Fall 2013: Arduino Light and Signal Control for the eBike

Introduction

One of the key safety aspects of any bicycle is its lighting. Several options are currently available for riders to choose from. At present, the lighting devices sold are small compact units that can attach to the bike or rider. Most are powered and controlled from internal circuitry. Because of this common design, these devices aren't easily controlled. Lighting or signaling devices mounted out of reach of the rider are simply not controllable. Those that can be reached, require the rider to let go of the steering, posing another safety issue. This is demonstrated in Matt Sondermann's YouTube video, http://www.youtube.com/watch?v=c6wCaNMJwQM.

This project will utilize the Arduino to control the lighting and signaling on an electric bicycle. A switch cluster installed on the handle bars, next to the grips, will allow the rider a safe option of control. The switch cluster will set the mode of operation for the Arduino to control the lighting, without regard to were the lighting is placed on the bicycle.

The added benefit of using the Arduino for control is a low cost feature rich system. This will allow multiple operation modes of lighting and signaling to satisfy almost any riding condition. Due to the nature of the Arduino architecture, expandability and integration with other systems is available as an option. This design is ideal because it eliminates identifying the type of usage the system will be deployed in.

Project History

One of the main reasons for this project was the practicality of it. Anyone that has experience riding in large groups with other bicycle riders can appreciate brake signals, or any type of signal for that matter. During one ride with my family, my wife had mentioned how much more relaxing it would be to know when I was slowing down. It just seemed the most sensible choice to take this task on.

The project was originally submitted for the midterm paper. The initial work started out with using existing lighting devices and extending the on/off switches within reach. The hardware became a significant challenge. The devices would require modifications that would compromise the resilience to weather conditions. One attempt at modifying hardware resulted in a pile of parts. A brake and turn signal unit, featured in Matt Sondermann's video above, was special ordered. When it arrived, it was damaged. It became clear after working with the part that it would not work. Lastly, changes to the bicycle would be permanent. So the idea of making long lasting modifications without knowing the outcome was scrapped.

This shift in the project approach was late in the development phase. This would have resulted in a project that would not be completed by the due date. However, the project was allowed to be shelved and used for the term paper.

Theory of Operation

From an operator point of view, the system will be fairly simple to operate. Pressing either of the brake levers will turn the brake light on. Releasing both of the brake levers will turn the brake light off.

The turn signals can be turned on either left or right, by way of a 2 way switch. Sliding the switch to the left will blink the left handle bar cap LED and cause the tail lights to strobe to the left. Sliding the switch to the right will blink the right handle bar cap LED and cause the tail lights to strobe to the right. Sliding the signal switch to the center will turn off all of the signal lights.

The head lamp can be turned on by way of an on/off switch housed with the signal switch, in an input cluster located next to the handle bar grips. Sliding the head lamp switch on will turn the head light on in a default low beam mode.

The head lamp modes can be changed by means of a momentary switch also housed in the input cluster. Pressing the mode button changes the intensity and modulation of the head light. There will be 5 modes that the operator can cycle through by sequential pressing the mode button. These modes are low beam, mid beam, high beam, warning strobe, alert strobe, and search beam.

The head light beam mode can be quickly reset by sliding the head light switch off and on again. This will return the head lamp mode to default low beam.

When the head light is on, regardless of the mode, the brake light will be dimly lit. Since the brightness is a control of PWM to the shift register, if the signals are activated, the brake light will turn off.

The logical work flow follows these steps. When power is supplied to the circuit, the voltage regulators will stabilize the load voltages and power the respective systems. The Arduino board will power up and initialize. Next, the variable registers will be set to zero, then the system will enter into its loop sequence.

First the system will check the states of the input devices. As it checks, it will register the state values into respective variable registers. Next, the system will perform actions based on the register values. It will do this for each of the register variables until it reaches the last. Finally, the system will loop back to the beginning of the state checks and repeat the process until power is removed.

The benefit of using a register based action set allows the code set to be modular. This will be demonstrated in the Program Code section below. Also, the system can be expanded to utilize the registers for secondary functions outside the scope of this project.

The hardware is comprised of 4 distinct sectional groups. The first section consists of the input group. An instrument control cluster that is designed for this use was chosen as the main input device. The control cluster is positioned close to the steering grips, allowing the rider to control both the input device and the bicycle at the same time. The control cluster consists of a 2 way switch for left and right signaling, an on/off switch to turn the head light on or off, and a momentary switch to change the head light modes. The secondary input devices are brake sensors that are in line with the brake cabling. These are located between the brake handle levers and the brake cables feeds. The sensor type is a hall sensor that detects pressure on the brake line when force is applied to it.

The second section consists of the control group. The control device is an Arduino Uno board with modular shields for the input and output devices. The control device is housed in a case that is kept inside a weather proof pannier bag, along with the battery pack and eBike power control systems. The input, output, and power systems connect to the control device housing by means of connector plug dongles. This is similar to the eBike controller design.

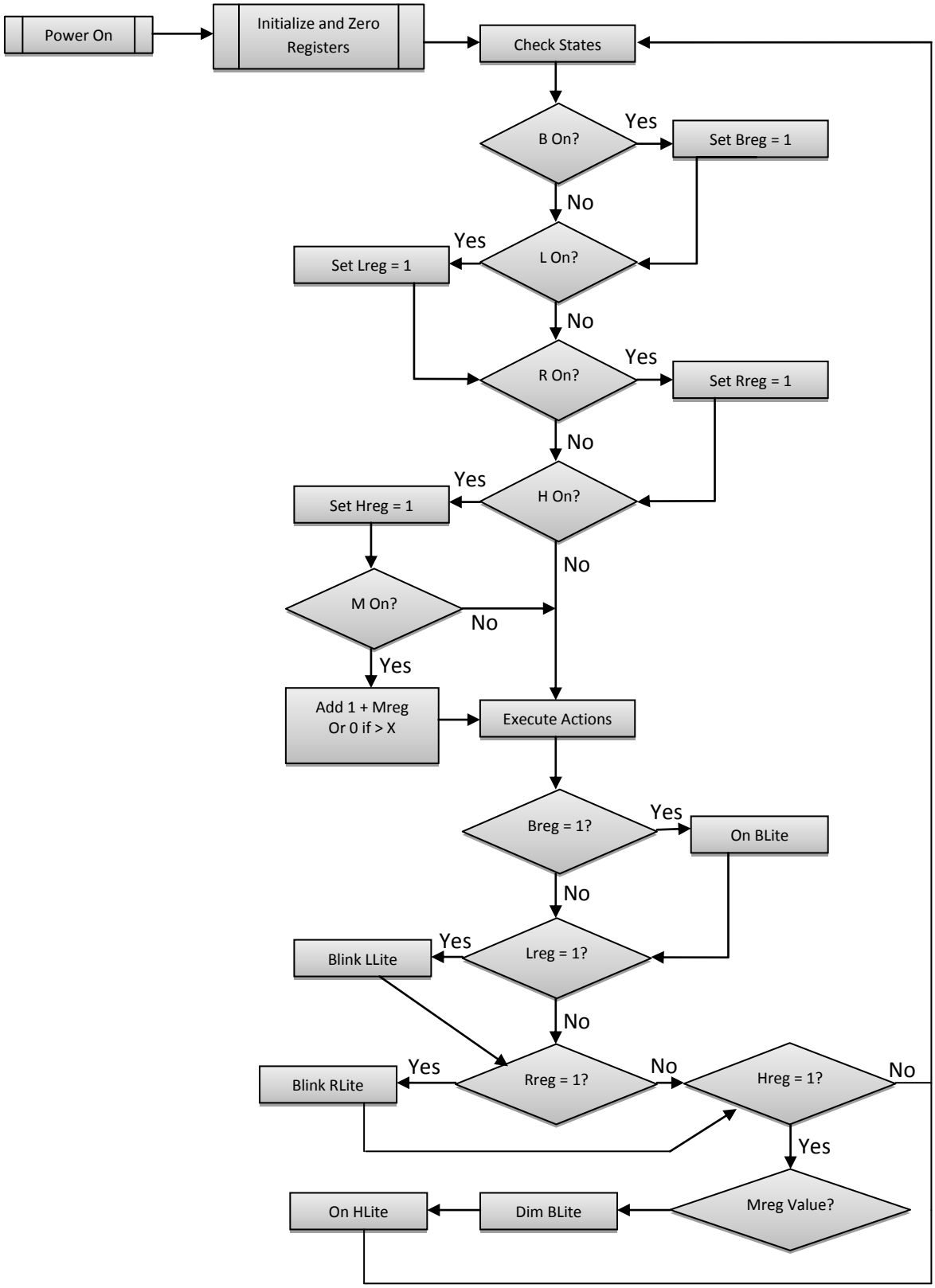
The third section consists of the output group. There are 4 output devices. The first is a Rupse 27 watt square LED head lamp, which is intended for use on off road motor vehicles. This is a meaty device with a die cast aluminum housing that is resistant to water, dust, and vibration.

The next device is a modified Planet Bike Blinky 3 tail light. The factory stock electronics and power systems have been removed. In their place are 8 high intensity LEDs, 6 yellow and 2 red. The LEDs are lit by a 74HC595 shift register that gets control signals from the Arduino. The shift register circuitry is also housed inside the tail light.

Finally, two shop built signal lights are mounted inside the ends of the handle bars. The signal lights are simply high powered LEDs with yellow reflectors hot glued to end caps. The wiring to the signal lights run between the grips and the handle bars from the control device.

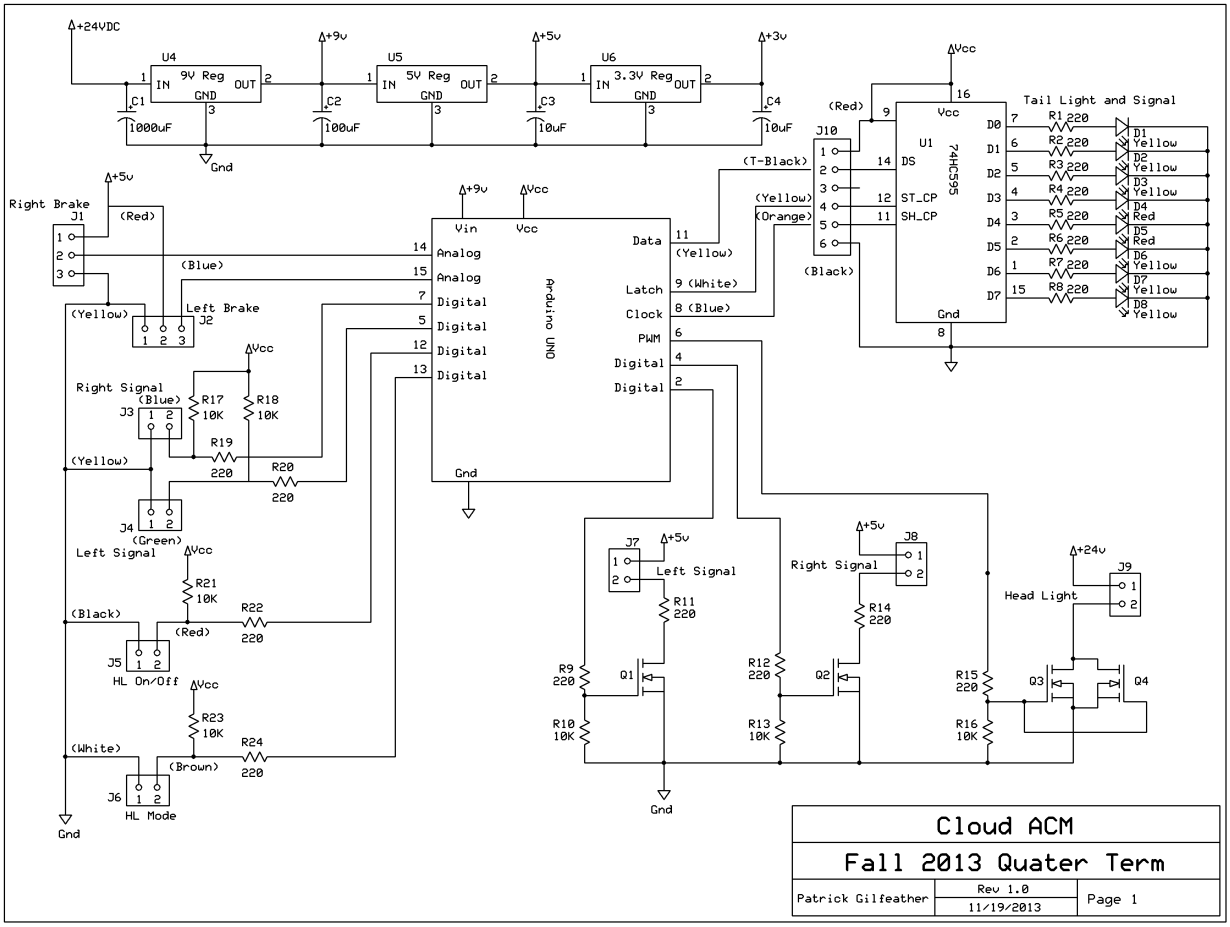
The fourth and final section consists of the power group. All of the devices get their power from the eBike battery pack via voltage regulators. There are various voltage splits to supply power the each of the loads independently of each other. The final result is load balancing without noticeable draining effects on the output devices.

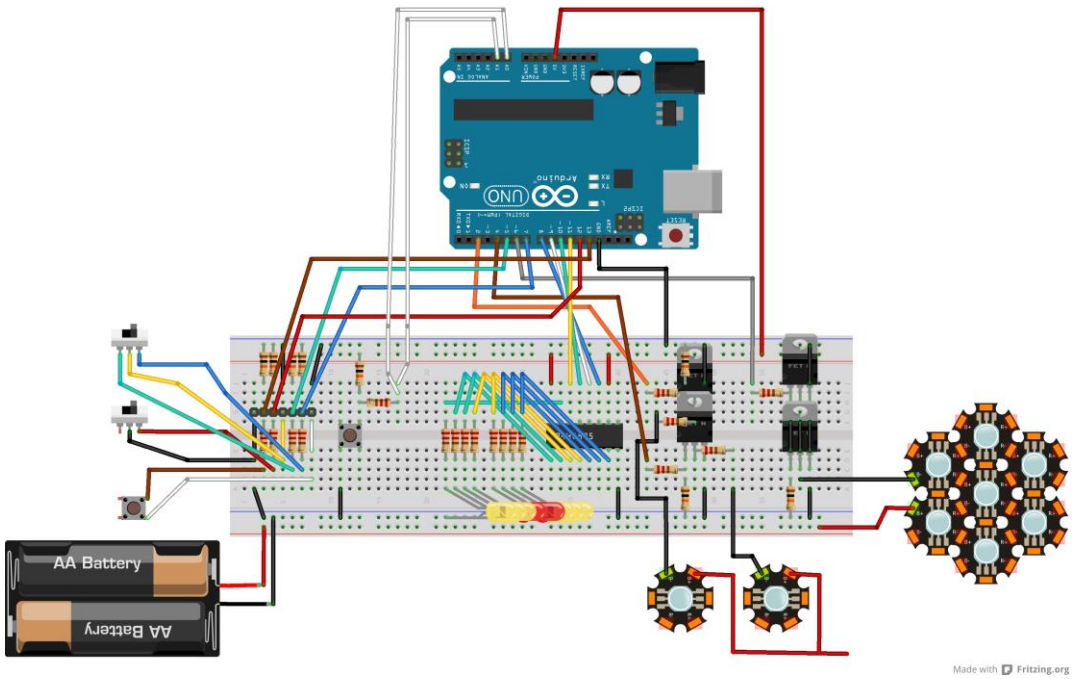
Operational Flow Chart



Author: Patrick Gilfeather / Cloud ACM / Seattle, WA

Schematics of Light and Signal Controls





Author: Patrick Gilfeather / Cloud ACM / Seattle, WA

Materials Parts List

Quantity	Component
1	27 Watt White LED Head Lamp
1	Handlebar Switch Cluster
2	Anderson Power Pole Jump Wires
1	Arduino UNO Board
1	USB Cable
2	1 Watt White LEDs
18	220 Ohm 1/4Watt Resisters
8	10K Ohm 1/4Watt Resisters
6	Ultra Bright Yellow LEDs
2	Ultra Bright Red LEDs
1	74HC595N Serial to Parallel Shift Register
4	IRF510 N-Channel MOSFETs
1	Push Button Switch
1	24 VDC Power Pack
1	64 Dual Power Bread Board
4	ExpressPCB Proto Boards

Program Code

```
/*
  -----[ Program Description ]----------------------
 Fall Quarter 2013 Term Project - Arduino Light and Signal Control for eBike
 */

// -----[ I/O Definitions ]------------------------
// Input Devices
const int HeadLampOn = 12;          // Head Lamp Switch
const int HeadLampMode = 13;        // Head Lamp Mode Button
const int LeftOn = 5;               // Left Turn Signal
const int RightOn = 7;              // Right Turn Signal
const int LeftBrake = 15;           // Left Brake Sensor
const int RightBrake = 14;          // Right Brake Sensor

// Output Devices
const int HeadLamp = 6;             // Head Lamp
const int LeftLamp = 2;             // Left Turn Signal
const int RightLamp = 4;            // Right Turn Signal
const int LatchPin = 9;             // Parallel Out - Serial Out Control Pin
const int ClockPin = 8;             // Synchronous Clock Data In
const int DataPin = 11;             // Synchronous Serial Data In

// -----[ Variables ]------------------------------
// Device Register variables used to make decisions based on value settings
int RegLeftBrake = 0;
int RegRightBrake = 0;
int RegLeftOn = 0;
int RegRightOn = 0;
int RegHeadLampOn = 0;
int RegHeadLampMode = 0;
int HeadLampModeCounter = 0;
int LastRegHeadLampMode = 0;
```

```
// -----[ Initialization ]-------------------------
void setup() {
  // Inputs Pins
  pinMode(HeadLampOn, INPUT);
  pinMode(HeadLampMode, INPUT);
  pinMode(LeftOn, INPUT);
  pinMode(RightOn, INPUT);
  pinMode(LeftBrake, INPUT);
  pinMode(RightBrake, INPUT);
  // Outputs Pins
  pinMode(HeadLamp, OUTPUT);
  pinMode(LeftLamp, OUTPUT);
  pinMode(RightLamp, OUTPUT);
  pinMode(LatchPin, OUTPUT);
  pinMode(ClockPin, OUTPUT);
  pinMode(DataPin, OUTPUT);
}

// -----[ Main Routine ]---------------------------
void loop() {
  // Read the state of the inputs into a registers:
  RegLeftBrake = digitalRead(LeftBrake);
  RegRightBrake = digitalRead(RightBrake);
  RegLeftOn = digitalRead(LeftOn);
  RegRightOn = digitalRead(RightOn);
  RegHeadLampOn = digitalRead(HeadLampOn);
  RegHeadLampMode = digitalRead(HeadLampMode);

  // Execute code based on register values.
  if (RegHeadLampMode != LastRegHeadLampMode) {
                  // This code checks if the head lamp mode register is active
    if (RegHeadLampMode == LOW) {
                  // If the register is currently low then run the code
      HeadLampModeCounter++;
                  // counter to cycle through the mode levels
      if (HeadLampModeCounter  == 7) {
                  // counter reset for operating 6 modes
        HeadLampModeCounter = 0;
                  // default operating mode to reset to
      }
    }
  LastRegHeadLampMode = RegHeadLampMode;
                  // deactivates the lamp mode register


 // Head Lamp Operating Modes
  if (RegHeadLampOn == 0) {
                  // Head Lamp is off
    BrakeOff();
    FullOff();
    HeadLampModeCounter = 0;}
  if (HeadLampModeCounter  == 0 && RegHeadLampOn == 1) {
                  // Head Lamp is Low Beam with dim brake lit
    Brake();
    LowBeam();}
  if (HeadLampModeCounter  == 1 && RegHeadLampOn == 1) {
                  // Head Lamp is Mild Beam with dim brake lit
    Brake();
    MildBeam();}
  if (HeadLampModeCounter  == 2 && RegHeadLampOn == 1) {
                  // Head Lamp is Mid Beam with dim brake lit
    Brake();
    MidBeam();}
  if (HeadLampModeCounter  == 3 && RegHeadLampOn == 1) {
                  // Head Lamp is High Beam with dim brake lit
    Brake();
    HighBeam();}
  if (HeadLampModeCounter  == 4 && RegHeadLampOn == 1) {
                  // Head Lamp is Bright Beam with dim brake lit
    Brake();
    BrightBeam();}
  if (HeadLampModeCounter  == 5 && RegHeadLampOn == 1) {
                  // Head Lamp is Search Beam with dim brake lit
    Brake();
    SearchBeam();}
  if (HeadLampModeCounter  == 6 && RegHeadLampOn == 1) {
                  // Head Lamp is Full On Beam with dim brake lit
    Brake();
    FullOn();}
```

```
 // Turn Signal and Brake Operating Modes
  if (RegLeftBrake == 1 || RegRightBrake == 1) {Brake();delay(10);}
                    // Turn on brake light if either brake sensor is active
    else {BrakeOff();}
  if (RegLeftOn == 0 && RegLeftBrake == 0) {LeftTurn();}
                    // Turn on the Left signals only
   else {BrakeOff();}
  if (RegRightOn == 0 && RegLeftBrake == 0) {RightTurn();}
                    // Turn on the Right signals only
   else {BrakeOff();}
  if (RegLeftOn == 0 && RegLeftBrake == 1) {LeftTurnBrake();}
                    // Turn on the Left signals and brake light
   else {BrakeOff();}
  if (RegRightOn == 0 && RegLeftBrake == 1) {RightTurnBrake();}
                    // Turn on the Right signals and brake light
   else {BrakeOff();}
}


// -----[ Sub Routines ]--------------------------
void FullOff()  {analogWrite(HeadLamp, 0);}
                    // Head Lamp is Off

void LowBeam()  {analogWrite(HeadLamp, 2);}
                    // Head Lamp is Low Beam

void MildBeam()  {analogWrite(HeadLamp, 5);}
                    // Head Lamp is Mild Beam

void MidBeam()  {analogWrite(HeadLamp, 10);}
                    // Head Lamp is Mid Beam

void HighBeam()  {analogWrite(HeadLamp, 25);}
                    // Head Lamp is High Beam

void BrightBeam()  {analogWrite(HeadLamp, 50);}
                    // Head Lamp is Bright Beam

void SearchBeam()  {analogWrite(HeadLamp, 100);}
                    // Head Lamp is Search Beam

void FullOn()  {analogWrite(HeadLamp, 255);}
                    // Head Lamp is Full On Beam

void Brake() {
                    // Shift Register command to turn brake light On
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 24);
    digitalWrite(LatchPin, HIGH); }

void BrakeOff() {
                    // Shift Register command to turn brake light Off
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 0);
    digitalWrite(LatchPin, HIGH);  }

void LeftTurn() {
                    // Shift Register command to strobe Left Turn signal
    digitalWrite(LeftLamp, HIGH);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 4);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 6);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 7);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(LeftLamp, LOW);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 3);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 1);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(LatchPin, LOW);
```

```
    shiftOut(DataPin, ClockPin, MSBFIRST, 0);
    digitalWrite(LatchPin, HIGH);
    delay(120);    }

void RightTurn()  {
                  // Shift Register command to strobe Right Turn signal
    digitalWrite(RightLamp, HIGH);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 32);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 96);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 224);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(RightLamp, LOW);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 192);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 128);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 0);
    digitalWrite(LatchPin, HIGH);
    delay(120);    }

void LeftTurnBrake() {
                  // Shift Register command to strobe Left Turn signal with Brake Light lit
    digitalWrite(LeftLamp, HIGH);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 28);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 30);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 31);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(LeftLamp, LOW);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 27);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 25);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 24);
    digitalWrite(LatchPin, HIGH);
    delay(120);    }

void RightTurnBrake()  {
                  // Shift Register command to strobe Right Turn signal with Brake Light lit
    digitalWrite(RightLamp, HIGH);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 56);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 120);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 248);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(RightLamp, LOW);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 216);
    digitalWrite(LatchPin, HIGH);
```

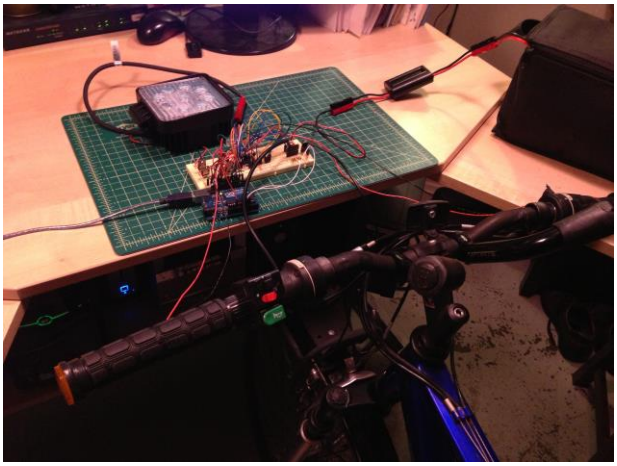
```
    delay(120);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 152);
    digitalWrite(LatchPin, HIGH);
    delay(120);
    digitalWrite(LatchPin, LOW);
    shiftOut(DataPin, ClockPin, MSBFIRST, 24);
    digitalWrite(LatchPin, HIGH);
    delay(120);    }
```

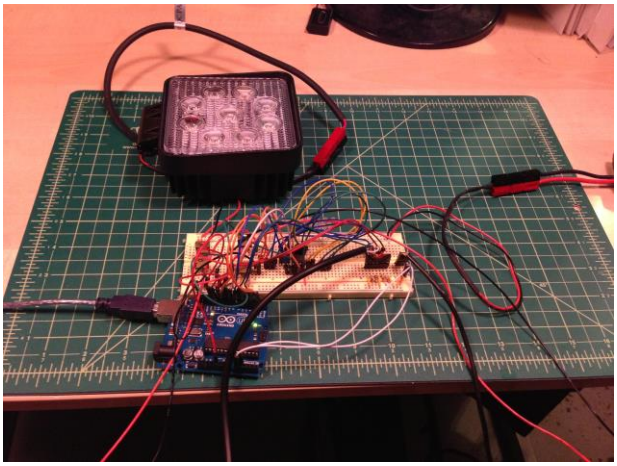
Bench Testing Results

The bench test demonstrated limitations to the design. The original design called for a modulating flasher mode. This mode was causing timing delays that would require more time to resolve. As a result, the project functions were compromised to allow the system to work in an acceptable state. This video, http://youtu.be/Dhz9qGgj6to demonstrates the operation of the input devices and resulting output on the various lights.

The cycling through the head lamp modes and reset worked as expected. The brake signals responded in near real time during all the head lamp modes. The brake light dim mode also worked as expected through all of the head lamp modes.

The turn signals did present a small problem. When the turning signals were active, there was a noticeable delay in response when cycling through the head lamp modes. This delay was noticeable when the brake lights were cycled on and off. The delay was annoying but not long enough to be addressed.





Field Testing Results

One of the biggest challenges with previous field tests was the durability of the equipment. The QuickFire Challenge from earlier this quarter, failed throughout the ride. I was fortunate to get this much data to present the video, http://www.youtube.com/watch?v=5vDrvEA1Yyk. This is because a breadboard was used instead of a soldered proto board.

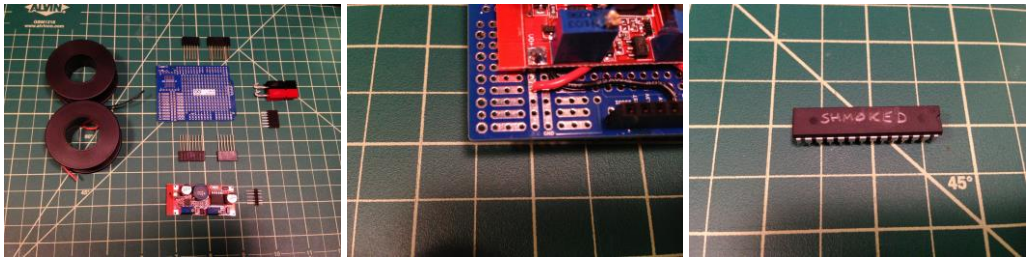
Another issued that was encountered was the several power sources and inefficient power converters. On the day of the QuickFire, it was in cold. I'm not sure if this caused issues, but I suspect it had.

With that said, the hardware for this project has to be resilient. The input, output, and control devices have to be built to withstand a harsh environment. The bicycle is particularly jarring on electronics. The temperature extremes of hot and cold, along with shaking and sudden shocks, really give the equipment a thrashing.

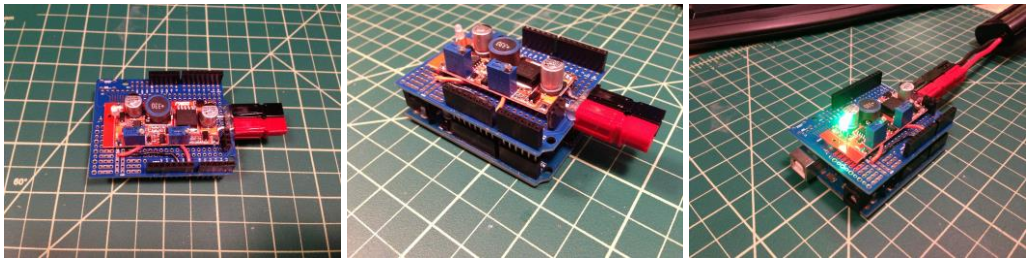
The Arduino shields offer a practical solution to this problem. The stacking nature also allows the control device to be compact, serviceable, and expandable. The costs of the prebuilt shields are low, making them a better choice.

In this project, there will be 3 shields stacked on top of an Arduino Uno board. The first will be the power shield, which provides all of the power requirements for the devices to operate at. The second will be the input/output shield that will contain the components for handing switching and interfacing to the input and output devices. The final shield will be the connector shield that will interface with the cabling to the external devices.

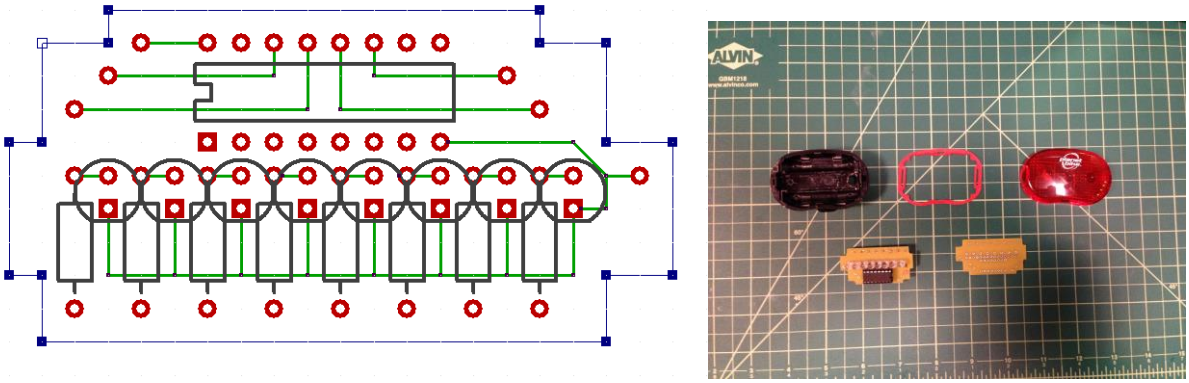
During the construction of the power shield, no layout preparation was made since all the components were modular and small in quantity. This was a costly mistake. It was only after the most of the shield was constructed that it a problem was discovered. As a result, more time was spent during disassembly and reassembly. This haste to correct the oversight led to another issue the ended with damage to the microcontroller.



Once the power shield was successfully rebuilt and tested, it was concluded that the CAD program would have lowered the build time that was trouble free. The testing of the power shield also reaffirmed that the modular approach was a solid choice for this project.



The brake light was built using a Planet Bike Blinky tail light. The electronics inside the brake light would comprise 8 LEDs, 8 resistors, and a Serial to Parallel Shift register. The housing of the tail light was just right for this design. A circuit board was designed in ExpressPCB CAD and an order was made with the vendor to manufacture the board used in this project.



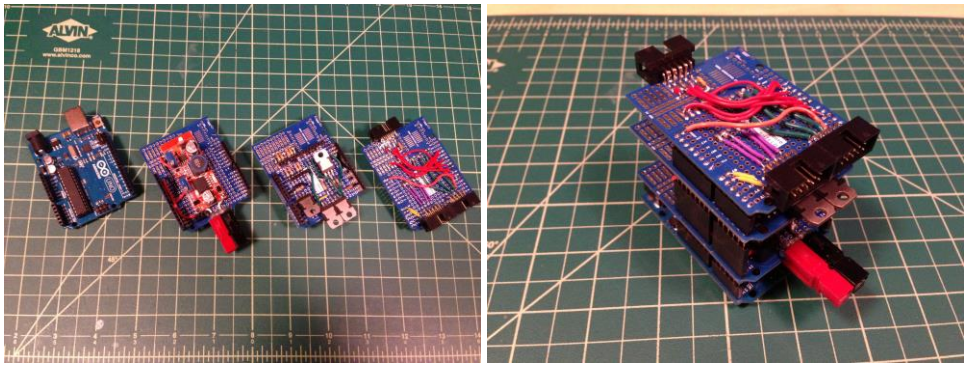
This designed proved to be extremely rewarding. The size of the board and placement of the components was optimal. Each step of the build fit like a puzzle. There was one problem with a shorting wire run from the limiting resistor to the shift register legs, but this was corrected. Once the board and wiring were attached, the board was hot

glued to the housing to fasten and protect it from moisture. Testing of the brake light was successful, as seen in this demonstration, http://youtu.be/CUc_xuuVmIw.

The IO shield layout was designed in the ExpressSCH program. No Arduino proto shields were available, so the board layout had to be drawn from scratch. The quality control after the drawing was created failed to identify several discrepancies. Many of the pads were incorrectly laid out and the dimensions of the board were not accurate.

As a result proposed drawing was ignored during the shield build. This resulted in the same issue that occurred with the power shield. As a consequence, an additional 5 hours were used to eventually correct the problems and test the board for correct operation. The final build of the IO shield also prompted the need for a connector shield. Originally, this was going to be incorporated with the IO shield.

The final shield was properly built and tested throughout the build. This solved the issued that plagued the earlier shields.



The cabling for the input and output devices were terminated at the cargo bag and underneath the head light. An enclosure was used, that was fastened to a bracket mount already available on the headlamp. The enclosure houses a proto board that the cabling is soldered to.

This design helped lower the amount of cable runs to the cargo bag and helped tidy up the front of the bike. Each cable was tested during the termination for proper continuity; this eliminated any issues with quality, but resulted in a longer build time.

Once all the hardware was in place and fastened, s test of the system was run. There were a couple of issues with the right brake sensor and rear right turn signal. The brake sensor was not activating because the sensor housing was pushed up against the headlamp assembly. Once the sensor was repositioned, the brake light operated. The rear right turn signal has a flicker when operated, but does scroll. This could be corrected with code, but is considered a minor issue in the grand scheme of the design.

During the test ride, the brake and turn signals operated as expected. The head lamp operation was also responsive and did not show any trouble during the test ride. The light levels were sufficient for the headlamp and signaling devices.

Summary and Conclusion

Yes, I could revisit everything already presented and type it all out again. I will not. I will finish with this. It is this that matters and what made possible the success of the learning process.

This paper is more than my attempt at design, development, and implementation of a gadget. It is a result of the support and good will of strangers. The open nature of the Arduino community was largely responsible for the availability of free material. The sharing of information was the success of this project.

I would like to point out as many of those that made this possible. To the creators of the Atmel microcontroller, it is the ground floor to some fantastic stuff. The Arduino team, they succeeded in making technology accessible to those who wouldn't have otherwise. To the companies Sparkfun, Adafruit, Element14, Jameco, Mouser, Digikey, Parallax, Vetco, Fry's, Lowes, Tacoma Screw, Stoneway Hardware, Radio Shack, ProDCtoDC, Amazon, Hobby King, PowerWerx, BatterySpace, Headway Headquarters, BMSBattery, EVAssemble, Grin Cycle, Recycle Cycles, USPS, UPS, FedEx, DHL, Bafang, ExpressPCB, REI, Niagara Cycle Works, Wheelsmith, Planet Bike, Tap Plastics, Zeeks Pizza, and Fosters Beer. If I missed you it was because of the Fosters.

Here's to the individuals that I've happened upon.

Russ William, he was the first to demo his electric bicycle to me, albeit via a YouTube video (http://www.youtube.com/watch?v=zKKvP9wWrlY). He showed it was possible to do this sort of thing.

Here's to those crazy kids up in Canada. Justin Lemire-Elmore at Grin Tech for riding across Canada on an electric assist cargo bike, this is a bucket item (http://www.treehugger.com/natural-sciences/crossing-canada-on-an-electric-bike-using-only-10-of-electricity.html). Stephane Melançon (aka Doctorbass), for his insane implementation of star/delta motor wiring (http://www.youtube.com/watch?v=L5HceFJp7aM). Steve Cesario (aka Steveo) for showing what a high powered bicycle is capable of (http://www.youtube.com/watch?v=JaDtIZJylbg).

Jenna deBoisblanc for sharing her setup of a light and signal system using Arduino (http://www.youtube.com/watch?v=O5YYsm_BqJQ). This was inspirational because the process was broken down without concealing the complexity.

Jeremy Blum for taking the time to organize and spell out what an Arduino can do and how to do it with his online tutorials (http://www.youtube.com/watch?v=fCxzA9_kg6s).

Richard Feynman, for the pleasure of finding things out (http://www.youtube.com/watch?v=AmiFnOqFWfM)

Thank you all for your help.

Finally, I would like to dedicate this project to my brother Kenneth. He would have been 54 just a few days after this paper is submitted. It is still difficult to fully understand what my family, friends, and I lost when he passed. I can't say with any certainty that I ever will. He truly lives in those who knew him.