

Senior Design Project Document
Due: December 3rd, 2018

Smart Garden Controller



Department of Electrical Engineering and Computer Science

University of Central Florida

Dr. Lei Wei and Dr. Richie

Sponsored by: self-sponsored

Group 12

Alexander Burns	Electrical Engineering
Geovanny Chirino	Electrical Engineering
Temple Corson	Computer Engineering
Renan Coelho Silva	Computer Engineering

Table of Contents

List of Figures	vi
List of Tables	vii
1.0 Executive Summary.....	1
2.0 Project Description.....	2
2.1 Project Background.....	2
2.2 Motivation.....	2
2.3 Goals/Objectives	3
2.4 Requirements Specifications.....	5
2.5 Marketing and Engineering Requirements	7
3.0 Research.....	13
3.1 Existing Projects and Products.....	13
3.1.1 Existing Projects	13
3.1.2 Existing Products	14
3.2 Relevant Technologies	17
3.2.1 Communication.....	17
3.2.2 Sensors	20
3.2.3 Electronic Valve / Flow Control.....	21
3.2.4 DC-DC Power Management	22
3.2.5 Rechargeable Battery Management	22
3.2.6 Software	22
3.2.7 Digital Data Storage	23
3.2.8 Liquid Crystal Display (LCD).....	24
3.2.9 Smart Speaker	24
3.3 Strategic Components and Part Selections.....	24
3.3.1 Microcontroller	25
3.3.2 Battery Management	29
3.3.3 Sensors	32
3.3.4 Water Flow Controller	41
3.3.5 LCD Screen Module.....	42
3.3.6 Smart Speaker	43
3.4 Possible Architecture and Related Diagrams.....	43

3.4.1 Design Choices	44
3.5 Part Selection Summary.....	46
4.0 Related Standards and Realistic Design Constraints.....	47
4.1 Standards	47
4.1.1 Wi-Fi (802.11) Standards	47
4.1.2 Power Supply Standards	48
4.1.3 AC Power Supply Standards.....	49
4.1.4 Sensor Interface & I2C Standards	49
4.1.5 C Programming Standards	50
4.1.5 Java Programming Standards.....	51
4.1.6 TypeScript Programming Standards	51
4.1.7 Html Programming Standards.....	52
4.2 Realistic Design Constraints	53
4.2.1 Economic and Time Constraints	53
4.2.2 Environmental, Social, and Political Constraints	55
4.2.3 Ethical, Health, and Safety Constraints.....	57
4.2.4 Manufacturability and Sustainability Constraints.....	59
5.0 Hardware Design Details.....	61
5.1 Initial Design Architectures and Related Diagrams.....	61
5.2 Microcontroller	64
5.2.1 CC3220MODA Power	65
5.2.2 CC3220MODA No Connects.....	66
5.2.3 CC3220MODA JTAG	66
5.2.4 CC3220MODA Reset	67
5.2.5 CC3220MODA Configuration Pins.....	68
5.2.6 CC3220MODA FLASH Serial Peripheral Interface (FLASH_SPI).....	69
5.2.7 CC3220MODA GPIOs.....	70
5.2.8 CC3220MODA GPIO Pin Multiplexing	70
5.3 Sensor Interface.....	72
5.3.1 Moisture, Temperature, & Ambient Light	72
5.3.2 Wind Speed.....	73
5.3.3 Humidity & Pressure	74
5.4 LCD and Pushbutton Interface.....	74

5.5 Water Flow Control Subsystem	75
5.6 Power Subsystem.....	77
5.7 Summary of Hardware Design	78
6.0 Software Design Details	80
6.1 Database	81
6.1.1 Database Tables	83
6.2 Microcontroller Software	84
6.2.1 Micro Controller Logic and Functions	85
6.2.2 Micro Controller Software Flow Diagram	87
6.3 Web Server.....	88
6.3.1 Web Server Methods Overview	90
6.4 Web Application.....	91
6.4.1 Development Environment & Tools.....	92
6.4.2 App Component & Module.....	93
6.4.3 Log In Component.....	93
6.4.4 User Component	94
6.4.5 Data Component.....	95
6.4.6 Schedule Component.....	96
6.4.7 On-Demand Component.....	97
6.5 Smart Speaker Integration	97
7.0 Project Prototype Construction and Coding	99
7.1 Integrated Schematics	99
7.1.1 Power Schematic.....	99
7.1.2 Sensor and User Interface Schematic	102
7.1.2 Solenoid Flow Control Schematic	106
7.1.3 External Flash Memory Schematic.....	107
7.1.4 Programming and Debug Schematic.....	108
7.1.5 Microcontroller Module.....	110
7.2 PCB Vendor and Assembly	111
8.0 Project Prototype Testing Plan	114
8.1 Hardware Test Environment	114
8.2 Hardware Specific Testing.....	116
8.2.1 Sensor Testing.....	116

8.2.2 LCD Testing.....	119
8.2.3 Valve Switch Testing	120
8.2.4 Microcontroller Testing	121
8.2.5 Chassis Testing	122
8.2.6 Miscellaneous Hardware Testing.....	123
8.3 Software Test Environment	123
8.4 Software Specific Testing	123
8.4.1 Database Software	124
8.4.2 Web Server Software	124
8.4.3 MCU Software	125
8.4.4 Web Application.....	127
8.4.5 Smart Speaker Integration	127
9.0 Administrative Content.....	129
9.1 Milestone Discussion	129
9.2 Budget and Finance	133
9.2.1 Sponsorship.....	134
Appendices.....	136
Appendix A: Image Permissions.....	136
Citations	141

List of Figures

Figure 1 - Smart Controller Mockup, Image by Author	5
Figure 2 - House of Quality	8
Figure 3 - CC3220MODA Module Courtesy of TI	27
Figure 4 - Soil Moisture Sensor	33
Figure 5 - Temperature Sensor Data.....	34
Figure 6 - Soil Moisture Sensor Data.....	35
Figure 7 – Anemometer Courtesy of Adafruit	37
Figure 8 – BME280 Courtesy of Adafruit	40
Figure 9 - LCD Module.....	42
Figure 10 - Optional Zone Layout 1.....	44
Figure 11 - Optional Zone Layout 2.....	45
Figure 12 - System Architecture.....	64
Figure 13 - CC3220MODA Block Diagram	72
Figure 14 - High Side Switch Configuration.....	77
Figure 15 - Web Interface Block Diagram	80
Figure 16 - Database Configuration	84
Figure 17 - Initial Microcontroller Software Flow Diagram	88
Figure 18 - Software Communication Diagram.....	90
Figure 19 – Log In Component Example	94
Figure 20 – User Component Example	94
Figure 21 – Data Component Example 1	95
Figure 22 – Data Component Example 2	96
Figure 23 – Schedule Component Example	96
Figure 24 – On-Demand Component Example	97
Figure 25 - Main Board Schematic	99
Figure 26 - Power Input Schematic.....	100
Figure 27 - 12VDC Power Schematic.....	100
Figure 28 - 5VDC Power Schematic.....	101
Figure 29 - 3.3VDC Power Schematic.....	102
Figure 30 - LCD and Anemometer Interface Schematic.....	103
Figure 31 - Soil Moisture and Humidity Sensor Interface Schematic	104
Figure 32 - User Interface Pushbuttons Schematic.....	105
Figure 33 - High Side Switch Configuration Schematic.....	106
Figure 34 - Flash Memory Schematic.....	107
Figure 35 - JTAG Header Schematic	108
Figure 36 - USB to UART Schematic	109
Figure 37 - Microcontroller Schematic	110
Figure 38 - Moisture Level Test Data	117
Figure 39 – Anemometer Voltage Test	119
Figure 40 – LCD Power up test.....	120

List of Tables

Table 1 - Requirements Table	7
Table 2 - Microcontroller Trade Analysis	29
Table 3 - Battery Chemistry Trade Analysis	31
Table 4 – Soil Moisture Sensor Data	36
Table 5 – Humidity and Pressure Sensor Data.....	37
Table 6 - Anemometer Data.....	38
Table 7 – HTU21D-F Sensor Data	39
Table 8 – BMP280 Sensor Data.....	39
Table 9 - BME280 Additional Data	40
Table 10 - LCD Module Data	43
Table 11 - Part Selection Summary	46
Table 12 - Power Supply Standards	49
Table 13 - I2C Standards	50
Table 14 - CC3220MODA Power Specs	66
Table 15 - CC3220MODA SOP Config.....	69
Table 16 - Database Field Tables	82
Table 17 - MCU Functions.....	87
Table 18 - Class Functionality Table	91
Table 19 – Valve current consumption.....	121
Table 20 – Estimated Cost.....	133
Table 21 – Final Bill of Materials.....	134
Table 22 – Guard Dog Valves Data.....	135

1.0 Executive Summary

Cultivating a garden is a common past time in the United States. Home gardening can be relaxing and provides fresh fruits and vegetables. Planting your own fruits and vegetables provides a certain freedom from depending in others for your food. There is a certain pride in eating something you have grown yourself. Other benefits of gardening include immune regulation, brain health, dexterity, heart health, and mental health. With such benefits, at a low cost it is a surprise more people do not have a home garden. However, gardening is not all fun. It can take a considerable amount of time to know how much water your plants need and how often they need water. Having to actually go outside and water the plants is actually worse than doing the research of how much watering your plants need, and if not properly watered your plants will die. These two previously mentioned cons are two of the strongest reasons some people decide not to have a home garden. The smart controller aims at solving these issues and providing metrics based on data collected by various sensors.

Watering the garden as often as possible and for a long time may seem like an easy solution. However, different varieties of plants require different amounts of water. Overwatering could drown the plants. Roots need oxygen and not enough oxygen in the soil could kill your plants. Not enough oxygen will be present if there is too much water in the soil. To make matters more intricate, the amount of watering required may change depending on environmental factors such as temperature and air moisture levels. The soil may dry up faster if the air is dry and hot. Not taking into consideration weather forecasts can lead to watering the plants in days which would rain. A lot of water could be wasted when trying to water the plants too often and not taking into consideration the weather forecast. Therefore, water as well as money could be wasted. The smart garden controller would help conserve your resources.

Our product, the smart garden controller, will provide a set of features which we believe will make it easier for any person to grown their own garden. Sensors for moisture will regularly upload data to the cloud for review and letting the gardener know if their plants got enough water. Wind data will be recorded to provide weather insight. Sunlight exposition sensors will help the gardener determine a proper location for the garden where the plants get enough light. Temperature information gathered through sensors will be available and helpful in determining how long to water the plants. In all, the various sensors and the data collected will guide the gardener in growing their plants. Therefore, a smart garden controller would be a vital addition to a personal garden. The expectation is that adding technology to home gardening would help the gardener focus on the parts of gardening which are rewarding while the system takes care of the negatives.

2.0 Project Description

2.1 Project Background

According to a special report by the National Gardening Association completed in 2014, “the number of households participating in food gardening from 2008 to 2013 grew from 36 million households to 42 million households. That’s an overall increase in participation of 17% in 5 years and a compound annual growth rate of 3% a year.” (“Garden to Table.” *Garden.org*, garden.org/special/pdf/2014-NGA-Garden-to-Table.pdf) Newer reports show even greater numbers growing food at home. Watering the plants is a common problem to all such home farmers.

This project describes the smart garden controller. The smart garden controller is a solution to the work involved in watering a home garden or potentially a small farm. The smart garden controller also collects data regarding soil moisture, temperature, and light. The data collected by the smart garden controller is useful in determine how often to water the garden. The project details the motivation, requirements, similar products, design, specifications, objectives and any relevant detail to the solution proposed to the problem.

2.2 Motivation

The motivation for this project can be considered from an experience, financial, and personal interest perspective. The motivation of this project as far as experience is that it puts to use a lot of the theoretical knowledge learned in college. This project should help the group to learn and understand realistic work practices of getting a product from design to reality. The project should build upon the subjects learned in previous classes and labs. The knowledge and experience learned in this project should also be helpful in getting into a career in the field of interest. The project is also paramount experience to be shared in potential interviews.

From a financial perspective, this project could lead to a product which could prove to be profitable. Per the business insider website, the Do It Yourself yard and garden industry continues to grow at a steady rate and as of April 2017 represented a \$37-billion-dollar sector. Current products in the market which provide high end features can cost over \$100. Therefore, a profit could be turned by creating a product which could provide some of the high-end features at a lower price point.

Finally, from a personal experience perspective, some of the members of the group have long been interested in home gardening. The members have been interested if it is financially worth to have a home garden. Also, one of the group members has been interested in growing plants which are original to its country of origin and not readily available in the United States.

2.3 Goals/Objectives

The Goal of this project is to design and build a smart garden controller which will help home gardeners ensure that their plants thrive with the right amount of irrigation provided. In addition, the system should also provide important data to the user which will be helpful in tuning the watering settings of the device. The controller should be affordable when compared to the competition while providing similar features.

The primary goal for the smart garden controller is to take away the manual labor involved in watering plants and automate the watering maintenance. The smart garden controller will allow the gardener to set a schedule for each zone of the garden independent of one another. The watering will be triggered and the soil irrigated until a threshold is met. Irrigation will stop once the threshold is met. The gardener will be able to control how often to trigger the irrigation as well as for how long to water the plants. Moisture sensors in the soil will send data back to the system and automatically determine whether watering based on the user schedule is necessary. Based on the metrics for the garden, the user will be able to adjust how often the different zones get watered while taking into account environmental factors such as rainfall and temperature.

The watering needs will not always be the same since the weather changes throughout the year. Depending on location, there may be times throughout the year when irrigation will not be necessary and water from the rain alone will be enough. The user will be able to set thresholds in the system and decide what to do if such thresholds are met. Using thresholds, irrigation may be skipped when the soil is already wet enough due to natural resources. Using such setting will allow for the system to save water and money to the garden owner. The efficiency and potential cost savings associated with the elimination of unnecessary watering, as well as the likelihood of a higher crop yield are just two of the main business cases for this device.

Another goal of the smart garden controller is to shorten the research time a user would spend in determining how often to water a plant. For example, strawberries have different watering needs than tomatoes. A set of parameters would be provided which the user could pick from for a variety of fruits and vegetables. The different parameters would be available to the end user via a web application. Through the web application the user should be able to edit the smart garden controller's schedule and settings. The end user would simply choose the schedule based on what is being planted and let the smart garden take care of the irrigation. Additional schedules would be provided by a community of users and the number of schedules would grow over time. The user would not be limited to the watering schedules provided out of box and would be able to modify the schedules to meet its needs. Community input will be very important. Instead of trial and error the user could look up in the community resource for other people planting in similar location and weather. The long-term theory is that users would be able to upload

their own schedules and profiles for different garden items, and the collective community would be able to rate and offer feedback, ultimately garnering an environment where the most successful profiles become easily evident. This increases the ease of use for the consumer to have a successful personal garden that does not require much maintenance.

The smart garden controller will also collect metrics throughout the day such as moisture, temperature, and light level data. The data will be available through the web application where it may be viewed or exported for download. There will be no personally identifiable information on the server or health related data, therefore, data will not need to be protected according to HIPAA (Health Insurance Portability and Accountability Act) standards. Nonetheless, the remote server will have basic authentication enabled to protect user data. Review of the data will be useful in determining how long the watering system should be kept on for each zone. For example, review of the data could point that at w temperature it takes x minutes for the soil to dry. Based on the metrics collected, it could be determined instead that the watering valve should be on for y minutes when the temperature is at z. These metrics will eventually be available to the entire user community so that the learned metrics of one can benefit many. The hope is that eventually there will be a user feedback system where users are able to grow their crop during each season according to certain profiles and at the end of the season the user can upload feedback about crop yield or issues that occurred throughout the season. This feedback paired with the weather metrics collected could help improve the profiles performance once the data is analyzed.

The smart garden controller will be light weight and relatively portable, although it is mainly intended for stationary operation. The smart garden controller will be easy to use and configured via a web interface. The smart garden controller will also be configurable via a display and physical buttons on the device. The smart garden controller will be able to control multiple sprinklers and therefore allow for the garden to grow in size over time. Finally, the smart garden controller will be affordable. Similar system can range from \$100 with basic features (4 zone watering system scheduler, no metrics, hoses, sprinklers, connectivity) to \$3,000 with more features (Professionally installed sprinkler system that does not include soil and weather feedback). The smart garden controller should provide more than basic feature and allow for proper watering while at a lower price tag than the high-end sprinkler systems currently available.

The smart controller will be built at a price point lower than what the competition sells their controllers for. With a price point lower than the competition, the controller could be sold at online retailers for a profit. Therefore, in summary, the goal of the project is to create a device which solves a problem, is built at a lower cost than the competition to allow for a profit and provides the students with real life experience. The following image created on draw.io pictures our first design blueprint for what the controller should look like.

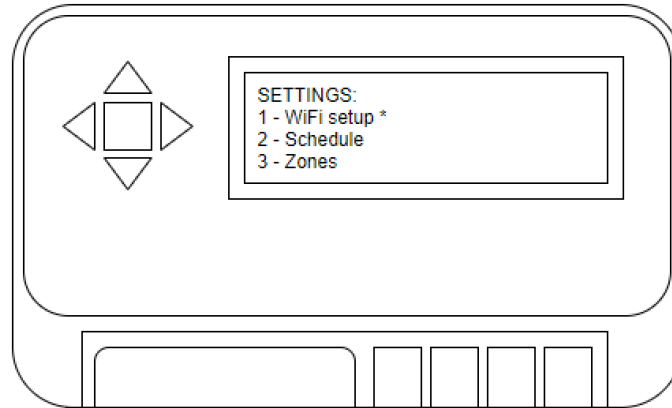


Figure 1 - Smart Controller Mockup, Image by Author

2.4 Requirements Specifications

The following are the detailed requirements for the system. At the end of the requirements specification section a table is provided for review.

The controller should be able to control, regulate watering service, at least three zones. Each plot will be independent of the others, meaning one zone is watered at a time and each zone has its own schedule. Controlling three zones would allow for three different profiles. Each profile could have a different species of plant(s).

The system shall have a schedule for each zone. The schedule shall contain information regarding when to water and for how long to water. The system shall turn on the watering as schedule without user interaction.

The system should have a low power consumption. 'Low' as used here to mean compared to other high functionality water regulation technology such as sprinkler systems. Typical standby power consumption of smart watering systems is from 3W to 6W.

The system should be easy to setup, as the end user may not have deep knowledge of technology. Users should not have to perform extensive setup tasks when installing the system, meaning they must only connect a water and power sources, connect the included sensors, start the system, and confirm operation. Systems which are too complicated could discourage users.

It should not take long to setup the system. The tasks detailed above should not take long to complete. The setup should be completed in a time frame comparable to other small water regulation systems. This timeframe would include any wireless connectivity that would need to take.

The system should be lightweight. 'Lightweight' as used here to mean compared to other high functionality water regulation technology such as water sprinkler

controller systems. The system should be easily transportable by a single capable adult.

The system should be compact. If a system were too large it could be an issue to be transported by the user and shipping. The system, including sensors, control board, and independent power supply (if included) should be able to be transported at short distances by a single capable adult.

The physical system shall have a display which would be used when configuring the physical unit. This display interface should allow the user to view the configurations of the device. The display should provide some of the functionality included in the web application interface, including but not limited to viewing current sensor readings and performing on-demand watering.

The system should be able to collect data which would be of interest to the user. Data of interest for the user would be collected via sensors. The data to be collected would be soil moisture, temperature, light, relative humidity, and atmospheric pressure.

The system shall upload data collected by the sensors to a database management system. This data shall be reviewed via a web application and such web application shall be secure. The data which belongs to user x should only be accessible by user x. In addition, user x should only be able to see its own data. Basic authentication shall be employed by the system to keep the data secure. Basic authentication consists of a username and password.

The system shall be configurable via a web application. In addition to presenting the data collected by the system, the web application shall allow the user to control the schedule for a controller.

The system shall be able to connect to the internet. This connection should be via a Wi-Fi connection.

Finally, further development of the system should integrate with one of the current smart home device solutions. The system should be able to receive commands from Alexa such as “water zone one for five minutes”.

The following table summarizes the requirements above and provided values as necessary.

Table 1 - Requirements Table

Requirement	Value
Communication	
Protocol for network communication	WiFi (802.11)
Protocol for data transfer over network	HTTP
Protocol for sensor communication	I2C
Security	Basic(username/pass)
Configuration	
LCD to review and configure hardware	True
Push buttons for direct configuration	True
Simple to configure	True
Time to configure	Less than 15 Min
Web Application for configuration	True
Data Collected	
Humidity	%
Pressure	hPa
Moisture	pF
Light	% (calculated)
Temperature	°C
Number of zones	3+
Hardware	
Dimensions	Less than 10'x10'x10'
Number of zones	3
Number of zones watered at once	1
Weight	Less than 10 lbs.

2.5 Marketing and Engineering Requirements

The House of Quality is a key component in any project that compares both the Engineering and End-User Requirements. It also decides what pros and cons each aspect of the design has in relation to one another to evaluate for any needed adjustments that may result in the future. This assists in understanding what exactly the projects main advantages and weaknesses may be and what to look at changing without disrupting any of the other requirements. Below is the tradeoff matrix for Smart Garden Controller and how the different requirements correlate with each other, as well as the goals for engineering requirements set for our final project.

			Engineering Requirements							
			Sensor Precision	Power Consumption	Weight	Dimensions	Cost	Set-up Time	Water Use	
			+	-	-	-	-	-	-	
End-User Requirements	Cost	-	↓↓	↓	↓	↓	↓	↓	↓	↓
	# of zones (Total Area)	+		↓	↓	↓↓	↓	↓↓	↑↑	
	Battery life	+	↓	↑	↓	↑	↓	↑		
	Ease of use	+			↓	↑	↓	↑		
	Ease of installation	+	↓		↑	↑↑	↓	↑		
	Water use reduction	+			↓		↓↓		↑↑	
	Data analysis	+	↑↑	↓			↓			
	Web application	+		↓			↓	↓		
	User interface	+		↓	↓	↓	↓	↓		
	Goal for engineering requirements			Less than +/- 2% Error	Less than 6Watts	Less than 5 Pounds	Less than 6"x6"x6"	Less than \$350	Less than 5 Minutes	Less than 5 Gallons/Wk

Key	
+	Positive Polarity
-	Negative Polarity
↑	Positive Correlation
↓	Negative Correlation
↑↑	Strong Positive Correlation
↓↓	Strong Negative Correlation

Figure 2 - House of Quality

The first step in our analysis is to figure out the end-user requirements and how the average user would like to utilize and prioritize different aspects of the Smart Garden Controller. One of the most important goals for the Smart Garden Controller is the cost because the main focus is to provide an efficient, functional, and well-rounded device, while also being affordable to the average person. If the product is too expensive it will limit the potential customers who may be interested in the product. On the other hand, if the product is too inexpensive some features may be lacking or missing and overall hurt the intention of the Smart Garden Controller.

The number of zones or total area the product will manage helps add more options for the customer as they are able to add more different fruits/vegetables to the garden or more of the same ones in the different zones. If more zones are added to the Smart Garden Controller, water use, and power consumption would escalate causing the overall price to go up as well. Lowering the number of zones would limit the customer to only a few options when using the Smart Garden Controller. More zones are appealing to the customer, but this will also make the project more

complex and overall more expensive, which may deter some users from buying the Smart Garden Controller.

Battery life of the system is essential to the Smart Garden Controller because it allows the device to run for long periods of time without the customer having to constantly worry. If the battery of the system is not able to handle an extended period without user interference the Smart Garden Controller could stop working while it is supposed to be managing the garden, potentially ruining the fruits and vegetables already planted. Having a long battery life for the Smart Garden Controller is very important for the user because it also means they will spend less time and money on changing old batteries for new ones.

Water reduction of the Smart Garden Controller is also important because it allows the device to run at optimal conditions without having excess water being wasted and thus increasing the price for the customer. This could be a huge issue because the garden needs to be watered at specific times and with specific quantities and if there is an excess overflow of water being pushed, it can damage the fruits and vegetables in the garden. If the water use of the Smart Garden Controller is carefully regulated and distributed correctly, the customer will have minimized the overall cost they are spending on the garden.

Ease of use, installation, data analysis, web application, and user interface all aim to help the customer able to interact with the Smart Garden Controller in a more simple and organized way. The added data analysis and web application helps record and store information about the garden and shows the customer the best way to set up their garden and what to look out for. This appeals a lot to the customer because they will be able to customize their own garden based on what previously has been recorded and adjust if something was to happen that was not intended. The installation time and how easy the Smart Garden Controller can be operated helps attract more types of customers. If the Smart Garden Controller is too complex or cluttered, many customers will have a hard time using the features and overall not enjoy the product.

After analyzing the different aspects of the end-user requirements, the next step is to take into account the engineering requirements and how important the different goals set for the Smart Garden Controller will interact with the other goals, as well as the end-user requirements. This is an important step because even though the end-user requirements give us a criterion for the requirements based on the user it is important to also base it off how impactful the engineering side of the project is as well. With both of these requirements taken into account, analyzing and understanding which aspects of the project affect each other and are most important to prioritize should be simple and valuable to both the end-user and our team.

Sensor precision is important for our Smart Garden Controller because it is how we will be receiving the data of the soil, water, temperature, and environment. This will be used to know when and how much to water the zones and how accurately the user will be able to see their gardens condition. Without accurate and reliable sensors, the Smart Garden Controller may not work at optimal conditions and may end up hurting the garden if it is getting inaccurate data. One of the main goals it to make sure the user is getting an accurate and reliable system, which is why we set the tolerance to be about a maximum of two percent error for our sensors to be as safe as possible.

Power consumption is another engineering requirement that the Smart Garden Controller will take into account because the less power it uses less heat will be dissipated and more battery life will be available. This is a key requirement in any project because with more battery life, less time will be spent by the user worrying about if the Smart Garden Controller will need more batteries. The Smart Garden Controller will be constantly monitoring the zones of the garden and recording data, thus making this an important aspect to consider. The team has taken into account other similar products and has set the goal for this requirement to be less than six watts of power.

The next engineering requirement for the Smart Garden Controller is the weight and dimensions of the device. These are important because having a heavy and huge design may be too much to handle and the focus is having the Smart Garden Controller be as compact and user-friendly as possible. Too much weight and space taken means that there are a lot of extra components and features in the design, which is not the main focus of the Smart Garden Controller as of right now. The team set the goal for both of these requirements to be less than five pounds and under 6 inches by 6 inches to keep the project fairly simple for the user and compact as possible.

The set-up time is the next engineering requirements that has a lot of importance for the Smart Garden Controller because the device should not have too much complexity to warrant a very long set up time. As stated before, the main goal is to keep the Smart Garden Controller as simple and compact as possible, while performing the same tasks as the other competitors of similar products. If the set-up time is too long, this could mean the Smart Garden Controllers design is clunky and needs to be more organized. Having a long set-up time is also bad for the user and may be difficult for some users to utilize. The goal the team set for this requirement is less than five minutes, which should be a reasonable and appealing time for the user.

Water use of the Smart Garden Controller is another engineering requirement to be wary of because the device is constantly running and trying to be as efficient and economical as possible. If the water is not distributed correctly then this means that too little or too much may be given to the garden, which may hurt the crops the user is trying to grow. It is also important to save as much water as possible to save money and use the resource as needed. This can be done by making sure we have efficient and correct paths for the water to flow through and the sprinkler heads we use cover the zone as well as possible. The team researched how much water is typically used to water a small garden every week and set the goal for this requirement to be less than five gallons of water every week. This will allow the Smart Garden Controller to be as resourceful as possible with the water and make sure it is not being wasted.

Finally, as with the end-user requirements the cost is another important requirement to look into in the engineering side of things. Cost is important because the project should not cost too much in order to be appealing to the team's budget, as well as be efficient as possible with respect to cost. For example, if a sensor has a 2% error for about \$30 value versus another sensor that has a 1% error for about \$150 value, then the more obvious choice would be the 2% error sensor because it is saving a lot of money for only a small benefit in quality which the cost does not justify. This will also help keep the Smart Garden Controller at a low cost for the user and help the team advertise it as a potential competitor to other products of similar nature. The team has decided that after viewing other similar products, a budget of \$350 will be a good goal to achieve for this project.

Now that both the end-user requirements and engineering requirements have been identified and valued, the tradeoff matrix can be made as shown in the House of Quality diagram to compare both sides of the end-user and engineering requirements. In the tradeoff matrix, the "+" shows a positive aspect that we consider for the project, while the "-" shows a negative aspect that isn't beneficial for the project. For example, the user would want the cost to be low so the cost has a "-", while the user would want more battery life, which is shown by the "+". Then the tradeoff matrix compares both by either an upward or downward arrow to show either a positive or negative correlation, respectfully. We decided to add an extra upward or downward arrow to show an extremely positive or negative correlation to better contrast the comparisons we are making. A positive correlation means in this case that both the end-user requirement and engineering requirement being compared are both positive to each other and work in tandem. On the other hand, a negative correlation means that the end-user requirement and engineering requirement are working inversely to each other and thus oppose each other. For example, for the end-user requirement we have the data analysis, while for the engineering requirement we have the sensor precision. If we compare both of these together it would be a strong positive correlation because the sensors

accuracy directly affects the data analysis the user will receive from it, thus making it very important to have this in mind when choosing the sensors. The tradeoff matrix helped keep our team mindful of the many different aspects of our requirements for the Smart Garden Controller.

3.0 Research

This chapter covers the research done for this project. The first section covers similar projects and products found. The projects and products already existent can be used to help better design and prototype the product. The next section goes over relevant technologies found which will be used in this project. Finally, the last section covers the research and reasoning in picking the components which will be put into the smart controller.

3.1 Existing Projects and Products

Most technologies build upon existing ideas and technologies, the device built in this project does as well. Comparing our requirements and design to that of existing technologies is time well spent. Review of previous products can help a project stay on track by keeping requirements realistic. Also, evaluation similar technology and previous projects can help avoid pitfalls seen in other projects. Finally, knowledge of existing device similar to our project will show if our product is well developed and avoid tunnel vision.

3.1.1 Existing Projects

The following are some projects which are similar to our Smart Garden Controller. The following projects were found through UCFs senior design projects webpage. The main difference to note is that some of the following projects are for hydroponics rather than soil based growing. Another difference is that most of the mentioned projects were geared towards growing plants inside the home, while our Smart Garden Controller focuses on growing plants outside the home. Also, most of the following projects were fully contained systems, while the Smart Garden Controller's goal is to integrate with an existing garden.

3.1.1.1 AutoBott

The AutoBott project was created by group two for the spring of 2015 senior design class. Initially, it started out as a smart garden with soil and regular sprinklers. However, eventually the project moved to hydroponics. The project had PH, light, and a water sensor. Based on sensor readings the water level and PH could be adjusted. Also, the project had wireless communication. The project aimed at scaling down an outdoor garden to fit into a house. This project was similar to smart controller presented here in which it aimed at growing plants. However, the AutoBott aimed at growing plants inside the house while the smart garden controller aims at the outside garden. More information for this project can be found on the EECS Senior Design website.

3.1.1.2 Autonomous Hydroponics System

The Energy Sustainable Hydroponics with Automated Reporting and Monitoring was created by group fifteen for the spring of 2015 senior design class. The system aimed at automating some of the tasks necessary for maintaining a hydroponics system. The system was similar to the AutoBott project previously mentioned, and in addition uploaded data collected from the system to a web server. More information regarding this project can be found at the following web page <http://www.eecs.ucf.edu/seniordesign/sp2015su2015/g15/>.

3.1.1.3 Plant Automated Sustainable System

The project that we found which most closely matched our smart controller was the PASS, Plant Automated Sustainable System. The PASS was a senior design project from fall of 2013. The PASS did not have a web interface and was controlled via Bluetooth. The PASS used regular soil, compared to most of the other projects found which focused on hydroponics. The user needed either a tablet or a phone to interact with the device. The system used a network of sensors to determine when to water the plants. However, the system did not provide metrics for the user for review. Also, the main focus of the system seemed to be portability and for an audience which needed to plant indoors. More information about the project can be seen at eecs.ucf.edu/seniordesign/fa2013sp2014/g31/.

3.1.2 Existing Products

In this section we will take a look at some of the existing products which are similar to the device our group is building. The research into existing products will be helpful in fine tuning our design. Review of customer feedback on such products will be helpful in determining what features other products are missing which we could improve upon.

3.1.2.1 Orbit B-hyve 21005/21004 Wi-Fi/Bluetooth Timer

The Orbit B-hyve 21005 Bluetooth Hose Faucet Timer provides control of a single hose. The device is controlled via a Bluetooth connection. The Bluetooth only version needs a mobile device in order to connect and configure the device via its application. The device can be upgraded to work with the WIFI hub at an additional cost. The device has a water meter which allows the user to track water usage. The Bluetooth only version can be found for as low as \$59.99 on the company's website store.orbitonline.com.

The Wi-Fi hub, purchased separately, can control multiple Orbit timers. With the addition of the Wi-Fi hub, the user can control the valves with Amazon Alexa voice control commands. The Wi-Fi hub version is priced at \$82.99 on the Orbit's website. The Wi-Fi hub also provides smart watering features. Orbit's

WeatherSense technology will trigger the watering system based on site conditions such as soil type, slope, and live weather feeds.

The Orbit B-hyve 21005 is a good option for a hobbyist gardener. However, the Orbit 21005 is not necessarily cheap per unit. Therefore, the cost of using this device in a home setting with multiple valves can get high as more valves are added. Additionally, the Orbit 21005 does not provide data from the garden such as soil moisture level, light, wind, or temperature. Although it is a good watering mechanism at a reasonable price, it does not provide the metrics needed for a home garden. Therefore, it seems the Orbit 21005 would be a better option for lawns rather than an actual garden.

3.1.2.2 Rachio WiFi Smart Lawn Sprinkler Controller

The Rachio WiFi Smart Lawn Sprinkler controller can control eight zones with the cheaper version and up to sixteen with the most expensive version. The device can be controlled over WiFi through the companion app using a smartphone, tablet or laptop. The watering can also be configured via the controller directly as well. The device installation is simple and can be done without special tools. An enclosure for weather protection can be purchased separately, if the device is to be used outside.

The device has smart features to help save water. The device can have zones configured by plant type. The controller will automatically determine how often to water the plants when the plant type and sun exposure is configured, there is no actual sensor for light. The controller has a rain and wind feature which can be used to skip watering based on local weather conditions forecast. The controller can be configured to work with Nest, Amazon Alexa, and Google Assistant. When paired with Alexa for example, the device can trigger watering of a specific zone via voice commands. For example, the user can simply say “Alexa, tell Rachio to water the front yard” and this would trigger watering for the zone configured for the front yard.

The smart controller feature list has rain sensors, soil sensor, and wired flow sensor. However, no further information was seen on how the sensors were integrated. It was found that this controller is usually configured to collect data from a weather station closest to it. Based on the data returned by the weather station, the algorithm decides how often to water the plants. However, the weather station picked may not be the closest to the actual location of the controller. Therefore, data collected based on physical sensors placed on the garden would be more reliable in conjunction with weather station data. In all, the Rachio comes with many useful features at a starting price of \$179.

3.1.2.3 RainMachine Touch HD-12, Cloud Independent

The RainMachine Touch HD-12 is the high-end sprinkler controller. The RainMachine comes equipped with a capacitive touch screen. The RainMachine looks very flashy and could be regarded as the MacBook pro of the sprinkler controllers. However just like the MacBook pro, it does not provide any vital features which are not found on the cheaper Rachio system. Also similar to the compared MacBook pro, the RainMachine comes as the most expensive of the mentioned systems at \$240 for the 12 zones version. There is also a 16 zones version.

The RainMachine is very similar in features to the Rachio controller, except for the touchscreen. The RainMachine can save water with the use of weather adjustments by collecting weather information and deciding if watering can be skipped. The RainMachine provides access to control it via android, iPhone, and PC browser. The RainMachine can also be integrated with Alexa, Nest, Wink, and SmartThings through the free platform IFTTT (If This, Then That).

3.1.2.4 Summary of Existing Similar Projects and Products

After going through and reviewing all the existing projects and products that are similar to the Smart Garden our team is planning to build, the team has a more concrete and baseline of what to expect when the actual project is being designed and built. The other projects and products help the team to know what might be too much to put into the Smart Garden Controller given the time restraint our team has and what innovative and interesting things can be added to the device to make it stand out from the other existing products. It is important for the Smart Garden Controller to stand out from the other products because we want to target an audience that is looking for something similar to the other existing products, but provides more features that the user and utilize to make the overall experience more enjoyable and simpler to use. It is also important that the team looks at the success and failures of the other projects to take this into account for the Smart Garden Controller to make sure we repeat the good aspects and improve on them and avoid the negative things the other products and projects may have encountered.

The success of the Smart Garden Controller is important for the team because it means that whole project was worthwhile and had a positive impact on some users, which is one of the main goals the team strived for. The team will focus on understanding what the other existing products and projects struggled with during their design and manufacturing phase and making sure that it is minimized for the Smart Garden Controller. It is also important that the team tries to see what positive things helped the other products and projects be successful and trying to strive for the same amount of success. If the team was successful on making the garden experience simpler and enjoyable for people who used to believe it couldn't be, then it shows that the Smart Garden Controller has achieved this goal.

3.2 Relevant Technologies

The evaluation of existing technologies allows us to examine our initial system design in an effort to determine the feasibility of the design. The relevant technologies that comprise the system are the essential building blocks that will be leveraged to complete a successful design. The ability to use and leverage relevant technology removes much of the research and testing required to develop a new product and mature it through the production process to market. Our use of the existing relevant technologies today is evident in the subsections to follow.

3.2.1 Communication

In this section we will focus on technologies which provide communication between devices. The project will need to communicate at different levels and will require multiple technologies for communications. There will be communication between the microcontroller and the multiple sensors. There will be communication between the microcontroller and the cloud server, which will store all the data collected by the sensors for review. Therefore, we will analyze multiple technologies in order to decide what technologies better fit our product requirements. We begin by first taking a look at wireless technologies which will enable control of the microcontroller and upload of data connected by the sensors to the cloud, such as WiFi and Bluetooth. Then, we will look into technologies which can pass information between the sensors to the microcontroller, such as I2C and UART.

The history of Wi-Fi or WLAN (Wireless Local Area Network) begins with the ruling of the Federal Communication Commission in 1985 which released the ISM band for unlicensed use. This ruling eventually led to the creation and release of the 802.11 standard set by the IEEE which would regulate the wireless landscape. The 802.11 standard is a set of standards for wireless devices, designed to govern and regulate how wireless devices are designed and how they interface with one another. Wi-Fi is used in many applications today and has become an overwhelming force in the constant transmission and downloading of data throughout each day in our daily lives.

The 802.11 standard governs the interfacing between wirelessly enabled devices. There are many different types and combinations of Wi-Fi devices that use the many different 802.11 standards for operation, so for simplicities sake we will stick to the most general cases of Wi-Fi. For a basic Wi-Fi setup, the user requires 3 things; a wirelessly enabled device, a hard-wired internet connection with active service, and a wireless router. The router is plugged into the hard-wired internet connection. The wirelessly enabled device is paired with the router so that security and encryption can take place based on the user's needs. The wireless device either requests information from the internet or requests to post information to the internet and these requests are compiled digitally according to the 802.11 standard

or transmitted wirelessly at approximately 2.4 GHz to the router. The router then decodes the wirelessly received messages and transmits the data to the internet.

The Bluetooth technology was originally created to allow wireless communications for headsets. The technology allows for data exchanges over short distances. Bluetooth also allows for personal area networks, where more than one device can exchange data. On a Bluetooth connection there is a master device which connects up to seven slave devices. All the slave devices must use the master's basic clock. The Bluetooth technology is used in a multitude of applications.

There are three classes of Bluetooth devices. The classes mostly determine how far the devices can be from each other. Class 3 allows for connections of up to 1 meter. Class 2 is the most often used and allows for connections of up to 10 meters. Finally, class 1 has the highest range and in theory allows for connections of up to 100 meters. However, in practice class 1 allows for up to 30 meters. The range will depend on the quality of the link at both ends and air conditions between the devices.

Bluetooth was standardized by the IEEE as standard 802.15.1. However, this standard is no longer maintained by the IEEE. Bluetooth is currently controlled by the Bluetooth Special Interest Group (SIG). The Bluetooth Special Interest group is composed of over thousands of member companies in multiple areas including telecommunication, computing, and consumer electronics. The development of the specification, and protection of the trademarks is controlled by the Bluetooth SIG. To market a device as Bluetooth, the manufacturer should meet the Bluetooth SIG standards.

For wireless communication and upload of data to the cloud WiFi is currently the chosen technology for the controller. For the most part Bluetooth would work fine. Most devices, including phones and laptops, have Bluetooth and therefore would allow management of the controller and review of data. As per requirements we need to have the data collected by the sensors uploaded to the remote web servers and be available for review. If using Bluetooth, the controller would only be able to do so when in conjunction with another Bluetooth device which could provide network connectivity. The secondary device would then upload the data to the remote web server. However, with WiFi the controller can be connected directly to a network wireless access point and upload the data directly without the need to of a "bridge" device. Because wireless access points are widely available WiFi is the chosen technology for management of the controller.

Regarding communications between the microcontroller to the peripherals we start with a review of I2C. I2C was invented in 1982 by Phillips Semiconductor. I2C is used commonly to provide communication between microcontrollers and processors to lower speed devices and within a short distance. I2C is often used in situations where cost is more important than speed. A master device is connected to multiple slave devices, similar to Bluetooth as far as the master and

slave design except for the hardware connection. This technology is relevant as it could be used to provide communication between the microcontroller unit and the various sensors and outputs connected.

I2C uses only two bidirectional lines for communication, one line for data transfer line and one line for serial clock line. The master controls what devices should be communicating and also control the base clock. The master start communications by sending a start command together with the address of a slave device. Communication is usually stopped once the master sends the stop command. Voltages used are usually between 3.3 and 5 V. The commands sent from the master to the slave must be commands as outlined per the slave device documentation.

I2C is not fast. However, there is the option of using clock stretching to provide faster communications between master and slave devices. Because I2C only has 7 bits for addressing, the address range is limited. Some of the benefits of using I2C are very low current consumption, extensive supply voltage, broad temperature range, and high noise immunity. The communication between the sensors and the controller will not need to be fast, therefore the low speed of I2C will not be an issue. Low current consumption is a positive as the controller may need to work solely on battery power at times. The broad temperature range is also a positive as the controller may be located outside. I2C lacks in some areas, however such areas are not important for the smart controller. Given the needs of the project, I2C is a viable option for communicating between the controller and the sensors.

A universal asynchronous receiver-transmitter, also known as UART, is a hardware component which provides serial asynchronous communication between devices to its peripherals. The UART usually receives eight bits in parallel before converting them to serial and transmitting. The receiver gets the serial communication and converts to eight bits in parallel. UART has transmission speeds and data formats which are configurable. Communication between two UART devices can be simple to setup. However, it is usually required to have one UART per peripheral. This can be worked around by time multiplexing the UART, when not all UARTs are not used simultaneously by the controller.

Bit banging is when software is used instead of hardware to provide communication with a protocol. Bit banging can be used to save money and space on a project. For example, with bit banging it would be possible to remove the UART from the circuit and thus save the space which would be occupied by the UART as well as the cost of the UART itself. Bit banging would require more development time from the group and UART does not seem ideal at this point for communicating with the sensors, as there will be multiple sensors to gather data from. However, UART may still be used in other parts of the project.

3.2.2 Sensors

This section briefly describes relevant sensors which will be used in this project. The sensors will be a critical part of the project as one of the main requirements for the smart garden is to provide sensor metrics for the gardener. The data collected by the sensors will help in deciding how often to water the plants. The same data will also be used in deciding when to skip a watering session.

3.2.2.1 Soil Moisture Sensor

The soil moisture sensor is perhaps the most important of the sensors. With the help of this sensor we will be able to determine if the soil received enough water. We will set thresholds depending on what type of plant is being grown in each zone. Once a threshold is reached the controller can stop watering a section or skip the watering altogether, if for example the threshold was already reached before a watering session. This data will also help determine if a zone needs to be watered again earlier than expected. This data will be available to the user for analysis over time, and moisture patterns and average levels will be recorded. The water moisture sensor is the only sensor that we will need one per zone.

3.2.2.2 Temperature Sensor

The temperature sensor will help us determine how much water we should put into the soil. While the soil moisture sensor will confirm if the soil got enough water, the temperature sensor will help us predict how much water we need to put into the soil. On a hot day, we may need to water longer than on a cold day. On the hotter days, the water will evaporate faster. Therefore, the hotter the day, the more water we will need to put into the soil. Having real time temperature information will help the controller to adjust to the conditions in real time. Because the temperature should be about the same in all zones we will only need one temperature sensor for the controller, instead of using one per zone.

3.2.2.3 Wind Speed Sensor

The wind speed sensor, anemometer, will provide wind speed data. Wind speed may also alter how long it takes for the soil to dry out. The wind speed sensor, however, will be more useful in determining when to skip a watering cycle. When the winds are very strong it is best to skip a watering session. It can be observed that water may not hit the intended zone when winds are fast. Wind speed data could be collected from a nearby weather tower. However, we believed it would be best to have a wind speed sensor on the garden for higher reliability on the data. We should only need one wind speed sensor per controller. Upon further research and development, this sensor was omitted from the final design, in favor of dedicating developmental resources to more essential features of the system.

3.2.2.4 Ambient Light Sensor

The ambient light sensor will help the gardener keep track of how much light the garden has received. Most plants require a minimum amount of light per day as sun light is used in photosynthesis. Photosynthesis is the method used by plants to create their food from water and carbon dioxide. There is not much a gardener can do to control how much sun light is in a day, however, with this data the gardener may determine what variety of plants can be planted on his garden. Lastly, this data can be useful in finding out if the sun is blocked at any part of the day by an object.

3.2.2.5 Relative Humidity Sensor

The Relative Humidity sensor will aid the user by informing them of the current humidity percentage as well as providing the system the ability to collect humidity data continuously, to analyze then provide to the user. Humidity is a useful metric for gardening because it can imply certain weather conditions that would result in more or less need for watering the garden, and this data could provide the user with insight regarding how individual plants grow in different levels of humidity. In the local operational environment for example, if high humidity percentages are common, the user could use this data to choose plants that thrive in humid climates.

3.2.2.6 Atmospheric Pressure Sensor

The atmospheric pressure sensor, or barometer, will allow consumers to use the system to collect data relevant to local weather patterns. This information could be helpful in anticipating and/or confirming expected weather forecasts that could influence the health and yield of the user's garden. Measuring this kind of data could also aid the user in choosing plants that thrive better in low or high-pressure climates, depending on the data the user collects with the system. Lastly, in the event of extreme weather such as hurricanes (relatively frequent weather phenomena in the operational environment) this sensor will record vital metrics related to those extreme weather events that could be useful in scientific research.

3.2.3 Electronic Valve / Flow Control

The electronic valve is what the micro controller will use to turn the watering system on or off. The valves will be placed between the home, or business, watering system and the pipes connected to the sprinklers. There are basic valves which simply open and close, and there are valves which also provide flow control and pressure control. The flow control regulates the flow or pressure of a fluid.

3.2.4 DC-DC Power Management

The DC-DC power management subsystem will control the electricity to the diverse parts of the system. The power management system accepts an input voltage and can convert to a different voltage, as diverse parts of the system may need another voltage. This is similar to what the electronic valve will do for the watering system, controlling which zones water will run, however controlling what components are on or off instead as well as their voltages.

3.2.5 Rechargeable Battery Management

The smart controller will use batteries as a source of power when the main power input fails. Precise voltage and amperage must be provided to the batteries to ensure they are successfully recharged and ready for use. The rechargeable battery management system will control how the batteries are recharged. Providing incorrect voltage to the rechargeable batteries could damage them, therefore the BMS will be very important to the reliability and safety of the system. Going above the minimum and maximum voltages for a battery can dramatically decrease lifecycle, damage, or even explode.

3.2.6 Software

The controller will water plants and collect data. The data will be accessible by the user. To provide access to this data to the user over the internet will require a database and an application server. To view the data, we will need a web application. The next subsection will cover such relevant software.

3.2.6.1 Database Management System

The project will need a system to keep all the data collected by the sensors. Over time there will be too much data to keep in the controller. The controller has a limited amount of memory. A database management system (DBMS) would be the perfect place to keep all this data. The database management system would keep the data in a remote location accessible to the controller via a network. Database management systems offer fast access to data and an easy way to build relationships between different data sets. A good solution to keep the data would take a lot of time and effort to design, build, and implement. Using a proved system which had been tested by many other projects will greatly simplify the management of the data collected by the controller.

3.2.6.2 Web Server / Application Server

A web server is a program which accepts HTTP requests and serves static HTML pages. Over time web servers became more and more popular. Web servers became the perfect program to access data over the internet. Serving static HTML

pages was no longer enough. Web servers started to behave more like actual applications. Such web servers with capabilities to behave as an application are called application servers. For our project, the application server will be the perfect solution as a bridge between the data collected by the sensors, in the database, and the multiple clients.

3.2.6.3 Web Application Framework

The database management system will keep the data. The application server will be a bridge between the database and any client, be it the controller or the user. Viewing all the data collected by the controller directly on the small LCD provided with the controller would not be a good solution. Therefore, a web application would be a good fit as the software to retrieve data from the application server and present to the user.

3.2.6.4 Hypervisor

Computers and servers used to be very expensive in the earlier days of computing, some business purpose systems still are. It was noticed that the CPUs for these expensive machine would spend a considerable percentage of their time idling, not processing anything. Hypervisors allow a system administration to install multiple operating system and run them simultaneously on the same hardware. Each system installed on a hypervisor would normally be called a virtual machine instance. This way multiple resources can run in the same system without affecting each other. This is relevant to this project because a lot of the software in this project will be developed and tested on virtual machines. Also, the final code may be uploaded to a cloud container, which is in a way a virtual machine running inside a hypervisor.

3.2.7 Digital Data Storage

Digital systems need a way to store all the data being processed, the execution code, and data which should be kept even when the system is turned off. Some of this data is volatile and some is not. Volatile data will be lost once electricity is turned off. Data storage systems have greatly improved over the years, both increase data density and decreased hardware size. The main data storage solutions these days are spinning discs hard drives and flash memory drives, SSDs. While spinning hard drives are a bit too big and have moving parts, flash storage can be very small and does not have moving parts. Flash storage devices such as microSD cards are usually good solutions for most projects. Some MCUs do provide flash memory. However, in some cases the data is kept in the cloud and it does not matter much what type of drives are used.

3.2.8 Liquid Crystal Display (LCD)

The Liquid Crystal Display (LCD) screen on the system will serve as a low-level user interface component. From the LCD screen the user will be able to manipulate the system to perform several operational functions. The user will be able to manually set the system to water or stop watering the garden from this screen. The user will also be able to view some data available from the sensors regarding wind speed, relative humidity, ambient light, temperature, soil moisture level, and Atmospheric pressure. The user will also have the ability to turn the system off or initiate system sleep mode from the LCD interface.

The Display will also serve as a backup user interface in the case that the user is unable to access the web application, or if there is a server failure or any incident that renders the web service unusable. In the case of complete web service failure, the LCD screen will serve as the main user interface for project demonstration.

3.2.9 Smart Speaker

Smart speakers allow users to interact with digital assistants. Such digital assistants are very common these days. All new smart phones have their own versions of digital assistants. These digital assistants have skills. Skills allow a user to speak commands that would perform an action. Such skills include setting alarms, controlling lights, turning on and off devices, setting reminders, and even asking questions about the weather. Smart speakers are relevant to this project as the team will work on getting the controller to accept commands from a smart speaker. The user should be able to give a command to the smart speaker which in turns on a valve in the controller and therefore waters the plants. Upon further research and development, this technology was omitted from the final design, in favor of dedicating developmental resources to more essential features of the system.

3.3 Strategic Components and Part Selections

Thorough research and examination was expended in the effort to properly design and implement a suitable and successful project. The system design in any project is a crucial piece of design work that takes the projects end goals and makes them realizable using the technology that is available, and in a way that is in line with the financial and functional goals of the end product or user. The system design must account for the financial, economic, technological, and ethical requirements and constraints throughout the life of the project to ensure that all goals are met within the limits of the constraints. A good system design requires research on relevant technologies and is often accompanied by a number of trade studies that offer a tradeoff analysis of certain aspects and characteristics that pertain to the overall system performance. These tradeoffs are often compared between competing technologies so that the designer is easily able to discern what options

seem best suited for an application, and what the tradeoff with that selection would be.

The following subsections detail our research in each specific area and provide clarification on the final part selections with supporting evidence for each item. For each subsection there will be a trade analysis that demonstrates the different options that were available and the pros and cons of each option that ultimately led to the final decision.

3.3.1 Microcontroller

A microcontroller is an embedded system that contains a processor, memory, and peripheral interfaces all contained within a single chip or module. Microcontrollers are used in almost all electronics and embedded devices. Microcontrollers are cost effective. Most microcontrollers use very little power and have sleep options that greatly improve battery performance. Some of today's microcontrollers even provide wireless connectivity. In a way, microcontrollers are platforms which abstract the circuitry necessary to control hardware. Therefore, microcontrollers are great for improving reliability and efficiency of products.

For the purposes of this project, the microcontroller will serve as the brains of the entire system. The microcontroller will be responsible for running the main firmware of the system which executes the decision algorithm based on the sensor and user inputs. The microcontroller will also be responsible for all timer based and calendar functions, including setting up and properly running and monitoring the necessary timers, and executing the necessary functions based on the timer interrupts. The microcontroller will be responsible for handling the wireless interface and communicating with the web server used for the user interface. This user interface will allow the user to access the device via WiFi and update settings as well as monitor sensor analytics. Finally, the microcontroller is responsible for running all peripheral devices including; LCD display, user interface buttons, weather sensors, soil sensors, water valve control, etc.

Our microcontroller needs for this project drove us to evaluate the following three microcontrollers as possible solutions. The main requirements that were evaluated in our microcontroller selection were the microcontrollers WiFi capabilities and ease of use and integration for WiFi. The microcontroller needs to be low power so that battery life could be optimized. The microcontroller needs to have sufficient interface options to easily optimize the system design.

3.3.1.1 ATWINC1500

The Microchip ATWINC1500 is an Internet of Things (IoT) module that is optimized for low power consumption and intended for use in Internet of Things devices. The main features of the ATWINC1500 module are that it is an IEEE 802.11 b/g/n 20 MHz, 2.4 GHz single band ISM device with an integrated Transmit/Receive switch

as well as an integrated PCB antenna. The module garners superior sensitivity and range via PHY signal processing and utilizes advanced equalization and channel estimation as well as advanced carrier and timing synchronization techniques. The module supports WEP, WPA, and WPA2 security, has integrated flash memory, and contains a built in 26 MHz crystal. The module has multiple power save modes as well as fast boot options. The module leverages an on-chip network stack to off load the MCU and is able to use WiFi hardware accelerators. All of these features and more make the ATWINC1500 one of the most popular Internet of Things WiFi modules on the market today.

The ATWINC is currently a popular choice for Internet of Things development due to its ease of use for integrating WiFi functionality as well as its low power consumption making it perfect for battery powered wireless devices. This part was taken under much consideration for use in this project. The ATWINC development environment is Atmel studio, which is one of the nicer tools provided by microcontroller developers. Unfortunately, we only have a limited experience with Atmel studio, so there would be an added development time for each group member to investigate and learn the new tool.

3.3.1.2 ESP8266

The Espressif Systems ESP8266 is a popular microcontroller WiFi module. The module runs a real-time operating system (RTOS) and a WiFi stack to enable up to 80% of the processor to be available for the user application. A RTOS allows the embedded system to appear that it is performing multiple tasks in parallel by switching between the multiple tasks very quickly. This is useful when an embedded system needs to respond to events and inputs in real time. A RTOS allows this to happen by allowing the programmer to give a priority to different threads of execution. This prioritization allows for a deterministic execution pattern of the program which enables the embedded system to complete tasks in real time and meet deadlines.

While the price as well as the low power consumption and the fact that it runs a RTOS are very big upsides for the ESP8266, there are some downsides to this module as well. First, the ESP8266 only contains 50 KB of SRAM and has no flash memory integrated into the module. This means that if we were to select this module for our project we would have to integrate our own flash memory which will add risk, because if there is a mistake in the layout of the flash part or the interface between the module and the flash part, we will not even be able to store our firmware image on the device. Another drawback or risk that would be associated with the selection of the ESP8266 is the fact that the development environment that is recommended for use with this module while running RTOS is to use FreeRTOS, which is an open source development environment. The fact that FreeRTOS is open source means that there is a very heavy use of customization and modification throughout the environment. The team believes that this may cause some confusion and frustration during the development process and

believes that a more bounded environment would be beneficial given our experience level and the timeframe given to complete the development.

3.3.1.3 CC3220MODA (Selected)



Figure 3 - CC3220MODA Module Courtesy of TI

The Texas Instruments CC3220MODA is a low power IoT controller microcontroller. The wireless microcontroller module that contains the built in Wi-Fi connectivity is FCC, IC, CE, and Wi-Fi Certified. The CC3220MODA uses an ARM Cortex-M4 processor running at 80 MHz for faster more reliable processing. The microcontroller contains a wide array of peripheral interfaces, including I2S, SD/MMC, UART, SPI, I2C, and a 4 channel ADC. The Wi-Fi network subsystem contains an additional ARM MCU that off loads Wi-Fi tasks from the application MCU. The network subsystem contains the 802.11 radio, a baseband layer, MAC layer, Driver layer, Supplicant layer, TCP/IP layer, SSL layer, and IP layer. The network subsystem on this module would allow for 256-bit encryption and supports WPA2 personal and enterprise, as well as WPS 2.0. All of these features combined lead Texas Instruments to claim that this module requires no prior Wi-Fi experience for development. The CC3220MODA utilizes Texas Instruments' Integrated Development Environment (IDE), Code Composer Studio. This IDE is familiar with all team members as all members of the team have used this IDE for development with the MSP430 microcontroller in previous projects. The level of comfort the team has with Code Composer Studio did play a part in the decision between these three microcontroller modules. The CC3220MODA also comes equipped with an on-board chip antenna, which would offer better wireless performance than the trace antennas of the alternative modules. Some wireless microcontrollers did not have an antenna. For such cases, an antenna would need to be built into the PCB or connected to the device. Therefore, the built in antenna contained in the CC3220MODA MCU removes the risk and effort involved in either creating our own antenna or attaching it.

Knowing that the CC3220MODA is intended to be interfaced with using Code Composer Studio and taking into account the fact that the module is designed for Internet of Things low power applications with a focus on easy Wi-Fi integration, it was not a difficult choice to select the CC3220MODA. Along with the obvious advantages of using the CC3220MODA by analyzing the features out of the box,

our group is also able to leverage historical knowledge of development with this part. At least one team member has been intimate in the electrical design and system application of the CC3220MODA in a real-world project that was the result of an internship opportunity. Documentation can often make or break the efficiency of a team when developing software for an MCU. Texas Instruments has multiple online resources which will be helpful in developing, troubleshooting, and understanding the MCU used. Some of the resources include online tutorials, pdf, and an active user community.

While in the battle of price per module the CC3220MODA came in last place, there were many other considerations that influenced the decision to move forward with the CC3220MODA as our microcontroller/WiFi module for the Smart Home Garden project. The CC3220MODA uses the Cortex-M4 processor, which is a well-known high performance, and reliable processor. The current consumption was comparable to the ATWINC1500 during receive and transmit conditions, but neither compared to the low current consumption of the WiFi activity of the ESP8266. While this low power usage for WiFi activity would help the unit conserve battery life, therefore creating a longer run time between charges. But the determination was made that, in the trade between low current consumption and complexity of design and implementation, a less complex design and easier to use integration environment were more favorable than the lower current consumption.

The memory allocation between the three different modules was another major factor that was taken into consideration, and in this category, the CC3220MODA reigned supreme with 256KB of RAM, which is double that of the ATWINC1500, and more than quadruple that of the ESP8266. This memory would allow for larger code and execution space and reduces the risk of having to optimize code that is too large to run on a given microcontroller. The flash memory is less than that of the ATWINC1500, but with the SD/MMC interface of the CC3220MODA we maintain the ability to upgrade the unit with an SD Card or SD memory interface to expand the flash memory if it is deemed necessary.

The final characteristic that led us to settle on the CC3220MODA is the fact that it is able to operate down to 2.3V rather than 2.6V as in the ESP8266 and ATWINC1500. This extra 0.3V of operating voltage will allow the unit to maintain its normal operational state for a longer period of time as the battery degrades and the voltage drops lower and lower. The ability to leverage the experience of the previous implementation of this module in an electrical design, along with the functionality provided by the module itself and the deterministic factors that were derived from the trade study, led us to choose this part as the microcontroller and wireless interface for the Smart Home Garden project.

Table 2 - Microcontroller Trade Analysis

Microcontroller Trade Analysis			
Module	ATWINC1500	ESP8266	CC3200
Price	\$ 8.08	\$ 5.00	\$ 9.35
Voltage range	2.7V-3.6V	2.7V-3.6V	2.3V-3.6V
Processor	Cortus APS3	Tensilica L106	Cortex-M4
Bits	32-bit	32-bit	32-bit
Receive Current	61mA	56mA	59mA
Transmit Current	265mA	170mA	278mA
RAM	128kB	50kB	256KB
Flash	4MB	n/a	64KB
Wifi Protocol	802.11 b/g/n	802.11 b/g/n	802.11 b/g/n
Data Rate	72.2Mbps	72.2Mbps	54Mbps

As outlined above the table CC3220MODA as the Microcontroller to use for the Smart Garden Controller. It is important for any project to have a list of parts and their different positive and negative aspects to help in choosing what to use in the final design. Many different devices have different things they are more utilized for and thus the team should make sure the right hardware is being used for the right circumstances on our project. It is valuable that the team makes a chart of each potential hardware elements for the Smart Garden Controller because it is easy to compare and contrast each part side by side. This makes it simple for the team to decide what part is best for the cost and the different features it may or may not provide.

When selecting a hardware component for the project our team had to make sure it provided the right features we would be expecting to use for the Smart Garden Controller, while also not being too expensive or difficult to element in the final design. It is simple to choose the most expensive part for the Smart Garden Controller, but having a budget and not too many constricting requirements means that the team is looking for the most affordable and design friendly component to make the project not too complicated and cheaper for the user and thus more appealing.

3.3.2 Battery Management

3.3.2.1 Rechargeable Battery

In the current technological ecosystem, batteries have become an integral part of consumer electronics design. The ability for a user to take their device with them wherever they go, without having to find an outlet to charge a device or maintain its operation has led to a revolution in mobile electronics and has created a new

intense demand for better battery technology. There are four main types of rechargeable batteries that exist currently.

The Lead Acid battery is rugged and economical but has a low cycle count and low specific energy and these batteries are typically used in wheelchairs and golf carts.

Nickel-cadmium (NiCd) batteries are utilized for long service life applications where high discharge current specifications are required as well as extreme temperature ratings. Nickel-cadmium chemistry allows for ultra-fast charging regardless of the batteries current state of charge. Main applications for NiCd batteries are power tools, aviation, and medical devices.

Nickel-metal-hydride (NiMH) batteries are becoming more popular as environmental concerns are being raised with the use of NiCd batteries. NiMH batteries are replacements for NiCd batteries, and their chemistry provides a higher specific energy. NiMH batteries are used in medical instruments and industrial applications but are also manufactured in AA and AAA battery form for consumer use.

Lithium-ion batteries have become much more popular which can be attributed to the fact that the high cycle count and reduced maintenance significantly reduces the cost per cycle of the battery. There are safety concerns with Lithium-ion batteries, and all batteries require a protection circuit that protects the battery from over/under charge scenarios, as well as over-voltage protection, short circuit protection, and over-current protection. This protection circuit adds to the price of the Lithium-ion battery and can complicate the design.

After evaluating the rechargeable battery chemistry options and evaluating our system requirements, it was determined to use a Lithium-ion chemistry battery pack in our system. This choice was based upon the cycle life of that particular battery chemistry as well as its cost per mAh of capacity, specific energy, and availability. A larger capacity battery pack is not as cost effective in a Nickel-metal-hydride chemistry battery and a large capacity battery is needed for our application, as our device will need to run on battery power for an extended periods of time. However, it was determined that the Lithium-ion battery pack will be the optimal solution for our system. The decision was based off of the current availability of NiMH battery management components as well as the availability of the battery packs themselves. The data in the following battery chemistry trade analysis table was used in the decision making process.

Table 3 - Battery Chemistry Trade Analysis

Battery Chemistry Trade Analysis				
Specifications	Lead Acid	NiCd	NiMH	Li-ion
Specific Energy (Wh/kg)	30-50	45-80	60-120	90-120
Internal Resistance	Very Low	Very Low	Low	Very Low
Cycle Life	200-300	1000	300-500	1000-2000
Charge Time	8-16h	1-2h	2-4h	1-2h
Overcharge Tolerance	High	Moderate	Low	Low
Self Discharge/month	5%	20%	30%	<5%
Cell Voltage	2V	1.2V	1.2V	3.3V
Charge Cutoff Voltage	2.4	Full Charge Detection by voltage signature		3.6
Discharge Cutoff Voltage	1.75	1.00V	1.00V	2.50V
Peak Load Current	5C	20C	5C	>30C
Charge Temp (Celcius)	-20-50	0-45	0-45	0-45
Discharge Temp	-20-50	-20-65	-20-65	-20-60
Maintenance Required	3-6 months	Full Discharge every 90 days		None
Safety	Thermally Stable	Fuse protection		Protection Circuit
Toxicity	Very High	Very High	Low	Low
Cost	Low	Moderate	Moderate	High

3.3.2.2 Battery Management Controller

The battery management controller is responsible for monitoring the health of the battery as well as reporting crucial information back to the microcontroller so it may be determined if there are any errors or faults that need to be addressed. There are two main types of battery interface controllers that are pertinent to our system, the battery charging controller, and the fuel gauge.

3.3.2.2.1 Battery Charging Controller

The battery charging controller is responsible for monitoring the health of the battery. The battery charging controller is also responsible for instituting any protections that may be provided with the particular battery charging controller, whenever battery conditions necessitate it. The battery charging controller is responsible for charging the batteries in a safe manner and monitoring the health of the batteries to ensure that no dangerous conditions arise within the battery pack. The battery charging controller is the direct interface to the battery pack and allows the pack to be charged in the most optimal way. The battery charging controller is under its own control and is able to make decisions about the battery charging state without instruction from the microcontroller.

The firmware that allows the battery charging controller to perform these functions is configurable by the programmer. This ensures that the proper criteria that are necessary for optimal operation are input into the battery charging controller's algorithm. The battery charging controller is able to indicate to the microcontroller, via a variety of interfaces, the state of the battery pack and whether or not charging is occurring. All of these combined features allow the battery charging controller to

efficiently and optimally charge the battery pack and report status back to the microcontroller for system level decision making.

3.3.2.2.2 Battery Fuel Gauge

The battery fuel gauge is an integral part of the battery management system. Without a fuel gauge there would be no way to properly report battery life statistics back to the end user. It is paramount for our system that the user gets prompt feedback regarding remaining battery life to ensure that the system is properly powered to avoid any periods of down time which would result in crop loss. If the system were to go down because the battery was drained beyond the point of being operational, the automatic control of the system is lost and the user is unable to rectify this until power is restored.

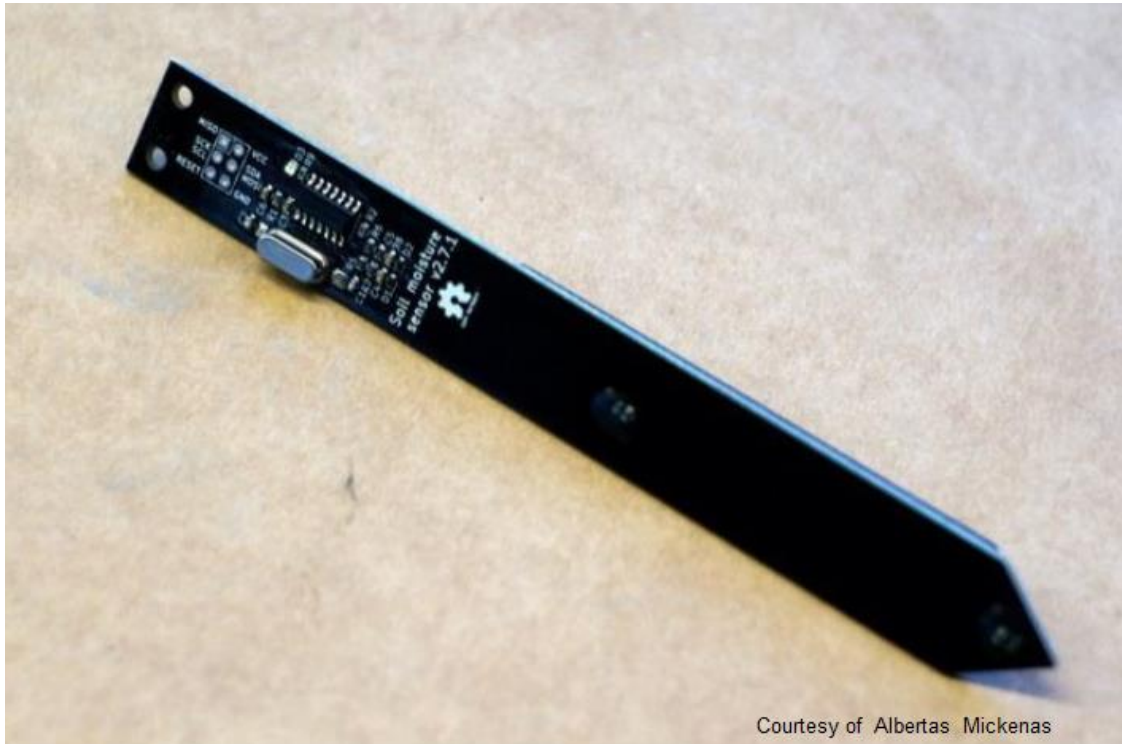
To truly monitor the status of the battery there are many different factors that must be taken into consideration and evaluated for proper calculation of remaining battery life. One may think that the battery life remaining calculations of a simple rechargeable battery system would be trivial algebra, that describes the battery charge capacity as a linear function of current consumption and time, but in reality, the chemistry and life cycle of a battery plays a much more significant role in battery management.

The state of health of the battery is independent of the state of charge of the battery, and these two metrics combined can help lead us to an accurate indication of the current state of charge, as well as the remaining capacity of the battery pack allocated to our system. The state of charge of a battery is commonly thought of as the battery's remaining capacity, and many would assume it to be indicative of remaining battery capacity. Unfortunately, the chemistry within battery packs causes degradation over time. This means that the more life cycles the battery experiences, or the more times the battery is charged and discharged, the less efficient that battery becomes at charging, retaining charge, and optimal discharge. This means that the older and more used a rechargeable battery becomes, the less efficient it is at being a battery. To account for this, fuel gauge controllers keep track of the number of charges and discharges and adjust the status of the battery life based on this knowledge. This allows the end user to obtain accurate battery data even though the battery life is changing over time.

3.3.3 Sensors

The collection and intelligent analysis of environmental data is a key objective of the automated Smart Garden. In order to connect users to the surroundings of their gardens our system must employ sensors to observe and record natural phenomena occurring constantly in the environment. The following sections will provide insight and information regarding the sensors used and considered in the design of our system.

3.3.3.1 Soil Moisture, Temperature, & Ambient Light Sensor



Courtesy of Albertas Mickenas

Figure 4 - Soil Moisture Sensor

The sensor chosen for our application is the I2C Soil Moisture Sensor version 2.7.5 by Catnip Electronics. It delivers readings for relative moisture level, Temperature, and Ambient light. This sensor was chosen considering its communication protocol (I2C), its price, and its measurement methods and capabilities. Compared to other sensors which can only read a single metric, this choice was able to cover three of our six major data points, and therefore was a clear superior.

The moisture level measurements are taken by this sensor using a capacitive moisture sensor. This technology is widespread and reliable, as it is used in several fields on consumer electronics including but not limited to touchscreen devices, position & acceleration sensors, and humidity sensors. When the sensor is taking measurements in soil, the capacitive surface will form a circuit with conductive substances, water for our purposes, and cause a specific voltage drop corresponding to a moisture level datum that is sent to the microcontroller. The capacitance of this circuit will increase or decrease depending on the amount of water in the soil touching the sensor (respectively). This relative capacitance value is the information that will be observed and analyzed by the system [C:Tindie].

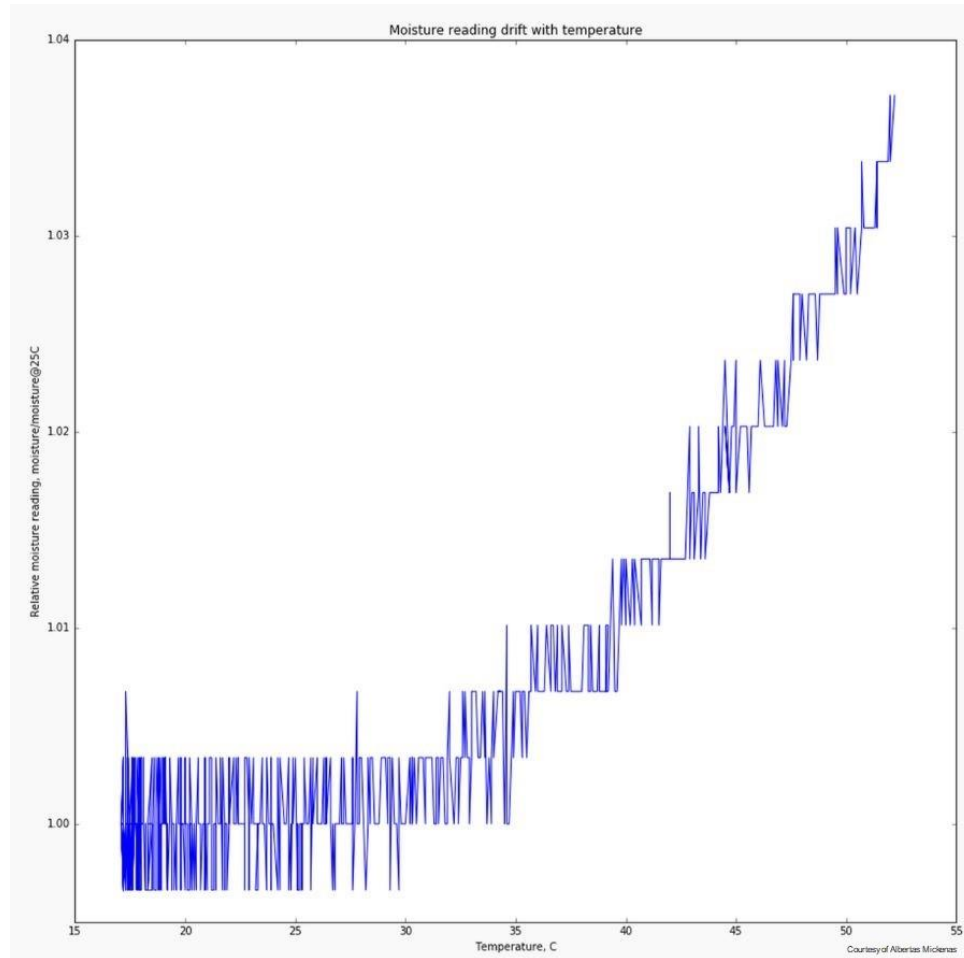


Figure 5 - Temperature Sensor Data

This sensor operates reliably within the temperature range 0°C to 85°C (32°F to 185°F). Temperature readings below this range within reason are not expected to damage the sensor, but the temperature measurement will never read a negative value. Temperature readings above this range are not possible in the operational environment. Moisture readings taken in an environment above 30°C (86°F) are expected to show some drift from the expected value, but this variation is well within reasonable error. The figure above shows that the moisture reading reaches a maximum of less than 4% error in environments above 50°C (122°F). Any measurement error above these temperatures is negligible due to the impracticality of the systems use in environments with temperatures consistently above that threshold.

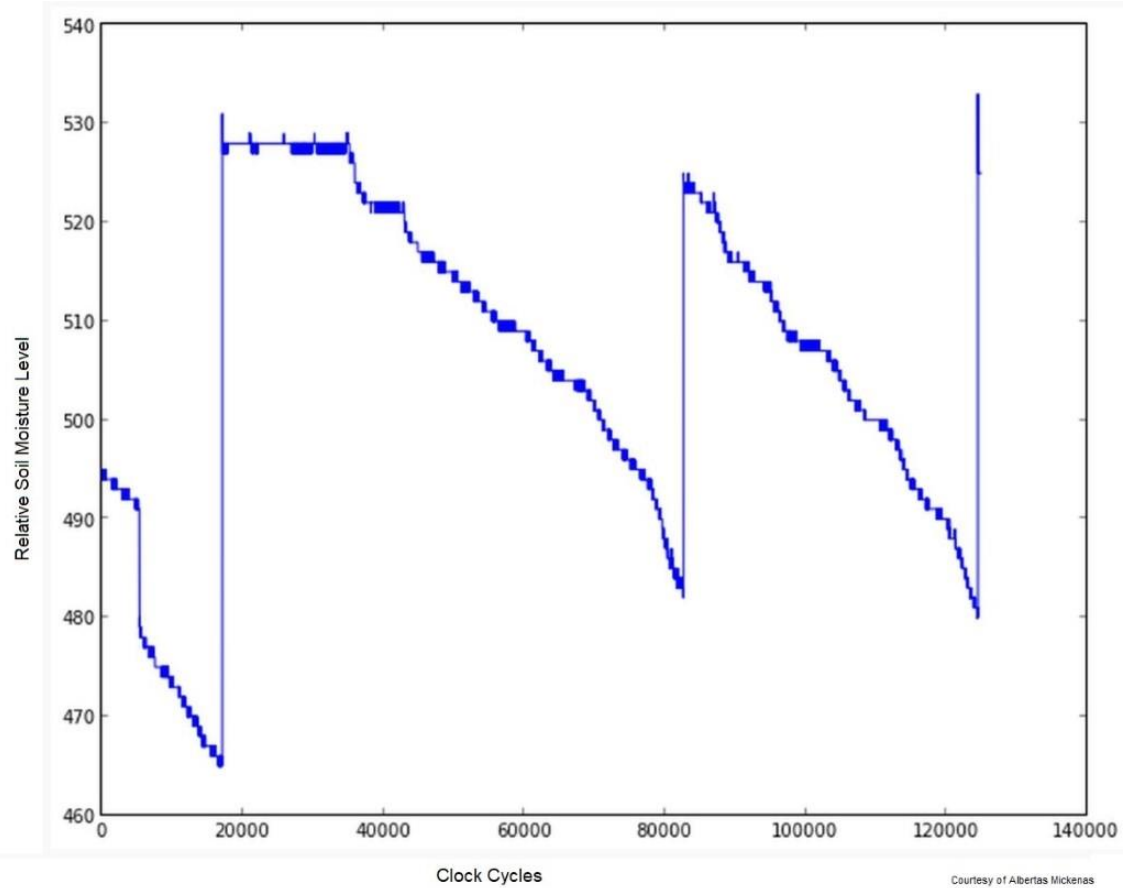


Figure 6 - Soil Moisture Sensor Data

This sensor also has the ability to continuously sample data at a maximum rate of 16 MHz. When the sensor is in the state, it uses an average of 4.5mA with a supply voltage of 5.0V (2.8mA at 3.3V), which is significantly less current than the single measurement consumption rate. Considering this, if several measurements are necessary in a short period of time, the constant polling mode is superior in terms of current consumption. The figure above shows an example of data collected during constant polling mode. The variation seen is a voltage drop over time due to loss in the capacitive circuit. This drop is corrected consistently with a change in applied voltage to the capacitive surface. The peaks of the data indicate the current relative moisture level, approximately 530, which is based on the capacitance of the sensor.

The temperature data recorded by this sensor is saved to the I2C bus in terms of tenths of degrees Celsius, giving our application three significant figures of accuracy for temperature. Ambient light and moisture levels are recorded and saved in terms of a single scaler integer of three significant figures and five, respectively. Both of these measurements are approximately linear, and for the scope of this system, they will be scaled to reasonable values for the user to

understand. See below for technical data regarding this sensor [C:Tindie- See appendix for image permissions].

Table 4 – Soil Moisture Sensor Data

Description	Value
Version	2.7.5
Supply Voltage	3.3V - 5V
Current Consumption (idle)	0.7mA - 1.1mA
Current Consumption (measure)	7.8mA - 14mA
Current Consumption (constant)	2.8mA - 4.5mA
Operating Temperature	0°C - 85°C
Crystal Speed	16MHz
Default I2C Address	0x20

Some acknowledgements must be made regarding this sensor, as it is not perfect. The default sensor color is black, which when exposed to direct sunlight for extended periods of time could cause inaccurate temperature readings. Developers working with this sensor in other projects advise covering most of the sensor with white tape or paint to counteract this anomaly. Also, while the sensor does come equipped with PRF202 protective coating, it is not indefinitely waterproof. The seller and other users suggest using a secondary method of waterproofing such as a "polyester or epoxy resin coat," PlastiDip®, or a standard rubber balloon over the sensor (this method, while easiest of the three, also invalidates ambient light measurements taken using this method)[C9]. Finally, the sensor should not be hotplugged into an active I2C bus. The seller warns that this could cause the I2C address to be randomly reset. Our development team had no intention to use the sensor in this manner, so this issue does not affect our design.

3.3.3.2 Humidity & Atmospheric Pressure Sensor

The sensor chosen to measure Humidity and Atmospheric Pressure is the Bosch BME280 Humidity and pressure sensor. This sensor was chosen considering its communication protocol (I2C), its price, and its measurement capabilities. Several versions of this sensor are available, with different target data points including temperature and ambient light, however this version was chosen in conjunction with the other sensors to reduce the number of sensors needed for the system.

The atmospheric pressure is recorded by this sensor and saved to the I2C bus in terms of hectopascals (hPa) with a range of 300 hPa to 1100 hPa, and the Humidity

is recorded in terms of Percentage of relative humidity with a standard range of 0% to 100%. See below for more Technical data regarding this sensor [Cmouser].

Table 5 – Humidity and Pressure Sensor Data

Description	Value
Supply Voltage	1.71V - 3.6V
Temperature range	-40°C - 85°C
Interface	I2C, SPI
Package size (Metal lid LGA)	2.5x2.5x0.93mm ³
Current consumption @ 1Hz	3.6µA
Operating supply voltage	1.8V
Operating supply current	0.5µA

3.3.3.3 Wind Speed Sensor: Anemometer

The Anemometer chosen for our application is a simple wind speed sensor with analog voltage output. This sensor was ideal because of its data output, reliability, and size. Other sensors similar were larger or more complex, while other sensors that also detected wind direction were prohibitively expensive.



Figure 7 – Anemometer Courtesy of Adafruit

This Anemometer can measure wind speeds between 0 m/s (0 mph) and 32.4 m/s (72.48 mph). This data is delivered to the microcontroller via analog voltage output, with 0.4V corresponding to 0 m/s and 2.0V corresponding to 32.4 m/s. Outside of this detection range the voltage will not increase unexpectedly which is ideal to protect the system, while such speeds would be unlikely and otherwise dangerous regardless. See below for technical data regarding this anemometer [C:Adafruit see appendix for image permission]. Upon further research and development, this sensor was omitted from the final design, in favor of dedicating developmental resources to more essential features of the system.

Table 6 - Anemometer Data

Description	Value
Output Voltage	0.4V - 2.0V
Testing Range	0.5 m/s - 50 m/s
Resolution	0.1 m/s
Accuracy (Worst Case)	+/- 1 m/s
Max wind speed	70 m/s
Mass	111.8 g
Hight	105mm
Arm Length	70mm

3.3.3.4 Other Sensors Considered

In addition to the sensors listed above other sensors were researched and considered for use in our system. The following sections detail a few of those sensors that were the closest to ideal for use in the system, and the reasons they were not chosen for use in our design.

3.3.3.4.1 HTU21D-F Temperature & Humidity Sensor

One sensor considered to measure relative humidity and temperature was the Adafruit HTU21D-F sensor breakout board. This sensor was not ideal because the combination of sensors chosen provided the most measurements with the fewest sensors to use, however if our target metrics were only temperature and humidity this sensor would have been ideal. This sensor delivers accurate results within +/- 2% from 5% to 95% relative humidity and +/- 1°C from -30°C to 90°C. See below for technical data regarding this sensor.

Table 7 – HTU21D-F Sensor Data

Description	Value
Supply voltage	3.3V - 5.0V
Dimensions	18mm x 16mm x 2mm
Mass	1.0 g
Default I2C address	0x40
Temperature read range	-30°C - 90°C
Humidity read range	5% - 95%

3.3.3.4.2 BMP280 Barometric Pressure & Altitude Sensor

The Adafruit BMP280 I2C or SPI Barometric pressure & Altitude sensor was considered for use in our system because of its communication protocol, price, and ability to measure atmospheric pressure. While Altitude was not a target metric for our team when searching for sensors such a measurement seemed like it could be a possible addition. Eventually however we found a better combination of sensors to measure pressure and our other target metrics. See below for technical details regarding this sensor.

Table 8 – BMP280 Sensor Data

Description	Value
Interface	I2C / SPI
Supply voltage	3.0V - 5.0V
Pressure accuracy	+/- 1 hPa
Altimeter accuracy	+/- 1 m
Average altitude noise	0.25 m

3.3.3.4.2 BME680 Temperature, Humidity, Pressure, & Gas Sensor

This sensor was considered for use in our system because of its interface and its ability to measure two of our target metrics: humidity and pressure. This sensor was determined to be less than ideal because of its price and its redundant ability to measure temperature. The ability of this sensor to measure the overall Volatile Organic Compound (VOC) concentration in the surrounding air. The VOC content in the air would indicate the amount of gasses such as alcohol, ethanol, and carbon monoxide (just the total amount not individual readings). This data would be very

interesting to developers and users alike, but it would not be as useful as other metrics. See below for technical data regarding this sensor [C:Adafruit see appendix for image permission].

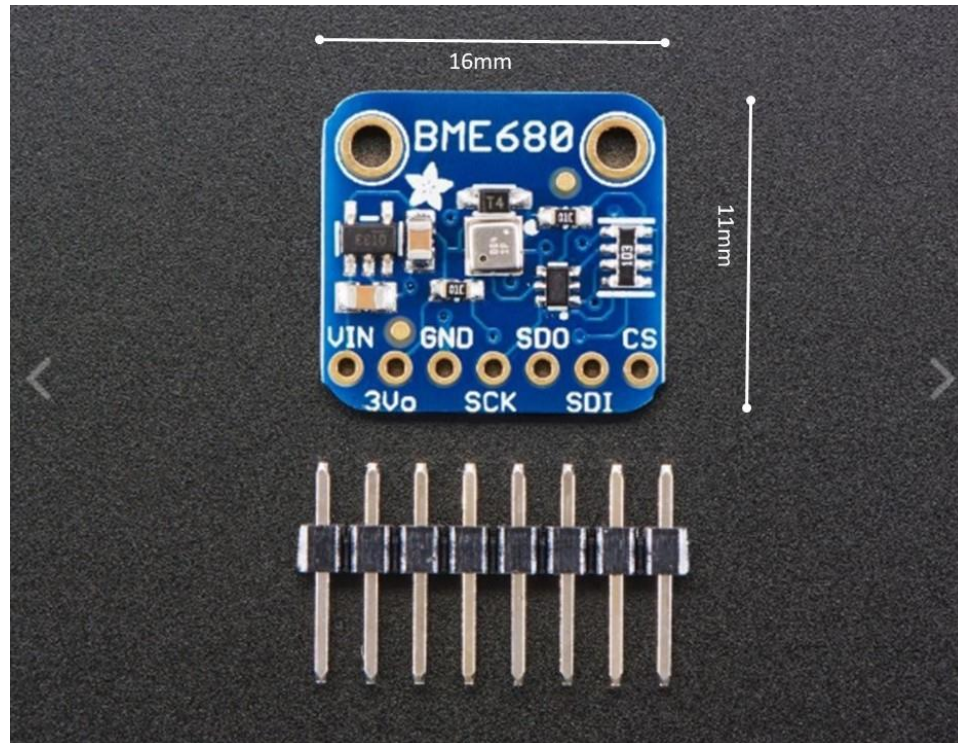


Figure 8 – BME280 Courtesy of Adafruit

Table 9 - BME280 Additional Data

Description	Value
Interface	I2C / SPI
Humidity accuracy	+/- 3%
Pressure accuracy	+/- 1 hPa
Temperature accuracy	+/- 1°C
Supply voltage	3.3V - 5.0V
Mass	3.0 g
Dimensions	16mm x 11mm x 2.8mm

3.3.4 Water Flow Controller

The water flow controller is really where the boots hit the ground in our project. This interface is what is going to keep our plants alive is Mother Nature does not prove bountiful enough for our crop. All the analytics and predictive algorithms in the world will not help us if we are unable to distribute water to the proper zones when they require it. Our water flow controller interface is the instrument with which our microcontroller is able to turn on and off the water flow to each zone individually. There are multiple ways to control the flow of water using electronics.

One simple way to control the flow of water using an electrical signal is to use an automatic multi-zone valve. These valves change orientation based on an electrical signal. The change in location of this valve allows water to flow from the input source out to the different zones. To shut the system off the valve usually returns to a location that does not provide a path forward for the water and the system lays at rest until the next electrical signal is given and the valve switches zones. This was an original option for us on this project but was ruled out due to size cost and overall feasibility. These units are usually quite large and expensive and are mostly designed for sprinkler systems that have a well dug to provide high volumes of pressurized water through pumping. For our purposes we plan to use a simple garden hose and outdoor faucet setup and would not require this heavy-duty valve.

Another solution to electrically control the flow of water is to use solenoid valve technology. A solenoid valve works by translating an electrical signal into a mechanical motion that opens and closes a valve to allow the passage of some fluid. Luckily for us solenoid valves are relatively inexpensive and readily available. After some research, it was determined that using a DC voltage-controlled solenoid valve would be vastly superior in our design as an AC solenoid would require us to do DC to AC inversion and intensely complication the power design of the system. The lower the voltage of the solenoid valve the better for us as the valve will likely need to be held in open or closed position and this will cause significant drain on our battery.

The water flow controller solenoid valve that has been selected for our design is a 12-volt DC solenoid valve with a plastic housing that is manufactured for liquid system valve controls. This solenoid contains $\frac{1}{2}$ " openings that will be utilized for our zone control hoses. This solenoid is a normally closed or NC type of solenoid. Normally closed implies that when there is no voltage supplied to the solenoid it acts as a closed valve. For the valve to open, the proper voltage must be supplied. In our system this is performed using field effect transistors or electromechanical relays.

3.3.5 LCD Screen Module

The LCD screen selected for use in this system is the LCD-013-420 Display Module. This component can display four 20-character lines of white text on a blue surface with a backlight. The backlight will improve readability in low light conditions, for example in case the user has to configure the device at night and outside. The LCD turned out to be a bit bigger than expected, however we believe this turned out to be a good point for readability. The following image shows the LCD-013-420 next to a ruler for size reference.

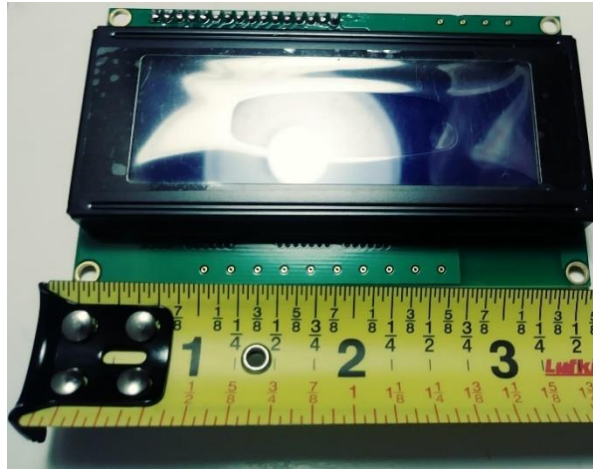


Figure 9 - LCD Module

This module uses I2C serial interface protocol to communicate with the CC3200MODA microcontroller. This screen was ideal for use in our application due to its price, size, interface protocol, and its ability to meet the functional requirements of the design. This screen can display all necessary characters essential to user understanding and manipulation. See below table for more details regarding this component[C:Superdroidrobots].

Table 10 - LCD Module Data

Description	Value
Interface	I2C
Default Address	0x27
Pins	Gnd, Vcc, SDA, SCL
Text dimensions	4 lines x 20 column
Backlight color	Blue
Text color	White
Supply Voltage	5.0V

3.3.6 Smart Speaker

There are two main smart speakers currently in the market, which are Google's Home and Amazon's Alexa (Echo, Echo dot, etc). Their functionality is almost identical from a user perspective, especially for not tech savvy customers. Price point is also very similar. Therefore, the main factor when deciding which one to use was documentation available on how to integrate with the device. Amazon's Alexa hit the market first and is well ahead of Google Home in terms of integrations already available. Due to having a larger community, Alexa was the winning smart speaker to be a part of the project. There are also solutions available where the integration could happen to both smart speakers. However, the initial setup time seemed longer then for integrating with either or. For the scope of this project it was determined by the team that only one would be necessary. Upon further research and development, this component was omitted from the final design, in favor of dedicating developmental resources to more essential features of the system.

3.4 Possible Architecture and Related Diagrams

Throughout this section the different architectural designs for Smart Garden Controller that were considered are explained and how the pros and cons of each individual architecture were valued and ranked. Every different architecture had its own positive and negative aspects to the Smart Garden Controller and thus careful planning and knowledge of each architecture was a key role in helping to decide which would be best suited for the device. The main significant influence that decided the initial design architecture could be seen by the house of quality matrix where we valued the overall cost, precision, and quality of the device. The Smart Garden Controller is meant to control at minimum two zones that will be regulated by water with minimal power consumption. It must also be able to precisely record

and log the readings it picks up with its sensors in a way that will be useful for the user to help future use of the Smart Garden Controller.

3.4.1 Design Choices

Throughout the development stage of the Smart Garden Controller many architectures were considered to help distribute the water to the different zones in the garden in an efficient and reliable way. The group wanted to make sure that there was enough space and area that the water flow controller would properly be able to handle the zones and make sure the fruits and vegetables received the right amount of water. The first system design the group had in mind had the different zones in a straight line that would use the soil, temperature, moisture, and light sensors to distribute the water to each zone when needed. The straight-line design was considered because it would allow the user to uniformly put different fruits and vegetables in each zone and be able to control them in a numerical way.

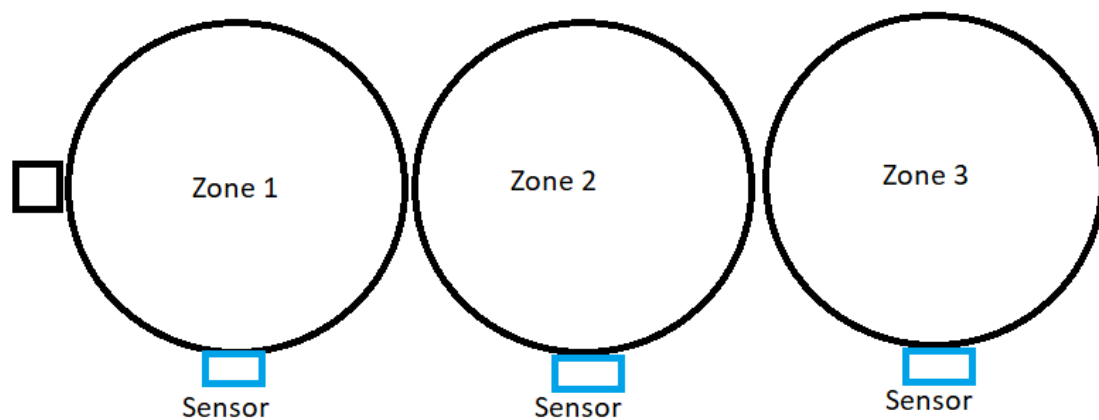


Figure 10 - Optional Zone Layout 1

The pros of this design help the user easily distinguish which zone is where and what exactly is growing without needing much assistance. However, being that the controller is on one side and the zones go in the positive direction, it can result in a very long clunky garden which may not be appealing to some customers.

Another design that our group thought up of was putting the controller in the center of the zones and having each zone go clockwise around it. This seemed to be more organized and allowed for a closer and more compact garden that could be controlled by the Smart Garden Controller and use the sensors in the same exact way as the other model. This would also allow for less temperature variance since the zones are more closely together. The sensors would trigger when they detect an abnormal condition and water the zone without much hassle.

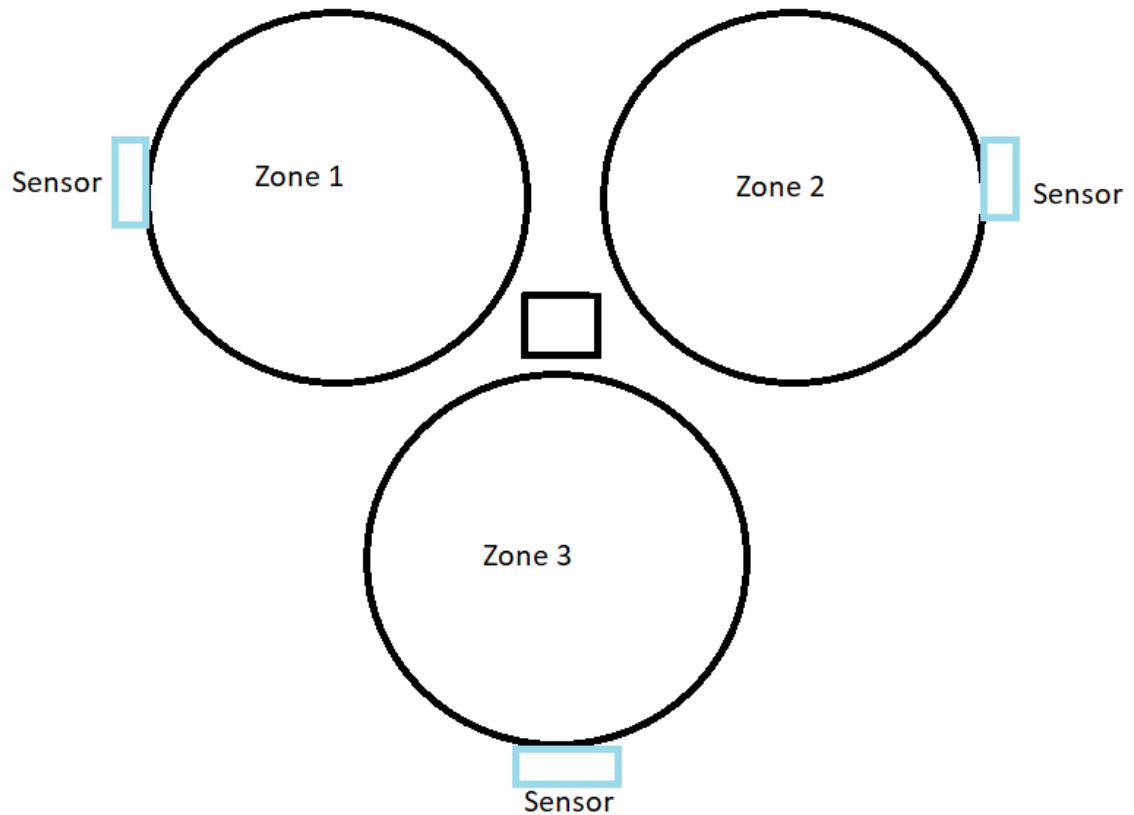


Figure 11 - Optional Zone Layout 2

The pros of this design as mentioned is that it provides the user the same number of zones but within a much smaller volume. This can be helpful because it would be able to fit into smaller areas. This design is also good because the controller is now in the center and can be viewed easily and the user would know the surrounding area is being monitored. However, it may be difficult to tell exactly which zone is which due to the zones being in a circle. This design would also improve cable length. While for the linear design we would need a cable three zones long or daisy chained, the centered controller would only need cables in length equal to the radius of a zone.

The final design implemented depends most significantly on the user's desired garden shape. The above decisions apply for prototype demonstration, however when the user installs the system, the location of their plants and sensors is determined by them.

3.5 Part Selection Summary

The table below details summaries all significant parts used in our system (see section 3.3 for more information on individual parts).

Table 11 - Part Selection Summary

Item	Name/ID	Manufacturer / Distributor	Cost
Anemometer	1733	Adafruit	\$44.95
Humidity & pressure sensor	BME280	Digi-Key Electronics	\$16.62
Moisture, temperature, and light sensor	I2C Soil Moisture Sensor	Catnip Electronics	\$13.00
Microcontroller	CC3200MODA	Texas Instruments	\$10.00
Liquid Crystal Display (LCD)	LCD-013-420	SuperDroid Robots	\$24.90

4.0 Related Standards and Realistic Design Constraints

The scope on our project and all other is limited and informed by several factors, including standards and constraints. The following sections will provide information regarding the relationship between our system and those scope limiting factors, including information about how standards and constraints effect our design choices, component choices, developer actions, and environmental considerations of the system.

4.1 Standards

A project that meets its goals while adhering to the necessary and informed standards applicable is a successful one. There are several sets of standards that are essential to the design of our automated Smart Garden, including but not limited to standards issued by the Institute of Electrical and Electronics Engineers (IEEE), The American National Standards Institute (ANSI), and other organizations accredited by ANSI. The following sets of standards apply to each subsystem utilized in our design.

4.1.1 Wi-Fi (802.11) Standards

The microcontroller we have selected can implement Wi-Fi protocols 802.11b, 802.11g, and 802.11n which is the newest protocol with the fastest maximum communication speed. 802.11n will be the target protocol to use in this system to ensure the fastest and most reliable communication environment. The IEEE standards for each protocol are included below.

4.1.1.1 802.11b Standards

The maximum transmission speed using 802.11b is 11 Megabits per second (Mbps) within a 2.4 GHz RF range and is compatible with 802.11g. The speed however can dip as low as 1 Mbps to comply with other older Wi-Fi protocols [C:Pearson].

4.1.1.2 802.11g Standards

The maximum transmission speed using 802.11g is 54 Mbps within a 2.4 GHz RF range and is compatible with 802.11b. The transmission distance range is up to approximately 150 feet or 45 meters [C:Pearson].

4.1.1.3 802.11n Standards

The average transmission speed of 802.11n is approximately 100 Mbps with maximum speed of around 600 Mbps. This protocol can operate at either 2.4GHz or 5GHz RF range[C:Pearson].

4.1.2 Power Supply Standards

The system's usage of Lithium-Ion rechargeable batteries will operate entirely within the scope of the IEEE Standard 1679-2010, which details exactly which technologies are accepted for current use and which technologies are being considered for use in stationary application. These standards dictate the following materials be used for their respective applications (taken from IEEE standard 1679.1-2017) [C:IEEEExplore].

Negative (anode)

- Unstructured/hard carbon
- Layered carbon (graphite)
- Lithium titanate (LTO)
- Silicon compounds

Positive (cathode)

- Lithium cobalt oxide (LCO)
- Lithium Manganese oxide (LMO)
- Lithiated mixed-metal oxide [Lithium nickel-manganese-cobalt oxide (NMC) or Lithium nickel-cobalt-aluminum oxide (NCA)]
- Lithium Iron Phosphate (LFP)

Electrolyte

- Lithium hexafluorophosphate (LiPF_6) salt in organic carbonate solution
- Lithium tetrafluoroborate (LiBF_4) salt in organic carbonate solution [C:IEEEExplore]

The particular power supply component that the Smart Garden will use must be compatible with the microcontroller which uses 2.3V to 3.6V (see sections 3.3.1.3, 3.3.2, & 5.8)

4.1.3 AC Power Supply Standards

After careful consideration of the use of a lithium-ion battery in our system as a sole power supply, our development team decided to use Alternating Current (AC) grid electricity as the sole power supply for our design. The system's development, testing, and demonstration will be performed in the United States; All AC power standards for the country will apply to our systems use of said power supply. Such standards have been established in order to regulate and simplify the use of such resources as widespread grid electricity. Several Bodies including IEEE and ANSI publish standards used by engineers, technicians, and users every day. By maintaining these standards especially when using AC power, our team ensures the safety and security of ourselves, our potential users, and the electrical grid and its users. Without them, our system could possibly not be compatible with a non-standardized grid system. These standards dictate the following options:

Table 12 - Power Supply Standards

Description	Value
Plug types	A, B, 14-30P, 14-50P
National Plug Standard (NEMA)	1-15, 5-15, 14-30, 14-50
Residential Voltage	120V, 240V
Utility Frequency	60Hz

Our design will be connected to a 120V source at 60Hz during development, testing, and demonstration.

4.1.4 Sensor Interface & I2C Standards

All the sensors used in the system communicate data using I2C protocol (see section 5.4). This protocol is an easy and reliable way to receive measurements from the sensors. This protocol needs only two I/O pins per sensor and can transfer data at up to 5 Mbps. This protocol is superior to others because it is used commonly in embedded devices and is synchronous, meaning that the signals sent between devices are aligned with the clock signal of our selected microcontroller. The table below summarizes the protocol [C:Circuitbasics].

Table 13 - I2C Standards

Description	Value
Wires used	2
Maximum speed (standard)	100 kbps
Maximum speed (Fast)	400 kbps
Maximum speed (High Speed)	3.4 Mbps
Maximum speed (Ultra Fast)	5 Mbps
Synchronous/Asynchronous	Synchronous
Serial/Parallel	Serial
Maximum number of masters	N/A
Maximum number of slaves	1008

4.1.5 C Programming Standards

The C programming Language will be used for Firmware on the microcontroller, and for collecting data from the connected sensors, and for interface between the LCD screen and the pushbutton hardware. All C language code will adhere to common conventions agreed upon by all participants, including but not limited to the following[C:Barrgroup]:

- Each file will have the name and Identifying number of every person who wrote that code
- All code will be commented as thoroughly as possible such that said commenting is enough for an individual to recreate the code, but not so verbose that it impedes a reasonable walkthrough of the program.
- All variables names will be in camelCase except where not applicable
- All logic control flow statements and applicable loops with a single execution line will not include brackets
- All variable names will be declared at the start of any function, except where prohibited by dynamic allocation
- All non-essential return values will be 0

- The Big-O runtime complexity and space complexity of a function will be included in a comment at the start, when applicable.

4.1.5 Java Programming Standards

The Java Programming Language will be used to write the code for the web server and database. All Java code will adhere to common conventions (similar to section 4.1.4) agreed upon by all participants including but not limited to the following[C:Oracle]:

- Each file will have the name and Identifying number of every person who wrote that code
- All code will be commented as thoroughly as possible such that said commenting is enough for an individual to recreate the code, but not so verbose that it impedes a reasonable walkthrough of the program.
- All field and method names will be in lowerCamelCase except where not applicable, and Class names will be in CapitalizedCamelCase.
- All logic control flow statements and applicable loops with a single execution line will not include brackets
- Field names do not have to be declared at the start of a method or class, only when necessary.
- All non-essential return values will be 0
- The Big-O runtime complexity and space complexity of a method will be included in a comment at the start, when applicable.
- If a class already exists to achieve desired functionality in the application program interface (API) that class will be used instead of programming a new class from scratch, when possible.

4.1.6 TypeScript Programming Standards

The TypeScript Scripting Language will be used to write code for the web application user interface. All TypeScript code will adhere to common conventions agreed upon by all participants including, but not limited to the following:

- Each file will have the name and Identifying number of every person who wrote that code

- All code will be commented as thoroughly as possible such that said commenting is enough for an individual to recreate the code, but not so verbose that it impedes a reasonable walkthrough of the program.
- All field and method names will be in lowerCamelCase except where not applicable, and Class names will be in CapitalizedCamelCase.
- All logic control flow statements and applicable loops with a single execution line will not include brackets
- Field names do not have to be declared at the start of a method or class, only when necessary.
- The 'let' keyword will be used instead of 'var' where applicable.
- All non-essential return values will be 0
- All functionality should be extracted from the 'ngOnInit' method such that only method calls and single line instantiations/assignments are present, when possible.
- The Big-O runtime complexity and space complexity of a method will be included in a comment at the start, when applicable.
- If a class already exists to achieve desired functionality in the application program interface (API) that class will be used instead of programming a new class from scratch, when possible.

4.1.7 Html Programming Standards

The Hypertext Markup Language will be used to write templates for the web application user interface. All html code will adhere to common conventions agreed upon by all participants including, but not limited to the following:

- All elements will be closed when applicable.
- All elements will be in lower case.
- All attribute names will be in lower case.
- All attribute values will be in quotes.
- No spaces will be used around equal signs in tags when possible.

- Lines will be no longer than 80 characters when possible.
- Line breaks (the '
' tag), blank lines, and indentation will only be used when necessary.

4.2 Realistic Design Constraints

When designing a new product, system, component, or process, a designer must be cognizant of the economic, political, social, environmental, safety, and many other realistic design constraints that will affect the product while it fulfills its function. These design constraints limit certain aspects of a design based on regulatory, safety, environmental, or even ethical issues. Constraints are really the bounds of what a product can function within to achieve its goals. Without any design constraints, there would be seemingly no limits on crucial pieces of the design and the design would inevitably fail.

4.2.1 Economic and Time Constraints

The cost of designing a new prototype device is usually considered to be a much higher cost than if the device were mature and in production. A product in production has the advantage of an entire system being in place and tested for procuring materials, assembling and testing the device, and shipping and servicing the customer. High volume production takes advantage of better pricing for higher quantity piece buys from vendors. For a prototype there are no high quantities involved, so typically the price per part is significantly higher. The prototype phase also carries many unknowns, as usually a new prototype design is something that has not been done before. This adds risk as there are unknowns that must be vetted before the design can be realized.

To minimize these risks, it is common to breadboard prototype designs using development boards from part manufacturers. This allows the designer to work with their selected components to prove out the concept of the design. The cost of the development boards as well as the cost of the time of the designer to create and operate the breadboard setup are also things that have a large economic impact on the prototype design. These non-recurring engineering costs (NRE) are costs that are associated with the design or development of a new product, part, or device, but they would not have an impact on the production price of the product. A lot of times nonrecurring engineering costs can be considered a "one-time fee" for a certain design service or fabrication.

Economically our project is constrained by the fact that we are an unsponsored group of students who all work but will be forced to pay for the development of this

prototype out of our own pockets. This constraint limits our ability to purchase high dollar parts and development kits, as well as limiting our ability to hire third party services companies to assist with certain technical aspects of the prototype design process. Our team goal is to keep development costs low to lessen the impact personally on each of us.

Economically, we are also constrained by the current market of existing similar products. Our product hopes to offer superior functionality at a competitive price, which means our part and component selection must take into account pricing at a prototype level, as well as at a production level. If we are able to create a device that offers a more robust feature set for a price that is comparable with an existing device on the market that contains less of a feature set, the assumption is the product would be a success.

Along with the cost associated with building a new prototype and delivering it to market for production, there is almost always a time constraint. Time constraints come from the need to get technology to the market to better compete at the cutting edge of whatever technology is being developed.

A time constraint is the timeframe for which the product is to be developed. There are many reasons for time constraints including beating competitors to market, displaying the device at a trade show, meeting a customer deadline to achieve a goal, or simply to deliver a product to a customer on time. Sometimes time constraints are imposed on design teams for seemingly no reason. Management will often do this with research and development projects to keep the design team striving towards a goal to keep the development from lagging. It is a technique that leverages the fact that it is much easier to never reach a goal if you have no deadline to meet.

The time constraints that our team is facing for this senior design project are imposed by the university faculty. These constraints require our design documentation to be completed within the first semester and for a fully functional prototype design to be completed by the end of the second semester. These time constraints were simply imposed and were portrayed at the outset of the development of this device.

The first major time constrained delivery, is the submission of the 120-page design document that is to be created in an effort to describe and justify our design. This delivery has a time constraint that it must be submitted at the end of the first semester of senior design. Failure to submit this documentation on time will result in sever action that would be detrimental to the grades of all group members.

The second major time constrained delivery is the final delivery of the fully functional prototype device. This delivery includes a demonstration of the device that is fully operational. This delivery is due at the end of the second semester of senior design. A failure to meet this delivery deadline will result in a failing grade

for senior design, and all group members would be forced to begin the process of senior design again. Like a real-world situation, this prototype delivery has a real-world impact on all members of the team if the delivery is not met.

4.2.2 Environmental, Social, and Political Constraints

Environmental, social, and political constraints deal with how products and devices effect the world and everyone who exists on it. Any product is likely to have a number of these types of constraints to ensure that the environment and society are not adversely affected by the device.

Environmental constraints that will affect the smart home garden project will largely be water usage. The use and over use of water for agricultural purposes has become somewhat of a hot button issue over the last decade. With a vastly more populated earth, resources are becoming harder and harder to come by. There is some major resistance for the use of massive amounts of water for capitalist agricultural purposes instead of supplying affordable drinking water to everyone around the planet. With this project designed to be an automated agricultural device on a very small scale, there will be an economic impact. The smart home garden is intended as an environmentally friendly device and should assist in the aid of water conservation.

The impact of water use on a home garden can be quite large depending on the crops being cultivated and the size of the garden. The smart home garden intends to use predictive algorithms with real time data in an attempt to predict the weather and save water accordingly. If the soil is wet when watering time comes the watering cycle is skipped. If rain is expected near after a watering cycle the cycle may be skipped or at least delayed to take advantage of the watering provided by mother nature. These real time analytics and predictive weather analysis will save the garden from being overwatered which will conserve water and money for the user. Beyond being predictive, when the garden does run a watering cycle it will be watering according to a known value of soil saturation. This means that since the smart home garden is in control of the watering system it can stop watering specific zones when they have reached their desired moisture level. This will aid in the prevention of overwatering and will also provide water saving benefits.

The other major environmental issue that would be caused by the smart home garden device would be the use of lithium batteries to wirelessly power the device. Lithium is not a nice chemical to deal with and must be disposed of properly. For this reason, care will be taken to provide the end user with the proper information about disposing of lithium batteries if that were to become necessary. The expectation is that the initial battery pack will be sufficiently rechargeable for the life of the device and will not need to be replaced as lithium batteries have a long lifespan. The other environmental concerns with the battery would be the volatility of lithium batteries. It is our intent to use a battery pack that contains the necessary safety PCB that prevents the battery from operating in unsafe conditions such as

over temperature or over voltage scenarios. This protection will allow the safe use of the battery pack without contributing negatively to the environment.

There are many social aspects and impacts that a new design may have on society that must be accounted for during the design and development of a new product. The social constraints for the smart home garden project include location, affordability, internet access, gardening ability, and ease of installation.

The location based social constraint stems from the fact that not all people have the opportunity to own or maintain a personal garden. The smart home garden is intended for small to medium sized home outdoor gardens. This creates a constraint for those potential users who may live in an apartment or residence where they do not have access to an outdoor yard or they may not be allowed to plant their own garden.

The affordability of the smart home garden is another social constraint because not all individuals will be able to afford this system. The intent is to keep the cost of the system as low as possible to allow for the widest range of the market to be able to purchase the smart home garden. That being said, there will still be some portion of the population that would not consider purchasing the system regardless of cost due to their lack of disposable income or the fact that they simply do not want or need the smart home garden.

Internet access is not absolutely necessary for the smart home garden, as there is an LCD screen and user interface buttons on the device to allow for user programming of the system. Without internet access the user is simply relegated to using the device as a programmable watering system that monitors soil moisture levels. This will likely leave the user feeling like they did not receive what they paid for. With internet access, the user is able to upload and download watering profiles, use programmable predictive weather techniques, gain wireless access and control of the system without having to physically interface with the device outside, and gain access to an online community for tips tricks and advice.

Prior gardening knowledge before first using the product is definitely a plus for the smart home garden. The device is being created by novice gardeners with little to no experience gardening and is intended to leverage an online community's collective wealth of gardening knowledge and automation to greatly assist a novice gardener in yielding a successful crop. That being said, prior gardening knowledge will without a doubt be helpful when operating the smart home garden. Users that may have little or no gardening know how may be unsuccessful in their initial gardening attempts.

The ease of installation of any product is a large consideration when any consumer is looking to purchase a new device. If a product is very complicated to assemble and use it is less likely that a large portion of the consumer community will be willing to sacrifice the time and effort necessary to learn how to build and use the

device. For this reason, the intent is for the smart home garden to be as simple to assemble, install, and move as possible. The intent is to have a very few number of pieces that the user must assemble, and the assembly for these parts should be simple connections that require little to no tools. Once assembled the smart home garden will be easy to connect to and initially configure so that the user can be operating their smart home garden in less than an hour from unboxing.

Research into any possible political constraints of the smart home garden system has returned very little. There are no restrictions in our country that prevent a citizen from privately growing legal crops. If a large movement of users began using the smart home garden to grow their own food it may become an issue with industrial agriculture corporations when their bottom lines are impacted. If this were to be the case the agriculture corporations could politically lobby to ban such devices but that is not the current climate.

4.2.3 Ethical, Health, and Safety Constraints

Realizing the ethical, health, and safety constraints for the Smart Garden Controller helps the team understand the potential risks of the device and how it can affect the potential user. It also helps the team take into account the accepted norms of society and what some people may consider to be right or wrong. This is important to take into account because we do not want to hurt or offend the target audience of the Smart Garden Controller and make sure that the product is delivered in the most appropriate way.

The health and safety aspect of the Smart Garden Controller is also key because the user's safety is always the number one priority and making sure nothing goes wrong is important. Without taking these constraints into account a lot could go wrong and the Smart Garden Controller may not be safe for our users and cause a lot of users to not find the device appealing.

The ethical constraints for the Smart Garden Controller are few and limited because it does not deal too much with what society is constantly changing their viewpoints on in the world. One possible argument for the ethical constraint for the Smart Garden Controller could be that it is making the gardening hobby into an autonomous process and thus ruining the experience of gardening. This is a potential deterrent to users who may think that the world is being run over by the machines and want more human interaction for their everyday tasks. There is not much the team can do about this because the overall goal of the Smart Garden Controller is to make the gardening process simpler by making it automated and providing the user data of what is happening in their garden.

The team was mindful of this ideology, but overall think it will help a lot of people that are not able to garden because of either health problems or time constraints. Another potential ethical issue is that some people may think that gardens should only have organic and the right fruits/vegetables for the area, as to not hurt the

surrounding area. This can be easily addressed by the Smart Garden Controller by providing an option to give more organic options in the web application. The Smart Garden Controller should also allow to use the recorded data to find the best fruits and vegetables for the area.

The health and safety constraints of the Smart Garden Controller are also not too major because gardening is proven to be good for your health. Some potential problems could arrive from not properly watering the zones correctly and causing some of the fruits/vegetables to be spoiled or rotten and then ingested by the user. This can be a serious problem and thus it is important for the Smart Garden Controller to be properly operating and provide the right amount of water to the right zones for the user's health. This can be easily addressed by making sure that the user confirms when they select an option for the zone they will be watering to make sure they know which zone is for what fruit/vegetable. Some other health constraints for the Smart Garden Controller may be that there are a lot of potential health hazards that gardening in general is subject to. For example, accidentally hurting yourself when using sharp tools or handling plants that have thorns are a possible issue.

There are also many insects around the garden area that can be carrying diseases that may be harmful to the user since they may be attracted to the garden. Another issue with gardening that can arise is the hot temperature that can lead to heat stroke, dizziness, or even dehydration. The hot temperature can also lead to sunburns, which may lead to skin cancer if the user is not properly protected by the UV rays. The user may also need to take into account poisonous plants and creatures, such as snakes or spiders that may be around the garden. Not much can be done to address these issues other than to help make sure that the user of the Smart Garden Controller knows basic gardening practices and that they are always protected and have the right equipment when they are out in the garden. This can be done by including a prompt when the user uses the web application that tells them to be cautious around the garden and to take care not to be a victim to one of the problems mentioned above. The Smart Garden Controller can also alert the user to stay hydrated when outside in the heat and to always have a phone nearby if they potential fall outside and need assistance.

For the actual controller itself, the device should make sure not to damage the user when it is functioning. The Smart Garden Controller waters the plants in the zones, making it important that the water supply is not contaminated and safe for the fruits/vegetables and the user. It should be placed in a safe location, such that it will not hurt the user or any potential people walking around the garden. It should be placed in an area that the water is easily spread throughout the zones and to make sure that there is enough space around each zone to not cause any problems with walking around the garden for the user. The Smart Garden Controller should also be placed in an area where it is not easily stepped on and not a health/safety hazard for the user. This will be addressed when the user receives the Smart Garden Controller and how to properly set it up, which would

not be too difficult nor take a lot of time. The device itself should not be too clunky or take up a lot of space, making a lot of these potential risks not too much of a problem to deal with.

Overall, the Smart Garden Controller does not have too many ethical, health, or safety constraints to be looked into and should help the user with their garden. The team believes the Smart Garden Controller to be a good option for a lot of people deterred with the normal garden risks because it will minimize the time spent outside in the sun and help the user record data that can be used to properly manage the garden. This opens the possibility of gardening to people who cannot be in the heat for long periods of time or have a disability that doesn't allow them to go outside as frequently to water the garden. This makes the Smart Garden Controller an amazing option for people who want to enjoy the benefits of gardening, like fresh fruits/vegetables and having a hobby in their spare time.

4.2.4 Manufacturability and Sustainability Constraints

Manufacturability and Sustainability constraints for the Smart Garden Controller is an important matter to look into because of the potential problems that can arise from them. Manufacturability for the Smart Garden Controller has to deal with how quickly and easily the device can be produced with as little resources as possible.

The overall goal of this constraint is to make the Smart Garden Controller easy to manufacture and make the process easy to reproduce for mass production. This means that to optimize the process of creating the Smart Garden Controller by using less parts and making the labor shorter and not too difficult. This helps make the product able to be created and used for the useful in a little amount of time but may limit the creation of more advanced features that can cause the design to be more complicated. It is also useful to help lower the cost for our team, especially when it comes to making the final PCB. If the team can get a part done by being completely automated it is a lot cheaper and faster to manufacture than if manual labor was required. Overall the manufacturability is helpful when the team wants to make their project appealing to consumers because it is generally cheaper and not a hassle to manufacture, as well as simplifying the overall design.

Sustainability Constraints for the Smart Garden Controller is important because the device will be running for long periods of time ideally to watch over the zones and water them whenever needed. It is also important to have it running to record the data of the soil and surrounding area for use by the user. The longer the Smart Garden Controller is able to run the longer it is able to do all of this and making it more convenient for the user, since less time would be dedicated to constantly checking back on the device. This problem is mainly addressed by the battery life and how much power consumption the Smart Garden Controller will require to operate under normal conditions. This can be optimized by making sure we do not have too many hardware elements that require a lot of power and that we can limit the amount of time the device is using unnecessary features. It can also be

optimized by making sure that the device is reliable and does not run into any problems while running, that may cause it to malfunction while watering the garden.

Overall, both the manufacturability and sustainability constraint for the Smart Garden Controller is key to making sure that the device is efficient, cheap, and works for long periods of time to make the user have a positive experience. Ignoring these valuable constraints can cause the user to not enjoy the device or end up making the manufacturing process take up too much time, given the deadlines the team is faced with. The team will keep these constraints in mind when working on the Smart Garden Controller in order to better understand what exactly to aim for when designing the final PCB layout and choosing the hardware and manufacturing process.

5.0 Hardware Design Details

This chapter gives an in depth look into the design process for this prototype and provides justification for the different electrical and computer engineering principles and practices that are utilized in this design. This chapter details out the different architectures of the different components of the design and walks through the design for clarity and better understanding.

5.1 Initial Design Architectures and Related Diagrams

The initial architecture for this design is intended to assist the user in the proper maintenance and care for a home garden. The architecture is intended to allow the user to set up and operate the system with ease. The architecture for this design was based around the fact that wireless connectivity via WiFi would play an essential role in this system. The idea was that with the ability for the system to access the internet and collect data, as well as giving the user wireless control, would set this device apart in the marketplace.

With this in mind the initial step in the system architecture design was to determine the method of gaining internet connectivity. It was determined by the team that, because the device is intended for use by the amateur gardener at home, WiFi connectivity would be the most feasible as well as offering the broadest customer base possible.

If we were to have chosen a different wireless interface that is not as commonplace as WiFi the product would likely have suffered due to that fact that most users do have easy access to WiFi but are less likely to have access to another form of standalone wireless technology. This also means that without WiFi there may have been additional hardware that would need to be supplied to the user for proper operation of the device. For these reasons and more, the decision was made to use WiFi 802.11 as our wireless protocol for this proof of concept prototype design.

After selecting the wireless interface, it was necessary to evaluate the market for the appropriate wireless chipset that would allow for our desired connectivity. Much research was put into the decision to move forward with the CC3220MODA as you can see in previous sections of this document, but the decision was based on the familiarity with the part and the use of modules from the same chipset family in other successful designs. The CC3200MODA is a robust module with a highly integrated WiFi stack and standalone WiFi controller so that the wireless interface does not burden the processor. While the option of using a chip down version of this part was kicked around, it was determined that using the module involved less risk as the amount of integrated circuitry on the module, including the antenna, could be quite burdensome to design from scratch.

Once the microcontroller and wireless interface portions of the architecture were solidified, the next step was to determine the proper process for moving, querying, and storing data from the internet. For this it was determined that there should be a web client for the user interface. This web client would be hosted by the web server that is created to handle the wireless interface portion of our system. The web server would access a database that stores all relevant data for the system as well as the user. This database is the powerhouse of the wireless infrastructure and will allow the device to operate the appropriate algorithms based on collected data for weather prediction and watering profiles that add a deeper level of intelligence to the Smart Home Garden. These aspects of the design collectively make up the wireless portion of the system that can be thought of as a small cloud for the device.

While the database and web server allow the device to operate a small functional home garden with ease using predictive weather algorithms for watering patterns is the main selling point of the device, consideration was made for users who may not have access to internet as well as a situation where a user may lose access to WiFi temporarily but still need to interface with the Smart Home Garden. For these cases it was necessary to give the user some form of manual control of the device. This would require both inputs and outputs as the user would need some verification that whatever operations are being entered manually into the device are being properly acknowledged. This led us to incorporate an LCD display as well as pushbuttons for the user. This manual user interface will give the user the device's basic scheduling functionality without having access to WiFi. This means the user is able to program a watering schedule that is zone specific, as well as being able to manually control the watering at the touch of a button.

Aside from the user interface, the device has other system components that are contained in the main chassis housing. At the top of this list is very obviously the CC3320MODA. While this wireless module handles the wireless portion of the system, it is also the master microcontroller for the entire system. All analytics and algorithms are run on this module which is able to determine complex metrics based on a variety of inputs. The CC3200MODA is the master for all communications within the system, and will collect, store, analyze, distribute, and report the appropriate data based on the configuration. The microcontroller will handle all memory allocations and data transmissions. This is really the heart and brain of the entire system.

Outside of the microcontroller, but still on our custom circuit card, there are a number of other components that allow the Smart Home Garden system to operate properly and efficiently. There is external flash memory for additional data storage. This additional storage will allow for a higher rate of data sampling which should, in turn, result in more accurate predictions. There is a JTAG interface for device programming. This JTAG interface is a standard programming interface that allows the designer to easily program the device. There is a USB to UART interface that allows terminal access to the microcontroller for debug. This interface is controlled

using an FTDI chipset that converts USB terminal data into UART data automatically so that you can use terminal functions on the microcontroller for debug and testing purposes. Also, on the main board within the chassis housing is the humidity/barometric sensor that is used to measure the atmospheric pressure in the air as well as the humidity level of surrounding air. This combination sensor will aid in the weather data collection and prediction.

To power the microcontroller and all of its peripherals an architecture for a power system needed to be designed and incorporated into our custom circuit card assembly. The initial plan for the power architecture was to use battery power to operate the device in most cases and send an alert to the user when the battery needed to be charged.

This initial architecture was determined to be unfeasible after investigation into the power consumption of solenoid valves for water flow control deemed to be far too power hungry for a purely battery powered implementation. Once the determination to move away from battery power was made, it was decided to power the device off of the grid using the user's home AC power. With this determination, a power architecture was created that implemented the use of an AC to DC power adapter that converts the higher voltage AC into a lower voltage DC. This power adapter is located outside of the Smart Home Garden chassis and will require access to a power outlet. This AC to DC power adapter feeds our custom circuit board with DC voltage so that the electronics are able to operate.

Once the DC voltage hits the board, it is run through multiple power components that translate the DC voltage to the appropriate levels needed to operate the different components on the board. Special care was taken in choosing these power components to ensure that the current draw of downstream components would be easily handled by the power design with enough overhead to account for worst case situations.

Outside of the main chassis housing is where the real purpose of the Smart Home Garden becomes clear. Cabled into the main chassis housing is a series of sensors that are placed in the user's different gardening zones. These sensors are able to collect data about the moisture levels of the soil, as well as collecting temperature and ambient light data all individually. These sensor cables are coupled with a hose that will provide each zone the ability to water separately when it is appropriate.

The intent is that the user runs the coupled cable and hose to the desired location, which would likely be in the center of what is being considered by the system a zone of the garden. The sensor is able to collect data that is specific to that zone, and this will allow the user to adjust the watering cycle for each specific zone based on what that zone may require dependent on its crop. The hoses will have sprinkler heads attached to the end so that each zone can be watered effectively using the

proper sprinkler head. The watering is controlled by solenoid valves that are actuated by the microcontroller.

The valves are normally closed unless the proper voltage is introduced across the terminals. This means that the water to the system would have to be on and ready so that when the system determined watering was necessary, there would be water available. Users should dedicate a certain spigot to the Smart Home Garden so that this spigot can be left on to supply water to the system at all times.

This overall system architecture is what has led this project down the path to create the design we have today.

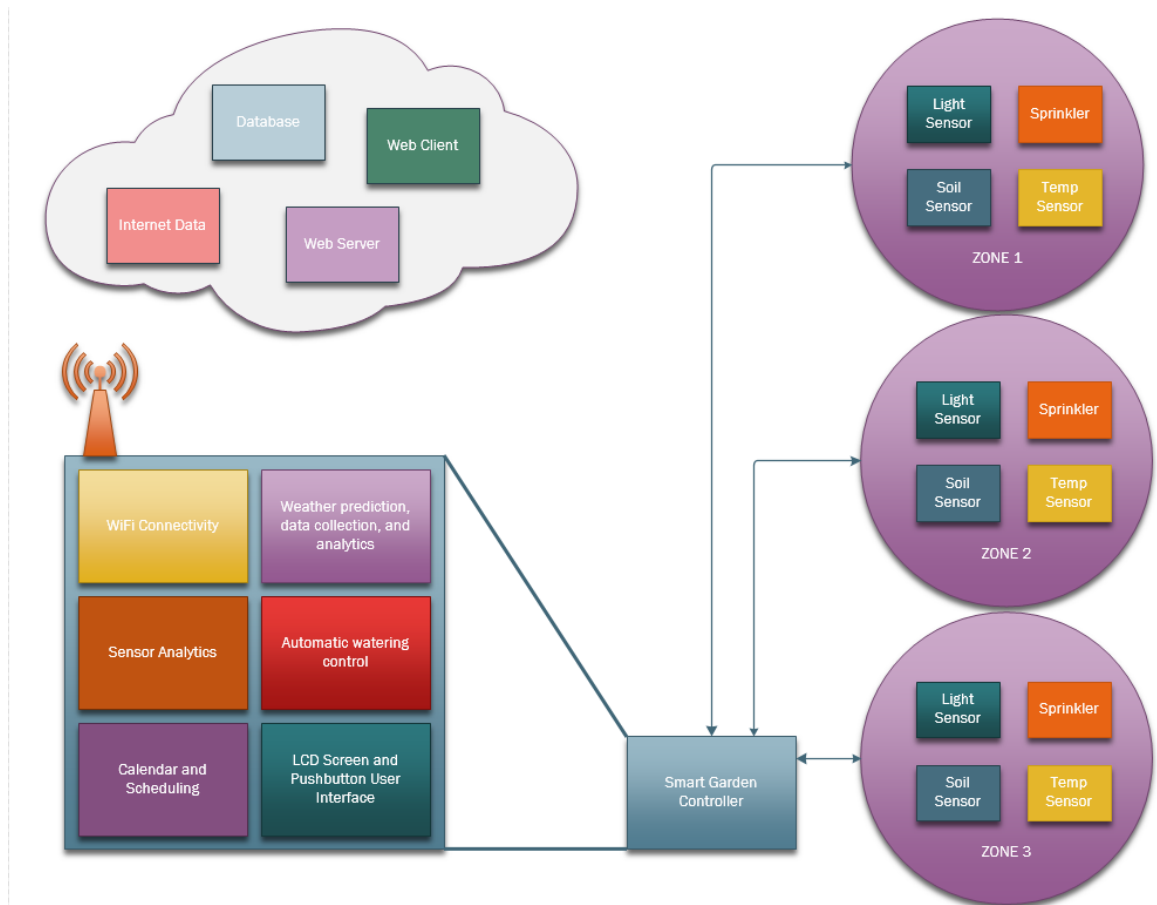


Figure 12 - System Architecture

5.2 Microcontroller

The Texas Instruments CC3220MODA is a WiFi certified wireless MCU module. This specific version of the chipset was chosen based on the included features and functions of the device as well as the low risk option of using a module that has an integrated antenna which helps relieve the burden of RF design from the

team. The CC3220MODA is an FCC, IC, CE, MIC, and SRRC certified wireless microcontroller module with built in wireless connectivity that integrates two physically separated on-chip microcontroller units. The two processors consist of an ARM Cortex M4 processor with 256KB of RAM and optional 1MB of serial Flash, alongside a wireless network processor microcontroller unit that runs all WiFi and internet logical layers. The network processor uses a ROM based subsystem which includes an 802.11 radio, baseband, and MAC. The network processor also contains a crypto engine that allows for 256-bit encryption for additional wireless security.

5.2.1 CC3220MODA Power

The CC3220MODA has a recommended operating voltage range between 2.3 volts and 3.6 volts. We chose to run the CC3220MODA using a 3.3-volt power rail as that is the widely accepted standard for many microcontrollers. Careful consideration was taken into account when choosing the power components for this subsystem to ensure that the maximum required current draw for this part in conjunction with any other components that may share this power rail, will not exceed the maximum current rating of the power component.

For the CC3220MODA, 3.3-volt power is applied to only two pins of the module, VBAT1 and VBAT2. These pins are called VBAT as the intended use by the manufacturer is that this microcontroller module should be utilized for Internet of Things devices which are usually small battery powered electronics, hence VBAT. These pins are designed to have small value decoupling capacitors placed very close to the module physically.

These capacitors aid in noise reduction on the power rail and they are said to decouple the underlying circuit from the noise from the rest of the circuit environment. Three decoupling capacitors are used for the CC3220MODA module. These two power signals feed the internal power management system of the module with the supply current needed to run the module and all of its functions. Aside from the two power supply pins on the module, there are 9 ground pins and 9 thermal ground pads under the part. These 20 pins and 3 decoupling capacitors collectively create the overall power system that runs the module.

Table 14 - CC3220MODA Power Specs

CC3220MODA Power Specifications			
Absolute Maximum ratings			
	Min	Max	Units
Vcc	-0.5	3.8	V
Digital I/O	-0.5	V _{cc} +0.5	V
RF pin	-0.5	2.1	V
Analog Pins	-0.5	2.1	V
Operating Temp	-40	85	C
Storage Temp	-40	85	C
Junction Temp		120	C
ESD Ratings			
		Value	Units
Electrostatic Discharge		+/-500	V
Recommended Operating Conditions			
	Min	Max	Units
Vcc	2.3	3.6	V
Operating Temp	-40	85	C
Ambient Thermal Slew	-20	20	C/minute
Current Consumption			
		Typical	Units
MCU Active		286	mA
MCU Sleep		285	mA
MCU Hibernate		5	uA
MCU Shutdown		1	uA
Peak Calibration Current		450	mA

5.2.2 CC3220MODA No Connects

Aside from the power and ground pins that make up the module, there are a number of pins that are marked by the manufacturer as NC or no connect pins. These pins are intentionally left unconnected by the designer. The exact reason for this is not perfectly known to us currently, but according to the CC3220MODA datasheet, these pins are reserved and are meant to be left floating unconnected. Our assumption from this is that these pins do have a function relative to the module, but this function is not necessary knowledge for our development so the pins will simply be left unconnected as intended by the design.

5.2.3 CC3220MODA JTAG

Now that we have power to the module and we don't have to worry about all of the no connect pins, we can utilize and exercise the different interfaces of the device. The first interface we will discuss may be the most important of all due to the fact

that without it, we would be unable to program the device easily. This is the JTAG interface. JTAG stands for Joint Test Action Group and is an IEEE standard that defines a test access port for digital integrated circuits and provides a standardized serial interface to control test logic. The JTAG interface is what we used to program our hardware with our developed firmware. JTAG is a widely accepted standard, therefore it is quite versatile in its interfacing.

Our design intent is to use an off the shelf JTAG programmer in conjunction with Code Composer Studio to program the CC3220MODA. This programmer interfaces to our custom circuit card assembly via a small ten pin connector that is standard for JTAG connections. This small header connector is routed to the appropriate pins on our microcontroller module so that we have the proper access to that portion of our device.

The pins that are interfaced to through this connector are defined in the CC3220MODA datasheet as JTAG_TDI, JTAG_TDO, JTAG_TCK, and JTAG_TMS. While it is not defined in the same way the other JTAG pins are by the datasheet for this module, the nRESET pin is a very important part of the JTAG interface as it is nearly always required to either reset the device to begin a JTAG process, or to hold the device in reset while some JTAG process is taking place. Either way it is important for the designer to have access to all of these signals when programming and debugging the system.

The different names for the JTAG signals are quite intuitive when you know what to look for. TDI is data in, TDO is data out. TCK is the clock signal. Finally, and probably the least intuitive of the four signals TMS, which is test mode select. TMS is sampled at the rising edge of the TCK clock and is used to determine the next state.

5.2.4 CC3220MODA Reset

Since we began talking about the nRESET pin in the previous paragraph, we will finalize its discussion now. The nRESET signal into the module is an external input to the module that effectively resets the module when the input signal is held low. This allows the designer to design an opportunity to create an easier test environment during development by adding a button attached to this signal with the appropriate logic that will perform a reset when the button is pressed. This reset signal may also be used in normal device operation.

It is common practice to put a hard-reset button onto electronic devices as they are prone to software malfunctions that require a device reset to alleviate. It is our intention to route the reset signal to the JTAG connector as well as routing it to a pushbutton that can be used both by the developers as well as the end users.

The final aspect of the reset signal design is the fact that it is an active low input, which indicates that when that pin sees a logic level 0 it will perform a reset

function. With this in mind it is a good design practice to place a pull-up resistor onto the nRESET signal so that if there are any transient signals or noise on the reset line, they will not accidentally fall below the hysteresis of the signal producing a false reset input and mistakenly resetting the module. This pull-up resistor puts the reset signal in a known state that must be actively pulled low by either the JTAG programmer or by the user pressing the button.

5.2.5 CC3220MODA Configuration Pins

Now that we are able to power on the module, program it, and reset it, we can move on to some configuration interfaces in our design. The first configuration interface we encounter with the CC3220MODA is the sense-on-power interface. The sense-on-power, or SOP, interface is used to determine the proper boot sequence when the module is powered on. The CC3220MODA has multiple boot options including, UARTLOAD, FUNCTIONAL_2WJ, FUNCTIONAL_4WJ, UARTLOAD_FUNCTIONAL_4WJ, and RET_FACTORY_IMAGE.

UARTLOAD is defined in the datasheet as “Factory, lab Flash, and SRAM loads through the UART. The device waits indefinitely for the UART to load code. The SOP bits then must be toggled to configure the device in functional mode. Also puts JTAG in 4-wire mode”.

FUNCTIONAL_2WJ mode is defined in the datasheet as “Functional development mode. In this mode, 2 pin SWD is available to the developer. TMS and TCK are available for debugger connection”.

FUNCTIONAL_4WJ is defined in the datasheet as “Functional development mode. In this mode, 4-pin JTAG is available to the developer. TDI, TMS, TCK, and TDO are available for debugger connection. The default configuration for CC3220MODx and CC3220MODAx modules.”

UARTLOAD_FUNCTIONAL_4WJ is defined in the datasheet as “Supports Flash and SRAM load through UART and functional mode. The MCU bootloader tries to detect a UART break on UART receive line. If the break signal is present, the device enters the UARTLOAD mode, otherwise, the device enters the functional mode. TDI, TMS, TCK, and TDO are available for debugger connection.”

RET_FACTORY_IMAGE is defined in the datasheet as “When module reset is toggled, the MCU bootloader kickstarts the procedure to restore factory default images.”

The SOP pins can be configured so that each of these boot procedures is followed according to the needs of the developer. The module contains internal pull-downs on the SOP lines and the datasheet claims that these signals may be multipurposed once the device is powered up, but for our purposes, and for the

purpose of not mistakenly interfacing with the boot pins during normal operation, in our design these SOP pins are single purpose solely for boot procedure.

Table 15 - CC3220MODA SOP Config

Sense-on-Power Specifications				
Name	SOP[2]	SOP[1]	SOP[0]	SOP Mode
UARTLOAD	Pullup	Pulldown	Pulldown	LDfrUART
FUNCTIONAL_2WJ	Pulldown	Pulldown	Pullup	Fn2WJ
FUNCTIONAL_4WJ	Pulldown	Pulldown	Pulldown	Fn4WJ
UARTLOAD_FUNCTIONAL_4WJ	Pulldown	Pullup	Pulldown	LDfrUART_FnWJ
RET_FACTORY_IMAGE	Pulldown	Pullup	Pullup	RetFactDef

The final configuration interface for the CC3220MODA are the antenna selection pins, ANT_SEL1 and ANT_SEL2. This signals control RF switching that is performed within the module when an external input requires it. The switching between antennas is not necessary for the purposes of our design but the datasheet is not overtly clear on the total functionality of these pins and what impact they may have on the modules performance. For this reason, we have decided to connect the ANT_SEL configuration signals to test points. This will allow us to analyze these signals within our design environment and will also give us the ability to interface with these signals via white wire rework if such a rework is deemed necessary.

5.2.6 CC3220MODA FLASH Serial Peripheral Interface (FLASH_SPI)

Now that we have the configuration pins taken care of we can turn our attention to the communication interfaces of the CC3220MODA. The first interface that we will address is the FLASH_SPI communication interface. This is the Flash memory specific serial peripheral interface that is intended for use with an external flash device, which is our intention for this design. This interface consists of 4 signals. FLASH_SPI_MISO is the master in slave out data signal that will transmit data from the module to the flash device. FLASH_SPI_MOSI is the master out slave in data from the flash device to the module. FLASH_SPI_CLK is the interface clock signal that is generated by the master, which in this case is the CC3220MODA.

Finally, the FLASH_SPI_nCS_IN is the chip select signal that addresses the device individually if there are multiple devices on the SPI bus. In our design there will only be a single device on this SPI bus so we will not have to worry about any sort of chip select signal conflicts or data miscommunication. In the Smart Home Garden design, these SPI interface signals are routed directly to the Flash memory device. Zero-ohm resistors may be an optional addition to the signals, in series, so that if there are any schematic or layout errors the resistor can be easily removed and reworked to fix the issue.

5.2.7 CC3220MODA GPIOs

The final pin assignment effort that we must undertake to complete the CC3220MODA hardware design is the assignment of the many general-purpose input/output pins, more commonly known as GPIOs. The CC3220MODA has a total of 21 GPIO pins that can be used exactly as the name implies, for general purpose inputs and outputs. These signals are often used as interrupts or reset control signals when a single logic state change can affect a result on the device being interfaced with. Beyond being just general-purpose pins, the CC3220MODA allows for multiplexing of these pins so that they may be configured in a number of different ways to communicate over a multitude of interfaces. We leveraged some of these multiplexed signals in our design, which will be discussed later in this chapter.

Our architecture and functional design require the use of multiple GPIOs for our system to operate optimally. We need 3 GPIO signals to control the solenoid valves for the watering system. These signals are utilized as simple logic signals to control a high-side switch to activate and open the solenoid valve. These change in logic state events are triggered by the programming of the CC3220MODA. There are an additional 3 GPIO pins needed to operate on board debug LEDs that were used by the developer and the programmer to debug and test different device functionality.

Finally, our system requires the use of 5 user interface pushbuttons to operate the device without having WiFi connectivity. These pushbuttons are simple logic using pull-ups to determine when a button has been pressed. Any button debouncing that is performed is done by the device firmware as it is assumed as less risk to compensate for debouncing with software rather than attempt to install a debouncing circuit on our hardware.

Those are all of the GPIOs that our current architecture requires, but it has been found good practice to break out unused GPIO signals in the case where an additional function or feature may be easily added if the hardware interface is available. With this knowledge we will break out some of the unused GPIO signals to a header connector so that we may add additional components or features to the design at a later stage without having to invest in completely new hardware.

5.2.8 CC3220MODA GPIO Pin Multiplexing

The CC3220MODA makes use of extensive pin multiplexing to offer a wide variety of peripheral functions while maintaining a small footprint and without having an overabundance of pins. Pin multiplexing on this module is handled using a combination of hardware configuration at module reset and register control. It is detailed in the datasheet that the pin multiplexing configuration is the responsibility of the developer and extra care must be taken in deciding the appropriate pinmux architecture so that the designed system is able to meet all of its goals.

Luckily for us Texas Instruments provides us with the PinMux tool that is a Texas Instruments specific tool that targets a specific module or integrated circuit and allows the developer to select the different interfaces and functions that may be required and automatically assigns that configuration to the available pins. This is extremely helpful in determining which interface to put on which multiplexed signal as there are many options for each. For our system we will require the module to be multiplexed to include interfaces for JTAG, I2C, and UART. We described the JTAG interface in an earlier section of this chapter, so we will go into no further detail on it here other than to say the JTAG signals are assigned to pins 12, 18, 20, and 22.

5.2.8.1 CC3220MODA I2C Interface

The CC3220MODA includes one I2C interface that operates in either standard transmission mode (100kbps) or fast transmission mode (400kbps). The I2C interface consists of 2 signals, SDA and SCL. SDA is the data line that transmits the data back and forth between the controller and the device. SCL is the clock signal that is used to synchronize the timing between two devices so that they may handshake and exchange data.

The I2C interface is an integral part of the Smart Home Garden design as we have optimized our communications interface to be primarily I2C based which is anticipated to save time and effort during the coding phase. The intention there is to focus more on the wireless interface and the web client rather than the hardware communications interface. That being said, the I2C bus will house the 3 soil moisture sensors, the LCD display, and the Barometric/Humidity sensor. In using the Texas Instruments provided PinMux tool it was determined that the ideal multiplexing location for the I2C interface is to utilize pin 3 for SCL and pin 4 for SDA. It is worth noting here that I2C signals do require pull-up resistors to function properly and these pull-ups were designed in.

5.2.8.2 CC3220MODA UART Interface

The CC3220MODA contains 2 UART interfaces that both include programmable baud rate generators, separate RX and TX FIFOs, programmable FIFO length, and many other very useful features of UART. For the purposes of this design and our project, the UART interface is going to be used primarily for debug and development purposes. The intention is to route the UART signals to a common FTDI USB to UART device that will allow us to interface with the microcontroller on our board through a terminal that we can access simply through a USB connection.

This feature was immensely helpful during the software development and test phases of this design and will allow us to better view what is going on inside of our code and what needs to be done to optimize it. The UART interface signal pin

assignments were also derived from the Texas Instruments PinMux tool. The pin assignments for the UART signals are pin 46 is TX, pin 47 is RX, pin 44 is CTS, and pin 51 is RTS.

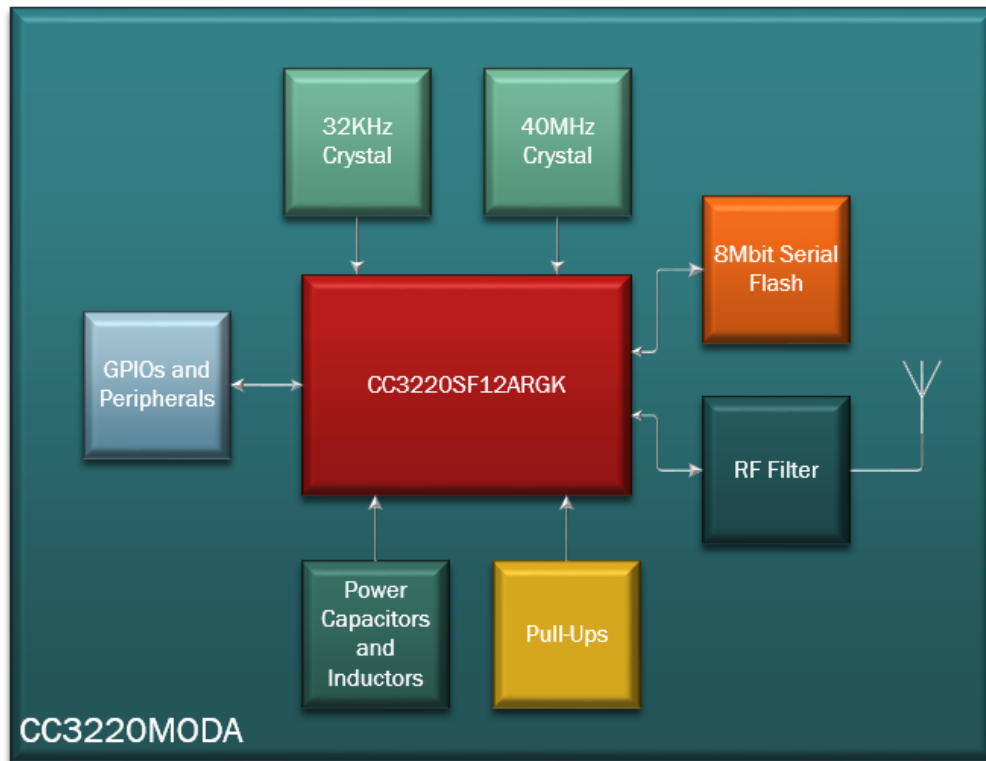


Figure 13 - CC3220MODA Block Diagram

5.3 Sensor Interface

The Automated Smart Garden employs several different sensors to collect data regarding the subjects of the garden and the condition of the surrounding environment. All of this data is made available to the user via the web-based user interface and the LCD screen manual user interface as requested. The sensors measure and record the following metrics, in order of importance to goal of the system: Soil moisture level, Temperature, ambient light levels, humidity, and atmospheric pressure.

5.3.1 Moisture, Temperature, & Ambient Light

Each plot in the automated Smart Garden includes a soil moisture sensor that records temperatures and ambient light levels along with its main moisture data

collection (see section 3.3 for more sensor equipment details). The I2C protocol is used to transfer recoded data to the microcontroller (see section 4.1.3 for more information on I2C protocol). This sensor will have a single connecting wire to the microcontroller on the PCB housing a serial data line (SDA) and a serial clock line (SCL).

Data collected by this sensor is read by the firmware of the microcontroller and saved to the nonvolatile memory on the chip, where it is periodically read by the web-based user interface to aggregate and display the data to the user in more useful ways. Three individual records are maintained, one for each metric, these records are as dense as necessary, saving values periodically as frequently as is practical and possible given the constraints of memory and the firmware's ability to process data (see section 4.2 for more information on limitations & constraints). That data is also available to the user via the Liquid Crystal Display (LCD) screen on the device (see section 5.5 for more information regarding the LCD display).

5.3.2 Wind Speed

The automated Smart Garden will include an anemometer to continuously record wind speed (see section 3.3.3 for more information on the anemometer). Since this system's scope is restricted to monitoring a small garden sized area, only a single anemometer would be used to record a single wind speed metric (see section 4.2 for more information on constraints affecting the system's scope).

The wind speed measurements will not be taken on a uniform schedule of discrete instants like the other metrics but instead periods of several measurements would be taken during periods of atmospheric activity. This data recording method is designed to practically respond to the random and unpredictable nature of the environment, and to take advantage of the analog data delivery system.

When the environment is demonstrating low atmospheric activity with wind speeds equal to or significantly close to 0 m/s, the anemometer will apply a voltage to the PCB equal to or significantly close to 0.4V. During these periods of inactivity, the firmware will consider wind speed negligible, and indefinitely record wind inactivity.

During periods of measurable atmospheric activity with wind speeds up to 32.4 m/s (72.47674 mph) the anemometer will apply the corresponding voltage to the PCB, between 0.4V and 2.0V. During this active period the voltages would be recorded and converted to measurements in meters per second (m/s), then saved in memory by the firmware.

During periods of extreme atmospheric activity with winds speeds greater than 32.4 m/s, or consistent wind speeds equal to or significantly close to 32.4 m/s, assuming the anemometer is still functioning normally, a consistent 2.0V signal would be applied to the PCB. Such measurements indicate hurricane force winds[C:NOAA]; this case is unlikely and the system is not recommended or designed for use in such conditions.

After extensive research and consideration, our team decided to omit the implementation of wind speed measurement from the final design of the prototype. This decision was made to dedicate more developmental resources to maximizing the system's functionality regarding the design requirements. Upon an increase in the scale of this design or further development, wind speed measurement should absolutely be implemented.

5.3.3 Humidity & Pressure

The automated Smart Garden will include a Humidity and Atmospheric pressure sensor [C:Mouser]. Similar to the sensor in section 5.4.1 this sensor will communicate with the PCB via I2C protocol (see section 4.1.3). Since this system's scope is restricted to monitoring a small garden sized area, a single sensor is used (see section 4.2).

Atmospheric pressure data is collected by the sensor and sent to the microcontroller in terms of hectopascals (hPa). Humidity data is measured and sent to the microcontroller in terms of a value between 0 and 100 (inclusive); this scaler corresponds to a percentage of relative humidity, from 0% to 100%. Both of these metrics are sampled at practical intervals determined considering memory and software constraints.

The environment should never present conditions such that the humidity measurements are out of range, due to the relative nature of that measurement. The environment is not expected to exit the range of atmospheric pressure sensor which is 300 hPa (0.296 atm) to 1100 hPa (1.086 atm); should measurements of extreme low pressure persist, this would indicate a pressure drop associated with catastrophic storms; the system is not designed or for use in such conditions.

5.4 LCD and Pushbutton Interface

The Smart Garden Controller houses a Liquid Crystal Display (LCD) screen in the custom enclosure with the PCB. This screen in conjunction with five pushbuttons comprise a manual user interface on the device itself. Using this interface, the user will configure their device with the username and password they created via the web application, and the user number assigned to them. After entering that personal information, the user will use this interface to configure the device's connection to their home WiFi network, by entering an SSID and, if applicable, a password. After these two configuration steps are completed, the user is not required to use the manual interface again (barring any change in WiFi network information or a reset of the device). This manual interface will however continue to provide the following optional functionality:

- SoilSensor() – when selected this option will show all soil sensors connected (up to three), and their most recent corresponding temperature, ambient light, and soil moisture readings.
- AirSensor() – when selected this option will show the most recent readings from the air sensor: relative humidity, atmospheric pressure, and temperature. This reading refers to the temperature inside the device itself, which is usually 5°C - 10°C higher than the temperature readings taken by the soil sensors. This is due to the heat generated by PCB components, and often the heat absorbed from sunlight, because the device is in a black box and almost certainly outdoors.
- OnDemand – when selected this option will prompt the user to select a zone and a duration for watering, and a waterNow() option. The waterNow() option will immediately initiate watering in the selected zone for the specified duration.

5.5 Water Flow Control Subsystem

The water flow control subsystem consists mainly of the solenoid valves that perform the opening and closing operation that allows the flow of water down the hoses and through the sprinkler heads to water the garden. This is the crucial feature of the entire system and without it, the garden would fail.

The solenoid valve that were selected for use in this project were selected for their cost effectiveness as well as their availability and user reviews. As discussed previously in this document, the original intention was for the system to run continuously for an extended period of time on battery power, but it was the solenoids that really constrained that requirement. The solenoid is an electromechanical device that uses the current flow and generation of magnetic field to create mechanical movement that opens a valve. Unfortunately for us this mechanical movement does not come cheap as far as the power budget is concerned. Ultimately the current consumption of the solenoid valves made it unrealistic to operate the system with battery power. Once this determination was made the design was adjusted to use AC grid power, which is a situation where power hungry components have a significantly smaller impact on the overall design.

The solenoid valves that were chosen for this project design are 12-volt devices. The functionality of the device is simplified to the point where when 12-volts is applied to the device, if there is sufficient current from the source, the solenoid valve is open and held open until the power source is removed. In that state the solenoid valve has a typical current draw of 320mA. The major concern of attempting to run off of battery power is that when the garden requires watering, for every minute that the water is distributed to the zones, a significant amount of

power is being consumed by the solenoid devices. The power consumption led us to believe the battery would have to be recharged far too often to be considered a beneficial feature, so the battery design aspect was scrapped.

The fact that the solenoid valves run on 12-volts and have a significant amount of current draw led to an interesting design challenge. We need the microcontroller to activate the solenoid, but the microcontroller operates at 3.3-volts and only consumes an average of 180mA itself, so there is no way it would be able to supply the power to open the solenoid valve itself. For this reason, it was decided that the use of a high side switch would be an easy to implement and effective solution for this problem. The high side switch component is an integrated circuit that, for all intents and purposes, contains two MOSFETs. One of the FETs is a PNP and the other is an NPN. In the high-side switch configuration the P-channel FET is has its source connected to the source that we are attempting to supply with power, in this case the solenoid valve. The drain of the P-channel FET is connected to the power source that is intended to power the device. The gate of the P-channel FET is connected to the source of the N-channel FET. The drain of the N-channel FET is connected to ground or some other reference depending on the design. The gate of the N-channel FET is then connected to the control signal, for our purposes this is considered to be the signal from the microcontroller. There are some passive components that need to be added as well to determine slew rate, add protection, and add decoupling, but for the most part the high-side switch is just the two FETs.

What this high side switch configuration allows us to do is to turn on a higher voltage level P-channel FET using a lower voltage level N-channel FET. This allows us to control a 12-volt part using 3.3-volt logic, as long as the 12-volt power supply is active. This high side switch configuration is an easy and common way to control higher voltage components without risking damage to the controller components.

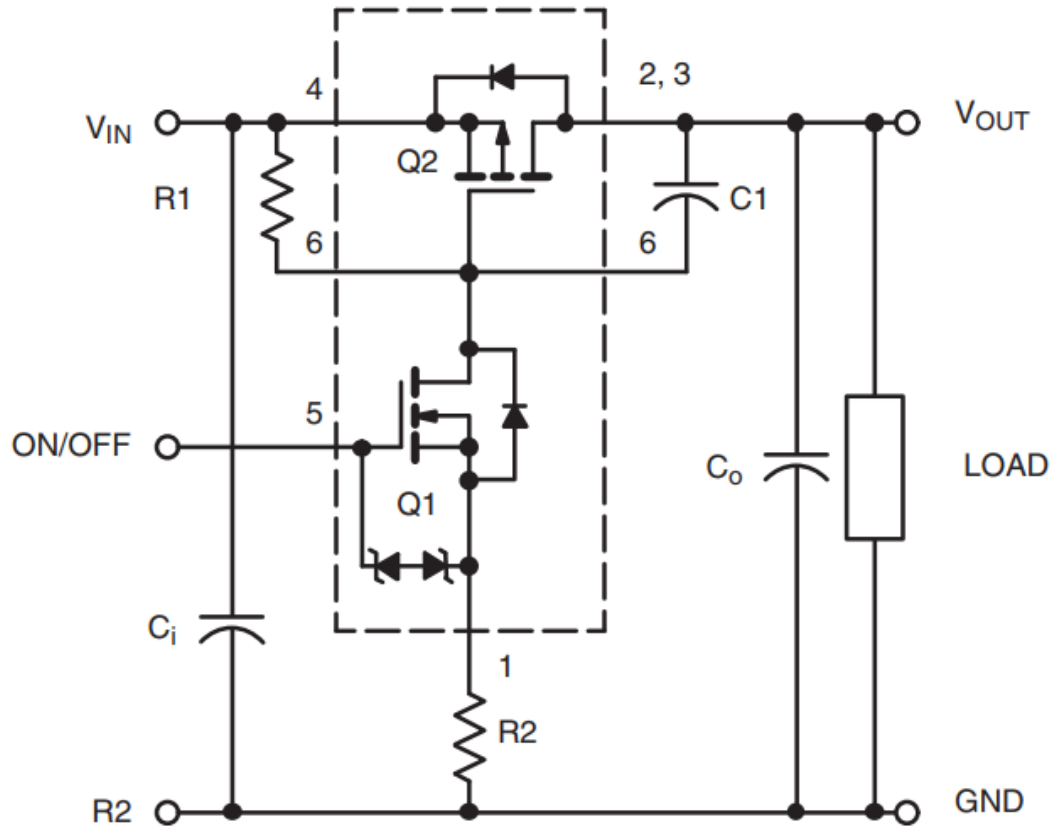


Figure 14 - High Side Switch Configuration

5.6 Power Subsystem

The power subsystem for the Smart Home Garden design is comprised of multiple levels of voltage conditioning to allow the appropriate power environment to be created for an optimal design. The design of the power subsystem begins with the input. In the case of the Smart Home Garden, the power input is a wall-wort type of AC-DC power adapter.

This power adapter has been investigated and evaluated to determine that it meets the necessary minimum specification to operate optimally within our design. The design that we chose has a maximum output current of 1.5 Amps and delivers this current at a 24-volt potential. While there are many devices that exist that would allow for a much higher current consumption by the system, but this model was settled on because it was the most reasonably priced option that met all of our immediate needs and requirements. This power adapter is provided with the system for each user.

The AC to DC power adapter is plugged into a standard wall outlet and the barrel connector that is attached to the end of the adapter cable is plugged into the Smart Home Garden System. This supplies the Smart Home Garden system with 24-volts from the adapter. This input voltage will immediately pass a protection circuit that will include a fuse as well as a transient voltage suppression circuit to prevent damage to the Smart Home Garden System from electrostatic discharge or any inrush current conditions. Once the input power has passed through the protection circuit, it is routed to a DC-DC power supply that will convert the 24-volt input into a 12-volt power rail.

This 12-volt power rail is the driving force behind the solenoid valves and is controlled using the high-side switches that were discussed in a previous section. From the 12-volt power rail we route the signal to another power management device that will provide us with the ability to generate a 3.3-volt power rail. This 3.3-volt power rail will power the microcontroller as well as all the sensors, the debug components, and the flash memory.

5.7 Summary of Hardware Design

The Smart Garden Controller our team designed will help in assisting the user by making the home gardening experience a lot simpler and automated, allowing the user to spend less time worrying about when and how to water their fruits/vegetables. The Smart Garden Controller will supervise a select number of zones for the user and water them based on what the user wants or a preset option that the team has inputted for a specific type of fruit/vegetables. The use of water as a resource is important to the device and thus making sure to conserve as much as it and be as efficient with it is important. It will also record data from the sensors for the user about the garden, such as temperature, humidity, soil, wind, etc., allowing the user to utilize it for further use. In order to achieve this, the Smart Garden Controller must be equipped with many sensors that allow it to record such data in a reliable and accurate way.

These components would have to be attached to our microcontroller in order to function and allow for the device to interpret the data so it can know when a zone would require water or if something was to go wrong. The Smart Garden Controller would also have Wi-Fi capability to connect to the internet and LCD display that allows the user to be able to view the scheduling of the garden and all the data being recorded. The device should be able to withstand the normal weather conditions of a typical garden and be as compact as possible.

As previously mentioned the team decided to use the Texas Instruments CC3220MODA microcontroller because of the lower power it requires, which overall gives it a longer battery life for the user. The team agrees that Texas Instruments is a good choice because of the many different resources they have, as well as the Code Composer Studio it uses. It was also chosen for its simple Wi-Fi capability and the lower voltage it is able to operate under. This version of the

microcontroller also comes with an antenna that will prove to be useful in helping with wireless information. Choosing the right microcontroller was essential to the Smart Garden Controller because it acts as the brains of our device and controls all of our components.

The sensors chosen for the Smart Garden Controller were also vital because it allows for us to record and use the data. It is key that the device has accurate and reliable readings because the team wants to give the user the best experience possible. This is why the team has chosen the Catnip Electronics Soil Moisture Sensor (2.7.5), since it gives data for the temperature, light, and moisture of the soil. The price of it was reasonable for all the data it will provide and how accurate it will be. The next sensor the group picked was the Bosch BME280 sensor that will read the humidity and atmospheric pressure of the garden. This sensor is good for the Smart Garden Controller because was also fairly cheap and had good measuring capabilities. Most importantly, both the above sensors have I2C communication protocols, allowing for simple data transfer between the devices. One final sensor that the team picked for the Smart Garden Controller was the anemometer, which measures the wind speed and direction of the garden. It was important to include this sensor because it can allow the user to know when a storm occurred.

The next step was to find the right water flow controller and sprinkler heads in order to provide the right amount of flow rate to our garden and an LCD monitor in order to display the data and allow the user to interact with the Smart Garden Controller. These were important to choose because if we do not have a good flow rate and distribution of our water, it may not properly water the zones or use an excess amount of water, when it could be more efficient with it. The sprinkler heads we used will allow for the zones to be equally distributed with the water to make sure no areas of the gardens' zones are left dry.

6.0 Software Design Details

In this section we will go over all the software needed used in the project. This section be split into five main parts. The first part is the code which will control the microcontroller. The second part will be the software keeping the database. Then the web user interface. Finally, the server which will serve all the data, to both the controller and the web interface, this software will be a middle man between the controller and clients. Very complex software would be needed in order to provide most of the functionality given by web server, application, and database. Therefore, open-source software which already provides such functionality will be used. This way the software design can be focused on the parts which are specific to this project. Such software will be covered in this section as well where relevant.

The MCU and the web app will communicate to the web server via representational state transfer (REST). REST or “Representational State Transfer is an architectural style that defines a set of constraints and properties based on HTTP. Web Services that conform to the REST architectural style, or RESTful web services, provide interoperability between computer systems on the Internet. REST-compliant web services allow the requesting systems to access and manipulate textual representations of web resources by using a uniform and predefined set of stateless operations” [1]. In other words, both the controller and the web interface will retrieve data from the database via HTTP requests. The results of such requests will be in plain text formatted either JSON or XML, which can then be processed by either the controller or the web user interface.

The web server will communicate with the database directly using a database connector. The connector is the code called to perform queries against the database and return the data back to the web server. The following image is a high graphical representation of the communication between the controller and web user interface to the web service. Because each of these sections are decoupled, not dependent on each other, development should be simplified.

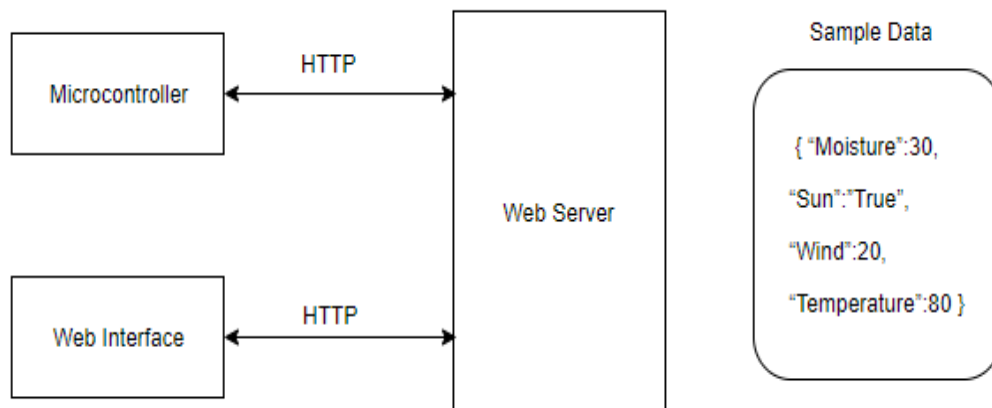


Figure 15 - Web Interface Block Diagram

6.1 Database

The database will hold all the data collected by the sensors, user data, etc. The controller will periodically send data to the web server. The web server will process the data and insert it into the database. The data in the database is used by the web application and by the microcontroller. The user via the web application will connect to the database to review the data collected by the controller. The user, based on the data collected, can then make decisions for future watering schedules, whether to increase or decrease watering. The microcontroller will connect to the database and use it mostly for scheduling. The database is the “memory” of the system.

For the database design we need first to decide what data will be stored. Then the data will be distributed amongst different tables. Once all the tables are created we must normalize the data. By normalizing the data, we will ensure that no data is duplicated or unnecessary. The first level of normalization is to ensure that no columns are repeated on a table. The next level of normalization is to ensure columns are not repeated on different tables, unless deemed necessary. If a column is needed in more than one table, then a reference column can be created. For a reference, values are pulled from the table which the reference points to.

For this project, the database will not be too complex and will consist of only a few tables. There is one table for the user, which will keep the username and password along with some data regarding the user. Then another table for the US states, which is referenced by the users table. One table to keep track of battery levels, this table may be used to trigger a warning if battery levels go to low. One table for data collected by weather stations, which is used in determining if a watering schedule should be skipped or not based on weather forecast. Finally, a table for the metrics collected by the sensors. At this point in the project tables may still be created or modified. More details on the tables and columns will be at the end of this section.

There are many options available for databases. A couple of the most widely used ones are flat file databases and relational databases. Relational databases are more flexible. The database in this project is a relational database. Both types should be fine given the scope of the project. However, a relational database system was chosen due to the simplicity of setting up a relational database. The relational database should also perform better as the system scales. A relational database will allow for data to be divided into tables which is a good representation of the data being used. The tables can then be joined in queries to get more meaning information from the data collected.

For the DBMS, database management system, there are a multitude of choices. Some at a very high premium, such as Oracle, and some which are open source and free to use, such as MySQL. Using an Oracle database would fall outside the

financial realm of this project, as a single license can cost thousands of dollars per processor. Also, at this time there is no clear advantage on having an Oracle database for this project. MySQL, an open source DBMS, has a very good reputation as being reliable, it has a large online community, and plenty of learning resources available to research and get help when setting it up.

Once the database is setup, SQL queries were run to create the database and the tables. Such queries can be run directly against the command line. However, another software was used to connect to the database for the purposes of setting up the data. The software, MySQL Workbench, will allow for configuration of the database via a GUI, a graphical user interface. Using this open source software will help in development and implementation time. The workbench will simplify how the database and the tables are created. More time would be needed for creating queries, tables, and relationships without the workbench. The workbench will also be utilized for inserting dummy data for troubleshooting and developing, which will be covered in more detailed in the software testing section. The work bench will also be used for creating table diagrams, which are helpful in understanding the logical layout of the database. More detailed information about the tables is given next. As in most software, the database is likely to change as the project evolves. The tables mentioned next should be a solid foundation for the data needs of the controller. The following table is a summary of the tables and the data it keeps.

Table 16 - Database Field Tables

Table Name	Table Data
user	Information about the user
location	User location and controller location
state	List of US states
weather_forecast	Weather forecast for each controller's location
controller	Each controller and unique sensor data
controller_sensors_data	Sensor data specific to a controller
zone	Data about the plant planted in each zone
zone_sensors_data	Sensor data specific to a zone
schedule	Watering schedule information

6.1.1 Database Tables

The User table will keep data about the user. This information will be the user_number, user_name, password, first_name, last_name, email. This data will be used for most of the queries. Each query should pass a username and password to be authenticated before accessing the database. Therefore, this table will be used for security. When querying the other tables the user id will be used to find data relevant to a user as well.

The location table will keep data regarding where the controller is installed. This data will be used when collecting data from weather stations. It is important for this information to be correct as the proper weather station needs to be connected to in order to collect helpful information. The information from local weather stations will be used when deciding if to skip a watering session or not. The information in this table will be street, number, city, zip_code, and state. State will be a reference field to another table. This table will also keep location data specific to the user, in addition to the controllers.

States will be a table of its own. The reason being that states will likely be repetitive information as there are only fifty states. Therefore, we can make this a table on its own and have a column which refers to it instead of repeating this data over and over on the location table. This table will only have two columns, the number and name.

The weather forecast table will keep data polled from the local weather station, given the controller location. This data should be updated at least once every day. Only two days forward are of interest at this time. This table will have the predicted probability of rain, temperature, wind, and humidity level. This data will be used when trying to decide if a watering session can be skipped.

The controller table will keep records for each controller a user has. A user could potentially have multiple controllers. Controllers could be in different locations, however managed from a central location. The controller will also have sensors. A separate table will keep data collected by sensors which are individual per controller, such as the anemometer.

The zone table will have information specific to what is being planted per zone. Information, which will be user configurable, such as tolerance, last_watering, and plant_type will all be used for water saving features. The primary index for this table will be the I2C address for the sensor. The zone sensor data table will keep all the data collected by zone sensors.

The schedules table will keep information regarding how often each zone should be watered. Columns will include each day of the week, time, duration, and update number. The update number column will be used when determining whether to

update the database schedule or the local controller schedule. The following image contains the database diagram for the tables required for the project.

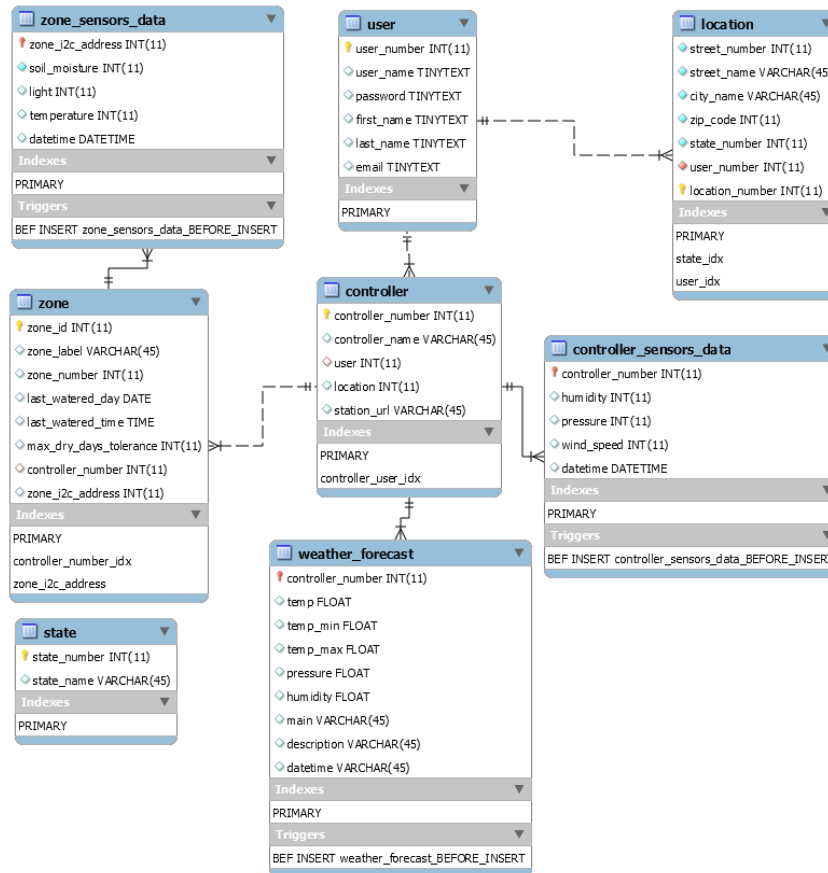


Figure 16 - Database Configuration

6.2 Microcontroller Software

This section covers the software running on the micro controller. This software controls all the physical hardware involved in the smart garden controller. This software will communicate with the sensors to get metrics to upload to the web server. It will control the valves which will let water through to water the plants. It will also interact with the user via push buttons and an LCD. Therefore, in a way this software will be the brain of the device. This software alone with the hardware will be enough to have the device working and watering the plants based on a user set schedule. However, it will still need the web server for the more advanced features.

The software controlling the smart garden controller will be written in C, assembly language was another viable option. However, assembly language could be considerably less efficient, development time wise, depending on the level of developer's experience. Also, the team working on the project is more developing

in C then in assembly. Using a language which the developers is familiar with should help the efficiency of how fast the team can code. Although assembly would give more direct control of the device, there are functions in C which will provide the same functionality. Also, at this point in our research it seems most libraries with built in functionality which would be helpful to our project are written in C. Therefore, when deciding what language to program the device, C seemed the best option.

The code will be developed using Texas instrument's code composer studio (CCS). Code composer studio is an integrated development environment (IDE). Code composer will provide most of the feature necessary to program the device and upload the code to the controller. Per Texas instruments website a subscription is no longer needed to use CCS.

6.2.1 Micro Controller Logic and Functions

The controller will basically run a loop. The loop will compare data collected by sensors, weather stations, and user configuration to decide when to water the plants, as well as upload data to the database. The code running on the SGC will be composed of multiple functions and libraries. We will go over some of the main functions used to control the flow of the data and control the SGC. Then we shall go over the overall logic controlling the SGC. The functions may be changed as the project matures.

The databaseDAO() function will control how the SGC communicates to the web service. DAO stands for data access object. One of the parameters this function will receive is the web address. The web address will determine the table and database operation, insert, update, delete, and select. The other parameter will be a String with the information to be passed to the web server. This function will be called by many of the other functions.

The updateSchedule() function will use the push buttons to view the current schedule running the controller and updated it.

The updateSchedules() function will compare the current schedule's update number with the schedule update number on the database for a controller. If the update numbers are not the same the same, then one of the schedules need to be updated. The updateSchedules() function will determine what schedule was updated most recently. The most recently updated schedule will become the schedule in both the controller memory and on the database. If the update numbers are the same, there will be no need to update any of the schedules.

The waterGarden() function will compare the current time up to the minute with the schedule and check for a match. If communication with the web server is not available, the waterGarden() function will trigger the watering system if a match in the schedule for the current time is found and the current soil moisture level is not

met. Weather station forecast will also be used, unless there is no internet connectivity. If connection to the web server is available, the waterGarden() function will let the web server determine if the garden should be watered or not.

The waterNow() function will be used to trigger watering of a specific zone regardless of weather forecast, schedule, or soil moisture. This function should receive a zone and a timeout. This function could be used as a fallback if the sensor metrics were wrong for any reason. This function could possibly be used as well with home automation system integrations as well.

The sensorData() function will communicate to each zone and collect sensor data. This function will receive the zone number as a parameter and return an array with the sensor data.

The weatherStation() function will connect to a weather station via REST and collect weather forecast data. This data will be later used when determining whether to trigger the watering of specific zones or not. This function will receive a web address which will be used to connect to the correct weather station.

Not all the functions used by the microcontroller will be listed. There will be many helper functions which will be used by the above-mentioned functions. The controller will start by checking if it can communicate with the web server. If the web server is available, meaning we successfully connected to it, then the controller will check if the data for the local schedule matches the data on the web server. If the data is not the same, say the schedule was updated on the database, then the SGC will need to update the local data or the database data depending on what was most recently updated. Once the data is synchronized between the SGC and the web server, the controller will collect sensor data. After collecting sensor data, the SGC will also collect weather station data, the weather station will be configured either via the push buttons or the web application.

At this point all the data should be synchronized between web server and the SGC. The SGC should have most of the data necessary to determine if it should water a zone or not. The SGC will check each zone if their moisture is above a threshold. The threshold will be set per zone. If the zone moisture level is above the threshold then there will be no need to water the zone at that moment. If the soil is not wet enough, the SGC will next check if it is supposed to rain or not in the next day or two. Each zone will have a tolerance number which will be used when determining if a zone needs watering or not. For example, if a zone tolerance is zero then the zone should be watered without regards to if it will or not rain in the next day or two. If the zone tolerance is one, then the SGC will only check it will rain on the next day. The tolerance will reflect up to how many days the plant type in the zone can go without being watered. The following table contains the main function names and their operation for reference.

Table 17 - MCU Functions

Function	Operation
databaseDAO	Provide communication to web server, sends and retrieves.
updateSchedule	Edit local schedule
updateSchedules	Synchronize schedules between controller and web server
waterGarden	Trigger watering system
waterNow	Trigger immediate watering of a zone for specified time in minutes
sensorData	Collect sensor data and upload to web server
weatherStation	Query local weatherStation for weather forecast in order to update both the controller and database with forecast

6.2.2 Micro Controller Software Flow Diagram

The following software flow diagram is a representation of the software logic and a good resource to better understand how the SGC works as a whole. The image was created on draw.io (see permissions).

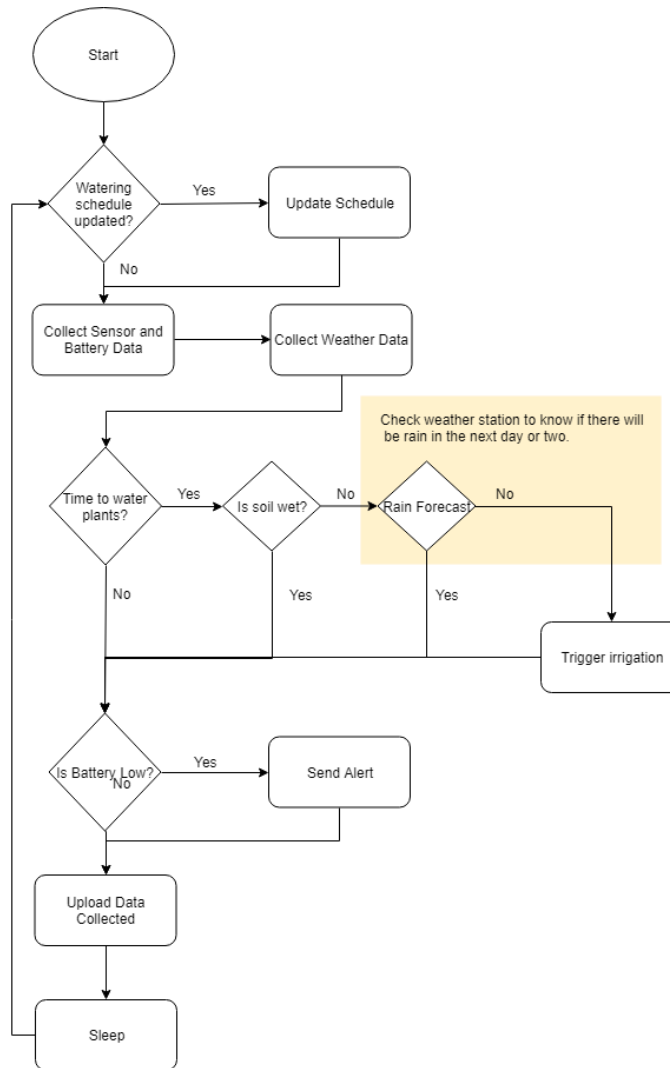


Figure 17 - Initial Microcontroller Software Flow Diagram

6.3 Web Server

A web server is a software which accepts HTTP requests and processes them. Web servers usually are known to host static HTML pages. In a way, the evolution of web servers lead to application servers. Like a web server, an application server serves web content. However, application servers also act as a container where business logic and processes can be provided to clients. However, very often these terms are used interchangeably these days as simply serving HTML static content is not very common these days. Therefore, for the purposes of this project the application server will be called the web server. Calling the application server, a web server will avoid confusion when going over the application server and the web application. A Tomcat server will be used as the web server to run the Java code, which will be used for the business logic. The Apache Tomcat is open-source and therefore free to use.

The web server software will control the flow of communication from the controller to the database, and from the web application to the database. Very often a web server will provide data which is used in building a web page. However, to simplify development we decided that the web server would mostly be processing REST requests. This way it will be easier to develop each integration independently. Because the web server will not provide any HTML, we will be able to use the same web application interface for both the SGC and the web application. The web server will also host the web application which will contain HTML, JavaScript, and CSS code. However, the way the setup works will allow for almost complete independence from the java code and the rest of the code used by the single page application.

The main role of the web server will be to accept a request based on a web address. From such address the web server will build a query. This query will be sent to the database and the data retrieved. Once this data is retrieved the result will be converted into JSON (JavaScript Object Notation) and returned to the client, which could be either the web application or the controller. The same code will be able to convert the data to XML format as well depending on the request parameters. As the format for the requests for the web server are the same regardless of the source, we will be able to later add other integrations to the database. For example, we could later choose to create an android application, or an iPhone application, without having to change how the web server handles input. Therefore, there would be no need to change or update the web server code, unless extra functionality was to be added.

The web server will also control authentication and authorization. Authentication will confirm the right user has access to the right information. Authorization will control what data the user is allowed to retrieve. Authorization for this project will be quite simple as all the data will be linked to a user name. At this point multiple users per account will not be supported. Therefore, once a user is authenticated it will have access to all the data it is authorized to. All the queries will take into account the username and password. The following image, created on draw.io, is a graphical representation of a REST query with username and password passed in the request for basic authentication. The image also illustrates that future integrations will be possible without modifying the back-end code in the web server.

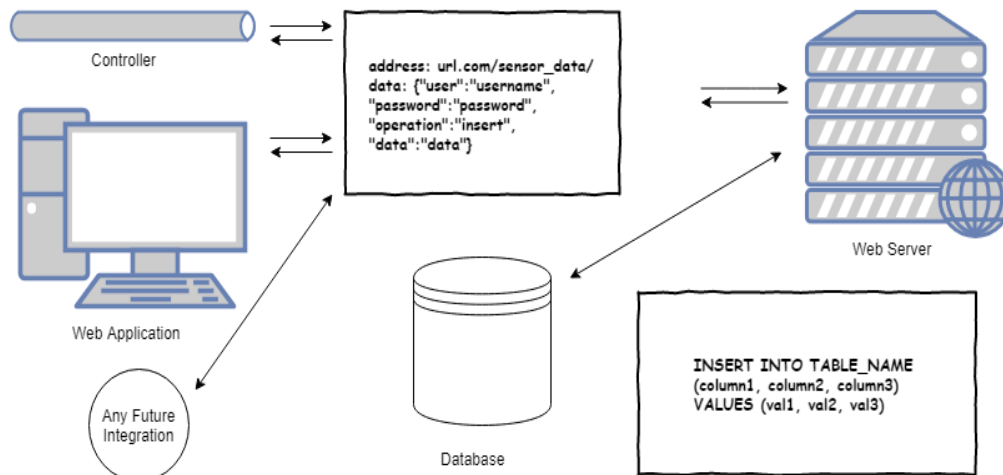


Figure 18 - Software Communication Diagram

The open source web services framework Jersey will be used. This framework will “abstract away the low-level details of the client-server communication” [3]. Using Jersey will allow us to focus on the business logic provided by the application server, rather than how to handle REST requests. Similarly, MySQL-connector-java will be used to provide communication to the database and abstract away the code needed to create connections to and retrieve data from the database.

The Functions on the web server will focus on providing data to the clients, which are the controller and web application. These functions will be accessed based on the web address given to the web server. For example, to access the `getUser()` function, which returns a user object given a user number, the client will browse to “<web_address>/webapi/user/<user_number>”, <web_address> and <user_number> would be replaced by actual values. The web server will then call the `getUser()` function. The `getUser()` function will reach the database and query the user table. Once the data is retrieved from the database it will be converted to a user object. The user object will be converted to JSON and returned as a string.

6.3.1 Web Server Methods Overview

Most of the classes provided in the application server will serve the purpose of querying the database. The class will have methods for query, insert, update, and delete. These classes will be very similar, the difference being mostly the class properties, getters/setters methods, and how they update the database. The zone table for example will have a matching class called zone which will provide a method for getting zone records from the database. Then there will be classes which will provide logic for the controller. The schedule class will be used to determine if watering can be skipped. This class will have methods which will analyze the data retrieved by the sensors, as well as the weather stations, and return true or false to the controller. The following table summarizes the classes and their use.

Table 18 - Class Functionality Table

Class Name	Functionality
Location	Provide getter and setter methods for location objects
LocationResource	Receives and returns JSON formatted locations
ResourcesDAO	Provides communication to the database
Schedule	Provide getters, setters, and methods for schedule objects
ScheduleResource	Receives and returns JSON formatted schedules
Sensor	Provide getter and setter methods for sensor objects
SensorResource	Receives and returns JSON formatted sensor data
StateResource	Receives and returns JSON formatted states
User	Represent user objects, provides getter and setters
UserResource	Receives and returns JSON formatted users
Weather	Provide getter and setter methods for weather objects
WeatherResource	Receives and returns JSON formatted weather data
Zone	Provide getter and setter methods for zone objects
ZoneResource	Receives and returns JSON formatted zone data

6.4 Web Application

The system implements a web-based application to serve as the primary user interface for the Smart Garden Controller. This web application is independent of any device operating system and therefore accessible to users with any device with an internet connection and a web browser. This 'web app' is implemented in the form of a single page application (SPA), created using the Angular 4 development framework. This implementation removes the need for an extracted routing service that would normally be used for large scale web applications that handle larger data sets and more user action. Due to the small scale of the systems development this is an ideal strategy for programming the user interface.

The web app consists of three main displays to fulfill the major requirements of a user interface for the system, and 2 ancillary displays which enable the user to efficiently and easily use the web app and maintain a relationship with their

personal garden. Each display has a corresponding html template, CSS file, and TypeScript file.¹ The following sections will detail those main displays and the development process by which they were created.

6.4.1 Development Environment & Tools

The most essential tool used in designing the web app was the Angular platform. “Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges.” [70] This allows the programming of scripts and templates for the web app to be divided into multiple ‘components,’ simplifying & concentration software development of each individual display that the user interacts with. Using Angular, our team produced an original, dynamically compiled multi-level html template that implements multiple JavaScript controlled components, CSS styling control, and TypeScript (TS) programming. The primary IDE used to write the html template, TS components, and style sheets was Virtual Studio Code (VScode), while some component creation was performed using Angular CLI (Command Line Interface).

The aforementioned web server and database were run concurrently while developing the web app, using Apache Tomcat via the Eclipse-Oxygen Java IDE and MySQL (respectively). Angular CLI and Node.js were used via the command line to serve the web app during development, however during prototype testing & usage the Tomcat server was used for both the database and the web app. Due to the small scale of this systems implementation CORS (cross-origin resource sharing) was not implemented, and therefore during development, while the database and web app were being served by different sources, it was required to disable web security manually by editing the browser executable. Upon expanding the scale of this system or further development CORS would absolutely be implemented to improve the security and portability of the Smart Garden Controller.

Bootstrap 4 was used to simplify the styling of the web app’s forms and navigation bars. Bootstrap is a free and open-source front-end framework. Bootstrap provides templates of how user interface elements look. This saves time that would otherwise be spent extensively editing the CSS for the html templates to achieve the same appearance to the user. [71]

¹ “TypeScript is a typed superset of JavaScript that compiles to plain JavaScript” [72]

6.4.2 App Component & Module

The Angular framework divides source code for web applications in development into constructs called modules, which contain components. Each module consists of a TS class file that imports the components for that module, and each component consists of an html template file (which can be injected with the html templates of sub-components), a CSS file, a testing TS file, and a TS class file. Due to the small scale of this system, our development consisted of a single app module containing the main app component, which included the multiple sub-components that controlled the corresponding displays seen by the user. Directives² are used in the html template of the app component to switch the view of the user interface between the displays mentioned in section 6.4. Upon further development or an expansion of the scale or features of this system, multiple modules would likely be used, and the single page implementation of the web app would be forgone in favor of multiple page routing. The following sections will detail the sub-components written for the user interface.

6.4.3 Log In Component

The log in component displays a bootstrap card with a login form that prompts the user for their username and password, and contains a button that switches the view to a 'create account' form. This form prompts the user to enter information to create an account, then upon submission switches the view back to the login card.

Upon submission of the login form the view switches to the schedule display, and upon navigation back to the log in display, switches the view to a card informing the user that they are logged in, which contains a logout button that clears the information for the current user and refreshes the page. This component performs the http requests to the server which create the user account and get the user's information when logging in.

² Directives: Classes included in the Angular API that, when used in the html template, attach custom behavior to elements in the DOM. [70]

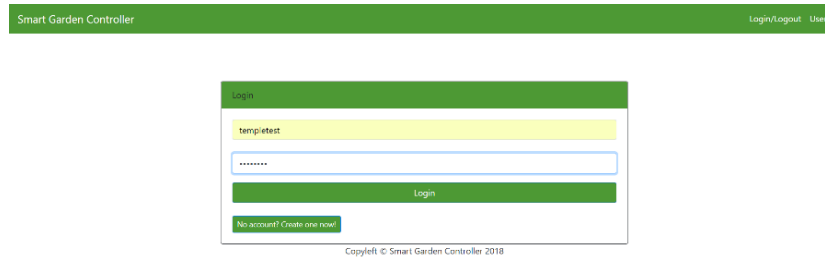


Figure 19 – Log In Component Example

Upon further development or an expansion of the scale or features of this system, the requests to the server (made by this and all other components) should be extracted to a component dedicated to http requests, and all data sent and received should be encrypted.

6.4.4 User Component

The user component displays two forms which the user can navigate between using bootstrap pills. The 'Edit Personal Info' form is for the user to update information such as their name, email address, username, and password. The 'Edit Address' form is for the user to update their street address. This component performs the http requests to update the above information for that user in the database.



Figure 20 – User Component Example

6.4.5 Data Component

The data component displays the measurements collected by the system's sensors in two formats which the user can navigate between. The 'Right Now' display shows a card containing the most recently collected data from the prototype. Assuming that no sensors are damaged or otherwise disabled and the device is running continuously at the time the user views this page, the data shown here will be no more than one minute old. The data component performs an http requests to get data recorded by the soil sensor and by the humidity & pressure sensor, which is stored and accessed separately in the database. When the 'Right Now' display is shown those requests pull data from all time, then the component extracts the most recent JSON objects to display data from. When the 'Time Range' display is shown those requests pull data from the range selected by the user as detailed later.

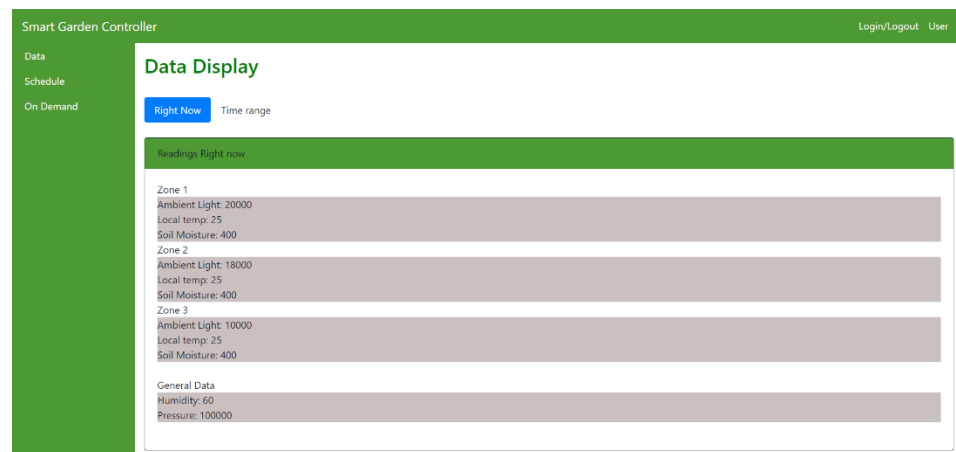


Figure 21 – Data Component Example 1

The 'Time Range' display prompts the user to enter into a bootstrap form the date range for which they wish to view data. Upon submission the view is changed to a display containing multiple bootstrap cards, each containing a canvas that is filled by a method in the data component TS class with data recorded by the device during that date range. These methods, and the JS file imported to the data component class that they require, are modified from the open-source Chart.js software (see permissions). [73]



Figure 22 – Data Component Example 2

6.4.6 Schedule Component

The schedule component includes three main displays that the user can navigate between. The 'Current Schedule' display shows a weekly calendar with the information for each scheduled watering in its corresponding day. The component performs http requests to get the schedule for the current user before it is displayed.

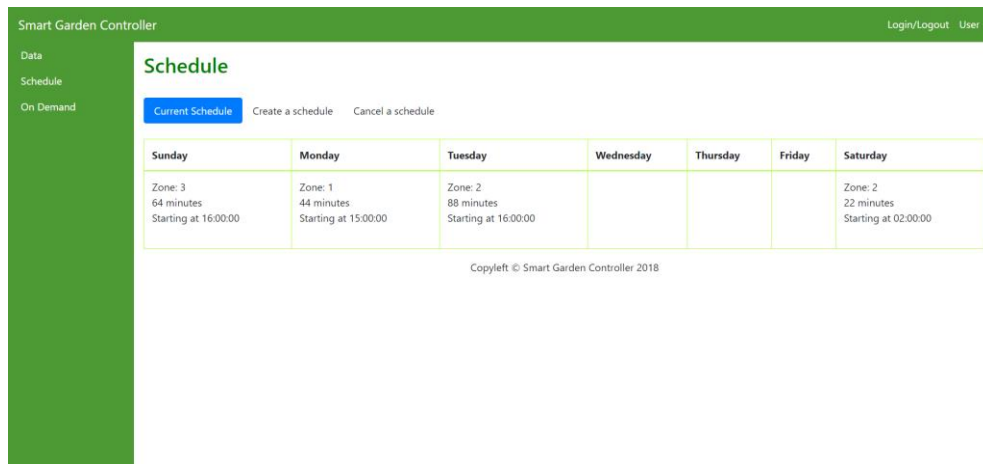


Figure 23 – Schedule Component Example

The 'Create a schedule' display allows the user to create a scheduled watering by entering day, duration, and starting time information into a bootstrap form. The user can navigate to one of these forms for each zone within the 'Create a schedule' display. Each zone can be scheduled to be watered at most once per day, however additional watering can be performed via the on-demand component detailed in the next section.³ The schedule component performs http requests to

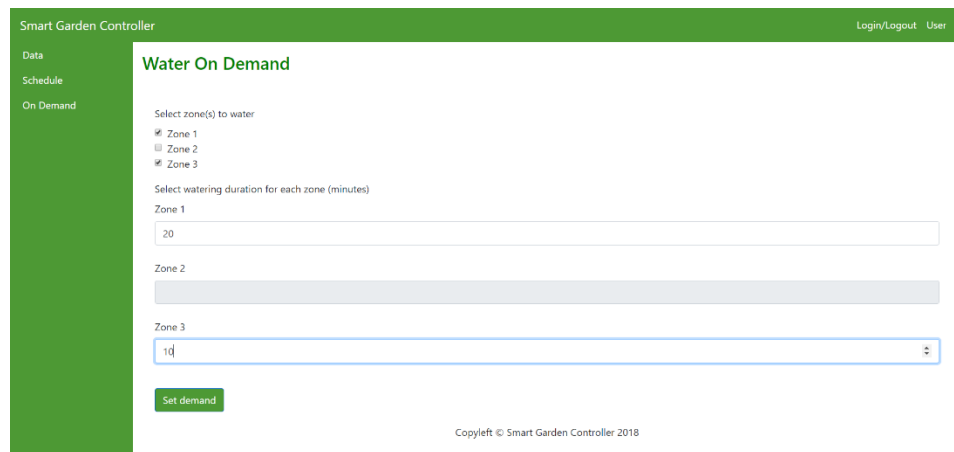
³ Beware: federal, state, and local governments may restrict the frequency of residential watering permitted.

create schedule objects in the database upon the submission of the aforementioned form.

The 'Cancel a schedule' display shows a list of the scheduled waterings, each one with a corresponding 'Cancel' button. When the user clicks one of these buttons the schedule component performs an http request to delete the corresponding schedule object from the database, then the view changes back to the 'Current schedule' display where the deleted watering is no longer shown.

6.4.7 On-Demand Component

The on-demand component first performs an http request to get any demand objects from the database, and if there are none, displays a form for the user to set a demand for any number of zones. Upon submission of that form the component performs an http request to create a demand object in the database, and the prototype will execute those demands immediately after its next database check, watering one zone at a time.



The screenshot shows a web interface for a 'Smart Garden Controller'. The page title is 'Water On Demand'. On the left, there is a green sidebar with navigation links: 'Data', 'Schedule', and 'On Demand'. The main content area has a green header with 'Smart Garden Controller' and 'Login/Logout User'. Below the header, there are three radio buttons for 'Select zone(s) to water': 'Zone 1' (checked), 'Zone 2', and 'Zone 3'. Underneath, there are three input fields for 'Select watering duration for each zone (minutes)'. The 'Zone 1' field contains '20', 'Zone 2' is empty, and 'Zone 3' contains '10'. A green 'Set demand' button is at the bottom left. A copyright notice 'Copyright © Smart Garden Controller 2018' is at the bottom center.

Figure 24 – On-Demand Component Example

After a demand is successfully set, the view changes to a display informing the user that there are demands set, which includes a button which, upon the user's click, causes the component to perform an http request to delete all the demands for that user. This is the same display that is shown if the user navigates to the on-demand component and the initial request finds any demand objects in the database.

6.5 Smart Speaker Integration

The smart speaker integration will allow the user to use voice commands to control the watering of the zones. The smart speaker chosen to integrate with the smart controller was Amazon's Echo. This device uses Amazon's smart assistant.

Amazon has a variety of tools and tutorial on how to create skills for the Echo, each skill is a new functionality which can be performed by the Echo. There were a few options on how to integrate the Echo with the smart controller. The simplest options seemed to be to use REST calls to communicate with either the CC3220MODA or to the web server, already used for other integrations in the project.

The first option would be for the web application server to receive the REST calls. It should be fairly simple to receive any REST calls from the Echo because a REST server will already be setup for other parts of the project, to perform communication between the database and any integrations. Amazon has canned code libraries which would allow to send REST calls to a target server. The challenge therefore would be getting familiarized with creating the skill and using the code to generate the REST calls. Amazon has plenty of resources which should help in this process. The second option would involve having the echo device send calls directly to the CC3220MODA MCU. The CC3220MODA is capable of running a lightweight web server. The CC3220MODA could receive such queries as interrupts. The interrupts would parse the data sent in the GET/POST request and perform the action.

Option one has a drawback that it may be too verbose to send a call to the web application server used for other integrations. Each call would need to contain the username and password for the user. It may not be desired for users to have to say username and password in front of other people. Therefore we would need a way to configure the skill keep username and passwords so that the user would not have to say it out loud. At the moment it is not clear how difficult that would be. There would also be a drawback in sending the REST calls directly to the CC3220MODA controller. In such a scenario, the CC3220MODA would need to keep a web server running constantly. Clear numbers on resources required to maintain such web server running 24/7 on the MCU were not found yet. Running the web server could end up being too much of a performance issue for the MCU.

During research regarding how to integrate smart home devices our plan was to implement this by sending REST calls to the web server which would then interface with the smart home devices, instead of communicating directly from the CC3220MODA to those devices. This was because the web would have more resources to handle calls, such as a faster CPU and more memory & storage.

Upon final prototype design & construction our team decided to omit any smart home device integration and dedicate more development resources to achieving maximum functionality regarding the essential system requirements. Upon an increase in the scale of the system or further development, smart home device integration should absolutely be implemented.

7.0 Project Prototype Construction and Coding

7.1 Integrated Schematics

This section details the schematic design for the Smart Home Garden project. This schematic was captured in Altium Designer.

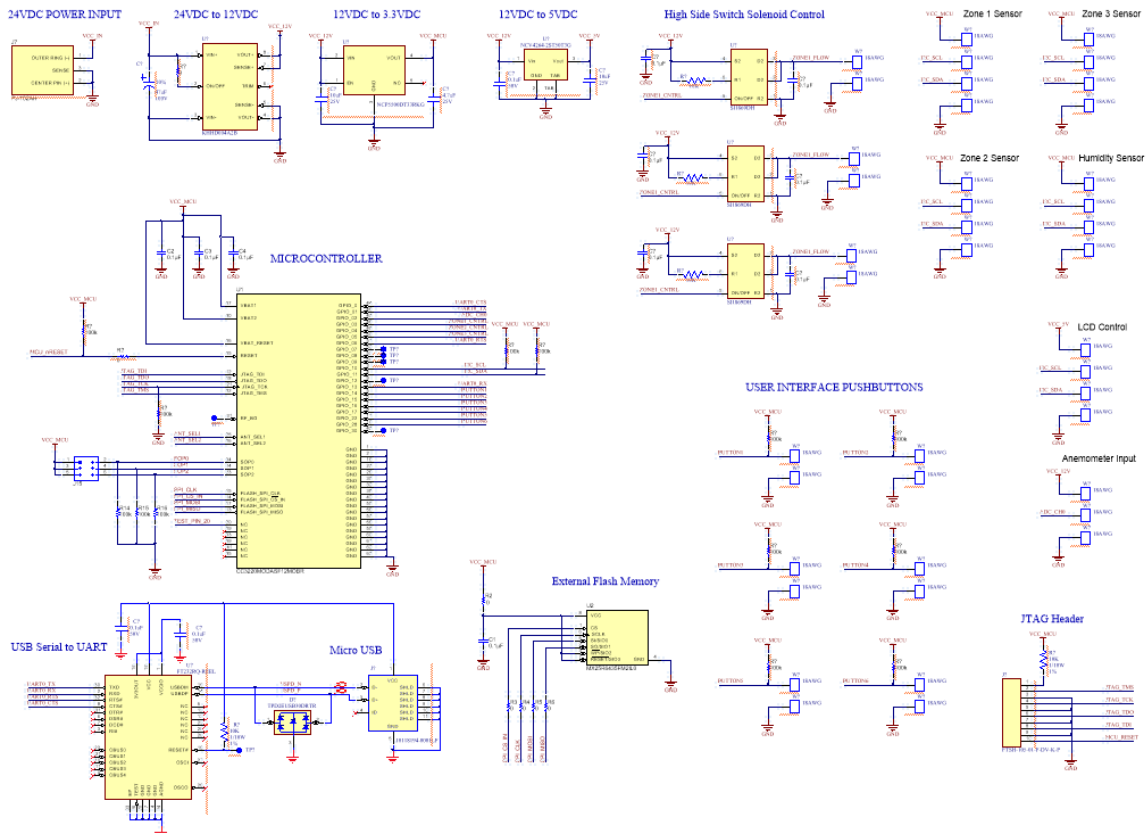


Figure 25 - Main Board Schematic

The schematic was able to be placed on a single schematic sheet. There are multiple sections of the schematic design that had to be considered and properly executed to meet design goals and deal with constraints.

7.1.1 Power Schematic

This section details the power design for the Smart Home Garden project.

24VDC POWER INPUT

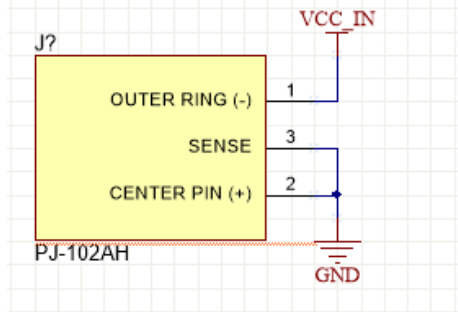


Figure 26 - Power Input Schematic

The power input connector is a barrel style connector that will allow us to use a AC-DC converter and plug the barrel connector into our board. The AC-DC converter that was chosen provides a 24VDC output. This input connector is rated for use with 24VDC input and is capable of handling the current that will be needed for proper operation of our board and design.

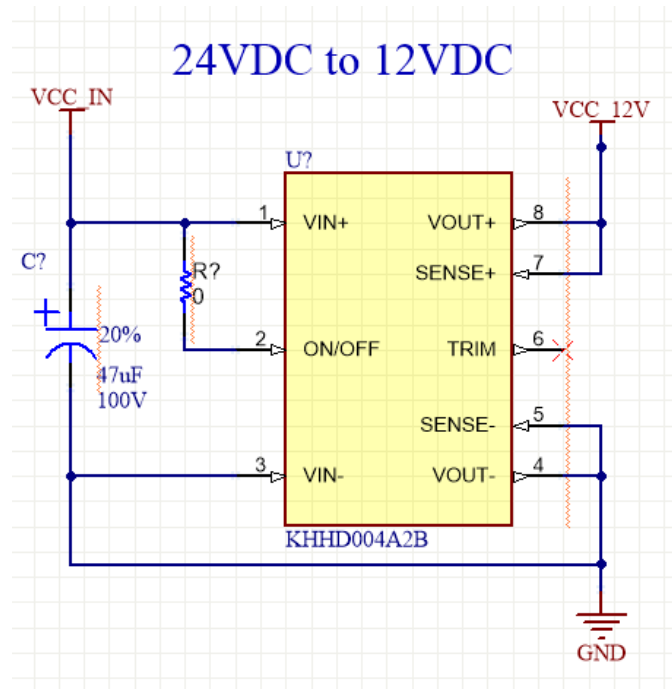


Figure 27 - 12VDC Power Schematic

Once the power is brought on board by the input power connector, it is immediately passed into the main power module that generates our 12VDC power rail. This power module is a wide input robust part that is capable of handling much more power-hungry designs. This power module was chosen due to its high current rating and also for the fact that there is previous design experience with using this part. The reason that the high current rating is desired is because the solenoid

valves that operate the watering process for each zone require a significant amount of current to operate properly. This power module is very simple to implement and has many peripheral components incorporated onto the module. For this reason there is only the need for an external input stability capacitor whose value was chosen based on previous design experience.

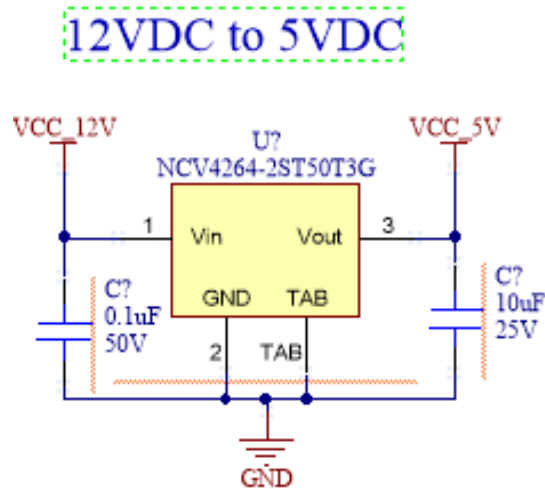


Figure 28 - 5VDC Power Schematic

After the power is conditioned by the main DC to DC power module, we are able to condition the power further to suit our needs. For the purposes of this design it was necessary to have two separate logic rails. The image above shows the 5VDC power component that conditions the 12VDC power rail down to 5VDC. This power rail operates the LCD screen and is an optional power rail for the soil moisture sensors if the desired performance is not achieved using the 3.3VDC power rail. This power IC is a linear dropout regulator and requires an input capacitor as well as an output capacitor for stability of the power rails. The values for these capacitors were determined from the part datasheet.

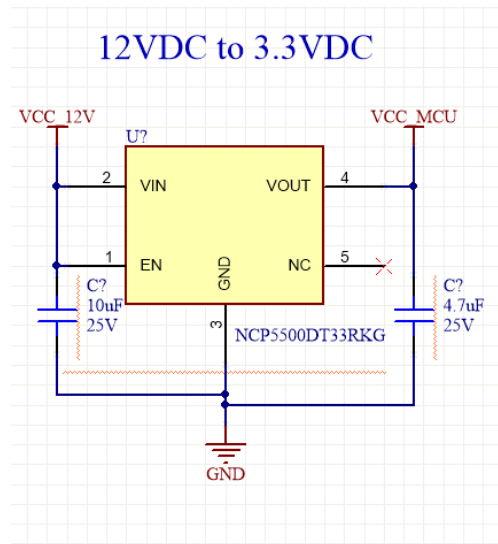


Figure 29 - 3.3VDC Power Schematic

The 12VDC to 3.3VDC power component is what provides the microcontroller and the bulk of the sensors and peripheral parts on the board with power. This module was specifically chosen for its current rating and due to the fact that there is previous experience with this part in designs. The part is a linear dropout regulator that requires an input capacitor as well as an output capacitor just as the 5VDC regulator did.

Together, the input power connector, the 12VDC power module, the 5VDC regulator, and the 3.3VDC regulator make up the power design for the Smart Home Garden project schematic.

7.1.2 Sensor and User Interface Schematic

This section details the schematic design of the sensor interface portion of the Smart Home Garden project. The sensor interface schematic varies for different sensors that must be interfaced with but there are many similarities.

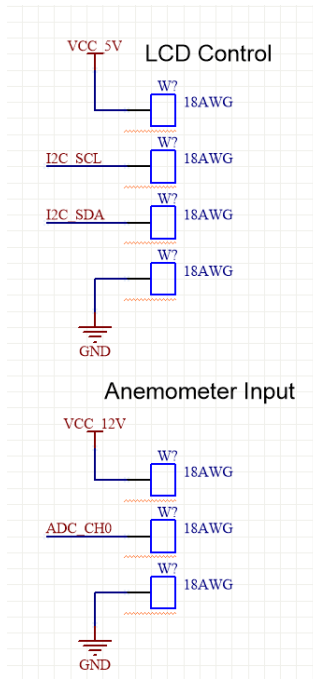


Figure 30 - LCD and Anemometer Interface Schematic

The LCD Control and Anemometer input portions of the schematic detail how these two components are interfaced with using our custom circuit card. The LCD control board requires the 5VDC rail to operate properly as well as a ground signal. Beyond the power, the LCD screen is an I2C command based module and only requires the data and clock signals from the microcontroller to function properly. The anemometer is an analog input device that runs off of 12VDC. For this reason we supply the sensor with 12VDC and ground, and provide an input to the microcontroller for the sensor output data. Both of these external components require wiring the signals off of the board. For this reason it was decided to provide conductor plated through holes specifically sized for 18 AWG wire. This allows us to manually choose cable lengths and hole placement so that we are not constrained by mechanical features this early in the design process. Using these through holes we will solder the appropriate wires for each sensor onto the board and route the wires as needed.

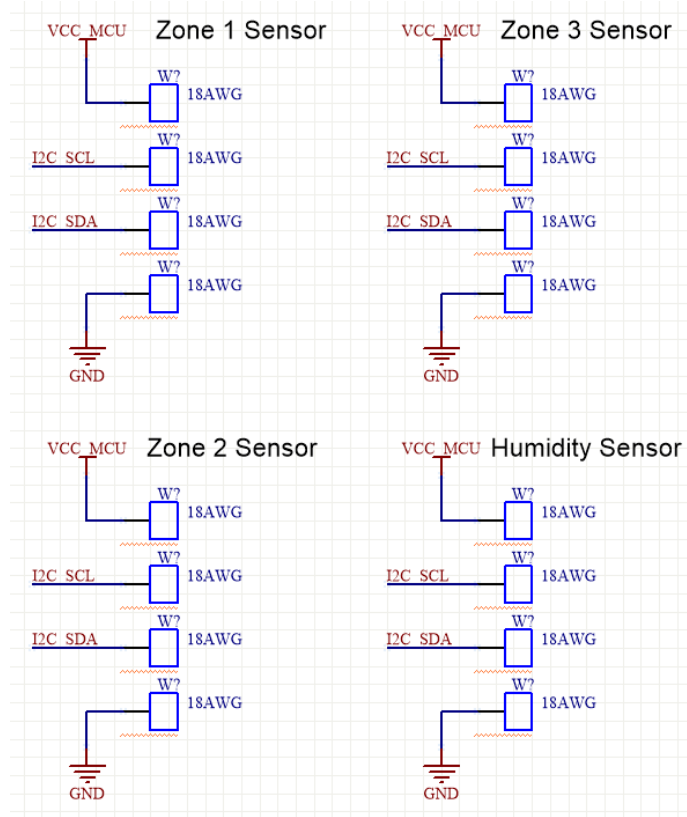


Figure 31 - Soil Moisture and Humidity Sensor Interface Schematic

The soil moisture sensors as well as the humidity sensor used in our design require cabling much like the anemometer and LCD screen. For this reason, the same method was used to interface to the sensors using through holes on our custom circuit card. For the three zone soil moisture sensors as well as the humidity sensor, the only signals required are the I2C bus signals and power and ground. This made the design easy as we were able to simply copy and paste the same circuit multiple times.

USER INTERFACE PUSHBUTTONS

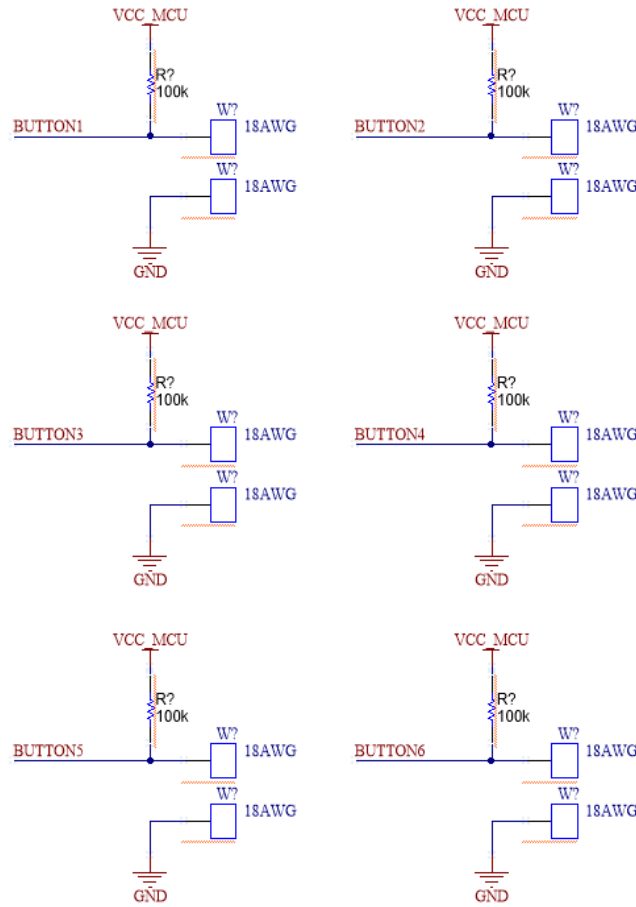


Figure 32 - User Interface Pushbuttons Schematic

The user interface pushbuttons are a crucial part of our design and allow the device to be used in the event that wireless connectivity is not available. The pushbuttons for our device must be mounted to the exterior of the chassis so that the user has convenient access to them. This lends itself to yet another case where wiring the components to the board using through holes is the best solution. This will allow us to place the pushbuttons and wires where it is best for our design and if changes need to be made there is no impact on the electrical design. The pushbutton interfaces were schematically designed so that when the button is pushed, the microcontroller will see the signal line pulled low to a logic 0 state. To prevent the inputs to the microcontroller from resting in an unknown state, which is likely to cause false positive interrupt signals, a weak pull-up is placed on the signal line which will always place the signal in a known state. If the button is not pressed a logic level 1 is on the signal line, and when the button is pressed there is a transition to a logic level 0. No debouncing circuitry was added for the pushbutton interface as it was determined, if debouncing is necessary, it can be performed in software rather than using hardware.

7.1.2 Solenoid Flow Control Schematic

This section of the schematic deals with the solenoid valve control which handles the watering operation to the different zones of the system. The solenoid valves operate by opening when 12VDC is applied and then closing again when the 12VDC is removed. This causes us to control a 12VDC rail using a 3.3VDC logic microcontroller. To achieve this a high side switch configuration was used.

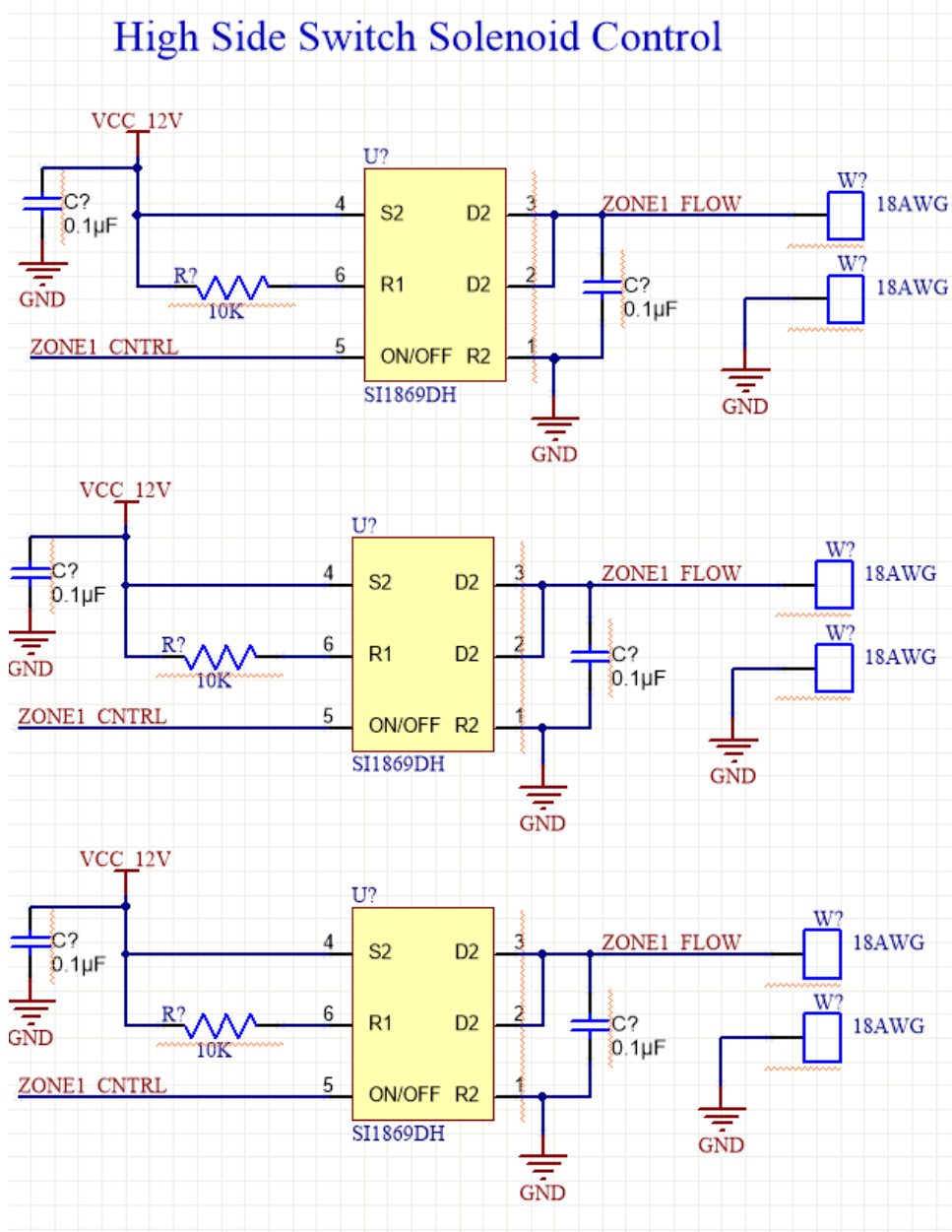


Figure 33 - High Side Switch Configuration Schematic

The high side switch is essentially an NPN FET and a PNP FET combined within a single device package and internally configured to operate high voltage switching using a lower voltage logic, hence the term high side switch. The high side switch operates by passing the high voltage signal from the input to the output when the input signal to the ON/OFF pin is pulled high. The toggling of the ON/OFF pin logic is performed using the microcontroller. When the logic level is 1, the 12VDC signal is passed through the high side switch and to the through hole where the solenoid will be wired. Each solenoid valve is also supplied its own grounding through hole so that each valve can be properly wired to the board. The high side switch requires input and output capacitors much like most integrated circuit devices, and it also requires a 10K resistor pull-up on the enable signal R1. This pull-up can be configured to disable the device in certain configurations, but that is not necessary in our design, so the 10K pull-up resistor allows the device to be in an always enabled state. This specific high side switch was selected because there has been multiple previous designs that have proven this parts ease of use and functionality.

7.1.3 External Flash Memory Schematic

The CC3220MODA comes with onboard RAM and a small amount of flash memory, but an external device was desired to ensure that enough memory would be available for whatever needs may arise. The CC3220MODA has a specific pinout that uses an external flash memory device and the development kit provided us with sufficient documentation to implement the same flash memory design on our custom circuit card.

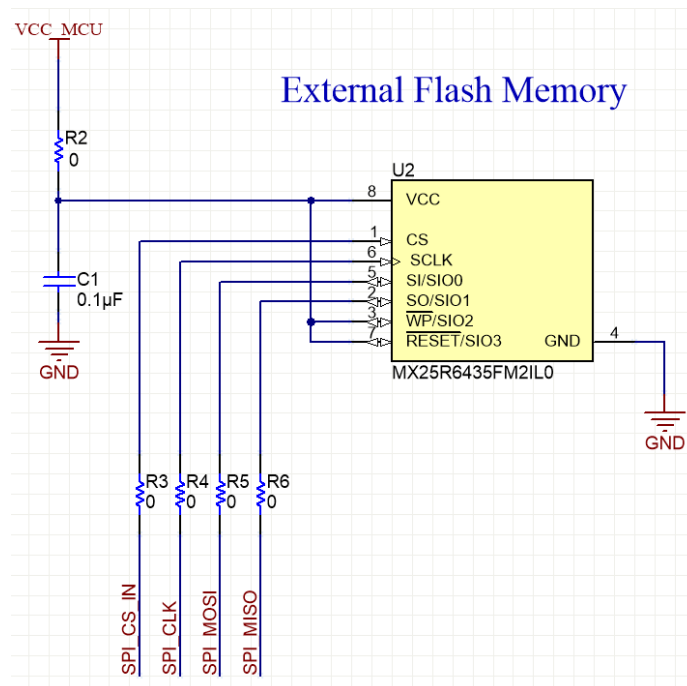


Figure 34 - Flash Memory Schematic

This flash memory part uses a SPI interface that is specifically intended for use with a SPI flash part from the microcontroller module. The flash memory part is schematically designed to be held out of reset and be in continuous operation. As stated previously, this design was derived from the development board documentation, so the confidence is high that this interface will have no hardware issues.

While this flash memory was included on the PCB of our final prototype, it was never necessary, as the standard memory was more than enough for the systems final design. Upon and increase in the system's scale or further development, features could be added easily to the design to take advantage of this surplus memory.

7.1.4 Programming and Debug Schematic

Programming and debug are essential functions in developing new hardware and performing the software integration and testing of a new device. The device is easily programmed using a simple JTAG interface which is standard. The device has been designed to also utilize a serial terminal functionality using a FTDI USB to Serial integrated circuit part.

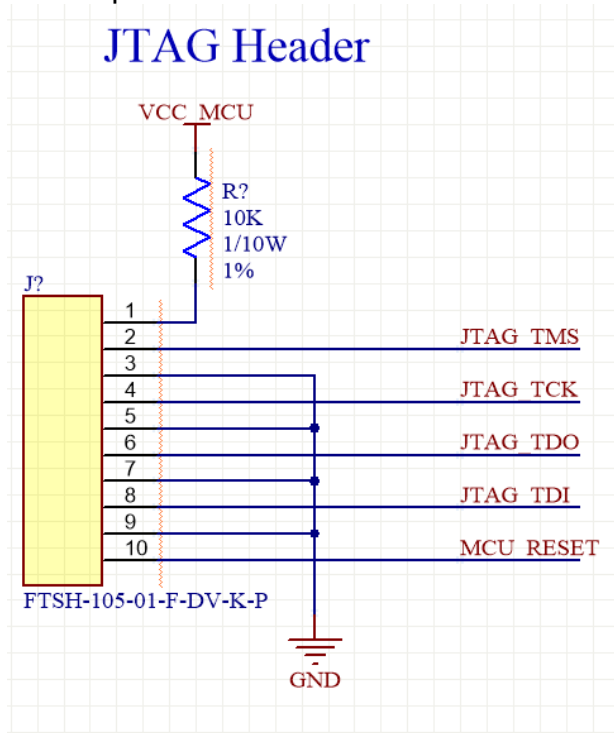


Figure 35 - JTAG Header Schematic

The JTAG interface is an industry standard. A simple Samtec 10 pin JTAG header was used to break out the JTAG signals for programming. The reset signal is also

routed to this connector as JTAG uses the reset line to instigate a programming sequence.

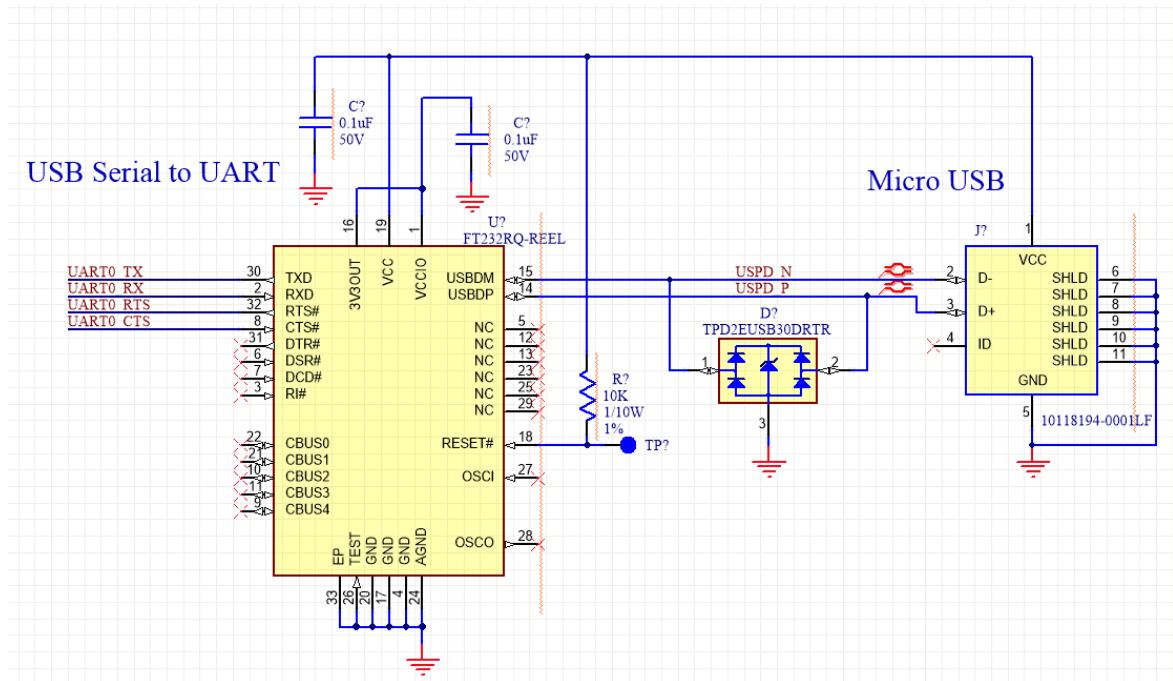


Figure 36 - USB to UART Schematic

The USB Serial to UART interface takes the UART interface from the microcontroller module, using the TX, RX, CTS, and RTS signals. These signals are routed to the FTDI part. The FTDI part is activated by the presence of a 5VDC signal when a powered USB device is plugged into the micro USB connector. When this happens, the FTDI enumerates itself and begins the process of converting the UART data into a serial terminal for use in a debug and test environment. This circuit has been used in previous designs by members of the team and is known to be fully tested and functional. There is no complex circuitry here only bypass capacitors and a simple transient voltage suppression part to protect against electrostatic discharge events when plugging in the USB cable.

This small amount of circuitry will allow us to easily program and debug our prototype device and allow for an easy way to derive test data from our system. This is perfect for the Smart Garden Controller because it will allow the team to spend less time worrying about the complex programming that some of the other systems similar to our project have. It is also important because the smaller the circuitry the less space and money would be required to make such a circuit. The

less complicated we can keep the design for our project, the more time our team can focus on other aspects of the device and have the project running in time.

7.1.5 Microcontroller Module

The microcontroller module is the heart and brains of the Smart Home Garden device. It is crucial that a proper electrical design be done to ensure all necessary functionality of the device. There was a wealth of helpful documentation that aided in the electrical design of the microcontroller as well as previous experience using this module. For these reasons there is a high level of confidence that the electrical design for our microcontroller module will be operational without the need for an obnoxious amount of debug and testing to obtain proper functionality.

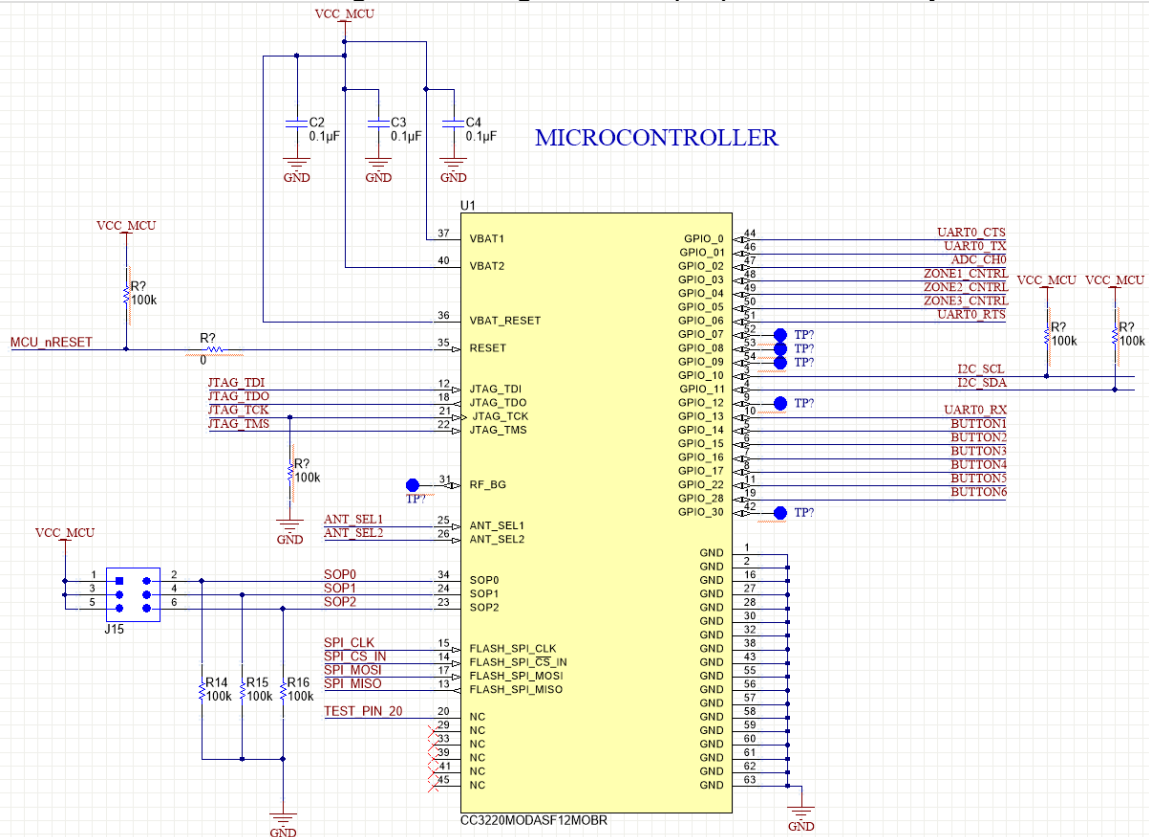


Figure 37 - Microcontroller Schematic

The microcontroller schematic is the most in depth portion of the design. As was discussed in an earlier section of this document, the pins for the microcontroller module are configurable to act as many different interfaces and inputs and outputs. The Texas Instruments PinMux tool was quite helpful in determining the appropriate pinout for our device schematic. As you can see from the image of the

microcontroller schematic, the use of the module form has been extremely beneficial in the simple implementation of the power design. The CC3200MODA only requires a 3.3VDC power rail to 3 pins, each of which also gets a bypass capacitor, and that is the power design for the microcontroller. There is a reset pin which is active low and has been pulled up using a weak resistor to leave it in a known state at all times. The JTAG signals are broken out according to the datasheet and a pulldown resistor is added to the TCK signal, which was derived from the development board documentation. The SOP signals which boot the device in different firmware load modes, have been routed to a jumper header where we will be able to change the state of these three pins base on our firmware programming needs. The FLASH_SPI signals are broken out according to the documentation from the module datasheet as well as the development board documentation. The GPIO signals have been configured according to the PinMux tool and all unused GPIO signals have been broken out into test points so that they may be easily accessed during development if an addition feature is needed. The I2C lines have the standard pull-ups installed to set the bus to a known state and ensure proper functionality. The only other signals remaining are the ground pins which are obviously grounded, and the no connect pins which are left unconnected. A single no connect pin has been connected to a signal due to the fact that during the electrical design there was some inconsistent documentation that put a question as to whether or not JTAG_TCK is connected to pin 20 or pin 21 of the microcontroller module. For this reason we elected to break both signals out in case the one anticipated to be the correct TCK signal is not, we still have access to it.

7.2 PCB Vendor and Assembly

Printed circuit board, or PCB, technology has been the backbone of the electronics industry since the advent of computational electronics. Printed circuit boards offer an electrical and mechanical mounting solution that has the ability to realize electrical circuits. A printed circuit boards gives the designer the ability to shrink the footprint of the design significantly compared to bread boarding components together. Printed circuit boards exist in nearly all electronics and it has become more common for even hobbyists to generate custom printed circuit boards for their designs.

The printed circuit board process begins with the design. The electrical design can be thought of as the blueprints for the printed circuit board, all of the necessary connections, restraints, and attributes accounted for. The electrical design begins with the schematic. The schematic contains all of the individual parts that will be housed on the printed circuit board and contains each of those unique parts attributes. These attributes range from part to part but can largely be thought of as all of the identifying information that would make that part unique (i.e. resistance, capacitance, impedance, voltage rating, tolerances, part number, manufacturer, price, etc.). Depending on the complexity of the part, these attributes can become very detailed and in-depth. It is important in this stage to be sure that all appropriate

relevant information is input into these symbols within the schematic so that all of the necessary data is available for analysis and synthesis by the tool. All of this information is important for the proper design of the printed circuit board due to many environmental and electrical factors. The attributes for each part are entered into their respective schematic symbol. These symbols also contain information about the part regarding their pin count, pin arrangement, PCB footprint, schematic configuration, simulation model, etc.

Once the symbols are created, they are input into the schematic and the pins are wired to each other according to the electrical design requirements using nets. These nets act like wires inside the schematic diagram, and they create a wire list or netlist that describes each part in the schematic with all of its attributes, as well as which pins each device has and how all of the pins from all of the different devices are interconnected. This netlist is the master list that the printed circuit board CAD software uses to properly create the layout for a printed circuit board. This list is what allows the designed electrical system to function properly based on the very meticulously designed electrical interconnections.

After the schematic is finalized and the netlist has been generated, it is imported into the printed circuit board design file. Inside this file, all of the information that was entered into each symbol comes to life to define what the part is, how it is mounted to the board, what keep outs exist around it, what are the constraints of the connections between the pins, etc. Once the netlist is imported you can define your board outline and set up the layer scheme.

The layers of a printed circuit board are what allows for the ability to contain a large number of electrical connections within a small area. To understand how the layers of a circuit board works it is easy to think of a system of water pipes in a building. Certain pipes are connected to a lot of things, other pipes are connected to only a few things, and some pipes are connected to nothing. These pipes represent the nets from the schematic, and the end points that are connected to them are the pins of the different components on the printed circuit board. Now just as in a building water system, you must take care in your design that only the proper pipes are allowed access to certain pipes of water for proper operation. Just as with water it is hard to route different bodies of water around each other while keeping them separate. Certain configurations become impossible with only 2 degrees of movement available. This is the condition that takes place on a single layer of a printed circuit board. Multiple routes of copper move around each other to connect different pins, but there are circumstances where pins must be connected but another route is in the way. We cannot simply pass through the route as it would short the two signals together. The only other option would be to jump over it or duck under it. This is where multiple layers come into play. Just like in a building water system, pipes must route up and down around each other inside the walls, the copper signal traces must route around each other inside the layers of the printed circuit board to connect to proper pins.

The feature of the printed circuit board that allows us to perform these jumping over and ducking under other signals is called a via. A via is essentially, a small hole drilled in the board, and then either coated or filled with conductive materials, likely copper. These vias act as elevators for electrical signals between the layers. This allows the different signals on the board to easily route around each other by simply jumping over and under other signals. This feature also makes it easy to distribute one signal over a large area, such as a ground plane. This allows you to connect many different components to a single signal on the board without having to individually find a path for each one. To perform this type of feature, the printed circuit board designer usually uses an internal layer as a ground or power plane layer and is able to distribute that signal across a large area of the board. Vias are then placed in strategic locations to provide that large signal near the proper pins on the outer layers of the board.

For our design we are expecting to design a four-layer printed circuit board. The four-layer design is an industry standard and is cost effective as well. Through research and investigation, it has been discovered that many two-layer boards are for quite simple and very cost sensitive designs. Boards that require more than four layers are often very dense or very complex configurations that would require a high level of printed circuit board design experience to execute properly. For these reasons we have decided to strive for a four-layer board design. Our expectation is that the printed circuit board layer stack-up will have two external signal layers and two internal power and ground plane layers. The power plane will offer us the appropriate power rails at the locations that are necessary for our design. The power plane will likely contain multiple polygons for each voltage rail. The ground plane will likely be nearly the entire area of the printed circuit board. This will allow easy access to ground signals from anywhere on the board. The two external signal layers will route all signal traces as well as some power and ground shapes. There is an added advantage of routing all signal traces on the external layers of the board by having the opportunity to directly interface with the signals for test and debug purposes, as well as the ability to cut the trace for rework purposes if the need arises.

8.0 Project Prototype Testing Plan

Our development team produced a prototype system prior to the final system demonstration. This prototype met the minimum engineering requirements (see section 2.4). This prototype was tested for complete functionality; all components used in the prototype were tested individually (when applicable). Our team aimed to develop two full working prototypes prior to the final demonstration, to confirm the accuracy of all measurements taken and actions performed by the system, as well as to provide an emergency backup if one of the prototypes were to otherwise fail. Instead a single working prototype was produced, and a backup PCB was fully fabricated. A working implementation of the design, using a dev board version of our selected microcontroller was also produced and functional. The following sections detail the testing procedures and environments with which our prototype and its components were tested.

8.1 Hardware Test Environment

All electrical components chosen for the design of the Automated Smart Garden will be individually tested in a controlled breadboard environment immediately upon receipt of said components. Such testing is necessary and required to ensure that each component is compatible with the power supply environment they will be working on, able to send and receive necessary data to and from exactly the other components intended, and efficient enough to meet the requirements of the system. These components included but are not limited to: all sensors (see section 5.4), the Microcontroller with WIFI module (see section 3.3.1, 5.2, & 5.3), the power supply (see section 3.3.2), and the LCD screen with pushbutton interface (see section 5.5).

The hardware test environment for the breadboard setup will be in an engineering lab environment that contains appropriate measurement and test equipment for proper circuit analysis and verification. This test environment will allow all designers to verify proper power generation or regulation within the breadboard environment, and also verify proper power sourcing and distribution to all components within the breadboard environment. The test environment will also give the designer the ability to probe and debug all components of the hardware to verify the appropriate hardware configurations and settings for the proper design. Finally, the test environment will allow the designer to debug the running software using either a provided debugging tool, or by probing the communication lines to capture data for verification. This hardware test environment will allow the designer to fully verify the high-level system design interoperability and allow for the development of customized firmware without having to wait for the full hardware development cycle to be completed. Once breadboard testing has been finalized and the prototype development process moves from development on the breadboard setup to the custom designed hardware, the breadboard prototype will

be safely stored for possible future development or use if the custom hardware is unable to demonstrate appropriate functionality.

Once the finalized custom hardware is developed and delivered, it will become necessary to test the custom hardware for proper functionality. A test of the prototype will be conducted in a controlled breadboard environment. This testing will include preliminary visual inspection that will ensure that there are no parts missing or misplaced. The visual inspection also verifies that there are no solder balls or other foreign pieces of material on the board from the manufacturing process that could cause shorts and contribute to board failure. The visual inspection also checks for the proper orientation of parts on the board. This means that the pin 1 location for all major components will be verified, as well as verification of the proper orientation of diodes and other polarized components.

Once the visual inspection is complete and it is verified that there are no issues that need to be addressed or corrected, impedance checks will be performed on the prototype board. These impedance checks are performed prior to initial power up of the board and are used as a way to verify that there were no mistakes that would cause a catastrophic board failure, if power were applied to the board. The impedance check process is performed by verifying that there are no shorts to ground from any power rail on the board. If this were the case, when the board was powered on there would be a direct short to ground and many components would likely be damaged instantly. Impedance checks are also performed between power rails themselves to verify that two power supply outputs are not connected by mistake, which would cause damaging failures as well.

Once the visual inspection and the impedance checks have been performed, and no major issues exist on the board that cannot be resolved, it is time to power up the board. It is important to think about this step early in the design process to make sure that this prototype test must incorporate all components that would be included in a final design and must demonstrate the performance of every component. Knowing this at the outset of the board design allows the designer to put certain features into place that can minimize risk during testing. One such tactic is to use unpopulated zero-ohm resistors to separate power rails from components downstream. This allows the tester to bring up each power rail individually without affecting other components. This is extremely beneficial when there are errors in the power design that cause rails to exceed their expected voltage levels. Without isolating the power rails during board bring up these errors could cause devastating damage to components on the board. Once the rails have been isolated and the power rails are verified to be at their proper voltage levels, we can reinstall the zero-ohm resistors to feed power to all components on the board.

Once all power rails are up and running and the board is being supplied with the proper power, the only item left to test is the hardware functionality. This process is usually very intertwined with the software integration effort as more often than not, testing hardware functionality requires software manipulation. For this a test

build will be created and loaded onto the microcontroller that will exercise the functionality of the hardware. Once all components are verified to be functional, the hardware testing is complete.

8.2 Hardware Specific Testing

8.2.1 Sensor Testing

Each sensor will be tested individually to ensure that it can capture accurate and consistent measurements from the surrounding environment. These tests will be conducted in controlled situations intended to mimic the conditions of the anticipated operational environment. It is important that all the sensors for the Smart Garden Controller are reliable and accurate for the data that will be recorded. The sensors are one of the main focus for the project due to the zones being monitored constantly for specific variations in the temperature, moisture, etc. If any of our sensors are faulty or don't live up to the error tolerance the group is looking for then the Smart Garden Controller may not be working properly for the user. The following sections will discuss the testing conditions regarding the Humidity & pressure sensor, the Anemometer, and the Soil moisture, temperature, & ambient light sensor.

8.2.1.1 Soil Moisture, Temperature, & Ambient Light Sensor Testing

The soil sensor used in our team's development will be tested for accuracy and consistency of measurement in a controlled soil sample environment, with dimensions of 10cm x 10cm x 10cm. To test the sensor's ability to measure soil moisture, and to set guidelines for our system's interpretation of the relative data returned by the sensor, the soil will first be measured in dehydrated soil, saturated soil, free air, and submerged in water. These edge cases will provide maximum and minimum relative readings so our system can scale the measurements to a useful data points, as well as indicating when the sensor is operating in non-ideal conditions outside of a soil plot, such as in free air or in liquid.

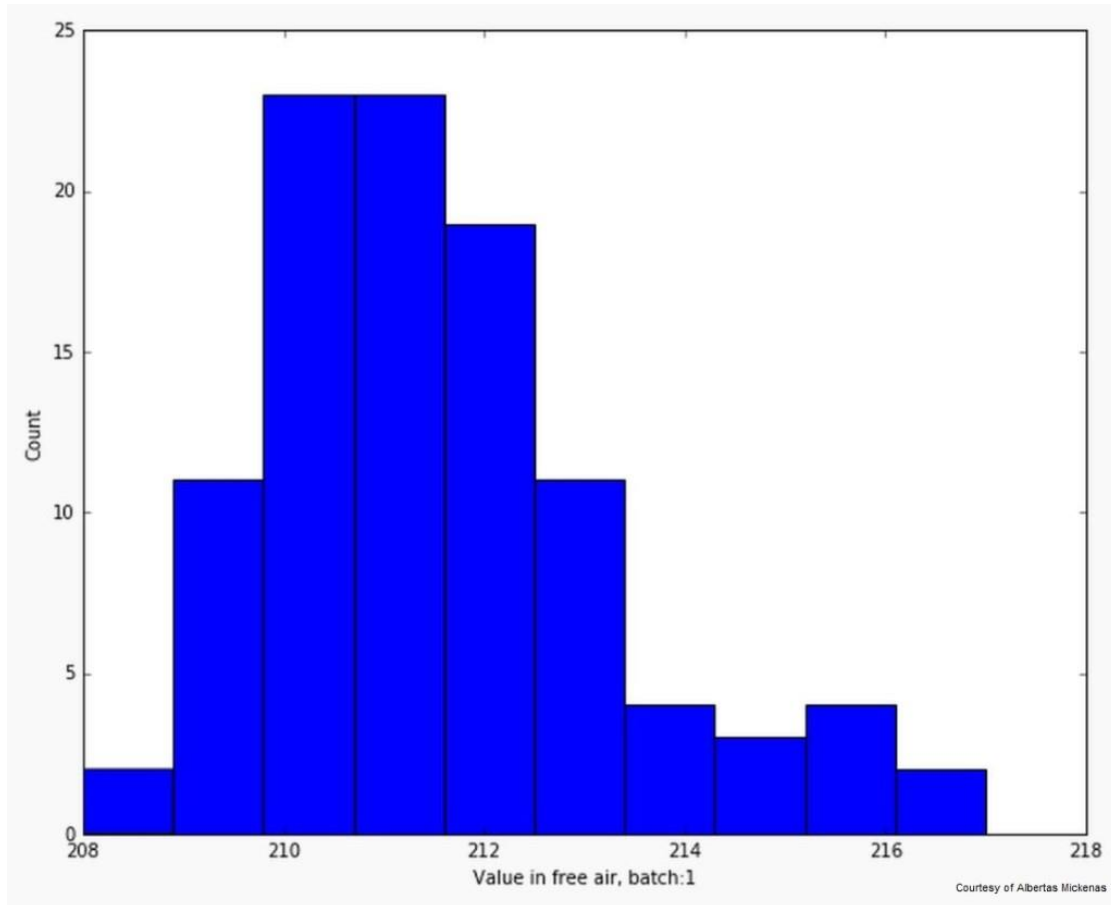


Figure 38 - Moisture Level Test Data

The Figure above shows the expected results of measuring the moisture levels in free air with this sensor, with an approximate average value of 212. These results are those of the seller before our team's purchase of these sensors. Our testing will include free air testing to estimate the accuracy of the sensors received compared to pre-purchase batch testing. Free air moisture levels should be the minimum possible values measured, as thus will be interpreted by the system accordingly. Ideally our system will present moisture data to the user as a value between the minimum (free air) and the maximum (submerged in water) values[C:Tindie].

The temperature measurements will be taken and compared to measurements made by valid thermometers in a local setting. The sensor's readings will be confirmed at 0°F, 32°F, 80°F, 100°F, and 120°F (and their respective Celsius equivalent temperatures). These temperature cases will incorporate minimum, maximum, average, and extreme temperature conditions that could be seen in the operational environment.

Similar to the relative measurements delivered as soil moisture data, ambient light measurements are recorded in relative units. The sensor will take ambient light measurements while physically obstructed to obtain a measurement as close as possible to absolute darkness. The sensor will then be tested under direct sunlight to measure average ideal conditions, then finally under variable energy artificial light, including but not limited to LED, Halogen, fluorescent, and standard incandescent-style bulbs. These testing scenarios will provide our team with maximum, minimum, and average ambient light readings such that the relative measurements taken by the sensor will be interpreted by the system as useful data.

8.2.1.2 Anemometer Testing

The anemometer component chosen for use in our team's development will be tested for accuracy of measurement against known wind speeds and compared to measurements taken using the Holdpeak 866B digital anemometer, a reliable secondary standalone anemometer. Measurements will be taken at 0 m/s wind speed (zero wind activity) in a controlled environment to confirm the minimum wind speed reading delivered by the component. Average and maximum wind speed measurements will also be taken and compared to known quantities to confirm accuracy on the higher end of the component's functionality.

The analog values delivered by the component should range between 0.4V and 2.0V, relative to wind speed between 0 m/s and 32.4 m/s, respectively. Our preliminary tests will provide us a baseline to accurately interpret those voltages as wind speed measurements to be used in our system's operation.

To confirm the sensor was working, a power supply was connected to the sensor providing 9.0 V. The red wire provided +9 v while the black wire was ground. A voltmeter was connected to the signal wire, the blue wire, and the signal voltage was measured. It was confirmed that at wind speed of 0 m/s the signal was at 0.4 V. Next, the signal voltage as seen by the digital voltmeter was recorded and compared to the speeds seen by the Holdpeak 866B digital anemometer. We found that at 1.7 m/s we had 0.435 V recorded, and at 2.6 m/s we had 0.44 V recorded. Although this setup proved useful in confirming the anemometer was in working condition it was not very precise. Therefore, as a next step in testing the Launchpad will be used to collect different voltage values. Storing the voltage values via the Launchpad should prove more accurate. These values will be needed in order to build the linear function to calculate the wind speed. Per the item description online the values should follow a linear function, however the function is not provided. As initial and final voltages, as well as min and max wind speed recorded, are known this should be enough information to build the simple linear function. However, this sensor having mechanical parts it is vital to do some testing for comparison. The following collage of images show the initial signal at 0 m/s and demonstrates the sensor is in working condition.

The above hardware specific testing in conjunction with the prototype was not performed for the anemometer because wind speed measurement was omitted from the final design of the system, and therefore the necessary analog to digital conversion of input from this sensor was never implemented. Upon further development or an increase in the scale of this system that includes wind speed measurement, ADC should be implemented and hardware specific testing should be performed.



Figure 39 – Anemometer Voltage Test

8.2.1.3 Humidity & Pressure Sensor Testing

Unlike the other sensor components used in the development of this system, the Bosch BME280 Humidity and Pressure sensor delivers measurements in the standard terms with which they are regularly used. Humidity measurements are saved as an integer between 0% and 100% relative humidity, and pressure measurements are saved as an integer between 300hPa and 1100hPa.

Both humidity and pressure readings taken by the prototype were consistently reasonably similar to reported weather information.

8.2.2 LCD Testing

The LCD component of our system will be tested for functionality and for compatibility with the CC3200MODA microcontroller. This testing will consist of displaying output on all possible segments provided by the LCD. Our team will also confirm that the screen is able to display the entire English Alphabet in at least one case size (ideally lower and upper case), numerical digits 0-9, a decimal point, and miscellaneous symbols and punctuation necessary to the operation of our system, and ease of understanding for potential users.

Initial testing will consist of checking if the device is functional. The positive connector of the power supply was connected to the 5 V input of the LCD. The LCD lit up as expected and this confirmed the LCD was working. Full functionality will require coding. The libraries available for this LCD are geared towards Arduino. Therefore, some development time will be required before full testing is possible. If the controller were using an Arduino a simple copy and paste would be possible. As the team has Arduinos available, Arduinos may be used first just to confirm all sections of the LCD are working, before putting forth more time and effort to migrate the libraries. The following image shows a successful power up test of the LCD display.

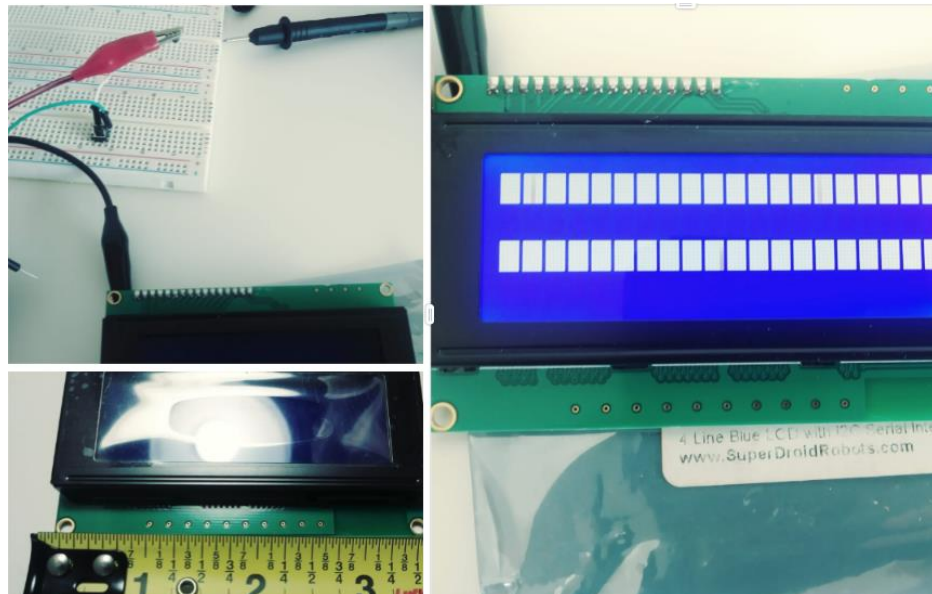


Figure 40 – LCD Power up test

8.2.3 Valve Switch Testing

The valve switch component of our system will be tested for functionality adhering to the system requirements, and compatibility with the CC3200MOD microcontroller. The valve switch must be able to perform binary water flow changes to at least one (1) plot of soil. This means the valve must be able to toggle the water flow at the direction of the program running on the microcontroller. If the switch requires manual intervention from developer(s) in order to perform such toggling, it is not viable for use in our system. It should be noted that the valve used in our design only enables a single direction of water flow and is intended for use with liquid water only, so testing acknowledges these constraints. The following table includes data regarding the valve's current consumption when used with different applied voltages ranging from 6Volts to 12Volts [CAdafruit-977].

Table 19 – Valve current consumption

Voltage	Current
6V	160 mA
7V	190 mA
8V	220 mA
9V	240 mA
10V	270 mA
11V	300 mA
12V	320 mA

8.2.4 Microcontroller Testing

The CC3200MOD Microcontroller will be tested for functionality and compatibility during the full system test. During hardware specific testing the microcontroller will be tested for standalone functionality and tested in conjunction with individual sensors that require the microcontroller for operation, including but not limited to the anemometer, the humidity & pressure sensor, the soil moisture, temperature, & ambient light sensor, and the LCD screen.

During standalone testing the microcontrollers ability to execute C code files will be confirmed using simple HelloWorld style programs and sample programs similar to software intended for full system operation. If the microcontroller fails to run these programs and return the appropriate output, then the component is not viable for use in our system.

During individual testing with the LCD screen the microcontroller must be able to display all messages in accordance with LCD testing, and the appropriate data for use in the operational environment, including but not limited to wind speed, temperature, humidity, pressure, soil moisture level, & ambient light level. This data must be visible on the LCD screen as directed by the program running on the microcontroller.

During individual testing with the humidity & pressure sensor the Microcontroller must facilitate the collection of measurements from the sensor. The sensor must receive adequate operational voltage and current consistently. The microcontroller must also be able to save such data for extended periods of time.

During individual testing with the anemometer the microcontroller must facilitate the collection of measurements from the sensor and run software to interpret such

analog data into useful numerical data. The microcontroller must also be able to save such data for extended periods of time.

During individual testing with the soil moisture, temperature, & ambient light sensor the microcontroller must facilitate the collection of measurements from the sensor. The measurements that are taken in relative terms (soil moisture & ambient light) must be interpreted into useful integer units by the software running on the microcontroller. The microcontroller must also be able to save such data for extended periods of time. Temperature data is recorded in degrees Celsius, so no interpretation computation must be performed by the microcontroller for temperature data, only the collection and preservation of such data is essential.

8.2.5 Chassis Testing

The 3-D printed plastic chassis included in our design will be tested for functionality and compatibility with other components. To function properly in the operational environment the chassis must be rain resistant from above, to protect the microcontroller and other electrical components. The chassis must also be large enough to house such components and must allow those components to be connected to the soil sensors, the anemometer, and the AC power supply from the grid. The humidity and pressure sensor must be able to gather accurate measurements from either inside the chassis or through a specialized opening in the container so as to maintain the integrity of the system, while still enabling the sensor. The chassis must also allow the user to access the housed components if necessary. This chassis will be made using a 3D printer one of the team member owns called the Originator i1. It is able to print using a variety of plastics and can heat as high as 250 degrees Celsius. It has a print volume of 150mm x 150mm x 140mm and has a heated bed that can go as high as 90 degrees Celsius. The group will be testing different plastic material ranging from PLA, ABS, PETG, etc., which all have different pros and cons when it comes to printing. As mentioned before, the group would like the chassis to be as protective as possible, but allowing for accurate sensor readings from within. This can be done by making sure the perimeters of the chassis design are thick while keeping the inside fairly uncluttered. It is also important to make sure that the team uses a material that has fairly high heat tolerance due to the Smart Garden Controller being in an outside environment for most of the time and wanting to avoid the possibility of something melting and causing failure to the device. The team will be using mainly Solidworks and Autocad to design the chassis to guarantee that the design is optimal and make sure that nothing is incorrect. Slic3r will be used to slice the design onto the 3D printer and allow us to change certain aspects of the design, such as the number of perimeters, layer height, speed at which the printer is printing, how it prints, etc. Overall, the team is very confident that the chassis will be able to provide a nice and secure shield from the outside and make sure that the information from the sensors that is recorded is accurate.

8.2.6 Miscellaneous Hardware Testing

Other miscellaneous and non-electrical hardware used in our system must be tested or somehow evaluated for viability in our design. These hardware components include but are not limited to: hoses, hose connectors, sprinkler heads & bulbs, and electrical cords. These components will be testing for their physical integrity when applicable, to ensure that they are viable for outdoor use in the operational environment. These components will also be tested for functionality to ensure that they are ideal for use in our system, and to confirm that (in the case of hose components) no water escapes the system and is wasted.

8.3 Software Test Environment

As software grows and becomes more complex it may become difficult to keep track of all the functionality. Documentation will be used to ensure all groups are on the same page. However, with all precautions, the software may still not behave as expected with. Software not working as expected would often be called a bug. Testing of each part of the software developed will be completed before implementation, to avoid having any surprises once deployed.

The combination of all the software used in this project will consist of three main categories. The categories will be the microcontroller software, the web server, and the web application. Each of these software will be coded in a different integrated development environment. Each development environment will have its own testing features. For the MCU software code composer studio will be used. Eclipse will be used for the web server software. Finally, Visual Studio Code will be used for the web application.

Although each IDE has very complete testing functionality, other software will be used for troubleshooting as well. A terminal application, such as Tera Term, will be used to review debug messages for the MCU code. Postman will be used to troubleshoot and debug the web server and database connection. Finally, Firefox with the firebug plugin will be used for debugging the web application.

Testing the smart speaker interaction will be the last part of the software testing. Testing the smart speaker part of the software will be done using the tools provided by amazon to develop skills. Also, an Echo dot will be used.

8.4 Software Specific Testing

The following section and subsection will go over the steps to test each part of the software. This is important because it helps go over the area that will control the data we collect and how it is used. The software used for testing, objective, procedure, and logic will be described in the following subsections.

8.4.1 Database Software

The database will need to store and retrieve all the permanent data used by the web application and the controller. Testing of the database will ensure the data storage system is ready.

Objective: Confirm the database system is ready to store and provide data.

Procedure: To test the database software the following steps will be performed.

1. Confirm the database management software (DBMS) is running. This can be done by running “MySQL -u root” on the server hosting the DBMS software. Following is an example output when the process is running.

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

```
Your MySQL connection id is 143
```

```
Server version: 5.5.60-0+deb8u1 (Raspbian)
```

2. Confirm the DBMS is listening in the correct port, default is 3306. We can do this by running “netstat -tln” command in Linux to look for listening ports. The following is an example output.

```
Active Internet connections (only servers)
```

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	0.0.0.0:3306	0.0.0.0:*	LISTEN

3. Run queries against the database to confirm functionality. This will be done using MySQL Workbench.

Expected Result: The port will be listening correctly and records will be created and retrieved successfully. The following is example user records data returned from the database for “select user_number, user_name, password from user”.

user_number	user_name	password
1	renancoelho	Welcome1
2	billnelson	Welcome1
3	geosn	Welcome1
4	restinserted	NULL

8.4.2 Web Server Software

The web server software will be the broker between the database and the client integrations. The following tests will confirm the system is providing the functionality required. The first test will focus on the web server status, followed by tests to retrieve data from the database from numerous tables.

Test: Web Server Status.

Objective: Confirm the web server is ready to provide data to the clients to the database and from the database to the client. If the web server is down, the clients will not be able to access the database.

Procedure:

1. On the server the tomcat application is installed run “netstat -tln” and confirm it is listening on the port configure. The default it 8080.
2. Open a browser and navigate to the URL http://<tomcat_server_ip>:8080. If the web server is working correctly a webpage should be displayed.
- 3.

Expected Result: Default webpage for the tomcat web server will be retrieved.

Test: Get records from DB via web server in JSON format

Objective: Confirm the class providing records information is successfully retrieving data from the database. In the following procedure the user table will be used as an example.

Procedure:

1. Open up a browser.
2. Navigate to <tomcat_server_ip>:8080/restserver/webapi/user/<number>, where <number> is the user_number for the user we want to retrieve.

Expected Result: A string formatted as JSON will be returned. The following output is an example.

```
{"user_name": "renancoelho", "user_number": 1}
```

8.4.3 MCU Software

The MCU software will interact with all the hardware in the garden controller. The tests will consist of ensuring each component can receive commands and respond appropriately. Some other components will require testing of specific functionality, such as wireless capabilities.

Test: Anemometer Sensor Data Collection.

Objective: Confirm the written code can successfully collect and use the anemometer. The anemometer will provide analog output from 0.4V to 2V linear.

Procedure:

1. Connect the anemometer to the garden controller, TI's LaunchPad at this moment.
2. Run method function to read output voltage from anemometer.
3. Transform data based on linear function built from comparing primary anemometer output data to handheld anemometer.
4. Send reading to terminal.

Expected Result: Wind speed displayed on terminal should match that as seen on handheld anemometer.

Test: Soil moisture, temperature, and light.

Objective: Test code which will communicate with soil sensors. The same sensor provides moisture, temperature, and light.

Procedure:

1. Connect sensor cables to TI's LaunchPad.
2. Run method function to read controlled inputs, known moisture, temperature, and light.
3. Print inputs to terminal.
4. Run method function to read new set of controlled inputs, known moisture, temperature, and light.
5. Print inputs to terminal.
6. Compare with new values.

Expected Result: Moisture, temperature, and light outputs to terminal should match that of the controlled inputs. A 5% margin of error should be acceptable.

Test: WiFi Connectivity

Objective: Verify software can correctly connect to a wireless network and upload data to database via the web server.

Procedure:

1. Provide parameters for network SSID and password.
2. Send HTTP Post request to web server with sample sensor input
3. Print confirmation code to terminal

Expected Result: From the database it should be seen that test input was correctly added to the database.

Test: Humidity and pressure.

Objective: Test code which will communicate with humidity and pressure sensor.

Procedure:

1. Connect sensor to TI's LaunchPad via breadboard.
2. Run function to read controlled inputs for humidity and pressure.
3. Print inputs to terminal.

Expected Result: Inputs should successfully be printed to the terminal.

8.4.4 Web Application

The web application should be the software the user interacts directly the most, once the controller is setup and connected to the internet. The web application will allow the user to review all metrics collected as well as configure the controller without having to go outside. The following are some of the tests to ensure this part of the system works as expected.

Test: Create user and login

Objective: To successfully create a user account. A user account will be needed to review any metrics. After a user account is created, a username and password will be sent with each request to retrieve data.

Procedure:

1. On a browser, open the login/new user page.
2. Click on “New User”.
3. Complete form and click submit.
4. Go to login page.
5. Complete login form and click submit.

Expected Result: User record should be created. After the user record is created, the user should be able to view his controller management page.

Test: Review metrics.

Objective: To confirm the metrics collected by the controller can be viewed in a browser successfully.

Procedure:

1. Login to web application.
2. Click on metrics.
3. Confirm diagrams are available and correctly display.

Expected Result: All metrics collected by the controller and the zones should be available for review by the user.

8.4.5 Smart Speaker Integration

The smart speaker should provide the wow factor of the project. Many of the new products these days can be integrated with smart speaker to receive commands from it. The following test will check if the integration is working properly.

Test: Trigger zone watering via voice command.

Objective: To successfully trigger a valve to be turned on and water a zone.

Procedure:

1. Say “Alexa, ask smartcontroller to water zone one for five minutes”.

Expected Result: Zone one should turn on the sprinkler for five minutes.

9.0 Administrative Content

The following sections provide clerical information regarding our development team's design process, meetings, decisions, and interpersonal communication between members. These sections also discuss the possible methods of funding for our project.

9.1 Milestone Discussion

The key to success for any project is to have proper communication with teammates and goals set in order to efficiently and reliably complete the project without stressing a lot on what is or isn't done and who is responsible. Without proper scheduling a project can become unorganized and leave team members confused on who is responsible for what and when they need to complete something by.

This can lead to team members procrastinating on their role in the project and leaving others worried if something will not be complete. For the Smart Garden Controller, the project will use the next several months as efficiently as possible to maximize the team member's contribution to the project and to make sure the project is ahead of the deadlines. The team has agreed to meet at minimum once per week and to split the course load evenly between the four members. The milestones set will help the team by having a concrete date of when a specific task is to be completed by, as well as, giving peace of mind to the other team members.

For the Smart Garden Controller, the team will be focusing the first couple of months on researching what components and features the project should have and testing to make sure they are what the project requires to function properly. The team does not want to put too many difficult features that may require a lot of time and testing because of the short time frame the project has to be completed by. Proper testing of the hardware elements and starting the software is essential to making sure the Smart Garden Controller is efficient and reliable as possible. It is also essential that the team recognizes that some of the milestones set will take longer to complete than others and thus may require more effort from our team to have them completed on time. The team wants to make sure that none of the members goes past any of the deadlines set in order to stay on track, unless an unusual circumstance occurs.

Throughout the first semester of Senior Design the team will be concentrating on making sure the requirements and goals of the project are explicitly set and known, as to avoid any potential confusion. A lot of time will go into researching past Senior Design projects and similar products to the Smart Garden Controller to help better the team's knowledge about what is to be expected. It is essential that we order all of the parts during this semester in order to save a lot of time for the next semester

when we are actually building the Smart Garden Controller, since to keep a minimum cost ordering parts outside of the United States may be the better option.

The second semester of Senior Design will be focused on designing the Smart Garden Controller and putting all of the hardware and software elements together to get the device working. This will put all the research and testing that the team acquired in the first semester to the test and lead to the creation of the Smart Garden Controller.

There will be a working prototype by the end of the second semester with a PCB made and all the subsystems working together. At any point in the schedule if a team member needs help with a specific part, they should ask the team for advice and not be afraid to communicate. The team may decide to meet more often than what the schedule actually says because the more time there is to discuss about the project, the faster it will be completed. The schedule below shows what exactly our team is expecting to complete and the date of when it should be completed by. Some of these dates are looking into the future and subject to change.

Senior Design 1 Schedule:

14 May 2018 ~ 28 May 2018: Form group and introduce each team member to each other. Each member come up with an idea for a project and post it for other members to see and rate which project to pursue.

29 May 2018: First group meeting. Discussed the pros/cons of the four projects each team member posted and what each member felt was the best project to take as our Senior Design project. Also set a mandatory weekly meeting every Tuesday or Thursday to discuss the project and each member's current progress.

31 May 2018: Senior Design Boot Camp. Discussed how to succeed in having a good project and the pros/cons of each team member. Also went through valuable team building exercises to gain more unity within our teammates.

05 June 2018: Second group meeting. Discussed requirements to the project and tentative schedule of deadlines.

08 June 2018: 10 Page Initial Documentation due. Each group member should be responsible for about 2 to 3 pages each.

12 June 2018: Senior Design Project idea approved by professor Lei Wei and Samuel Richie.

19 June 2018: Third group meeting. Discussed parts for the Smart Garden Controller and updated 10 Page Initial Documentation with comments by professor Lei Wei. Acquired four CC3200 Launchpads for the project to due testing on. Also

Discussed 60 Page Draft Documentation and assigned specific sections for each team member to work on.

29 June 2018: Fourth group meeting. Follow up on the 60 Page Draft Documentation and set deadline for completion by the first of July.

03 July 2018: Fifth group meeting. Final check up on the 60 Page Draft Documentation. Discussed plans for 100 Page draft documentation and assigned roles to teammates.

06 July 2018: 60 Page Draft Documentation due. Each group member should be responsible for about 15 pages each.

10 July 2018: 60 Page Draft Documentation reviewed by Professor Samuel Richie. Small changes to be made on the draft, including adding table and figure numbers and slight format changes.

12 July 2018: Sixth group meeting. Discussed some of the hardware elements and changes we may make to the design. Assigned what sensors and parts each group member will be responsible for purchasing and how many extra components to buy in case something fails.

19 July 2018: Seventh group meeting. Further discussed the hardware elements that are being purchased and what LCD screen to buy for the Smart Garden Controller.

20 July 2018: 100 Page Draft Documentation due. Each group member should be responsible for about 25 pages each.

22 July 2018: Hardware purchased and tested. Each group member should be responsible for purchasing sensors/components as needed for the Smart Garden Controller and have spares.

26 July 2018: Eighth group meeting. Discussed the end of the semester and overview on the final documentation to make sure everything is in order. Also tried setting some deadlines for the next semester and got each team members schedule in order to make sure the project will be finished by the end of the next semester.

30 July 2018: 120 Page Final Documentation due. Each group member should be responsible for about 30 pages each. Focus of the team after this is now to build the Smart Garden Controller and making sure it is running by the end of the following semester.

Senior Design 2 Schedule:

- 07 August 2018: Ninth group meeting.
- 15 August 2018: Software Design complete.
- 16 August 2018: Tenth group meeting.
- 23 August 2018: Eleventh group meeting
- 30 August 2018: Twelfth group meeting.
- 01 September 2018: Schematic Complete.
- 06 September 2018: Thirteenth group meeting.
- 13 September 2018: Wi-fi portion of firmware complete.
- 20 September 2018: Fourteenth group meeting.
- 27 September 2018: Fifteenth group meeting.
- 01 October 2018: Layout complete.
- 04 October 2018: Sixteenth group meeting.
- 11 October 2018: Seventeenth group meeting.
- 15 October 2018: Calendar/Scheduling Firmware complete.
- 21 October 2018: PCB printed and tested.
- 25 October 2018: Eighteenth group meeting.
- 31 October 2018: Midterm Demonstration.
- 08 November 2018: Nineteenth group meeting.
- 15 November 2018: Web application code complete.
- 22 November 2018: Twentieth group meeting.
- 27 November 2018: Final Project Demonstration

9.2 Budget and Finance

The project completely is self-sponsored. The students will pay for any and all expenses. The following budget is not final as the components have not been decided upon at this moment. As research continues, components may be removed or added to the project due to requirements, constraints, and budgeted limitations. The following prices are based on online research for low quantities of components and no current production level costing effort has been performed. The budget and the prices of the components will be updated once decisions are finalized and there will be a continuing effort to reduce bill of materials costs. To take into account faulty parts and testing malfunctions or errors that result in damage to the parts, most parts will be ordered three times more than necessary. Therefore, the budget will have a column to display the value for a single final product and another column for the total cost of the project.

Table 20 – Estimated Cost

Description	Unit Cost	Project Parts Count	Project Cost Estimate	Final Product Parts Count	Final Product Cost
MCU/Wi-Fi Module	10	3	30	1	10
Anemometer	45	2	90	1	45
Humidity & Pressure sensor	16	3	48	1	7
Moisture, Light, & Temperature sensor	13	3	39	1	13
Batteries	2	8	16		
Water Valves	7	8	56	4	28
PCB	30	3	90	1	30
LCD screen	27	3	81	1	27
Chassis Enclosure	10	3	30	1	10
Power Source	10	2	20	1	10
Miscellaneous			50		
Total Cost			\$550 (\$1100 buffer)		\$190

Note: Some items do not need to be included in the final product which would go to an end user. For example, the end user could buy its own battery. Therefore, battery pricing would be empty for final product. Also, some components do not have a price set at the moment and are to be left empty.

Included below is the final bill of all materials purchased for this system's development.

Table 21 – Final Bill of Materials

Item	Cost	Item	Cost
Buttons & connectors	\$93.63	Launchpad	\$59.99
PCB parts	\$236.47	Anemometer-test	\$22.99
PCB fabrication	\$199.40	BME sensors-test	\$19.86
Hoses, valves, etc	\$68.45	BME sensors	\$61.40
Cable	\$12.79	Soil sensors-test	\$74.20
Solenoid valves	\$105.53	Soil Sensors	\$73.25
TI debug emulator	\$99.00	Dirt & plants	\$19.76
SD1 Documentation	\$106.03	Showcase materials	\$5.98
Garden box	\$106.95	3D printer filament	\$27.38
Anemometer-sensor	\$58.95	SD2 documentation	\$75.00
LCD screens	\$83.45	Total project cost:	\$1,668.40
Buttons-test	\$6.57		
Router	\$17.11		
Cables	\$6.98		
Breadboard	\$9.99		
Regs Tubin	\$17.09		

9.2.1 Sponsorship

Building a relationship between our development team and third-party companies or individuals would open up opportunities for the development of our system to sponsored or subsidized by those parties. In absence of formal sponsorship, our development team must fund our own system. The following sections detail possible relationships with third parties that would be closer to ideal than a self-sponsored project.

9.2.1.1 Guard Dog Valves

Our team's primary target for corporate sponsorship is a partnership or involvement with Guard Dog Valves Inc. The company, and their CEO Doug Guidish, have strong environmental goals regarding the preservation of the planet's potable water resources. The main product sold by Guard Dog Valves is a preventative valve addition that prevents water waste in the home. See below for technical details regarding this product[C:Guarddogvalves].

Table 22 – Guard Dog Valves Data

Description	Value
Power Supply	6.0V (4 AA batteries)
Minimum expected life span	1 year
Maximum approximate water saved	757 Liters/day
Approximate Installation time	30 minutes

Our team feels that the objectives of our proposed system are in the same vein as the product offered by Guard Dog Valves, while much more complex. Our system is designed to minimize water waste and ensure that consumers use exactly the water that is necessary for a garden to thrive. The involvement of users with our system is a step further to achieve those goals by creating a relationship between consumers, the resources they use, and their homes and gardens.

9.2.1.2 Orange County Utilities

While any sort of sponsorship with local government seems unlikely, our team is interested in a relationship with Orange County Utilities. Much like the private industry example discussed above, OCU have been involved with water conservation efforts for at least 30 years during their participation in the Water Conserve II cooperative reuse project [C:OCFL]. This project is an effort by Orange County, the City of Orlando, and the local agricultural community to work together to reduce water waste and increase the health and integrity of local crops yields. The project used advances in irrigation technology, an aquifer recharge technology called rapid infiltration basins (RIBs), and the Florida Department of Environmental Protections access to publicly reclaimed water to reuse waste water, improve crop irrigation, and retain the strength of Florida's aquifers.

Our team hopes that a possible relationship with OCU or other municipal governments or institutions could further our efforts in designing a system with the same key objectives as the projects performed in the past. Our team is also interested in cooperative opportunities to work with local and governmental organizations during or after the production of our proposed system. While the current scope of our system doesn't prioritize large scale water conservation, our team would be interested in expanding that scope to encompass large scale projects with other organizations.

Appendices

Appendix A: Permissions

7/23/2018

Mail: [REDACTED]

Re: [[Press/Media]

Adafruit Industries <support@adafruit.com>

Fri 7/20/2018 11:02 AM

To: Temple Corson <[REDACTED]>

hi temple,

that link does not work, however, if they are photos of adafruit products, it's OK to use.

thanks,
adafruit support, phil

On Thu, Jul 19, 2018 at 7:02 PM, Temple Corson <support@adafruit.com> wrote:

contactname: Temple Corson

email address: [REDACTED]

message text: Temple A. Corson IV

[REDACTED]

University of Central Florida

My senior design team ordered and is using your product at the link below.
We would like to use the images from this page in our documentation with
credit to Adafruit and/or other necessary parties.

https://www.dropbox.com/referrer_cleansing_redirect?hmac=%2Fw7a%2BF%2B9rWJEvEG%2F4PI58gBlavGhWVJKbFBRE%3D&url=https%3A%2F%2Fwww.adafruit.com%2Fproduct%2F733%3Fqclid%3DCjwKCAjw06LZBRBNFwA2vgMVBW%2FD87NNeGQ6sac6pIMNIVTY1PKwNIVNaEzLd5goA6nW1cEapGDRoCQlgQAvD_BwE

Thanks,

Temple A. Corson IV
Client IP: [REDACTED]

<https://outlook.office.com/owa/>

1/1

7/23/2018

Mail - [REDACTED]

[Tindie] Re: Image Permissions

mic (tindie) <support@tindie.zendesk.com>

Mon 7/23/2018 10:48 AM

To: Temple Corson <[REDACTED]>

##- Please type your reply above this line -##

Your request (#11655) has been updated. Reply to this email or click the link below:
<http://tindie.zendesk.com/hc/requests/11655>



mic

Jul 23, 7:48 AM PDT

Hi Temple,

Yes you can use pictures from the store no problem. If you want to credit me, my name is Albertas Mickenas. Also you can provide a link to the source files of the sensor:

<https://github.com/Miceuz/i2c-moisture-sensor>

Attachment(s)

[signature.asc](#)



James N (Tindie)

Jul 20, 1:31 PM PDT

Hi Temple,

Thanks for writing in.

If you'd like permission to use an image, you will want to reach out to the seller directly. I have CC'd the seller, Albertas, here to best assist you, but you can contact any seller directly from the product page by clicking on the "contact" button for the seller. It is on the right side of the page under the store information.

Albertas, Temple from University of Central Florida would like permission to use your product images in one of their projects. I'll include their full message below:

"Temple A. Corson IV

[REDACTED]

<https://outlook.office.com/owa/>

1/2

7/23/2018

Mail - [REDACTED]

[REDACTED]
University of Central Florida

My Senior Design team ordered and is using the product at the link below, and we would like to use the product images in our documentation with credit to Tindie and/or the seller(s).

<https://www.tindie.com/products/miceuz/i2c-soil-moisture-sensor/#specs>

Thanks,

Temple A. Corson IV"

Thanks,
James



Temple Corson

Jul 19, 3:56 PM PDT

Temple A. Corson IV

[REDACTED]
[REDACTED]
University of Central Florida

My Senior Design team ordered and is using the product at the link below, and we would like to use the product images in our documentation with credit to Tindie and/or the seller(s).

<https://www.tindie.com/products/miceuz/i2c-soil-moisture-sensor/#specs>

Thanks,

Temple A. Corson IV

This email is a service from Tindie. Delivered by [Zendesk](#).

[V0VMZL-0M08]

<https://outlook.office.com/owa/>

2/2

The MIT License

SPDX short identifier: MIT

[Further resources on the MIT License](#)

Copyright <YEAR> <COPYRIGHT HOLDER>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Citations

- [1] http://batteryuniversity.com/index.php/learn/article/secondary_batteries
- [2] <http://i2c.info/i2c-bus-specification>
- [3] <http://i2c.info/i2c-bus-specification>
- [4] <http://ww1.microchip.com/downloads/en/DeviceDoc/70005304B.pdf>
- [5] <http://www.aoml.noaa.gov/hrd/tcfaq/A5.html>
- [6] <http://www.cablefree.net/wireless-technology/history-of-wifi-technology/>
- [7] <http://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/>
- [8] <http://www.circuitstoday.com/pcb-manufacturing-process>
- [9] http://www.csd.uoc.gr/~hy428/reading/i2c_spec.pdf
- [10] <http://www.electronicdesign.com/4g/talk-multiple-devices-one-uart>
- [11] <http://www.futureelectronics.com/en/microcontrollers/microcontrollers.aspx>
- [12] <http://www.hydraulicspneumatics.com/200/TechZone/HydraulicValves/Article/False/6409/TechZone-HydraulicValves>
- [13] <http://www.informit.com/articles/article.aspx?p=23760&seqNum=3>
- [14] <http://www.ocfl.net/?tabid=371#.Wz59F9JKiUk>
- [15] <http://www.oracle.com/technetwork/java/codeconventions-150003.pdf>
- [16] <http://www.pearsonitcertification.com/articles/article.aspx?p=1329709&seqNum=4>
- [17] <http://www.powerelectronics.com/basics-design/power-management-basics-power-supply-fundamentals>
- [18] <http://www.seeedstudio.com/blog/2017/04/05/pcb-manufacturing-process/>
- [19] <http://www.ti.com/lit/an/slva704/slva704.pdf>
- [20] <http://www.ti.com/lit/ug/sprugp1/sprugp1.pdf>
- [21] <http://www.ti.com/product/bq40z80>
- [22] <http://www.ti.com/product/bq78350-r1>
- [23] http://www.ti.com/solution/appliances_user_interface_connectivity_modules?variantid=22543&subsystemid=23716

- [24] <http://www.vleo.net/i2c-anemometer-cheap-wind-data-logger/>
- [25] <http://www.waterconservii.com/>
- [26] <https://barrgroup.com/Embedded-Systems/Books/Embedded-C-Coding-Standard>
- [27] https://en.wikipedia.org/wiki/Bit_banging
- [28] <https://en.wikipedia.org/wiki/Bluetooth>
- [29] https://en.wikipedia.org/wiki/Bluetooth_Low_Energy
- [30] https://en.wikipedia.org/wiki/Flow_control_valve
- [31] <https://en.wikipedia.org/wiki/Garden>
- [32] <https://en.wikipedia.org/wiki/I2C>
- [33] https://en.wikipedia.org/wiki/Rechargeable_battery
- [34] https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter
- [35] <https://github.com/weewx/weewx/wiki/Raspberry-Pi-weather-station-with-i2C-sensors>
- [36] https://greeniq.com/product_tech.htm
- [37] <https://hackaday.io/project/12397-modular-weather-station>
- [38] <https://ieeexplore.ieee.org/document/7586376/>
- [39] <https://ieeexplore.ieee.org/document/7786995/>
- [40] <https://ieeexplore.ieee.org/document/8262521/>
- [41] <https://ieeexplore-ieee-org.ezproxy.net.ucf.edu/stamp/stamp.jsp?tp=&arnumber=8262521>
- [42] <https://whatis.techtarget.com/definition/UART-Universal-Asynchronous-Receiver-Transmitter>
- [43] https://www.adafruit.com/product/1733?gclid=CjwKCAjw06LZBRBN EiwA2vgMVfBWD87NNeGQ6sao6piMNIPTY1PKwN1VNaEzLd5goA6nWt LcEapGDRoCQIgQAvD_BwE
- [44] <https://www.adafruit.com/product/996>
- [45] https://www.adafruit.com/product/997?gclid=CjwKCAjwmufZBRBJE iwAPJ3LpvpEqNYrzU_Ewgsbn3h042Qjzr6ckHO76OXAGQyCpsaFzwuIM vV77xoCICgQAvD_BwE

- [46] <https://www.bluetooth.com/specifications/bluetooth-core-specification>
- [47] <https://www.egr.msu.edu/classes/ece480/capstone/fall14/group04/assets/application-paper-of-august.pdf>
- [48] <https://www.espressif.com/en/products/hardware/esp8266x/overview>
- [49] <https://www.espressif.com/en/support/explore/get-started/esp8266/getting-started-guide>
- [50] https://www.espressif.com/sites/default/files/documentation/0c-esp-wroom-02_datasheet_en.pdf
- [51] <https://www.flomatic.com/valves/automatic-control-valves/flow-control-valves/>
- [52] <https://www.freertos.org/about-RTOS.html>
- [53] <https://www.goodhousekeeping.com/home/gardening/advice/a25635/gardening-dangers/>
- [54] <https://www.guarddogvalves.com>
- [55] <https://www.guarddogvalves.com/about-us>
- [56] <https://www.homedepot.com/b/Outdoors-Garden-Center-Watering-Irrigation-Sprinkler-Timers/N-5yc1vZc63j>
- [57] <https://www.lifewire.com/what-is-802-11-wireless-protocol-2378244>
- [58] <https://www.livestrong.com/article/68617-pros-cons-growing-own/>
- [59] <https://www.microchip.com/wwwproducts/en/ATWINC1500>
- [60] https://www.mouser.com/new/bosch/bosch-bme280/?gclid=CjwKCAjw06LZBRBNEiwA2vgMVfhOVUpBKB9FYJdoA9kJ9rYLSz8xYI_EWSJxZ-SOHyZQJUSON6usghoCx4QQA_vD_BwE
- [61] https://www.mouser.com/new/bosch/bosch-bme280/?gclid=CjwKCAjw06LZBRBNEiwA2vgMVfhOVUpBKB9FYJdoA9kJ9rYLSz8xYI_EWSJxZ-SOHyZQJUSON6usghoCx4QQA_vD_BwE
- [62] <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>
- [63] <https://www.pcbcart.com/article/content/PCB-manufacturing-process.html>

- [64] <https://www.postscapes.com/smart-irrigation-controllers/>
- [65] <https://www.scientificamerican.com/article/how-does-wi-fi-work/>
- [66] <https://www.superdroidrobots.com/shop/item.aspx/4-line-blue-lcd-with-i2c-serial-interface/1838/>
- [67] <https://www.the-ambient.com/reviews/best-smart-garden-tech-465>
- [68] <https://www.tindie.com/products/miceuz/i2c-soil-moisture-sensor/#specs>
- [69] <https://www.tindie.com/products/miceuz/i2c-soil-moisture-sensor/#specs>
- [70] <https://angular.io/docs>
- [71] <https://getbootstrap.com/>
- [72] <https://www.TypeScriptlang.org/>
- [73] <https://www.chartjs.org/>