

EREBUS LABS SENSOR TECHNICAL REFERENCE MANUAL

SCOTT LAWSON

BRYAN BUTTON

CHRIS CLARY

MAX COPE

Version 1.0

6/4/2014

VERSION HISTORY

Version #	Implemented By	Revision Date	Reason
1.0	Scott Lawson	1/28/2014	Initial Release Adapted from Design Specification V1.6

TABLE OF CONTENTS

TABLE OF CONTENTS	1
INTRODUCTION	3
Purpose of The Document	3
Overview	3
Theory of Operation.....	3
HARDWARE DESIGN	4
Overview	4
Block Diagrams	4
Base Unit Level 0 Block Diagram.....	4
Base Unit Level 1 Block Diagram.....	5
Base Unit Level 2 Block Diagram.....	6
PSoC3 Block Diagram	7
Implementation	8
Microcontroller	8
Sensor Design.....	8
Power Supply	8
Schematics & Layouts	9
FIRMWARE DESIGN	14
Firmware Overview	14
Initialization Tasks.....	15
Interrupt Service Routines.....	15
Event Handlers.....	16
USB Waiting	16
Take Sample Waiting	18
Start Sampling Waiting	18
Stop Sample Waiting.....	18
Check Battery Waiting	19

Erebus Labs STEM Sensor Technical Reference Manual

Low Battery Blink Waiting.....	19
Sample Blocks	19
Data Samples	20
Sample Exporting	21
USB Packets During Sample Export	21
Real-Time Clock Updates	22
USER APPLICATION DESIGN	23
User Interface	23
Device Interface Mode	23
Available Commands	23
Incoming Messages from Sensor	24
Virtual COM Port.....	24
Command Resolution	24
APPENDIX A: ACRONYMS.....	25
APPENDIX B: SYSTEM ARCHITECTURE	26

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

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 internal workings of both the hardware and software are made available for the benefit of the end user and engineering community. The user is encouraged to modify and improve the system, and share the results with the development community.

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.

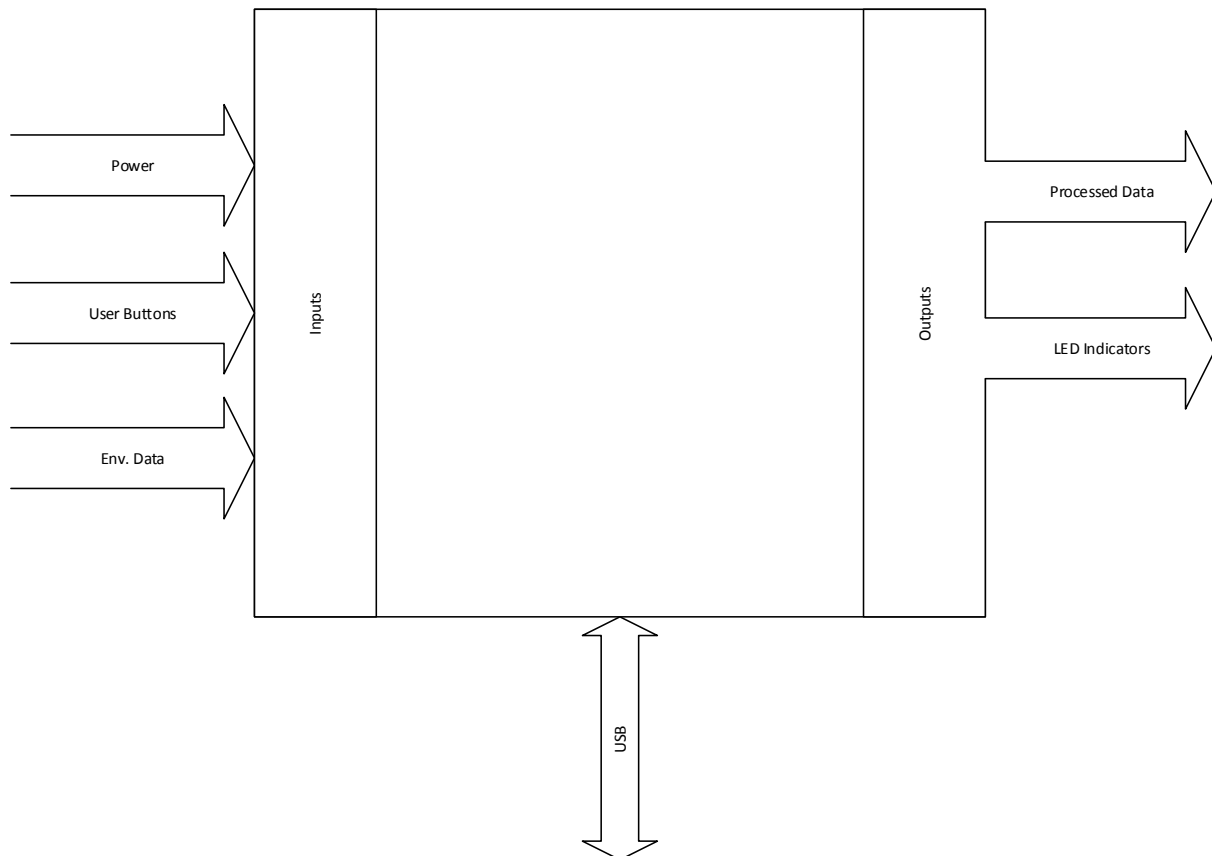
Hardware Design

Overview

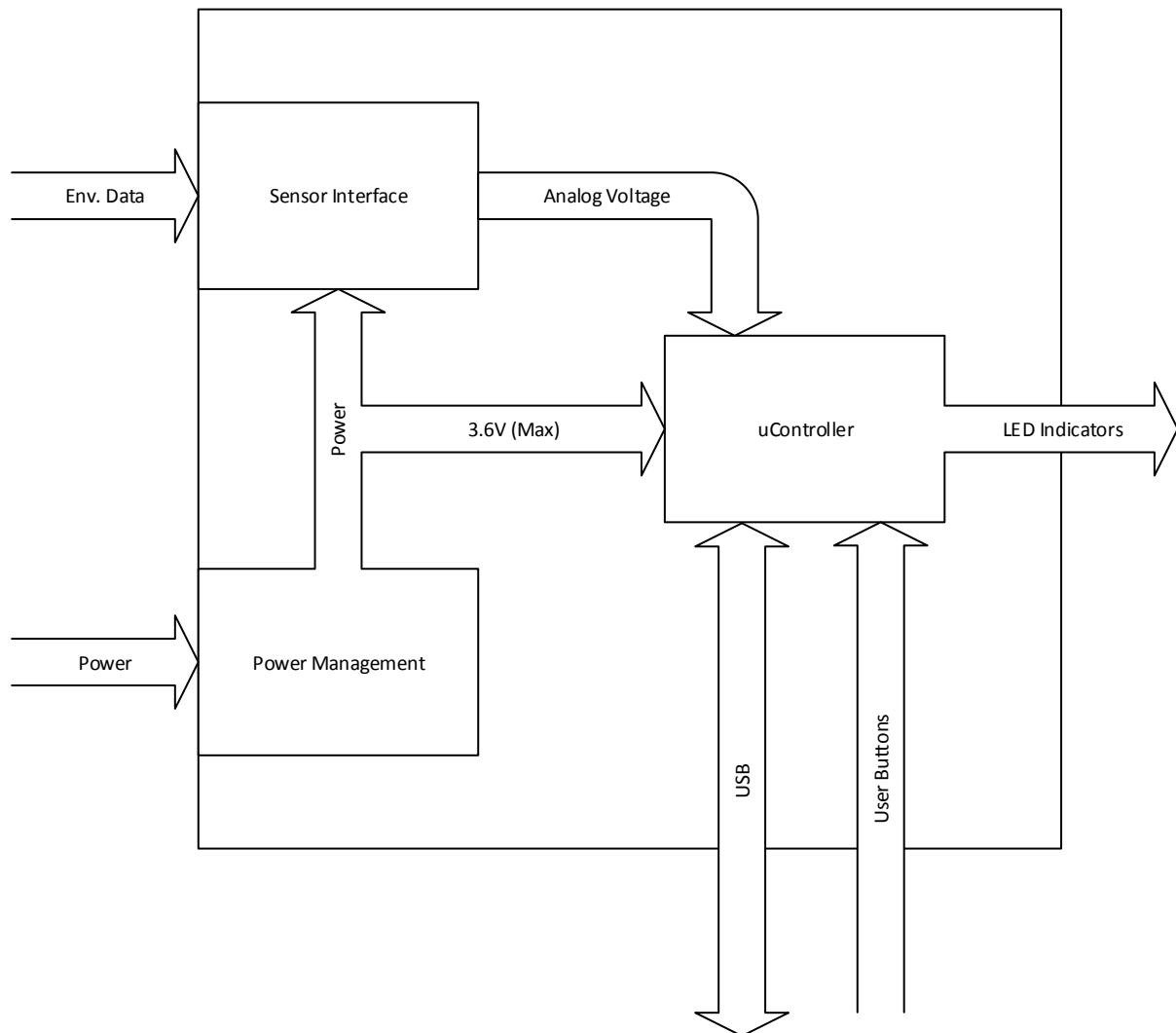
The base unit is comprised of a Cypress PSoC3 microcontroller, two voltage regulators, a sensor interface and a USB port. The sensors are on individual boards connected by a CAT-5 Ethernet cable plugged into RJ-45 jacks on the sensor boards and base units.

Block Diagrams

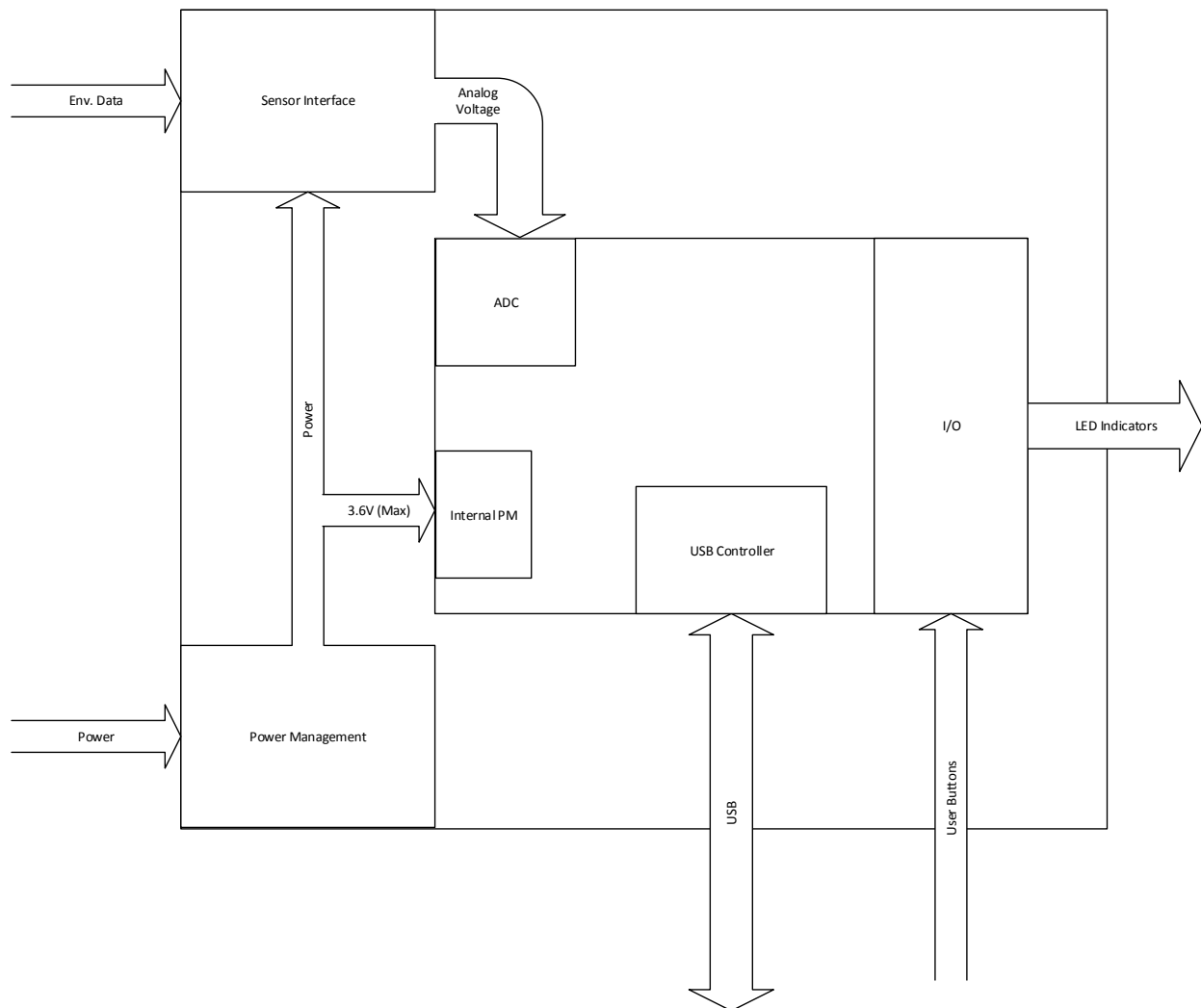
Base Unit Level 0 Block Diagram



Base Unit Level 1 Block Diagram



Base Unit Level 2 Block Diagram





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

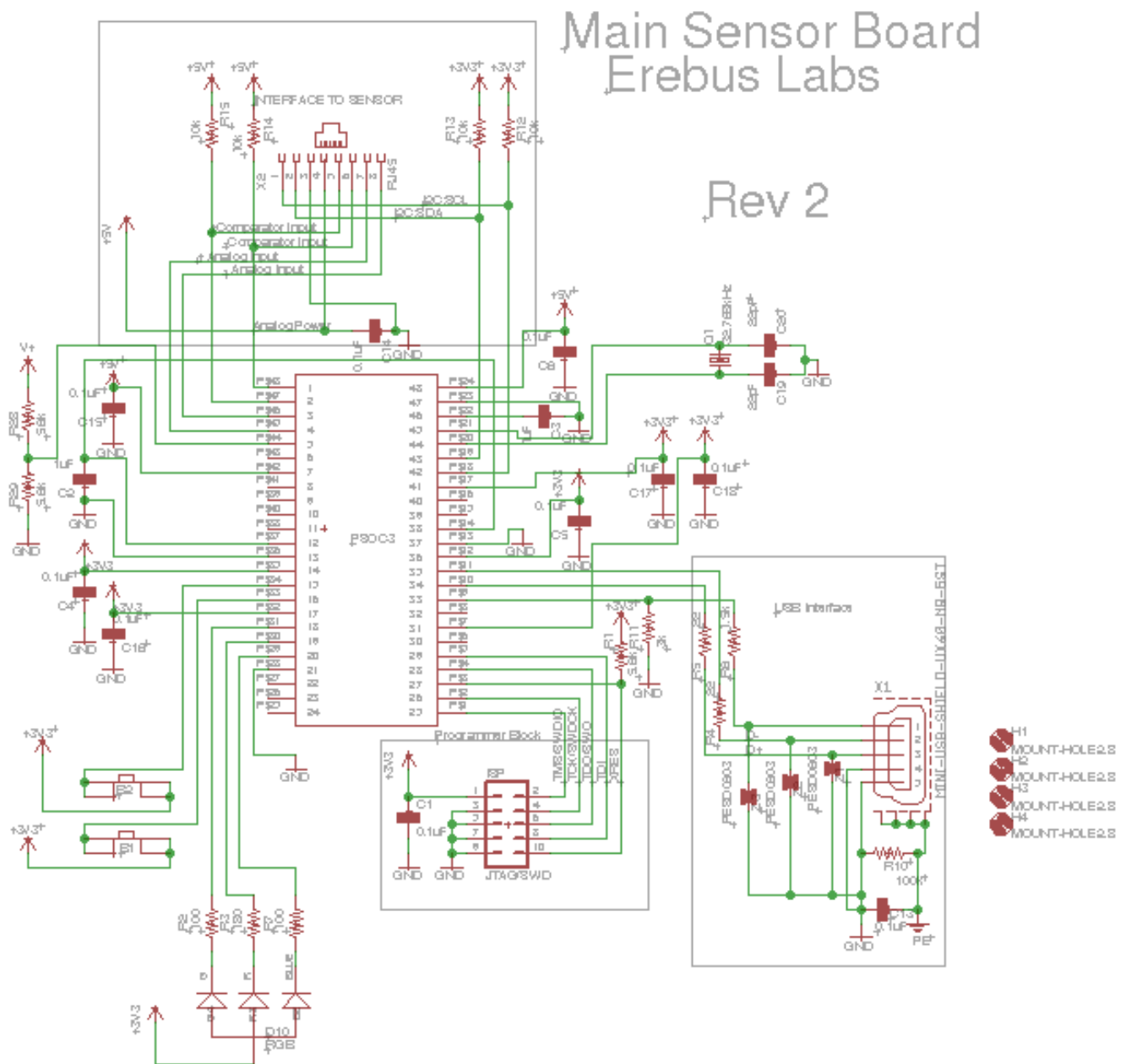
Sensor Design

The based unit will come 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 unit including: 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.

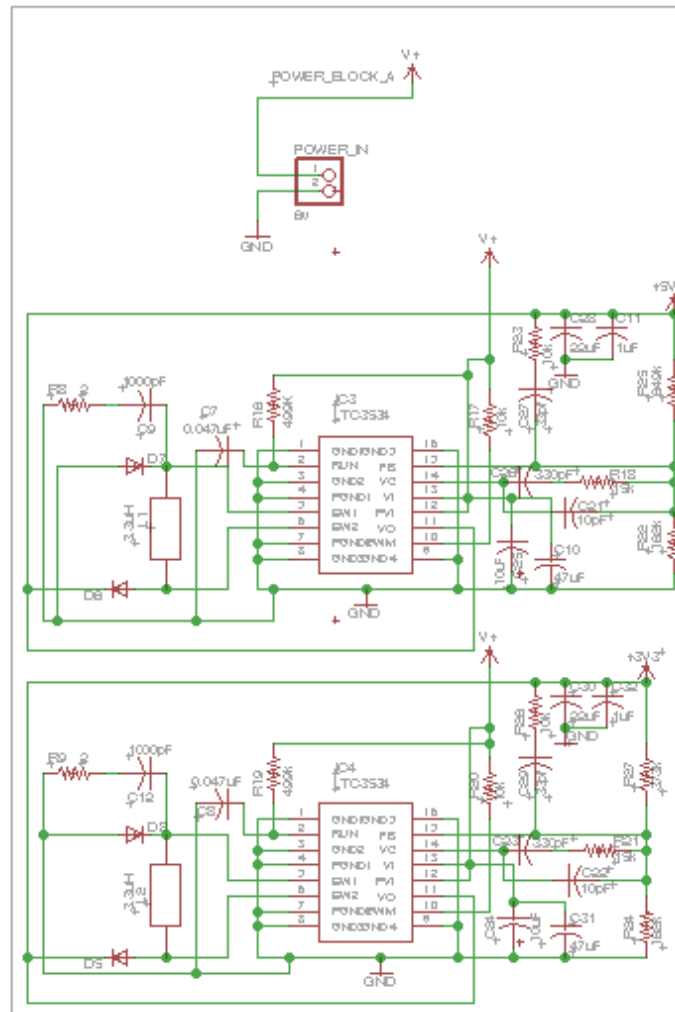
Power Supply

The Base will made 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.

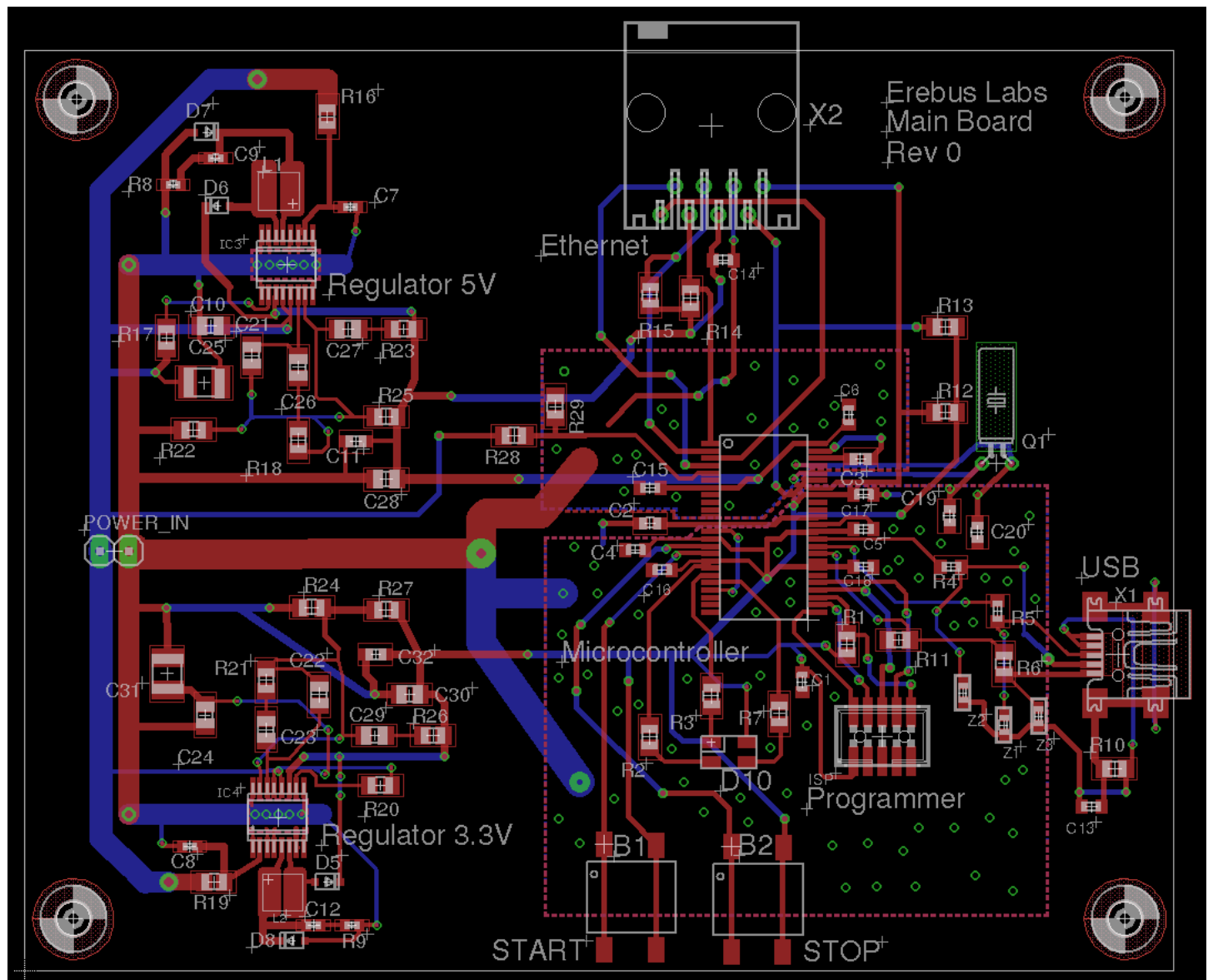
Base Unit Microcontroller Schematic



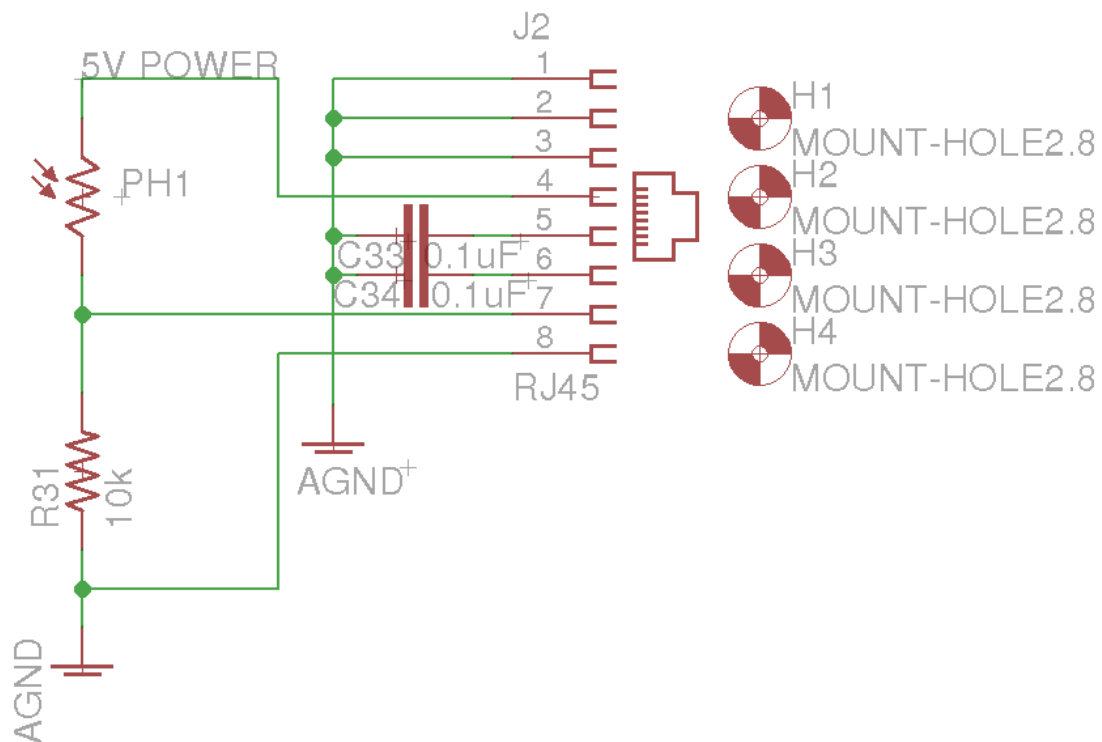
Base Unit Power System Schematic



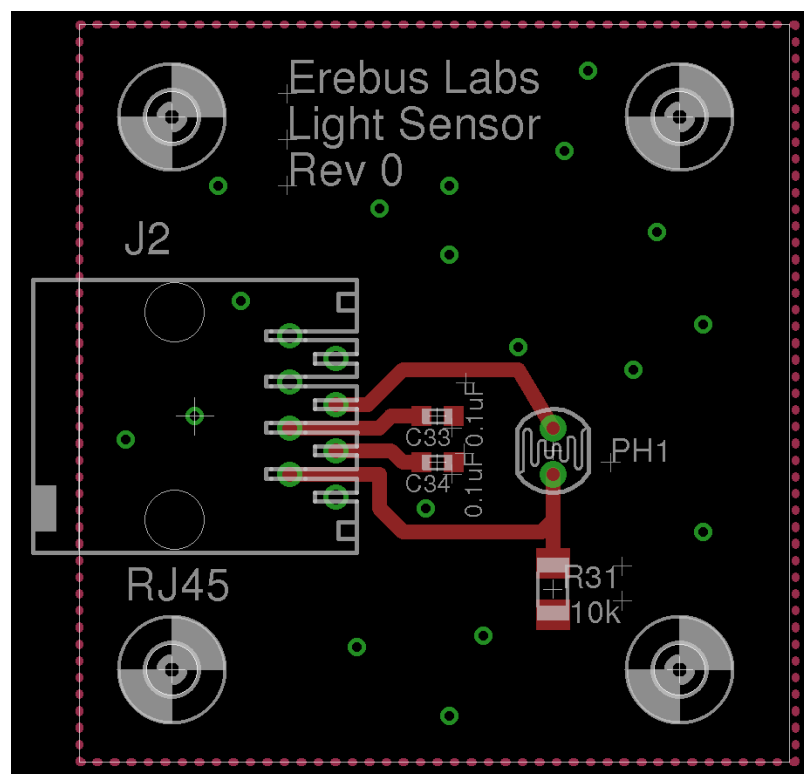
Base Unit Layout



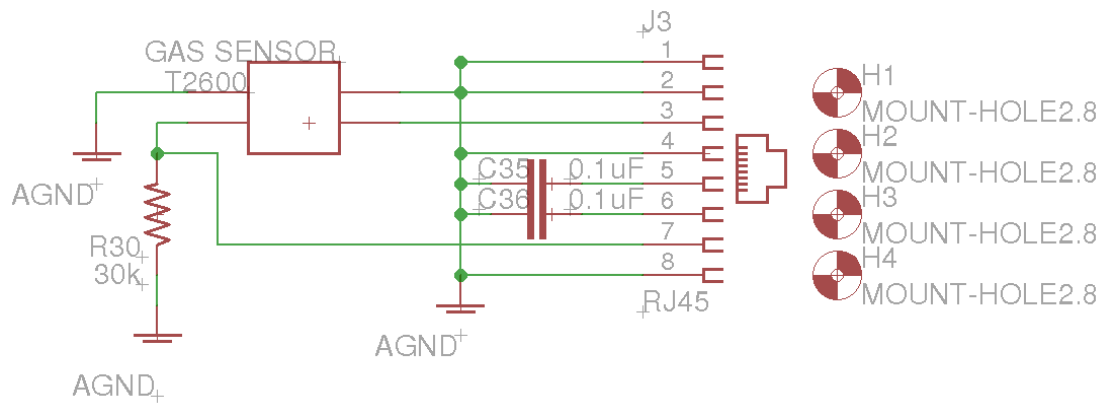
Light Sensor Schematic



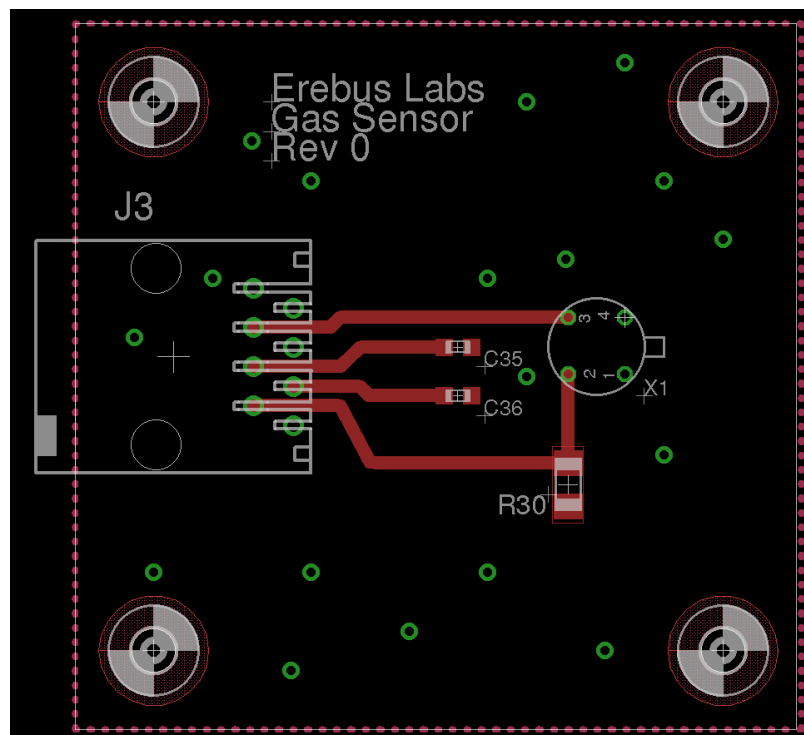
Light Sensor Layout



Gas Sensor Schematic



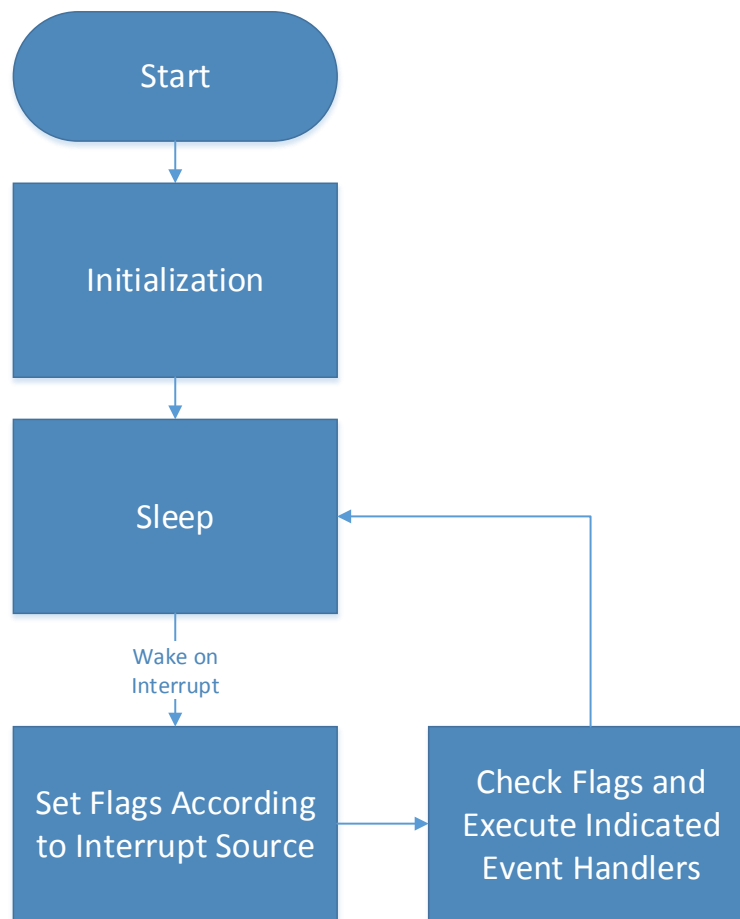
Gas sensor Layout



Firmware Design

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 will operate 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.

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.

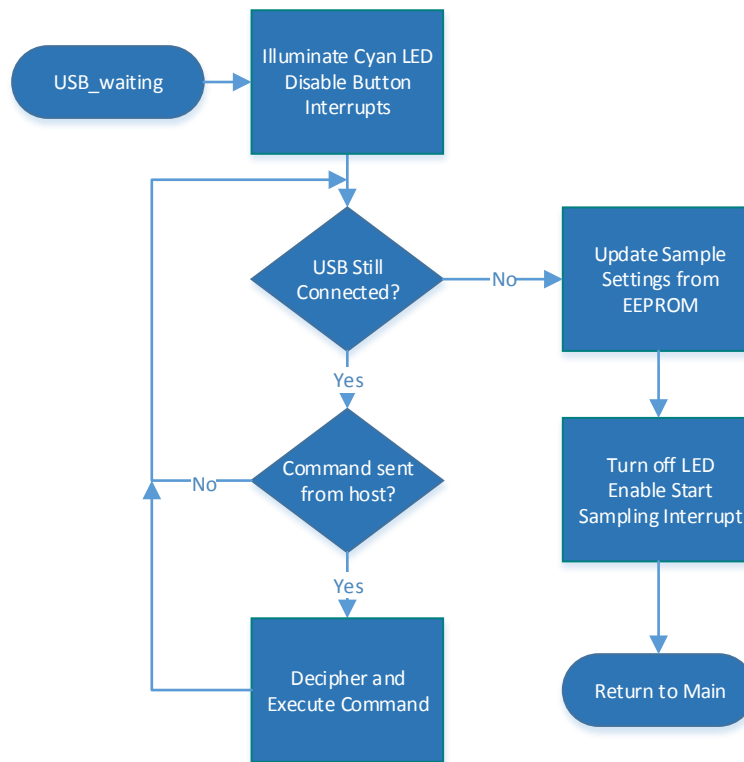
Note: RTC interrupt routines can be found in Generated_Source/PSoC3/RTC/RTC_Int.c. They all share one interrupt vector titled RTC_isr in Erebus_Sensor.cydwr -> Interrupts.

Event Handlers

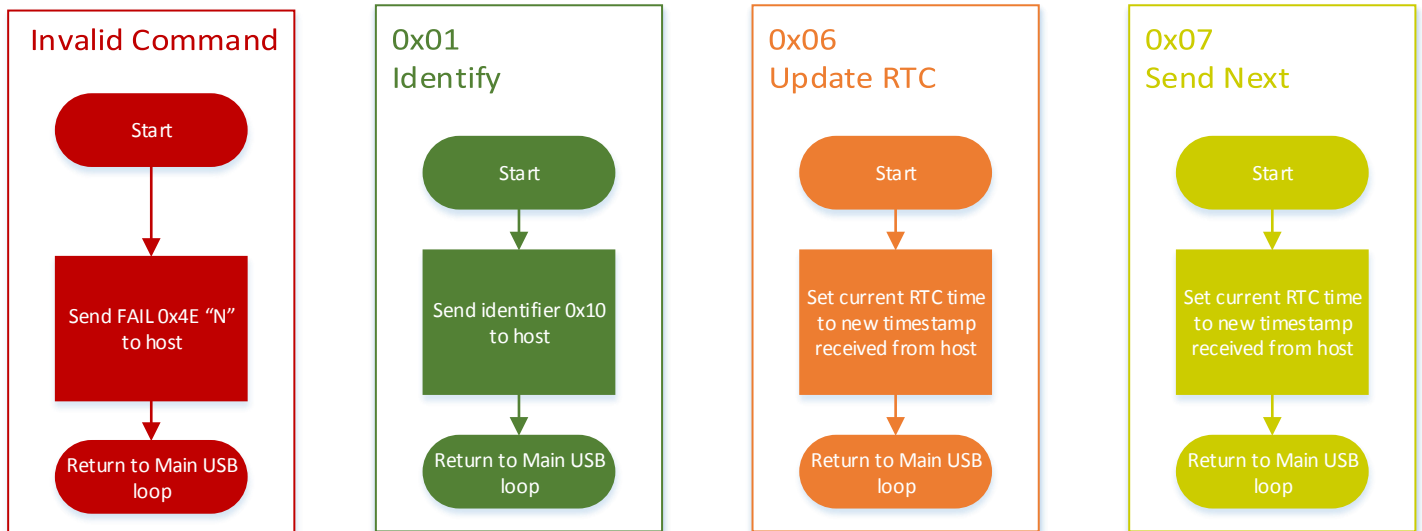
The following events are to be handled by function calls from main by the procedures described in these flowcharts.

USB Waiting

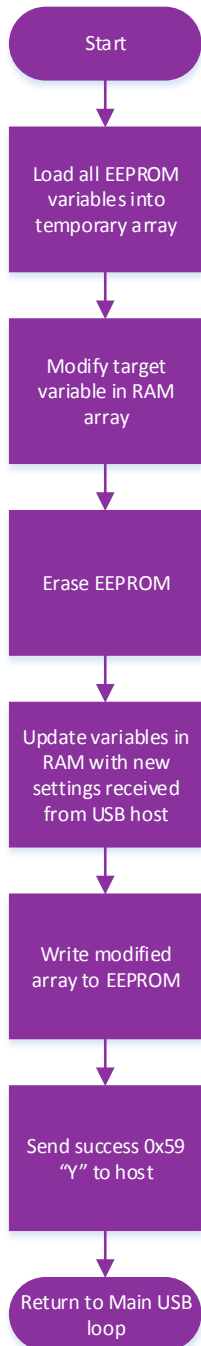
The USB_waiting flag indicates that the device's USB port has been attached to a host. The various commands that may be sent by the host are handled according to the USB Command Handling flowcharts below.



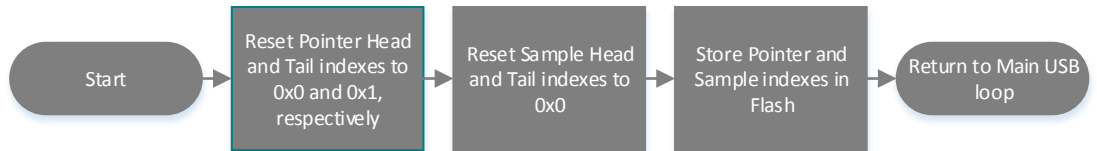
USB Command Handling



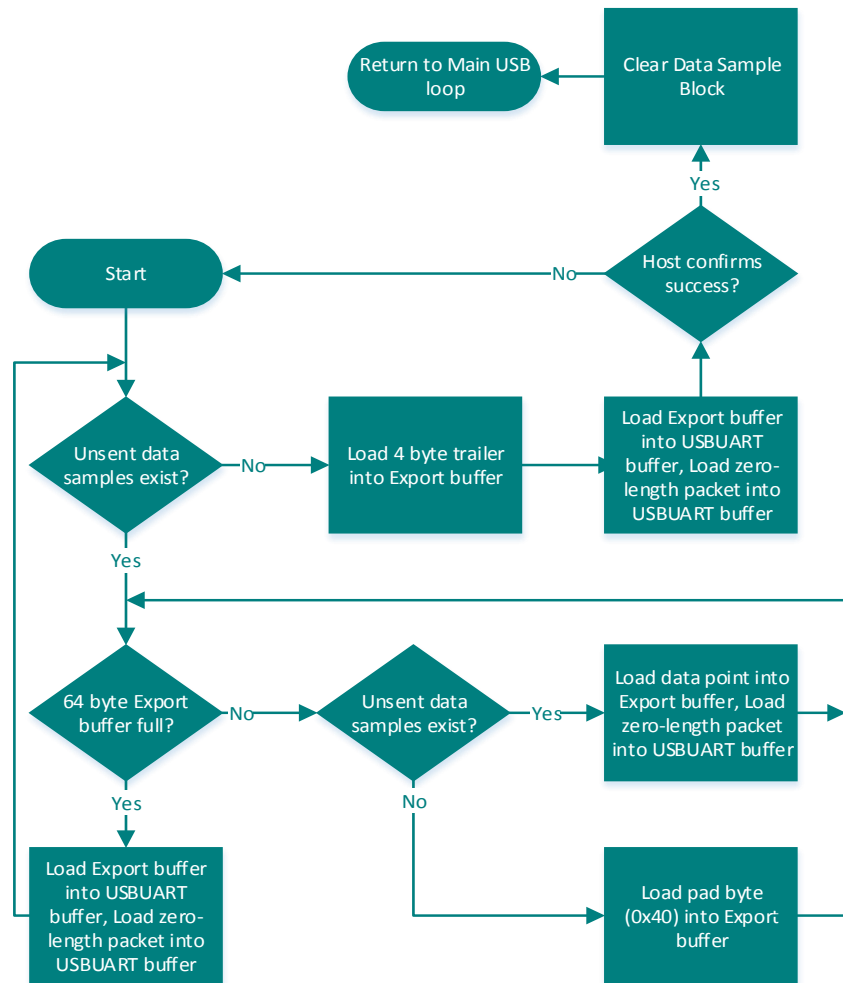
0x04 Change Settings



0x05 Reset Pointers

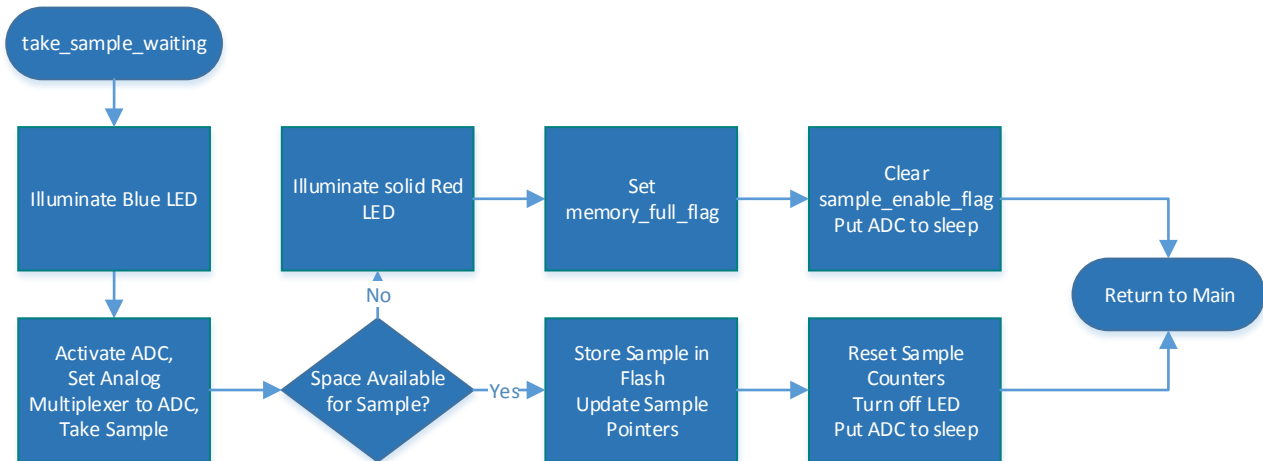


0x02 Dump Data



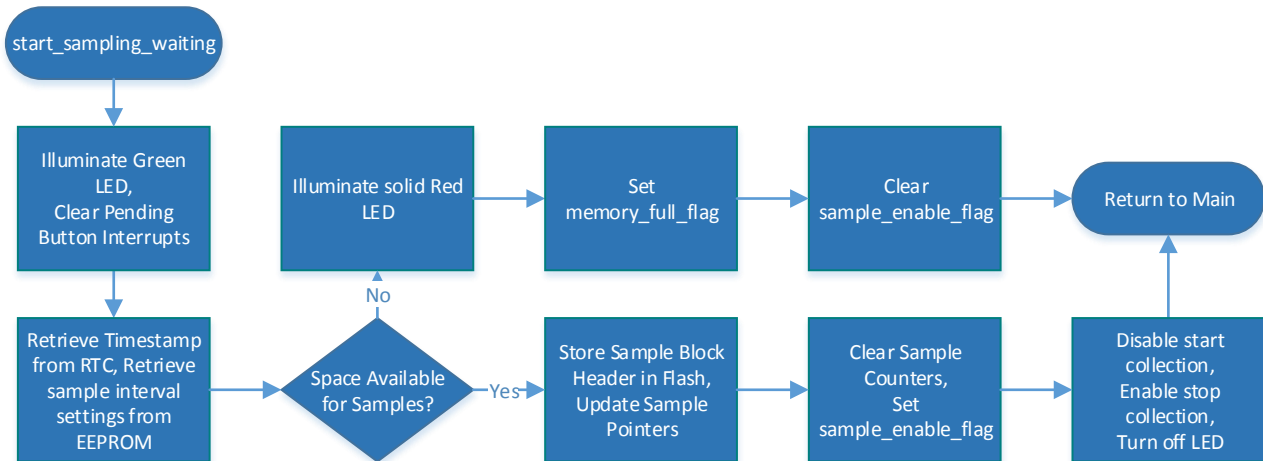
Take Sample Waiting

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.



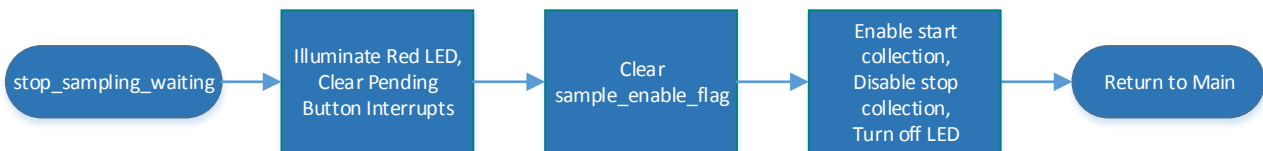
Start Sampling Waiting

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



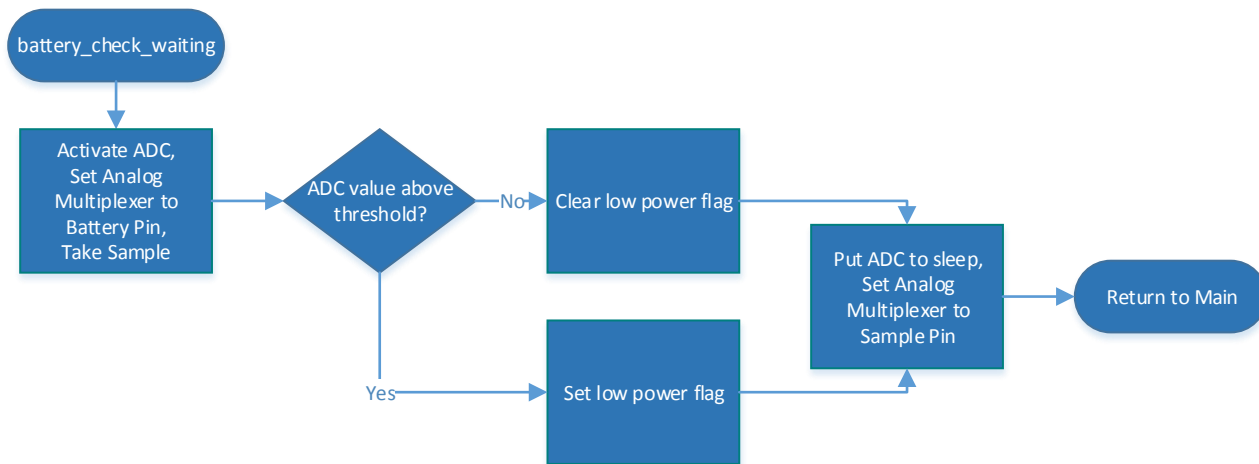
Stop Sample Waiting

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



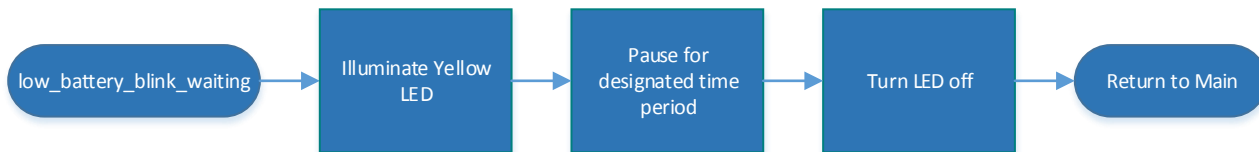
Check Battery Waiting

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

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 will use 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. 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.

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 Design

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 will remain in Interface Mode. During this time, the PSoC3 will continually monitor its input buffer for commands from the host computer and respond to them accordingly. The microcontroller will exit 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

Incoming Messages from Sensor

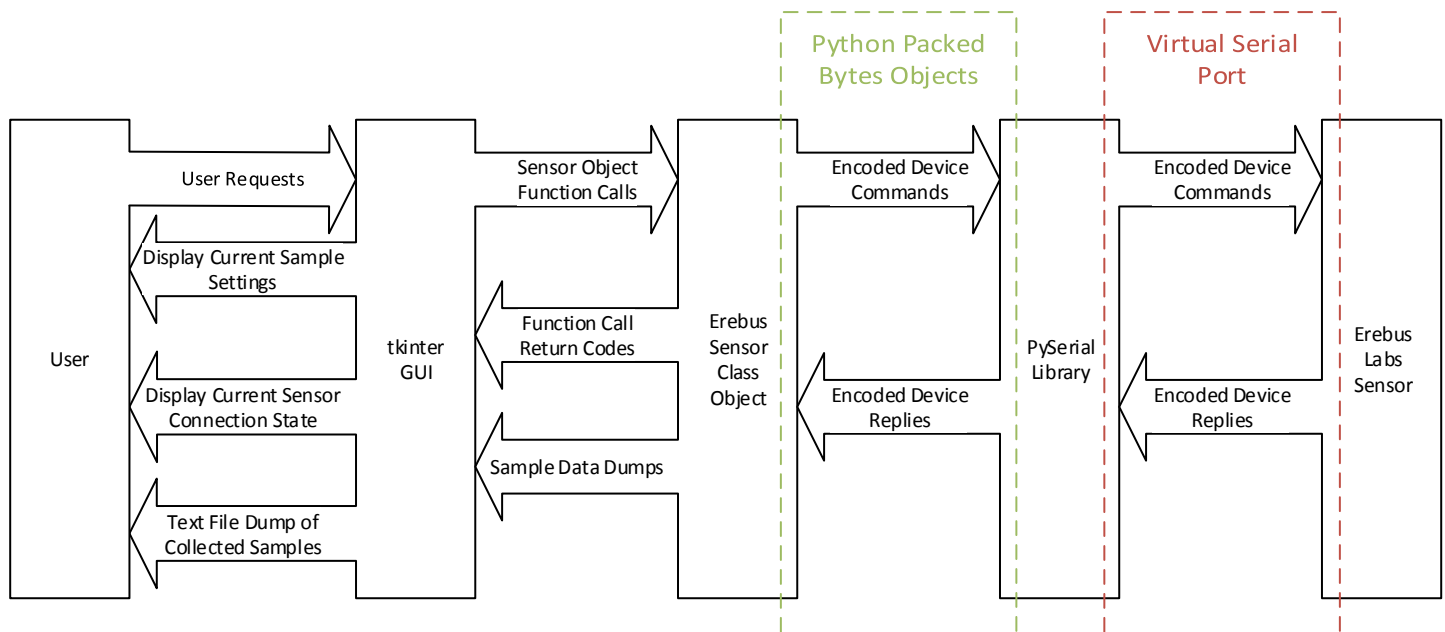
Messages	Bit Pattern	Meaning
IDENTIFIER	0x10	Differentiates Erebus Labs Sensor from other serial devices
SUCCESS	0x20	Last action was successful, ready for next command
FAILURE	0x30	Last action failed, reattempt command

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.