Software Formalization

Year: 2024 Semester: Spring Team: 12 Project: Microphone Interface
Creation Date: 2/17/2024 **Last Modified:** September 15, 2024
Author: Liam Roach **Email: leroach@purdue.edu**

Assignment Evaluation: See Rubric on Brightspace Assignment

1.0 Utilization of Third Party Software

In this project we are using three pieces of third-party software. The first being the software provided in the STM32Cube IDE, which includes the HAL drives and the TouchGFX library. We are using this software for building and providing our graphical user interface, interrupts, DMA, I2C, SAI, and SPI.

The second is an ILI9341 driver[1] found on GitHub that we are using to provide a layer of abstraction between our code and the controller on our LCD. The driver is licensed under the MIT license.

The third is a programming language called Faust[2] which is specifically made for digital signal processing and can be compiled to various languages including C and C++. We will be using it to write out DSP functions in this project. Faust is licensed under the GNU LGPL version 2.1.

2.0 Description of Software Components

Our software has four major components: an audio data buffer, a DSP parameter data structure, DSP functions, and a graphical user interface.

The audio data buffer is an array of 16-bit audio samples. The size of the array will be determined based on testing so that audio quality is maximized whilst delay is minimized. For the sake of keeping processed and unprocessed data separate, there will be two buffers: one for incoming audio from the codec, and one for processed audio that will be sent back to the codec. The data in these buffers will be transferred from and to the codec via circular DMA. As soon as a sample is transferred into the input buffer, an interrupt will kick off processing of that sample, which will then be written to the output buffer.

The DSP parameter data structure will contain a value for each user-controlled parameter of each DSP function: frequency, width, and gain for each band of the five band EQ, level for the distortion, and time and feedback for the delay. Each parameter will be stored as an 8-bit integer and will range from zero to one hundred. All parameters will be initialized at zero, except for the frequency parameters, which will be spaced out evenly, and gain, which will be initialized to one hundred. Each time a rotary encoder is turned, an interrupt will trigger that will update the values in this structure. The DSP functions will directly access this structure to obtain the relevant parameters each time they are called.

There are three DSP functions that will be called each time a sample is read from the codec: EQ, delay, and distortion. The output from each effect will be fed to the input of the next in that order, and the last output will be written to the output buffer. The EQ function will implement a standard digital biquad filter[3], with a bandpass for each band of the EQ. The delay function will add the sample at i – time multiplied by feedback to the current sample. The distortion function will amplify the incoming sample by an amount given by level, and cap it to a maximum level if it goes above that level.

Finally, the graphical user interface, which will be built using the TouchGFX library, will display each effect parameter on a series of pages. There will be a page for each band of the EQ and the delay and distortion will split a page. Only the parameters on the page currently being displayed will be able to be modified via the rotary encoders. Which page is being selected can be changed via the two push buttons; one button will navigate forwards, and one button will navigate backwards. Each time a button is pressed to change a page or an encoder is turned to change a parameter, and interrupt will trigger to redraw the display to reflect the change.

3.0 Testing Plan

For the sake of debugging initialization, our board will feature a few status LEDs, which be used to check if each stage of initialization completes successfully. Once the initialization is complete, the LCD screen can be used for debugging purposes.

To test the audio data buffer we will pass the audio directly to the output without any DSP and display the waveform on the screen and check that it matches what we expect based on the input, and check that the waveform of the output matches what is stored in the buffer. To test the DSP parameter data structure we will display all of the values in the structure on the LCD screen and we will move the rotary encoders and check if they update as expected. To test the EQ portion of the DSP functions, we will pass in a signal with a few prominent frequencies and check, via FFT on an oscilloscope, that the function is attenuating frequencies as expected based on the parameters. Delay will be tested by passing in a short burst of sound, and using an oscilloscope to see if a decaying echo of that pulse is present on the output. Finally, distortion will be tested by passing in a signal, and using an oscilloscope to check that the output is an amplified version of that signal with the higher amplitude portions clamped to a certain level. The last component, the GUI, will be tested similarly to the way that the DSP parameter data structure will be tested, but with the GUI being displayed the whole time.

4.0 Sources Cited:

[1] dtnghia2206, “STM32_Peripherals,” GitHub, https://github.com/dtnghia2206/STM32_Peripherals/tree/f15e568eee03854cc8ee80febeece6df16a1b81c (accessed Feb. 17, 2024).

[2] Grame-Cncm, “Faust - Programming Language for Audio Applications and Plugins,” GitHub, https://github.com/grame-cncm/faust (accessed Feb. 17, 2024).

[3] “Digital biquad filter,” Wikipedia, https://en.wikipedia.org/wiki/Digital_biquad_filter (accessed Feb. 17, 2024).

Appendix 1: Software Component Diagram

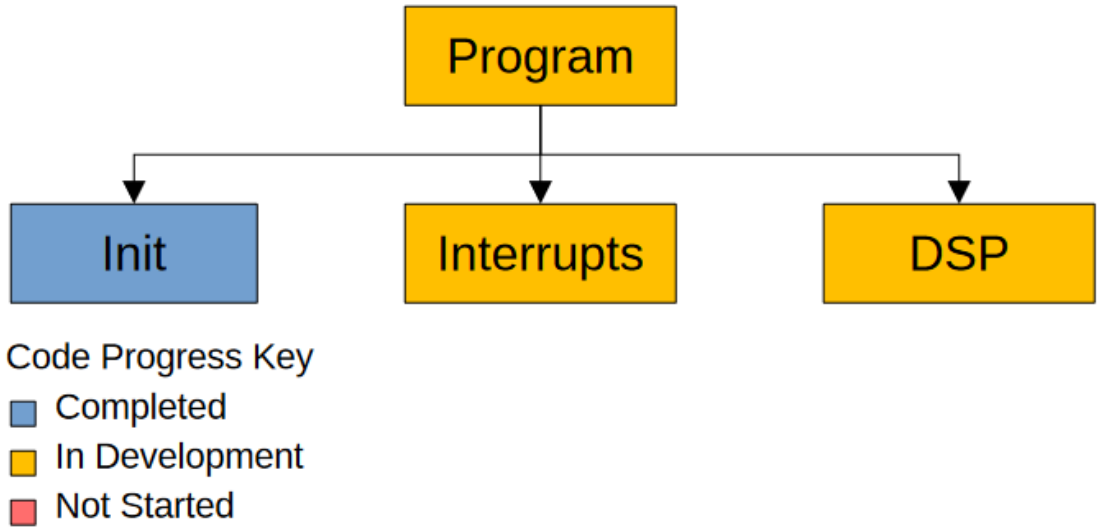

Figure 1. Overall Structure

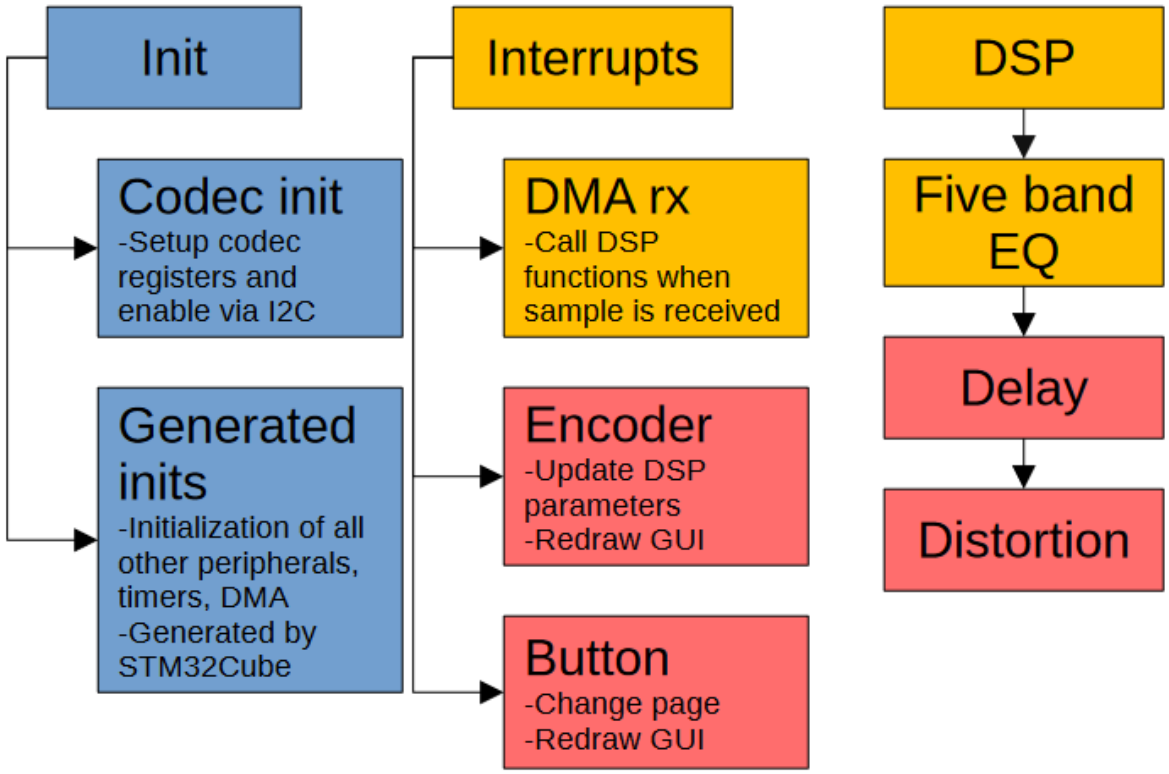

Figure 2. Initialization, Interrupts, and DSP Structure

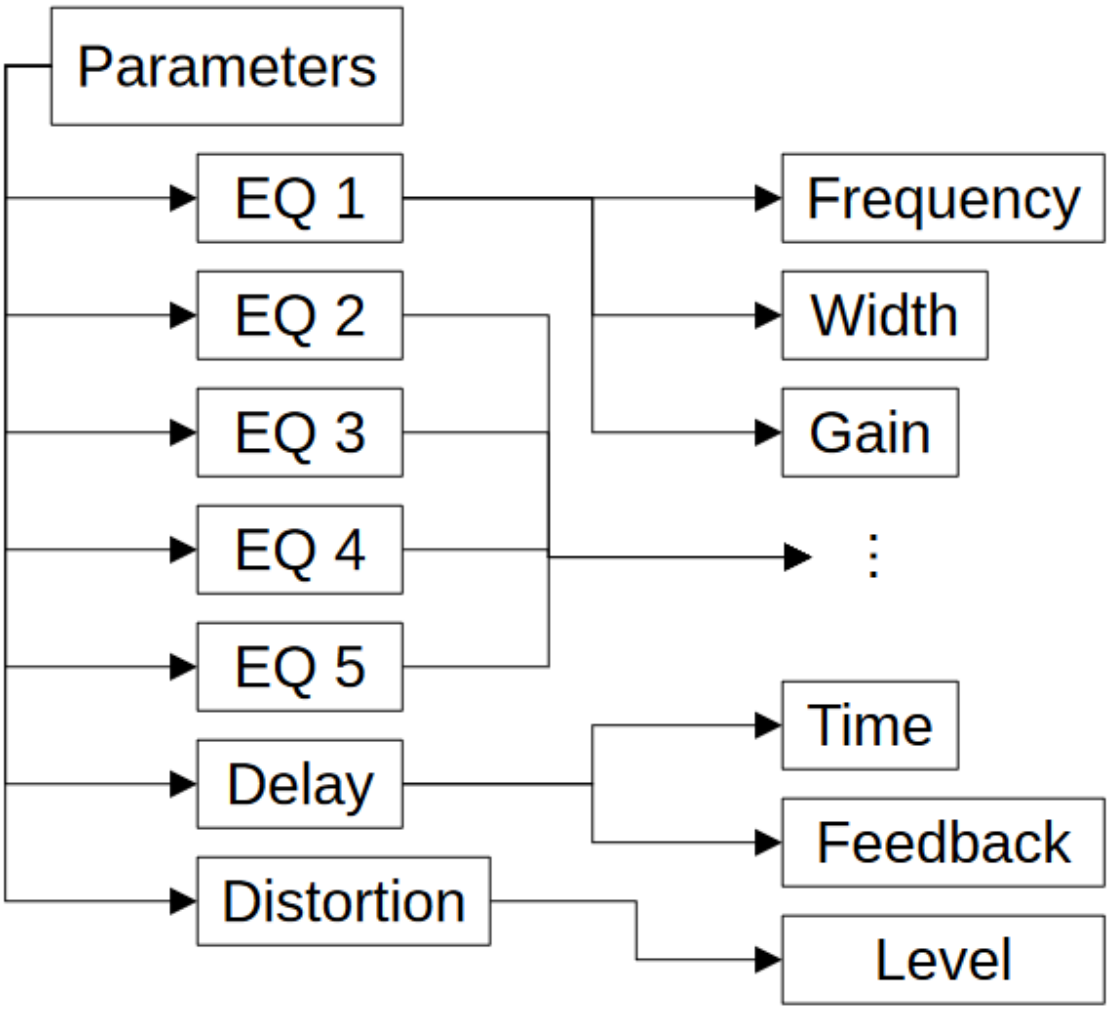

Figure 3. DSP Parameters Data Structure