InteLEDs Midterm SDP Report

Justin Lad, EE, Alden Michaels, CSE, Sade Luwoye, EE, and Arseny Izotov, EE

***Abstract*—The InteLEDs vision is an LED Music Controller featuring an automated lightshow where colorful LEDs react in harmony with the music being played, and a remote to control the lights for a price under $200. The user may play songs via auxiliary input or Shairplay. This paper describes the current state and future goals of each major subsystems, and the plan to integrate them into a final product.**

I. Introduction

Noncommercial products which connect the auditory experience of music with the visual experience of suitably synchronized LEDs are limited. This is due to a couple of obstacles: it is difficult to market lightshow products to a public when only targeted for party settings, and it is challenging to do processing on sound and control lights accordingly in real-time. Consequently, products such as the CUBEecho[0] target only commercial and wealthy audiences with a price around $500. Cheaper laser light show products exist[1], but yield poor coordination with sound, spending more time collecting dust than actually plugged in. These, and other current lightshow products on the market utilize simplified methods such as amplitude detection to give an appearance of beat recognition. Such oversimplification limits possibilities for what the lightshow can provide. InteLEDs strives to overcome both obstacles by acting as the central lighting system in the room with simple controls from any device with WIFI access. InteLEDs invites itself into one's home and everyday use, by acting as a unifying system between the lights and speakers of one's home while delivering comprehensive music analysis. Controlling the delivery of both audio and optics, the InteLEDs system can smoothly achieve synchrony.

Because the InteLED's music controller flashes bright lights at high speeds, analysis on the effect of our product on the public is necessary to ensure the safety of all users. Three percent of epileptics report a sensitivity to flashing lights yielding seizures[2]. We will make it clear to those users that they should use our product with caution and at their own risk, voiding us of liability, and ethical responsibility.

II. Design

A. Overview

We will make a LED music controller. We decided the ideal candidate for a processing system is the BeagleBone Black[3] (BBB), a popular development platform, for two major reasons. First, there were not many other candidates. A DSP seemed too specific and lacked the ability to use existing python libraries and standard Linux features. A laptop was considered for speed and convenience, but it didn't fit our embedded system vision. The last choice is a Raspberry Pi, which pales in comparison to specs[1]. Not to mention, it's fairly cheap ($50). For development boards, the BBB is as good as it gets.

Python has been chosen as the primary language to complete the project, over lower level languages for various reasons. Primarily, it is perfectly suited for prototype development by offering powerful built in functionality and libraries that makes writing code natural and pain-free. The major drawback of using python is performance, but because we're allowing a maximum buffer time of 10 seconds, execution time of our programs isn't critical to the functionality of our project. Additionally, many of the python libraries being used (pyAudio, numpy.fft) wrap C libraries, so those processes are almost as efficient as C. The most important aspect is the coordination of light signals and music playback, which will be simpler using Python.

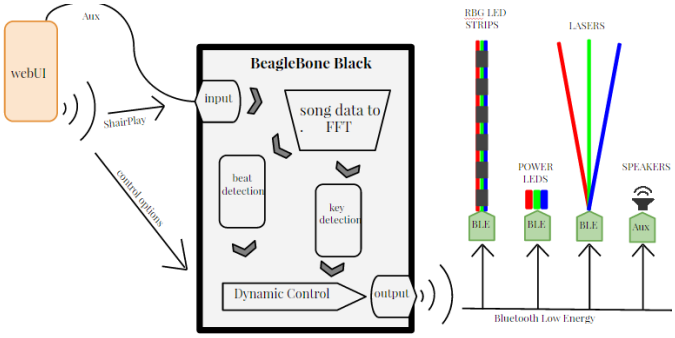

Figure 1: The System Block Diagram

Figure 1 outlines the system as a whole and the interaction between subsystems, which will be analyzed in depth in addition to the specific technologies used in the sections below. Sade is responsible for block 1, Alden for block 2, Justin for block 3, and Arseny for block 4. The functionality of each subsystem guarantees the complete system will work because we have already demonstrated each subsystem works. The real challenge now is to coordinate the integration.

B. Block 1: Remote Light Controller

This block includes the Web App interface and Shairplay[4]. The Web App will be compatible with iPhone, Android, and computers. The Web App gives the user the ability to choose a specific color for the LEDs. The user will also have the

J. L. Author from Bangor, ME (jlad@umass.edu).
A. M. Author, Jr., from Auburn, MA (apmichae@umass.edu).
S. L. Author from Fall River, MA (sluwoye@umass.edu).
A. I. Author from Acton, MA (aizotov@umass.edu).

ability to choose whether they would like the lights to flash to the beat of the music or choose a light show demonstration.

The color wheel was created using HTML5. It is a visual representation of colors arranged in a circle to show the relationship between primary and secondary colors. Figure 4 displays the color wheel image used for the Web App. The color wheel consists of two main components: the preview pane and the color wheel. When the preview pane is clicked, the color wheel will be displayed. First a new canvas is created, then a png image of a color wheel is drawn on the canvas. Event handlers were added for when the user clicks the preview pane and the color wheel. When the user moves over the color wheel, information about the current color is refreshed and updated. A server was written for interfacing with the LEDs. It allows a webpage to be called when the user clicks on a point in the color wheel. For example, if the user clicks on red, the hex RGB value is 0xFF0000, so http://10.10.10.101/rgb/FF0000 would be called.

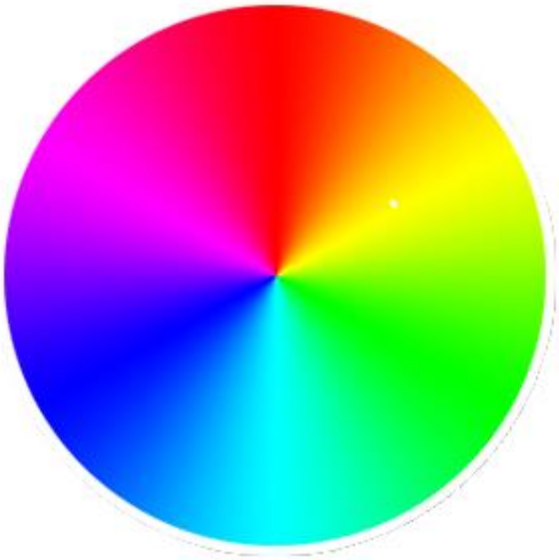

Figure 2: Color Wheel for Light Color Control

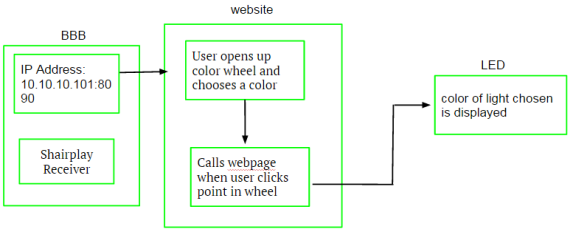

Figure 3: Remote Light Control Block Diagram

To allow the BeagleBone Black to become a wireless audio receiver, Shairplay was installed. Shairplay is a free portable AirPlay server implementation. It allows music to be streamed from a mobile device to the BeagleBone black[5, 6].

C. Block 2: LED Modules

The LED Module system consists of two systems: the server that sends control signals to the module, and the LED module itself.

The block shown in figure 3 describes the software and hardware stack for LED control on the BBB. First, there is a static Python web server which serves Sade's WebUI HTML file, a static file that includes HTML for layout and a JS program for asynchronous network control of our lighting system. In addition to that static server, there is a dynamic server which takes HTTP requests and uses them to control the lights. The commands come in the format of http://beagleboneaddress/rgb/RRGGBB, where RR is the 8 bit hex value of the red intensity (from 0-100%), GG is the same for green, and BB for blue. This standard API allows any programmer to write an interface in any language on any internet-connected device that is active in the user's home. This allows for a tight level of integration into existing products, leveraging the extensive trove of internet connected devices most households contain. The dynamic Python server takes these commands and outputs them via a standard USB Bluetooth 4.0 transceiver, a popular item available for under $10 from any computer retailer. If we were to create custom boards for the BBB, this transceiver would be replaced with a Nordic Semiconductor nRF51822, or similar, and place that on the board as well.

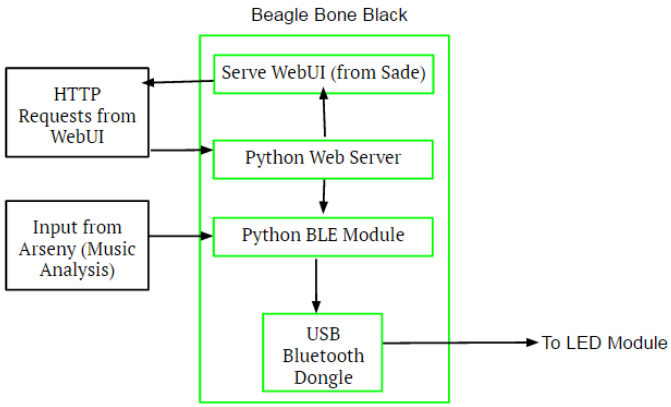

Figure 4: LED Control from BBB

Figure 4 depicts the LED Module, the device responsible for controlling the LEDs electrically. It is powered by a standard 12V "wall-wart," which allows us to leverage the economies of scale for that component. This 12V is fed to a 3.3V LDO (low-dropout regulator), which powers the logic of the mainboard. The module has a 2.4GHz antenna to receive Bluetooth signals, which are fed into a dedicated BLE radio. The Nordic Semiconductor's nRF51822[7] was chosen, as it has an integrated BLE radio and a powerful 32 bit ARM Corext-M0+ processor on one SoC die, with a low price point of under $2 in quantity. The ARM processor is programmed with custom control logic, which uses the hardware PWM capabilities of the internal timers to create RGB signals corresponding to the colors received via Bluetooth. These signals are sent to MOSFET driver circuits, which are connected to 5M long RGB LED strips. These strips are

waterproof, have over 300 RGB LEDs (900 total LEDs), and consume about 48W of power at full brightness. They cost less than $20 per 5M roll, so we are once again leveraging the economies of scale here.

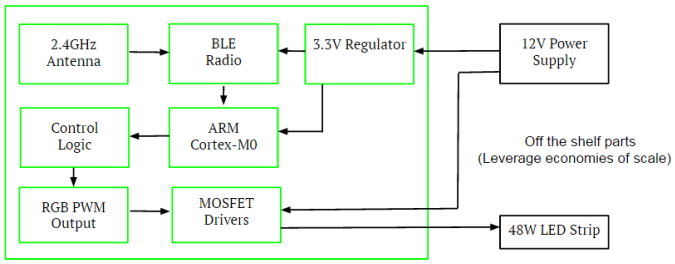

Figure 5: LED Module Block Diagram

D. Block 3: Live Stream & Frequency Transform

The specifications this block must accomplish are recording music streaming on the USB sound card, transforming the song information into its frequency representation (which is then sent as an input to block 4), and finally coordinate playing the lightshow and music for every moment of the song.

This block will consists of three python programs, tied together by a python script depicted in figure 5. The first python program will continuously record 2.75 second samples of music input from the sound card and send the .wav file with its associated time stamp (to keep track of how to piece the song back together) as inputs to two programs: songAnalysis.py and controller.py. The songAnalysis program takes a .wav file and a timestamp as an input, opens the wav file, converts it to the frequency domain, and outputs the beat and key of that song segment (this is block 4). It will then send its output (time stamp and songInfo) to the controller program. The controller program will listen for a .wav file and songInfo and keep two queues: one for .wav files and one for songInfo. The buffer is implemented by checking if there are a couple matching timestamps each time a new .wav file or songInfo is received. Matching time stamps in each queue indicates that our system will be able to play the song segment and control the lights accordingly.

In order to complete this block, research and testing must be done to learn how to make the controller program listen for asynchronous inputs, and how to coordinate the programs together. In order to develop the functionality of my block, we will first continuously record and pipe the output to analyzeSong.py. Then complexity will be added by sending the recorded output to analyzeSong.py and controller.py. At that point, controller.py will be able to add its input to the correct queue. Analysis of my results are fairly self-evident—either it works or it doesn't. Once the system is fully integrated, the ultimate test is whether the program controls the lights in sync with the music being played.

A significant portion of prototyping this subsystem has been accomplished at this mid-year junction. A major aspect of this block has been achieved: converting a song in .wav format to its frequency domain representation, by employing the use of a python wrapper of a C library called scipy.fft, which hase been successful in recording and playing back audio using the pyAudio library; however, implementing the controlling system will be fairly involved and the crux of the deliverable for CDR.

Completing this block will require drawing upon a culmination of skills accrued during my UMass education. Namely, software courses like data structures and algorithms that taught me how to formulate programmatic algorithms, and software engineering (ECE 373), which taught me parallel processing, Unix, and scripting fundamentals.

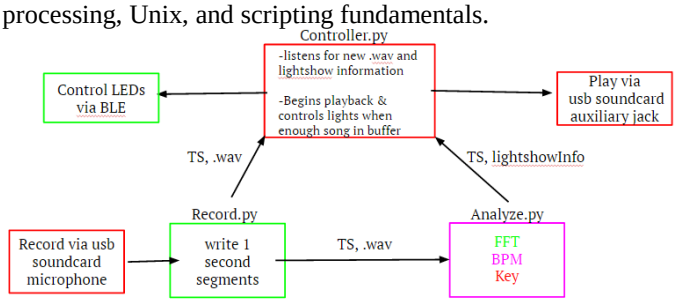

Figure 6: Live Stream Block Diagram

E. Block 4: Song Analysis

The main purpose of this block is to analyze the song, and create a lightshow accordingly. The analysis will consist of finding the tempo, individual beats, and possibly key detection of the song, to allow a programmatic approach to color selection. This block will receive sampled song data points over time, compute and store the analysis, then output lightshow information mapped to appropriate times, cutting out computation time wherever possible. All parts of the system take into account a faster reaction time of human perception: 100 milliseconds[8].

The beat and tempo detection is completed, and ready for optimization when integrating the blocks. The beat and tempo detection are both done by applying a comb filter on simplified frequency energy points. Those simplified frequency energy points are samplings of the energy of the song, 20 times a second, in multiple different frequency ranges. The multiple frequency ranges allow for more intricate song analysis.

Each frequency is treated separately, differentiated to find the biggest changes in energy, and then the comb filter is applied. The comb filter takes a dot product of evenly spaced rectangular pulses with one of the frequency energy arrays. Sweeping over different spacing of the rectangular pulses, and the different possible starting points of the rectangular pulses, the highest dot product values are recorded.

Recorded spacing of the rectangular pulses map to a certain number of beats per minute (BPM). The start of the first rectangular pulse maps to the start of the first beat. This is repeated for every frequency range, once a second, operating on the past 2.5 seconds. Due to the nature of music, different frequency ranges produce different results, which can later be used for more complex lightshows. For now, the fastest BPM is recorded per one second interval.

To deal with memory issues as seen in Computer Systems Lab, and to learn from previous mistakes, old data is cleared to

make room for new music and analysis. This data is also structured accordingly to linked list systems as exemplified in Data Structures at UMass.

What remains for a simple lightshow is integrating different patterns of flashing and fading to the LED strip. For a more complicated lightshow, the key of the song may be found by mapping the most persistent frequencies to chords. More research needs to be done in regards to music theory to deal with overtones and undertones, but the concept is present. Specific keys are known to be sadder or happier than other ones, which can map to warmer or cooler colors. Also, with more LED strips available, more complicated optics can be programmed, and switched between tempo changes.

Due to the complexity of the algorithm, it is important to keep the code maintainable by keeping it readable. Along with the benefits of ease of integration with other blocks, and the availability of C libraries for the fast Fourier transform, python is again used.

To test the accuracy of the program, multiple methods were used. One was to see how the algorithmically detected BPM of the song related to audible BPM. Matching a metronome to the beats of several tested songs, the audible BPM was found. The two BPMs were accurate within the human 100 millisecond recognizable margin. In essence, a song with BPM of 117 was detected to be either 110 BPM or 120 BPM over 95% of the time. Another test was to write down audible beats for a 10 second span of music by ear, to mimic the human ability of detecting beats. The programmatically detected beats matched up at most beats excluding two conditions: around random samples included in modern music (such as an eagle chirp) and around tempo changes. This can be overcome during integration by having oversimplified lightshow transitions during those tempo changes.

Key detection may be more difficult to analyze in modern music, but by using songs that stay in a known key, this can be easily tested. Future tests for whether lightshow patterns match up can be done by handling the lightshows with a consistent BPM, and again measuring with a metronome to see if the lights react accordingly.

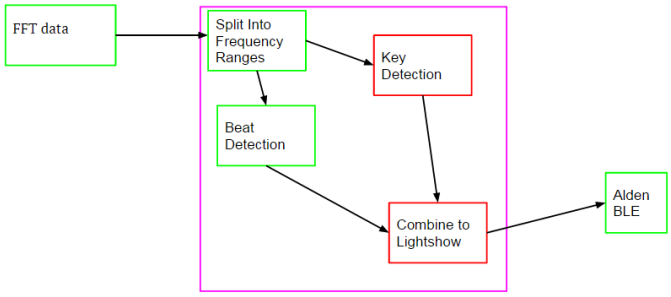

Figure 7: Song Analysis Block Diagram

III. Project Management

MDR Goal	Status	Estimated Time until Delivery
Play song on BBB via Shairplay	Completed	N/A
Control Lights via webUI	Completed	N/A
Convert .wav file to frequency domain representation	Completed	N/A
Calculate BPM for every moment of song	Needs Optimization After Integration	CDR
Control lights to blink to beat of song	Not Started	CDR 2/15

Table 1: Listing of MDR Deliverables

Generally speaking, our team works well together. There have been times where individual commitment is challenged, but we bounce back and come together with motivation and energy to get the project working. Each member possess unique abilities that makes focusing on individual parts natural. And beyond our individual responsibilities within our project, we assist each other during group meetings. Scheduling and general communication is done via GroupMe, which is essentially a group SMS application, so every team member gets information sent immediately to their phone.

We have weekly meetings with our advisor Professor Kelly and as group every Thursday. Our group meetings largely consist of discussing our progress and discussing technical issues we're struggling with. Arseny has a strong ability in algorithm development; consequently, he's handling the difficult problems of beat and dominant key recognition. Sade is our organization savior by noting important comments and keeping the rest of the team informed. She will deliver the Wep App interface and Shairplay functionality. Justin acts as the manager and has a strong ability to program. He is delivering the live stream functionality and coordinating playback and light controls. Alden Michaels is the technical backbone of this project, with deep knowledge in most aspects of the project. He delivered a functional LED Module by MDR, and keeps our group in line when the urge to slack creeps in.

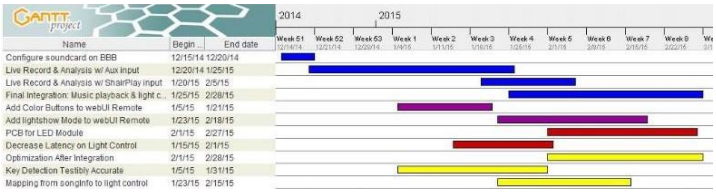

Figure 8: Gantt Chart for Proposed Progress

IV. Conclusion

InteLEDs is at a reasonable halfway point. All of the major subsystems are functional but there are a couple of features that must be delivered (live stream and key detection). Our plans for future progress are outlined in the Gantt chart (figure 7). We plan to achieve a significant portion during winter break, so we may come together as a group and focus exclusively on integrating the system and all the unforeseen issues associated with that process. Each member will describe his status, accomplishments, and future goals below.

Currently for Block 1, the user is able to access the website and choose a specific color from the color wheel for the LED lights. With the help of Alden, we were able to accomplish this task. For CDR, Sade plans to add more control features such as individual color buttons to minimize the lagging that occurs from the color wheel. She also plan to create a light show demo for the user. This will allow the user to choose a specific light choreography instead of flashing to the beat of the music. This is similar to how Christmas lights work. Sade is currently researching light patterns of Christmas tree lights and how they are developed.

Block 2 meets all of the proposed goals, but exists solely on a breadboard. Additionally, a Linux bluetooth program is still being used as the intermediary from my groupmate's Python scripts to the USB Bluetooth radio attached to the Beaglebone Black. Thus, two main things to remain for the coming semester; one will be to minimize the latency of the BBB software side of my project, bypassing the intermediary tool and talking directly to the kernel driver, and the other is to move the LED Module onto a custom PCB. Currently, the BBB must handshake with the LED Module each time it changes color, this is a result of using the intermediary program. By writing a custom library and talking to the kernel driver directly, we can greatly reduce the latency by only performing the handshake at the stand of a session, rather than on every color change. The PCB will reduce the footprint from roughly the size of a sheet of paper, to roughly one half of a dollar bill. It will include an integrated 2.4GHz PCB antenna for BLE communications, an ARM Cotrex-M0+ 32 bit microcontroller, and all the power electronics to handle the large LED load. All components will be surface mount, save for the barrel jack power connector. This mimics real life LED controllers, and would be the most cost effective option in a real manufacturing scenario.

Block 3's live stream is a critical feature, and needs to be functional by the end of January. Difficulties can certainly be foreseen adapting songAnalysis.py (Arseny's block) to work with the live recording, due to it constantly receiving song segments instead of reading an entire .wav file. Also, a kernel interface might be necessary to take the audio information sent from shairplay in order to send it to block 3, which will certainly add complications. However, we believe we can record the song in the same way as it's done with the auxiliary input. Once we can record and analyze continuously, we can start the final system integration: writing the controller.py program shown in figure 5, that will playback the previously recorded songs and send signals to control the LEDs. Lastly, the USB soundcard must be configured to work on the BBB, which will be done before leaving for winter break.

The current state of block 4 is a completed beat detection algorithm, which can be further optimized for speed and accuracy. Integration with lights is needed, along with colors from an uncompleted key detection. Running into difficulties with overtones and undertones could potentially leave the need for randomly selected colors. Creating different algorithms for different lightshow patterns is also a need.

References

[0] Eternal Lighting, "Eternal Lighting CUBEecho RGBWA DMX" CUBEecho-DMX RGBWA datasheet.

[1] American DJ Technical Staff, Micro Galaxian Instruction Manual, American DJ.

[2] Shafner, P. O. "Epilepsy Foundation." *Photosensitivity and Seizures*. 1 Nov. 2013. [Online]. [Accessed 15 Dec. 2014].

[3] L., Michael. "How to Choose the Right Platform: Raspberry Pi or BeagleBone Black?" *Makezine*. 25 Feb. 2014. [Online]. [Accessed 15 Dec. 2014].

[4] Juhovh. "Juhovh/shairplay." GitHub. GitHub, n.d. [Online]. Available: https://github.com/juhovh/shairplay. [Accessed 21 Sept. 2014].

[5] "Play Content from Your IPhone, IPad, IPod touch, or Mac on Your HDTV." *Airplay*. Apple, n.d. [Online]. Available: http://www.apple.com/airplay/. [Accessed 13 Sept. 2014].

[6] Crawford, Stephanie. "How AppleAirPlay Works" 27 June 2011. HowStuffWorks. [Online]. Availbe: http://electronics.howstuffworks.com/airplay.htm [Accessed 13 Sept. 2014.]

[7] Nordic Semiconductor, "Multi-protocol Blueototh Low Energy and 2.4 GHz proprietary system-on-chip," nRF51822 datasheet.

[8] J. Shelton and G. P. Kumar, "Comparison between Auditory and Visual Simple Reaction Times," SciRes Neuroscience & Medicine, Vellore, India. Final Rep., Aug. 7, 2010.