Laser Harp

Vlad Slyusar

Jay Lindland

ECE 4983/4 ELECTRICAL/COMPUTER ENGINEERING DESIGN

Winter 2016

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

THE UNIVERSITY OF MICHIGAN-DEARBORN

Evergreen Road, Dearborn MI 48128-1491

Tel: (313) 593 - 5420 Fax: (313) 593 - 9967

TABLE OF CONTENTS

ABSTRACT XXI

ACKNOWLEDGEMENTS XXII

1. INTRODUCTION *(Page 1)*

2. PROBLEM STATEMENT *(Page 1)*

3. TASKS AND DESIGN ALTERNATIVES *(Page 2)*

3.1 Design Tasks of Each Member *(Page 2)*

3.2 List of Specifications and Engineering Standards Employed *(Page 3)*

3.3 Discussion of design alternatives *(Page 6)*

3.4 Discussion of design evaluation and measurements *(Page 9)*

4. DESIGN CHOICES AND PERFORMANCE CRITERIA

4. DETAILS OF DESIGN *(Page 10)*

4.1 Design Tasks for Each Team Member *(Page 10)*

4.2 FINAL SYSTEM *(Page 17)*

4.3 TEST RESULTS AND DISCUSSION *(Page 19)*

5 NONTECHNICAL ASPECTS OF DESIGN *(Page 22)*

5.1 Detailed Cost Analysis *(Page 22)*

5.2 Economic Benefits *(Page 24)*

5.3 Ethics and Safety Issues *(Page 26)*

5.4 Lifelong Learning *(Page 28)*

5.5 Contemporary Issues and Global Impact *(Page 29)*

6 CONCLUSIONS AND EXECUTIVE SUMMARY FROM EACH TEAM MEMBER *(Page 31)*

7 REFERENCES *(Page 34)*

8 APPENDICES *(Page 36)*

8.1 APPENDIX 8.1: ORIGINAL PROJECT PROPOSAL (First Semester) *(Page 36)*

8.2 APPENDIX 8.2: DETAILED PROGRESS REPORT (FIRST SEMESTER) *(Page 39)*

8.3 APPENDIX 8.3: POWERPOINT HANDOUT SLIDES (Final Presentation) *(Page 55)*

8.4 APPENDIX 8.4: PRELIMINARY EFFORTS – PRELIMINARY EFFORTS *(Page 59)*

8.5 APPENDIX 8.5: SOURCE CODE - ITERATION 1 *(Page 77)*

8.6 APPENDIX 8.6: SOURCE CODE – ITERATION 2 (Testing DSP) *(Page 86)*

8.7 APPENDIX 8.7: SOURCE CODE – ITERATION 3 (Senior Design Demo Code) *(Page 90)*

8.8 APPENDIX 8.8: ADDITIONAL SOURCE CODE AND LIBRARIES *(Page 98)*

8.9 APPENDIX 8.9: ONE PAGE VITA *(Page 144)*

8.10 APPENDIX 8.10: ONE PAGE VITA *(Page 145)*

XXI. ABSTRACT

Laser Harp

Project Team:
Vlad Slyusar
Jay Lindland

Project Advisors:
John Miller, Ph.D.
Samir Rawashdeh, Ph.D.

Project Funding:
JVTech LC.

Abstract:

This report presents the process of developing a low-cost laser harp synthesizer. The project relies on an array of 8 laser dot diodes each activating a detection circuit based on a phototransistor (Figure 1). Interruption of any of the beams signals a Teensy microcontroller to output a programmed sound effect. The sound is processed on an audio board with a DSP chip and output to a standard 3.5mm connector (Figure 2). Additionally, with the application of a MIDI driver, the Teensy directly integrates with most commercial music and DJ applications, such as FL Studio, allowing the harp to simulate hundreds of instruments and control digital effects. The novel aspect of this project centers on the fact that no low-cost laser-based synthesizer currently exists on the market and the conducted research paves the way for mass production.

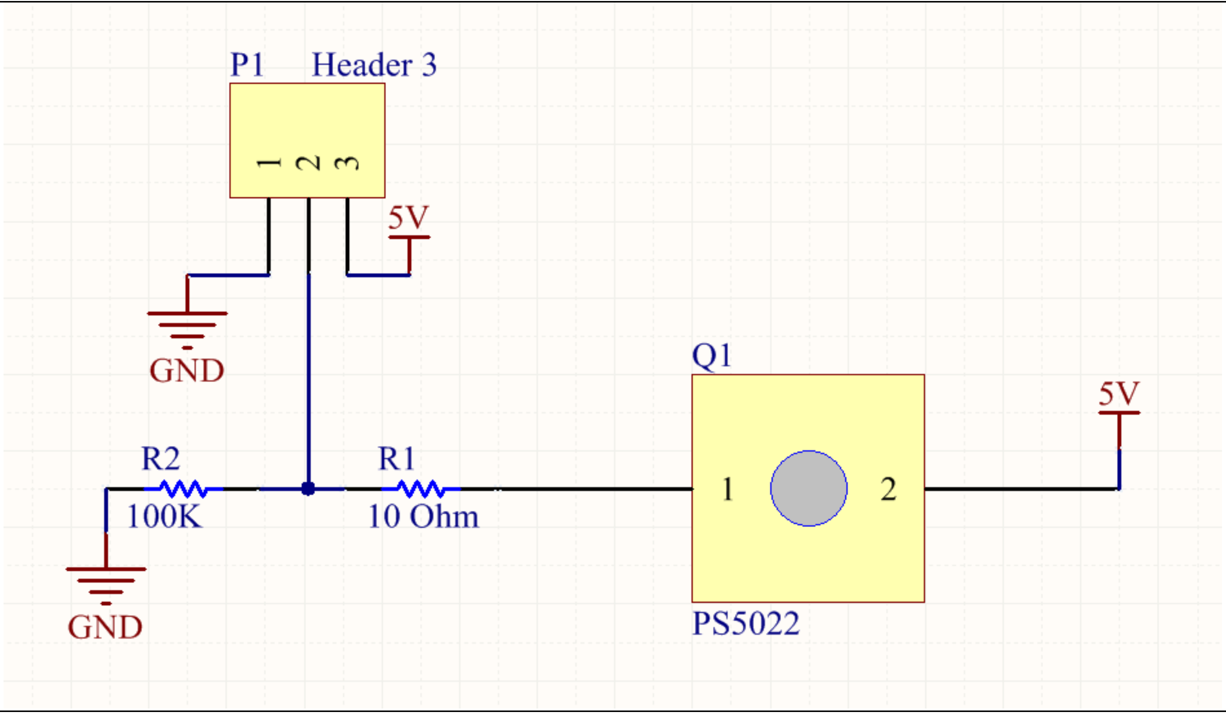


Figure 1. Laser Detection Circuit

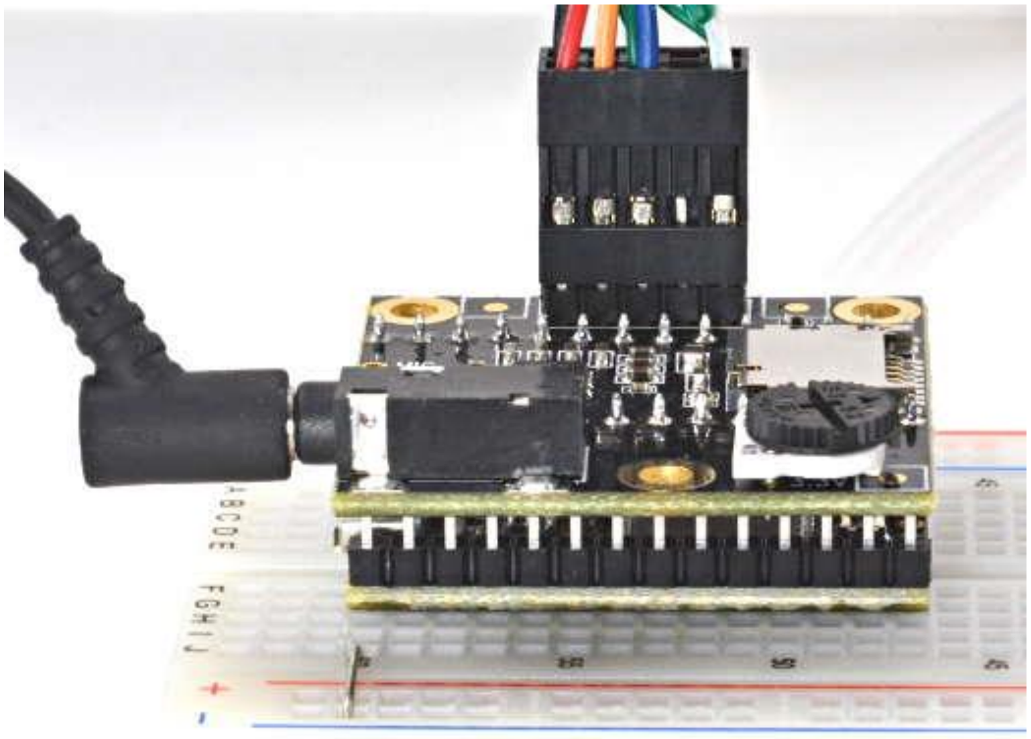


Figure 2. Teensy Board Connected to Audio Adapter with Sound Output

XXII. Acknowledgements

We would like to acknowledge Dr. Samir Rawashdeh for providing valuable advice regarding the project design as well as Dr. John Miller for help with initial project planning. Furthermore we want to thank Osram for providing sample laser diodes and JVTech LC for funding the research and development of the project.

1. INTRODUCTION

This senior design group consists of Vlad Slyusar, and Jay Lindland. The project is essentially a digital synthesizer that relies on the use of laser dot diodes that activate detection circuitry. When the circuit detects an interrupt, this sends a signal to the microcontroller and lets it know to play a sound. The outcome of this design is a low cost instrument that can be customized to play different sounds per the user's choice. To gain experience within the sphere of product design engineering, the end goal for the project was chosen as delivering a market-ready product at a significantly lower cost than all competition in the market. The general approach was to use low-cost components and apply cost-centered engineering design principles as discussed in the sections below.

2. PROBLEM STATEMENT AND GENERAL DESIGN APPROACH

The initial idea for the project focused on creating a low cost laser based synthesizer designed for mass production for the consumer market. Thorough search of global online markets for laser based instruments showed virtually no competition, with the exception of several multi-thousand dollar products. No company in the world currently sells a laser harp type product at an affordable cost, and that is the problem JVTech LC set out to solve. A goal of under $100 was set as a target sale price to draw in consumers and still generate a profit. Through considerable research, practice, and ideation, design techniques were selected and optimized on the path to creating a stylish, full featured, and low cost laser harp. Despite the requirement to meet a cost target, numerous engineering constraints were defined for benchmarking project success. Research on laser diode regulations and safety led to the requirement of using lasers with a power draw of 5mw or less. Although the laser beams would not be visible at such power levels without fog, haze, or smoke particles in the air, their operation does not require protective eyewear. Perhaps the most challenging engineering constraint centered on laser alignment and detection. Detectors had to work in night/daylight conditions, but also allow a high degree of tolerance for laser alignment, which required a careful balance of the detection threshold. As for the total power limitations, standard USB 2.0 ports only provide 500mA of power at 5V thus the total drain from the lasers, detectors, teensy, and DSP was limited to 2.5 Watts. In terms of sound, a critical requirement of any digital instrument centers on minimizing auditory lag. The maximum latency for the laser detection boards was set at 2ms, and the maximum delay for processing the sound sample on the DSP or computer was set at 5ms. This allows a total maximum audio lag of 7ms between playing a string and hearing sound. This constraint influenced initial research for a fast DSP, fast memory for storing audio samples, and a speedy communication to control the digital signal processor. The implementation of MIDI drivers allowed the project to interface with DJ software and rely on the user's computer for digital signal processing providing a cost-saving alternative design without the DSP board and memory chip. The project frame size constraint was derived from manufacturing machinery size limits. The entire frame had to be cut from a single sheet of acrylic for maximum efficiency in manufacturing. The maximum size sheet to fit inside the working region of the CO2 laser cutter used for the project was 24"x18" and set the overall limits on frame dimensions. Furthermore, after defining the manufacturing strategy for the final product as a kit, the frame assembly requirements heavily guided the direction of final design. The final frame had to be easily constructed without the use of glue while also providing a high degree of structural integrity.

3. TASKS AND DESIGN ALTERNATIVES

The following sections break down the work performed by each partner as well throughout the term. Design specifications for measuring project success as well as the engineering standards are also provided along with alternatives to major aspects of design. The testing and evaluation procedure for all major parts of the design are also provided.

3.1 Design Tasks of Each Member

The tasks for Vlad Slyusar are as follows:

1. Perform microcontroller research and integration into project
2. Design and manufacture printed circuit boards for laser detection

3. Develop project software

The tasks for Jay Lindland are as follows:

1. Design the laser harp frame
2. Research power and battery options
3. Design wire harness and power distribution network for lasers and detection boards
4. Solve laser alignment issues for product manufacturing
5. Laws & Regulations Research

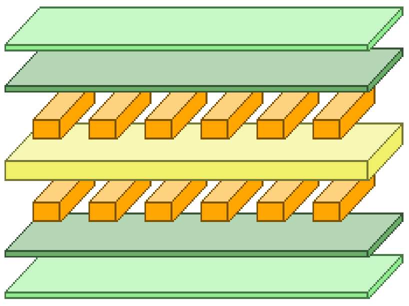
3.2 List of Specifications and Engineering Standards Employed

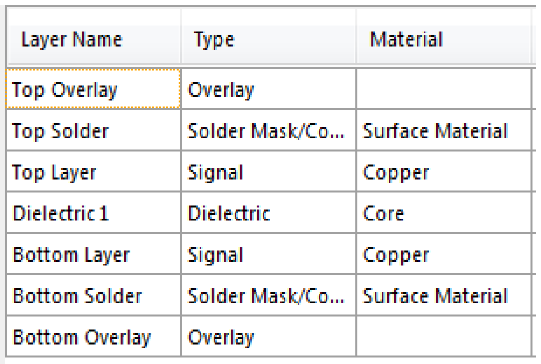
Specification Table	**Proposed**	**Achieved**	**Criteria Met?**
Total system current draw at 5v	Under 500mA	Maximum recorded current draw: 115mA	Success
Laser power	Under 5mW	5mW (0.2mW tolerance)	Success
Detector Boards	Fully operational in daylight conditions, responsive to partially aligned lasers	Detectors are not set off by daylight, not even phone flash light, but even part of laser beam does.	Success
Detection Latency (Phototransistor react + teensy input read)	Under 2ms	100us (20us tolerance)	Success
DSP Playback Latency	Under 5ms	500us-2ms (Dependent on sound sample)	Success
Frame Size	Cut from single 24”x18” sheet	Total dimensions: 23”x11”	Success
Frame Construction	High structural integrity, no requirement for adhesive	Locking pin mechanisms (acrylic) hold frame together firmly (4x)	Success

Final Product Cost	Under $100	$90 (at gross marginal profit of 30%)	Success

The following section breaks down engineering standards relevant to the project:

- SPI – Serial Periferal Interface. Used for communication between teensy microcontroller and audio adapter board. SPI is a synchronous serial communication bus with dedicated send and receive lines, a clock line, as well as a dedicated chip select line.
- Teensy 3.1/3.2 – 32 bit ARM Cortex M-4 Processor based microcontroller clocked at 72 MHz. The main difference is that the 3.2 version includes a more powerful 3.3 volt regulator. Peripherals, memory, and processor are identical for the two boards.
- Teensy Audio Board – SGTL5000 audio codec chip from NXP. Onboard DSP performs real-time FFT calculations necessary for outputting audio samples. This is an improvement to single tone output previously generated from a PWM pin driving an amplifier.
- Serial NOR Flash Memory – W25Q128FV chip, 128Mbit. Used for storing audio samples for direct sound synthesis.
- MIDI - Musical Instrument Digital Interface. USB output to computer for use with DJ studio software such as FL Studio.
- RCA audio – 3.5mm female adaptor to fit standard headphones/speakers.
- Laser Dot Diodes
  - Class 3R (IEC 60825-1 Standard) – 5mW power rating
  - Comply with CFR - Code of Federal Regulations Title 21 1040.10 and 11. (Qualify as pointer devices).
- PCB Design
  - Altium PCB Design Suite
  - 2 Layer Board



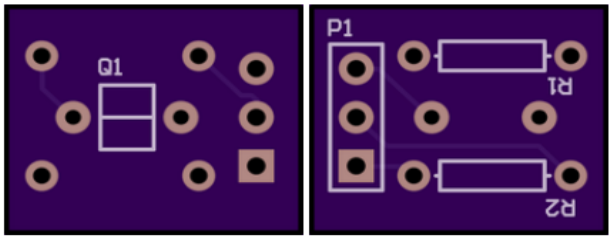


Layer Name	Type	Material
Top Overlay	Overlay	
Top Solder	Solder Mask/Co...	Surface Material
Top Layer	Signal	Copper
Dielectric 1	Dielectric	Core
Bottom Layer	Signal	Copper
Bottom Solder	Solder Mask/Co...	Surface Material
Bottom Overlay	Overlay	

Project Layer Stackup

- Trace Width: 10mil track diameter, 6mil minimum allowed by board house.
- Trace Spacing: 10mil minimum spacing between tracks.
- Drill Size: 13mil minimum hole
- Annual Ring Size: 7mil minimum specification for area on the pad surrounding a via.

- PCB Manufacture
  - OSH Park Fabrication Facility



Board Design Files Interpreted by OSH Park System

  - FR4 grade, glass-reinforced epoxy laminate (flame retardant).
    - In compliance with UL94V-0 standard. (Plastic flammability with Underwriters Laboratories). Material burning stops within 10 seconds on a vertical specimen.
  - 170Tg (glass transition temperature) / 290Td (deposition temperature)
  - Lead Free Process – ROHS Compliant
  - ENIG finish surface plating (Electroless nickel with immersion gold to protect from oxidation)
  - 1.6mm final thickness, 1 ounce copper on each side

3.3 Discussion of design alternatives

High integrity frame ready for production:

- Wood construction
- 3D Printing
- Laser-cut acrylic
- Thermal Injection Molding

The project frame started out as a wooden construction held together by screws. The frame pieces were cut using a band saw and a drill press was used to create 6mm holes to house the laser dot diodes. Small copper perfboards were built by hand to accommodate the laser detection circuitry. In the second iteration, the design was updated to use a ¼ inch acrylic frame etched out of a 24”x18” sheet of plastic cut out on a 60W Epilog CO2 laser cutter, held together by industrial hot glue. The frame was designed as a 2-dimensional hairline image using vector graphics software Corel Draw X5. Moving away from a wooden frame prototype to plastic increases aesthetic design value and allows for more accurate and automated part production. The third and final iteration of design focused on the product manufacturing strategy. The frame was fully recreated applying a puzzle-piece design that can be assembled by the customer. The design choice resulted from the decision to produce the product as a kit, until funding is raised to mass manufacture the frame using a thermal injection machine and hire personnel to assemble the product. The 3D printing option was initially considered, and a CAD model was constructed, however size limitations on the available equipment as well as well as printing resolution and long printing time, and high error rate proved unfeasible for the project goal.

Lasers:

- Green handheld laser pointer
- Red laser dot diode - 650nm 5mW 5V module head
- Industrial green laser diodes with driver circuits – 5mW to 50mW, 532nm
- Osram Samples - PL 520 Green visible lasers TO38 packaging, 520nm, 50mW

Since the unique aspect of the project centers around employing laser beams as strings, a lot of research was devoted to selecting the laser diodes for the project. Initial idea of disassembling green handheld lasers was crushed by duty cycle specifications allowing the laser to be on for only minutes without custom heat sinks. Laser dot diodes themselves were researched and unfortunately no green laser diode modules were found for under $10/piece. Red laser dot diodes, on the other hand, cost almost 2 orders of magnitude less. The initial build was performed with a bulk order of 650nm 5mW 5V laser dot diodes which carried through to the final design. At a cost of under 30 cents/unit, viable alternatives were difficult to find. Sample diode heads were also provided by Osram, but required external driver circuitry as well as heat sinks bringing the total price per unit well over $30.

Laser Detection

- Photo resistor/Photodiode/Phototransistor
- PCB with through-hole components
- Fully assembled SMD PCBs

Although the light detection principle was simple in nature, the right device had to be selected for the design. Since response speed was a vital factor, photo resistors were abandoned for their faster counterparts. The choice between transistor vs diode required a tradeoff between higher sensitivity and slight speed improvement. Since the design constraint called for a large alignment tolerance of the laser beams, phototransistors were selected for their higher sensitivity to allow proper detection even when the laser only partially aligns. Once circuitry was designed, breadboard tested, and prototyped, custom PCBs were designed using Altium designer and manufactured at OSH Park board house. The current latest prototype of the project utilizes these boards; however, since they require self-assembly this approach does not work for a product outside of kit requiring soldering. For future iterations, the PCBs require redesign with SMD components and mass volume production including assembly. This step was not yet completed due to high investment costs for board assembly.

Low cost, small and effective electronics core:

- Tiva C
- Teensy (Models: LC, 3.1, 3.2)
- DSP – SGTL5000 NXP audio codec
- External Memory for sound storage
    - SD card – Standard class 10 microSD
    - Serial Flash Memory – W25Q128FV Serial NOR with quad SPI
- MIDI – Directly simulate standard electronic music instrument over USB

On the electronics side, the heart of the project relies on a microcontroller to read sensor inputs and generate sound for each interrupted laser beam. Initially a Tiva C microcontroller was used, but the much smaller and popular teensy micro was selected for the new prototype. While both microcontrollers are built around a Cortex-M4 processor, the Teensy's incredibly small size as well as a versatile selection of hardware shields proves more effective to meeting the goals of this project. Three versions of the teensy were ordered and tested: Teensy 3.2, 3.1, and the LC low cost model. Unfortunately many issues were encountered with the low cost version of the board when trying to setup an SPI channel to interface with the audio board. Integration would require custom routing of the signals instead of a simple header stackup. Both the 3.1 and 3.2 however performed well and were both selected as candidates for final design depending on availability and cost at the time of large volume production. As far as the audio board, the initial idea of relying on an SD card for audio storage was rejected to keep costs down and the W25Q128FV Serial NOR Flash memory chip was selected as a viable low cost alternative. Finally, the development of drivers for MIDI protocol and full integration with commercially available DJ software provided a significant cost savings approach which would completely eliminate the audio adapter board and allow the low cost teensy to be used once again. Although the project direction changed significantly at different stages of design to bring down system cost, the process itself provided valuable experience in product design as well as experience with a wider variety of subsystems than included in the final project.

3.4 Discussion of design evaluation and measurements

Total power draw specification was measured for the third and final prototype. A lab bench power supply was used to provide 5V to the Vin port on the microcontroller and ground pin was connected for return path, the power supply also provided the value of output current. Since the board provides power to both lasers and detection circuitry, the single point of power allowed for measuring the total current draw all at once. Since the measurement goal was testing maximum power draw, random math computation loop was flashed to the board before the test which would guarantee CPU usage and prevent sleep mode which would otherwise conserve power. The maximum current drain measured was 115mA which corresponds to 0.575mW power draw. This is well within the 2.5mW maximum previously set.

Laser power was tested in a similar fashion but repeated for 4 individual diodes each averaged over a 20 second cycle. Total power draw ranged from 4.8mW to 5.2mW depending on the lasers tested. The detector boards were connected to analog inputs on the microcontroller to provide accurate feedback as multiple sources of light were applied. With the biasing resistors adjusting phototransistor operation, standard sources of light failed to provide greater than 14% of the voltage at the input. Laser beams, on the other hand, provided nearly 100% in perfect alignment and well above the 30% range for most alignment offsets tested. Thus 25% was set as the cutoff value in software providing the perfect balance. This corresponds to ~0.8V of the 3.3V applied to the collector of the phototransistor.

To test detection latency, the lasers were provided power and shined directly at the phototransistors on the boards. The power bus providing current to the boards was configured as an output pin and a timer was used to measure the time (derivative of processor cycles dependent on clock speed) between providing power to the boards and receiving a signal from each. The highest measured time was 118us which was well within our 2ms target. Testing the DSP playback latency was much more challenging. Sound samples between 3sec to 30sec were loaded into memory before the test. The challenge was syncing the play signal to analog sound output. This was achieved by using 2 oscilloscope probes, the first was connected to a digital output pin which got enabled as soon as the chip send SPI command to DSP for playback of sample. The second probe hooked up to the solder joint on headphone jack output. Time between the start peak, and higher amplitude readings on the analog output line was recorded as the latency. Depending on multiple measurement factors and type of sample, the delay ranged between 0.5-1.8ms. Since microSD cards would not be used in final design due, their access speed was not tested.

For the frame construction, the final design successfully implemented a puzzle-piece structure that required no adhesives and does not fall apart due to employed use of locking-pin mechanisms. The total dimensions of the finished product came out to 23”x11” fully cut out of a single sheet 24”x18” in size. The final and most important specification to the original intent of the project was meeting the cost target of under $100 to consumer. With the MIDI-only version of the product, the final BOM cost comes out to $38.59. Production cost includes 1 hour laser cutter time and packaging and shipping totaling $25/project. With a 30% gross marginal profit total product price comes out to $90.84 (More info in cost analysis section).

4. Details of the design

The following sections provide the complete details of the project design. Additional content is provided in the appendix section.

4.1 Design Tasks for each member

Vlad

Throughout the senior design term, significant progress was made within the sphere of engineering to make the project operational and develop improvements after thorough research and experimentation. The first iteration of design employed a Tiva C board, which was large and could only play single tone output through a pulse-width modulated pin fed through a small audio amplifier. After significant research, the Teensy was identified as a potential prospect and multiple Teensy microcontrollers were obtained for evaluation. An image of the board is provided below illustrating the relatively small size with the micro-USB interface on the left side as reference.

Teensy 3.2 Microcontroller

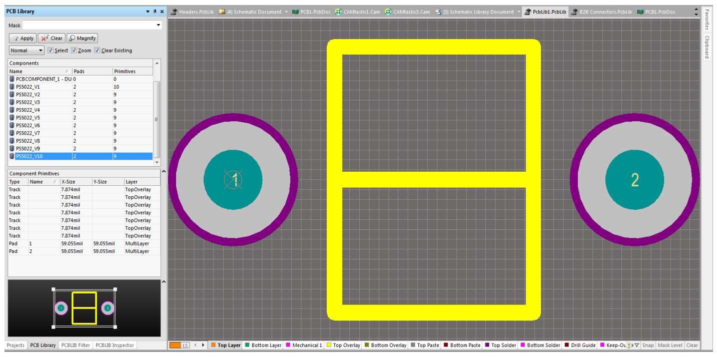
To play real sound samples, a powerful digital signal processor was needed. Two copies of the audio shield from PJRC were also ordered for experimentation. The image below illustrates the physical construction of the audio board showing both a 3.5mm RCA audio interface as well as microSD card slot.

Teensy Audio Adapter Board

The major difficult task relied on setting up a communication channel between the micro and audio adapter using the SPI protocol. Once communication was successfully established, the next major challenge was selecting the type of music output and physically achieving the result. Many techniques were tested including playing WAV files from an SD card formatted in FAT32, FAT16, and extFAT. Latency issues with reading bandwidth caused problems playing more than 3 strings at a time and the cost of upgrading to a faster SD card would increase project cost or require imposing limitations on end user SD card purchasing.

As a result, the idea was completely scrapped for a $1.63 flash memory chip replacement to be optionally added in the final design.

As far as the laser string implementation, a simple design was developed relying on strong beams of laser light activating detector circuits. This major deviation from Theremin based designs with expensive lasers and sensors is precisely why the project goal focuses on low cost design as compared to multi-thousand dollar instruments. For this reason distance sensing functionality was no part of the design. A PS5022 phototransistor was used at the heart of the detection circuitry with 10 Ohm and 100K resistor for biasing the circuit to react only to the laser light. Experimental method was used in determining the values successfully eliminating sensitivity to daylight and even the brightest room lighting while at the same time providing an error margin for imperfect laser positioning. In other words only the laser triggers the phototransistor, even if only partially aimed at its surface. Bright light from cell phone flash was still ineffective in triggering a response from the detection circuits. As far as the physical circuitry, the route of manufacturing custom PCBs was chosen to gain knowledge and experience as well as simplify prototype manufacturing efforts. Altium Designer Suite software was used for creating custom components (phototransistor), drawing up schematics, creating accurate component footprints, and performing the circuit layout. The image below illustrates the interface for creating a custom footprint designed for the phototransistor. Sizing specifications for physical dimensions were carefully followed to ensure product would properly fit.



Components

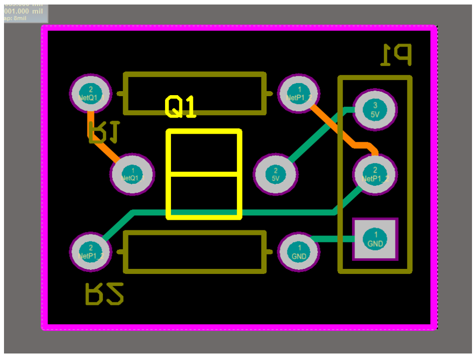
Name	Pads	Primitives
PCBCOMPONENT_1 - DU	0	0
PS5022_V1	2	10
PS5022_V2	2	9
PS5022_V3	2	9
PS5022_V4	2	9
PS5022_V5	2	9
PS5022_V6	2	9
PS5022_V7	2	9
PS5022_V8	2	9
PS5022_V9	2	9
PS5022_V10	2	9

Component Primitives

Type	Name	X-Size	Y-Size	Layer
Track		7.874mil		TopOverlay
Track		7.874mil		TopOverlay
Track		7.874mil		TopOverlay
Track		7.874mil		TopOverlay
Track		7.874mil		TopOverlay
Track		7.874mil		TopOverlay
Track		7.874mil		TopOverlay
Pad	1	59.055mil	59.055mil	MultiLayer
Pad	2	59.055mil	59.055mil	MultiLayer

Altium Custom Footprint Creation

Multiple iterations of design led to the decision of placing the phototransistor on the top side of the board to face the laser and resistors and header on the bottom for connectivity. This design approach creates a maximally flat surface at the top of the PCB allowing simpler fixation to the frame. The image below represents the finalized design with bright outline components and orange tracks representing top layer and the rest bottom. Since through-hole components were used for each part, no excess VIAs were required.



Laser Detection PCB Designed Using Altium

As far as fabrication, the design rules were first checked to ensure compliance with manufacturing minimums of board house. Next, drill files were generated to comply with OSH Park standards and several attempts were made until design was finally accepted. An important part not automatically added by Altium Designer is the board outline the lack of which caused all sorts of issues. Once the small problems were resolved, an order was placed and the boards arrived within 2 weeks.

The software aspect of the project breaks down into two major categories: testing software and software used in the final design. Since the initial direction of the project included the digital signal processor, testing level code was created for setting up SPI communication with the DSP, programming sounds into memory, and controlling the playback functionality. The software was successful at showcasing the functionality of hardware design, but due to extra cost of over $20 for the DSP as well as upgraded teensy required to work with it, the implementation was dropped for the consumer build. The DSP currently remains as an optional add-on to be implemented in the future depending on consumer requests. As far as the production project, successful implementation of MIDI drivers for the Teensy board allowed us to send MIDI encoded notes to commercial DJ software (FL Studio). After significant research on music theory, notes, and the MIDI standard, the software flashed to the Teensy would send on and off signals for notes corresponding to each of the laser beams. Serial input allowed control of variables to manipulate the octave offset allowing the simulation of multiple sections of the scale; however, the next code iteration added a custom functionality that currently does not exist in any digital instrument on the market. We call it cheat mode, but in reality it provides functionality similar to auto-tune for voice artists. Enabling cheat mode level 1 through serial command (currently set as 11) will help the user perform nearly any song flawlessly. The software keeps track of notes and automatically switches octaves. Cheat mode level 2 also keeps track of proper timing for each note. At the time, only the song Fur Elise by Beethoven is programmed for the assistance function, but future iterations will allow integration of almost all sheet music files through a custom conversion program. All software was written using C language and is provided in the appendix section of report.

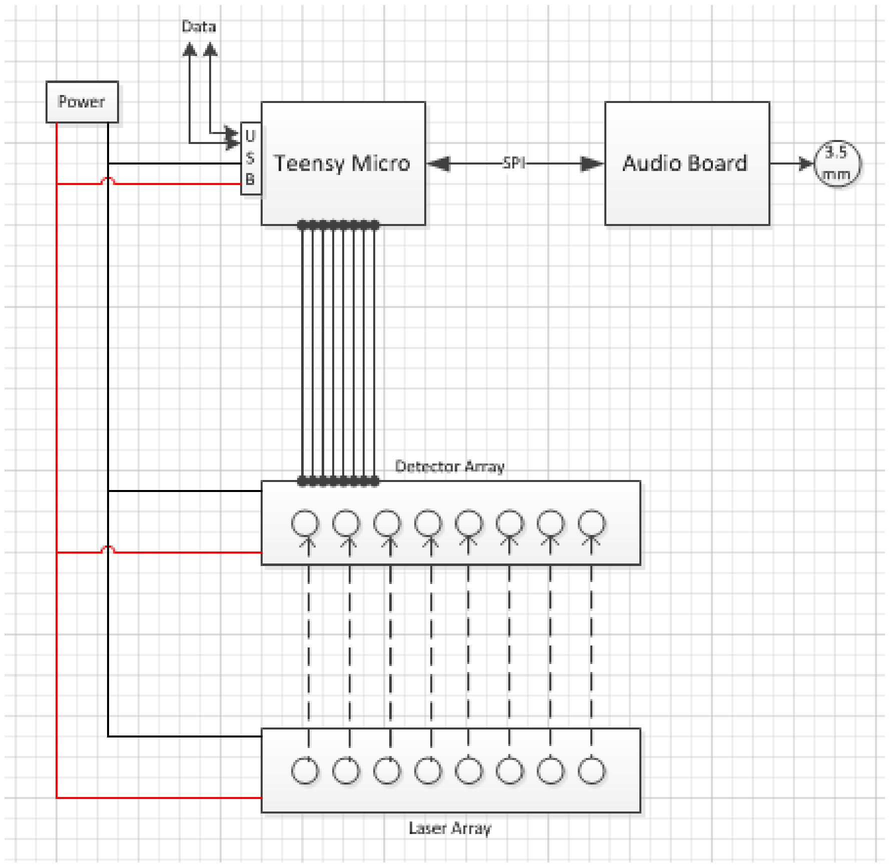
Jay

As far as the physical frame, ¼" acrylic was selected due to availability as well as pleasing surface. Clear plastic was used due to lower costs and to provide an informational prototype. Corel Draw X5 was obtained and used for designing the outline for frame components. An Epilog CO2 laser cutter was used at Tech Shop facilities to perform clean and accurate cuts. Outside of general debugging and researching parts, the final bulk of work involved the frame construction itself. Acrylic sheets were obtained from the laser cutter etched to size per design. Significant problems were encountered using hot glue as a binding agent as the working time was very short. Furthermore, because of the clear acrylic used, uneven glue marks could be seen from the outside and masking tape was added at the project's corners as a fix. Once the frame was complete, work began on assembling the electronic components. Power and ground bus wires were routed through the top and bottom portions of the frame. The lasers and phototransistor circuits require 5V and 3.3V respectively to as well as a current return path to operate. Furthermore, individual wires had to be cut to length and stripped for each of the 8 detection circuits. The assembly of custom PCBs proved much more pleasant than the initial perf-board prototypes and several design improvements were noted for future boards. A large improvement in manufacturing time involved an idea to place the boards on the bottom plane rather than affixed to the top of the acrylic. Then a single strip of Velcro can be applied to the entire line and each board can be easily positioned and firmly affixed in place directly in view of its laser. Finally, wires were routed out to the temporary setup with teensy and audio board.

It was clear that the initial design was only purely functional. A more optimal approach was then researched. The main idea going into the design for this was the ability to construct the frame without the use of glue, and with minimal effort. Another aspect that had to be considered was how easily it could be manufactured with as little waste as possible. Overall, these factors were the driving force behind the design of the new frame. For it to be marketable, it had to be appealing to the customer, and also be efficient to produce.

A way to design a frame that would stay together without glue proved to be somewhat of a test of ingenuity. Several ideas were considered on paper before trials were made with the laser cutter. The main idea behind the new frame turned into having pieces that would interlock and then stay in place. The size of each interlocking mechanism had to be precise so that the frame would stay rigid. Testing was then done to cut connector pieces ~~out~~ using the laser cutter. The edges where the frame would come together were designed using Corel, and cut on the CO2 laser cutter. Until a tight enough fit between the pieces was achieved, the tolerance on the interlocking pieces was lowered. Once a way to achieve a perfect fit between the two pieces was achieved, the final design of the frame could start.

The final design of the frame came out something like puzzle pieces. The whole frame was cut from the same sheet of acrylic, and each individual piece was interleaved with ~~all of~~ the pieces next to it on the sheet. This design allowed for the most optimal use of space on the acrylic sheet providing us with the least amount of waste. Once the frame was cut by the laser cutter, it could be broken up into individual pieces and assembled. The main assembly of the frame was broken up into four parts: the upper structure, lower structure, and two side structures. Each structure has four walls that interlock, and come together to make the whole structure. The top and bottom structures could them be connected to the side structures, and locked in place with a small pinning piece.



Project Block Diagram

With a complete frame assembled, the next thing to implement was the power distribution. The figure above shows all of the data lines, and power lines that would have to connect throughout the whole frame. Having wires run through the whole frame was definitely not the most optimal solution. The design for this frame was to have the easiest assembly possible for the customer. This is where the idea to use copper conductive tape came from. The copper tape can be put onto the correct piece of the frame before it is assembled. Once the frame is assembled, the copper tape will come into contact with the pieces on the other ends of the frame. This will proved a power bus that travels along the whole frame making it easy to provide power to anything that would need it.

The only part of the design that would still require running wires was the data lines for the detector circuits. There still have to be eight individual wires running between the teensy and the detector circuits. The easiest approach to making this was creating a wiring harness out of a ribbon cable. The point on the bottom structure for each detector circuit was measured out, and the wire for that piece was cut to length. The ribbon cable turned out to be able to have all eight detector circuits attached to it, and could easily be placed on the bottom structure of the frame.

A huge problem with the first two designs of the frame was that the lasers wouldn't align properly. The main cause of this was the fact that cheap lasers were used, and they had lenses that weren't perfect. High powered lasers were researched, but were dismissed when they would almost double the end cost of the product. So a solution had to be made in the design of the frame itself. The holes cut for the lasers themselves had their tolerance lowered so that it could be inserted, and held in place without glue. The fact that clear acrylic was used to build the frame allowed for the detector circuits to be placed within the lower structure of the frame, and not need holes to be cut. The laser would shine though the acrylic above the detector, and would still be bright enough to trigger the detector circuit on the other side. The detector circuits could then be moved to any point on the lower structure of the frame to allow for alignment. The detector circuit itself could be held in place with a strip of Velcro or two-sided automotive tape.

All of these aspects built up to a new frame that didn't require glue to assemble. The frame was designed in Corel to utilize the most acrylic possible. The frame could then be have the copper tape put in the correct spots, and have the lasers inserted into their holes. The bottom structure could be assembled by placing the piece of Velcro, or tape, and aligning the detector circuits to match the position of the laser. Once each of the structures is assembled, they could all come together and make the completed harp.

4.2 Final System

Since the goal of the project from the very start centered on a production-ready design, a clear vision of the final system has been developed throughout the course of the year. As the current progress of design, with the third iteration of prototyping, the product can be sold as a kit for assembly by consumer. The production volume is limited to 50 units a month as per limitations of laser cutter usage as set by Techshop policy. Larger production volumes for the frame can be achieved by arranging a contract with a large scale industrial shop to perform that step of production; however, the current design approach is only effective for low volume production and a new iteration of the product is required for large quantity assembly. The puzzle piece frame design kit needs to shift to a fully assembled product to reach the large percentage of consumer base looking for a finished product. From an engineering standpoint, it makes little sense to sell a final product from a kit design. Lower costs and aesthetic appeal can be achieved by a new iteration of the frame that can be mass produced faster, more efficiently, and at lower cost of materials and production. Thermal injection molding is a solid candidate for such a task, which will require finding a production house with large machinery.

As far as the electronics components in the design, the custom PCBs are ready to ship along with through-hole components to be soldered. For large volume production, the boards will need to be assembled at the fabrication house, and hence redesigned with smaller and more space-efficient surface mount components. The prohibitively high initial investment costs for pick and place machine setup and reflow profiling required to produce fully assembled boards led to the decision of postponing full board production until the next design iteration. In terms of the microcontroller, the question of purchase vs self-assembly arises once again as the cost of the board approaches $12. For any production of 1000 units or less, such a cost is very difficult to beat for a custom design, thus the consumer version of the microcontroller will be employed in final designs until production volumes heavily increase.

The aspect of packaging and shipment, as well as product assembly, will require an employee base, customer assistance, sales representatives, and a company management structure. The current design promises market-readiness, but only as a kit requiring assembly and component soldering. Although this method bypasses the requirement of a workforce, packaging and shipment are a vital necessity for product delivery. The current plan heavily depends on the flat 24”x18” acrylic sheet design. After laser etching, the entire sheet can be placed inside packaging typically used for paintings. The microcontroller, power distribution tape, laser detection wire harness, printed circuit boards, and components will be included in a small sub-package with bubble warp protection for laser diodes. Despite the low volume production limitations, the final construction provides stylish appeal and a sturdy construction once assembled. For future iterations focusing on a fully assembled product, custom packaging will need to be developed including attractive graphics and company logo, as well as standard product information. The creation of refund/return policies will be required for final product sales strategy as well.

The final major subcomponent of successful project delivery to consumer involves the development, distribution, and update plan for consumer software. The prototype software already exists, but requires a significant number of additional features before consumer release. Many of the planned features are hardware dependent, for example implementing a push-button array for switching octaves and note pitch instead of the current serial input interface used. The distribution plan for consumer software as well as future updates is heavily based around the company website. Product drivers and software will be provided in that fashion, thus eliminating the cost of CDs in the shipment of the product.

4.3. Test results and conclusions

Based on the initial idea, the main goal of the project focused on delivering a laser based instrument to market for under $100. Besides meeting the cost target, the project had to play high fidelity sounds linked to each string and exhibit low latency undetectable to the end user. The majority of more specific design principles were developed and updated along the way. Each iteration of design provided new information and often heavily shifted the direction of the final project, as with the case of excluding the DSP board for a MIDI only product. The first design involved a hand built prototype built using wood and prototyping boards. A Tiva C microcontroller was employed to read outputs from the 8 photodetector circuits and a power network was created to power the lasers and detection boards. The project was successfully integrated with a remote microcontroller (Arduino) using Bluetooth modules on each end and the Arduino was programmed to control professional DJ lighting over DMX protocol. For sound output, a pulse width modulation controlled pin was configured with an audio amplifier and successfully produced single tone output to an analog speaker. The tone frequency was linked to each of the strings and output was controlled by interruption of the beams triggering the base input of phototransistors on detection boards. The first prototype proved out the feasibility of a laser based instrument based on the novel design. Although the prototype satisfied criteria for cost and functionality, the implementation could not provide sound output beyond single tone pitch. Furthermore the wood frame did not meet the requirements for aesthetically pleasing design and production feasibility, which called for a full redesign of the entire system. A smaller, more versatile microcontroller was selected along with a digital sound processing board for handing high fidelity sound. The system was tested with low level software which successfully played sound samples stored in onboard memory. To house the new system, an acrylic frame was laser cut and held together using hot glue adhesive. Power and signal bus networks were constructed from wires and custom printed circuit boards were developed and manufactured for detection circuitry. The application of custom boards provided a path towards scaling production but the requirement of assembly for both the boards as well as the frame provided a manufacturing challenge for making the project a reality. Furthermore, the DSP code proved significantly complex and required WAV files converted to a custom format and flashed directly to memory before playback by the DSP. For implementation of a customer product, a wrapper application with a custom graphic user interface would require development. These steps were left for the next iteration and focus shifted to developing drivers for the device to communicate with commercial music software over a MIDI USB channel. The device connection required a modification of the standard serial protocol employed for communicating with the board to allow serial commands as well as sending MIDI note encoded values. Once the software was functional, the ability to send digital notes allowed for the device to simulate any instrument over any range of the musical scale. The possibilities discovered with such an implementation significantly overpowered the functionality of the onboard DSP. Rather than providing a limited subset of functionality onboard, any commercial software with MIDI support could be used along with thousands of encompassed features. Not only could each string play a single note on an instrument, the instrument could easily be changed, filtered, and modulated creating nearly infinite possibilities for end user implementation. As a result of the findings, the costly and labor intensive option of onboard digital signal processor was pushed off for potential future implementation depending on market demand. With the new protocol in place, the manufacturing problem of the frame as well as aesthetic appeal of final product had to be solved. The second version of the design used a three dimensional frame cut from clear acrylic but required adhesive binding at all the edges to keep the structure together. Since custom acrylic glue was found to have a 72 hour curing time, hot glue was used for the prototype build. Neither option provided a viable implementation for a final product as both required significant effort and hot glue binding created a displeasing appearance. At this point the direction of the project design changed to accommodate a realistic sales strategy, the product would sell as a low volume production kit for assembly and soldering by end user. Although this did not meet the initial plans for reaching a high consumer base, it provides a viable stepping stone for measuring the project’s potential within consumers. Thus the new design task centered on creating a fully autonomous frame that provides high structural integrity without the use of glue and provides a nice looking finished product. Many iterations of puzzle piece designs were created and etched out of acrylic for testing. After significant efforts and numerous failed attempts, a final prototype was completed. The research used three full 24”x18” sheets for designing the final prototype; however, it led to the creation of a fully autonomous master file to cut the entire frame from a single sheet. An image is provided in appendix reference. Besides the frame, user assembly would require laser alignment, power distribution networks for both lasers and detectors, and a signal bus for the 8 detection PCBs. The use of conductive copper tape in a creative wrap-around fashion allowed for joining multiple power rails by simple contact with pressure holding the frame together from the locking pins. The tape was easy and quick to apply and provided a power and ground line to the entire array of lasers along the top and array of detectors along the bottom. Laser alignment issues caused significant turmoil and the only viable solution required significantly more expensive laser diodes which would provide better tolerance for laser beam angle. As a result, a unique solution was implemented by employing a clear sheet at the top of the frame base, and positioning the detectors to proper location based on the laser position. Double-sided automotive grade 3M tape was used for the application, as a strip along the entire bottom plane. For the individual signal lines from each pcb, an 8-bit parallel cable was used with each wire cut to proper length corresponding to the detection board location. Hence the combination of clever manufacturing techniques allowed for a kit ready product. The true and final test of the results was playback capability and user experience metric. The cost target was met by extremely cost-conservative design decisions, but despite these measures the final project exhibited an extreme level of success and producing high fidelity sound with undetectable latency for the strings. Despite the engineering time specifications for latency previously discussed, the true test for the project revolved around consumer experience. Based on the overwhelming popularity and positive reviews from the large sample set of people during the senior design showcase, the functional aspect of product design was deemed successful.

5. Nontechnical aspects of design

Because research and development was funded by a private company (JVTech LC.) started by both parties involved in the senior design project, a flexible budget was available for product development within realistic boundaries. Hence, the cost analysis most relevant to the project focused on the bill of materials and production cost of the final product, which would have a significant effect on product popularity and size of consumer base. Product design and pricing strategies are discussed in the sections below, as well as economic growth and job creation which would result from successful implementation of a development and sales strategy for a fully assembled product.

5.1 Cost Analysis

Vlad

Although the final prototype developed throughout the design term can be produced and sold at low volumes, a significant overhaul of the entire system is required for successful implementation of a large scale production, fully assembled product. As far as the microcontroller currently employed in the design, the low cost version of the Teensy board retails for $11.65. For small volume production, custom microcontroller boards cannot compete with such pricing due to extremely expensive pricing for board assembly. Comparatively, the 18 PCBs printed during the first run cost a total of $8.10 including shipping. Quotes for full assembly boards of similar quantities ranged from 3-5 thousand due to high setup costs for pick and place machine and reflow profiling. Since the failure of any aspect of the project does not pose any significant consumer harm, a potential strategy for startup would encompass employing foreign board houses where 100 units were quoted under $1000. Low cost fully assembled PCBs would provide a significant reduction in the effort required for product assembly. With large sales volumes, the same strategy could be used for the microcontroller boards as long as significant cost advantage can be created through such means. As for the marketing strategy, the current plan for testing product popularity and consumer response hinges on the successful application of a kickstarter project. Bulk initial funds would provide the necessary startup costs to attain the cost benefits of large scale production and bulk parts ordering. Finally, a customer support structure would be required for a successful consumer product. On the software side, the company website would host the latest software updates and serve as the main distribution channel for initial software as well as updates. The website would also allow for comments and submission of user bug reports which would be used to improve the code and release new iterations.

Jay

The cost for building the frame is mostly time and material dependent once the design was completed. All that needs to be done to make frames from here on out is the acrylic sheet has to be inserted into the laser cutter, and press a button that will upload the schematic to the laser cutter and have it start. For an entire frame to be cut, it takes roughly an hour. A Techshop membership only allows for around two hours of cutting per day. Assuming only two boards a day, a $5 a day worker membership and $10 per hour to observe the laser cutter while it cuts, the manufacturing cost would be $25. The production cost of the frame and wiring components can be cut by using efficient designs that produce the least amount of waste possible. The acrylic sheet is almost entirely used up when all of the pieces are cut out, and ends up costing $18.02. The wiring and connectors used to make the power distribution for the frame, and the wiring harness for the detector circuits comes out to less than $2 for the conductive tape and ribbon cable. The lasers themselves are bought in bulk and cost around $2 for all eight of them. The whole product can be sold as a kit, so the packaging would be easy. The main focus would be to ensure that the frame has all of the components needed, and they are not damaged. All of the electrical components such as the PCBs, lasers, wiring supplies, etc. would be counted and packaged together. All of this could then be put into a box and sent to the customer. The customer support side of frame assembly would be helping people figure out how to assemble the frame, line up the detector circuits to the lasers, wire the detector circuits to the Teensy, trouble-shooting the power distribution, dealing with people breaking components of the frame and sending them replacements, and an infinite number of unseen troubles that the customer will be able to come up with. The main thing is trying to come up with a solution to their problem before they even have it. Every aspect of the product has to be broken down into its failure modes. From here we can come up with a solution for each, make this information available to the customer, and this would decrease the amount of customer support that would have to be offered.

5.2 Economic Benefits

Vlad

Because of stringent constraints placed on the final product pricing, every stage of engineering design reflected a cost-savings approach. The ideal manufacturing technique for the project would employ US based boardhouses for PCB assembly. This would provide a small stimulus to the general economy and bring more money into circulation with product sales. Unfortunately, due to a large cost gap, initial builds would attempt production using foreign manufacturers and perform cost studies encompassing quality factors and pricing to determine the best overall strategy. In the case of the microcontroller, initial builds will focus on the prebuilt Teensy LC boards currently in use by the final prototype. The parts are supplied by PJRC, a US based company in Portland Oregon. Hence the purchase of large unit volumes will provide a small boost to overall US economy. Finally, the software and support structure of the company will provide the greatest direct benefit by creating US jobs. To maintain a high quality image of the company, US workers will be employed to handle sales and customer support. If necessary, local programming efforts will be employed for performing updates and developing new features. Despite significantly higher costs, a local workforce for the software side of the project will allow simpler management and help prevent software theft commonly involved with outsourcing code development. Unlike most consumer products in the sphere of digital music instruments, the software driving the laser harp provides the optional functionality of auto-tuning user's musical abilities to perfect performance. Such a novel implementation could potentially have latent effects on deep underlying aspects of society by providing a calming and satisfying feeling of control over high quality music creation to, for example, people with depression who also lack the knowledge and experience in music theory which would be otherwise required for attaining the satisfaction of live performance. (Numerous studies link playing music to lower blood pressure and long-lasting cognitive benefits, see references section)

Jay

Manufacturing the frame has a large initial trade-off. Almost all of the prototyping was done using the local Techshop facilities. By using their machines and advertising where they were produced, they get a lot of advertising just by having us use their facilities. But only about two hours of laser cutting can be done there each day due to their membership policies. This limits the number of frames that can be made each day. This is where investing in a laser cutter would be beneficial. This would allow for as much cutting to be done each day as possible. The machine could be operated longer, and more frames could be produced. The bad side of this though is the initial cost for acquiring the machine, cost for maintenance and repair, and the potential that the next iteration of the frame wouldn't even use a laser cutter. A huge economic benefit about this project is the fact that it is sold as a kit. This requires the customer to gain deeper knowledge into how the entire thing works. They would have to understand how to wire up the detector circuits to the power bus, and then to the Teensy by making a wiring harness. They would have to align the detectors to the lasers. They would have to build each section of the frame and fit the whole thing together. Overall, they would gain a huge amount of knowledge just constructing the project and it could possibly stimulate their desire to learn more about the science or mechanics behind how it works. The best possible economic benefit behind this project is getting people interested in the endless and amazing amount of things that can be designed and built with the knowledge that you gain from engineering, and having them either enroll at a local Techshop class, or got to school for it.

5.3 Safety and ethical issues

Vlad

Since the final product design intention encompasses a consumer product, a multitude of safety constraints were considered during design. The greatest threat due to failure or product misuse was high power laser beams initially intended for the project. The design decision to employ 5mW laser pointers within the 3R laser classification (standard pointer devices) eliminates potential of lawsuits due to consumer eye damage. Nonetheless, from an ethical and legal standpoint, instructions provided with the final product will clearly prohibit staring directly into the laser beams to avoid eye damage. As far as the microcontroller and custom detection boards, the only real concern in terms of safety centers on fire prevention. During development, a Teensy module was burned through a high voltage discharge from improper integration with powerful DJ lighting. Protective circuitry within the microcontroller took the hit and fried the chip but no fire hazard was created. Final product will require extensive testing under high-stress environments to determine potential sources of combustion ranging from laser diodes overheating, to short circuits developed as a result of hardware failure. Besides fire safety, since no part of the product will be employed in safety critical applications, the main devastating result of failure of any system functions would result in an unsatisfied consumer. With regards to ethical issues concerning the system design aspect, little issues arose during the year. Since both partners shared the same vision from the start, teamwork and honesty came naturally to achieve final success. Tasks were divided evenly and utilized the strengths of each partner; hence failure to assume responsibility for any portion of design would have prevented the success obtained by the final iteration of design. Although not all deadlines during the project were always met, honesty and communication provided a method for the entire team to stay informed and structure proper plans to make up time and deliver the final product by the presentation deadline.

Jay

A major safety issue behind producing our frame is that a high powered laser is used to cut the frame. This laser can potentially cause a fire, and that is why it has to be observed throughout the entire process where it is cutting the frame. This would result in damaging the laser cutter, and potentially harming the operator when they try to put out the fire. Ultimately, the decision to sell the project as a kit was a huge ethical issue. The initial start-up cost to get everything assembled and shipped proved to be too much of a risk, so the design was switched to be something that could reasonably assembled by the customer. This creates potential safety issues while they’re assembling it because they have to solder components to the PCBs, or working on the power distribution. Once there is enough interest in the project, the PCBs will be fabricated with SMD components removing the need for the use so solder components. Very few issues happened throughout the past few semesters. There has been a drive for this project to be completed since before senior design started. This lead to both partners putting in a large amount of work whenever they could to get things done ahead of schedule. This allowed for a focus on school or other projects when it was important. The main strength of each partner was shared with the other so there was always progress. Honesty is what really made the project come together in the end because it really laid out a clear plan for what had to be done to get the project completed before the deadline. Without this level of honesty, some part of the project could have been completed to a sub-par level because the person in charge of it was too scared the other would be angry or disappointed. But there was always a high level of honesty, so it would be known right away what part of the project needed more time to be focused on, or if someone needed a little help with their part of the design.

5.4 Lifelong Learning

Vlad

Whether in the field of fringe research to discover and invent or simply improve product design or perform routine testing, the level of entry knowledge has significantly increased over the course of history. Furthermore, as a result of the constant innovation in technology, a modern engineer's lifelong success highly depends on keeping up with the novel aspects of the field. No longer can an engineer perform the same duties their entire life, technologies evolve at an extremely high pace, and the competitive aspect of modern economy forces adaptation. Furthermore, progress in communication and long distance collaboration has created a significant problem for US graduates: underpriced competition from abroad. The simplicity of a US based workforce, better communication skills, and higher quality work are the main factors current and future engineers within the US must rely on to prevent outsourcing of their jobs. Although many turn to politics for creating barriers and preventing outsourcing, such tactics arise from inherent greed and stifle the overall progress of humanity as a whole. It remains the responsibility of the individual engineer to compete in such a market by providing a superior skill set, communication, and overall benefits of employments. Thus the long term career success of most modern engineers highly depends on lifelong learning and adaptability to innovations in the field and overall superior company benefit over their potential outsourcing counterpart abroad.

Jay

To remain successful as an engineer, you have to keep gaining new knowledge about how to solve issues in the best way possible. If you don't keep learning how to make things better, or finding a better solution, then you'll fall behind. There is constant technological growth, and an engineer should be aware of this. An engineer should embrace this technological growth, and try to learn as much as they possibly can about it. Lifelong learning is incredibly important for an engineer because we are uniquely qualified to learn this material, and potentially improve society as a whole. Engineers as a whole have to keep learning about new technologies so that they can benefit themselves, and the people around them. By continuing to learn throughout your whole life, you ensure that you will be able to solve the problems you're faced with to the best of your abilities. Engineering is about having the best solution to the problem you're given and for an engineer to do that, they have to keep learning throughout their whole life. Without a commitment to learning about new technologies, and truly understanding them, an engineer isn't living up to their full potential. Engineers have a unique toolset; not everyone can be an engineer. Someone has made it so far when they can finally call themselves an engineer. So for them to waste their potential and not continue to learn is a true shame. Engineers should continue throughout their lives because they shouldn't waste their potential.

5.5 Contemporary Issues and Global Impact

Vlad

With the exponential technological progress achieved by humanity every year, the modern state of evolutionary progress may be exhibiting a stage never before encountered within historical cycles of humanity. Engineers, mathematicians, physicists, programmers, and scientists drive the progress of humanity's base knowledge with regards to the laws of nature which govern our universe and can be exploited to advance humanity into further stages of civilization. Technological progress has been the single most important aspect in improving the standard of living for modern humans, providing access to abundant food and clean water. Within developed countries, modern technology provides the capability for most of the population to meet the bottom tiers of Maslov's self-actualization triangle: physiological and safety concerns. Such abundance was never the case in historic times; however life was also much less complex for the average person as well as the engineers at the brink of science. As the circle of human collective knowledge has increased, fueled by limitless information exchange through the application of the world-wide-web, the radius of that circle, representing the limits of our knowledge and number of new questions humanity cannot answer has also grown exponentially. On the other hand, although human innovation cannot keep up with the lust for answers, the exponential rate of progress can have profound negative effects for a large number of people. Many would argue that such a topic as technology displacing human labor has cyclically brought the general population into a panic state; however, after each such historical phase, the general labor population has always maintained its volume without widespread devastating effects. Applying the pattern to the modern technological revolution, one could argue that the net displacement of jobs from automation techniques will lead to the creation of new positions to be filled by the workforce. Based on the careful analysis of the types of jobs that modern technology could successfully automate, however, a major portion of the population may soon find themselves unemployable without advancement of personal knowledge. Successful application of autonomous vehicles, alone, threatens the over 3 million-strong workforce of the transportation industry within the United States. The application of modern processing power, as well as potential future benefits of applying nanotechnology, photonics or quantum gate computers, has created a unique situation never before experienced within the history of humanity. One must consider the widespread potential consequences of cheaper and more effective robotics and software automation replacing the majority of low-skilled labor. Although highly technical fields may not notice significant detriment from such technology, advancement of artificial intelligence promises possible automation of much labor intensive intellectual work as well. A particular example encountered within the research phases within this project was the use of software for music composition. A small application, cgMusic, was effectively able to compose entire symphonies and melodies never before heard in the history of mankind with exemplary results. The long term effects of ever-advancing possibilities within the sphere of AI are not yet fully understood and despite their massive potential benefits, they may bring significant harm onto humanity in the form of complete job displacement or malicious intent with tremendous consequences.

Jay

The push toward reaching a sustainable society is a huge turning point for our society. It can help do so many things that will be a huge benefit. The main benefit is that it will take a huge amount of work to achieve. This will cause an economic resurgence that hasn’t been seen since the interstate highway was created. Society as a whole is trying to get on the path of sustainability, and stability. This includes implementing new sources of energy like solar, hydroelectric, wind, etc. This will continue to be affected by the advancement of technologies. There will always be a push for a more efficient solar panel, or for a hydroelectric system that has less of an environmental impact. The main point is that for our society to actually change, is that it’s going to take an incredible amount of work. New technologies will make it so our power grid can be able to handle renewable power systems better; the only problem is implementing them. A lot of sustainable systems don't have regular outputs, so there has to be a redistribution of power. The power grid has to be change so that it can handle this shift of power when it needs it. This will limit the amount of a switch to sustainable energy until there is a power grid that is able to redistribute power like this. At this point, it's not so much of a technological issue as much as it's an economical one. The technology exists; it's being implemented on a small scale. There just isn't enough of a driving economic force to make such a drastic change. This leads it into being a political issue. Again, the technology is there, it could be funded and a lot of people would get jobs just like in the project to create the interstate highway. But there are companies out there lobbying for change not to happen. Natural gas, coal, and oil companies don't want this shift toward sustainable energy because their jobs, their lively hoods rely one change not happening. Without a huge push from the society as a whole, it will take a long time for our society to change. For change to truly happen, the voice of more than corporations has to be heard. Historically, the greatest advancement in technology is the use of electricity. Our ability to produce electricity has continued to grow, and reached a point where it is available to everyone. Electricity has almost gotten to the point where it is a basic human right. Technological advancements are continuing to happen in the realm of energy production, and will shape how our society is able to function in the future. But to continue to grow, and provide an even better life for the future generations, this is an issue that needs to be solved before it gets too large. If it comes to a point where the power grid can't handle the load it's under, and something happens, there will be a huge amount of chaos that follows.

6. Conclusions and Executive Summaries

The following conclusions were formulated as a result of the combined experiences throughout the senior design term:

- The third iteration of design was successfully completed and made to fully comply with initial requirements.
- Production plan was developed to sell the product as a kit and the design was tailored

to allow autonomous production of the frame pieces.

- The final product design aligned with cost target of under $100 to consumer and proved exemplary functionality without noticeable latency for the end user.
- Beyond the scope of the original project to create a digital music instrument, custom software was successfully developed to augment the user's playing abilities and provide professional level performance without the required knowledge of music theory.
- When creating a new product, achieving excellence requires many iterations of engineering design.
- Throughout the engineering design process, the final product direction may change on multiple occasions with the availability of new information through conducted research and testing results.
- Although product design and functionality often fills the initial vision of an engineer, cost savings tactics as well as production capable strategies must be employed for successful integration of the product into the consumer market.

Vlad

The senior design experience proved quite positive overall. The process was successful in providing highly versatile experience in the sphere of engineering design, product manufacturing, and business planning. The knowledge gained attributed to the realization of the level of work required for an end consumer product and provided a sense of appreciation for the engineering labor behind the majority of commercial products used in daily life. In terms of the laser harp project, the initial goal set out to create a prototype, and fell into a pattern of iterations of product redesign to fit with the next best image for the product. The cost constraints placed on the final product from the start of the design provided a close approximation of engineering challenges encountered in the workplace. Realizations were made with regards to the tradeoffs of functionality and reliability vs product cost. Within the workforce, engineering teams generally receive constraints from upper management and the business divisions and have to operate both within limited research budgets as well as low cost of final product. The initial goals, design process, and finished product created throughout the senior design term provided significant experience emulating many such aspects of real engineering applications within the workforce. Although the project was closely tailored to the work of product design engineering, testing and validation tactics were also employed to perform comparisons of product functionality to the initial specifications to benchmark success. The real success of the product, however, lies within the consumer demand for the final product and net long term profit generated. These profits must account for product returns, software updates, and customer support. Therefore good design practices are a vital component of final success to provide a difficult to break product with versatile functionality and clear instructions for use. The combined experience from the performed research and development, manufacturing insight, and overall engineering design process paves a pathway into successful integration into the engineering workforce. Although accompany was created for the project, a substantial amount of field experience, business connections, and management tactics is still required before large success can be achieved. Hence, JVTech LC will remain as a complementary project on the side of an industry engineering position.

Jay

Senior design has been a great experience. The best part of it was the teamwork. The process showed what can be done if you can work well with someone. The level of honesty, and trust made the process what it was. It showed the importance of these values when you’re working on a project. This process wouldn’t have been the same without that. Also the fact that the project was something truly interesting made it that much better. There was a good amount of work to go around between two members. The tasks to design and manufacture a working laser harp were eye opening. They truly showed what you have to do if you want to design something from scratch, and then get it to the point where you can sell it, and be proud that it is your product. Overall, the best part of senior design was being able to work with someone who is truly brilliant. Senior design wouldn’t have been the same otherwise. There is a good future for the project that was worked on. There are still so many aspects that have to be learned before it is successful. Nonetheless it was a great experience, and so many useful things were learned throughout the process.

7. REFERENCES

"Altium Designer." *Overview*. N.p., n.d. Web. 14 Dec. 2015. <http://www.altium.com/altium-designer/overview>.

"CFR - Code of Federal Regulations Title 21." *CFR - Code of Federal Regulations Title 21*. N.p., n.d. Web. 14 Dec. 2015. <http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/cfrsearch.cfm?fr=1040.1>.

"CorelDrawX5 Reviewer's Guide." *PsycEXTRA Dataset* (1981): n. pag. Web.

"Epilog Legend Laser Series." *Epilog Legend Series: Epilog Mini 18, 24 and Helix*. N.p., n.d. Web. 14 Dec. 2015. <https://www.epiloglaser.com.au/products/legend-laser-series.htm>.

"Freesound.org - Freesound.org." *Freesound.org - Freesound.org*. N.p., n.d. Web. 14 Dec. 2015. <http://www.freesound.org/>.

"Music Therapy for Depression: It Seems to Work, but How? | The British Journal of Psychiatry." *Music Therapy for Depression: It Seems to Work, but How? | The British Journal of Psychiatry*. N.p., n.d. Web. <http://bjp.rcpsych.org/content/199/2/92>.

N.p., n.d. Web. <http://www.byteparadigm.com/kb/article/AA-00255/22/Introduction-to-SPI-and-IC-protocols.html>.

"OSH Park." ~ *Design Submission Guidelines*. N.p., n.d. Web. 14 Dec. 2015. <https://oshpark.com/guidelines>.

"Printed Circuit Board Surface Finishes – Advantages and Disadvantages." *Printed Circuit Board Surface Finishes*. N.p., n.d. Web. 14 Dec. 2015.

<http://www.epectec.com/articles/pcb-surface-finish-advantages-and-disadvantages.html>.

"PS5022 Datasheet -." *PS5022 Datasheet* -. N.p., n.d. Web. 14 Dec. 2015. <http://www.digchip.com/datasheets/parts/datasheet/459/PS5022-pdf.php>.

Pappas, By Stephanie. "Making Music Proves to Be Powerful Antidepressant." *LiveScience*. TechMedia Network, 05 Aug. 2011. Web. <http://www.livescience.com/15422-making-music-powerful-antidepressant.html>.

Publication Release Date: October 09,, 2013, and Revision H. *W25Q128FV 3V 128M-BIT SERIAL FLASH MEMORY WITH DUAL/QUAD SPI & QPI* (n.d.): n. pag. Web.

Semiconductor, Inc. Freescale. *SGTL5000, Low Power Stereo Codec with Headphone Amp - Data Sheet* (n.d.): n. pag. Web.

"Tech Specs & Info." *Tech Specs & Info*. N.p., n.d. Web. 14 Dec. 2015. <http://www.midi.org/techspecs/>.

"TechShop Is America's 1st Nationwide Open-Access Public Workshop -- TechShop Detroit, Michigan." *TechShop Is America's 1st Nationwide Open-Access Public Workshop -- TechShop Detroit, Michigan*. N.p., n.d. Web. 14 Dec. 2015. <http://www.techshop.ws/ts_detroit.html>.

"Teensy USB Development Board." *PJRC Store*. N.p., n.d. Web. 14 Dec. 2015. <https://www.pjrc.com/store/teensy32.html>.

"Webstore International Electrotechnical Commission." *IEC 60825-1:2007*. N.p., n.d. Web. 14 Dec. 2015. <https://webstore.iec.ch/publication/17996>.

8. APPENDICES

The following sections provide reference information for the project as well as preliminary work efforts.

a. *APPENDIX 8.1: ORIGINAL PROJECT PROPOSAL (First Semester)*

ECE 4985/6 ELECTRICAL/COMPUTER ENGINEERING DESIGN

<Fall> <2015>

NAME: Vlad Slyusar, Jay Lindland **DATE: 10/12/2015**

TITLE: Laser Harp Synth

ADVISOR: Professor Rawashdeh

DESCRIBE BRIEFLY AIMS AND GOALS OF THE PROJECT:

The goal of the project is to design a digital music synthesizer which relies on an array of lasers replacing the strings of a harp-like instrument. Currently, no such product exists on the market with the exception of a few multi-thousand dollar custom made devices. Thus the focus of this project involves applying a heavy cost-savings approach to design, build, and mass produce such an instrument for under $100 to the end consumer. Such a cost difference is quite feasible to accomplish by deviating from the design principles of expensive laser-based instruments and implementing a brand new approach from the ground up.

DESCRIBE RELEVANT PRIOR WORK:

A large portion of prior work was undergone in Embedded systems ECE473. A prototype device was built relying on the Tiva C board used in the class. The focus of the project deviated from a cost effective model by relying on multiple microcontrollers connected over bluetooth and encompassing external subsystems to control professional DJ lighting equipment. An audio amplifier chip was used to drive a speaker from a PWM pin on the microcontroller generating tones for each of the laser strings. For commercial implementation, the light controller attracts only a very narrow customer spectrum and single frequency tone output has very little practical uses outside of demonstration. Thus, a complete redesign of the project will involve a new smaller microprocessor, external subsystem with a digital signal processor chip, and flash memory to provide real music as expected out of a digital synthesizer.

PRELIMINARY IDEAS AND METHODS:

The end goal of this project is to create a low-cost final prototype with custom PCBs and mass-production ready physical design. The majority of the work throughout the year will involve developing, creating, and testing multiple iterations of the design allowing us to experiment with various features and determine the direction for the final design balancing costs and features to incite the largest possible customer base. Initial ideas involve relying on a teensy microprocessor with audio DSP shield as the heart of the project. Our goal is to test a variety of approaches and equipment for various uses. Currently we plan to test a distance sensor or horizontal lasers for controlling pitch of an individual string. The majority of the work will be centered around the microprocessor and DSP chip for creating beautiful sounds. We intend to use a custom flash memory chip to store pre-recorded sounds, perhaps as .wav files, and play back files corresponding to each laser string. This approach allows multiple instruments programmed into the device at a single time, depending on the memory size and flash read speed and latency. The general method for the project will involve running through multiple iterations generating new ideas for improvements of the project.

SPECIFICATIONS AND ENGINEERING STANDARDS:

- SPI for flash memory access
- UART for programming micro and serial communication
- Bluetooth - possible addon for external communication
- DSP - Digital signal processor to perform FFT and audio playback
- RAM - Experimental chip for adding audio delay line
- MCU - Microprocessor Unit, Teensy 3.2 for initial testing

PROJECT COST ANALYSIS:

The project will be fully funded by our group, the cost analysis most relevant to this project is the bill of materials and manufacturing costs of the design which will determine the final price and market feasibility of the product. Current BOM with approximate costs not involving volume-based discounts:

Part	**Cost**	**Quantity**	**Total**
Laser Dot Diode	$0.20	8	$1.60
PS5022 Phototransistor	$0.10	8	$0.80
10ohm resistor	$0.04	8	$0.32
100k resistor	$0.03	8	$0.20
1" square pcb board (max)	$1.67	8	$13.36
Teensy	$11.65	1	$11.65
Body Construction (approx.)	$20	1	$20.00
3.3v 2x AA battery holder	$2.92	1	$2.92
3.5mm Female Socket	$0.73	1	$0.73
Total			$51.58

The projected costs are subject to change as we explore various technologies for user connectivity, power, and actual manufacture cost quotes. The cost breakdown for development of the product is currently set at 3 times the BOM cost for first build(3x prototypes) followed by an additional 3x the cost for second prototype stage. Further $200 will be allocated for trying out new components and to allow spillover of material costs currently defined. Thus total project budget for R&D will be set at 3*$50 + 3x$50 + $200 = $500. Project will be funded out of pocket by group members.

b. *APPENDIX 8.2: DETAILED PROGRESS REPORT (FIRST SEMESTER)*

Laser Harp Synthesizer

Project Team:
Vlad Slyusar
Jay Lindland
Project Advisors:
John Miller, Ph.D.
Samir Rawashdeh, Ph.D.

Abstract:

This report presents the process of developing a low-cost laser harp synthesizer. The project relies on an array of 8 laser dot diodes each activating a detection circuit based on a phototransistor (Figure 1). Interruption of any of the beams signals a Teensy microcontroller to output a programmed sound effect. The sound is processed on an audio board with a DSP chip and output to a standard 3.5mm connector (Figure 2). The novel aspect of this project centers on the fact that no low-cost laser-based synthesizer currently exists on the market and the conducted research paves the way for mass production.

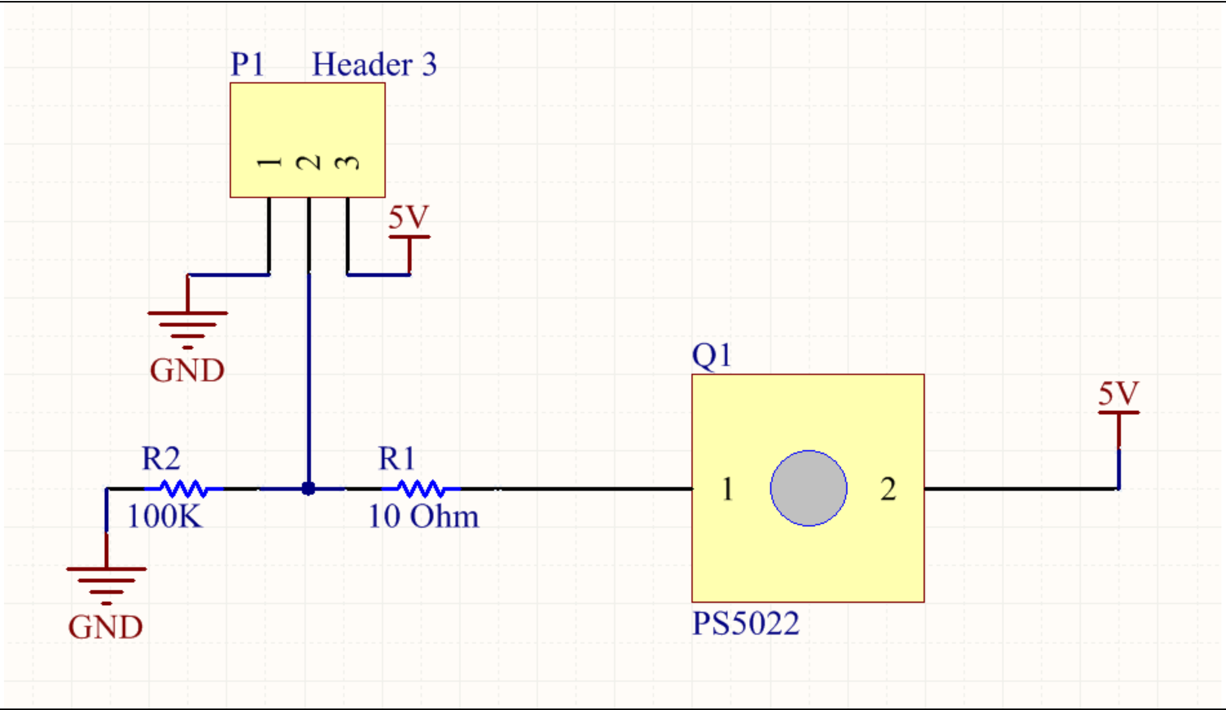


Figure 1. Laser Detection Circuit

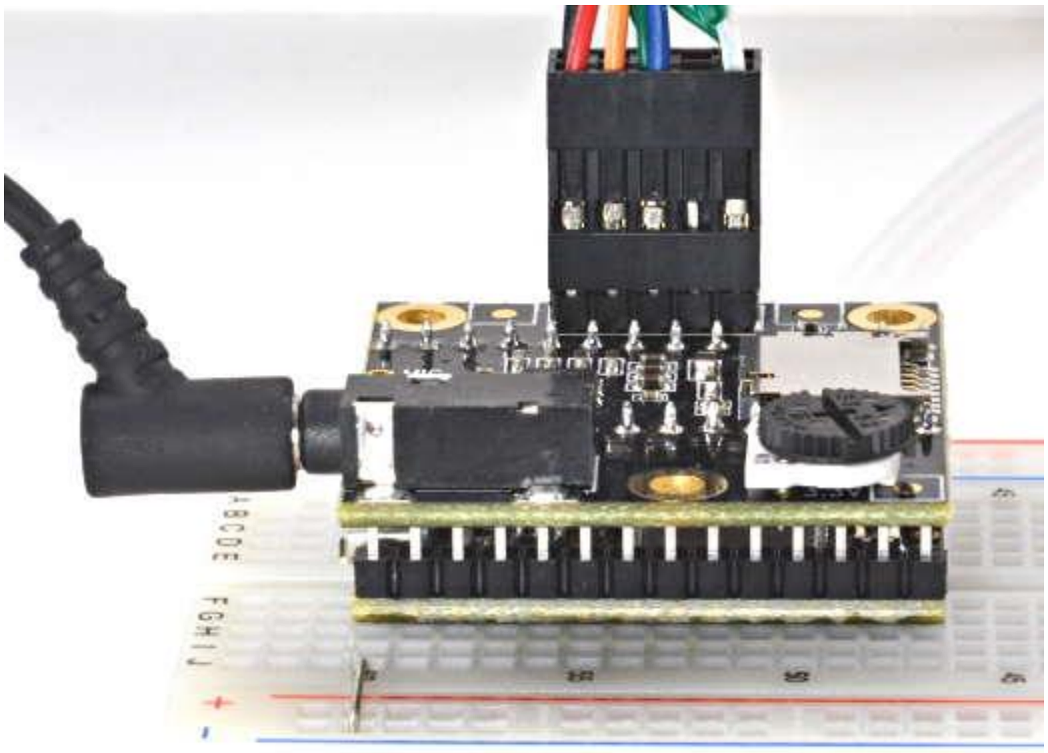
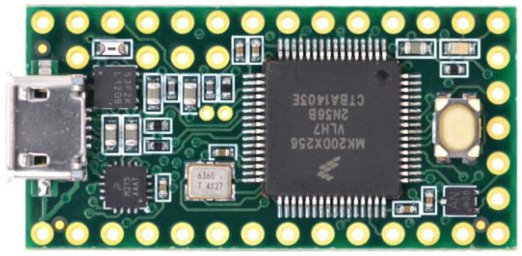


Figure 2. Teensy Board Connected to Audio Adapter with Sound Output

The initial idea for the project focused on creating a low cost laser based synthesizer designed for mass production for the consumer market. A goal of under $100 was set as a target sale price to draw in consumers and still generate a profit. Through considerable research, practice, and ideation, design techniques were selected and optimized on the path to creating as stylish, full featured, and low cost laser harp. The project frame started out as a wooden construction held together by screws. The frame pieces were cut using a band saw and a drill press was used to create 6mm holes to house the laser dot diodes. Small copper perfboards were built by hand to accommodate the laser detection circuitry. The design was updated now using a ¼ inch acrylic frame etched out of a 24”x18” sheet of plastic cut out on a 60W Epilog CO2 laser cutter, held together by industrial hot glue. The frame was designed as a 2-dimensional hairline image using vector graphics software Corel Draw X5. Moving away from a wooden frame prototype to plastic adds aesthetic design value and allows for more accurate and automated part production. The heart of the project relies on a microcontroller to read sensor inputs and generate sound for each interrupted laser beam. Initially a Tiva C microcontroller was used, but the much smaller and popular teensy micro was selected for the new prototype. While both microcontrollers are built around a Cortex-M4 processor, the Teensy’s incredibly small size as well as a versatile selection of hardware shields proves more effective to meeting the goals of this project. Three versions of the teensy were ordered and tested: Teensy 3.2, 3.1, and the LC low cost model. Unfortunately many issues were encountered with the low cost version of the board including insufficient memory and processing power. An image of the board is provided below illustrating the relatively small size with the micro-USB interface on the left side as reference.



Teensy 3.2 Microcontroller

Both the 3.1 and 3.2 however performed well and were both selected as candidates for final design depending on availability and cost at the time of large volume production.

As far as the audio board, the initial idea of relying on an SD card for audio storage was rejected to keep costs down and the W25Q128FV Serial NOR Flash memory chip was selected as a viable low cost alternative. The bill of materials was updated to reflect material costs for the current prototype:

Part	**Cost**	**Quantity**	**Total**
Laser Dot Diode	$0.20	8	$1.60
PS5022 Phototransistor	$0.10	8	$0.80
10ohm resistor	$0.04	8	$0.32
100k resistor	$0.03	8	$0.20
Custom PCB (No assembly)	$0.45	8	$3.6
Teensy	$17.00	1	$17.00
Body (Clear Acrylic)	$17.00	1	$17.00
3.3v 2x AA battery holder	$2.92	1	$2.92
Audio Board	$14.25	1	$14.25
Total			$57.69

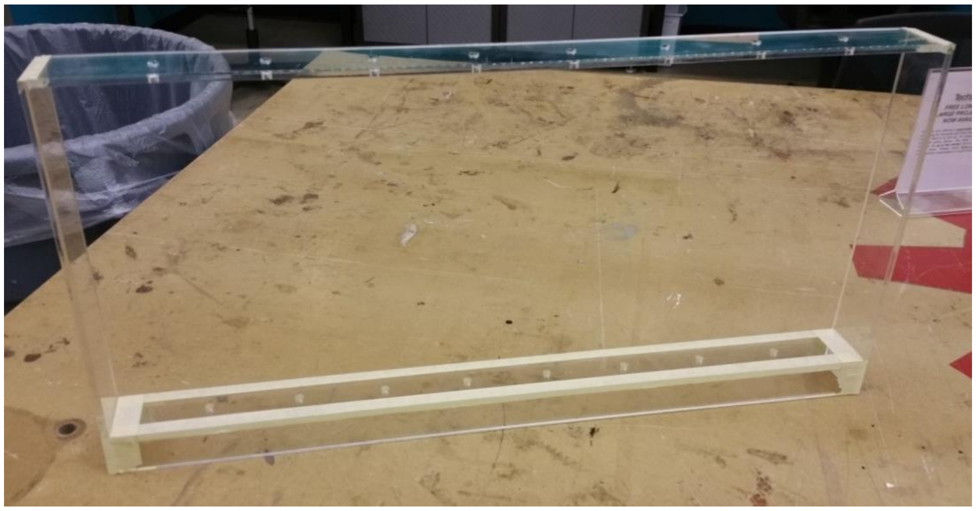
This reflects a slight increase from the initial BOM approximation of $51.58. Although PCB production and frame material costs were reduced, the added cost of the audio adapter board generated a net increase in cost. The image below illustrates the physical construction of the audio board showing both a 3.5mm RCA audio interface as well as microSD card slot.

Teensy Audio Adapter Board

Finally, the timeline was heavily altered from the initial proposal. Prototype deliverable was pushed to the end of Semester 1 and all tasks successfully completed. First prototype stage was completed, experimentation was conducted into alternative technologies, and a custom frame was designed and built. PCB boards for the laser detection circuits were designed, manufactured, tested, and finally used in the creation of a finalized prototype. Since most of the work was completed ahead of schedule, Semester 2 goals will focus on overall improvements in further prototype iterations as well as the development of software for the end user. Custom GUI interface for programming the laser functions and uploading custom sounds must be created before the laser harp can be commercialized. Furthermore, a replacement for the teensy and audio shield will be considered using a single custom PCB board. Final manufacture decision will rely on production costs including assembly of custom boards as compared to the teensy ultimately the lower cost solution will be employed.

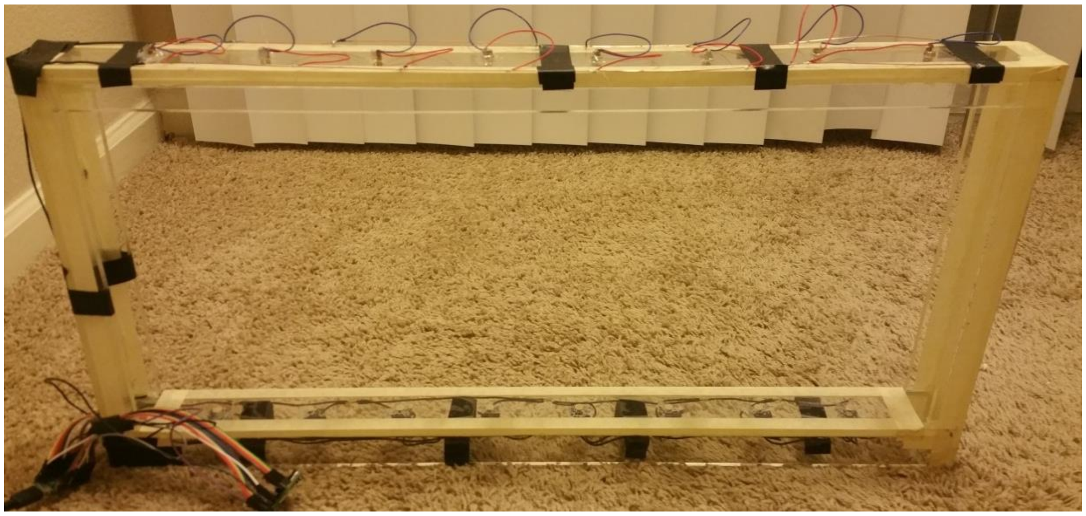
The main problem inherent to this project relies on not only developing a working prototype but also keeping final production costs below $100 to the consumer while generating a suitable profit margin. While engineering design issues are generally easy to fix with methods that add cost to the product, a tougher approach was chosen for this project focusing on final product pricing rather than complying with a limited development budget. At the current breakdown, the material costs come out to $60. The fabrication cost assessment varies from $20 - $40, leaving an approximate profit margin of only up to $20 per unit. Such a low profit margin would pose a serious issue limiting opportunities for growth as well as curbing investment interest. Thus significant effort was applied to solving this issue from an engineering standpoint. One of the design alternatives which could bring down the costs significantly centers on discarding the teensy and audio shield and developing a fully custom microcontroller as replacement. In order to determine the feasibility of this approach, working boards must be designed and assembly quotes obtained from several board houses. Hughes electronics will be considered as the first choice. The PCBs in the current prototype were obtained at very low cost due to small size, requirement of only 2 layers, and most importantly the lack of assembly. While the detection circuits used through-hole components easily soldered by hand, the custom microcontroller board will have a number of small SMD components as well as QFN package chips requiring pick-and-place machine and reflow soldering for optimal results. The cost for such work will inevitably exceed simple PCB printing; however, as long as the net cost falls below the $31.25 cost of teensy and audio board this approach will provide cost savings. Outside of engineering practices, the major factor in reducing costs relies on volume production. A Kickstarter project will be created during Semester 2 in order to raise money up front. Minimum investment cost will be calculated and product price will be based accordingly. Initial batch will be offered at minimal to zero profit to gain popularity.

Aside from cost reduction, the design of an easy to construct product brought about a significant number of challenges requiring engineering solutions. Perhaps the most difficult problem came as a result of non-ideal laser diodes with significant variation in beam direction. For successful operation, the laser beam must point directly at the phototransistor located 12” away. While with the first prototype, inaccuracy of wood and drilling process was believed to cause the misalignment, the laser etched acrylic frame experienced similar problems to a smaller degree. With the wooden construction, the detector boards were placed after the lasers and affixed with tape in positions effective to picking up the laser beam. As a result, the detectors were not evenly spaced out and several beams appeared crooked attributing to a disproportional look. With the finalized prototype using acrylic, 6mm holes were etched for both the lasers as well as detectors thus forcing perfectly straight beams of light. However, despite nearly perfectly straight containment holes, several lasers curved and the only solution required affixing the dot diodes in proper position with hot glue. More effective manufacturing techniques are still undergoing research. Another problem with the frame resulted from the large dimensions of the frame (24”x12”) which allowed a slight degree flex of the top plane housing the lasers. The design is illustrated in the image below illustrating the initial flaw.



Laser Harp Frame – Acrylic Design (Flexing Problem)

Even a small deformation proved enough to alter the direction of several lasers and trip several sensors. Such a defect would cause several notes to play at once without the user’s intent and thus a supporting structure was added to the initial design etched out of the same acrylic sheet. The result produced a three-dimensional frame providing significant rigidity and a design impervious to standard bending pressure. The frame is illustrated in the following image of the completed project.



Laser Harp Frame with Structural Support

Another engineering problem currently requiring exploration is the power drain from the entire project. Since the product is designed for portability, a battery will be used in the final design. The nominal current draw must be measured for the processor running the code as well as the instantaneous current draw by the audio amplifier when playing sound. The eight red lasers rated at 5mW of power will draw 12.2mA of current at the 3.3V provided by the teensy but actual measurements must be made to calculate exact total drain. Furthermore, in order to determine an accurate runtime an algorithm must be used to approximate general usage based on nominal and active drain when playing sound. In case the 2x AA batteries produce severely short battery life, a more costly solution of integrating a lithium ion battery will be favored. In that case, the battery must be integrated into a single output port shared with the microcontroller and circuitry must be designed to allow charging the battery as well as interfacing with the Teensy.

Another engineering decision that had to be made required a tradeoff between user safety and product appearance. The greatest novel factor of the project is the replacement of conventional strings with laser beams. While the 5mW red lasers currently used are safe for production, the power limitation requires haze, fog, or smoke for beam visibility. While the intended use for the project calls for some sort of air particle generator, research was conducted into an alternative solution allowing beam visibility in any dark environment. A class 3B or class 4 laser would be used with an array mirrors to split the single powerful beam into 8. Due to cost and safety considerations, no deviation from the current design will be made at the current time.

The following section breaks down engineering standards relevant to the project:

- SPI – Serial Periferal Interface. Used for communication between teensy microcontroller and audio adapter board. SPI is a synchronous serial communication bus with dedicated send and receive lines, a clock line, as well as a dedicated chip select line.

- Teensy 3.1/3.2 – 32 bit ARM Cortex M-4 Processor based microcontroller clocked at 72 MHz. The main difference is that the 3.2 version includes a more powerful 3.3 volt regulator. Peripherals, memory, and processor are identical for the two boards.

- Teensy Audio Board – SGTL5000 audio codec chip from NXP. Onboard DSP performs real-time FFT calculations necessary for outputting audio samples. This is an improvement to single tone output previously generated from a PWM pin driving an amplifier.

- Serial NOR Flash Memory – W25Q128FV chip, 128Mbit. Used for storing audio samples for direct sound synthesis.

- MIDI - Musical Instrument Digital Interface. USB output to computer for use with DJ studio software such as Abbleton Live. MIDI output currently in development and planned for release with commercial software in Semester 2 of Senior Design.

- RCA audio – 3.5mm female adaptor to fit standard headphones/speakers.

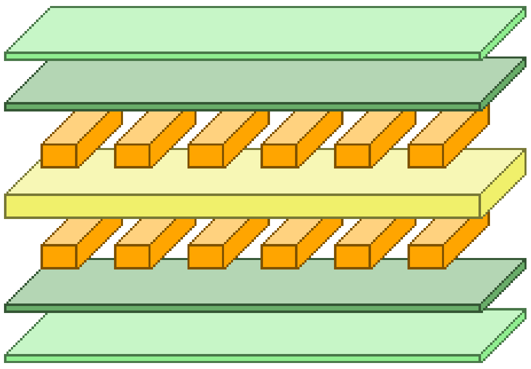
- Laser Dot Diodes

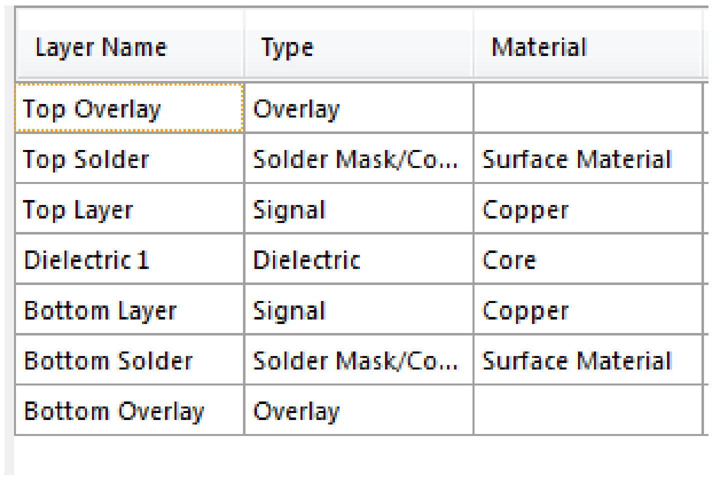
  - Class 3R (IEC 60825-1 Standard) – 5mW power rating
  - Comply with CFR - Code of Federal Regulations Title 21 1040.10 and 11. (Qualify as pointer devices).

- PCB Design

  - Altium PCB Design Suite

- 2 Layer Board





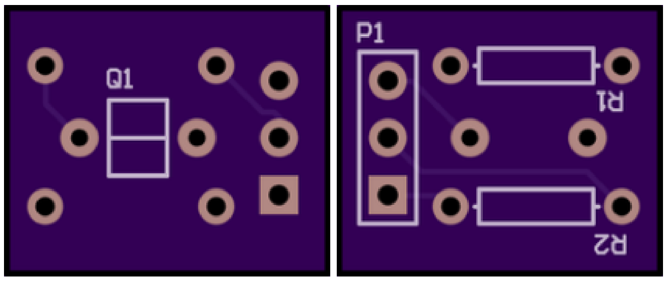
Layer Name	Type	Material
Top Overlay	Overlay	
Top Solder	Solder Mask/Co...	Surface Material
Top Layer	Signal	Copper
Dielectric 1	Dielectric	Core
Bottom Layer	Signal	Copper
Bottom Solder	Solder Mask/Co...	Surface Material
Bottom Overlay	Overlay	

Project Layer Stackup

- Trace Width: 10mil track diameter, 6mil minimum allowed by board house.
- Trace Spacing: 10mil minimum spacing between tracks.
- Drill Size: 13mil minimum hole
- Annual Ring Size: 7mil minimum specification for area on the pad surrounding a via.

● PCB Manufacture

- OSH Park Fabrication Facility



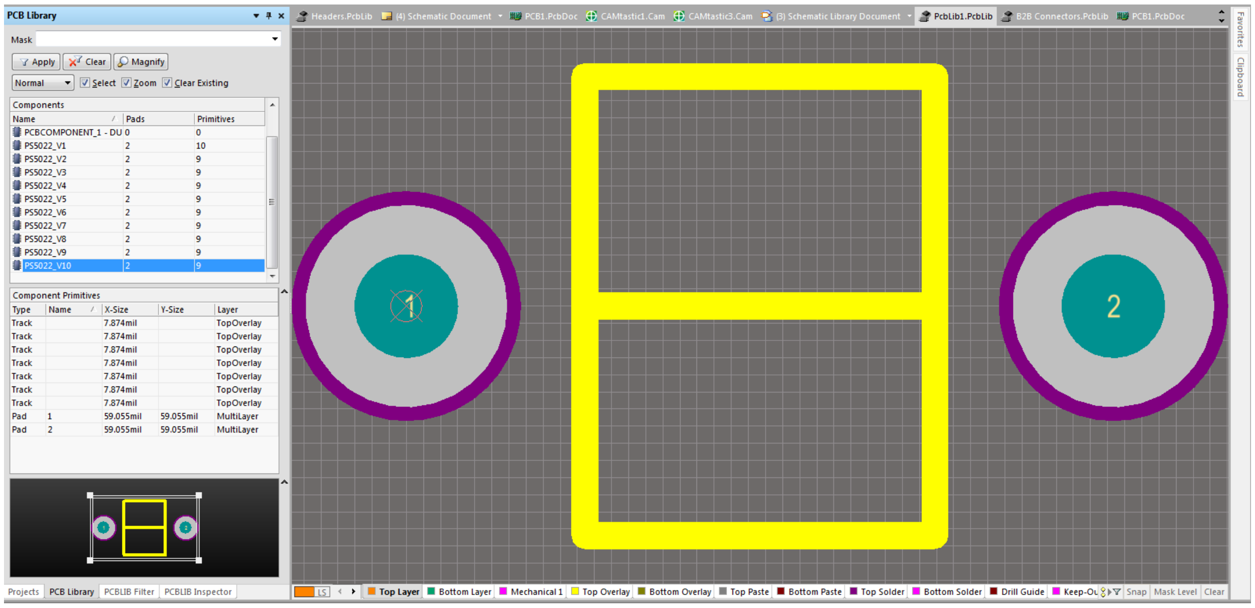
Board Design Files Interpreted by OSH Park System

- FR4 grade, glass-reinforced epoxy laminate (flame retardant).
  - ▪ In compliance with UL94V-0 standard. (Plastic flammability with Underwriters Laboratories). Material burning stops within 10 seconds on a vertical specimen.
- 170Tg (glass transition temperature) / 290Td (deposition temperature)
- Lead Free Process – ROHS Compliant

- ENIG finish surface plating (Electroless nickel with immersion gold to protect from oxidation)
- 1.6mm final thickness, 1 ounce copper on each side

In order to accelerate the timeline and complete a solid prototype this semester, significant effort was exerted on behalf of both partners. The task of frame assembly and product research was taken up by Jay Lindland. Jay also applied prior knowledge as well as research based expertise in music theory to solve many initial issues with producing sound output and playing real samples using the DSP. Research into legal issues as well as general aesthetics was also handles by Jay to improve looks and reliability as well as limit potential lawsuits. Vlad Slyusar took on the task of PCB design per past experience at work, although the entire team was actively involved in talks with the board house and fabrication process. The acrylic frame was produced by Vlad with the help of utilities at the Detroit-Allen Park Techshop. Embedded programming was also handled by Vlad. Although specific tasks were taken up by individual team members, the majority of work throughout the semester centered on fixing problems with the design as well as embedded code.

Stepping back from a cost centered approach, significant progress was made within the sphere of engineering to make the project operational and develop improvements after thorough research and experimentation. As previously mentioned, multiple Teensy microcontrollers were obtained for evaluation. Two copies of the audio shield from PJRC were also ordered for experimentation by each partner. The major difficult task relied on setting up a communication channel between the micro and audio adapter using the SPI protocol. Once communication was successfully established, the next major challenge was selecting the type of music output and physically achieving the result. Many techniques were tested including playing WAV files from an SD card formatted in FAT32, FAT16, and extFAT. Latency issues with reading bandwidth caused problems playing more than 3 strings at a time and the cost of upgrading to a faster SD card would increase project cost or require imposing limitations on end user SD card purchasing. As a result, the idea was completely scrapped for a $1.63 flash memory chip replacement to be optionally added in the final design. As far as the physical frame, ¼” acrylic was selected due to availability as well as pleasing surface. Clear plastic was used due to lower costs and to provide an informational prototype. Corel Draw X5 was obtained and used for designing the outline for frame components. An Epilog CO2 laser cutter was used at Tech Shop facilities to perform clean and accurate cuts. As far as actual laser harp operation, a simple design was developed from scratch relying on strong beams of laser light activating detector circuits. This major deviation from Theremin based designs with expensive lasers and sensors is precisely why the project goal focuses on low cost design as compared to multi-thousand dollar instruments. A PS5022 phototransistor was used due to availability and a 10 Ohm and 100K resistor were used for biasing the circuit to react only to the laser light. Experimental method was used in determining the values successfully eliminating sensitivity to daylight and even the brightest room lighting while at the same time providing an error margin for imperfect laser positioning. In other words only the laser triggers the phototransistor, even if only partially aimed at its surface. Bright light from cell phone flash was still ineffective in triggering a response from the detection circuits. As far as the physical circuitry, the route of manufacturing custom PCBs was chosen to gain knowledge and experience as well as simplify prototype manufacturing efforts. Altium Designer Suite software was used for creating custom components (phototransistor), drawing up schematics, creating accurate component footprints, and performing the circuit layout. The image below illustrates the interface for creating a custom footprint designed for the phototransistor. Sizing specifications for physical dimensions were carefully followed to ensure product would properly fit.



Mask

Normal ☑ Select ☑ Zoom ☑ Clear Existing

Components

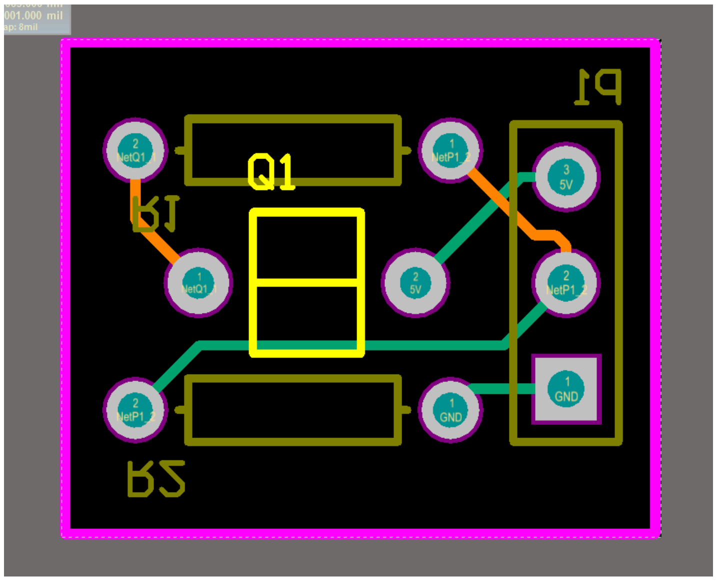
Name	Pads	Primitives
PCBCOMPONENT_1 - DU	0	0
PS5022_V1	2	10
PS5022_V2	2	9
PS5022_V3	2	9
PS5022_V4	2	9
PS5022_V5	2	9
PS5022_V6	2	9
PS5022_V7	2	9
PS5022_V8	2	9
PS5022_V9	2	9
PS5022_V10	2	9

Component Primitives

Type	Name	X-Size	Y-Size	Layer
Track		7.874mil		TopOverlay
Track		7.874mil		TopOverlay
Track		7.874mil		TopOverlay
Track		7.874mil		TopOverlay
Track		7.874mil		TopOverlay
Track		7.874mil		TopOverlay
Track		7.874mil		TopOverlay
Pad	1	59.055mil	59.055mil	MultiLayer
Pad	2	59.055mil	59.055mil	MultiLayer

Altium Custom Footprint Creation

Multiple iterations of design led to the decision of placing the phototransistor on the top side of the board to face the laser and resistors and header on the bottom for connectivity. This design approach creates a maximally flat surface at the top of the PCB allowing simpler fixation to the frame. The image below represents the finalized design with bright outline components and orange tracks representing top layer and the rest bottom. Since through-hole components were used for each part, no excess VIAs were required.



Laser Detection PCB Designed Using Altium

As far as fabrication, the design rules were first checked to ensure compliance with manufacturing minimums of board house. Next, drill files were generated to comply with OSH Park standards and several attempts were made until design was finally accepted. An important part not automatically added by Altium Designer is the board outline the lack of which caused all sorts of issues. Once the small problems were resolved, an order was placed and the boards arrived within 2 weeks. Outside of general debugging and researching parts, the final bulk of work involved the frame construction itself. Acrylic sheets were obtained from the laser cutter etched to size per design. Significant problems were encountered using hot glue as a binding agent as the working time was very short. Furthermore, because of the clear acrylic used, uneven glue marks could be seen from the outside and masking tape was added at the project's corners as a fix. Once the frame was complete, work began on assembling the electronic components. Power and ground bus wires were routed through the top and bottom portions of the frame. Both the lasers and phototransistor circuits each require 3.3V and a Ground connection to operate. Furthermore, individual wires had to be cut to length and stripped for each of the 8 detection circuits. The assembly of custom PCBs proved much more pleasant than the initial perf-board prototypes and several design improvements were noted for future boards. A large improvement in manufacturing time involved an idea to place the boards on the bottom plane rather than affixed to the top of the acrylic. Then a single strip of Velcro can be applied to the entire line and each board can be easily positioned and firmly affixed in place directly in view of its laser. Finally, wires were routed out to the temporary setup with teensy and audio board. Future iterations will employ a stacking header and actually fit the boards inside of the frame.

Although considerable progress was achieved throughout this semester's work and a finalized prototype was built, much more work, research, and optimization remains before the design can be considered production ready. With each iteration of prototype improvement, new ideas have sprung up and a large amount of work awaits next semester in order to make this project a success. Starting with the frame, while the acrylic design looks nice, an updated design will etch cutouts along the sides of each piece to tightly fit together like a puzzle. This approach will add significant strength to the frame even without a binding agent. Furthermore, a custom adhesive solution specifically designed for binding acrylic will be used with a working time of at least 30 minutes allowing time to accurately and properly apply the binding substance. A chemical mixture to physically fuse the acrylic together will not only provide superior binding strength but will eliminate visible globs of hot glue and remove the need for covering up regions of the frame. In terms of software, a custom program with GUI interface must be built including drivers for interfacing with the teensy and its memory. Conversion algorithms and software must be studied to provide users a large variety of sound formats to program into each of the strings. Furthermore, a small button array could be added to the project allowing user to switch between multiple instruments on the fly. The software would be responsible for programming each of the corresponding buttons with either an instrument or collection of user selected sounds. Another add-on to the project would include a Bluetooth or radio interface for wireless communication with other microcontrollers. This option would be particularly feasible if the route of custom microcontroller production is chosen as a Bluetooth chip and radio tracks can be directly embedded into the PCB at the cost of individual parts. The idea itself, although quite challenging, could potentially reduce the project cost significantly. A custom PCB with a cortex M-4 or updated processor will be designed and in the case of appropriate manufacturing costs, adopted as the new controller for the project. On the embedded software side, without requiring any extra hardware, a significant amount of consumers can be enticed by offering MIDI support through the micro-USB port already existing on the Teensy. MIDI is a standard used by many DJ boards, piano keyboards, and synthesizers. Although simple in operation, drivers will have to be written to emulate a popular production controller that already has support with most commercial software. Instead of generating and playing sound from the Teensy and DSP chip, a specific note will be sent to the computer corresponding to the laser string being interrupted. Then the commercial software such as FL Studio or Abbleton Live can be used to tie in audio samples and effects and the laser harp will function no different from a digital piano or mix-board.

Milestones to be met by end of Semester 2:

- Redesigned frame with better structural support, acrylic glue, and cut out in a single run on a laser cutter out of no more than 24”x18” material.

- Explore feasibility of custom microcontroller design, receive quote from at least one board house. (Most likely Hughes electronics).
- Explore distance detection techniques for personal project revision
- Test Bluetooth/RF communication feasibility and cost
- Fully enclosed frame with cut holes for 3.5mm audio cord as well as micro-USB cable. Custom structures to keep microcontroller and all pieces in place.
- Solution to laser direction problem. Each individual laser has different tolerance and they do not all point straight. Must come up with solution to ease construction despite this fact.
- Software with GUI interface for programming sounds into the Teensy
- Kickstarter project. Attempt to raise minimum investment costs to gain volume discounts and make the project a reality. Will require video, building process, as well as possible advertising schemes.

Although significant progress was accomplished throughout the semester, a large amount of work remains ahead in order to make the laser harp production ready. As outlined above, Semester 2 efforts will focus on perfecting the design as well as creating end-user software to simplify reprogramming the project’s functionality. The main goal will focus on lowering costs and making the product easy to use in order to cater to a large consumer base and not just a small community of specialists and engineers.

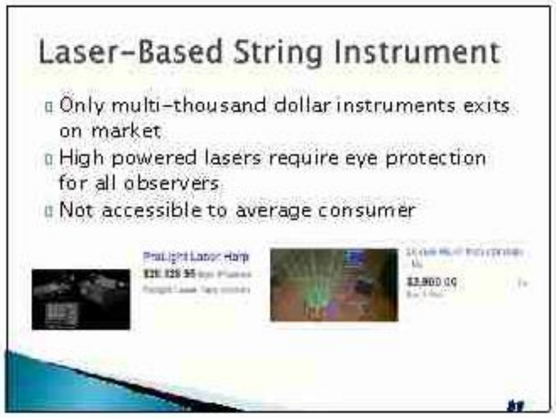
c. *APPENDIX 8.3: POWERPOINT HANDOUT SLIDES (Final Presentation)*

Laser Harp

Senior Design Project

Vlad Slyusar & Jay Lindland

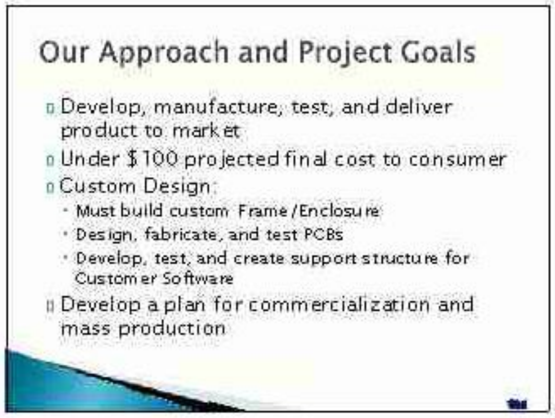
Advised by: Samir Rawashdeh & [illegible]



Laser-Based String Instrument

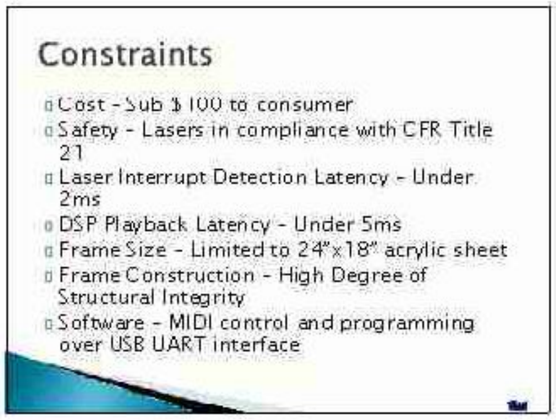
- Only multi-thousand dollar instruments exits on market
- High powered lasers require eye protection for all observers
- Not accessible to average consumer

ProLight Laser Harp
$2,900.00



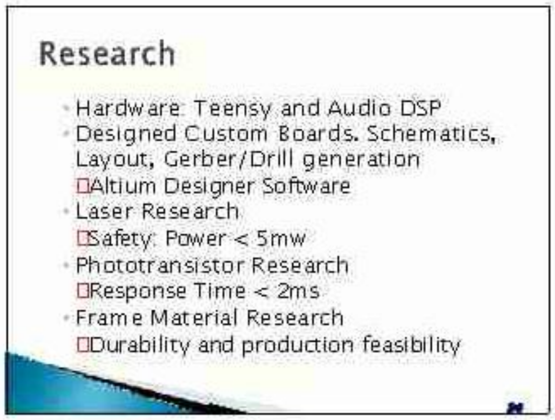
Our Approach and Project Goals

- Develop, manufacture, test, and deliver product to market
- Under $100 projected final cost to consumer
- Custom Design:
  - Must build custom Frame/Enclosure
  - Design, fabricate, and test PCBs
  - Develop, test, and create support structure for Customer Software
- Develop a plan for commercialization and mass production



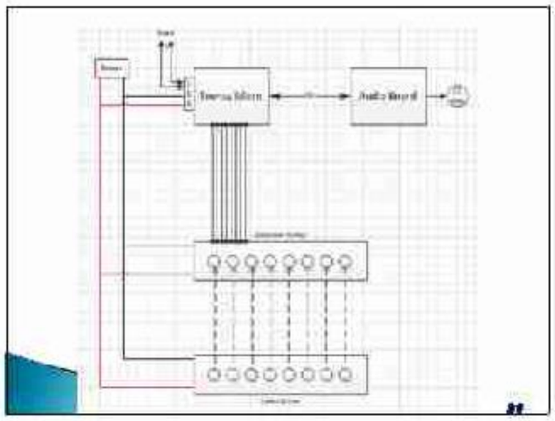
Constraints

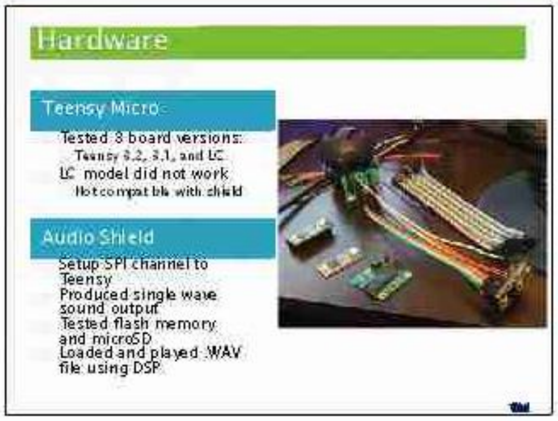
- Cost - Sub $100 to consumer
- Safety - Lasers in compliance with CFR Title 21
- Laser Interrupt Detection Latency - Under 2ms
- DSP Playback Latency - Under 5ms
- Frame Size - Limited to 24"x18" acrylic sheet
- Frame Construction - High Degree of Structural Integrity
- Software - MIDI control and programming over USB UART interface



Research

- Hardware: Teensy and Audio DSP
- Designed Custom Boards. Schematics, Layout, Gerber/Drill generation
  - Altium Designer Software
- Laser Research
  - Safety: Power < 5mw
- Phototransistor Research
  - Response Time < 2ms
- Frame Material Research
  - Durability and production feasibility





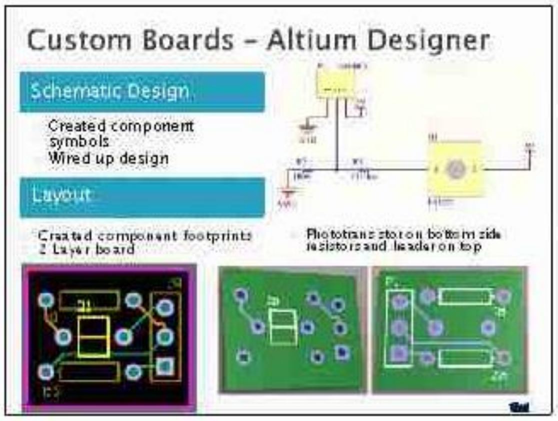
Hardware

Teensy Micro

- Tested 3 board versions:
  - Teensy 3.2, 3.1, and LC
- LC model did not work
  - Not compatible with shield

Audio Shield

- Setup SPI channel to Teensy
- Produced single wave sound output
- Tested flash memory and microSD
- Loaded and played .WAV file using DSP



Custom Boards – Altium Designer

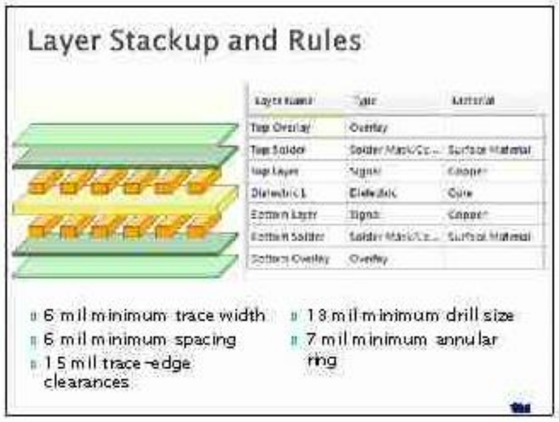
Schematic Design

- Created component symbols
- Wired up design

Layout

- Created component footprints
- 2 Layer board

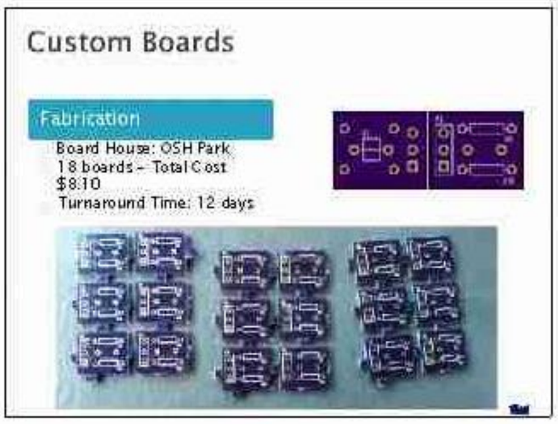
Phototransistor on bottom side resistors and header on top



Layer Stackup and Rules

Layer Name	Type	Material
Top Overlay	Overlay	
Top Solder	Solder Mask/Co...	Surface Material
Top Layer	Signal	Copper
Dielectric 1	Dielectric	Core
Bottom Layer	Signal	Copper
Bottom Solder	Solder Mask/Co...	Surface Material
Bottom Overlay	Overlay	

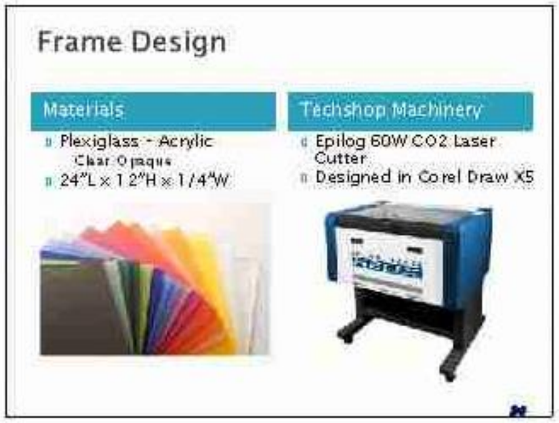
- 6 mil minimum trace width
- 6 mil minimum spacing
- 15 mil trace-edge clearances
- 13 mil minimum drill size
- 7 mil minimum annular ring



Custom Boards

Fabrication

- Board House: OSH Park
- 18 boards – Total Cost $8.10
- Turnaround Time: 12 days



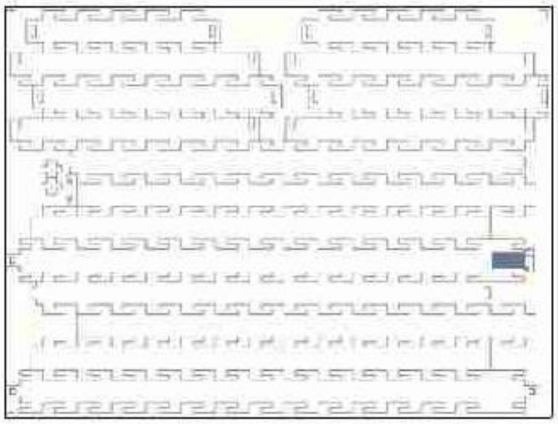
Frame Design

Materials

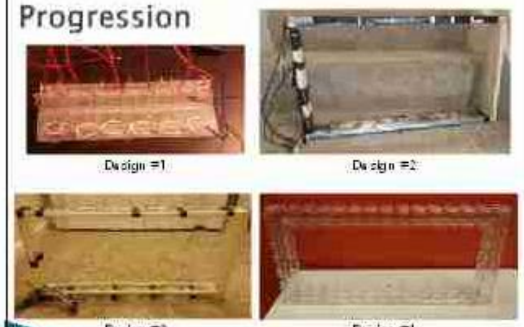
- Plexiglass – Acrylic
  - Clear Opaque
- 24"L x 12"H x 1/4"W

Techshop Machinery

- Epilog 60W CO2 Laser Cutter
- Designed in Corel Draw X5

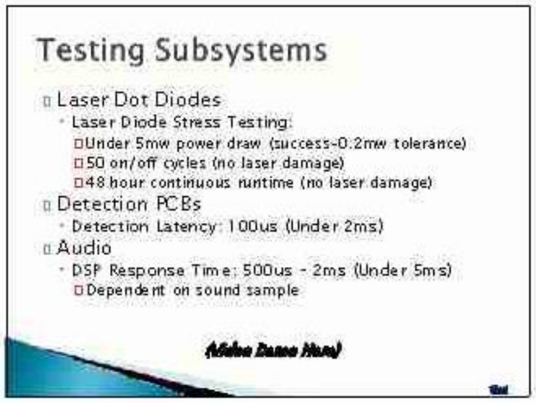


Progression



Design #1 Design #2 Design #3 Design #4

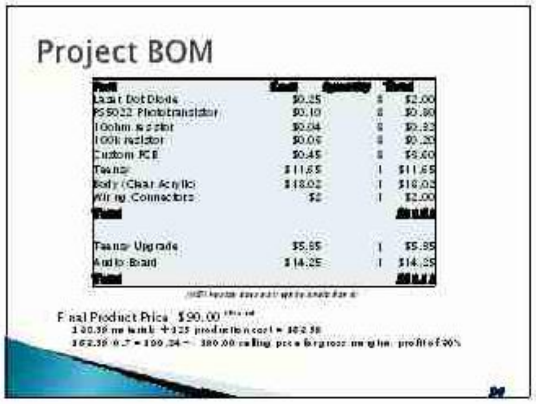
Testing Subsystems



- Laser Dot Diodes
  - Laser Diode Stress Testing:
    - Under 5mw power draw (success-0.2mw tolerance)
    - 50 on/off cycles (no laser damage)
    - 48 hour continuous runtime (no laser damage)
- Detection PCBs
  - Detection Latency: 100us (Under 2ms)
- Audio
  - DSP Response Time: 500us - 2ms (Under 5ms)
    - Dependent on sound sample

[illegible]

Project BOM

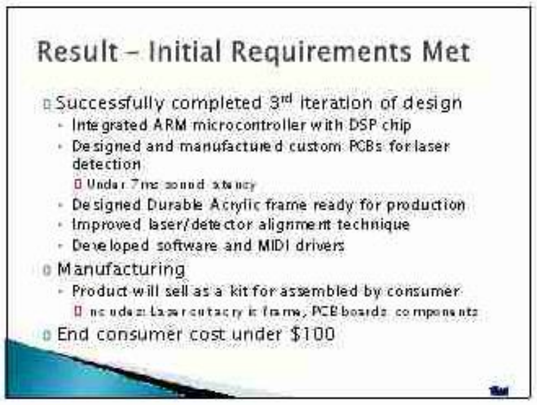


Part	Cost	Quantity	Total
Laser Dot Diode	$0.25	8	$2.00
PS5022 Phototransistor	$0.10	8	$0.80
10ohm resistor	$0.04	8	$0.32
100k resistor	$0.05	8	$0.20
Custom PCB	$0.45	8	$3.60
Teensy	$11.65	1	$11.65
Body (Clear Acrylic)	$18.02	1	$18.02
Wiring Connectors	$2	1	$2.00
Total			[illegible]
Teensy Upgrade	$5.95	1	$5.95
Audio Board	$14.25	1	$14.25
Total			[illegible]

[illegible]

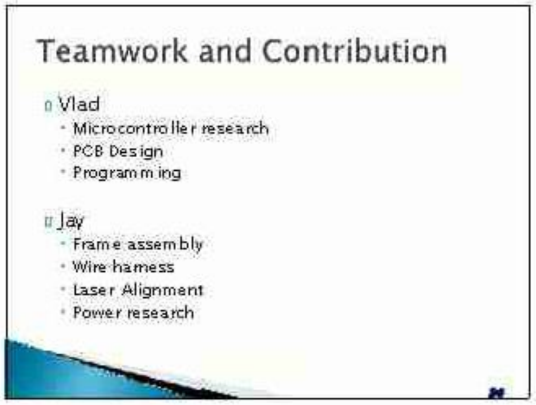
Final Product Price $90.00 [illegible]
$ 30.39 materials + $25 production cost = $62.38
$62.38 / 0.7 = $90.34 ~ $90.00 selling price for gross margin profit of 30%

Result – Initial Requirements Met



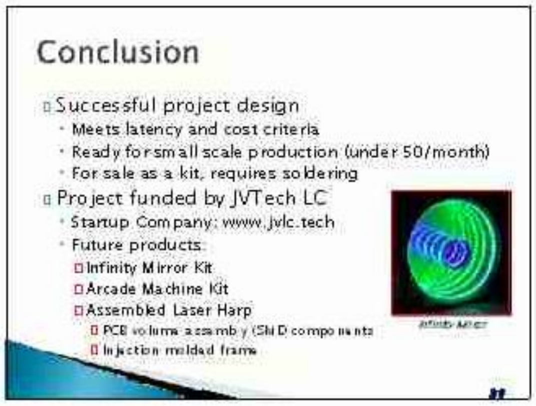
- Successfully completed 3rd iteration of design
  - Integrated ARM microcontroller with DSP chip
  - Designed and manufactured custom PCBs for laser detection
    - Under 7ms sound latency
  - Designed Durable Acrylic frame ready for production
  - Improved laser/detector alignment technique
  - Developed software and MIDI drivers
- Manufacturing
  - Product will sell as a kit for assembled by consumer
    - Includes: Laser cut acrylic frame, PCB boards, components
- End consumer cost under $100

Teamwork and Contribution



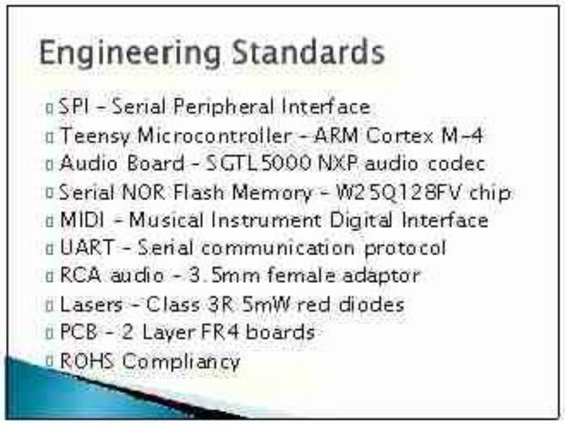
- Vlad
  - Microcontroller research
  - PCB Design
  - Programming
- Jay
  - Frame assembly
  - Wire harness
  - Laser Alignment
  - Power research

Conclusion



- Successful project design
  - Meets latency and cost criteria
  - Ready for small scale production (under 50/month)
  - For sale as a kit, requires soldering
- Project funded by JVTech LC
  - Startup Company: www.jvlc.tech
  - Future products:
    - Infinity Mirror Kit
    - Arcade Machine Kit
    - Assembled Laser Harp
      - PCB volume assembly (SMD components
      - Injection molded frame

Infinity Mirror



Engineering Standards

- SPI – Serial Peripheral Interface
- Teensy Microcontroller – ARM Cortex M-4
- Audio Board – SGTL5000 NXP audio codec
- Serial NOR Flash Memory – W25Q128FV chip
- MIDI – Musical Instrument Digital Interface
- UART – Serial communication protocol
- RCA audio – 3.5mm female adaptor
- Lasers – Class 3R 5mW red diodes
- PCB – 2 Layer FR4 boards
- ROHS Compliancy

d. *APPENDIX 8.4: PRELIMINARY EFFORTS – PRELIMINARY EFFORTS*

Embedded Systems Design (ECE473) - LASER HARP REPORT
(Referenced as Iteration 1 of the Design)

Objective

The main objective of this project encompassed designing and implementing a musical instrument driven by the interruption of laser beams representing the strings of a single octave harp. The Tiva Launchpad C board will control photo detector circuits for each laser beam and play a range of pitches on a speaker driven by small monoblock amplifier. Furthermore, the system must use bluetooth to remotely control professional lighting(LED Light Bar) through a second microcontroller with output to a DMX512 network cable. In full combination, the interruption of lasers in the harp must play sounds corresponding to the octave note and light up the proportionate segment of a light bar.

Equipment Used

- Tiva C Series LaunchPad Evaluation Kit(TM4C123G) with an ARM Cortex M4F Microcontroleler
- Arduino Uno board with ATmega328P Microcontroller
- DMX/RDM Shield for Arduino
- HC-05 Bluetooth Master/Slave module
- HC-06 Bluetooth Slave module
- American DJ Mega Bar RGBA 1 meter light fixture with 320 LEDs
- Mini class D audio amplifier with PAM8403 chip
- 8x10 ohm resistors
- 8x100 Kohm resistors
- 8x phototransistors
- 8x PS5022 phototransistors
- 8x 650nm 5V Mini Laser Dot Diode Module Heads (Red laser)

Theory/Procedure

The operation of the main laser harp circuit is based on principles of photo detection and laser illumination. The light emitted from a laser dot diode carries a significantly higher photonic energy than the light intensity of daylight and most standard indoor lights. Employing a photosensitive material at the region of light impact, a stable and versatile laser detection circuit can be built with very limited sensitivity to ambient lighting changes as compared to the laser beam. This allows for a laser beam detection circuit to operate properly under most lighting conditions and even a range of voltages when properly biased with control resistors. A phototransistor was chosen over cheaper photoresistors due to faster operation, increased sensitivity, and discrete turn-on characteristics which would alert the digital GPIO subsystem on the Tiva board when laser light is interrupted. A simple resistor ratio between power and ground can be used to bias the phototransistor to proper detection levels with the output of the voltage divider directly connected to a digital read port. An array of lasers and small photodetector circuits can be used to create a digital tripwire fence or in this case a photo-based array of strings for a digital instrument.

Once an array of digital inputs is properly controlled by the interruption of lasers, a sound must be generated for each input to audialize the light based instrument. One of the simplest methods of playing sound on a microcontroller relies on utilizing pulse width modulation on an output pin and generating a square wave of a particular frequency representing a specific tone. The pitch of the sound depends on the frequency and duty cycle driving the pin which can be connected to a small speaker with a common ground. For the purpose of this lab, a digital speaker was used through a simple digital amplifier chip to prevent overdrawing current from the Tiva board pins. The amplifier, as well as the laser dot diodes are powered directly from a 5 volt power source thus protection the microcontroller with limited current capability.

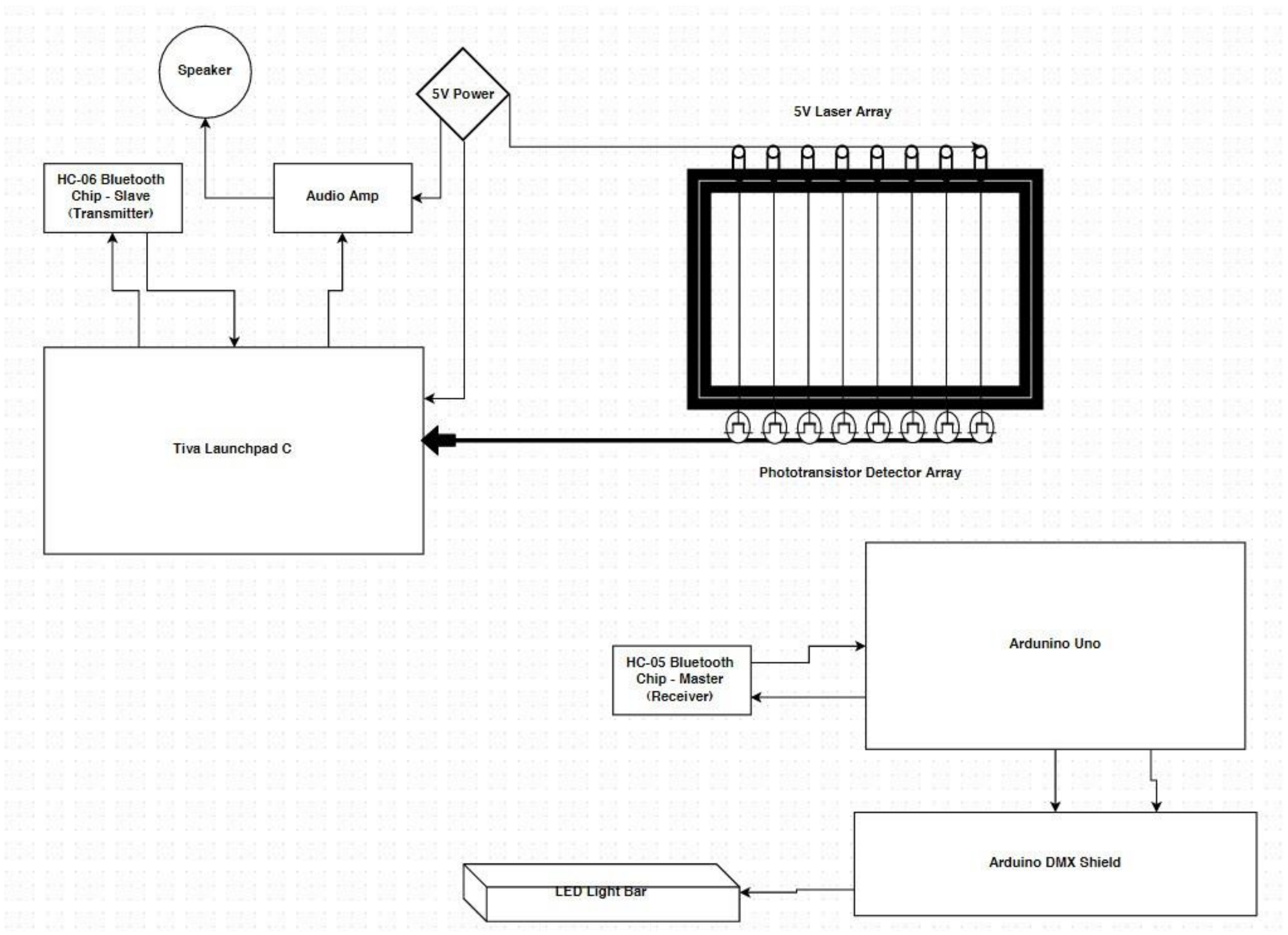
The next portion of the project relied on utilizing Bluetooth communication to another microprocessor to remotely control lighting equipment based on the current state of the laser harp system. Bluetooth was chosen mainly due to availability but 2-way communication could have been established using rf links, wifi modules, xbee, or a number of other wireless communication platforms. Two different bluetooth chips were used for each of the microcontrollers: HC-06 slave device, and HC-5 master/slave Bluetooth module. The slave device utilized only 4 pins for operation: 5/3.3V power, ground, Tx, and Rx while the master device employed an additional pin(Key) to determine operation state. Initially, much effort was undergone to simply use 2 slave devices and coordinate them through a Bluetooth master on a PC but significant issues with latency and general operation were encountered and a more stable but harder to implement approach was chosen(Bluetooth devices talk directly to each other). The slave device's receive and transmit pins were simply connected to a UART port on the TIVA board and a serial library was used to setup communication. The remote microcontroller relied on a similar principle for communicating with the Bluetooth device but first had to act as a serial programmer of the Master mode functionality to force the chip to connect with the remote Tiva subsystem. An arduino uno was used as the second microcontroller due to availability and simplicity of using an expansion shield for later part of the project. Since the board itself only had 1 hardware serial, a software serial library was used to connect the bluetooth device (HC-05) at the same time as the standard hardware serial port was used by the board for programming and light control(DMX512 shield). Such an implementation required a special programming script to run on the board to relay commands from its usb COM port to the software serial Rx and Tx pins of the Bluetooth module. The program acted as a simple relay of commands between the 2 serial ports to allow the Bluetooth chip to be programmed over the hardware serial interface connected to a standard USB port and controlled with tera-term. Once communication with the board was established, the master mode operation was defined using a series of AT commands as per manufacturer specifications. AT commands are one of the most common protocols for defining modem and similar subsystem operation based on a command set developed by Dennis Hayes for early smartmodems. The device characteristics were configured in this way and a communication link with the remote Bluetooth device (HC-06 on Tiva board) was established.

Once the two microcontroller boards established a serial communication channel over Bluetooth, the laser strings could control circuit functionality on the remote board. Essentially any time a string was plucked(laser beam interrupted), a hardware interrupt routine on the Tiva board would execute a subroutine and the current state of the entire system, all 8 strings, would be encoded into a single byte and transmitted over the air. (8 strings, each represented by either 1 or 0, thus 2^8 = 256 bits required to store state of entire system). The output from the Bluetooth chip here can also be connected to a computer or phone(used for initial testing) which opens us a range of possible uses and further expansions for the project.

Once the remote microcontroller(Arduino) had an established communication channel with the laser harp system, the data received could be used to control a variety of external hardware without any wires running back to the harp. This requirement presented the reason for using Bluetooth links as the goal was to control lighting on the other side of the room. Although a simple array of 8 leds could illustrate the functionality, a personal goal of this project required controlling intelligent light fixtures. The DMX512 serial protocol is an industry standard for professional lighting within the US and is used at almost every major music concert for controlling stage lighting equipment(DALI protocol used in Europe). One of the main advantages of such a protocol is the ability to chain link many slave devices creating a single bus controlled by a single master controller. The 512 represents the number of channels a single controller can control by driving a single byte value on each channel. An RBGA Bega Bar from ADJ was used for implementation and allowed control of up to 34 channel mode. The bar encompassed an array of 8 blocks of lights with each block consisting of an array of 4 colors diagonally strung with 4 lines each. Each color at every block could be controlled by the first 32 channels(8 blocks * 4 individual colors) with the last 2 channels limited to strobe and dimming functionality. A value of 0-255 sent to each channel would drive the leds with proportional intensity. In order to physically communicate with the light fixture, an DMX shield was connected to the microcontroller sharing the main hardware serial port. A dmx library(conceptinetics) was used for setting up the proper communication and custom functions were written to provide a simpler API control and decode segments and colors into proper channel numbers for the specific device.

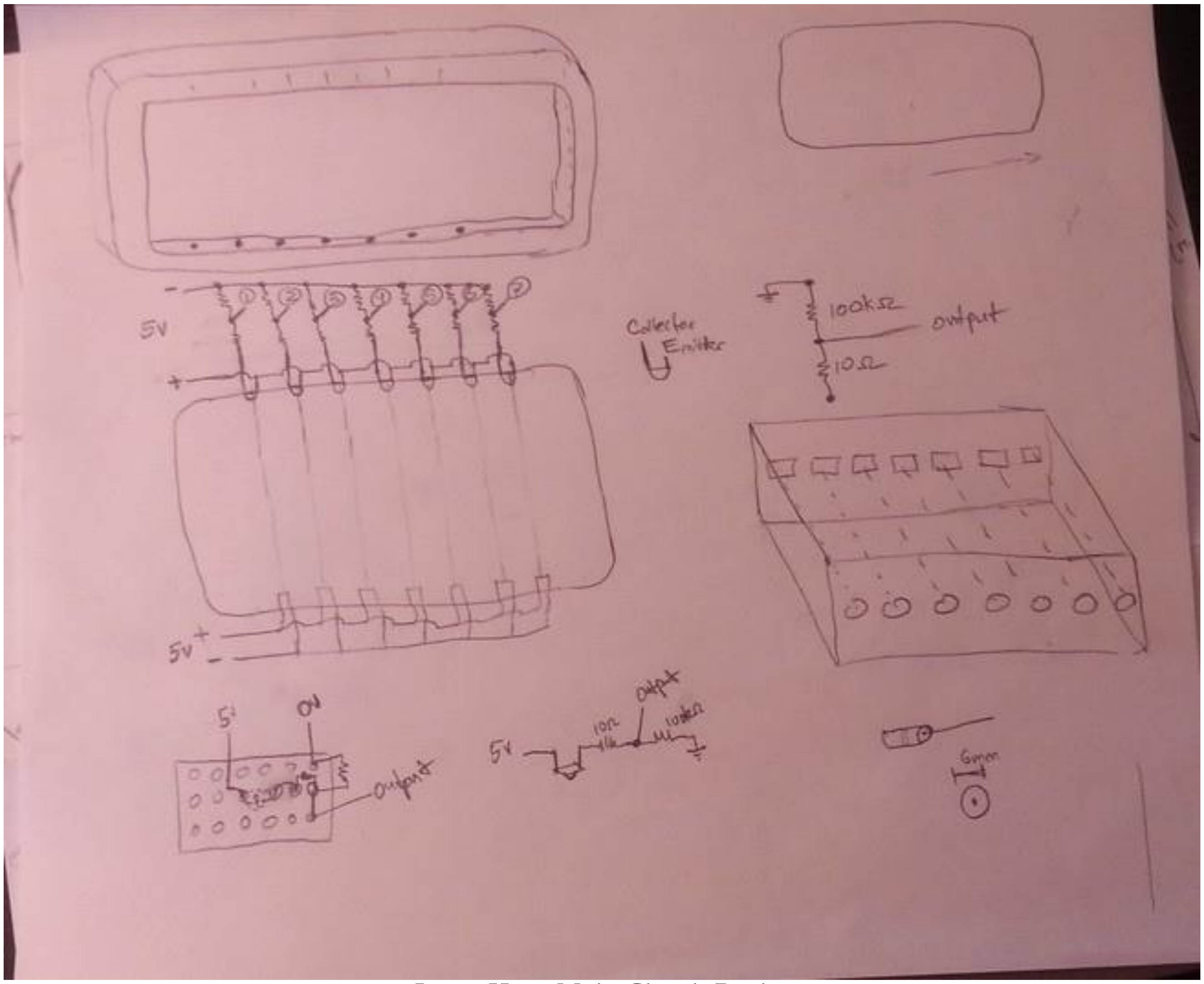
Implementation

The high level schematic involving all subsystems used is as follows:

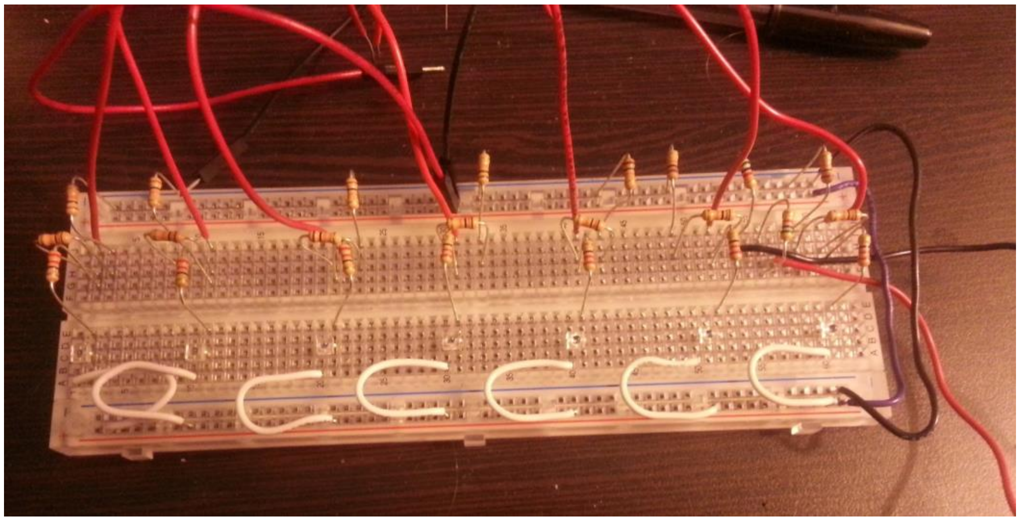

Speaker
5V Power
5V Laser Array
HC-06 Bluetooth Chip - Slave (Transmitter)
Audio Amp
Tiva Launchpad C
Phototransistor Detector Array
Ardunino Uno
HC-05 Bluetooth Chip - Master (Receiver)
Arduino DMX Shield
LED Light Bar

Device Schematic. Bluetooth chips allow the 2 systems to communicate

The first subsystem for design and implementation was the physical laser harp involving 8 red laser dot diodes(just the head part of a laser pointer), and 8 custom circuits for laser detection built using a phototransistor and 2x biasing resistors. Initially tested on a breadboard, schematics and images are provided below.

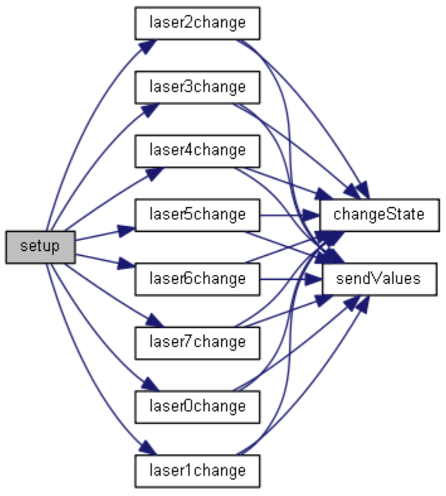


Laser Harp Main Circuit Design

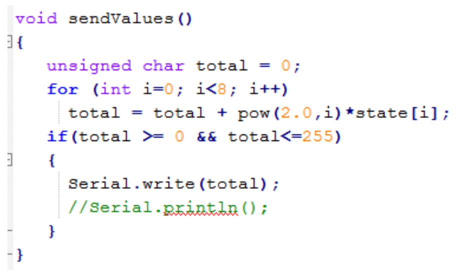


Testing photodetector circuits using handheld laser (reverse operation)

Once a physical device prototype was created, the controller software had to be defined. The laser detectors circuit resistor biasing creates a state in which the pins will read high when the lasers are uninterrupted, and will bring the line low when the string is played. The functionality was performed by attaching a hardware interrupt to 8 GPIO pins on the tiva(PE3, PE2, PE1, PE0, PD3, PD2, PD1, PD0) triggering a subroutine function on every change. The function performs an additional check to determine value state allowing for a slight debounce delay on the strings. Each ISR will call 2 functions: changeState, and sendValues. The change state function updates an array to keep track of the current state of the pin, this handles the issue with interrupts being triggered on any change. The next function mathematically encodes the laser array into a single character and sends it to the bluetooth transceiver over serial.

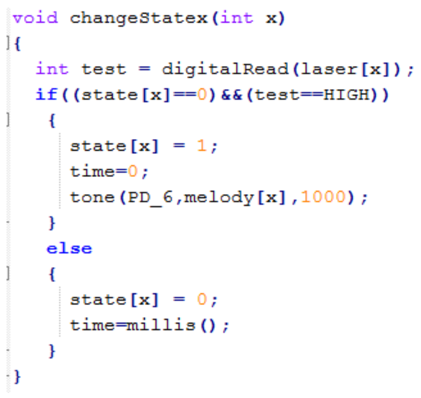


Laser Harp Controller - Main Call Graph showing Interrupt Routines



```
void sendValues()
{
    unsigned char total = 0;
    for (int i=0; i<8; i++)
      total = total + pow(2.0,i)*state[i];
    if(total >= 0 && total<=255)
    {
      Serial.write(total);
      //Serial.println();
    }
}
```

Send Values function encodes 8 bit array state[] into single char

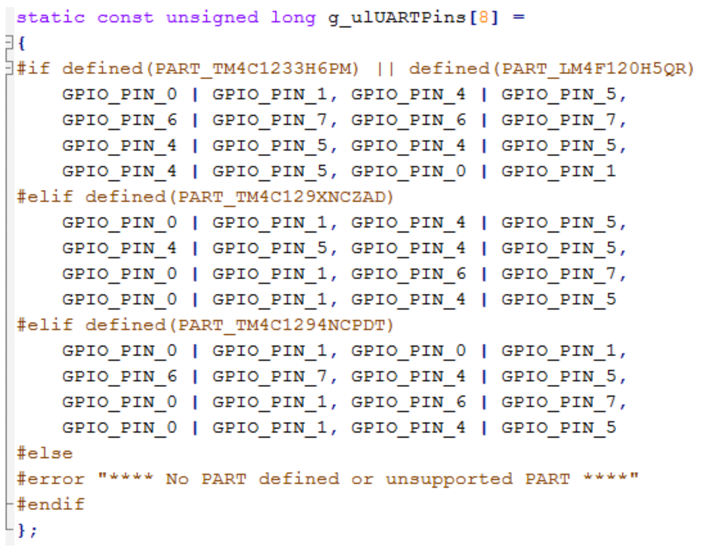


```
void changeStatex(int x)
{
  int test = digitalRead(laser[x]);
  if((state[x]==0)&&(test==HIGH))
   {
     state[x] = 1;
     time=0;
     tone(PD_6,melody[x],1000);
   }
   else
   {
     state[x] = 0;
     time=millis();
   }
}
```

changeState function updates the laser state array and plays proportionate pitch

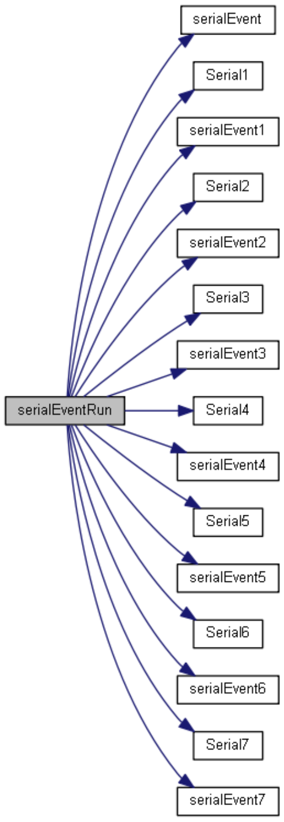
The pitches.h file is included to access a pitch library defining the sinusoid frequencies matching their respective notes. This allows calling atone function to pulse width modulate a pin that can drive a speaker. With this approach the sound is limited to a single frequency and cannot replicate a real instrument. A goal for the future is to access the Cortex M4 DSP on the tiva board, or use an external powerful digital signal processor such as laptop over bluetooth, to play real sounds and even encode full songs turning the harp into a dj music synthesizer.

Configuring the bluetooth slave device on the tiva board was as simple as connecting the Rx and Tx pins of the device to those on the board. This allows data replication on both bluetooth as well as direct serial connection on the board. Such a feature was extremely helpful for initial configuration of the device. The only catch in this case involved switching the pins as the main serial here was on the programmer side with switched Tx and Rx lines with respect to the board. Utilizing a different hardware serial port as the tiva has 8:



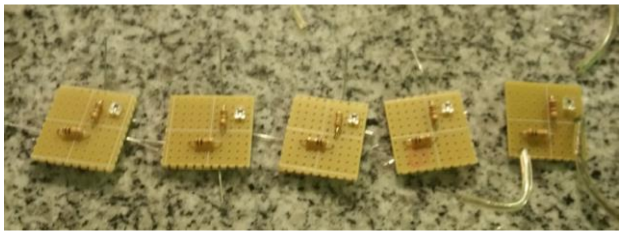
```
static const unsigned long g_ulUARTPins[8] =
{
#if defined(PART_TM4C1233H6PM) || defined(PART_LM4F120H5QR)
    GPIO_PIN_0 | GPIO_PIN_1, GPIO_PIN_4 | GPIO_PIN_5,
    GPIO_PIN_6 | GPIO_PIN_7, GPIO_PIN_6 | GPIO_PIN_7,
    GPIO_PIN_4 | GPIO_PIN_5, GPIO_PIN_4 | GPIO_PIN_5,
    GPIO_PIN_4 | GPIO_PIN_5, GPIO_PIN_0 | GPIO_PIN_1
#elif defined(PART_TM4C129XNCZAD)
    GPIO_PIN_0 | GPIO_PIN_1, GPIO_PIN_4 | GPIO_PIN_5,
    GPIO_PIN_4 | GPIO_PIN_5, GPIO_PIN_4 | GPIO_PIN_5,
    GPIO_PIN_0 | GPIO_PIN_1, GPIO_PIN_6 | GPIO_PIN_7,
    GPIO_PIN_0 | GPIO_PIN_1, GPIO_PIN_4 | GPIO_PIN_5
#elif defined(PART_TM4C1294NCPDT)
    GPIO_PIN_0 | GPIO_PIN_1, GPIO_PIN_0 | GPIO_PIN_1,
    GPIO_PIN_6 | GPIO_PIN_7, GPIO_PIN_4 | GPIO_PIN_5,
    GPIO_PIN_0 | GPIO_PIN_1, GPIO_PIN_6 | GPIO_PIN_7,
    GPIO_PIN_0 | GPIO_PIN_1, GPIO_PIN_4 | GPIO_PIN_5
#else
#error "**** No PART defined or unsupported PART ****"
#endif
};
```

Launchpad C Hardware Serial Mapping, used first element for demonstartion

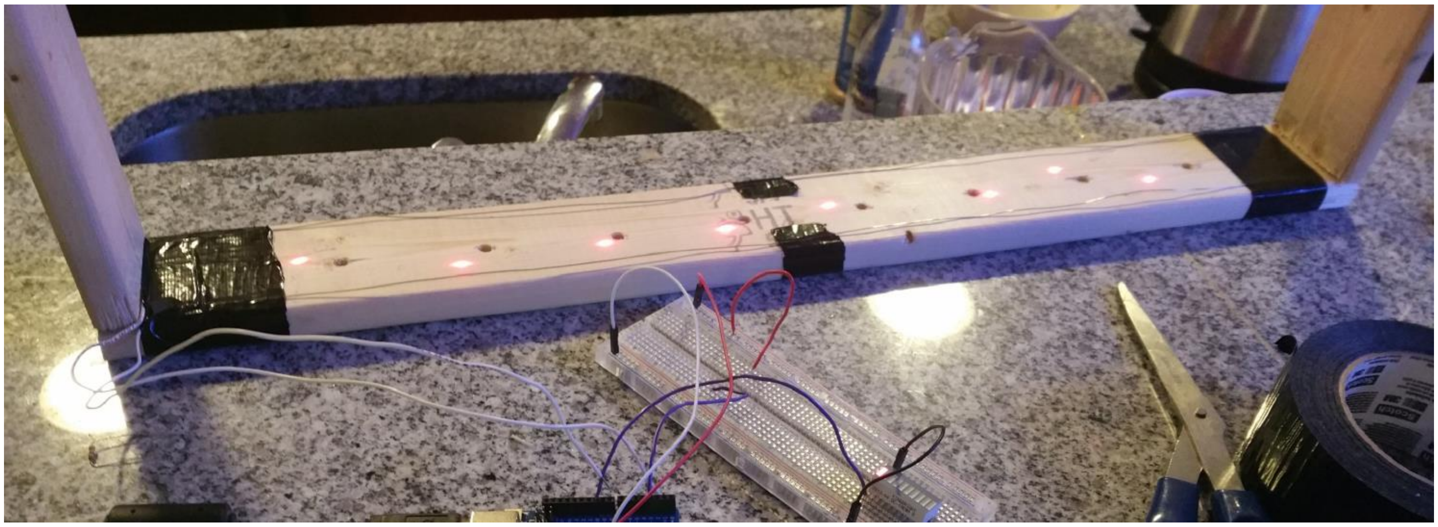


Hardware Serial Call Graph

Full source code for laser harp controller as well as libraries is provided in Appendix section. The final part of the Tiva system involved connecting an audio amp to the PWM output pin to amplify the power and drive a speaker. The power for the tiva board was provided by a 5v phone battery with a second USB cable directly driving the laser array and sound amplifier.



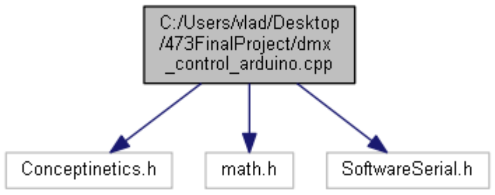
Building Photodetector Ciurcuts



Testing hardware operation, positive and negative rails to power transistors and led array to test reading values.

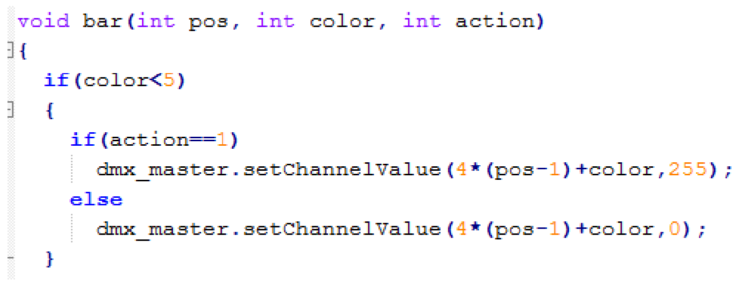
The next part of the project involved configuring a second microcontroller and wiring up a bluetooth master device. Since a second Tiva board was unavailable, an Arduino Uno had to suffice which led to some complications due to the inherent hardware limitations: single hardware serial port. Ideally, a microcontroller with at least 3 hardware UART GPIO ports should be used: Serial0 - programming interface, Serial1 - Bluetooth device communicator, Serial2 - DMX equipment controller. In this case, the single port was shared for programming and controlling lights, and the Bluetooth device was connected to general purpose IO with a software serial driver. The issue with software serial is delay due to hardware interrupts and the significant programming overhead on the main processor. Since this board does not perform critical functions, software serial was a successful workaround.

In order to communicate with the DMX shield and properly control DMX based light fixtures, a small library was used under the name Conceptinetics. This library provides calls to initialize a microcontroller as a DMX master device and directly assign values to individual, or a range of channels.


C:/Users/vlad/Desktop/473FinalProject/dmx_control_arduino.cpp
Conceptinetics.h
math.h
SoftwareSerial.h

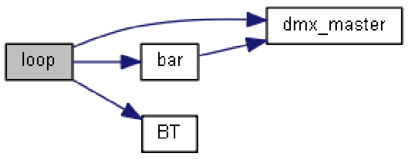
Dependency call graph for Arduino light driver

The DMX512 protocol uses serial UART communication at a baud rate of 250000 and the conceptinetics library takes care of handling Tx and Rx lines for the multiple device bus structure of the implementation. When configured as a slave device, individual part of the LED light bar can be controlled using a single function call dmx_master.setChannelValue(channel number, value 0-255). The limitation with lights used in this case came as a result of 4 colors for each block meaning a custom library was required to drive lights based on sector and color instead of channel number.



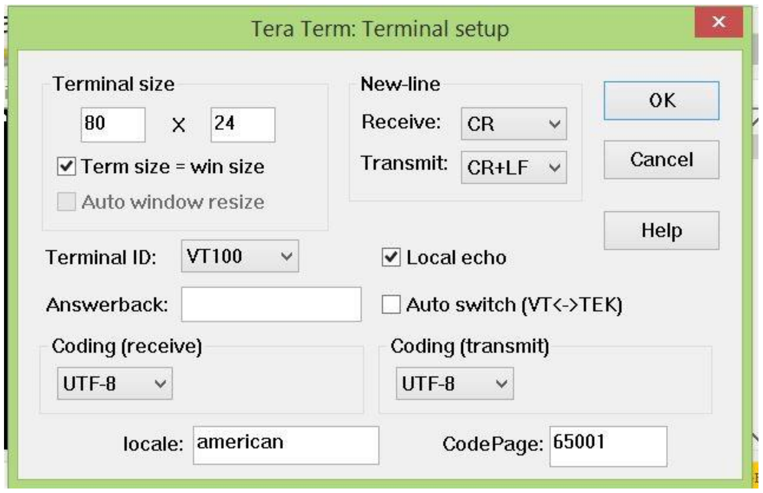
```
void bar(int pos, int color, int action)
{
  if(color<5)
  {
    if(action==1)
      dmx_master.setChannelValue(4*(pos-1)+color,255);
    else
      dmx_master.setChannelValue(4*(pos-1)+color,0);
  }
```

The bar function mathematically encodes block position and color into the proper channel number for the light fixture used.



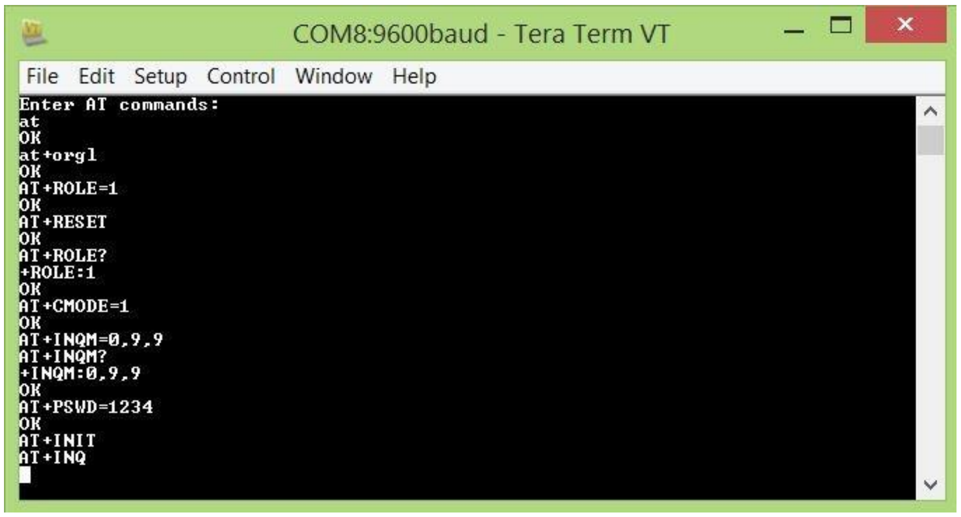
Main Arduino system call graph. dmx_master sets up communication, bar sets channel values, and BT function handles Bluetooth data from remote device.

The final part of this system required configuring a Bluetooth master device and receiving data from the Tiva board with changes in the laser harp. Due to limited hardware serial ports, first a custom program was programmed to the board which relayed signals from hardware USB serial to the software serial pins of the HC-05 Bluetooth device. The key pin on the device was brought high before powering on enabling master mode which does not allow other devices to directly connect over bluetooth. The serial data forwarding allowed a list of AT commands to be entered into the device to force connection with partner device(HC-06 Bluetooth slave on Tiva board). An important step to note here is configuring terminal to transmit both carriage return and line feed and show local echo to properly communicate with the device using a terminal.



Terminal size: 80 x 24
☑ Term size = win size
☐ Auto window resize
New-line
Receive: CR
Transmit: CR+LF
Terminal ID: VT100
Answerback:
☑ Local echo
☐ Auto switch (VT<->TEK)
Coding (receive): UTF-8
Coding (transmit): UTF-8
locale: american
CodePage: 65001

Must Enable CR+LR and Local echo to send serial commands to bluetooth device over device hardware serial port.


COM8:9600baud - Tera Term VT
File Edit Setup Control Window Help

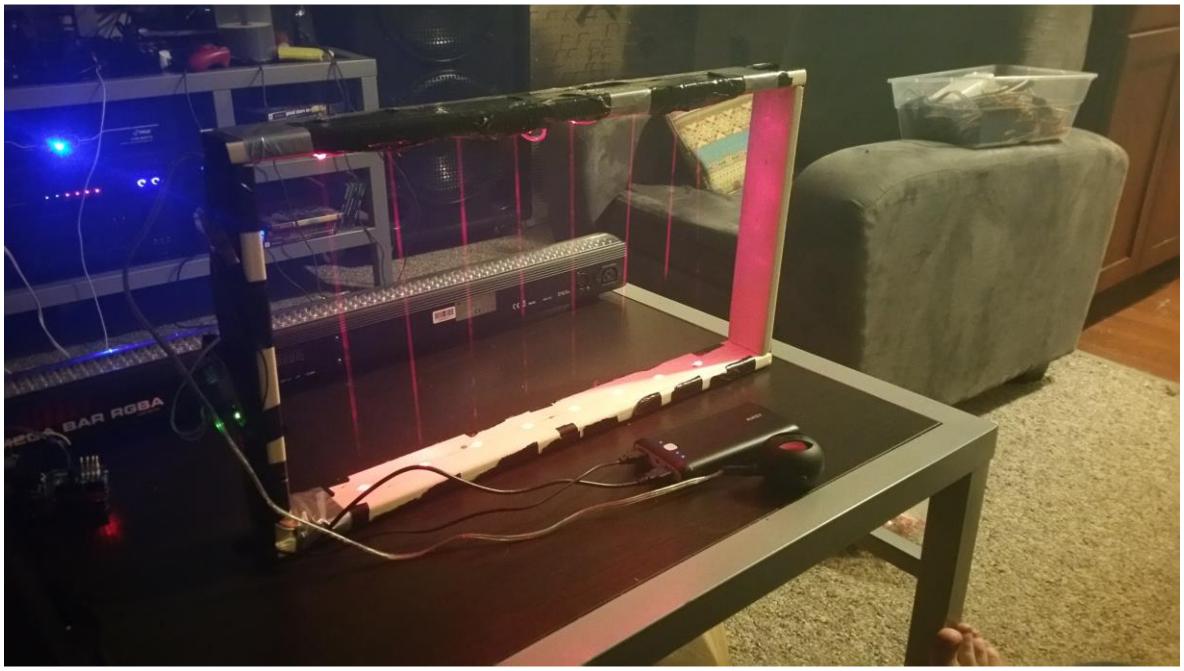
```
Enter AT commands:
at
OK
at+orgl
OK
AT+ROLE=1
OK
AT+RESET
OK
AT+ROLE?
+ROLE:1
OK
AT+CMODE=1
OK
AT+INQM=0,9,9
AT+INQM?
+INQM:0,9,9
OK
AT+PSWD=1234
OK
AT+INIT
AT+INQ
```

AT Command HC-05 Module Configuration

Once the Bluetooth device is configures, the microcontroller is flashed with the real program which still reads data from the Bluetooth device but reserves the hardware serial for use by the DMX shield. The functionality defined for the demonstration program simply recreates the state of the laser system on the light bar using a single color at a time. Color changes occur when the last laser is triggered with alternating direction.

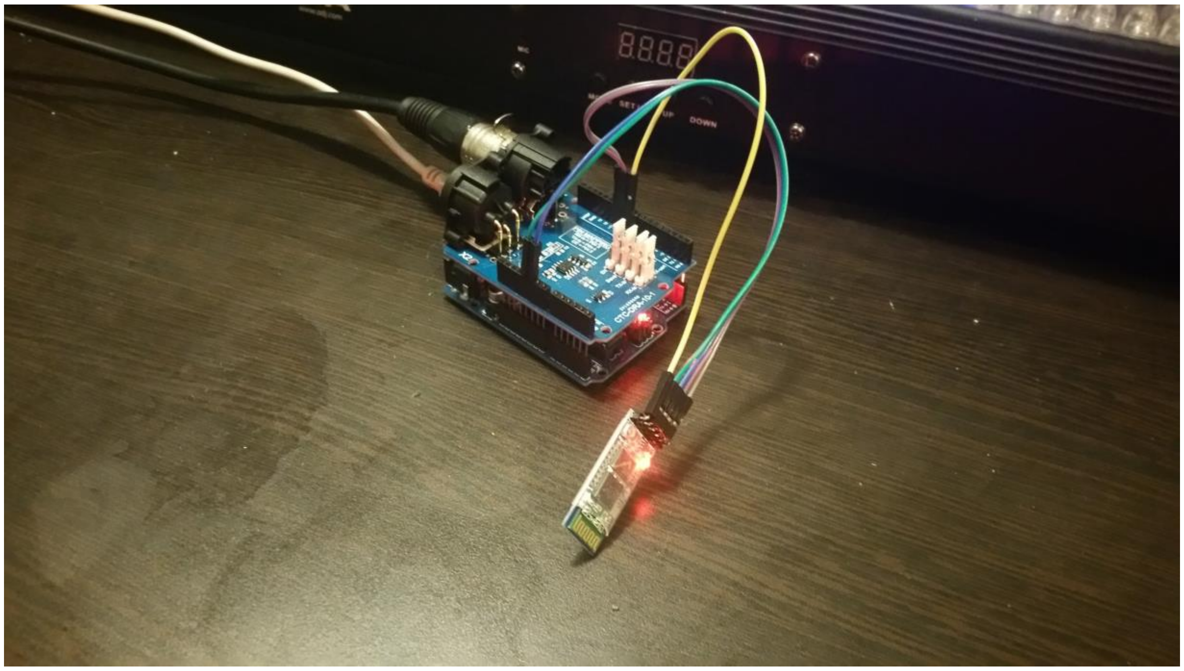
Results

The final result of combining all the subsystems created a musical instrument with laser strings which controls lighting equipment. Initially, many different approaches were attempted for various parts of the project with the final operation defined as per this report. The C4 octave was programmed into the device allowing each laser to control a musical note(left to right): C4, D4, E4, F4, G4, A4, B4, C5. The full effects of the sound and lighting are illustrated in the accompanying project video.

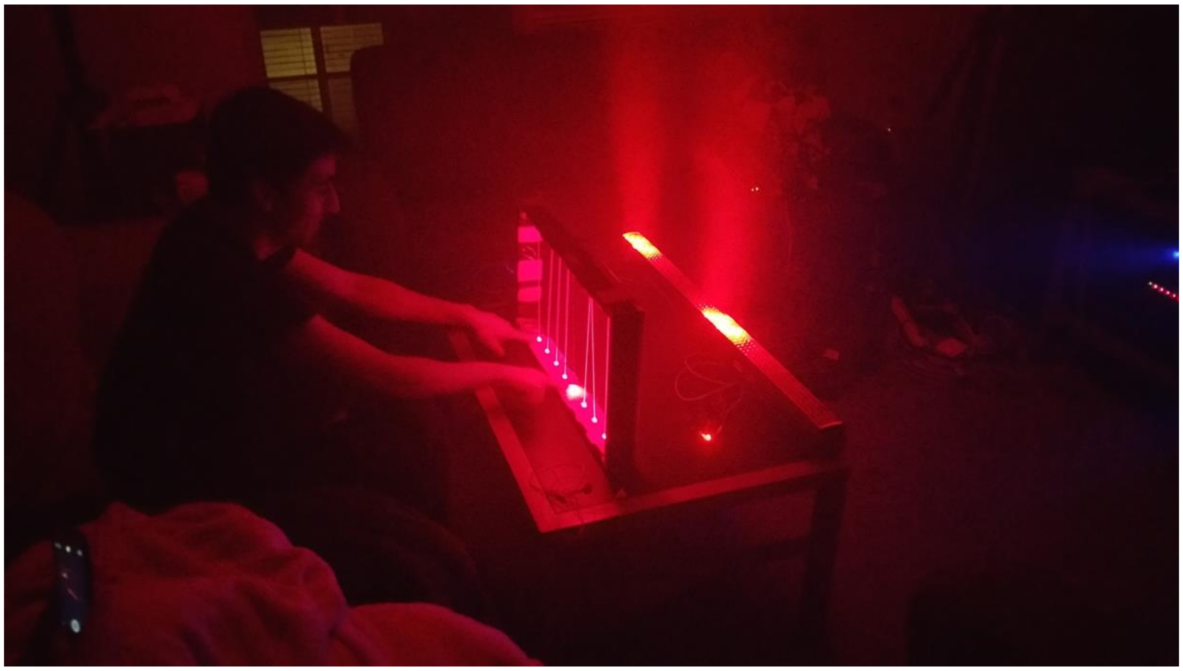


Laser harp system powered by external battery.

Tiva Launchpad C microcontroller, HC-06 bluetooth device and digital audio amp attached to Laser Harp Frame



Second Microcontroller: Arduino Uno with DMX shield and HC-05 Bluetooth module. Serial cable for board power, and DMX cable connecting to light fixture.

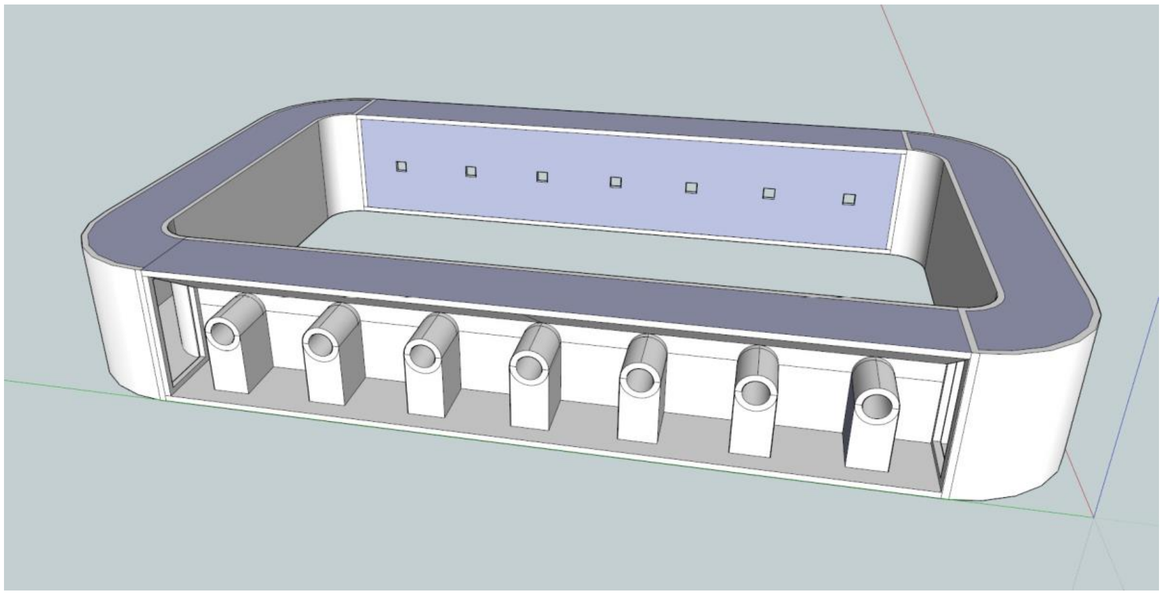


Complete system combining music and lighting

Conclusion/Future Goals

Throughout the completion of this project, a significant amount of knowledge and experience with various subsystems was gained. Perhaps the most difficult challenge encountered was selecting the most efficient and implementable solution to each individual portion of the project. Initially 2 Bluetooth slave devices were used and a computer master was meant to relay the information between the devices. After numerous approaches connecting each Bluetooth device to a virtual com port, a direct link between devices was chosen for stability. Matlab, Chuck, TTL and direct com port redirection were among the languages/processes attempted for connecting between bluetooth devices with various issues encountered in each case. (Matlab slow music playback and instability when dealing with Bluetooth device com port, Chuck could not connect to BT chips, Tera Term Language problem connecting to multiple com ports without setup delay or using relay file(could be fixed with interprocess communication), and direct com port redirect produced unstable data relaying.)

Although these approaches were not implemented currently, given more time such problems could be resolved allowing for example to play real music on computer/phone from laser harp Bluetooth data. Also, as previously mentioned, the internal DSP functionality of the Tiva board could be used in the future to provide realistic music output directly from the board. In terms of the lighting, a single LED bar was used for demonstration. Up to 3 such bars could be daisy chained with such an approach and controlled from the harp or any Bluetooth device sending data to the microcontroller. Furthermore, the implementation is not limited to bar lighting but the laser signals could be used to control rotation/movement of rotating beam lighting, laser scanning systems, or even the output from a DMX controlled fog/haze machine. The design for the Laser Harp system was not based on any designs online, rather developed from scratch including the circuitry and thus the next steps for this project include creating a nicer looking prototype and possibly manufacture in the future. A CAD model image to be created on a 3D printer is provided below.



Computer Aided Design drawing for Laser Harp Body

e. *APPENDIX 8.5: SOURCE CODE - ITERATION 1*

```
//Laser Harp Control Code for Tiva Launchpad C Microcontroller(Energia
IDE)
#include "pitches.h"

int laser[8] = {PE_3, PE_2, PE_1, PE_0, PD_3, PD_2, PD_1, PD_0}; int
state[8] = {0,0,0,0,0,0,0,0};
int melody[] = {NOTE_C4, NOTE_D4,NOTE_E4, NOTE_F4, NOTE_G4,NOTE_A4,
NOTE_B4, NOTE_C5};
int pitch = 0; int num = 0;
unsigned long time; void setup() {
Serial.begin(9600);

pinMode(laser[0], INPUT_PULLUP); pinMode(laser[1], INPUT_PULLUP);
pinMode(laser[2], INPUT_PULLUP); pinMode(laser[3], INPUT_PULLUP);
pinMode(laser[4], INPUT_PULLUP); pinMode(laser[5], INPUT_PULLUP);
pinMode(laser[6], INPUT_PULLUP); pinMode(laser[7], INPUT_PULLUP);

attachInterrupt(laser[0], laser0change, CHANGE);
attachInterrupt(laser[1], laser1change, CHANGE);
attachInterrupt(laser[2], laser2change, CHANGE);
attachInterrupt(laser[3], laser3change, CHANGE);
attachInterrupt(laser[4], laser4change, CHANGE);
attachInterrupt(laser[5], laser5change, CHANGE);
attachInterrupt(laser[6], laser6change, CHANGE);
attachInterrupt(laser[7], laser7change, CHANGE);
}

void sendValues()
{
unsigned char total = 0; for (int i=0; i<8; i++)
total = total + pow(2.0,i)*state[i]; if(total >= 0 && total<=255)
{
Serial.write(total);
}
}

void loop(){
}

void play(int x)
{
if
((state[0]==0)&&(state[1]==0)&&(state[2]==0)&&(state[3]==0)&&(state[4]==0
)&&(s tate[5]==0)&&(state[6]==0)&&(state[7]==0))
{
pitch=0; num=0;
}
if(pitch==0||time==0)
{
time=millis();
}
else if(pitch>0)
{
noTone(PD_6); tone(PD_6,pitch/num,100000); time=0;
}
}

void changeState(int x)
{
int test = digitalRead(laser[x]); if(test==HIGH)
{
tone(PD_6,melody[x],200); state[x]=1;
}
else
state[x]=0;
}

void changeStatex(int x)
{
int test = digitalRead(laser[x]); if((state[x]==0)&&(test==HIGH))
{
state[x] = 1; time=0;
tone(PD_6,melody[x],1000);
}
else
{
state[x] = 0; time=millis();
}
}

void laser0change()
{
changeState(0);

sendValues();
}

void laser1change()
{
changeState(1); sendValues();
}

void laser2change()
{
changeState(2); sendValues();
}

void laser3change()
{
changeState(3); sendValues();
}

void laser4change()
```

```
{
changeState(4); sendValues();
}

void laser5change()
{
changeState(5); sendValues();
}

void laser6change()
{
changeState(6); sendValues();
}

void laser7change()
{
changeState(7); sendValues();
}

//AT Configuration Commands to setup Bluetooth master to communicate with
//slave device AT+ORGL AT+RMAAD AT+ROLE=1 AT+RESET AT+CMODE=1

AT+INQM=0,9,9 AT+PSWD=1234 AT+UART=9600,0,0 AT+INIT
AT+INQ

//Arduino HC-05 Bluetooth device setup code
#include "math.h"
#include  <SoftwareSerial.h> SoftwareSerial BTSerial(10,11); // RX | TX
int offset = 0;
void setup()
{
pinMode(9, OUTPUT); // this pin will pull the HC-05 pin 34 (key pin) HIGH
to switch module to AT mode
pinMode(13, OUTPUT); digitalWrite(9, HIGH); Serial.begin(9600);
Serial.println("Enter AT commands:");
BTSerial.begin(38400);  // HC-05 default speed in AT command more
}

void loop()
{
//char message; char message;
// Keep reading from HC-05 and send to Arduino Serial Monitor if
(BTSerial.available())
{
message = BTSerial.read(); if(message=='1')
digitalWrite(13,HIGH); else if(message=='0')
digitalWrite(13,LOW); Serial.write(message);
}

// Keep reading from Arduino Serial Monitor and send to HC-05 if
(Serial.available())
{
message = Serial.read(); BTSerial.write(message);
```

```
}
}

//Arduino DMX controller and Bluetooth receiver code
#include <Conceptinetics.h>
#define DMX_MASTER_CHANNELS 100
#define RXEN_PIN    2

#include "math.h"
#include <SoftwareSerial.h> SoftwareSerial BT(10,11); // RX | TX
DMX_Master  dmx_master ( DMX_MASTER_CHANNELS, RXEN_PIN ); int offset = 0;

String message; unsigned long time;

void setup()
{
dmx_master.enable ();
pinMode(9, OUTPUT); // this pin will pull the HC-05 pin 34 (key pin) HIGH
to switch module to AT mode
pinMode(13, OUTPUT); digitalWrite(9, HIGH);
BT.begin(38400);    // HC-05 default speed in AT command more
}

void bar(int pos, int color, int action)
{
if(color<5)
{
if(action==1)
dmx_master.setChannelValue(4*(pos-1)+color,255); else
dmx_master.setChannelValue(4*(pos-1)+color,0);
}
}

int binArr[8]; int msg;
int color = 1;
int nextChange = 0; void loop()
{
int x = millis();  if(time>=0 && x>(time+1000))
{
dmx_master.setChannelRange(1,33,0);
}
else
{
for(int i=0;i<8;i++)
{
int colorx = color%4+1; if(colorx!=1)
bar(i+1,1,0); if(colorx!=2)
bar(i+1,2,0);

if(colorx!=3) bar(i+1,3,0);
if(colorx!=4) bar(i+1,4,0);
bar(i+1,colorx,binArr[i]);
}
```

```
}

if(BT.available())
{//while there is data available on the serial monitor msg = BT.read();
}
else if(!BT.available())
{
if(msg!=0)
{
time=millis(); for(int i=7;i>=0;i--)
{
if(msg>=pow(2.0,i))
{
binArr[i] = 1;
msg = msg - pow(2.0,i);
}
else
binArr[i] = 0;
}
if(nextChange==0&&binArr[0]==1)
{
color++; nextChange=7;
}
else if(nextChange==7&&binArr[7]==1)
{
color++; nextChange=0;
}
}
}
}

//Tiva Launchpad Pin Definition used for Energia Implementation
/*
**********************************************************************
*   pins_energia.h
*
*   Energia core files for LM4F
*   Copyright (c) 2012 Robert Wessels. All right reserved.
*
*   Contribution: Rei VILO
*
*********************************************************************

Derived from:
pins_arduino.h - Pin definition functions for Arduino Part of Arduino -
http://www.arduino.cc/

Copyright (c) 2007 David A. Mellis

This library is free software; you can redistribute it and/or modify it
under the terms of the GNU Lesser General Public  License as published by
the Free Software Foundation; either version 2.1 of the License, or (at
your option) any later version.
```

```
This library is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License
along with this library; if not, write to the Free Software Foundation,
Inc., 59 Temple Place, Suite 330, Boston, MA   02111-1307  USA
*/

#ifndef Pins_Arduino_h
#define Pins_Arduino_h

#define   LM4F120H5QR
#define   TM4C123GH6PM

#ifndef BV
#define BV(x) (1 << (x))
#endif
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "driverlib/adc.h"

//
// Pin names based on the silkscreen
//
static const uint8_t PB_5 = 2; static const uint8_t PB_0 = 3; static
const uint8_t PB_1 = 4; static const uint8_t PE_4 = 5; static const
uint8_t PE_5 = 6; static const uint8_t PB_4 = 7; static const uint8_t
PA_5 = 8; static const uint8_t PA_6 = 9; static const uint8_t PA_7 = 10;
static const uint8_t PA_2 = 11; static const uint8_t PA_3 = 12; static
const uint8_t PA_4 = 13; static const uint8_t PB_6 = 14;

static const uint8_t PB_7 = 15; static const uint8_t PF_0 = 17; static
const uint8_t PE_0 = 18; static const uint8_t PB_2 = 19; static const
uint8_t PD_0 = 23; static const uint8_t PD_1 = 24; static const uint8_t
PD_2 = 25; static const uint8_t PD_3 = 26; static const uint8_t PE_1 =
27; static const uint8_t PE_2 = 28; static const uint8_t PE_3 = 29;
static const uint8_t PF_1 = 30; static const uint8_t PF_4 = 31; static
const uint8_t PD_7 = 32; static const uint8_t PD_6 = 33; static const
uint8_t PC_7 = 34; static const uint8_t PC_6 = 35; static const uint8_t
PC_5 = 36; static const uint8_t PC_4 = 37; static const uint8_t PB_3 =
38; static const uint8_t PF_3 = 39; static const uint8_t PF_2 = 40;

static static   const const uint8_t uint8_t A0 A1   =
=   29;
28; //PE_3
//PE_2
static  const   uint8_t A2  =   27; //PE_1
static  const   uint8_t A3  =   18; //PE_0
static  const   uint8_t A4  =   26; //PD_3
static  const   uint8_t A5  =   25; //PD_2
static  const   uint8_t A6  =   24; //PD_1
static  const   uint8_t A7  =   23; //PD_0
static  const   uint8_t A8  =   6;  //PE_5
static  const   uint8_t A9  =   5;  //PE_4
static const uint8_t A10 =  7; //PB_4 static const uint8_t A11 =    2;
//PB_5

static const uint8_t RED_LED = 30; static const uint8_t GREEN_LED = 39;
static const uint8_t BLUE_LED = 40;

static const uint8_t PUSH1 = 31; static const uint8_t PUSH2 = 17; static
const uint8_t TEMPSENSOR = 0;

#ifdef ARDUINO_MAIN
const uint32_t port_to_base[] = { NOT_A_PORT,
(uint32_t) GPIO_PORTA_BASE, (uint32_t) GPIO_PORTB_BASE, (uint32_t)
GPIO_PORTC_BASE, (uint32_t) GPIO_PORTD_BASE, (uint32_t) GPIO_PORTE_BASE,

(uint32_t) GPIO_PORTF_BASE
};
const uint8_t digital_pin_to_timer[] = { NOT_ON_TIMER,  /*      dummy */
NOT_ON_TIMER,  /*       1 - 3.3V */ T1B0,       /*  2 - PB5 */
T2A0,    /*  3 - PB0 */
T2B,     /*  4 - PB1 */ NOT_ON_TIMER,    /*  5 - PE4 */ NOT_ON_TIMER,
/*  6 - PE5 */ T1A0,    /*  7 - PB4 */ NOT_ON_TIMER,    /*  8 - PA5 */
NOT_ON_TIMER,    /*  9 - PA6 */ NOT_ON_TIMER,    /*  10 - PA7 */
NOT_ON_TIMER,   /*  11 - PA2 */ NOT_ON_TIMER,   /*  12 - PA3 */
NOT_ON_TIMER,   /*  13 - PA4 */ T0A0,    /*  14 - PB6 */
T0B0,    /*  15 - PB7 */ NOT_ON_TIMER,  /*  16 - RST */ T0A1,    /*  17 -
PF0 */ NOT_ON_TIMER,   /*  18 - PE0 */ T3A,    /*  19 - PB2 */
NOT_ON_TIMER, /* 20 - GND */ NOT_ON_TIMER,     /*  21 - VBUS */
NOT_ON_TIMER,            /*  22 - GND */ WT2A,   /*  23 - PD0 */
WT2B,    /*  24 - PD1 */
WT3A,    /*  25 - PD2 */
WT3B,    /*  26 - PD3 */ NOT_ON_TIMER, /* 27 - PE1 */ NOT_ON_TIMER, /* 28
- PE2 */ NOT_ON_TIMER, /*  29 - PE3 */ T0B1,   /*  30 - PF1 */
T2A1,           /*  31 - PF4 */ WT5B,    /*  32 - PD7 */
WT5A,    /*  33 - PD6 */
WT1B,    /*  34 - PC7 */
WT1A,    /*  35 - PC6 */
WT0B,    /*  36 - PC5 */
WT0A,    /*  37 - PC4 */
T3B,     /*  38 - PB3 */
T1B1,    /*  39 - PF3 */
T1A1,    /*  40 - PF2 */
};
const uint8_t digital_pin_to_port[] = { NOT_A_PIN,  /*      dummy */
NOT_A_PIN, /*       1 - 3.3V */ PB,      /*  2 - PB5 */
PB, /*  3 - PB0 */
PB,         /*  4 - PB1 */ PE,  /*  5 - PE4 */

PE, /*  6 - PE5 */
```

```
PB,         /*  7 - PB4 */ PA,  /*  8 - PA5 */
PA, /*  9 - PA6 */
PA, /*  10 - PA7 */
PA, /*  11 - PA2 */
PA, /*  12 - PA3 */
PA, /*  13 - PA4 */
PB, /*  14 - PB6 */
PB,         /*  15 - PB7 */ NOT_A_PIN,     /*          16 - RST */ PF,
/*  17 - PF0 */ PE,     /*          18 - PE0 */ PB,         /*  19 - PB2
*/ NOT_A_PIN,  /*      20 - GND */ NOT_A_PIN,  /*      21 - VBUS */
NOT_A_PIN,      /*  22 - GND */ PD, /*  23 - PD0 */
PD, /*  24 - PD1 */
PD, /*  25 - PD2 */
PD,         /*  26 - PD3 */ PE, /*  27 - PE1 */
PE, /*  28 - PE2 */
PE, /*  29 - PE3 */
PF, /*  30 - PF1 */
PF, /*  31 - PF4 */
PD, /*  32 - PD7 */
PD, /*  33 - PD6 */
PC, /*  34 - PC7 */
PC, /*  35 - PC6 */
PC, /*  36 - PC5 */
PC, /*  37 - PC4 */
PB, /*  38 - PB3 */
PF, /*  39 - PF3 */
PF, /*  40 - PF2 */
};
const uint8_t digital_pin_to_bit_mask[] = { NOT_A_PIN,  /*      dummy */
NOT_A_PIN, /*      1 - 3.3V */ BV(5),      /*  2 - PB5 */
BV(0),  /*  3 - PB0 */ BV(1),                 /*  4 - PB1 */ BV(4),
/*      5 - PE4 */
BV(5),  /*  6 - PE5 */
BV(4),  /*  7 - PB4 */
BV(5),  /*  8 - PA5 */
BV(6),  /*  9 - PA6 */
BV(7),  /*  10 - PA7 */
BV(2),  /*  11 - PA2 */
BV(3),  /*  12 - PA3 */
BV(4),  /*  13 - PA4 */
BV(6),  /*  14 - PB6 */

BV(7),          /*  15 - PB7 */ NOT_A_PIN,     /*          16 - RST */
BV(0),          /*  17 - PF0 */ BV(0),      /*          18 - PE0 */
BV(2),          /*  19 - PB2 */ NOT_A_PIN,  /*      20 - GND */
NOT_A_PIN,  /*      21 - VBUS */
NOT_A_PIN,      /*  22 - GND */ BV(0),  /*  23 - PD0 */
BV(1),  /*  24 - PD1 */
BV(2),  /*  25 - PD2 */
BV(3),  /*  26 - PD3 */
BV(1),  /*  27 - PE1 */
BV(2),  /*  28 - PE2 */
BV(3),  /*  29 - PE3 */
```

```
};

const uint32_t timer_to_offset[] = { TIMER0,
TIMER0, TIMER0, TIMER0, TIMER1, TIMER1, TIMER1, TIMER1, TIMER2, TIMER2,
TIMER2, TIMER3, TIMER3, WTIMER0, WTIMER0, WTIMER1, WTIMER1, WTIMER2,
WTIMER2, WTIMER3, WTIMER3, WTIMER5, WTIMER5,

};

const uint8_t timer_to_ab[] = { TIMA,
TIMA, TIMB, TIMB, TIMA, TIMA, TIMB, TIMB, TIMA, TIMA, TIMB, TIMA, TIMB,
TIMA, TIMB, TIMA, TIMB, TIMA, TIMB, TIMA, TIMB, TIMA, TIMB,
};
const uint32_t timer_to_pin_config[] = {
GPIO_PB6_T0CCP0, GPIO_PF0_T0CCP0, GPIO_PB7_T0CCP1, GPIO_PF1_T0CCP1,
GPIO_PB4_T1CCP0, GPIO_PF2_T1CCP0, GPIO_PB5_T1CCP1, GPIO_PF3_T1CCP1,
GPIO_PB0_T2CCP0, GPIO_PF4_T2CCP0, GPIO_PB1_T2CCP1, GPIO_PB2_T3CCP0,
GPIO_PB3_T3CCP1, GPIO_PC4_WT0CCP0, GPIO_PC5_WT0CCP1, GPIO_PC6_WT1CCP0,
GPIO_PC7_WT1CCP1, GPIO_PD0_WT2CCP0, GPIO_PD1_WT2CCP1, GPIO_PD2_WT3CCP0,
GPIO_PD3_WT3CCP1, GPIO_PD6_WT5CCP0, GPIO_PD7_WT5CCP1,
};

const uint32_t digital_pin_to_analog_in[] = { ADC_CTL_TS,   /*      0 -
TempSensor  */ NOT_ON_ADC,  /*      1 - 3.3V*/ ADC_CTL_CH11,        /*  2
- PB5 */ NOT_ON_ADC,  /*      3 - PB0 */ NOT_ON_ADC,      /*  4 - PB1 */
ADC_CTL_CH9,     /*  5 - PE4 */ ADC_CTL_CH8,     /*  6 - PE5 */
ADC_CTL_CH10,    /*      7 - PB4 */ NOT_ON_ADC,      /*  8 - PA5 */
NOT_ON_ADC, /*  9 - PA6 */
NOT_ON_ADC, /*  10 - PA7 */
NOT_ON_ADC, /*  11 - PA2 */
NOT_ON_ADC, /*  12 - PA3 */
NOT_ON_ADC,     /*  13 - PA4 */ NOT_ON_ADC, /*      14 - PB6 */
NOT_ON_ADC, /*      15 - PB7 */ NOT_ON_ADC,     /*  16 - RST */
NOT_ON_ADC, /*      17 - PF0 */ ADC_CTL_CH3,        /*  18 - PE0 */
NOT_ON_ADC, /*      19 - PB2 */ NOT_ON_ADC,     /*  20 - GND */
NOT_ON_ADC, /*  21 - VBUS */ NOT_ON_ADC,    /*  22 - GND */
ADC_CTL_CH7,    /*      23 - PD0 */ ADC_CTL_CH6,    /*      24 - PD1 */
ADC_CTL_CH5,    /*      25 - PD2 */ ADC_CTL_CH4,        /*  26 - PD3 */
ADC_CTL_CH2,        /*  27 - PE1 */ ADC_CTL_CH1,        /*  28 - PE2 */
ADC_CTL_CH0,        /*  29 - PE3 */ NOT_ON_ADC, /*      30 - PF1 */
NOT_ON_ADC,     /*  31 - PF4 */ NOT_ON_ADC, /*      32 - PD7 */
NOT_ON_ADC, /*      33 - PD6 */ NOT_ON_ADC, /*      34 - PC7 */
NOT_ON_ADC, /*      35 - PC6 */ NOT_ON_ADC, /*      36 - PC5 */
NOT_ON_ADC, /*      37 - PC4 */ NOT_ON_ADC,     /*  38 - PB3 */
NOT_ON_ADC,     /*  39 - PF3 */ NOT_ON_ADC, /*      40 - PF2 */
};
#endif
#endif
```

f. *APPENDIX 8.6: SOURCE CODE – ITERATION 2 (Testing DSP)*

```
/*
This code was used for functionality testing and prototyping
The purpose was to showcase the functionality of using DSP
*/
#include <Audio.h>
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include <SerialFlash.h>
#include <Bounce.h>

// WAV files converted to code by wav2sketch
#include "AudioSampleSnare.h"        //
http://www.freesound.org/people/KEVOY/sounds/82583/
#include "AudioSampleTomtom.h"       //
http://www.freesound.org/people/zgump/sounds/86334/
#include "AudioSampleHihat.h"        //
http://www.freesound.org/people/mhc/sounds/102790/
#include "AudioSampleKick.h"         //
http://www.freesound.org/people/DWSD/sounds/171104/
#include "AudioSampleGong.h"         //
http://www.freesound.org/people/juskiddink/sounds/86773/
#include "AudioSampleCashregister.h" //
http://www.freesound.org/people/kiddpark/sounds/201159/

AudioPlayMemory    sound0;
AudioPlayMemory    sound1;  // six memory players, so we can play
AudioPlayMemory    sound2;  // all six sounds simultaneously
AudioPlayMemory    sound3;
AudioPlayMemory    sound4;
AudioPlayMemory    sound5;
AudioMixer4        mix1;    // two 4-channel mixers are needed in
AudioMixer4        mix2;    // tandem to combine 6 audio sources
AudioOutputI2S     headphones;
AudioOutputAnalog  dac;     // play to both I2S audio board and on-chip
DAC

const int button2 = 1;
// Create Audio connections between the components
AudioConnection c1(sound0, 0, mix1, 0);
AudioConnection c2(sound1, 0, mix1, 1);
AudioConnection c3(sound2, 0, mix1, 2);
AudioConnection c4(sound3, 0, mix1, 3);
AudioConnection c5(mix1, 0, mix2, 0);   // output of mix1 into 1st input
on mix2
AudioConnection c6(sound4, 0, mix2, 1);
AudioConnection c7(sound5, 0, mix2, 2);
AudioConnection c8(mix2, 0, headphones, 0);
AudioConnection c9(mix2, 0, headphones, 1);
AudioConnection c10(mix2, 0, dac, 0);
```

```
AudioControlSGTL5000 audioShield;

int state0=0;
int state1=0;
int state2=0;
int state3=0;
int state4=0;
int state5=0;
int state6=0;
int state7=0;


void setup() {

  pinMode(0, INPUT);
  pinMode(1, INPUT);
  pinMode(2, INPUT);
  pinMode(3, INPUT);
  pinMode(4, INPUT);
  pinMode(5, INPUT);
  pinMode(17,INPUT);
  pinMode(16,INPUT);
  pinMode(20,OUTPUT);
 digitalWrite(20,HIGH);
 pinMode(15,INPUT);
  AudioMemory(10);

  audioShield.enable();// turn on the output
  audioShield.volume(0.5);

  mix1.gain(0, 0.4);
  mix1.gain(1, 0.4);
  mix1.gain(2, 0.4);
  mix1.gain(3, 0.4);
  mix2.gain(1, 0.4);
  mix2.gain(2, 0.4);
}

void loop() {

  if(state0==0)
  {
    if(digitalRead(0)==0)
    {
      //sound1.play(AudioSampleSnare);
      sound1.play(AudioSampleTomtom);
      state0=1;
    }
  }
  else
    if(digitalRead(0)==1)
      state0=0;
```

```
if(state1==0)
{
  if(button2.fallingEdge())//digitalRead(1)==0)
  {
    sound1.play(AudioSampleTomtom);
    state1=1;
  }
}
else
  if(digitalRead(1)==1)
    state1=0;

if(state2==0)
{
  if(digitalRead(2)==0)
  {
    sound1.play(AudioSampleHihat);
    state2=1;
  }
}
else
  if(digitalRead(2)==1)
    state2=0;

if(state3==0)
{
  if(digitalRead(3)==0)
  {
    sound1.play(AudioSampleKick);
    state3=1;
  }
}
else
  if(digitalRead(3)==1)
    state3=0;

if(state4==0)
{
  if(digitalRead(4)==0)
  {
    sound1.play(AudioSampleGong);
    state4=1;
  }
}
else
  if(digitalRead(4)==1)
    state4=0;

if(state5==0)
{
  if(digitalRead(5)==0)
  {
        sound1.play(AudioSampleCashregister);
        state5=1;
      }
    }
    else
      if(digitalRead(5)==1)
        state5=0;

    if(state6==0)
    {
      if(digitalRead(17)==0)
      {
        sound1.play(AudioSampleKick);
        state6=1;
      }
    }
    else
      if(digitalRead(17)==1)
        state6=0;

    if(state7==0)
    {
      if(digitalRead(16)==0)
      {
        sound1.play(AudioSampleHihat);
        state7=1;
      }
    }
    else
      if(digitalRead(16)==1)
        state7=0;
  }
```

g. *APPENDIX 8.7: SOURCE CODE – ITERATION 3 (Senior Design Demo Code)*

```
/*
------------------------------------------
Digital Laser Harp Synthesizer Source Code
------------------------------------------
-The code provided was used to run the project demo for Senior Design
presentations
-This code is specific to the implementation used but showcases
production functionality
-Used in conjunction with provided libraries and DMX controller code
*/


//Required for implementing I2C communication
#include <Wire.h>

/*
-This section includes processor directives for the program
-These values are essentially static variables
*/
#define delayTime 1 //entire loop time
#define debounce delayTime*5 //millisecond debouce time for lasers
#define startChannel 36 //Midi imcrement for note c0 of first octave
(skipping lowest note segment)
#define setup1 0 //setup1 disables normal mode for custom setup
funcionality, used for 3D visual effect syncing
#define numNotes 85 //number of notes implemented for Fur Elise, can be
replaced with size(array)/sizeof(array[0])
/*

/*
-Conversion block for programming simplicity
-maps note values to proper note values within DMX protocol
-requires combination with proper offset value for octave to play
*/
#define c 0
#define cs 1
#define d 2
#define ds 3
#define e 4
#define f 5
#define fs 6
#define g 7
#define gs 8
#define a 9
#define as 10
#define b 11

/*
The following sectino defines global variables used throughout the
program
```

```
*/
const int channel = 1; //Channel value to use for DMX communication
int noteCounter = 0;  //keeps track of next note for cheat mode
byte dmxLight = 0; //encodes all active strings into single byte, 8
strings so 2^8 bits, simplifies communication
int values[8]; //analog values of each laser input
int old1[8] = {1,1,1,1,1,1,1,1}; //notes not yet sent play singnal for
int old2[8] = {0,0,0,0,0,0,0,0}; //notes play signal already sent for
int counterOn[8] = {0,0,0,0,0,0,0,0}; //time counter for each note on
time, used for debouncing
int counterOff[8] = {0,0,0,0,0,0,0,0}; //counter for off time
int octet[8] = {0,2,4,5,7,9,11,12}; //hardcoded offset values for
standard notes in octet, skips sharp notes
int channelIncrement=1; //specifies octave harp will play
int oldIncrement; //stores old value
int vis1 = 0; //counter for visual effects controller
int cheatMode = 0; //Initially player has full control over music
int offQueue[8]; //keep track of number of cycles each note was off,
based off delayTime

/*
-The three arrays below implement a structure for the optional
augmentation feature
-Notes, as well as their correspoinding octave values and timing are
stored
-Delays are only used for 'cheatMode2' (takes away control from player)
-Only one song is currently implemented: Fur Elise by Beethoven
-Future version will include integration with online note database
*/
int notes[] = {
  e, ds, e, ds, e, b, d, c, a,
  c, e, a, b, e, a, b, c,
  e, ds, e, ds, e, b, d, c, a,
  c, e, a, b, e, c, b, a,
  b, c, d, e, g, f, e, d, e, e, d, c, e, d, c, e,
  e, ds, e, ds, e, e, d, c, a,
  c, e, a, b, e, a, b, c,
  e, ds, e, ds, e, b, d, c, a,
  c, e, a, b, e, c, b, a
};
int notesIncrement[] = {
  4,4,4,4,4,3,4,4,3,
  3,3,3,3,3,3,3,4,
  4,4,4,4,4,3,4,4,3,
  3,3,3,3,3,4,3,3,
  3,4,4,4,3,4,4,4,3,4,4,3,4,4,4,4,
  4,3,4,4,3,3,3,3,3,
  3,3,3,4,4,4,4,4,
  4,3,4,4,3,3,3,3,3,
  3,4,3,3,3,4,3,3
};

int notesDelay[] = {
  350,350,350,350,350,400,400,400,1000,
  350,400,400,1000,400,400,400,1000,
  400,400,400,400,400,400,400,400,1000,
  400,400,400,1000,400,400,400,1000,
  400,400,400,1000,400,400,400,1000,400,400,400,1000,400,400,400,1000,
  400,350,350,350,350,400,400,400,1000,
  350,400,400,1000,400,400,400,1000,
  400,400,400,400,400,400,400,400,1000,
  400,400,400,1000,400,400,400,1000
};

void setup()
{
  Serial.begin(38400);//Define serial communication at 38400 baud rate
  Wire.begin(); //Open channel for I2C communication
}

/*
-Custom function specific to I2C communication to external
microcontroller
-Used to talk to arduino with DMX shield for controlling custom DJ
equipment
-This functionality is only part of demonstration, not final product
release
*/
void dmxControl()
{
  Wire.beginTransmission(8); // transmit to device #8
  Wire.write(dmxLight+1);   // ends one byte, offset by 1 to properly
send 0 value
  Wire.endTransmission();   //stop transmitting

}

/*
-Gets input from user through serial console
-Integers 1-5 switch instrument octave
-Integer 10 disables normal notes for live setup of 3D graphics within FL
Studio
-Integer 11 and 12 each activate progressive levels of user augmentation
code
*/
void parseUserInput()
{
    if (Serial.available())
    {
        oldIncrement = channelIncrement;
        channelIncrement = Serial.parseInt();  // will not be -1
        if (channelIncrement!=0)
        {
          Serial.println(channelIncrement);
          dmxLight = 0;
          dmxControl();
          if(channelIncrement<8)
          {
            cheatMode=0;
            for (int i=0; i<8; i++)
            {
              usbMIDI.sendNoteOff(startChannel+octet[i]+(oldIncrement-
1)*12, 0, channel);
            }
          }
          else if(channelIncrement==10)
            cheatMode=10;
          else if(channelIncrement==11)
          {
            noteCounter=0;
            cheatMode = 1;
          }
          else if(channelIncrement==12)
          {
            noteCounter=0;
            cheatMode = 2;
          }
        }
        else channelIncrement=oldIncrement;
    }
}

/*
-Store current voltage level of each laser detector (Percent of max)
-Inputs 5 and 4 were used for I2C, hence 2 loops
*/
void readLaserDetectors()
{
    for(int i=0;i<4;i++)
  {
    values[i] = analogRead(9-i);
  }
  for(int i=4;i<8;i++)
  {
    values[i] = analogRead(7-i);
  }
}

/*
-Handler code for a aprticular laser string (passed by value: int i)
-Analog inputs with configurable cutoff value are used for the laser
detector, this fixes improper laser alignment
-Cutoff value 200 represents 20% of maximum voltage
-Polling of inputs is used instead of interrupd driven appraoch for this
reason
*/
void programResponse(int i)
{
    if (values[i] < 200)
    {
      counterOff[i]=0;
      if (old1[i]==1)
      {
        old1[i]=0;
        counterOn[i]=0;
      }
      else if(counterOn[i]>debounce)
      {
        if(old2[i]==0)
        {
          vis1=vis1+i*14;
          if (cheatMode==10)
          {
            usbMIDI.sendNoteOn(1, vis1, channel);
          }
          else
          {
            usbMIDI.sendNoteOn(1, vis1, channel);
            if(cheatMode==0)
              usbMIDI.sendNoteOn(startChannel+octet[i]+(channelIncrement-
1)*12, 99, channel);
            else if(cheatMode==1)
            {

usbMIDI.sendNoteOn(startChannel+notes[noteCounter%numNotes]+(notesIncreme
nt[noteCounter%numNotes]-1)*12, 99, channel);
            offQueue[i] = noteCounter;
            noteCounter++;
            }
            else if(cheatMode==2)
            {

usbMIDI.sendNoteOn(startChannel+notes[noteCounter%numNotes]+(notesIncreme
nt[noteCounter%numNotes]-1)*12, 99, channel);
              delay(notesDelay[noteCounter%numNotes]*.75);

usbMIDI.sendNoteOff(startChannel+notes[noteCounter%numNotes]+(notesIncrem
ent[noteCounter%numNotes]-1)*12, 99, channel);
              noteCounter++;
            }
          }
          dmxLight+=pow(2.0,i);
          dmxControl();
          old2[i]=1;
        }
      }
      else
      {
        counterOn[i]++;
      }
    }
    else if (values[i] >= 200)
    {
      counterOn[i]=0;
      if (old1[i]==0)
      {
        old1[i]=1;
        counterOff[i]=0;
      }
      else if(counterOff[i]>debounce)
      {
        if(old2[i]==1)
        {
          vis1=vis1-i*14;
          if(!setup1)
            if(cheatMode==0)

usbMIDI.sendNoteOff(startChannel+octet[i]+(channelIncrement-1)*12, 0,
channel);
            else if(cheatMode==1)

usbMIDI.sendNoteOff(startChannel+notes[offQueue[i]%numNotes]+(notesIncrem
ent[offQueue[i]%numNotes]-1)*12, 0, channel);
            dmxLight-=pow(2.0,i);
            dmxControl();
          old2[i]=0;
        }
      }
      else
      {
        counterOff[i]++;
      }
    }
}

/*
Main driver program for the harp functionality
*/
void loop()
{
    parseUserInput(); //Read serial to check if user entered new command
    readLaserDetectors(); //Store current state of laser detecors into
array
    for(int i=0;i<8;i++) //run for each laser string
    {
        programResponse(i); //play or stop note and update light
controller
    }
    delay(delayTime); //wait 1ms (Can be removed if required, used mainly
for simple debouncing delay implementation)
}
```

```
/*
Supplemental program used on Arduino
Receives data from laser harp over I2C and controls a DMX based DJ light
bar
*/
#include <Conceptinetics.h>
#define DMX_MASTER_CHANNELS   100
#define RXEN_PIN                2
#include "math.h"
#include <Wire.h>
#include <stdlib.h>

DMX_Master        dmx_master ( DMX_MASTER_CHANNELS, RXEN_PIN );
//int offset = 0;
const int led_pin = 6;
const int transmit_pin = 12;
const int receive_pin = 11;
const int transmit_en_pin = 3;

int msg=0;
int binArr[8];

void setup()
{
  delay(1000);
  dmx_master.enable ();
  Wire.begin(8);                // join i2c bus with address #8
  Wire.onReceive(receiveEvent); // register event
}

void receiveEvent(int howMany) {
  msg = Wire.read();    // receive byte as an integer
}

void bar(int pos, int color, int action)
{
  if(color<5)
  {
    if(action==1)
      dmx_master.setChannelValue(4*(pos)+color,255);
    else
      dmx_master.setChannelValue(4*(pos)+color,0);
  }

}

int colorValue[] = {1,1,1,1,1,1,1,1};

void loop()
{
    if (msg!=0)
    {
      msg--;
      for(int i=7;i>=0;i--)
        {
          if(msg>=pow(2.0,i))
          {
            binArr[i] = 1;
            msg = msg - pow(2.0,i);
          }
          else
            binArr[i] = 0;
        }
      for(int i=0;i<8;i++)
      {
        bar(i,colorValue[i],binArr[i]);
        if(binArr[i]==0)
          colorValue[i] = rand()%4+1;
      }
    }
}
```

h. *APPENDIX 8.8: ADDITIONAL SOURCE CODE AND LIBRARIES*

```
//USB MIDI library used for serial communication using MIDI protocol
#include "usb_dev.h"
#include "usb_midi.h"
#include "core_pins.h" // for yield()
#include "HardwareSerial.h"

#ifdef MIDI_INTERFACE // defined by usb_dev.h -> usb_desc.h
#if F_CPU >= 20000000

uint8_t usb_midi_msg_channel;
uint8_t usb_midi_msg_type;
uint8_t usb_midi_msg_data1;
uint8_t usb_midi_msg_data2;
uint8_t usb_midi_msg_sysex[USB_MIDI_SYSEX_MAX];
uint8_t usb_midi_msg_sysex_len;
void (*usb_midi_handleNoteOff)(uint8_t ch, uint8_t note, uint8_t vel) =
NULL;
void (*usb_midi_handleNoteOn)(uint8_t ch, uint8_t note, uint8_t vel) =
NULL;
void (*usb_midi_handleVelocityChange)(uint8_t ch, uint8_t note, uint8_t
vel) = NULL;
void (*usb_midi_handleControlChange)(uint8_t ch, uint8_t control, uint8_t
value) = NULL;
void (*usb_midi_handleProgramChange)(uint8_t ch, uint8_t program) = NULL;
void (*usb_midi_handleAfterTouch)(uint8_t ch, uint8_t pressure) = NULL;
void (*usb_midi_handlePitchChange)(uint8_t ch, int pitch) = NULL;
void (*usb_midi_handleSysEx)(const uint8_t *data, uint16_t length,
uint8_t complete) = NULL;
void (*usb_midi_handleRealTimeSystem)(uint8_t rtb) = NULL;
void (*usb_midi_handleTimeCodeQuarterFrame)(uint16_t data) = NULL;

// Maximum number of transmit packets to queue so we don't starve other
endpoints for memory
#define TX_PACKET_LIMIT 6
static usb_packet_t *rx_packet=NULL;
static usb_packet_t *tx_packet=NULL;
static uint8_t transmit_previous_timeout=0;
static uint8_t tx_noautoflush=0;


// When the PC isn't listening, how long do we wait before discarding
data?
#define TX_TIMEOUT_MSEC 40

#if F_CPU == 168000000
  #define TX_TIMEOUT (TX_TIMEOUT_MSEC * 1100)
#elif F_CPU == 144000000
  #define TX_TIMEOUT (TX_TIMEOUT_MSEC * 932)
#elif F_CPU == 120000000
  #define TX_TIMEOUT (TX_TIMEOUT_MSEC * 764)
```

```
#elif F_CPU == 96000000
  #define TX_TIMEOUT (TX_TIMEOUT_MSEC * 596)
#elif F_CPU == 72000000
  #define TX_TIMEOUT (TX_TIMEOUT_MSEC * 512)
#elif F_CPU == 48000000
  #define TX_TIMEOUT (TX_TIMEOUT_MSEC * 428)
#elif F_CPU == 24000000
  #define TX_TIMEOUT (TX_TIMEOUT_MSEC * 262)
#endif


void usb_midi_write_packed(uint32_t n)
{
      uint32_t index, wait_count=0;

      tx_noautoflush = 1;
      if (!tx_packet) {
            while (1) {
                  if (!usb_configuration) {
                        //serial_print("error1\n");
                              return;
                  }
                  if (usb_tx_packet_count(MIDI_TX_ENDPOINT) <
TX_PACKET_LIMIT) {
                              tx_packet = usb_malloc();
                              if (tx_packet) break;
                  }
                  if (++wait_count > TX_TIMEOUT ||
transmit_previous_timeout) {
                              transmit_previous_timeout = 1;
                        //serial_print("error2\n");
                              return;
                  }
                  yield();
            }
      }
      transmit_previous_timeout = 0;
      index = tx_packet->index;
      //*((uint32_t *)(tx_packet->buf) + index++) = n;
      ((uint32_t *)(tx_packet->buf))[index++] = n;
      if (index < MIDI_TX_SIZE/4) {
            tx_packet->index = index;
      } else {
            tx_packet->len = MIDI_TX_SIZE;
            usb_tx(MIDI_TX_ENDPOINT, tx_packet);
            tx_packet = usb_malloc();
      }
      tx_noautoflush = 0;
}

void usb_midi_send_sysex(const uint8_t *data, uint32_t length)
{
        // TODO: MIDI 2.5 lib automatically adds start and stop bytes
        while (length > 3) {
                usb_midi_write_packed(0x04 | (data[0] << 8) | (data[1] <<
16) | (data[2] << 24));
                data += 3;
                length -= 3;
        }
        if (length == 3) {
                usb_midi_write_packed(0x07 | (data[0] << 8) | (data[1] <<
16) | (data[2] << 24));
        } else if (length == 2) {
                usb_midi_write_packed(0x06 | (data[0] << 8) | (data[1] <<
16));
        } else if (length == 1) {
                usb_midi_write_packed(0x05 | (data[0] << 8));
        }
}

void usb_midi_flush_output(void)
{
      if (tx_noautoflush == 0 && tx_packet && tx_packet->index > 0) {
            tx_packet->len = tx_packet->index * 4;
            usb_tx(MIDI_TX_ENDPOINT, tx_packet);
            tx_packet = usb_malloc();
      }
}

void static sysex_byte(uint8_t b)
{
      // when buffer is full, send another chunk to handler.
      if (usb_midi_msg_sysex_len == USB_MIDI_SYSEX_MAX) {
            if (usb_midi_handleSysEx) {
                  (*usb_midi_handleSysEx)(usb_midi_msg_sysex,
usb_midi_msg_sysex_len, 0);
                  usb_midi_msg_sysex_len = 0;
            }
      }
      if (usb_midi_msg_sysex_len < USB_MIDI_SYSEX_MAX) {
            usb_midi_msg_sysex[usb_midi_msg_sysex_len++] = b;
      }
}


int usb_midi_read(uint32_t channel)
{
      uint32_t n, index, ch, type1, type2;

      if (!rx_packet) {
            if (!usb_configuration) return 0;
            rx_packet = usb_rx(MIDI_RX_ENDPOINT);
            if (!rx_packet) return 0;
            if (rx_packet->len == 0) {
                  usb_free(rx_packet);
                  rx_packet = NULL;
```

```
                        return 0;
                }
        }
        index = rx_packet->index;
        //n = *(uint32_t *)(rx_packet->buf + index);
        n = ((uint32_t *)rx_packet->buf)[index/4];
        //serial_print("midi rx, n=");
        //serial_phex32(n);
        //serial_print("\n");
        index += 4;
        if (index < rx_packet->len) {
                rx_packet->index = index;
        } else {
                usb_free(rx_packet);
                rx_packet = usb_rx(MIDI_RX_ENDPOINT);
        }
        type1 = n & 15;
        type2 = (n >> 12) & 15;
        ch = ((n >> 8) & 15) + 1;
        if (type1 >= 0x08 && type1 <= 0x0E) {
                if (channel && channel != ch) {
                        // ignore other channels when user wants single channel
read
                        return 0;
                }
                if (type1 == 0x08 && type2 == 0x08) {
                        usb_midi_msg_type = 0;                  // 0 = Note off
                        if (usb_midi_handleNoteOff)
                                (*usb_midi_handleNoteOff)(ch, (n >> 16), (n >>
24));
                } else
                if (type1 == 0x09 && type2 == 0x09) {
                        if ((n >> 24) > 0) {
                                usb_midi_msg_type = 1;          // 1 = Note on
                                if (usb_midi_handleNoteOn)
                                        (*usb_midi_handleNoteOn)(ch, (n >> 16), (n
>> 24));
                        } else {
                                usb_midi_msg_type = 0;          // 0 = Note off
                                if (usb_midi_handleNoteOff)
                                        (*usb_midi_handleNoteOff)(ch, (n >> 16), (n
>> 24));
                        }
                } else
                if (type1 == 0x0A && type2 == 0x0A) {
                        usb_midi_msg_type = 2;                  // 2 = Poly
Pressure
                        if (usb_midi_handleVelocityChange)
                                (*usb_midi_handleVelocityChange)(ch, (n >> 16),
(n >> 24));
                } else
                if (type1 == 0x0B && type2 == 0x0B) {
                  usb_midi_msg_type = 3;              // 3 = Control
Change
                  if (usb_midi_handleControlChange)
                        (*usb_midi_handleControlChange)(ch, (n >> 16), (n
>> 24));
            } else
            if (type1 == 0x0C && type2 == 0x0C) {
                  usb_midi_msg_type = 4;              // 4 = Program
Change
                  if (usb_midi_handleProgramChange)
                        (*usb_midi_handleProgramChange)(ch, (n >> 16));
            } else
            if (type1 == 0x0D && type2 == 0x0D) {
                  usb_midi_msg_type = 5;              // 5 = After Touch
                  if (usb_midi_handleAfterTouch)
                        (*usb_midi_handleAfterTouch)(ch, (n >> 16));
            } else
            if (type1 == 0x0E && type2 == 0x0E) {
                  usb_midi_msg_type = 6;              // 6 = Pitch Bend
                  if (usb_midi_handlePitchChange)
                        (*usb_midi_handlePitchChange)(ch,
                          ((n >> 16) & 0x7F) | ((n >> 17) & 0x3F80));
            } else {
                  return 0;
            }
            return_message:
            usb_midi_msg_channel = ch;
            usb_midi_msg_data1 = (n >> 16);
            usb_midi_msg_data2 = (n >> 24);
            return 1;
      }
      if (type1 == 0x04) {
            sysex_byte(n >> 8);
            sysex_byte(n >> 16);
            sysex_byte(n >> 24);
            return 0;
      }
      if (type1 >= 0x05 && type1 <= 0x07) {
            sysex_byte(n >> 8);
            if (type1 >= 0x06) sysex_byte(n >> 16);
            if (type1 == 0x07) sysex_byte(n >> 24);
            usb_midi_msg_data1 = usb_midi_msg_sysex_len;
            usb_midi_msg_sysex_len = 0;
            usb_midi_msg_type = 7;                    // 7 = Sys Ex
            if (usb_midi_handleSysEx)
                  (*usb_midi_handleSysEx)(usb_midi_msg_sysex,
usb_midi_msg_data1, 1);
            return 1;
      }
      if (type1 == 0x0F) {
            // TODO: does this need to be a full MIDI parser?
            // What software actually uses this message type in practice?
            if (usb_midi_msg_sysex_len > 0) {
                  // From David Sorlien, dsorlien at gmail.com,
http://axe4live.wordpress.com
                  // OSX sometimes uses Single Byte Unparsed to
                  // send bytes in the middle of a SYSEX message.
                  sysex_byte(n >> 8);
            } else {
                  // From Sebastian Tomczak, seb.tomczak at gmail.com
                  // http://little-scale.blogspot.com/2011/08/usb-midi-
game-boy-sync-for-16.html
                  usb_midi_msg_type = 8;
                  if (usb_midi_handleRealTimeSystem)
                        (*usb_midi_handleRealTimeSystem)(n >> 8);
                  goto return_message;
            }
      }
      if (type1 == 0x02) {
            // From Timm Schlegelmilch, karg.music at gmail.com
            // http://karg-music.blogspot.de/2015/06/receiving-midi-time-
codes-over-usb-with.html
            usb_midi_msg_type = 9;
            if (usb_midi_handleTimeCodeQuarterFrame)
                  (*usb_midi_handleTimeCodeQuarterFrame)(n >> 16);
            return 1;
      }
      return 0;
}
#endif // F_CPU
#endif // MIDI_INTERFACE



//Software Serial implementation on Arduino
/*
SoftwareSerial.cpp (formerly NewSoftSerial.cpp) -
Multi-instance software serial library for Arduino/Wiring
-- Interrupt-driven receive and other improvements by ladyada
(http://ladyada.net)

-- Tuning, circular buffer, derivation from class Print/Stream,
multi-instance support, porting to 8MHz processors,
various optimizations, PROGMEM delay tables, inverse logic and
direct port writing by Mikal Hart (http://www.arduiniana.org)
-- Pin change interrupt macros by Paul Stoffregen (http://www.pjrc.com)
-- 20MHz processor support by Garrett Mace (http://www.macetech.com)
-- ATmega1280/2560 support by Brett Hagman
(http://www.roguerobotics.com/)
This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.
*/
// When set, _DEBUG co-opts pins 11 and 13 for debugging with an
// oscilloscope or logic analyzer. Beware: it also slightly modifies
// the bit times, so don't rely on it too much at high baud rates
#define _DEBUG 0
#define _DEBUG_PIN1 11
#define _DEBUG_PIN2 13
//
// Includes
//
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <Arduino.h>
#include <SoftwareSerial.h>
#include <util/delay_basic.h>
//
// Statics
//
SoftwareSerial *SoftwareSerial::active_object = 0;
char SoftwareSerial::_receive_buffer[_SS_MAX_RX_BUFF];
volatile uint8_t SoftwareSerial::_receive_buffer_tail = 0;
volatile uint8_t SoftwareSerial::_receive_buffer_head = 0;
//
// Debugging
//
// This function generates a brief pulse
// for debugging or measuring on an oscilloscope.
inline void DebugPulse(uint8_t pin, uint8_t count)
{
#if _DEBUG
volatile uint8_t *pport = portOutputRegister(digitalPinToPort(pin));
uint8_t val = *pport;
while (count--)
{
*pport = val | digitalPinToBitMask(pin);
*pport = val;

}
#endif
}
//
// Private methods
//
/* static */
inline void SoftwareSerial::tunedDelay(uint16_t delay) {
_delay_loop_2(delay);
}
// This function sets the current object as the "listening"
// one and returns true if it replaces another
bool SoftwareSerial::listen()
{
if (!_rx_delay_stopbit)
return false;
if (active_object != this)
{
if (active_object)
```

```
active_object->stopListening();
_buffer_overflow = false;
_receive_buffer_head = _receive_buffer_tail = 0;
active_object = this;
setRxIntMsk(true);
return true;
}
return false;
}
// Stop listening. Returns true if we were actually listening.
bool SoftwareSerial::stopListening()
{
if (active_object == this)
{
setRxIntMsk(false);
active_object = NULL;
return true;
}
return false;
}
//
// The receive routine called by the interrupt handler
//
void SoftwareSerial::recv()

{
#if GCC_VERSION < 40302
// Work-around for avr-gcc 4.3.0 OSX version bug
// Preserve the registers that the compiler misses
// (courtesy of Arduino forum user *etracer*)
asm volatile(
"push r18 \n\t"
"push r19 \n\t"
"push r20 \n\t"
"push r21 \n\t"
"push r22 \n\t"
"push r23 \n\t"
"push r26 \n\t"
"push r27 \n\t"
::);
#endif
uint8_t d = 0;
// If RX line is high, then we don't see any start bit
// so interrupt is probably not for us
if (_inverse_logic ? rx_pin_read() : !rx_pin_read())
{
// Disable further interrupts during reception, this prevents
// triggering another interrupt directly after we return, which can
// cause problems at higher baudrates.
setRxIntMsk(false);
// Wait approximately 1/2 of a bit width to "center" the sample
tunedDelay(_rx_delay_centering);
DebugPulse(_DEBUG_PIN2, 1);
```

```
// Read each of the 8 bits
for (uint8_t i=8; i > 0; --i)
{
tunedDelay(_rx_delay_intrabit);
d >>= 1;
DebugPulse(_DEBUG_PIN2, 1);
if (rx_pin_read())
d |= 0x80;
}
if (_inverse_logic)
d = ~d;
// if buffer full, set the overflow flag and return
uint8_t next = (_receive_buffer_tail + 1) % _SS_MAX_RX_BUFF;
if (next != _receive_buffer_head)
{
// save new data in buffer: tail points to where byte goes
_receive_buffer[_receive_buffer_tail] = d; // save new byte

_receive_buffer_tail = next;
}
else
{
DebugPulse(_DEBUG_PIN1, 1);
_buffer_overflow = true;
}
// skip the stop bit
tunedDelay(_rx_delay_stopbit);
DebugPulse(_DEBUG_PIN1, 1);
// Re-enable interrupts when we're sure to be inside the stop bit
setRxIntMsk(true);
}
#if GCC_VERSION < 40302
// Work-around for avr-gcc 4.3.0 OSX version bug
// Restore the registers that the compiler misses
asm volatile(
"pop r27 \n\t"
"pop r26 \n\t"
"pop r23 \n\t"
"pop r22 \n\t"
"pop r21 \n\t"
"pop r20 \n\t"
"pop r19 \n\t"
"pop r18 \n\t"
::);
#endif
}
uint8_t SoftwareSerial::rx_pin_read()
{
return *_receivePortRegister & _receiveBitMask;
}
//
// Interrupt handling
//
```

```
/* static */
inline void SoftwareSerial::handle_interrupt()
{
if (active_object)
{
active_object->recv();
}
}
#if defined(PCINT0_vect)

ISR(PCINT0_vect)
{
SoftwareSerial::handle_interrupt();
}
#endif
#if defined(PCINT1_vect)
ISR(PCINT1_vect, ISR_ALIASOF(PCINT0_vect));
#endif
#if defined(PCINT2_vect)
ISR(PCINT2_vect, ISR_ALIASOF(PCINT0_vect));
#endif
#if defined(PCINT3_vect)
ISR(PCINT3_vect, ISR_ALIASOF(PCINT0_vect));
#endif
//
// Constructor
//
SoftwareSerial::SoftwareSerial(uint8_t receivePin, uint8_t transmitPin,
bool
inverse_logic /* = false */) :
_rx_delay_centering(0),
_rx_delay_intrabit(0),
_rx_delay_stopbit(0),
_tx_delay(0),
_buffer_overflow(false),
_inverse_logic(inverse_logic)
{
setTX(transmitPin);
setRX(receivePin);
}
//
// Destructor
//
SoftwareSerial::~SoftwareSerial()
{
end();
}
void SoftwareSerial::setTX(uint8_t tx)
{
// First write, then set output. If we do this the other way around,
// the pin would be output low for a short while before switching to
// output hihg. Now, it is input with pullup for a short while, which
// is fine. With inverse logic, either order is fine.
```

```
digitalWrite(tx, _inverse_logic ? LOW : HIGH);
pinMode(tx, OUTPUT);
_transmitBitMask = digitalPinToBitMask(tx);
uint8_t port = digitalPinToPort(tx);

_transmitPortRegister = portOutputRegister(port);
}
void SoftwareSerial::setRX(uint8_t rx)
{
pinMode(rx, INPUT);
if (!_inverse_logic)
digitalWrite(rx, HIGH); // pullup for normal logic!
_receivePin = rx;
_receiveBitMask = digitalPinToBitMask(rx);
uint8_t port = digitalPinToPort(rx);
_receivePortRegister = portInputRegister(port);
}
uint16_t SoftwareSerial::subtract_cap(uint16_t num, uint16_t sub) {
if (num > sub)
return num - sub;
else
return 1;
}
//
// Public methods
//
void SoftwareSerial::begin(long speed)
{
_rx_delay_centering = _rx_delay_intrabit = _rx_delay_stopbit = _tx_delay
=
0;
// Precalculate the various delays, in number of 4-cycle delays
uint16_t bit_delay = (F_CPU / speed) / 4;
// 12 (gcc 4.8.2) or 13 (gcc 4.3.2) cycles from start bit to first bit,
// 15 (gcc 4.8.2) or 16 (gcc 4.3.2) cycles between bits,
// 12 (gcc 4.8.2) or 14 (gcc 4.3.2) cycles from last bit to stop bit
// These are all close enough to just use 15 cycles, since the inter-bit
// timings are the most critical (deviations stack 8 times)
_tx_delay = subtract_cap(bit_delay, 15 / 4);
// Only setup rx when we have a valid PCINT for this pin
if (digitalPinToPCICR(_receivePin)) {
#if GCC_VERSION > 40800
// Timings counted from gcc 4.8.2 output. This works up to 115200 on
// 16Mhz and 57600 on 8Mhz.
//
// When the start bit occurs, there are 3 or 4 cycles before the
// interrupt flag is set, 4 cycles before the PC is set to the right
// interrupt vector address and the old PC is pushed on the stack,
// and then 75 cycles of instructions (including the RJMP in the
// ISR vector table) until the first delay. After the delay, there
// are 17 more cycles until the pin value is read (excluding the

// delay in the loop).
// We want to have a total delay of 1.5 bit time. Inside the loop,
// we already wait for 1 bit time - 23 cycles, so here we wait for
// 0.5 bit time - (71 + 18 - 22) cycles.
_rx_delay_centering = subtract_cap(bit_delay / 2, (4 + 4 + 75 + 17 - 23)
/
4);
// There are 23 cycles in each loop iteration (excluding the delay)
_rx_delay_intrabit = subtract_cap(bit_delay, 23 / 4);
// There are 37 cycles from the last bit read to the start of
// stopbit delay and 11 cycles from the delay until the interrupt
// mask is enabled again (which _must_ happen during the stopbit).
// This delay aims at 3/4 of a bit time, meaning the end of the
// delay will be at 1/4th of the stopbit. This allows some extra
// time for ISR cleanup, which makes 115200 baud at 16Mhz work more
// reliably
_rx_delay_stopbit = subtract_cap(bit_delay * 3 / 4, (37 + 11) / 4);
#else // Timings counted from gcc 4.3.2 output
// Note that this code is a _lot_ slower, mostly due to bad register
// allocation choices of gcc. This works up to 57600 on 16Mhz and
// 38400 on 8Mhz.
_rx_delay_centering = subtract_cap(bit_delay / 2, (4 + 4 + 97 + 29 - 11)
/
4);
_rx_delay_intrabit = subtract_cap(bit_delay, 11 / 4);
_rx_delay_stopbit = subtract_cap(bit_delay * 3 / 4, (44 + 17) / 4);
#endif
// Enable the PCINT for the entire port here, but never disable it
// (others might also need it, so we disable the interrupt by using
// the per-pin PCMSK register).
*digitalPinToPCICR(_receivePin) |=
_BV(digitalPinToPCICRbit(_receivePin));
// Precalculate the pcint mask register and value, so setRxIntMask
// can be used inside the ISR without costing too much time.
_pcint_maskreg = digitalPinToPCMSK(_receivePin);
_pcint_maskvalue = _BV(digitalPinToPCMSKbit(_receivePin));
tunedDelay(_tx_delay); // if we were low this establishes the end
}
#if _DEBUG
pinMode(_DEBUG_PIN1, OUTPUT);
pinMode(_DEBUG_PIN2, OUTPUT);
#endif
listen();
}
void SoftwareSerial::setRxIntMsk(bool enable)
{
if (enable)

*_pcint_maskreg |= _pcint_maskvalue;
else
*_pcint_maskreg &= ~_pcint_maskvalue;
}
void SoftwareSerial::end()
{
```

```
stopListening();
}
// Read data from buffer
int SoftwareSerial::read()
{
if (!isListening())
return -1;
// Empty buffer?
if (_receive_buffer_head == _receive_buffer_tail)
return -1;
// Read from "head"
uint8_t d = _receive_buffer[_receive_buffer_head]; // grab next byte
_receive_buffer_head = (_receive_buffer_head + 1) % _SS_MAX_RX_BUFF;
return d;
}
int SoftwareSerial::available()
{
if (!isListening())
return 0;
return (_receive_buffer_tail + _SS_MAX_RX_BUFF - _receive_buffer_head) %
_SS_MAX_RX_BUFF;
}
size_t SoftwareSerial::write(uint8_t b)
{
if (_tx_delay == 0) {
setWriteError();
return 0;
}
// By declaring these as local variables, the compiler will put them
// in registers _before_ disabling interrupts and entering the
// critical timing sections below, which makes it a lot easier to
// verify the cycle timings
volatile uint8_t *reg = _transmitPortRegister;
uint8_t reg_mask = _transmitBitMask;
uint8_t inv_mask = ~_transmitBitMask;
uint8_t oldSREG = SREG;
bool inv = _inverse_logic;

uint16_t delay = _tx_delay;
if (inv)
b = ~b;
cli(); // turn off interrupts for a clean txmit
// Write the start bit
if (inv)
*reg |= reg_mask;
else
*reg &= inv_mask;
tunedDelay(delay);
// Write each of the 8 bits
for (uint8_t i = 8; i > 0; --i)
{
if (b & 1) // choose bit
*reg |= reg_mask; // send 1
else
*reg &= inv_mask; // send 0
tunedDelay(delay);
b >>= 1;
}
// restore pin to natural state
if (inv)
*reg &= inv_mask;
else
*reg |= reg_mask;
SREG = oldSREG; // turn interrupts back on
tunedDelay(_tx_delay);
return 1;
}
void SoftwareSerial::flush()
{
if (!isListening())
return;
uint8_t oldSREG = SREG;
cli();
_receive_buffer_head = _receive_buffer_tail = 0;
SREG = oldSREG;
}
int SoftwareSerial::peek()
{

if (!isListening())
return -1;
// Empty buffer?
if (_receive_buffer_head == _receive_buffer_tail)
return -1;
// Read from "head"
return _receive_buffer[_receive_buffer_head];
}


//Hardware Serial library used to implement UART communication on Tiva
Launchpad
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <inttypes.h>
#include "wiring_private.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_ints.h"
#include "inc/hw_uart.h"
#include "driverlib/gpio.h"
#include "driverlib/debug.h"
#include "driverlib/interrupt.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/pin_map.h"
```

```
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "HardwareSerial.h"
#define TX_BUFFER_EMPTY (txReadIndex == txWriteIndex)
#define TX_BUFFER_FULL (((txWriteIndex + 1) % txBufferSize) ==
txReadIndex)
#define RX_BUFFER_EMPTY (rxReadIndex == rxWriteIndex)
#define RX_BUFFER_FULL (((rxWriteIndex + 1) % rxBufferSize) ==
rxReadIndex)
#define UART_BASE g_ulUARTBase[uartModule]
static const unsigned long g_ulUARTBase[8] =
{
UART0_BASE, UART1_BASE, UART2_BASE, UART3_BASE,
UART4_BASE, UART5_BASE, UART6_BASE, UART7_BASE
};
//*************************************************************************
*****
*
//

// The list of possible interrupts for the console UART.
//
//*************************************************************************
*****
*
static const unsigned long g_ulUARTInt[8] =
{
INT_UART0, INT_UART1, INT_UART2, INT_UART3,
INT_UART4, INT_UART5, INT_UART6, INT_UART7
};
//*************************************************************************
*****
*
//
// The list of UART peripherals.
//
//*************************************************************************
*****
*
static const unsigned long g_ulUARTPeriph[8] =
{
SYSCTL_PERIPH_UART0, SYSCTL_PERIPH_UART1, SYSCTL_PERIPH_UART2,
SYSCTL_PERIPH_UART3, SYSCTL_PERIPH_UART4, SYSCTL_PERIPH_UART5,
SYSCTL_PERIPH_UART6, SYSCTL_PERIPH_UART7
};
//*************************************************************************
*****
*
//
// The list of UART GPIO configurations.
//
//*************************************************************************
*****
```

```
*
static const unsigned long g_ulUARTConfig[8][2] =
{
#if defined(PART_TM4C1233H6PM) || defined(PART_LM4F120H5QR)
{GPIO_PA0_U0RX, GPIO_PA1_U0TX}, {GPIO_PC4_U1RX, GPIO_PC5_U1TX},
{GPIO_PD6_U2RX, GPIO_PD7_U2TX}, {GPIO_PC6_U3RX, GPIO_PC7_U3TX},
{GPIO_PC4_U4RX, GPIO_PC5_U4TX}, {GPIO_PE4_U5RX, GPIO_PE5_U5TX},
{GPIO_PD4_U6RX, GPIO_PD5_U6TX}, {GPIO_PE0_U7RX, GPIO_PE1_U7TX}
#elif defined(PART_TM4C129XNCZAD)
{GPIO_PA0_U0RX, GPIO_PA1_U0TX}, {GPIO_PQ4_U1RX, GPIO_PQ5_U1TX},
{GPIO_PD4_U2RX, GPIO_PD5_U2TX}, {GPIO_PA4_U3RX, GPIO_PA5_U3TX},
{GPIO_PK0_U4RX, GPIO_PK1_U4TX}, {GPIO_PH6_U5RX, GPIO_PH7_U5TX},
{GPIO_PP0_U6RX, GPIO_PP1_U6TX}, {GPIO_PC4_U7RX, GPIO_PC5_U7TX}
#elif defined(PART_TM4C1294NCPDT)
{GPIO_PA0_U0RX, GPIO_PA1_U0TX}, {GPIO_PB0_U1RX, GPIO_PB1_U1TX},
{GPIO_PA6_U2RX, GPIO_PA7_U2TX}, {GPIO_PA4_U3RX, GPIO_PA5_U3TX},
{GPIO_PK0_U4RX, GPIO_PK1_U4TX}, {GPIO_PC6_U5RX, GPIO_PC7_U5TX},
{GPIO_PP0_U6RX, GPIO_PP1_U6TX}, {GPIO_PC4_U7RX, GPIO_PC5_U7TX}
#else
#error "**** No PART defined or unsupported PART ****"
#endif
};

static const unsigned long g_ulUARTPort[8] =
{
#if defined(PART_TM4C1233H6PM) || defined(PART_LM4F120H5QR)
GPIO_PORTA_BASE, GPIO_PORTC_BASE, GPIO_PORTD_BASE, GPIO_PORTC_BASE,
GPIO_PORTC_BASE, GPIO_PORTE_BASE, GPIO_PORTD_BASE, GPIO_PORTE_BASE
#elif defined(PART_TM4C129XNCZAD)
GPIO_PORTA_BASE, GPIO_PORTQ_BASE, GPIO_PORTD_BASE, GPIO_PORTA_BASE,
GPIO_PORTK_BASE, GPIO_PORTH_BASE, GPIO_PORTP_BASE, GPIO_PORTC_BASE
#elif defined(PART_TM4C1294NCPDT)
GPIO_PORTA_BASE, GPIO_PORTB_BASE, GPIO_PORTA_BASE, GPIO_PORTA_BASE,
GPIO_PORTK_BASE, GPIO_PORTC_BASE, GPIO_PORTP_BASE, GPIO_PORTC_BASE
#else
#error "**** No PART defined or unsupported PART ****"
#endif
};
static const unsigned long g_ulUARTPins[8] =
{
#if defined(PART_TM4C1233H6PM) || defined(PART_LM4F120H5QR)
GPIO_PIN_0 | GPIO_PIN_1, GPIO_PIN_4 | GPIO_PIN_5,
GPIO_PIN_6 | GPIO_PIN_7, GPIO_PIN_6 | GPIO_PIN_7,
GPIO_PIN_4 | GPIO_PIN_5, GPIO_PIN_4 | GPIO_PIN_5,
GPIO_PIN_4 | GPIO_PIN_5, GPIO_PIN_0 | GPIO_PIN_1
#elif defined(PART_TM4C129XNCZAD)
GPIO_PIN_0 | GPIO_PIN_1, GPIO_PIN_4 | GPIO_PIN_5,
GPIO_PIN_4 | GPIO_PIN_5, GPIO_PIN_4 | GPIO_PIN_5,
GPIO_PIN_0 | GPIO_PIN_1, GPIO_PIN_6 | GPIO_PIN_7,
GPIO_PIN_0 | GPIO_PIN_1, GPIO_PIN_4 | GPIO_PIN_5
#elif defined(PART_TM4C1294NCPDT)
GPIO_PIN_0 | GPIO_PIN_1, GPIO_PIN_0 | GPIO_PIN_1,
GPIO_PIN_6 | GPIO_PIN_7, GPIO_PIN_4 | GPIO_PIN_5,
GPIO_PIN_0 | GPIO_PIN_1, GPIO_PIN_6 | GPIO_PIN_7,
GPIO_PIN_0 | GPIO_PIN_1, GPIO_PIN_4 | GPIO_PIN_5
#else
#error "**** No PART defined or unsupported PART ****"
#endif
};
// Constructors
////////////////////////////////////////////////////////////////
HardwareSerial::HardwareSerial(void)
{
txWriteIndex = 0;
txReadIndex = 0;
rxWriteIndex = 0;
rxReadIndex = 0;
uartModule = 0;
txBuffer = (unsigned char *) 0xFFFFFFFF;
rxBuffer = (unsigned char *) 0xFFFFFFFF;
txBufferSize = SERIAL_BUFFER_SIZE;

rxBufferSize = SERIAL_BUFFER_SIZE;
}
HardwareSerial::HardwareSerial(unsigned long module)
{
txWriteIndex = 0;
txReadIndex = 0;
rxWriteIndex = 0;
rxReadIndex = 0;
uartModule = module;
txBuffer = (unsigned char *) 0xFFFFFFFF;
rxBuffer = (unsigned char *) 0xFFFFFFFF;
txBufferSize = SERIAL_BUFFER_SIZE;
rxBufferSize = SERIAL_BUFFER_SIZE;
}
// Private Methods
//////////////////////////////////////////////////////////////
void
HardwareSerial::flushAll(void)
{
// wait for transmission of outgoing data
while(!TX_BUFFER_EMPTY)
{
}
txReadIndex = 0;
txWriteIndex = 0;
//
// Flush the receive buffer.
//
rxReadIndex = 0;
rxWriteIndex = 0;
}
void
HardwareSerial::primeTransmit(unsigned long ulBase)
{
```

```
//
// Do we have any data to transmit?
//
if(!TX_BUFFER_EMPTY)
{
//
// Disable the UART interrupt. If we don't do this there is a race
// condition which can cause the read index to be corrupted.
//
ROM_IntDisable(g_ulUARTInt[uartModule]);
//
// Yes - take some characters out of the transmit buffer and feed
// them to the UART transmit FIFO.
//

while(!TX_BUFFER_EMPTY)
{
while(ROM_UARTSpaceAvail(ulBase) && !TX_BUFFER_EMPTY){
ROM_UARTCharPutNonBlocking(ulBase,
txBuffer[txReadIndex]);
txReadIndex = (txReadIndex + 1) % txBufferSize;
}
}
//
// Reenable the UART interrupt.
//
ROM_IntEnable(g_ulUARTInt[uartModule]);
}
}
// Public Methods
//////////////////////////////////////////////////////////////
void
HardwareSerial::begin(unsigned long baud)
{
baudRate = baud;
//
// Initialize the UART.
//
ROM_SysCtlPeripheralEnable(g_ulUARTPeriph[uartModule]);
//TODO:Add functionality for PinConfigure with variable uartModule
ROM_GPIOPinConfigure(g_ulUARTConfig[uartModule][0]);
ROM_GPIOPinConfigure(g_ulUARTConfig[uartModule][1]);
ROM_GPIOPinTypeUART(g_ulUARTPort[uartModule], g_ulUARTPins[uartModule]);
ROM_UARTConfigSetExpClk(UART_BASE, F_CPU, baudRate,
(UART_CONFIG_PAR_NONE | UART_CONFIG_STOP_ONE |
UART_CONFIG_WLEN_8));
//
// Set the UART to interrupt whenever the TX FIFO is almost empty or
// when any character is received.
//
ROM_UARTFIFOLevelSet(UART_BASE, UART_FIFO_TX1_8, UART_FIFO_RX1_8);
flushAll();
ROM_UARTIntDisable(UART_BASE, 0xFFFFFFFF);
```

```
ROM_UARTIntEnable(UART_BASE, UART_INT_RX | UART_INT_RT);
ROM_IntEnable(g_ulUARTInt[uartModule]);
//
// Enable the UART operation.
//
ROM_UARTEnable(UART_BASE);

// Allocate TX & RX buffers
if (txBuffer != (unsigned char *)0xFFFFFFFF) // Catch attempts to re-init
this Serial instance by freeing old buffer first
free(txBuffer);
if (rxBuffer != (unsigned char *)0xFFFFFFFF) // Catch attempts to re-init
this Serial instance by freeing old buffer first
free(rxBuffer);
txBuffer = (unsigned char *) malloc(txBufferSize);
rxBuffer = (unsigned char *) malloc(rxBufferSize);
SysCtlDelay(100);
}
void
HardwareSerial::setBufferSize(unsigned long txsize, unsigned long rxsize)
{
if (txsize > 0)
txBufferSize = txsize;
if (rxsize > 0)
rxBufferSize = rxsize;
}
void
HardwareSerial::setModule(unsigned long module)
{
ROM_UARTIntDisable(UART_BASE, UART_INT_RX | UART_INT_RT);
ROM_IntDisable(g_ulUARTInt[uartModule]);
uartModule = module;
begin(baudRate);
}
void
HardwareSerial::setPins(unsigned long pins)
{
if(pins == UART1_PORTB)
{
ROM_GPIOPinConfigure(GPIO_PB0_U1RX);
ROM_GPIOPinConfigure(GPIO_PB1_U1TX);
ROM_GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
}
else
{
//Default UART1 Pin Muxing
ROM_GPIOPinConfigure(g_ulUARTConfig[1][0]);
ROM_GPIOPinConfigure(g_ulUARTConfig[1][1]);
ROM_GPIOPinTypeUART(g_ulUARTPort[1], g_ulUARTPins[1]);
}
}
HardwareSerial::operator bool()
{
```

```
return true;
}
void HardwareSerial::end()
{
unsigned long ulInt = ROM_IntMasterDisable();
flushAll();
//
// If interrupts were enabled when we turned them off, turn them
// back on again.
//
if(!ulInt)
{
ROM_IntMasterEnable();
}
ROM_IntDisable(g_ulUARTInt[uartModule]);
ROM_UARTIntDisable(UART_BASE, UART_INT_RX | UART_INT_RT);
}
int HardwareSerial::available(void)
{
return((rxWriteIndex >= rxReadIndex) ?
(rxWriteIndex - rxReadIndex) : rxBufferSize - (rxReadIndex -
rxWriteIndex));
}
int HardwareSerial::peek(void)
{
unsigned char cChar = 0;
//
// Wait for a character to be received.
//
if(RX_BUFFER_EMPTY)
{
return -1;
//
// Block waiting for a character to be received (if the buffer is
// currently empty).
//
}
//
// Read a character from the buffer.
//
cChar = rxBuffer[rxReadIndex];
//
// Return the character to the caller.
//

return(cChar);
}
int HardwareSerial::read(void)
{
if(RX_BUFFER_EMPTY) {
return -1;
}
```

```
//
// Read a character from the buffer.
//
unsigned char cChar = rxBuffer[rxReadIndex];
rxReadIndex = ((rxReadIndex) + 1) % rxBufferSize;
return cChar;
}
void HardwareSerial::flush()
{
while(!TX_BUFFER_EMPTY);
}
size_t HardwareSerial::write(uint8_t c)
{
unsigned int numTransmit = 0;
//
// Check for valid arguments.
//
ASSERT(c != 0);
/*
//this is not necessary: https://github.com/energia/Energia/issues/225
//
// If the character to the UART is \n, then add a \r before it so that
// \n is translated to \n\r in the output.
//
// If the output buffer is full, there's nothing for it other than to
// wait for the interrupt handler to empty it a bit
if(c == '\n')
{
while (TX_BUFFER_FULL);
txBuffer[txWriteIndex] = '\r';
txWriteIndex = (txWriteIndex + 1) % txBufferSize;
numTransmit ++;
}
*/
//
// Send the character to the UART output.
//
while (TX_BUFFER_FULL);

txBuffer[txWriteIndex] = c;
txWriteIndex = (txWriteIndex + 1) % txBufferSize;
numTransmit ++;
//
// If we have anything in the buffer, make sure that the UART is set
// up to transmit it.
//
if(!TX_BUFFER_EMPTY)
{
primeTransmit(UART_BASE);
ROM_UARTIntEnable(UART_BASE, UART_INT_TX);
}
//
// Return the number of characters written.
```

```
//
return(numTransmit);
}
void HardwareSerial::UARTIntHandler(void){
unsigned long ulInts;
long lChar;
// Get and clear the current interrupt source(s)
//
ulInts = ROM_UARTIntStatus(UART_BASE, true);
ROM_UARTIntClear(UART_BASE, ulInts);
// Are we being interrupted because the TX FIFO has space available?
//
if(ulInts & UART_INT_TX)
{
//
// Move as many bytes as we can into the transmit FIFO.
//
primeTransmit(UART_BASE);
//
// If the output buffer is empty, turn off the transmit interrupt.
//
if(TX_BUFFER_EMPTY)
{
ROM_UARTIntDisable(UART_BASE, UART_INT_TX);
}
}
if(ulInts & (UART_INT_RX | UART_INT_RT))
{
while(ROM_UARTCharsAvail(UART_BASE))
{
//
// Read a character

//
lChar = ROM_UARTCharGetNonBlocking(UART_BASE);
//
// If there is space in the receive buffer, put the character
// there, otherwise throw it away.
//
uint8_t volatile full = RX_BUFFER_FULL;
if(full) break;
rxBuffer[rxWriteIndex] =
(unsigned char)(lChar & 0xFF);
rxWriteIndex = ((rxWriteIndex) + 1) % rxBufferSize;
//
// If we wrote anything to the transmit buffer, make sure it
actually
// gets transmitted.
//
}
primeTransmit(UART_BASE);
ROM_UARTIntEnable(UART_BASE, UART_INT_TX);
}
```

```
}
void
UARTIntHandler(void)
{
Serial.UARTIntHandler();
}
void
UARTIntHandler1(void)
{
Serial1.UARTIntHandler();
}
void
UARTIntHandler2(void)
{
Serial2.UARTIntHandler();
}
void
UARTIntHandler3(void)
{
Serial3.UARTIntHandler();
}
void
UARTIntHandler4(void)
{
Serial4.UARTIntHandler();

}
void
UARTIntHandler5(void)
{
Serial5.UARTIntHandler();
}
void
UARTIntHandler6(void)
{
Serial6.UARTIntHandler();
}
void
UARTIntHandler7(void)
{
Serial7.UARTIntHandler();
}
void serialEvent() __attribute__((weak));
void serialEvent() {}
void serialEvent1() __attribute__((weak));
void serialEvent1() {}
void serialEvent2() __attribute__((weak));
void serialEvent2() {}
void serialEvent3() __attribute__((weak));
void serialEvent3() {}
void serialEvent4() __attribute__((weak));
void serialEvent4() {}
void serialEvent5() __attribute__((weak));
void serialEvent5() {}
void serialEvent6() __attribute__((weak));
void serialEvent6() {}
void serialEvent7() __attribute__((weak));
void serialEvent7() {}
void serialEventRun(void)
{
if (Serial.available()) serialEvent();
if (Serial1.available()) serialEvent1();
if (Serial2.available()) serialEvent2();
if (Serial3.available()) serialEvent3();
if (Serial4.available()) serialEvent4();
if (Serial5.available()) serialEvent5();
if (Serial6.available()) serialEvent6();
if (Serial7.available()) serialEvent7();
}
HardwareSerial Serial;
HardwareSerial Serial1(1);
HardwareSerial Serial2(2);

HardwareSerial Serial3(3);
HardwareSerial Serial4(4);
HardwareSerial Serial5(5);
HardwareSerial Serial6(6);
HardwareSerial Serial7(7);


//Conceptinetics Library for DMX control
/*
Conceptinetics.cpp - DMX library for Arduino
Copyright (c) 2013 W.A. van der Meeren <danny@illogic.nl>. All right
reserved.
This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 3 of the License, or (at your option) any later version.
This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.
You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
*/
/*
This code has been tested using the following hardware:
- Arduino UNO R3 using a CTC-DRA-13-1 ISOLATED DMX-RDM SHIELD
- Arduino MEGA2560 R3 using a CTC-DRA-13-1 ISOLATED DMX-RDM SHIELD
*/
#include "pins_arduino.h"
#include "Conceptinetics.h"
#include <inttypes.h>
#include <stdlib.h>
```

```
#include <avr/interrupt.h>
#include <avr/io.h>
#include <util/delay.h>
#if defined (USE_DMX_SERIAL_0)
#if defined (USART__TXC_vect)
#define USART_TX USART__TXC_vect
#elif defined(USART_TX_vect)

#define USART_TX USART_TX_vect
#elif defined(USART0_TX_vect)
#define USART_TX USART0_TX_vect
#endif
#if defined (USART__RXC_vect)
#define USART_RX USART__RXC_vect
#elif defined(USART_RX_vect)
#define USART_RX USART_RX_vect
#elif defined(USART0_RX_vect)
#define USART_RX USART0_RX_vect
#endif
#if defined UDR
#define DMX_UDR UDR
#elif defined UDR0
#define DMX_UDR UDR0
#endif
#if defined(UBRRH) && defined(UBRRL)
#define DMX_UBRRH UBRRH
#define DMX_UBRRL UBRRL
#elif defined(UBRR0H) && defined(UBRR0L)
#define DMX_UBRRH UBRR0H
#define DMX_UBRRL UBRR0L
#endif
#if defined(UCSRA)
#define DMX_UCSRA UCSRA
#elif defined(UCSR0A)
#define DMX_UCSRA UCSR0A
#endif
#if defined(UCSRB)
#define DMX_UCSRB UCSRB
#elif defined (UCSR0B)
#define DMX_UCSRB UCSR0B
#endif
#if defined(TXEN) && defined(TXCIE)
#define DMX_TXEN TXEN
#define DMX_TXCIE TXCIE
#elif defined(TXEN0) && defined(TXCIE0)
#define DMX_TXEN TXEN0
#define DMX_TXCIE TXCIE0
#endif
#if defined(RXEN) && defined(RXCIE)
#define DMX_RXEN RXEN
#define DMX_RXCIE RXCIE
#elif defined(RXEN0) && defined(RXCIE0)
#define DMX_RXEN RXEN0
```

```
#define DMX_RXCIE RXCIE0
#endif
#if defined(FE)
#define DMX_FE FE
#elif defined(FE0)
#define DMX_FE FE0
#endif
#define RX_PIN 0
#define TX_PIN 1
#elif defined (USE_DMX_SERIAL_1)
#define USART_RX USART1_RX_vect
#define USART_TX USART1_TX_vect
#define DMX_UDR UDR1
#define DMX_UBRRH UBRR1H
#define DMX_UBRRL UBRR1L
#define DMX_UCSRA UCSR1A
#define DMX_UCSRB UCSR1B
#define DMX_TXEN TXEN1
#define DMX_TXCIE TXCIE1
#define DMX_RXEN RXEN1
#define DMX_RXCIE RXCIE1
#define DMX_FE FE1
#define RX_PIN 19
#define TX_PIN 18
#elif defined (USE_DMX_SERIAL_2)
#define USART_RX USART2_RX_vect
#define USART_TX USART2_TX_vect
#define DMX_UDR UDR2
#define DMX_UBRRH UBRR2H
#define DMX_UBRRL UBRR2L
#define DMX_UCSRA UCSR2A
#define DMX_UCSRB UCSR2B
#define DMX_TXEN TXEN2
#define DMX_TXCIE TXCIE2
#define DMX_RXEN RXEN2
#define DMX_RXCIE RXCIE2
#define DMX_FE FE2
#define RX_PIN 17
#define TX_PIN 16
#elif defined (USE_DMX_SERIAL_3)
#define USART_RX USART3_RX_vect
#define USART_TX USART3_TX_vect
#define DMX_UDR UDR3
#define DMX_UBRRH UBRR3H
#define DMX_UBRRL UBRR3L
#define DMX_UCSRA UCSR3A
#define DMX_UCSRB UCSR3B
#define DMX_TXEN TXEN3
#define DMX_TXCIE TXCIE3

#define DMX_RXEN RXEN3
#define DMX_RXCIE RXCIE3
```

```
#define DMX_FE FE3
#define RX_PIN 14
#define TX_PIN 15
#endif
#define LOWBYTE(v) ((uint8_t) (v))
#define HIGHBYTE(v) ((uint8_t) (((uint16_t) (v)) >> 8))
#define BSWAP_16(x) ( (uint8_t)((x) >> 8) | ((uint8_t)(x)) << 8 )
namespace isr
{
enum isrState
{
Idle,
Break,
DmxBreak,
DmxBreakManual,
DmxStartByte,
DmxRecordData,
DmxTransmitData,
RdmBreak,
RdmStartByte,
RdmRecordData,
RdmTransmitData,
RDMTransmitComplete,
};
enum isrMode
{
Disabled,
Receive,
DMXTransmit,
DMXTransmitManual, /* Manual break... */
RDMTransmit,
RDMTransmitNoInt, /* Setup uart but leave interrupt disabled */
};
};
DMX_Master *__dmx_master;
DMX_Slave *__dmx_slave;
RDM_Responder *__rdm_responder;
int8_t __re_pin; // R/W Pin on shield
isr::isrState __isr_txState; // TX ISR state
isr::isrState __isr_rxState; // RX ISR state

void SetISRMode ( isr::isrMode );
DMX_FrameBuffer::DMX_FrameBuffer ( uint16_t buffer_size )
{
m_refcount = (uint8_t*) malloc ( sizeof ( uint8_t ) );
if ( buffer_size >= DMX_MIN_FRAMESIZE && buffer_size <=
DMX_MAX_FRAMESIZE )
{
m_buffer = (uint8_t*) malloc ( buffer_size );
if ( m_buffer != NULL )
{
memset ( (void *)m_buffer, 0x0, buffer_size );
m_bufferSize = buffer_size;
}
else
m_buffer = 0x0;
}
else
m_bufferSize = 0x0;
*m_refcount++;
}
DMX_FrameBuffer::DMX_FrameBuffer ( DMX_FrameBuffer &buffer )
{
// Copy references and make sure the parent object does not dispose our
// buffer when deleted and we are still active
this->m_refcount = buffer.m_refcount;
(*this->m_refcount)++;
this->m_buffer = buffer.m_buffer;
this->m_bufferSize = buffer.m_bufferSize;
}
DMX_FrameBuffer::~DMX_FrameBuffer ( void )
{
// If we are the last object using the
// allocated buffer then free it together
// with the refcounter
if ( --(*m_refcount) == 0 )
{
if ( m_buffer )
free ( m_buffer );
free ( m_refcount );
}
}
uint16_t DMX_FrameBuffer::getBufferSize ( void )

{
return m_bufferSize;
}
uint8_t DMX_FrameBuffer::getSlotValue ( uint16_t index )
{
if (index < m_bufferSize)
return m_buffer[index];
else
return 0x0;
}
void DMX_FrameBuffer::setSlotValue ( uint16_t index, uint8_t value )
{
if ( index < m_bufferSize )
m_buffer[index] = value;
}
void DMX_FrameBuffer::setSlotRange ( uint16_t start, uint16_t end,
uint8_t
value )
{
if ( start < m_bufferSize && end < m_bufferSize && start < end )
memset ( (void *) &m_buffer[start], value, end-start );
}
```

```
void DMX_FrameBuffer::clear ( void )
{
memset ( (void *) m_buffer, 0x0, m_bufferSize );
}
uint8_t &DMX_FrameBuffer::operator[] ( uint16_t index )
{
return m_buffer[index];
}
DMX_Master::DMX_Master ( DMX_FrameBuffer &buffer, int readEnablePin )
: m_frameBuffer ( buffer ),
m_autoBreak ( 1 ) // Autobreak
generation is default on
{
setStartCode ( DMX_START_CODE );
__re_pin = readEnablePin;
pinMode ( __re_pin, OUTPUT );
::SetISRMode ( isr::Disabled );
}

DMX_Master::DMX_Master ( uint16_t maxChannel, int readEnablePin )
: m_frameBuffer ( maxChannel + DMX_STARTCODE_SIZE ),
m_autoBreak ( 1 ) // Autobreak
generation is default on
{
setStartCode ( DMX_START_CODE );
__re_pin = readEnablePin;
pinMode ( __re_pin, OUTPUT );
::SetISRMode ( isr::Disabled );
}
DMX_Master::~DMX_Master ( void )
{
disable (); // Stop sending
__dmx_master = NULL;
}
DMX_FrameBuffer &DMX_Master::getBuffer ( void )
{
return m_frameBuffer; // Return reference to
frame buffer
}
void DMX_Master::setStartCode ( uint8_t value )
{
m_frameBuffer[0] = value; // Set the first byte
in our frame buffer
}
void DMX_Master::setChannelValue ( uint16_t channel, uint8_t value )
{
if ( channel > 0 ) // Prevent overwriting
the start code
m_frameBuffer.setSlotValue ( channel, value );
}
void DMX_Master::setChannelRange ( uint16_t start, uint16_t end, uint8_t
value )
{
```

```
if ( start > 0 ) // Prevent overwriting
the start code
m_frameBuffer.setSlotRange ( start, end, value );
}
void DMX_Master::enable ( void )
{
__dmx_master = this;
if ( m_autoBreak )

::SetISRMode ( isr::DMXTransmit );
else
::SetISRMode ( isr::DMXTransmitManual );
}
void DMX_Master::disable ( void )
{
::SetISRMode ( isr::Disabled );
__dmx_master = NULL; // No active master
}
void DMX_Master::setAutoBreakMode ( void ) { m_autoBreak = 1; }
void DMX_Master::setManualBreakMode ( void ) { m_autoBreak = 0; }
uint8_t DMX_Master::autoBreakEnabled ( void ) { return m_autoBreak; }
uint8_t DMX_Master::waitingBreak ( void )
{
return ( __isr_txState == isr::DmxBreakManual );
}
void DMX_Master::breakAndContinue ( uint8_t breakLength_us )
{
// Only execute if we are the controlling master object
if ( __dmx_master == this && __isr_txState == isr::DmxBreakManual )
{
pinMode ( TX_PIN, OUTPUT );
digitalWrite ( TX_PIN, LOW ); // Begin BREAK
for (uint8_t bl=0; bl<breakLength_us; bl++)
_delay_us ( 1 );
// Turn TX Pin into Logic HIGH
digitalWrite ( TX_PIN, HIGH ); // END BREAK
__isr_txState = isr::DmxStartByte;
// TX Enable
DMX_UCSRB |= (1<<DMX_TXEN);
_delay_us ( 12 ); // MAB 12µSec
// TX Interupt enable
DMX_UCSRB |= (1<<DMX_TXCIE);
}
}
void (*DMX_Slave::event_onFrameReceived)(unsigned short
channelsReceived);
DMX_Slave::DMX_Slave ( DMX_FrameBuffer &buffer, int readEnablePin )

: DMX_FrameBuffer ( buffer ),
m_startAddress ( 1 )
{
__dmx_slave = this;
__re_pin = readEnablePin;
```

```
pinMode ( __re_pin, OUTPUT );
::SetISRMode ( isr::Disabled );
}
DMX_Slave::DMX_Slave ( uint16_t nrChannels, int readEnablePin )
: DMX_FrameBuffer ( nrChannels + 1 ),
m_startAddress ( 1 )
{
__dmx_slave = this;
__re_pin = readEnablePin;
pinMode ( __re_pin, OUTPUT );
::SetISRMode ( isr::Disabled );
}
DMX_Slave::~DMX_Slave ( void )
{
disable ();
__dmx_slave = NULL;
}
void DMX_Slave::enable ( void )
{
::SetISRMode ( isr::Receive );
}
void DMX_Slave::disable ( void )
{
::SetISRMode ( isr::Disabled );
}
DMX_FrameBuffer &DMX_Slave::getBuffer ( void )
{
return reinterpret_cast<DMX_FrameBuffer&>(*this);
}
uint8_t DMX_Slave::getChannelValue ( uint16_t channel )
{
return getSlotValue ( channel );
}
uint16_t DMX_Slave::getStartAddress ( void )
{
return m_startAddress;

}
void DMX_Slave::setStartAddress ( uint16_t addr )
{
m_startAddress = addr;
}
void DMX_Slave::onReceiveComplete ( void (*func)(unsigned short) )
{
event_onFrameReceived = func;
}
bool DMX_Slave::processIncoming ( uint8_t val, bool first )
{
static uint16_t idx;
bool rval = false;
if ( first )
{
// We could have received less channels then we
// expected.. but still is a complete frame
if (m_state == dmx::dmxData && event_onFrameReceived)
event_onFrameReceived (idx);
m_state = dmx::dmxStartByte;
}
switch ( m_state )
{
case dmx::dmxStartByte:
setSlotValue ( 0, val ); // Store start code
idx = m_startAddress;
m_state = dmx::dmxWaitStartAddress;
case dmx::dmxWaitStartAddress:
if ( --idx == 0 )
m_state = dmx::dmxData;
break;
case dmx::dmxData:
if ( idx++ < getBufferSize() )
setSlotValue ( idx, val );
else
{
m_state = dmx::dmxFrameReady;
// If a onFrameReceived callback is register...
if (event_onFrameReceived)
event_onFrameReceived (idx-2);
rval = true;

}
break;
}
return rval;
}
uint16_t RDM_FrameBuffer::getBufferSize ( void ) { return sizeof ( m_msg
); }
uint8_t RDM_FrameBuffer::getSlotValue ( uint16_t index )
{
if ( index < sizeof ( m_msg ) )
return m_msg.d[index];
else
return 0x0;
}
void RDM_FrameBuffer::setSlotValue ( uint16_t index, uint8_t value )
{
if ( index < sizeof ( m_msg ) )
m_msg.d[index] = value;
}
void RDM_FrameBuffer::clear ( void )
{
memset ( (void*)m_msg.d, 0x0, sizeof( m_msg ) );
m_state = rdm::rdmUnknown;
}
bool RDM_FrameBuffer::processIncoming ( uint8_t val, bool first )
{
static uint16_t idx;
```

```
bool rval = false;
if ( first )
{
m_state = rdm::rdmStartByte;
m_csRecv.checksum = (uint16_t) 0x0000;
idx = 0;
}
// Prevent buffer overflow for large messages
if (idx >= sizeof(m_msg))
return true;
switch ( m_state )
{
case rdm::rdmStartByte:
m_msg.startCode = val;
m_state = rdm::rdmSubStartCode;

break;
case rdm::rdmSubStartCode:
if ( val != 0x01 )
{
rval = true; // Stop processing data
break;
}
m_msg.subStartCode = val;
m_state = rdm::rdmMessageLength;
break;
case rdm::rdmMessageLength:
m_msg.msgLength = val;
m_state = rdm::rdmData;
m_csRecv.checksum = 0xcc + 0x01 + val; // set initial checksum
idx = 3; // buffer index for next
byte
break;
case rdm::rdmData:
m_msg.d[idx++] = val;
m_csRecv.checksum += val;
if ( idx >= m_msg.msgLength )
m_state = rdm::rdmChecksumHigh;
break;
case rdm::rdmChecksumHigh:
m_csRecv.csh = val;
m_state = rdm::rdmChecksumLow;
break;
case rdm::rdmChecksumLow:
m_csRecv.csl = val;
if ((m_csRecv.checksum % (uint16_t)0x10000) == m_csRecv.checksum)
{
m_state = rdm::rdmFrameReady;
// valid checksum ... start processing
processFrame ();
}
m_state = rdm::rdmUnknown;
rval = true;
break;
};
return rval;
}

bool RDM_FrameBuffer::fetchOutgoing ( volatile uint8_t *udr, bool first )
{
static uint16_t idx;
static uint16_t cs;
bool rval = false;
if ( first )
{
m_state = rdm::rdmData;
cs = 0x0;
idx = 0;
}
switch ( m_state )
{
case rdm::rdmData:
cs += m_msg.d[idx];
*udr = m_msg.d[idx++];
if ( idx >= m_msg.msgLength )
{
m_state = rdm::rdmChecksumHigh;
}
break;
case rdm::rdmChecksumHigh:
*udr = (cs >> 8);
m_state = rdm::rdmChecksumLow;
break;
case rdm::rdmChecksumLow:
*udr = (cs & 0xff);
m_state = rdm::rdmUnknown;
rval = true;
break;
}
return rval;
}
void (*RDM_Responder::event_onIdentifyDevice)(bool);
void (*RDM_Responder::event_onDeviceLabelChanged)(const char*, uint8_t);
void (*RDM_Responder::event_onDMXStartAddressChanged)(uint16_t);
void (*RDM_Responder::event_onDMXPersonalityChanged)(uint8_t);
//
// slave parameter is only used to ensure a slave object is present
before
// initializing the rdm responder class
//

RDM_Responder::RDM_Responder ( uint16_t m, uint8_t d1, uint8_t d2,
uint8_t d3, uint8_t d4, DMX_Slave &slave )
: RDM_FrameBuffer ( ),
m_Personalities (1), // Available personlities
m_Personality (1) // Default personality eq 1.
```

```
{
__rdm_responder = this;
m_devid.Initialize ( m, d1, d2, d3, d4 );
// Default software version id = 0x00000000
memset ( (void*)m_SoftwareVersionId, 0x0, 0x4 );
// Rdm responder is disabled by default
m_rdmStatus.enabled = false;
}
RDM_Responder::~RDM_Responder ( void )
{
__rdm_responder = NULL;
}
void RDM_Responder::onIdentifyDevice ( void (*func)(bool) )
{
event_onIdentifyDevice = func;
}
void RDM_Responder::onDeviceLabelChanged ( void (*func) (const char*,
uint8_t) )
{
event_onDeviceLabelChanged = func;
}
void RDM_Responder::onDMXStartAddressChanged ( void (*func) (uint16_t) )
{
event_onDMXStartAddressChanged = func;
}
void RDM_Responder::onDMXPersonalityChanged ( void (*func) (uint8_t) )
{
event_onDMXPersonalityChanged = func;
}
void RDM_Responder::setDeviceLabel ( const char *label, size_t len )
{
if ( len > RDM_MAX_DEVICELABEL_LENGTH )
len = RDM_MAX_DEVICELABEL_LENGTH;
memcpy ( (void *)m_deviceLabel, (void *)label, len );
}
#define UID_0 0x12
//ESTA device ID

#define UID_1 0x34
#define UID_2 0x56
#define UID_3 0x88
#define UID_4 0x00
#define UID_5 0x00
#define UID_CS (0xFF *6 +UID_0 +UID_1 +UID_2 +UID_3 +UID_4 +UID_5)
void RDM_Responder::repondDiscUniqueBranch ( void )
{
// Set shield to transmit mode (turn arround)
digitalWrite ( __re_pin, HIGH );
#if defined(UCSRB)
UCSRB = (1<<TXEN);
#elif defined(UCSR0B)
UCSR0B = (1<<TXEN0);
#endif
```

```
uint16_t cs = 0;
uint8_t response[24] =
{
0xfe, 0xfe, 0xfe, 0xfe, 0xfe, 0xfe, 0xfe, 0xaa, // byte 0-7
m_devid.m_id[0] | 0xaa, m_devid.m_id[0] | 0x55, // byte 8, 10 MSB
manufacturer
m_devid.m_id[1] | 0xaa, m_devid.m_id[1] | 0x55, // byte 10, 11 LSB
manufacturer
m_devid.m_id[2] | 0xaa, m_devid.m_id[2] | 0x55, // byte 12, 13 MSB
device
m_devid.m_id[3] | 0xaa, m_devid.m_id[3] | 0x55, // byte 14, 15 .
m_devid.m_id[4] | 0xaa, m_devid.m_id[4] | 0x55, // byte 16, 17 .
m_devid.m_id[5] | 0xaa, m_devid.m_id[5] | 0x55, // byte 18, 19 LSB
device
0x0, 0x0, 0x0, 0x0 // Checksum space
};
// Calculate checksum
for ( int i=8; i<20; i++ )
cs += (uint16_t)response [i];
// Write checksum into response
response [20] = (cs>>8) | 0xaa;
response [21] = (cs>>8) | 0x55;
response [22] = (cs&0xff) | 0xaa;
response [23] = (cs&0xff) | 0x55;
// Table 3-2 ANSI_E1-20-2010 <2ms
_delay_us ( MIN_RESPONDER_PACKET_SPACING_USEC );
for ( int i=0; i<24; i++ )

{
// Wait until data register is empty
#if defined (UCSR0A) && defined (UDRE0)
while((UCSR0A & (1 <<UDRE0)) == 0);
#elif defined (UCSRA) && defined (UDRE)
while((UCSRA & (1 <<UDRE)) == 0);
#endif
DMX_UDR = response[i];
}
// Wait until last byte is send
#if defined (UCSR0A) && defined (UDRE0)
while((UCSR0A & (1 <<UDRE0)) == 0);
#elif defined (UCSRA) && defined (UDRE)
while((UCSRA & (1 <<UDRE)) == 0);
#endif
// TODO:...
_delay_us (100);
// Restore ISR operations
::SetISRMode ( isr::Receive );
}
void RDM_Responder::populateDeviceInfo ( void )
{
RDM__DeviceInfoPD *pd = reinterpret_cast<RDM__DeviceInfoPD *>(m_msg.PD);
pd->protocolVersionMajor = 0x01;
pd->protocolVersionMinor = 0x00;
```

```
pd->deviceModelId = BSWAP_16(m_DeviceModelId);
pd->ProductCategory = BSWAP_16(m_ProductCategory);
memcpy ( (void*)pd->SoftwareVersionId, (void*)m_SoftwareVersionId, 4 );
pd->DMX512FootPrint = BSWAP_16(__dmx_slave->getBufferSize()-
1); // eq buffersize-startbyte
pd->DMX512CurrentPersonality = m_Personality;
pd->DMX512NumberPersonalities = m_Personalities;
pd->DMX512StartAddress = BSWAP_16(__dmx_slave-
>getStartAddress());
pd->SubDeviceCount = 0x0; // Sub devices are not supported by
this library
pd->SensorCount = 0x0; // Sensors are not yet supported
m_msg.PDL = sizeof (RDM__DeviceInfoPD);
}
const uint8_t ManufacturerLabel_P[] PROGMEM = "Conceptinetics";

void RDM_Responder::processFrame ( void )
{
// If packet is a general broadcast
if (
m_msg.dstUid.isBroadcast (m_devid.m_id) ||
m_devid == m_msg.dstUid
)
{
// Set default response type
m_msg.portId = rdm::ResponseTypeAck;
switch ( BSWAP_16(m_msg.PID) )
{
case rdm::DiscUniqueBranch:
// Check if we are inside the given unique branch...
if ( !m_rdmStatus.mute &&
reinterpret_cast<RDM_DiscUniqueBranchPD *>(m_msg.PD)-
>lbound < m_devid &&
reinterpret_cast<RDM_DiscUniqueBranchPD *>(m_msg.PD)-
>hbound > m_devid )
{
// Discovery messages are responded with data only and no
breaks
repondDiscUniqueBranch ();
}
break;
case rdm::DiscMute:
reinterpret_cast<RDM_DiscMuteUnMutePD *>(m_msg.PD)->ctrlField
= 0x0;
m_msg.PDL = sizeof ( RDM_DiscMuteUnMutePD );
m_rdmStatus.mute = true;
break;
case rdm::DiscUnMute:
reinterpret_cast<RDM_DiscMuteUnMutePD *>(m_msg.PD)->ctrlField
= 0x0;
m_msg.PDL = sizeof ( RDM_DiscMuteUnMutePD );
m_rdmStatus.mute = false;
break;
```

```
case rdm::SupportedParameters:
//
// Temporary solution... this will become dynamic
// in a later version...
//
m_msg.PD[0] = HIGHBYTE(rdm::DmxStartAddress); // MSB
m_msg.PD[1] = LOWBYTE (rdm::DmxStartAddress); // LSB
m_msg.PD[2] = HIGHBYTE(rdm::DmxPersonality);
m_msg.PD[3] = LOWBYTE (rdm::DmxPersonality);

m_msg.PD[4] = HIGHBYTE(rdm::ManufacturerLabel);
m_msg.PD[5] = LOWBYTE (rdm::ManufacturerLabel);
m_msg.PD[6] = HIGHBYTE(rdm::DeviceLabel);
m_msg.PD[7] = LOWBYTE (rdm::DeviceLabel);
m_msg.PDL = 0x6;
break;
// Only for manufacturer specific parameters
// case rdm::ParameterDescription:
// break;
case rdm::DeviceInfo:
if ( m_msg.CC == rdm::GetCommand )
populateDeviceInfo ();
break;
case rdm::DmxStartAddress:
if ( m_msg.CC == rdm::GetCommand )
{
m_msg.PD[0] = HIGHBYTE(__dmx_slave->getStartAddress ());
m_msg.PD[1] = LOWBYTE (__dmx_slave->getStartAddress ());
m_msg.PDL = 0x2;
}
else // if ( m_msg.CC == rdm::SetCommand )
{
__dmx_slave->setStartAddress ( (m_msg.PD[0] << 8) +
m_msg.PD[1] );
m_msg.PDL = 0x0;
if ( event_onDMXStartAddressChanged )
event_onDMXStartAddressChanged ( (m_msg.PD[0] << 8) +
m_msg.PD[1] );
}
break;
case rdm::DmxPersonality:
if ( m_msg.CC == rdm::GetCommand )
{
reinterpret_cast<RDM_DeviceGetPersonality_PD *>
(m_msg.PD)->DMX512CurrentPersonality = m_Personality;
reinterpret_cast<RDM_DeviceGetPersonality_PD *>
(m_msg.PD)->DMX512CurrentPersonality =
m_Personalities;
m_msg.PDL = sizeof (RDM_DeviceGetPersonality_PD);
}
else // if ( m_msg.CC == rdm::SetCommand )
{
m_Personality =
reinterpret_cast<RDM_DeviceSetPersonality_PD *>

(m_msg.PD)->DMX512Personality;
m_msg.PDL = 0x0;
if ( event_onDMXPersonalityChanged )
event_onDMXPersonalityChanged ( m_Personality );
}
break;
case rdm::IdentifyDevice:
if ( m_msg.CC == rdm::GetCommand )
{
m_msg.PD[0] = (uint8_t)(m_rdmStatus.ident ? 1 : 0);
m_msg.PDL = 0x1;
}
else if ( m_msg.CC == rdm::SetCommand )
{
// Look into first byte to see whether identification
// is turned on or off
m_rdmStatus.ident = m_msg.PD[0] ? true : false;
if ( event_onIdentifyDevice )
event_onIdentifyDevice ( m_rdmStatus.ident );
m_msg.PDL = 0x0;
}
break;
case rdm::ManufacturerLabel:
if ( m_msg.CC == rdm::GetCommand )
{
memcpy_P( (void*)m_msg.PD, ManufacturerLabel_P,
sizeof(ManufacturerLabel_P) );
m_msg.PDL = sizeof ( ManufacturerLabel_P );
}
break;
case rdm::DeviceLabel:
if ( m_msg.CC == rdm::GetCommand )
{
memcpy ( m_msg.PD, (void*) m_deviceLabel, 32 );
m_msg.PDL = 32;
}
else if ( m_msg.CC == rdm::SetCommand )
{
memset ( (void*) m_deviceLabel, ' ', 32 );
memcpy ( (void*) m_deviceLabel, m_msg.PD, (m_msg.PDL <
32 ? m_msg.PDL : 32) );
m_msg.PDL = 0;
// Notify application
if ( event_onDeviceLabelChanged )
event_onDeviceLabelChanged ( m_deviceLabel, 32 );
}

break;
default:
// Unknown parameter ID response
m_msg.portId = rdm::ResponseTypeNackReason;
m_msg.PD[0] = rdm::UnknownPid;
m_msg.PD[1] = 0x0;
m_msg.PDL = 0x2;
break;
};
}
//
// Only respond if this this message
// was destined to us only
if ( m_msg.dstUid == m_devid )
{
m_msg.startCode = RDM_START_CODE;
m_msg.subStartCode = 0x01;
m_msg.msgLength = RDM_HDR_LEN + m_msg.PDL;
m_msg.msgCount = 0;
/*
switch ( m_msg.msg.CC )
{
case rdm::DiscoveryCommand:
m_msg.msg.CC = rdm::DiscoveryCommandResponse;
break;
case rdm::GetCommand:
m_msg.msg.CC = rdm::GetCommandResponse;
break;
case rdm::SetCommand:
m_msg.msg.CC = rdm::SetCommandResponse;
break;
}
*/
/* Above replaced by next line */
m_msg.CC++;
m_msg.dstUid.copy ( m_msg.srcUid );
m_msg.srcUid.copy ( m_devid );
_delay_us ( MIN_RESPONDER_PACKET_SPACING_USEC );
SetISRMode ( isr::RDMTransmit );
}
}
void SetISRMode ( isr::isrMode mode )
{

uint8_t readEnable;
#if defined(USE_DMX_SERIAL_0)
#if defined(UCSRB) && defined (UCSRC)
UCSRC |= (1<<UMSEL)|(3<<UCSZ0)|(1<<USBS);
#elif defined(UCSR0B) && defined (UCSR0C)
UCSR0C |= (3<<UCSZ00)|(1<<USBS0);
#endif
#elif defined(USE_DMX_SERIAL_1)
UCSR1C |= (3<<UCSZ10)|(1<<USBS1);
#elif defined(USE_DMX_SERIAL_2)
UCSR2C |= (3<<UCSZ20)|(1<<USBS2);
#elif defined(USE_DMX_SERIAL_3)
UCSR3C |= (3<<UCSZ30)|(1<<USBS3);
```

```
#endif
switch ( mode )
{
case isr::Disabled:
#if defined(UCSRB)
UCSRB = 0x0;
#elif defined(UCSR0B)
UCSR0B = 0x0;
#endif
readEnable = LOW;
break;
case isr::Receive:
DMX_UBRRH = (unsigned char)(((F_CPU + DMX_BAUD_RATE * 8L) /
(DMX_BAUD_RATE * 16L) - 1)>>8);
DMX_UBRRL = (unsigned char) ((F_CPU + DMX_BAUD_RATE * 8L) /
(DMX_BAUD_RATE * 16L) - 1);
// Prepare before kicking off ISR
//DMX_UDR = 0x0;
__isr_rxState = isr::Idle;
readEnable = LOW;
DMX_UCSRB = (1<<DMX_RXCIE) | (1<<DMX_RXEN);
break;
case isr::DMXTransmit:
DMX_UDR = 0x0;
readEnable = HIGH;
__isr_txState = isr::DmxBreak;
DMX_UCSRB = (1<<DMX_TXEN) | (1<<DMX_TXCIE);
break;
case isr::DMXTransmitManual:
DMX_UBRRH = (unsigned char)(((F_CPU + DMX_BAUD_RATE * 8L) /
(DMX_BAUD_RATE * 16L) - 1)>>8);
DMX_UBRRL = (unsigned char) ((F_CPU + DMX_BAUD_RATE * 8L) /
(DMX_BAUD_RATE * 16L) - 1);

DMX_UDR = 0x0;
DMX_UCSRB = 0x0;
readEnable = HIGH;
__isr_txState = isr::DmxBreakManual;
break;
case isr::RDMTransmit:
DMX_UDR = 0x0;
readEnable = HIGH;
__isr_txState = isr::RdmBreak;
DMX_UCSRB = (1<<DMX_TXEN) | (1<<DMX_TXCIE);
break;
}
// If read enable pin is assigned
if (__re_pin > -1)
digitalWrite ( __re_pin, readEnable );
}
//
// TX UART (DMX Transmission ISR)
//
```

```
ISR (USART_TX)
{
static uint16_t current_slot;
switch ( __isr_txState )
{
case isr::DmxBreak:
DMX_UCSRA = 0x0;
DMX_UBRRH = (unsigned char)(((F_CPU + DMX_BREAK_RATE * 8L) /
(DMX_BREAK_RATE * 16L) - 1)>>8);
DMX_UBRRL = (unsigned char) ((F_CPU + DMX_BREAK_RATE * 8L) /
(DMX_BREAK_RATE * 16L) - 1);
DMX_UDR = 0x0;
if ( __isr_txState == isr::DmxBreak )
__isr_txState = isr::DmxStartByte;
break;
case isr::DmxStartByte:
DMX_UCSRA = 0x0;
DMX_UBRRH = (unsigned char)(((F_CPU + DMX_BAUD_RATE * 8L) /
(DMX_BAUD_RATE * 16L) - 1)>>8);
DMX_UBRRL = (unsigned char) ((F_CPU + DMX_BAUD_RATE * 8L) /
(DMX_BAUD_RATE * 16L) - 1);
current_slot = 0;
DMX_UDR = __dmx_master->getBuffer()[ current_slot++ ];
__isr_txState = isr::DmxTransmitData;
break;

case isr::DmxTransmitData:
// NOTE: we always send full frames of 513 bytes, this will bring us
// close to 40 frames / sec with no interslot delays
#ifdef DMX_IBG
_delay_us (DMX_IBG);
#endif
DMX_UDR = __dmx_master-
>getBuffer().getSlotValue( current_slot++ );
// Send 512 channels
if ( current_slot >= DMX_MAX_FRAMESIZE )
{
if ( __dmx_master->autoBreakEnabled () )
__isr_txState = isr::DmxBreak;
else
SetISRMode ( isr::DMXTransmitManual );
}
break;
case isr::RdmBreak:
DMX_UCSRA = 0x0;
DMX_UBRRH = (unsigned char)(((F_CPU + DMX_BREAK_RATE * 8L) /
(DMX_BREAK_RATE * 16L) - 1)>>8);
DMX_UBRRL = (unsigned char) ((F_CPU + DMX_BREAK_RATE * 8L) /
(DMX_BREAK_RATE * 16L) - 1);
DMX_UDR = 0x0;
__isr_txState = isr::RdmStartByte;
break;
case isr::RdmStartByte:
```

```
DMX_UCSRA = 0x0;
DMX_UBRRH = (unsigned char)(((F_CPU + DMX_BAUD_RATE * 8L) /
(DMX_BAUD_RATE * 16L) - 1)>>8);
DMX_UBRRL = (unsigned char) ((F_CPU + DMX_BAUD_RATE * 8L) /
(DMX_BAUD_RATE * 16L) - 1);
// Write start byte
__rdm_responder->fetchOutgoing ( &DMX_UDR, true );
__isr_txState = isr::RdmTransmitData;
break;
case isr::RdmTransmitData:
// Write rest of data
if ( __rdm_responder->fetchOutgoing ( &DMX_UDR ) )

__isr_txState = isr::RDMTransmitComplete;
break;
case isr::RDMTransmitComplete:
SetISRMode ( isr::Receive ); // Start waitin for new data
__isr_txState = isr::Idle; // No tx state
break;
}
}
//
// RX UART (DMX Reception ISR)
//
ISR (USART_RX)
{
uint8_t usart_state = DMX_UCSRA;
uint8_t usart_data = DMX_UDR;
//
// Check for framing error and reset if found
// A framing most likely* indicate a break in our ocasion
//
if ( usart_state & (1<<DMX_FE) )
{
DMX_UCSRA &= ~(1<<DMX_FE);
__isr_rxState = isr::Break;
return;
}
switch ( __isr_rxState )
{
case isr::Break:
if ( __dmx_slave && usart_data == DMX_START_CODE )
{
__dmx_slave->processIncoming ( usart_data, true );
__isr_rxState = isr::DmxRecordData;
}
else if ( __rdm_responder &&
usart_data == RDM_START_CODE &&
__rdm_responder->m_rdmStatus.enabled )
{
// __rdm_responder->clear ();
__rdm_responder->processIncoming ( usart_data, true );
__isr_rxState = isr::RdmRecordData;
}
else
{
__isr_rxState = isr::Idle;
}

break;
// Process DMX Data
case isr::DmxRecordData:
if ( __dmx_slave->processIncoming ( usart_data ) )
__isr_rxState = isr::Idle;
break;
// Process RDM Data
case isr::RdmRecordData:
if ( __rdm_responder->processIncoming ( usart_data ) )
__isr_rxState = isr::Idle;
break;
}
}

//Pitches.h file used for encoding notes to be used with Tone library
/*************************************************
* Public Constants
*************************************************/
#define NOTE_B0 31
#define NOTE_C1 33
#define NOTE_CS1 35
#define NOTE_D1 37
#define NOTE_DS1 39
#define NOTE_E1 41
#define NOTE_F1 44
#define NOTE_FS1 46
#define NOTE_G1 49
#define NOTE_GS1 52
#define NOTE_A1 55
#define NOTE_AS1 58
#define NOTE_B1 62
#define NOTE_C2 65
#define NOTE_CS2 69
#define NOTE_D2 73
#define NOTE_DS2 78
#define NOTE_E2 82
#define NOTE_F2 87
#define NOTE_FS2 93
#define NOTE_G2 98
#define NOTE_GS2 104
#define NOTE_A2 110
#define NOTE_AS2 117
#define NOTE_B2 123

#define NOTE_C3 131
#define NOTE_CS3 139
#define NOTE_D3 147
```

```
#define NOTE_DS3 156
#define NOTE_E3 165
#define NOTE_F3 175
#define NOTE_FS3 185
#define NOTE_G3 196
#define NOTE_GS3 208
#define NOTE_A3 220
#define NOTE_AS3 233
#define NOTE_B3 247
#define NOTE_C4 262
#define NOTE_CS4 277
#define NOTE_D4 294
#define NOTE_DS4 311
#define NOTE_E4 330
#define NOTE_F4 349
#define NOTE_FS4 370
#define NOTE_G4 392
#define NOTE_GS4 415
#define NOTE_A4 440
#define NOTE_AS4 466
#define NOTE_B4 494
#define NOTE_C5 523
#define NOTE_CS5 554
#define NOTE_D5 587
#define NOTE_DS5 622
#define NOTE_E5 659
#define NOTE_F5 698
#define NOTE_FS5 740
#define NOTE_G5 784
#define NOTE_GS5 831
#define NOTE_A5 880
#define NOTE_AS5 932
#define NOTE_B5 988
#define NOTE_C6 1047
#define NOTE_CS6 1109
#define NOTE_D6 1175
#define NOTE_DS6 1245
#define NOTE_E6 1319
#define NOTE_F6 1397
#define NOTE_FS6 1480
#define NOTE_G6 1568
#define NOTE_GS6 1661

#define NOTE_A6 1760
#define NOTE_AS6 1865
#define NOTE_B6 1976
#define NOTE_C7 2093
#define NOTE_CS7 2217
#define NOTE_D7 2349
#define NOTE_DS7 2489
#define NOTE_E7 2637
#define NOTE_F7 2794
#define NOTE_FS7 2960
```

```
#define NOTE_G7 3136
#define NOTE_GS7 3322
#define NOTE_A7 3520
#define NOTE_AS7 3729
#define NOTE_B7 3951
#define NOTE_C8 4186
#define NOTE_CS8 4435
#define NOTE_D8 4699
#define NOTE_DS8 4978
```

i. *APPENDIX 8.9: ONE PAGE VITA*

THE UNIVERSITY OF MICHIGAN-DEARBORN

Department of Electrical & Computer Engineering

1. Name and Academic Rank: VLADYSLAV SLYUSAR, Student

2. Degrees, with fields, institutions and dates:

BSE	Computer Engineering	(In Progress, expected graduation: May 2016)
BSE	Electrical Engineering	(In Progress, expected graduation: May 2016)
Minor	Computer Science	(In Progress, expected graduation: May 2016)

3. Number of years of post-secondary education: 4 years

4. Work Experience

Apr 2014 – Present   Ford Motor Company, Dearborn MI

May 2013 – Sep 2013   Valeo, Troy MI

Mar 2012 – Jun 2012   Vector CANtech, Novi MI

5. Honors & Awards

James B. Angell Scholar Award – March 2016, 7 Consecutive Semesters

Chancellor's Scholarship – August 2012, 4 years

6. Patents

Trap for Rodents – Ukraine Patent #64952

j. *APPENDIX 8.10: ONE PAGE VITA*

THE UNIVERSITY OF MICHIGAN-DEARBORN

Department of Electrical & Computer Engineering

1. Name and Academic Rank: Jay Lindland, Student

2. Degrees, with fields, institutions and dates:

BSE Electrical Engineering (In Progress, expected graduation: May 2016)

3. Number of years of post-secondary education: 5 years

4. Work Experience

May 2015 – August 2015 Terumo Cardiovascular, Ann Arbor MI