

EREBUS LABS SENSOR DESIGN SPECIFICATION

SCOTT LAWSON

BRYAN BUTTON

CHRIS CLARY

MAX COPE

Version 1.6
6/4/2014

VERSION HISTORY

| Version # | Implemented By | Revision Date | Reason |
|-----------|--------------------------|---------------|--|
| 1.6 | Scott Lawson Max Cope | 6/4/2014 | Updated Software/Firmware details Added Layouts and Schematics Added Sensor and power supply details |
| 1.5 | Scott Lawson | 3/31/2014 | Added interface mode flowchart Added error messages |
| 1.4 | Scott Lawson | 3/30/2014 | Updated 3.1 IRQ information Updated 3.2 Interface Mode details Adjusted page numbering to start on Table of Contents |
| 1.3 | Scott Lawson | 3/3/2014 | Added 3.2.3 Virtual COM Port Added 3.2.4 Interface Mode Updated 3.1.5 with VBUS details |
| 1.2 | Scott Lawson | 2/1/2014 | Fixed header typos Added Engineering Requirements to Introduction |
| 1.1 | Scott Lawson | 1/30/2014 | Changed data sample format Changed Appendix A name to "Glossary" from "Terminology" |
| 1.0 | Scott Lawson | 1/28/2014 | Initial Release Converted from Software Plan V1.0 |

TABLE OF CONTENTS

| | |
|---------------------------------------|-----------|
| TABLE OF CONTENTS | 1 |
| INTRODUCTION | 3 |
| Purpose of The Document | 3 |
| Overview | 3 |
| Objective Statement | 3 |
| Theory of Operation..... | 3 |
| ENGINEERING REQUIREMENTS | 4 |
| HARDWARE PLAN..... | 6 |
| Overview | 6 |
| Block Diagrams | 6 |
| Base Unit Level 0 Block Diagram..... | 6 |
| Base Unit Level 1 Block Diagram..... | 7 |
| Base Unit Level 2 Block Diagram..... | 8 |
| PSoC3 Block Diagram | 9 |
| Implementation | 10 |
| Microcontroller | 10 |
| Sensor Design..... | 11 |
| Power Supply | 11 |
| Schematics & Layouts | 12 |
| FIRMWARE PLAN..... | 17 |
| Firmware Overview | 17 |
| Initialization Tasks..... | 18 |
| Interrupt Service Routines..... | 18 |
| Event Handlers..... | 19 |
| USB Waiting | 19 |
| Take Sample Waiting | 21 |

STEM Sensors Design Specification

| | |
|--|-----------|
| Start Sampling Waiting | 21 |
| Stop Sample Waiting..... | 21 |
| Check Battery Waiting | 22 |
| Low Battery Blink Waiting..... | 22 |
| Sample Blocks | 22 |
| Data Samples | 23 |
| Sample Exporting | 24 |
| USB Packets During Sample Export | 24 |
| Real-Time Clock Updates | 25 |
| USER APPLICATION PLAN..... | 26 |
| User Interface | 26 |
| Device Interface Mode | 26 |
| Available Commands | 26 |
| Incoming Messages from Sensor | 27 |
| Virtual COM Port..... | 27 |
| Command Resolution | 27 |
| APPENDIX A: ACRONYMS..... | 28 |
| APPENDIX B: SYSTEM ARCHITECTURE | 29 |

Introduction

Purpose of The Document

This document describes the implementation of both the hardware and software components of the Erebus Labs STEM Sensor. It is intended to be an in-depth technical description of the sensor's software and firmware for those interested in developing custom sensors or modifying the operation of the system.

Overview

Objective Statement

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.

Theory of Operation

The Erebus Labs STEM Sensor system is an open-source electronic device for collecting environmental data over a period of time and presenting it for analysis. The system is comprised of the following components:

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 is run on a laptop or desktop computer that allows the user to analyze the data collected.

The base unit is designed to have one sensor attached to it and passively collect data without being attached to a computer system. The data collection site is chosen by the user. The user interface is a simple GUI for displaying collected data and exporting the data to a text file for analysis with a third-party program.

Engineering Requirements

See proposal for marketing requirements.

| 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 |

STEM Sensors Design Specification

| | | |
|----------|--|---|
| 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 |

STEM Sensors Design Specification

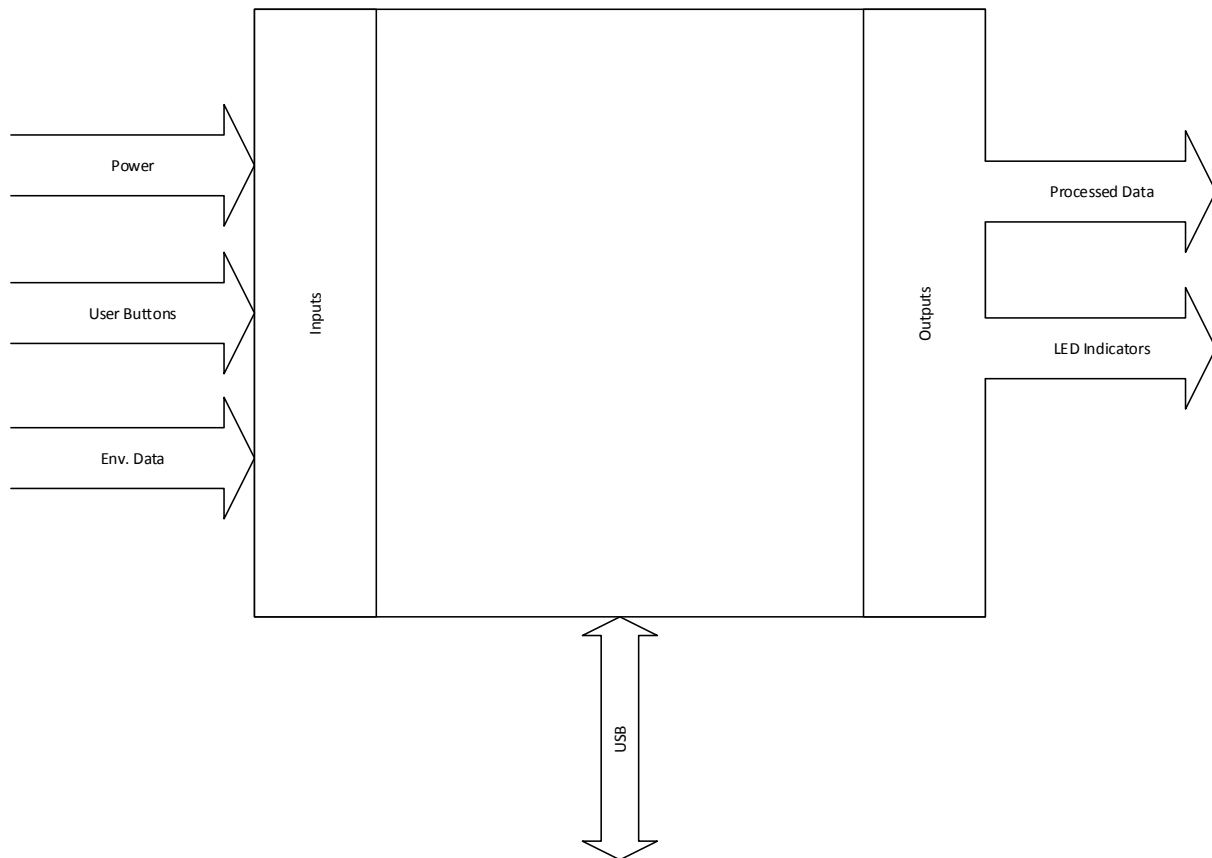
Hardware Plan

Overview

The base unit is comprised of a Cypress PSoC3 microcontroller, two voltage regulators, a sensor interface and a USB port.

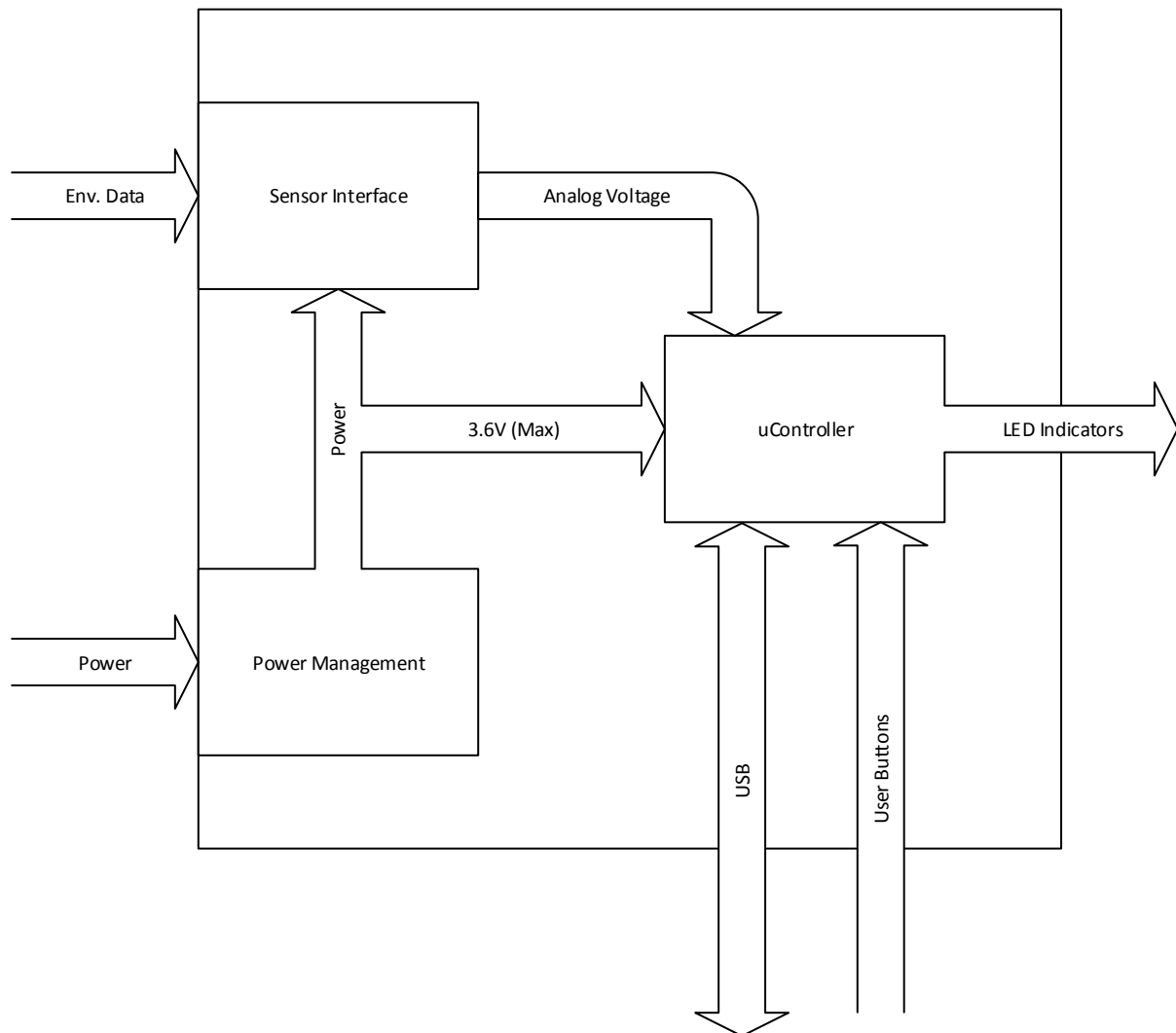
Block Diagrams

Base Unit Level 0 Block Diagram



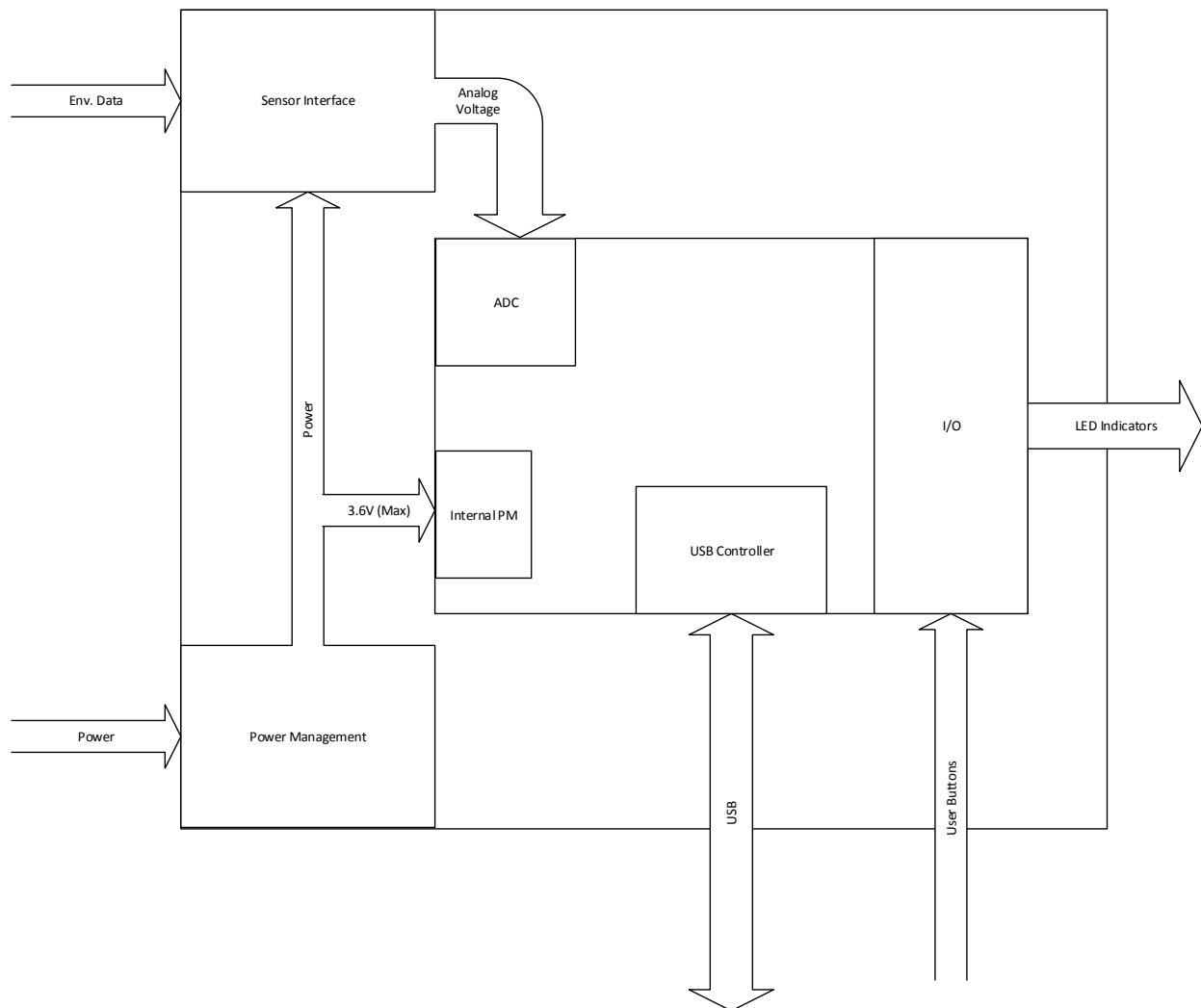
STEM Sensors Design Specification

Base Unit Level 1 Block Diagram

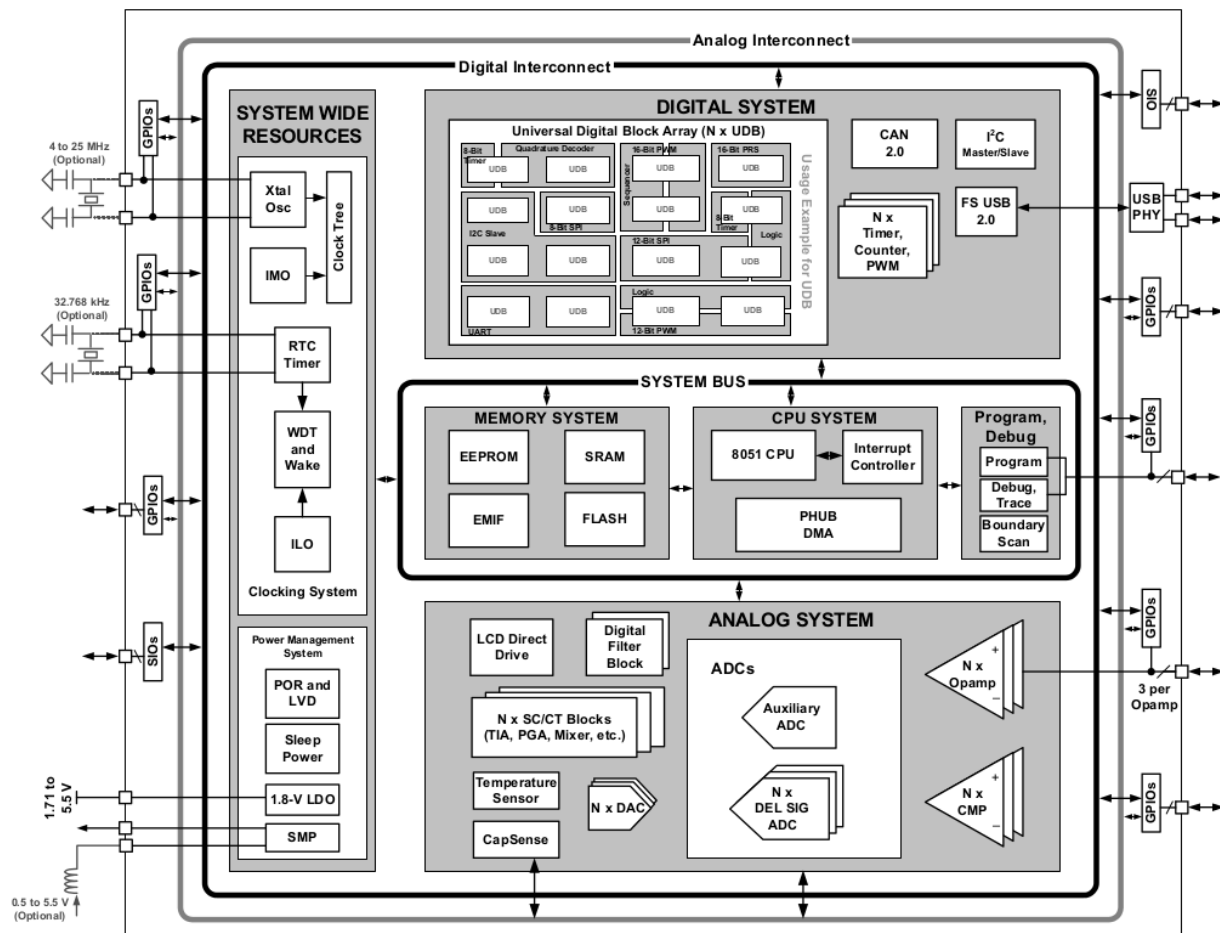


STEM Sensors Design Specification

Base Unit Level 2 Block Diagram



PSoC3 Block Diagram



STEM Sensors Design Specification

Implementation

Microcontroller

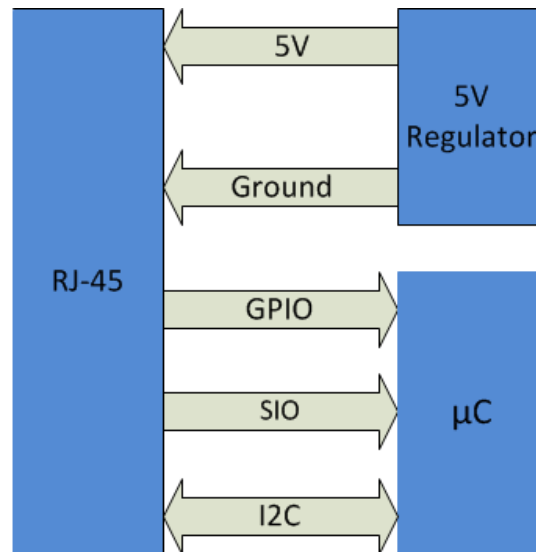
The PSoC3 microcontroller from Cypress Semiconductor was selected because of its balance between flexibility and cost. It contains several embedded programmable logic blocks that can be used to implement a Full-Speed USB controller and real-time clock. Additionally, the programmable blocks also provide an interface for utilizing the chip's on-board Flash memory in place of an external EEPROM chip. Therefore, the PSoC3 provides a one-chip solution.

| | |
|-------------------|-------------------------------------|
| Manufacturer | Cypress Semiconductor |
| Family | PSoC3 |
| Model Number | CY8C3246PVI-147 |
| Architecture | 8-bit 8051 |
| Clock Speed | 50MHz |
| Operating Voltage | 1.71V – 5.5V |
| Current Draw | 0.8mA@3MHz, 1.2mA@6MHz, 6.6mA@48MHz |
| ADC | 12-bit Delta-Sigma |
| Program Memory | 64KB Flash |
| EEPROM | 1KB |
| UDBs | 24 |
| Package | 48-pin SSOP |

STEM Sensors Design Specification

Sensor Design

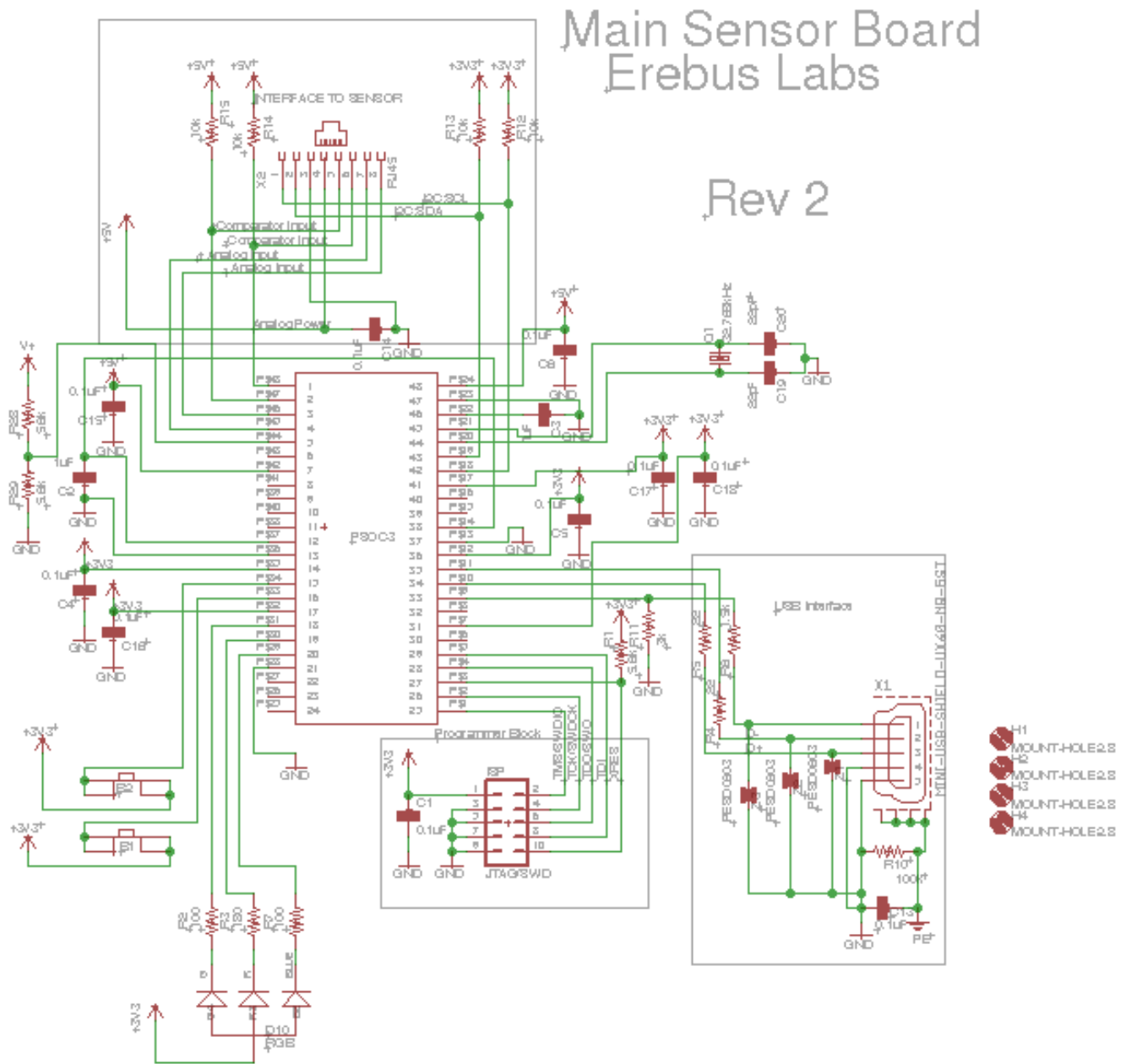
The base unit comes with two sensors. The first is a light sensor to provide high controllability during testing. The second is a Figaro T2600 low oxygen sensor to provide results in gas detection. The two stock sensors were chosen for high reliability and proof of concept. The sensor interface has up to six available outputs to the base these include: two outputs that go the microcontroller's comparator, two outputs that go to the microcontroller's ADC, and two outputs that go to the microcontroller's I2C input. This flexibility provides the user the ability to use a variety of sensors requiring only software changes. The sensor modules are connected to the base unit via a standard CAT5 cable plugged into RJ-45 jacks on both the base unit and sensor modules.



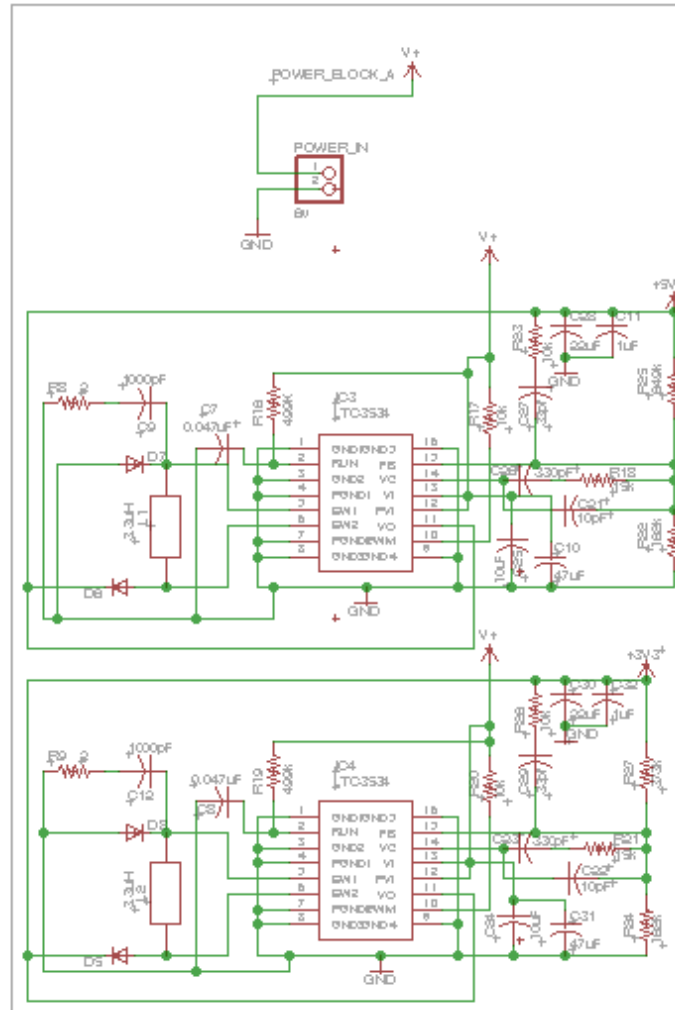
Power Supply

The Base is designed to run on four AA batteries. To do this the most efficiently, Linear Technologies' LTC3435 buck-boost regulators are used. Using these regulator allow the maximum use of the batteries by operating in all three regions: over voltage, at regulated voltage, and under voltage. There are two supplies, 3.3V (for digital needs) and 5V (for analog needs), that consume power with 90% efficiency.

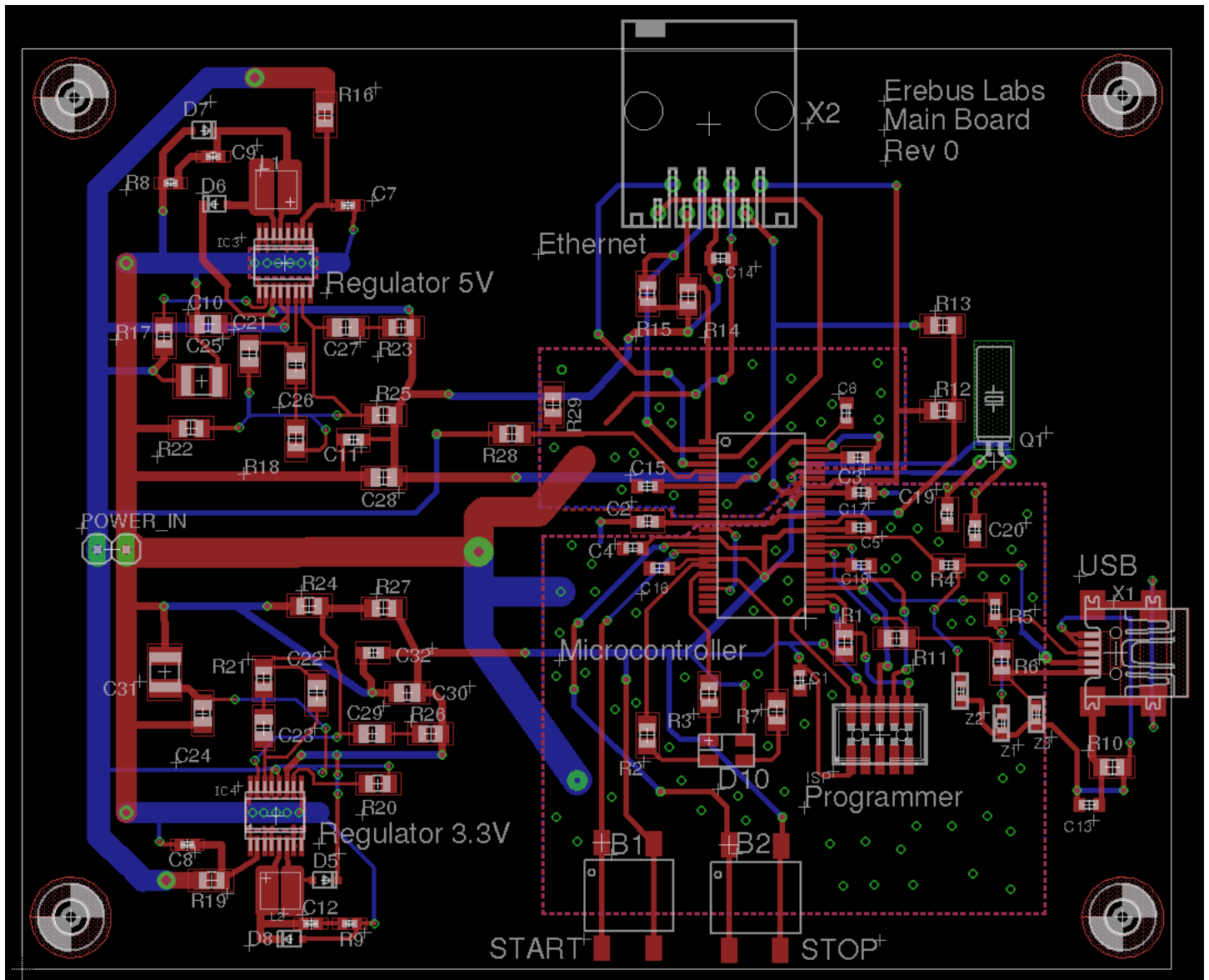
Base Unit Microcontroller Schematic



Base Unit Power System Schematic

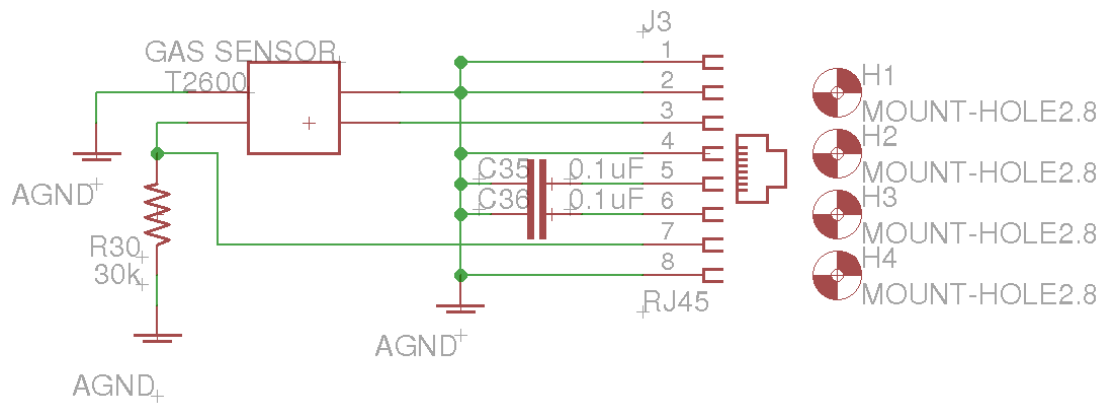


Base Unit Layout

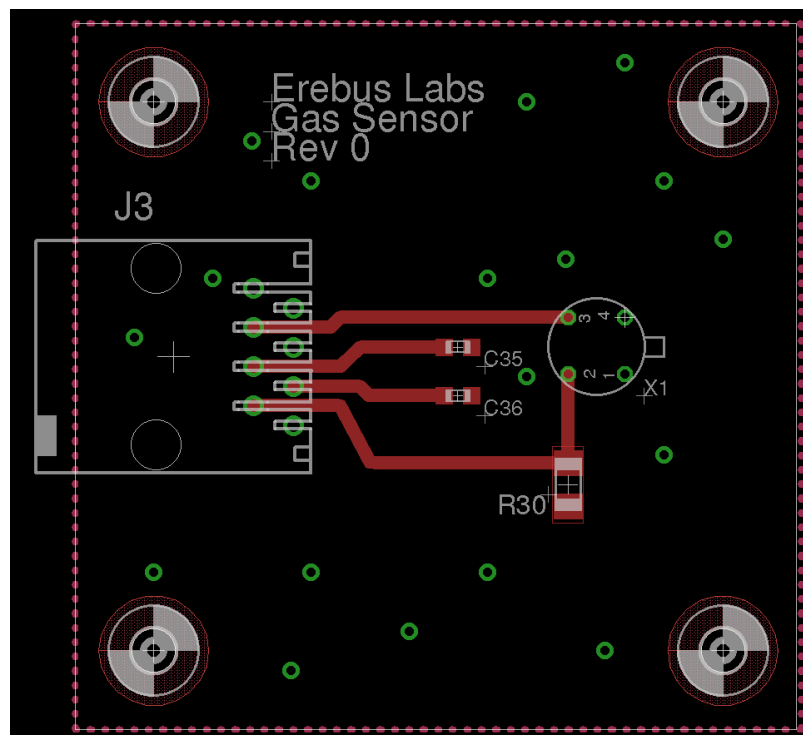




Gas Sensor Schematic



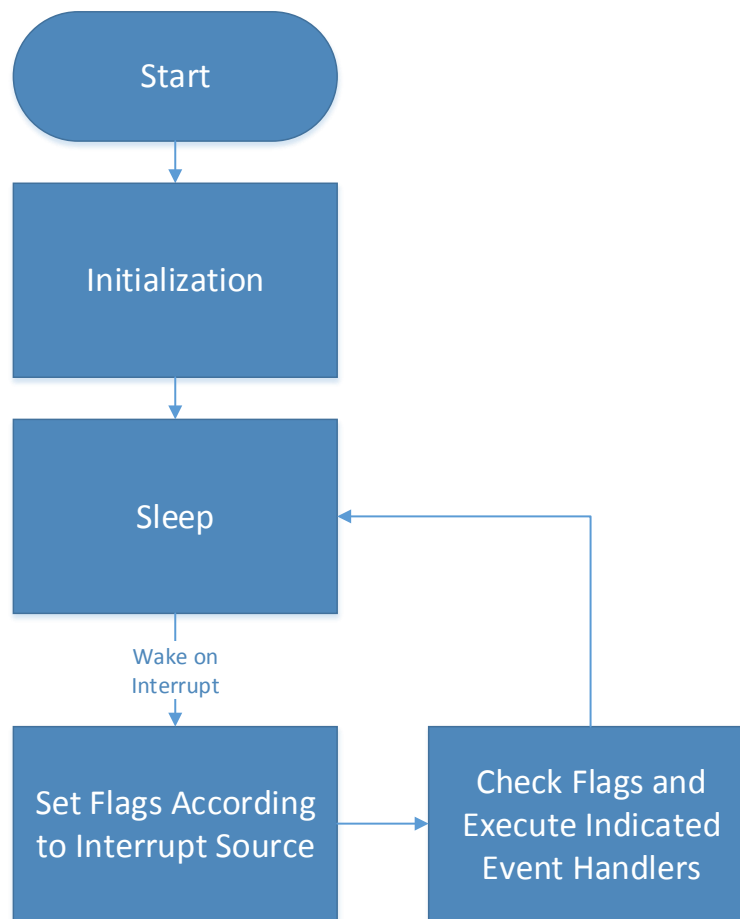
Gas sensor Layout



Firmware Plan

Firmware Overview

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:



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.

STEM Sensors Design Specification

Initialization Tasks

1. The following initialization tasks are performed upon reset:
2. Retrieve data sample pointers from Flash
3. Start Components:
EEPROM, Real-Time Clock, PWM Controller, ADC, Analog Multiplexer
4. Retrieve and apply sampling parameters from EEPROM
5. Enable VBus_IRQ, ModifyCollection_IRQ, and global interrupts
6. Enable sample initiation

Interrupt Service Routines

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_EverySecondHandler | 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_EveryMinuteHandler | 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_EveryHourHandler | RTC Every Hour Periodic Interrupt | If sample unit is hours and sampling enabled, increment counter; if counter == sample interval, take sample. |
| RTC_EveryDayHandler | RTC Every Day Periodic Interrupt | If sample unit is days and sampling enabled, increment counter; if counter == sample interval, take sample. |

Note: RTC interrupt routines can be found in Generated_Source/PSoC3/RTC/RTC_Int.c. They all share one interrupt vector titled RTC_isr in Erebus_Sensor.cydwr -> Interrupts.

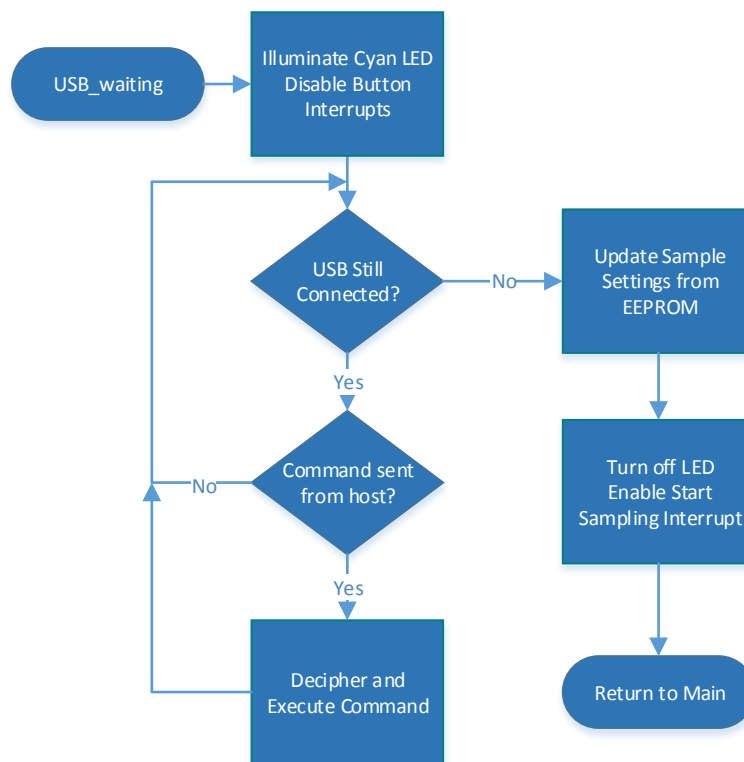
STEM Sensors Design Specification

Event Handlers

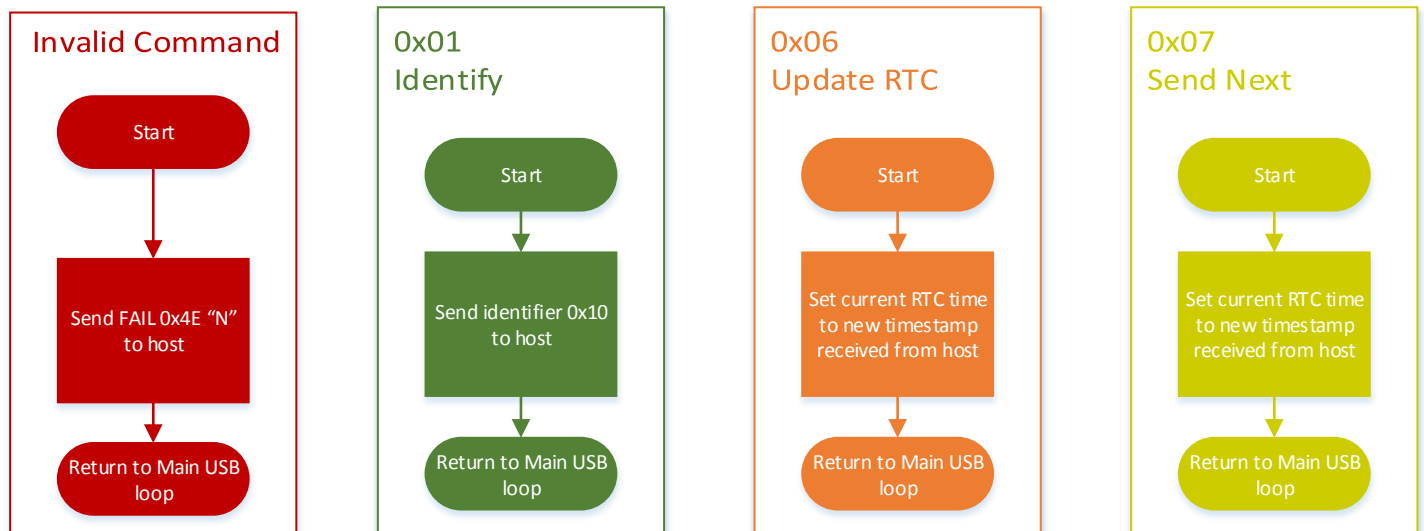
The following events are to be handled by function calls from main by the procedures described in these flowcharts.

USB Waiting

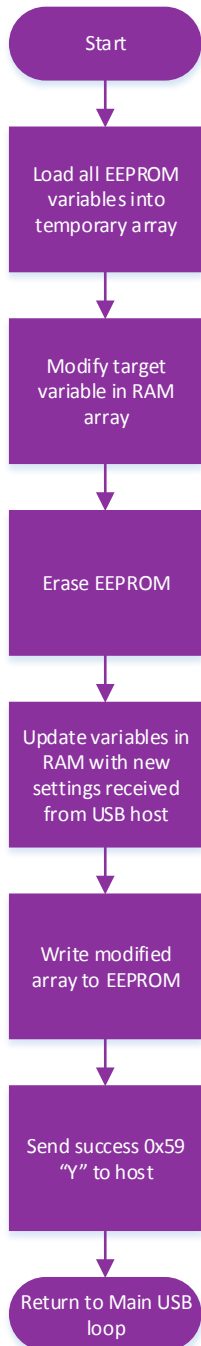
The USB_waiting flag indicates that the device's USB port has been attached to a host. The various commands that may be sent by the host are handled according to the USB Command Handling flowcharts below.



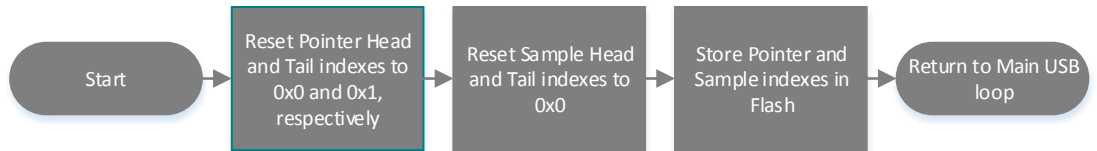
USB Command Handling



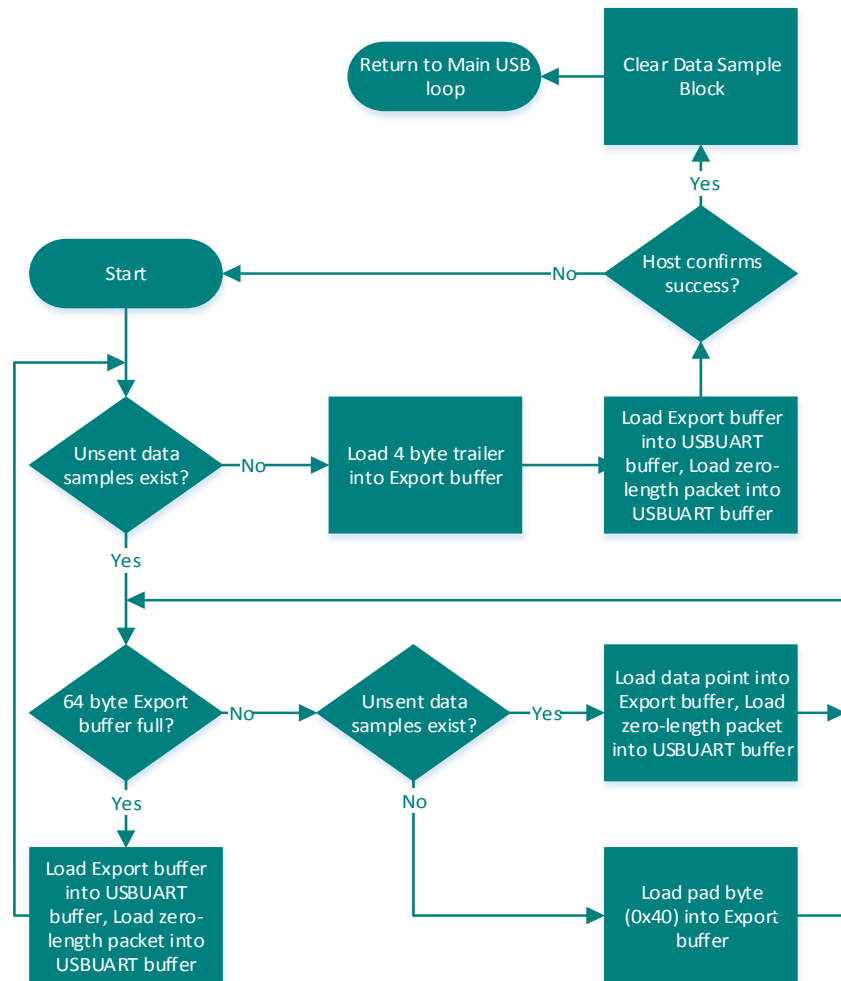
0x04 Change Settings



0x05 Reset Pointers



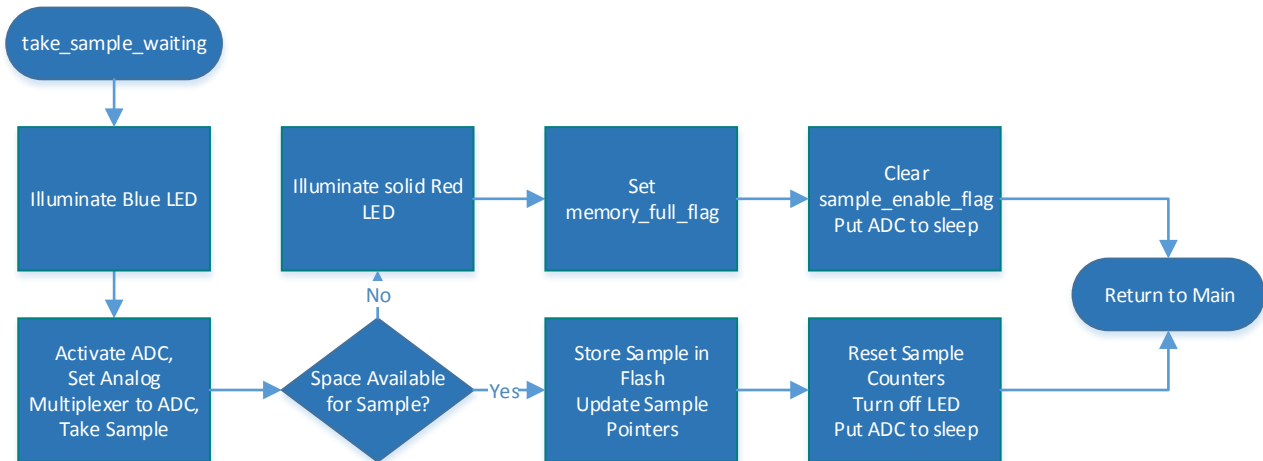
0x02 Dump Data



STEM Sensors Design Specification

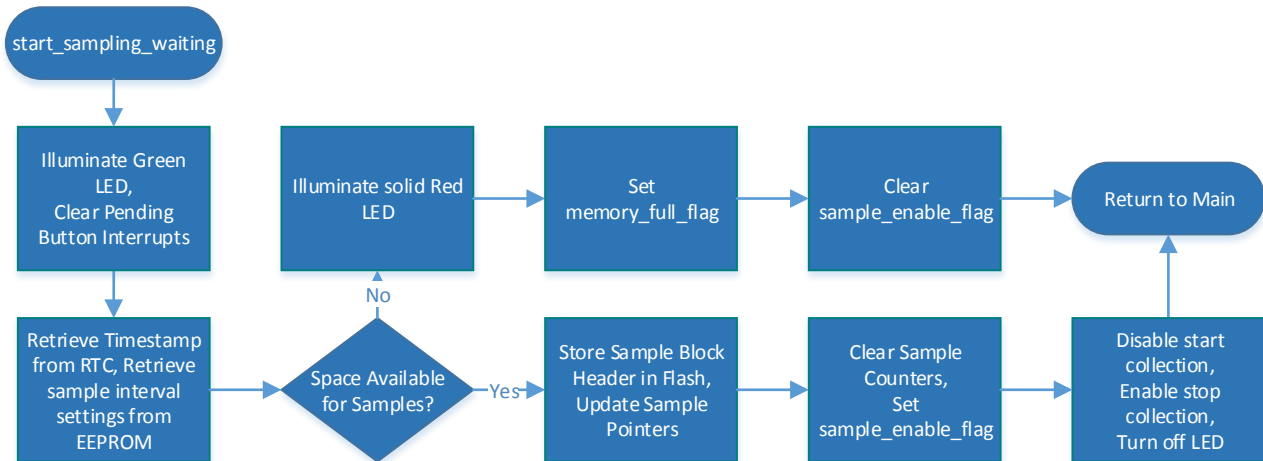
Take Sample Waiting

The take_sample_waiting flag indicates that the sample interval has been reached according to the Real-Time clock and a sample must be taken.



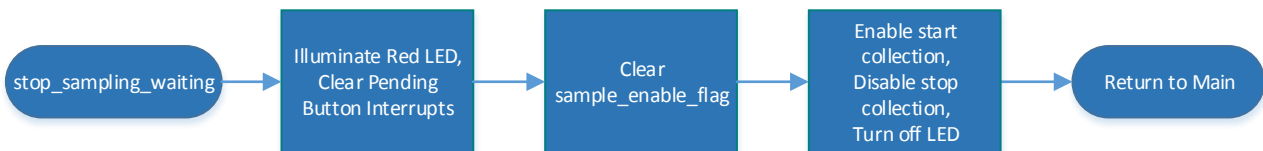
Start Sampling Waiting

The start_sampling_waiting flag indicates that the user has pressed the “Start Sampling” button.



Stop Sample Waiting

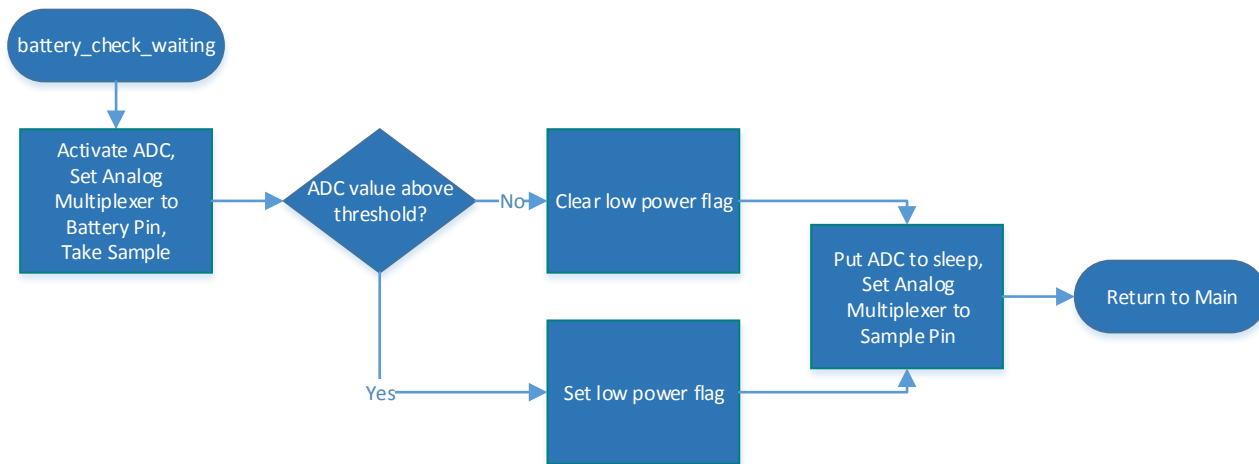
The stop_samping_waiting flag indicates that the user has pressed the “Stop Sampling” button.



STEM Sensors Design Specification

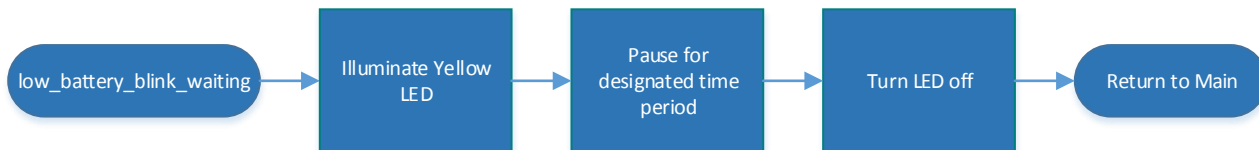
Check Battery Waiting

The `check_battery_waiting` flag indicates that it is time to check the battery level.



Low Battery Blink Waiting

The `low_battery_blink_waiting` flag indicates that as of the last battery sample, the voltage level of the battery is below the threshold, so the LED must be used to alert the user.



Sample Blocks

Data samples are stored in Flash memory along with program code by utilizing the EEPROM emulator 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.

The header has the following 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 |

STEM Sensors Design Specification

| | | |
|-----------|---|---|
| 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 Samples

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.

STEM Sensors Design Specification

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. It has the following structure:

| Bits | Number of Bytes | Description |
|-----------|-----------------|--|
| TR[31:16] | 2 | Number of data points 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.

IMPORANT NOTE: After each 64-byte packet, a zero length packet **MUST** be sent to the host to notify it that the sensor is done with that transaction and is awaiting the NEXT command. Failure to do this may cause 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.

STEM Sensors Design Specification

Real-Time Clock Updates

When the current time of the RTC is to be updated, the current time stamp is expected to be received in an array of bytes (CT) of the following 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. |

User Application Plan

User Interface

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.

The user interface is written in Python 3.2x utilizing the PySerial library for serial communications and the tkinter library for GUI rendering.

Device Interface Mode

While plugged into the computer, the sensor remains in Interface Mode. During this time, the PSoC3 continually monitors its input buffer for commands from the host computer and respond to them accordingly. The microcontroller exits interface mode when power from the USB VBUS pin is no longer detected.

All commands to the sensor from the host and replies from the sensor are one byte long, with the exception of sampled data dumps and when returning current sampling settings. All command and data packets use Big-Endian byte-ordering.

Available Commands

| Command | Bit Pattern | Sensor Action |
|----------------|-------------|---|
| IDENTIFY | 0x01 | Replies with identifier |
| DUMP_DATA | 0x02 | Exports sampled data points to host |
| GET_SETTINGS | 0x03 | Retrieves current sampling settings from EEPROM and sends them to the user application |
| CHANGE_SETTING | 0x04 | Stores new sampling settings provided in EEPROM |
| RESET_PTRS | 0x05 | Resets data sample block pointers to default values |
| UPDATE_RTC | 0x06 | Updates the time and date of the Real-Time Clock with the values provided |
| NEXT | 0x07 | During sample block dump, sensor proceeds with next sample transaction – illicit a FAILURE response if sent when not transferring sample data |

STEM Sensors Design Specification

Incoming Messages from Sensor

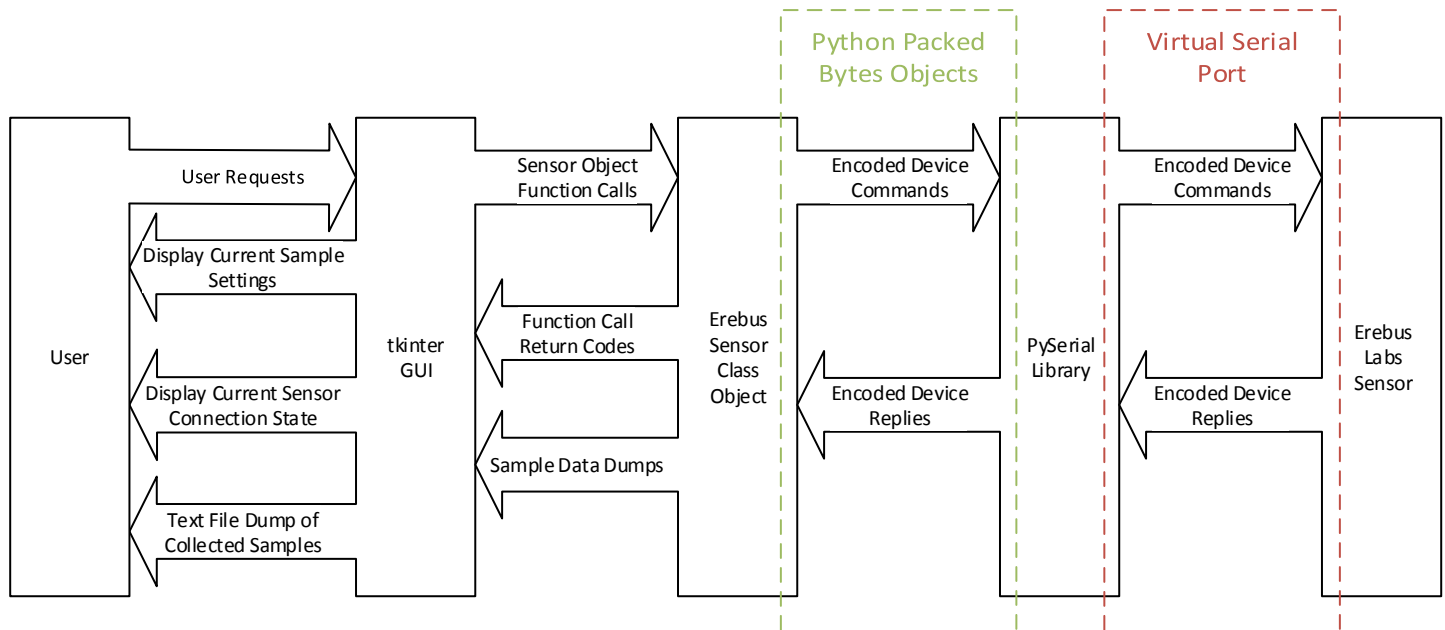
| Messages | Bit Pattern | Meaning |
|------------|-------------|---|
| IDENTIFIER | 0x10 | Differentiates Erebus Labs Sensor from other serial devices |
| SUCCESS | 0x20 | Last action was successful, ready for next command |
| FAILURE | 0x30 | Last action failed, reattempt command |

Virtual COM Port

The PSoC3 microcontroller USBFS_UART component is used for communication with the host computer. The Python library PySerial is used to communicate over the serial port with the base unit. Communicating using a virtual serial port provides a cross-platform method of enumerating on and communicating with a host without the need for additional drivers.

Command Resolution

User communication with the device involves four layers as shown in the following diagram:



As the user selects menu options or enters new settings, the GUI executes callbacks into the Erebus Sensor class object which maintains a handle to the attached sensor. It is the responsibility of the Erebus Sensor object to resolve the user's commands into the appropriate bit fields and transfer them to the device through calls to the PySerial routines. The Erebus Sensor object must also interpret the device replies to notify the user whether the last command succeeded or failed.

Appendix A: Acronyms

| Acronym | Meaning |
|------------------|--|
| ADC | Analog-to-Digital Converter |
| BOM | Bill of Materials |
| CO | Carbon Monoxide |
| CSV | Comma-separated-value formatted file |
| EEPROM | Electrically Erasable Programmable Read-Only Memory |
| EPL | The Portland State University Engineering and Prototyping Lab |
| GUI | Graphical User Interface |
| LED | Light Emitting Diode |
| I ² C | The Inter-Integrated Circuit communication protocol |
| ISR | Interrupt Service Routine |
| K-12 | Kindergarten through 12 th grade school |
| LED | Light Emanating Diode |
| PCB | Printed Circuit Board |
| PICU | Port Interrupt Control Unit |
| PSoC | Programmable System On Chip |
| SI | Silicon |
| SPI | Serial Peripheral Interface Bus |
| STEM | Science, Technology, Engineering and Math |
| TRM | Technical Reference Manual |
| USB | Universal Serial Bus |
| USBUART | Universal Serial Bus Universal Asynchronous Receiver/Transmitter (refers to the PSoC3 USBUART component) |

Appendix B: System 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 is 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.