: LO_BYTE(D=1680)
        AIC3204_rset( 30, 0x88 );  // For 32 bit clocks per frame in Master
            mode ONLY
                                   // BCLK=DAC_CLK/N =(12288000/8) = 1.536MHz =
                                       32*fs
        AIC3204_rset( 5,  0x91 );  // PLL setting: Power up PLL, P=1 and R=1
        EZDSP5535_waitusec(10000); // Wait for PLL to come up
        AIC3204_rset( 13, 0x00 );  // Hi_Byte(DOSR) for DOSR = 128 decimal or
            0x0080 DAC oversamppling
        AIC3204_rset( 14, 0x80 );  // Lo_Byte(DOSR) for DOSR = 128 decimal or
            0x0080
        AIC3204_rset( 20, 0x80 );  // AOSR for AOSR = 128 decimal or 0x0080 for
            decimation filters 1 to 6
        AIC3204_rset( 11, 0x82 );  // Power up NDAC and set NDAC value to 2
        AIC3204_rset( 12, 0x87 );  // Power up MDAC and set MDAC value to 7
        AIC3204_rset( 18, 0x87 );  // Power up NADC and set NADC value to 7
        AIC3204_rset( 19, 0x82 );  // Power up MADC and set MADC value to 2

        /* DAC ROUTING and Power Up */
        AIC3204_rset( 0,  0x01 );  // Select page 1
        AIC3204_rset( 12, 0x08 );  // LDAC AFIR routed to HPL
        AIC3204_rset( 13, 0x08 );  // RDAC AFIR routed to HPR
        AIC3204_rset( 0,  0x00 );  // Select page 0
        AIC3204_rset( 64, 0x02 );  // Left vol=right vol
        AIC3204_rset( 65, 0x00 );  // Left DAC gain to 0dB VOL; Right tracks
            Left
        AIC3204_rset( 63, 0xd4 );  // Power up left,right data paths and set
            channel
        AIC3204_rset( 0,  0x01 );  // Select page 1
        AIC3204_rset( 16, 0x00 );  // Unmute HPL , 0dB gain
        AIC3204_rset( 17, 0x00 );  // Unmute HPR , 0dB gain
        AIC3204_rset( 9 , 0x30 );  // Power up HPL,HPR
        EZDSP5535_waitusec(100 );  // Wait

        /* ADC ROUTING and Power Up */
        AIC3204_rset( 0,  0x01 );  // Select page 1
        AIC3204_rset( 52, 0x30 );  // STEREO 1 Jack
                                   // IN2_L to LADC_P through 40 kohm
        AIC3204_rset( 55, 0x30 );  // IN2_R to RADC_P through 40 kohmm
        AIC3204_rset( 54, 0x03 );  // CM_1 (common mode) to LADC_M through 40
            kohm
        AIC3204_rset( 57, 0xc0 );  // CM_1 (common mode) to RADC_M through 40
            kohm
        AIC3204_rset( 59, 0x00 );  // MIC_PGA_L unmute
        AIC3204_rset( 60, 0x00 );  // MIC_PGA_R unmute
        AIC3204_rset( 0,  0x00 );  // Select page 0
        AIC3204_rset( 81, 0xc0 );  // Powerup Left and Right ADC
        AIC3204_rset( 82, 0x00 );  // Unmute Left and Right ADC
        AIC3204_rset( 0,  0x00 );  // Select page 0
        EZDSP5535_waitusec(100 );  // Wait

        return 0;
}

Int16 AIC3204_rset( Uint16 regnum, Uint16 regval )
{
    Uint16 cmd[2];
    cmd[0] = regnum & 0x007F;       // 7-bit Device Register
    cmd[1] = regval;                // 8-bit Register Data

    EZDSP5535_waitusec( 300 );

    return EZDSP5535_I2C_write( AIC3204_I2C_ADDR, cmd, 2 );
}
```

```
#include "ezdsp5535.h"

Int16 aic3204_init( void );
Int16 AIC3204_rset( Uint16 regnum, Uint16 regval );
```

```
/*
 * envelope.c
 *
 *  Created on: May 29, 2017
 *      Author: evan
 */
#include "envelope.h"

void createEnvelopeConfig(EnvelopeConfig *ec, Int16 a, Int16 d, Int16 s, Int16
    r) {
    ec->attack = a;
    ec->decay = d;
    ec->sustain = s;
    ec->release = r;

    ec->attack_step_cnt = a * 4;
    ec->decay_step_cnt = d * 4;
    ec->release_step_cnt = r * 4;
}

Envelope createEnvelope(EnvelopeConfig *ec) {
    Envelope e;
    e.env_config = ec;
    e.env_state = ENV_ATTACK;
    e.env_val = 0;
    e.step_cnt = 0;
    return e;
}
Int16 envelopeIncrement(Envelope *e) {
    if (e->env_state == ENV_ATTACK) {
        if (e->env_val <= 255) {
            if (e->step_cnt < e->env_config->attack_step_cnt) {
                e->step_cnt++;
            } else {
                e->step_cnt = 0;
                e->env_val++;
            }
        } else {
            e->env_state = ENV_DECAY;
            e->step_cnt = 0;
        }
    }
    else if (e->env_state == ENV_DECAY) {
        if (e->env_val > e->env_config->sustain) {
            if (e->step_cnt < e->env_config->decay_step_cnt) {
                e->step_cnt++;
            } else {
                e->step_cnt = 0;
                e->env_val--;
            }
        } else {
            e->env_state = ENV_SUSTAIN;
            e->env_val = e->env_config->sustain;
        }
    } else if (e->env_state == ENV_RELEASE) {
        if (e->env_val > 0) {
```

```c
            if (e->step_cnt < e->env_config->release_step_cnt) {
                e->step_cnt++;
            } else {
                e->step_cnt = 0;
                e->env_val--;
            }
        } else {
            e->env_state = ENV_INACTIVE;
            e->env_val = 0;
        }
    }
    return e->env_val;
}
```

```
/*
 * envlope.h
 *
 *  Created on: May 29, 2017
 *      Author: evan
 */

#ifndef ENVLOPE_H_
#define ENVLOPE_H_

#include <std.h>

typedef enum {
    ENV_ATTACK, ENV_DECAY, ENV_SUSTAIN, ENV_RELEASE, ENV_INACTIVE
} EnvelopeState;



typedef struct {
    Int16 attack;
    Int16 attack_step_cnt;

    Int16 decay;
    Int16 decay_step_cnt;

    Int16 sustain;

    Int16 release;
    Int16 release_step_cnt;

} EnvelopeConfig;

typedef struct {
    EnvelopeConfig *env_config;
    // State variables
    EnvelopeState env_state;
    Int16 env_val;
    Int16 step_cnt;
} Envelope;

void createEnvelopeConfig(EnvelopeConfig *ec, Int16 a, Int16 d, Int16 s, Int16
    r) ;
Envelope createEnvelope(EnvelopeConfig *ec);
Int16 envelopeIncrement(Envelope *e);



#endif /* ENVLOPE_H_ */
```

```
/* Standard C includes */
#include <stdio.h>

/* DSP/BIOS headers */
#include <std.h>
#include <tsk.h>
#include "hellocfg.h"

/* ezDSP C5535 board specific headers */
#include "ezdsp5535.h"

/* C55xx chip support library headers */


/* Danalog headers */
#include "../pconfig/aic3204.h"
#include "../pconfig/i2s_dma.h"
#include "../pconfig/spi_config.h"
#include "../io/midi_queue.h"
#include "../global_vars.h"

volatile Int16 nothing = 0;

Void spi_get_midi( void )
{
    while (1) {
        Uint16 message = SPI_MIDI_CMD;
        //TSK_disable();

        while (1) {
            spi_write(&message, 1);
            spi_read(midi, 3);
            // if the fist byte is 0, no new midi information
            if ( midi[0] == 0x00) {
                break;
            }
            else if (midi[0] != 0x90 && midi[0] != 0x80) {
                break;
            }
            //if ( (midi[0] & 0x80) == 0) { break; } // this is a hack for the
                slow avr isr

            MidiPacket p;
            p.midi_cmd = midi[0];
            p.note_id  = midi[1];
            p.velocity = midi[2];

            midi_buffer_write(p);

        }
        //TSK_enable();
        TSK_sleep(2);
    }
}

Void spi_get_interface_controls( void )
```

```
{
    Uint16 counter = 0;
    Uint16 message;
    while (1) {
        switch(counter) {
        case 0:
            message = SPI_SWT_CMD;
            TSK_disable();
            spi_write(&message, 1);
            spi_read(&switches, 1);
            TSK_enable();
            break;
        case 1:
            message = SPI_ENC_CMD;
            TSK_disable();
            spi_write(&message, 1);
            spi_read(encoders, 19);
            TSK_enable();
            break;
        case 2:
            message = SPI_POT_CMD;
            TSK_disable();
            spi_write(&message, 1);
            spi_read(pots, 8);
            TSK_enable();
            break;
        default:
            while (1); // error
        }

        counter = (counter + 1) % 3;
        TSK_sleep(1000);
    }
}
```

```c
/*
 * gen_sound.c
 *
 *  Created on: May 6, 2017
 *      Author: evan
 */

#include "fm.h"

/* Standard C includes */
#include <std.h>
#include <stdio.h>
#include <tsk.h>

/* DSP/BIOS headers */
#include "hellocfg.h"

/* ezDSP C5535 board specific headers */
#include "ezdsp5535.h"

/* Danalog headers */
#include "../audio/singen.h"
#include "../audio/sintable.h"
#include "../global_vars.h"
#include "../io/midi.h"
#include "envelope.h"
#include "ringbuf.h"

EnvelopeConfig mod_env_cfg, car_env_cfg;

FMNote note;


Int16 mod_ratio = 1;
Int16 mod_depth = 1;


FMNote midi_to_fm_note(MidiPacket* p) {
    FMNote n;
    n.pitch = convert_to_freq(p->note_id);
    n.velocity = (Int16) p->velocity;
    n.mod_env = createEnvelope(&mod_env_cfg);
    n.car_env = createEnvelope(&car_env_cfg);
    sin_compute_params(&n.mod_sin, n.pitch * mod_ratio);
    sin_compute_params(&n.car_sin, n.pitch);
    return n;
}


Void generate_samples_tsk( Void )
{



    createEnvelopeConfig(&car_env_cfg, 0, 0, 250, 100);
    createEnvelopeConfig(&mod_env_cfg, 0, 0, 100, 100);
    while (1) {
        SEM_pend(&ping_pong_sem, SYS_FOREVER);
        //createEnvelopeConfig(&car_env_cfg, encoders[10], encoders[11],
            encoders[12], encoders[13]);
        //createEnvelopeConfig(&mod_env_cfg, encoders[14], encoders[15],
            encoders[16], encoders[17]);

        MidiPacket p;
        if (midi_buffer_size() > 0) {
            while (midi_buffer_size() > 0) {
                p = midi_buffer_read();

                if (midi_packet_type(p) == MIDI_NOTE_ON) {
                    add_note(&p, mod_ratio);
                }
                else if (midi_packet_type(p) == MIDI_NOTE_OFF) { //
                   MIDI_NOTE_OFF
                    release_note(&p);
                }
            }


        }




        // determine which buffer to fill
        Int16 *left_output, *right_output;
        if (CSL_DMA1_REGS->DMACH0TCR2 & 0x0002) { // last xfer: pong
            left_output = left_pong;
            right_output = right_pong;
        } else {
            left_output = left_ping;
            right_output = right_ping;
        }

        Int16 i;
#pragma MUST_ITERATE(I2S_DMA_BUFFER_SIZE,I2S_DMA_BUFFER_SIZE)
        for (i = 0; i < I2S_DMA_BUFFER_SIZE; i++) {
            Int16 output = 0;
            Int16 counter;
            for (counter = 0; counter < NOTE_BUF_LEN; counter++) {
                FMNote *n = &note_buf[counter];
                if (n->car_env.env_state != ENV_INACTIVE) {
                    Int32 mod = ((envelopeIncrement(&n->mod_env) * (Int32)
                        sin_gen(&n->mod_sin, 0))) >> 8;
                    Int32 mod_scaled = (mod >> 3) * mod_depth;

                    output += ((envelopeIncrement(&n->car_env) * (Int32)
                        sin_gen(&n->car_sin, mod_scaled))) >> 10;
                }
            }

            left_output[i] = output;
            right_output[i] = output;
        } // end for
    } // end while (1)
}
```

```
/*
 * gen_sound.h
 *
 *  Created on: May 6, 2017
 *      Author: evan
 */

#ifndef GEN_SOUND_H_
#define GEN_SOUND_H_

#include <std.h>

#include "../io/midi.h"
#include "envelope.h"
#include "../audio/singen.h"


typedef struct {
    Int16 pitch;
    Int16 velocity;
    Envelope mod_env;
    Envelope car_env;
    SinState mod_sin;
    SinState car_sin;


} FMNote;

FMNote midi_to_fm_note(MidiPacket* p);





#endif /* GEN_SOUND_H_ */
```

```c
/*
 * global_vars.h
 *
 *  Created on: May 25, 2017
 *      Author: evan
 */

#ifndef GLOBAL_VARS_H_
#define GLOBAL_VARS_H_

#include "ezdsp5535.h"
#include "io/midi_queue.h"
#include "fm/fm.h"
#include "audio/singen.h"

// SPI recieving data structures
extern Uint16 encoders[19];
extern Uint16 pots[8];
extern Uint16 switches;
extern Uint16 midi[3];

// I2S/DMA buffers for audio output
#define I2S_DMA_BUFFER_SIZE 128
extern Int16 left_ping[I2S_DMA_BUFFER_SIZE];
extern Int16 left_pong[I2S_DMA_BUFFER_SIZE];

extern Int16 right_ping[I2S_DMA_BUFFER_SIZE];
extern Int16 right_pong[I2S_DMA_BUFFER_SIZE];

// Midi buffer
extern MidiPacket midi_buffer[];

extern FMNote note;
extern SinState ss_carrier;
extern SinState ss_mod;



#endif /* GLOBAL_VARS_H_ */
```

```c
/*
 * i2s_dma.c
 *
 *  Created on: May 8, 2017
 *      Author: evan
 */


#include <stdio.h>
#include "ezdsp5535_i2s.h"
#include "i2s_dma.h"

#include "hellocfg.h"

#include "soc.h"
#include "cslr.h"
#include "cslr_sysctrl.h"

#include "csl_gpio.h"
#include "csl_i2s.h"
#include "csl_intc.h"

#include "../global_vars.h"



#pragma DATA_ALIGN (left_ping, 4)
Int16 left_ping[I2S_DMA_BUFFER_SIZE];

#pragma DATA_ALIGN (left_pong, 4)
Int16 left_pong[I2S_DMA_BUFFER_SIZE];

#pragma DATA_ALIGN (right_ping, 4)
Int16 right_ping[I2S_DMA_BUFFER_SIZE];

#pragma DATA_ALIGN (right_pong, 4)
Int16 right_pong[I2S_DMA_BUFFER_SIZE];





CSL_DmaRegsOvly dma_reg;

void i2s_dma_init( void )
{

    CSL_Status          status;

    // Configure I2S
    CSL_I2sHandle i2sHandle;
    I2S_Config i2sConfig;

    i2sHandle = I2S_open(I2S_INSTANCE2, DMA_INTERRUPT, I2S_CHAN_STEREO);

    /* Set the value for the configure structure */
```

```
i2sConfig.dataFormat     = I2S_DATAFORMAT_LJUST;
i2sConfig.dataType       = I2S_STEREO_ENABLE;
i2sConfig.loopBackMode   = I2S_LOOPBACK_DISABLE;
i2sConfig.fsPol          = I2S_FSPOL_LOW;
i2sConfig.clkPol         = I2S_RISING_EDGE; //I2S_FALLING_EDGE;
i2sConfig.datadelay      = I2S_DATADELAY_ONEBIT;
i2sConfig.datapack       = I2S_DATAPACK_DISABLE;
i2sConfig.signext        = I2S_SIGNEXT_DISABLE;
i2sConfig.wordLen        = I2S_WORDLEN_16;
i2sConfig.i2sMode        = I2S_SLAVE;
i2sConfig.clkDiv         = I2S_CLKDIV2; // don't care for slave mode
i2sConfig.fsDiv          = I2S_FSDIV32; // don't care for slave mode
i2sConfig.FError         = I2S_FSERROR_DISABLE;
i2sConfig.OuError        = I2S_OUERROR_DISABLE;

status = I2S_setup(i2sHandle, &i2sConfig);
CSL_I2S2_REGS->I2SINTMASK &= 0xFF80;
I2S_transEnable(i2sHandle, TRUE);

// Init DMA
status = DMA_init();


/* Set the reset clock cycle */
CSL_FINS(CSL_SYSCTRL_REGS->PSRCR, SYS_PSRCR_COUNT,
CSL_DMA_RESET_CLOCK_CYCLE);
CSL_FINST(CSL_SYSCTRL_REGS->PRCR, SYS_PRCR_DMA_RST, RST);

/* Enable the corresponding DMA clock from PCGCR Registers */
CSL_FINST(CSL_SYSCTRL_REGS->PCGCR1, SYS_PCGCR1_DMA0CG, ACTIVE);
CSL_FINST(CSL_SYSCTRL_REGS->PCGCR2, SYS_PCGCR2_DMA1CG, ACTIVE);
CSL_FINST(CSL_SYSCTRL_REGS->PCGCR2, SYS_PCGCR2_DMA2CG, ACTIVE);
CSL_FINST(CSL_SYSCTRL_REGS->PCGCR2, SYS_PCGCR2_DMA3CG, ACTIVE);



/* enable ch4 DMA interrupts */
CSL_SYSCTRL_REGS->DMAIER = 0x0010;

/* Clear all DMA interrupt flags */
CSL_SYSCTRL_REGS->DMAIFR = 0xFFFF;
//IRQ_clear(DMA_EVENT);.


// Configure DMA

// Left DMA config
CSL_DMA_Handle      left_dmaHandle;
CSL_DMA_Config      left_dmaConfig;
CSL_DMA_ChannelObj  left_dmaChannelObj;

left_dmaConfig.pingPongMode = CSL_DMA_PING_PONG_ENABLE;
left_dmaConfig.autoMode     = CSL_DMA_AUTORELOAD_ENABLE;
left_dmaConfig.burstLen     = CSL_DMA_TXBURST_1WORD;
left_dmaConfig.trigger      = CSL_DMA_EVENT_TRIGGER;
left_dmaConfig.dmaEvt       = CSL_DMA_EVT_I2S2_TX;
```

```
    left_dmaConfig.dmaInt       = CSL_DMA_INTERRUPT_ENABLE;
    left_dmaConfig.chanDir      = CSL_DMA_WRITE;
    left_dmaConfig.trfType      = CSL_DMA_TRANSFER_IO_MEMORY;
    left_dmaConfig.dataLen      = I2S_DMA_BUFFER_SIZE * 4;
    left_dmaConfig.srcAddr      = (Uint32)left_ping;
    left_dmaConfig.destAddr     = (Uint32)0x2A08;

    left_dmaHandle = DMA_open(CSL_DMA_CHAN4, &left_dmaChannelObj, &status);
    DMA_config(left_dmaHandle, &left_dmaConfig);
    dma_reg = left_dmaHandle->dmaRegs;
    DMA_start(left_dmaHandle);

    // Left DMA config
    CSL_DMA_Handle      right_dmaHandle;
    CSL_DMA_Config      right_dmaConfig;
    CSL_DMA_ChannelObj  right_dmaChannelObj;

    right_dmaConfig.pingPongMode = CSL_DMA_PING_PONG_ENABLE;
    right_dmaConfig.autoMode     = CSL_DMA_AUTORELOAD_ENABLE;
    right_dmaConfig.burstLen     = CSL_DMA_TXBURST_1WORD;
    right_dmaConfig.trigger      = CSL_DMA_EVENT_TRIGGER;
    right_dmaConfig.dmaEvt       = CSL_DMA_EVT_I2S2_TX;
    right_dmaConfig.dmaInt       = CSL_DMA_INTERRUPT_DISABLE; // rely on
        iterrupt from left
    right_dmaConfig.chanDir      = CSL_DMA_WRITE;
    right_dmaConfig.trfType      = CSL_DMA_TRANSFER_IO_MEMORY;
    right_dmaConfig.dataLen      = I2S_DMA_BUFFER_SIZE * 4;
    right_dmaConfig.srcAddr      = (Uint32)right_ping;
    right_dmaConfig.destAddr     = (Uint32)0x2A0C;

    right_dmaHandle = DMA_open(CSL_DMA_CHAN5, &right_dmaChannelObj, &status);
    DMA_config(right_dmaHandle, &right_dmaConfig);
    dma_reg = right_dmaHandle->dmaRegs;
    DMA_start(right_dmaHandle);



    /* Clear DMA Interrupt Flags */
    IRQ_clear(DMA_EVENT);

    /* Enable DMA Interrupt */
    IRQ_enable(DMA_EVENT);





}

void dma_isr(void) {
    if (CSL_SYSCTRL_REGS->DMAIFR & 0x0010) { // ch4 interrupt, left channel
        SEM_post(&ping_pong_sem);
        CSL_SYSCTRL_REGS->DMAIFR |= 0x0010; // clear interrupt
    } else {
        while(1);
    }
```

```
}
```

```
/*
 * i2s_dma.h
 *
 *  Created on: May 8, 2017
 *      Author: evan
 */



#ifndef I2S_DMA_H_
#define I2S_DMA_H_

#include "csl_dma.h"




void i2s_dma_init( void );
void dma_isr(void);

#endif /* I2S_DMA_H_ */
```

```
/* Standard C includes */
#include <stdio.h>

/* DSP/BIOS headers */
#include <std.h>
#include "hellocfg.h"

/* ezDSP C5535 board specific headers */
#include "ezdsp5535.h"
#include "ezdsp5535_i2c.h"

/* C55xx chip support library headers */
#include "csl_pll.h"
#include "csl_general.h"
#include "csl_pllAux.h"


/* Danalog headers */
#include "pconfig/aic3204.h"
#include "pconfig/i2s_dma.h"
#include "pconfig/spi_config.h"
#include "io/midi_queue.h"

PLL_Obj pllObj;
PLL_Config pllCfg1;

PLL_Handle hPll;


PLL_Config pllCfg_12p288MHz = {0x8173, 0x8000, 0x0806, 0x0000};
PLL_Config pllCfg_40MHz     = {0x8988, 0x8000, 0x0806, 0x0201};
PLL_Config pllCfg_60MHz     = {0x8724, 0x8000, 0x0806, 0x0000};
PLL_Config pllCfg_75MHz     = {0x88ED, 0x8000, 0x0806, 0x0000};
PLL_Config pllCfg_100MHz    = {0x8BE8, 0x8000, 0x0806, 0x0000};
PLL_Config pllCfg_120MHz    = {0x8E4A, 0x8000, 0x0806, 0x0000};


//PLL_Config pllCfg_12p288MHz = {0x82ED, 0x8000, 0x0806, 0x0200};
//PLL_Config pllCfg_40MHz     = {0x8262, 0x8000, 0x0806, 0x0300};
//PLL_Config pllCfg_60MHz     = {0x81C8, 0xB000, 0x0806, 0x0000};
//PLL_Config pllCfg_75MHz     = {0x823B, 0x9000, 0x0806, 0x0000};
//PLL_Config pllCfg_100MHz    = {0x82FA, 0x8000, 0x0806, 0x0000};
//PLL_Config pllCfg_120MHz    = {0x8392, 0xA000, 0x0806, 0x0000};


PLL_Config *pConfigInfo;

#define CSL_TEST_FAILED         (1)
#define CSL_TEST_PASSED         (0)



Void main()
{

    printf("Initializing bsl\n");
```

```
EZDSP5535_init( );

/**** PLL init *****/
CSL_Status status;

status = PLL_init(&pllObj, CSL_PLL_INST_0);
if(CSL_SOK != status)
{
   printf("PLL init failed \n");
   return (status);
}

hPll = (PLL_Handle)(&pllObj);

PLL_reset(hPll);

status = PLL_bypass(hPll);
if(CSL_SOK != status)
{
   printf("PLL bypass failed:%d\n",CSL_ESYS_BADHANDLE);
   return(status);
}

/* Configure the PLL for 60MHz */
pConfigInfo = &pllCfg_120MHz;

status = PLL_config (hPll, pConfigInfo);
if(CSL_SOK != status)
{
   printf("PLL config failed\n");
   return(status);
}

status = PLL_getConfig(hPll, &pllCfg1);
if(status != CSL_SOK)
{
    printf("TEST FAILED: PLL get config... Failed.\n");
    printf ("Reason: PLL_getConfig failed. [status = 0x%x].\n", status);
    return(status);
}

printf("REGISTER --- CONFIG VALUES\n");

printf("%04x --- %04x\n",pllCfg1.PLLCNTL1,hPll->pllConfig->PLLCNTL1);
printf("%04x --- %04x Test Lock Mon will get set after PLL is up\n",
       pllCfg1.PLLCNTL2,hPll->pllConfig->PLLCNTL2);
printf("%04x --- %04x\n",pllCfg1.PLLINCNTL,hPll->pllConfig->PLLINCNTL);
printf("%04x --- %04x\n",pllCfg1.PLLOUTCNTL,hPll->pllConfig->PLLOUTCNTL);

EZDSP5535_waitusec(4000);

status = PLL_enable(hPll);
if(CSL_SOK != status)
{
   printf("PLL enable failed:%d\n",CSL_ESYS_BADHANDLE);
   return(status);
```

```
    }

    printf("Init i2c\n");
    EZDSP5535_I2C_init( );

    printf("Initializing aic3204\n");
    aic3204_init();

    printf("Initializing dma with i2s");
    i2s_dma_init();

    printf("Initializing spi");
    midi_buffer_init();
    spi_init();

    /* fall into DSP/BIOS idle loop */
}
```

```c
/*
 * midi_queue.c
 *
 *  Created on: May 28, 2017
 *      Author: evan
 */
#include "midi_queue.h"
#include "../global_vars.h"

Uint16 writeLoc, readLoc;
MidiPacket midi_buffer[MIDI_BUFFER_SIZE];

void midi_buffer_init()
{
    writeLoc = 0;
    readLoc  = 0;
}

void midi_buffer_write( MidiPacket p )
{
    midi_buffer[writeLoc] = p;
    writeLoc = (writeLoc + 1) % MIDI_BUFFER_SIZE;
}

MidiPacket midi_buffer_read( void )
{
    if (midi_buffer_size() == 0) {
        while (1); // tried to read from empty buffer
    }

    MidiPacket p = midi_buffer[readLoc];
    readLoc = (readLoc + 1) % MIDI_BUFFER_SIZE;
    return p;
}

Uint16 midi_buffer_size( void )
{
    return (writeLoc - readLoc + MIDI_BUFFER_SIZE) % MIDI_BUFFER_SIZE;
}
```

```
/*
 * midi_queue.h
 *
 *  Created on: May 28, 2017
 *      Author: evan
 */

#ifndef MIDI_QUEUE_H_
#define MIDI_QUEUE_H_

#include "ezdsp5535.h"
#include "midi.h"

#define MIDI_BUFFER_SIZE 16

extern Uint16 writeLoc, readLoc;

void midi_buffer_init();
void midi_buffer_write( MidiPacket p );
MidiPacket midi_buffer_read( void );
Uint16 midi_buffer_size( void );

#endif /* MIDI_QUEUE_H_ */
```

```
/*
 * midi.c
 *
 *  Created on: May 29, 2017
 *      Author: evan
 */
#include "midi.h"

MidiCommand midi_packet_type(MidiPacket p)
{
    if (p.midi_cmd == 0x90) {
        return MIDI_NOTE_ON;
    }
    else if (p.midi_cmd == 0x80) {
        return MIDI_NOTE_OFF;
    }
    else {
        return MIDI_UNKNOWN;
    }
}

Int16 midi_freq[88] = {
        28,29,31,33,35,37,39,41,44,46,49,52,55,58,62,65,69,73,78,82,
        87,93,98,104,110,117,124,131,139,147,156,165,175,185,196,208,
        220,233,247,262,277,294,311,330,349,370,392,415,440,466,494,523,
        554,587,622,659,699,740,784,831,880,932,988,1047,1109,1175,1245,
        1319,1397,1480,1568,1661,1760,1865,1976,2093,2218,2349,2489,2637,
        2794,2960,3136,3322,3520,3729,3951,4186
};

Int16 convert_to_freq(Uint16 midi_note) { // NOTE the signed types
    if (midi_note < 21 || midi_note > 108) {
        return 0;
    } else {
        return midi_freq[midi_note - 21];
    }
}
```

```c
/*
 * midi_utils.h
 *
 *  Created on: May 29, 2017
 *      Author: evan
 */

#ifndef MIDI_UTILS_H_
#define MIDI_UTILS_H_

#include <std.h>

enum midi_cmd_type {
    MIDI_NOTE_ON,
    MIDI_NOTE_OFF,
    MIDI_UNKNOWN
};

typedef enum midi_cmd_type MidiCommand;


struct midi_packet_struct {
    Uint16 midi_cmd;
    Uint16 note_id;
    Uint16 velocity;
};
typedef struct midi_packet_struct MidiPacket;

MidiCommand midi_packet_type(MidiPacket);


Int16 convert_to_freq(Uint16 midi_note);


#endif /* MIDI_UTILS_H_ */
```

```
/*
 * ringbuf.c
 *
 *  Created on: May 31, 2017
 *      Author: evan
 */
#include "ringbuf.h"

Int16 note_buf_head = 0;
FMNote note_buf[NOTE_BUF_LEN];

void add_note(MidiPacket *p, Int16 mod_ratio) {
    FMNote *n = &note_buf[note_buf_head];
    *n = midi_to_fm_note(p);

    sin_compute_params(&n->mod_sin, n->pitch * mod_ratio);
    sin_compute_params(&n->car_sin,  n->pitch);

    note_buf_head = (note_buf_head + 1) % NOTE_BUF_LEN;
}

void release_note(MidiPacket *p) {
    Int16 i;
    for (i = 0; i < NOTE_BUF_LEN; i++) {
        FMNote *n = &note_buf[i];
        if (n->pitch == convert_to_freq(p->note_id)) {
            n->car_env.env_state = ENV_RELEASE;
            n->mod_env.env_state = ENV_RELEASE;
        }
    }
}
```

```
/*
 * ringbuf.h
 *
 *  Created on: May 31, 2017
 *      Author: evan
 */

#ifndef RINGBUF_H_
#define RINGBUF_H_

#include "fm.h"
#include "../io/midi.h"

#define NOTE_BUF_LEN 2

extern FMNote note_buf[NOTE_BUF_LEN];

void add_note(MidiPacket *p, Int16 mod_ratio);
void release_note(MidiPacket *p);


#endif /* RINGBUF_H_ */
```

```c
#include "singen.h"

/* Danalog headers */
#include "sintable.h"


void sin_compute_params(SinState *state, Int32 frequency) {
    state->frequency = frequency;
    state->step_delta = (frequency * 8192) / SAMPLE_RATE;
    state->position = 0;
}

Int16 _decompress_sin(Int16 index) {
    if (index > 8192|| index < 0) {
//      printf("ERROR: Index out of range. index = %d", index);
        while(1);
    }

//  if (index < SINTABLE_LENGTH) {
//      return sintable[index];
//  }
//  else if (index < SINTABLE_LENGTH * 2) {
//      index = SINTABLE_LENGTH - (index - SINTABLE_LENGTH) - 1;
//      return sintable[index];
//  }
//  else if (index < SINTABLE_LENGTH * 3) {
//      index = index - 4096;
//      return _sneg(sintable[index]);
//  }
//  else {
//      index = SINTABLE_LENGTH - (index - SINTABLE_LENGTH) - 1;
//      return _sneg(sintable[index]);
//  }

    Int16 multiplier = 1;
    if (index >= 4096) {
        multiplier = -1;
        index = index - 4096;
    }

    if (index >= SINTABLE_LENGTH) { // reflect 3999->2000 to 1999->0
        index = SINTABLE_LENGTH - (index - SINTABLE_LENGTH) - 1;
    }

    return sintable[index] * multiplier;


}

Int16 sin_gen(SinState *state, Int16 mod) {
    Int16 position_mod;
    state->position = (state->step_delta + state->position) & 8191;
    position_mod = (state->position + mod) & 8191;
    return _decompress_sin(position_mod);
}
```

```c
/*
 * singen.h
 *
 *  Created on: May 2, 2017
 *      Author: evan
 */

#ifndef SINGEN_H_
#define SINGEN_H_

#include "ezdsp5535.h"

#define SAMPLE_RATE 96000

struct sin_state {
    Int32 position;
    Int32 frequency;
    Int32 step_delta;

};
typedef struct sin_state SinState;



Int16 _decompress_sin(Int16 index);
Int16 sin_gen(SinState *state, Int16 mod);

void sin_compute_params(SinState *state, Int32 frequency);


#endif /* SINGEN_H_ */
```

```
/*
 * sintable.c
 *
 *  Created on: May 2, 2017
 *      Author: evan
 */

#include "sintable.h"



const Int16 sintable[SINTABLE_LENGTH] = {

        0, 25, 50, 75, 101, 126, 151, 176,
        201, 226, 251, 277, 302, 327, 352, 377,
        402, 427, 453, 478, 503, 528, 553, 578,
        603, 629, 654, 679, 704, 729, 754, 779,
        805, 830, 855, 880, 905, 930, 955, 980,
        1006, 1031, 1056, 1081, 1106, 1131, 1156, 1182,
        1207, 1232, 1257, 1282, 1307, 1332, 1357, 1383,
        1408, 1433, 1458, 1483, 1508, 1533, 1558, 1583,
        1609, 1634, 1659, 1684, 1709, 1734, 1759, 1784,
        1809, 1835, 1860, 1885, 1910, 1935, 1960, 1985,
        2010, 2035, 2060, 2086, 2111, 2136, 2161, 2186,
        2211, 2236, 2261, 2286, 2311, 2336, 2362, 2387,
        2412, 2437, 2462, 2487, 2512, 2537, 2562, 2587,
        2612, 2637, 2662, 2687, 2712, 2738, 2763, 2788,
        2813, 2838, 2863, 2888, 2913, 2938, 2963, 2988,
        3013, 3038, 3063, 3088, 3113, 3138, 3163, 3188,
        3213, 3238, 3263, 3288, 3313, 3338, 3363, 3388,
        3413, 3438, 3463, 3488, 3513, 3538, 3563, 3588,
        3613, 3638, 3663, 3688, 3713, 3738, 3763, 3788,
        3813, 3838, 3863, 3888, 3913, 3938, 3963, 3988,
        4013, 4038, 4063, 4088, 4113, 4138, 4163, 4188,
        4213, 4237, 4262, 4287, 4312, 4337, 4362, 4387,
        4412, 4437, 4462, 4487, 4512, 4536, 4561, 4586,
        4611, 4636, 4661, 4686, 4711, 4736, 4760, 4785,
        4810, 4835, 4860, 4885, 4910, 4935, 4959, 4984,
        5009, 5034, 5059, 5084, 5109, 5133, 5158, 5183,
        5208, 5233, 5257, 5282, 5307, 5332, 5357, 5382,
        5406, 5431, 5456, 5481, 5505, 5530, 5555, 5580,
        5605, 5629, 5654, 5679, 5704, 5728, 5753, 5778,
        5803, 5827, 5852, 5877, 5902, 5926, 5951, 5976,
        6001, 6025, 6050, 6075, 6099, 6124, 6149, 6174,
        6198, 6223, 6248, 6272, 6297, 6322, 6346, 6371,
        6396, 6420, 6445, 6470, 6494, 6519, 6544, 6568,
        6593, 6617, 6642, 6667, 6691, 6716, 6740, 6765,
        6790, 6814, 6839, 6863, 6888, 6913, 6937, 6962,
        6986, 7011, 7035, 7060, 7085, 7109, 7134, 7158,
        7183, 7207, 7232, 7256, 7281, 7305, 7330, 7354,
        7379, 7403, 7428, 7452, 7477, 7501, 7526, 7550,
        7575, 7599, 7624, 7648, 7673, 7697, 7721, 7746,
        7770, 7795, 7819, 7844, 7868, 7892, 7917, 7941,
        7966, 7990, 8014, 8039, 8063, 8087, 8112, 8136,
        8161, 8185, 8209, 8234, 8258, 8282, 8307, 8331,
        8355, 8379, 8404, 8428, 8452, 8477, 8501, 8525,
        8550, 8574, 8598, 8622, 8647, 8671, 8695, 8719,
        8744, 8768, 8792, 8816, 8840, 8865, 8889, 8913,
        8937, 8961, 8986, 9010, 9034, 9058, 9082, 9106,
        9131, 9155, 9179, 9203, 9227, 9251, 9275, 9299,
        9324, 9348, 9372, 9396, 9420, 9444, 9468, 9492,
        9516, 9540, 9564, 9588, 9612, 9637, 9661, 9685,
        9709, 9733, 9757, 9781, 9805, 9829, 9853, 9877,
        9901, 9924, 9948, 9972, 9996, 10020, 10044, 10068,
        10092, 10116, 10140, 10164, 10188, 10212, 10235, 10259,
        10283, 10307, 10331, 10355, 10379, 10403, 10426, 10450,
        10474, 10498, 10522, 10546, 10569, 10593, 10617, 10641,
        10664, 10688, 10712, 10736, 10760, 10783, 10807, 10831,
        10854, 10878, 10902, 10926, 10949, 10973, 10997, 11020,
        11044, 11068, 11091, 11115, 11139, 11162, 11186, 11210,
        11233, 11257, 11280, 11304, 11328, 11351, 11375, 11398,
        11422, 11446, 11469, 11493, 11516, 11540, 11563, 11587,
        11610, 11634, 11657, 11681, 11704, 11728, 11751, 11775,
        11798, 11822, 11845, 11869, 11892, 11915, 11939, 11962,
        11986, 12009, 12032, 12056, 12079, 12103, 12126, 12149,
        12173, 12196, 12219, 12243, 12266, 12289, 12313, 12336,
        12359, 12382, 12406, 12429, 12452, 12475, 12499, 12522,
        12545, 12568, 12592, 12615, 12638, 12661, 12684, 12708,
        12731, 12754, 12777, 12800, 12823, 12847, 12870, 12893,
        12916, 12939, 12962, 12985, 13008, 13031, 13054, 13077,
        13101, 13124, 13147, 13170, 13193, 13216, 13239, 13262,
        13285, 13308, 13331, 13354, 13377, 13399, 13422, 13445,
        13468, 13491, 13514, 13537, 13560, 13583, 13606, 13629,
        13651, 13674, 13697, 13720, 13743, 13766, 13788, 13811,
        13834, 13857, 13880, 13902, 13925, 13948, 13971, 13993,
        14016, 14039, 14062, 14084, 14107, 14130, 14152, 14175,
        14198, 14220, 14243, 14266, 14288, 14311, 14333, 14356,
        14379, 14401, 14424, 14446, 14469, 14492, 14514, 14537,
        14559, 14582, 14604, 14627, 14649, 14672, 14694, 14717,
        14739, 14762, 14784, 14806, 14829, 14851, 14874, 14896,
        14918, 14941, 14963, 14986, 15008, 15030, 15053, 15075,
        15097, 15120, 15142, 15164, 15186, 15209, 15231, 15253,
        15276, 15298, 15320, 15342, 15364, 15387, 15409, 15431,
        15453, 15475, 15498, 15520, 15542, 15564, 15586, 15608,
        15630, 15652, 15674, 15697, 15719, 15741, 15763, 15785,
        15807, 15829, 15851, 15873, 15895, 15917, 15939, 15961,
        15983, 16005, 16027, 16048, 16070, 16092, 16114, 16136,
        16158, 16180, 16202, 16224, 16245, 16267, 16289, 16311,
        16333, 16354, 16376, 16398, 16420, 16442, 16463, 16485,
        16507, 16528, 16550, 16572, 16594, 16615, 16637, 16659,
        16680, 16702, 16723, 16745, 16767, 16788, 16810, 16831,
        16853, 16875, 16896, 16918, 16939, 16961, 16982, 17004,
        17025, 17047, 17068, 17090, 17111, 17133, 17154, 17175,
        17197, 17218, 17240, 17261, 17282, 17304, 17325, 17346,
        17368, 17389, 17410, 17432, 17453, 17474, 17495, 17517,
        17538, 17559, 17580, 17602, 17623, 17644, 17665, 17686,
        17707, 17729, 17750, 17771, 17792, 17813, 17834, 17855,
        17876, 17897, 17919, 17940, 17961, 17982, 18003, 18024,
        18045, 18066, 18087, 18108, 18129, 18149, 18170, 18191,
        18212, 18233, 18254, 18275, 18296, 18317, 18337, 18358,
        18379, 18400, 18421, 18441, 18462, 18483, 18504, 18525,
        18545, 18566, 18587, 18607, 18628, 18649, 18669, 18690,
        18711, 18731, 18752, 18773, 18793, 18814, 18834, 18855,
        18876, 18896, 18917, 18937, 18958, 18978, 18999, 19019,
        19040, 19060, 19081, 19101, 19121, 19142, 19162, 19183,
        19203, 19223, 19244, 19264, 19284, 19305, 19325, 19345,
        19366, 19386, 19406, 19426, 19447, 19467, 19487, 19507,
        19527, 19548, 19568, 19588, 19608, 19628, 19648, 19669,
        19689, 19709, 19729, 19749, 19769, 19789, 19809, 19829,
        19849, 19869, 19889, 19909, 19929, 19949, 19969, 19989,
        20009, 20029, 20049, 20068, 20088, 20108, 20128, 20148,
        20168, 20187, 20207, 20227, 20247, 20267, 20286, 20306,
        20326, 20346, 20365, 20385, 20405, 20424, 20444, 20464,
        20483, 20503, 20522, 20542, 20562, 20581, 20601, 20620,
        20640, 20659, 20679, 20698, 20718, 20737, 20757, 20776,
        20796, 20815, 20835, 20854, 20873, 20893, 20912, 20931,
        20951, 20970, 20989, 21009, 21028, 21047, 21067, 21086,
        21105, 21124, 21143, 21163, 21182, 21201, 21220, 21239,
        21258, 21278, 21297, 21316, 21335, 21354, 21373, 21392,
        21411, 21430, 21449, 21468, 21487, 21506, 21525, 21544,
        21563, 21582, 21601, 21620, 21639, 21658, 21676, 21695,
        21714, 21733, 21752, 21770, 21789, 21808, 21827, 21846,
        21864, 21883, 21902, 21920, 21939, 21958, 21976, 21995,
        22014, 22032, 22051, 22070, 22088, 22107, 22125, 22144,
        22162, 22181, 22199, 22218, 22236, 22255, 22273, 22292,
        22310, 22328, 22347, 22365, 22384, 22402, 22420, 22439,
        22457, 22475, 22494, 22512, 22530, 22548, 22567, 22585,
        22603, 22621, 22639, 22658, 22676, 22694, 22712, 22730,
        22748, 22766, 22784, 22802, 22820, 22839, 22857, 22875,
        22893, 22911, 22929, 22946, 22964, 22982, 23000, 23018,
        23036, 23054, 23072, 23090, 23107, 23125, 23143, 23161,
        23179, 23196, 23214, 23232, 23250, 23267, 23285, 23303,
        23320, 23338, 23356, 23373, 23391, 23409, 23426, 23444,
        23461, 23479, 23496, 23514, 23531, 23549, 23566, 23584,
        23601, 23619, 23636, 23654, 23671, 23688, 23706, 23723,
        23740, 23758, 23775, 23792, 23810, 23827, 23844, 23861,
        23879, 23896, 23913, 23930, 23947, 23964, 23982, 23999,
        24016, 24033, 24050, 24067, 24084, 24101, 24118, 24135,
        24152, 24169, 24186, 24203, 24220, 24237, 24254, 24271,
        24288, 24305, 24321, 24338, 24355, 24372, 24389, 24406,
        24422, 24439, 24456, 24473, 24489, 24506, 24523, 24539,
        24556, 24573, 24589, 24606, 24622, 24639, 24656, 24672,
        24689, 24705, 24722, 24738, 24755, 24771, 24788, 24804,
        24820, 24837, 24853, 24870, 24886, 24902, 24919, 24935,
        24951, 24968, 24984, 25000, 25016, 25033, 25049, 25065,
        25081, 25097, 25114, 25130, 25146, 25162, 25178, 25194,
        25210, 25226, 25242, 25258, 25274, 25290, 25306, 25322,
        25338, 25354, 25370, 25386, 25402, 25418, 25434, 25449,
        25465, 25481, 25497, 25513, 25528, 25544, 25560, 25576,
        25591, 25607, 25623, 25638, 25654, 25670, 25685, 25701,
        25717, 25732, 25748, 25763, 25779, 25794, 25810, 25825,
        25841, 25856, 25872, 25887, 25902, 25918, 25933, 25949,
        25964, 25979, 25995, 26010, 26025, 26040, 26056, 26071,
        26086, 26101, 26117, 26132, 26147, 26162, 26177, 26192,
        26207, 26222, 26237, 26253, 26268, 26283, 26298, 26313,
        26328, 26343, 26357, 26372, 26387, 26402, 26417, 26432,
        26447, 26462, 26477, 26491, 26506, 26521, 26536, 26550,
        26565, 26580, 26595, 26609, 26624, 26639, 26653, 26668,
        26682, 26697, 26712, 26726, 26741, 26755, 26770, 26784,
        26799, 26813, 26828, 26842, 26856, 26871, 26885, 26900,
        26914, 26928, 26943, 26957, 26971, 26985, 27000, 27014,
        27028, 27042, 27056, 27071, 27085, 27099, 27113, 27127,
        27141, 27155, 27169, 27183, 27198, 27212, 27226, 27240,
        27253, 27267, 27281, 27295, 27309, 27323, 27337, 27351,
        27365, 27378, 27392, 27406, 27420, 27434, 27447, 27461,
        27475, 27488, 27502, 27516, 27529, 27543, 27557, 27570,
        27584, 27597, 27611, 27625, 27638, 27652, 27665, 27678,
        27692, 27705, 27719, 27732, 27746, 27759, 27772, 27786,
        27799, 27812, 27826, 27839, 27852, 27865, 27879, 27892,
        27905, 27918, 27931, 27944, 27957, 27971, 27984, 27997,
        28010, 28023, 28036, 28049, 28062, 28075, 28088, 28101,
        28114, 28127, 28139, 28152, 28165, 28178, 28191, 28204,
        28216, 28229, 28242, 28255, 28267, 28280, 28293, 28306,
        28318, 28331, 28343, 28356, 28369, 28381, 28394, 28406,
        28419, 28431, 28444, 28456, 28469, 28481, 28494, 28506,
        28518, 28531, 28543, 28556, 28568, 28580, 28592, 28605,
        28617, 28629, 28641, 28654, 28666, 28678, 28690, 28702,
        28714, 28727, 28739, 28751, 28763, 28775, 28787, 28799,
        28811, 28823, 28835, 28847, 28859, 28870, 28882, 28894,
        28906, 28918, 28930, 28942, 28953, 28965, 28977, 28989,
        29000, 29012, 29024, 29035, 29047, 29059, 29070, 29082,
        29093, 29105, 29116, 29128, 29139, 29151, 29162, 29174,
        29185, 29197, 29208, 29220, 29231, 29242, 29254, 29265,
        29276, 29288, 29299, 29310, 29321, 29332, 29344, 29355,
        29366, 29377, 29388, 29399, 29410, 29422, 29433, 29444,
        29455, 29466, 29477, 29488, 29499, 29510, 29520, 29531,
        29542, 29553, 29564, 29575, 29586, 29596, 29607, 29618,
        29629, 29639, 29650, 29661, 29672, 29682, 29693, 29703,
        29714, 29725, 29735, 29746, 29756, 29767, 29777, 29788,
        29798, 29809, 29819, 29830, 29840, 29850, 29861, 29871,
        29881, 29892, 29902, 29912, 29923, 29933, 29943, 29953,
        29963, 29974, 29984, 29994, 30004, 30014, 30024, 30034,
        30044, 30054, 30064, 30074, 30084, 30094, 30104, 30114,
        30124, 30134, 30144, 30154, 30163, 30173, 30183, 30193,
        30203, 30212, 30222, 30232, 30241, 30251, 30261, 30270,
        30280, 30290, 30299, 30309, 30318, 30328, 30337, 30347,
        30356, 30366, 30375, 30385, 30394, 30403, 30413, 30422,
        30431, 30441, 30450, 30459, 30469, 30478, 30487, 30496,
        30505, 30515, 30524, 30533, 30542, 30551, 30560, 30569,
        30578, 30587, 30596, 30605, 30614, 30623, 30632, 30641,
        30650, 30659, 30668, 30677, 30685, 30694, 30703, 30712,
        30721, 30729, 30738, 30747, 30755, 30764, 30773, 30781,
        30790, 30799, 30807, 30816, 30824, 30833, 30841, 30850,
        30858, 30867, 30875, 30883, 30892, 30900, 30909, 30917,
        30925, 30934, 30942, 30950, 30958, 30967, 30975, 30983,
        30991, 30999, 31007, 31016, 31024, 31032, 31040, 31048,
        31056, 31064, 31072, 31080, 31088, 31096, 31104, 31112,
        31119, 31127, 31135, 31143, 31151, 31159, 31166, 31174,
        31182, 31190, 31197, 31205, 31213, 31220, 31228, 31235,
        31243, 31251, 31258, 31266, 31273, 31281, 31288, 31296,
        31303, 31311, 31318, 31325, 31333, 31340, 31347, 31355,
        31362, 31369, 31377, 31384, 31391, 31398, 31405, 31413,
        31420, 31427, 31434, 31441, 31448, 31455, 31462, 31469,
        31476, 31483, 31490, 31497, 31504, 31511, 31518, 31525,
        31531, 31538, 31545, 31552, 31559, 31565, 31572, 31579,
        31586, 31592, 31599, 31606, 31612, 31619, 31625, 31632,
        31639, 31645, 31652, 31658, 31665, 31671, 31677, 31684,
        31690, 31697, 31703, 31709, 31716, 31722, 31728, 31735,
        31741, 31747, 31753, 31759, 31766, 31772, 31778, 31784,
        31790, 31796, 31802, 31808, 31814, 31820, 31826, 31832,
        31838, 31844, 31850, 31856, 31862, 31868, 31874, 31879,
        31885, 31891, 31897, 31903, 31908, 31914, 31920, 31925,
        31931, 31937, 31942, 31948, 31953, 31959, 31965, 31970,
        31976, 31981, 31987, 31992, 31997, 32003, 32008, 32014,
        32019, 32024, 32030, 32035, 32040, 32045, 32051, 32056,
        32061, 32066, 32071, 32077, 32082, 32087, 32092, 32097,
        32102, 32107, 32112, 32117, 32122, 32127, 32132, 32137,
        32142, 32147, 32151, 32156, 32161, 32166, 32171, 32175,
        32180, 32185, 32190, 32194, 32199, 32204, 32208, 32213,
        32217, 32222, 32227, 32231, 32236, 32240, 32245, 32249,
        32254, 32258, 32262, 32267, 32271, 32275, 32280, 32284,
        32288, 32293, 32297, 32301, 32305, 32310, 32314, 32318,
        32322, 32326, 32330, 32334, 32338, 32342, 32346, 32350,
        32354, 32358, 32362, 32366, 32370, 32374, 32378, 32382,
        32386, 32390, 32393, 32397, 32401, 32405, 32408, 32412,
        32416, 32419, 32423, 32427, 32430, 32434, 32437, 32441,
        32444, 32448, 32451, 32455, 32458, 32462, 32465, 32469,
        32472, 32475, 32479, 32482, 32485, 32489, 32492, 32495,
        32498, 32502, 32505, 32508, 32511, 32514, 32517, 32520,
        32523, 32526, 32529, 32533, 32535, 32538, 32541, 32544,
        32547, 32550, 32553, 32556, 32559, 32562, 32564, 32567,
        32570, 32573, 32575, 32578, 32581, 32583, 32586, 32589,
        32591, 32594, 32596, 32599, 32602, 32604, 32607, 32609,
        32612, 32614, 32616, 32619, 32621, 32624, 32626, 32628,
        32630, 32633, 32635, 32637, 32639, 32642, 32644, 32646,
        32648, 32650, 32652, 32655, 32657, 32659, 32661, 32663,
        32665, 32667, 32669, 32671, 32672, 32674, 32676, 32678,
        32680, 32682, 32684, 32685, 32687, 32689, 32691, 32692,
        32694, 32696, 32697, 32699, 32701, 32702, 32704, 32705,
        32707, 32708, 32710, 32711, 32713, 32714, 32716, 32717,
        32718, 32720, 32721, 32722, 32724, 32725, 32726, 32727,
        32729, 32730, 32731, 32732, 32733, 32735, 32736, 32737,
        32738, 32739, 32740, 32741, 32742, 32743, 32744, 32745,
        32746, 32747, 32747, 32748, 32749, 32750, 32751, 32752,
        32752, 32753, 32754, 32754, 32755, 32756, 32756, 32757,
        32758, 32758, 32759, 32759, 32760, 32760, 32761, 32761,
        32762, 32762, 32763, 32763, 32764, 32764, 32764, 32765,
        32765, 32765, 32765, 32766, 32766, 32766, 32766, 32766,
        32767, 32767, 32767, 32767, 32767, 32767, 32767, 32767,

};
```

```c
/*
 * sintable.h
 *
 *  Created on: May 1, 2017
 *      Author: evan
 */

#ifndef SINTABLE_H_
#define SINTABLE_H_

#include "ezdsp5535.h"

#define SINTABLE_LENGTH 2048

extern const Int16 sintable[SINTABLE_LENGTH];


#endif /* SINTABLE_H_ */
```

```
/*
 * spi_config.c
 *
 *  Created on: May 25, 2017
 *      Author: evan
 */

#include "spi_config.h"
#include "csl_gpio.h"

#include "../global_vars.h"

#define SPI_RW_WAIT 100

SPI_Config      spi_hwConfig;

// SPI data containers
Uint16 encoders[19];
Uint16 pots[8];
Uint16 switches;
Uint16 midi[3];

void spi_init( void ) {
    /*********************************
     *    Configure SPI peripheral   *
     *********************************/

    // Copy and paste from CSL, but changed
    // pin multiplexing mode to PPMODE_MODE6
    volatile Uint16 delay;
    ioport volatile CSL_SysRegs *sysRegs;

    sysRegs = (CSL_SysRegs *)CSL_SYSCTRL_REGS;
//  CSL_FINS(sysRegs->PCGCR1, SYS_PCGCR1_SPICG, CSL_SYS_PCGCR1_SPICG_ACTIVE);
//
//  /* Value of 'Reset Counter' */
//  CSL_FINS(sysRegs->PSRCR, SYS_PSRCR_COUNT, 0x20);
//
//  CSL_FINS(sysRegs->PRCR, SYS_PRCR_PG4_RST, CSL_SYS_PRCR_PG4_RST_RST);

    for(delay = 0; delay < 100; delay++);


    CSL_FINS(sysRegs->EBSR, SYS_EBSR_PPMODE, CSL_SYS_EBSR_PPMODE_MODE1);
    // End of CSL copy paste

    hSpi = &SPI_Instance;
    hSpi->mode = SPI_CS_NUM_1;
    hSpi->opMode = SPI_POLLING_MODE;

    spi_hwConfig.spiClkDiv  = 25;
    spi_hwConfig.wLen       = SPI_WORD_LENGTH_8;
    spi_hwConfig.frLen      = 1;
    spi_hwConfig.wcEnable   = SPI_WORD_IRQ_DISABLE;
    spi_hwConfig.fcEnable   = SPI_FRAME_IRQ_DISABLE;
    spi_hwConfig.csNum      = SPI_CS_NUM_1;
    spi_hwConfig.dataDelay  = SPI_DATA_DLY_0;
    spi_hwConfig.csPol      = SPI_CSP_ACTIVE_LOW;
    spi_hwConfig.clkPol     = SPI_CLKP_LOW_AT_IDLE;
    spi_hwConfig.clkPh      = SPI_CLK_PH_RISE_EDGE;

    SPI_config(hSpi, &spi_hwConfig);

    /*********************************************
    *      Enable level shifter (set OE high)    *
    *      GPIO pin 13                           *
    **********************************************/
    CSL_GpioObj    gpioObj;
    CSL_GpioObj    *hGpio;
    CSL_Status     status;

    hGpio = GPIO_open(&gpioObj, &status);

    CSL_GpioPinConfig    config;
    config.pinNum    = CSL_GPIO_PIN13;
    config.direction = CSL_GPIO_DIR_OUTPUT;
    config.trigger   = CSL_GPIO_TRIG_CLEAR_EDGE;

    GPIO_configBit(hGpio, &config);
    GPIO_write(hGpio, CSL_GPIO_PIN13, 1);
}

void spi_write( Uint16 *write_buf, Uint16 buf_len ) {
    // set frame length
    CSL_FINS(CSL_SPI_REGS->SPICMD1, SPI_SPICMD1_FLEN, buf_len-1);
    //SPI_write(hSpi, write_buf, buf_len);

    // swipped from cls_spi.c SPI_read
    volatile Uint16     bufIndex;
    Uint16  spiStatusReg;
    volatile Uint16     spiBusyStatus;
    volatile Uint16     spiWcStaus;
    volatile Uint16     delay;

    /* Write Word length set by the user */
    bufIndex = 0;
    while(bufIndex < buf_len)
    {
        CSL_SPI_REGS->SPIDR2 = (Uint16)(write_buf[bufIndex] << 0x08);
        CSL_SPI_REGS->SPIDR1 = 0x0000;
        bufIndex++;

        /* Set command for Writting to registers */
        CSL_FINS(CSL_SPI_REGS->SPICMD2, SPI_SPICMD2_CMD, SPI_WRITE_CMD);

        for(delay = 0; delay < SPI_RW_WAIT; delay++);

        do
        {
            spiStatusReg = CSL_SPI_REGS->SPISTAT1;
            spiBusyStatus = (spiStatusReg & CSL_SPI_SPISTAT1_BSY_MASK);
            spiWcStaus = (spiStatusReg & CSL_SPI_SPISTAT1_CC_MASK);
        }while((spiBusyStatus == CSL_SPI_SPISTAT1_BSY_BUSY) &&
                (spiWcStaus != CSL_SPI_SPISTAT1_CC_MASK));
    }
}

void spi_read( Uint16 *read_buf, Uint16 buf_len ) {
    // set frame length
    CSL_FINS(CSL_SPI_REGS->SPICMD1, SPI_SPICMD1_FLEN, buf_len-1);

    // swipped from cls_spi.c SPI_read
    volatile Uint16     bufIndex;
    Int16   spiStatusReg;
    volatile Int16  spiBusyStatus;
    volatile Int16  spiWcStaus;
    volatile Uint16 delay;

    bufIndex = 0;

    /* Read Word length set by the user */
    while(bufIndex < buf_len)
    {
        // Clear spi data regs
        CSL_SPI_REGS->SPIDR1 = 0;
        CSL_SPI_REGS->SPIDR2 = 0;

        /* Set command for reading buffer */
        CSL_FINS(CSL_SPI_REGS->SPICMD2, SPI_SPICMD2_CMD,
                                        CSL_SPI_SPICMD2_CMD_READ);

        for(delay = 0; delay < SPI_RW_WAIT; delay++);

        do
        {
            spiStatusReg = CSL_SPI_REGS->SPISTAT1;
            spiBusyStatus = (spiStatusReg & CSL_SPI_SPISTAT1_BSY_MASK);
            spiWcStaus = (spiStatusReg & CSL_SPI_SPISTAT1_CC_MASK);
        } while ((spiBusyStatus == CSL_SPI_SPISTAT1_BSY_BUSY) &&
                (spiWcStaus != CSL_SPI_SPISTAT1_CC_MASK));

        // read data
        read_buf[bufIndex] = (CSL_SPI_REGS->SPIDR1 & 0xFF);
        bufIndex++;
    }
}
```

```c
/*
 * spi_config.h
 *
 *  Created on: May 25, 2017
 *      Author: evan
 */

#ifndef SPI_CONFIG_H_
#define SPI_CONFIG_H_

#include "csl_spi.h"

#define SPI_MIDI_CMD 0x01
#define SPI_ENC_CMD  0x02
#define SPI_POT_CMD  0x03
#define SPI_SWT_CMD  0x04


extern CSL_SpiHandle    hSpi;
extern SPI_Config       spi_hwConfig;

void spi_init( void );
void spi_write( Uint16 *write_buf, Uint16 buf_len );
void spi_read( Uint16 *read_buf, Uint16 buf_len );


#endif /* SPI_CONFIG_H_ */
```

ATMega directory structure

```
├── SPI.c
├── SPI.h
├── SynthMux.c
├── SynthMux.h
├── encoder.c
├── encoder.h
├── k25m.c
├── k25m.h
├── lcd.c
├── lcd.h
├── main.c
├── main.hex
├── makefile
├── midi.c
├── midi.h
├── potadc.c
├── potadc.h
├── serial_usb.c
└── serial_usb.h
```

```
#include "encoder.h"

#include <avr/io.h>

void check_encoder(Encoder* enc) {
    if ((*enc->a_port & enc->a_pin) == enc->a_pin  && enc->timer != 0 ) {
      enc->timer--;
    }

    if ((*enc->a_port & enc->a_pin) == 0 && enc->timer == 0) {
      if ((*enc->b_port & enc->b_pin) == 0) {
        enc->count--;
      } else {
        enc->count++;
      }
      enc->timer = 4;
    }
}

void encoder_init(void) {
  DDRE &= ~((1<<ENC_SYNTH_PRESET_A)|(1<<ENC_SYNTH_PRESET_B));
  DDRL &= ~(0xFF); // entire port L
  DDRD &= ~((1<<ENC_FX2_SELECT_B));
  DDRG &= ~((1<<ENC_FX2_PARAM_3_B)|(1<<ENC_FX2_SELECT_A)|(1<<ENC_FX2_PARAM_3_A)
      );
  DDRC &= ~(0xFF); // entire port C
  DDRA &= ~(0xFF); // entire port A
  DDRK &= ~(0xFF); // entire port k

  // Set pull ups
  PORTE |= (1<<ENC_SYNTH_PRESET_A)|(1<<ENC_SYNTH_PRESET_B);
  PORTL |= 0xFF; // entire port L
  PORTD |= (1<<ENC_FX2_SELECT_B);
  PORTG |= (1<<ENC_FX2_PARAM_3_B)|(1<<ENC_FX2_SELECT_A)|(1<<ENC_FX2_PARAM_3_A);
  PORTC |= 0xFF; // entire port C
  PORTA |= 0xFF; // entire port A
  PORTK |= 0xFF; // entire port k

  encoders[0].a_pin  = 1<<ENC_FX1_SELECT_A;
  encoders[0].b_pin  = 1<<ENC_FX1_SELECT_B;
  encoders[0].a_port = &PINC;
  encoders[0].b_port = &PINC;

  encoders[1].a_pin  = 1<<ENC_FX1_PARAM_0_A;
  encoders[1].b_pin  = 1<<ENC_FX1_PARAM_0_B;
  encoders[1].a_port = &PINC;
  encoders[1].b_port = &PINC;

  encoders[2].a_pin  = 1<<ENC_FX1_PARAM_1_A;
  encoders[2].b_pin  = 1<<ENC_FX1_PARAM_1_B;
  encoders[2].a_port = &PINA;
  encoders[2].b_port = &PINA;

  encoders[3].a_pin  = 1<<ENC_FX1_PARAM_2_A;
  encoders[3].b_pin  = 1<<ENC_FX1_PARAM_2_B;
  encoders[3].a_port = &PINA;
```

```
encoders[3].b_port = &PINA;

encoders[4].a_pin  = 1<<ENC_FX1_PARAM_3_A;
encoders[4].b_pin  = 1<<ENC_FX1_PARAM_3_B;
encoders[4].a_port = &PINC;
encoders[4].b_port = &PINC;

encoders[5].a_pin  = 1<<ENC_FX2_SELECT_A;
encoders[5].b_pin  = 1<<ENC_FX2_SELECT_B;
encoders[5].a_port = &PING;
encoders[5].b_port = &PIND;

encoders[6].a_pin  = 1<<ENC_FX2_PARAM_0_A;
encoders[6].b_pin  = 1<<ENC_FX2_PARAM_0_B;
encoders[6].a_port = &PINA;
encoders[6].b_port = &PINA;

encoders[7].a_pin  = 1<<ENC_FX2_PARAM_1_A;
encoders[7].b_pin  = 1<<ENC_FX2_PARAM_1_B;
encoders[7].a_port = &PINA;
encoders[7].b_port = &PINA;

encoders[8].a_pin  = 1<<ENC_FX2_PARAM_2_A;
encoders[8].b_pin  = 1<<ENC_FX2_PARAM_2_B;
encoders[8].a_port = &PINC;
encoders[8].b_port = &PINC;

encoders[9].a_pin  = 1<<ENC_FX2_PARAM_3_A;
encoders[9].b_pin  = 1<<ENC_FX2_PARAM_3_B;
encoders[9].a_port = &PING;
encoders[9].b_port = &PING;


encoders[10].a_pin  = 1<<ENC_SYNTH_PARAM_0_A;
encoders[10].b_pin  = 1<<ENC_SYNTH_PARAM_0_B;
encoders[10].a_port = &PINL;
encoders[10].b_port = &PINK;

encoders[11].a_pin  = 1<<ENC_SYNTH_PARAM_1_A;
encoders[11].b_pin  = 1<<ENC_SYNTH_PARAM_1_B;
encoders[11].a_port = &PINL;
encoders[11].b_port = &PINK;

encoders[12].a_pin  = 1<<ENC_SYNTH_PARAM_2_A;
encoders[12].b_pin  = 1<<ENC_SYNTH_PARAM_2_B;
encoders[12].a_port = &PINL;
encoders[12].b_port = &PINK;

encoders[13].a_pin  = 1<<ENC_SYNTH_PARAM_3_A;
encoders[13].b_pin  = 1<<ENC_SYNTH_PARAM_3_B;
encoders[13].a_port = &PINL;
encoders[13].b_port = &PINK;

encoders[14].a_pin  = 1<<ENC_SYNTH_PARAM_4_A;
encoders[14].b_pin  = 1<<ENC_SYNTH_PARAM_4_B;
encoders[14].a_port = &PINL;
```

```
  encoders[14].b_port = &PINK;

  encoders[15].a_pin  = 1<<ENC_SYNTH_PARAM_5_A;
  encoders[15].b_pin  = 1<<ENC_SYNTH_PARAM_5_B;
  encoders[15].a_port = &PINL;
  encoders[15].b_port = &PINK;

  encoders[16].a_pin  = 1<<ENC_SYNTH_PARAM_6_A;
  encoders[16].b_pin  = 1<<ENC_SYNTH_PARAM_6_B;
  encoders[16].a_port = &PINL;
  encoders[16].b_port = &PINK;

  encoders[17].a_pin  = 1<<ENC_SYNTH_PARAM_7_A;
  encoders[17].b_pin  = 1<<ENC_SYNTH_PARAM_7_B;
  encoders[17].a_port = &PINL;
  encoders[17].b_port = &PINK;

  encoders[18].a_pin  = 1<<ENC_SYNTH_PRESET_A;
  encoders[18].b_pin  = 1<<ENC_SYNTH_PRESET_B;
  encoders[18].a_port = &PINE;
  encoders[18].b_port = &PINE;

  uint8_t i;
  for (i = 0; i < 19; i++) {
    encoders[i].count = 0;
    encoders[i].timer = 0;
  }
}
```

```
#include <avr/io.h>


// Port E
#define ENC_SYNTH_PRESET_A 3
#define ENC_SYNTH_PRESET_B 4
// Port L
#define ENC_SYNTH_PARAM_0_A 0
#define ENC_SYNTH_PARAM_1_A 1
#define ENC_SYNTH_PARAM_2_A 2
#define ENC_SYNTH_PARAM_3_A 3
#define ENC_SYNTH_PARAM_4_A 4
#define ENC_SYNTH_PARAM_5_A 5
#define ENC_SYNTH_PARAM_6_A 6
#define ENC_SYNTH_PARAM_7_A 7
// Port D
#define ENC_FX2_SELECT_B 7
// Port G
#define ENC_FX2_PARAM_3_B 0
#define ENC_FX2_SELECT_A 1
#define ENC_FX2_PARAM_3_A 2
// Port C
#define ENC_FX1_PARAM_3_B 0
#define ENC_FX1_SELECT_A 1
#define ENC_FX1_PARAM_3_A 2
#define ENC_FX1_SELECT_B 3
#define ENC_FX2_PARAM_2_B 4
#define ENC_FX1_PARAM_0_A 5
#define ENC_FX2_PARAM_2_A 6
#define ENC_FX1_PARAM_0_B 7
// Port A
#define ENC_FX1_PARAM_2_B 7
#define ENC_FX2_PARAM_0_A 6
#define ENC_FX1_PARAM_2_A 5
#define ENC_FX2_PARAM_0_B 4
#define ENC_FX2_PARAM_1_B 3
#define ENC_FX1_PARAM_1_A 2
#define ENC_FX2_PARAM_1_A 1
#define ENC_FX1_PARAM_1_B 0
// Port K
#define ENC_SYNTH_PARAM_0_B 7
#define ENC_SYNTH_PARAM_1_B 6
#define ENC_SYNTH_PARAM_2_B 5
#define ENC_SYNTH_PARAM_3_B 4
#define ENC_SYNTH_PARAM_4_B 3
#define ENC_SYNTH_PARAM_5_B 2
#define ENC_SYNTH_PARAM_6_B 1
#define ENC_SYNTH_PARAM_7_B 0


struct enc_struct {
  volatile uint8_t *a_port;
  volatile uint8_t *b_port;
  uint8_t a_pin;
  uint8_t b_pin;
  uint8_t timer;
  uint8_t count;
};
typedef struct enc_struct Encoder;

Encoder encoders[19];

void check_encoder(Encoder*);
void encoder_init(void);
```

```
#define F_CPU 16000000
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <string.h>

#include "k25m.h"

#include "serial_usb.h"
#include "midi.h"

void init_keys (void) {
    DDRB &= ~(1<<PB6);
  DDRH |= (1<<PH3)|(1<<PH4)|(1<<PH5)|(1<<PH6);
  DDRB |= (1<<PB4)|(1<<PB5)|(1<<PB7);

  // for debug
//  DDRD |= (1<<PD0);

  int i, j;
  for (i = 0; i < KEY_ROW_COUNT; i++) {
    for (j = 0; j < KEY_COL_COUNT; j++) {
      keys[i][j].status = KEY_OFF;
      keys[i][j].contact_delta = 0;
      keys[i][j].note_id = i + 8*j + 60;
    }
  }

  strcpy(keys[0][0].note_name, "C 0");
  strcpy(keys[1][0].note_name, "C#0");
  strcpy(keys[2][0].note_name, "D 0");
  strcpy(keys[3][0].note_name, "D#0");
  strcpy(keys[4][0].note_name, "E 0");
  strcpy(keys[5][0].note_name, "F 0");
  strcpy(keys[6][0].note_name, "F#0");
  strcpy(keys[7][0].note_name, "G 0");

  strcpy(keys[0][1].note_name, "G#0");
  strcpy(keys[1][1].note_name, "A 0");
  strcpy(keys[2][1].note_name, "A#0");
  strcpy(keys[3][1].note_name, "B 0");
  strcpy(keys[4][1].note_name, "C 1");
  strcpy(keys[5][1].note_name, "C#1");
  strcpy(keys[6][1].note_name, "D 1");
  strcpy(keys[7][1].note_name, "D#1");

  strcpy(keys[0][2].note_name, "E 1");
  strcpy(keys[1][2].note_name, "F 1");
  strcpy(keys[2][2].note_name, "F#1");
  strcpy(keys[3][2].note_name, "G 1");
  strcpy(keys[4][2].note_name, "G#1");
  strcpy(keys[5][2].note_name, "A 1");
  strcpy(keys[6][2].note_name, "A#1");
  strcpy(keys[7][2].note_name, "B 1");

  strcpy(keys[0][3].note_name, "C 2");
```

```
}

void init_timer0( void ) {
  TCCR0A = 1<<WGM01; // oc0a oc0b both disconnected
  TCCR0B = 1<<CS01; // clk prescale = 256
  OCR0A  = 80; // 16MHz / 8 / 80 = 25kHz = 40us
  TIMSK0 = 1<<OCIE0A;
}

static uint8_t key_column_count = 0;
static uint8_t key_row_count = 0;
static uint8_t cache_h, cache_b;
//ISR(TIMER0_COMPA_vect) {
void poll_k25m() {
//      PORTD |= (1<<PD0);
      cache_h = PORTH;
      cache_h &= ~COL_MSK;
      cache_h |= COL_MSK & (key_column_count << COL_SFT);
      PORTH = cache_h;

      cache_h = PORTH;
      cache_h &= ~ROW_MSK_H;
      cache_h |= ROW_MSK_H & (key_row_count << ROW_SFT_H);
      cache_b = PORTB;
      PORTH = cache_h;
      cache_b &= ~ROW_MSK_B;
      cache_b |= ROW_MSK_B & (key_row_count << ROW_SFT_B);
      PORTB = cache_b;


      k = &(keys[key_row_count][key_column_count/2]);
      if (PINB & (1<<PB6)) {
        if (key_column_count & 1) { // if column is odd (first contact)
          if ( k->status == KEY_OFF ) {
            k->status = KEY_CONTACT;
            k->contact_delta = 0;
          }
          else if (k->status == KEY_CONTACT) {
            k->contact_delta++;
          }
        }
        else { // if column is even (second contact)
          if ( k->status == KEY_CONTACT ) {
            PORTD ^= (1<<PD0);
            k->status = KEY_ON;
            MidiPacket p;
            p.status = MIDI_NOTE_ON;
            p.note = k->note_id;
            p.velocity = ~k->contact_delta;
            mbuffer_write(p);
            print_serial_usb("Key press detected. Buffer length = %u\n",
                mbuffer_count());
//            print_serial_usb("Key press detected.");
          }
        }
      }
      else { // the switch is open, i.e. key is not pressed
        if ( k->status == KEY_ON ) {
          k->status = KEY_OFF;
          MidiPacket p;
          p.status = MIDI_NOTE_OFF;
          p.note = k->note_id;
          p.velocity = 0;
          mbuffer_write(p);
        } // KEY_ON
      } // else: switch open

      if ( key_column_count == 6 && key_row_count == 0) {
        key_column_count = 7;
      }
      else if (key_column_count == 7 && key_row_count == 0) {
        key_column_count = 0;
      }
      else if ( key_row_count == 7 ) {
        key_row_count = 0;
        key_column_count++;
      } else {
        key_row_count++;
      }
//      PORTD &= ~(1<<PD0);
}
```

```
#define COL_SFT 3
#define COL_MSK (7<<3)

#define ROW_SFT_H 6
#define ROW_MSK_H (1<<6)
#define ROW_SFT_B 3
#define ROW_MSK_B (6<<3)

extern FILE uart_output;

enum key_status_enum {KEY_OFF, KEY_CONTACT, KEY_ON};
typedef enum key_status_enum KeyStatus;

struct key_struct {
  KeyStatus status;
  uint32_t contact_delta;
  char note_name[4];
  uint8_t note_id;
};
typedef struct key_struct Key;

#define KEY_ROW_COUNT 8
#define KEY_COL_COUNT 4
Key keys[8][4];
Key *k;


void init_keys( void );
void init_timer0( void );
void poll_k25m( void );
```

```
#include "lcd.h"

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

FILE fx_lcd = FDEV_SETUP_STREAM(USART3_write_char, NULL, _FDEV_SETUP_WRITE);
FILE synth_lcd = FDEV_SETUP_STREAM(USART1_write_char, NULL, _FDEV_SETUP_WRITE);

void fx_cursor_move( int line, int row ) {
  USART3_write_char(0xFE, NULL);
  int pos;
  switch(line) {
    case 0:
      pos = 128 + row;
      break;
    case 1:
      pos = 192 + row;
      break;
    default:
      return;
      break;
  }
  USART3_write_char(pos, NULL);
}

void synth_cursor_move( int line, int row ) {
  USART1_write_char(0xFE, NULL);
  int pos;
  switch(line) {
    case 0:
      pos = 128 + row;
      break;
    case 1:
      pos = 192 + row;
      break;
    case 2:
      pos = 148 + row;
      break;
    case 3:
      pos = 212 + row;
      break;
    default:
      return;
      break;
  }
  USART1_write_char(pos, NULL);
}

// LCD Synth
#ifndef F_CPU
#define F_CPU 16000000UL
#endif
```

```
#define BAUD 9600
#include <util/setbaud.h>
void fx_init( void ){
    /* Set baud rate */
    UBRR3H = UBRRH_VALUE;
    UBRR3L = UBRRL_VALUE;

  #if USE_2X
    UCSR3A |= _BV(U2X0);
    #else
    UCSR3A &= ~(_BV(U2X0));
    #endif

    /* Enable transmitter */
    UCSR3B = (1<<TXEN3)|(1<TXCIE3);
    /* Set frame format: 8data, 1stop bit */
    UCSR3C = (3<<UCSZ30);

  cbuffer_init(&fx_buf);
}

int USART3_write_char( char c, FILE *stream) {
  if (c == '\n') {
    USART3_write_char('\r', stream);
  }
  cbuffer_write(&fx_buf, c);
  return 0;
}

ISR(USART3_TX_vect)
{
  fx_flush();
}

void fx_flush( void ) {
  if (cbuffer_count(&fx_buf) > 0) {
    UDR3 = cbuffer_read(&synth_buf);
  }
}

// Synth
#ifndef F_CPU
#define F_CPU 16000000UL
#endif

#define BAUD 9600
#include <util/setbaud.h>
void synth_init( void ){
    /* Set baud rate */
    UBRR1H = UBRRH_VALUE;
    UBRR1L = UBRRL_VALUE;

  #if USE_2X
    UCSR1A |= _BV(U2X0);
    #else
    UCSR1A &= ~(_BV(U2X0));
    #endif

    /* Enable transmitter */
    UCSR1B = (1<<TXEN1)|(1<<TXCIE1);
    /* Set frame format: 8data, 1stop bit */
    UCSR1C = (3<<UCSZ30);

  cbuffer_init(&synth_buf);
}

int USART1_write_char( char c, FILE *stream) {
  if (c == '\n') {
    USART1_write_char('\r', stream);
  }
  cbuffer_write(&synth_buf, c);
  return 0;
}

ISR(USART1_TX_vect)
{
 // PORTD ^= (1<<PD0);
  synth_flush();
//  PORTD &= ~(1<<PD0);
}

void synth_flush( void ) {
  if (cbuffer_count(&synth_buf) > 0) {
    UDR1 = cbuffer_read(&synth_buf);
  }
}

void cbuffer_init(CharBuffer *b)
{
  b->write_loc = 0;
  b->read_loc = 0;
}

void cbuffer_write(CharBuffer *b, char c)
{
  while (cbuffer_count(b) == CHAR_BUF_LEN - 1);
  b->char_buffer[b->write_loc] = c;
  b->write_loc = (b->write_loc + 1) % CHAR_BUF_LEN;

}

char cbuffer_read(CharBuffer *b)
{
  char c = '\0';
  if (cbuffer_count(b) > 0) {
    c = b->char_buffer[b->read_loc];
    b->read_loc = (b->read_loc + 1) % CHAR_BUF_LEN;
  }
  return c;
}

uint8_t cbuffer_count (CharBuffer *b) {
  return (b->write_loc - b->read_loc + CHAR_BUF_LEN) % CHAR_BUF_LEN;
}
```

```
#include <stdio.h>

#define CHAR_BUF_LEN 128


// Ring buffer
typedef struct {
  volatile uint8_t write_loc;
  volatile uint8_t read_loc;
  uint8_t char_buffer[CHAR_BUF_LEN];
} CharBuffer;

void cbuffer_init(CharBuffer *b);
void cbuffer_write(CharBuffer *b, char c);
char cbuffer_read(CharBuffer *b);
uint8_t cbuffer_count (CharBuffer *b);


// LCD
CharBuffer fx_buf;
void fx_cursor_move( int line, int row );
void fx_init( void );
int USART3_write_char( char c, FILE *stream );
void fx_flush( void );
extern FILE fx_lcd;
#define print_fx(...) fprintf(&fx_lcd, __VA_ARGS__)

// Synth
CharBuffer synth_buf;
void synth_cursor_move( int line, int row );
void synth_init( void );
int USART1_write_char( char c, FILE *stream );
void synth_flush( void );
extern FILE synth_lcd;
#define print_synth(...) fprintf(&synth_lcd, __VA_ARGS__)
```

```
/*
 * DANALOG.c
 *
 * Created: 5/25/2017 12:20:26 AM
 * Author : Vikrant
 */

#define F_CPU 16000000UL
#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>

#include "k25m.h"
#include "serial_usb.h"
#include "SPI.h"
#include "SynthMux.h"
#include "lcd.h"
#include "potadc.h"
#include "encoder.h"
#include "midi.h"

int main(void)
{
    /*Initialize Serial Print*/
//  USART0_Init();

//  /*Initialize Keyboard*/
    init_keys();
////    init_timer0();
  mbuffer_init();
//
//  /*Initialize Switches and Buttons*/
//  Synth_Mux_Init();
//
//  /*Initialize Encoders*/
    encoder_init();
//
//  /*Initialize ADC*/
//  PotADC_Init();
//
//  /*Initialize LCDs*/
    synth_init();
//  fx_init();
//
//  /*Initialize SPI*/
    SPI_SlaveInit();
//
//  /*Enable Interrupts*/
    sei();

  // for debug
  DDRD |= (1<<PD0);

 uint32_t counter = 0;
  while (1)
 {

    /*Check Swtich and Button Positions*/
//   if (counter == 0) {
//    Synth_Mux_Select();
//   }

    /*Check Encoders*/
  uint8_t i;
    for (i = 0; i < 19; i++) {
      check_encoder(&encoders[i]);
    }
//   synth_cursor_move(0,0);
//   print_synth("%03u", encoders[12].count);
//   print_synth("buffer count =  %04d", cbuffer_count(&synth_buf));
//   synth_cursor_move(1,0);
//   print_synth("counter =  %04d", counter++);
//   synth_flush();
//   _delay_ms(1);

    /*Check Potentiometer Positions*/
//   if (counter == 50) {
//    PotADC_Poll();
//   }
//
  poll_k25m();

  if (cbuffer_count(&synth_buf) == 0) {
   if (counter++ == 1000) {
    counter = 0;
    synth_cursor_move(0,0);
    print_synth("%03u  %03u  %03u  %03u  ", encoders[11].count, encoders[13
      ].count, encoders[15].count, encoders[17].count);
    print_synth("mDEP mATK mSUS DECAY");
    print_synth("%03u  %03u  %03u  %03u  ", encoders[10].count, encoders[12
      ].count, encoders[14].count, encoders[16].count);
    print_synth("mR8O cATK cSUS PHASE");
    synth_flush();
   }
  }


//   print_serial_usb("MIDI buffer length = %u\r", mbuffer_count());
 }
}
```

```
PORT=/dev/cu.usbmodem1411
MCU=atmega2560
CFLAGS=-g -Wall -mcall-prologues -mmcu=$(MCU) -O3
LDFLAGS=-Wl,-gc-sections -Wl,-relax
CC=avr-gcc
TARGET=main
OBJECT_FILES=main.o k25m.o lcd.o serial_usb.o encoder.o SynthMux.o
SPI.o potadc.o midi.o

all: $(TARGET).hex

clean:
	rm -f *.o *.hex *.obj *.hex

%.hex: %.obj
	avr-objcopy -R .eeprom -O ihex $< $@

%.obj: $(OBJECT_FILES)
	$(CC) $(CFLAGS) $(OBJECT_FILES) $(LDFLAGS) -o $@

program: $(TARGET).hex
	avrdude -p $(MCU) -c wiring -P $(PORT) -b 115200 -F -U
flash:w:$(TARGET).hex -D
```

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include "midi.h"

MidiPacket midi_buffer[M_BUF_LEN];
uint8_t buf_write_loc, buf_read_loc;

#undef BAUD
#define BAUD 31250
#include <util/setbaud.h>
void USART2_Init( void ){
    /* Set baud rate */
    UBRR2H = UBRRH_VALUE;
    UBRR2L = UBRRL_VALUE;

  #if USE_2X
    UCSR2A |= _BV(U2X2);
    #else
    UCSR2A &= ~(_BV(U2X2));
    #endif

    /* Enable reciver with interupts  */
    UCSR2B = (1<<RXEN2) | (1<<RXCIE2);
    /* Set frame format: 8data, 1stop bit */
    UCSR2C = (3<<UCSZ20);
}

ISR(USART2_RX_vect)
{
  // need to fix this later
  //buffer_write(UDR2);
}

void mbuffer_write( MidiPacket p ) {
  if (!mbuffer_full()) {
    midi_buffer[buf_write_loc] = p;
    buf_write_loc = (buf_write_loc + 1) % M_BUF_LEN;
  }
}

MidiPacket mbuffer_read( void ) {
  MidiPacket p;
  if (!mbuffer_empty()) {
    p = midi_buffer[buf_read_loc];
    buf_read_loc = (buf_read_loc + 1) % M_BUF_LEN;
  }
  return p;
}

void mbuffer_init( void ) {
  buf_write_loc = 0;
  buf_read_loc = 0;
}

uint8_t mbuffer_empty( void ) {
  return buf_read_loc == buf_write_loc;
}

uint8_t mbuffer_full( void ) {
  return mbuffer_count()  == M_BUF_LEN - 1;
}

uint8_t mbuffer_count( void ) {
  return (buf_write_loc - buf_read_loc + M_BUF_LEN) % M_BUF_LEN;
}
```

```
// Init UART RX to be compatible with MIDI
void USART2_Init( void );

#define MIDI_NOTE_ON  0b10010000
#define MIDI_NOTE_OFF 0b10000000

/* ====  MIDI buffer ==== */
#define M_BUF_LEN 32

struct midi_packet_struct {
  uint8_t status;
  uint8_t note;
  uint8_t velocity;
};

typedef struct midi_packet_struct MidiPacket;

// MIDI state variables
extern MidiPacket midi_buffer[M_BUF_LEN];
extern uint8_t buf_write_loc, buf_read_loc;

// Buffer helper functions
void mbuffer_init( void );
void mbuffer_write( MidiPacket data );
MidiPacket mbuffer_read( void );
uint8_t mbuffer_empty ( void );
uint8_t mbuffer_full ( void );
uint8_t mbuffer_count ( void );
```

```c
//potadc.c

#include "serial_usb.h"
#include "potadc.h"
#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>

uint8_t adc_cache;
uint8_t adc_array[8];

void PotADC_Init(void)
{
    ADMUX = (1<<REFS0)|(1<<ADLAR);
    ADCSRA = (1<<ADEN);
}

void PotADC_Poll(void)
{
    int i, j;
    for(i=0; i<8; i++)
    {
        adc_cache = ADMUX;
        adc_cache &= ~(ADC_MSK);
        adc_cache |= i;
        ADMUX = adc_cache;
        ADCSRA |= (1<<ADSC);
        while(ADCSRA & (1<<ADSC));
        adc_array[i] = ADCH;


    }
    adc_array[5] = ~(adc_array[5]);
    //
    for(j=0; j<8; j++)
    {
        //print_serial_usb("%u\n", adc_array[j]);
    }

}
```

```
//potadc.h

#include <avr/io.h>
#ifndef F_CPU
#define F_CPU 16000000
#endif

#define ADC_MSK 7

extern uint8_t adc_array[8];

void PotADC_Init(void);
void PotADC_Poll(void);
```

```
#include "serial_usb.h"
#include <avr/io.h>

FILE serial_usb = FDEV_SETUP_STREAM(USART0_write_char, NULL, _FDEV_SETUP_WRITE)
    ;

#ifndef F_CPU
#define F_CPU 16000000
#endif

#define BAUD_TOL 5
#define BAUD 115200
#include <util/setbaud.h>
void USART0_Init( void ){
  /* Set baud rate */
  UBRR0H = UBRRH_VALUE;
  UBRR0L = UBRRL_VALUE;

  #if USE_2X
    UCSR0A |= _BV(U2X0);
  #else
    UCSR0A &= ~(_BV(U2X0));
  #endif

  /* Enable and transmitter */
  UCSR0B = (1<<TXEN0);
  /* Set frame format: 8data, 1stop bit */
  UCSR0C = (3<<UCSZ00);
}

int USART0_write_char( char c, FILE *stream) {
  while ( !( UCSR0A & (1<<UDRE0)) );
  UDR0 = c;
  if (c == '\n') {
    USART0_write_char('\r', stream);
  }
  return 0;
//  while ( !(UCSR0A && (1<< TXC0)) ); // loop until tx complete is set
}
```

```
#include <stdio.h>

void USART0_Init( void );
int USART0_write_char( char c, FILE *stream );
extern FILE serial_usb;
#define print_serial_usb(...) fprintf(&serial_usb, __VA_ARGS__)
```

```
//SPI.c

#include "potadc.h"
#include "encoder.h"
#include "midi.h"

#include "SPI.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#include "serial_usb.h"
#include <util/delay.h>
uint8_t spidata, spistat = 0;
uint8_t switches;

/*ISR State Variables*/
uint8_t spi_tx_cnt = 0;
uint8_t spi_tx_type = 0;


// MIDI transfer state
#define MIDI_TX_STATUS_SENT 2
#define MIDI_TX_NOTE_SENT 1
MidiPacket current_packet;

void SPI_SlaveInit(void)
{
    /* Set MISO output, all others input */
    DDRB |= (1<<DDB3);
    /* Enable SPI */
    SPCR = (1<<SPE)|(1<<SPIE);
    sei();
}

ISR(SPI_STC_vect)
{
        if(spi_tx_cnt>0)
        {
            if(spi_tx_type == SPI_MIDI_CMD)
            {
        switch (spi_tx_cnt) {
          case MIDI_TX_STATUS_SENT:
            SPDR = current_packet.note;
            break;
          case MIDI_TX_NOTE_SENT:
            SPDR = current_packet.velocity;
            break;
        }
                spi_tx_cnt--;
            }
            else if(spi_tx_type == SPI_ENC_CMD)
            {
                SPDR = encoders[19-spi_tx_cnt].count;
                spi_tx_cnt--;
            }
            else if(spi_tx_type == SPI_POT_CMD)
            {
                SPDR = adc_array[8-spi_tx_cnt];
                spi_tx_cnt--;
            }
        }
        else {
            spidata = SPDR;
            if(spidata == SPI_MIDI_CMD)
            {
        if (!mbuffer_empty()) {
          current_packet = mbuffer_read();
          SPDR = current_packet.status;
          spi_tx_cnt = MIDI_TX_STATUS_SENT;
          spi_tx_type = SPI_MIDI_CMD;
        }
        else {
          SPDR = 0;
        }
            }
            else if(spidata == SPI_ENC_CMD)
            {
                SPDR = encoders[0].count;
                spi_tx_cnt = 18;
                spi_tx_type = SPI_ENC_CMD;
            }
            else if(spidata == SPI_POT_CMD)
            {
                SPDR = adc_array[0];
                spi_tx_cnt = 7;
                spi_tx_type = SPI_POT_CMD;
            }
            else if(spidata == SPI_SWT_CMD)
            {
                SPDR = switches;
            }
        }

}
```

```
#include <avr/io.h>
#include <stdio.h>

#define SPI_MIDI_CMD 0x01
#define SPI_ENC_CMD 0x02
#define SPI_POT_CMD 0x03
#define SPI_SWT_CMD 0x04

void SPI_SlaveInit(void);
char SPI_SlaveReceive(void);
extern uint8_t switches, spidata;
```

```
//Synth Mux C file
#include "SynthMux.h"
#include "serial_usb.h"
#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>

uint8_t switches;

void Synth_Mux_Init(void)
{
        PORTJ |= (1<<PJ0);   //Set Pullup on PJ0
        DDRG |= (1<<DDG5);   //Set PG5, PE5, PB7, and PD2 as outputs
        DDRE |= (1<<DDE5);
        DDRB |= (1<<DDB7);
        DDRD |= (1<<DDD2);
}

uint8_t Synth_Mux_Select(void)
{
    uint8_t cache_g = 0;
    uint8_t cache_e = 0;
    uint8_t cache_b = 0;
    uint8_t cache_d = 0;
    uint8_t i=0;
    uint8_t sw1=0, sw2=0, sw3=0, sw4=0, sw5=0;


    for(i=0; i<12; i++)
    {
        // Set S0
        cache_g = PORTG;
        cache_g &= ~(1<<PG5);
        cache_g |= (i&S0_MSK)<<S0_SFT;
        PORTG = cache_g;

        // Set S1
        cache_e = PORTE;
        cache_e &= ~(1<<PE5);
        cache_e |= (i&S1_MSK)<<S1_SFT;
        PORTE = cache_e;

        //Set S2
        cache_b = PORTB;
        cache_b &= ~(1<<PB7);
        cache_b |= (i&S2_MSK)<<S2_SFT;
        PORTB = cache_b;

        //Set S3
        cache_d = PORTD;
        cache_d &= ~(1<<PD2);
        cache_d |= (i&S3_MSK)>>S3_SFT;
        PORTD = cache_d;


        _delay_us(1);
        if ((PINJ&(1<<PINJ0)) == 0)
        {
            switch(i)
            {
                case 0 :
                sw1 = 5;
                break;
                case 1 :
                sw1 = 3;
                break;
                case 2 :
                sw1 = 4;
                break;
                case 3 :
                sw1 = 2;
                break;
                case 4 :
                sw1 = 1;
                break;
                case 5 :
                sw2 = 2;
                break;
                case 6 :
                sw2 = 3;
                break;
                case 7 :
                sw2 = 1;
                break;
                case 8 :
                sw3 = 1;
                break;
                case 9 :
                sw5 = 1;
                break;
                case 10 :
                sw4 = 1;
                break;
            }
        }
    }
    switches = (sw1<<5)|(sw2<<3)|(sw3<<2)|(sw4<<1)|sw5;
    //print_serial_usb("%u\n",switches);
    return switches;
}
```

```
//Synth Mux H file
#include <avr/io.h>

#ifndef F_CPU
#define F_CPU 16000000
#endif

#define S0_MSK 1
#define S1_MSK 2
#define S2_MSK 4
#define S3_MSK 8

#define S0_SFT 5
#define S1_SFT 4
#define S2_SFT 5
#define S3_SFT 1

extern uint8_t switches;

void Synth_Mux_Init(void);
uint8_t Synth_Mux_Select(void);
```