3. APPROACH

SCDLS is designed to help aid pedestrians while using crosswalks by actively lighting the crosswalk and making it visible to approaching vehicles. By dynamically changing the status of the crosswalk, this system has the capability to save numerous lives by gaining the attention of approaching motorists.

3.1. System Overview

SCDLS consists of several subsystems in order to actively track pedestrians as they enter and exit the crosswalk and dynamically trigger the LED lighting to alert oncoming traffic. The team has gone through great lengths to keep power consumption to a minimum in order to ensure that the system as a whole will stay functional for an average of 5 years.

Below, Figure 3.1 shows an overview of the various subsystems that compose a SCDLS unit. The devices are powered solely by solar power and are intelligently controlled via an on-board microcontroller in order to further conserve power.

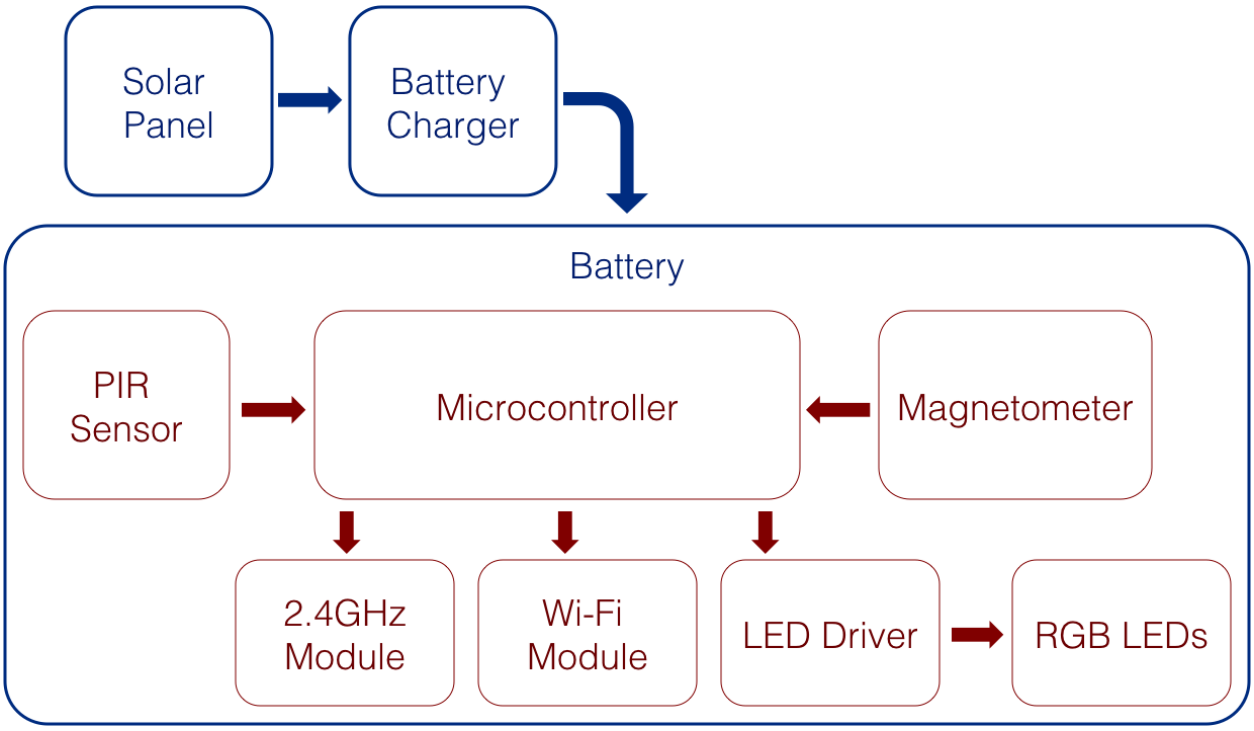

Figure 3.1 – SCDLS System Overview

3.2. Hardware

SCDLS necessitates a self-contained sustainable power architecture for each module. This architecture is displayed below in figure 3.2. Each of these components were selected based on their effectiveness and compatibility with other components in the system.

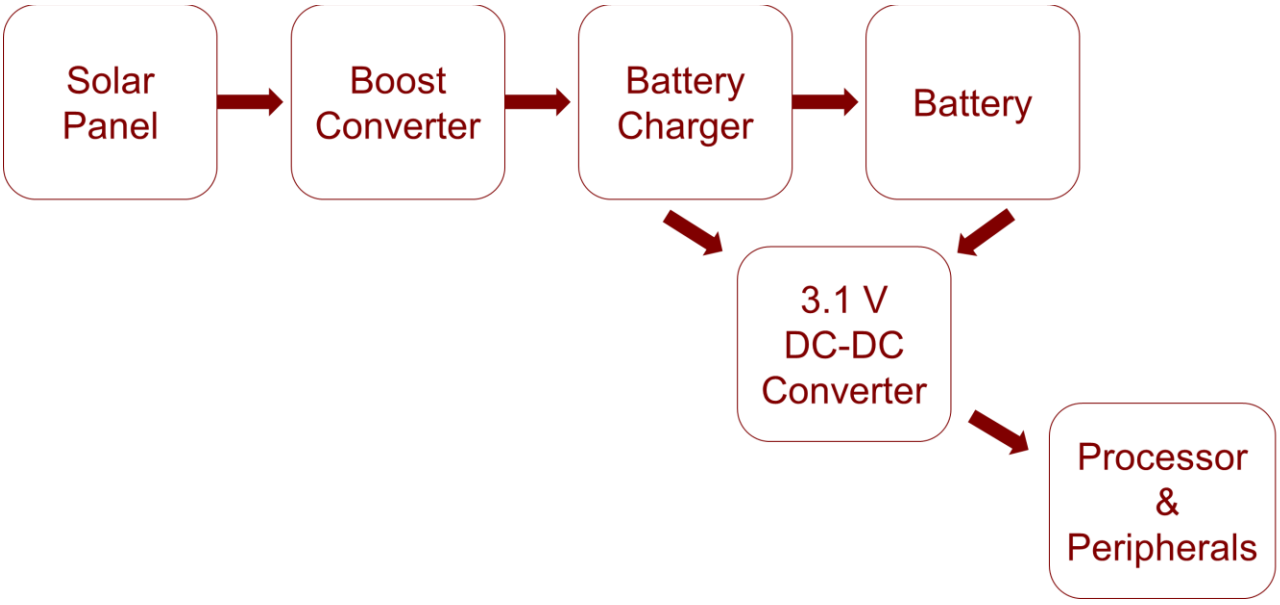

Figure 3.2 – Power Architecture for SCDLS Module

The team estimated power consumption based data gathered from the individual components in the system to determine what amount of power would be necessary. The worst-case calculations for a node and a hub are located below. The calculations assume that the module is located in Atlanta, Georgia. This assumption was made because SCDLS is not capable of withstanding an impact from a snowplow or other large machinery used for clearing snow form a road. The modules will only illuminate at night because it is unfeasible for the LEDs to compete with the light from the sun during the daytime, while still relying solely on solar power. Additionally, the duty cycles are team-estimated values that will depend highly on the traffic of a specific crosswalk. For ease of power calculation, the primary voltage is 3.1V because it is compatible with all components, which will be explained further in this document.

3.2.1. Node Energy Requirement Calculations

$$\text{Energy Required} = (24\ \text{Hours} - \text{Hours of Sunlight}) \times (\text{Average Active Power Consumption})$$

$$\text{Average Active Power Consumtion} = \frac{\text{NRF Avg Power}}{\left(\frac{\text{NRF Duty Cycle}}{100\%}\right)} + \frac{\text{Microcontroller Avg Power}}{\left(\frac{\text{Microcontroller Duty Cycle}}{100\%}\right)} + \frac{\text{LED Driver Avg Power}}{\left(\frac{\text{LED Driver Avg Power}}{100\%}\right)}$$

$$\text{Average Active Power Consumption} = \frac{(13\text{mA})(3.1\text{V})}{\left(\frac{10\%}{100\%}\right)} + \frac{(4.3\text{mA})(3.1\text{V})}{\left(\frac{5\%}{100\%}\right)} + \frac{(10\text{mA})(3.1\text{V})}{\left(\frac{5\%}{10\%}\right)}$$

$$= 9.7322\text{mW}$$

$$\text{Energy Required} = (24\ \text{Hours} - 9.92\ \text{Hours}) \times 6.24\text{mW} = 494\frac{\text{J}}{\text{Day}}\ [1].$$

3.2.2. Hub Energy Requirement Calculations

$$\text{Energy Required} = (24\ \text{Hours} - \text{Hours of Sunlight}) \times (\text{Average Active Power Consumption})$$

$$\text{Average Active Power Consumption} = \frac{\text{NRF Avg Power}}{\left(\frac{\text{NRF Duty Cycle}}{100\%}\right)} + \frac{\text{WiFi Avg Power}}{\left(\frac{\text{WiFi Duty Cycle}}{100\%}\right)} + \frac{\text{Microcontroller Avg Power}}{\left(\frac{\text{Microcontroller Duty Cycle}}{100\%}\right)} + \frac{\text{PIR Sensor Avg Power}}{\left(\frac{\text{PIR Sensor Cycle}}{100\%}\right)} + \frac{\text{LED Driver Avg Power}}{\left(\frac{\text{LED Driver Duty Cycle}}{100\%}\right)} + \frac{\text{Magnometer Avg Power}}{\left(\frac{\text{Magnometer Duty Cycle}}{100\%}\right)}$$

$$\text{Average Active Power Consumption} = \frac{(13\text{mA})(3.1\text{V})}{\left(\frac{100\%}{100\%}\right)} + \frac{(197\text{mA})(3.1\text{V})}{\left(\frac{0.35\%}{100\%}\right)} + \frac{(4.3\text{mA})(3.1\text{V})}{\left(\frac{100\%}{100\%}\right)} + \frac{(0.17\text{mA})(3.1\text{V})}{\left(\frac{20\%}{100\%}\right)} + \frac{(10\text{mA})(3.1\text{V})}{\left(\frac{20\%}{100\%}\right)} + \frac{(0.1\text{mA})(3.1\text{V})}{\left(\frac{20\%}{100\%}\right)}$$

$$\text{Energy Necessary} = (24\ \text{Hours} - 9.92\ \text{Hours}) * 6.24\text{mW} = 1327\frac{\text{J}}{\text{Day}}\ [1].$$

Multiple methods of power generation were considered before solar power was selected. Comparisons of the options can be seen in Table 3.2.2. Based on this analysis, the solar panel was a standout choice for powering SCDLS modules. Solar panels are also used by industry for similar applications.

Table 3.2.2 - Possible Power sources for SCDLS Module [2]

Generation Technology	Solar	Thermoelectric	Acoustic	Piezoelectric	RF
Power output	15mW/cc [x]	40µW/cc [x]	960nW/cc [x]	330µW/cc [x]	116µW[x]
Reliability	Medium	High	Low	Low	Medium
Ease of Implementation	Medium	Low	Low	Low	Very Low

Based on the power analyses that are outlined above, it was clear that solar power was the only feasible option for supplying power to this application. Therefore, multiple solar panels were compared. In order to account for any unforeseen issues, and to have a functional prototype before completing all necessary power optimizations, a safety factor above 4 was desired. The two primary requirements for selecting a solar panel was that it meets the desired power output with a safety factor and is highly durable panel. For that reason, a 5V, 250mA peak, epoxy-covered solar panel was chosen. Figure 1.1.X shows the power output from the solar panel during a sunny February day in Starkville, Mississippi.

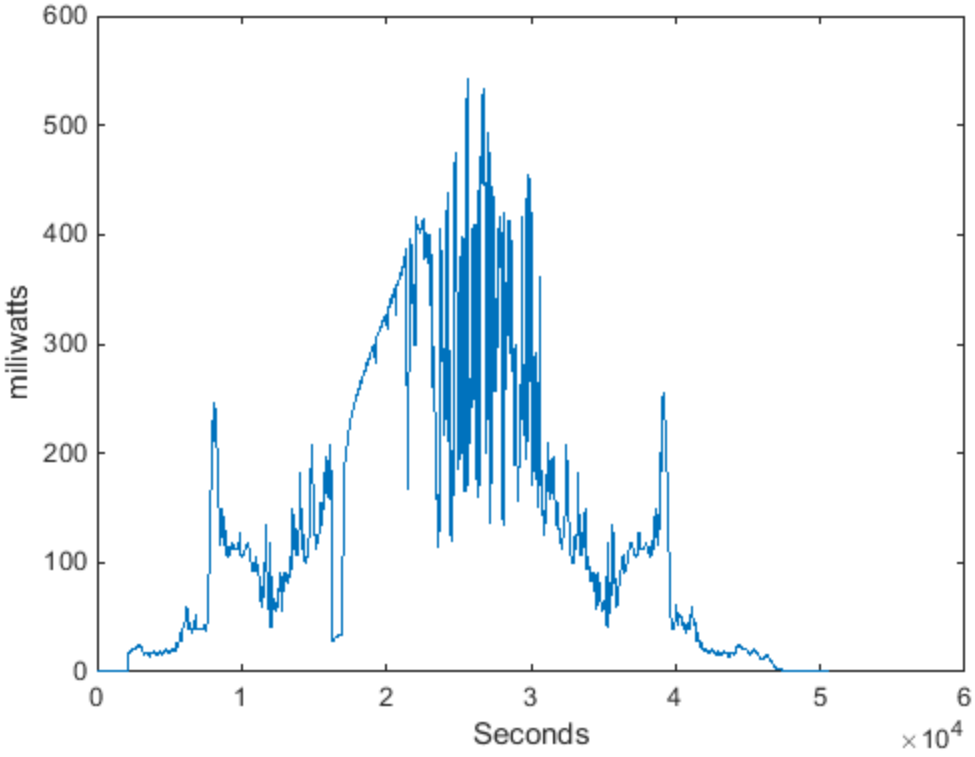

Figure 3.2.2a – Solar Panel Power Output

When integrated using the MATLAB `trapz` function, this data yields a single day power output of 6614 $\frac{\text{J}}{\text{Day}}$, which gives our team a safety factor of 4.98 for the hub, and a factor of safety of 13.4 for the node on that particular day. While this is not representative of the worst possible day the team plans to use a substantially sized battery to allow the system to operate for extended periods without sunlight. When evaluating different battery options, the primary evaluation factors were use of protected cells, energy density, packaging efficiency, and temperature requirements. Protected cells are standard battery cells with an added current interrupt device (CID) that will disconnect the battery terminals from the battery in the event of an overcharge, undercharge or overcurrent. This is necessary for product safety and reliability. Based on figure 3.2.2b it was clear that the Li-Ion were the most energy dense batteries, and were capable of meeting the temperature requirements of -10°C to 50°C [Battery University]. A few packages were considered, but the most efficient package for our system is a rectangular prism due to the design of the casing. As such, a 3500mAh lithium-ion Samsung Galaxy Note 2 battery was selected because of its flat rectangular shape. As well, the Galaxy Note 2 battery has built in cell protection and is manufactured by reputable and experienced company [3, 4].

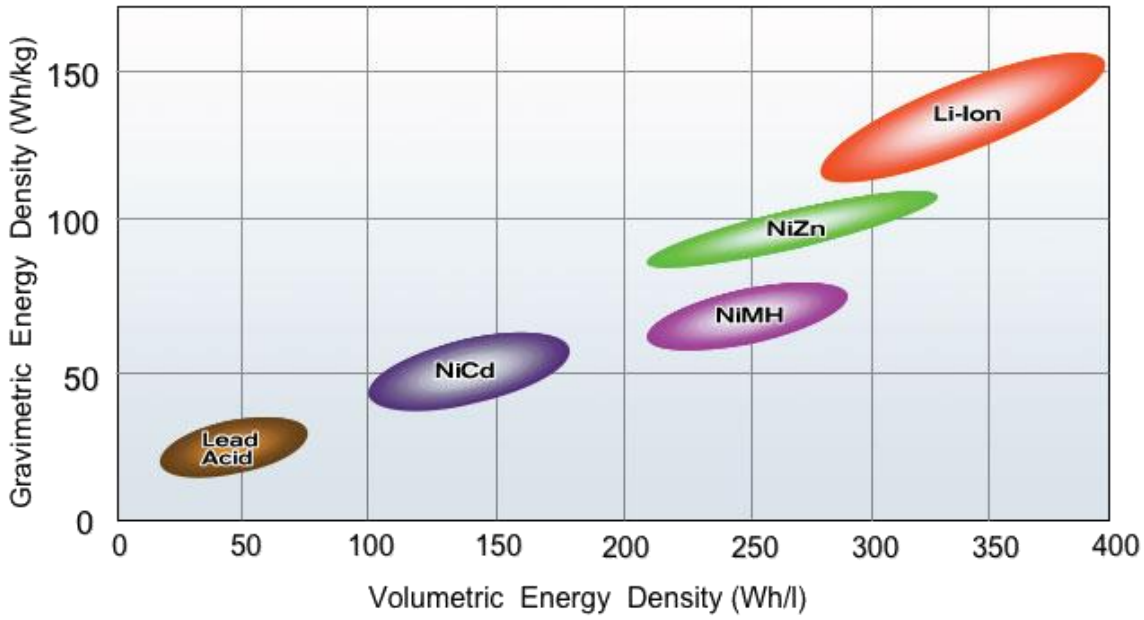

Figure 3.2.2b – Energy Density of Different Battery Technologies [5]

The team desired for a full charged battery to be able to power a module for 30 days without receiving any energy from the solar panel. Assuming the voltage regulation circuit is 90% efficient, the amount of time the battery can power a module is as follows:

$$Hub\ Battery\ Capacity = \frac{(90\%)(3500mAH)(3.7V)(3.6\frac{J}{mWH})}{(1328\frac{J}{Day})} = 32\ Days$$

In order to transfer the energy from the solar panel to the battery, an energy conversion circuit is necessary. Our requirement for this application was minimum efficiency of 90% and compatibility with the solar panel. Some possible solutions to this problem were integrated circuits, a custom made circuit, or a premade solar panel charger circuit board by Adafruit. After analyzing the different options, it was determined that the Adafruit board was not sufficiently efficient, and was larger than necessary. The custom-made circuit designs with discrete components soon exceeded the complexity the team felt comfortable with at no benefit. As a result, integrated circuits were the front-runner. Multiple ICs were evaluated, but the ST Microelectronics SPV1040 and L6924D shined ahead of the pack due to the high quality documentation and a reference design that was closely applicable to our design. This reference design also came with test data indicating its performance in applications similar to SCDLS's [6].

Once energy is stored in the battery, it needs to be converted to a consistent voltage that is compatible with the microcontroller and all other peripherals. 3.1V was chosen because it is well within the input limits of all of the components chosen. The team set several constraints on the conversion of energy from the battery to power the other components: compatibility with input and output voltages, sufficient current handling capacity, and 85% efficiency under typical load conditions. Possible solutions meeting these requirements included linear regulators, low dropout regulators, and switching regulators, but after some analysis it was soon apparent that only a switching regulator would meet the efficiency requirement. In order to find an integrated circuit with these requirements a parametric search was performed that only

displayed components meeting these requirements, and the lowest cost component with quality documentation was chosen. This component is the TPS62240 switching regulator from Texas Instruments. The switching regulator design has been implemented following the reference designs, but if there are any voltage regulation or ripple issues those will be resolved by the addition of a smoothing capacitor.

3.2.3. Pedestrian Detection

One of the major advantages of SCDLS over competitors is the ability to detect pedestrians accurately without user interaction. To do this, the team considered a number of various sensors before choosing passive infrared (PIR) technology as the sensor type to continue forward with. Other considerations included ultrasonic sensors and infrared (IR) distance sensors. There were many considerations associated with each sensor type. Before expanding on the background theory of each sensor, Table 3.1 below gives an overview of the major characteristics of each of the sensor types previously mentioned.

Table 3.2.3 – Considerations of Different Sensor Types

Technology	Maximum Distance	Power consumption	Weather Considerations	Field of view
PIR	16 feet	30 µA	Easily weatherproofed	22° conical
Ultrasonic	6.5 feet	15 mA	Easily weatherproofed	10° conical
IR	6.2 feet	40 mA	Susceptible to Humidity	2° conical

Based on the information above, the team found it fitting to eliminate IR sensors from further consideration based on their extremely small field of view. The small field of view would mean than an array of IR sensors would be needed in order to accurately detect when a pedestrian enters or exits the crosswalk. As well, the IR sensors do not fare well in standard weather conditions. The PIR and ultrasonic sensors accomplish similar tasks in different ways. To start, PIR sensors fundamentally work by detecting changes in amounts of infrared radiation produced by objects. Usually the module consists of a comparator and a Fresnel lens. The infrared radiation values change significantly as individuals move in front of the module. Through the use of the Fresnel lens, these changes can be compared to the sensor's previous value to see if movement has been detected.

In comparison, ultrasonic sensors work by detecting sound waves. The module we were considering was unique in that it basically acts as both a speaker and a microphone. The sensor first emits a precise high frequency sound. The sensor then listens for a response back to emitted sound and uses the time for the response to occur to determine the distance from the object.

Both modules had the following pros and cons. The ultrasonic sensors could give us distance data. However, data was inherently noisy and would need to be filtered using fairly advanced techniques in order to obtain usable data. In comparison, the PIRs were one-third of the size, but only recorded if an object had passed by. The team chose to use PIR sensors because they are more accurate than ultrasonic sensors and produce data that does not require as much filtering. As well, the detection threshold for the PIR sensors is greater than the ultrasonic sensors and thus false positives would be reduced. Finally, the PIR sensors are more energy efficient than the ultrasonic sensors.

The normal width of a crosswalk is approximately 6 feet, however, in larger cities, this is oftentimes doubled. This presented the unique challenge of finding sensors that could work at distances of up to 14 feet, while still being small enough to physically fit in the modules. The PIR is the only sensor that met this requirement. In addition, with a standby current of 1µA, they were the obvious choice for our power and life-of-use constraints.

3.2.4. Vehicle Detection

Although PIR sensors work very well for pedestrian detection, our tests thus far have shown that they are not as effective at detecting vehicles. Because of the high rate of speed that vehicles can be moving and their direction relative to the crosswalk, the PIRs might not accurately detect vehicular traffic. To counter this, the team decided to add another component to the hubs, a magnetometer. Fundamentally, a magnetometer works by detecting changes in the magnetic field around itself. Because all vehicles contain large quantities of metal, when a vehicle crosses a magnetometer, it detects a dramatic change in the magnetic field around it. The microcontroller can use an algorithm to determine when a car passes over the hub using the data produced by the magnetometer.

3.2.5. Light Emitting Diodes

A number of light emitting diodes (LEDs) were considered before the team decided to use common anode red, green, blue (RGB) LEDs. In conjunction with the team's constant current LED driver, these modules will allow the team to display alerts to drivers in an array of colors at a varying brightness. This was an important consideration as to not inhibit the driver's field of view with exceedingly bright light. The team chose not to use a single color LED because different states and countries have different laws on what colors are permitted for use on roadways. By using an RGB LED, these colors can be set and changed after the fact to choose the best color light for the setting.

3.2.6. LED Drivers

The two options for driving the LEDs on the modules are either using the GPIO pins on the microcontroller or using an LED driver IC. Using the GPIO pins is unsuited for our application because it would require placing current limiting resistors in series with the pins of the LEDs. The current limiting resistors would consume power unnecessarily, which is undesirable since one of the system's main constraints is power consumption. Constant current LED drivers contain hardware to regulate the current through each LED. As such, constant current LED drivers do not require the use of current limiting resistors, and they also ensure that all of the LEDs receive the same amount of current while preventing the LEDs from drawing more than their maximum rated current. Therefore, the team decided to use constant current LED drivers.

The characteristics used to evaluate the different LED drivers were the following: minimum supply voltage, communication bus, number of outputs, package, dimming capabilities, constant current capabilities, and cost. Originally, the chosen minimum supply voltage was 2.7V, however later in the design, the minimum supply voltage was increased 3.1V due to factors that will be discussed later. The communication bus chosen was I^2C due to the ease of use using the Arduino Wire library and the team's familiarity with it. Since each module will use four RGB LEDs, the minimum number of outputs required on the LED driver is twelve, because each RGB LED has three cathodes that must be connected to the driver. However, since I^2C supports multiple devices on the same bus, multiple LED drivers could be used to bring the total number of outputs to twelve if the drivers have enough addressing pins to support the required number of drivers on the I^2C bus. The preferred package for the LED driver is shrink small outline package (SSOP) or an SSOP variant due to the packages' small sizes and ease of soldering. However, an LED driver in a quad-flat no-leads (QFN) package is acceptable even though it is harder to

solder than SSOP. The LED driver needs dimming capabilities in order to reduce power consumption by the LEDs and to allow control of the brightness of the LEDs in order to prevent unnecessarily distracting drivers or pedestrians. As discussed previously, the LED driver needs constant current capabilities. Finally, cost was considered as a final discriminating factor.

The following is a table showing the characteristics of four different LED drivers considered for use.

Table 3.2.6 – Comparison of LED Drivers

Manufacturer	Part Number	Minimum Supply Voltage	Communication Bus	Number of Outputs	Package	Dimming Capabilities	Constant Current	Cost (USD)
Maxim Integrated	MAX8647	2.7V	I^2C	6	Thin QFN	PWM	Yes	$5.46
NXP Semiconductors	PCA9532	2.7V	I^2C	16	TSSOP, HVQFN	PWM	No	$2.65
ISSI	IS31FL3218	2.7V	I^2C	18	SOP, QFN	PWM	Yes	-
Texas Instruments	TLC59116	3.0V	I^2C	16	TSSOP, VQFN	PWM	Yes	$3.15

The Maxim LED driver was the initial candidate for use in the system. However, the Maxim LED driver has a hardcoded I^2C address and thus only one LED driver can be used on the I^2C bus. As well, the Maxim LED driver was one of the most expensive drivers found. The NXP LED driver would be suitable for the system if it were constant current. Since the NXP LED driver is not constant current, it was not chosen for use in the system. The ISSI LED driver met all of the requirements for the system. However, Digi-Key does not sell the ISSI LED driver, and Mouser only sells the ISSI LED driver in multiples of 550 units. The team was unsuccessful at finding a supplier for the ISSI LED driver. Finding LED drivers that met the 2.7V minimum supply voltage requirement proved to be extremely difficult, so the team decided to increase the supply voltage of the system to 3.1V. Given the increased supply voltage, the Texas Instruments LED driver was chosen because it met or exceeded all of the requirements for the LED driver while being significantly less expensive than the other components.

3.3. Software

3.3.1. Implementation Details

The SCDLS software stack is written in C++. This choice was made for a few reasons: first and foremost, much of the Arduino libraries are written in C++. Using C++ allows us to utilize these libraries with little to no additional effort. Because we utilize these libraries, our code's interaction with basic hardware features of the CPU is greatly simplified. C++ also has modern language features that make designing reusable software very easy. Of course, many of these features, such as run-time type information, are too resource-intensive to be used on a microcontroller, so we carefully considered each feature of the language to use. Done correctly, C++ is a great system to work in.

All the subsystems in our firmware are implemented as C++ classes. This approach provides encapsulation and facilitates asynchronous algorithms, which are crucial to maintaining the responsiveness of the system. The only downside to this approach is that there is a small amount of runtime overhead, because instance pointers must be passed to all member functions, consuming a hardware register. However, this would have to be done anyway for many of the subsystems, because they maintain state between scheduler events.

3.3.2. Subsystems

Our project incorporates various software subsystems to meet our constraints and to provide a maintainable codebase. These subsystems include the scheduler, the device drivers, and the command layer.

3.3.3. Scheduler

The scheduler is responsible for keeping system time and for executing events at the appropriate point. Other software modules communicate with the scheduler to schedule events. The scheduler will also ultimately be responsible for managing the power saving modes on the microcontroller (when this functionality is implemented). The scheduler will awaken the CPU only when tasks are scheduled to be executed, leaving the device in a low-power state at all other times.

Modules wishing to use the scheduler are required to inherit from a C++ abstract base class, which allows the scheduler to call their callback function at the appropriate time without knowing exactly what system it is invoking. Although there is a small runtime cost associated with virtual function calls, it is quite small, even on embedded platforms such as ours. Furthermore, much of the Arduino stack already makes extensive use of virtual calls. We feel that any small runtime cost is easily offset by the ease of implementation. The primary alternative to the use of virtual functions is passing pointers to raw functions; however, because most modules maintain state, the scheduler would need to pass them an instance pointer anyway. Furthermore, code involving pointers to functions is generally less readable and maintainable than code that uses proper inheritance.

The scheduler relies on cooperative multitasking between modules, and does not make any strong real-time guarantees. Therefore, it is the responsibility of the subsystems' authors to ensure that the subsystems do not block the CPU while waiting for external events. Although it would be advantageous to have a real-time operating system (RTOS) from a responsiveness perspective, the very limited system resources of the microcontroller prohibit an advanced task-switching framework. Therefore, the cooperative scheduler provides a good compromise between a full RTOS and no multitasking framework whatsoever.

3.3.4. Device Drivers

The device drivers are the subsystems, which communicate directly with the hardware. These modules are responsible for handling input from sensors, sending data over the wireless links, and turning on and off the lights. These systems forward incoming events to the command layer. The implementation details for most of these driver systems are fairly straightforward.

3.3.5. Network Driver Subsystem

In contrast to most of our device drivers, the networking subsystem is fairly complex. It provides several features beyond those provided by the nRF24L01+ radio hardware. The radio uses a simple data-link protocol called ShockBurst. A ShockBurst frame consists of a five-byte address, a 32-byte payload, and a

CRC16 checksum. The radio can be configured to receive frames addressed to up to five addresses. Our system uses the first four bytes of the address as a "network ID." Each crosswalk system will have a unique network ID, assigned by the system administrators at installation time. This allows multiple crosswalks to be installed in close proximity without interfering with each other's communications. The last byte of the address is the node ID; each module assigns itself a random ID when it boots. ID collisions are handled in the command protocol, discussed later.

Offset	0x00	0x01	0x03
Data	Frame Type	Frame Data Size	Packet ID

Figure 3.3.5a – Network Subsystem Large Frame Header

Our network subsystem is a combined network and transport layer implemented on top of the radio's data link layer. It allows nodes to send messages to any other node, as discussed above. It also allows nodes to send packets of arbitrary size, not limited to the 32-byte ShockBurst frame size. Large packets, which do not fit into a single frame, are broken into pieces and sent sequentially. The receiving node(s) reassemble the packets upon delivery. This model incurs an overhead of one byte on all frames sent, and an additional overhead of three bytes per frame on large packets. This overhead consists of frame information such as the packet ID, the relative frame number, and the total number of bytes in the packet. Because the nodes could potentially be transmitting relatively large data, such as sending stored traffic statistics to the master node, this overhead is rather small and is acceptable.

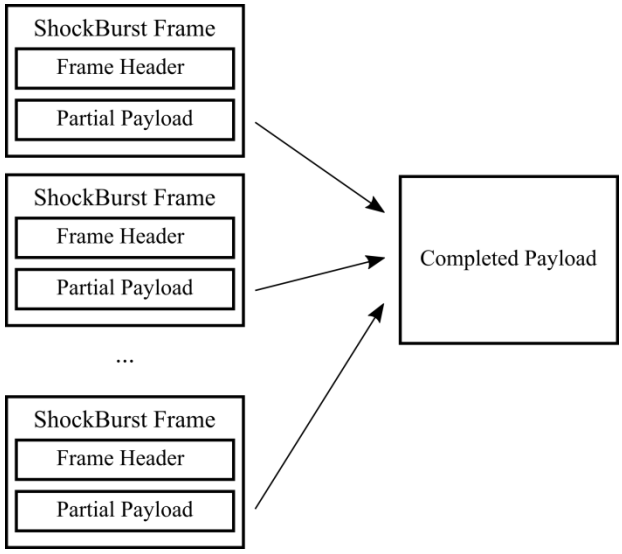

Figure 3.3.5b – Large Payload Reassembly

The last main feature provided by the networking subsystem is broadcast packets. In addition to sending packets to other individual modules in the network, modules can broadcast a packet that will be received and processed by all nodes. This functionality is used by the hubs to broadcast sensor data, which is received and interpreted simultaneously by all nodes. The networking subsystem uses the multiple-address functionality of the radio IC to not only receive packets addressed to its specific node ID, but also packets addressed to a reserved "broadcast ID" (currently 0x00). This design eliminates the need for each hub to keep track of every other node in the network and manually send copies of sensor data to each of them, thereby reducing power consumption and network congestion.

3.3.6. Command Layer

The command layer is responsible for taking inputs from the sensors and performing actions appropriately. The primary actions performed by the command layer are dispatching commands to other subsystems; due to the asynchronous nature of our software stack, little CPU time is spent in the command layer itself.

Because no sensor is perfect, there will be some false readings. One of the functions of the command layer is to filter out sensor noise and to only turn on the lights when it is highly probable that there are pedestrians present. There are several options for filtering algorithms for the PIR sensors we chose. The algorithm we are currently using is a simple threshold detection algorithm where two sensors must be triggered within a preset number of seconds, currently five seconds. This algorithm is simple but should prevent occasional false positives caused by random errors.

The following flowchart gives an overview of the actions performed by the command layer. Note that only the hubs have onboard sensors; therefore, the shaded grey section of the flowchart is only implemented on hubs.

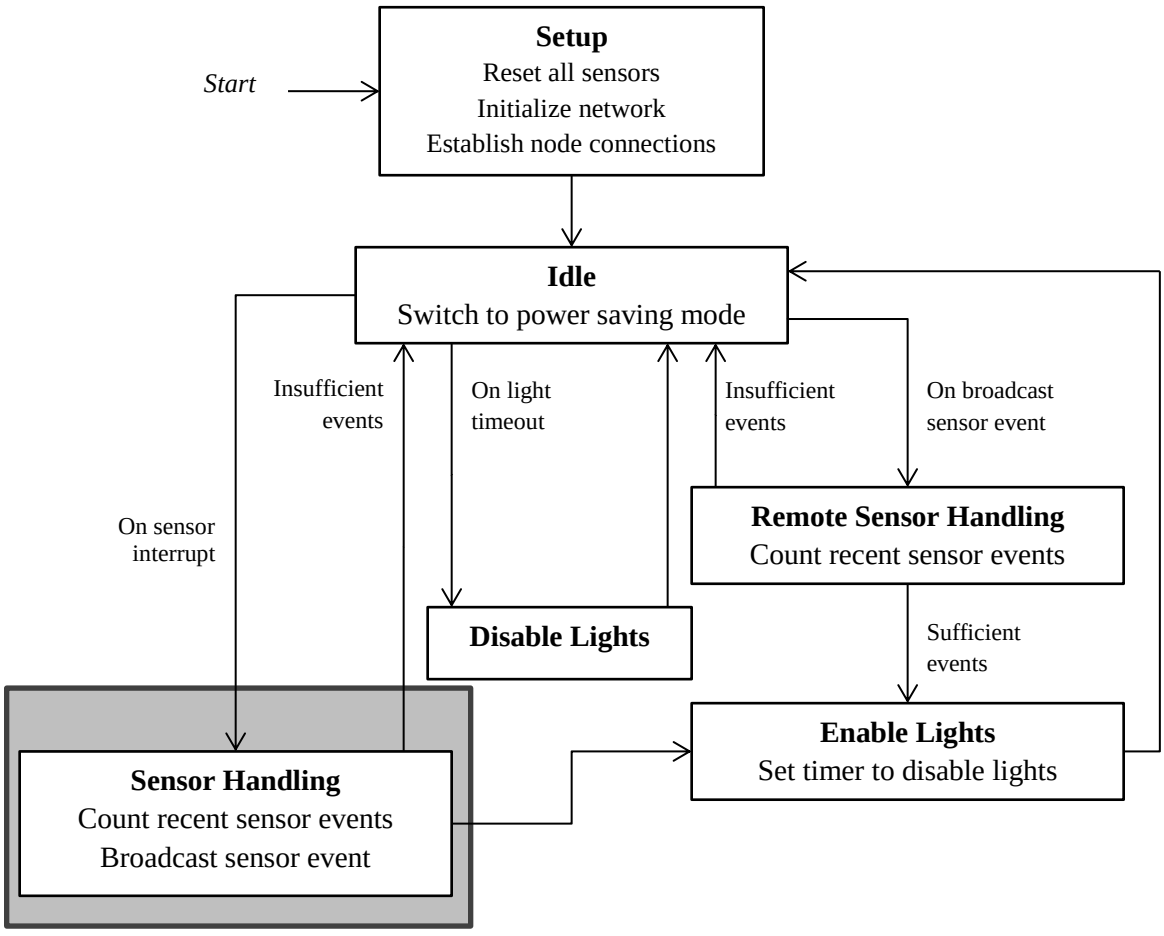

Figure 3.3.6 – Command Layer Logic Flowchart

The modules spend most of their time in power save "idle" mode. Because all the subsystems are activated by external events or on a timed basis, the command layer does not have to spend most of its

CPU time in a busy-waiting loop (that is, executing no-operation instructions while waiting for an event). When an event occurs, the subsystem's handler runs. Then, if necessary, the handler invokes the appropriate routine in the command layer. The command layer will handle the sensor event and dispatch a command to the light driver if necessary. It will also schedule a lights-off event, so that the CPU does not busy-wait for the lights to turn off. The scheduler will then wake the processor and return control to the command layer at the appropriate time.

3.3.7. Use Cases

Using the above high-level flowchart, several user interactions are possible. Note that in each case, the goal of the system is to provide as much lighting to the crosswalk as possible.

The ideal "sunny day" user interaction is as follows.

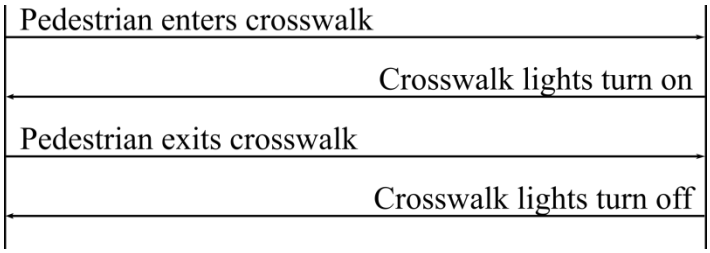

Figure 3.3.7a – Sunny Day User Interaction

This is a very simple ideal interaction; this is by design. SCDLS should work with no interaction from its primary users. Of course, this diagram is from the perspective of the user; from the standpoint of the system, this interaction gets somewhat more complicated.

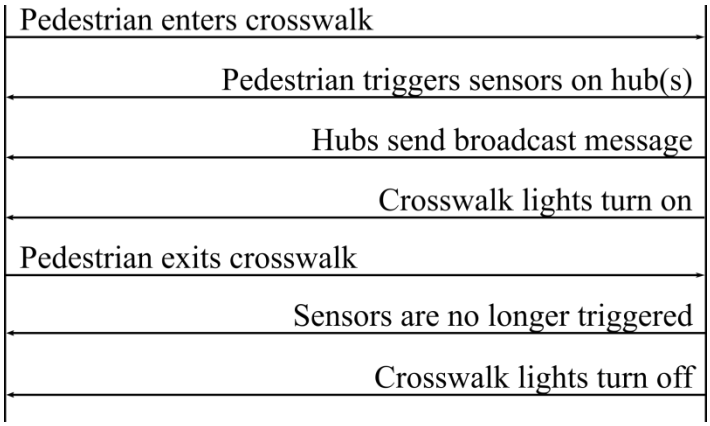

Figure 3.3.7b – Sunny Day System Interation

A worse-case scenario is when there is interference on the 2.4GHz spectrum. In this scenario, not all nodes may receive the broadcasts. Even though these messages may not be delivered, the nodes that do receive the message turn on their lights, providing some protection to the pedestrian. Although this may look unusual to users, we decided that this degraded operation provides some protection to pedestrians and is better than total system failure. The following diagram shows the events for this degraded operation:

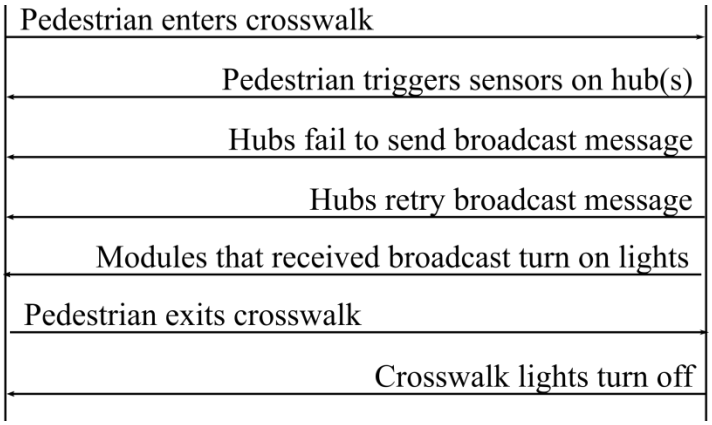

Figure 3.3.7c – Rainy Day User Interaction

WORKS CITED

[1] Washington Post, "Winter begins today: Five questions and answers about the solstice," 21 December 2014. [Online]. Available: http://www.washingtonpost.com/blogs/capital-weather-gang/wp/2014/12/21/winter-begins-today-five-questions-and-answers-about-the-solstice/. [Accessed 25 March 2015].

[2] National Electronics Manufacturing Center of Excellence, "Energy Harvesting from Wireless Sensor Networks," January 2011. [Online]. Available: http://www.empf.org/empfasis/2011/January11/energy.html. [Accessed 25 March 2015].

[3] "The Anatomy of a Protected LiIon Battery," [Online]. Available: http://www.lygte-info.dk/info/battery%20protection%20UK.html. [Accessed 25 March 2015].

[4] I. Buchmann, "Charging at High and Low Temperatures," [Online]. Available: http://batteryuniversity.com/learn/article/charging_at_high_and_low_temperatures. [Accessed 25 March 2015].

[5] A. Composites, "Nickel Zinc (NiZn) batteries are a great COST-EFFECTIVE choice for many applications.," [Online]. Available: http://www.automotivecomposites.com/batteries.htm. [Accessed 25 March 2015].

[6] ST Microelectronics, "Application Note: Lithium-ion solar battery charger," [Online]. Available: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/application_note/DM00048561.pdf. [Accessed 25 March 2015].