Software Overview (Spring 2020)

Team: 8
Project: AudioBeamer
Creation Date: 1/29/2020
Last Modified: 1/31/2020

Name: Aditya Thagarthi Arun **Email: athagart@purdue.edu**

Assignment Evaluation:

Item	Score (0-5)	Weight	Points	Notes
Assignment-Specific Items				
Software Overview		x2		
Description of Algorithms		x2		
Description of Data Structures		x2		
Program Flowcharts		x3		
State Machine Diagrams		x3		
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:
Relevant overall comments about the paper will be included here

1.0 Software Overview

We are using software/firmware to configure our DSP (digital signal processing) from a library for audio equalization, distortion, etc. For audio equalization, we plan to use digital filters (high pass, low pass, and bandpass or potentially shelf filters) to reduce/enhance frequencies. We plan to adjust some coefficient to the user's desired level to adjust certain frequencies. These coefficients will be received by the microcontroller via Bluetooth from an Android application. The packet sent via Bluetooth from the Android application will include values for equalization and distortion. This packet will be read from a Bluetooth device via UART. The microcontroller will communicate the digital audio signal between itself and an audio codec via I^2S.

To explain this process temporally, first the microcontroller will configure Bluetooth, DSP, I^2S, etc. The input signal received from I^2S will be put into the Low Energy Accelerator for Signal Processing, which will run digital signal processing on the signal in parallel to the main microprocessor. While the Low Energy Accelerator is doing the heavy lifting of DSP, the main microprocessor will check for input from the user (likely through an interrupt). The user can use an Android app to send a message over Bluetooth which will be communicated to the microcontroller via UART. These messages will be sent immediate when the app opens and when any app preferences are changed. When such a message is received by the microcontroller, an interrupt will be triggered which will update the DSP parameters. After the update, the interrupt will end. The processed signal will then be outputted to the audio codec again via I^2S. Additionally, a Power Management Integrated Circuit (PMIC) will monitor the power level of the battery powering the device. The microcontroller will check this power level on a regular interval of time (i.e. 5 minutes) and update the batter power indicator LEDs if necessary. This completes the main loop of the software.

Furthermore, the Android application will display options for equalization and various audio effects. Upon a change to the application's values by a user, the Android device will send a message to the receiver device.

2.0 Description of Algorithms

As mentioned beforehand, we plan to use high pass, low pass, and bandpass filtering for equalization. Audio distortion effects can be achieved via linear and nonlinear functions depending on the effect (amplification, for example, would be linear while overdrive would be nonlinear). These manipulations can be performed through the MSP 430 DSP library structures and functions (see "Description of Data Structures" for more information").

The overall algorithm will be to read a signal from the audio codec via I^2S, sending it through the LEA for DSP, and then outputting the signal back through the audio codec again via I^2S. Furthermore, the algorithm will continuously check for DSP parameter updates and battery life.

DSP parameters will be updated whenever a signal is picked up via UART to a Bluetooth device that will be communicating with an Android app on a user's phone. The application will send a DSP update when it first starts running and whenever any parameter is changed (for equalization, distortion, etc.) via Bluetooth.

To check our battery life, we plan to use datasheets for our PMIC and either AA or lithium-ion batteries. When battery voltage levels go below a certain threshold, an LED indicator will change color to indicate the battery life. We may make this implementation discrete, with the LED(s) changing color/brightness only at certain values. Alternatively, we may make this implementation continuous, with our LED(s) gradually changing color over time depending on the voltage level read. The former option is more likely at this point. Either way, the LED(s) will change to display an indication that battery levels need to be changed.

3.0 Description of Data Structures

The DSP library for the MSP430 family has various data structures. We will have to construct different instances of these structs and they can now be used with the functions in the library for our purpose. For all fixed-point data types, the MSP430 DSP library uses a combination of 16-bit signed integers and 32-bit signed integers with certain types representing whole numbers and others representing fractional decimal numbers. The data types are int16_t (16-bit integer), int32_t (32-bit integer), _q15 (15-bit fraction), and _iq31 (31-bit fraction). Precision of values depends on using either _q15 or _iq31 data types. All complex numbers are saved as "pairs of consecutive data". The library also contains data structures for matrices and for the following matrix calculation: add, subtract, transpose, multiply, negate, absolute value, offset, scale, and shift. There are also data structures listed for low energy accelerometer support, which includes further matrix manipulation techniques. Additionally, there are structures for fast Fourier transform function for both real and complex values. Lastly, there are structures for real and complex vector mathematics. Functionality for this includes vector functions for add, subtract, multiply, accumulate, negate, absolute value, offset, scale, shift, maximum, and minimum.

Type	Integer Bits	Fraction Bits	Min	Max	Resolution
int16_t	16	0	-32,768	32,767	1
int32_t	32	0	-2,147,483,648	2,147,483,648	1
_q15	1	15	-1.0	0.9999695	0.00003051758
_iq31	1	31	-1.0	0.9999999995	0.00000000046

For specific names and functionality, see source 3 listed below.

The data structures used for DSP manipulations are msp_fill_q15_params, msp_add_q15_params, msp_copy_q15_params, msp_scale_q15_params, and msp_biquad_cascade_df1_q15_params.

"msp_fill_q15_params" takes two values: length of the array and value with which to fill the array.

"msp_add_q15_params" contains the length of the array.

"msp_copy_q15_params" contains the length of the array.

"msp_scale_q15_params" contains the length of the array, the scale value, and a shift value by which to bit shift.

"msp_biquad_cascade_df1_q15_params" contains the length of the array, the coefficients of the relevant IIR filter, and the states of the relevant IIR filter.

"msp_biquad_df1_q15_states" contains four int values representing the states of the IIR filter.

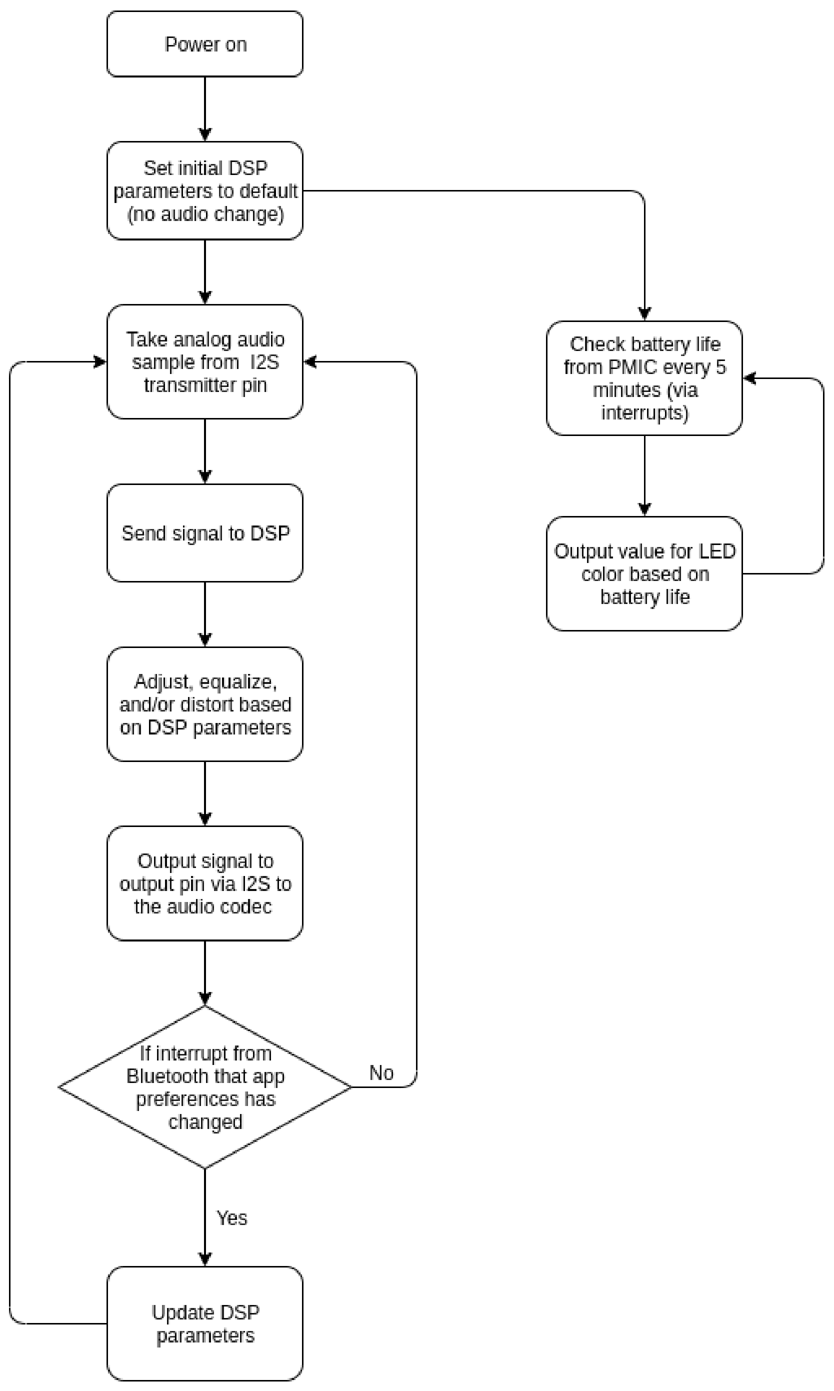
"msp_biquad_df1_q15_coeffs" contains six int values representing the coefficients of the IIR filter.

4.0 Sources Cited:

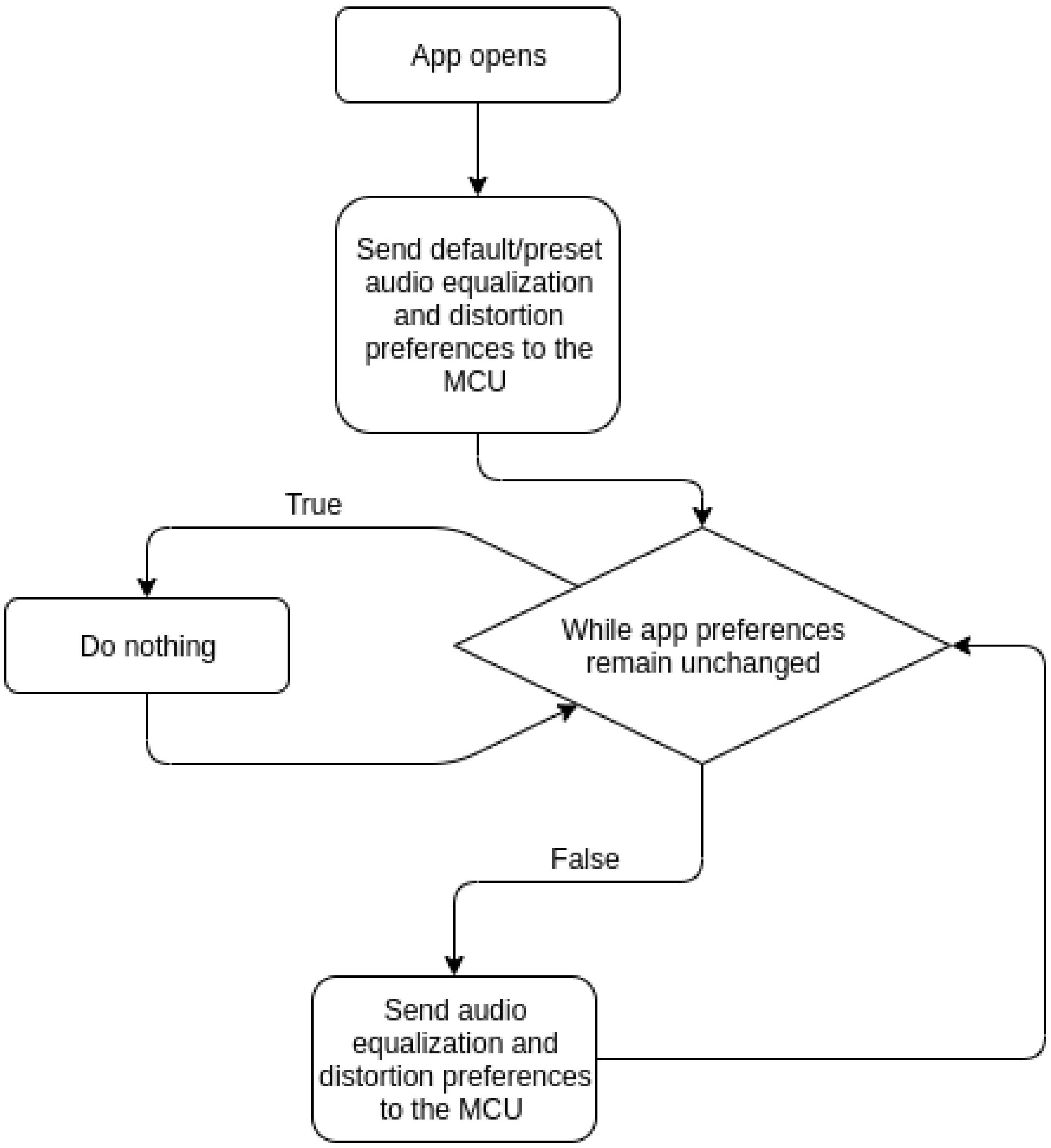
1. "Parametric Audio Equalizer," 2015. [Online]. Available: https://www.mathworks.com/help/dsp/examples/parametric-audio-equalizer.html [Accessed January 29, 2020].
2. Eric Tarr. "Distortion Effects," 2018. [Online]. Available: https://www.hackaudio.com/digital-signal-processing/distortion-effects/ [Accessed January 29, 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 January 30, 2020].

Appendix 1: Program Flowcharts

MCU

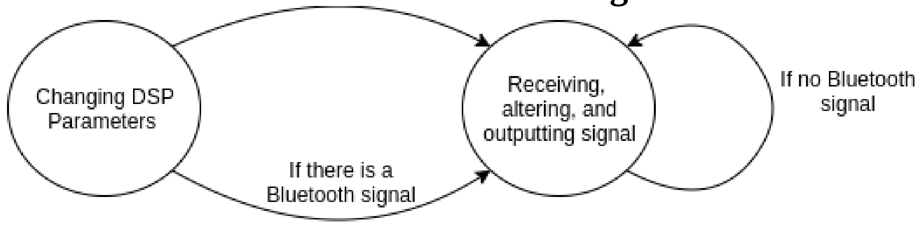

Power on
Set initial DSP parameters to default (no audio change)
Take analog audio sample from I2S transmitter pin
Send signal to DSP
Adjust, equalize, and/or distort based on DSP parameters
Output signal to output pin via I2S to the audio codec
If interrupt from Bluetooth that app preferences has changed
No
Yes
Update DSP parameters
Check battery life from PMIC every 5 minutes (via interrupts)
Output value for LED color based on battery life

Application

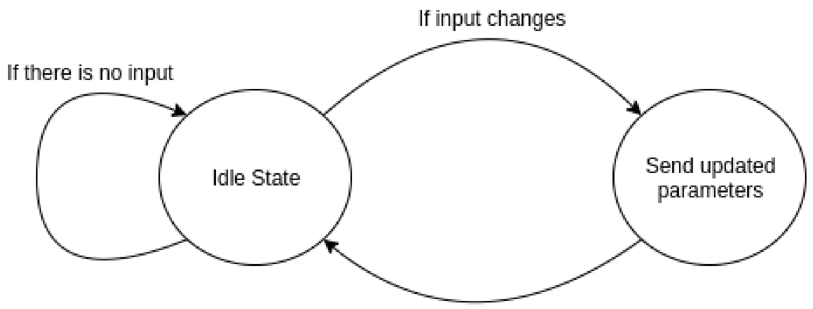

App opens
Send default/preset audio equalization and distortion preferences to the MCU
True
Do nothing
While app preferences remain unchanged
False
Send audio equalization and distortion preferences to the MCU

Appendix 2: State Machine Diagrams

MCU State Transition Diagram


Changing DSP Parameters
Receiving, altering, and outputting signal
If there is a Bluetooth signal
If no Bluetooth signal

Application State Transition Diagram


If input changes
If there is no input
Idle State
Send updated parameters