Spring 2020

Creature Care

Alexander Weber
ajw138@zips.uakron.edu

Creature Cage

Midterm Design Report

DT17

Cassandra Allender

Shane Benner

Alex Weber

Kim Wilson

Faculty Advisor: Okan Boler

04/23/20

Table of Contents

List of Figures

List of Tables

1. Abstract (A.W.)

Caring for a pet is a formidable responsibility that requires intense care and monitoring in order to protect and prolong the life of the animal. Traditionally, caring for a reptile presents a unique challenge for pet owners, who not only need to care for the animal but for the environment as well. Individual species distinctly require precise monitoring and control of environmental parameters such as temperature, humidity, and air quality; and deviation from the nominal values often leads to serious health problems for the reptile. To simplify reptile ownership, a system was designed that measures the critical parameters of a reptile enclosure and automatically regulates the local environment, notifying the user of any issues that require manual intervention. The system provides live and historical data, accessible on the Web via any computer or mobile device, and is capable of alerting the user by email or text message when any problems occur within the system. Additional reminders for feeding and cleaning are also configurable based on the requirements of the species. This design gives peace of mind for the user, ensuring that the animal within the enclosure is being provided the most ideal local environment unique to the species.

2. Problem Statement

2.1. Need (C.A.)

Owning a reptile comes with great responsibility. To keep the animal healthy, many factors must be taken into consideration. For example, a reptile enclosure must be kept at within a certain temperature and humidity. These can be difficult to monitor and regulate, especially for a reptile owner that is away for long periods of time during the day. This can lead to the accidental mistreatment of an animal. The creature cage allows for the monitoring and visualization of humidity, temperature, and air quality of the tank. These factors can be controlled and regulated from any location through an application on a smart device. The creature cage supports various animal species.

2.2. Objective (A.W.)

The objective of this project is to design an intelligent reptile enclosure that will monitor and control the environment according to the needs of the animal. The enclosure will feature a central microprocessor that will be connected to various peripherals in order to facilitate the collection of environmental data and the regulation of the parameters in response to changes in the local environment and as prescribed by the needs of the animal. The microcontroller will use a standardized format for the environmental parameters required by the species, and the structure will be expandable to allow for additional species to easily be integrated into the existing design. The microcontroller will be connected to the Internet in order to send and receive data to facilitate both the monitoring and visualization of the environment as well as the control of peripherals as necessary.

2.3. Background

2.3.1. General Reptile Care (C.A.)

Owning a reptile comes with the responsibility of being knowledgeable about the natural habitat of the reptile in order to simulate the environment in the enclosure it is housed in. The main factors of its environment that need to be taken into consideration are heat, light, and humidity. Failing to have the correct balance of these factors can be detrimental to the health of the animal. According to the Institute for Laboratory Animal Research (ILAR), reptiles are ectotherms, meaning they do not generate enough metabolic heat to raise their body temperatures above ambient temperature. Their regulated temperature may change in response to internal and external conditions which includes daytime and nighttime conditions, elevated body temperature after feeding, pregnancy, behavioral fever during bacterial infection, and thermogenesis during egg brooding. The conditions must facilitate their thermoregulatory responses [1].

2.3.2. Effects of Outside Heat and Humidity (S.B.)

As previously stated, reptiles require a specific outside temperature to keep their body temperature within a specific range. There is a small window of that temperature range that is called the tolerable range. If the temperature is too far out of this tolerable range, it will cause stress and eventually death to the animal. In nature, reptiles have naturally have the ability to adapt and find a way to remain in their preferred optimum temperature range. In captivity, an animal's options are greatly reduced within the confinements of an often too small enclosure. This

enclosure can still be set up without proper consideration for the animal's necessary optimum temperature range. There are ample signs showing when an animal in an enclosure is outside of this temperature range. Many reptiles live in climates where the temperature and humidity does not fluctuate much. Therefore, the enclosure environment needs to be similar to this. If these humidity and temperature levels fluctuate too much, it can be unhealthy for the animal. This results in the animal not being able to shed properly or cause fungus and bacteria to grow on them. The size of the pool of water that is in the enclosure can change the ambient humidity [4]. People own reptiles in different locations around the world. Some of these locations have very high fluctuating temperatures due to the seasons. This needs to be taken into account for when the ambient temperature outside of the enclosure begins to change. During these temperature fluctuations, the enclosure should be measured in three locations: basking areas, warm areas, and cool areas. One thing reptile owners need to consider is that when they are home, the temperature might remain constant because of heating and ventilation systems that are left on when the owner is home, but that may not be the case when the owner is away. These seasonal fluctuations will also affect the humidity within the house as well as inside the reptile enclosure, so it is important that the enclosure will adjust to these changes [5].

2.3.3. Existing Designs (C.A.)

Creating an environment that can be automatically regulated will increase the ease of owning a reptile and decrease the chances of accidental mistreatment of the animal. A similar product exists: the Comprehensive Reptilian Environment Control

System. This design is a control system made as an addition to a reptile enclosure, rather than the reptile enclosure itself. It keeps the enclosure within desired heat and humidity ranges, with the heat and humidity sources included. The user sets the desired temperature and humidity settings with dials and the control system maintains the set conditions [2]. Another similar product is the Reptile Cage Apparatus. It is a transparent enclosure with a thermostatically controlled resistance heater, a spaced infrared light source, both mounted in association with a thermostat to regulate the temperature in the enclosure. It has a humidifier in association with a humidistat to regulate the humidity [3]. The smart-tank will include a UV light and heat lamp, automatic humidifier, motor-controlled food storage door, humidity sensors, heat sensors, ammonia sensors, live video capability, and the circuitry involved in connecting the components. The device will be networked and it will have the functionality to send the user notifications via web hooks (i.e. for SMS or email).

2.3.4. Environmental Sensors (A.W.)

Several comprehensive solutions for simultaneous monitoring and control of various scales of environments exist. Fundamentally, such systems are very similar in nature, focusing primarily on reactive control of the environment based on the measured values of one or more parameters. For a reptile enclosure, it is imperative to maintain a temperature, humidity, and air quality within a range dictated by the traditional climate of the species contained therein. Such an enclosure would need to react quickly and accurately to changes in the local environment that disrupt the

ideal parameters for sustaining the animal, while also maintaining a clean, stable, and hospitable climate.

Accurate and precise measurements and control is therefore of utmost importance with respect to maintaining and adjusting the enclosure environment. Quick reaction to parameters outside of the acceptable range is greatly emphasized as it affects the health of the reptile. A network-based device for sensing and controlling several of these parameters currently exists, which includes reading in values constantly, calculating statistics based on the values, and performing calibrations and adjustments in order to maximize the precision of the climate control [7]. This device focuses on real-time data transmission and reactive control of the environment based on the constant stream of data. Additionally, devices and methods for regulating the environment for aquatic, semi-aquatic, and terrestrial organisms can vary widely based on the needs of the consumer, so allowing for a flexible and scalable is important to maximize the usefulness and range of application of such an enclosure [8].

One central system responsible for both monitoring and controlling relevant parameters is ideal because it reduces the complexity and cost of environmental control. A multi-sensor system allows for relatively simple expansion, as additional sensors for parameters such as ambient light or air quality can be installed without requiring significant modification to the existing design [8]. Further, the use of a central controller easily facilitates the possibility for automatic latent calibration, enabling the system to constantly enhance the precision of measurement and the resulting control [8].

Sensors for temperature, humidity and air quality can vary widely in how data about the environment is collected and transmitted. For example, some devices will give relative readings in response to changes in the environment, whereas others may give an absolute reading. Also, such sensors can communicate via a common protocol such as I2C, SPI, or CAN, while others may provide an analog output of the measure. Given the low cost and high availability of the former, digital devices can provide a simpler and more useful measure of the local environment. Additionally, such digital devices allow for several sensors to use the same communication bus, simplifying design and reducing the complexity and cost of the system. This also allows for multiple sensors of the same type to be used within the enclosure, enabling more accurate measurements by strategically placing the devices throughout the enclosure to ensure a uniformly stable environment [9].

2.4. Marketing Requirements (A.W., S.B., C.A.)

1. The system will focus primarily on the safety and well-being of the animal.
2. The system will require minimal manual human intervention.
3. The system will be scalable with other systems on the same network.
4. The system will provide essential metrics for visualization of the environment.
5. The system will be affordable to consumers.
6. The system will be able to operate in any household environment.

2.5. Objective Tree (C.A.)

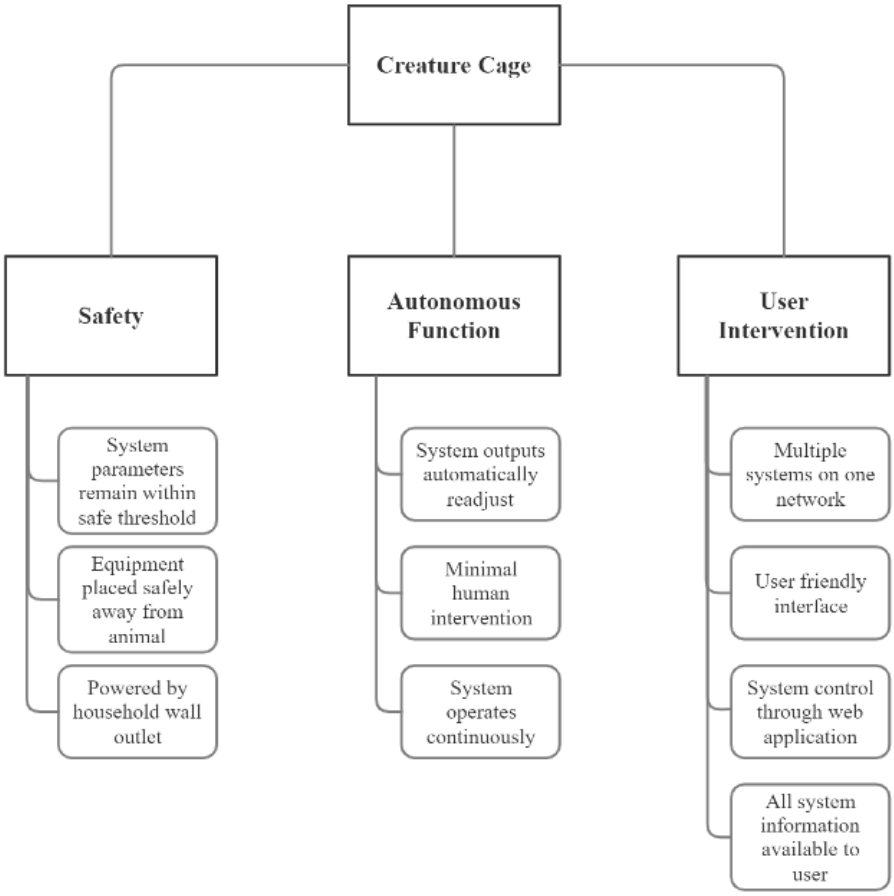

Figure 1: Objective Tree

3. Engineering Analysis

3.1. Circuits (C.A.)

The hardware design must provide power to multiple sensors measuring temperature, humidity, and air quality, a microcontroller, a liquid crystal display, micro-switches on the tank lid, an alarm, and the necessary equipment needed for maintaining a reptile's environment including ventilation fans (one bringing air in and one bringing air out) and a night and day heat lamp. The animal owner will provide a water source that will provide the necessary humidity to the environment.

The microcontroller will be used for communication with the web application and communicating with the sensors. It will send the data collected from the sensors to the web application as well as a liquid crystal display. The data collected from the sensors will include the temperature, humidity, air quality, and lid status of the enclosure.

The microcontroller will also be used for the sensor feedback. It will control the night and day heat lamp, the buzzer, and the ventilation fans. More specifically, the heat lamp will remain on at all times, but will switch from the daylight bulb and nighttime bulb based on the time of day. The ventilation fans will turn on and off if temperature, humidity, or air quality passes their max value set points. A piezo buzzer will sound if the enclosure lid is not secured, meaning all values of the micro-switches are not equal to zero.

The use of several voltage values will be used for different devices of the design. The heat lamp will require 120VAC. The ventilation fans, liquid crystal display, environmental sensors, micro-switches, and microcontroller will all require 3.3V. The liquid crystal display and microcontrollers will require 5V. An AC to DC converter is

used to convert the AC voltage from a wall outlet to a DC voltage that can be used for the other components with a voltage regulator that drops the voltage to 3.3V. A power relay with an opto-isolator allows for switching between the daylight bulb and nighttime bulb within the heat lamp.

3.2. Electronics (S.B. & C.A.)

This project is composed of the following electronics: microcontroller, environmental control devices, sensors in the environment, and a display to the user. These electronics will be used to gather the necessary conditions for the environment, apply these necessary changes to the environment and actively monitor the environment sending this information to the user.

The microcontroller used is the Tiva C Series Launchpad, manufactured by Texas Instruments. The processors on them is the TM4C123GH6PM [10]. The microcontroller will be used for the following:

- Taking the environment conditions gathered from the web and the current environmental states, and calculating the necessary changes needed for the environment gathered from the sensors in the environment in order to meet the system conditions.
- Connecting to the internet to perform web-scraping for the animal species given from the user, which will take the environment condition information found for the specific animal species and send it to the first controller to make the decision on the change

necessary for the environmental states to meet the system conditions.

- Controlling the signal processing and outputting the information in a way that is perceivable to the user (i.e. visual display and audio cues).

The next electronic components needed for this design are environmental control devices. These devices would consist of a day and night heat lamp and ventilation fans. Each of these would be able to manipulate the environment in such a way that the system states of the environment would be the same as the system conditions gathered and processed from the controllers. (e.g. if temperature is too high, the ventilation fans will turn on until the desired temperature set point is met).

The next electronic components needed for this design is the environmental sensors. These would consist of the BME680 that measures temperature, humidity, air quality of the environment, and mini pushbutton switches placed around the enclosure lid connected to a digital piezo buzzer that alerts users when the enclosure lid is not secured. This will consist of switches and sensors communicating information via serial communication or sensors communicating information based on the change of voltage outputs to determine a specific parameter of the environment. This will send the information back to the controller to calculate if the environment states match the environment conditions.

3.3. Signal Processing (K.W.)

Processing the data will require scaling the raw data and switching set points set by the particular levels set by the webserver. The hysteresis should be decided with specific consideration of the environmental variables being controlled. The gas sensor values are humidity and temperature dependent so a correction can be calculated in software and applied if there are changes in value due to environmental conditions and such a correction can be estimated from engineering data on the device datasheet or determined experimentally if needed.

3.4. Communications (A.W.)

The design of the system heavily relies on digital communications in order to maintain proper function. The system will contain a number of devices, communicating over different protocols in order to both read data about the local environment and send control data to other subsystems responsible for environmental regulation. Given the estimated volumetric constraints of the system, there should not be any sensors located significantly more than one meter away from the central unit. Because of this, the maximum cable length for any sensors placed outside of the main control unit can be assumed to be less than a meter. Additionally, the majority of the sensor devices will be using the I2C or One-Wire communication protocols, so a single device cable should not need to exceed four wires, including device power.

The I2C device protocol specification consists of two signals: serial data (SDA) and serial clock (SCL) [11]. In addition to these two lines for communication, each of the devices will have a voltage in (VDD) and ground (GND) connection, resulting in a total

of four wires to each device. The wiring must be designed such that crosstalk on the lines is minimized, and shielding may be necessary because of the relatively high impedance of pull-up devices [12]. If such shielding is employed, the shielded cable must have a low coupling capacitance between the data and power lines, as the data line is most susceptible to crosstalk [12]. As each of the devices will be utilizing bidirectional communication on the I2C bus, one of the four bidirectional data rate specifications will need to be used. Standard-mode (Sm), Fast-mode (Fm), Fast-mode Plus (Fm+), and High-speed mode (Hs-mode) allow for bit rates of up to 100 kbit/s, 400 kbit/s, 1 Mbit/s, and 3.4 Mbit/s, respectively [12]. Lastly, the devices will need to be selected such that the addressing mode can allow for the required number of devices on the I2C bus. The I2C specification allows for both 7-bit and 10-bit addressing mode, theoretically allowing for up to 128 or 1024 devices on one bus, respectively [12]. However, many devices to not support this full range of addresses, and as such, the devices will need to be chosen so that address conflicts can be avoided.

One-Wire communication, as the name suggests, uses only one wire for data communication. This single data line is a bidirectional data line with an open-drain output and a weak pullup on the line [13]. Additionally, the specification requires a timing circuit capable of reliably producing a 1 µs delay at the standard speed, or a 0.25 µs delay if the device is in overdrive mode [13]. This protocol does not allow for multiple devices on one bus, so no addressing is necessary in order to communicate with such devices.

3.5. Electromechanics (S.B. & C.A.)

This project will consist of a few electromechanical devices, although the main goal of the design does not necessarily require any electromechanical components. These components will be used as environment control devices. Specifically these devices will be used for air quality control and temperature control.

One of the electromechanical control devices that will be used is a fan, more specifically two fans. One of the fans will be used to assist in air quality control in the environment (i.e. when the air quality of the environment is passed a set threshold, a fan will turn on to blow out the toxic air at a more rapid rate then the natural airflow in the environment). The other fan will be used to assist in temperature control in the environment (i.e. if the environment gets too hot, a fan will blow in air from outside of the environment to cool the environment).

3.6. Computer Networks (A.W.)

As per the marketing requirements, the system is to be designed such that the user can place it anywhere without requiring any cables other than to power the system. As such, the networking of the system should be designed wirelessly to avoid any location restrictions. The system will use WiFi communication to connect to the user's wireless network and send and receive data over the network. The network impact should be relatively low and unobtrusive, with the standard metrics logging being responsible 1MB of the user's network bandwidth with daily use. To achieve this, a lightweight messaging protocol must be employed to communicate with the central database.

In order to provide networking capabilities to the system, the ESP8266 module will be utilized, allowing for 802.11 b/g/n wireless communication according to the WiFi Alliance specification. The module supports up to 72.2 Mbps at 2.4 GHz using 802.11n, so the module will be more than capable to support the infrequent and relatively lightweight transmissions of the system [14]. Additionally, the module supports several low-power modes with a quiescent current as low as 20 μA while the device is in "Deep-sleep" [14]. This mode is recommended for data logging applications with infrequent transmissions, on the order of minutes, and uses less than 1 mA of current overall on average. The module will be able to support much higher data rates than the system is designed to use, extending the scalability of the system.

In order to send and receive data over the network, an application-layer protocol must be selected to control the flow of data. The Message Queueing Telemetry Transport (MQTT) protocol is an ISO standard publisher-subscriber communication protocol that works within the TCP/IP layers. MQTT is a lightweight protocol that is designed for remote systems with low bandwidth requirements, and the communication is handled by a broker that can lie outside the local network. In this case, the system would consist of a single publisher that will send the data to the broker on an external server, and a subscriber on the same server will process the data. Maintaining this data processing outside of the user's network simplifies and reduces the network footprint of the system while also allowing for many different devices to communicate with the same server. The individual MQTT packets consist of only a few bytes of header information, and different modes are available to increase or decrease the quality of service (QoS) in order to further control the bandwidth required [15]. The payload is simply byte-encoded

data, so each message can be as simple or as detailed as desired, given the constraints of the system. Additionally, the MQTT protocol supports the Advanced Encryption Standard (AES), the most widely-used encryption algorithm, though not all embedded processors support the feature [15].

3.7. Embedded Systems (K.W.)

The embedded system will require a unique id. The parameters of its operation and its operating program all need a novitile memory space as yet undefined in size. The interface with the wifi device will output the system variables. Indicators prompt necessary user interaction. The wifi interface will also communicate with the web server to update the environmental values and receive updates for setpoints and user interaction. The wifi chip has a variety of interface options and I2C is among them.The use of I2C interface which uses a clock and a bidirectional data line to communicate over an approximately two meter distance at a baud rate of 100k or 400k will be the choice for sensors and wifi communication. The interface uses a master slave relationship with pull up resistors on both lines. The devices interfaced need a unique ID.The one wire interface is also a very good choice with lower data rates and longer runs similar to I2C. Both interfaces exceed the system requirements of thermal equilibrium time constants and allow multiple devices to be accessed with bidirectional communication over the same data line in a serial manner allowing a minimal hardware overhead for multiple devices. Temperature humidity and combined sensors are available. The gas sensor is a resistive device and can be accessed with an A/D

converter, RC with timer, or a level comparator. The data rates and processing time will be small compared to the thermal environmental rates of change.

Interfacing the controlled items like fans and heat lamps will use a dedicated data line for each device to switch heat lamp, fan, and humidity control in as on off control.

4. Engineering Requirements Specification (C.A. & A.W.)

Table 1: Engineering requirements specifications

Marketing Requirement	Engineering Requirement.	Justification
1, 2	Maintain the temperature within 3 degrees Celsius of the species' environmental constraints within 10 minutes.	Safe temperature and low fluctuation in order to maintain a healthy body temperature of the animal.
1, 2	Maintain the relative humidity within 10% of the species' environmental constraints within 10 minutes.	Safe humidity and low fluctuation in order to maintain a healthy body temperature of the animal.
1	Maintain safe levels of unsafe gases under 20%.	Keep the tank clean and avoid toxicity.
1, 4	Monitor enclosure opening's security.	Alert user if enclosure lid is open.
4	Display will provide general overview of the status of the system as well as current environmental data.	Quick environment status without use of web application.
1, 2	The user will be able to manually adjust environmental limits.	Ability to make environmental changes if need be.
3	Assign a QR code to each enclosure in use.	Easy way for the user to maintain multiple enclosures.
5, 6	The system will be able to control a 100W, 120V heating element.	The average sized heating element used for reptiles.

5. Engineering Standards Specification (C.A.)

Table 2: Engineering standards specifications

	Standard	Use
Safety	OSHA 1910.303	Electrical safety standard followed to keep user and animal safe in or near the enclosure.
Communications	IEEE 802.11	Wi-Fi is used for communication between the enclosure and web application.
Data Formats	PNG	Used for documentation figures.
Programming Language	C Python	C is used for programming the microcontrollers. Python is used for seeing messages in the MQTT channels.
Connector Standards	USB NEMA 5-15	USB is used for connection to the microcontroller. NEMA 5-15 is the power connection to the system.

6. Accepted Technical Design

6.1. Hardware Design (S.B., C.A., A.W.)

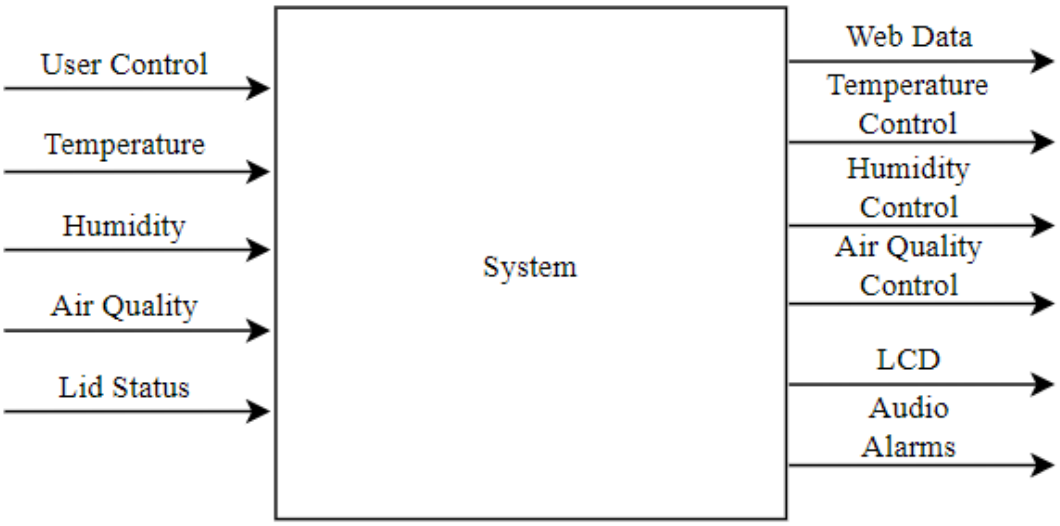

Figure 2: Level 0 Hardware Diagram

Table 3: System Module, Level 0

Module	System
Designer	Shane Benner, Cassie Allender
Inputs	User Control, Temperature, Humidity, Air Quality, Lid Status
Outputs	Web Data, Temperature Control, Display, Audio Alarms, Humidity Control, Air Quality Control, LCD Display
Description	Based on the animal species and the user input, the environment control system will change to the specifications of the species and the user input.

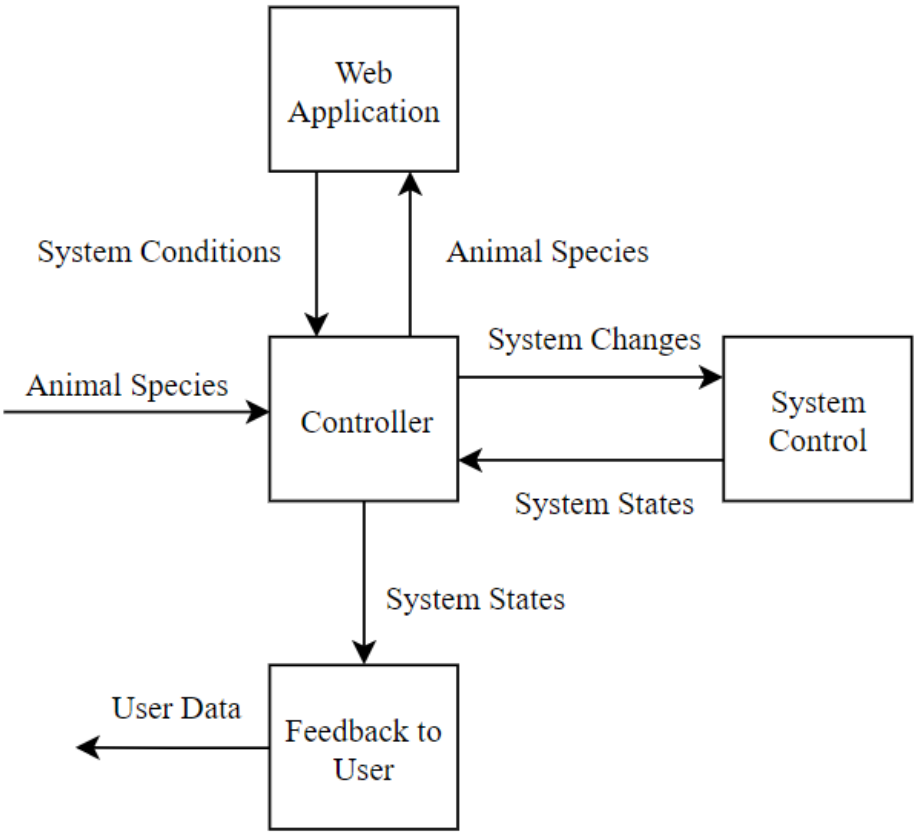

Figure 3: Level 1 Hardware Diagram

Table 4: Controller Module, Level 1

Module	Controller
Designer	Shane Benner
Inputs	Animal Species, System Settings, System States
Outputs	Animal Species, System Changes, System States
Description	The controller will have the specified animal species and will web scrape all the necessary information for the system. It will calculate the changes needed for the system based on the current system states. It will provide feedback of the current system states to the user.

Table 5: Web Module, Level 1

Module	Web
Designer	Shane Benner
Inputs	Animal Species
Outputs	System Conditions
Description	This will handle web scraping system settings depending on the species of animal. It will be sent back to the controller.

Table 6: System Control Module, Level 1

Module	System Control
Designer	Shane Benner
Inputs	System Changes
Outputs	System States
Description	This will be changed based on system settings gathered. The necessary changes of the system will be made until the proper conditions are met.

Table 7: Feedback to User Module, Level 1

Module	Feedback to the User
Designer	Shane Benner
Inputs	Processed Data
Outputs	User
Description	These are the displays, including the web application display and the liquid crystal display.

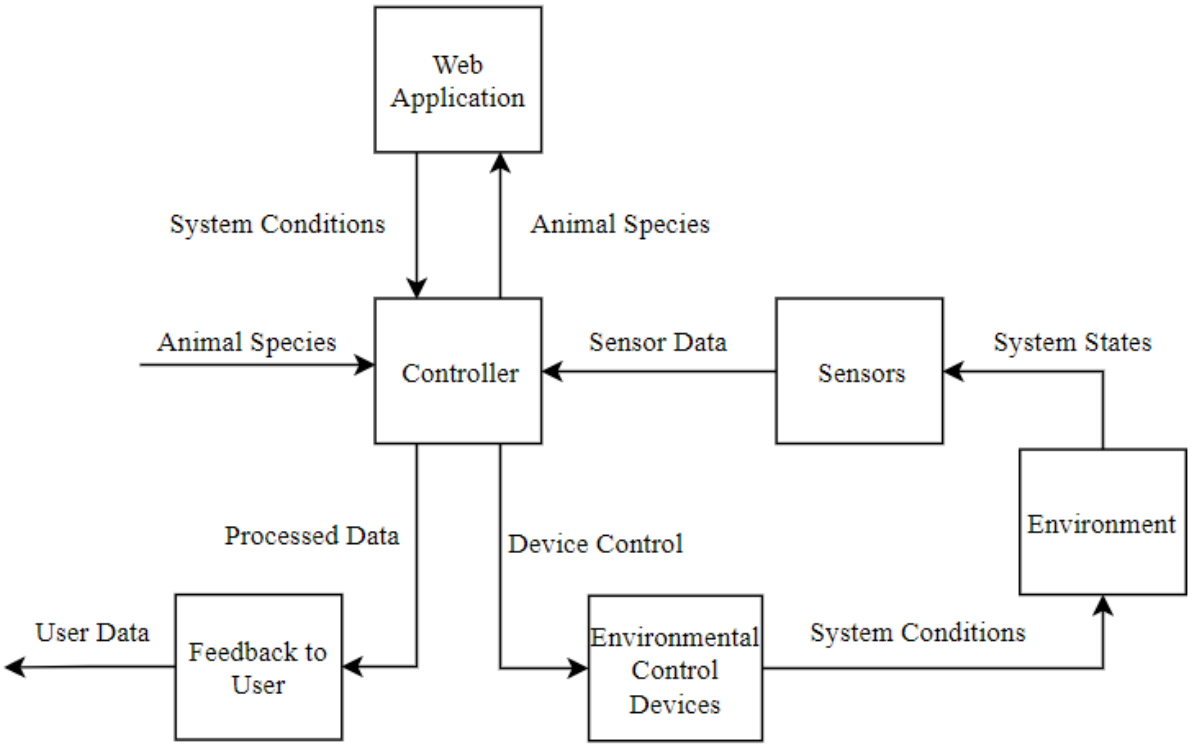

Figure 4: Level 2 Hardware Diagram

Table 8: Web Module, Level 2

Module	Web
Designer	Shane Benner
Inputs	Animal Species
Outputs	System Conditions
Description	This will handle web scraping system settings depending on the species of animal. It will be sent back to the controller.

Table 9: Controller Module, Level 2

Module	Controller
Designer	Shane Benner
Inputs	Animal Species, System Conditions, Sensor Data
Outputs	Animal Species, Processed Data, Device Control
Description	The controller will have the specified animal species and will web scrape all the necessary information for the system. It will calculate the changes needed for the system based on the current system states. It will provide feedback of the current system states to the user.

Table 10: Sensors Module, Level 2

Module	Sensors
Designer	Shane Benner
Inputs	System States
Outputs	Sensor Data
Description	The sensors in the environment will send information about the environment to the controller. This includes sensors measuring temperature, humidity, air quality, movement, and enclosure lid open/close.

Table 11: Environment Module, Level 2

Module	Environment
Designer	Shane Benner
Inputs	System Conditions
Outputs	Sensor States
Description	This is the animal's enclosure, where changes are to be made.

Table 12: Environment Control Devices Module, Level 2

Module	Environment Control Devices
Designer	Shane Benner
Inputs	Device Control
Outputs	System Conditions
Description	These are the devices that control the conditions of the environment. This includes a heating fixture, a humidifier/dehumidifier, fans, and an alarm.

Table 13: Feedback Module, Level 2

Module	Feedback to the User
Designer	Shane Benner
Inputs	Processed Data
Outputs	User
Description	These are the displays, including the web application display and the liquid crystal display.

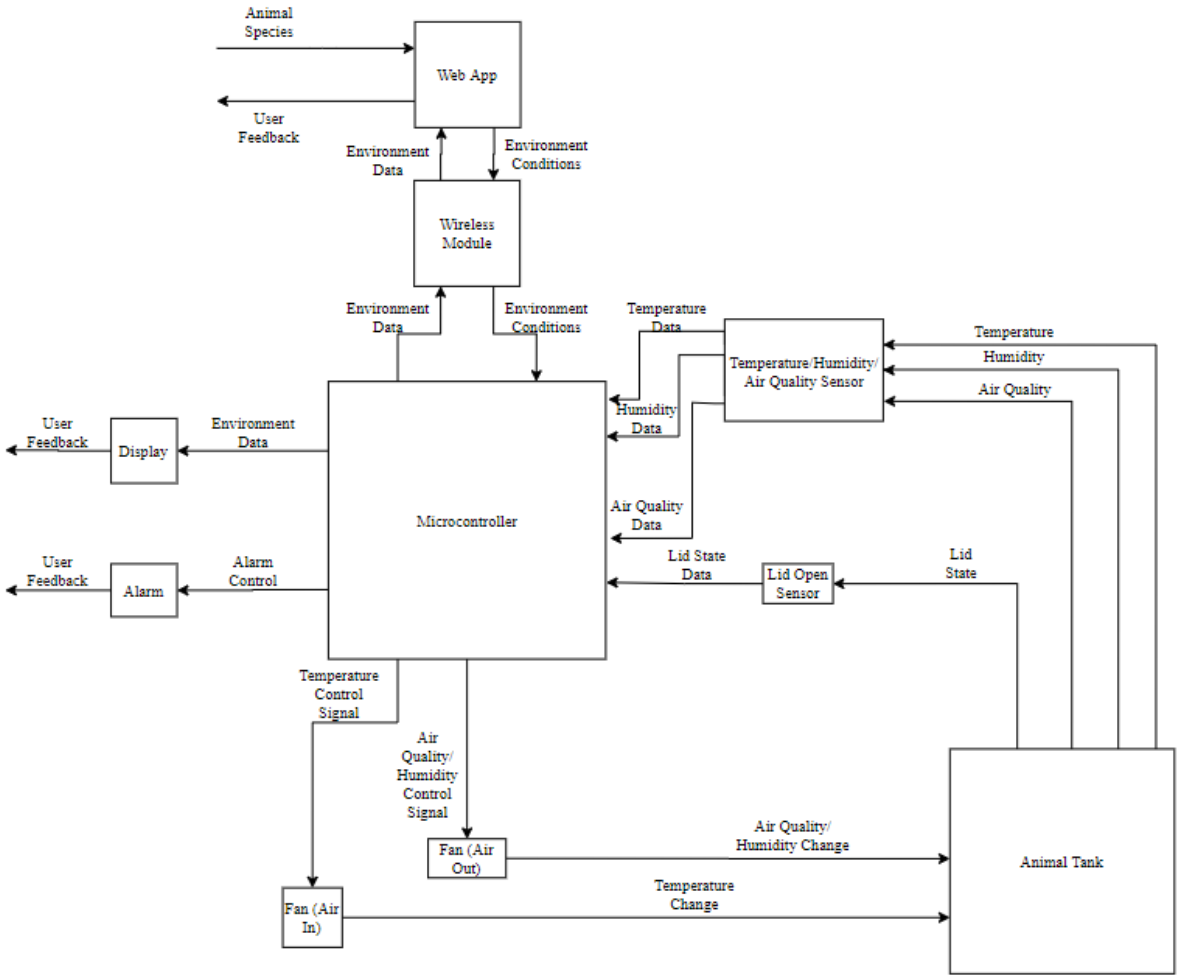

Figure 5: Level 3 Hardware Diagram

Table 14: Web Application Module, Level 3

Module	Web App
Designer	Shane Benner
Inputs	Animal Species, Environment Data
Outputs	User Feedback, Environment Conditions
Description	This will handle web scraping system settings depending on the species of animal. It will be sent back to the controller. This will also handle the user input to send to the system.

Table 15: Web Module, Level 3

Module	Web Module
Designer	Shane Benner
Inputs	Environment Data, Environment Conditions
Outputs	Environment Data, Environment Conditions
Description	This will handle the wireless communication between the Microcontroller and the Web App. Receiving the necessary Environment Conditions from the Web App, and the Environment Data from the MicroController.

Table 16: Microcontroller Module, Level 3

Module	Microcontroller
Designer	Shane Benner
Inputs	Environment Conditions, Temperature Data, Air Quality Data, Visual Data, Lid State Data, Water Level Data
Outputs	Environment Data, Alarm Control Signal, Temperature Control Signal, Humidity Control Signal
Description	The Microcontroller will take the environment conditions sent from the web app. It will calculate the changes needed for the Animal Tank based on the current Environment Data and will Output Control Signals to reach the Environment Conditions. It will provide feedback of the current system states to the user.

Table 17: Temperature/Humidity Sensor Module, Level 3

Module	Temperature/Humidity/Air Quality Sensor
Designer	Shane Benner
Inputs	Temperature, Humidity, Air Quality
Outputs	Temperature Data, Humidity Data
Description	The Temperature/Humidity/Air Quality Sensor will monitor the temperature. humidity, and air quality of the Animal Tank. It will then send the data Data to the Microcontroller.

Table 18: Lid Sensor Module, Level 3

Module	Lid Open Sensor
Designer	Shane Benner
Inputs	Lid State
Outputs	Lid State Data
Description	The Lid Open Sensor will monitor Lid State of the Animal Tank. It will then send the Lid State Data to the Microcontroller.

Table 19: Enclosure Module, Level 3

Module	Animal Tank
Designer	Shane Benner
Inputs	Humidity Change, Air Quality Change, Temperature Change
Outputs	Water Level, Lid State, Visuals, Air Quality, Temperature, Humidity
Description	This is the animal's enclosure. The state of the environment will be monitored and changed based on the Environment Conditions for the given species.

Table 20: Fan Module, Level 3

Module	Fan (Air Out)
Designer	Shane Benner, Cassie Allender
Inputs	Air Quality and Humidity Control Signal
Outputs	Air Quality and Humidity Change
Description	The Fan will be used to change the air quality and/or humidity of the Animal Tank. The Microcontroller will control this based on the Environment Conditions and will do so with an Air Quality and Humidity Control Signal.

Table 21: Heat Lamp/Fan Module, Level 3

Module	Fan (Air In)
Designer	Shane Benner
Inputs	Temperature Control Signal
Outputs	Temperature Change
Description	The Heat Lamp/Fan will be used to change the Temperature of the Animal Tank. The Microcontroller will control this based on the Environment Conditions and will do so with a Temperature Control Signal.

Table 22: Alarm Module, Level 3

Module	Alarm
Designer	Shane Benner
Inputs	Alarm Control Signal
Outputs	User Feedback
Description	The Alarm will be used to alert the user that the Lid to the Animal Tank is open.

Table 23: Display Module, Level 3

Module	Display
Designer	Shane Benner
Inputs	Environment Data
Outputs	User Feedback
Description	The Display will display the conditions of the environment through the use of an LCD screen.

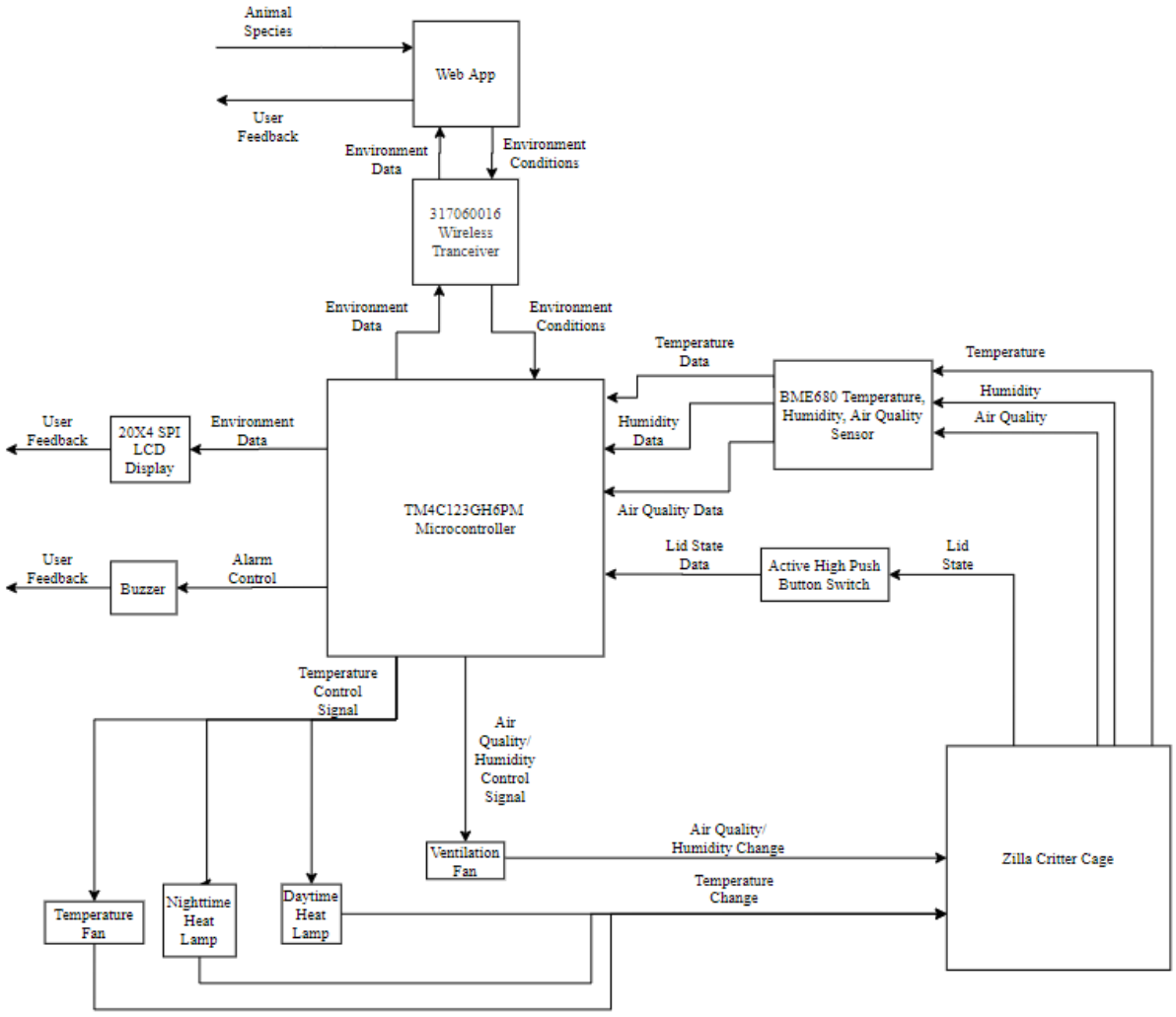

Figure 6: Level 4 Hardware Diagram

Table 24: Web Application Module, Level 4

Module	Web App
Designer	Shane Benner
Inputs	Animal Species, Environment Data
Outputs	User Feedback, Environment Conditions
Description	This will handle web scraping system settings depending on the species of animal. It will be sent back to the controller. This will also handle the user input to send to the system.

Table 25: Serial Wireless Transceiver

Module	Serial Wireless Transceiver
Designer	Shane Benner
Inputs	Environment Data, Environment Conditions
Outputs	Environment Data, Environment Conditions
Description	This will handle the wireless communication between the TM4C123GH6PM Microcontroller and the Web App. Receiving the necessary Environment Conditions from the Web App, and the Environment Data from the TM4C123GH6PM Microcontroller.

Table 26: Microcontroller Module, Level 4

Module	TM4C123GH6PM Microcontroller
Designer	Shane Benner
Inputs	Environment Conditions, Temperature Data, Air Quality Data, Visual Data, Lid State Data, Water Level Data
Outputs	Environment Data, Alarm Control Signal, Temperature Control Signal, Humidity Control Signal
Description	The TM4C123GH6PM Microcontroller will take the environment conditions sent from the web app. It will calculate the changes needed for the Animal Tank based on the current Environment Data and will Output Control Signals to reach the Environment Conditions. It will provide feedback of the current system states to the user through a Buzzer and a 20X4 SPI LCD Display.

Table 27: Temperature and Humidity Sensor Module, Level 4

Module	BME680 Digital Temperature, Humidity, Air Quality Sensor
Designer	Shane Benner, Cassie Allender
Inputs	Temperature, Humidity, Air Quality
Outputs	Temperature Data, Humidity Data, Quality Data
Description	The BME680 Digital Temperature, Humidity & Air Quality Sensor will monitor the Temperature, Humidity, and Air Quality of the Zilla Critter Cage. It will then send the Temperature Data, Humidity Data, and Air Quality Data to the TM4C123GH6PM Microcontroller.

Table 28: Push Button Switch Module, Level 4

Module	Active High Push Button Switch
Designer	Shane Benner
Inputs	Lid State
Outputs	Lid State Data
Description	The Active High Push Button Switch will monitor Lid State of the Zilla Critter Cage. It will then send the Lid State Data to the TM4C123GH6PM Microcontroller.

Table 29: Cage Module, Level 4

Module	Zilla Critter Cage
Designer	Shane Benner
Inputs	Humidity Change, Air Quality Change, Temperature Change
Outputs	Water Level, Lid State, Visuals, Air Quality, Temperature, Humidity
Description	This is the animal's enclosure. The state of the environment will be monitored and changed based on the Environment Conditions for the given species.

Table 30: Ventilation Fan Module, Level 4

Module	Ventilation Fan
Designer	Shane Benner, Cassie Allender
Inputs	Air Quality/Humidity Control Signal
Outputs	Air Quality/Humidity Change
Description	The Ventilation Fan will be used to change the air quality of the Zilla Critter Cage. The TM4C123GH6PM Microcontroller will control this based on the Environment Conditions and will do so with an Air Quality/Humidity Control Signal.

Table 31: Daytime Heat Lamp Module, Level 4

Module	Daytime Heat Lamp
Designer	Shane Benner
Inputs	Temperature Control Signal
Outputs	Temperature Change
Description	The Daytime Heat Lamp will be used to change the Temperature of the Zilla Critter Cage. The TM4C123GH6PM Microcontroller will control this based on the Environment Conditions and will do so with a Temperature Control Signal.

Table 32:Night-time Heat Lamp Module, Level 4

Module	Nighttime Heat Lamp
Designer	Shane Benner
Inputs	Temperature Control Signal
Outputs	Temperature Change
Description	The Nighttime Heat Lamp will be used to change the Temperature of the Zilla Critter Cage. The TM4C123GH6PM Microcontroller will control this based on the Environment Conditions and will do so with a Temperature Control Signal.

Table 33: Temperature Fan Module, Level 4

Module	Temperature Fan
Designer	Shane Benner
Inputs	Temperature Control Signal
Outputs	Temperature Change
Description	The Temperature Fan will be used to change the Temperature of the Zilla Critter Cage. The TM4C123GH6PM Microcontroller will control this based on the Environment Conditions and will do so with a Temperature Control Signal.

Table 34: Buzzer Module, Level 4

Module	Buzzer
Designer	Shane Benner
Inputs	Alarm Control Signal
Outputs	User Feedback
Description	The Buzzer will be used to alert the user that the Lid to the Animal Tank is open.

Table 35: LCD Display Module, Level 4

Module	20X4 SPI LCD Display
Designer	Shane Benner
Inputs	Environment Data
Outputs	User Feedback
Description	The 20X4 SPI LCD Display will display the conditions of the environment.

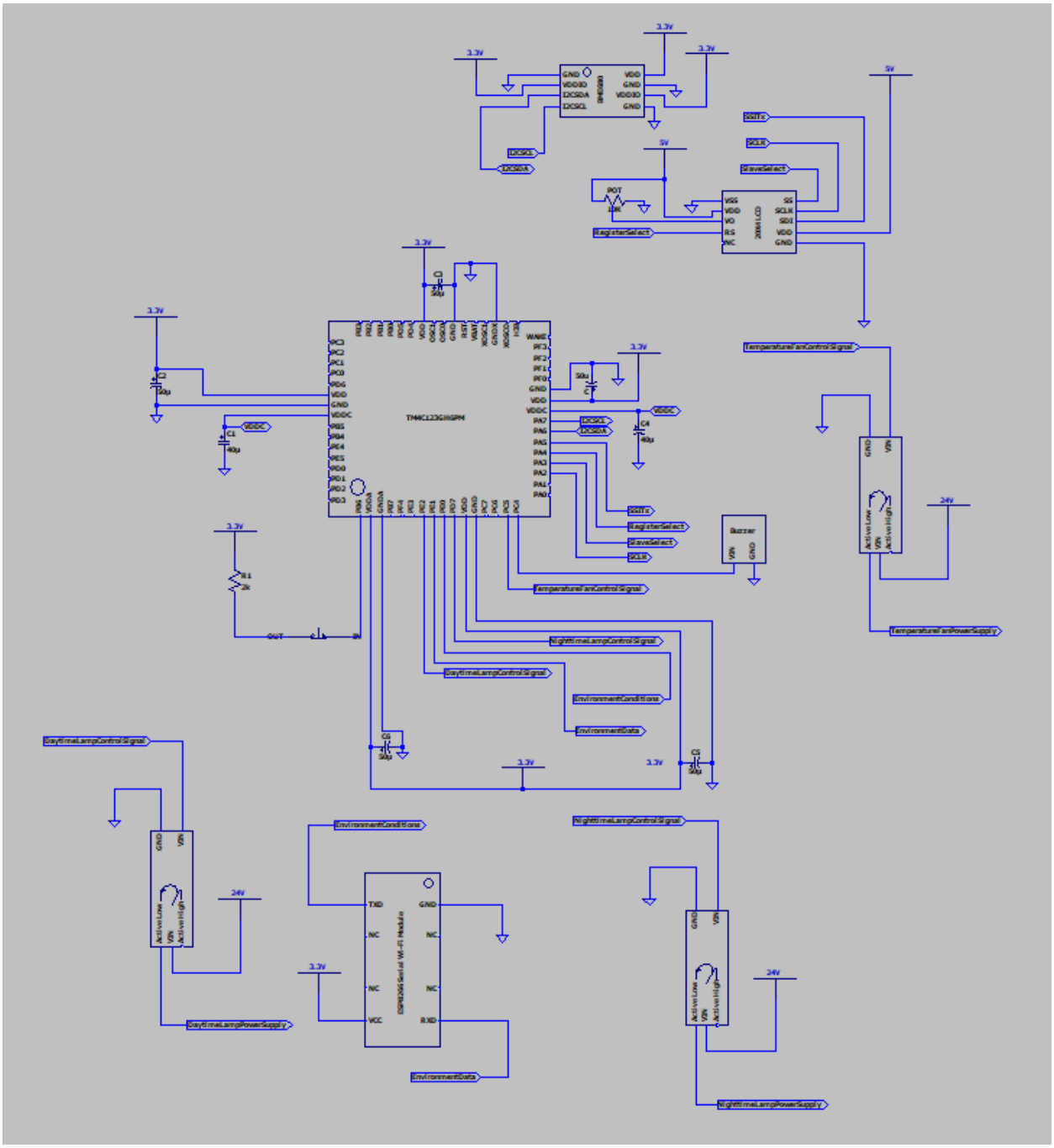

Figure 7: System Circuit Diagram (S.B. & C.A.)

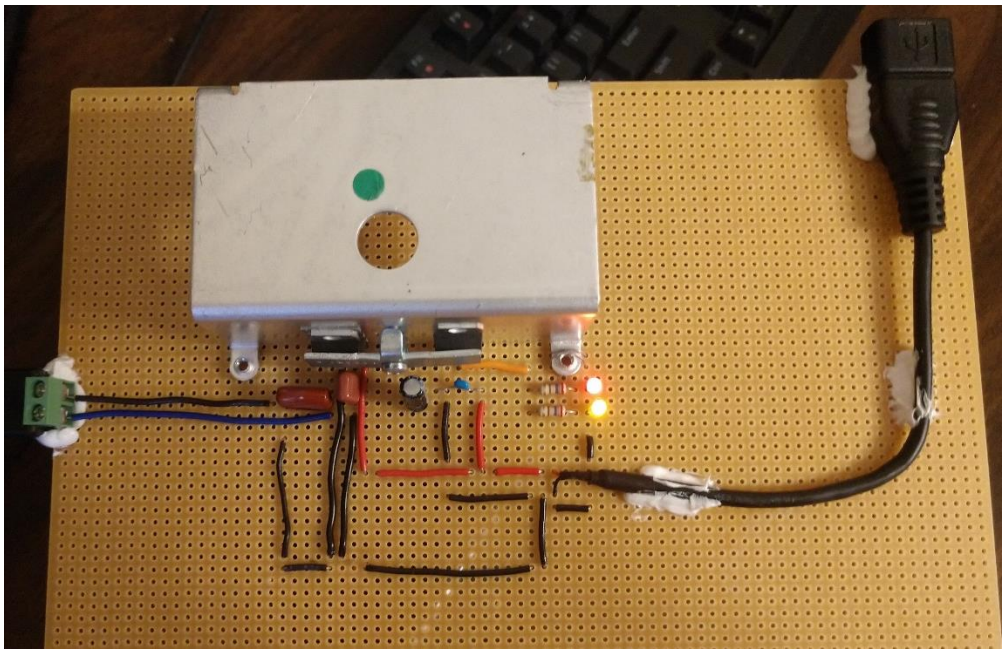

Figure 8: Power supply and distribution (A.W.)

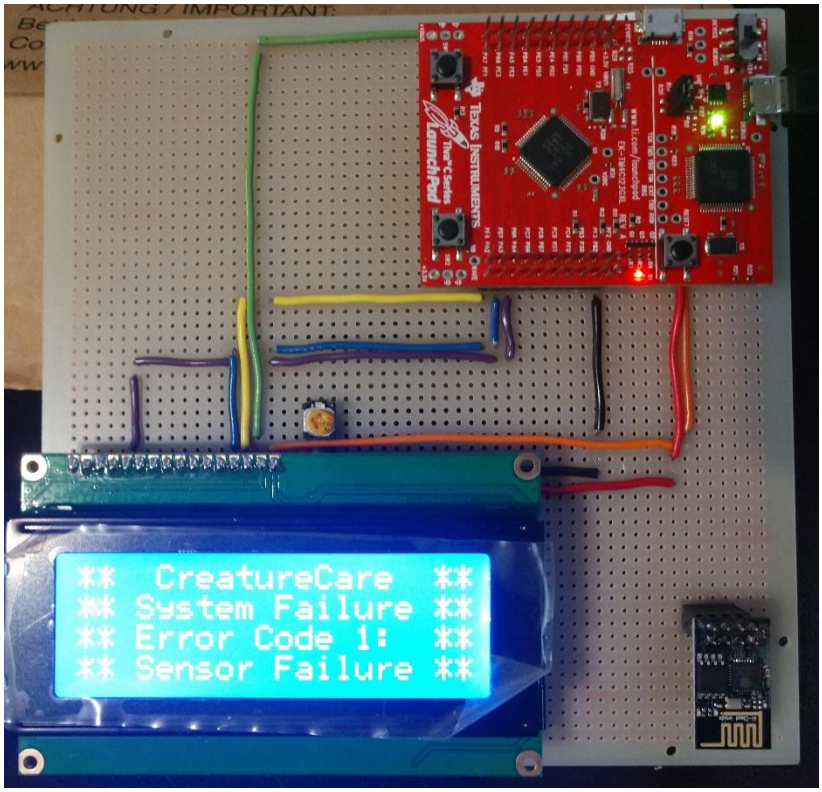

Figure 9: Liquid Crystal Display (A.W.)

6.1.1 Theory of Operation (C.A.)

Based on the species of the animal and/or user preference, the enclosure controls the temperature, humidity, and air quality. The values of each parameter measured by the BME680 sensor will be read by the Tiva C microcontroller via I2C, along with mini pushbutton switches placed around the cage lid that will be read in binary values by the microcontroller. This data is sent to the liquid crystal display placed on the cage and the web application. Commands are sent to the system control devices, causing them to turn on and off. Specifically, two fans will be placed on the top of the lid, one that brings outside air in and one that brings inside air out. Also on top of the lid is the heat lamp, which includes a daylight bulb and a nighttime bulb. The heat lamp will run at all times, switching between the day and night bulb, syncing with the natural changes of light outside. If the temperature inside the tank passes the maximum temperature set point, the fan blowing outside are in will turn on until the temperature is back to an acceptable value. When air quality or humidity passes their maximum set points, the fan blowing inside air out will turn on. When the lid is not secure and all values of the mini pushbutton switches are not the same, a digital piezo buzzer will sound and a warning will be sent to the liquid crystal display and the web application.

6.2 Software Design (A.W. & C.A.)

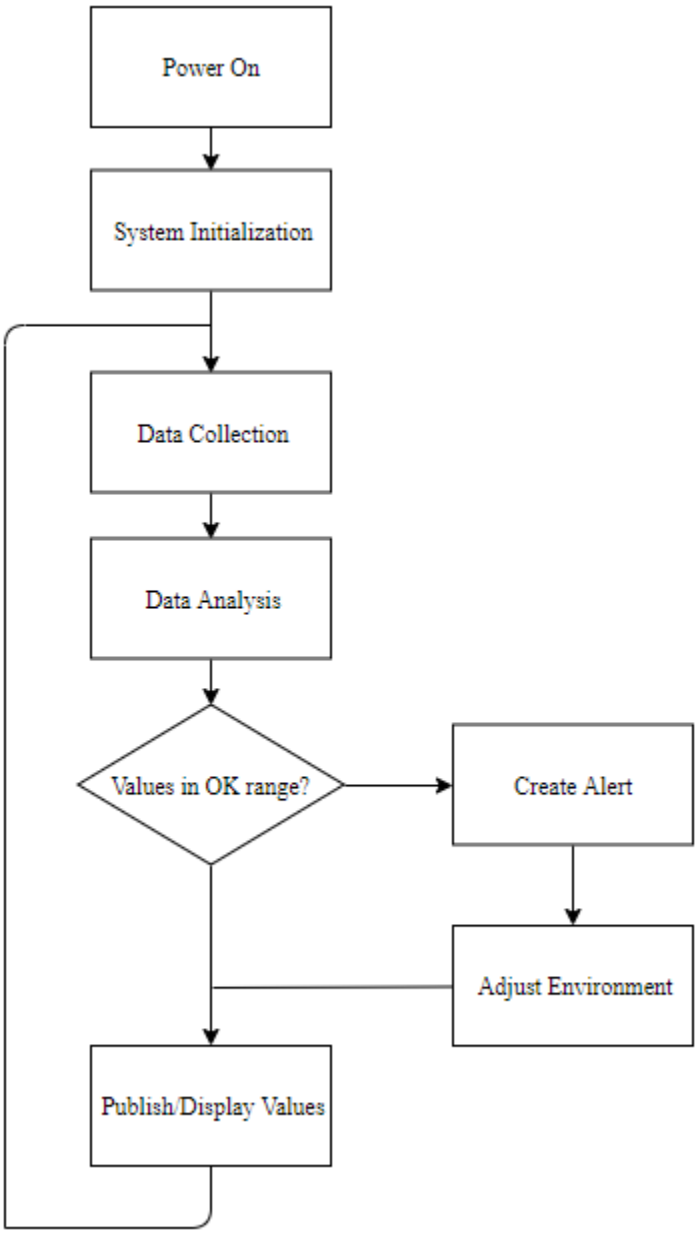

Figure 10: Level 0 Software Flowchart

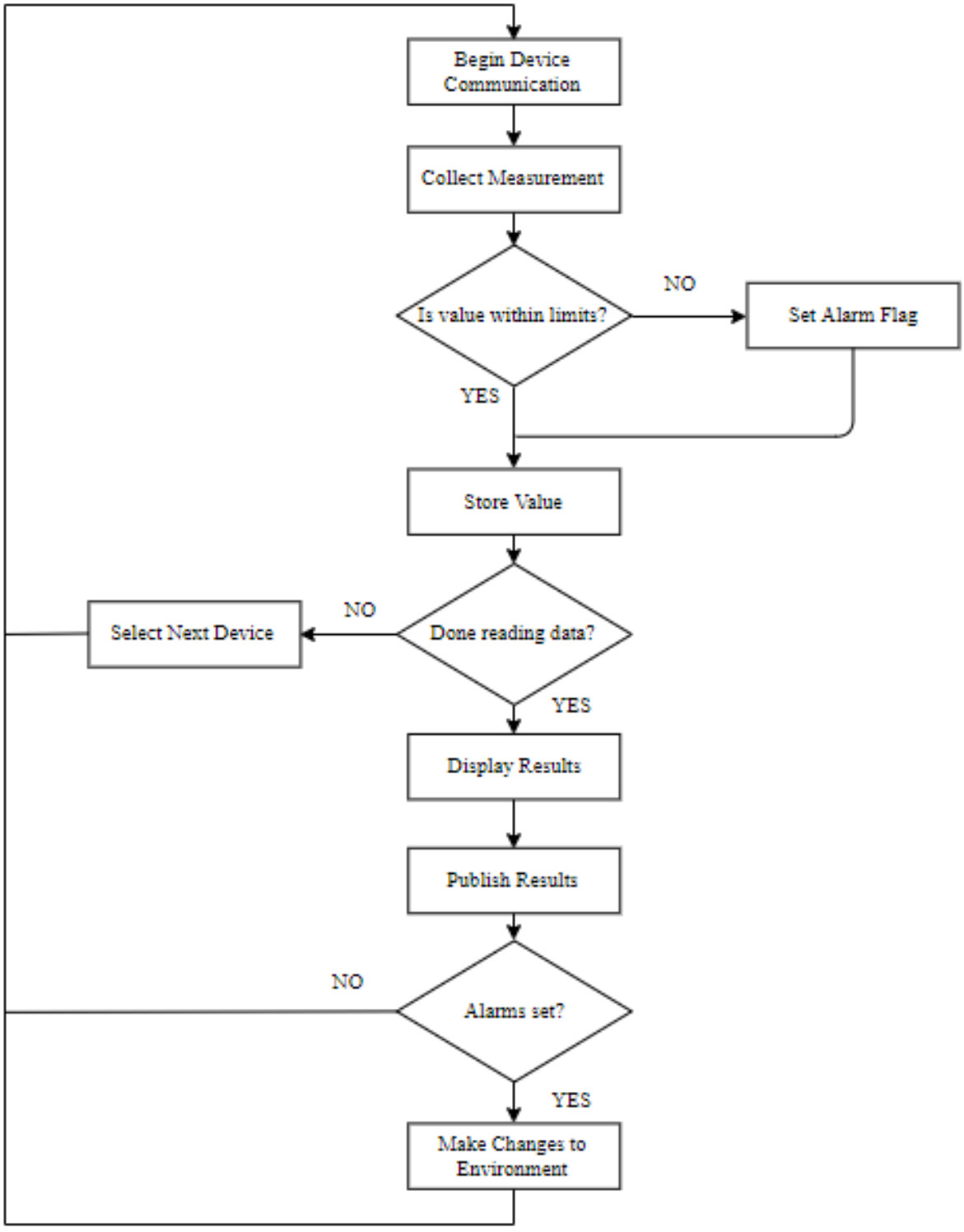

Figure 11: Level 1 Data Collection and Analysis Software Flowchart

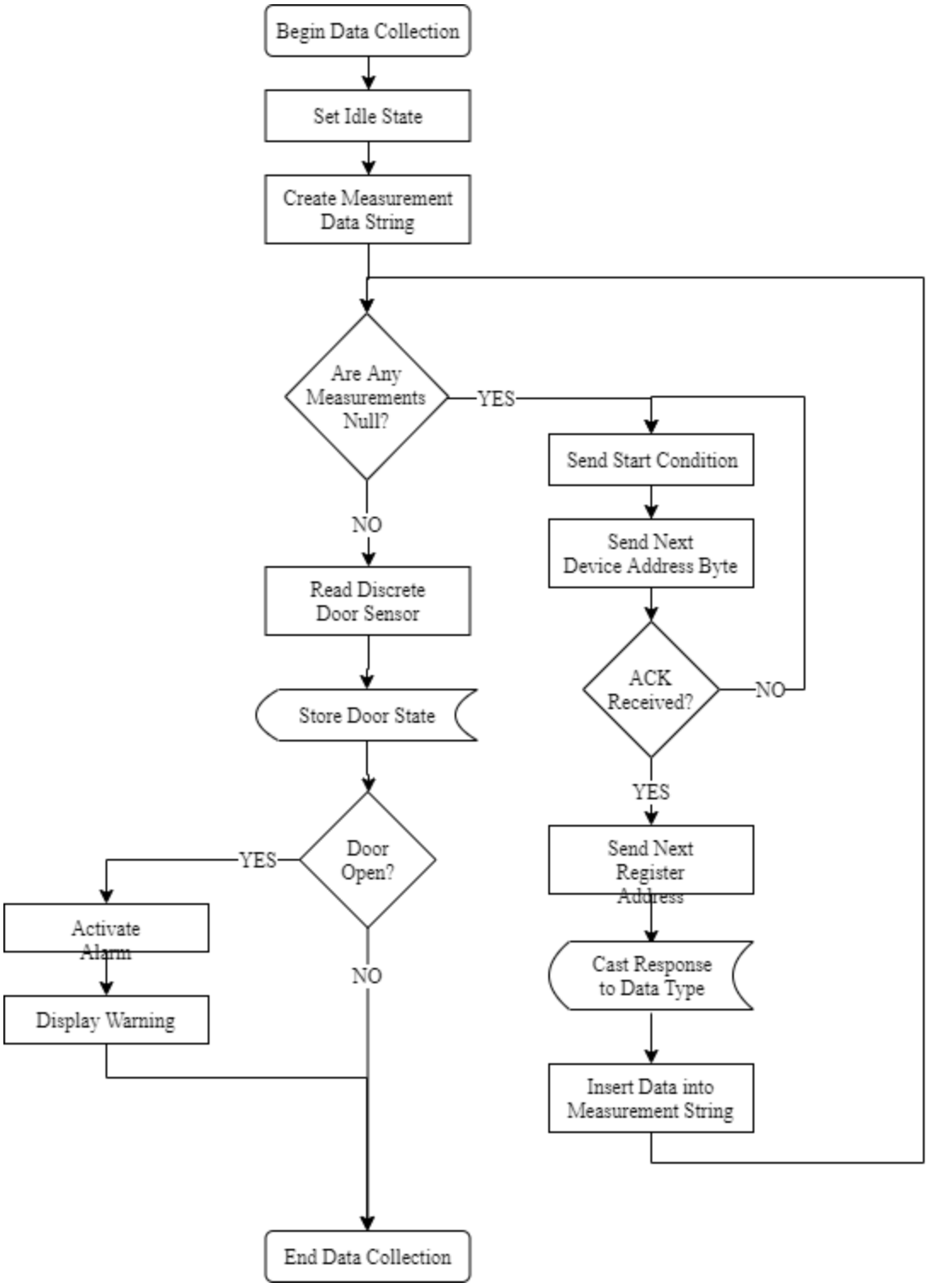

Figure 12: Data Collection Level 2 Flowchart

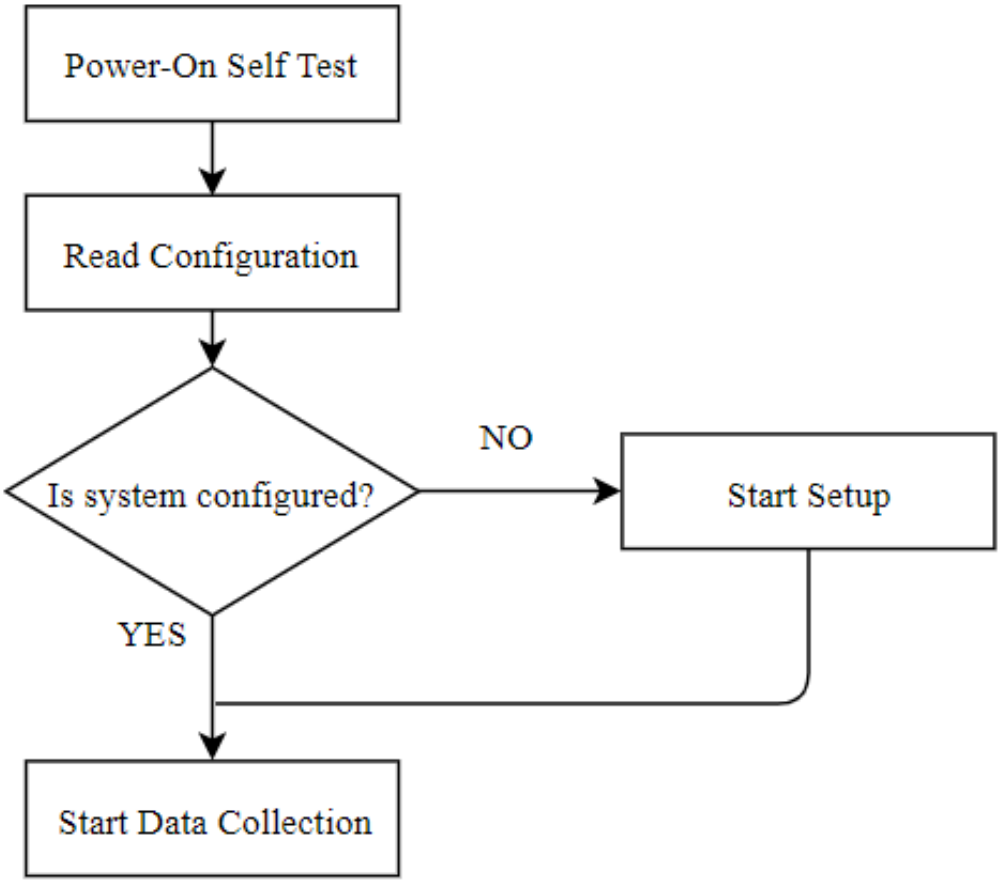

Figure 13: Level 1 Data Initialization Software Flowchart

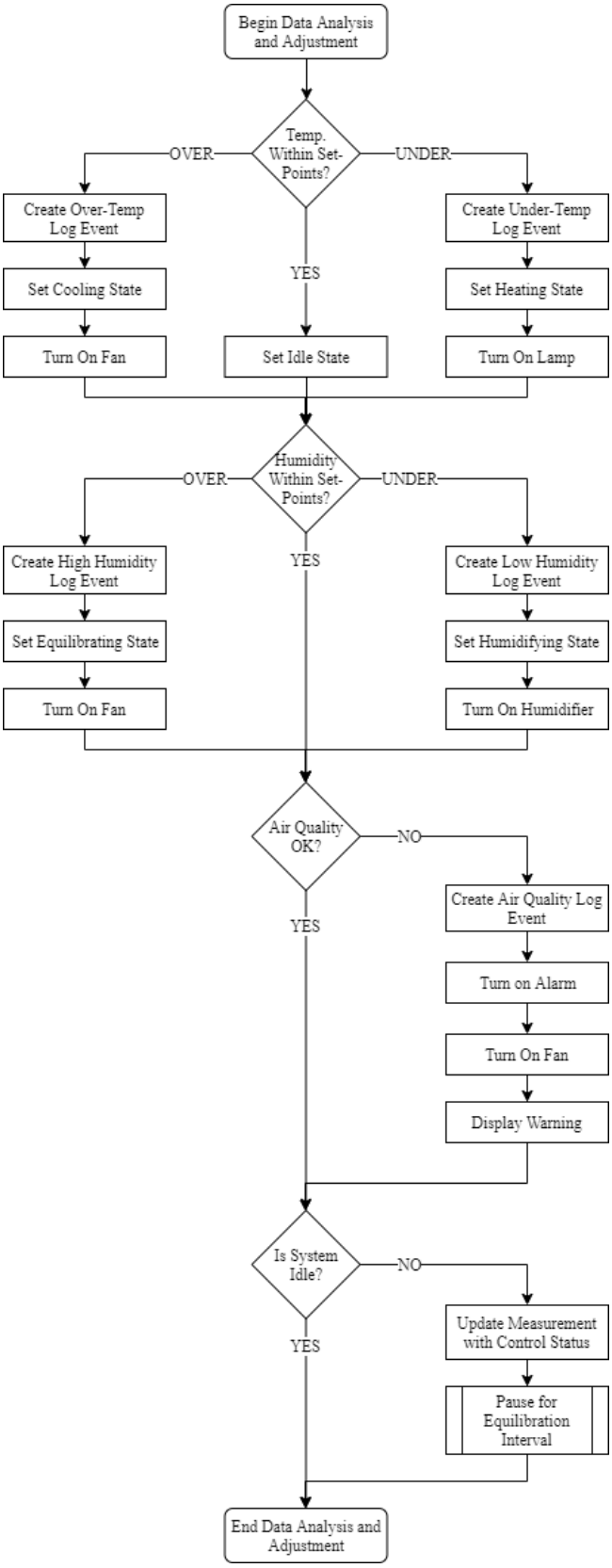

Figure 14: Data Analysis Level 2 Flowchart

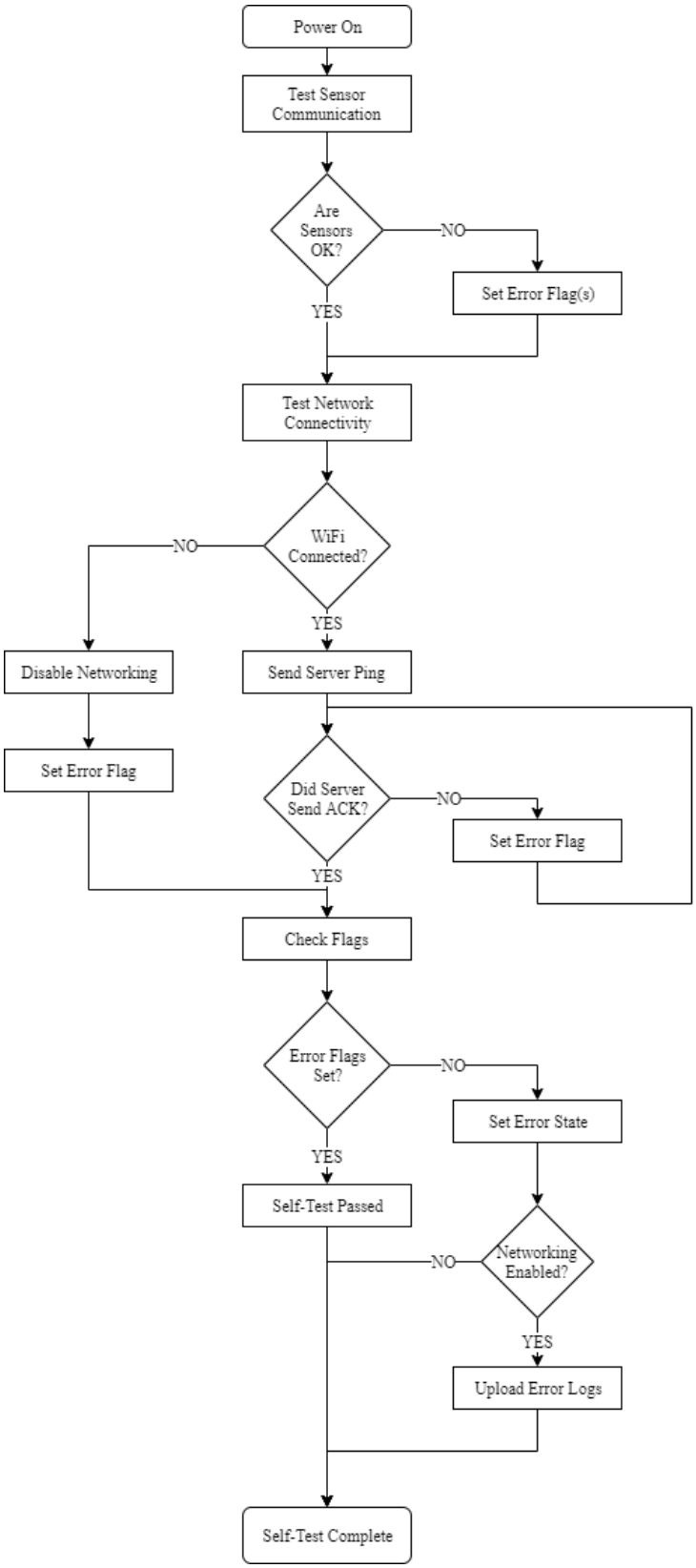

Figure 15: Power-On Self Test Level 2 Flowchart

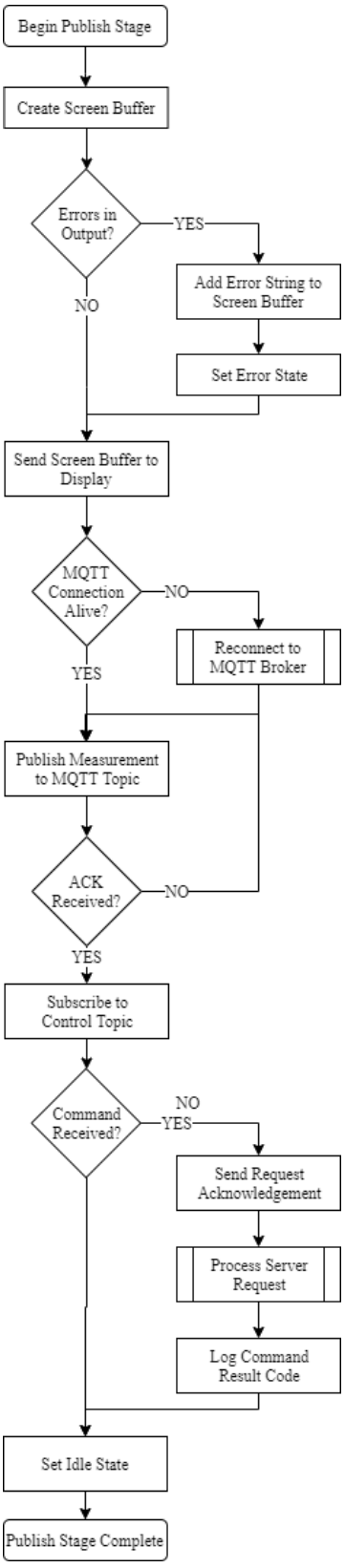

Figure 16: Publish/Display Data Level 2 Flowchart

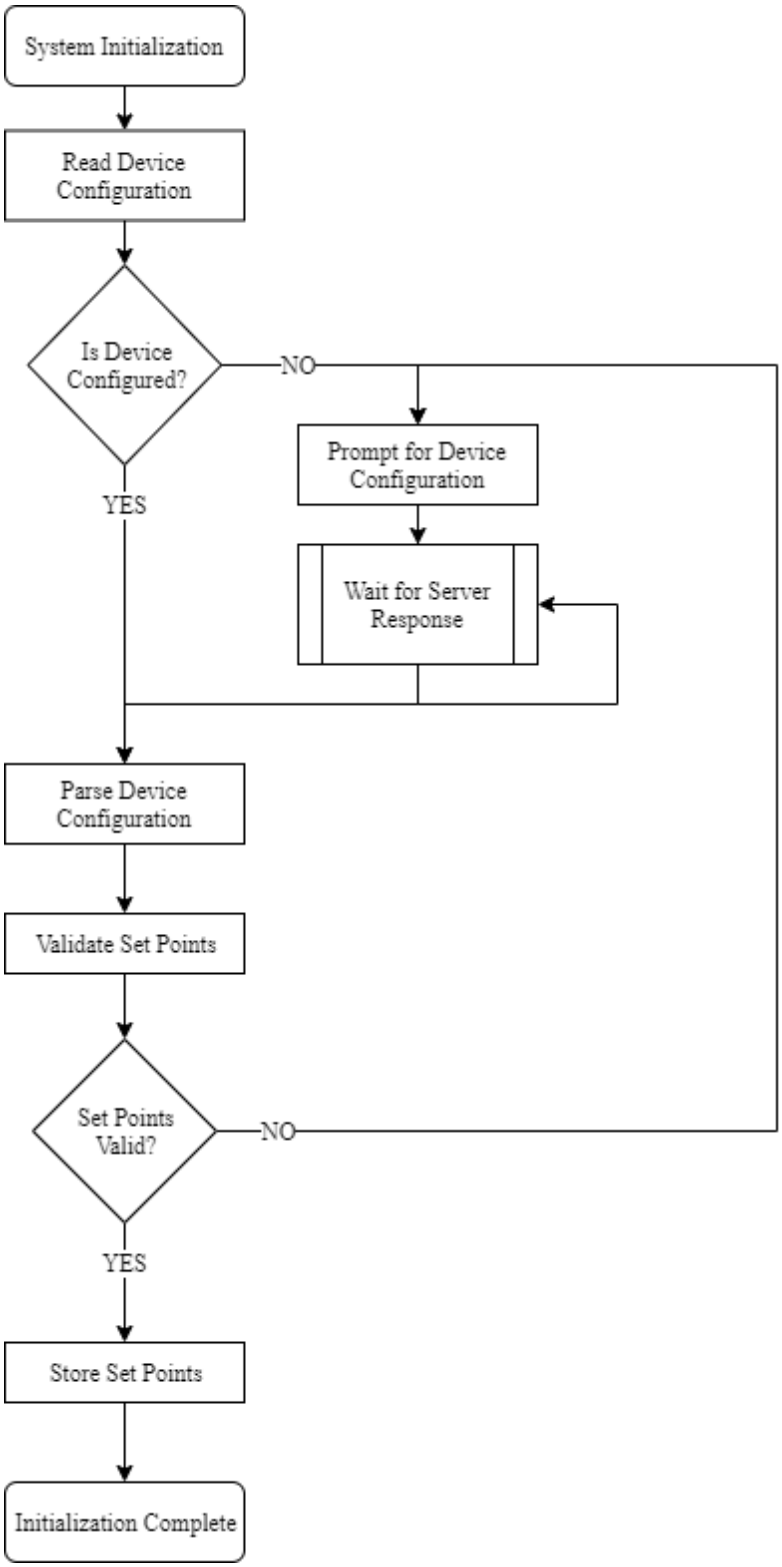

Figure 17: System Initialization Level 2 Flowchart

Table 36: Device Error Codes

Bit Pattern	Error ID	Error Description
00	0	No error
01	1	Sensor/module communication timeout
10	2	Unable to connect to WiFi network
11	3	MQTT server connection timeout

Table 37: Device States

Bit Pattern	State ID	State Description
0000	0	Unknown
0001	1	System Boot
0010	2	Boot Failure
0011	3	Configure
0100	4	Configuration Failure
0101	5	Initialize
0110	6	Idle
0111	7	Equilibrating
1000	8	Equilibrating – Cooling
1001	9	Equilibrating – Heating
1010	10	Equilibrating – Humidifying

Table 38: Database Measurement Data

Measurement ID	Measurement Description	Data Type	Unit
Time	Microsecond UNIX Timestamp	Long integer	µs
Air_qual	Air quality indication	Boolean	*None*
Temp_int	Internal temperature	Float	°C
Temp_ext	External temperature	Float	°C
Rh	Relative humidity	Float	%
Fan_ctrl	Fan control active	Boolean	*None*
Hum_ctrl	Humidity control active	Boolean	*None*

6.2.1 C-Code: Gas Sensor (C.A.)

```
/*
 * Title: Tiva I2C proof of concept
 *
 * Author: Cassie Allender
 *
 * Objective: Communicate to an air quality sensor from a TIVA C Series launchpad.
 *                    Use LED to show air quality results. Simulates fan turning on
and off.
 */

/**
 * gas.c
 */

#define PART_TM4C123GH6PM
#define LEDbase GPIO_PORTF_BASE
#define LEDred GPIO_PIN_1
#define LEDblue GPIO_PIN_2

#include <stdint.h>
#include <stdbool.h>
#include "stdlib.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_uart.h"
#include "inc/hw_gpio.h"
#include "inc/hw_pwm.h"
#include "inc/hw_types.h"
#include "driverlib/adc.h"
#include "driverlib/timer.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/udma.h"
#include "driverlib/pwm.h"
#include "driverlib/ssi.h"
#include "driverlib/systick.h"
#include "driverlib/adc.h"
#include "utils/uartstdio.h"
#include <string.h>

volatile uint32_t millis=0;

// This array is used for storing the data read from the ADC FIFO.
uint32_t ADCValues[1];

// This value is used to store the conversions for air quality measurement
```

```
// of the gas sensor.
uint32_t PPMValue;

void SycTickInt()
{
  millis++;
}
void SysTickbegin()
{
  SysTickPeriodSet(80000);
  SysTickIntRegister(SycTickInt);
  SysTickIntEnable();
  SysTickEnable();
}

void Wait(uint32_t time)
{
       uint32_t temp = millis;
       while( (millis-temp) < time){}
}

void InitConsole(void)
{
    // Enable GPIO port A which is used for UART0 pins.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    //
    // Configure the pin muxing for UART0 functions on port A0 and A1.
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);

    // Enable UART0 so that we can configure the clock.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

    // Use the internal 16MHz oscillator as the UART clock source.
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);

    // Select the alternate (UART) function for these pins.
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    // Initialize the UART for console I/O.
    UARTStdioConfig(0, 115200, 16000000);
}

int main()
.{

       SysCtlClockSet(SYSCTL_SYSDIV_2_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16
MHZ);
           SysTickbegin();
           InitConsole();

           // The ADC0 and GPIO peripherals must be enabled for use.
           SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
           SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
```

```
            SysCtlDelay(3);

            // Configuration for use of LEDs.
            GPIOPinTypeGPIOOutput(LEDbase, (LEDred|LEDblue));

            // Enable sample sequence 3 with a processor signal trigger.  Sequence 3
            // will do a single sample when the processor sends a signal to start the
            // conversion.
            ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);

            //
            // Configure step 0 on sequence 3.
            ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH0 | ADC_CTL_IE |
                                     ADC_CTL_END);

            // Since sample sequence 3 is now configured, it must be enabled.
            ADCSequenceEnable(ADC0_BASE, 3);

            // Clear the interrupt status flag.
            ADCIntClear(ADC0_BASE, 3);

            // Continuously sample gas. Using UART, values will be displayed using
Putty.
            while(1)
            {
                // Trigger the ADC conversion.
                ADCProcessorTrigger(ADC0_BASE, 3);

                // Wait for conversion to be completed.
                while(!ADCIntStatus(ADC0_BASE, 3, false))
                {
                }

                // Clear the ADC interrupt flag.
                ADCIntClear(ADC0_BASE, 3);

                // Read ADC Value.
                ADCSequenceDataGet(ADC0_BASE, 3, ADCValues);

                //
                // Use non-calibrated conversion provided in the data sheet. I use
floats in intermediate
                // math but you could use intergers with multiplied by powers of 10
and divide on the end
                // Make sure you divide last to avoid dropout.
                //
                //PPMValue = (uint32_t)(147.5 - ((75.0*3.3 *(float)ADCValues[0])) /
4096.0);
                PPMValue = (uint32_t) (((float)ADCValues[0] - 2048) / 20.48) - 15;
                //
                // Display the value on the console.
                //
                UARTprintf("Air Quality = %d%%\n", PPMValue);

                // LED remains green when air quality is good.
```

```
                // LED turns red when air quality is bad and fan must turn on.

                if(PPMValue > 10)
                     GPIOPinWrite(LEDbase,(LEDred|LEDblue), LEDred);

                else
                     GPIOPinWrite(LEDbase,(LEDred|LEDblue), LEDblue);

                // This function provides
ahttp://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf means of generating a constant
length
                // delay.  The function delay (in cycles) = 3 * parameter.  Delay
                // 250ms arbitrarily.
                SysCtlDelay(80000000 / 12);
           }
}
```

6.2.2 C-Code: Temperature Sensor Output and Fan Control (S.B.)

```
/*
 * Title: Demo Rev 0
 *
 * Author: Shane Benner
 *
 * Objective: Present a successful demonstration of reading temperature values from a
temperature sensor using I2C
 * communication protocol, and demonstrating control of a fan when a defined temperature
has been exceeded. The fan
 * will be powered from an external power supply but the control comes from a relay
controlled by TM4C123GHPM. Some
 * status LED's will also be included
 */

/*

*********************************************************************************
***************
 ***********************************Libraries and Header
Files*****************************************

*********************************************************************************
***************
 */

#include <stdarg.h>
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_i2c.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/i2c.h"
#include "driverlib/sysctl.h"
```

```
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"


/*
 *****************************************************************************
 ***************************Pre-Processor Directives**************************
*****************************************************************************
 */

#define TEMPSENSOR 0x48           // Slave address for the temperature sensor
#define CONFIGREG 0x01            // Configuration register address
// Command to set the configuration register of the temp sensor into standby mode
#define TURNON 0x00
#define REDLED GPIO_PIN_2         // Pin that will control the good status indicator
#define GREENLED GPIO_PIN_3       // Pin that will control the good status indicator
#define FANCONTROL GPIO_PIN_6     // Pin that will control the fan switch


/*

*****************************************************************************
*************************************Error Check*****************************
*****************************************************************************
 */
#ifdef DEBUG
void
__error__(char *pcFilename, uint32_t ui32Line)
{
}
#endif


/*
*****************************************************************************
*********************************Function Prototypes*************************
*****************************************************************************
 */

void ConfigureUART(void);
void InitI2C0(void);
void ConfigureGPIO(void);
void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...);
void I2CSendString(uint32_t slave_addr, char array[]);
uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg);



/*
*****************************************************************************
*************************************Main Function***************************
*****************************************************************************
 */

int main(void)
{
```

```
    uint32_t celsius;   // Variable to hold the raw data from the temp sensor, this raw
data represents temperature in celsius
    int fahrenheit;     // Variable to hold the integer value of degrees in fahrenheit

    /*
     * Set the clocking to run directly from the external crystal/
     * oscillator
     */
    SysCtlClockSet(SYSCTL_SYSDIV_2 | SYSCTL_USE_PLL | SYSCTL_OSC_INT |
SYSCTL_XTAL_16MHZ);

    ConfigureUART();    // Setup UART communications
    InitI2C0();         // Setup I2C on module 0
    ConfigureGPIO();    // Setup GPIO

        while(1)
           {
               // Sends a bit asking for register 0, the data register of the TC74 sensor
               I2CSend(TEMPSENSOR, 1 ,0);

               SysCtlDelay(100);  // Wait 1ms for transmission to finish

               // Read the temperature from the sensor
               celsius = I2CReceive(TEMPSENSOR, 0x0);

               SysCtlDelay(100);  // Wait 1ms for transmission to finish

               UARTprintf("%d\n", celsius);

               if(celsius >= 23)
               {
                 GPIOPinWrite(GPIO_PORTF_BASE, GREENLED, 0x00);
                 GPIOPinWrite(GPIO_PORTF_BASE, REDLED, REDLED);
                 GPIOPinWrite(GPIO_PORTD_BASE, FANCONTROL, FANCONTROL);
               }
               else
               {
                 GPIOPinWrite(GPIO_PORTF_BASE, REDLED, 0x00);
                 GPIOPinWrite(GPIO_PORTD_BASE, FANCONTROL, 0x00);
                 GPIOPinWrite(GPIO_PORTF_BASE, GREENLED, GREENLED);

               }

               SysCtlDelay(10000);
           }



       return 0;
}
/*
*********************************************************************************
*******************************Function Definitions******************************
*********************************************************************************
 */

// Initialize UART communication
void ConfigureUART(void)
```

```
{
    // UART is accessible via Debug USB port, as a virtual serial port

    // Enable the GPIO peripheral used by the UART
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    /*
     * Configure the pin muxing for UART0 functions on port A0 and A1. This step is not
necessary if your
     * part does not support pin muxing.
     */
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);

    // Enable UART0 so that we can configure the clock
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

    // Use the internal 16MHz oscillator as the UART clock source
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);

    // Select the alternate (UART) function for these pins
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    UARTStdioConfig(0, 115200, 16000000);
}



// Initialize I2C module 0
// Slightly modified version of TI's example code
void InitI2C0(void)
{
    // Enable I2C module 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);

    // Reset module
    SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);

    // Enable GPIO peripheral that contains I2C 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

    // Configure the pin muxing for I2C0 functions on port B2 and B3.
    GPIOPinConfigure(GPIO_PB2_I2C0SCL);
    GPIOPinConfigure(GPIO_PB3_I2C0SDA);

    // Select the I2C function for these pins
    GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
    GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);

    /*
     * Enable and initialize the I2C0 master module. Use the system clock
     * for the I2c0 mosule. The last parameter sets the I2C data transfer
     * rate. If false the data rate is set to 100kbps and if true the
     * data rate will be set to 400kbs.
     */
    I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), false);

    // Clear I2C FIFOs
```

```
    HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 80008000;
}

// Initializes GPIO Output pins to use for control
void ConfigureGPIO(void)
{
       // Enable GPIO ports for control peripherals
       SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

       while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOF));    // Wait

       SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);

       while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOD));

       //
       // Configure the GPIO port for the LED operation.
       //
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, REDLED);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GREENLED);
    GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, FANCONTROL);

}



// Sends an I2C command to the specified slave
void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...)
{
    uint8_t i;

    /*
     * Tell the master module what address it will place on the bus when
     * communicating with the slave.
     */
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);

    // Stores list of variable number of arguments
    va_list vargs;

    /*
     * Specifies the va_list to "open" and the last fixed argument so
     * vargs knows where to start looking
     */
    va_start(vargs, num_of_args);

    // Put the data to be sent into the FIFO buffer
    I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));

    /*
     * If there is only one argument, then we only need to use the single
     * send I2C function
     */
    if(num_of_args == 1)
    {
        // Initiate send of data from the MCU
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);
```

```
        // Wait until MCU is done transferring
        while(I2CMasterBusy(I2C0_BASE));

        // "Close" variable argument list
        va_end(vargs);
    }

    // Otherwise, we start transmission of multiple bytes on the I2C bus
    else
    {
        // Initiate send of data from the MCU
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);

        /*
         * Send num_of_args-2 pieces of data, using the
         * BURST_SEND_CONT command of the I2C module
         */
        for(i = 1; i < (num_of_args - 1); i++)
        {
            //put next piece of data into I2C FIFO
            I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
            //send next data that was just placed into FIFO
            I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);

            // Wait until MCU is done transferring.
            while(I2CMasterBusy(I2C0_BASE));
        }

        // Put the last piece of data into the I2C FIFO buffer
        I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
        // Send the data that was just placed into the FIFO buffer
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
        // Wait until the MCU is done transferring
        while(I2CMasterBusy(I2C0_BASE));

        // "Close" variable args list
        va_end(vargs);
    }
}


// Sends an array of data via I2C to the specified slave
void I2CSendString(uint32_t slave_addr, char array[])
{
    /*
     * Tell the master module what address it will place on the bus when
     * communicating with the slave.
     */
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);

    // Put data to be sent into the FIFO buffer
    I2CMasterDataPut(I2C0_BASE, array[0]);

    /*
     * If there is only one argument, we only need to use the single
     * send I2C function
     */
    if(array[1] == '\0')
```

```
    {
        // Initiate send of data from the MCU
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);

        // Wait until MCU is done transferring.
        while(I2CMasterBusy(I2C0_BASE));
    }

    // Otherwise, we start transmission of multiple bytes on the I2C bus
    else
    {
        // Initiate send of data from the MCU
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);

        // Wait  until MCU is done transferring.
        while(I2CMasterBusy(I2C0_BASE));

        // Initialize index into array
        uint8_t i = 1;

        /*
         * Send num_of_args - 2 pieces of data, using the
         * BURST_SEND_CONT command of the I2C module
         */
        while(array[i + 1] != '\0')
        {
            // Put the next piece of data into I2C FIFO buffer
            I2CMasterDataPut(I2C0_BASE, array[i++]);

            // Send next data that was just placed into FIFO buffer
            I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);

            // Wait until the MCU is done transferring
            while(I2CMasterBusy(I2C0_BASE));
        }

        // Put the last piece of data into I2C FIFO buffer
        I2CMasterDataPut(I2C0_BASE, array[i]);

        /*
         * Send the piece of data that was just placed into the FIFO
         * buffer
         */
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);

        // Wait until the MCU is done transferring
        while(I2CMasterBusy(I2C0_BASE));
    }
}

// Read specified register on slave device
uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg)
{
    /*
     * Specify that we are writing (a register address) to the slave
     * device
     */
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);
```

```
    // Specify the register to be read
    I2CMasterDataPut(I2C0_BASE, reg);

    // Send control byte and a register address byte to slave device
    while(I2CMasterBusy(I2C0_BASE));

    //Specify that we are going to read from slave device
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, true);

    // Send control byte and read from the register we specified
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);

    // Wait for MCU to finish transaction
    while(I2CMasterBusy(I2C0_BASE));

    // Return data pulled from the specified register
    return I2CMasterDataGet(I2C0_BASE);
}
```

6.2.3 Python Code: MQTT Snooper (Shows messages in the MQTT channels) (A.W.)

```
#!/usr/bin/env python3.6
import sys
import ssl
import paho.mqtt.client as MQTT

# Connection details
MQTT_BROKER = "broker.hivemq.com"
MQTT_PORT = 1883
MQTT_CLIENT_NAME = "ESP_snooper"
MQTT_TOPIC = 'DT17/#' # Topic '#' subscribes to all

# Callback function on client connect
def on_connect(client, userdata, flags, rc):
	client.subscribe(topic=MQTT_TOPIC, qos=2)
	print('[INFO] Client connected', client._client_id.decode())

# Callback function on message receipt
def on_message(client, userdata, message):
	print('------------------------------')
	print('[MESSAGE] TOPIC:', message.topic)
	print('[MESSAGE] QoS: %d' % message.qos)
	print('[MESSAGE] PAYLOAD:', message.payload.decode())

# Main function
def main():
	client = MQTT.Client(client_id=MQTT_CLIENT_NAME, clean_session=True)
```

```
	client.on_connect = on_connect
	client.on_message = on_message
	client.connect(MQTT_BROKER, MQTT_PORT)

	try:
		client.loop_forever()
	except KeyboardInterrupt:
		client.disconnect()
		print('----------------------------')
		print("[INFO] Client disconnected!")
		print("[INFO] Bye!")
		return

# Main function call
if __name__ == '__main__':
	main()
	sys.exit(0)
```

6.2.4 Python Code: MQTT Subscriber (Publishes MQTT measurements to database) (A.W.)

```
#!/usr/bin/env python3.6
import sys
import paho.mqtt.client as MQTT_Client
import paho.mqtt.publish as MQTT_Publish
from influxdb import InfluxDBClient
import json
import time

## MQTT CONNECTION DETAILS ##
MQTT_BROKER = "broker.hivemq.com"
MQTT_PORT = 1883
MQTT_CLIENT_NAME = "public_subscriber"
MQTT_REQ_CLIENT_NAME = "pub_req_hdlr"
MQTT_TOPIC_ROOT = "DT17"

## INFLUXDB CONNECTION DETAILS ##
INFLUXDB_HOST = "localhost"
INFLUXDB_PORT = "8086"
INFLUXDB_USER = "mercury"
INFLUXDB_PASS = "***********"
INFLUXDB_DB = "sdp"

def on_connect(client, userdata, flags, rc):
	client.subscribe(topic=MQTT_TOPIC_ROOT + "/#", qos=2)
	print('[INFO] Client connected', client._client_id.decode())

def on_message(client, userdata, message):
```

```
	print('------------------------------')
	# Check if the data received is a JSON object (aka check if the JSON decoder fails)
	try:
		data = json.loads(message.payload.decode())
	except json.decoder.JSONDecodeError as e:
		print("[WARNING] INVALID DATA OBJECT:", e)
		print("+--[WARNING] Object interpreted as string\n")
		data = message.payload.decode()

	# Check if the object is a list
	if not isinstance(data,(list,)):
		print("[WARNING] OBJECT NOT A LIST - DATA IGNORED:")
		print('+--[MESSAGE] Topic:', message.topic)
		print('+--[MESSAGE] Payload:', message.payload.decode())
		print('+--[MESSAGE] QoS: %d' % message.qos)

# If the format is correct (i.e. the values for measurements/keys)
	try:
		# If it contains the "measurement" key, it's verified and added as-is into the
database
		# The objects given to this are the same JSON format that InfluxDB accepts, so
they're very easy to verify
		if "measurement" in data[0]:
			# Add logs to "logs" measurement, data to "data", et cetera ad infinitum
			if (data[0]['measurement']) == "tank_env":
				print("Received tank environment data from
{}!".format(data[0]['tags']['id']))
			else:
				print("Received unexpected '{}' data from
{}!".format(data[0]['measurement'], data[0]['tags']['id']))

			# Create influxDB connection
			influx = InfluxDBClient(INFLUXDB_HOST, INFLUXDB_PORT, INFLUXDB_USER,
INFLUXDB_PASS, INFLUXDB_DB)

			influx.write_points(data)

except Exception as e:
		print("[WARNING] INVALID LIST STRUCTURE")
		print("+--[WARNING] Exception:", e)
		print('+--[MESSAGE] Topic:', message.topic)
		print('+--[MESSAGE] Payload:', message.payload.decode())
		print('+--[MESSAGE] QoS: %d' % message.qos)

# Main Function
def main():
	client = MQTT_Client.Client(client_id=MQTT_CLIENT_NAME, clean_session=False)
	client.on_connect = on_connect
	client.on_message = on_message
	client.connect(MQTT_BROKER, MQTT_PORT)

	try:
		client.loop_forever()
	except KeyboardInterrupt:
		client.disconnect()
```

```
            print('----------------------------')
            print("[INFO] Client disconnected!")
            print("[INFO] Bye!")
            return

def publish_response(response, serial):
      MQTT_Publish.single(
            topic=MQTT_TOPIC_ROOT + '/response/' + serial,
            payload=response,
            qos=2,
            hostname=MQTT_BROKER,
            port=MQTT_PORT,
            client_id=MQTT_REQ_CLIENT_NAME,
            tls={
                  'ca_certs': MQTT_CERT_PATH,
                  'cert_reqs': ssl.CERT_NONE,
                  'tls_version': ssl.PROTOCOL_TLSv1
            }
      )

# Main function call
if __name__ == '__main__':
      main()
      sys.exit(0)
```

6.2.5 Main Program Loop (A.W.)

```
/*****************************************************************
    main.c - The main program for the DT17 Creature Care project

    Purpose: The main C source file that runs the program and all of
             its features.

    Authors: Alexander Weber, (guys: add your names here)

    Copyright (c) 2019-2020. University of Akron Senior Design Team 17
*****************************************************************/

/*   Preprocessor Directives   */
#include <stdbool.h>
#include <stdint.h>
#include <math.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"

#include "CFAH2004AP.h"
#include "boot.h"
```

```
#include "control.h"

/*   Main Program   */

int main(void)
{
    systemBoot();

    while(1);
}
```

6.2.6 LCD Header File (A.W.)

```
/*****************************************************************
    CFAH2004AP_LCD.h - The header file for LCD functions

    Purpose: This file contains function headers for SSI write
             operations to control and send data to the LCD over SPI.

    Authors: Alexander Weber

    Copyright (c) 2019-2020. University of Akron Senior Design Team 17
*****************************************************************/

// Include guard for LCD header file
#ifndef HW_DRIVERS_CFAH2004AP_H_
#define HW_DRIVERS_CFAH2004AP_H_

// Define the SSI bus for the LCD
#define LCD_ADDRESS     SSI0_BASE

// Define a "safe" number of clock cycles to delay in between commands
// In actual practice, 1000 is NOT enough
#define LCD_PACKET_DELAY    2000

// Define common LCD function codes as available in the documentation
#define CLEAR_LCD       0x01
#define CURSOR_HOME     0x02
#define SET_LCD_MODE    0x06
#define DISPLAY_ON      0x0F
#define DISP_ON_NO_CUR  0x0C
#define SET_SPI_MODE    0x38

// Define DDRAM addresses for the LCD's four lines
#define LINE_ONE        0x80
#define LINE_TWO        0xC0
#define LINE_THREE      0x94
#define LINE_FOUR       0xD4

// The LCD is 4x20: 4 lines, 20 chars. Define the line size as 20 = 0x14
#define LINE_SIZE       0x14

// Define LCD function headers
```

```
void Initialize_LCD(void);
void LCDWriteChar(uint16_t chr);
void LCDWriteCmd(uint16_t cmd);
void LCDWriteLine(uint8_t size, uint8_t *line);
void demoLCD(void);

// SPI Initialization function
void initSPI(void);

#endif /* HW_DRIVERS_CFAH2004AP_H_ */
```

6.2.7 LCD Source File (A.W.)

```
/*****************************************************************
    CFAH2004AP_LCD.c - The source file for LCD functions

    Purpose: This file contains SSI write operations to control and
             send data to the LCD over SPI.

    Authors: Alexander Weber

    Copyright (c) 2019-2020. University of Akron Senior Design Team 17
*****************************************************************/

/*   Preprocessor Directives   */
#include <stdbool.h>
#include <stdint.h>
#include <math.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/ssi.h"

#include "CFAH2004AP.h"

// Fires up the ol' CFAH2004AP display
void Initialize_LCD() {
    // These functions are pretty self-explanatory
    LCDWriteCmd(CLEAR_LCD);
    LCDWriteCmd(CURSOR_HOME);
    LCDWriteCmd(SET_LCD_MODE);
    LCDWriteCmd(DISP_ON_NO_CUR);
    LCDWriteCmd(SET_SPI_MODE);

    // Start the LCD at the DDRAM address for line one
    LCDWriteCmd(LINE_ONE);

    // Delay a bit more since the LCD isn't always ready A$AP rocky:
    SysCtlDelay(LCD_PACKET_DELAY);
```

```
}

// Writes a character to the LCD
void LCDWriteChar(uint16_t chr){
    // Pull RS HIGH, (data register selected)
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_4, 0xFF);

    // Send the character to the LCD over the SPI bus
    SSIDataPut(LCD_ADDRESS, chr);

    // Wait for the SPI bus to become available again
    while(SSIBusy(LCD_ADDRESS));

    // Pull the RS bin back to LOW
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_4, 0x00);

    // Delay for a "safe" number of clock cycles
    SysCtlDelay(LCD_PACKET_DELAY);
}

// Sends a command to the LCD
void LCDWriteCmd(uint16_t cmd)
{
    // Pull RS LOW, (control register selected)
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_4, 0x00);

    // Send the character to the LCD over the SPI bus
    SSIDataPut(LCD_ADDRESS, cmd);

    // Wait for the SPI bus to become available again
    while(SSIBusy(LCD_ADDRESS));

    // Pull the RS pin HIGH (data register selected again)
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_4, 0xFF);

    // Delay for a "safe" number of clock cycles
    SysCtlDelay(LCD_PACKET_DELAY);
}

// Loops through a character array and prints each character to the screen
void LCDWriteLine(uint8_t size, uint8_t *line)
{
    uint8_t i;
    for (i = 0; i < size; i++)
    {
        // Write the next character to DDRAM
        LCDWriteChar(line[i]);
    }
}

// Initializes SPI for LCD communication
void initSPI(void)
{
    // Enable the SSI0 peripheral, and GPIO on ports A and B
    // The SSI0 peripheral is on Port A and pins 2,3,4 and 5.
```

```
    SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    // Set up the system clock, and set the SPI mode
    uint32_t clockk=SysCtlClockGet();
    SysCtlClockGet();
    SSIConfigSetExpClk(LCD_ADDRESS, SysCtlClockGet(), SSI_FRF_MOTO_MODE_0,
SSI_MODE_MASTER, 100000, 8);

    // This function/s configures the pin muxing on port A pins 2,3,4 and 5
    GPIOPinConfigure(GPIO_PA2_SSI0CLK);
    GPIOPinConfigure(GPIO_PA3_SSI0FSS);
    //GPIOPinConfigure(GPIO_PA4_SSI0RX); // We're not using an RS pin because we're
only sending data
    GPIOPinConfigure(GPIO_PA5_SSI0TX);
    SysCtlDelay(4);

    // Configures the pins for use by the SSI; give the pins on Port A for SSI
    GPIOPinTypeSSI(GPIO_PORTA_BASE, GPIO_PIN_5 | GPIO_PIN_3 | GPIO_PIN_2);
    SysCtlDelay(4);

    // Configures the SSI port for SPI master mode, it takes 6 parameters
    //SSIConfigSetExpClk(SSI0_BASE, 2000000 , SSI_FRF_MOTO_MODE_3, SSI_MODE_MASTER,
1600 , 8 );

    GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_4);
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_4, 0x00);
    // Enables the SSI0 module
    SSIEnable(LCD_ADDRESS);
    SysCtlDelay(4);
    uint32_t scrap;
    while(SSIDataGetNonBlocking(LCD_ADDRESS, &scrap));
}

// Displays demo text on the LCD
void demoLCD(void) {
    // Example of writing a  00000000001111111111
    // 20 chars to the LCD   01234567890123456789
    LCDWriteLine(LINE_SIZE, "** Univ. of Akron **");

    LCDWriteCmd(LINE_TWO);
    LCDWriteLine(LINE_SIZE, "** Sr. Des. Proj. **");

    LCDWriteCmd(LINE_THREE);
    LCDWriteLine(LINE_SIZE, "**    TEAM #17    **");

    LCDWriteCmd(LINE_FOUR);
    LCDWriteLine(LINE_SIZE, "**  CreatureCare  **");
}
```

6.2.8 BME680 Sensor Header Stub (A.W.)

```
/*****************************************************************
    BME680.h - The header file for BME680 sensor functions

    Purpose: This file contains function headers for I2C communication
             operations to read temperature, humidity, and air
             quality data from the BME680 sensor.

    Authors: Alexander Weber

    Copyright (c) 2019-2020. University of Akron Senior Design Team 17
*****************************************************************/

#ifndef HW_DRIVERS_BME680_H_
#define HW_DRIVERS_BME680_H_

// Define LCD function headers
int Initialize_BME680(void);


#endif /* HW_DRIVERS_BME680_H_ */
```

6.2.9 BME680 Sensor Source Stub (A.W.)

```
/*****************************************************************
    BME680.c - The source file for BME680 sensor functions

    Purpose: This file contains functions for I2C communication
             operations to read temperature, humidity, and air
             quality data from the BME680 sensor.

    Authors: Alexander Weber

    Copyright (c) 2019-2020. University of Akron Senior Design Team 17
*****************************************************************/

#include "BME680.h"

// Attempt communication with the BME680 sensor via I2C
int Initialize_BME680(void) {
    // WIP - not yet implemented
    return 1;
}
```

6.2.10 Boot Header File (A.W.)

```
/**********************************************************************
    boot.h - The header file for boot functions

    Purpose: This file contains function headers for the boot
             sequence operations.

    Authors: Alexander Weber

    Copyright (c) 2019-2020. University of Akron Senior Design Team 17
**********************************************************************/

// Include guard for boot driver
#ifndef SW_DRIVERS_BOOT_H_
#define SW_DRIVERS_BOOT_H_

// Define the RGB LED colors (Port F, pins 1, 2, and 3
#define RED_LED   GPIO_PIN_1
#define BLUE_LED  GPIO_PIN_2
#define GREEN_LED GPIO_PIN_3

// Define a base period for the blink function. About 0.25 sec
#define BLINK_PER 500000

// Define boot function headers
void systemBoot(void);
void initializeLED(void);
void setStatusLED(int color, int period, int cycles);


#endif /* SW_DRIVERS_BOOT_H_ */
```

6.2.11 Boot Source File (A.W.)

```
/**********************************************************************
    boot.c - The source file for boot functions

    Purpose: This file contains functions involved in the boot sequence

    Authors: Alexander Weber

    Copyright (c) 2019-2020. University of Akron Senior Design Team 17
**********************************************************************/

/*   Preprocessor Directives   */
#include <stdbool.h>
#include <stdint.h>
#include <math.h>
#include "inc/hw_memmap.h"
```

```
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/ssi.h"

#include "boot.h"
#include "CFAH2004AP.h"
#include "BME680.h"
#include "control.h"

// Boot the entire system, initialize all subsystems
void systemBoot(void) {
    // Enable the built-in LED
    initializeLED();

    // Do a boot indicator self-test of the LED
    setStatusLED(GREEN_LED, 0, 3);

    // Enable the system clock, SPI communication, and LCD functions
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);
    initSPI();
    Initialize_LCD();

    // Since the LCD is initialized, print the demo screen
    demoLCD();

    // Blink for a bit, just for fun
    setStatusLED(BLUE_LED, 1, 7);

    // Try to initialize the BME680 sensor, report failure if unsuccessful
    int status = 0;
    status = Initialize_BME680();
    if (status) {
        setError(ERROR_SENSOR);
    }
}

// Initialize the built-in LED (PORTF)
void initializeLED(void) {
    // Enable GPIO Port F
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    // The RGB LED is on pins 1, 2, and 3
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, RED_LED | BLUE_LED | GREEN_LED);
}

// Function to blink the LED given a color and delay
void setStatusLED(int color, int period, int cycles) {
    // Bit shift the default delay period by the multiplier given
    long per = BLINK_PER << period;

    // Blink the LED for the given number of cycles
```

```
    int i;
    for (i = 0; i < cycles; i++) {
        GPIOPinWrite(GPIO_PORTF_BASE, color, 0xFF);
        SysCtlDelay(per);

        GPIOPinWrite(GPIO_PORTF_BASE, color, 0x00);
        SysCtlDelay(per);

    }
}
```

6.2.12 Main Control Header File (A.W.)

```
/*****************************************************************
    control.h - The header file for main control functions

    Purpose: This file contains function headers for the general
             control functions.

    Authors: Alexander Weber

    Copyright (c) 2019-2020. University of Akron Senior Design Team 17
*****************************************************************/

// Include guard for the main control driver
#ifndef SW_DRIVERS_CONTROL_H_
#define SW_DRIVERS_CONTROL_H_

// Define the various error states
#define ERROR_NONE      0x00
#define ERROR_SENSOR    0x01
#define ERROR_WIFI      0x02
#define ERROR_MQTT      0x03
#define ERROR_STATES    0x04


// Define the default blink period for the error states
#define ERROR_BLINK     1

// Define control function headers
void setError(int error);
void getErrorText(uint8_t* error_text, int error);

#endif /* SW_DRIVERS_CONTROL_H_ */
```

6.2.13 Main Control Source File (A.W.)

```
/*****************************************************************
    control.c - The source file for main control functions

    Purpose: This file contains the general control functions
             for the operation of the device.

    Authors: Alexander Weber

    Copyright (c) 2019-2020. University of Akron Senior Design Team 17
*****************************************************************/

/*   Preprocessor Directives   */
#include <stdbool.h>
#include <stdint.h>
#include <math.h>
#include <stdio.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"

#include "control.h"
#include "boot.h"
#include "CFAH2004AP.h"

// Sets a given error bit and halts the system
// Displays the error on the LCD if possible
void setError(int error) {
    // Display the error text on the LCD

    // Write the error text  00000000001111111111
    // to the LCD            01234567890123456789
    LCDWriteCmd(CLEAR_LCD);
    LCDWriteCmd(CURSOR_HOME);

    LCDWriteCmd(LINE_ONE);
    LCDWriteLine(LINE_SIZE, "**  CreatureCare  **");

    LCDWriteCmd(LINE_TWO);
    LCDWriteLine(LINE_SIZE, "** System Failure **");

    LCDWriteCmd(LINE_THREE);
    LCDWriteLine(14, "** Error Code ");
    LCDWriteChar((char) error + 0x30);
    LCDWriteLine(5, ":  **");


    // Create the error string buffer as a one-line length object
    uint8_t error_text[LINE_SIZE];
    getErrorText(error_text, error);
```

```
    LCDWriteCmd(LINE_FOUR);
    LCDWriteLine(LINE_SIZE, error_text);

    // Blink the error LED and halt the system
    long pause = BLINK_PER << ERROR_BLINK;
    while(1) {
        // Blink the red LED according to the error number
        // This way the error is known even if the LCD is unavailable
        setStatusLED(RED_LED, ERROR_BLINK, error);
        SysCtlDelay(pause);

    }
}

// Return the error text as a uint8_t string for the LCD to display
void getErrorText(uint8_t* error_text, int error) {
    // Dump the appropriate error string into the buffer
    switch(error)
    {
    case ERROR_SENSOR:
        sprintf((char *)error_text, "** Sensor Failure **");
        break;

    case ERROR_WIFI:
        sprintf((char *)error_text, "** Network Offline**");
        break;

    case ERROR_MQTT:
        sprintf((char *)error_text, "**  MQTT Timeout  **");
        break;

    case ERROR_NONE:
    default:
        sprintf((char *)error_text, "**    No error    **");
        break;
    }
}
```

6.2.14 Live Monitoring Graphic (A.W.)

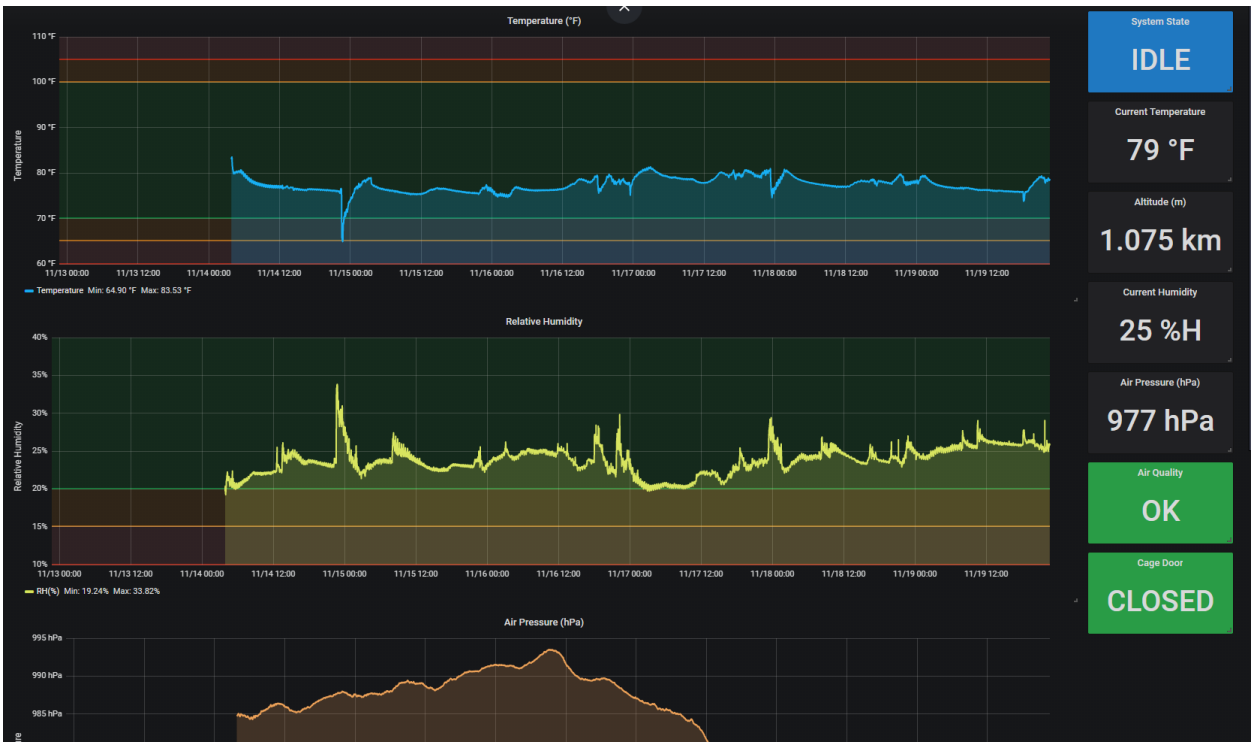

Figure 18: Live monitoring graphic

7. Financial Budget (C.A.)

Table 39: Financial Cost (Original Budget)

Product	**Cost**	**Quantity**
Serial WiFi Transceiver	$6.95	2
Humidity and Temp Sensor	$3.82	3
Cage	$38.24	1
Daylight Bulb (100W)	$9.59	1
Nightlight Bulb (50W)	$11.99	1
Heat Lamp Enclosure	$37.79	2
Ventilation Fan	$7.99	1
Air Quality / Gas Sensor Breakout	$22.50	1
Mini Push Button Switch (5 pcs)	$1.35	1
Optical Infrared Water Liquid Level Sensor	$8.80	1
3-24V Piezo Buzzer	$5.99	2
20x4 SPI Character LCD Module	$14.64	1
1-Channel Power Relay with Optoisolator	$1.95	4
Voltage Regulator Linear Positive Fixed 1 Output 5V 1.5A	$6.99	1
3.3V 950mA, Voltage Regulator, LD33V	$6.40	1
DC Power Supply 24 Volt 5A 120W Power Adapter 100V~240V AC to DC Converter	$18.21	1

Table 40: Financial Budget (Actual Cost)

Product	**Cost**	**Quantity**
Serial WiFi Transceiver	$6.95	2
Humidity and Temp Sensor	$3.82	3
Daylight Bulb (100W)	$9.59	1
Nightlight Bulb (50W)	$11.99	1
Heat Lamp Enclosure	$37.79	2
Ventilation Fan	$7.99	1
Air Quality / Gas Sensor Breakout	$22.50	1
Mini Push Button Switch (5 pcs)	$1.35	1
3-24V Piezo Buzzer	$5.99	2
20x4 SPI Character LCD Module	$14.64	1
1-Channel Power Relay with Optoisolator	$1.95	4
Voltage Regulator Linear Positive Fixed 1 Output 5V 1.5A	$6.99	1
3.3V 950mA, Voltage Regulator, LD33V	$6.40	1
DC Power Supply 24 Volt 5A 120W Power Adapter 100V~240V AC to DC Converter	$18.21	1

8. Team Information (C.A.)

- Cassie Allender, Electrical Engineering, Project Lead
- Shane Benner, Electrical Engineering, Hardware Manager
- Alexander Weber, Computer Engineering, Software Manager
- Kim Wilson, Computer Engineering, Archivist

9. Project Schedule (C.A.)

Midterm Report	40 days	Fri 8/30/19	Wed 10/9/19		
Cover page	39.38 days	Fri 8/30/19	Tue 10/8/19		Cassie
T of C, L of T, L of F	39.38 days	Fri 8/30/19	Tue 10/8/19		Cassie
Problem Statement	**40 days**	**Fri 8/30/19**	**Wed 10/9/19**		
Need	39.38 days	Fri 8/30/19	Tue 10/8/19		Cassie
Objective	39.38 days	Fri 8/30/19	Tue 10/8/19		Alex
Background	39.38 days	Fri 8/30/19	Tue 10/8/19		Alex,Shane,Cassie
Marketing Requirements	39.38 days	Fri 8/30/19	Tue 10/8/19		Alex,Shane
Engineering Requirements Specification	40 days	Fri 8/30/19	Wed 10/9/19		
Engineering Analysis	**40 days**	**Fri 8/30/19**	**Wed 10/9/19**		
Circuits (DC, AC, Power, ...)	39.38 days	Fri 8/30/19	Tue 10/8/19		Cassie
Electronics (analog and digital)	39.38 days	Fri 8/30/19	Tue 10/8/19		Shane
Signal Processing	39.38 days	Fri 8/30/19	Tue 10/8/19		Kim
Communications (analog and digital)	39.38 days	Fri 8/30/19	Tue 10/8/19		Alex
Electromechanics	39.38 days	Fri 8/30/19	Tue 10/8/19		Shane
Computer Networks	39.38 days	Fri 8/30/19	Tue 10/8/19		Alex
Embedded Systems	39.38 days	Fri 8/30/19	Tue 10/8/19		Kim
Accepted Technical Design	**40 days**	**Fri 8/30/19**	**Wed 10/9/19**		
Hardware Design: Phase 1	**39.38 days**	**Fri 8/30/19**	**Tue 10/8/19**		**Cassie,Shane**
Software Design: Phase 1	**39.38 days**	**Fri 8/30/19**	**Tue 10/8/19**		**Cassie,Alex**
Mechanical Sketch	39.38 days	Fri 8/30/19	Tue 10/8/19		Cassie
Team information	39.38 days	Fri 8/30/19	Tue 10/8/19		Cassie
Project Schedules	**40 days**	**Fri 8/30/19**	**Wed 10/9/19**		**Cassie**
References	39.38 days	Fri 8/30/19	Tue 10/8/19		Cassie
Hardware Design	**62.38 days**	**Fri 8/30/19**	**Thu 10/31/19**		
Dimmer switch layout	55.38 days	Fri 8/30/19	Thu 10/24/19		Cassie
Sensor Communication	62.38 days	Fri 8/30/19	Thu 10/31/19		Shane
Display Unit Layout	55.38 days	Fri 8/30/19	Thu 10/24/19		Shane
Inner Controller Communication	62.38 days	Fri 8/30/19	Thu 10/31/19		Shane,Cassie
Build air quality sensor	62.38 days	Fri 8/30/19	Thu 10/31/19		Cassie
Software Design	**62.38 days**	**Fri 8/30/19**	**Thu 10/31/19**		
Sensor Data Processing	62.38 days	Fri 8/30/19	Thu 10/31/19		Kim
Data Scraping	62.38 days	Fri 8/30/19	Thu 10/31/19		Alex
Wifi Communication	55.38 days	Fri 8/30/19	Thu 10/24/19		Alex
Display Communication	55.38 days	Fri 8/30/19	Thu 10/24/19		Kim
Sensor Communication	55.38 days	Fri 8/30/19	Thu 10/24/19		Alex

Figure 19: Gantt Chart (Design Phase)

Creature Cage

DT17
Cassie Allender

Project Start Date:	4/23/2020
Scrolling Increment:	1

Milestone Description	Category	Assigned To	Progress	Start	No. Days
Hardware Implementation					
Voltage Regulator Circuit	Goal	Cassie	25%	4/23/2020	3
Test Light/Fan Control	Goal	Cassie/Alex	75%	4/30/2020	7
Implement Buzzer/Switch Circuit	Goal	Cassie	100%	4/23/2020	7
Solder Components	Goal	Cassie/Alex	75%	4/30/2020	1
Test Hardware for Demonstration	Goal	Cassie/Alex	25%	5/7/2020	6
Software Implementation					
Program LCD	Goal	Alex	100%	4/23/2020	13
Program BME680	Goal	Kim	0%	4/23/2020	9
Test WiFi Module	Goal	Alex	50%	4/23/2020	14
Test Software for Demonstration	Goal	Alex	50%	4/23/2020	6

Figure 20: Gantt Chart (Implementation Phase)

Creature Cage

DT17
Cassie Allender

Project Start Date:	4/23/2020
Scrolling Increment:	1

Milestone Description	Category	Assigned To	Progress	Start	No. Days
Hardware Implementation					
Voltage Regulator Circuit	Goal	Cassie/Alex	100%	4/23/2020	3
Test Light/Fan Control	Goal	Shane	75%	4/30/2020	7
Implement Buzzer/Switch Circuit	Goal	Cassie	95%	4/23/2020	7
Solder Components	Goal	Cassie/Alex	100%	4/30/2020	1
Test Hardware for Demonstration	Goal	Cassie/Alex	25%	5/7/2020	6
Software Implementation					
Program LCD	Goal	Alex	100%	4/23/2020	13
Program BME680	Goal	Kim	0%	4/23/2020	9
Test WiFi Module	Goal	Alex	50%	4/23/2020	14
Test Software for Demonstration	Goal	Alex	50%	4/23/2020	6

Figure 21: Gantt Chart (Actual)

10. Conclusions and Recommendations (C.A.)

This design team faced a large tragedy after the loss of an incredible person, engineer, classmate, and friend, Shane Benner and unfortunately, the team did not have the opportunity to finish the project in his memory. The team dynamic in some ways worked well, and in other ways did not. With Shane and Alex being extremely knowledgeable with skills beyond what is taught in the classroom, it was a great learning experience for the rest of the team. With a team of three, the project was then simplified, so the future of the project looked promising. The number one recommendation for other students is to learn how to hold teammates accountable, and for faculty to give more direction on how to handle difficult situations.

11. References (C.A.)

[1] Pough, Harvey F., “Recommendations for the Care of Amphibians and Reptiles in Academic Institutions”, *ILAR Journal*, Volume 33, Issue 4, 01 October 1991 <https://academic.oup.com/ilarjournal/article/33/4/S1/744425>

[2] Patent-Greenwood, John Brady, *Comprehensive Reptilian Environment Control System*, Patent Number: 5,799,614, Date of Patent: 01 September 1998

[3] Patent-Azpurua, Diane M., Azpurua, Francisco J., *Reptile Cage Apparatus*, Patent Number: 5,010,845, Date of Patent: 30 April 1991

[4] Kaplan, Melissa, “Microclimates For Your Reptiles”, *Herp Care Collection*, 01 January 2014

[5] Kaplan, Melissa, “Winter Advisory”, *Herp Care Collection*, 01 January 2014

[6] El Amraoui, Lilia, Khediri, Jalel, Lachheb, Aymen, “Control of linear switched reluctance motor for sliding door application”, IEEE, 14-17 January 2017

[7] Patent-Hughes, Kenneth D., *Temperature Controlled Containers*, Patent Number: US 2010/0218728, Date of Patent: 02 September 2010

[8] Patent-Eckel, David, Bonasia, Gaetano, Porter, James A., *Network based multiple sensors and control device with temperature sensing and control*, Patent Number: US 6,798,341 B1, Date of Patent: 28 September 2004

[9]Patent-Richards, John H., Kolarovic, Ronald S., *Humidity Sensor for Incubator*, Patent Number: US 6,711,937 B2, Date of Patent: 30 March 2004

[10] *Tiva TM4C123GH6PM Microcontroller Datasheet*, 12 June 2014, <http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf>

[11] *I2C Info-I2C Bus, Interface and Protocol*, <https://i2c.info/i2c-bus-specification>

[12] "UM10204", *I2C-bus specification and user manual*, 04 April 2014, <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>

[13] Maxim Integrated, *1-Wire Communication through Software*, 30 May 2002, <https://www.maximintegrated.com/en/design/technical-documents/app-notes/1/126.html>

[14] Expressif Systems, *ESP8266EX Datasheet*, August 2019, <https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf>

[15] Oasis, *MQTT Version 5.0 Oasis Standard*, 07 March 2019, <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>