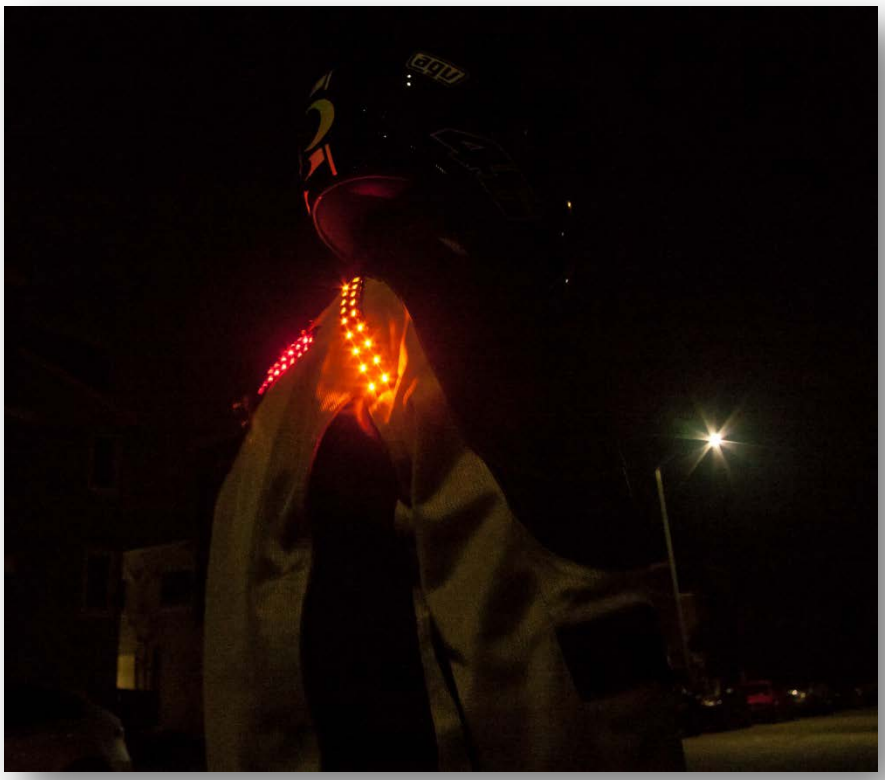
Wireless LED Motorcycle Jacket



Sean O'Brien
Cal Poly State University San Luis Obispo
Electrical Engineering
Advisor: Bridget Benson, Computer Engineering

June 10th 2015

Table of Contents

List of Tables and Figures

Abstract

In 2013 there were over 120,000 motorcycle fatalities in the U.S. alone, 73% of which were in the evening or during poor lighting conditions [1]. Motorcyclists are at a disadvantage when it comes to their visibility on the road due to their size and location of lighting indicators. Because of this, motorcyclists are 82% more likely to be in an accident than other automobiles [2]. One of the most common phrases among drivers involved in car vs motorcycle collisions is “I just didn’t see him” [3].

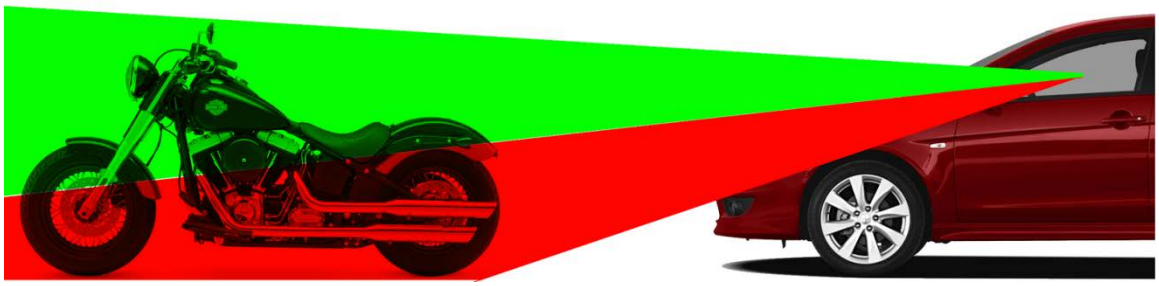


Figure 1: Visibility of turn and braking indicators from a sedan

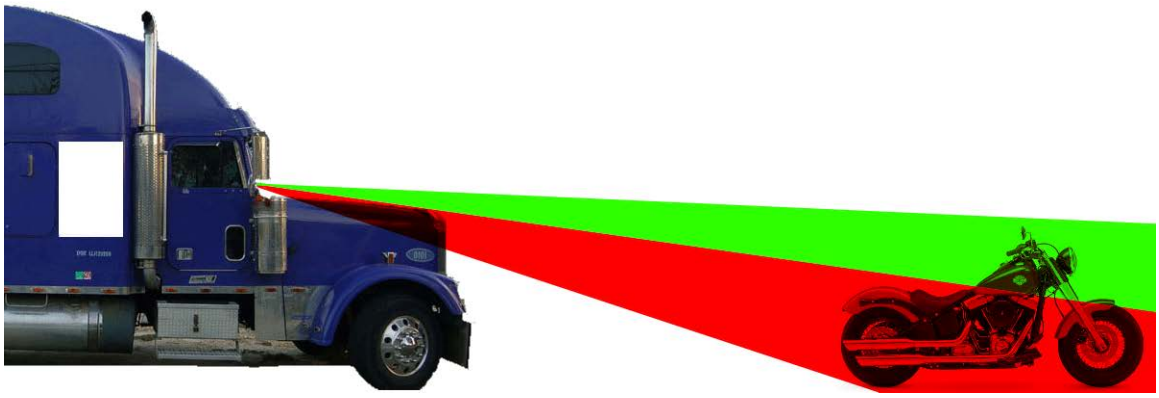


Figure 2: Visibility of turn and braking indicators from a semi-tuck

Many products exist on the market to assist with visibility of motorcyclists including high-visibility clothing and gear which make use of retro reflective materials. Jackets, vests, gloves, pants, reflective tape, and other items exist, but all of the safety equipment currently available is passive: they still rely on an outside source of light in order to work properly. While any increased visibility is good, passive equipment provides only a static level of increased visibility which does not alert other drivers when motorcyclists are slowing or turning, arguably some of the most critical scenarios. Motorcycle safety studies done by the US Department of Transportation show that wearing reflective clothing or high-visibility equipment can reduce the chance of an accident by 46% or higher [2]. How many accidents could be prevented if visibility equipment reacted dynamically?

Figure 3: Example of rear-end accident that could be prevented

Acknowledgements & Dedication

Special thanks to:

- My senior project advisor, Dr. Bridget Benson, for her help throughout the course of my projects development.

- Department chair, Dr. Dennis Derickson, for his help during the planning and documentation phases of my project, and for his support throughout my education at Cal Poly.

- Trust Automation, my internship location for 2014-2015, for allowing me use of their lab and fabrication facilities.

- The Electrical Engineering department for help with funding and support.

- My friends, Nolan Reker and Drew Troxell, for assistance selecting Bluetooth Low Energy hardware.

- My girlfriend, Maggie Moore, for her assistance with fabrics, stitching, and assembly.

My project is dedicated to motorcyclists on the road today; that projects like mine will help develop products to keep them a little safer on the road.

Introduction

Unlike passive visibility improvements, my solution increases the ability to see motorcycles at night as well as during the day, and also alerts drivers to when motorcyclists are slowing or turning. Many pieces of safety equipment are forgotten or simply ignored due to them being intrusive or difficult to use. In addition to increasing visibility and indicating the slowing or turning of the rider, the LED motorcycle jacket will integrate seamlessly into the habits of most riders. Once configured, the jacket will automatically connect, disconnect, and sleep based on the parameters of the microcontroller.

Aside from being a high-quality synthetic motorcycle safety jacket, this wearable tech would be a light-weight high-energy-density lithium ion battery pack. The battery would drive a wireless receiver and microcontroller responsible for duplicating the turn signal and braking indicators of the motorcycle being ridden. The replicated indicators of the bike would use flexible LED strips which would also be stitched into the back of the jacket.

The other side of the product is an integrated transmitter module for the motorcycle itself. The module would easily clip into the wiring harness of any existing bike. The module would be universal for motorcycles, mopeds, ATV's, and other motorized vehicles; in other words, it would support 6V or DC as well as 12V DC. The module would detect when a brake or turn signal was used, and wirelessly convey the digital information to the jacket. The wireless protocol would be over Bluetooth Low Energy (BLE) for lower power consumption. Encryption or at least a unique ID would be required so that multiple jackets do not accidentally link to the wrong bike and so that nefarious users could not take control of the indicators on another's jacket.

Market Alternatives

Let's compare some of the currently available market alternatives and look at their relative advantages and disadvantages.

Company	Product	Advantages	Disadvantages
impulse JACKETS	Jacket	High quality jacket Wireless LED's Rechargeable Armored	Cost Compatibility Nighttime brightness
HALO BELT	Belt	Cost High brightness LED's Rechargeable	No smart indication Flashing can be confusion on a bike
Street Lighting	Jacket	Cost LED's	No smart indication Not rechargeable
ILUMIN ADDIC	Jacket	Armored Cost	EL wire Not rechargeable No smart indication
Lighting de Soleil	Jacket or Backpack	Wireless Cost LED's Rechargeable Smart signaling	Non-solar rechargable
Helstar	Helmet lamp	Wireless LED's Smart signaling Universal	Wired No turn signals
Muza Moto	Bar-end and Bolt-on turn signals	LED's Style	Cost Visibility No brake indicator
RYDE BRIGHT	Jacket	LED's	Cost Visibility Availability

Table 1: Market alternatives for a visibility-increasing motorcycle gear and accessories

As we can see, no complete solution exists for the market segment this project aims to fill. The closest design is Impulse Jackets, which are not currently manufactured or available for purchase. Impulse also has a few downsides by being proprietary in nature instead of being designed for all riders and bikes.

Requirements

Marketing Requirements

1. Device must be easy to use
2. Device must be universal (all motorcycles)
3. Device must appeal to a wide range of riders (different style bikes)
4. Device must last a full day of use, with or without sunlight (24 hr)
5. Device must increase visibility during nighttime and daytime for motorcyclists
6. Device must cost only marginally more than a nice jacket alone. Build cost of $250 or less
7. Device must be light weight, small, unobtrusive

Engineering Specifications

1. Device must accept 6V DC or 12V DC nominal input voltage
   Applicable marketing requirements: 1, 2, 3
   Justification: Motorcycles, mopeds, and ATV's do not all use the same voltage levels or type

2. Device must have battery large enough to feed LED's and wireless circuitry without solar energy for 24h. 5AH battery
   Applicable marketing requirements: 4
   Justification: Based on power estimates, 5 AH should be long enough for "continuous" usage for 24 hours. Continuous usage is defined as normal usage while riding a motorcycle. Lithium batteries will be used based on their high energy density.

3. LED's must be above 200 lumens in brightness
   Applicable marketing requirements: 6
   Justification: In order to be daylight visible, light must be brighter than daylight. It is estimated that the lumens of an average car taillight is around 150 lumens.

4. Jacket will use high-density Lithium Iron batteries

Applicable marketing requirements: 8
Justification: Lithium Ion batteries are high energy density. Lithium Iron batteries have the highest energy density and are the safest. They will be small and unnoticeable in the bulk of the jacket.

5. Transmitter must be less than 2”x3” in size
   Applicable marketing requirements: 8
   Justification: PCB must fit under a seat or in the existing wiring harness. 2” x 3” is about the size of a credit card, and should have no problem slipping into most motorcycles wiring harnesses. 2x3 is also the footprint of many development platforms.

Hardware Design

This section describes the hardware components and their implementation into the Transmitter and Receiver modules.

Functional Hardware Block Diagrams

Overall

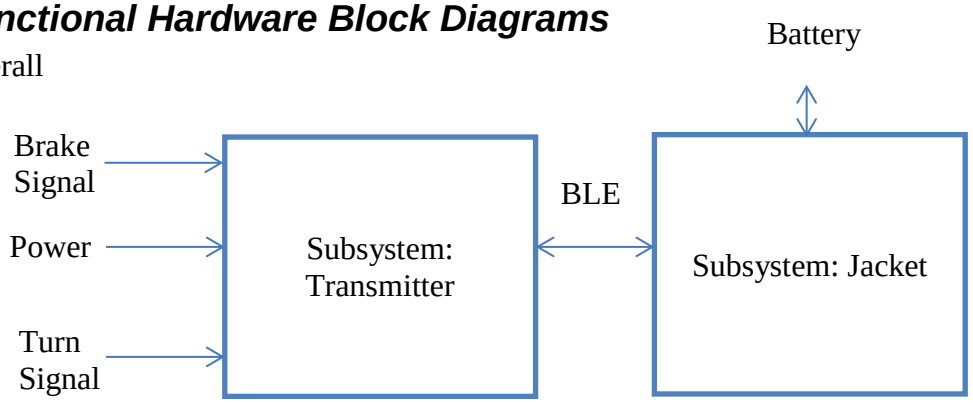


Figure 4: Example of rear-end accident that could be prevented

	Name	Description
Inputs	**Power**	Power to the transmitter will be supplied from the bike itself. The power input should be universal, meaning it will accept 6-12V DC.
	Turn Signals	The existing wiring of the bike will be used to detect when the turn signal indicators on turned on.
	Brake Signal	Similarly, the existing brake light wiring will be used to detect when the braking indicator is lit.
I/O	**Bluetooth Low Energy (BLE)**	A wireless Bluetooth Low Energy connection will be used between the jacket and the bike for triggering the signals and also detecting when the rider has left the bike.
	Battery	The battery, a pack of lithium-ion 18650 cells, will hold enough charge for all-day usage without being charged. The batteries will use an external charger for simplicity.
Outputs	**LED Indicators**	The high-brightness LED's will be driven according to the input signals read by the transmitter. The LED's should be visible during the day.
Description	The figure above shows an overall system diagram comprised of the two independent subsystems. The transmitter subsystem will be mounted on the motorcycle and will be transmitting the turn and brake indication signals.	

Table 2: Breakdown of top level block diagram

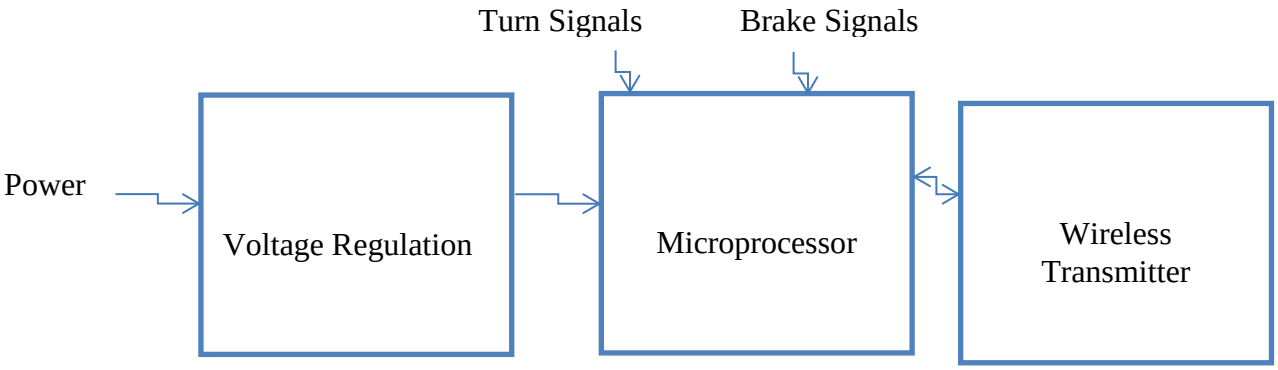


Figure 5: Transmitter Block Diagram

	Name	Description
Inputs	**Power**	Power to the transmitter will be supplied from the bike itself. The power input should be universal, meaning it will accept 6-12V DC, so the regulator will have to be capable of regulating these voltages down to those required by the microcontroller (3.3v).
	Turn Signals	The existing wiring of the bike will be used to detect when the turn signal indicators on turned on. A level converter will be used in order to input to a digital I/O pin.
	Brake Signals	Similarly, the existing brake light wiring will be used to detect when the braking indicator is lit.
I/O	**Busses**	Busses between the microprocessor and the wireless transmitter are internal and included in the SOC.
Outputs	**Bluetooth Low Energy (BLE)**	BLE will be used because it is low power and easily integrates with most small microcontrollers. It can facilitate communications between the transmitter and the jacket bi-directionally.
Description	The figure above looks closely at the first subsystem, the transmitter module. This subsystem will be comprised of a voltage regulation block, a microprocessor block, and a transmitter block. The voltage regulator works to control the voltages running the microprocessor. The microprocessor controls the wireless transmitter, and determines what signals to send to the jacket based on its input.	

Table 3: Breakdown of transmitter block diagram

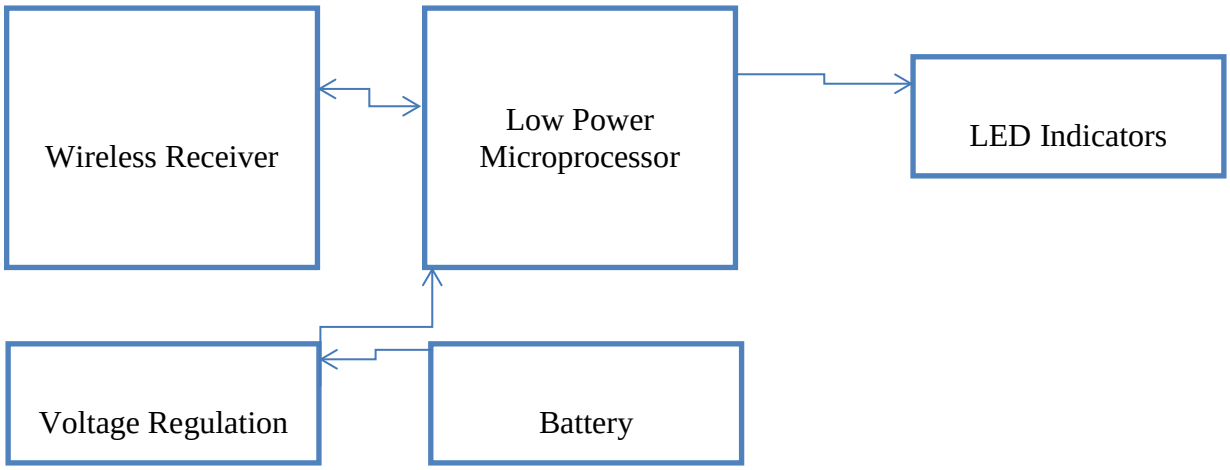


Figure 6: Jacket Block Diagram

	Name	Description
Inputs	**Bluetooth Low Energy (BLE)**	BLE will be used because it is low power and easily integrates with most small microcontrollers. It can facilitate communications between the transmitter and the jacket bi-directionally.
	Power Input	For charging manually, the power input will be 12V DC which can be plugged into a bike accessory port or a wall outlet (with provided transformer / 12V regulator).
I/O	**Busses**	Busses between the microprocessor and the wireless transmitter are internal and included in the SOC.
	Battery	The battery, a lithium-ion pouch cell, will hold enough charge for all-day usage without being charged. The batteries can be charged from the solar panels as well as a wall plug or accessory port. The battery will output to a voltage regulator, similar to the regulator in the transmitter block, to power the LED's and microcontroller circuity. On the input side, the battery will be charged from a charge controller.
Outputs	**LED Indicators**	The high-brightness LED's will be driven according to the input signals read by the transmitter. The LED's should be visible during the day. The LED's will require a driver circuit connected to the microprocessor.
Description	This figure the most involved part of the project. The jacket will contain similar hardware as the transmitter as far as the microprocessor, wireless transceiver, and voltage regulation circuitry. One notable difference is that since this circuit runs off of a battery, power usage is critical. A low power microprocessor and voltage regulator will be employed to achieve this.	

Table 4: Breakdown of jacket block diagram

Design Considerations

Power

The jacket system must draw as little power as possible when it is idle (i.e. not turning on the brake or turn signal indicators). To achieve this, the low-power sleep states of the TI CC2541 will be utilized. The processor is so efficient that it can actually go to sleep while maintaining a GPIO state. For this project, the processors sleep state will kick in not only when the jacket is idle for long periods of time, but the sleep state will also kick in between turn-signal flashes. The only way to drive such a system is with interrupts, as polling requires the processor to be awake.

The CC2541 chipset inside the BLE113 module does not support pin-change interrupts. As such, the software enables rising-edge interrupts on P1_x pins and falling-edge interrupts on P0_x pins. By electrically tying P0_x pins to P1_x pins, and by using the same interrupt service routine (ISR) in software, we can emulate pin-change interrupt behavior. This configures all 8 available pins to work as if they were driven by change interrupts:

P0_0 & P1_0, P0_4 & P1_4 through P0_7 & P1_7 are unused

Turn Signal L = P0_1 & P1_1
Turn Signal R = P0_2 & P1_2
Brake Indicator = P0_3 & P1_3

This method does not make very good use of the available GPIO pins on the CC2541 chip, but does accomplish our task of keeping the processor in a low-power state.

Four series Lithium cells will be selected for a nominal pack voltage of around 14.6 volts. This will allow a 12V Series LED string to operate at its full brightness without modification.

Safety

It is important to identify the most dangerous part in any system design. In this design, the lithium ion batteries are of the highest concern. Often, when lithium ion batteries are pierced, shorted, or otherwise damaged, catastrophic failure can occur. If a battery pack were to catch fire while in the pocket of a rider, serious injury or worse could occur. This safety concern drives a careful battery selection process. Several Lithium battery chemistries were considered.

Chemistry	Lead Acid	NiCd	NiMH	Rechargeable Alkaline	LiTiO	LiFePO	LiCoO	LiMnO	LiNCM
Conversational description	Lead Acid	NiCad	Nickel Metal Hydride	Reusable alkaline	Lithium Titanate	Lithium Iron	Lithium Cobalt	Lithium Manganese	Lithium Nickel Cobalt Manganese
Nominal Cell Voltage (V)	2.0	1.2	1.25	1.5	2.2	3.2	3.7	3.8	3.7
Specific Energy Density(Wh/kg)	40	60	90	80	48	115	174	135	150
Energy Density(Wh/l)	94		222		106	230	370		735
Discharge Rate (C-rate)	5	20	5	0.5	20	5	5	10	10
Charge Rate (C-rate)	0.08	1.00	0.33	0.40	10	0.50	0.50		
Discharge temperature (degC)	-20 - +60	-40 - +60	-20 - +60	0 - +65	-20 - +55	-30 - +55	-30 - +55	-20 - +60	-20 - +60
Charge Temperature (degC)					-40 - +55	0 - +40	0 - +40	0 - +40	0 - +40
Maintenance	3-6 mo	45 days	45 days	none	none	none	none	none	none
Cycle Life at 25degC, 1C load	250	150	400	50	100,000	3,000	1,000	1,000	2,000
Cathode stability	good	good	good	good	good	good	poor	moderate	moderate
Self discharge per month (25C)	5%	20%	30%	<1%	~ 0.7%	~ 0.7%	~ 0.7%	~ 0.7%	~ 0.7%

Table 5: Battery Chemistry Comparison

The table above eliminates almost all chemistries due to their poor lifespan, maintenance, or instability. Lithium Titanate (LTO) and Lithium Iron Phosphate (LiFe) are the two most likely candidates, with LTO cells having a few advantages over Lithium Iron. Unfortunately, Lithium Titanate cells are not manufactured yet in a form-factor suited for small personal electronics, and so Lithium Iron is the obvious solution.

Hardware Components

- 2 BLE 113 (TI CC2541 SOC)
- 2 LT1763 3.3V LDO Regulators
- 5 25v 350uF capacitors
- 1 74HC04 Hex inverter
- 3 NTE3041 OptoIsolators
- 3 470 Ω ½ W Resistors
- 3 10 kΩ ¼ W Resistors
- 3 UV Resistant Weatherproof Red LED Strings (SMD 5050 LED's)
- 4 UV Resistant Weatherproof Yellow LED Strings (SMD 5050 LED's)
- 1 DB9 Male
- 1 DB9 Female
- 2 ft CAT5 Cable
- 0.5 yd Polyester / nylon mesh
- 0.5 yd Polyester / nylon backing

- Thread
- Wire
- 1 Receiver PCB (breadboard for development)
- 1 Transmitter PCB (breadboard for development)
- 2 2 position Phoenix screw terminal block
- 2 6 position Phoenix screw terminal block
- 4 Lithium Iron Phosphate 18650 Cells
- 1 4 series 18650 holder

Schematics

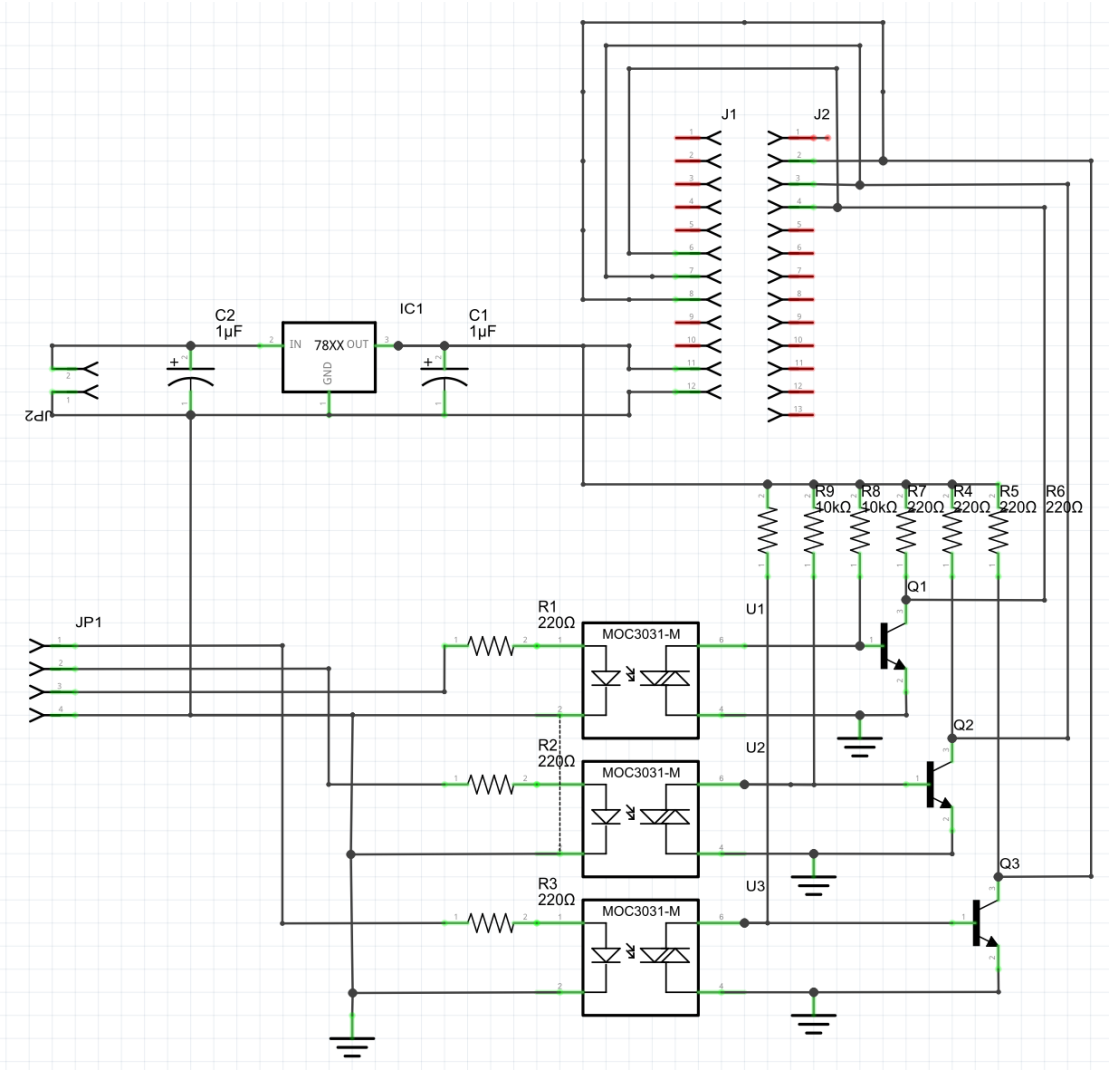


Figure 7: Transmitter Schematic

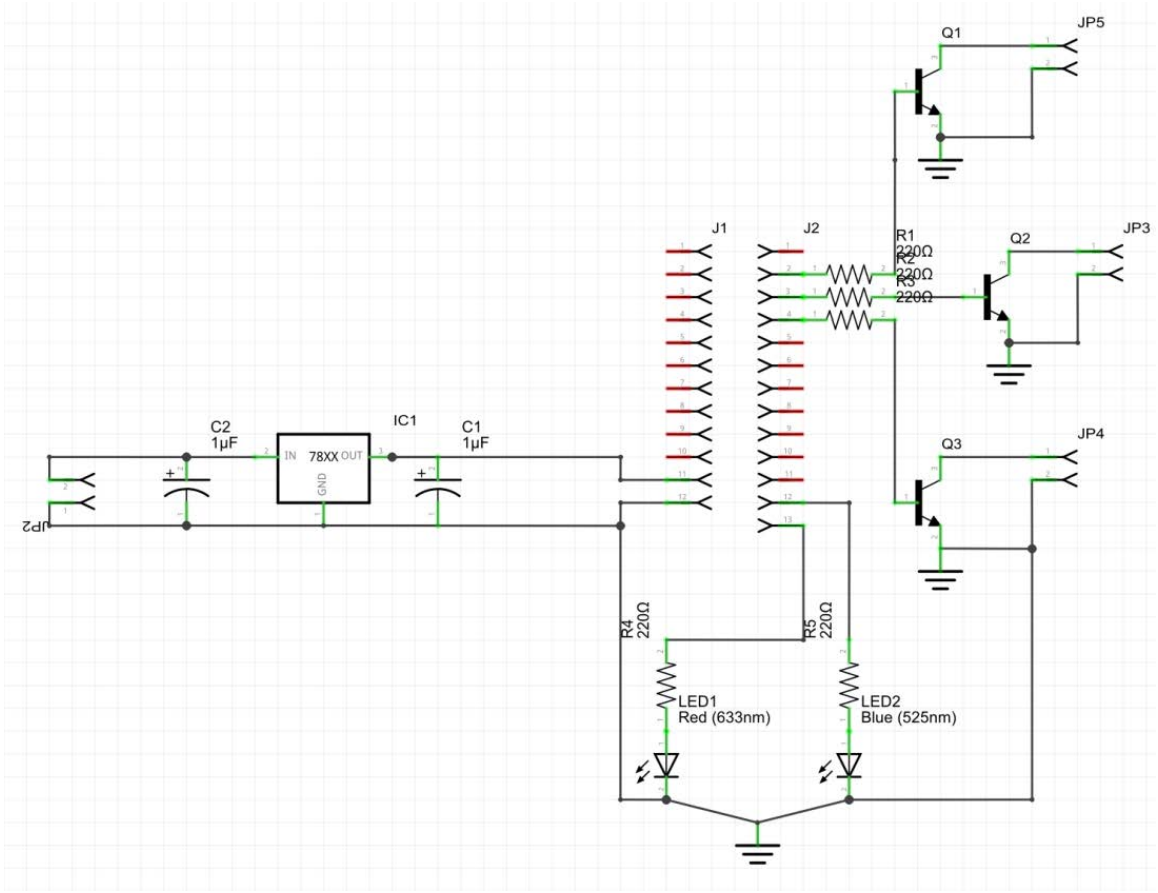


Figure 8: Receiver Schematic

Assembly and PCB Design

The BLE113 chip is incredibly small, measuring only 9.15 x 15.75 x 2.1mm. For prototyping purposes, a breakout board was required. This allows easy connecting of peripherals, power supplies, and other hardware without designing my own breakout board. The breakout board also allows for connecting of TI's CC USB debugger for programming and debugging the chip.

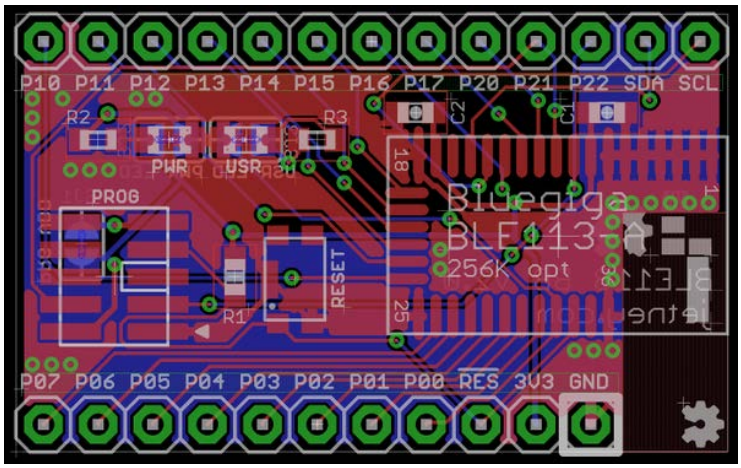


Figure 9: BLE113 Breakout PCB Provided by Jeff Rowberg

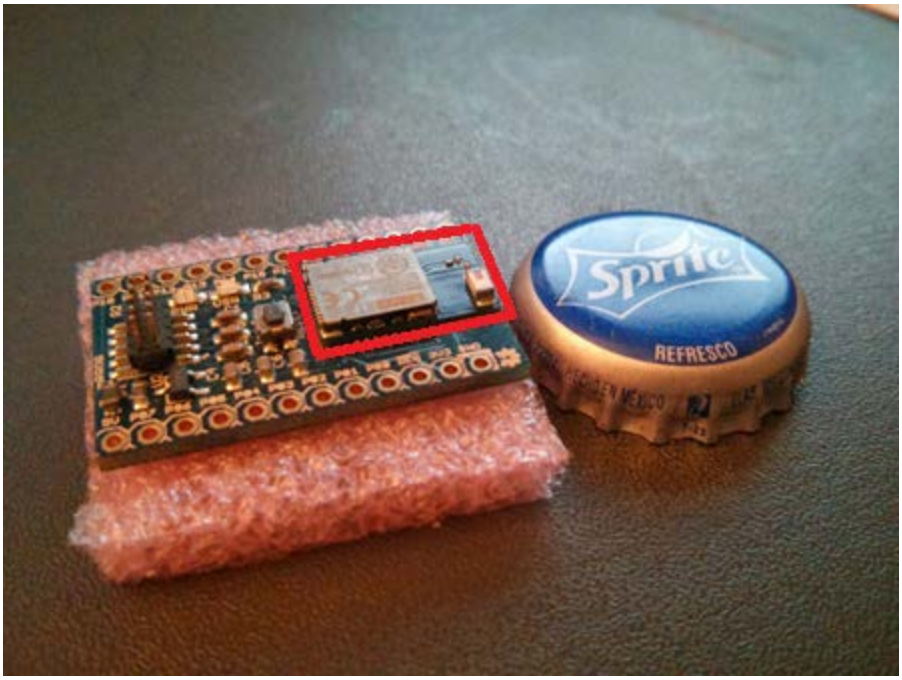


Figure 10: BLE113 Breakout PCB Assembled (BLE113 in red)

There are two separate PCB's: one for the receiver and one for the transmitter. Both PCB's were designed to be small and unobtrusive for enclosure in the jacket and on the motorcycle.

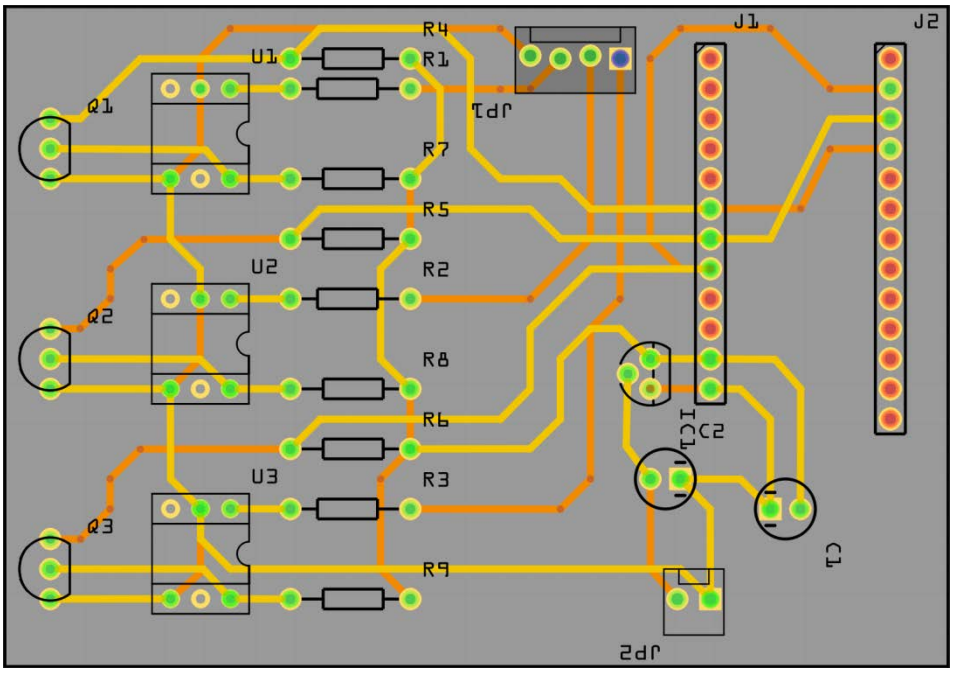


Figure 11: Transmitter PCB

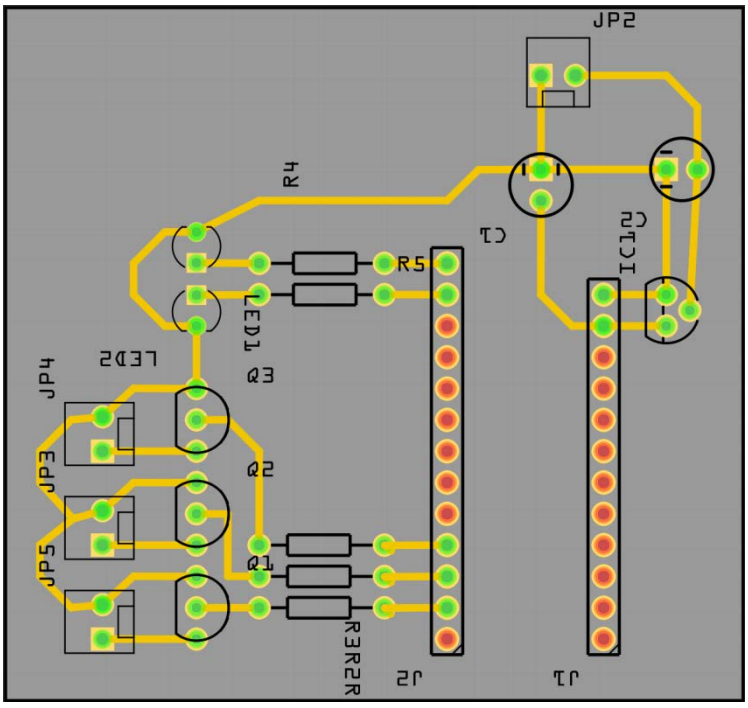


Figure 12: Receiver PCB

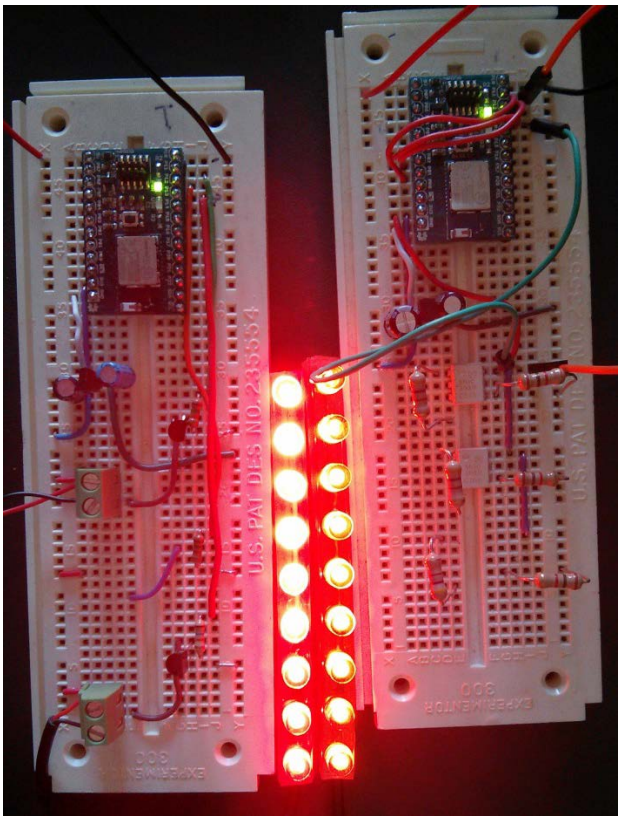


Figure 13: Assembled Functional Prototype

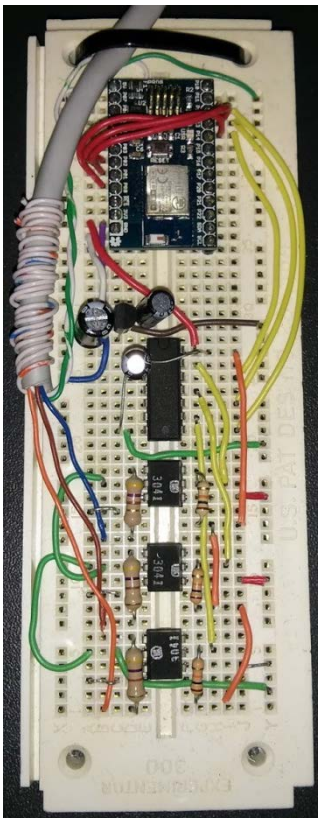


Figure 14: Final Transmitter Breadboard

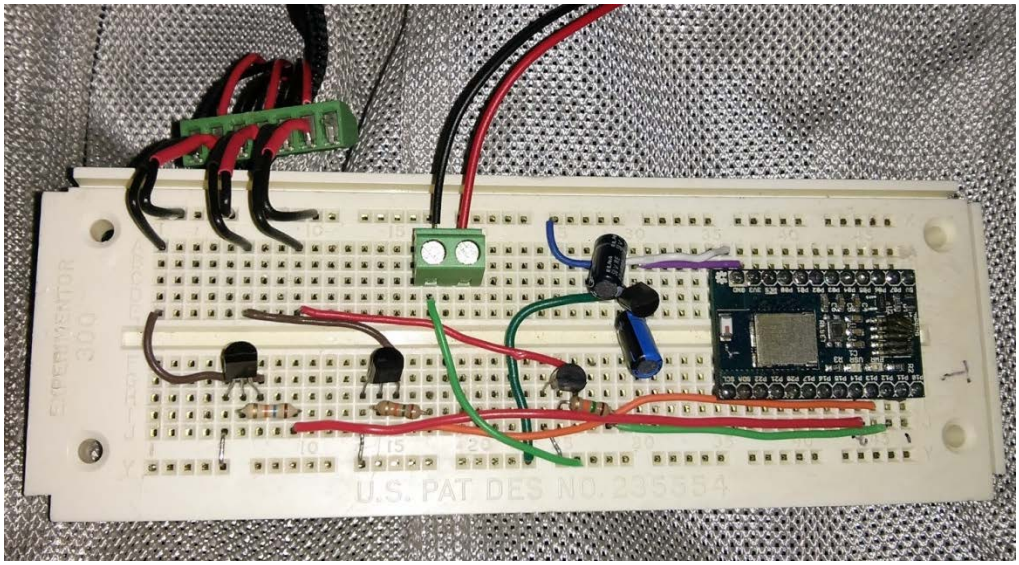


Figure 15: Final Receiver Breadboard

Jacket Construction

For this stage of production, a budget motorcycle jacket was purchased specifically for this project. The strips, wiring, and circuitry needed to be added in a safe and permanent fashion. Each set of LED strips is contained within a nylon and polyester sleeve. The sleeve maintains flexibility of the strips while keeping them firmly affixed to the jacket.

Figure 16: LED Sleeve Construction

Figure 17 demonstrates the final attachment of all 7 LED strips (3 brake, 4 turn signal).

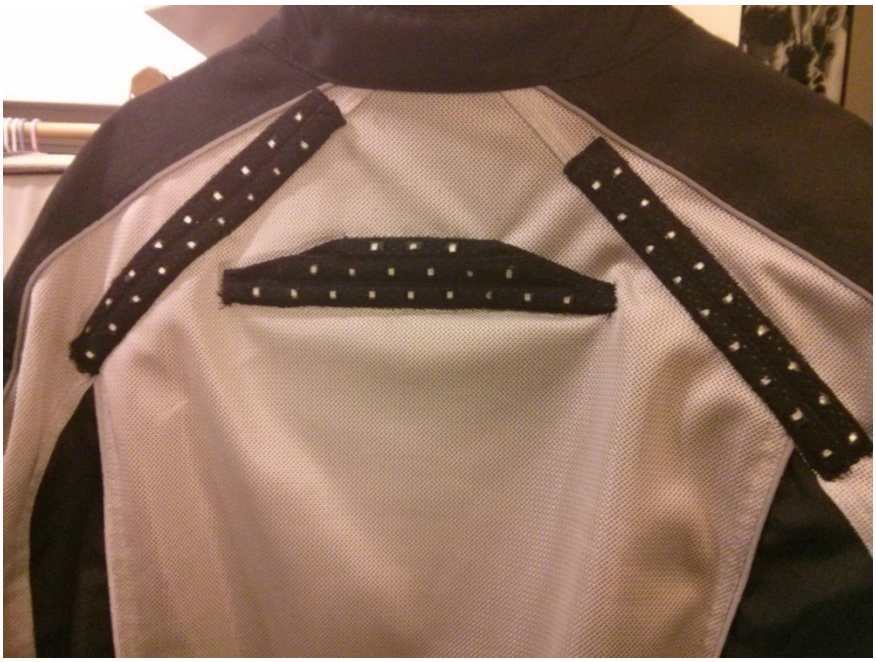


Figure 17: LED's Attached to Jacket

LED's are internally connected in series groups of 3, and connected in parallel. Any single LED failure will not bring down the entire array. After assembly, the LED's were tested again for functionality (figure 18).

Figure 18: LED's Operational

Software Design

As noted in the marketing requirements, the jacket and transmitter must be easy to use for any rider. This drove a very simple software design which boots and automatically connects, regardless of which side is powered on first. If connection failure occurs, the default state is to attempt re-connection. The sleep-state of the processor is inherently handled by the processor and does not need to be asserted manually.

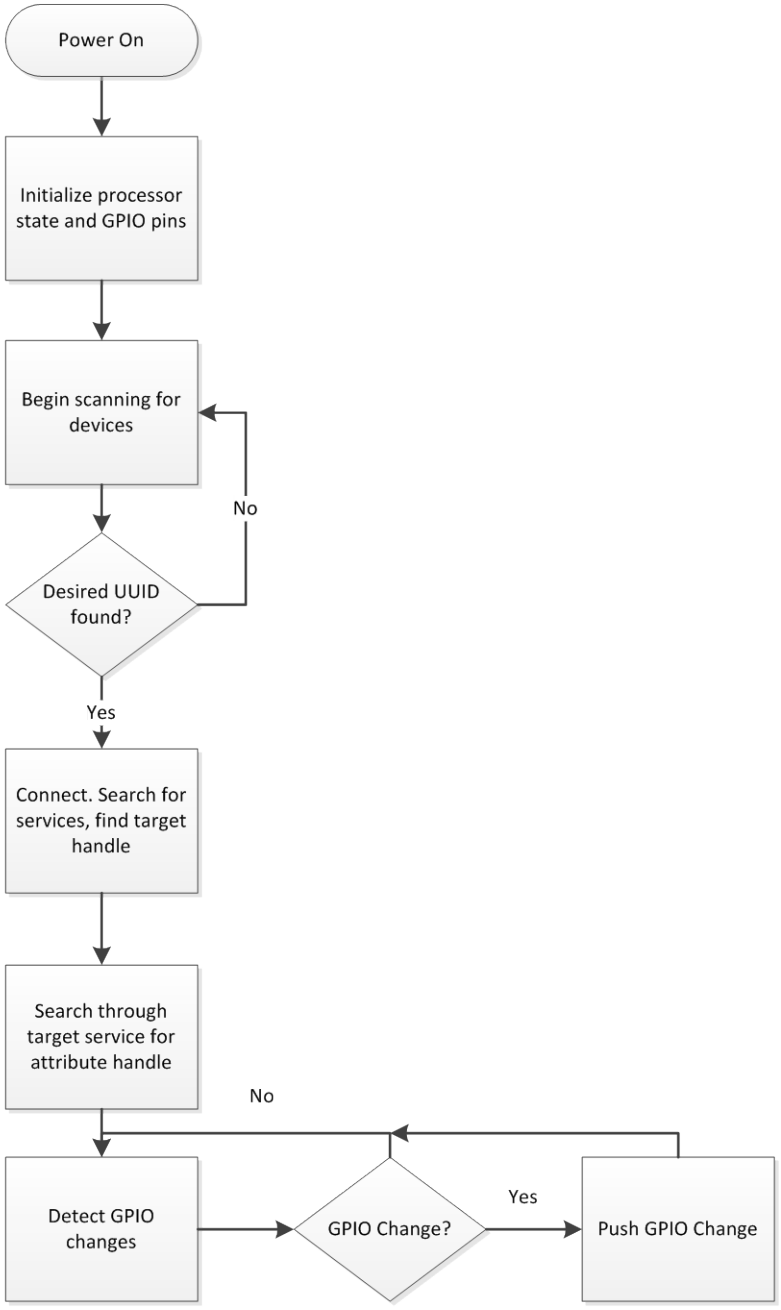


Figure 19: Transmitter Flow Diagram

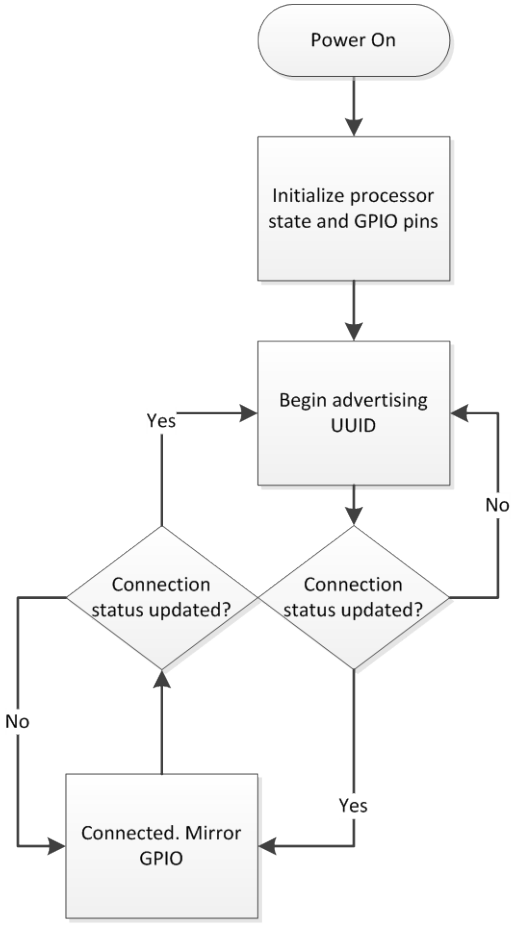


Figure 20: Receiver Flow Diagram

Testing

Problems Encountered

There were not many critical problems encountered during development. The most difficult problem was working with such a large test object, a motorcycle. Since I could not wheel my bike into on-campus labs, equipment was used at Trust Automation, my internship location for the 2014-2015 year. There, however, space and time was limited since I could not interrupt regular production during the day. Testing and troubleshooting analysis was typically limited to an hour or two at night.

A summary of the technical problems encountered is given below:

Problems	Solutions
Noise on signal lines	Use of opto-isolators increases “on” threshold and effectively removes noise from the input to the microcontroller. See figures 19 and 20.
Noise on power lines	Addition of decoupling capacitors near critical components (inverter, 12V rail, 3.3V rail).
Occasional “stuck” LED	Swapped inverting transistors with hex inverter to reduce lag and increase signal slope.
Occasional edge transitions missed	Increased TX power to 0dbm.
Orange and Red LED’s looked too similar	Changed wavelength of LED on next batch of strings.
PCB’s had internal shorts	PCB’s designed with components further apart, and a different manufacturer was selected.

Table 6: Project Problems and Solutions

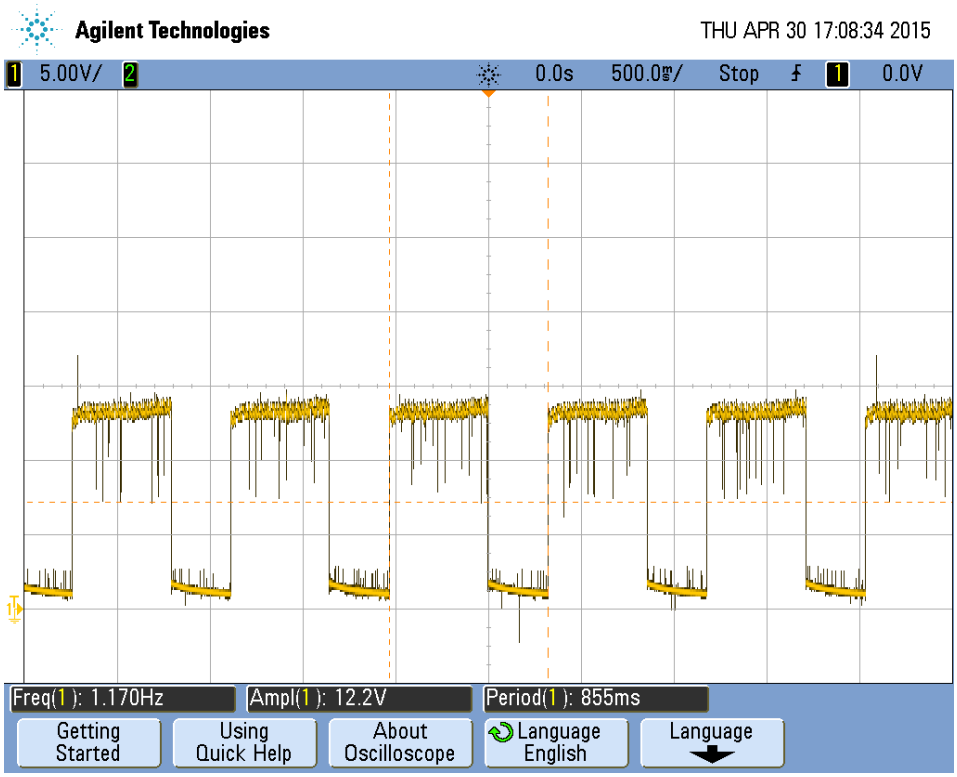


Figure 21: Scope capture of turn signal (engine on)

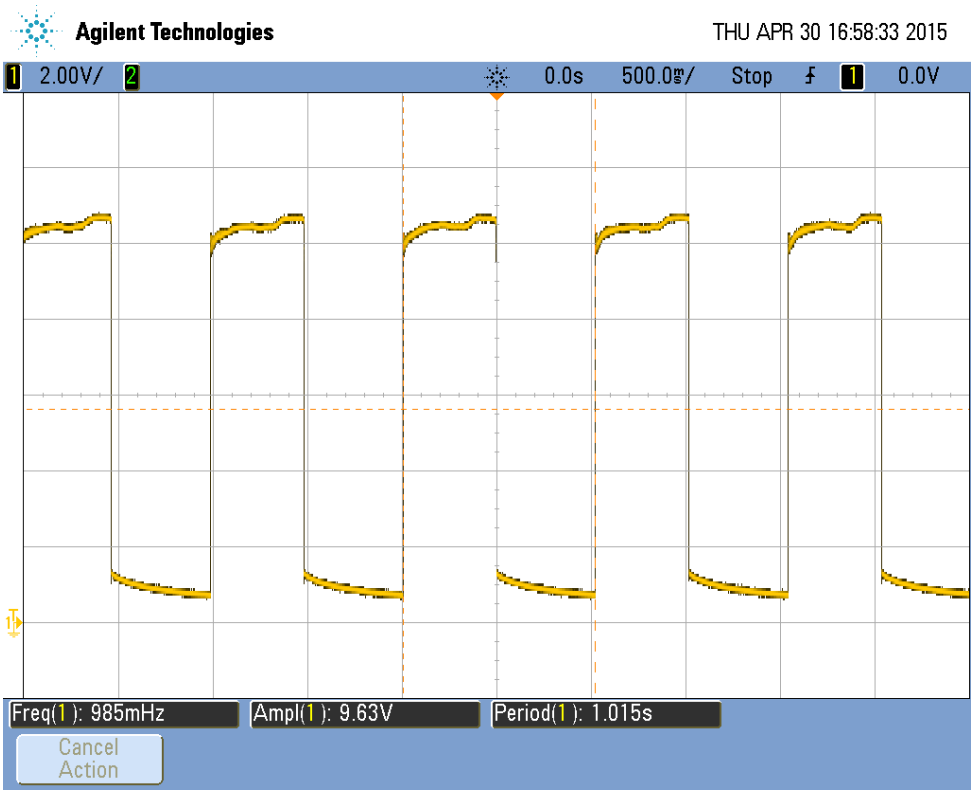


Figure 22: Scope capture of turn signal (engine off)

Test Cases

The basic functionality and corner cases were tested using, in part, the table below. The distance and TX / RX delay tests were not as critical as the basic functionality tests, but interesting nonetheless. Testing combinations of input, sequences of input, and other corner cases helped to debug remaining hardware and software issues.

Test	**Observations**	**Result**
Left Turn Signal	Blinks faster than the right turn signal	Success
Right Turn Signal	Blinks more slowly than the left turn signal	Success
Brake Indicator	No issues observed	Success
Rapid braking	Rapid oscillations caused “stuck” LED’s. Fixed issue by replacing inverting transistors with hex inverter IC.	Success
Alternating Left / Right Turn Signal	Occasional “stuck” LED’s if switching on a rising or falling edge. Fixed issue by replacing inverting transistors with hex inverter IC	Success
Left Turn + Brake Indicator	No issues observed	Success
Right Turn + Brake Indicator	No issues observed	Success

Jacket worn with backpack	Signal not impeded	Success
Distance from bike	50+ ft without signal interruption	Success
Daytime Brightness Test	Not highly visible, especially at steep angles	Partial Success
Nighttime Brightness Test	Very visible at large distances. Difficult to photograph due to overloading the camera sensor and reflections in the lens	Success
Current Idle	0.048 uA	Success
Current TX	18.0 mA	Success
Current RX	14.4 mA	Success
TX / RX Delay	<22 mS	Success
Battery life	Jacked was worn riding 1 hour every day for 3 weeks. Battery voltage dropped from the 4.2V finish voltage to their nominal 3.65V. It is estimated that the jacket could last for up to twice this long before needing to be recharged	Success / inconclusive

Table 7: Test Cases and Results

Figure 23: Daytime Brightness Test

Figure 24: Nighttime Brightness Test

Test Results and Conclusions

In all of the above tests, the jacket and Bluetooth Low Energy link performed very well. The exception to this was the daylight viewing test. With limited heat dissipation in the UV strips, the largest LED's possible for use were SMD 5050's. These LED's, while bright, are no match for full-brightness riding. The LED indicators are still visible, but do not necessarily increase the visibility of the motorcyclist. Additional testing will need to be done to see whether or not the LED's enhance visibility.

The difference in blink speed can be explained by the way flasher units work and by differences in the current conducted by each bulb. Inherently, more current though a lightbulb will cause it to blink faster. This is due to the thermal effects on a bi-metal conductor which creates an oscillator. More current, more heat, faster oscillating. Although peculiar, this is not a problem with the jacket but a symptom of the bike's flasher unit.

Timing tests revealed that even in worst-case scenario where both Transmitter and Receiver are asleep, the delay between the input and output of the system was less than 22 milliseconds. This system delay is certainly faster humans can typically perceive, and is very close to the time the "real" turn signals or brake indicators light up.

Future Development

This project merely scratched the surface for what is possible with wearable electronics and Bluetooth Low Energy. Continued work is planned for this project. Some of the future development includes:

- Built-in Lithium cell balancing
  Cell balancing is important for Lithium cells to extend their life and achieve maximum battery utilization.

- Built-in smart charge controller
  It would be useful to have a built-in charge controller to accept charging from a number of sources, including solar, wall-wart, USB, or other available resources.

- BLE Encryption to eliminate nefarious uses
  The BLE113 does support encrypted connections. This can be enabled to prevent accidental or intentional misuse of the jacket.

- Flexible solar panels to recharge while riding
  Adding flexible solar panels to the back of the jacket would recharge the batteries during normal day-time riding. 10W panels could likely charge the pack fully in just an hour of riding.

- Brighter LED's for day-time use
  As noted in testing, day-time use of the jacket is not as helpful as nighttime use. This can be remedied by using brighter LED's with a thicker substrate to dissipate heat.

- Smaller battery pack coupled with more efficient electronics
  The LDO regulators can be swapped for switching regulators, and the resistor-driven LED arrays can be switched to a more intelligent driver. These two things combined would allow for the battery pack to be reduced in size.

- Smartphone app for battery monitoring
  As noted in the test table, calculating battery life is difficult. The microcontroller would be much more capable than external circuitry to determine battery level. This level, along with the current sync status

could be relayed to a smartphone from the transmitter, receiver, or both.

- LED indicator for sync-status

  A quick “am I connected?” LED indicator can be added to the jacket to alert the user when they have successfully paired. This would be useful for troubleshooting if there was a problem with the bike or indicator circuit.
- RGB LED Strips

  RGB LED’s could be used for customization or safety. For example, in the case of an emergency stop the turn signals could also flash red instead of being orange or yellow.

- More compact PCB for transmitter and receiver

  At 2 in x 3 in, the current PCB may be too large or bulky for some newer bikes or smaller jackets.

- Weatherproof enclosure design

  The LED’s and jacket support riding in all weather, and so should the PCB’s! A weatherproof enclosure on the bike and jacket would allow riders to increase their visibility in the rain, where it is arguably the most important.

- Simplified wiring hookup for motorcycles

  Current hookup methods require users to understand their wiring harness and clip-in directly to it. While not terribly complex, this obstacle may turn away some less technical users.

- More advanced utilization of the TI CC2541

  The TI CC2541 is a very useful System On Chip, and could be utilized more fully. Examples of such utilization would be using the chip as a security alarm, monitoring OBD2 stats of the bike, or including a GPS tracker.

Appendix I: Works Cited and Referenced

Works Cited

[1] "List of Motorcycle Deaths in U.S. by Year." *Wikipedia*. Wikimedia Foundation, 24 Oct. 2014. Web. 05 Nov. 2014. <http://en.wikipedia.org/wiki/List_of_motorcycle_deaths_in_U.S._by_year>.

[2] "Motorcycle Accident Statistics and Possible Causes." *MotorcycleAccidentorg*. N.p., n.d. Web. 03 Nov. 2014. <https://motorcycleaccident.org/motorcycle-accidents-statistics-and-possible-causes/>.

[3] "Motorcycle Accidents: Common Causes | Nolo.com." *Nolo.com*. N.p., n.d. Web. 02 Nov. 2014. <http://www.nolo.com/legal-encyclopedia/motorcycle-accidents-common-causes-30330.html>.

[4]"Motorcycle Trends in the United States | Bureau of Transportation Statistics." *Motorcycle Trends in the United States | Bureau of Transportation Statistics*. N.p., n.d. Web. 04 Nov. 2014. <http://www.rita.dot.gov/bts/sites/rita.dot.gov.bts/files/publications/special_reports_and_issue_briefs/special_report/2009_05_14/html/entire.html>.

Works Referenced

"ATmega328P." *ATmega328P*. N.p., n.d. Web. 05 Nov. 2014. <http://www.atmel.com/devices/atmega328p.aspx>.

“[BGScript]: gpio_remote - Remote GPIO control with master and slave components.” BLE113. Web. 26 Aug. 2014.

“BLUEGIGA BLUETOOTH SMART SOFTWARE V.1.3 API DOCUMENTATION.” BLE113. Web. 10 Sep. 2014. >

“BGSCRIPT SCRIPTING LANGUAGE DEVELOPER GUIDE.” BLE113. Web. 3 Sep. 2014. >

“BLUETOOTH SMART PROFILE TOOLKIT DEVELOPER GUIDE.” BLE113. Web. 10 Sep. 2014. >

“BLUETOOTH SMART MODULE CONFIGURATION GUIDE.” BLE113. Web. 2 Sep. 2014. >

"The Complete Motorcycle Jacket Buying Guide." *EBay*. N.p., n.d. Web. 05 Nov. 2014. <http://www.ebay.com/gds/The-Complete-Motorcycle-Jacket-Buying-Guide-/10000000177630025/g.html>.

"Explaining Lithium-ion Chemistries." – *Battery University*. N.p., n.d. Web. 04 Nov. 2014. <http://batteryuniversity.com/learn/article/explaining_lithium_ion_chemistries>.

"The Hard Truth About Armor | CE or Not CE?" *Motorcyclist*. N.p., n.d. Web. 05 Nov. 2014. <http://www.motorcyclistonline.com/how-to/hard-truth-about-armor-ce-or-not-ce>.

"HelSTAR: The Wireless Helmet Brake & Signal Light." *Kickstarter*. N.p., n.d. Web. 05 Nov. 2014. <https://www.kickstarter.com/projects/2134363764/helstar-the-wireless-helmet-brake-and-signal-light>.

"Home." *HALO BELT*. N.p., n.d. Web. 05 Nov. 2014. <http://www.halobelt.com/>.

"Iluminadic Protective Illuminated Riding Clothing & Jackets." *Iluminaddic Protective Clothing & Jackets*. N.p., n.d. Web. 05 Nov. 2014. <http://www.iluminadic.com/>.

"Impulse Jackets Motorcycle Jackets." *Impulse Jackets Motorcycle Jackets*. N.p., n.d. Web. 05 Nov. 2014. <http://www.impulsejackets.com/>.

"Instant Cool: A Buyer's Guide Motorcycle Jackets." *The Guide to Street Motorcycle Jackets*. N.p., n.d. Web. 05 Nov. 2014. <http://www.bikebandit.com/community/guides/street-jackets-guide>.

"Lithium Battery Types." *Battery University*. N.p., n.d. Web. 12 Jun. 2014. <http://batteryuniversity.com/>.

"Motorcycle Crashes." *III*. N.p., n.d. Web. 04 Nov. 2014. <http://www.iii.org/issue-update/motorcycle-crashes>.

"Motorcycle Crash-Related Data." *Centers for Disease Control and Prevention*. Centers for Disease Control and Prevention, 14 June 2012. Web. 04 Nov. 2014. <http://www.cdc.gov/features/dsMotorcycleSafety/>.

"Motorcycle Personal Protective Equipment." *Wikipedia*. Wikimedia Foundation, 24 Oct. 2014. Web. 05 Nov. 2014. <http://en.wikipedia.org/wiki/Motorcycle_personal_protective_equipment>.

"MuzaMoto." *MuzaMoto*. N.p., n.d. Web. 05 Nov. 2014. <http://www.muzamoto.com/>.

"Photon Blaster." *Skene Design Motorcycle Visibility Lights*. N.p., n.d. Web. 05 Nov. 2014. <http://www.lights.skenedesign.com/>.

[Slus083*], Texas Instruments Incorporated. *Low-Dropout Li-Ion Charge-Control IC With AutoComp Charge-Rate Compensation* (n.d.): n. pag. Web.

[Snvs043B, Texas Instruments Incorporated, and]. *LM3622 Lithium-Ion Battery Charger Controller (Rev. B)* (n.d.): n. pag. Web.

"Solar Flex Kits and Modules (Flat, Bendable Solar)." *Mobile Power Systems and Solutions for RV, Utility, Fleet and Marine Application*. N.p., n.d. Web. 05 Nov. 2014. <http://gpelectric.com/products/solar-flex-kits-modules>.

"TI LaunchPad." *Ultra-Low-Power MSP430 Microcontroller LaunchPad Kits*. N.p., n.d. Web. 04 Nov. 2014. <http://www.ti.com/ww/en/launchpad/launchpads-msp430.html>.

"Welcome to Street Lightning, Inc." *Street Lightning, Inc.* N.p., n.d. Web. 05 Nov. 2014. <http://streetlightning.net/>.

"Wireless Controlled - LED Motorcycle Backpack & Jacket!" *Kickstarter*. N.p., n.d. Web. 05 Nov. 2014. <https://www.kickstarter.com/projects/541527645/lighting-de-soleil-wireless-led-motorcycle-gear/?ref=kicktraq>.

Appendix II: Senior Project Analysis

Summary of Function Requirements

The primary functional requirement of the LED motorcycle jacket is improve daytime and nighttime visibility for motorcycle riders. This is done via a wireless Bluetooth Low Energy synchronization between the rider's motorcycle and the proposed jacket. Indicators built into the back and shoulders of the jacket will alert other vehicles on the road to changes in speed or lane.

Primary Constraints

The most difficult constrain of the project is making a product easy enough to use and streamlining it into riders' everyday habits. This enumerates into a number of specifications including long-lasting batteries, no-fuss auto connectivity, and a high-quality sleek looking jacket.

Economic

The development and manufacturing of the LED motorcycle jacket will not stimulate as many jobs as the marketing and selling of the jacket. One of the hardest parts of introducing a new product is making your target audience believe they need it. The motorcycle scene is no different, and marketing and sales teams must carefully advertise such a jacket to appeal to as many riders as possible.

If Manufactured on a Commercial Basis

An estimated 500 jackets would be sold the first year, growing on that each consecutive year. The total cost of the jacket (see above) is estimated to be. Estimated purchase price for each jacket is $500. Cost to operate the device could be zero if a solar charging device was used exclusively. Otherwise, a small operating cost exists for owners to charge the jacket with a wall AC adapter. Little to no operating cost would be incurred per unit sold to the manufacturer.

Environmental

One of the major concerns with such a product is disposal. While the jacket could still be worn after an electrical failure or battery malfunction, there are more toxic chemicals and elements in the PCB's and batteries. It is recommended that a battery-return service be offered to accept old battery modules free of charge from customers in order to keep them out of the landfill.

Manufacturability

Most of the parts of the LED jacket are easily manufactured by any company without special skills. The most complicated pieces are the jacket and batteries, both of which would be manufactured by a third party. If production increase past the abilities of a small work force, assembly of major components (receiver, transmitter modules) would be outsourced to a 3rd party manufacturer.

Sustainability

No major parts of this project are currently scarcely available or show signs of being discontinued. The market for an LED jacket continues to grow each year as the number of motor cycle riders (and accidents) increases each year.

Ethical

An LED motorcycle jacket as proposed in this document would have many positive ethical implications. Such a jacket would save lives, prevent injuries, and save thousands of dollars in insurance payout for accidents caused by lack of visibility. No obvious negative ethical implications arise from such a product.

Health and Safety

This project poses no health or safety threat to the user or those around them. The health and safety danger to those in the manufacturing of the product is very low, but precautions must be taken. The manufacture of electronics, especially batteries, inherently involves some very toxic chemicals which must be handled properly by trained professionals.

Social and Political

As something that does not have many nefarious uses and does not negatively affect its user or those around it, the jacket should not induce any social or political issues. Decisions on manufacturing would have to be made (US or overseas) and import / export regulations must be considered and honored when going into full scale production.

Development

Throughout the course of this project I personally developed many skills. This included:

- Analysis of rise / fall times with respect to interrupt-driven hardware
- Bluetooth Low Energy architecture, connectivity, and usage
- SMD LED hardware, specifications, selection, and manufacturing
- Battery construction, safety, chemistry, management, storage, charging, discharging, trade-space, and selection for projects
- The BGScript language utilized by BlueGiga hardware

Appendix III: BGS Code

Master / Transmitter

```
# ================================================================
# Smart LED Jacket, transmitter side bgs file
# Sean O'Brien
# ----------------------------------------------------------------
#
# ================================================================

# ================================================================
# BASIC ARCHITECTURAL OVERVIEW:
#
#       a. Initialize GPIO pins for correct interrupt edges
#       b. Begin scanning for devices
#       c. If the desired UUID is found in an ad packet, connect to that device
#       d. Search for all "service" descriptors to find the target service handle range
#       e. Search through the target service to find the target attribute handle
#       f. Detect GPIO changes and push to remote slave device
#
# FUNCTION ANALYSIS:
#
# 1. system_boot:
#     Raised on boot/reset. This event handler enables relevant interrupt edge
#     detection on desired, pins, then sets scan parameters and initiates a
#     scan with the "gap_discover" command.
#
# 2. gap_scan_response:
#     Raised during scanning whenever an advertisement packet is detected. The
#     data provided includes the MAC address, RSSI, and ad packet data payload.
#     This payload includes fields which contain any services being advertised,
#     which allows us to scan for a specific service. In this demo, the service
#     we are searching for has a custom 128-bit UUID, which is compared against
#     any existing 128-bit UUID values found in the advertisement packet. Once
#     a match is found, the application initiates a connection request with the
#     "gap_connect_direct" command.
#
# 3. connection_status
#     Raised when the connection status is updated. This happens when the
#     connection is first established, and the "flags" byte will contain 0x05 in
#     this instance. However, it will also happen if the connected devices bond
#     (i.e. pair), or if encryption is enabled (e.g. with "sm_encrypt_start"),
#     or if either side of the link updates the connection parameters used. Once
#     a connection is established, the script begins a service discovery with
#     the "attclient_read_by_group_type" command.
#
# 4. attclient_group_found
#     Raised for each group found during the search started in #3. If the right
#     service is found (matched by UUID), then its start/end handle values are
#     stored for usage later. We cannot use them immediately because the ongoing
#     read-by-group-type procedure must finish first.
#
# 5. attclient_find_information_found
#     Raised for each attribute found during the search started after the service
#     search completes. We look for one specific attribute during this process:
#     the 128-bit UUID designating the GPIO remote control characteristic's data
#     attribute. If/when this is found, it is stored in another variable for use
#     later.
#
# 6. attclient_procedure_completed
#     Raised when an attribute client procedure finishes, which in this script
#     means when the "attclient_read_by_group_type" (service search) or the
#     "attclient_find_information" (descriptor search) completes. Since both
#     processes terminate with this same event, we must keep track of the state
#     so we know which one has actually just finished. The completion of the
```

```
#     service search will (assuming the service is found) trigger the start of
#     the descriptor search, and the completion of the descriptor search will
#     (assuming the attribute is found) set the correct application state so
#     that GPIO data is sent over the air to the slave when needed.
#
# 7. connection_disconnected
#     Raised when a connection is terminated. This is used only to put the
#     BLE device back into a scanning state, essentially starting the process
#     over again.
#
# 8. hardware_soft_timer
#     Raised when a scheduled soft timer interval elapses. This is used only
#     during the connection process to detect if too much time has elapsed
#     since the connection attempt began (4 seconds in this demo). If the
#     timer fires, then the connection attempt is cancelled and the module is
#     returned to the scanning state. The timer is cancelled preemptively if
#     the connection is fully established within 4 seconds.
#
# 9. hardware_io_port_status
#     Raised when a GPIO interrupt occurs. Once the connection is established
#     and all necessary GATT discovery has occurred successfully, GPIO data is
#     written to the GATT server (slave) when necessary. Since the CC254x
#     chipset inside the module does not support CHANGE interrupts, the script
#     enables rising-edge interrupts on P1_x pins and falling-edge interrupts
#     on P0_x pins, and pulls pins on both ports low. Electrically, you can
#     then connect a button (or any signal) to BOTH matching pins on each port
#     (e.g. P0_5 and P1_5), and effectively get "pin change" interrupt data.
#     This configures all 8 available pins to work this way, assuming the
#     following electrical connections:
#
#         Vdd -> button -> P0_0 & P1_0
#         Vdd -> button -> P0_1 & P1_1
#         Vdd -> button -> P0_2 & P1_2
#         Vdd -> button -> P0_3 & P1_3
#         Vdd -> button -> P0_4 & P1_4
#         Vdd -> button -> P0_5 & P1_5
#         Vdd -> button -> P0_6 & P1_6
#         Vdd -> button -> P0_7 & P1_7
#
# ================================================================

const PIN_SELECT_MASK = $FF         # 0b11111111 = Px_0-Px_7 are outputs
#const PIN_SELECT_MASK = $E0         # 0b11100000 = Px_5/6/7 are outputs

const STATE_STANDBY = 0
const STATE_SCANNING = 1
const STATE_CONNECTING = 2
const STATE_FINDING_SERVICES = 3
const STATE_FINDING_ATTRIBUTES = 4
const STATE_CONTROL_READY = 5

dim app_state                       # keep track of application state

dim pending_connection_handle       # handle for pending connection attempt

dim att_handlesearch_start          # GPIO remote GATT service definition range handle
start
dim att_handlesearch_end            # GPIO remote GATT service definition range handle
end
dim att_handle_gpio_control         # local var for discovering/storing remote handle

dim ad_field_length
dim ad_field_type

dim i
dim j
dim k
dim ret_result
dim temp_buf(16)

event system_boot(major, minor, patch, build, ll_version, protocol_version, hw)
```

```
    # initialize all status/tracking vars
    app_state = STATE_STANDBY
    pending_connection_handle = $ff
    att_handlesearch_start = 0
    att_handlesearch_end = 0
    att_handle_gpio_control = 0

    # enable falling-edge interrupts on Port0 pins
    call hardware_io_port_config_pull(0, $00, 0)
    call hardware_io_port_irq_enable(0, PIN_SELECT_MASK)
    call hardware_io_port_irq_direction(0, 1)

    # enable rising-edge interrupts on Port1 pins
    # (dual-pin connections allows CHANGE-style interrupts)
    call hardware_io_port_config_pull(1, $00, 0)
    call hardware_io_port_irq_enable(1, PIN_SELECT_MASK)
    call hardware_io_port_irq_direction(1, 0)

    # set scan parameters to 125ms/125ms interval/window, and use passive scanning
    call gap_set_scan_parameters(200, 200, 0)

    # start discovery
    app_state = STATE_SCANNING
    call gap_discover(gap_discover_generic)
end

# catch scan response event while scanning
event gap_scan_response(rssi, packet_type, sender, address_type, bond, data_len,
data_data)
    # only check for main ad packets (packet_type = 0)
    if packet_type = 0 then
    # advertisement packet found during scan, so check for demo status/control service
    # searched UUID is defined in the slave GATT as 47f1de41-c535-414f-a747-1184246636c6
        # NOTE: LITTLE-ENDIAN BYTE ORDER
        temp_buf(0:16) =
"\xc6\x36\x66\x24\x84\x11\x47\xa7\x4f\x41\x35\xc5\x41\xde\xf1\x47"

        i = 0
        while i < data_len
            ad_field_length = data_data(i:1)
            ad_field_type = data_data(i + 1:1)
            if ad_field_type = $06 || ad_field_type = $07 then
                # partial ($06) or complete ($07) list of 128-bit UUIDs

                j = 0
                while j < ad_field_length - 1
                    if memcmp(data_data(i + j + 2), temp_buf(0), 16) && app_state =
STATE_SCANNING then
                    # found GPIO remote service, so connect (stops scanning automatically)
                        call gap_connect_direct(sender(0:6), address_type, $20, $30, 100,
0)(ret_result, k)
                        pending_connection_handle = k

                        # update application state
                        app_state = STATE_CONNECTING

                        # start 4-second one-shot timer to detect connection timeout
                        call hardware_set_soft_timer(32768*4, 0, 1)

                        # exit gap_scan_response event handler immediately
                        return
                    end if
                    j = j + 16
                end while
            end if
            i = i + ad_field_length + 1
        end while
    end if
end

# catch connection update event
```

```
event connection_status(connection, flags, address, address_type, conn_interval, timeout,
latency, bonding)
    # check for "new connection established" update
    if (flags & $05) = $05 then
        # make sure this is the first "connection_update" event
        # (may be triggered again for other updates, encryption, etc.)
        if app_state = STATE_CONNECTING then
            # cancel connection timeout timer
            call hardware_set_soft_timer(0, 0, 1)
            pending_connection_handle = $ff

            # start searching through service groups (UUID = 0x2800) to find GPIO remote
service
            call attclient_read_by_group_type(connection, $0001, $ffff, 2, "\x00\x28")

            # update application state
            app_state = STATE_FINDING_SERVICES
        end if
    end if
end

# catch group found event (during GATT service discovery)
event attclient_group_found(connection, start_handle, end_handle, uuid_len, uuid_data)
    # found a service group, so check to see if it's the GPIO remote service
    # searched UUID is defined in the slave GATT as 47f1de41-c535-414f-a747-1184246636c6
    # ^^^ NOTE: "temp_buf" will still contain this value from the GAP device scan earlier
    if uuid_len = 16 && memcmp(uuid_data(0), temp_buf(0), 16) then
        # found it! save the handle range
        att_handlesearch_start = start_handle
        att_handlesearch_end = end_handle
    end if
end

event attclient_find_information_found(connection, chrhandle, uuid_len, uuid_data)
    # found a descriptor, so check to see if it's the the right one
    # ^^^ NOTE: "temp_buf" will contain the correct value for this search, having been
written
    #           in the "attclient_procedure_completed" handler before starting the search
    if uuid_len = 16 && memcmp(uuid_data(0), temp_buf(0), 16) then
        # found the GPIO control attribute, so save the handle
        att_handle_gpio_control = chrhandle
    end if
end

# catch procedure completed event (during GATT discovery and upon write acknowledgement)
event attclient_procedure_completed(connection, result, chrhandle)
    if app_state = STATE_FINDING_SERVICES then
        # just finished scanning for services
        if att_handlesearch_start > 0 then
            # found GPIO remote service, so now find the control characteristic value
handle
            temp_buf(0:16) =
"\xe7\xa9\x1e\x26\xbc\x50\x93\x84\x4a\x4b\xc0\x06\xc7\xb6\x08\xf4"
            call attclient_find_information(0, att_handlesearch_start,
att_handlesearch_end)

            # update application state
            app_state = STATE_FINDING_ATTRIBUTES
        #else
            # if the "else" matches, we didn't find the service...uh oh
        end if
    else
        if app_state = STATE_FINDING_ATTRIBUTES then
            # just finished scanning for attributes_read
            if att_handle_gpio_control > 0 then
                app_state = STATE_CONTROL_READY
            #else
                # if the "else" matches, we didn't find the control attribute...uh oh
            end if
        #else
            # just acknowledged a GATT write operation from a GPIO logic change
```

```
            # (not important for this example)
        end if
    end if
end

# catch disconnection event
event connection_disconnected(handle, result)
    # restart discovery
    app_state = STATE_SCANNING
    call gap_discover(gap_discover_generic)
end

# catch timer tick (used to update the slave and detect connection attempt timeouts)
event hardware_soft_timer(handle)
    if app_state = STATE_CONNECTING && pending_connection_handle != $ff then
        # end connection attempt (1st line required, 2nd line failsafe)
        call gap_end_procedure()
        call connection_disconnect(pending_connection_handle)

        # restart discovery
        app_state = STATE_SCANNING
        call gap_discover(gap_discover_generic)
    end if
end

# catch GPIO interrupts to pass on logic states to the slave
event hardware_io_port_status(timestamp, port, irq, state)
    if app_state = STATE_CONTROL_READY then
        # connected, so write new GPIO port bitmask value to remote device
        call attclient_attribute_write(0, att_handle_gpio_control, 1, state)
    end if
end

<!--
Smart LED Jacket, transmitter side gatt file
Sean O'Brien
</!-->

<?xml version="1.0" encoding="UTF-8" ?>
<configuration>

    <!-- 1800: org.bluetooth.service.generic_access -->
    <service uuid="1800" id="generic_access">
        <description>Generic Access</description>

        <!-- 2A00: org.bluetooth.characteristic.gap.device_name -->
        <characteristic uuid="2A00" id="c_device_name">
            <description>Device Name</description>
            <properties read="true" const="true" />
            <value>GPIO Remote Master</value>
        </characteristic>

        <!-- 2A01: org.bluetooth.characteristic.gap.appearance -->
        <characteristic uuid="2A01" id="c_appearance">
            <description>Appearance</description>
            <properties read="true" const="true" />
            <value type="hex">0100</value>
        </characteristic>

    </service>

    <!-- 180A: org.bluetooth.service.device_information -->
    <service uuid="180A" id="device_information">
        <description>Device Information</description>

        <!-- 2A29: org.bluetooth.characteristic.manufacturer_name_string -->
        <characteristic uuid="2A29" id="c_manufacturer_name">
            <description>Manufacturer Name</description>
            <properties read="true" const="true" />
            <value>Bluegiga</value>
        </characteristic>
```

```
        <!-- 2A24: org.bluetooth.characteristic.model_number_string -->
        <characteristic uuid="2A24" id="c_model_number">
            <description>Model Number</description>
            <properties read="true" const="true" />
            <value>GPIO Remote Master Device</value>
        </characteristic>

    </service>

</configuration>
```

Slave / Receiver

```
# ================================================================
# Smart LED Jacket, receiver side bgs file
# Sean O'Brien
# ----------------------------------------------------------------
#
# ================================================================

# ================================================================
# BASIC ARCHITECTURAL OVERVIEW:
#
#       a. Initialize GPIO pins for correct output/low logic
#       b. Begin advertising as connectable
#       c. After connecting, detect and apply new logic states from master
#
# FUNCTION ANALYSIS:
#
# 1. system_boot:
#     Raised on boot/reset. This event handler sets the desired Port1 pins to
#     OUTPUT mode and initialized them to a logic low state, then puts the
#     module into an advertising/connectable state.
#
# 2. connection_status (PLACEHOLDER ONLY, NOT ACTUALLY USED)
#     Raised when the connection status is updated. This happens when the
#     connection is first established, and the "flags" byte will contain 0x05 in
#     this instance. However, it will also happen if the connected devices bond
#     (i.e. pair), or if encryption is enabled (e.g. with "sm_encrypt_start"),
#     or if either side of the link updates the connection parameters used. This
#     demo does not require any unique behavior based on a new connection.
#
# 3. connection_disconnected
#     Raised when a connection is terminated. This is used only to put the
#     BLE device back into an advertising/connectable state.
#
# 4. attributes_value
#     Raised when a remote GATT client writes a new value to a local GATT
#     characteristic. In this demo, that means that the remote device has sent
#     a new GPIO logic value which we need to apply to our Port1 output pins.
#
# ================================================================

const PIN_SELECT_MASK = $FF         # 0b11111111 = Px_0-Px_7 are outputs
#const PIN_SELECT_MASK = $E0         # 0b11100000 = Px_5/6/7 are outputs

event system_boot(major, minor, patch, build, ll_version, protocol_version, hw)
    # set desired Port1 pins to output/low mode
    call hardware_io_port_config_direction(1, PIN_SELECT_MASK)
    call hardware_io_port_write(1, PIN_SELECT_MASK, $00)

    # set advertisement interval to 200-300ms
    call gap_set_adv_parameters(320, 480, 7)

    # put module into discoverable/connectable mode
```

```
    call gap_set_mode(gap_general_discoverable, gap_undirected_connectable)
end

# catch connection update event
#event connection_status(connection, flags, address, address_type, conn_interval,
timeout, latency, bonding)
    # the slave side does not care about connection details,
    # but this event handler is left here for convenience in
    # case additional functionality is needed
#end

# catch disconnection event
event connection_disconnected(handle, result)
    # put module back into discoverable/connectable mode
    call gap_set_mode(gap_general_discoverable, gap_undirected_connectable)
end

# catch remote write of characteristic value
event attributes_value(connection, reason, handle, offset, value_len, value_data)
    if handle = c_gpio_levels then
        # write single byte to Port1 (masked by output pin selection value)
        call hardware_io_port_write(1, PIN_SELECT_MASK, value_data(0:1))
    end if
end



<!--
Smart LED Jacket, receiver side gatt file
Sean O'Brien
</!-->

<?xml version="1.0" encoding="UTF-8" ?>
<configuration>

    <!-- 1800: org.bluetooth.service.generic_access -->
    <service uuid="1800" id="generic_access">
        <description>Generic Access</description>

        <!-- 2A00: org.bluetooth.characteristic.gap.device_name -->
        <characteristic uuid="2A00" id="c_device_name">
            <description>Device Name</description>
            <properties read="true" const="true" />
            <value>GPIO Remote Slave</value>
        </characteristic>

        <!-- 2A01: org.bluetooth.characteristic.gap.appearance -->
        <characteristic uuid="2A01" id="c_appearance">
            <description>Appearance</description>
            <properties read="true" const="true" />
            <value type="hex">0100</value>
        </characteristic>

    </service>

    <!-- 180A: org.bluetooth.service.device_information -->
    <service uuid="180A" id="device_information">
        <description>Device Information</description>

        <!-- 2A29: org.bluetooth.characteristic.manufacturer_name_string -->
        <characteristic uuid="2A29" id="c_manufacturer_name">
            <description>Manufacturer Name</description>
            <properties read="true" const="true" />
            <value>Bluegiga</value>
        </characteristic>

        <!-- 2A24: org.bluetooth.characteristic.model_number_string -->
        <characteristic uuid="2A24" id="c_model_number">
            <description>Model Number</description>
            <properties read="true" const="true" />
            <value>GPIO Remote Slave Device</value>
        </characteristic>
```

```
    </service>

    <!-- custom service for providing and controlling demo slave behavior -->
    <service uuid="47f1de41-c535-414f-a747-1184246636c6" advertise="true">
        <description>GPIO Remote Control Service</description>

        <!-- custom characteristic for acknowledged status and control point -->
        <characteristic uuid="f408b6c7-06c0-4b4a-8493-50bc261ea9e7" id="c_gpio_levels">
            <description>GPIO Remote Control Characteristic</description>
            <properties write="true" indicate="true" />
            <value length="1" />
        </characteristic>

    </service>

</configuration>
```