Bike Safe

An Automatic Turn Signal System

by

Betty Chan
Ricardo Duran

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT
California Polytechnic State University,
San Luis Obispo

2013-2014

Abstract

Bicycles are vehicles that do not require a license to operate and share the road with cars, motorcycles, and other fast moving vehicles. An improper lane change from a bicyclist, such as one without any warning, can lead to great injury or death (especially after sundown). Bike Safe addresses this safety concern by providing a low-cost device that mimics the warning system motorists' use every day. The system consists of a red brake light and amber turning signal lights that mounts to the rear of the bike, as well as a white headlamp that mounts to the front handlebars. The bicyclist safely and easily controls the turning lights signal without removing their hands from the handlebars. In addition, when the bicyclist applies the brakes, the red brake light significantly brightens until the bicyclist releases the brakes. Bike Safe also has a solar panel that charges the battery in order to provide a more reliable power source and a longer battery life for night-time bike rides.

Table of Contents

Lists of Tables and Figures

Table **Page**

Figures **Page**

Acknowledgements

We would like to thank Professor Bryan Mealy for being an outstanding advisor and Dr. Dennis Derrickson for his encouragement throughout the project. We would also like to thank Jaime Carmo for all the tools, resources, and services that he provided us in order to complete this project.

Chapter 1. Introduction

Bike Safe is a device that allows bicyclists to indicate that they are making a turn without having to take their hands off the handle bar. The system consist of a red brake light, a white headlight, and two amber turning lights, one on each side of the brake light to indicate left and right. The user activates the flashing amber lights from the control system mounted on the handlebar by using their fingers, thus, eliminating the use of hand signals. The Bike Safe's brake alert system consists of consist of red LEDs that increase in brightness when the bicyclist applies the brakes. This system is a necessary bicycle improvement when riding at night because it displays the bicyclist's intention to surrounding vehicles, thus preventing injuries or death.

Chapter 2. Background

Bicycle riders that shares that public road with other motorists are subject to the same rules and regulations, such as having working brakes at all times and having lights and reflectors when riding at night. According to the National Highway Traffic Safety Administration, there were 726 bicyclist deaths and 49,000 bicyclist injuries in 2012 and nearly 29% of those injuries are caused collisions with cars. 48% of these fatalities occurred between 4:00pm to 11:59pm when the sun is setting. We can reduce these numbers by having the proper equipment to be more visible to motorist at night, especially when the bicyclists are weaving through traffic lanes with other motorists.

Chapter 3. Requirements and Specifications

For marketing appeal the device is light in weight in order to maintain the performance and comfort of the bicyclist. The low retail price will peak people's interest in investing with our product. Once customers view first hand the easy installation and usability as well as the safety concerns it addresses, they will be inclined to purchase the product for their bicycle.

To support these marketing needs, TABLE I defines the design specifications of the system. The following explains TABLE I:

- Lightweight at three pounds or less.
- Small dimension at 5" x 2.25" x 2.5".
- Slow flashing amber lights.
- Brighter red lights when squeezing brakes.
- Harvest energy from solar panels and stores energy in battery.

TABLE I

BIKE SAFE REQUIREMENTS AND SPECIFICATIONS

Marketing Requirements	Engineering Specifications	Justification
1,2,4	System should weigh no more than 3 lb.	The weight of the system should not reduce the performance of the bicyclist.
1,2,4	The panel dimensions, without any mounting brace, should not exceed 5" x 2.25" x 2.5".	The system should be large enough to be easily noticeable by others and fit a seat post at its lowest position.
2,3	Amber lights will flash at 1 Hz.	Faster frequency may produce discomfort to others.
2,3	Red light will run at 90% duty cycle when brakes are applied and 20% duty cycle when not pressing brakes.	This captures others attention and acts as a warning that the biker is slowing down.
4,5	System will be powered by a rechargeable battery with a solar panel as backup.	Rechargeable batteries are better for the environment since they support a significant amount of reuse. A solar panel extends system battery life between recharges.
Marketing Requirements 1. Light-weight and portable. 2. Easy to see. 3. Easy to operate. 4. Low cost. 5. Maintainable.		

Chapter 4. Functional Decomposition (Level 0 and Level 1)

The Bike Safe device operates from user input. ***Figure 1*** shows that the user determines the direction they want to turn and when they want to brake. After the user initiates their input, the system outputs amber LEDs and/or red LEDs. The white light input turns on/off the white headlamp and red back light for night-time riding. Rechargeable batteries power the system, which also uses a solar cell to slow down discharge during the day. ***TABLE II*** provides a brief description of the input and output signals of the Bike safe system.

Figure 1: Level-0 Bike Turning Signal Functionality

TABLE II

BIKE SAFE LEVEL 0 FUNCTIONALITY TABLE

Module	**Bike Turning Signal**
Inputs	• Power: rechargeable batteries and solar panel • Turning input: user selects left or right turn • Brake input: user applies the brakes • White light input: user determines white/red light to be on
Outputs	• Amber LED: Left or right Flashing LED output • Red brake LED: Red Light Brightens • White/Red LED: Front and back light turns on
Module	This module flashes amber LEDs relative to user's input of right or left turn. The flashing LEDs should automatically turn off under specific circumstances. The switch turns on the headlamp and back light. The red light brightens from 20% duty cycle to 90% duty cycle.

The Level-1 block diagram shown in ***Figure 2*** consists of four types of inputs: Power, turning input, white light switch input, and brake input. Power provides energy to the system and allows it to function. The user provides the turning input from the turning signal control; the Yellow LED Driver processes the signal and determines which amber LED flashes. The white light switch input allows the user to determine one of three options: White and red LED off, white and red led at 60% duty cycle and 20% duty cycle respectfully, or white and red led at 90% duty cycle and 20% duty cycle. The red LED brightness control processes the brake input, which outputs an increased brightness of the red LED.

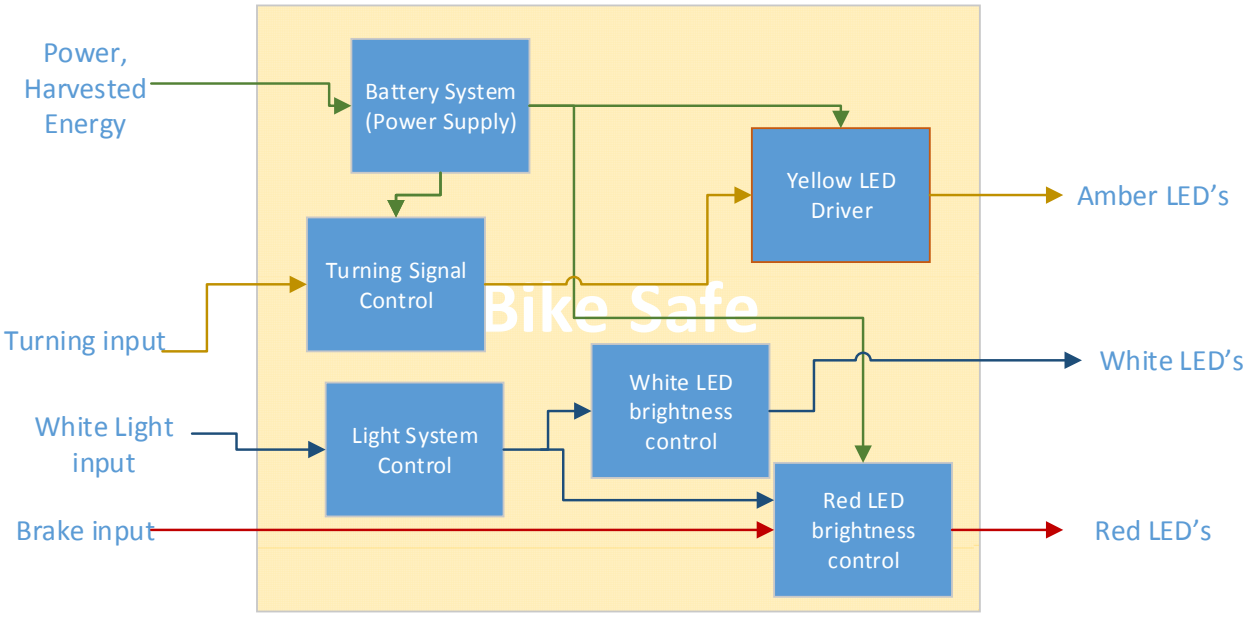


Figure 2: Level 1 Bike Turning Signal Functionality

Chapter 5. Design

This chapter describes the hardware and software design of the Bike Safe system.

Hardware Design

The system uses the ATmega328p microprocessor from the Arduino Nano development board because of its compact and its simplistic nature. ***Figure 3*** shows the complete layout and schematic of the device based on the pinouts from the Arduino Nano.

- Power

The system obtains its power from a 9.6V 1600mAh rechargeable NiMH battery with a 12V 1.5W solar panel to extend the battery life. The solar panel has a diode at the positive terminal to prevent discharging the battery through the solar panel when it is dark. The amount of time the solar panel charges the battery depends on the battery level being read by the Analog Pin 0 of the Arduino Nano. In order to protect the microprocessor from overvoltage on the analog pin, the system uses a voltage divider to scale the 9.6V battery voltage down to an acceptable voltage, 5V, without harming the pin. The voltage divider has a 1MΩ and 510kΩ resistor allowing a maximum of 15V input from the battery using Vout=Vin(R2/(R1+R2)).

- Turn Sensor

In order to detect that a completed turn, a unipolar hall-effect sensor AH201 in front of the bike detects a magnet that is in parallel to the hall-effect sensor when the wheels are straight. When the hall-effect sensor detects a magnet, the output of the sensor changes to from high to low. The Analog Pin 1 of the Arduino Nano detects the signal indicating that the wheels are back in the straight position and to turn off the turn signal lights.

- LED Outputs

The system consists of two sets of three amber LEDs for the left and right signal, a red 12V trailer light module, and a white 9-LED headlight. The digital pins of the Arduino Nano controls these outputs. The outputs depend solely on the buttons being pushed and the state of the hall-effect sensor.

- Buttons/Switch Input

There are multiple input buttons in the system. There is a pair of buttons to input left or right turn, two buttons specially customized to fit on the brakes, and a switch for turning the white and red lights on when it gets dark.

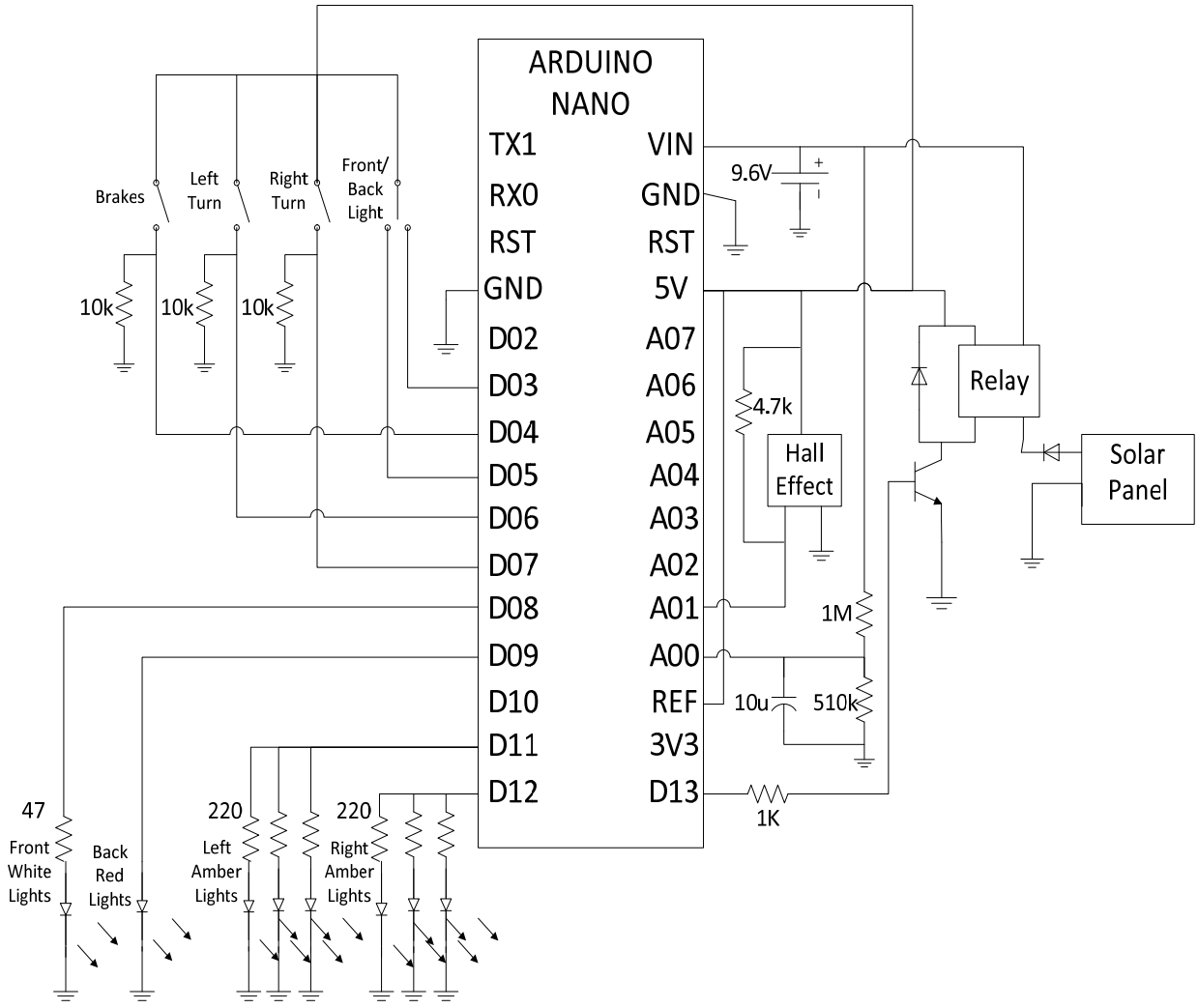


Figure 3: Layout of Bike Safe

Software Design

The Bike Safe project uses C with Atmel Studio 6 to program the ATmega328p microprocessor. In order to minimize the size of the project and components used, a majority of the functionalities are done through software.

Figure 4 shows the flowchart of the main function where the Timers initializes and interrupts enables. In the main function, Analog Pin 1 initializes to check the state of the hall effect output and will either clear or set a flag. The main function also checks the battery voltage level and places the Arduino Nano into power-saving mode when it does not sense anything.

Figure 5 shows the flowchart of the battery charge control function that utilizes Analog Pin 0. First, the function reads the battery voltage from the voltage divider because the analog pin's maximum input is 5V. For accurate ADC readings, the function takes 20 samples and averages the values of those points to determine the ADC value. The value determines which state the relay should be in. The solar panel charges the battery when the ADC value is less than 747 (battery at 10.8V) and will stop charging when the ADC value reaches 768 (battery at 11.1V).

Figure 6 shows the flowchart of the sleep mode function. The microprocessor goes to power-saving mode when there are no interrupts or when the switch is in the off position. It wakes up when there is a pin change.

Figure 7 shows the flowchart of the SPDT switch position function. There are 3 positions: off, low-beam, and high-beam. In the off position, the white headlight and red back lights are off. In

the low-beam position, the white headlight is on at 60% duty cycle and the red back light at 20% duty cycle. Finally, in the high-beam position, the white headlight is on at 90% duty cycle and the red back light at 20% duty cycle.

Figure 9 shows the flowchart of the white and red LED brightness control. In this Timer 2 interrupt, there are 100 counts per period for an easy way to set the duty cycle. For example, a 20% duty cycle will need 20 counts out of the 100 count each period. This duty cycle is set in the pin change interrupt.

Figure 10 shows the flowchart of the interrupt that checks the battery every 30 minutes by using the Timer 2 overflow interrupt. The timer counts up to 30 minutes and once it overflows, the battery voltage gets checked, and the timer clears and starts counting again. The Timer 2 overflow interrupt only occurs when the microprocessor is in sleep mode.

Figure 8 shows the flowchart of the turning signal LEDs. In this Timer 0 interrupt, the duty cycle is set to 50% at a 1Hz frequency. When the user presses the left or right button, it triggers an interrupt and the respective LEDs will flash.

Figure 11 shows the pin change interrupt. This interrupt controls when the LEDs turn on or turn off based on switch position, buttons pressed, and the state of the hall-effect sensor.

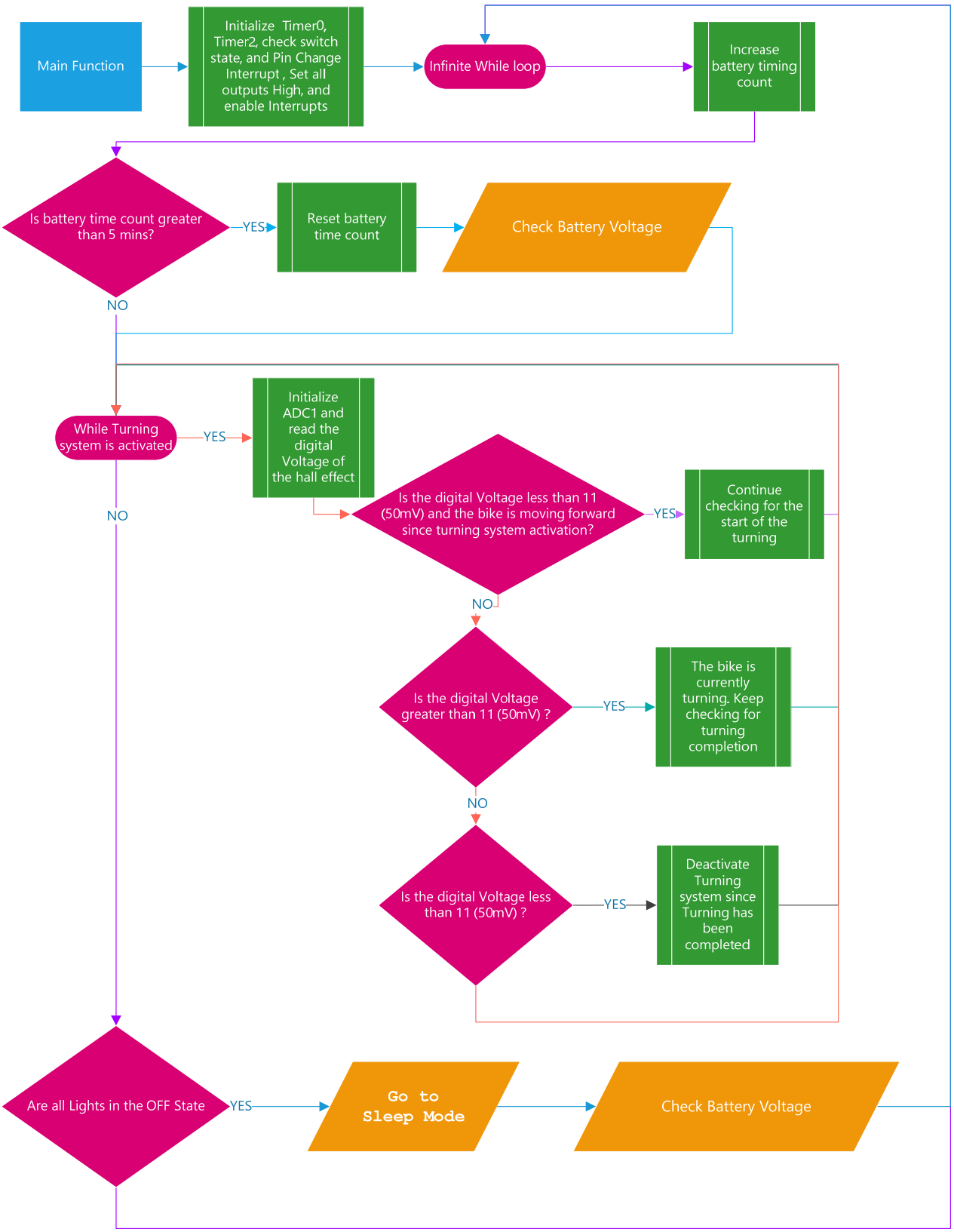


Figure 4: Flowchart of Main Function

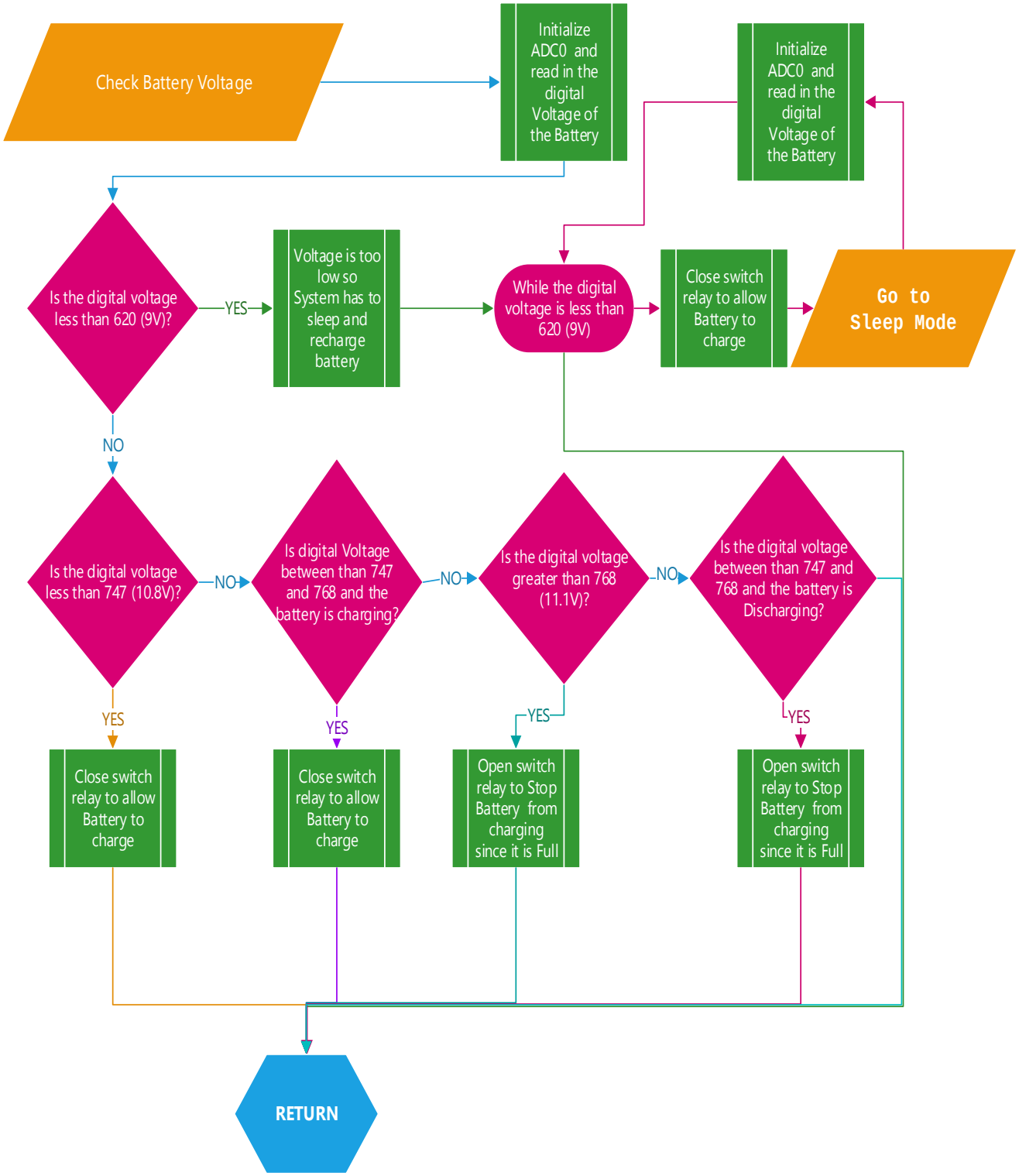


Figure 5: Flowchart of Battery Charge Control

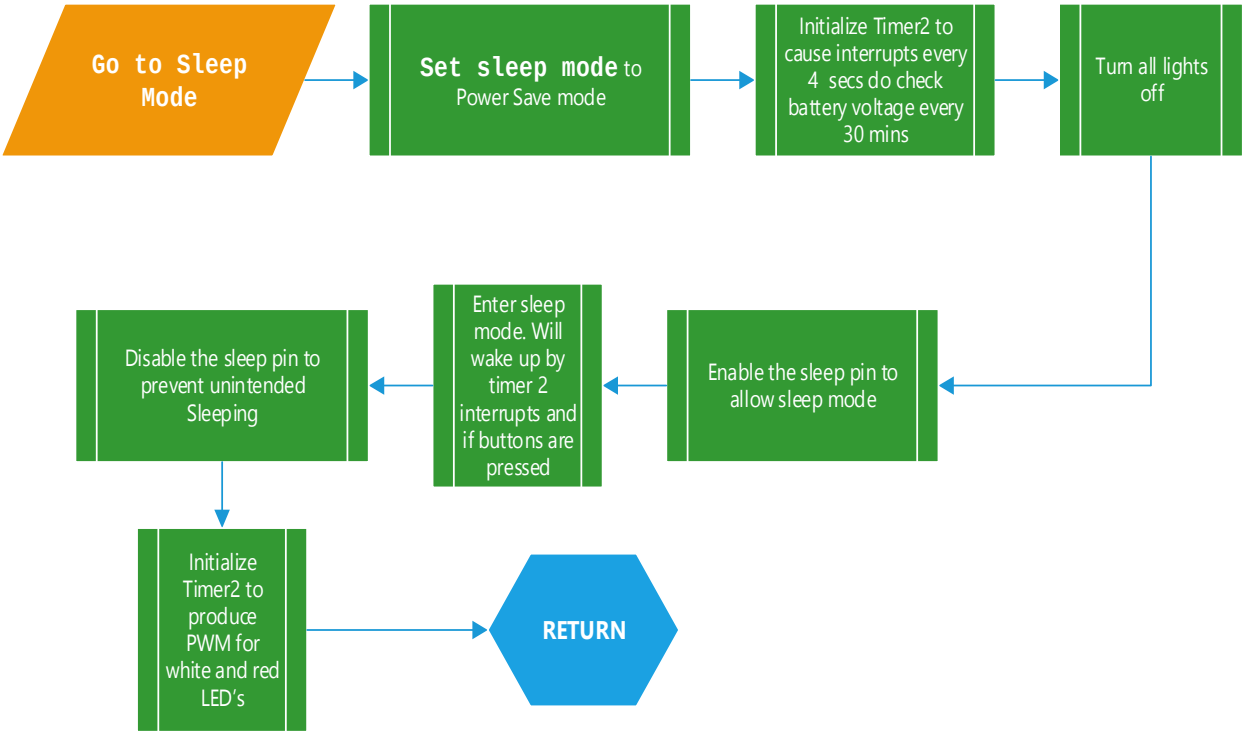


Figure 6: Flowchart of Sleep Mode

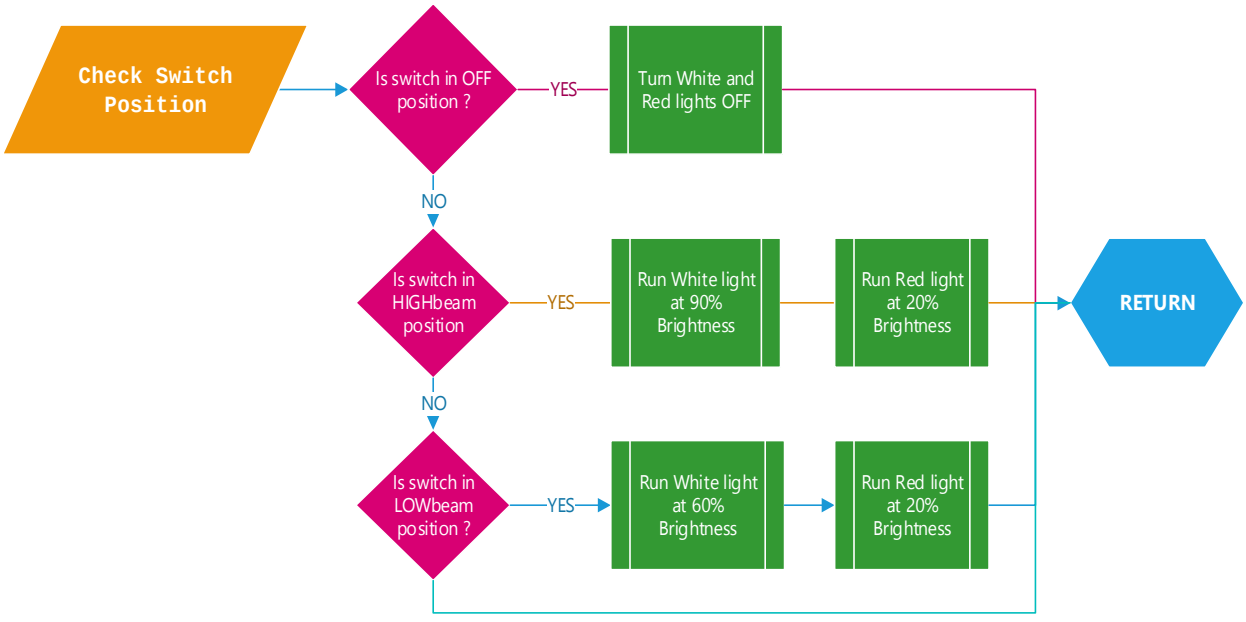


Figure 7: Flowchart of Switch Position Function

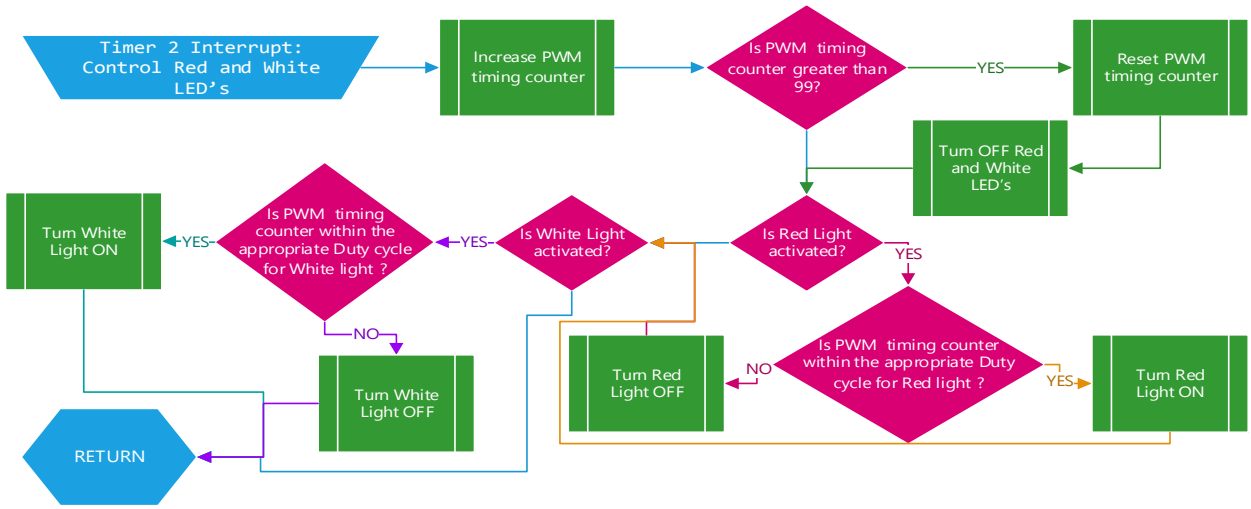


Figure 9: Flowchart of Red and White LEDs Interrupt

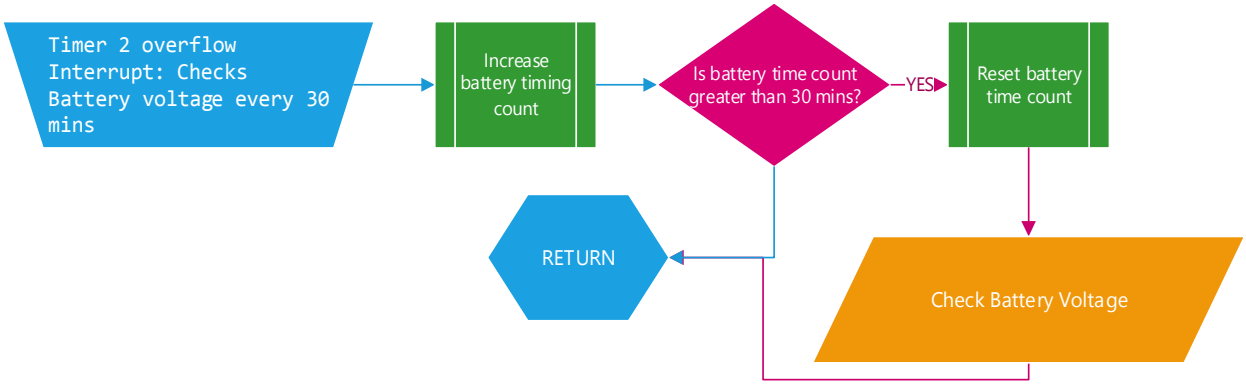


Figure 10: Flowchart of Battery Check Interrupt

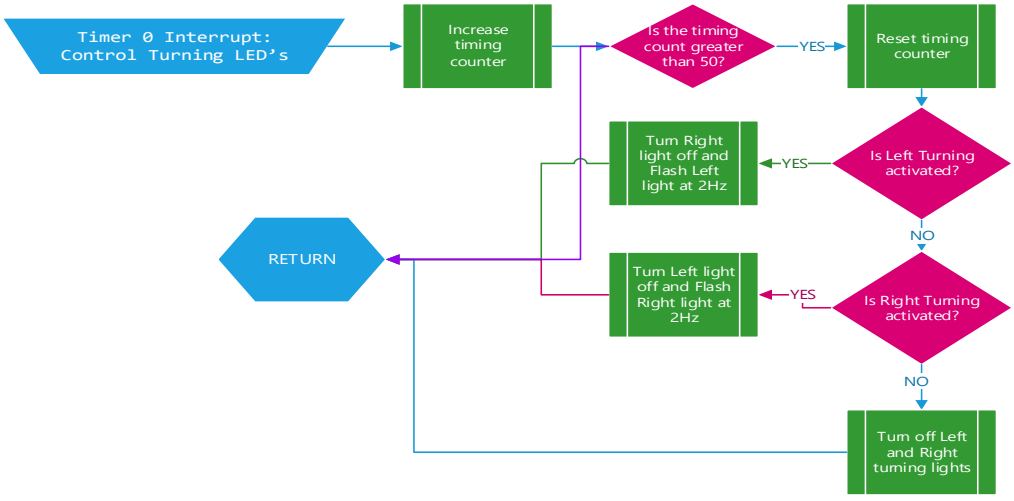


Figure 8: Flowchart of Turning Signals Interrupt

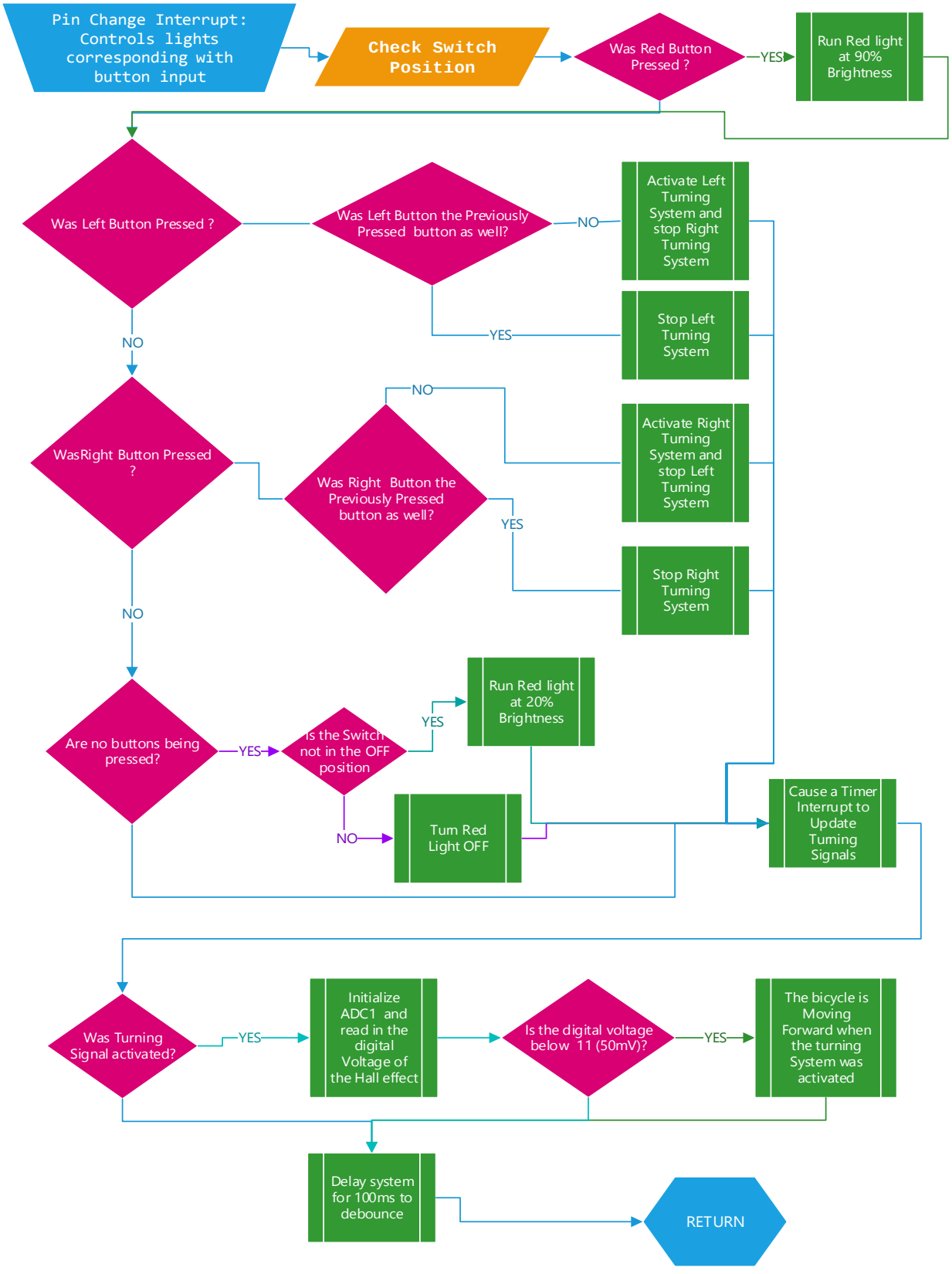


Figure 11: Flowchart of Pin Change Interrupt

Chapter 6. Construction

There are multiple components in this device that mounts directly to both the front and back of the bike. ***Figure 13*** shows the turning signal lights, the brake lights, the battery, and the solar panel. The battery is one of the heaviest components in this project so that the bulk of the weight gets mounted to the back of the bike to maintain stability when biking. The power unit attaches to the project box with velcros so that they can be taken off easily to charge elsewhere. The project box mounts to a basket on the back or right under the saddle with two bungee cords and the attached hooks on the project box sides, as shown in ***Figure 13*** and ***Figure 12***.

Figure 13: Final Product of Turning and Brake Lights

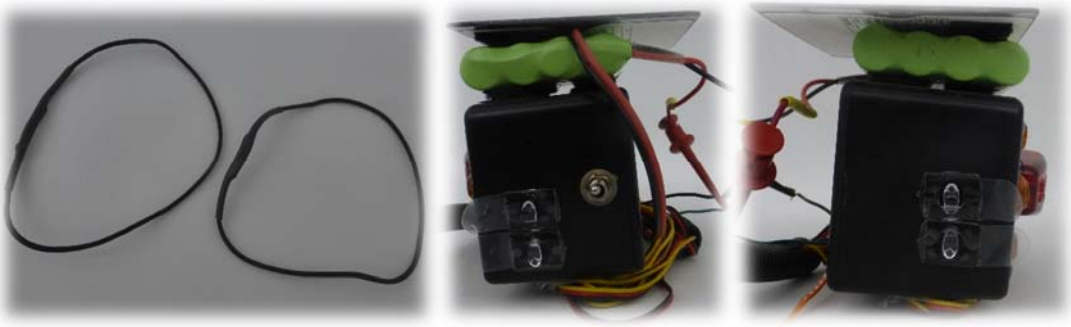


Figure 12: Final Product of securely mounting onto bike

The first constructed component was the back component of the project. First step is to drill holes the size of your LEDs so that the LEDs can be attached securely and screwed on snugly. The power and ground rail for the left and right turn LEDs connect to their respective sides so that the LEDs can be powered in sets of three as shown in ***Figure 14***. The switch for the white light and red light also attaches to this box.

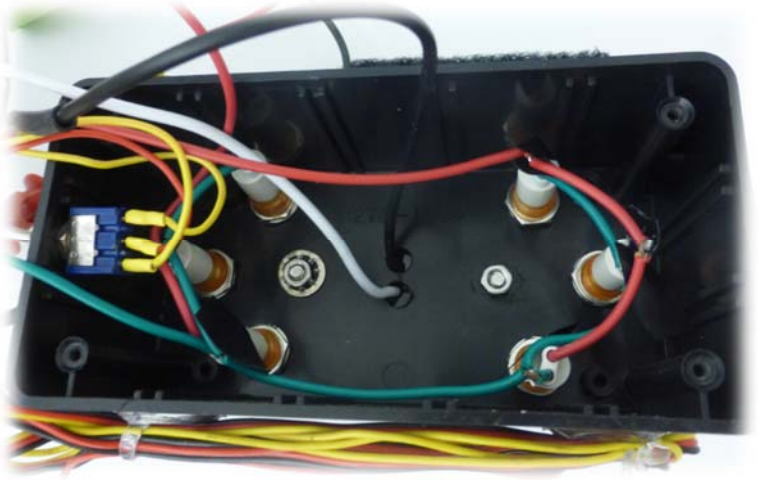


Figure 14: Turning and Brake Lights inside Project Box

Figure 15 shows the circuit board to control the LEDs; the layout to this circuit can be found in ***Figure 3***. ***Figure 16*** shows the circuit board that gets fitted inside the project box.

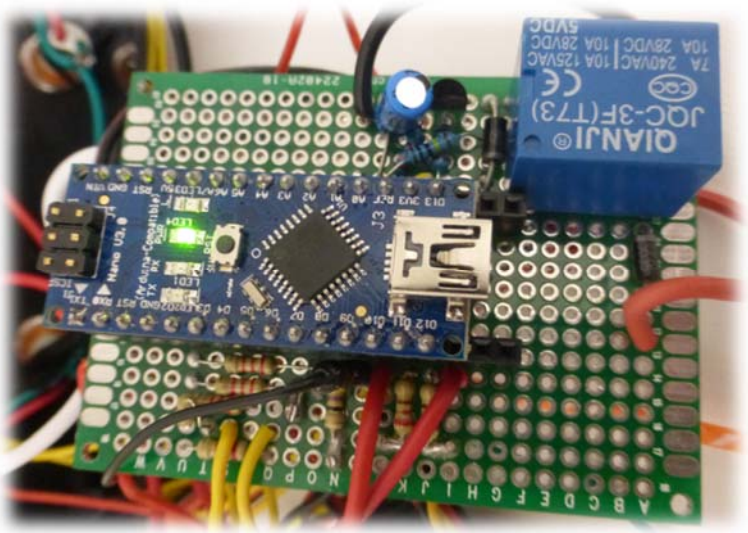


Figure 15: Circuit board of Bike Safe with Arduino Nano

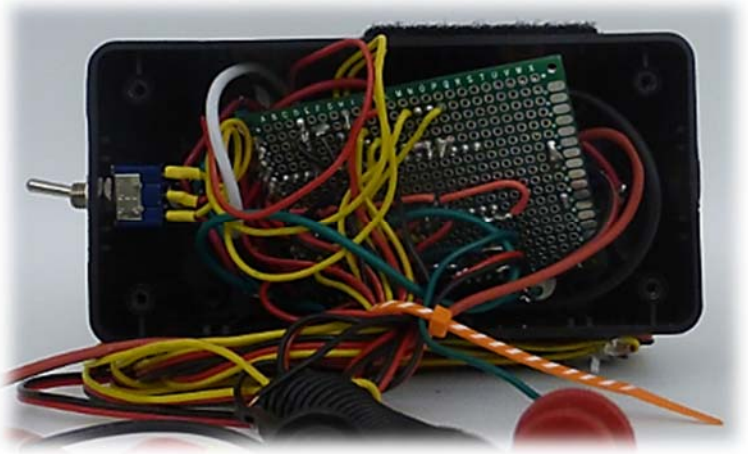


Figure 16: Complete inside of Project Box

In order for the turning signals to flash in the back, the user provides input with the buttons that are mounted to the front of the bike. ***Figure 17*** shows buttons placed near the handlebars so a bicyclist can push it without removing their hands from the handlebar. The headlight turns on with the switch on the back project box.

Figure 17: Final Product of Headlight and Turn Buttons

A hall-effect sensor mounts to front frame of the bike right below the pivot area of the handlebars and a small magnet mounted to the ring that attaches to the pivot area of the handlebar extends forward so that it is parallel to the hall-effect sensor when the wheels are straight, as shown in ***Figure 18***. This allows the hall-effect sensor to detect the magnet when it is in front signaling that a turn is made. The hall-effect stays in a fixed position while the magnet moves of the rotation of the handlebars.

Figure 18: Final Product of Headlight and Turn Buttons

The brake lights activates by special triggers located on the brake cable near the brakes as shown in ***Figure 19***. When the user presses the brake lever, the brake cable squeezes together to press the button. The left image of ***Figure 19*** shows the position of the button when not braking, while the right shows the button when braking.

Figure 19: Final Product of braking Buttons

Since there are many wires routing from the front to the back, the system uses wire wraps to route the wires along the frame of the bicycle. Zip ties prevent the wire wrap from moving out of place and endangering the bicyclist. The wire wrap makes the system more manageable and more appealing to the eyes.

Figure 20: Final Product of wire wrap

Chapter 7. Testing

Figure 21 thru ***Figure 26*** are the outputs of all the LEDs in the system. This confirms that the LEDs are flashing at the correct frequencies and duty cycle as specified.

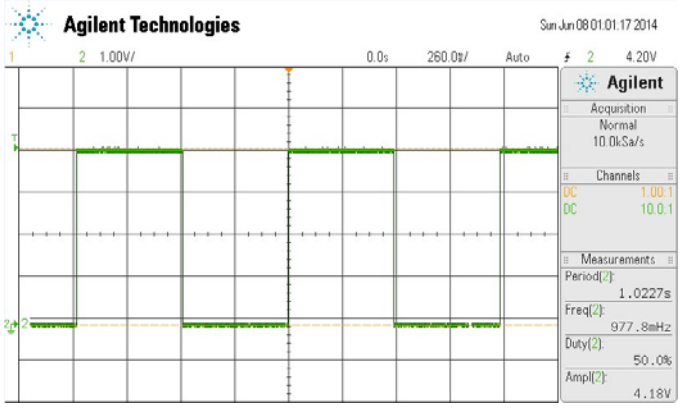


Figure 21: Left Turning Flashes at 1 Hz

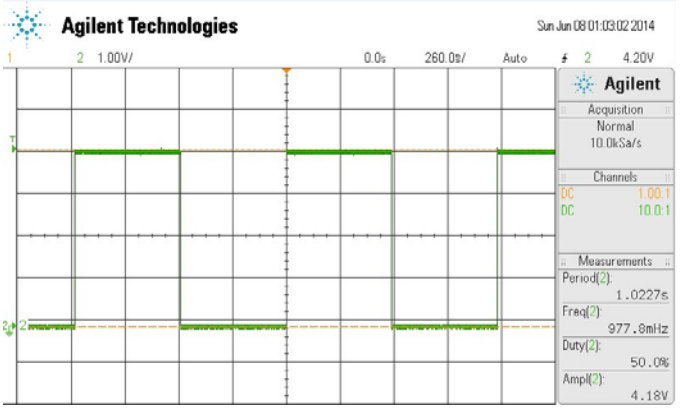


Figure 22: Right Turning Flashes at 1 Hz

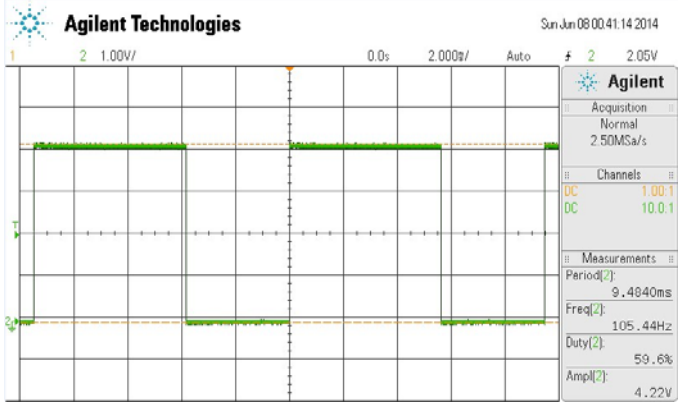


Figure 23: White Light Operates at 60% brightness when switch is in low-Beam position

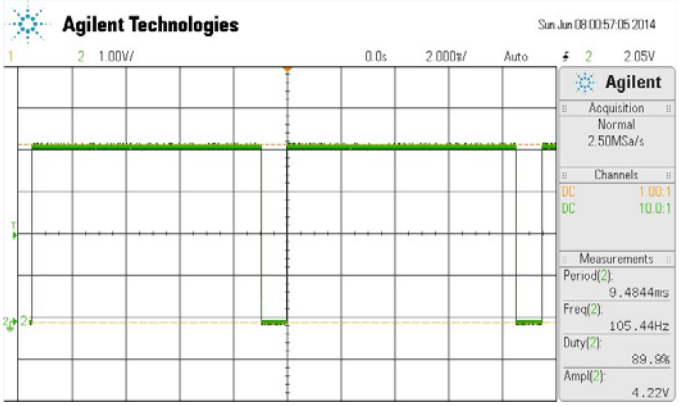


Figure 24: White Light Operates at 90% brightness when switch is in high-Beam position

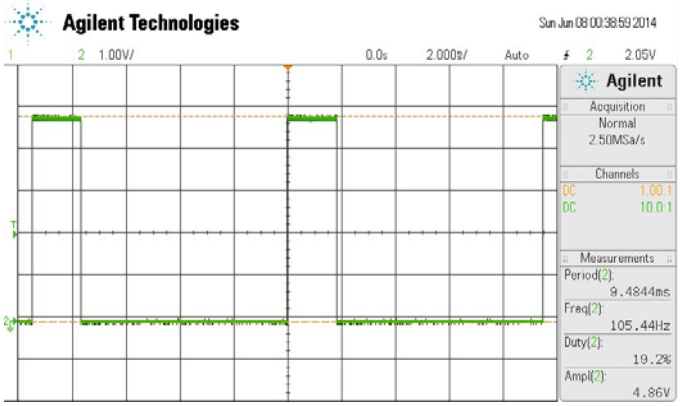


Figure 25: Red Light Operates at 20% brightness when switch is in low-Beam and High-Beam position

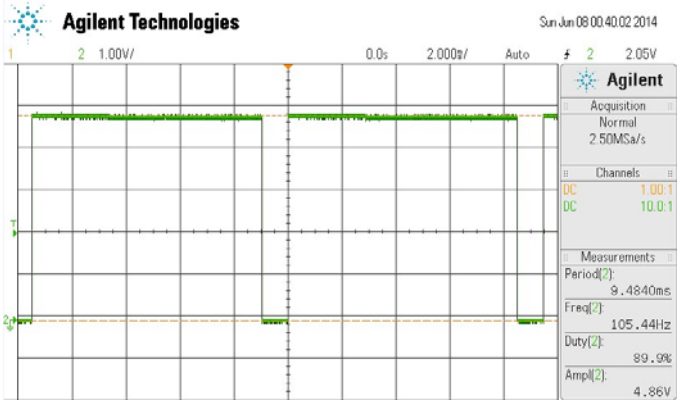


Figure 26: Red Light Operates at 90% brightness when brakes are applied

TABLE III shows the operational truth table of the Bike Safe device. The variables on the left of the ***TABLE III*** are the inputs from the user and the variables on the right of the ***TABLE III*** indicate which systems should be activated. Testing is done with these parameters and conditions resulted in 100%.

TABLE III

BIKE SAFE OPERATION TRUTH TABLE

Switch (off)	Switch (low-beam)	Switch (high-beam)	Left Button	Right Button	Brake Button	Hall-effect*	Left Lights	Right Lights	Red Back Lights	Red Brake Lights	White Lights
1	0	0	0	0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	0	0	0	1	0
0	1	0	0	0	0	1	0	0	0	0	1
0	1	0	1	0	0	0	1	0	0	0	1
0	1	0	0	1	0	0	0	1	0	0	1
0	1	0	0	0	0	0	0	0	0	0	1
0	1	0	0	0	1	1	0	0	0	1	1
0	0	1	0	0	0	1	0	0	0	0	1
0	0	1	1	0	0	0	1	0	0	0	1
0	0	1	0	1	0	0	0	1	0	0	1
0	0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	1	1	0	0	0	1	1

*Note: 0 when there is a magnetic field, 1 when there is not.

Figure 27 on the next page shows a collection of images of our successful testing. Images **(a)** and **(c)** successfully respectively indicate when the left and right turning signals are activated. When these indicators are off it represents either the turn has been completed or the user turned off the signal. When these lights are on the amber lights are also on to alert others of the impeding turn as shown in image **(g)** and **(k)** of ***Figure 27***. Images **(d)** show how the head light is powered on by only two wires while image **(b)** illustrates the working correctly when the switch is at low beam position. Image **(h)** is another view of the working head light when the switch is at high beam position. Images **(e)** and **(f)** demonstrate how easy it is to reposition the solar panel and

battery pack, in case of battery failure. Image **(i)** shows when the system is in sleep mode. Image **(j)** indicates that the switch is no longer in the off position and that the red light is dim when the white light is on. Image **(k)** shows the turning signal working when the white light is on. Finally, image **(l),** illustrates how brighter the red light turns when the brakes are applied.

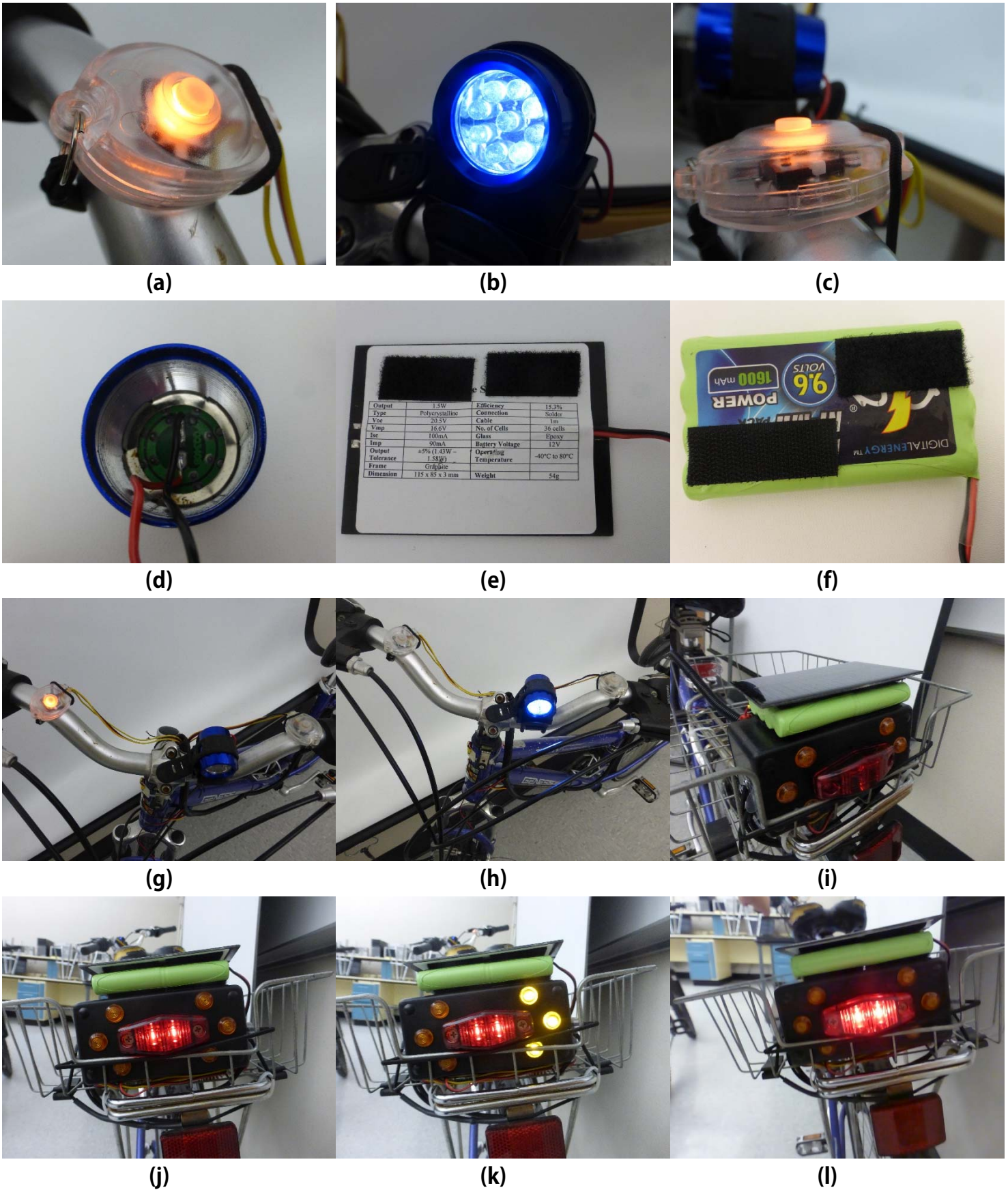


Figure 27: A collection of images illustrating successful testing

Future Improvements

This project has potential for improvement. The Arduino Nano in this system has a power indicator LED that uses 15mA. By removing this LED, we will save 15mA and will improve battery life. The project can also be made compact by using surface mount components on a custom PCB. By minimizing the PCB, the battery will fit inside the project enclosure. By adding a buzzer to the project, it will alert more motorist within the vicinity of the bicyclist providing a safer experience for the bicyclist.

Conclusion

Making this project happen has been a great learning experience, especially on time management and project planning. We were able to quickly get a prototype up and running on a breadboard, but the most time-consuming part of the project was the construction of the project. In order to make the project more permanent, we decided to transfer the prototype to a small PCB and have all the components of this project routed to this PCB. Since there was a lot of soldering involved and not a lot of room to work with on the PCB, mistakes were made, but in the end, we got the device up and running.

References

[1] Mari Lynch, *Be Cool Be Safe Bike Law Summary and Resources*, http://marilynch.com/blog/wp-content/uploads/Be-Cool-Be-Safe-Bike-Law-Summary-from-Bicycling-Monterey.pdf: BicyclingMonterey.com, Nov 2013.

[2] Pedestrian and Bicycle Information Center, *Pedestrian and Bicyclist Crash Statistics*, http://www.pedbikeinfo.org/data/factsheet_crash.cfm: Pedbikeinfo.org, Apr 2014.

[3] NHTSA's National Center for Statistics and Analysis, *2012 BICYCLISTS And OTHER CYCLISTS Traffic Safety Fact Sheet*, http://www-nrd.nhtsa.dot.gov/Pubs/812018.pdf: nhtsa.gov, Apr 2014.

[4] "Power Supply Matters," *tangentsoft.net*, para. 6, May 17, 2014. [Online]. Available: http://tangentsoft.net/audio/pimeta2/ps.html. [Accessed May 1, 2014].

[5] Atmel, "8-bit Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash," ATmega328p datasheet, Oct. 2010.

[6] Fairchild Semiconductor, "NPN General Purpose Amplifier," PN2222A datasheet, Aug. 2010.

[7] Langir, "PCB Relay," JQC-3F datasheet.

[8] Sentech, "AH201 Hall-Effect Switch Integrated Circuit," AH201 datasheet.

Appendix A. Senior Project Analysis

Project Title: Bike Safe

Student's Name: Betty Chan and Ricardo Duran **Student's Signature:**

Advisor's Name: Bryan Mealy **Advisor's Initials:** **Date:**

Summary of Functional Requirements

- **Describe the overall capabilities or functions of your project or design. Describe what your project does. (Do *not* describe how you designed it).**

  Bike Safe is a device that allows bicyclists to indicate that they are making a turn without having to take their hand off the handle bars. The system mounts to the seat post of the bicycle where the bicyclist controls the flashing amber LED lights from the handle bar. The system also has a headlamp system and a brake system where the red LEDs increase in brightness when the user applies the brakes.

Primary Constraints

- **Describe significant challenges or difficulties associated with your project or implementation. For example, what were limiting factors, or other issues that impacted your approach?**

  The goal is to produce a device that weighs no more than one pound and have dimensions less than 5" x 2.25" x 2.5" so it can be as lightweight and portable as possible. The amber turning lights should also flash at 1 Hz while the red brake lights brightens up when the user applies the brakes. A rechargeable battery powers the system and has a solar panel as backup.

- **What made your project difficult? What parameters or specifications limited your options or directed your approach?**

  It was difficult to construct this project in a more compact packaging because there are a lot of wires soldered on the circuit board that gets routed elsewhere. We needed a sufficient amount of space to construct the board with the wires so we could not make them shorter.

Economic

- **Human Capital**

  This device creates a safer environment for people to bike on the road and to provide a more universal signaling system for bikes. Bicyclists will not have to take their hands off the handle bars to signal on the road when making a turn. With this device, bicyclists will only have to press a button with their thumb, thus, creating a safer and more stable bike ride. This also provides a more obvious alert to motorists that are sharing the road.

- **Financial Capital**

  This product should be low-cost to produce and should also sell at an affordable price in the consumer market. At first, this device will probably have insignificant monetary gain, but later, production cost should decrease because the production labor will be faster and more efficient. After the initial prototype stage, the components should also be bought in bulk to keep component cost low.

- **Manufactured or Real Capital**

  Production cost will decrease when the laborers figure out a system where they can produce a device faster and more efficiently. The labor cost will remain the same while more products are being completed in the same amount of time.

- **Natural capital**

  This device is uses rechargeable batteries and a solar panel for power. The components and materials used to produce this device are RoHS compliant.

- **When and where do costs and benefits accrue throughout the project's lifecycle?**

  The cost will go into parts for creating prototypes and testing since different types of parts will be purchased to see which works best. After that stage, the money will go into labor, components, marketing, and honoring warranties.

- **What inputs does the experiment require? How much does the project cost? Who pays?**

  The device requires the bicyclist to push a button to determine which turn signal should turn on and to flick a switch to turn on the headlight and back light. It also requires pressure on the brakes to increase the brightness of the red brake lights. The device obtains power from a rechargeable battery and a solar panel. To prototype and test, the estimated cost is $200 because a higher quantity of what is needed is bought in order to account for mistakes such as frying LEDs and/or microprocessors. It also accounts for design changes. The students involved in this project pay first and then get reimbursed by the Senior Project Fund from the Cal Poly Electrical Engineering Department.

- **Actual final cost of component parts (at the end of your project)**

  At the end of the project, a list was compiled with all the components used and the cost was calculated to be $77.16.

- ***Attach is** TABLE IV **which is the final bill of materials for all components***

TABLE IV

BILL OF MATERIALS FOR PROJECT CONSTRUCTION

Components	Quantity	Price
Arduino Nano	1	$13.99

Red LED Trailer Light	1	$4.95
Amber Trailer Lights	6	$5.64
12V 1.5W Solar Panel	1	$13.98
5x7cm PCB	1	$1.00
Project Box	1	$5.49
Hall Effect Sensor	1	$0.60
Turtle Shaped Bike Lights	2	$1.99
LED Tactile Button - Orange	2	$3.90
9 LED flashlight	1	$2.99
Headers	1	$0.75
Various Resistors	14	$1.40
10uF Capacitor	1	$0.50
9.6V 1600maH Battery	1	$11.24
2N222 NPN	1	$0.25
1N4001 Diode	2	$0.50
SPDT Switch	1	$1.00
Wire Wrap		$6.99

- **Additional equipment costs (any equipment needed for development?)**

  A bicycle is needed for this project in order to figure out the best configuration and set up for the Bike Safe device. The students can provide this so no additional costs.

- **How much does the project earn? Who profits?**

  The project will not make a monetary profit. The profit will be the decrease amount of deaths due to improper signaling and not being seen on a bike.

- **Timing**

  - **When do products emerge? How long do products exist? What maintenance or operation costs exist?**

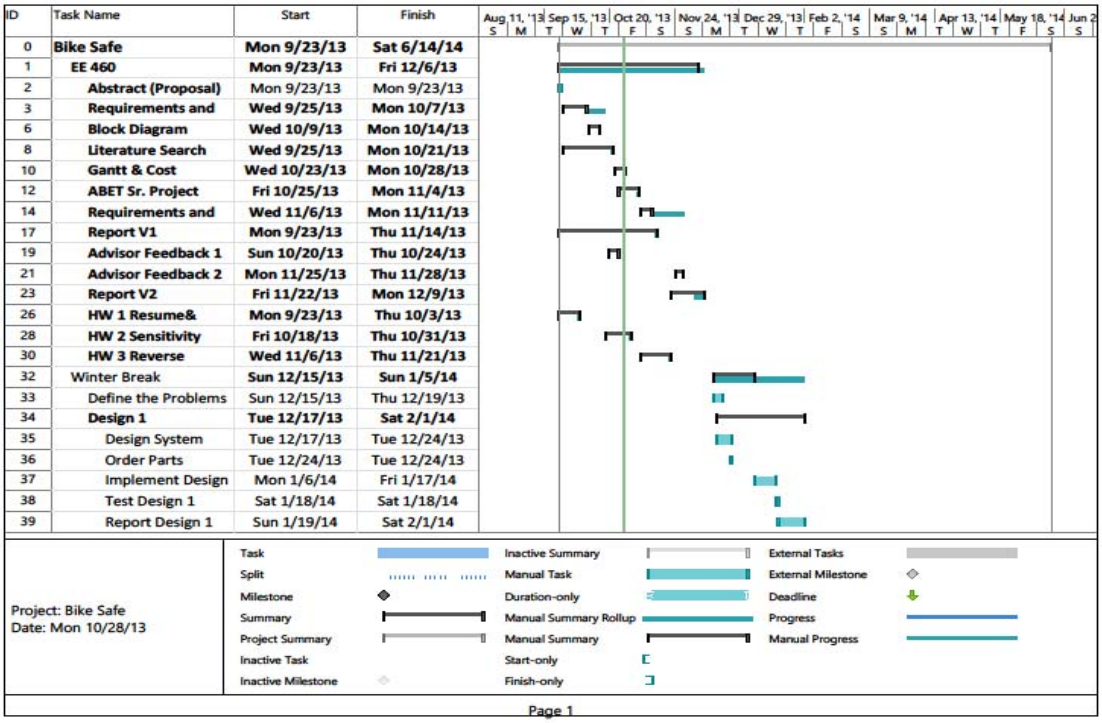
    The product will be ready by May 2014. Product will be kept on the bike until it fails. It will cost time to charge batteries because the solar panel is not able to fully recharge a drained battery.

  - **Original estimated development time (as of the start of your project), as Gantt or Pert chart**

    It will take three months to plan, two months to design, and three months to build.

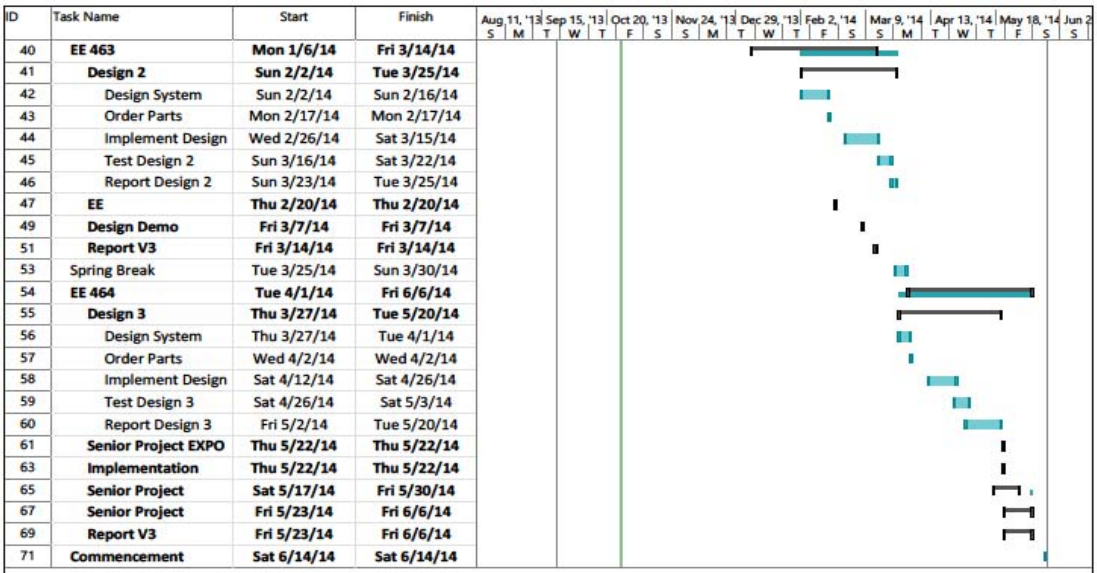
- **Actual development time (at the end of your project), as Gantt or Pert chart**

See next page.



ID	Task Name	Start	Finish
0	**Bike Safe**	**Mon 9/23/13**	**Sat 6/14/14**
1	**EE 460**	**Mon 9/23/13**	**Fri 12/6/13**
2	**Abstract (Proposal)**	Mon 9/23/13	Mon 9/23/13
3	**Requirements and**	**Wed 9/25/13**	**Mon 10/7/13**
6	**Block Diagram**	**Wed 10/9/13**	**Mon 10/14/13**
8	**Literature Search**	**Wed 9/25/13**	**Mon 10/21/13**
10	**Gantt & Cost**	**Wed 10/23/13**	**Mon 10/28/13**
12	**ABET Sr. Project**	**Fri 10/25/13**	**Mon 11/4/13**
14	**Requirements and**	**Wed 11/6/13**	**Mon 11/11/13**
17	**Report V1**	**Mon 9/23/13**	**Thu 11/14/13**
19	**Advisor Feedback 1**	**Sun 10/20/13**	**Thu 10/24/13**
21	**Advisor Feedback 2**	**Mon 11/25/13**	**Thu 11/28/13**
23	**Report V2**	**Fri 11/22/13**	**Mon 12/9/13**
26	**HW 1 Resume&**	**Mon 9/23/13**	**Thu 10/3/13**
28	**HW 2 Sensitivity**	**Fri 10/18/13**	**Thu 10/31/13**
30	**HW 3 Reverse**	**Wed 11/6/13**	**Thu 11/21/13**
32	Winter Break	**Sun 12/15/13**	**Sun 1/5/14**
33	Define the Problems	Sun 12/15/13	Thu 12/19/13
34	**Design 1**	**Tue 12/17/13**	**Sat 2/1/14**
35	Design System	Tue 12/17/13	Tue 12/24/13
36	Order Parts	Tue 12/24/13	Tue 12/24/13
37	Implement Design	Mon 1/6/14	Fri 1/17/14
38	Test Design 1	Sat 1/18/14	Sat 1/18/14
39	Report Design 1	Sun 1/19/14	Sat 2/1/14

Page 1



ID	Task Name	Start	Finish
40	**EE 463**	**Mon 1/6/14**	**Fri 3/14/14**
41	**Design 2**	**Sun 2/2/14**	**Tue 3/25/14**
42	Design System	Sun 2/2/14	Sun 2/16/14
43	Order Parts	Mon 2/17/14	Mon 2/17/14
44	Implement Design	Wed 2/26/14	Sat 3/15/14
45	Test Design 2	Sun 3/16/14	Sat 3/22/14
46	Report Design 2	Sun 3/23/14	Tue 3/25/14
47	**EE**	**Thu 2/20/14**	**Thu 2/20/14**
49	**Design Demo**	**Fri 3/7/14**	**Fri 3/7/14**
51	**Report V3**	**Fri 3/14/14**	**Fri 3/14/14**
53	Spring Break	Tue 3/25/14	Sun 3/30/14
54	**EE 464**	**Tue 4/1/14**	**Fri 6/6/14**
55	**Design 3**	**Thu 3/27/14**	**Tue 5/20/14**
56	Design System	Thu 3/27/14	Tue 4/1/14
57	Order Parts	Wed 4/2/14	Wed 4/2/14
58	Implement Design	Sat 4/12/14	Sat 4/26/14
59	Test Design 3	Sat 4/26/14	Sat 5/3/14
60	Report Design 3	Fri 5/2/14	Tue 5/20/14
61	**Senior Project EXPO**	**Thu 5/22/14**	**Thu 5/22/14**
63	**Implementation**	**Thu 5/22/14**	**Thu 5/22/14**
65	**Senior Project**	**Sat 5/17/14**	**Fri 5/30/14**
67	**Senior Project**	**Fri 5/23/14**	**Fri 6/6/14**
69	**Report V3**	**Fri 5/23/14**	**Fri 6/6/14**
71	**Commencement**	**Sat 6/14/14**	**Sat 6/14/14**

Page 2

- **What happens after the project ends?**

  The project will be implemented on additional bikes for additional testing and to obtain reviews for improvement.

If manufactured on a commercial basis:

- **Estimated number of devices sold per year**

  The estimated numbers of device sold per year is about 10,000 devices.

- **Estimated manufacturing cost for each device**

  The estimated manufactured cost for each device is about $40.

- **Estimated purchase price for each device**

  The estimated purchase price for each device is $50.

- **Estimated profit per year**

  The estimated profit per year (excluding labor cost) is estimated to be $100,000. Assuming that there are 2 workers paid $10 per hour to produce 10,000 devices a year, the profit is about $60,000 per year. This excludes other factors.

- **Estimated cost for user to operate device, per unit time (specify time interval)**

  It does not cost the user anything to operate the device.

Environmental

- **Describe any environmental impacts associated with manufacturing or use, explain where they occur and quantify.**

This project involves soldering and solder contains lead and create fumes. This may be a health hazard.

- **Which natural resources and ecosystem services does the project use directly and indirectly?**

  This project uses the sunlight for the solar panel.

- **Which natural resources and ecosystem services does the project improve or harm?**

  The project does not improve or harm any natural resources and ecosystem unless it is trashed in an improper way.

- **How does the project impact other species?**

  There will be a decrease of deaths due to bike accidents on the road. Motorist will be able to predict the bicyclist's movements more easily.

Manufacturability

- **Describe any issues or challenges associated with manufacturing.**

  This device will most likely be built by hands and not machines so it will mass-produce at a much slower pace than with machines. It requires strong soldering skills and some programming knowledge to troubleshoot and to implement the code on the microprocessor.

Sustainability

- **Describe any issues or challenges associated with maintaining the completed device, or system.**

  There will not be much maintenance to do after the device is mounted on the bike. The only maintenance that needs to be done is to recharge the batteries after a few hours of use.

- **Describe how the project impacts the sustainable use of resources.**

  The battery is rechargeable so it does not need replacing each time it runs out of charge. There is also a solar panel that slowly charges the battery with sunlight.

- **Describe any upgrades that would improve the design of the project.**

  The turning signals can be implemented and incorporated into the handle bars to alert motorist coming from the front, but the battery will drain faster.

- **Describe any issues or challenges associated with upgrading the design.**

  The battery will drain faster because it will be using more power for the extra feature.

Ethical

- **Describe ethical implications relating to the design, manufacture, use, or misuse of the project.**

  Analyze using one or more ethical frameworks in addition to the IEEE Code of Ethics.

  Like any electronic product, the battery has a chance of blowing up if something gets shorted. Since this device is implemented on a bike with moving parts, wires have a higher chance of breaking or moving to a place that it is not intended to go to. To prevent this from happening, all electrical contacts will be insulated and grounded.

Health and Safety

- **Describe any health and safety concerns associated with design, manufacture or use of the project.**

  The purpose of this device is to allow the biker to be better seen and to be more predictable, but the user also has to watch out for other motorist on the road. With this device, some users

may think they are safe to make a turn by only using this device and neglect to look over their shoulders. This can be a huge safety concern.

Social and Political

- **Describe social and political issues associated with design, manufacture, and use.**

  The more people see and use this device, the more they would want it to be a standard safety feature on bicycles.

- **Who does the project impact? Who are the direct and indirect stakeholders?**

  The project directly impacts the bicyclists who prefer to share the road with fast-moving vehicles because it can be the thing that saves their lives. The project indirectly impacts other drivers because as a stakeholder, they do not want to accidentally run over a bicyclist and get sued or have that on their conscience.

- **To what extent do stakeholders benefit equally? Pay equally? Does the project create any inequities?**

  The stakeholders (drivers) will be able to see the bicyclist earlier so they have less of a chance of accidentally running the bicyclist over. This can help them avoid jail time and being sued.

- **Consider various stakeholders' locations, communities, access to resources, economic power, knowledge, skills, and political power.**

  Bike shops can benefit from this product by providing a deal on this product with a purchase of a bike. The manufacturer can make a small profit. The biking community will have more members. There will be more people biking because it is safer with this device and this benefits the environment due to less pollution.

Development

- **Describe any new tools or techniques, used for either development or analysis that you learned independently during the course of your project.**

  For this project, we obtained a Red Tag for the machine shop and learned how to some of the equipment there. We used this resource to create holes in the project box.

- **Include a literature search.**

  See References on Page 28.

Appendix B. Code

```
/************************************************
//                  Bike Safe main c file
/************************************************
/*
* BikeSafe.c
*
* Created: 5/11/2014 12:25:03 AM
*  Author: Ricardo and Betty
*/

#define F_CPU 16000000

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <stdio.h>
#include "UART.h"
#include "ADC.h"

#define LEFT 7 // left Button
#define RIGHT 6 // Right Button
#define WHITElow 5 // white switch one
#define RED 4 //  Red Button
#define WHITEhigh 3 // white switch two

/*voltage divider*/
#define R1 1000000 //1 Mega Ohm
#define R2 510000 // 510 kOhm

/*ADC readings*/
#define Volt_9 620 // ADC 9V 622
#define Volt_10_8 747 // ADC 10.8V
#define Volt_11_1 768 // ADC 11.1V

/*red and white light brightness*/
#define lowPWM 20
#define midPWM 60
#define highPWM 90

/*timing*/
#define oneHour 219726
#define oneMin 3662

void initPCINT2(void);
void initTIMER0(void);
void initTIMER2(void);
void initTIMERBat(void);
void checkBatVoltage(void);
void sleepNow(void);
void checkWHITE(void);

int time = 0, mode;
int pushFlag = 0, redFlag = 0, whiteFlag = 0, whitePWM = 0, redPWM = 0, batFlag = 0;;
int magnetLoc = 0;
int myCounter = 0;
unsigned long batTime = 0L;

char buffer [50];
int main(void)
{

          DDRD = (0 << PIND7) | (0 << PIND6)| (0 << PIND5)| (0 << PIND4)| (0 << PIND3);
          DDRB = (1 << PINB4) | (1 << PINB3)|(1 << PINB0) | (1 << PINB1)|(1<<PINB5);
          usart_init(9600, F_CPU);
          initPCINT2();

          initTIMER0();
          initTIMER2();

          checkWHITE();
          /*Check Bat Voltage*/
          //checkBatVoltage();

          sei();

          while(1)
          {
                    /*Check battery every 5 mins*/
                    batTime++;
                    if(batTime>=100000000)
                    {
                              batTime = 0;
                              checkBatVoltage();
                    }

                    /*enter loop of turning signal activated*/
                    while (pushFlag>0)
                    {
                              //putty_print("on\n\r");
```

```
                                /*magnet aDC*/
                                Initialize_ADC1();
                                convertADC();
                                //sprintf (buffer, "Voltage = %d:\r\nmagnetLoc = %d:\r\n", voltage,magnetLoc);
                                //putty_print(buffer);

                                if ((magnetLoc == 1)&&(voltage<11))
                                {
                                        /*Moving forward when turning signal is on */
                                        magnetLoc = 1;
                                }
                                else if (voltage > 11)
                                {
                                        /*currently turning */
                                        magnetLoc = 0;
                                }
                                else if (voltage < 11) // For [0,1] volts, output 1 Hz
                                {
                                        /*turning Completed, turn signal off */
                                        pushFlag = 0;
                                }
                        }

                        if ((pushFlag == 0)&&(whiteFlag == 0)&&(redFlag == 0))
                        {
                                sleepNow();
                                checkBatVoltage();
                        }
                }
}

void checkBatVoltage()
{
        Initialize_ADC0();
        mean_volt = 0;
        for (int i=0; i<20; i++)// average 20 samples
        {
                convertADC();
                mean_volt +=voltage;
                sprintf (buffer, "i= %d:    V=:%d\r\n", i,voltage);
                putty_print(buffer);
        }
        mean_volt/=20;

        sprintf (buffer, "Voltage = %d:\r\nMEAN Voltage = %d:\r\n", voltage,mean_volt);
        putty_print(buffer);
        voltage = mean_volt;


        if (voltage < Volt_9) // bat is too low at 9V
        {
                batFlag = 2;
                while(voltage < Volt_9)
                {
                        // Starts charging and goes to sleep until a pin change occurs and the battery charge is over 9V
                        PORTB |= (1<<PINB5);
                        sleepNow();

                        putty_print("voltage Below: 9V\r\n");

                        Initialize_ADC0();
                        mean_volt = 0;
                        for (int i=0; i<20; i++)// average 20 samples

                        {
                                convertADC();
                                mean_volt +=voltage;
                                sprintf (buffer, "i= %d:\r\n", i);
                                putty_print(buffer);
                        }
                        mean_volt/=20;
                        sprintf (buffer, "Voltage = %d:\r\nMEAN Voltage = %d:\r\n", voltage,mean_volt);
                        putty_print(buffer);
                        voltage = mean_volt;


                }
        } else if (voltage <Volt_10_8)// bat has reached below 10.8V
        {
                putty_print("Battery is Charging.\r\n");
                batFlag = 1;
                PORTB |= (1<<PINB5); // battery is allowed to charge

        }

        else if ((voltage> (Volt_10_8 - 1)) && (voltage < Volt_11_1) && batFlag==1)
        {
                // bat is between 10.8V and 11.1V  and is charging
                putty_print("Battery is Charging.\r\n");

                PORTB |= (1<<PINB5); // battery is allowed to charge
        }
        else if (voltage > (Volt_11_1 - 1))// bat is 11.1 or above
        {
```

```
                    putty_print("Battery is Charged.\r\n");
                    batFlag = 0;
                    PORTB &= ~(1<<PINB5);
                    //mode =0;
          }
          else if ((voltage > (Volt_10_8 - 1)) && (voltage < Volt_11_1) && batFlag == 0)
          {
                    // bat is between 10.8V and 11.1V  and is discharging
                    putty_print("Battery is discharging.\r\n");
                    PORTB &= ~(1<<PINB5);
          }
          //Initialize_ADC1();
}

void sleepNow()
{
          set_sleep_mode(SLEEP_MODE_PWR_SAVE);   // sleep mode is set here
          initTIMERBat(); // sets up timer to check battery every 30 mins

          sprintf (buffer, "SLEEP MODE _______%d________\r\n", mode);
          putty_print(buffer);

          // resets flags and turns all lights off
          pushFlag = 0, redFlag = 0, whiteFlag = 0, whitePWM = 0, redPWM = 0;
          PORTB &= ~((1 << PINB3)|(1 << PINB4)|(1 << PINB0)|(1 << PINB1));

          sleep_enable();          // enables the sleep bit in the mcucr register
          // so sleep is possible. just a safety pin

          putty_print("SLEEP MODE ON:\r\n");

          sleep_mode();            // here the device is actually put to sleep!!
          // THE PROGRAM CONTINUES FROM HERE AFTER WAKING UP

          sleep_disable();         // first thing after waking from sleep:
          putty_print("SLEEP MODE OFF:\r\n");
          // disable sleep...

          initTIMER2();// sets up timer to control red and White light brightness
}

void initTIMERBat()
{
          TCCR2A = (0 << WGM21);  //Set Normal Mode
          OCR2A = 5;   //Calculated from http://eleccelerator.com/avr-timer-calculator/
          TIMSK2 = (1 << OCIE2A);
          TCCR2B = (1 << CS22) | (1 << CS21)| (1 << CS20);            //Set prescalar to clk/1024

}

void initTIMER0(void)
{
          TCCR0A = (1 << WGM01);  //Set CTC
          OCR0A = 156;   //Calculated from http://eleccelerator.com/avr-timer-calculator/
          TIMSK0 = (1 << OCIE0A);
          TCCR0B = (1 << CS02) | (1 << CS00);               //Set prescalar to 1024
}

void initTIMER2(void)
{
          TCCR2A = (1 << WGM21);  //Set CTC
          OCR2A = 5;   //Calculated from http://eleccelerator.com/avr-timer-calculator/
          TIMSK2 = (1 << OCIE2A);
          TCCR2B = (1 << CS22) | (1 << CS21);               //Set prescalar to 256
}

void initPCINT2(void)
{
          //PCIFR = (1 << PCIF2); //triggerd in portD
          PCICR = (1 << PCIE2);         //Enable PCINT22..23 (PD6 and PD7&PD4 and PD5 and PD3)
          PCMSK2 = (1 << PCINT22) | (1 << PCINT23)|(1 << PCINT20) | (1 << PCINT21)| (1 << PCINT19);
}

ISR(TIMER0_COMPA_vect)
{
          time++;
          if(time > 50)
          {
                    time = 0;
                    if (pushFlag == 1)
                    {
                              /*Right Light on*/
                              PORTB ^= (1 << PINB4);
                              PORTB &= ~(1 << PINB3);
                    }
                    else if (pushFlag == 2)
                    {
                              /*Left Light on*/
                              PORTB ^= (1 << PINB3);
                              PORTB &= ~(1 << PINB4);
                    }
                    else
                    {
                              /*turning lights off*/
                              PORTB &= ~((1 << PINB3)|(1 << PINB4));
```

```
                }
        }
}

ISR(TIMER2_OVF_vect)
{
        batTime++;
        if(batTime>610)
        {
                batTime = 0;
                checkBatVoltage();
                //_delay_ms(5000);
        }

}

ISR(TIMER2_COMPA_vect)
{
        /*Sets up PWM for red and white light brightness*/
        myCounter++;

        if (myCounter > 99){
                /* when 100 is reached , reset counter and turn off red and white lights*/
                myCounter = 0;
                PORTB &= ~((1<<PINB1)|(1<<PINB0));
        }
        if(redFlag >  0 ) {
                /*red light On*/
                if (myCounter > redPWM){
                        PORTB |= (1<<PINB1);
                }
                else{
                        PORTB &= ~((1<<PINB1));
                }
        }


        if(whiteFlag>0 ){
                /*white light On*/
                if (myCounter > whitePWM){
                        PORTB |= (1<<PINB0);
                }
                else{
                        PORTB &= ~((1<<PINB0));
                }
        }

}

ISR(PCINT2_vect)
{
        checkWHITE();

        //check red
        if((PIND & (1<< RED))&&(PIND | ~(1<<LEFT))&&(PIND | ~(1<<RIGHT)))
        {
                putty_print("RED:\r\n");
                redFlag =  2;
                redPWM = 99 - highPWM;

        }

        else if((PIND & (1<<LEFT))&&(PIND | ~(1<<RIGHT))&&(PIND | ~(1<<RED)))
        {
                //check left
                putty_print("LEFT:\r\n");
                switch(pushFlag){
                        case 0:
                        /*left Light on*/
                        pushFlag = 1;
                        break;
                        case 1:
                        /*left Light off*/
                        pushFlag = 0;
                        break;
                        case 2:
                        /*left Light on and right light off*/
                        pushFlag = 1;
                        break;
                        default:
                        pushFlag = 0;
                        break;
                }

        }


        else if((PIND | ~(1<<LEFT))&&(PIND & (1<<RIGHT))&&(PIND | ~(1<<RED)))
        {
                //check right
                putty_print("RIGHT:\r\n");
                switch(pushFlag){
                        case 0:
                        /*right Light on */
                        pushFlag = 2;
```

```
                                break;
                                case 1:
                                /*right Light on and left light off*/
                                pushFlag = 2;
                                break;
                                case 2:
                                /*right light off*/
                                pushFlag = 0;
                                break;
                                default:
                                pushFlag = 0;
                                break;
                        }
                }


                else if((PIND | ~(1<<RIGHT))&&(PIND | ~(1<<LEFT))&&(PIND | ~(1<<RED)))
                {
                        putty_print("depressed:\r\n");
                        if(whiteFlag > 0)
                        {
                                redFlag = 1;
                                redPWM = 99 - lowPWM;
                        }
                        else
                        redFlag =0;
                }

                TIFR0 |= (1<<OCF0A);// update lights

                switch (pushFlag)
                {
                        case 0:
                        putty_print("0: OFF turning\r\n");
                        break;
                        case 1:
                        putty_print("1: LEFT turning\r\n");
                        break;
                        case 2:
                        putty_print("2: RIGHT turning\r\n");
                        break;
                        default:
                        break;
                }


                if (pushFlag>0)
                {
                        Initialize_ADC1();
                        convertADC();
                        if (voltage<11)
                        {
                                magnetLoc = 1;
                        }
                }
                _delay_ms(100);

}

void checkWHITE(void)
{
                putty_print("WHITE:\r\n");
                if((PIND & (1<<WHITElow))&&(PIND | ~(1<<WHITEhigh))) // OFF
                {
                        whiteFlag = 0;
                        redFlag = 0;
                        putty_print("OFF\r\n");
                }

                else if((PIND & (1<<WHITEhigh))&&(PIND | ~(1<<WHITElow))) // High WHite
                {
                        whiteFlag = 2;
                        whitePWM = 99-highPWM;
                        redFlag = 1;
                        redPWM = 99 - lowPWM;
                        putty_print("High:\r\n");
                }

                else if ((PIND | ~(1<<WHITElow))&&(PIND | ~(1<<WHITEhigh))) // dim WHIte
                {
                        whiteFlag = 1;
                        whitePWM = 99-midPWM;
                        redFlag = 1;
                        redPWM = 99 - lowPWM;
                        putty_print("Dim\r\n");
                }
                else
                {
                        whiteFlag = 0;
                        redFlag = 0;
                        putty_print("ELSE OFF\r\n");
                }

}
```

```
/************************************************
//                  ADC Header file
/************************************************

/*
* ADC.h
*
* Created: 5/14/2014 8:41:48 PM
*  Author: Ricardo and Betty
*/


#ifndef ADC_H_
#define ADC_H_
int voltage, oldVoltage, mean_volt;

//this function allows the hall effect sensor to be read by the adc
void Initialize_ADC1(void);

//this function allows the battery voltage to be read by the adc
void Initialize_ADC0(void);

//the convert function converts the analog voltage into  a digital representation of the voltage
void convertADC(void);


#endif /* ADC_H_ */


/************************************************
//                  ADC C file
/************************************************
/*
* ADC.c
*
* Created: 5/14/2014 8:41:17 PM
*  Author: Betty and Ricardo
*/
#define F_CPU 16000000
#include <stdlib.h>                                     // Standard C library
#include <avr/io.h>                                     // Input-output ports, special registers
#include <util/delay.h>
#include "ADC.h"

void Initialize_ADC1(void)
{
          ADCSRA = 0x87;      //Turn On ADC and set prescaler (CLK/128)
          //ADCSRB = 0x00;    //Set gain & turn off autotrigger
          ADCSRB = 0x00;      //Set gain & turn off on compare match
          ADMUX = 0x01;       //Set ADC channel ADC0 with 1X gain
          voltage = 1023;
}

void Initialize_ADC0(void)
{
          ADCSRA = 0x87;    //Turn On ADC and set prescaler (CLK/128)
          ADCSRB = 0x00;    //Set gain & turn off autotrigger
          ADMUX = 0x00;        //Set ADC channel ADC0 with 1X gain
}

void convertADC(void){
          ADCSRA = 0xC7;                        // start conversion
          _delay_us(260);                       // ensure max sampling rate not exceeded__260
          voltage = ADC & 0x3FF;                // read voltage
}


/************************************************
//                  UART Header file
/************************************************
/*
* UART.h
*
* Created: 4/20/2014 10:40:28 PM
*  Author: rad
*/


#ifndef UART_H_
#define UART_H_


#define BAUD_PRESCALE 103


/* function sends strings of words to the uart by one character at a time*/
void putty_print(char words[]);

/* function setups the communication protocols the uart will transmit at.*/
void usart_init(uint16_t baudin, uint32_t clk_speedin);
```

```
//the send function will put 8bits on the trans line
void usart_send( uint8_t data ) ;

/* the receive data function. Note that this a blocking call
Therefore you may not get control back after this is called
until a much later time. It may be helpfull to use the
istheredata() function to check before calling this function
@return 8bit data packet from sender
*/
uint8_t  usart_recv(void);

/* function check to see if there is data to be received
@return true is there is data ready to be read */

uint8_t  usart_istheredata(void);



#endif /* UART_H_ */


/************************************************
//                  UART c file
/************************************************
/*
* UART.c
*
* Created: 4/20/2014 10:35:06 PM
*  Author: Betty and Ricardo */

#include <stdlib.h>                                     // Standard C library
#include <avr/io.h>                                     // Input-output ports, special registers
#include "UART.h"
#include <string.h>

#define BAUD_PRESCALE 103
void usart_init(uint16_t baudin, uint32_t clk_speedin)
{
          uint32_t ubrr = (clk_speedin/16UL)/baudin-1;
          UBRR0H = (unsigned char)(ubrr>>8);
          UBRR0L = (unsigned char)ubrr;
          /*UBRR0H = (BAUD_PRESCALE>>8);
          UBRR0L = BAUD_PRESCALE;*/
          /* Enable receiver and transmitter */
          UCSR0B = (1<<RXEN0)|(1<<TXEN0);
          /* Set frame format: 8data, 1stop bit */
          UCSR0C = (1<<USBS0)|(3<<UCSZ00);
          UCSR0A &= ~(1<<U2X0);
}

/*the send function will put 8bits on the trans line. */
void usart_send( uint8_t data )
{
          /* Wait for empty transmit buffer */
          while ( !( UCSR0A & (1<<UDRE0)) );
          /* Put data into buffer, sends the data */
          UDR0 = data;
}

/* the receive data function. Note that this a blocking call
Therefore you may not get control back after this is called
until a much later time. It may be helpfull to use the
istheredata() function to check before calling this function
@return 8bit data packet from sender
*/
uint8_t  usart_recv(void)
{
          /* Wait for data to be received */
          while ( !(UCSR0A & (1<<RXC0)) )
          ;
          /* Get and return received data from buffer */
          return UDR0;
}

/* function check to see if there is data to be received
@return true is there is data ready to be read */
uint8_t  usart_istheredata(void)
{
          return (UCSR0A & (1<<RXC0));
}

/* function, sends strings of words to the uart by one character at a time*/
void putty_print(char words[])
{
          for (char i=0; i < strlen(words); i++)
          {
                    usart_send(words[i]);
          }
}
```