6.S08 Interconnected Embedded Systems

H A R M O N I

Spring 2017

Created by:
Shannon Hwang
Clare Liu
Jessica Tang
Charleen Wang

TABLE OF CONTENTS

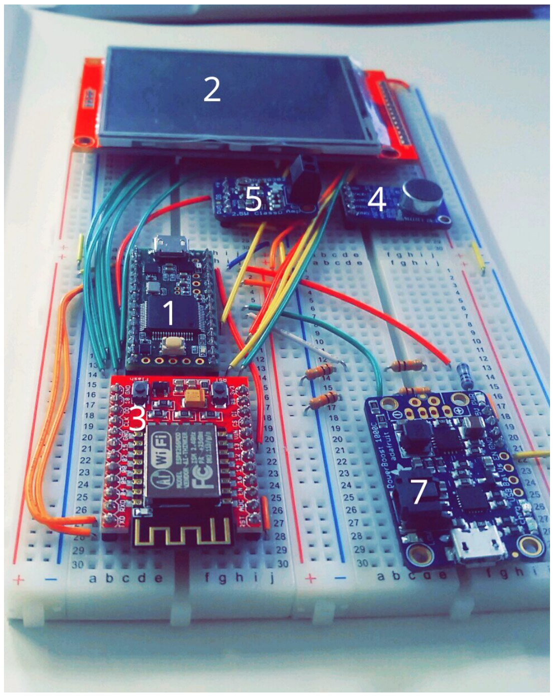
SYSTEM DOCUMENTATION **1**
Wiring 1
LIbraries 2
FUNCTIONAL BLOCK DIAGRAM **3**
STATE MACHINE BLOCK DIAGRAM **4**
DESIGN CHALLENGES & RATIONALE **4**
PARTS LIST **6**
DETAILED CODE DESCRIPTION **6**
Functions 6
Classes 7
Server-Side Code and Database 9
ENERGY MANAGEMENT **10**

SYSTEM DOCUMENTATION

Harmoni is a harmonizing system that allows its user to create and learn harmonies to any melody. Users are also able to share both harmony and melody with other users for them to learn, and compete with other users for the most accurate harmonization for any given harmony.

Harmoni is composed of 7 components:

1. Microcontroller (Teensy)
2. LCD Touch Screen
3. WiFi Module
4. Microphone
5. Amplifier
6. Speaker
7. PowerBoost / Battery



To the left is an example of the complete system (except the speakers, which are normally attached to the amplifier).

The speakers are attached to the amplifier with two small screws. On top of the amplifier module, there is a black trapezoidal piece with two small holes in the side. The speaker's leads go into these small holes, and the screws hold them in place.

Wiring
The breadboard was constructed by combining four smaller breadboards (two lengthwise and two widthwise). The connection necessitated breaking off the left PWR/GND railings on two breadboards before assembling (see picture on left for clarification).

The Wifi, microphone, and power modules are attached according to what we did during lab.

Connections between Amplifier and Teensy

Amplifier	**Teensy**
A+	5
A-	GND
SD	none
VIN	VIN

GND	GND

The A+ and A- handles differential inputs, but in our case, we only needed one. As a result, the A+ is connected to the teensy, and the A- is connected to ground. SD is not used and is not connected to anything.

Connections between Screen and Teensy

Screen	**Teensy**
VCC (power)	VIN
GND	GND
CS	9
RESET	3.3V
D/C	15
SDI (MOSI)	11 (DOUT)
SCK	13 (SCK)
LED	VIN
SDO (MISO)	12 (DIN)
T_CLK	13 (SCK)
T_CS	10
T_DIN	11 (DOUT)
T_DO	12 (DIN)
T_IRQ	18

The CS and D/C pins are used to set up the display on the screen and T_CS is used to register the touch screen. These three pins are used in the code; the rest are just connecting the screen to the teensy. The LED to VIN is connected through a 100Ω resistor. The CS and D/C pins have alternate teensy pin connections. Both of them can be connected to any of the Teensy's 9, 10, 15, 20, 21 pins. We used Teensy pin 9 for CS and pin 15 for D/C. The T_CS and T_IRQ can be connected through any available digital pin. We used 10 and 18 respectively. T_IRQ is also optional.

Libraries

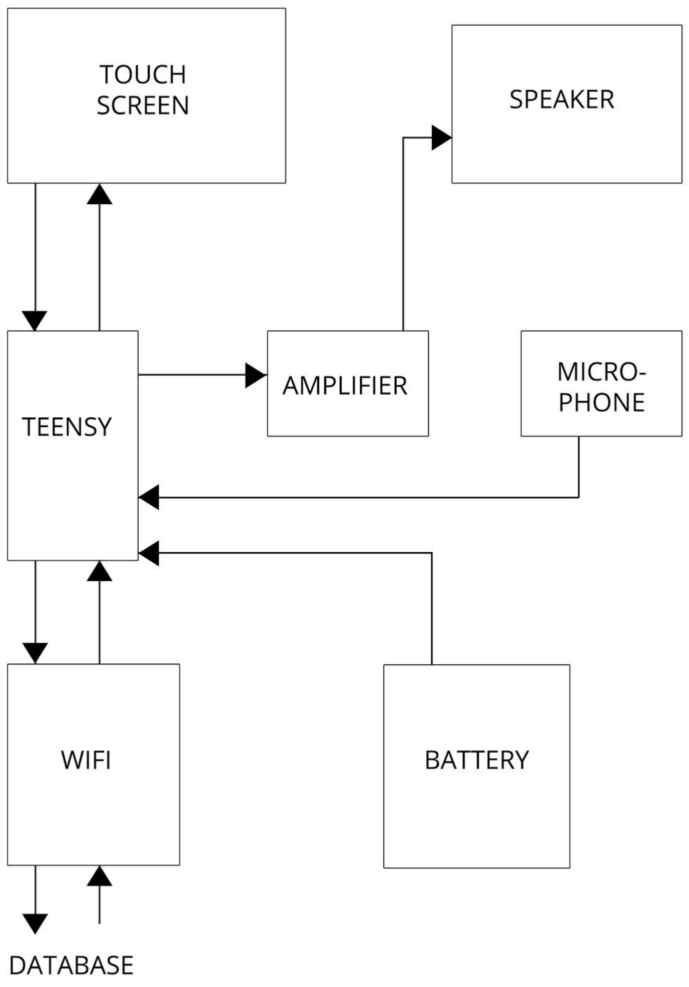
Several libraries were imported to help facilitate the creation of Harmoni. Quickstats is a library by dnubins (GitHub) with functions that can calculate statistics such as the mean or median of arrays passed to it. It was useful in the display, which calculated the average length of a note, correlated it to a set display length, and scaled other durations around that length. It was also used to find the average frequency of notes in a song in order to determine which octave the song was mostly in and draw notes on the staff in that octave.

The pitches library was used to determine note frequencies and write tones to the speaker.

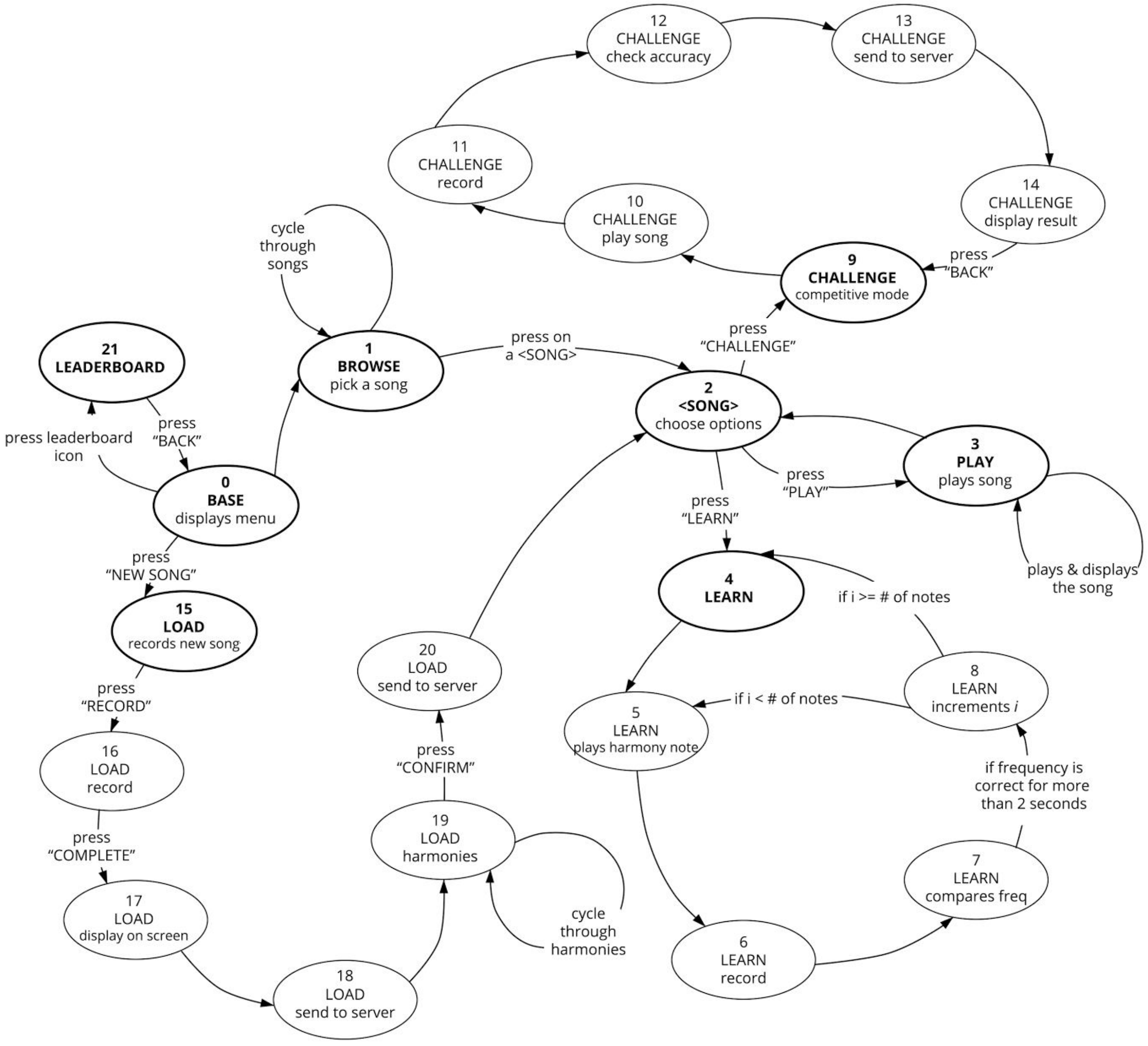
The optimized ILI9341 TFT Library by PaulStoffregen was used to better draw to our LCD screen.

Classes, functions, and further details regarding the code for Harmoni will be explained in the section DETAILED CODE DESCRIPTION.

FUNCTIONAL BLOCK DIAGRAM



STATE MACHINE BLOCK DIAGRAM



**NOTE: all states can return to state 0 (BASE) due to the [x] button on the screen in all instances.

DESIGN CHALLENGES & RATIONALE

One of the first challenges we came across was deciding how detailed we wanted the harmony algorithm to be. As we studied the music theory behind harmonies, we realized that there was a lot more than the scope and time of the project would allow us to delve into. Therefore, we decided to use a simpler, yet commonly heard approach to harmonizing: creating 3rds, 4ths, and 6ths moving in "tight harmony" with the initial melody (e.g. maintaining a interval of three notes, four notes, and six notes between harmony and melody for any given melody). Tight harmony is commonly used in pop songs, which we predicted the majority of users would attempt to sing, and its simpler implementation allowed us focus more time on the creation of Harmoni's other components, We allowed the user to select their preferred harmonizing interval via touchscreen.

One of Harmoni's major components, its LCD touchscreen, also posed major design challenges. We decided to use the touchscreen in order to avoid the inefficiency and general clunkiness accompanying the buttons/IMU input system used in class; however, in order to move ahead with the project before receiving the touchscreen, we initially coded graphics using our original OLED screens, and ported code to touchscreen later. This required us to learn unfamiliar graphics libraries for the touchscreen in a short amount of time.

A particular point of confusion was the fact that coordinate systems reported and used in different pieces of code we later ported did not align. We had to calibrate coordinates given by the touch screen to coordinates used by TFT.

The touch screen also uses considerably more power than the OLED due to its larger surface area and screen backlight, thus requiring more rigorous power management of other parts (see more in ENERGY MANAGEMENT).

One of the biggest challenges we came across was extracting frequencies from microphone input. First, we had to determine how to get frequencies from a set of retrieved samples. We considered using Fast Fourier Transform or Fast Hartley Transform, but, with some research, discovered they were inefficient and overly complex. We used autocorrelation instead, as it required minimal calculations and was fairly accurate. Autocorrelation fits the sampled data to a sine curve, and finds the frequency of the sine curve. It proved to be highly effective, with only a small margin of error we fixed by adding a scalar. We then binned the cleaned frequency to one of the notes in the pitches.h library to determine what note it was closest to.

Then, we had to determine how many samples to retrieve, and at what frequency, in order to conserve space without sacrificing accuracy. We experimented with different sample rates, viewing them on the Serial Plotter to get a rough estimate of how many samples drew a good enough curve. Then, we experimented with different sample rates and sample sizes by playing known frequencies via tone generator to the microphone, and seeing what frequency was measured through our refined autocorrelation and note binning code. Eventually we settled on a sample frequency of 22050, gathering a total of 221 samples per sampling period.

Lastly, we had to contemplate memory usage. We had to determine what data we wanted to keep on the Teensy side and how to best go about maximizing efficient use of space. At one point, we tried to keep all song details on the Teensy side but ran out of memory. In the end, we decided the best solution was to only ever keep one song stored on the Teensy, but all the ids and names of songs on the Teensy. This way song details could be pulled from the server by querying with the song id and then overwriting the one song stored on the Teensy with the new details.

PARTS LIST

(additional parts not included in the standard 6.S08 system)

1) LCD Touch Screen
   320x240 Color TFT Touchscreen, ILI9341 Controller Chip
   Used in place of the standard 6.S08 OLED screen to allow navigation between Harmoni's different modes via an easier-to-use, streamlined touchscreen interface.
2) Amplifier
   Adafruit Mono 2.5W Class D Amplifier
   Used for volume control of the 4-ohm+ speaker.
3) Speaker
   4-ohm+ speaker
   Used in place of the standard 6.S08 buzzer to play back melodies, harmonies, and various notes.

DETAILED CODE DESCRIPTION

(please refer to Main.ino)

Functions

int parsenotes(int *, int)

Given an array and value, it finds the closest value in that array to the given value. Used to bin frequencies to notes defined by pitches.h.

int counting(String, String)

Returns number of instances of a given String (second parameter) in another String (first parameter). Used to determine the number of notes stored in a string.

void strToArr(String, float *, float *)

Parses the String of compiled data, separates the data, and fills it into two arrays - one for frequencies (second parameter) and one for durations (third parameter).

void tsprint(int, int, String)

Prints a string on the LCD screen at the specified coordinates.

void tsclear( )

Clears the LCD screen by filling the screen with black.

void rectclear( )

Clears the portion of the LCD screen where notes are during play.

int findNote(int)

Returns the index of the given int (frequency) in NOTES[], an array of all note frequencies.

String getList(), String getNotes(String), String getLeaders()

Sends WiFi requests for the list of songs currently stored in the database, notes of a certain song (accessed via its id), and high scores for a song's challenge mode.

void drawMainMenu(), void drawBrowseSongs(String), drawSongOptions(String), drawLeaderboard(String)

> *Draws out various menu screens required in the device, such as the main menu and the leaderboard. Some instances take in String data to parse for display.*

void makeLists(String)

> *Given the String from* getList(), *generates instances and sets up objects of class Listing for each song.*

loop()

> *Determines whether LCD screen has been touched, and if so, where.*
> *Contains state machine for highest-level navigation of Harmoni between the main menu, browsing songs, song-specific menus (with links to play, learn, and challenge a certain song), playing songs or harmonies, learning songs or harmonies, challenging songs , recording new songs, and looking at the leaderboard for challenged songs.*

Classes

Note

> *Class that defines all the properties of a single note of a song, including its frequency, duration, and physical length on screen. It also includes functions to draw the note, as well as update its location when being played.*

Song

> *Class that defines and has all the functionalities of a song - a string of notes with frequency and duration. Includes an array of objects of class Note for each note in the song as well as the following functions for creating, drawing, and playing a song:*
>
> void setup(String)
>
> > *Takes in a data string with all the notes of a song and sets up objects of class Note in array notes[] corresponding to the related note frequency and duration. Based on the average note duration and frequency, it sets up the locations of a note on screen in relation to a 'center note' generated based on the closest note B to average frequency of the song; and sets up the physical length of each note depending on the average duration of a note.*
>
> int gety(int, int)
>
> > *Gets the y location of a note depending on its frequency and the base frequency being used as the 'center note'.*
>
> void drawStaff()
>
> > *Draws the staff on screen.*
>
> void restart()
>
> > *Returns all the notes to their original location.*
>
> void play(int); void hplay(int)
>
> > *Writes notes in the main melody or harmony of a song to pin (first input).*

void update(int, int)

Draws notes (both melody and harmony) on a staff, animates and plays either melody or harmony depending on input parameters.

void harmonize(int)

Generates harmony based on the main melody depending on the int interval given.

void addHarmony(String)

Takes in a data string with all the notes of a harmony and sets up objects of class Note in array hnotes[] corresponding to the related note frequency .

String getHarmony()

Returns a string of data with the frequency and duration of notes in harmony.

Listing

Class that defines the name, id, and kerberos of a song. Used to generate listings of songs taken from the server and displayed in Browse. By using a class that holds all the data of a listing, we only have to have one instance of the Song class and can simply overwrite the details by querying for a new data string with the id from a Listing. Functions in Listing include writing details to a Listing object and drawing a Listing.

Freq

Class with functions to capture samples from the the microphone and run autocorrelation, frequency cleaning, and binning to a note.

Recorder

Class with the following functions that record audio, call functions in class Freq to get frequency, and save data from audio pulled to a string:

int get_note()

If sound from audio is greater than a threshold, takes a sample and through functions in Freq return the frequency from the sample.

void record_audio()

Gathers duration of a note from the sample based on when sounds cross the threshold and then packages frequencies and notes into a data string.

void reset()

Clears the data string.

String output()

Returns the data string.

Button

Encapsulates all the functionalities of touchscreen “buttons” - drawn buttons on the screen that are linked to certain actions/events. Contains variables determining where *the button is and text, if any, inside said button, and methods to draw the button and determine if a touch is within the bounds of a button.*

MVP

Contains higher-level variables, methods, and state machines needed for all of Harmoni's Learn and Record modes. Below is a listing of their associated methods in the MVP class:

void learn(int, int, boolean, boolean, String)

Takes in coordinates of and information about the user's touch on the LCD screen, and the melody to be inputted. Contains a state machine allowing the user to move between starting the song, playing a song note, and judging the user's accuracy for the song note (via whether they are too low or too high); the user advances to the next note once they have successfully matched a note for two seconds or hit the "next note" button.

float ind(String)

Parses an inputted song string for the note to be learned; used by learn()

void learnNext(String)

Allows user to skip to learning next note in inputted song; used by learn() and ind()

int getLearnState()

Returns the current state of the learn state machine that the user is in.

void learn_reset()

Sets learn state machine state to zero (returning the user back to starting the song).

void record(int, int, boolean, boolean)

Takes in coordinates of and information about the user's touch on the LCD screen. Contains a state machine allowing the user to move between choosing to record a new song, recording, playing back the recording, accepting or rejecting the recording, sending it to servers, choosing harmonies, and accepting or rejecting the harmonies.

void record_reset()

Resets the record state machine's state to zero (returning the user back to choosing whether or not to record the song).

Server-Side Code and Database
(please refer to parsenotes.py)

Server side code was compiled into one python file. By specifying a call the server would query for different things. For example, the call "list" would return a string with a list of songs stored in the server in the melody table. And the call "melody" with the id of the melody in question would return a string with the frequencies and durations of notes in the song.

The database structure is comprised of three tables, one for storing melodies, one for harmonies, and one for the leadership board (shown below).

MELODIES TABLE					
ID	**Timestamp**	**Kerberos**	**Name**	**Notes**	**Test**
auto	*auto*	*kerberos of user (varchar)*	*name of song (varchar)*	*stored notes (varchar)*	*for testing purposes (int)*

HARMONIES TABLE					
ID	**Timestamp**	**Kerberos**	**Melody**	**Notes**	**Test**
auto	*auto*	*kerberos of user (varchar)*	*ID of the associated melody (int)*	*stored notes (varchar)*	*for testing purposes (int)*

TEAM6_LEADERBOARD TABLE					
ID	**Timestamp**	**Kerberos**	**Harmony**	**Score**	**Test**
auto	*auto*	*kerberos of user (varchar)*	*ID of the associated harmony (int)*	*score of accuracy (float)*	*for testing purposes (int)*

ENERGY MANAGEMENT

Though we did make design choices and considerations to optimize energy consumption, physical mobility of the overall system is not a crucial or even important component of Harmoni. Nevertheless, we lessened the energy consumed by the system's two most power-hungry components: the LCD screen and WiFi.

Navigating through Harmoni requires that the LCD touch screen be constantly on - but since it consumes relatively more power than the rest of our system, we attempted to minimize its power usage by re-drawing and adding more elements to the screen only when absolutely necessary (e.g. only when new or different information has to be displayed), and we minimized the number of graphics commands sent to the LCD.

Lastly, we also did try sleeping the Teensy, WiFi, and amplifier, however, ran into some issues. Because we put so much work onto the Teensy, we opted to not sleep it at all. As for the WiFi, there was no way to sleep it without using a timer, since all we wanted was to wake the WiFi when a response was called and sleep it after the response was received. We also wired the amplifier to be able to be slept but could not locate the libraries for it. Nevertheless, the system ultimately does not use that many power-consuming devices.