No Amp - E20

Informal report by Benjamin M’Sadoques

Intro

Say I’m looking to buy a guitar, and I find a few potentially good models; they sound excellent on someone else’s guitar setup, but I have a completely different setup. I want to know how this guitar would sound on my setup. If only I could record the guitar signal and play it at the same volume it was recorded at.

The goal of this project term was to design a hardware and software setup that could work for the project and to update the wiki page. I figured out a process that can record the guitar signal and play a good approximation of the original signal. I made a hardware setup that includes most of the parts necessary with different types of components, setup options, and suggestions for each part. I also started the software programming on the computer-side of the project.

Approach

The approach I took for the project was to research the project before starting any development. The research was mainly done using common search engines to quickly gain a surface-level understanding on the topics I needed to know. The first task was to create a hardware setup for the project. At first, I wanted to create a low-cost compact design and find boards with integrated ADC and DAC without amplifying the guitar signal. I thought you would only need a device to record the signal and not play it. I had some of the right ideas but barely understood how the hardware works. I decided to tackle the unfamiliar parts of the project first, then work towards the more-familiar computer-side software component.

Every week I picked a new topic I wanted to understand to make the project work. I wanted to generally understand how I could use different hardware components and what qualities of each were relevant for the project. I kept building the hardware setup based on what I learned and kept changing my earlier ideas. I changed my design philosophy from a compact design to a more realistic and flexible design where I could choose individual components. Later in the project when I addressed the software side, I could only work with the computer-side software since the embedded software depends too much on the board chosen. I had to create a cross-platform program but I was not sure what I was looking for in an environment. I decided it was best to find something simple that could work, rather than get every part I could possibly need. Overall I researched to build a basic understanding of how to build the system and what would actually work.

Milestones

Reduced setup

The first major milestone for the project was figuring out the recording procedure. The original plan for recording was to take the signal from the guitar, have the computer play the file it recorded, then alter the computer’s recording to match the signal’s level. I figured out an easier solution would be to simply record the guitar signal and send the recording to the computer. The board does not need to create the recording file, it only needs to send each sample through USB. The computer can create the recording file as needed and interact with the user. Next some other user will load the file onto their system and play it through the same hardware setup but reversed. The computer will pass samples to the board through USB and the board will play samples at the recorded sample rate. The samples will be changed back into a good approximation of the original guitar signal and sent to the user’s guitar setup.

WAV file construction

WAV files are the most common choice for uncompressed audio files so I had to understand how they worked to know how to work with them. Wav files are split into three sub chunks: the header, format, and data. The header contains the letters “RIFF” the subchunk size, and “WAVE” indicating the file type. The format mostly contains the letters “FMT” for format, the audio format PCM, number of audio channels, sample rate in Hz, and bits per sample. The Data subchunk contains the letters “DATA” and contains the recording data in samples [1]. This format is feasible to edit in hex digits so I could create my own file parser, the main problem is dealing with byte order, writing the samples, and writing parts of the file after the recording stops. (see appendix A for a wav header file) Likely there are some good APIs for C++ audio wav format recording so the work to create a parser may be unnecessary.

USB Communication

Given the new setup I had to figure out how to communicate with the computer. USB communication is a common choice for interfacing with many devices but I wanted to understand a bit about how it would work for the project. Lots of MCUs, DSPS, and some FPGAs implement USB communication in some form. USB offers different modes to transfer data such as isronconus, transfer that gives a guaranteed data rate; Interrupt transfers, that provides a quick low-latency data transfer; bulk transfer, that can send a lot of data at once; and control transfers, that are used for sending command messages to a USB device. USB also provides lots of speed options such as low speed at 1.5 Mbit/s, Full speed at 12 Mbit/s, High speed at 480 Mbit/s, SuperSpeed at 5 Gbit/s, and SuperSpeed+ at 10 Gbit/s [2]. For this project isochronous transfer can be used to transfer samples between the computer and the board at the same rate. Control transfer can be used to switch the board into different modes.

Recording a Guitar Signal

Figuring out how to record a guitar signal made up the bulk of the project. I had little knowledge on recording analog signals into a digital format. The first problem was the guitar signal itself. There is no consistent guitar voltage range, every guitar produces a different voltage range depending on the style of pickup [3] and the notes played from as low as 30 mv [4] to nearly 2 v [5]. Analog to Digital Converters (ADCs) convert analog signals to digital data depending on a reference voltage, the bit rate, sample rate, and many other factors. The Least Significant Bit

(LSB) is the smallest voltage difference the ADC can detect. The more bits the ADC has, the smaller the LSB as shown in this formula. $LSB = \frac{Vref}{2^N}$, where N is the number of bits. The ADC never reaches the voltage reference and the highest voltage the ADC can detect is the full scale voltage $FS = Vref - 1 * LSB$ [6]. We want the guitar signal maximum to be this full scale voltage. At first, there seemed to be two potential solutions, to change the reference voltage for any guitar, or to amplify the signal to reach the same reference voltage for every guitar. After looking at various ADCs I could not find any parts with a low enough reference voltage. The second solution requires another part to amplify the signal. Voltage Gain Amplifiers (VGAs) take some analog voltage to amplify an input voltage. We can use one of the board's analog DC supplies (usually 3.3v) as a reference voltage. We need at most 100x the original voltage or a 40 dB increase for the signal to reach the full scale voltage [7]. The VGA also needs a high enough bandwidth to not distort the guitar signal. Bandwidth is the difference between the lowest and highest frequencies in an analog signal. The theoretical minimum bandwidth we need is 2x the highest frequency the guitar can produce [8]. I do not know this value but I know that the bandwidth should be the same or higher than my sample rate of 44.1 kHz. The DAC had to reverse the process. To de-amplify the guitar signal without distorting it we can use an attenuator with a bandwidth of at least 44.1 kHz.

Hardware Setup

Once I knew how to record the signal I just needed to find the type of parts needed to create a basic setup. The goal was to create a simple setup that could work rather than a complex setup with specific parts. I have the knowledge to look for parts but not optimize a setup. The first component is a 6.35mm jack. Unfortunately I found very few articles explaining how these devices work. One chance to the original setup is to include a multiplexer/demultiplexer. This bi-directional component can be used to switch between 4 wires to 1 6.35mm jack instead of having multiple inputs/outputs. The next component is a Voltage Gain Amplifier (VGA) that is used to amplify the guitar signal to the ADC full-scale voltage. The ADC needs to communicate with the board or be integrated with the board. Then the board can send the samples to the computer through USB. The computer can manage creating the wave file and recording the values needed from calibrating the system for the guitar. When playing the recording, the computer can send the samples back to the board. The board can send the samples to the DAC to output the amplified signal. An attenuator is used to accurately de-amplify the analog signal back to a good approximation of the original guitar signal. (see Appendix B for hardware block diagram)

Programming Setup

The last major milestone was to create the software setup on the computer. I needed to create a cross platform program. I wanted to use C++ because the board will likely be programmed in C/C++. At first I tried to use Visual Studio 2019 to create a GUI-based application. The IDE works well with windows-based forms but cross platform GUI creation is somewhat difficult. My first choice was to use GTK+ as the cross-platform GUI component since it has Glade, a GUI builder program. Having a builder program is necessary for creating good-looking user interfaces. The programmer can focus on making the file visually rather than using code. The main problem with GTK+ was that the latest version of Glade was only available through a package distribution system rather than a standard download. I was unable to get the libraries to work with Visual Studio so I looked for other options. I considered using WX widgets since it was another cross-platform GUI creator but there were too many builders to choose from. Later I tried searching for all-in-one creators and settled on using QT to create both the code and the GUI. QT is built for creating C++ cross-platform GUI-based applications so it seemed like a perfect fit. It features many APIs and its own GUI builder. So far I created a project that uses Model/View design, a simple GUI, and can use a style sheet. (see appendix C for a picture of the GUI so far)

Setup

Here are all the components that seem necessary for the project to work. This list does not include any sort of wires, LEDs or other analog components that may be needed, just the core components with their purpose, requirements, types, and suggestions for each.

6.35 mm Jack Interface:

Purpose: To interface with a 6.35 mm jack for both input guitar and output amp. A switch would allow some pin to detect if the jack is inserted to turn some LED on.

Requirements:

- Bi-directional
- SPDT Switches to detect when a plug is inserted (optional)

Types: PCB, SDPT

Suggestions:

Notes: These parts are common, cheap, but confusing. There are many products with varying pin layouts (from 3 to 9 pins) and few datasheets. Searching for information mainly gives products but one article helped explain the jack layout. https://www.cuidevices.com/blog/understanding-audio-jack-switches-and-schematics

- https://www.amazon.com/ZXHAO-6-35mm-Female-Socket-Headphone/dp/B07M62VBZR

    - 6-pin
    - Has screw threads
- https://elabbay.myshopify.com/products/6-35aj-bo-v1b-6-35mm-stereo-audio-jack-breakout-board-elabguy
    - Breakout board
    - Understandable pin layout

1 to 4 MUX & DEMUX:

Purpose: to switch between one 6.35mm input/output interface to different routes. More interfaces would be easier to build but one will be more convenient for the end user.

Requirements:

- Digital controls (kind of a given)
- Differential voltage range -3.3v to 3.3v
- 1:4
- 48kHz bandwidth
- bi-directional

Types: Both multiplexer and demultiplexer (bi-directional)

Suggestions:

- 74CB3Q3253: https://www.nexperia.com/products/analog-logic-ics/analog/bus-switches/series/74CB3Q3253.html
    - bi-directional
    - 2-channel

    - -0.5v to 7v (does not support negative range)
    - 500 Mhz bandwidth
- CD74HC4051-EP: https://www.ti.com/lit/ds/symlink/cd74hc4051-ep.pdf?ts=1594583925909&ref_url=https%253A%252F%252Fwww.google.com%252F
    - Bi-directional
    - -5v to 5v
    - 10 Mhz bandwidth (approximate)

Voltage Converter: (optional)

Purpose: To power the MUX/DEMUX with positive and negative voltage. This component is optional because it may not be necessary to power the MUX/DEMUX. Guitar signals do have a negative component so this component is necessary to preserve the full signal.

Requirements:

- Supplies a negative and positive voltage of the original voltage

Types: N/A

Suggestions:

- LT1026: https://www.analog.com/media/en/technical-documentation/data-sheets/1026fb.pdf
    - Up to +/-18v output
    - Outputs both negative and positive voltage
- TC7660:

https://components101.com/ics/tc7660-voltage-converter

  - Up to +/- 10v output

VGA/PGA:

Purpose: To control amplifying the guitar signal to the ADC full scale voltage. The amplification needed will be different per guitar and decided when calibrating the guitar.

Requirements:

- Wide range of amplification
- At least 100x the original voltage (or 40dB, most manufactures seem to put this number in dB)
- 48kHz bandwidth

Types: VGA, PGA

Case 1: VGA

+ Wide voltage control
- Analog control and requires using the DAC

Case 2: PGA

+ Digitally controlled
- Limited voltage control (likely not wide enough to meet the requirements without a serial interface.)

Suggestions:

- AD8338: https://www.analog.com/media/en/technical-documentation/data-sheets/AD8338.pdf

    - Wide range
    - 0 dB to 80dB
    - 18MHz bandwidth
- THS7530:

  https://www.ti.com/product/THS7530
    - Wide range
    - 11.6 dB to 46.5dB
    - 300 MHz bandwidth

ADC:

Purpose: To accurately convert the amplified guitar signal to digital data based on the 3.3v reference voltage supplied by the board.

Requirements:

- At least 16-bits
- Capable of a 44.1kHs sample rate (44.1KSPS)
- Works well with a 3.3 v reference

Types: Successive approximation, Sigma Delta, Pipelined

- All of these types are highly accurate ADCs. Quality is more important than sample rate.
    - Successive approximation (SAR) works well for both accuracy and speed
    - Sigma Delta is very slow but works well for high quality audio processing

    - Pipelined ADCs are more refined SAR and serve as a good middle ground

Case 1: External ADC

+ Higher quality
+ Offers flexibility
+ Can have matching DACs
- More difficult to wire/build

Case 2: Integrated ADC

+ More compact design
+ Easier to build
- Design is fixed

Suggestions: (Only external ADCs)

- ADS8920B: https://www.ti.com/lit/ds/symlink/ads8922b.pdf?ts=1594564525365&ref_url=https%253A%252F%252Fwww.google.com%252F
    - 16-bit
    - 1MSPS
    - SAR
    - Vref: 2.5 to 5.2
- AD7723: https://www.analog.com/media/en/technical-documentation/data-sheets/AD7723.pdf
    - 16-bit
    - 460KSPS

    - Sigma Delta
    - Vref: 0.3v to 7.3v
- ADS5483: https://www.ti.com/lit/ds/slas565c/slas565c.pdf?ts=1594566624222&ref_url=https%253A%252F%252Fwww.google.com%252F
    - 16-bit
    - 135MSPS
    - Monolithic Pipelined
    - Vref: 1.2v (internal)

DAC:

Purpose: To accurately convert the digital guitar signal back into an analog signal based on the 3.3 reference voltage supplied by the board.

Requirements:

- At least 16-bits
- Capable of a 44.1kHs output rate
- 3.3 reference voltage

Types: PWM, R-2R, multi-bit, Delta Sigma

- All of these types could work for the conversion process.
    - PWM uses a signal to produce different voltage ranges, it can produce any amount of bits for a digital signal. These types are simple and fairly common. Boards with a PWM can also make a DAC using this technique.

    - R-2R use resistor ladders use lots of resistors to construct the output voltage.
    - Multi-bit has a high-resolution and quality. This option may be more expensive than the Delta Sigma option.
    - Delta Sigma uses oversampling to encode a high bit-rate signal into a high-sample rate. They are comparable to the multi-bit DAC but can cause some noise.

Suggestions: (Only external DACs)

- LTC1597
  https://www.analog.com/en/technical-articles/16bit-parallel-dac-has-1lsb-linearity-ultralow-glitch-and-accurate-4quadrant-resistors.html
    - 16-bit
    - +/- 10v output range
    - 100kHz bandwidth (some noise)
    - R/2R
- PCM270xC
  https://www.ti.com/product/PCM2705C
    - 16-bit
    - 3v to 3.6 ref (Vdd)
    - 48kHz sampling rate
    - Good for audio
    - Sigma Delta
- AD1959

https://www.analog.com/media/en/technical-documentation/data-sheets/AD1959.pdf

  - 24-bit
  - 4.50v to 5.50v (higher than desired but can work)
  - 192kHz
  - Multibit Sigma Delta

Attenuator:

Purpose: to accurately attenuate the analog DAC signal back to a good approximation of the original guitar signal.

Requirements:

- At latest 44.1kHz bandwidth
- Analog or Digital control
- Wide range of voltage
- At least -40db

Types:

- Step attenuator
- Continuously variable

Suggestions:

- LM1971: https://www.ti.com/product/LM1971#product-details##description
  - Digitally controlled
  - 0dB to -62 dB with 1dB steps (about 1.122* gain)
  - Audio oriented

DSP/Microcontroller/FPGA:

Purpose: To control/power all the other components. To configure the VGA to amplify the weak guitar signal's max voltage to 3.299 v (the ADC full-scale voltage). To record the digital signal output from the ADC and transport it to the computer via a USB cable along with other needed values to play the signal properly. To play the digital signal using the DAC. To control the Attenuator to set the DAC full scale voltage to the max voltage of the original guitar signal.

Requirements:

- Inexpensive
- Compact design
- Few on-board buttons, switches, etc. (will not be used)
- USB-C port programming and data transfer to and from the computer
- 20MHz+ clock speed
- 12+ io pins
- 2 or 3 3.3v DC voltage supplies

Type: DSP, Microcontroller, FPGA

- Digital Signal Processors are built for processing digital signals. They will perform better than general-purpose micontrollers and often have built in ADC/DAC.
- Microcontroller chips
- FPGA are Field Programmable Gate Arrays. They allow for people to program the hardware directly in VHDL or Verilog. These boards use lookup tables to distribute digital values. This option may produce the best performance.

Case 1: DSP/microcontroller with integrated ADC and DAC

+ Compact design
+ Convenient to use
- Cannot upgrade components for a higher sample rate, bitrate, signal to noise ratio, or bandwidth.
- DSP boards can have good integrated parts but few microcontrollers have both components.

  Recommendations:

  - Bella/Bella-mini:

    https://bela.io/products/

    - 16-bit ADC and DAC
    - Mid-cost ($188.26) or the mini ($151.62)
    - Browser-based IDE (I would need the board to actually see it)

      Also features an oscilloscope
    - 4GB of internal memory (that’s plenty for recording)
    - The Mini version is pretty compact
  - Mini DSP:

    https://www.minidsp.com/products/minidspkits/2-x-in-4-x-out

    - 24-bit ADC/DAC
    - 48 kHz sample rate
    - I2S ports
    - USB streaming
    - Low-cost ($80)

        - RCA connectors (we would need another component to convert to RCA)

Case 2: DSP/microcontroller with integrated ADC (no 16-bit DAC)

+ Semi-compact design
+ Convenient ADC
+ If the board has a PWM, it can be used to make a DAC
- Unbalanced design
- Cannot upgrade the ADC

Recommendations:

- Daisy: https://www.electro-smith.com/daisy/daisy
    - 16-bit ADC
    - Low-cost ($30)
    - 1 3.3v analog pin
    - Few integrated controls
    - USB data transfer
    - ARM Cortex-M7 MCU
    - Serial interfaces

Case 3: DSP/microcontroller with no ADC or DAC

+ Flexible design
+ Interchangeable parts
+ Good potential for high-quality parts
- May be difficult to use these parts depending on the interface provided.

Recommendations:

- AD1940: https://www.analog.com/media/en/technical-documentation/data-sheets/AD1940_1941.pdf
    - dsp
    - 192kHz sample rates
    - Expensive ($230)
    - Serial interfaces for controlling external ADC and DAC
- SHARC Audio Module: https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/sharc-audio-module.html#eb-overview
    - Dsp
    - USB connection
    - Rather large and doesn't seem to have an integrated ADC or DAC
    - Mid-cost ($195)

Case 4: FPGA with no ADC or DAC

+ Will have the best performance (operations are only limited to the clock)
+ Works well for controlling multiple clocks
- Requires hardware programming knowledge
- May be difficult to transport to the computer

Recommendations:

- Doppler: https://dadamachines.com/product/doppler/
    - Compatible with Arduino IDE
    - Both microcontroller and FPGA

  - Audio focused
  - Low-cost ($54.11)
  - Micro usb connection
  - ADC and DAC (not sure on their quality)
  - 1 3v supply

Other interesting kits I found

- TGA Pro: https://www.tindie.com/products/Blackaddr/arduino-teensy-guitar-audio-shield/
  - Don’t see much of any data sheet, might be a good idea to contact the manufacturer.
  - Made specifically for working with guitars (more for making a pedal though) If this kit has an ADC and DAC it would be an all-in-one setup
  - Low-cost ($63.20)
- https://www.arrow.com/en/products/adzs-audio-ex3/analog-devices?gclid=Cj0KCQjw6ar4BRDnARIsAITGzlBtxyuxHCpTS46ujO2azc5TYy8ADllkROaMjY6qLejiL-lUX4JdvkQaAmisEALw_wcB

Programming(On board):

This part of the project will depend on the type of board you choose and the programming language. Microcontrollers and DSPs can be programmed with many high-level languages but C/C++ is chosen most often. FPGAs can be programmed in any Hardware Definition Language such as VHDL or Verilog. Note that HDL languages are not software. They do not compile or execute code. Instead the program gives the board instructions on how to configure the digital hardware, thus the code behaves significantly differently from software.

Purpose:

To manage all of the hardware components previously mentioned and communicate with the software on the computer. The requirements are

Requirements:

- Receive messages from the computer through USB protocol using control transfer; to change states based on user input.
- When calibrating the guitar:
    - Record the highest voltage the guitar outputs and configure the DAC to output the voltage needed for the VGA stages to reach approximately 3.299 v (We can also record the max voltage, how loud the signal is, the bandwidth range)
    - Only allow calibration when a guitar is plugged in. It must send a message to tell the computer when a guitar is plugged in.
- When recording the guitar:
    - Record samples of at least 16-bits at a consistent 44.1kHz rate.
    - Transport these samples to the computer through USB protocol using Isochronous transfer.
    - Be able to restart the recording and erase the currently stored samples
    - Be able to pause the recording.
    - Be able to stop the recording.
- When recording is paused:
    - Stop receiving the guitar signal and transferring samples.

    - Be able to resume the recording cycle.
- When the recording is stopped:
    - Stop receiving the guitar signal and transferring samples.
    - Send and erase the currently stored samples.
    - Calculate the number needed to configure the attenuator using the DAC voltage. Send this value to the computer so it can be used when playing the signal later.
- Configuring playing the guitar
    - Configure the attenuator using the DAC to play a good approximation of the original guitar signal at the same voltage level as recorded.
- When playing the recording:
    - Download samples from the computer through USB protocol.
    - Play these 16-bit or larger samples at a consistent 44.1 kHz rate.
    - Be able to restart playing.
    - Be able to pause the playing.
    - Be able to stop the recording.
- When playing is paused:
    - Stop outputting the recording and receiving the samples from the computer.
    - Be able to resume the playing cycle.
- When playing is stopped:
    - Stop outputting the recording and receiving the samples from the computer.
    - Erase the currently held samples

    - Be able to play the recording from the start again

Recommendations:

- If the board you choose is a microcontroller or DSP: C is a good high-level systems programming language to use; it’s a common choice for embedded systems. C++ also works well for the same reasons but offers far more features.
- If the board you choose is FPGA: Verilog or VHDL can be used depending on preference. These Hardware Definition Languages may be hard to program correctly since they are not compiled software.
- The platform to program in depends on the board used. Some manufactures such as Texas Instruments and Xilinix have their own Eclipse-based IDE environments built to work with all of their boards. Some companies such as Arduino have simple browser-based IDEs. Any C/C++ IDE or text editor can be used but it needs the proper tools for embedded development.

Software(On Computer):

The computer-side software is currently set up. The setup uses the IDE QT to create both the GUI and back end programming in C++. (See appendix D for installation instructions). I chose QT because it's an all in one open source cross-platform IDE and GUI toolkit that fits the software requirements. QT provides many APIs and alternate methods to make a GUI centered application and good documentation. The only less documented area is Qmake. C++ is a convenient language to use since the board may be programmed in C.

Purpose: To communicate with the board through USB protocol so it can properly record the signal as the user commands. To store the recording and DAC value on the user's computer. To interface with the user so they can command the board through the software.

Requirements:

- Command the board to change states through USB protocol.
- Provide a GUI for recording that allows the user to
  - Start the calibration process
    - The GUI must Instruct the user how to calibrate the guitar for the system. Play the guitar with open strings as loud as you can.
    - Calibration can only work with the guitar plugged in.
    - Bit-rate and sample rate should be the default settings
  - Stop the calibration process; only when the process has started.
    - Records the DAC value needed to play the guitar later and the max voltage the guitar reached.
  - Change the sample size
    - Options: 8bit, 10bit, 12bit, 16bit (default), 24bit*, 32bit* (limited by the system ADC)
  - Change the sample rate
    - Options: minimal (2x the highest tone the, 32000 Hz, 44100 Hz (default), *48000 Hz, *96000 Hz, *192000 Hz
  - Enter a guitar name
    - Options: Enter any guitar name, the user should enter the brand and model of the guitar.

- Start the recording process
  - Sends a message to tell the board to start recording with the current recording information.
  - Disable the ability to calibrate the guitar and change the recording specifications.
  - Enable the ability to restart, pause, or stop the recording
  - Receive sample packets from the board.
  - Creates a WAV file to store the audio samples.
- Restart the recording process
  - Sends a message to tell the board to restart the recording with the same recording information.
  - Erase the currently stored recording.
- Pause the recording process
  - Send a message to the board to pause the recording.
  - Finnish recording the stored samples and stop recording new samples.
- Resume the recording process
  - Send a message to the board to resume the recording.
  - Resume recording attacks
- Stop the recording process
  - Send a message to the board to stop the recording.
  - Finnish recording the stored samples and stop recording new samples.
  - Enable saving the recording.

        - Enable configuring the guitar.
    - Save the recording and other values
        - Allow the user to change:
            - The location where the guitar recordings will be stored. (Default: the program directory ./Recordings)
            - The name of the recording (to make multiple recordings for the same guitar)
        - The saving process will combine the needed files into an archive folder
        - Default location is in the program directory.
- Provide a GUI for playing that allows the user to
    - Load any recordings made with the system
        - The GUI must display the sample size, sample rate, guitar name, and recording name for the current recording.
        - Send a message with the sample size, sample rate, and the DAC value to the board.
    - Play the recording
        - Send a message to the board to start playing the recording.
        - Send samples to the board through USB protocol.
        - Enable the ability to restart, pause, or stop the playing.
        - Disable the ability to load a recording.
    - Restart playing the recording
        - Send a message to the board to restart playing the recording and erase the currently held samples.

        - Restart sending samples to the board.
    - Pause playing the recording
        - Send a message to the board to finnish playing the current sample and pause playing the recording.
        - Stop sending samples to the board.
    - Resume playing the recording
        - Send a message to the board to start playing the recording again from the current position.
        - Start sending samples to the board.
    - Stop playing the recording
        - Send a message to the board to stop playing the recording and erase the currently held samples.
        - Stop sending samples to the board.
        - Enable the ability to load a recording.

Moving Forward

The next step is to start building the project hardware setup. The setup section provides most of the parts needed and many choices. Choose one of the possible hardware setups and look into the suggested parts. Of course there might be better parts than listed. After the hardware setup is built, start the device hardware/software coding (depending on the device). Research the procedure and practices for coding embedded hardware or FPGA and how to access specific parts of the board. Continue developing the computer-side part of the project with the downloaded file. Look more into C++, QT development, and any free-to-use libraries. Research guitar signals to learn how to properly calibrate any guitar. Finding the method to produce the highest possible voltage will take some experimentation.

What I learned

Before this project I knew very little about hardware. I have looked for devices and parts but I never knew what I wanted, and ADC was just an ADC. Through this project I learned how to look for what I want in hardware and understand some of the performance specs, pin layouts, and interfaces. I only understand a small group of devices and a few of their main features, but I have a good sense of how to look for parts. Given these parts I looked into recording an analog signal. I learned how ADCs are limited by their same size, sample rate, signal to noise ratio, and the process used to convert analog to digital. I learned the importance of looking at component bandwidth. If any component had lower bandwidth than I needed to record the signal, the component would not work for the setup. After the board receives the digital signal, I learned a bit how to send the signal to the computer. I learned how WAV files are organized in hex-bit form so I can place samples into them. On the computer-side of the project I learned how to decide on a software setup. Overall I mainly gained surface-level knowledge but enough to start the project work.

Resources

[1] "Microsoft WAVE soundfile format." http://soundfile.sapp.org/doc/WaveFormat/ (accessed Jun. 14, 2020).

[2] "USB Connectivity for MCUs: Which is Right for Your Next Design? | DigiKey." https://www.digikey.com/en/articles/usb-connectivity-for-mcus-which-is-right-for-your-next-design (accessed Jun. 14, 2020).

[3] "Basic Electric Guitar Circuits 1: Pickups | Amplified Parts." https://www.amplifiedparts.com/tech-articles/basic-electric-guitar-circuits-1-pickups (accessed Jul. 13, 2020).

[4] Tomsguitarprojects, "Tom's Guitar Projects: Electric Guitar Output Voltage Levels," *Tom's Guitar Projects*, Dec. 31, 2014. http://tomsguitarprojects.blogspot.com/2014/12/electric-guitar-output-voltage-levels.html (accessed Jul. 11, 2020).

[5] *What does your guitar signal looks like?* 2013.

[6] A. S. Nastase, "An ADC and DAC Least Significant Bit (LSB)," *Mastering Electronics Design*. https://masteringelectronicsdesign.com/an-adc-and-dac-least-significant-bit-lsb/ (accessed Jul. 13, 2020).

[7] "Amplifier Gain and Decibels." https://learnabout-electronics.org/Amplifiers/amplifiers13.php (accessed Jul. 13, 2020).

[8] "Sample Rates - Audacity Manual." https://manual.audacityteam.org/man/sample_rates.html (accessed Jul. 13, 2020).

Appendix A - Wav file header

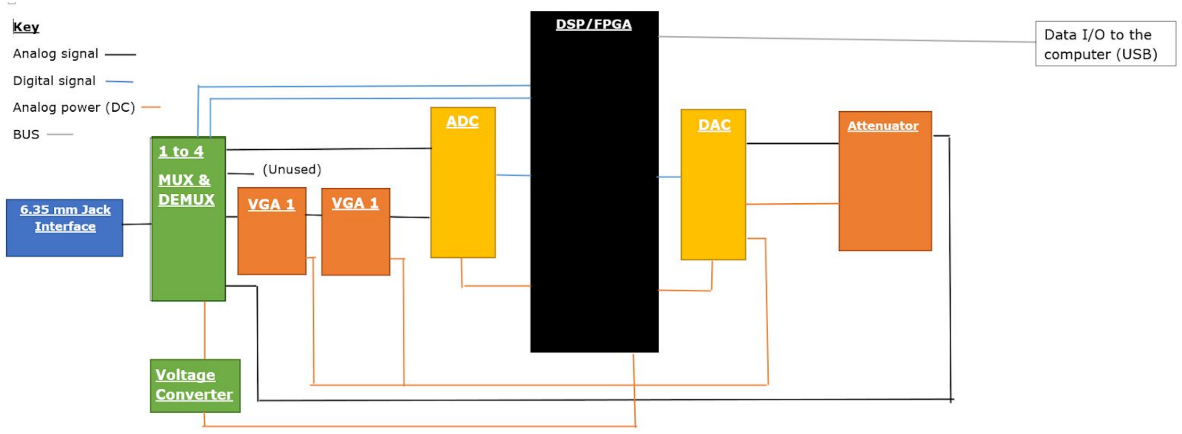
This is a C-style header, it may not be needed for the software side of the project so it is here instead of the project.

```
// wav file structs
//canonical WAV format header
typedef struct header
{
    const uint32_t chunkId;         //The letters "RIFF", big endian
    uint32_t chunkSize;             //36*data_subchunk.subchunk_size, little endian
    const uint32_t format;          //The letters "WAVE", big endian
} HEADER_DEFAULT = {0x52494646, 0x0, 0X57415645};

//format subchunk
uint32_t SAMPLE_RATE =      0x44AC0000
uint16_t BITS_PER_SAMPLE =  0X1000
typedef struct fmt_subchunk
{
    const uint32_t subchunkId;      //The letters "FMT", big endian
    const uint32_t subchunkSize;    //16 for PCM, Size of the rest of this subchunk,
little endian
    const uint16_t audioFormat;     //PCM = 1, little endian
    uint16_t numChannels;            //Number of channels, mono = 1, stereo = 2,
little endian
    uint32_t sampleRate;             //Sample rate in Hz, default = 44.1kHz, little
endian
    uint32_t byteRate;               //(sampleRate*numChannels*bitsPerSample)/8
    uint16_t blockAlign;             //(numChannels*bitsPerSample)/8, number of bytes
for one sample, little endian
    uint16_t bitsPerSample;          //bits per sample, default = 16 bits, little endian
} FMT_SUBCHUNK_DEFAULT = {0X666d7420, 0X10000000, 0x0100, 0x0100,
0x44AC0000, 0x88580100, 0x0200, 0X1000};

// custom type for a sample
typedef struct sample{
    void* smallestSample;            //The smallest sample allowed is a byte, marked
as void for a non-specific size
};

//data subchunk
typedef struct data_subchunk
{
    const uint32_t subchunkId;       //The letters "DATA", big endian
    uint32_t subchunkSize;           //(numSamples*numChannels*bitsPerSample)/8,
little endian
```

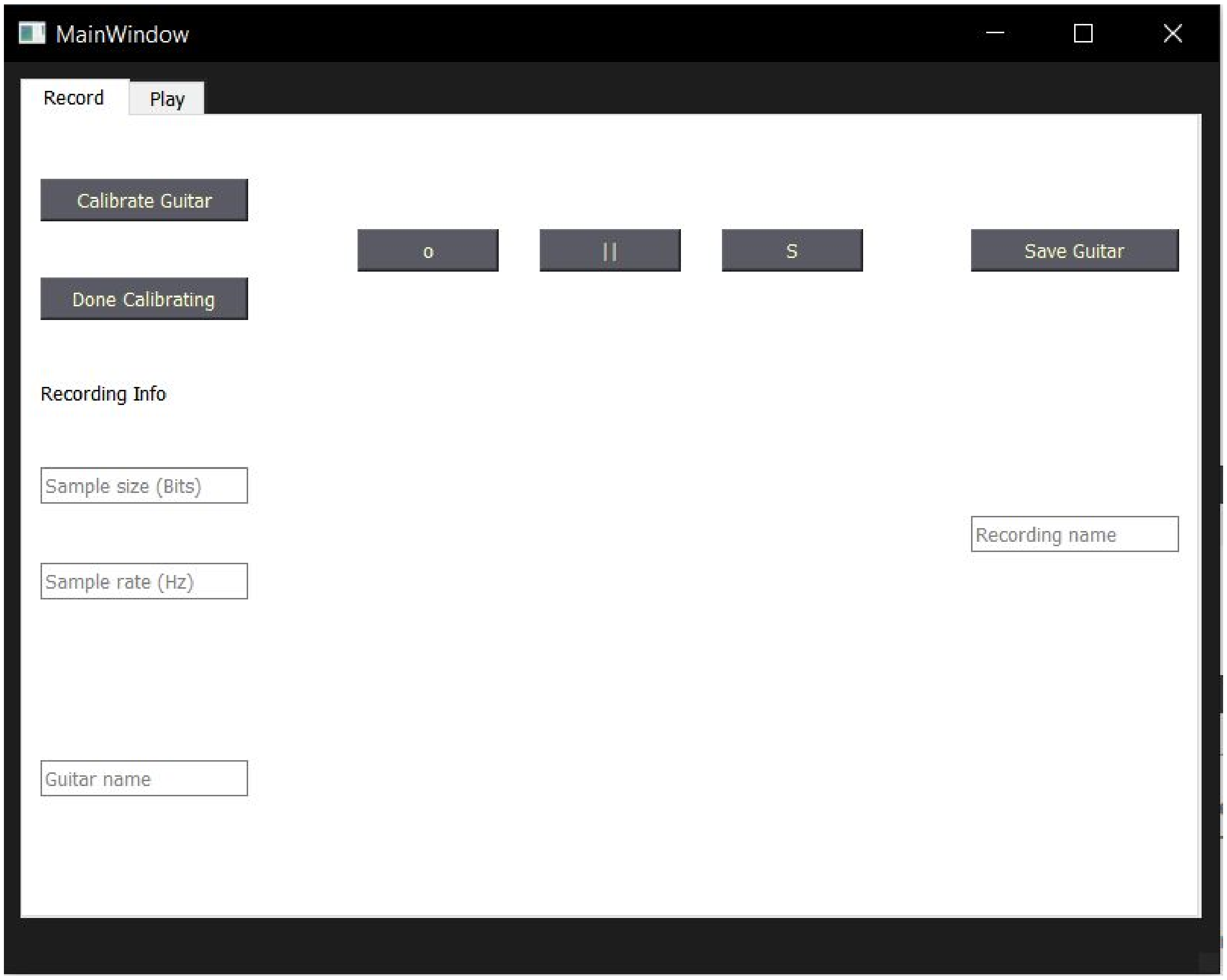
```
    sample *SAMPLE_DATA[];          //The sample data, each sub_sample is a byte,
little endian
} DATA_SUBCHUNK_DEFAULT = {0X64617461, 0x00000000, 0x0};
```

Appendix B - Block diagram

This block diagram contains most of the needed hardware components. This setup is not a full schematic. It includes some of the necessary connections and no real components.

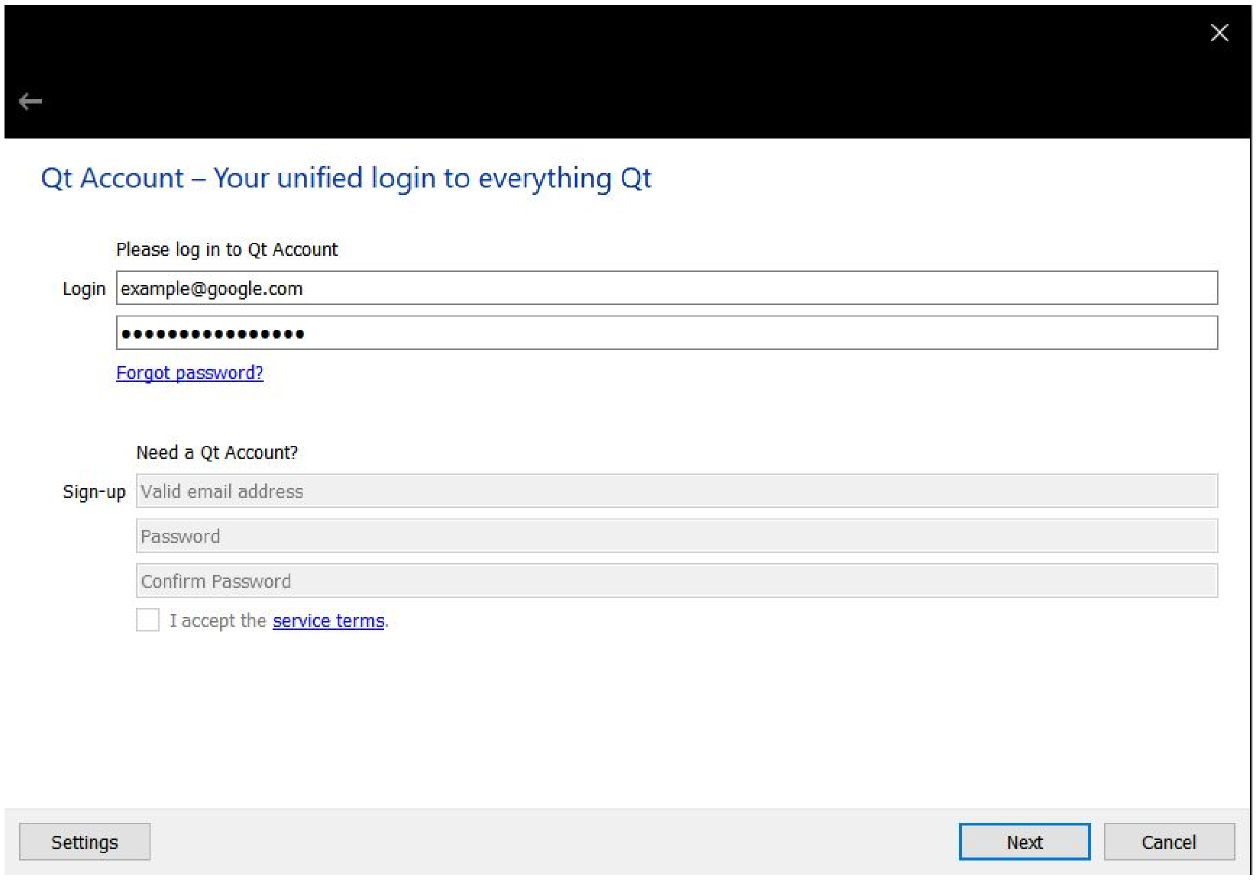


Appendix C - GUI



Appendix D - QT Installation instructions

1. Use the installer for QT version 2.3.2 for your operating system found in the repository under ./Setup/Installers/QT This installer is for the open source version of QT. The instructions will show the windows version of the installer.
2. Create a QT account



Qt Account – Your unified login to everything Qt

Please log in to Qt Account

Login: example@google.com

●●●●●●●●●●●●●●●●

Forgot password?

Need a Qt Account?

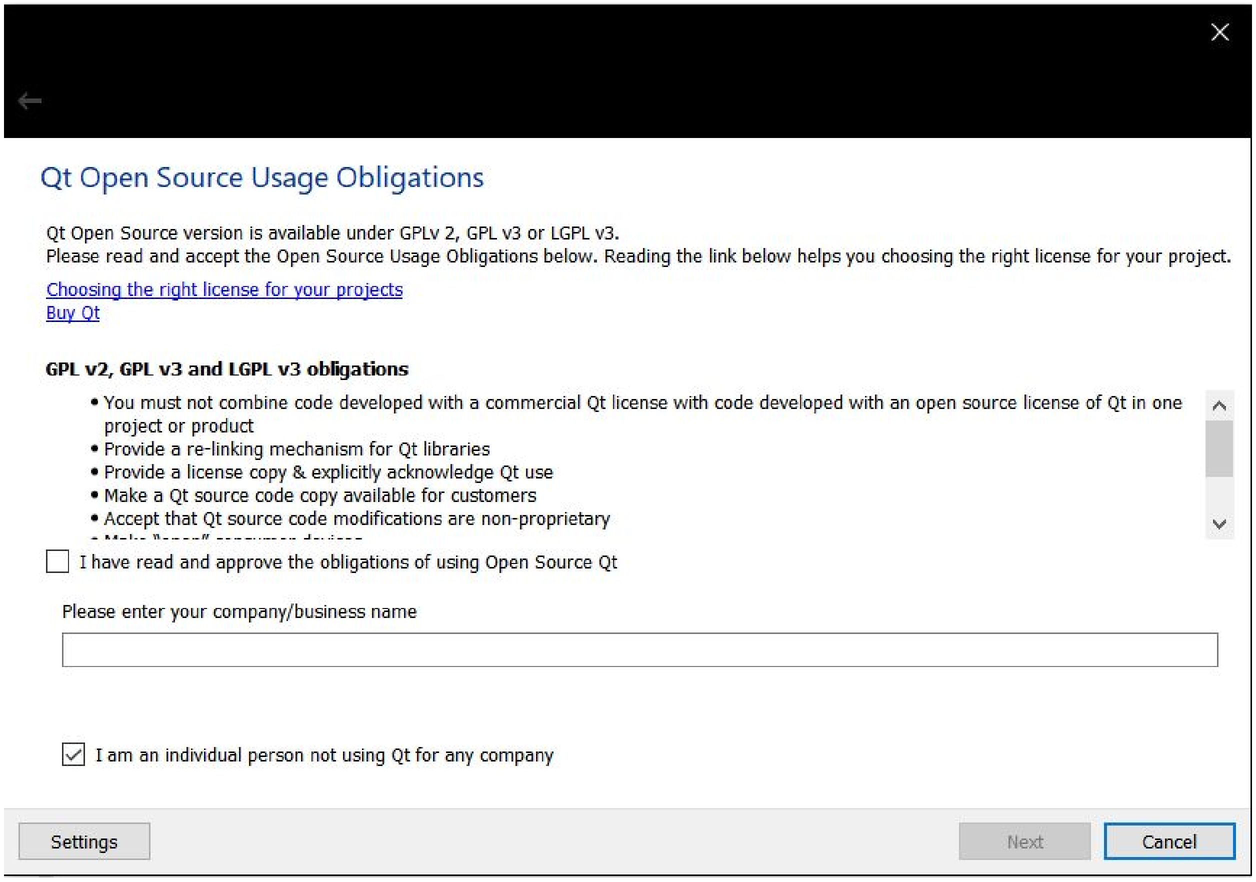
Sign-up: Valid email address

Password

Confirm Password

I accept the service terms.

3. Read the license agreement and check "I am an individual person not using QT for any company"



Qt Open Source Usage Obligations

Qt Open Source version is available under GPLv 2, GPL v3 or LGPL v3.
Please read and accept the Open Source Usage Obligations below. Reading the link below helps you choosing the right license for your project.

Choosing the right license for your projects
Buy Qt

GPL v2, GPL v3 and LGPL v3 obligations

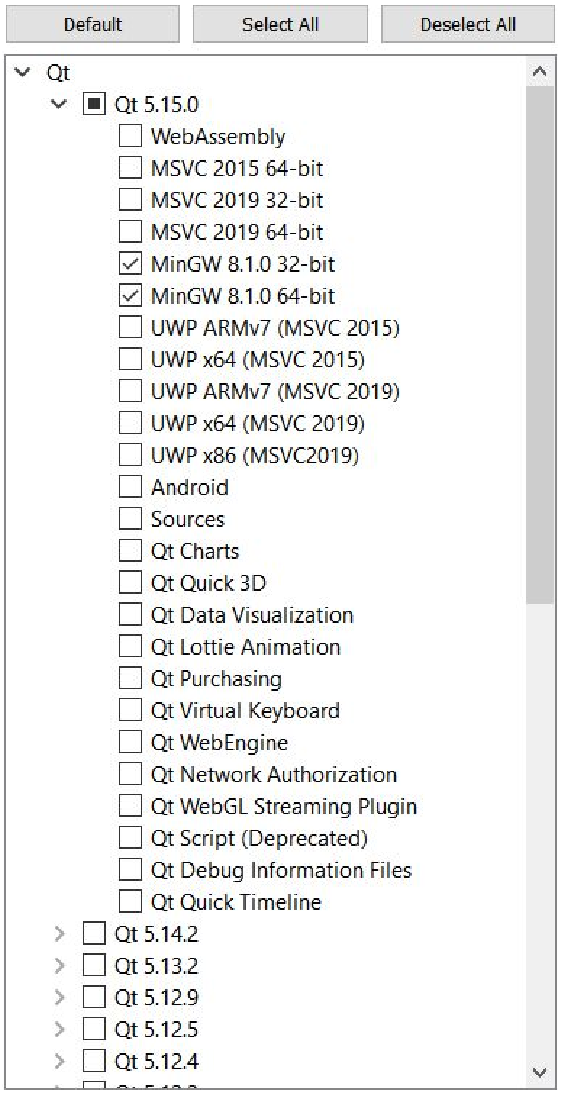
- You must not combine code developed with a commercial Qt license with code developed with an open source license of Qt in one project or product
- Provide a re-linking mechanism for Qt libraries
- Provide a license copy & explicitly acknowledge Qt use
- Make a Qt source code copy available for customers
- Accept that Qt source code modifications are non-proprietary

☐ I have read and approve the obligations of using Open Source Qt

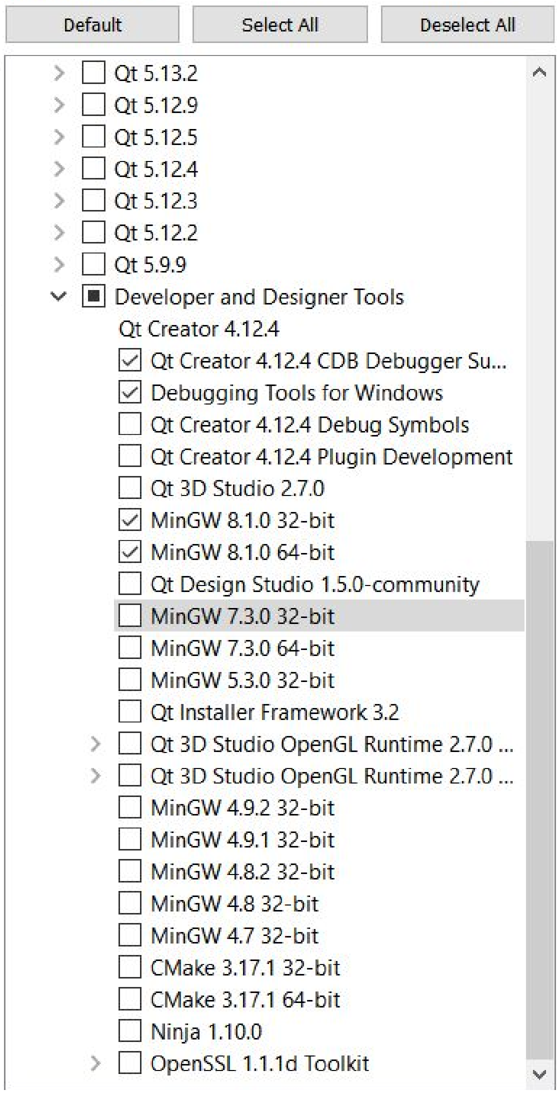
Please enter your company/business name

☑ I am an individual person not using Qt for any company

4. Install the following components.



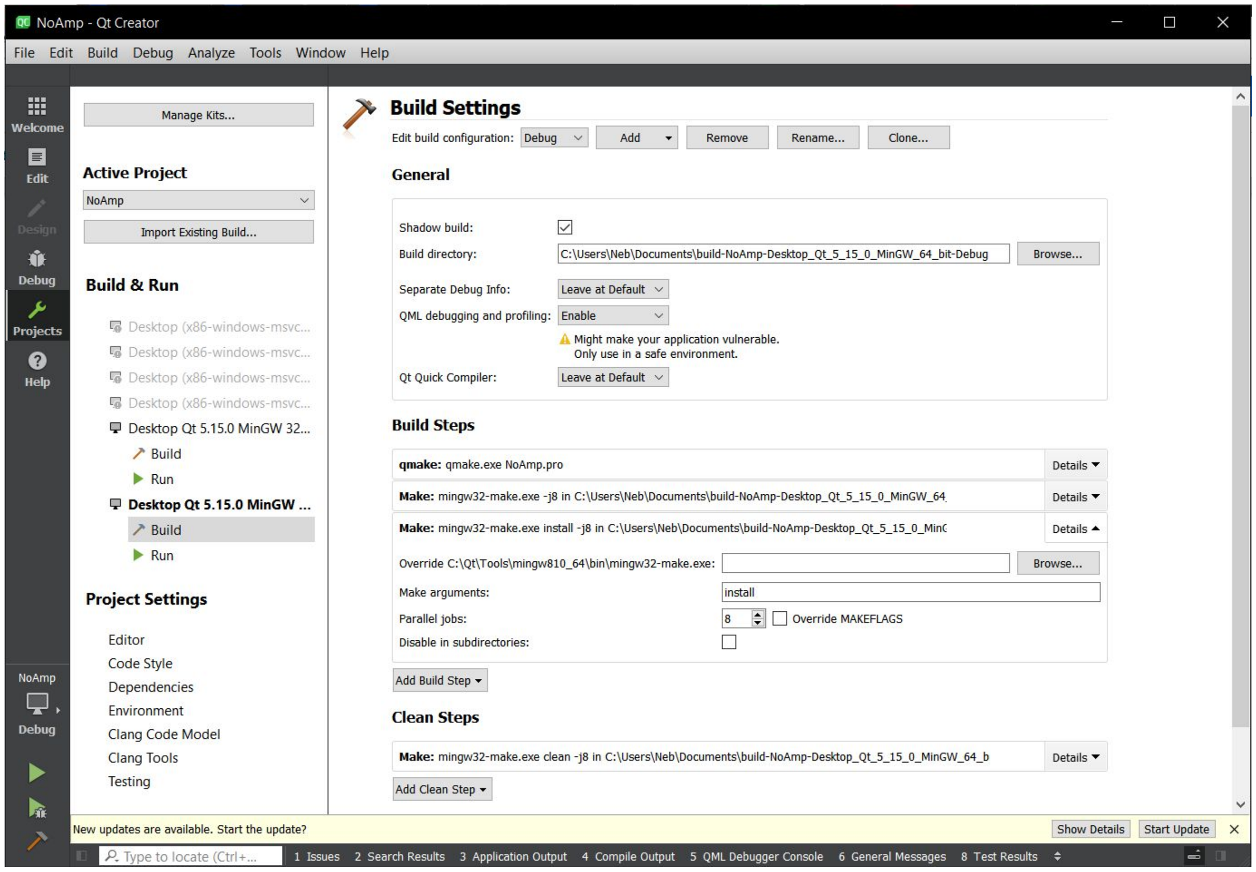
- Qt
  - ■ Qt 5.15.0
    - [ ] WebAssembly
    - [ ] MSVC 2015 64-bit
    - [ ] MSVC 2019 32-bit
    - [ ] MSVC 2019 64-bit
    - [✓] MinGW 8.1.0 32-bit
    - [✓] MinGW 8.1.0 64-bit
    - [ ] UWP ARMv7 (MSVC 2015)
    - [ ] UWP x64 (MSVC 2015)
    - [ ] UWP ARMv7 (MSVC 2019)
    - [ ] UWP x64 (MSVC 2019)
    - [ ] UWP x86 (MSVC2019)
    - [ ] Android
    - [ ] Sources
    - [ ] Qt Charts
    - [ ] Qt Quick 3D
    - [ ] Qt Data Visualization
    - [ ] Qt Lottie Animation
    - [ ] Qt Purchasing
    - [ ] Qt Virtual Keyboard
    - [ ] Qt WebEngine
    - [ ] Qt Network Authorization
    - [ ] Qt WebGL Streaming Plugin
    - [ ] Qt Script (Deprecated)
    - [ ] Qt Debug Information Files
    - [ ] Qt Quick Timeline
  - [ ] Qt 5.14.2
  - [ ] Qt 5.13.2
  - [ ] Qt 5.12.9
  - [ ] Qt 5.12.5
  - [ ] Qt 5.12.4



- [ ] Qt 5.13.2
- [ ] Qt 5.12.9
- [ ] Qt 5.12.5
- [ ] Qt 5.12.4
- [ ] Qt 5.12.3
- [ ] Qt 5.12.2
- [ ] Qt 5.9.9
- ■ Developer and Designer Tools
  - Qt Creator 4.12.4
  - [✓] Qt Creator 4.12.4 CDB Debugger Su...
  - [✓] Debugging Tools for Windows
  - [ ] Qt Creator 4.12.4 Debug Symbols
  - [ ] Qt Creator 4.12.4 Plugin Development
  - [ ] Qt 3D Studio 2.7.0
  - [✓] MinGW 8.1.0 32-bit
  - [✓] MinGW 8.1.0 64-bit
  - [ ] Qt Design Studio 1.5.0-community
  - [ ] MinGW 7.3.0 32-bit
  - [ ] MinGW 7.3.0 64-bit
  - [ ] MinGW 5.3.0 32-bit
  - [ ] Qt Installer Framework 3.2
  - [ ] Qt 3D Studio OpenGL Runtime 2.7.0 ...
  - [ ] Qt 3D Studio OpenGL Runtime 2.7.0 ...
  - [ ] MinGW 4.9.2 32-bit
  - [ ] MinGW 4.9.1 32-bit
  - [ ] MinGW 4.8.2 32-bit
  - [ ] MinGW 4.8 32-bit
  - [ ] MinGW 4.7 32-bit
  - [ ] CMake 3.17.1 32-bit
  - [ ] CMake 3.17.1 64-bit
  - [ ] Ninja 1.10.0
  - [ ] OpenSSL 1.1.1d Toolkit

5. Proceed to installation and let it finnish
6. Once QT is done go to Projects

7. For all building procedures add a step to make install. This step is needed by the project to copy certain files from the project directory (such as the style sheet) to the build directory so they can be used when running the program.



Active Project

NoAmp

Build & Run

- Desktop (x86-windows-msvc...
- Desktop (x86-windows-msvc...
- Desktop (x86-windows-msvc...
- Desktop (x86-windows-msvc...
- Desktop Qt 5.15.0 MinGW 32...
  - Build
  - Run
- **Desktop Qt 5.15.0 MinGW ...**
  - Build
  - Run

Project Settings

- Editor
- Code Style
- Dependencies
- Environment
- Clang Code Model
- Clang Tools
- Testing

NoAmp
Debug

Build Settings

Edit build configuration: Debug

General

Shadow build: ✓
Build directory: C:\Users\Neb\Documents\build-NoAmp-Desktop_Qt_5_15_0_MinGW_64_bit-Debug
Separate Debug Info: Leave at Default
QML debugging and profiling: Enable
Might make your application vulnerable.
Only use in a safe environment.
Qt Quick Compiler: Leave at Default

Build Steps

qmake: qmake.exe NoAmp.pro
Make: mingw32-make.exe -j8 in C:\Users\Neb\Documents\build-NoAmp-Desktop_Qt_5_15_0_MinGW_64
Make: mingw32-make.exe install -j8 in C:\Users\Neb\Documents\build-NoAmp-Desktop_Qt_5_15_0_MinC

Override C:\Qt\Tools\mingw810_64\bin\mingw32-make.exe:
Make arguments: install
Parallel jobs: 8 ☐ Override MAKEFLAGS
Disable in subdirectories: ☐

Clean Steps

Make: mingw32-make.exe clean -j8 in C:\Users\Neb\Documents\build-NoAmp-Desktop_Qt_5_15_0_MinGW_64_b

New updates are available. Start the update?