ECE 558 Final Report

Ryan Bentz
Ram Bhattarai

Speech Driven Light Show

Project Description

The project intention was to create a prototype device that could be scaled up and used as part of an interactive video display that could be deployed for art and entertainment purposes. There are two major components to the prototype: a mobile application providing a user interface and a hardware application controlling the underlying hardware. For this reason, we chose to go with the more difficult TLC5940 LED driver than the easier but more expensive individually addressable LEDs.

The software application allows users to add new words via speech-to-text. When the user click on any words, it will open up a new screen shown via dynamic fragment which allows the user to then set up to 8 custom colors for the word. The custom colors are saved and used in the custom lightshow for that word. The user can press a button to begin the lightshow when they are ready. Currently, the lightshow consists of a simple cycle through the customized colors. When the user indicates they want to play a lightshow, a special flag is updated in the database to signal to the hardware application the word to play the lightshow for. The hardware application retrieves the data, processes it, and begins transmitting the data to the low-level devices which handle the LED PWM.

The mobile application and the hardware application communicate with each other and share information through Google Firebase. The database stores the list of known words and that the mobile application uses to build the list it displays and that the hardware application uses to retrieve the color data to display the LEDs with.

Hardware

At the highest level, an Android app runs on a Raspberry Pi and listens to the database for changes. When a change is made to a special member to indicate it is time to play a light show, the app reads the database for the word then reads the database to get the color information for that word. The app cycles through each color and sends the information to the Arduino. To send a color, the app takes the RGB values for the color stored in a single int-type and separates them into their respective values. Each RGB 8-bit value is converted into a 12-bit PWM duty cycle value. The 12-bit PWM values are arranged into an array of 48 12-bit values for the 48 unique channels spread out over the 3 TLC5940 devices. The array of 12-bit PWM values is then deconstructed into a byte stream of 72 bytes. The byte stream is organized as 3 blocks of 24 bytes and each block is sent to the Arduino over I2C for displaying.

At the lowest level is the TLC5940 12-bit PWM 16-channel LED driver. The TLC5940 is controlled by an Arduino that acts as the intermediary between the Raspberry Pi and the LED's. The TLC requires that the PWM be shifted into it and stored in registers before the LEDs are turned on. The TLC5940 cannot control the LEDs on its own. It has a dedicated pin (GSCLK) that controls the PWM of each channel. Toggling the GSCLK pin increments an internal counter that counts from 0-4096. As the counter increments, the TLC5094 checks the PWM values for each channel. If a channel has a matching PWM value, that channel is turned off. When the counter reaches 4096, a second pin (BLANK) must be toggled and the process can repeat.

The needy nature of the TLC5940 requiring constant pulses on its input to get the PWM effect meant that controlling the TLC5940, while doable from the Raspberry Pi, was not a good use of its time and it was better to offload that work to a microcontroller like the Atmega328.

Software

Splash Activity

Splash Activity makes a use of AsyncTask to read from the database in background thread. If there are no contents available in the database, Toast message will be displayed letting the user know that there is no content available in the database. When it finds the content in the database, the word list will be added to the word manager. After successfully adding the content to the word manager, it will start the Main Activity for the setting up recycler view and fragments.

Main Activity

Main Activity manages dynamic fragments setting up home screen for the app. It inserts the recycler view with list of words into the top half of the screen and bottom half with the control buttons. When the user selects a word from the recycler view, the activity replaces the RecyclerView fragment with the specific word fragment.

WordList Fragment

WordList Fragment implements the RecyclerView used to display and navigate the list of known words.

Word Fragment

Word Fragment handles the process of selecting the colors for each color box and recording the data which will be used later to perform light show.

Word Manager

Word Manager is responsible for maintaining the list of known words locally on the app. It creates a centralized location for an activity to interact with the word list across all fragments and activities.

Control Fragment

Control Fragment implements the fragment that manages two control buttons playshow and speech to text button. Speech to text input button allows the user to add a new word to the recycler view and make their own custom light show.

Database Manager

The DatabaseManager class manages the interactions with the database on behalf of the other classes in the app. Primarily it handles the process of reading the complex WordList structure from the database and extracting the words and color data. It also writes the wordlist back to the database as needed.

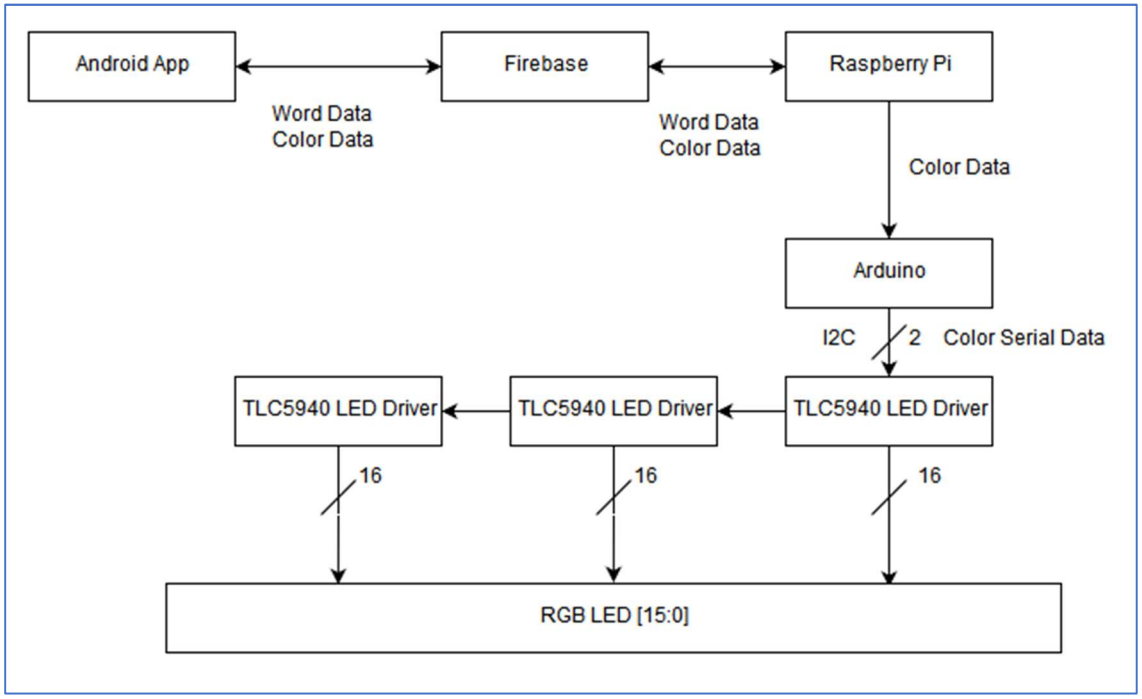
Database

The database architecture follows a specific format that was set by our app design. There is a top-level item called “word” which acts as the root for all the known words in the database. Child members of the

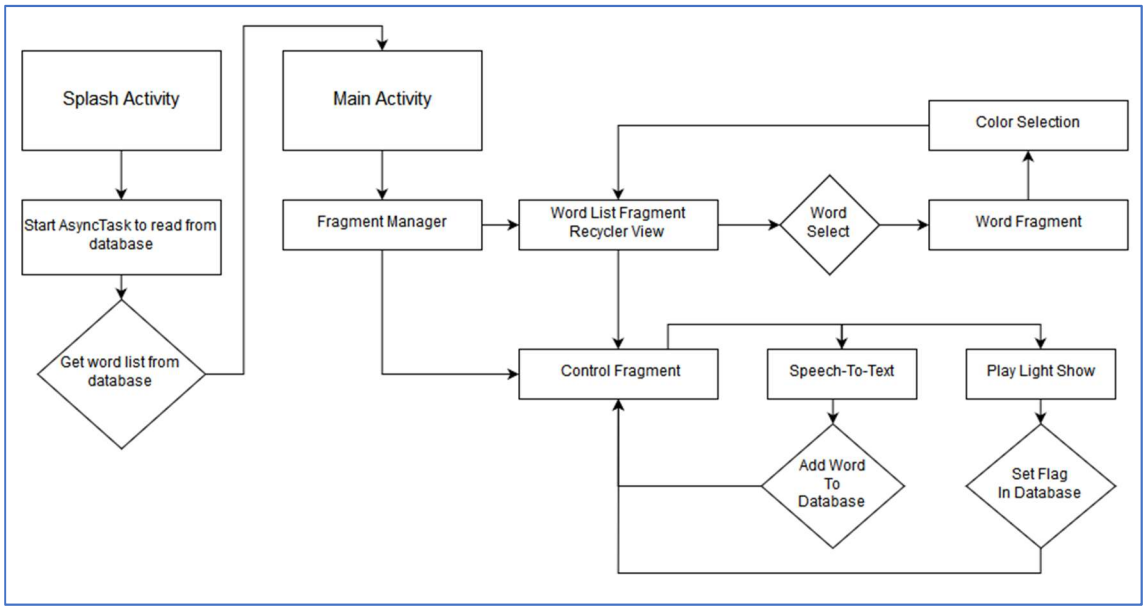
root contain the known words who also have child members containing the values of the user specific colors for the word.

Project Architecture

The below figure shows the overall design of the project.



The below figure shows the overall design of the software application running on the mobile device.



Challenges Faced

1. Creating the low-level driver for the LED Driver and interfacing with Arduino

   We had initially tried to use libraries available on the internet to interface with the TLC5940 but were not immediately successful. After trying to get them to work, we decided it would be more prudent to write our own low-level driver to at least make sure that we had something working that we could count on instead of hanging our hopes on a library written by others. We decided that the time spent learning to use a library would require some knowledge of how the TLC5940 worked and at that point, we could probably write our own driver anyways.

   We would like to share that our demo was not up to par due to a problem with the low-level driver. The problem was that during the time we were trying to debug the driver with the Arduino Serial Print, we moved the pins controlling the TLC5940 away from the same pin that was sending UART data to the Serial monitor for printing. In doing so, we created a conflict with a pin that was supposed to toggle an LED every time there was I2C data received on the Arduino. That LED pin also became the BLANK pin which is connected to the TLC5940 and toggled at the start of every PWM sequence. Indeed, a falling edge on the pin initiates the entire sequence. Toggling this pin on every I2C transfer caused its state to be inconsistent for the needs of the TLC5940 which caused the LEDs to have different colors and not consistently stay on. Needless to say, it was disappointing to find after the fact when we cannot make a video due to team members travel schedules (Ryan left Oregon only a few hours after the demo), but we hope this explanation at least sheds some light on the poor performance of the low-level driver during the demo.

2. Creating a decent lightshow given the chosen hardware design

   This was the most complicated part of the project. We had a few ideas of how to implement a lightshow but it added a level of complexity that we were not prepared for. Our idea of having a blending algorithm that would transition smoothly from one color to another added a level of complexity to the project that we were not ready for. We had wanted to include a color blend and flashing sequences that could be manipulated by sensor data gathered by the mobile application but those ideas in themselves could be projects and we ultimately had to scale down our ambitions.

3. Synchronizing information and events across fragments

   We were not ready for having to process information across fragments. One thing we had to contend with was how to transfer new words from the control fragment running the speech-to-text to the RecyclerView fragment. We initially had done this by creating our own class implementing the View Model abstract class and creating a runnable in the RecyclerView fragment that periodically polled the View Model object for new words. In the end, this method was abandoned and the WordManager was used as the intermediary between fragments.

4. Reading and writing to the database when the members are complex structures.

   When we created the database structure, we did so with the understanding from reading StackOverflow and Android Developer that our idea was not ideal for Firebase. But with the limited time and most of the app already structured a certain way, it would have cost more to

redesign the app than to deal with the database. We decided that, while not ideal, would be acceptable for a class project but certainly not for a professional app. We did not have familiarity with hash maps but thankfully the JSON information we received earlier in the term helped us to understand how Firebase was organizing the data and how we could parse it then put it back together.

Future Improvements

1. Implement color blend algorithm and additional chase/light sequences in TLC5940 driver on Raspberry Pi.
2. Implement "Record Show" methods to gather sensor data and create algorithm to tie sensor data to lightshow.
3. Full Speech recognition software that acts similar like siri but sends various commands to a device to produce a lightshow.
4. Correct the pin outputs of the low-level driver to eliminate the conflicts in pin usage (already done in submitted code).