Greenhouse Monitoring and Regulation Project

Talmor Kliman

University of California, Santa Cruz

Jack Baskin, Electrical Engineering

Senior Capstone Project 2018

Table of Contents

Abstract

The monitoring project is part of the greenhouse research facility at the University of California, Santa Cruz which aims to provide an interactive electrical system for producing successful harvests. It is an ongoing project that is in its second iteration, 2017-18', with the goal of remotely monitoring and regulating the greenhouse environmental conditions. The client's needs are to create a plug-n-play metering hub and web interface for students to conduct horticulture experiments using solar energy. This year's productivity involved gathering old documentation, revamping the server, developing a user-manual, and creating a website. The ability to monitor the electrical grid through serial communication was researched, forwarded, and documented. To complete the prototype of the metering system there must be power-data analytics on the website along with charts and graphs of conditions. This document serves to assist in understanding the current state of the system and for future work.

High-Level Greenhouse Monitoring Network

Sunstainability Lab Greenhouse at UC Santa Cruz

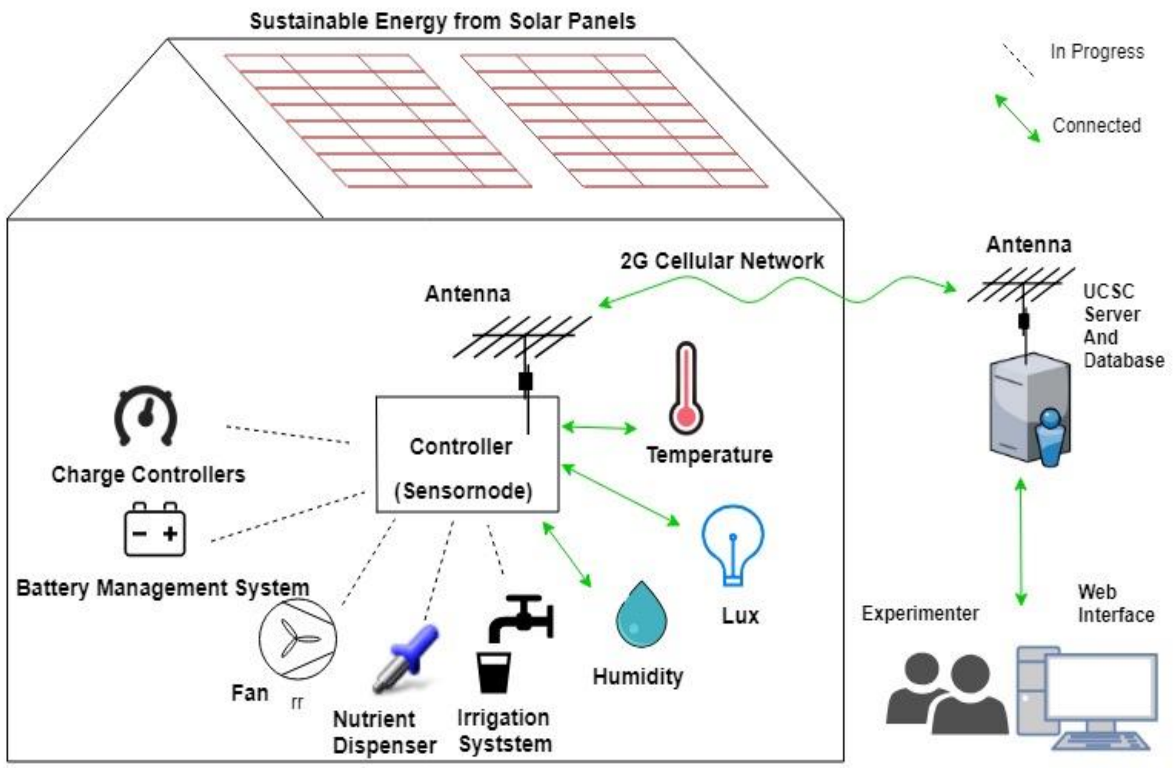


Diagram Summary

This diagram realizes the network of the monitoring system at the greenhouse at the University of California, Santa Cruz's Arboretum.The three main components are the following: microcontroller, UCSC server, and a website. The monitoring network is attached to a microcrid which is used for sustainable food system research solutions. The ultimate goal of the metering system is to have a plug-n-play sensor interface that can be accessed through a website to analyze and and regulate the plant environment.

Talmor Kliman
Revision 4
11-26-18

1 Introduction

1.1 Background

The methods and capabilities of growing crops have changed drastically along with the presence of the *internet of things* (IoT). Plant quality and health metrics were once gauged by a farmer's intuition and are now measured accurately by electronics. Computer networks and integrated circuits (ICs) combined are powerful and capable of growing food with high quality, anywhere in the world, at a low cost, with minimal effort. Cultivating food has become a science because the environment of crops can be controlled and monitored in depth when grown indoors. Growing food efficiently has become popular with global impact organization around the globe in pursuit of feeding everyone and reducing the human carbon footprint. Climate change has been a concern for the environment ever since the industrial revolution over 100 years ago. We have been seeing the effects of climate change and research has shown that the speed of humanity's carbon footprint will be counterproductive in the long-term. To aid to this massive-global cause, the University of California, Santa Cruz has pursued the mission of becoming carbon neutral by 2020. To do so, resources are funneled towards eco-friendly efforts at UCSC.

A home for innovation called S-Lab, short for *Sustainability Laboratory*, is an organization at UC Santa Cruz that provides students with the facility, resources, and mentorship to lead projects that contribute to the challenges posed by climate change. There are many ways to reduce carbon footprints and thus the S-Lab provides many facilities and resources to do so. One of the facilities, in importance to this paper, is a pair of greenhouses at the Arboretum. This branch of S-Lab aims to create sustainable food systems. In the past year, the greenhouses switched from city power to solar power (off-grid) and S-Lab is still working on completing the renovations. They desire a *metering/regulation system* for the greenhouses, which the *Monitoring Project* aimed to provide. The experimenters needed to effectively gauge resource usage and control/regulate the plant environment. This would allow them to accurately document their experimental solutions and communicate their findings to others effectively.

1.2 Metering System Orientation

This project began with some parts of the metering system already created from the sustainable-power team in 2017. An Arduino with a custom shield to easily plugin, software for easy sensor switching, and a server with a database for logging data was all working. A web interface did not exist before the Monitoring Project commenced. The team called their metering-hub *Sensornode*, which will be referred to in this document abundantly. As nice as this sounds, it was unusable because there wasn't enough documentation or a user-manual to modify it. It was found difficult to understand how to turn-on and use *Sensornode*. Nonetheless, with the efforts of former team member and engineer Isaak Cherdak, I was able to make significant contributions to the system. As of now, *Sensornode* has a website and is in a useable state for students. The greenhouse environment can be monitored, one can schedule the frequency that sensor values get logged, and one can add an entirely new *Sensornode* into the system. This is a remarkable tool for the S-Lab. One of the goals of this project that wasn't fully deployed was the ability to detect power. There aren't any power metrics due to the complications experienced with interfacing an Arduino with unique commercial devices, which is explained in the *Power Data Communication Section* for future students to build upon. The system has four main parts that are visualized in the *High-Level Greenhouse Monitoring System* diagram above and explained in the *Product Breakdown* section of this report. The sections include the design/implementation of the following subsystems: server (brain for operations), database (storage of memory), website (hands and eyes for user-input and monitoring), and hardware power-data (feet for logging the data).

2 System Level Overview

As stated in the introduction, the S-Lab is facilitating student research and community engagement. The greenhouse facility is home to student-led experiments in pursuit of finding innovative solutions to supply food with sustainable energy. The customer's call to action was to create a plug-n-play system that gives students full overwatch and control of the greenhouse online and ability to modify the sensor metrics to be flexible with experimentation needs. The design requirements were translated to design speculations that will be broken down in each subsection of the *Product Breakdown* section. A website is needed to monitor, regulate, and control the greenhouse. A modular-backend with a database is needed to grab data from the sensor-clusters and store it appropriately. Power data-analytics (from the MCC and BMS) are needed to give the user more control of the greenhouse environment. Lastly, to bind the system together, it ought to have a governing set of rules to integrate all the parts together to act as a single cohesive entity. The design is realized in diagram X that shows the components of the data-logger and the web of connections between the hardware and the user.

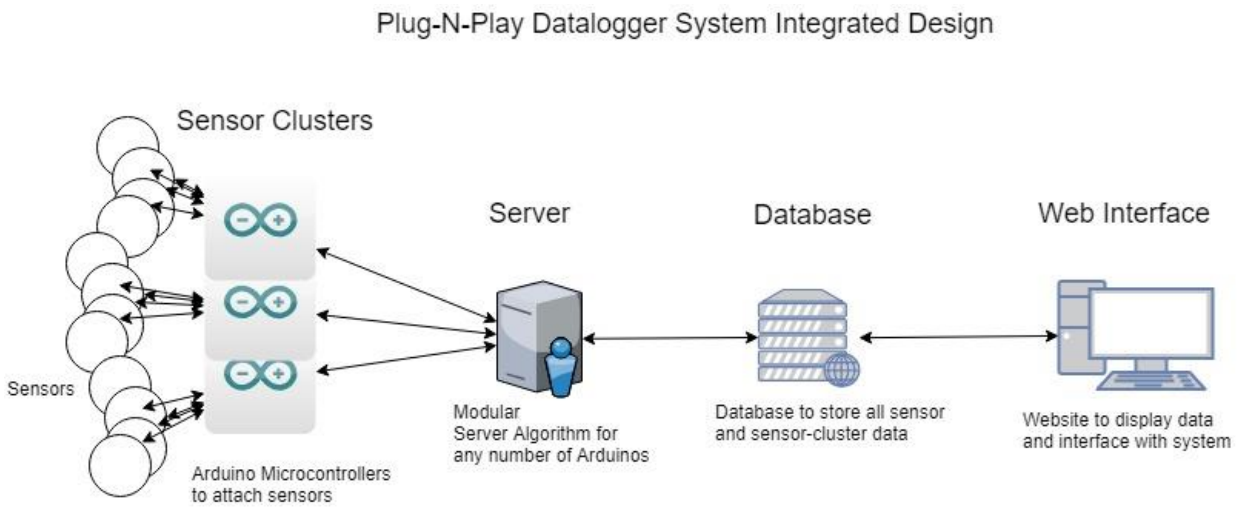

Plug-N-Play Datalogger System Integrated Design
Sensor Clusters
Server
Database
Web Interface
Sensors
Modular Server Algorithm for any number of Arduinos
Database to store all sensor and sensor-cluster data
Website to display data and interface with system
Arduino Microcontrollers to attach sensors

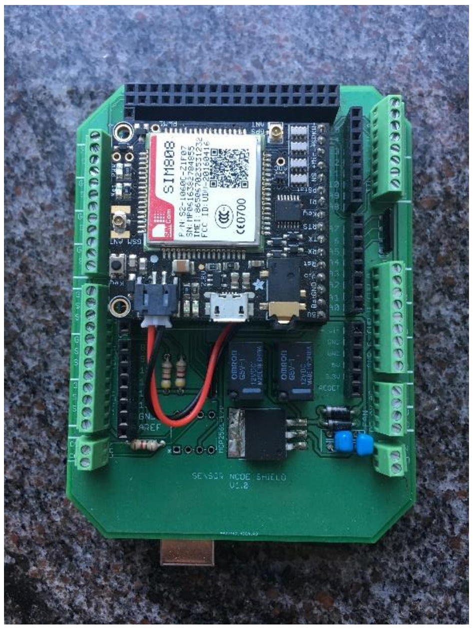
Diagram X Caption: The design requirements of the customer were to have full control of the greenhouse environment by having a plug-n-play system for student-led projects. The diagram shows each subsystem and integration in the data-logging system.

The system diagram shows how the highway for the data which flows throughout the system to achieve full control and a plug-n-play system. The Sensor Clusters, which are also called *Sensornodes*, connect to various sensors to monitor the environment and electrical system at the greenhouse. The server is designed to ping all the sensor-clusters to retrieve the data and store each in the database. The database is the medium between the server and the web interface. The web interface displays the information of each sensor, enables the user to adjust the ping-frequency of a sensor-cluster. This design aims to achieve the goals of a fully automated and modifiable system for all experiment styles and needs. The *Power Data Communication* section speaks about the contribution to the research needed for data from the power grid, the *Backend* section discusses the algorithm for the server to cycle through all *Sensornodes* to retrieve data, the *Website* section discusses the user-experience on the greenhouse website, and lastly the *Database* section discusses the structure and rules that binds the entire system together. This report section *Product Design Breakdown* will begin with the hardware (the source of the data) and make our way through the system (Server and Database) and end at the web interface (the destination of the data).

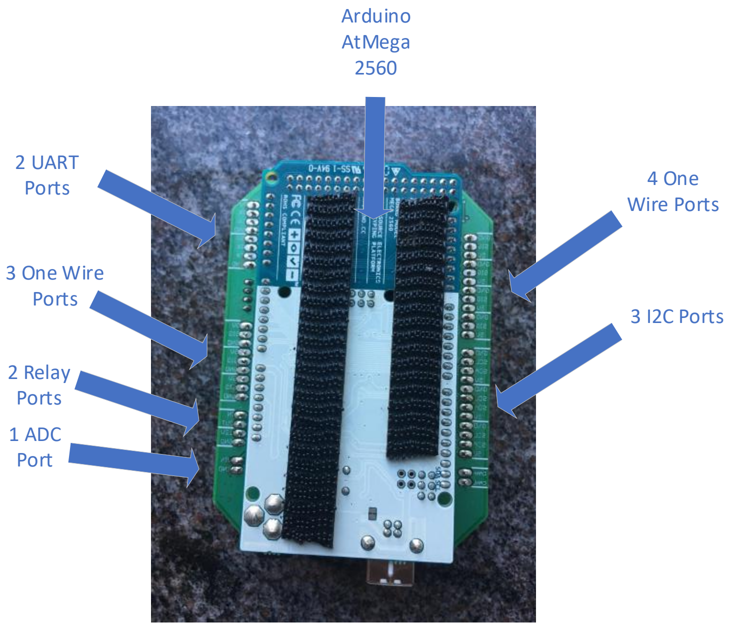
3 Product Description

(additional and clearer photos can be found in the application notes section)

Sensornode



Sensornode Ports for Sensors



3.1 Speculations and Capabilities

Overall System Speculations Capabilities:

- Logs Temperature, Humidity, Light (I^{2}C (3), SPI (7), I/O pins (42), UART or USART (2), ADC pins (1), Relay switches (2))
- Actuate Fans
- View live sensor data on website
- View live sensornodes on website
- View schedule for each sensornode on website
- User can edit schedule for sensornode

Additionally:

- User can Add/Remove sensors to the system
- User can Add/Remove sensornodes to the system

Subsystem Speculations:

Arduino AtMega2560:

- Custom easy-plug-in hardware interface (daughterboard)
- 2G Cellular Connection (FONA module with Ting subscription and SIM card)
- 256MB memory (remaining), 24G (Used)
- Requires 5V power supply

Note: additional AtMega 2560 specs can be found here

Database:

- MySQL data storage
- PHPmyadmin controlled website
- 1 and 2 provided by UCSC

Server:

- Unix Timeshare
- Provided by UCSC
- Hosts the website and the

Website:

- Node.js web framework
- Javascript, HTML, embedded EJS
- Hosted by Server

3.2 Installation and User Manuals

Video Tutorial Contents

- How to Use Overall System
    - Sensornode (Metering Device)
        - What are the components of sensornode
        - How to plug in sensors
        - How to program sensors
    - Website
        - What is the website used for?
        - How to interpret sensor data page on website
        - How to interpret scheduling page on webiste
        - How to set a sensornode’s schedule from website
- How to Setup Software to use Sensornode
- How to add a sensornode to the system
- How to use the system as a developer

Links to tutorials:

Sensornode and Website User Manual:

The video introduces Sensornode and its features. This is a step-by-step tutorial on how to connect sensors to the Arduino, run the code on the Arduino, and how to use the website interface for viewing the sensor readings.

Link: https://www.youtube.com/watch?v=YGanolWRI1Y&t=3s

Sensornode Set-Up Tutorial

The set-up video provides all the information for setting up the software for programming the Arduino. The set-up is required for connecting new sensors and sensornodes to the system.

Link:

Add Sensornode Tutorial:

This video demonstrates how to connect an additional Arduino to the metering system. The tutorial explains how to navigate the database, Sensornode source code, and the website for properly integrating another Sensornode.

Link: https://www.youtube.com/watch?v=3hHCDe_S5vY

4 The Product Breakdown

4.1 Modular Server and the Database

The backend enables the greenhouse to be monitored remotely through an online website. This piece of the system is the core of the network highway. The backend, nicknamed the *server*, grabs information from the sensors at the greenhouse and puts it in a database. Logging the data is one step away from presenting it to the researcher to be analyzed.

The server is the operator that connects the call between the experimenter and the sensors. The content of this section is intended to help modify, fix, or use the data-logger system for its full-potential. The intended takeaway from this reading is to understand the server operations and role in the data-logger system. The backend was designed to be malleable and easy to build upon as discussed in this reading.

At the start of the project the server could handle a single Arduino with a fixed number of sensors. The backend structure was revamped to achieve the goal of a plug-n-play system. As of now, the server can handle any number of sensors and any number of Arduinos. The once hardcoded design transformed into a modular and systematic design.

The backend consists of a server and database. The backend lives on the Unix Timeshare Server space provided by UC Santa Cruz. The software is written in C and compiled and run on the Unix Timeshare and accessed through a login and password provided by Tela Favaloro. The server-space is hosted by UCSC but is not supported by the IT department because it does not follow the policy rules. Less regulation means more freedom, but also more responsibility. This gives S-Lab engineers full control and design freedom meaning we can write our own code but we must fix our own bugs.

An important note on the server is that it is on a desktop computer connected to the internet in Baskin Engineering 2. The computer has BLANK MEMORY and runs Ubuntu as the operating system. The software needs to be updated periodically. Note: more information on the server technical details can be found BLANK.

The backend grabs information from Arduinos on a schedule and logs it in the database. It is designed to relay information between Arduino's based on a schedule it references in the database.

Key features of the backend

- The server connects and retrieves data from each and evert sensornode
- The server periodically updates its own information from the database (ping schedules, sensornode clients)
- The server creates tables and inserts data in the database for each sensornode
- The server uses timers, sockets, and mysql commands as a means for executing operations

The backend was designed with 3 main operations for simplicity. The states are contained within one governing statemachine shown in Diagram X. The customer desired the ability to scale the number of sensors and sensornodes up or down.

The main goal for the backend was to get sensor information from an Arduino any number of sensors. This meant the number of sensors could not be hardcoded but rather actively known. The method to get data was to request the number of sensors on each connection to an Arduino and then ask it to send each sensor's reading over. This method was chosen because it was reliable, simple, and it met the needs of the customer.

Another goal of the backend was to make it modular to make it able to talk to multiple clients. The method used was to have the server use a pre-existing list of Arduinos from the database and then call each individually. Using the database list met the needs of the customer and allowed for future modifications. If the list of sensornodes is modifiable through the website than the system is prone to being misused. It is important that an administrator is in control of the sensornode list until the system is more resilient to unproperly formatted information.

As of now, if the server is unable to make a connection with an Arduino then it freezes and waits. This is a known bug for the server and Arduino link. The problem is in the port-connection function which stops the program until a successful connection is made. There are flags that could be set to abort the function, but this option did not work and should be further experimented with. The open-source code for the port connection was not fully understood during this project and is a point of interest that needs to be documented in the future. The other attempt was to time-out after a failed connection or after a long period of time. This did not work either because once the port-function attempts a connection, it blocks the program until the Arduino responds. Another solution that was considered as recommended by Isaak Cherdak was threading. This is where the computer executes Arduino connections independently of one another. Threading can improve time efficiency of logging data, but the customer's needs are tolerant to the amount of time it takes to retrieve sensor data. It was considered but the idea was dismissed due to its complexity and it was not a priority. Threading should be considered for this system because of its benefits, but it may call for a major redesign of the system. Alternatives, benefits, and resources should be weighed before pursuing a threaded system approach.

The solution for communicating with multiple arduinos is explained with a statemachine in Diagram C which creates connections with arduinos one at a time based on information from the database. A complete redesign of the backend was required to increase the capacity of Arduinos. The answer to increasing the server's capabilities was to create a process in which a list of Arduinos is iterated over using a for-loop so each is communicated to one-by-one. In other words, each sensornode would be pinged by iterating over a list of known clients. Therefore, a masterlist of sensornodes had to exist in the database. On top of pinging sensornodes, the client requested that each sensornode get pinged at different times. A list of schedules to ping each sensornode exists in the database called *pingtiming*. To allow the user to actively change the schedules, the server has a periodic time-out to update its information from the database.

The server executes operations based on the database content as discussed prior. Therefore, changing something in the database causes a change in the server. By being able to modify the database through the website, the user can control the system online. As of now, the user only has access to the ping schedules of each sensornode through the website. In the future, there should be a secure way to edit critical information such as having an administrator page and login.

To add more controls and new features to the system, one should add a new list to the database and give the user access to it through the website. For example, if the fans will be controlled from the website than a list of fans with an on/off attribute should be created in the database and the user should be able to toggle it through the website. The server will respond by communicating with the right Arduino to turn the fan on/off.

The server grabs information from each sensornode based on a schedule. The algorithmic process is realized by Diagram C. A more in-depth explanation of how the software algorithm works is described in Diagram D. The subprocess was designed in a statemachine-style that can be easily built upon by adding another state. The *Machine* talks to all sensornodes that are scheduled and then resets their timers. There is an issue in the timing of each connection because the communication schema focuses on one client at a time. Therefore, if two Arduinos are scheduled within a minute of each other, one must wait until the other finishes. Also, all the timers reset after all Arduinos are communicated with, which is not coherent with when the Arduino was pinged. The timer should reset at the end of each sensornode's connection rather than at the end of the entire communication cycle. The delta of inaccuracy is about 3 minutes in delay per sensornode. So, if there are 3 sensornodes in the queue, then the first sensornodes timer will restart 6 minutes in delay and the second sensornode in the queue will delayed 3 minutes, and the last sensornode will not be delayed at all. In terms of the code, the *Reset Timers* state should be combined with the *Ping Sensornodes* state. The reason the statemachine exists is to set a standard for the design of the backend. If a statemachine is used, then the processes can be categorized into states which is scalable, modular, and easily to understand for others to modify. A future suggestion is that the *Update Information* process is given its own state in the statemachine and that all processes are contained within the statemachine for the simplicity it offers.

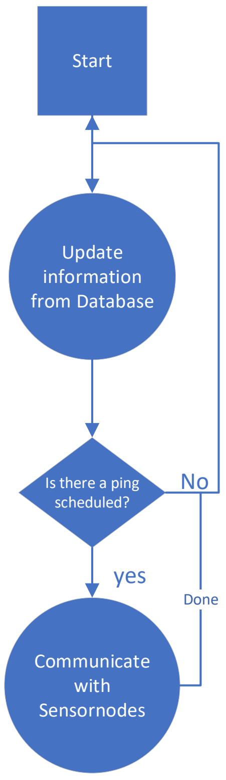

Start
Update information from Database
Is there a ping scheduled?
No
yes
Done
Communicate with Sensornodes

Diagram C: Server Algorithm for grabbing data from sensornodes

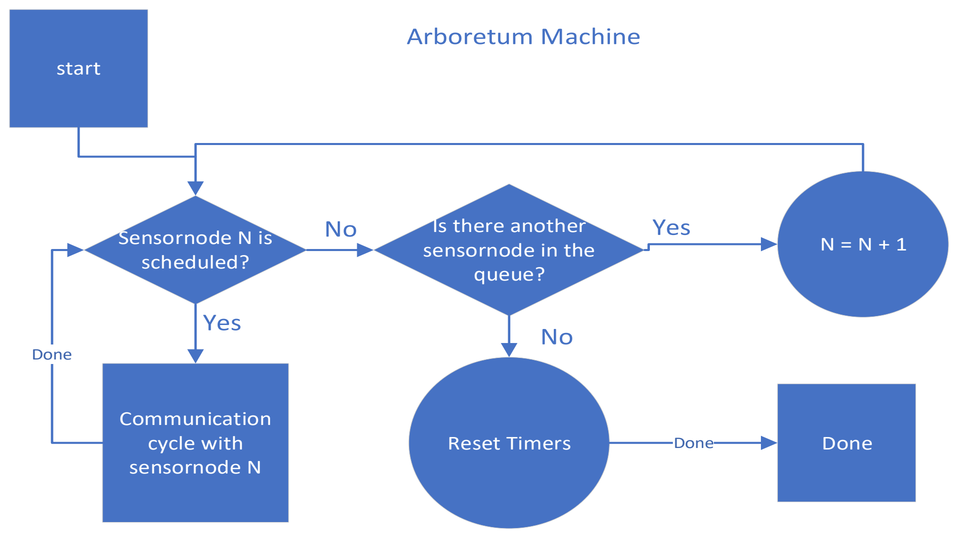

Arboretum Machine
start
Sensornode N is scheduled?
No
Is there another sensornode in the queue?
Yes
N = N + 1
Yes
No
Done
Communication cycle with sensornode N
Reset Timers
Done
Done

Figure 1 Arboretum Machine is the algorithm for cycling through all scheduled Sensornodes and is executed in the "Communicate with Sensornodes" state in Diagram C

Diagram D: The Algorithm for pinging all scheduled sensornodes and retrieving all of their sensor data.

The server uses a statemachine for contacting each sensornode and within each connection with a sensornode, a substatemachine is in effect. After establishing a connection with an Arduino, the server uses a custom protocol to extract the sensor information. The rules of the protocol are described by the cycle in Diagram E. Three valuable pieces of information are extracted in the protocol; name of the Arduino, number of sensors, and sensor information (name, type, location, value). First the server requests the Arduino name, then asks for the number of sensors, and finally it asks for each sensor's information one-by-one. Each sensor has a few pieces of information to describe it. That way the sensors can be grouped by any of their attributes such as location or serial type. Categorizing sensors makes it easy to display them in a readable fashion on the website.

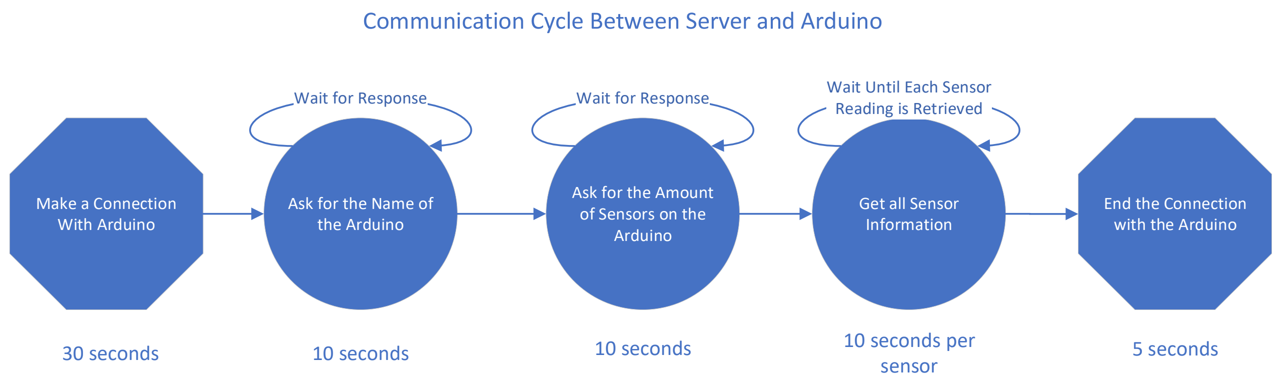

Communication Cycle Between Server and Arduino
Make a Connection With Arduino
Wait for Response
Ask for the Name of the Arduino
Wait for Response
Ask for the Amount of Sensors on the Arduino
Wait Until Each Sensor Reading is Retrieved
Get all Sensor Information
End the Connection with the Arduino
30 seconds
10 seconds
10 seconds
10 seconds per sensor
5 seconds

Diagram E: Shows the protocol for retrieving data from the sensors to the server. The data retrieved is the name of the Arduino, amount of sensors connected to the Arduino, and the sensor information (name,type,reading,location). The estimated duration of each process is described below it.

The information of each sensor is the following: name, location, type, serial type, reading. The implementation of the sensor attributes and categorization schema can be improved. The extra pieces of information such as the type, location, and serial type were included for a few reasons: to provide information for future design choices (modular), to allow for easy organization in the database, to describe the sensor to the user. There are probably more ways to improve organization to keep track of the sensor information, but this meets the client's needs. For our goals, retrieving all information during every cycle was practical. Some of the information is redundant and can be redesigned in the future if applicable to the client's goals. The style of retrieving information plays a role in how the data is queried in the database. Thus, the design of the database is contingent on the way the server interacts with the data when it pulls and pushes information in storage. On the contrary, the website must also be considered in the design of the storage architecture because its purpose is to display the information in an organized fashion.

The database is designed to easily access information about each MCU. The three main categories of information are:

- Sensornodes
- Schedules
- Active Sensor Data

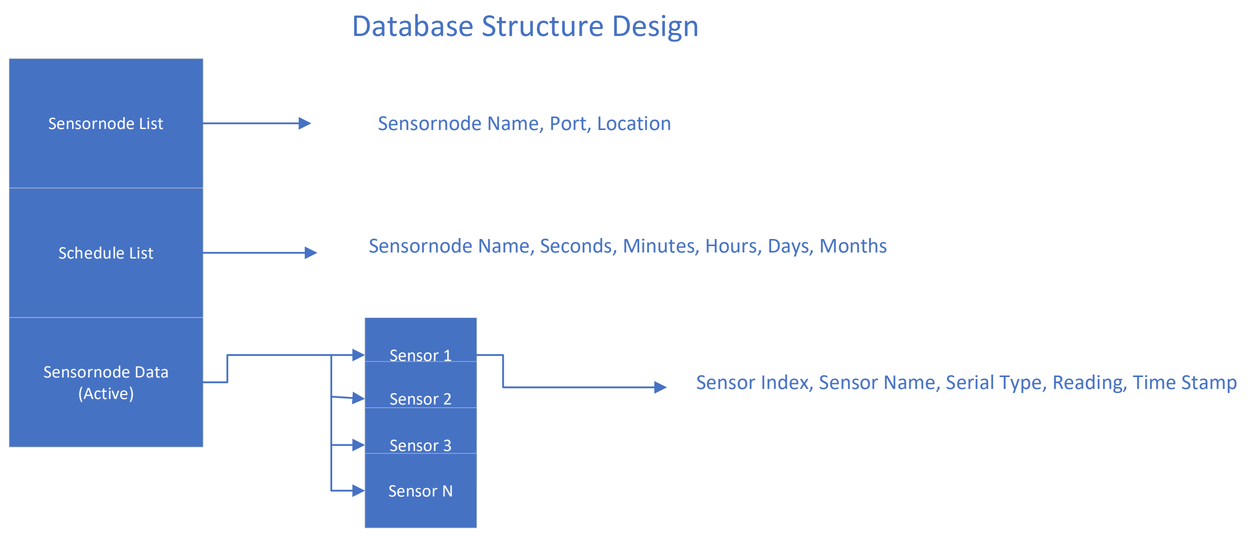

Database Structure Design
Sensornode List
Sensornode Name, Port, Location
Schedule List
Sensornode Name, Seconds, Minutes, Hours, Days, Months
Sensornode Data (Active)
Sensor 1
Sensor 2
Sensor 3
Sensor N
Sensor Index, Sensor Name, Serial Type, Reading, Time Stamp

Figure 2 Database Structure Design: Shoes the three types of lists that are stored in the database. This reflects the data displayed on the Arboretum-Backend website.

The database is accessed by using MYSQL commands provide by the C Library BLANK. The Sensornode list was created manually through the PHPmyadmin website as well as the Schedule List. The Server creates the Sensor Data tables.

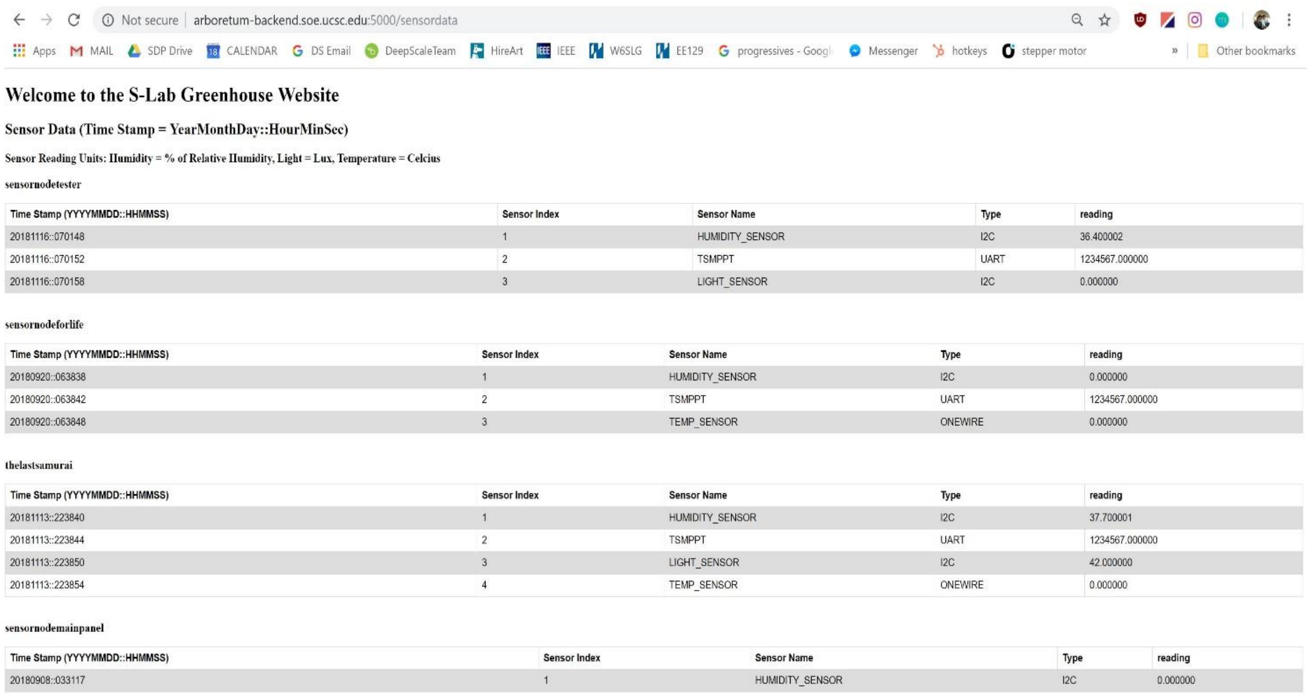
Every time the server gets information from the Arduino, it clears the table of its information and then logs the most recent data. By design, each sensor will have two lists, an active list and a history list. Another method that might be more efficient is to have one list for active and historical data. What to watch out for with a single-list design is the increased difficulty when trying to query the most recent sensor values. Since the amount of sensors are changing, I believe that having all the data in one chart will cause incoherence in the modular system. A suggestion for making the single-list method work is to have a list of active sensors for each sensornode to reference the names before querying the sensor data. This technique suggests that each sensor needs to have a unique name to access each active sensor's most recent reading. That way was not used because it did not seem practical for a modular design because the number of sensors will be changing. It is more practical to have a list of all active sensor data and then have a list historical data for each sensor. Sensors need to be able to be removed and added to the system and therefore they must standalone in the database to be easily accessed. By having a list of all active sensor data, the website can easily retrieve that information to display it to the user.

SensorIndex keeps track of the order the sensors were created on the Arduino which is useful for adding/removing sensors and identifying a sensor. Each sensor doesn't have a unique name so the SensorIndex attribute allows for recognition between two identical sensor types. In the future, each sensor should be able to have a unique name to identify it which should be done on Sensornode during initialization or by the user through the website.

4.2 Monitoring Website

The Website is a portal for the user to monitor the conditions at the greenhouse and configure the system. The goal is to have a user interface where the user can visualize data, rapidly orient themselves on any critical states, and regulate the features of ones experiment. Ultimately, the website enables the ability for effective horticulture experiments with interactive controls and historical data analysis. All these goals are aligned with the Sustainability Lab's pursuit to provide a research facility for innovate research to create holistic solutions to the challenges posed by climate change. The monitoring website produces an analytical experience for the experimenter which will encourage innovation and increase the percentage of fruitful harvests.

The main goal for this website is to be able to see the live sensor readings for each of the sensornodes. A page for all the sensor data is shown in Figure XY. The sensors are grouped into categories according to which sensornode they are attached to. The intended use of the sensornodes is to place them in various locations in the greenhouse(s) to regulate each subsection of the environment. Each table on the sensordata page is a specific sensornode with all the sensors created on the device with their attributes on that row.



Welcome to the S-Lab Greenhouse Website

Sensor Data (Time Stamp = YearMonthDay::HourMinSec)

Sensor Reading Units: Humidity = % of Relative Humidity, Light = Lux, Temperature = Celcius

sensornodetester

Time Stamp (YYYYMMDD::HHMMSS)	Sensor Index	Sensor Name	Type	reading
20181116::070148	1	HUMIDITY_SENSOR	I2C	36.400002
20181116::070152	2	TSMPPT	UART	1234567.000000
20181116::070158	3	LIGHT_SENSOR	I2C	0.000000

sensornodeforlife

Time Stamp (YYYYMMDD::HHMMSS)	Sensor Index	Sensor Name	Type	reading
20180920::063838	1	HUMIDITY_SENSOR	I2C	0.000000
20180920::063842	2	TSMPPT	UART	1234567.000000
20180920::063848	3	TEMP_SENSOR	ONEWIRE	0.000000

thelastsamurai

Time Stamp (YYYYMMDD::HHMMSS)	Sensor Index	Sensor Name	Type	reading
20181113::223840	1	HUMIDITY_SENSOR	I2C	37.700001
20181113::223844	2	TSMPPT	UART	1234567.000000
20181113::223850	3	LIGHT_SENSOR	I2C	42.000000
20181113::223854	4	TEMP_SENSOR	ONEWIRE	0.000000

sensornodemainpanel

Time Stamp (YYYYMMDD::HHMMSS)	Sensor Index	Sensor Name	Type	reading
20180908::033117	1	HUMIDITY_SENSOR	I2C	0.000000

Diagram XU: Shows the page for all the live sensor data logged by the Sensornode. Each table represents the group of sensors connected to a specific Arduino (Sensornode) that is displayed only if it is in the list of sensornode clients.

The tables in Diagram XU are representative of the typical use of the monitoring system. It displays a batch of sensornodes and their sensors. Lets take the third sensornode named *thelastsamurai* to understand the contents of a table.

The first column is the Time Stamp which represents the local time (PST) in which the sensor reading was inserted into the database which has a 10 second latency from the time of retrieval. The delay is due to the time from the moment of reading the sensor to sending that data over the 2G network to logging the value.

The second column is the Sensor Index which is indicative of the order in which the sensors were created on the sensornode device. By having an order of sensor instantiation, a user can do the following:

- Distinguish between sensors of the same type
- Command the Arduino by using the sensornode features (see microgrid 2017 report, section Sensornode)
- Track the chronological order of sensor creation

The third column is the Sensor Name which is a universal name given to the sensor when it is created. It represents the utility of the sensor. The types of sensors that can be created are the following: Humidity, Light, Temperature, AC Relay, and a switch (MCU I/O pin). Each sensor name is generic and not unique to any one sensor. The index given to the sensor is what makes it unique and recognizable. As it was not a priority of this project to uniquely name the sensors, it follows to be the task of the next project affiliate(s) to introduce a method for naming and identifying sensors that is coherent with system, conscious of the application, and beneficial for the user. The next step to follow, when the system is more developed and tested in real applications, a wider range of sensors should be made available according to the needs of the experimenter. To create new sensor types, a designer can utilize the object-style architecture implanted by Isaak Cherdak. Any type of sensor or utility can be created and added to the system naturally and efficiently by adding following the adopted Sensornode guidelines (Microgrid Report 2017).

The fourth column, the type, represents the serial communication required for using the sensor device. Whether the connection type is a simplex, half-duplex, full-duplex, etc. is necessary for the following reasons:

- Recognizing the availability of space on the Arduino for the type of sensor (see the speculations in section X and the *Sensornode User Tutorial*)
- Understanding how to wire the sensor to the Arduino (which pins to connect to, how many ports are needed, and matching wire types to pin types)
- Understanding how to classify a sensor to the appropriate serial communication requirements for creating a new sensor type on the Arduino

The reason for categorizing sensors into types is for universally passing information between devices. In the practices of telecommunication and data transmission there are standard protocols and practices that were adopted to effectively transferring data. The standard changes over time due to technological advancement and improved communication techniques. As of now, Sensornode is programmed to recognize the following communication protocols:

- Integrated-Integrated Circuit - I^2C
- Serial Peripheral Interface - SPI
- Analog to Digital Converter (ADC) pins (not a protocol but used for similar communication applications)
- Input/output (I/O) pins (not a protocol but used for control and monitoring)

In progress and Available:

- Universal Asynchronous Receiver Transmitter – UART
- Universal Synchronous or Asynchronous Receiver Transmitter – USART
- Controller Area Network - CAN bus
- Recommended Standard - RS-232

These programmed protocols are unique to the sensors that are currently used by sensornode. That is, the I^2C code is uniquely used for the BLANK humidity sensor and the SPI is used for the BLANK temperature sensor. The analog to digital converter (ADC) and input/output (I/O) pins and code are available for various applications. While the ADC and I/O pins do not fit into the category of the serial communication protocol category, they do have important roles in the monitoring system. They allow for example the user to enable/disable, read the states of on-board switches and configuration shunts, and drive LEDs. The application of the ADC and I/O pins are in the hands of the experimenter's imagination. These pins have more simple applications than the standard serial communication protocols, but they still require unique code and care to be useful for the user. One application is to allow the user to send an On/Off signal to the Arduino I/O pin to actuate a valve, LED, valve, or a simple switch or application of sorts.

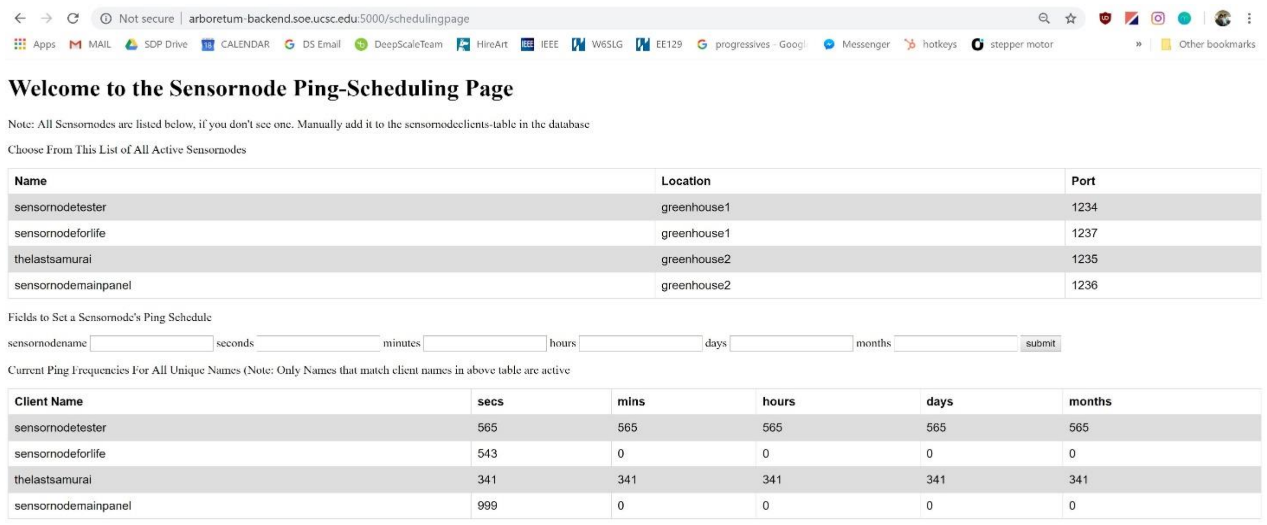
The fifth and last column is the reading which is the gauge or signifier of the sensor. It is the most important of the columns because it shows what value the sensor is detecting at a given moment. These values have different meanings for different sensors and the units for each are listed at the top of the page. The units are categorized by the sensor name such as TEMP_SENSOR is measured in Celsius. The following is the unit measurement for each of the sensor names.

Sensor Name	**Application**	**Units of Measurement**
TEMP_SENSOR	Temperature	Degrees in Celcius
HUMIDITY_SENSOR	Humidity	% of relative humidity
LIGHT_SENSOR	Light	Lux (luminous flux / unit area)

The sensor data on this page only displays the live sensor data meaning it is the most recent data that was retrieved by the server from each Arduino. The webpage is static and doesn't update itself which means a user needs to refresh the page to view the most recent sensor values. To use this page effectively it is important to know that the data displayed in the tables can be incorrect if a person loads the webpage at instance of data retrieval from a specific sensornode. It is an attribute of the system that the tables data in the tables are updated by clearing the rows and then repopulating them with the currently retrieved data. This design feature assures that the data displayed on the website is indicative of the current state of each sensornode and it sensors. If something is changed on an Arduino such as a deletion of a sensor or creation of a sensor, than the website quickly represents that change by listing the current number of sensors. The user is then able to interact with the metering system more actively to have more control of their experiment.

Another webpage on the arboretum website is the scheduling page that is used for changing the frequency at which the data is refreshed in the database. On this page shown in diagram ZP, a user can do the following:

- Check which sensornodes are active
- Check the frequency the server is scheduled to refresh each active Sensornode's data
- Input and change the frequency the server refreshes each active Sensornode's data



arboretum-backend.soe.ucsc.edu:5000/schedulingpage

Welcome to the Sensornode Ping-Scheduling Page

Note: All Sensornodes are listed below, if you don't see one. Manually add it to the sensornodeclients-table in the database

Choose From This List of All Active Sensornodes

Name	Location	Port
sensornodetester	greenhouse1	1234
sensornodeforlife	greenhouse1	1237
thelastsamurai	greenhouse2	1235
sensornodemainpanel	greenhouse2	1236

Fields to Set a Sensornode's Ping Schedule

sensornodename ______ seconds ______ minutes ______ hours ______ days ______ months ______ submit

Current Ping Frequencies For All Unique Names (Note: Only Names that match client names in above table are active

Client Name	secs	mins	hours	days	months
sensornodetester	565	565	565	565	565
sensornodeforlife	543	0	0	0	0
thelastsamurai	341	341	341	341	341
sensornodemainpanel	999	0	0	0	0

Diagram ZP: This is the webpage for viewing all the active Arduinos that are tracked by the metering system, viewing the frequency at which the data is refreshed, and there is a field for allowing the user to alter the frequency at which data is refreshed from a sensornode.

The scheduling page can also be called the control center. It serves the user information about which Arduinos are part of the network, at which frequency they are being communicated with, and allows the user to vary the pinging frequency.

The first table at the top of the page presents the list of all active sensornodes which are the sensornodes that are part of the communication cycle in the backend. The first column is the unique *Name* of a sensornode which is manually programmed in the header filed *self_identification.h* as part of the setup requirements (as explained in the tutorial video *Sensornode Setup* and *Installation Section*). The name of the Sensornode must perfectly match the name on the Arduino for it to work. The reason for naming a Sensornode is to distinguish between sensornodes, log data appropriately, and introduce light security in the network.

Column number two in the top table is the *Location* of a sensornode. The Location attribute is solely to group the sensornodes together by greenhouses, sections, purpose, etc.

The third column is the Port number that is used to for a communication link between the server and an Arduino. The port number must be unique for each sensornode and must match the port number manually programmed in the source file *fona.c* to work (as explained in the tutorial video Sensornode Setup and Installation and Installation Section). The Arduinos were unable to share the same ports for reasons that were not understood during this project. If two Arduinos were put on the same port number, the server would crash due to an unsuccessful bind on the socket. The C-Library Socket.h may have functions that allow for a single port to be used for all the Arduinos. This method was chosen to get it working and served the goal of communicating with multiple Arduinos.

The frequency threshold has a lower bound of 5 minutes because the system focuses on one Arduino at any given moment. If two sensornodes are scheduled within 5 minutes of each other, the second Arduino will have to wait for the first to finish.

The other feature of the scheduling page are its ability to allow the user to alter the pinging frequency. The fields in between the top and bottom table are used to interact with the server. The first field is for the name of the Sensornode that the user wants to configure. This field must directly match the name in the top table of active sensornodes for it to work. The rest of the fields are for inserting the period of time in between each data retrieval. By inserting numbers into the seconds, minutes, hours, days, weeks, and months fields, the user is varying the amount of time the server will wait between each time it renews the data table on the sensordata page for a given sensornode. The system responds within 2 minutes to user changes of the scheduling frequency. After notices a change in the scheduling frequency for a sensornode it will reset its waiting timer over with the new frequency for that sensornode. To implement the changes, the user must click submit and then they are routed to a new page called pingsubmitsuccess. This page displays the fields the user had just filled out for accuracy purposes and because Javascript needs to post a response to the user's request to submit the fields. It uses a Node.js feature called *Routes* that are used for getting information from the user. Once the form is filled out, it is important to confirm that the changes were recorded by the system. The Ping Frequency table is where the changes are checked and the fields should be updated immediately upon clicking the submit button.

The last table at the bottom of the scheduling page displays the scheduling frequency for each sensornode. This is used for checking the scheduling frequency for each sensornode and double-checking that the changes were recorded when submitting a schedule change for a sensornode on this page.

To conclude, the monitoring website includes two webpages that allows the user to alter ping schedules, monitor all active sensors, confirm the presence of sensornodes in the system. The knowledge required to properly use the monitoring website were discussed in this section. Recall that the website is intended to provide an experimenter with the ability to remotely monitoring the horticulture system at the greenhouse for an effective harvest. Insight was provided to understand the timing constraints of the data-logger which are based on the server's ability to handle only one Arduino at any one time within a 3 minute period. There can be improvements in the attributes given to the sensors and sensornodes to group them strategically. It is also important to note that the website doesn't have an alert system which is crucial for a successful harvest. The problem is that the website doesn't notify the user when a sensor is malfunctioning or unplugged. The website will still log the unusable data and present it on the sensordata webpage. The objective is to solve this problem is find a method to handle a corrupt sensor. I propose that if a sensor has a reading value of zero then the user must be notified by email. I also recommend that the sensor readings are color coded to signify the state of each of the readings such as red for critical and green for typical values. For the future design improvements should consider the intent of the website to be the control center of the greenhouse system with the necessary information for monitoring and data analysis. The client intends to configure

4.3 Communication Hardware for Power Data

MCC = Morningstar Charge Controller or Tristar

BMS = Battery Management System

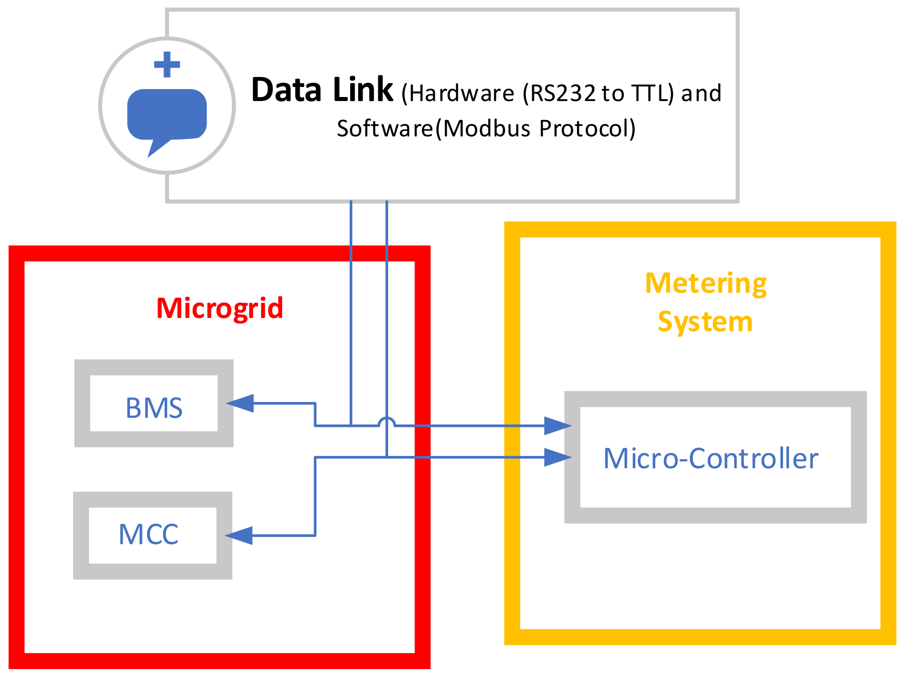

Data Link (Hardware (RS232 to TTL) and Software(Modbus Protocol)
Microgrid
BMS
MCC
Metering System
Micro-Controller

Diagram XX: The BMS and MCC hold information that the microcontroller can retrieve to send to the website. This diagram exposes the important components for power data retrieval. The data link (blue arrows) symbolize a hardware and software requirement to make a successful connection. The diagram shows the Microgrid (red) and Metering System (yellow) sub-blocks of the Greenhouse System, high-level, shown in Diagram X in the Introduction.

The acquisition and visualization of power-data is a key feature needed for regulating a microgrid which will allow for power-budgeting; it provides the ability to analyze solar power generation, energy storage levels, and the rate of energy consumption. The power feature is intended to give the user more control to improve the quality of research while protecting sensitive components, such as the battery bank, from negligent misuse. Since last year's team did not address the power task, it became a priority and key goal for this project. The following objectives were part of the greenhouse power-data goal:

1. Obtain power-data and identifying information about the micro-grid (battery temperature, battery state of charge, charge controller temperature, output/input power, daily battery voltage and others)
2. Convert this information to be comprehensible
3. Display it on a website for greenhouse users

It is possible to get power information in a few ways. Sensors are needed either way. These are the following ways: (1) placing voltage/current sensors on the grid, (2) retrieving the data from the power-controllers manually, and (3) purchasing a commercial data-logger. The most favorable choice is option number 2 because it saves money and allows the data to be seen on our custom website. The client desired a single location for all the data. Option 1 was a viable solution because we would have the raw data to display on the website. The con of option 1 is that more money must be spent and more code must be written. The more efficient and effective choice was grabbing the data from the source ourselves. Since there was a previous team who worked on the power-data efforts, their resources were utilized. It was a favorable choice to continue where the last team left-off because it would provide some initial leverage, but meant there would be start-up requirements to understand their work.

Nonetheless, the task of power-data was required and the existing code and hardware was scrapped for working parts.

To get power-data information it was important to understand the existing electrical system. The microgrid as designed by the last year's team featured two commercial regulation devices which store the data about the power in real-time and historically. There are two types of controllers called the Morningstar Charge Controller (MCC) and the Emus Battery Management System (EBMS). The MCC is a control and regulation device which is the head of electrical operations at the greenhouse. It acts as as a regulation and control device between the solar panels, batteries, and the loads. The MCC is multipurpose: it increases efficiency in charging the batteries, it actively protects the batteries, and it actuates devices. The MCC has valuable information about the electrical state of the micro-grid. The other device is the battery manager which controls how the batteries charge and discharge to supply power for the greenhouse. Each control-unit serves a different purpose to keep the microgrid healthy and operating properly. To learn more about the BMS and MCC, refer to the 2017 team report linked in the references.

The plan was to use an MCU to interface with the controller-units, the BMS and MCC. The Arduino has a UART which is a transceiver for communication applications. Since the Arduino and charge controller units are different in voltage requirements, a TTL to RS232 converter circuit was needed to link them together. Both the MCC and BMS share the same data-link requirements for proper communication, therefore a solution for one can be slightly changed to work for all the units. The data-link between the Arduino and the MCC was the focus for this project, but the goal of power-data acquisition was not successful as discussed later in this section. The experiences from this project are translated into useful documentation for future work on acquiring information about the microgrid at the S-Lab greenhouse.

The communication link between the power-controller and the microcontroller is analogous to using Facetime to get a dish-recipe from your mom. First you must establish a connection with your mom using a network (in this example, the *internet)*. Second, you and your mom must speak the same *language* to understand each other. Third, both of you must not speak over the other which means there are *rules*. The same is true with serial communication. To retrieve the information manually, there were the following requirements: Use RS-232 standard physical connection (analogous to linking to the internet) and the software Modbus Protocol standard (which provides language and rules). To fully understand the requirements of RS232 and Modbus Protocol, reference blank and blank. Communicating over computers requires many rules for a successful link. The takeaway is that serial communication must be understood for this portion of the project. The implementation of the link between the Arduino and the Morningstar Charge Controller is shown in diagram X5. The second block from the left, the intermediate block, is a MAX232 integrated circuit. This chip meets the constraints for adding an IC chip to the Arduino and the parameters for interfacing an MCU with the MCC.

- Needs 5V or less power supply
- 0 to 5V input voltage range
- -10 to +10v output voltage range
- Low Cost (less than 100$)
- Compact (smaller than 3 inches)
- Can handle a Baud Rate of 9600 or 104 micro-second signal frequency without causing distortion

The Arduino power supply is 5Volts which is the same as the MAX232 power requirement. The voltages being transmitted from the Arduino are in the range of 0 to 5V which are converted by the MAX232 to the range of -10V to +10V as required by the MCC. The MAX232 IC chip is also capable of handling two transceiver links. This meant that we need two chips rather than three for all the controller units to save scarce space on Sensornode.

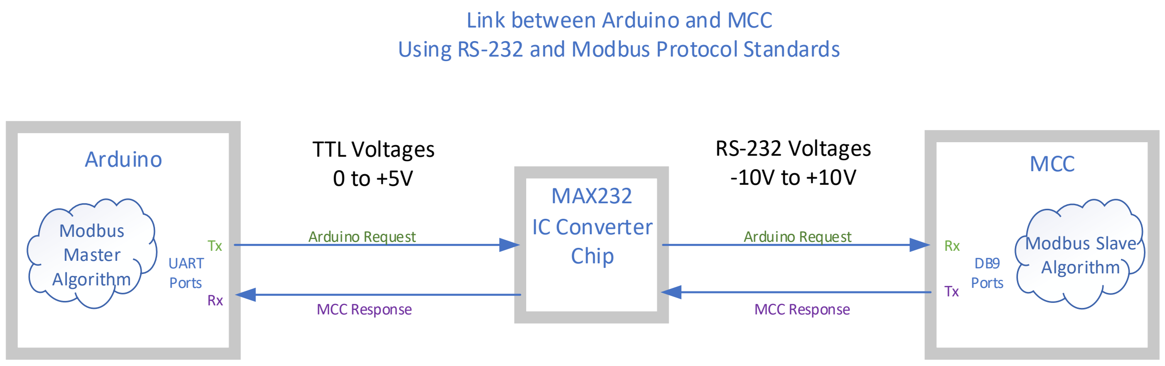


Diagram X5: The plan to get power data from the Morningstar Charge Controller is described by the blocks in this diagram. This diagram shows the hardware and software requirements for a successful link between the Arduino and the MCC.

As mentioned earlier, successful communication requires both a hardware link and a software link. First in the priority list is the hardware connection. The circuit for converting TTL to RS232 was designed with respect to the Maxim Integrated datasheet (datasheet in app notes). To verify this design, we had to run a series of tests.

- First, The RS232 to TTL circuit was tested by applying input voltages and confirming the intended output voltages according to the MAX232 datasheet. This was a sanity check to verify the chip was not defective.
- The next test was to see if the circuit performed properly with a bit rate of 9600 bits per second, specified by the MCC Modbus Protocol user-manual (in the references). The Arduino was programmed to transmit a bit vector at 9600 baud which was transmitted through the TTL to RS-232 converter. The *UART* was used to transmit the signal. The transmission signal of a group of characters was sent characters to the UART via uart1_putc (code is found in app notes). The signals observed at the input and output of the MAX232 chip were in accordance with the RS-232 standard. The frequency of the signal before and after the conversion was the same and the voltages were converted from TTL to RS-232 standards. This confirmed that the physical communication link is operating.

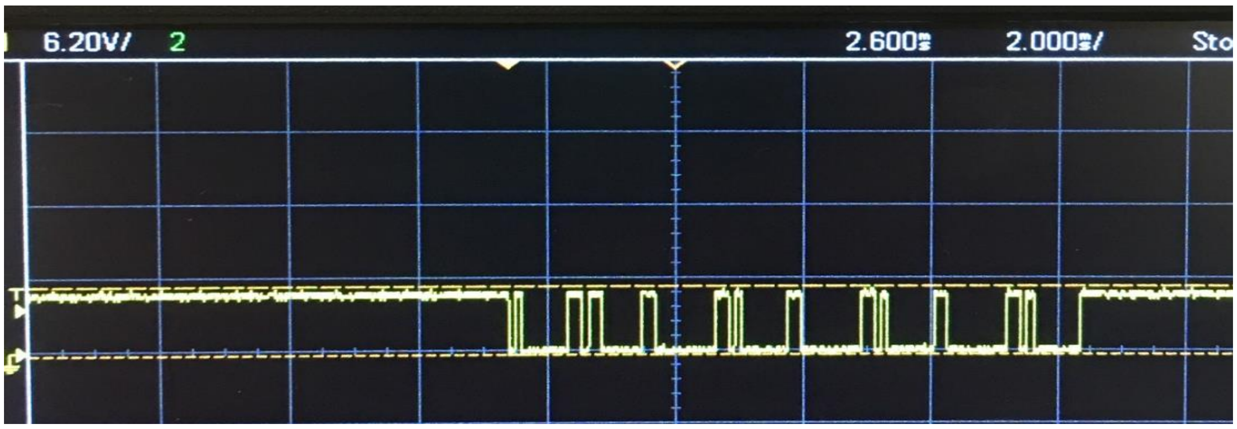


Diagram N: Shows an Arduino signal transmitted over UART1 (code found in the app notes) and sending the hex message [13000101] or in binary [0001 0011 0000 0000 0000 0001 0000 0001]. The TTL signal is in the range of 0 to +5V and its frequency is 9600 bits/second.

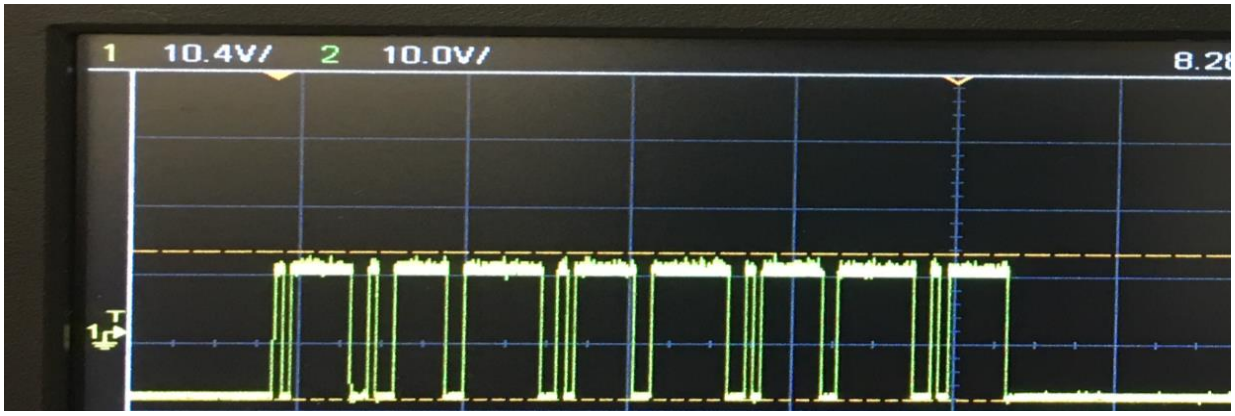


Diagram Z: Shows the signal from diagram N successfully converted to RS-232 standard voltages by the MAX232 chip. The voltages range from -7V (logical 0) to +7V (logical 1) and its frequency is 9600 bits/second.

Second in line of the priority list is the software to aid the interactive communication link. As specified by the MCC, the software protocol requirement is Modbus RTU. It is a protocol for a master and slave scenario. Modbus defines a set of parameters that describes language and rules for successful conversations which are listed below:

- Time Frame Durations - Baud Rate and hold-times
- Message Frame Format
- Function Codes
- Error Check - Cyclic Redundancy Check

The Modbus Protocol document was referenced to create a request message. For this experiment, refer to diagram X4 to see the test-bed used. Each message must follow the structure as seen in diagram U. Each package must have the following contents in the following order:

1. slave address also called Modbus ID
2. function call
3. data bits
4. cyclic redundancy check (CRC)

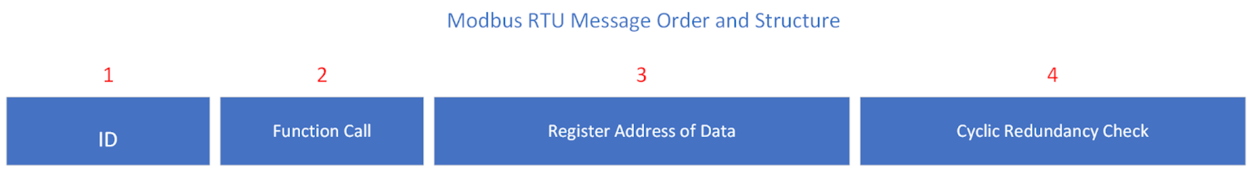


Diagram U: Shows the message format for transmitting by the modbus protocol standards needed to communicate with the Morningstar Charge Controller

The slave address also called the Modbus ID was configured in the *MSView* software (link to download in references). The function call needed was a read-command which has the value three according to the Modbus Protocol. The data bits in the packet refer to the location in memory that holds the power information. The location or address for the information of the microgrid is found in the MCC Modbus Protocol document found in the application notes. For this message, the voltage address 0x0001 was used. The last part of the Modbus transmission vector is the CRC or cyclic redundancy check. It is an error checking mechanism used for serial communication to prevent errors due to wire transmission vulnerabilities. This 16-bit error-check was hardcoded for this specific packet and calculated with an online calculator with the link found here. The other requirement is a 4-character silence in between transmission packets of information. This was calculated by using the duration of transmitting 8-bits and multiplying by 4. Duration of 1 character multiplied by 4. The list of requirements is the following: 9600 baud, no parity bits, 8 data bits, 1 or 2* stop bits, and no flow control, and 4-character long silence in between messages.

Daigram X4 Caption: The picture above is the experimental set-up that was used to simulate an on-site system. It was used to test and experiment with the communication link between the Arduino and the Morningstar Charge Controller (MCC). It is an identical set-up to the block diagram in Diagram X5 above.

The packet shown in diagram N was repeatedly sent from the Arduino to the MCC to muster a response. The MCC signal channel was being watched on an oscilloscope. The expected signal to be retrieved from the MCC should have been in the same format of the Modbus Protocol and should contain the voltage information in the packet. There is an error check. If the packet is sent with errors within it than the MCC is expected to send an error message abiding by the Modbus Protocol. Not a glimpse of any message or signal was observed on the oscilloscope as a response.

I believe that the connection didn't work because of the following reasons: flow control with the CTS and RTS pins are wired incorrectly in the circuit, there is a miswiring in the circuit due to misunderstanding of Modbus Protocol, or the Modbus software is not implemented correctly. The Modbus protocol was not fully grasped. The circuit and code were designed to the best of my knowledge and interpretation of the Modbus Protocol document. A promising method to check the connection requirements is to connect the MCC to a personal computer, observe the response/request signals from downloading the MCC configuration data, and then emulating the same signals in software. Note, I recommend that all pins are observed on the DB9 connector at the input of the MCC when a connection link is established. The notes on how to connect a PC to the MCC and trigger a conversation can be found in the reference folder in the Tri-Star (MCC) subsection.

To make it easier for the future research to continue, a list of recommendations and requirements are explained in the *Future Work Recommendations* list below/as a contribution to the power-data acquisition goal.

Future Work Recommendations

1. Stick with original Arduino and add a multiplexer circuit for increasing UART capacity, write modbus library C code or find a Modbus Protocol C Library and use that API
2. Use a different microcontroller (STMF32 because it has more UARTS and all the power data and sensors can coexist on one microcontroller unit). Find Modbus Open-source code or write Modbus API from scratch

3. Make a custom unit for the Power data. Use an Arduino with Arduino language (instead of C language on original sensornode) Arduino Open-Source Modbus Library solely for Power data.
4. Trash Sensornode Code and incorporate Open Source Arduino Modbus Library.
5. Get an RS232 compatible device to remove need for RS232 conversion and then use a program that extracts the data.
6. Tristar data-logger rather than microcontroller. Then, Interface with the data-logger to extract data and display on website. If data can't be extracted from data-logger, then use Tristar provided user interface to monitor microgrid.

I recommend a simpler solution using a commercial voltage/current sensor to achieve the client's goals of monitoring the power consumption/generation and protecting the batteries. Creating a custom power-data collection circuit will be faster and meet basic needs. The method would involve connecting voltage and current sensors strategically in the power network, logging the data with an Arduino or other microcontroller, and then in software, calculate the power metrics with the inputs. This would be a more appropriate approach for having a plug-n-play system that monitors and regulates the greenhouse. The only major consideration is product cost and measurement accuracy which are lenient. A quick and useful addition to this module would be a tiny LCD screen that displays the active power levels to prevent on-site misuse.

As a recap, the existence of power data analytics for the user helps them understand the status of the micro-grid's health and energy-levels. The client's needs were to have all data viewable on a remote computer. The plan was to interface a microcontroller with the power-regulation devices in the greenhouse which would then connect to the internet and send information to a website. The experiments conducted were inconclusive due to difficulties interfacing TTL with the RS232 standard and using the Modbus Protocol. To continue this work, the software for this module must be modified/rewritten with respect to the Modbus protocol and the RS232 standard must be confirmed. The experimental test-bed and circuit is set-up for further testing and experimentation between the Arduino and MCC. After studying the system in depth and understanding the system requirements I recommend attaching current sensors, voltage sensors, and others to an Arduino to monitor the electrical characteristics. This would be a more brute force method that is more promising and immediate than tackling the complexities of interfacing with commercial controller units. Another method would be to use an allocated Arduino with the open-source Arduino Modbus Library operating on it rather than writing a custom Modbus Library which is less reliable. In the future an Arduino should be used with the Arduino-Modbus Library to interface with the Tristar and Emus BMS.

5 Future Work

The Sustainability Lab's goal to remotely monitor the conditions at the greenhouse is now possible with the new metering system that modifications in the software, a creation of a website, and usage documentation. The product is in a beta state that needs more work to be fully scalable and reliable as a means for research. The metering system presents sensor data on a website for experiments by communicating with multiple microcontrollers whom log data about the greenhouse environment through a 2G wireless network. The ability to log information about the energy usage, storage, and generation presented difficulties that prevented the completion of power-data acquisition. Since power analytics were not achieved during this project, it is important that it gets high priority for the next project. There are suggestions and documents that will accelerate the completion of the power portion of this project found in this report and in the physical document binder for this project. In the future, the finalized metering system should have fully autonomous capabilities and should be designed in a modular, research friendly, and easy-to-use manner for its various applications. A complete product may include a tutorial video on how to use the product, an assembly video on how to build sensornodes from scratch, an in-greenhouse screen, data charts/graphs on the website, data about state-of-power at the greenhouse, alert-system to avoid critical conditions, and a self-regulation protocol to maintain a healthy plant environment and protect the electrical system.

List of next steps (priority high to low):

1. Ability to see amount of power available (consumed and generated) from the website
2. Adding Sensor History data to website
3. Ability to control the Arduino from the website (create, read, write, destroy commands), Suggestion: create a plug-in schema to allow the user to choose from a set of available pins that the Arduino tells the user to choose from)
4. Visualizing Sensor Readings vs time on a chart
5. User-Controlled Alert System for Critical Values (alerts via emails, color coding, text)
6. Optimized Hardware redesign with visible names on top layer of sensornode shield (consider MCU, removing CAN pins to name a few)

6 References

About the Sustainability Lab

https://slab.sites.ucsc.edu/about/

UCSC Campus Sustainability Plan Mission Statement 2017-2022, A letter by Chancellor George Blumenthal

https://sustainabilityplan.ucsc.edu/chancellors-letter/

UCSC Carbon Neutrality Initiative

https://ucop.edu/carbon-neutrality-initiative/index.html

Bonnie Reiss Carbon Neutrality Student Fellowship Program Email: StudentServices@ucop.edu

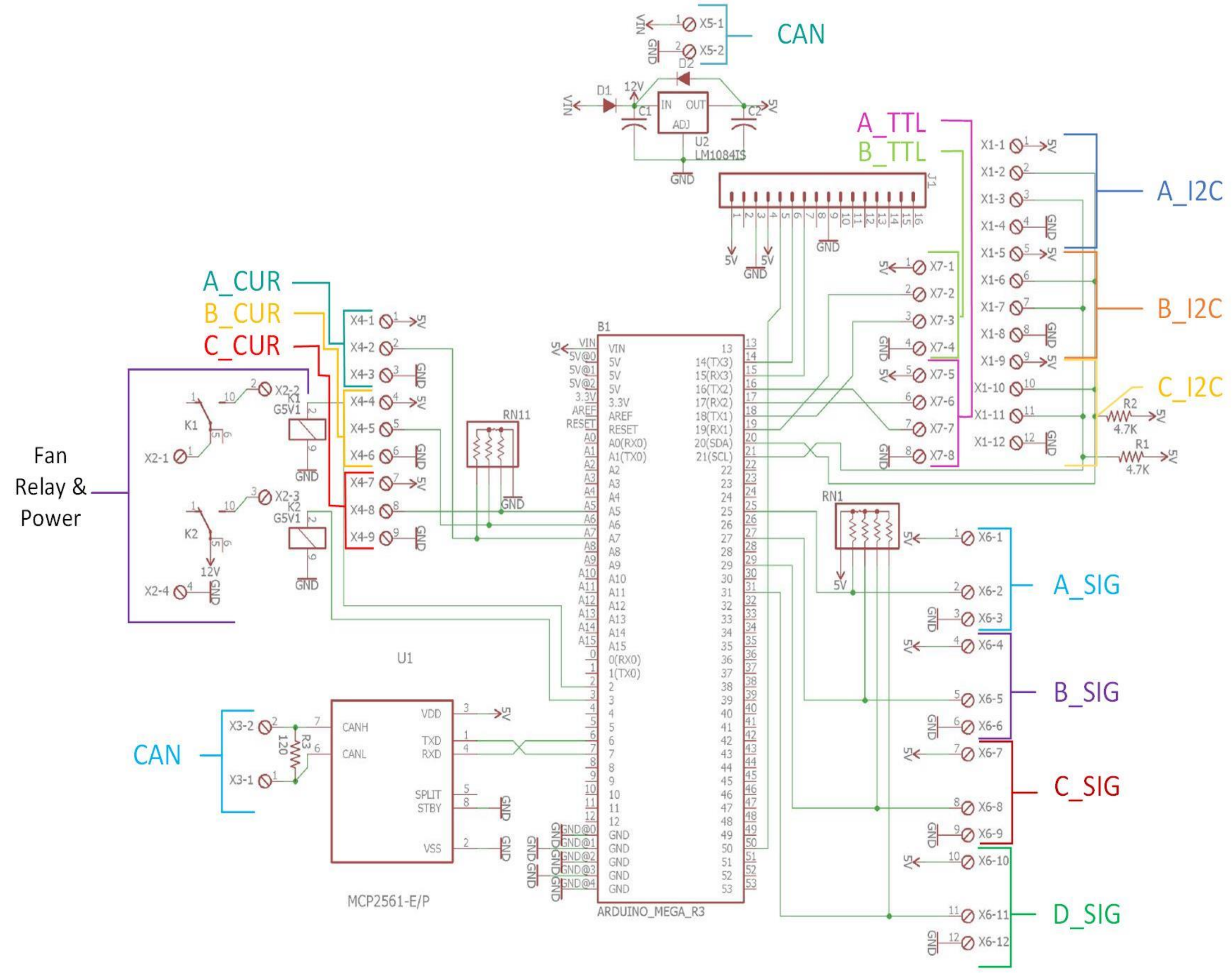
2017 Microgrid Project Final Report

7 Application Notes

7.1 Sensornode Tutorial Diagrams

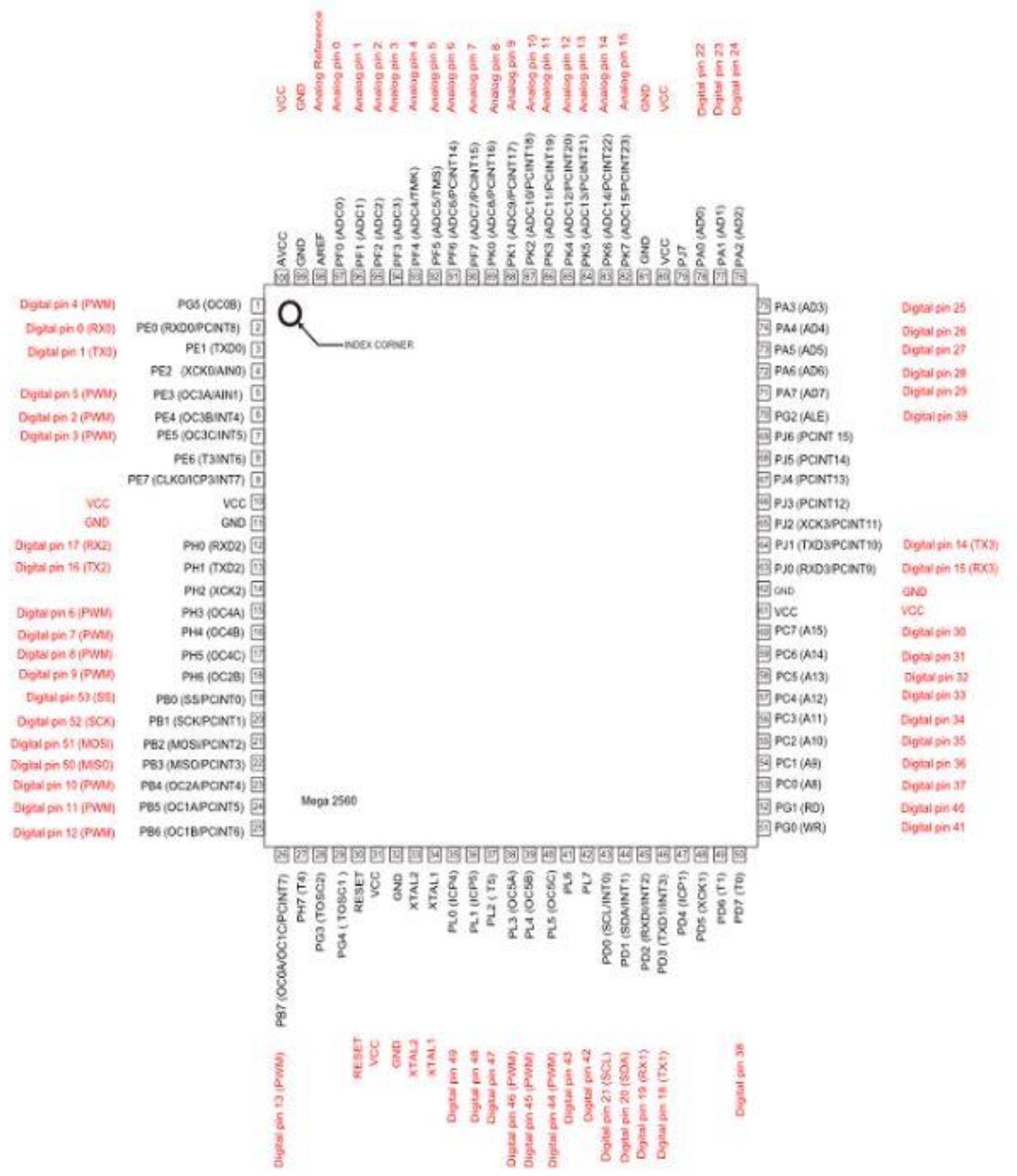
Pin Mapping Sensornode Shield Document

Sensornode Schematic for Pin Mapping to Physical Device



Pin Mapping for Arduino Atmega 2560

Arduino Mega 2560 PIN diagram



7.2 Communication Hardware for Power Data

MCC Modbus Document for Mapping Register Locations and definitions

TriStar MPPT 150V MODBUS Document

http://support.morningstarcorp.com/search/?document_section=&search_product=96

Morningstar Charge Controller

https://www.morningstarcorp.com/products/tristar-mppt/

List of Morningstar Charge Controller

http://support.morningstarcorp.com/search/?document_section=&search_product=96

MAX232 Document

https://pdfserv.maximintegrated.com/en/ds/MAX220-MAX249.pdf

CRC website

http://www4.ncsu.edu/~chou/course/Animations/Calculation%20of%20a%20CRC%20Checksum/crcinit.html

Modbus Protocol User-Manual

http://modbus.org/docs/PI_MBUS_300.pdf

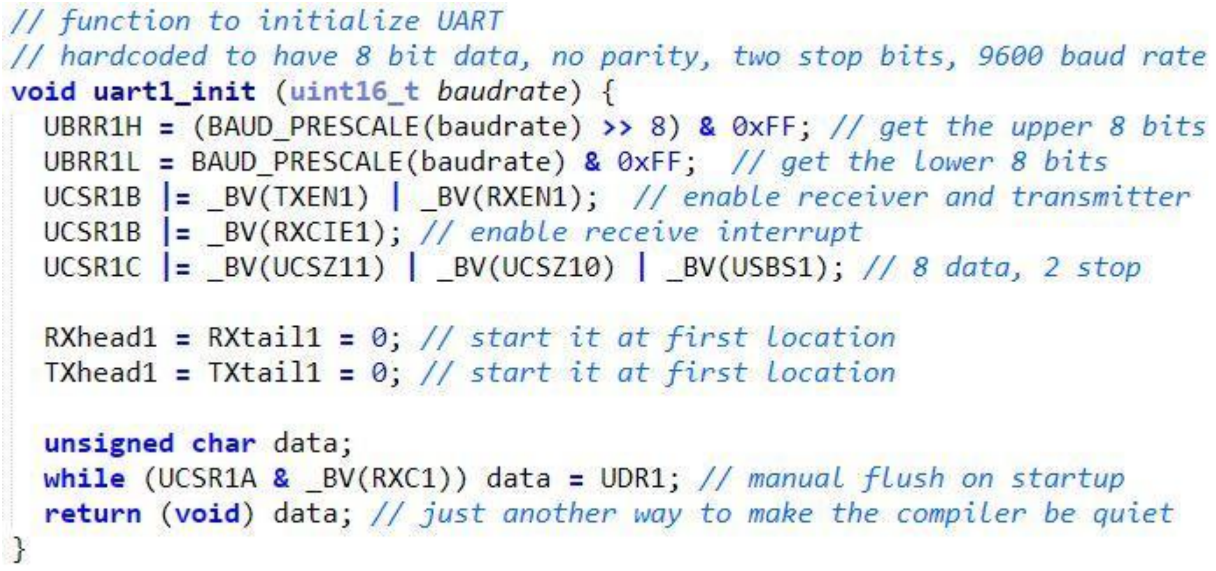
Modicon Modbus Protocol Reference Guide, June 1996 MODICON, Inc., Industrial Automation Systems One High Street North Andover, Massachusetts 01845

MS-View Program for connecting to Morningstar Charge Controller and other Morningstar Products

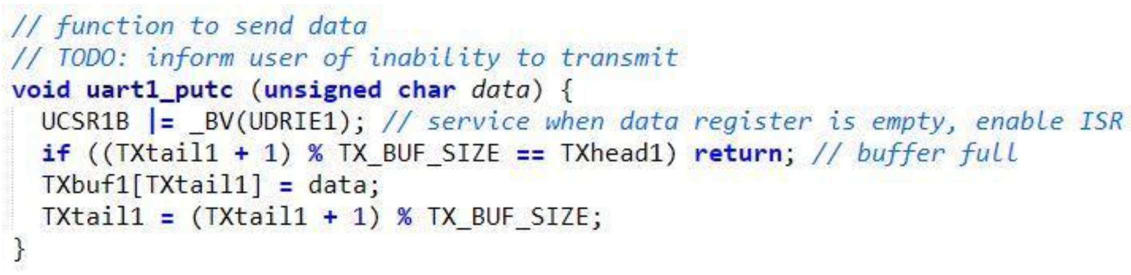
https://www.morningstarcorp.com/msview/

Code and Converter circuit for RS-232 and Modbus Protocol transmission signal

Uart.c/h

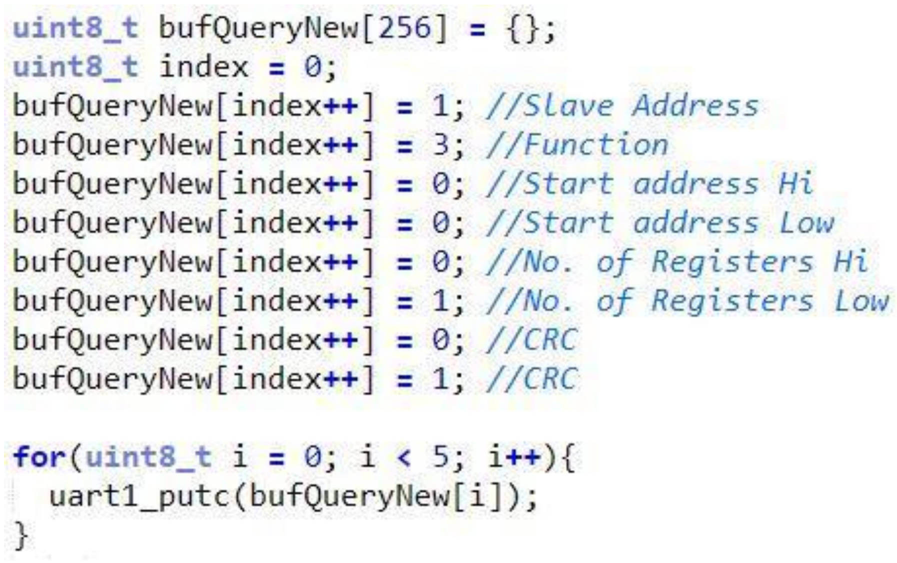


```c
// function to initialize UART
// hardcoded to have 8 bit data, no parity, two stop bits, 9600 baud rate
void uart1_init (uint16_t baudrate) {
  UBRR1H = (BAUD_PRESCALE(baudrate) >> 8) & 0xFF; // get the upper 8 bits
  UBRR1L = BAUD_PRESCALE(baudrate) & 0xFF;  // get the lower 8 bits
  UCSR1B |= _BV(TXEN1) | _BV(RXEN1);  // enable receiver and transmitter
  UCSR1B |= _BV(RXCIE1); // enable receive interrupt
  UCSR1C |= _BV(UCSZ11) | _BV(UCSZ10) | _BV(USBS1); // 8 data, 2 stop

  RXhead1 = RXtail1 = 0; // start it at first location
  TXhead1 = TXtail1 = 0; // start it at first location

  unsigned char data;
  while (UCSR1A & _BV(RXC1)) data = UDR1; // manual flush on startup
  return (void) data; // just another way to make the compiler be quiet
}
```

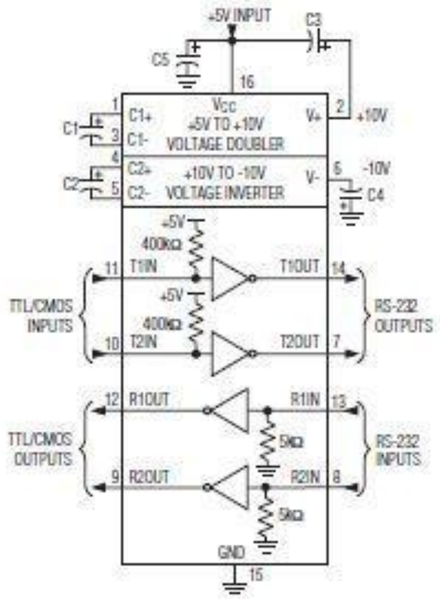


```c
// function to send data
// TODO: inform user of inability to transmit
void uart1_putc (unsigned char data) {
  UCSR1B |= _BV(UDRIE1); // service when data register is empty, enable ISR
  if ((TXtail1 + 1) % TX_BUF_SIZE == TXhead1) return; // buffer full
  TXbuf1[TXtail1] = data;
  TXtail1 = (TXtail1 + 1) % TX_BUF_SIZE;
}
```

Tsmppt.c/h

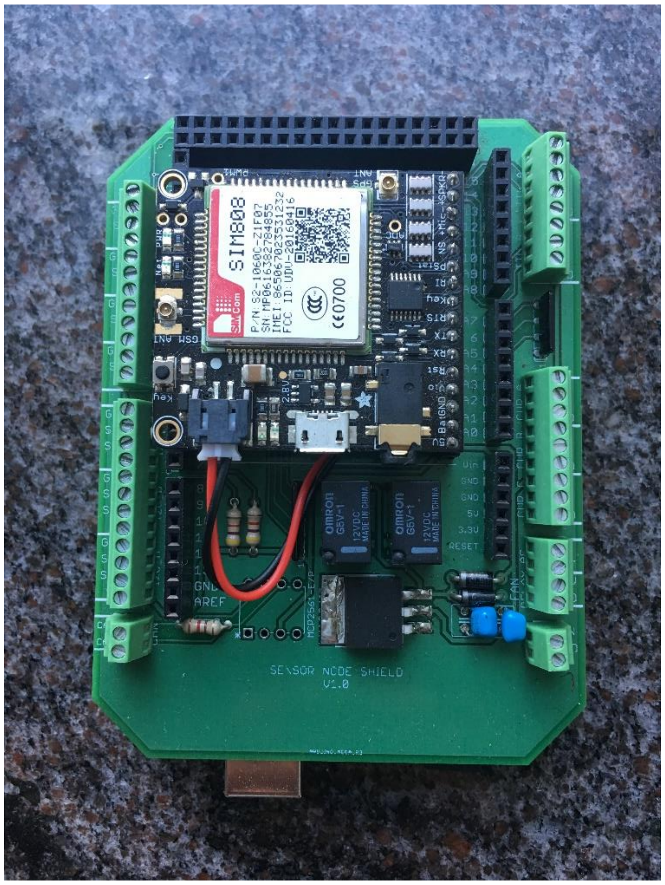


```
uint8_t bufQueryNew[256] = {};
uint8_t index = 0;
bufQueryNew[index++] = 1; //Slave Address
bufQueryNew[index++] = 3; //Function
bufQueryNew[index++] = 0; //Start address Hi
bufQueryNew[index++] = 0; //Start address Low
bufQueryNew[index++] = 0; //No. of Registers Hi
bufQueryNew[index++] = 1; //No. of Registers Low
bufQueryNew[index++] = 0; //CRC
bufQueryNew[index++] = 1; //CRC

for(uint8_t i = 0; i < 5; i++){
  uart1_putc(bufQueryNew[i]);
}
```

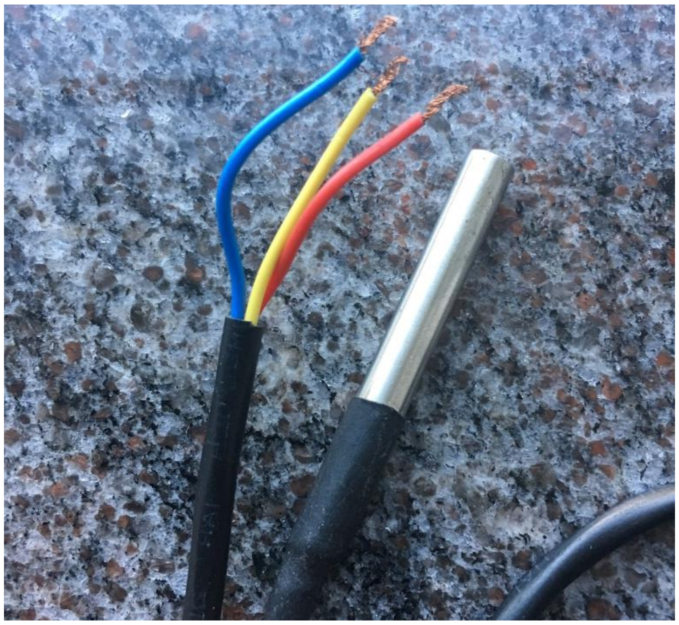


Converter Circuit for RS232 to TTL (Max232)

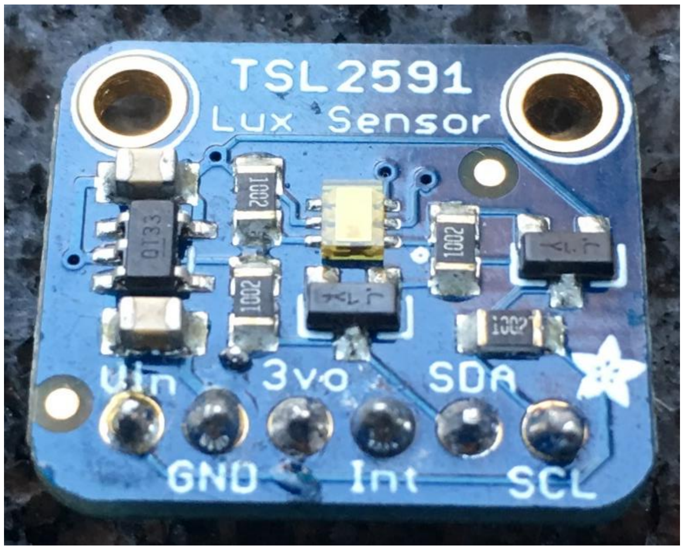
7.3 Pictures of Devices



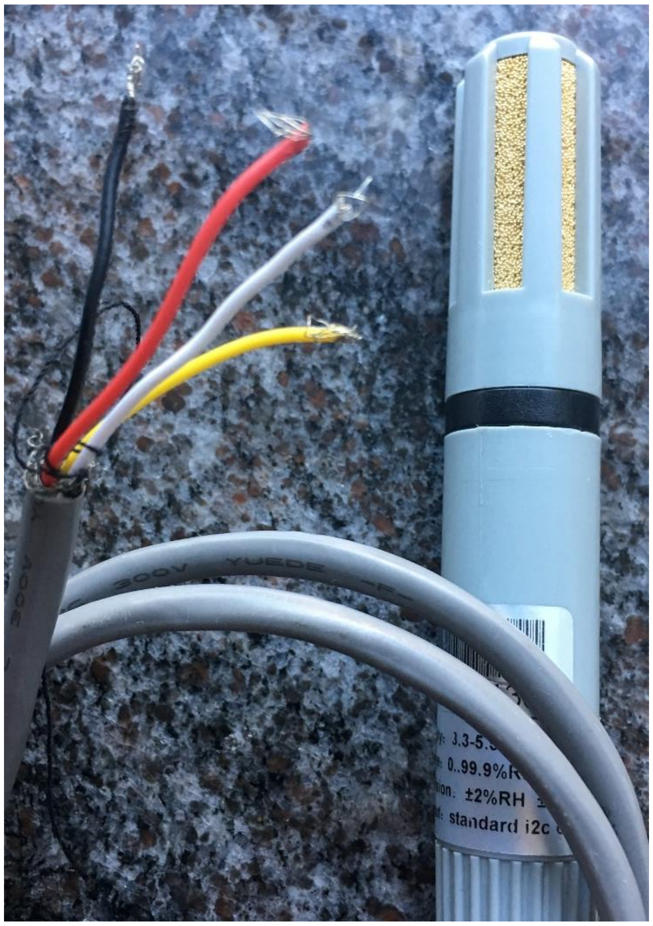
Sensornode Metering Device (Arduino AtMega2560, See section 3 for more details)



Temperature Sensor (One Wire (SPI))



Lux (Light) Sensor (I2C) TSL2591



Humidity Sensor (I2C) AM2315

7.4 Parts List

Name	Comment	Link
Arduino Shield PCB	Bay Area Circuits	https://stoı
FONA 808	Cellular Module for Arduino	https://ww
SMA to uFL	Cellular Connection Part for FONA	https://ww
SMA Antenna	Cellular Connection Part for FONA	https://ww
2G SIM Card	Ting Subscription	https://ww
Ting Subscription	To connect to 2G wireless cellular	https://ting
500 mAh Lipoly Battery	For Current Spikes with FONA Transmission	https://ww
Arduino AtMega 2560	Microcontroller Unit	https://ww
DS18B20 Temperature Sensor	Sensor	https://ww
AM2315 Humidity Sensor	Sensor	https://ww
TSL2591 Light Sensor	Sensor	https://ww
5V Regulator	For Arduino Power Source	http://wwv
Relay	Switching On/Off Loads	http://wwv
4.7k Resistor Network	for I2C/Onewire	http://wwv
10k Resistor Network	for I2C/Onewire	http://wwv
Capacitor	for Arduino Power Source	http://wwv
Diode	for Arduino Power Source	http://wwv
2 Port Screw Terminal	Connection Ports on PCB Shield	https://ww
4 Port Screw Terminal	Connection Ports on PCB Shield	https://ww
8 Port Screw Terminal	Connection Ports on PCB Shield	https://ww
9 Port Screw Terminal	Connection Ports on PCB Shield	https://ww
12 Port Screw Terminal	Connection Ports on PCB Shield	https://ww
Male DB9 Breakout Board	TSMPPT to Arduino Interface Research	https://ww
Female DB9 Breakout Board	TSMPPT to Arduino Interface Research	https://ww
DB9 Extension Cord	TSMPPT to Arduino Interface Research	https://www
DB9 to USB-A	TSMPPT to Arduino Interface Research	https://ww
USB-A to USB-B	TSMPPT to Arduino Interface Research	https://ww
UCSC Server Space on Unix Timeshare		
Ting Subscription for 2G Cellular Connection to Sensornode		
UCSC MySQL Database via PhpMyAdmin		