Erebus Labs

# *STEM Sensors*

# Project Report

Version 1

*6/11/2014*

# scott lawson

# Bryan Button

# Chris clary

# max cope

# VERSION HISTORY

|  |  |  |  |
| --- | --- | --- | --- |
| **Version #** | **Implemented**  **By** | **Revision**  **Date** | **Reason** |
| 1 | Bryan Button | 6/11/14 | Imported document from Google share to implement formatting. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**NOTE TO READER**

This is a template obtained from:

<http://www2.cdc.gov/cdcup/library/templates/default.htm>

Template Name: Product Design

**UP Template Version:** 12/31/07

Table Of Contents

[Abstract 5](#_Toc390274856)

[Introduction 5](#_Toc390274857)

[Solution Overview 7](#_Toc390274858)

[Hardware 7](#_Toc390274859)

[Software 12](#_Toc390274860)

[Roadmap for Rev. 2 28](#_Toc390274861)

[Lessons Learned 29](#_Toc390274862)

[Conclusion 31](#_Toc390274863)

Table of Figures

[Figure 1: Basic System Block Diagram 8](#_Toc390274864)

[Figure 2: RJ45 Connector 9](file:///C:\Users\bryan_000\Documents\GitHub\frack-n-sensor\Documents\source_documents\Report.docx#_Toc390274865)

[Figure 3: Sensor to System Block Diagram 9](file:///C:\Users\bryan_000\Documents\GitHub\frack-n-sensor\Documents\source_documents\Report.docx#_Toc390274866)

[Figure 4: Low Oxygen Sensor Schematic 11](file:///C:\Users\bryan_000\Documents\GitHub\frack-n-sensor\Documents\source_documents\Report.docx#_Toc390274867)

[Figure 5: Low Oxygen Sensor Layout 11](file:///C:\Users\bryan_000\Documents\GitHub\frack-n-sensor\Documents\source_documents\Report.docx#_Toc390274868)

[Figure 6: Light Sensor Schematic 11](file:///C:\Users\bryan_000\Documents\GitHub\frack-n-sensor\Documents\source_documents\Report.docx#_Toc390274869)

[Figure 7: Light Sensor Layout 11](file:///C:\Users\bryan_000\Documents\GitHub\frack-n-sensor\Documents\source_documents\Report.docx#_Toc390274870)

[Figure 8: PSoC Creator IDE 12](#_Toc390274871)

[Figure 9: Basic Firmware Flow 14](#_Toc390274872)

[Figure 10: USB Flow 16](#_Toc390274873)

[Figure 11: Wear Leveling Overview 23](#_Toc390274874)

[Figure 12: User Software GUI 26](#_Toc390274875)

# Abstract

Current educational curriculum in the US lacks a focus on Science, Technology, Engineering, and Mathematics (commonly referred to as STEM). Resistance to STEM begins in primary education classrooms where 50% of 8th graders have a hostile attitude towards math and science. Furthermore, according to the United States department of education, only 16% of American high school seniors are proficient in mathematics and interested in a STEM career.

Our goal was to create a device that would allow students, or anyone who is interested, the ability to easily measure the world around them. Getting kids interested in the ideas behind science and engineering is the first step to getting them interested in a STEM career later. We wanted to make data collection as plug-and-play as possible for simplicity, while still allowing access to internal hardware and software for those who have a desire to dig a little deeper.

# Introduction

## Basic STEM Intro

Science, Technology, Engineering and Math (commonly referred to as “STEM”) is an overarching category applied to all sorts of careers, fields, and interests that are highly technical in nature. This means that they are distinct from the arts and professional focuses such as philosophy or business.

STEM careers are generally considered drivers of industry; as such they are viewed as an important measure of economy. STEM fields introduce new technologies, and uses for those technologies, which are exploitable for both the improvement of business and mankind. Thus having a large number of people working in STEM jobs is very important for the growth of and betterment of mankind.

## Guiding Principles for the Project

### Ease of use

#### User Provided Hardware

Although the device is intended to ship to the user with everything necessary for operation, certain components such as batteries and cables may be lost, worn out or damaged by students. The desire to make replacements for these components easily accessible and inexpensive heavily influenced design decisions.

#### Minimize Development Complexity

A major goal of the final product is to get students interested in the engineering process and possibly design their own sensors. When multiple development software options existed, the simplicity of each software option was heavily weighted to support this goal.

#### Minimize Final Product Complexity

Electronics inherently contain some complexity, but an effort was made to minimize this complexity to make the final product more accessible to students. The user interfaces both on the device and the user application were simplified at the cost of additional features.

### Accessibility

#### Free/Open Source Development Software

To support the goal of encouraging students to design their own sensors, the process of selecting development software prioritized software packages that were open source. If suitable open-source options were not available, the only alternatives considered were closed-source but free.

#### Code Hosting

All code and development resources were hosted in GitHub. The files were hosted privately during development but are slated to be publicly released within two months, pending further review.

# Solution Overview

The device we developed has three distinct elements: a base station which collects data, a sensor module which facilitates the type of data being collected, and a Graphical User Interface (GUI) by which the user of the device may interact with the base station.

## Base Station

The base station (or just base) itself has a minimal user interface for collecting data; a couple of buttons and an LED generating system feedback for the user. The base is a stand-alone device that does not require a computer connection to operate. It was designed to be powered by AA batteries and left in a remote location for long periods of time to collect samples from the environment based on its configuration and the sensor connected to it.

Designed with accessibility in mind, the base was not designed to be obfuscated or difficult to learn. Instead, our hope was that it WOULD be inspected and that it would generate an interest in how it functions. It was also designed for flexibility and to be expandable. As will be discussed later, the base is not limited to only one sensor module; it is in fact ready to accept three additional sensor ports, though only one port was included in the initial build.

## Sensor Module

The sensor modules are incredibly simple attachments that interface with the base station by means of a sensor port. The port itself utilizes a type of cable to is flexible, robust, and widely used and available. The cable can also be custom made to a desired length which allows the user to

# Hardware

## Introduction and Diagram

One of the key goals in the makeup of the hardware was to maintain ease of access, while providing a flexible platform that is upgradable with minimal to no alterations. Every part that the user interacts with was to chosen with the user’s ability to gain access to, and operate, without requiring specialty stores or high cost. Keeping the future in mind, and that the system is built for continuing iterations, meant that the platforms adaptability needed to be accessible to a wide range of ages and knowledge backgrounds. Those at the introductory level will be able to operate the system and gain knowledge through simple data collection and interpolation. Meanwhile, those in more advanced stages would easily be able to make new sensors, using their preferred layout tool. Cadsoft’s free version of Eagle Cad was used to create the base station and sensors that come with the current build. Moreover, the interface allows for up to four analog sensors inputs and an I2C bus for the user to implement more complex sensor modules.

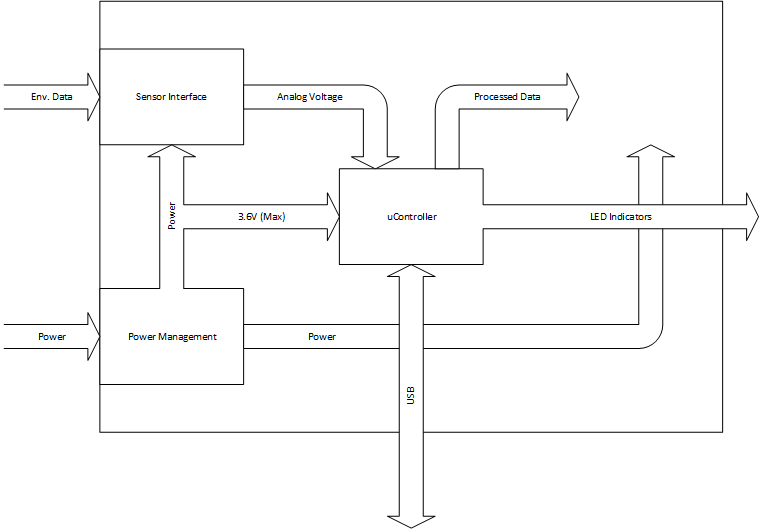


Figure : Basic System Block Diagram

## Components

### Power Supplies (including discussion of battery choice)

The power method chosen was four standard AA batteries. While proprietary batteries offer better longevity, their overall cost and limitations to the end user made them a secondary option.

For power regulation, switching regulators were chosen for their ability to both buck and boost the incoming battery voltage. This quality became appealing when realizing linear regulators would only buck the battery voltage and would function within a small window of operation. With the shorter life of the AA batteries, this could potentially render the device useless for any long term continuous data collection.

## Interface Methods

### USB

The choice of interface for the user to base station communication was USB. Every laptop and desktop in production today is fitted with several USB ports, with many unused most of the time. This means the user does not need to provide or purchase a port-to-port extension or other expensive interfacing tool that would get the collected data to something accessible.

### RJ-45

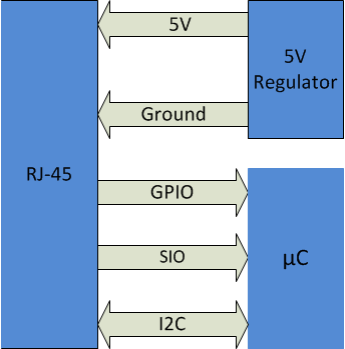
The interface between the base station and the sensor is RJ45, commonly referred to as ethernet. A key thing to point out here is that the ethernet protocols are not being implemented and to reduce confusion, will be referred to as RJ45. The main reason for choosing this kind of interconnection was due to it being an eight line adapter. With two lines being used for ground and power, there are six available lines open to the user. There are two primary single ended connections to microcontroller ADC mux. Also, a dual ended input to the microcontroller's comparator is supplied and can also be used as two additional single ended inputs sent the ADC mux. The final two lines are attached to the I2C bus on the base station. RJ45 cable is implemented with four twisted pairs and when used with any of the three previously mentioned sets of two differentially can greatly reduce noise transmission back to the base station. This set up provides a very high degree of flexibility in development of new sensor modules by way of the various combinations of the six RJ45 lines. 

Figure : RJ45 Connector

## PSoC 3

*Architecture*

*One-Chip solution*

*Dev kit*

## User Interface

### Buttons

A very simple two button start-stop system was implemented for field deployment. The ability to start and stop data collection at the desired site was meant to constrict the environmental errors added to the data set that can occur from moving the collection point around during sampling. This can create outliers in the data that distorts the results.

### RGB LED

A single LED notification system was implemented to simplify understanding of the current state of the base station. The LED is capable of the entire color spectrum but is will use only distinct colors to reduce confusion. The LED will blink green to notify sample process has started, and red when it is stopped. Also, it will blink blue every time a sample is collected. By using this kind of LED, distinct colors can be used for warnings such as memory full, low battery, or base station is connected to a computer and ready for data dumping or programming of sample parameters.

## Board Layout

Mix signal ground planes: Other than the analog sensors, all the peripherals that communicate with the base unit’s microcontroller operate digitally. Sensors that opt to use the I2C bus will use digital signals. With so much digital noise being introduced to the to the base unit board, much care needed to be taken to insure to signal integrity of the analog sensors. This was done in two significant ways.

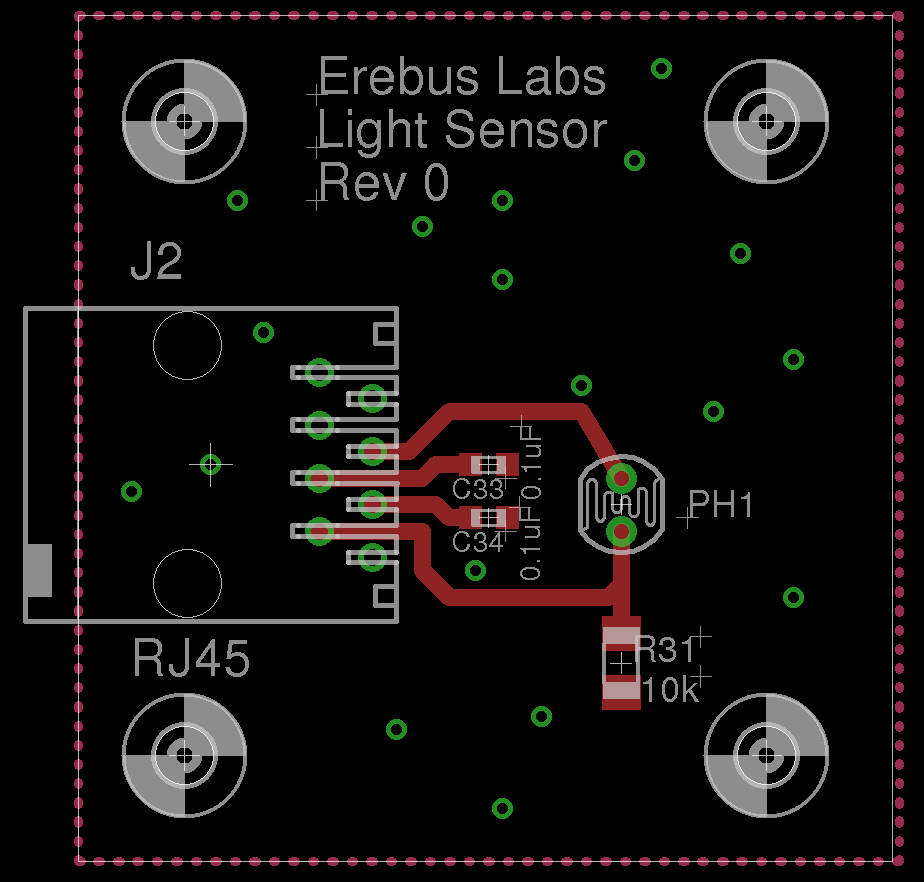
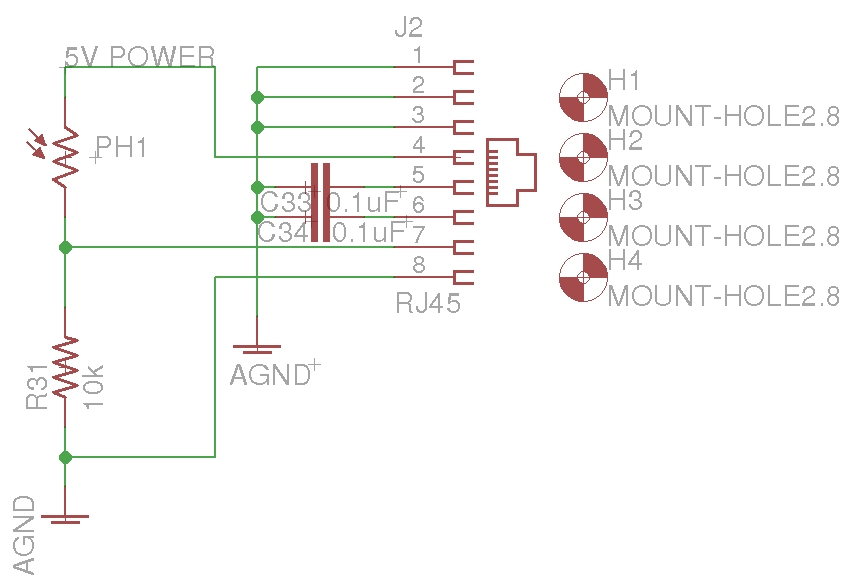
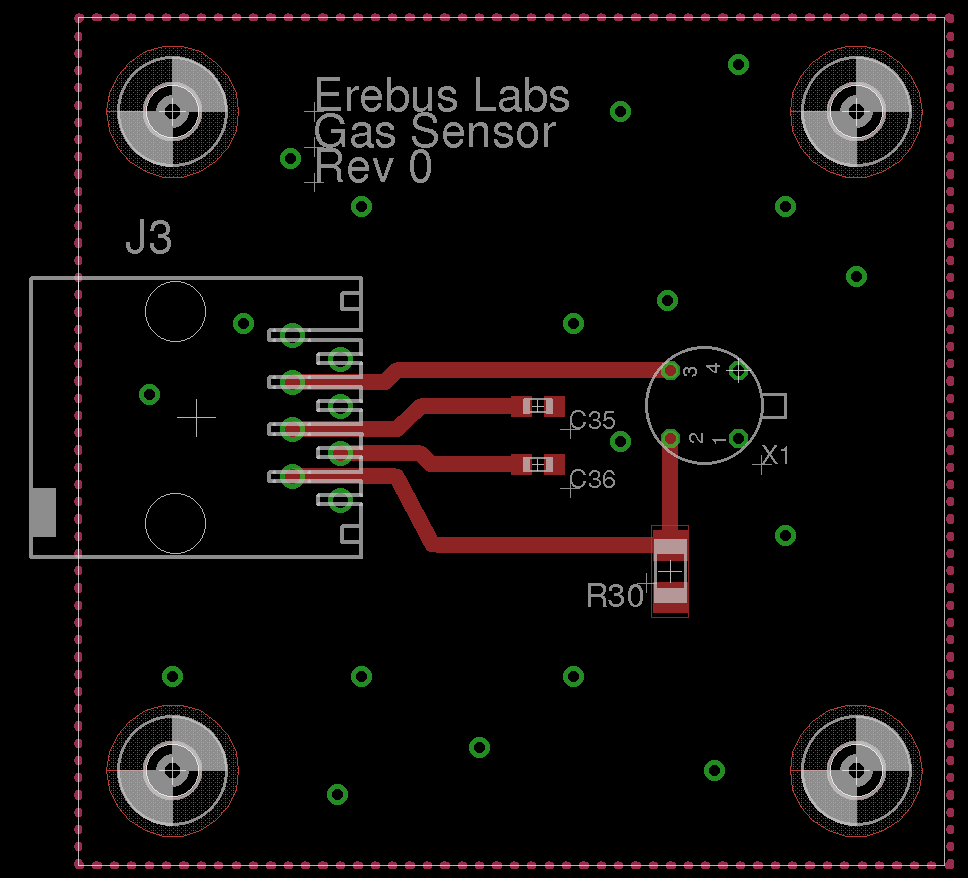
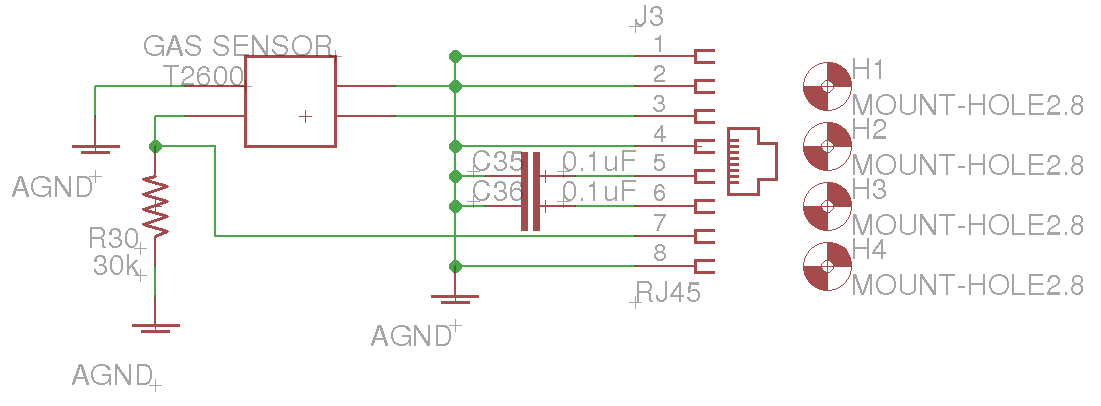
The first was to isolate the analog and digital ground paths. There is a small analog ground plane that wraps around all the analog pins of the microcontroller. Much is the same for the digital pins. Both return paths join as a star ground that connects to the negative battery terminal. This helps to maintain individual signal paths and reduce spurious noise that can be introduced from having a single ground plane with all access routing back to the lowest potential.

The second way noise was reduced on the analog line was to create differential signals in traces close together. In most cases where there a long traces on the printed circuit board, the input traces are plotted next to the output traces and interwoven as much as possible. This is done to reduce the inductive coupling and the resulting cross talk noise. Much is the same in the way the RJ45 cable was implemented, i.e. the ground and power are in the same twisted pair or how the other three sets of twisted pair can be used differentially.

## Sensors

To extrapolate on the ease of use theme, two resistive sensors were used in a simple voltage divider topology. One was a basic photocell while the other was the Figaro 2602 oxygen sensor. Both could be used in a wide variety of classrooms due to their easy theory of operation relative to other sensors on the market.

Both sensors were laid out on boards much smaller than the base station for more portability.



# Software

## IDE

The development environment used to develop firmware for our device and program the PSoC 3 was Cypress’s PSoC Creator software. This software is full-featured and includes support for development, programming, and debugging. It also comes with individual documentation for the APIs included. One of the reasons we chose to use the PSoC 3 chip in the first place was this software for reasons we shall now discuss.

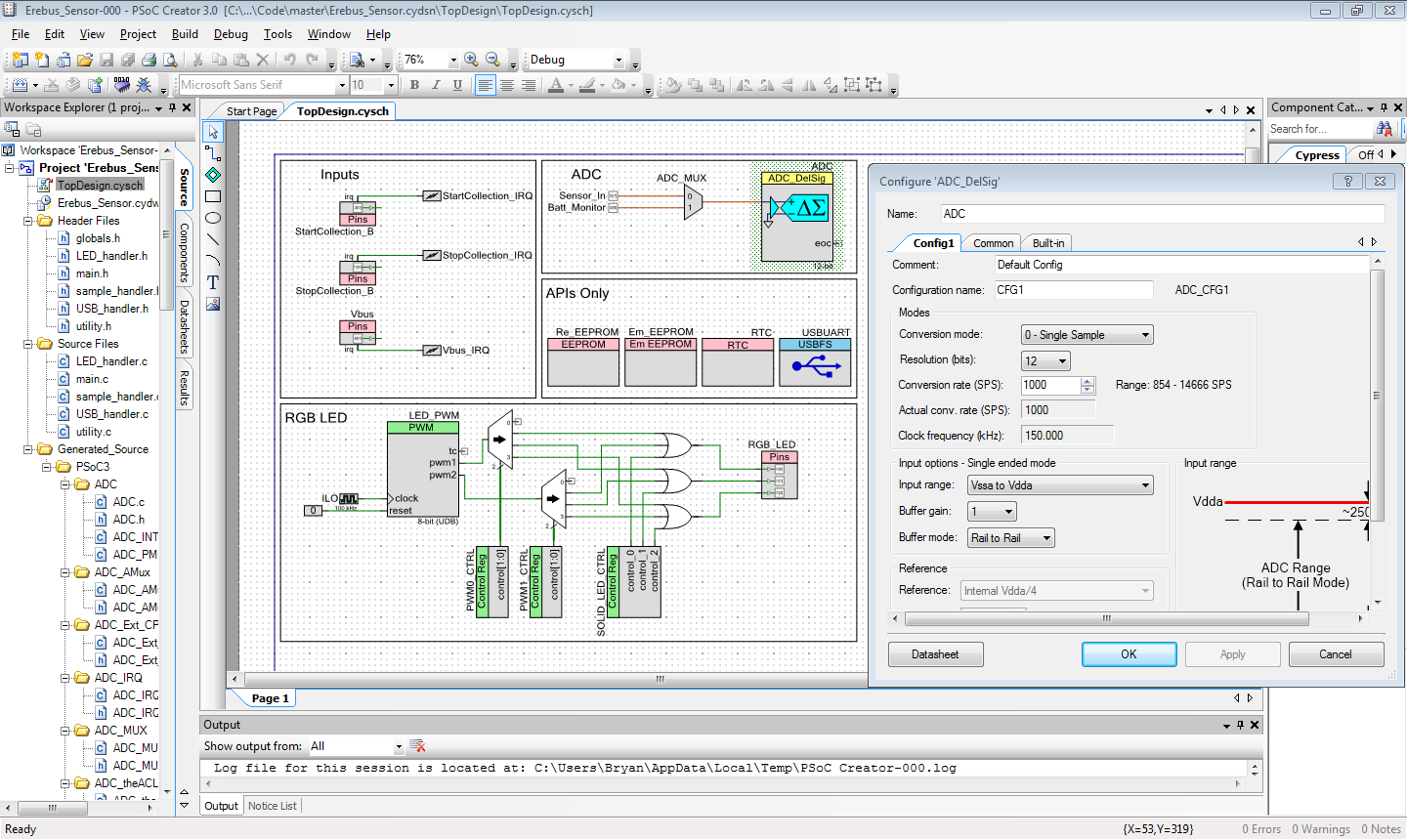


Figure : PSoC Creator IDE

The first and foremost reason was due to price of the software: free. Because the focus of this project was to get our device into the hands of students and anyone who is interested, we wanted to make sure that using the device didn’t have extra costs to it such as an expensive IDE. While the PSoC Creator software is not open source, its availability to anyone who wants it is the next best thing considering how comprehensive and powerful it is.

Another reason we like the PSoC Creator was the fact that it emphasized drag-and-drop programming. Creator features two different graphical programming methods: the first is the “schematic” method. This method allows you to add features to your firmware without initially requiring you to code anything. For example, one component we included in our code was the “Delta Sigma” or DelSig Analog to Digital Convertor (ADC). This component allowed us to take readings from the connected sensor, but requires a lot of configuration and additional components such as a multiplexer. Using Creator though, we were able to drop the ADC part onto our schematic, add a multiplexer to it, and route it to specific pins, all without needing to write a single line of code. In fact, just by double clicking on the part, a user has the ability to further customize the parameters of that part (if available) without needing to code anything.

The second graphical method used in Creator is the system configuration portion. This area allows users to customize the built in functionality of the chip itself and works together with the schematic method. For example, if you were to place a pin in the schematic, you would use the system configuration area to assign that pin to an actual physical pin on the chip. This method also allows you to configure things such as interrupts, clocks, EEPROM, flash, and other onboard components.

Of course, some code still needs to be written (and in our case a lot!). This code is what brings together the changes implemented in the graphical programing methods. When a user places parts on the schematic, a C file and/or header file are generated with the name of the part placed. This is the code that actually implements what the user did graphically. Most of this code is not user editable, as recompiling will overwrite any changes to these files. There are sections, however, that are set aside for user input. An example is the Real Time Clock (RTC) which we utilized for sampling and time-stamps. The RTC can be used to generate interrupts with very specific timing. In order to use those interrupts, we had to write our own code in one of the Creator generated files. Normally work like this would require a lot more code as well as writing to registers and other work which would have been time consuming to implement. With Creator, however, we were able to get this working rather quickly.

Which brings us to the last reason we chose to use the Cypress software for our project: time. With only 6 months to build an acquisition module with sensors from scratch, we needed to make sure that the tools we used would help us get through design and development as quickly as possible. Due to the nature of the discussed programming methods, we were able to accomplish this while using other environments might not have been so expedient. While it did speed up development time though, there are a lot of APIs available on this chip. Because of this, some information might be spread out over different documents or there might just be a lot to read through. So while writing all the code we did took quite a bit of time still, and while the is a decent learning curve associated with the Creator software, overall it met our requirements and helped us achieve our goals quite well.

## Firmware

The Erebus Labs STEM Sensors device firmware is written in standard C compliant with ISO/IEC 9899:2011 and developed in Cypress’ PSoC Creator 3.0. Upon reset, the controller operates as follows:

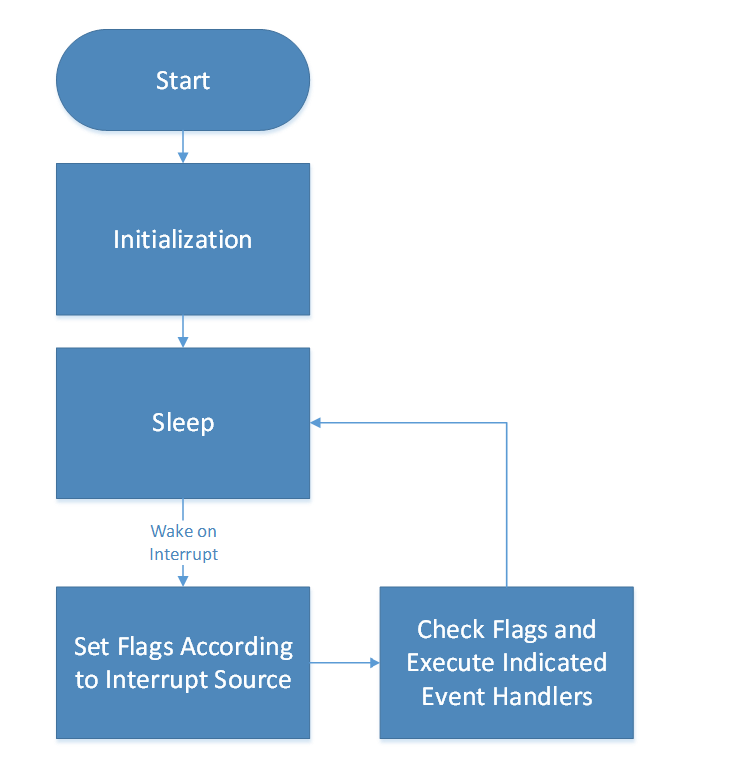


Figure : Basic Firmware Flow

## Initialization

The following initialization tasks are performed upon reset:

1. Retrieve data sample pointers from Flash
2. Start Components:
   1. EEPROM
   2. Real-Time Clock
   3. PWM Controller
   4. ADC
   5. Analog Multiplexer
3. Retrieve and apply sampling parameters from EEPROM
4. Enable VBus\_IRQ, ModifyCollection\_IRQ, and global interrupts
5. Enable sample initiation

### Main

The STEM Sensor’s main() function is responsible for putting the chip to sleep, restoring clock state on wakeup, and checking for flags that may have been set by an interrupt routine. Depending on flags set, the chip may either go back to sleep or call appropriate event handlers according to the flags.

### Interrupt Service Routines

The main program loop takes no actions besides going to sleep, waking, and calling event handlers when event flags are set. The event flags are set by the following interrupt service routines:

|  |  |  |
| --- | --- | --- |
| **Name** | **Trigger** | **Action** |
| VBus\_IRQ | Logic High voltage on VBUS pin | Activate USB Component, enumerate device on host, receive and store user settings, dump data samples to host, restores entry state on exit |
| ModifyCollection\_IRQ | Either start or stop collection button press | Examine PICU to identify interrupt source, enable or disable sampling accordingly |
| RTC\_EverySecond  Handler | RTC Every Second Periodic Interrupt | If sample unit is seconds and sampling enabled, increment counter; if counter == sample interval, take sample. Increment low battery blink counter; if battery power is low and blink count == blink interval, blink low battery LED indicator |
| RTC\_EveryMinute  Handler | RTC Every Minute Periodic Interrupt | If sample unit is minutes and sampling enabled, increment counter; if counter == sample interval, take sample. Increment battery check count; if battery check count == battery check interval, check battery level |
| RTC\_EveryHour  Handler | RTC Every Hour Periodic Interrupt | If sample unit is hours and sampling enabled, increment counter; if counter == sample interval, take sample |
| RTC\_EveryDay  Handler | RTC Every Day Periodic Interrupt | If sample unit is days and sampling enabled, increment counter; if counter == sample interval, take sample |

### Event Handlers

#### Handler Listing

The following event handlers are called in response to flags set by the interrupt service handlers:

|  |  |  |
| --- | --- | --- |
| **Handler Name** | **Flag** | **Set By ISR** |
| Run\_USB() | USB\_Waiting | VBus\_IRQ |
| take\_sample() | take\_sample\_waiting | RTC\_EverySecondHandler or  RTC\_EveryMinuteHandler or  RTC\_EveryHourHandler or  RTC\_EveryDayHandler |
| start\_collection() | start\_sampling\_waiting | ModifyCollection\_IRQ |
| stop\_collection() | stop\_sampling\_waiting | ModifyCollection\_IRQ |
| check\_battery() | battery\_check\_waiting | RTC\_EveryMinuteHandler |

#### Handler Flow of Execution

##### Run\_USB()

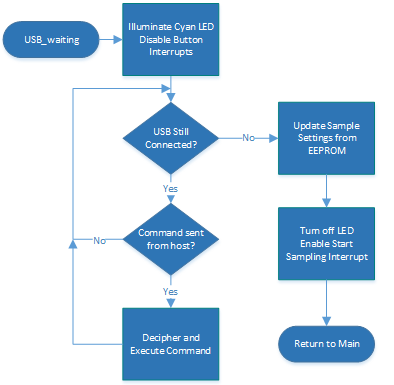
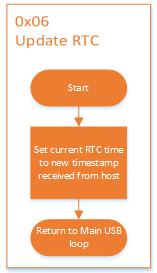
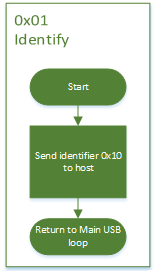
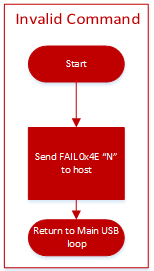


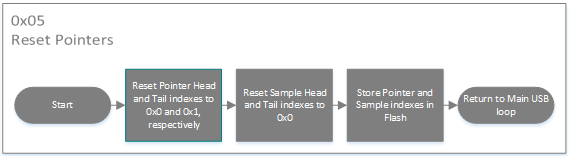
Figure : USB Flow

While the base unit is connected to a host system via USB, it continually polls the USBUART buffer checking for commands from the host. If a command is received, it is deciphered and the device takes the appropriate actions according to the corresponding flowchart below:

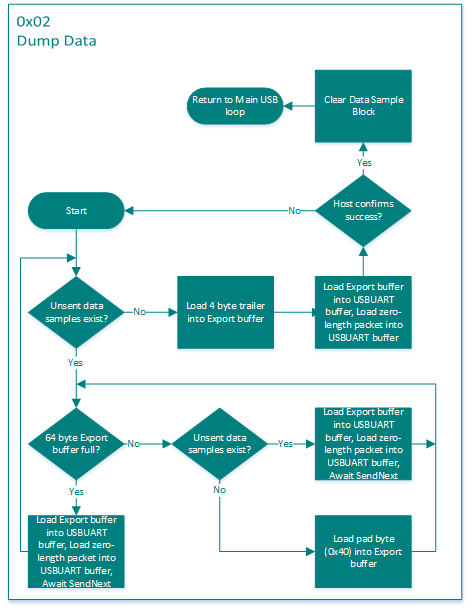
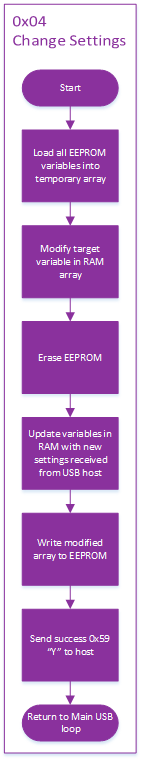
###### Invalid Command, Identify, Update RTC



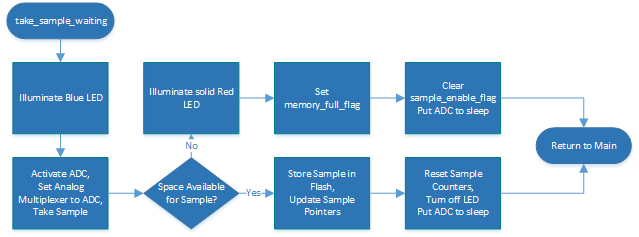
###### Reset Pointers



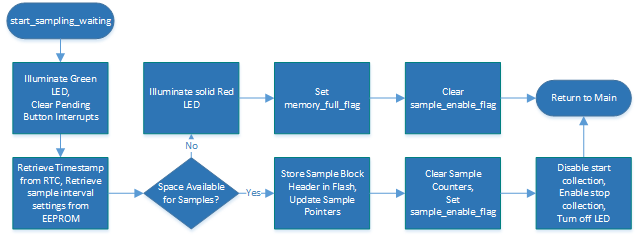
###### Change Settings, Dump Data



##### take\_sample()



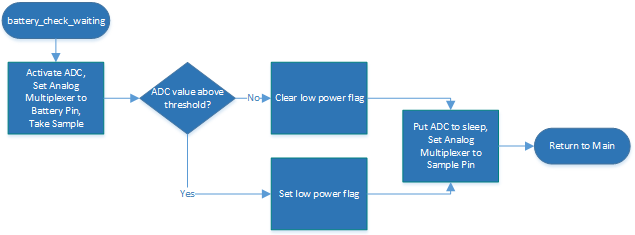
##### start\_collection()



##### stop\_collection()



##### check\_battery()



### Sampling

#### Sample Blocks

Data samples are stored in Flash memory along with program code by utilizing the Emulated EEPROM component available for the PSoC3. Each section of data samples begins with a 16-byte header (HE). The header contains information about the sensor used, the sample period, the sample interval, and a date/time stamp from when data collection was started. There are also four reserved bytes at the end of the header to force the header to be 16-byte aligned. A new sample block is initiated every time the user presses the Start Sampling button if the sensor is not currently collection samples.

##### Header Format

|  |  |  |
| --- | --- | --- |
| **Bits** | **Number of Bytes** | **Description** |
| HE[127:112] | 2 | Unsigned integer containing sample block start signature 0x200 |
| HE[111:96] | 2 | Unsigned integer indicating sample start year |
| HE[95:88] | 1 | Unsigned integer indicating sensor used  0 = Light Sensor  1 = Low Oxygen Sensor  2 = Custom Sensor |
| HE[87:80] | 1 | Unsigned integer indicating sample unit: must be in the range [0:3]  0 = Seconds  1 = Minutes  2 = Hours  3 = Days |
| HE[79:72] | 1 | Unsigned integer indicating sample interval |
| HE[71:64] | 1 | Unsigned integer indicating sample start second |
| HE[63:56] | 1 | Unsigned integer indicating sample start minute |
| HE[55:48] | 1 | Unsigned integer indicating sample start hour of day |
| HE[47:40] | 1 | Unsigned integer indicating sample start day of month |
| HE[39:32] | 1 | Unsigned integer indicating sample start month: must be in the range [1:12]  1 = January  2 = February  3 = March  Etc. |
| HE[31:0] | 1 | Reserved: write as 0x00000000 |

##### Data Sample Format

The data samples themselves are recorded in a 2-byte bit-field (DA) comprised of the 12-bit ADC output and a 4-bit reserved field:

|  |  |
| --- | --- |
| **Bits** | **Description** |
| DA[15:12] | Reserved: write as 0x0 |
| DA[11:0] | Unsigned raw ADC output |

During sampling, the “reserved” field of the sample should be written as 0x0. The data dump routine in the USB Handler uses these bits to communicate information to the host during dumping, such as End of Data or No Data Present.

#### Sample Exporting

When data sample blocks are exported to the host system over USB, the current block of data samples and headers are iterated over until the tail pointer matches the head pointer. If the end of the sample block is reached before a 64-byte packet can be filled, the remaining bytes are filled with the pad-byte signature (0x40) to indicate to the user application that those bytes do not contain sample data and should not be counted when calculating the total number of good bytes received.

The data export routine does no interpretation of the data it is iterating over; it simply loads bytes from the sample block into the USBUART buffer. After all data has been transferred, a four-byte trailer (TR) is sent to the host to confirm that the correct number of bytes were received.

##### Trailer Format

|  |  |  |
| --- | --- | --- |
| **Bits** | **Number of Bytes** | **Description** |
| TR[31:16] | 2 | Number of non-pad bytes transmitted |
| TR[15:8] | 1 | Pad byte: write as 0x00 |
| TR[7:0] | 1 | End of data transmission marker: write as 0x80 |

##### USB Packets During Sample Export

Host commands and device replies are sent in single byte packets. Data is transmitted in 64-byte packets, the maximum size allowed by the USB specification. Between each 64-byte packet sent, the firmware waits for the NEXT command (0x07) from the host before initiating the transmission of the next packet. This includes waiting after the last sample has been sent before sending the trailer.

After each 64-byte packet, a zero length packet is sent to the host to notify it that the sensor is done with that transaction and is awaiting the SEND NEXT command. Failure to do this causes the last 64-byte transmission to be held in a USB buffer on the host before being transferred to the Erebus Sensor application, causing both the firmware and user application to stall while waiting for data from the other. The design team was unaware of this requirement during development, and it caused seemingly unpredictable behavior of the export routine during early testing. Once the zero-length packet requirement was identified, data exporting behaved properly.

#### Wear Levelling

The use of the Emulated EEPROM component allowed us to realize the goal of minimizing the number of chips required on the main board, but it created certain challenges as well. The most significant challenge was the need for wear-levelling. Although the Flash memory in the PSoC3 is guaranteed for up to 100,000 write cycles, we wanted the base unit to be as durable as possible. To achieve greater product longevity, a wear-levelling scheme was implemented. The wear-levelling method employed is shown by the following diagram:

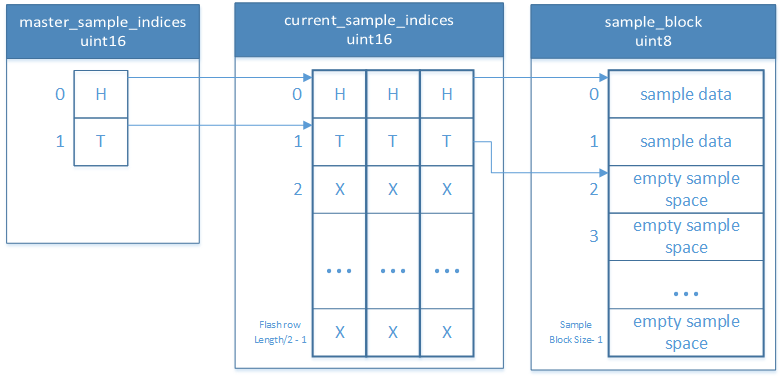


Figure : Wear Leveling Overview

### Both the master\_sample\_indices and current\_sample\_indices arrays are stored in flash so that stored data and array indices are retained in the event of power failure. The arrays perform the following roles:

##### sample\_block

The sample\_block array is an array of uint8’s (unsigned chars) and stores sample data and sample block headers. This array is only written to when beginning sampling (to store the block header) and when taking a sample. It behaves like a circular buffer, with the current head and tail indices stored in the current\_sample\_indices array.

##### current\_sample\_indices

The current\_sample\_indices array contains the indices of the sample\_block array that correspond to the current block’s head and tail pointers. As samples are taken and collection periods are started and stopped, the tail pointer is incremented through the sample\_block array. The head pointer does not move until the existing data is successfully exported to a host computer. When a data dump is successfully completed, the head index is set equal to the current tail index, effectively clearing existing data without subjecting the Flash memory to unnecessary erase cycles.

There are three pairs of head and tail indices in the current\_sample\_indices array that are separated a number of bytes equal to the length of a Flash row - 2. This arrangement ensures that the pairs of head and tail pointer pairs are in separate rows of Flash memory. Every time the head or tail indices are modified, the new index must be updated in the current\_sample\_indices array. The head and tail index pointers are in separate rows so that they can have their wear levelled as well.

##### master\_sample\_indices

The head and tail indices in the master\_sample\_indices array contains the index of the currently used head and tail index pair in the current\_sample\_indices array. As data headers and samples are stored in Flash, the current tail index in the current\_sample\_indices array is incremented, but the master\_sample\_indices array is untouched. Eventually, enough data will be collected that the current tail index is wrapped around the end of the sample\_block array because it is a circular buffer. When that data is dumped and setting the head index equal to the tail index requires the head to be wrapped around the end of the sample\_block array, the head/tail indices in the master\_sample\_indices are set equal to the next pair of head/tail indices in the current\_sample\_indices array.

##### Initialization

After the base unit has been reprogrammed, the indices must be reset to their default locations before the device can be used. To accomplish this, the base unit is connected to a host system and the Reset Device command is issued from the user application. From that point on every time the device is powered up, the master and current sample indices are retrieved from Flash allowing the device to store samples properly.

### Real-time Clock

#### Updating the Real-time Clock

When the current time of the RTC is to be updated, the current time is sent from the host to the base unit. It is sent in an array of bytes (CT).

##### Current Time Byte Array Format

|  |  |  |
| --- | --- | --- |
| **Bits** | **Number of Bytes** | **Description** |
| CT[55:40] | 2 | Unsigned integer indicating current year |
| CT[39:32] | 1 | Unsigned integer indicating current second |
| CT[31:24] | 1 | Unsigned integer indicating current minute |
| CT[23:16] | 1 | Unsigned integer indicating current hour of day |
| CT[15:8] | 1 | Unsigned integer indicating current day of month |
| CT[7:0] | 1 | Unsigned integer indicating current month: must be in the range [1:12]  1 = January  2 = February  3 = March  Etc. |

#### Real-time Clock Maintenance

The use of the on-board real-time clock created a unique challenge when trying to create a user-friendly interface. The RTC is reset to 01 January 1900 if power is lost. Since the RTC used is built into the chip, there was no way to provide a second power source to maintain the RTC while powering down the chip. This limitation restricted the final product’s firmware design because we were also unable to use the PSoC 3’s lowest power state, Hibernate, as that would also reset the RTC. To minimize the impact of the user being unable to fully power down the chip, the user application was configured to update the RTC with the host system’s current time every time the device is connected to the application. This solution is still less-than ideal though, because if the user needs to change the batteries, they still need to attach the base unit to a USB host and connect to it before taking any samples. If this is not done, the samples will still be collected properly, but the time stamp at the beginning of sampling will not be correct.

## User Application

The host computer user interface is a simple GUI that performs two functions:

1) Provide a method for the user to change data collection settings or reset the base

unit.

2) Allow the user export data from the device and present it in a text file. Python provides a platform-independent framework for a GUI and for interacting with the sensor.

Python was selected as the language for creating the application interface because it is free, open-source, and cross-platform compatible. Additionally, it is a very popular high-level language so there is a lot of community support for it. Students that are new to programming and application development can access many free resources and get questions answered on community websites such as StackOverflow.com. Python’s abundance of community support also enabled us to use reliable third-party libraries such as PySerial, rather than having to write all of our code from scratch.

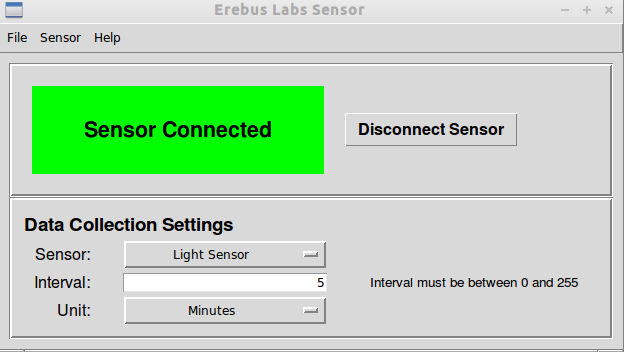


Figure : User Software GUI

One downside to the choice of Python is that until a proper installer is developed, launching the program is more challenging that simply double-clicking an executable. The right version of Python (3.2x) must be installed along with the PySerial and tkinter libraries, which are all obtained from different websites. Once Python and the libraries are installed, the host computer must be configured to execute the top-level script on double-click in the Python interpreter. If this configuration is not done, then the script must be launched from the command line.

Given more time, another Python library called cxFreeze could have been used to package the scripts into an executable that could run on a host without Python or any of the libraries installed, but limitations on the development time available prevented the team from having the time to utilize cxFreeze. Even if cxFreeze would have been used, the resulting executable still relies on multiple \*.pyd files (which are the Python equivalent of \*.dll files) remaining in the current working directory of the executable, cluttering up the program directory and potentially confusing novice computer users.

### Device Connection

User communication with the device is achieved through the use of the PSoC3’s Universal Serial Bus Universal Asynchronous Receiver-Transmitter (USBUART) component. When a USB cable is used to connect the base unit to a computer host, the USBUART component opens a virtual COM port (Windows) or virtual TTY port (Linux) that is accessible by the user application.

The PSoC3 USB component does provide full USB functionality, including device classes such as Human Interface Device (HID), Mass Storage Device (MSD), or Composite Device (combination of classes, such as HID and MSD in one device). However, none of the team members had any experience developing USB devices. Although the USB specification is freely available for implementation, the team decided that the complexity of a full USB implementation could not be achieved in the six-month time period allowed. In addition to firmware complexity, the user application would also increase in complexity with a standard USB implementation.

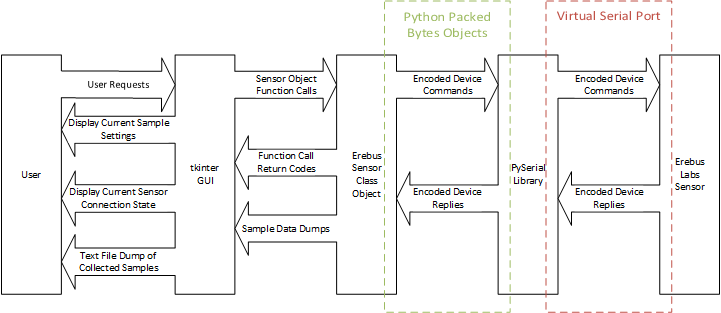
The USBUART provided a suitable replacement for a full-scale USB implementation. Some team members already had experience with serial devices and the complexity required on the firmware side was much less with the USBUART implementation than would be required for standard USB. For example, sending data to and retrieving data from the host is as simple as writing to and reading from a single buffer. On the user application side, the PySerial library required no configuration. Establishing communication with the base unit required only a handful of function calls.

### Operating system support

One downside to the USBUART solution is that it complicated support for multiple platforms. The final product is only supported on Linux. Windows support requires a driver provided by Cypress. During development, we had issues with the driver provided by Cypress; it would cause the host system to lock up and require a hard reboot when the device was plugged in and bound to the driver. Assistance was requested from Cypress, but they were unable to recreate the issue. Development on other aspects of the firmware and software continued in Linux, and the problem could not be resolved by final project delivery.

### Command Resolution

As commands are issued by the user, they must be translated into the byte-encoded commands recognized by the device. The following diagram illustrates the layers of translation the commands go through before being executed by the firmware:



# Roadmap for Rev. 2

As with any project, not every desired feature can be implemented in the first revision. In our case, the biggest factor defining what did and did not get included was the simple fact that we had such a short amount of time to implement our ideas. Though six months is a decent amount of time, a lot of that time in the beginning is spent on planning and research. Additionally, some of the ideas we had were not discovered quickly enough to implement them. So that the learning process we went through is not lost however, we are including here a roadmap of sorts for future work which may be done.

## Critical Battery Shutdown

While we did develop a method to detect when the battery level hits a certain threshold, we did not have a chance to implement any procedures for what should happen when that threshold has been reached (other than alerting the user through the LED). While we built the system to be robust in the event of a power loss or improper handling, we feel that adding critical battery power management to the system would be beneficial to mitigate any possible data corruption or other unforeseen issues resulting in critical battery level or power loss.

## Enclosure Mounted Buttons

Currently the user buttons for starting and stopping data are mounted on the board itself. While this allows for more flexibility as far as packaging is concerned, it also means that the board must be accessed in order to start or stop data collection. Should this device ever see a specific enclosure built for it such as a customized box, it might be beneficial to place the buttons (and possibly the LED as well), on the outside of the box for ease of access.

## Power-Wire Strain Relief

While working with the device in testing and assembly, one issue we ran into multiple times was that of the battery connection wires breaking off at the solder joint and becoming disconnected. This is not something we had thought about before this happened but was very obviously in need of a fix. For quick strain relief, we ended up drilling a couple of holes in the board near the power connections; in the end though, implementing something on-board would not only help to make the device look cleaner, it would also ensure that there is some sort of standard strain relief that goes along with the project as a whole.

## Weather Proof Box

While the device we built did include an enclosure, the enclosure itself is not at all close to being water-tight. We believe that a weatherproof box would allow for more flexibility in data collection as it would allow the device to be placed unattended in more types of locations. This type of box would be especially crucial for outdoor data collection or in areas with higher moisture.

## VOC Sensor

This project initially started out with the idea to build a sensor specifically for testing water quality in areas with large amounts of pollutants near water sources. Volatile Organic Compounds can be hazardous to humans, especially in large quantities, and providing a way for students to measure the water in their area would be not only educational, but relevant and a way to get them engaged in engineering and science which is what the goal of this project is now. By adding a VOC sensor, we could continue to accomplish our current goal while fulfilling the desire set forth in the original project description.

## Multiple Sensor Ports

Currently, as mentioned, the device we built has only one port by which a user can connect a sensor module. While more sensors could be added to a module, it would be beneficial to have multiple ports on the board as well. This would allow the same type measurement to be taken at the same time in multiple locations (a sensor array). One example of how this could be used would be to compare the heat outdoors in open air while simultaneously measuring the heat in an enclosure in the same spot. Another application would be taking acoustic pressure (sound) samples along a certain length.

## Jumper for Power

We did not include a power button for reasons discussed (see the RTC), but having the ability to remove power from the device would be quite beneficial. If work was being done on the device, or you wanted to store the device for a while without removing the batteries, having the means of easily doing so without risking users accidentally removing power would be helpful. For this reason, we suggest adding a jumper to the device. This would allow power to be removed conveniently, but not so easily done that it would be trivial to disconnect power inadvertently.

# Lessons Learned

## One-Chip Solution

One of the biggest takeaways of the project was that of what it means to design a device using one chip. Of course there are a couple power regulators on the main board, but all of the actual functionality of the device was contained within the one PSoC 3 microcontroller. While this meant that we have the flexibility to easily add things like additional sensor ports in the future, it also means that we are limited by one chip as well.

One example of this limitation can be seen with the RTC. Currently the RTC is a function of the PS0C 3 and as such is powered by the same lines as the PSoC. This means that when the device is completely powered off or even in its lowest power mode (hibernation) the power to the RTC is lost as well and all RTC data is lost as well. When this happens, the clock on the device is reset and timestamps are no longer accurate. Unfortunately the fact that the RTC is a part of the controller means that we cannot power the RTC only, whereas there are dedicated RTC chips on the market which would allow us to use an on-board backup battery (like a button cell battery) to keep the RTC running even while the board itself is powered off.

Another example of how breaking components off onto their own chips is the EEPROM. Normally wear leveling is not a huge issue with EEPROM as it is with flash. Due to the fact that our EEPROM is emulated within flash though, wear-leveling is something that we had to take into consideration which added additional development time. If we were to use a separate EEPROM chip not only would we be able to avoid wear-level implementation, we could also more easily change the capacity of the EEPROM later without the need to change out the entire controller.

## Other Thoughts

Individually we each had our own takeaways as well. Through this project we learned things such as USB implementation, wear-leveling (as mentioned before), embedded systems programming, proper design technique, and various other technical things. We also got a feel for the entire design process going from nothing to a working product.

Overall this project turned into a learning experience that help us not only hone our skills in hardware and software development, but additionally helped us understand how important it is to start out with firm and understandable goals. Had we not thought carefully about this project from the start and about who our target audience is, it could have easily turned into a device that was just like all the other data acquisition devices out there: difficult to use, heavy in programming, and geared toward a more technical audience. Instead, we have been able to deliver a device that we all believe can truly help those just starting out become interested in the fields we find so fascinating.

# Conclusion

All in all, this project was quite successful. In the end we were able to deliver a device that was simple to operate and modify and which allowed users access to the raw data collected. Below are the requirements we initially set for the project.

Requirement Met, ~~Requirement Removed~~, Requirement not tested, Requirement not met

## Marketing Requirements

1. *The system must be modular, allowing multiple sensor types to be employed by one base unit.*
2. *The system must be low cost.*
   1. The initial cost of the entire project ended up being larger than hoped for or anticipated. In the future, though, we believe that the cost could be brought down.
3. *The system must be simple enough to operate that K-12 teachers and students can effectively utilize the device in the classroom*
   1. We created a couple of simple experiments that could be performed in a classroom easily.
4. *The user interface must accommodate both novice and advanced users.*
   1. The UI is simple to use, but allows access to RAW data which is useable by both amateurs and professionals, though this device is meant to be for learning and not scietific data collection.
5. *Base unit with sensors attached must be able to operate outside of a classroom.*
   1. System is battery powered and is standalone.
6. *The system must provide a method for users to specify the data collection interval.*
7. *Hardware and software designs should be open-source.*
   1. All of our code and board design materials are available on GitHub.
8. *The base unit should be able to collect data for an extended period without user interaction.*
9. *The base unit should require minimal changes or adjustments for the use of different sensors.*
   1. While the GUI facilitates choosing what sensor is plugged in, it is only for labeling. The base station does not discriminate between devices.
10. *The system should be durable.*
11. *~~The base unit may also be modular and require user assembly.~~*
12. *The base unit may support simultaneous multiple sensor attachments.*
    1. While we did build in support for additional sensor interfaces in the future, we did not include multiple port in this release.
13. *The user interface should provide a method for organizing and comparing data from multiple base units employed simultaneously.*
    1. We decided that comparing data could be done by the user outside of our GUI and did not need to be included. This is something, however, that could be included in a future update.

## Engineering Requirements

1. All sensors must use the same interface to connect to the base unit
2. The user interface must provide a method for the user to access the raw data collected
   1. The GUI dumps all data to the user’s computer as a plain text data file.
3. A publicly-accessible repository must be used for code and documentation hosting
4. If third-party software is used, it must be open-source
   1. We ended up using CadSoft’s Eagle program for board layout and Cypress’s PSoC Creator, neither of which are open source. As they are at least freely availbe, however, we felt that this was sufficient.
5. The base unit with sensors attached must operate when exposed to temperatures between -10°C and +70°C
   1. According to datasheets everything on our boards met this requirement. As we were not able to fully test this though we could not say that it had been fully tested.
6. If the system is does not use a rechargeable power source, it must not use proprietary battery types
7. BOM for base unit should not exceed $20.00 each
8. BOM for sensors should not exceed $5.00 each
9. The base unit should identify the sensor(s) attached and configure itself appropriately
   1. The GUI is needed in order to specify which sensor is attached. This is only for labeling, however, and the system does not care about what is connected.
10. The system should be able to collect data points at rates between 1 Hz and 1 per day
11. ~~The system should be able to coordinate data collection between 6 base units simultaneously~~
    1. This requirement was removed as the multiple sensors requirement was more what we were looking for.
12. The base units and sensors should be operational after a 1.5m drop-test
13. The base unit with sensors attached should operate when exposed to temperatures between -20°C and +80°C
    1. Same issue as requirement 5.
14. The base unit should be able to collect data points for 90 days without user interaction
    1. We were not able to collect good battery drain data. Based on the data collection we did do, the system should still last for a long while even if not for 90 days.
15. ~~A wireless data dump interface should be utilized by the base unit~~
16. ~~If a wireless data dump interface is utilized by the base unit, it should not require the user to be closer to the base unit than 3 meters~~
17. The base and sensors may be constructed with a water-resistant case
18. The base unit may contain multiple attachment points to enable multiple sensors to be used simultaneously
    1. As mentioned previously, we ended up not including multiple sensor attachments. But we did leave the system open for them to be added later
19. The base unit may use sockets and connectors to attach the controller, power, and communications devices to the PCB
    1. Sockets are used for everything, and the battery case is permanently attached.

Appendix A: Available Documentation

The following documents are available for users or future groups as resources.

## User Documentation

### Technical Reference Manual

This manual is a full technical manual and discusses the details of how the device works. It is meant to be used by advanced users or those wishing to learn more about how the device operates and how it can be modified.

### System Architecture

A basic overview of the architecture of the system. While not as technical as the documentation for the PSoC chip, it gives enough technical information to understand how the system is connected and how it is run.

### User Manual

This is a basic user manual meant to assist users in getting started with the software and hardware. It is non-technical and can be used by anyone.

### Fundamentals of Electronics

A quick intro into the fundamentals of electronics on which some portions of the system (including the sensors) are based.

## Project Documents

### Design Spec

This document is similar to the Technical reference manual, only it goes into further detail to include detailed diagrams, descriptions, and flow charts of the most inner workings of the system.

### Budget

This document breaks down the costs of the project for those interested in pricing or to use for future estimates

### Timeline

The timeline our team followed to complete the project within the six-month time frame allowed.

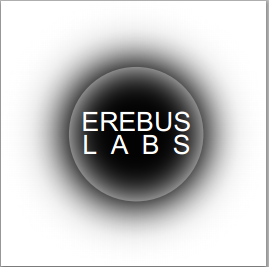
### Requirements

Marketing and design requirements. Our goals for the project which were discussed previously.

### Test Plan

Testing performed to ensure that the system operated as designed.

Appendix B: User Manual

**System Requirements**

To communicate with your new sensor, you will need to a version of Linux, Ubuntu is suggested. You can get it by following the link below.

http://www.ubuntu.com/download

Also, you will need to upgrade the stock version of Python, that came with Ubuntu, to version 3.4.1 by following the link below.

<https://www.python.org/downloads/release/python-341/>

Next, you will need to add the tkinter package from

<http://www.tcl.tk/software/tcltk/download.html>

and download and install version tk8.6.1-src.zip. Lastly, you will need to have pyserial which can be found with the following link.

<https://pypi.python.org/pypi/pyseria>

1

#### Commonly Asked Questions

How do I see what my current sampling settings are? You can check by connecting the base station to the laptop/desktop, hit the “Connect to Erebus Sensor” and wait for the connection icon to turn green. Now, go to “Sensor” in the tool bar Sensor🡪Get Current Configuration and click. The “Data Collection Settings” will be populated with the current base unit settings.

Can I change the sampling setting in the field? No, not unless you have brought a laptop to attach to the base unit.

How do I know what sensor I am using? The sensor type is printed on the top of the circuit board.

What if the timestamp on my data dump says it’s the year 1900? The clock that tracks the current date is reset if the power is disconnected. Reconnect power to the base unit, plug it into a laptop or desktop and connect to it through the Erebus Sensor program. This will reset the clock.

Is there any additional support? Yes, the help tab in the tool bar of the user interface will bring up options for the user guide or technical manual.

What if the base unit is not responding to configuration changes? You can go to sensors in the tool bar Sensor🡪Reset Sensor and click. WARNING: THIS WILL ERASE ALL DATA CURRENTLY STORED ON THE BASE UNIT, ONLY DO THIS WHEN ALL OTHER OPTIONS HAVE BEEN EXHAUSTED.

10

## How to Retrieve Your Data

1. Remove the lid from the base station and push the Stop button.
2. Replace the lid and bring it back to your desktop/laptop.
3. Insert the USB cable as before. You should see the solid blue teal LED. If not, unplug it and try again.
4. Bring up the user interface on your desktop/laptop as before and click the “Connect to Erebus Sensor” button.
5. Once, the connection icon has turned green and says “Sensor Connected”, click on sensor from the tool bar Sensor🡪Get Data and click.
6. The data will be stored in a file called “datadump.txt” in the folder that you launched the program from. If the file already exists, new data will be added to the end of the file.
7. The data dump file will be the raw data and needs to entered into Matlab or Excel to be interpreted.

9

Table of Contents

System Architecture……………………………………3

What is in the Box……………………….……………….5

Getting Started…………………………………………..6

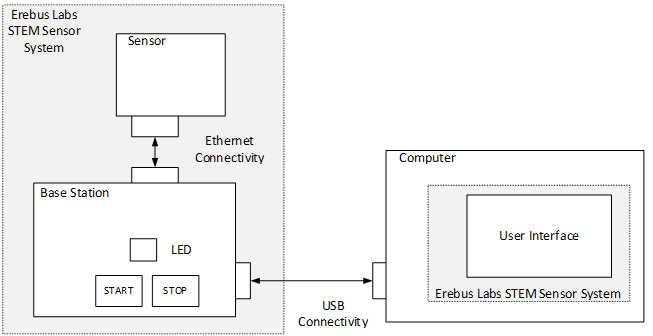
What do the Colors Mean………………………………8

How to Retrieve Your Data……...…….………………9

Commonly Asked Questions……….……….……….10

2

#### System Architecture



**System**

The operational product Compromised of the base unit with attached sensor and a user interface.

**Base Unit**

The central device that manages power, communication, data storage, and has one or more sensors attached to it.

**Sensors**

The individual data collection devices such as a gas detector or ambient light detector

**User Interface**

The program that will run on a laptop or desktop computer that allows the user to view and interact with the data collected.

3

**What Do the Colors Mean**

Solid Teal: USB is connected to both laptop/

desktop and base station are connected.

Blink Blue: Sample is being taken.

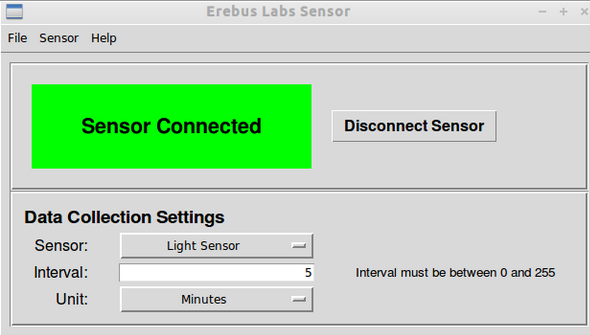
Blink Green: Sampling process has started.

Solid Red: The memory is full and needs to be dumped.

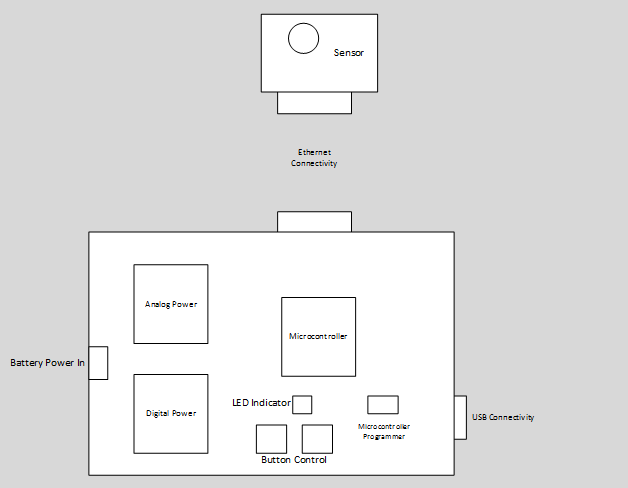
Blink Red: Sampling process has been stopped.

Blink Yellow: The battery is low and needs to be changed.

8

1. If the user interface has connected to the base unit the yellow square will turn green. 
2. If you do not see this screen repeat step 3.
3. Select the sensor you wish to use from the drop down menu “Sensor”
4. Enter a number between 0 and 255 in the “Interval” text box.
5. Choose the time unit for the interval (i.e seconds, minutes, etc.) in the “Unit” drop down box.
6. The next step is click sensor in the toolbar Sensor🡪Apply Setting and click.
7. You are now ready to disconnect base unit from your desktop/laptop.
8. Finally, connect the sensor to base station with the provided Ethernet cable, and take both to the location you wish to collect data, remove the cover, hit the start button, and replace the cover If you see the LED flashing blue at the correct interval, you are set to go.

7



**Power**

There is a 3.3V supply to run all digital peripheral. Also, there is a 5V supply to run the analog peripheral, as well as directly power the sensor.

**Microcontroller**

This is where the sample intervals are stored. The initiation of data collection and the storage of data is done here.

**User Control**

User Controllability with two buttons, start and stop sample collection, and an LED indicator

4

### What is in the Box

1Base Unit …………….



1 Ethernet Cable…….



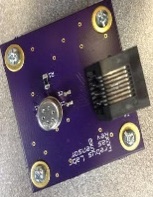
1 USB Cable………….



4 AA Batteries……….



1 Photocell Sensor…..

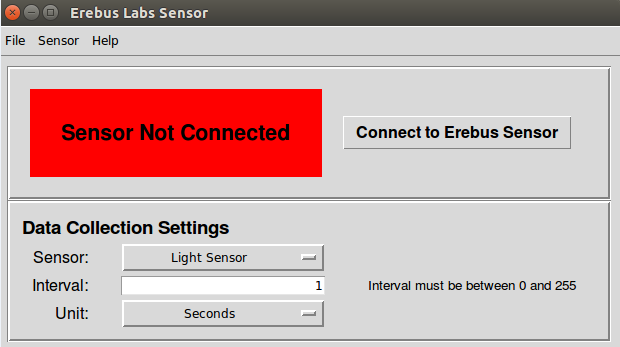


1 Gas Sensor…………...

5

**Getting Started**

1. Start up the Erebus Labs Sensor interface by opening terminal and enter navigate to the director containing erebus\_sensor.py. Enter the command “python3 ./erebus\_sensor.py”. The program will launch as shown below.



1. Take the USB cable and plug the mini in to the base unit, then plug the standard size end into your desktop/laptop. The LED should turn teal, this lets you know there is a connection.
2. Now you will need to connect the base station to the user interface. Do this by clicking the “Connect to Erebus Sensor” button. The red square should turn yellow and then a pop up will say, ”this may take up 30 seconds. Click OK to continue.

6

Appendix C: Fundamentals of Electronics

Erebus Labs

# *STEM Sensors*

Basic Resistive Sensor Fundamentals

Version *1.0*

*6/3/2014*

# scott lawson

# Bryan Button

# Chris clary

# max cope

Your Erebus Labs STEM Sensor system includes two sensor modules:

1. Figaro T2600 low oxygen sensor.
2. Photocell ambient light sensor.

Both sensors are resistive and are operating in a voltage divider configuration. Below is the general description of a voltage divider topology.



Basic electrical fundamentals tell us that the entire input voltage will be dissipated across the two resistors, with each resistor dissipating some portion of this voltage. With this in mind, the output voltage can be determined from the resistance values of R1 and R2. Below is a helpful mathematical equation:

Now examine a simple example of this topology. Imagine an input voltage of 5 Volts and a resistance of 8 ohms for R1 and 2 ohms for R2.

We see that the outputted voltage will be the original inputted voltage, 1 volt. With these results we can speculate that the higher the resistance, the more voltage will be dissipated across it. In this case R1 dissipated 4 volts of the entire 5 volts!

This concept can be used to represent the world around us using sensors and this topology.



Let us replace R1 with a resistive light sensor. The resistance of this component will change with respect to the amount of light seen by the sensor. This change in resistance will cause the output voltage to change as well. The world around us is now represented by the amount of voltage seen at the output of this voltage divider! This changing output voltage can be recorded and saved to be analyzed at a later time. Imagine recording the outputted voltages throughout the day, as the sun passes through the sky overhead.



This configuration, although basic, is widely used for scientific measurements and is how your Erebus Labs STEM Sensor system collects its data.



The sensor modules are connected via Ethernet where the input voltage of 5 volts is sent from the base station into the sensor module. That 5 Volts is then dissipated across R1 and R2 of the voltage divider circuit. The amount of ambient light on the light sensor will change its resistance, and will create a unique output voltage that will be recorded and stored in the microcontroller back on the base station.

The other sensor module included with your Erebus STEM Sensor system is the Figarao gas sensor. Although it is a little more complex than the photocell described above, it operates in the exact same way. Its resistance will change with respect to the amount of oxygen in its vicinity, outputting a unique voltage utilizing our voltage divider technique.

These are just two examples of resistive sensors! Can you think of where you would place your two sensors and what you would be trying to measure?

Appendix D: Project Proposal

Erebus Labs

# *STEM Sensors*

# Project Proposal

Version 1*.*3

*1/18/2014*

# scott lawson

# Bryan Button

# Chris clary

# max cope

# VERSION HISTORY

|  |  |  |  |
| --- | --- | --- | --- |
| **Version #** | **Implemented**  **By** | **Revision**  **Date** | **Reason** |
| 1.3 | Scott Lawson | 1/18/2013 | Fixed objective statement  Corrected formatting issues caused by LibreOffice  Removed “Approval” page  Removed “Approval” columns from this table  Reorganized requirements  Added Appendix A  Updated “General System Overview” diagram |
| 1.2 | Scott Lawson | 1/14/2014 | Updated Objective |
| 1.1 | Maxwell Cope | 1/13/2014 | Updated Objective  Removed extra signature blocks |
| 1.0 | Maxwell Cope | 1/12/2014 | Initial Merger of Individual Sections |

**NOTE TO READER**

This is a template obtained from:

<http://www2.cdc.gov/cdcup/library/templates/default.htm>

Template Name: Product Design

**UP Template Version:** 12/31/07

# TABLE OF CONTENTS

1 Introduction 52

1.1 Purpose of The Document 52

1.2 Background 52

1.3 Objective 53

2 Guidelines 53

2.1 Marketing Requirements 53

2.2 Engineering Requirements 55

3 Architecture 56

3.1 General System Overview 56

14.1 Sensor Interfacing 56

15 Timeline 56

15.1 Overview 57

15.2 Milestones 57

16 Budget 58

16.1 Development Costs 58

16.2 BOM costs 59

## Introduction

### Purpose of The Document

This document is the initial project proposal for the STEM (Science, Technology, Engineering, Math) Sensor system. Both signal diagrams and timeline will be extrapolated on depending initial approval.

### Background

Current educational curriculum in the US lacks a focus on Science, Technology, Engineering, and Mathematics (commonly referred to as STEM). According to the United States department of education, only %16 of American high school seniors are proficient in mathematics and interested in a STEM career[[1]](#footnote-1). With a growing demand for STEM related jobs and declining interest in such fields, it is important to get high school graduates into a STEM focused degree program in college. Doing so, however, requires an early start in younger students. In studies referenced by the Business Education Compact, a non-profit focused on bringing STEM education to younger and underserved classrooms, negative interest begins in elementary classrooms where %33 of fourth grade students’ attitudes are already hostile towards science and math; that number goes to %50 by 8th grade[[2]](#footnote-2).

Unfortunately, many student oriented projects currently available are not as practical or affordable as would be desired to get students encouraged. Many projects that students get their hands on involve simple data analysis which is not necessarily the best way to get young energetic kids involved the sciences. Other types of projects are available which are fairly cheap to start out, but require the use of chemicals which are usually required to be kept in the classroom and are harder to collect actual data from. Lastly, there are data acquisition modules which are great for collecting actual data but generally require more advanced setup (such as programming) and usually cost more than $100 per unit. An example of one such device is the LabJack data acquisition device.

Another issue facing the proliferation of STEM subjects in schools is teacher education and resources. Unfortunately many teachers do not have the time or resources to put together detailed lesson plans or learn complicated material with the small amount of time available. The department of education states that one issue with STEM proliferation is limited teacher resources and education.

In order to drive interest in STEM, students need access to a means of not only collecting data but analyzing it as well. Collection and analysis needs to be both educational and interesting, while remaining simple and easy to use. This simplicity needs to extend to the teachers as well, with example plans and experiments available to ensure a smooth teaching and learning experience.

### Objective

Encourage an interest in STEM in K-12 students by delivering a working prototype of an affordable, simple and flexible device to collect environmental data.

## Guidelines

### Marketing Requirements

1. The system must be modular, allowing multiple sensor types to be employed by one base unit.
2. The system must be low cost.
3. The system must be simple enough to operate that K-12 teachers and students can effectively utilize the device in the classroom
4. The user interface must accommodate both novice and advanced users.
5. Base unit with sensors attached must be able to operate outside of a classroom.
6. The system must provide a method for users to specify the data collection interval.
7. Hardware and software designs should be open-source.
8. The base unit should be able to collect data for an extended period without user interaction.
9. The base unit should require minimal changes or adjustments for the use of different sensors.
10. The user interface should provide a method for organizing and comparing data from multiple base units employed simultaneously.
11. The base unit should employ a wireless method of exporting data for analysis.
12. The system should be durable.
13. The base unit may also be modular and require user assembly.
14. The base unit may support simultaneous multiple sensor attachments.

### Engineering Requirements

See Appendix A for terminology definitions.

|  |  |  |
| --- | --- | --- |
| **Marketing Requirements** | **Engineering Requirements** | **Justification** |
| **1, 2, 3, 4** | All sensors must use the same interface to connect to the base unit | Minimizes cost and complexity for users while increasing versatility |
| **4** | The user interface must provide a method for the user to access the raw data collected | Allows advanced users to perform their own data analysis |
| **7** | A publicly-accessible repository must be used for code and documentation hosting | Encourages exploration and experimentation by students |
| **2, 7** | If third-party software is used, it must be open-source | Encourages exploration and experimentation by students, minimizes cost |
| **5, 12** | The base unit with sensors attached must operate when exposed to temperatures between -10°C and +70°C | Temperature range required for outdoor operation |
| **2, 3** | If the system is does not use a rechargeable power source, it must not use proprietary battery types | Using widely available batteries minimizes cost |
| **2** | BOM for base unit should not exceed $20.00 each | Necessary for adoption by K-12 classrooms with limited budgets |
| **2** | BOM for sensors should not exceed $5.00 each | Necessary for adoption by K-12 classrooms with limited budgets |
| **3, 4, 9** | The base unit should identify the sensor(s) attached and configure itself appropriately | Simplifies operation for younger users |
| **1, 4, 6** | The system should be able to collect data points at rates between 1 Hz and 1 per day | Accommodates a wide variety of data collection applications |
| **4, 10** | The system should be able to coordinate data collection between 6 base units simultaneously | Accommodates a wide variety of data collection applications |
| **5, 12** | The base units and sensors should be operational after a 1.5m drop-test | The system needs to survive daily use by K-12 students |
| **5, 12** | The base unit with sensors attached should operate when exposed to temperatures between -20°C and +80°C | Temperature range suggested for outdoor operation |
| **3, 8** | The base unit should be able to collect data points for 90 days without user interaction | Simplifies operation for all users |
| **11** | A wireless data dump interface should be utilized by the base unit | Provides a convenient method for users to retrieve data |
| **11** | If a wireless data dump interface is utilized by the base unit, it should not require the user to be closer to the base unit than 3 meters | Provides a convenient method for users to retrieve data |
| **5, 12** | The base and sensors may be constructed with a water-resistant case | The system needs to survive daily use by K-12 students |
| **2, 4, 14** | The base unit may contain multiple attachment points to enable multiple sensors to be used simultaneously | Enhances versatility for advanced data collection |
| **3, 13** | The base unit may use sockets and connectors to attach the controller, power, and communications devices to the PCB | Further modularity provides hardware interactivity and learning opportunities for younger users |

## Architecture

### General System Overview

### Sensor Interfacing

To reduce interfacing complexity, it is proposed that the sensors be modularized and all interfacing be done through one standard connection. This allows the capability to use a variety of sensors and sensor topologies yet still interface with the base unit for data collection. 

## Timeline

### Overview

The Timeline is currently only populated with milestones. These milestones are designed to dictate the general sequence of the project. Pending approval of the project as well as these milestones, the team will create a more robust timeline.

The main direction taken with these milestones is the functional prototype (3) at the end of the first term. This will be the half-way point of the project, and a functional prototype by then allots us the time to revise our approach to produce a more robust final product.

### Milestones

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Task Name** |  |  | **Deadline** | **Predecessors** |
| **1 Proposal** |  |  | **Sun 1/12/14** | **N/A** |
| **2 Design Review One/ Begin Prototype** |  |  | **Fri 1/31/14** | **1** |
| **3 Functional Prototype** |  |  | **Sun 3/23/14** | **2** |
| **4 Design Review Two and Revision** |  |  | **Sun 4/6/14** | **3** |
| **5 Begin Final Project Assembly** |  |  | **Sun 4/20/14** | **4** |
| **6 Final Unit Completion** |  |  | **Sun 5/18/14** | **5** |
| **7 Present/Wrap Up** |  |  | **Fri 6/6/14** | **5** |

## Budget

### Development Costs

All costs are given in U.S. dollars. Development costs assume low enough quantities that there is no volume discount. Quantities assume two breadboard prototypes and two assembled PCB-based prototypes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Item** | **Cost (Ea)** | **Quantity** | **Ext. Cost** | **Cost Factors** | |
| **Sensor** | 1.00 – 25.00 | 5 – 8 | 5.00 – 50.001 | Type, Quantity of each type | |
| **μController** | 1.00 – 5.00 | 5 – 8 | 5.00 – 40.00 | Onboard features2 | |
| **Batteries** | 0.75 – 12.00 | 5 – 163 | 12.00 – 60.003 | Rechargeable, composition, form factor | |
| **Passives4** | 0.05 – 2.00 | 50 – 100 | 20.00 – 30.001 | Values and tolerances required | |
| **Interfaces5** | 0.50 – 8.00 | 16 - 25 | 8.00 – 50.00 | Wireless vs. Wired | |
| **PCBs6** | 4.00 – 18.00 | 4 - 6 | 24.00 – 48.00 | PCB Area, sensor requirements | |
| **Other SI7** | 0.50 – 2.00 | 16 – 30 | 8.00 – 50.001 | Battery selection, sensor output | |
| **Packaging** | 3.00 – 75.00 | 2 – 3 | 9.00 – 225.00 | Materials: laser-cut acrylic vs. 3D-printer | |
|  | | | | | |
| **Total:** | | | | | 100.00 – 500.00 |

1. Extended cost does not scale linearly with quantity because it is assumed that the maximum quantity would not be entirely comprised of the most expensive components.
2. Onboard features include ADC, power conditioning, amount of memory, etc
3. Quantity and extended cost assume either a small amount of expensive rechargeable proprietary batteries, or a larger amount of cheap (AA or 9V) batteries.
4. Resistors, capacitors, inductors, LEDs
5. Antennas, transceivers, receivers, cable jacks
6. Assumes a PCB for the base unit as well as separate PCBs for the interchangeable sensors.

Example: Minimum extended cost is based on a 2in x 2in base unit PCB with two 1in x 1in sensor PCBs at $2.00 per square inch, x2 prototypes.

1. Other semiconductors: op-amps, voltage regulators, discrete transistors

### BOM costs

Low Price-point Example: one base unit with two external interchangeable sensors. Single unit cost assuming volume is low enough that there are no quantity discounts.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Cost (Ea)** | **Quantity** | **Ext. Cost** | **Notes** |
| **Sensor** | 0.19, 5.99 | 2 | 6.18 | Temperature Sensor, CO sensor |
| **μController** | 1.36 | 1 | 1.36 | ATTiny84 – onboard ADC, 8KB flash |
| **Batteries** | 2.14 | 1 | 2.14 | Standard 9v Battery |
| **Passives4** | 0.05 – 0.50 | 10 | 3.00 | Misc resistors, capacitors, LEDs |
| **Interfaces5** | 0.50 | 4 | 2.00 | Generic jack for sensors, USB for base |
| **PCBs6** | 4.00 – 8.00 | 3 | 16.00 | 2”x2” base PCB, 1”x2” PCB per sensor |
| **Other SI7** | 0.50 | 0 | 0.00 | Assuming μC ADC can handle sensor output |
| **Packaging** | 3.00 | 1 | 3.00 | Laser-cut acrylic from EPL, hand assembled |
| **Total:** | | | | $33.68 |

## Appendix A: Terminology

### Acronyms

|  |  |
| --- | --- |
| **Acronym** | **Meaning** |
| ADC | Analog-to-Digital Converter |
| BOM | Bill of Materials |
| CO | Carbon Monoxide |
| EPL | The Portland State University Engineering and Prototyping Lab |
| K-12 | Kindergarten through 12th grade school |
| LED | Light Emanating Diode |
| PCB | Printed Circuit Board |
| SI | Silicon |
| STEM | Science, Technology, Engineering and Math |
| USB | Universal Serial Bus |

### Requirements

**“Must”**

The system design is not considered complete if these requirements are not satisfied.

**“Should”**

These requirements describe strongly desired features or options, but the product may be considered complete without them.

**“May”**

It would be nice to implement these features if time and budget allow.

### Architecture

**Base Unit**

The central device that manages power, communication, and data storage, and has one or more sensors attached to it.

**Sensor**

The individual data collection devices such as VOC detectors and thermometers that are attached to the base unit.

**User Interface**

The program that will be run on a laptop or desktop computer that allows the user to view and interact with the data collected.

**System**

The operational product comprised of base units with attached sensors and a user interface.

1. www.ed.gov/stem [↑](#footnote-ref-1)
2. http://becpdx.org/nem/stemconnect.aspx [↑](#footnote-ref-2)