Software Formalization

Year: 2020
Semester: Spring
Team: 8
Project AudioBeamer
Creation Date: 2/25/2020
Last Modified: 2/28/2020
Author: Calvin Walter Heintzelman **Email: cheintze@purdue.edu**

Assignment Evaluation:

Item	Score (0-5)	Weight	Points	Notes
Assignment-Specific Items				
Third Party Software		x2		
Description of Components		X3		
Testing Plan		x3		
Software Component Diagram		x4		
Writing-Specific Items				
Spelling and Grammar		x2		
Formatting and Citations		x1		
Figures and Graphs		x2		
Technical Writing Style		x3		
Total Score				

5: Excellent 4: Good 3: Acceptable 2: Poor 1: Very Poor 0: Not attempted

General Comments:

1.0 Utilization of Third Party Software

The audio codec, the Bluetooth module, and the PMIC will not use any outside libraries and will be programmed through the MCU. We will program the codec using a 3-wire serial interface for mode control, and we will use SPI to implement the 2 I2S interfaces needed for digital in and digital out (form codec's perspective). We will program the Bluetooth module via UART. Lastly, we will program the PMIC via I2C.

Android App

Name	License	Description	Use
android.bluetooth	Apache 2.0	"Provides classes that manage Bluetooth functionality, such as scanning for devices, connecting with devices, and managing data transfer between devices. The Bluetooth API supports both "Classic Bluetooth" and Bluetooth Low Energy." [1]	We will be using this library to communicate to the RN4020 chip via low energy Bluetooth in our app.
android.widget	Apache 2.0	"The widget package contains (mostly visual) UI elements to use on your Application screen. You can also design your own." [2]	We will be using this library to create GUI elements in our app, including text, sliders, and buttons.

MCU

Name	License	Description	Use
MSP DSP library	TSPA	"The Texas Instruments® Digital Signal Processing library (DSPLib) is an optimized set of functions to perform common fixed point signal processing operations on MSP430 and MSP432 devices. Applications where processing intensive filters and transforms are needed can leverage the DSPLib to run in real time with minimal energy consumption for ultra low-power applications." [3]	We will be using this library for most (if not all) DSP functions. These will include the computations behind equalization, reverb, distortion, etc.

2.0 Description of Software Components

Android App

The Android application code is used to send slider data from the user to the Bluetooth module via low energy Bluetooth. It is written in Java using Android Studio Code. The product user can connect to the Bluetooth module and alter audio effects by changing sliders in the app. These effects include, but are not limited to, equalization, reverb, and distortion. The app has been developed completely by the team using the help of Android Studio Code documentation. We are using a Pixel 3a to run this application.

Our app code has the following four major functions:

1. Initialize the app and set up the GUI
2. Set up Bluetooth on the phone
3. Scan for the Bluetooth module to connect and bind to it
4. Send packages to the Bluetooth module whenever slider values are changed

Audio Codec

The codec will be used to convert analog signals from the guitar to digital signals needed as input for our DSP functionality. The codec will then convert the output of our DSP into analog signals which are used by the speaker to output sound. The codec is controlled by sending commands via a 3-wire serial interface (data line, clock, and strobe pulse). These commands are used to set the 4 registers that control the operations of the codec. The MSP430 (MCU) does not have native I2S functionality so we will use a single SPI interface to simulate 2 I2S interfaces needed to carry the digital signals to and from the MCU and the codec.

The audio codec does not truly have functions since we will be only be writing to registers, however part of its communication with the MCU will be implemented in software associated with the MCU. This involves setting up a communication link, sending data, and receiving data between the MCU and the codec.

Bluetooth Module

The Bluetooth module is used to connect our Android app (via low energy Bluetooth) to our MCU (via UART). The software for this module is very rudimentary and is programmed through UART.

The Bluetooth module code has the following four major functions:

1. Reset itself to its default settings
2. Unbind from any previous Bluetooth devices
3. Disconnect from any previous Bluetooth connections
4. Rename itself to “audiobeamer” so our Android app can recognize it

Note that this Bluetooth module searches for devices automatically

MCU

The software in our MCU is primarily used for digital signal processing and to send signals where they need to go when they are received. The MCU is connected to the Bluetooth module via UART, the audio codec via SPI, and the PMIC via I2C. Our team is using the MSP DSP library to help with creating audio effects, such as equalization, reverb, and distortion.

The MCU has the following eleven major functions:

1. Send set up commands/signals to the Bluetooth module
2. Send set up commands/signals to the audio codec
3. Send set up commands/signals to the PMIC
4. Receive data from the Bluetooth module for DSP specifications for equalization, reverb, and distortion.
5. Receive the audio signal from the audio codec
6. Receive the battery/voltage level value from the PMIC
7. Display battery/voltage level to the user through an LED UI.
8. Apply an equalization effect to the audio signal received by the audio codec (or an updated audio signal returned by functionalities listed in 9 and/or 10) based on the specification sent by the Bluetooth module
9. Apply a reverb effect to the audio signal received by the audio codec (or an updated audio signal returned by functionalities listed in 8 and/or 10) based on the specification sent by the Bluetooth module
10. Apply a distortion effect to the audio signal received by the audio codec (or an updated audio signal returned by functionalities listed in 8 and/or 9) based on the specification sent by the Bluetooth module
11. Send the fully updated audio signal to the audio codec as a final output

Note that additional audio effects may be added as desired and those functionalities will be like the functionalities listed in 8, 9, and 10.

PMIC

The software in our MCU needs to simply read battery level/voltage and send the information to the MCU via I2C. Programming for this device is relatively minimal and is done through the I2C channel.

The PMIC code has the following two major functions:

1. Initial set up to read the battery level/voltage
2. Send the battery level/voltage value to the MCU when requested

3.0 Testing Plan

Android App – Priority 2

Testing this component will be running the app on a Pixel 3a to check to see if GUI elements are visually functional and appropriate. We will also need to connect to the Bluetooth module which will be tested by printing a success statement on the app when it connects and binds to our Bluetooth module. Lastly, our app needs to be able to send information from a slider to the Bluetooth module. Since we do not want to waste battery life with an acknowledgement, this test will be done on the Bluetooth module's end by displaying the information received from the app via UART to a computer terminal during the testing stage.

Audio Codec – Priority 4

This module will be tested by checking its ability to take input and output audio signals. Initially, we will simply test its ability to output an unchanged input signal. This signal will be received by the audio codec, sent to the MCU, and then outputted through a jack to some amp. This test is considered successfully if we can send a simple note from a guitar to the amp without any modification. We will then work up from there, sending chords and short songs. To verify the signals are the same, we will read both the input and output signals with an oscilloscope.

Bluetooth Module – Priority 3

This module will be tested by checking its ability to set up correctly and its ability to communicate with the Android app and the MCU. To check that this device is set up correctly, we will see if it can be recognized through Bluetooth devices (such as our cell phones) with its correct name, "audiobeamer". Once this test is completed, we will test this module's ability to communicate with Android app by printing any slider values to a computer terminal via UART. Once the Bluetooth module is successfully sending signals through UART and is sending the correct slider values from the Android app, we will then test its ability to communicate with the MCU. We will connect this module to the MCU via UART and check that it is sending the correct values from the slider by reading the signal with an oscilloscope. This test is considered completed when the signal outputted to the computer terminal is the same as the signal sent to the MCU and the value sent correctly matches the slider value on the Android app. This module's default settings do most of what we need it to do, so there is fairly little programming involved.

MCU – Priority 1 (Highest)

Testing this component will likely take the longest because it has the most functionalities and communicates with three other components. We will test the MCU's ability to communicate with the audio codec, the Bluetooth module, and the PMIC by going into the debug mode in our IDE and checking if the correct values are being stored in the correct registers/variables. For example, for the Bluetooth module, we will send a slider value from the Android app to the Bluetooth module and then from the Bluetooth module to the MCU. If the value read by the MCU (which we will check through the debugger in our IDE) is the same as the value sent by the Android app, then we know that the MCU is successfully communicating with the Bluetooth module. Likewise, if the voltage level read by the MCU is the same as the voltage level read from the battery (using an oscilloscope or voltmeter), then we know that the MCU is correctly communicating with the PMIC. Lastly, to test if the MCU is correctly communicating with the Audio codec, we will send an audio signal from the audio codec through the MCU and send it

back to output it. If the output signal is the same as the input signal, then we will know that the MCU is correctly communicating with the audio codec. This method of testing is effective because it tests for multiple functionalities with each device: the ability for the MCU to send the correct set-up commands/signals and the ability for the MCU to receive meaningful data from each device. Note that we can only move on to higher level testing after these functionalities have been successfully tested and implemented.

To test if the MCU can accurately display the battery level/voltage from the PMIC, we will turn on a specific LED based on the battery level/voltage. This is somewhat subject to change (the color of the LED, the number of LEDs, etc.), however we will test this by measuring the voltage level across the battery using a voltmeter or oscilloscope and checking if the correct LED is lit up based on the voltage level. To speed up testing, we will use a voltage source to simulate voltage signals to the MCU to test for correct LED functionalities.

We will test our functions for equalization, reverb, distortion, and any other audio effects only after we are successfully able to communicate with the audio codec and Bluetooth module. We will test these functions by sending the audio codec a simple audio signal (probably just a single sustained note) and checking if it can be outputted by the MCU with any changes. Once that works, we will change an audio effect value from a slider on our Android app. We will then send the same simple audio signal and check if the desire effect is then outputted through the MCU to the audio codec and then finally through a jack and to an amp. We will repeat this process for many different slider values for each audio effect to assure the effect works for all slider values for a simple signal. We will then repeat these tests with gradually more complicated signals. We will likely start with a note, move to a chord, then to several chords, and then to a full song. Once we create the desired output signal from every input signal, we will consider this test completed.

PMIC – Priority 5 (Lowest)

We will need to test this component for two main functionalities: accurately measuring voltage levels and sending that data to the MCU via I2C. Much of the testing will be done on the MCU's end, as mentioned previously, however some testing can be done in isolation with this component. We will check if the MCU has correctly programmed this device via I2C by checking the PMIC's output on an oscilloscope and check if the value correctly corresponds to our batteries' voltage levels, measured using the same oscilloscope or a voltmeter. Similar to the Bluetooth module, this component hardly needs any programming, so this will likely take very little time. Furthermore, this component's functionality does not impair on our project's basic function (creating custom audio effects). For these two reasons, testing this component has the lowest testing priority.

4.0 Sources Cited:

1. "android.bluetooth," December 2019. [Online]. Available: https://developer.android.com/reference/android/bluetooth/package-summary [Accessed February 28, 2020].
2. "android.widget," February 2020. [Online]. Available: https://developer.android.com/reference/android/widget/package-summary [Accessed February 28, 2020].
3. "MSP DSP Library Documentation," 2018. [Online]. Available: http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/DSPLib/latest/exports/html/index.html [Accessed February 28, 2020].
4. "STC3100," January 2009. [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/datasheet/cb/7e/b5/82/36/e5/4c/f1/CD00219947.pdf/files/CD00219947.pdf/jcr:content/translations/en.CD00219947.pdf [Accessed February 28, 2020].

Appendix 1: Software Component Diagram