

Automated Greenhouse Watering and Heating System for the Schenectady ARC

Edwin Garcia-Flores and Lilah Kelly
ECE 499: Computer Engineering Capstone
Project Advisor: Professor James Hedrick

UNION COLLEGE

Mar 17, 2022

Introduction and Project Definition	4
Background	6
Design Requirements.....	8
Specifications & Constraints.....	8
Performance.....	8
Safety.....	10
Energy.....	11
Economics.....	11
Environmental.....	11
Interface & Accessibility.....	12
Electronic Constraints.....	12
Standards.....	13
Design Alternatives.....	15
Wireless Devices.....	16
Network Topology.....	17
Minimizing Power.....	18
Waterproofing.....	20
Design.....	22
Overview.....	22
End Device.....	23
Temperature Sensor.....	24
Moisture Sensor.....	27
Sleeping.....	27
Coordinator Device.....	28
Communication between end devices and coordinator.....	28
Graphical User Interface.....	29
Parts.....	31
Testing.....	32
Preliminary Results.....	32
Test Plan.....	33
Implementation Schedule.....	35
References.....	36
Appendices.....	37
Send Temperature.....	37

Read.....	37
Coordinator XCTU Configuration Screenshots using DigiMesh.....	38
End Device XCTU Configuration Screenshots using DigiMesh.....	40
Table of TMP36 Output Voltage vs Temperature.....	43
Previous team schematic.....	44

Table of Figures and Tables

Figure 1	9
Figure 2.....	15
Figure 3	22
Figure 4	23
Figure 5	25
Figure 6.....	28
Figure 7	30
Appendix Figure 1	37
Appendix Figure 2	37
Appendix Figure 3	38
Appendix Figure 4.....	38
Appendix Figure 5	39
Appendix Figure 6	40
Appendix Figure 7	40
Appendix Figure 8	41
Appendix Figure 9	41
Appendix Figure 10	42
Appendix Figure 11	43
Appendix Figure 12	44

Introduction and Project Definition

The ARC New York is an organization that “provides support and services to people with intellectual, developmental, and other disabilities, emphasizing choice and community engagement” [1]. In the Schenectady branch, one of the more notable ways community engagement is promoted is through the local greenhouse, where individuals can help care for and sell various plants. This way, those with disabilities are able to contribute to their community and gain experience for jobs in the future.

Although there are many commercial systems available for the ARC to purchase, a customized system is both cheaper for the branch and able to cater to specific accessibility needs of the community. The current system consists of a sprinkler structure that is activated with manual switches, heating pads for each plant bed, and a solid state relay connected to a power source. It also has moisture and temperature sensors connected to a controller and touchscreen to display data from each plant bed. However, there is still a need for improvement of both accessibility and safety for the project. Some aspects of the system require in-depth knowledge of electronics for troubleshooting, and manual labor to operate. In order to help with this issue, a graphical user interface will be implemented on a touchscreen which will offer both manual and automatic modes for the greenhouse. The automatic mode is intended to care for the plants when people are unavailable or unable to visit the greenhouse. The system also has numerous wired connections, which can be dangerous when used in the wet climate of the greenhouse. Reduction of wires by using wireless communication devices and waterproofing the system will help eliminate this hazard.

We have chosen to work on this project largely due to the positive and lasting impact it will have on the Schenectady community. There is also a strong personal connection to the

cause, as we have family members with disabilities. Thus, we have put in considerable effort with researching, testing, and implementing the improved system. Further information on this project and the intended improvements are detailed in the following Background section, as well as the design requirements we considered and applicable standards. The Design Alternatives section will delve into the research done on the improvements, with pros and cons of each choice explored before justifying the final decisions. Next, the Design section will have an in depth discussion covering each aspect of the system, including the roles and responsibilities of each component. Past and future intended testing of the system will be addressed in the Testing section, along with results that have been observed thus far. The Implementation Schedule section will display a mapping of the progress expected to be made in the future. Finally, the Appendix section will have code from the system, tables of sensor data, and pictures of component configurations and the previous team's schematic.

Background

The Schenectady ARC aims to provide individuals with developmental disabilities the resources, services, and support that can enable them to advocate and participate within their communities [1]. The ARC seeks to encourage these individuals to develop skills and hobbies that give them independence and purpose. One way that this is encouraged is through operating the ARC's greenhouse. Individuals at the Maple Ridge Center are responsible for operating the water system to irrigate the plant table daily and ensuring that the proper amount of water is distributed to the plants. This is a rewarding activity and offers the opportunity to develop useful skills in greenhouse management and maintenance. However, despite offering manual and automatic options, the current equipment of the greenhouse is not user-friendly and vulnerable if water leaks. The objective of this project is to develop an automatic controller for heating and water delivery that will be user intuitive, safe, robust, affordable, and easy to maintain. This project was started by Guo Qian Yue CPE class of 2016, kept on by Stengel Kyle CPE class of 2018, continued by Lisa Gu CPE class of 2019, followed by Larissa Umulinga CPE class of 2020, and further worked on by Aikaterini Petridou and Jason Meng class of 2021 [2] [3] [4] [5] [6].

The system that we will work on involves the optimization of wireless transmission of data to control the watering and heating system. A further part that will be worked on is a graphical user interface to make it easier to operate the system. This graphical user interface (GUI) is intended to help individuals simply press icons on the touchscreen in order to interact with the plants and their respective temperature and moisture sensors, providing a more widely-accessible option for working in the greenhouse. In this way, we choose to continue the work done by Aikaterini (Kat) Petridou and Jason Meng, Union College Class of 2021 CpE and

EE majors, respectively. Kat and Jason designed and built a wireless temperature and moisture sensor system to aid in the heating and water delivery systems in the greenhouse. To accomplish this, we will have multiple end device XBees communicating with a main controller XBee that will read in the values and display on a raspberry pi touchscreen. From here, we plan to continue to pursue how this experience can be optimized in terms of cost and operation.

Design Requirements

As this system is intended to help individuals with disabilities for a nonprofit organization, there are many requirements that are crucial to follow in order for the design to be successful. The specifications and constraints have been determined based on the performance needed, safety, energy, economics, environmental concerns, interface and accessibility, and the greenhouse's dimensions. Standards were selected after extensive research was made using the IEEE Xplore search engine.

Specifications & Constraints

The specifications for the design must comply with the requirements outlined by the Director of Schenectady ARC, Donna Vincent. As this is an ongoing project, the design must also be compliant with the current watering system in place at the greenhouse.

1. Performance

In order to integrate the design seamlessly with the rest of this project, the previous system in place had to be carefully analyzed and researched. A top level diagram representation of this previous system's design can be found in Figure 1. As seen in the diagram, the solid state relay is powered by a 120V DC power supply, which in turn powers the heating pads and water delivery system. The solid state relay behaves as a sort of switch that turns on the heating pads and watering system. This is done by having individual connections to each of the heating pads, and each of the solenoids of the watering system. There is also a controller that receives temperature and moisture readings from sensors. Using the data from the sensors, heating and watering will be done as appropriate. It is important to note that the controller will offer both manual and automatic modes. This is in compliance with the requirements given by Donna

Vincent; the system needs to both function automatically over weekends, breaks, and other leaves of absence are needed, as well as allowing for individuals to directly interact with the plants and system. Therefore, the two modes would allow the controller to directly turn on the watering and heating systems as needed when in automatic mode, and wait for user input on the touchscreen with a developed User Interface (UI) when in manual mode.

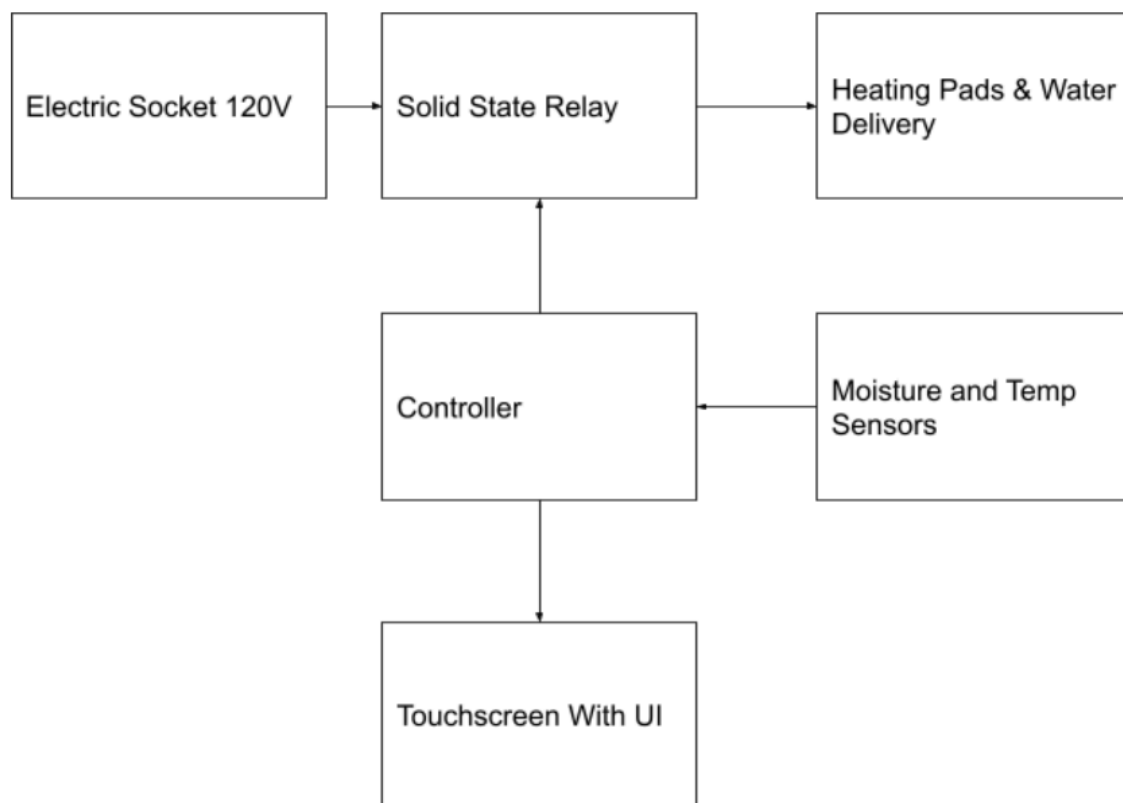


Figure 1. Top Level Diagram of system

From our research, we know that each planter bed is 8' by 11'. This large area is handled efficiently by splitting it into four quadrants, which was decided when the previous students Aikaterini Petridou and Jason Meng ('21) consulted with Donna Vincent. For the design, each

quadrant in each planter bed will get one moisture sensor and one temperature sensor. These sensors will read the levels of moisture in the soil and temperature of the soil, respectively. Health issues with plants in a greenhouse environment are often related to either moisture or temperature levels, which is why sensors that read these values were implemented into the system. In order to properly monitor the health of the plants, both the temperature and moisture levels will be measured hourly and sent to the controller. The heating pads will be distributed so there is one per planter bed. The ideal type chosen to be implemented is a 120V AC heating pad, as it is often used in greenhouses and consumes minimal power in the context of greenhouse-compliant heating pads.

Predetermined ranges for the data read from both sensors will indicate when the plants are good, will need help soon, or need immediate help. When the system is in manual mode, the ranges for the readings will be represented on a touchscreen, along with options to activate the sprinkler system and heating pads. When the system is in automatic mode, it will respond depending on what the data tells it.

2. Safety

Safety is essential when designing this project. Since we are designing a very practical system, we need to make sure that we cover all the safety concerns that the Schenectady ARC has and follow the IEEE safety standards. Upon consulting the Schenectady ARC, it is required that the hardware design should fit in its respective place without causing any harm to other activities carried out in the greenhouse.

The current system has numerous exposed wires, which could corrode from the greenhouse's climate and create a shock hazard. As the excess wiring could endanger people, both a reduction in wire usage and an effective containment plan for the system is necessary.

Effective wireless communication between the controller and sensors will considerably reduce the amount of wires used by the system. Remaining exposed wires should be waterproofed and placed in out of the way areas for minimal interactions and risks of shocks. The system should be contained in some manner to make it waterproof, as well as keep it in its respective place.

Overheating is also a concern. Upper limit temperature for the heating pads will be set in case of any malfunctions. The circuitry will be tested repeatedly to ensure that none of the components are strained enough to heat up beyond a reasonable amount.

3. Energy

As stated above, the most significant power consumption for this system is the heating pad, which uses 120V. The rest of the system needs to be designed to minimize power consumption. This is ideal for reducing power bills, keeping the system as eco friendly as possible, and helping extend the lifespan of the system. Thus, alternative power sources for other components of the system should be considered, along with how often the system needs to be on.

4. Economics

The Schenectady ARC is a non-profit organization, meaning that funding for this project is not readily available. Thus, the system has to be low-cost and simple enough for those without a strong background in electronics to operate and fix it. All necessary purchases will be done through funding from the ECBE department of Union College or an SRG grant, which is still pending. Funds from New York State Industries for the Disabled (NYSID) through the CREATE Project are currently being looked into, as funding was previously stopped due to COVID.

5. Environmental

As a system tasked with caring for plants, water is a big concern. Fine tuning and making sure the correct amount of water is being output is important for the plants which depend on the system. In terms of overall environmental concerns, wasting water should be avoided. This part of the system has been implemented in previous legs of this project, but could be improved further.

6. Interface & Accessibility

The system is intended to be used by people with various intellectual and developmental disabilities. As people have different needs based on both disabilities and personal preferences, it is difficult to predict which accommodations will be necessary for each person's specific needs. Thus, careful research for universal accessibility is essential for each aspect of the system, paired with an ability to change based on future developments. Additionally, the system should not require any special knowledge to be operated or maintained. No manuals or training should be needed for the user to control the system.

Although true for many electronic systems, it is especially important that for this system, the user does not need to understand the circuitry and each component to successfully interact with it. A Raspberry Pi 3 B+ with a touchscreen has been integrated into the system to create a more user-friendly interface and experience. Simple user interfaces will be used to both indicate sensor readings and turn on aspects of the system based on what the plants need and the decisions of the user. This subsystem will also offer the option to shift from manual to automatic care for when users are unavailable for long periods of time, such as the weekends, or are simply unable to interact with the system.

Electronic Constraints

As the system will have wired connections converting to wireless connections, research was done to get the dimensions of the greenhouse and the general layout. After consulting with Professor Hedrick and the work done by Aikaterini Petridou and Jason Meng, it has been determined that these wireless devices will need to successfully communicate with each other within a range of 50 feet [6].

Standards

The system being implemented uses an Xbee module for wirelessly transmitting data with the moisture sensor. It also uses another Xbee module which has an ethernet cable to then wirelessly transmit the data gathered to a remote location.

- I. IEEE C63.26-2015- IEEE/ANSI Standard for Compliance Testing of Transmitters Used in Licensed Radio Services

This standard addresses the proper procedure for using transmitters. The key is to repeatedly test these transmitters in order to assure that they are able to function properly.

“IEEE C63.26-2015- IEEE/ANSI Standard for Compliance Testing of Transmitters Used in Licensing Radio Services.” IEEE, January 15, 2016.

- II. IEEE C63.27-2017 American National Standard for Evaluation of Wireless Coexistence

This standard addresses the complete process (including design, testing, etc.) that is necessary

when working with a device that utilizes wireless transmitters. The key is to make sure that the wireless transmitter maintains a functional performance. In the case that the wireless devices cannot operate automatically, there would be a manual way to still get them to continue operation.

“IEEE C63.27-2017 American National Standard for Evaluation of Wireless Coexistence.”

IEEE, May 11, 2017.

III. 1-2000- IEEE Recommended Practice - General Principles for Temperature Limits in the Rating of Electrical Equipment and for the Evaluation of Electrical Insulation

This standard addresses the importance of documenting temperature limits when using electrical equipment that utilizes it. The key here is to use all temperature indexes for the equipment that is used. Considering the heating pads will be near plants, it is important to have this standard and ensure that no harm is done.

“1-2000-IEEE Recommended Practice - General Principles for Temperature Limits in the Rating of Electrical Equipment and for the Evaluation of Electrical Insulation.” IEEE, April 30, 2001.

Design Alternatives

As the project is a continuation of an already existing system pictured in Figure 1, the updated design is centered around improvement. The areas specifically being focused on are outlined in red in Figure 2.

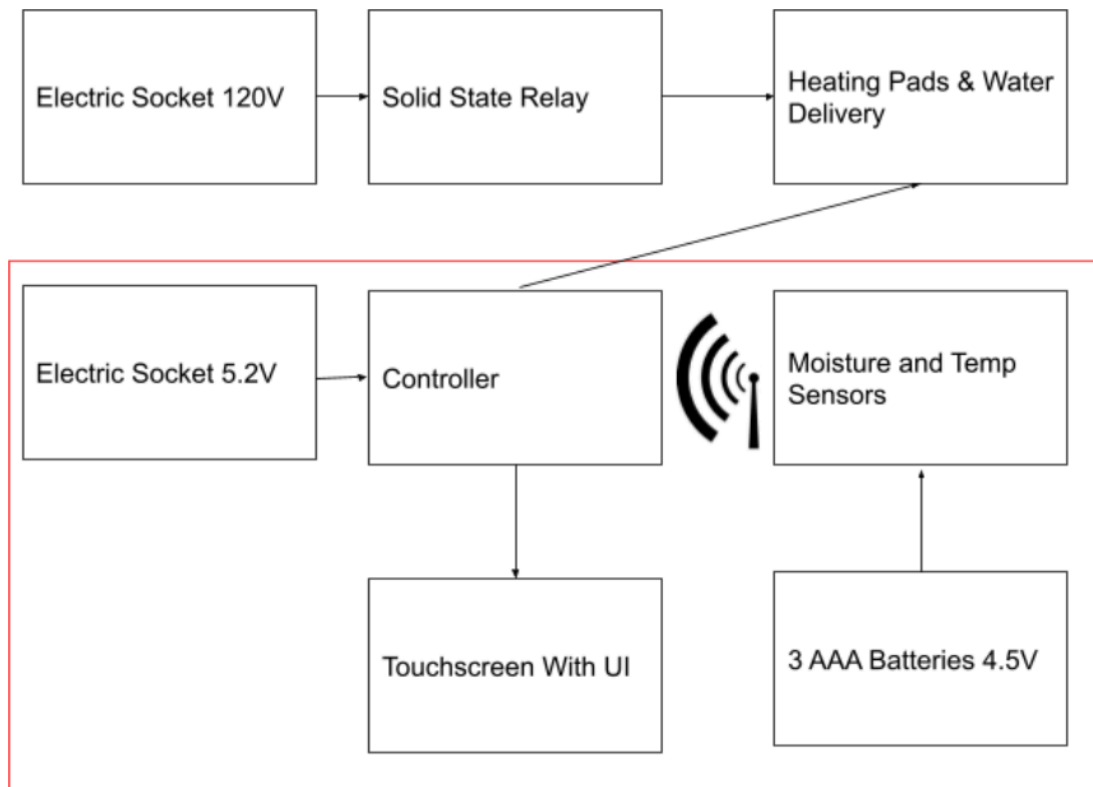


Figure 2. Updated Top Level Diagram of Current System

When comparing this updated diagram to Figure 1, it becomes clear that several significant changes have been made. As seen in Figure 2, the controller is now separately powered by 5.2 Volts via DC power. Its connection to the moisture and temperature sensors changed from an arrow to a symbol of an antenna with signal waves. This is meant to signify the replacement of a wired connection from the sensors to controller with a wireless connection. The controller is now in charge of receiving readings from the moisture and temperature sensors

through radio frequency waves. The controller is also to determine if heating or watering is needed in the heating pads and watering system.

Delving more into these changes, the wireless connection is accomplished through radio frequency (RF) modules. Radio frequency modules are essentially small radios that use transmitters and receivers to send data over varying ranges and frequencies. They are ideal for working with embedded systems, such as the Raspberry Pi. For the updated system design, there needs to be one RF module to send data from each plant bed in the greenhouse. These sensors combined with their RF module form a circuit powered by a battery pack of 3 AAA batteries. This portion of the system can be seen in the bottom right section of Figure 2.

1. Controller

The controller's purpose is to receive data from the sensors. It then needs to display the data received in some way that will be understood by the user. This data should also be processed by the controller to determine when the watering system and heating pads need to be employed. With this list of responsibilities, the controller is arguably the most important part of the entire system; it is what is making the system both accessible to the user and automated for convenience.

The jobs of the controller also indicate that a microcontroller is the type of component needed to successfully implement this essential block in the top level design. The question for the design of the controller then becomes which microcontroller should be used to achieve the best results. One microcontroller that was considered was the RedBoard Plus. This model was used for ESC-100 Exploring Engineering, a preliminary course required for all engineering students at Union College. Therefore, a huge advantage of using this microcontroller was the prior experience working with it. It is also Arduino-compatible, meaning that using the

RedBoard would come with the expansive Arduino software library. Unfortunately, the decision was not about finding the easiest method for implementation, but about optimizing the accessibility of the system for the user's experience. There were no clear advantages for accessibility for this device, and therefore other options needed to be considered.

The next microcontroller examined for this project was the PIC24HJ128GP502 with the Microstick II. This exact configuration was used in the course ECE-218 Embedded Microcontroller Projects. Specifically, this course featured labs and projects which involved processing data from temperature sensors. Therefore, the advantage for this option is also the prior experience and coursework done with the setup, as there could be a reliance on previously done work and code. However, once again, the device had no clear advantages for accessibility or power consumption. Although the PIC24 category also offered the PIC24F family, which features an eXtreme Low Power Technology, it still lacked accessibility options [7].

The final option explored was the Raspberry Pi 3 B+. This microcontroller did not come with any prior course experience or expansive software libraries to rely on when implementing it into the system. Therefore, out of the three options listed, the Raspberry Pi 3 B+ would require the most work when concerning familiarity and coding. However, as emphasized above, it did offer some remarkable features to help with accessibility. Unlike the other two microcontrollers, the Raspberry Pi 3 B+ came with the option to add a touchscreen interface. Majority of the users for this system are not expected to have a strong engineering and circuitry background, so a much more user friendly feature for the controller is preferable. In this way, even when factoring in the lack of features to reduce power consumption, the Raspberry Pi 3 B+ is the clear winner for this system.

2. Sensors

With the controller chosen, the second most important part of the system can be focused on. This part, the sensors, is crucial for the overall functionality of the system; without data readings, there is no system to be automated. The two different sensors needed to automate a system for taking care of plants are moisture and temperature. This decision was made during a consultation with Donna Reeds. Next was deciding which type of these sensors the system needed to capture these types of data.

For temperature sensors, two choices were considered. The first option was the one previously implemented into the system; the TMP36. This sensor had a number of advantages, including the fact that it was exactly the same once used in ECE-218 Embedded Microcontroller Projects, and that it was already implemented in the system. However, as this was so integral to the system, it was worth looking at more options. This other option looked at was the NOVUS RHT Climate-WM. This device offered both temperature and humidity readings, which was perfect for this system. However, the cheapest available version of this sensor was more than sixty times the price point for the TMP36. As the system needed to be cost-effective for the stakeholder, this option was not a viable one. Therefore, it was decided that continuing to use the TMP36 was the best choice.

As this expensive sensor that took care of both data readings was taken out of the running, a moisture sensor was still needed. The obvious choice was the sensor already implemented into the system along with the TMP36; the Gikfun Capacitive Soil Moisture Sensor. The advantages of the TMP36 also hold true for the Gikfun Capacitive Soil Moisture Sensor; it is both cost effective and already implemented into the system. It is worth noting that

the price for this moisture sensor is more expensive than the temperature sensor chosen, but it is still the lowest cost moisture sensor that is compatible with the system.

3. Wireless Devices

As mentioned previously, the system in place had too many wires, which can lead to numerous safety hazards. Thus, it was determined that the controller and sensors would communicate via radio frequency modules to reduce wire usage. These modules, or wireless devices, needed to be affordable, reliable, and easy to implement in the system. Various wireless devices were considered and weighed against each other to find the most suitable option for our project.

The first device that was considered was the Arduino NRF24L01 2.4 GHz RF Transceiver Module. Although the price point is low with high performance ratings, it does not support ADC inputs [8]. This is a problem for the system, as the temperature sensor and moisture sensor being used read ADC values, which must be read by the sensor and converted into data that can be interpreted. Without ADC inputs, the device would be unable to work with the system.

The second type of wireless RF module that was looked at was the Digi XBEE 3 Zigbee 3 RF Module. This module is affordable, has built in protocol stacks for 802, Digimesh and Zigbee, and has an on board processor with four 10-bit ADC inputs [9]. These protocol stacks are important because the 802 stack supports star network protocols, Digimesh supports battery-powered mesh networks, and Zigbee supports mains power mesh networks. These types of networks are options on how different wireless RF modules are able to communicate with each other. It also has a transceiver which eliminates the need to obtain another component for the system. Additionally, it can serve as a programmable microcontroller, which means the

physical complexity of the system would be reduced. In other words, separate microcontrollers with their own mains power requirements would not be needed for each XBee. A drawback of these devices would be that documentation for this specific implementation is very limited, so extensive testing and research would be needed. As the benefits of this device outweigh the drawbacks, the Digi XBEE 3 Zigbee 3 RF Module was selected as the wireless device for the design.

4. Network Topology

The network topology determines the arrangement of devices within a communication network. It is important to note that this project's communication network is not physically represented. This means that the XBee will not be moved around for the different network topology options. Instead, these formations shape the wireless connections between the devices. The type of topology chosen is an important decision. This is largely due to the variety of power consumption, security, and robustness that different topologies provide.

The first type of network topology that was considered was a mesh network. The main concept of a mesh network is that all nodes, which in this case are wireless devices, are interconnected to all other nodes. The pros of this topology include robustness, meaning that if one connection is broken, there are multiple other pathways the data can take to still arrive at the intended destination. Thus, the mesh topology would help stay consistent with the goals of making the system more user-friendly, as the network would have the ability to fix itself if something were to go wrong. The cons of this topology include high power consumption, as the network needs many connections between all nodes in order to be classified as a mesh network.

The second type explored was a star network. This style of network has one centralized node (referred to as a controller in this case) that is directly connected to all other nodes (known

as end devices for this project). Thus, each end device only has a single one-way connection. A pro for this choice is that it has a low power consumption, as there are only as many connections as end devices. It also has high security, meaning that if one end device was hacked, all other connections would still be secure. However, this also leads into the star network's major con; the topology is not robust and therefore would need outside assistance if a connection were to fail.

The final topology looked at was the bus network. The bus topology has all nodes attached to each other via a linear transmission medium, which is to say that there is a "route" which all nodes branch off from. The pros of this topology include robustness. This means that if one node fails, which in this case refers to the device breaking in some way, the rest of the system remains intact and communication is possible. Therefore, the information is still preserved and easy to intercept by any other device. Once again, this leads to the main drawback of a bus topology for this specific project, which is that all devices get all of the data all of the time, which consumes both energy and makes data analysis difficult.

After careful consideration, the star network was chosen as the best option out of the three discussed. This is because for this project, only one-way communication is needed, as seen by the arrows in Figure 2. This is because the controller needs data from the planter beds, which will each have an end device, but the end devices do not need data from the controller. It was also chosen because it consumes the least amount of power, which is a major design requirement as discussed above.

5. Sleep Cycles

The sleep cycles of the XBees played a huge role in the system's transformation from Fall term to Winter term. Originally, this was not a design consideration; the XCTU software had one sleep mode selected as a default. It was not until indepth research on the sleep modes was

being done during the end of Fall term that it was realized that other options could be considered. The XBee documentation discusses each of the eight sleep modes offered and the features of each one. Weighing the demands of the system against the features of each of the sleep modes narrowed these eight options down to two. This is because the system needed to be automatic, which meant that all sleep modes that required user interaction were automatically ruled out. The remaining two were the originally implemented asynchronous sleep cycle (Sleep Mode 4) and the synchronous cyclical sleep cycle (Sleep Mode 8).

The asynchronous sleep cycle's main advantage was also the synchronous cyclical sleep cycle's main disadvantage; the synchronous cyclical sleep cycle was only available on other firmwares, whereas the asynchronous sleep cycle was featured on the firmware already being used by the system. Therefore, the main advantage of the asynchronous option was that it would not require a complete overhaul of both the network protocol setup and sleep cycle configurations. However, the main disadvantage of this sleep mode was that it consumed a much higher amount of power than the synchronous option.

The synchronous cyclical sleep cycle was able to minimize the power consumption of the system by allowing all or most devices in the network to use low power mode, depending on the implementation. The general setup of this sleep mode is that one device is assigned the role of sleep coordinator. Its job is to send out a sync message at the beginning of each full cycle of sleeping and waking to update the devices on the new length of the sleep cycle. In this way, less power is consumed because the sleep coordinator only needs to be operating under full power when sending out the sync messages and when receiving data. This means that during sleep cycles, the sleep coordinator can enter low power mode. The method for the asynchronous option is that the end devices would wake up when they were done sleeping according to whatever sleep

length was hardcoded into the XCTU for that device and send data to the coordinator. This means that the coordinator had to be ready to receive data at any given time, as the devices could be operating at different time intervals, and therefore using much more power than with the synchronous option. This leads into the other main advantage of the asynchronous option, which is that it has the ability to change the length of the sleep cycles from the sleep coordinator. When comparing this to needing to plug each individual device into a computer with XCTU running to change the lengths, the synchronous option allowed for a much more user friendly option. These two advantages greatly outweighed the disadvantage of switching firmwares and rebuilding the sleep cycles from scratch. Therefore, during the break between Fall term and Winter term, painstaking research was done to seamlessly transition the system from the original asynchronous sleep cycle to the improved synchronous cyclical sleep cycle.

6. Firmwares

With the previously discussed decision of switching sleep cycles, the system also required a new firmware. It is worth noting that the word firmware is synonymous with the aforementioned protocol stacks offered for XBees. The default firmware for the XBees was the 802 protocol stack. The synchronous cyclical sleep cycle was only available on the other two firmwares: Digimesh and Zigbee. With the sleep decision already made, the next choice was which firmware would work best for the system.

On the official website for Digi products, a comparison was available for the Zigbee and Digimesh protocol stacks. Outside of having different networking protocols, the main difference is that the Zigbee has the capability to communicate up to two miles with a high gain antenna, as compared to the Digimesh's one mile [10]. This is a significant advantage for the Zigbee over the Digimesh, as higher ranges for communication could be useful for future evolutions of the

system. However, the major disadvantage is that the Zigbee does not offer a star network configuration in its protocol stack. This is an issue because the network topology options for the system were already analyzed, and the star network was the best choice.

Fortunately, the Digimesh protocol stack did offer the option for a star network protocol. The one drawback was redoing much of the work over Fall term for the new firmware. This work, as briefly mentioned above, was done during the break between the two terms. Outside of this one disadvantage, this stack was the only one out of the three that offered the abilities to both configure the devices in a star network and allow the devices to use the synchronous cyclical sleep cycle. Therefore, the Digimesh firmware was chosen for the system.

7. Minimizing Power

As the ARC is a nonprofit organization, keeping costs low is another top priority for the design of the updated system. The cost for the system can be split into two groups: purchasing of components, and power consumption. All of the above mentioned design choices were made with these considerations in mind. Additionally, numerous different approaches to reducing power consumption were explored to solve this problem.

One option outside of the other previously mentioned design decisions was to add rechargeable batteries to the system. Benefits of this idea include less wiring, meaning that the system would not need a direct connection to an outlet, and less power consumption overall. However, the main disadvantage of this choice is that it could decrease accessibility, as it would require the user to monitor battery life, then replace the battery if needed.

The other power consumption option explored was to continue using mains power for the entire system. Although it does not minimize power consumption, it does prioritize accessibility.

This is because the system would stay plugged in to the outlets, meaning no user engagement would be required.

Because these two options have equal weights for advantages and disadvantages, a mixture of the two was chosen as the best design choice. Specifically, the circuits with the XBees would use standard battery packs, as the circuits do not require a high amount of power and therefore would not need the batteries changed too frequently. A proposed solution for the issue of monitoring battery life would be a system similar to smoke detectors. The battery packs would have a date written on them indicating when they were last changed and a second date for when the next change should occur. Testing of the system with the battery packs would indicate when the batteries would need to be changed. The other aspects of the system would stay with mains power, meaning the controller and the 120V needed for the heating pads and watering systems. This is because purchasing batteries to handle the amount of power needed for these parts of the system would cost more than keeping the mains power implementation.

As mentioned before in this section, other decisions were made to minimize power which were discussed in other sections. In order to centralize these decisions, the design decisions made to improve power consumption are implementing battery packs, using a star network topology for XBee communication, and selecting the synchronous cyclical sleep cycle as the sleep mode.

8. Waterproofing

As the system is located in an extremely wet and humid environment, waterproofing the system is one of the top priorities. The design choice to satisfy this condition would need to be 100% effective all of the time, as any water exposure to the system could cause electric corrosion, which leads to a risk of shock hazards.

The simplest proposal for waterproofing the system would be to wrap it in plastic. This choice would be extremely easy to implement, as plastic is simple to work with. However, there is always the possibility of the plastic getting punctured, which could expose the system to water. Additionally, it would make the system less accessible for maintenance and interactions, as the plastic would need to be replaced each time.

Another solution was hydrophobic coating for the system. The exact hydrophobic coating explored was epoxy, which is a simple and affordable coating to add to the system. A major con of this solution is that the epoxy is much more permanent than a plastic coating, meaning that it would be hard to remove if the system needed maintenance, and would need to be replaced each time with new epoxy.

The final option that was considered was a sealable box. The box would encase the entire system and would be easy to open for all users. The sealant would create a waterproof barrier that would automatically work once the box is closed, and allow access when the box is opened. The main issue with this idea is that more research is needed to decide on both the box and sealant.

Once again, two out of the three choices were chosen for the design. This is because keeping the system as safe as possible is a top priority, and all elements that can optimize safety in our design are extremely important. The option of having a sealable box for the system was chosen as it keeps the circuitry easily accessible and safe. A consideration for this box is it would need a clear access point for the rechargeable battery so it could be taken out and recharged. As the XBees would be inside boxes, the sensors would need to be wired from the boxes to their respective planter beds. Thus, there would still be exposed wiring, which was the original issue. This is why the decision to also incorporate hydrophobic coating was made. The design will have

heat shrinking wire covers to protect the wires from the climate. poxy would be used to cover the temperature sensors so they can be safely placed in the wet soil. As the moisture sensors are specifically designed to measure moisture, it is unnecessary to waterproof them.

9. GUI

A huge aspect of designing this system is optimizing the accessibility of each component. It was thus proposed to have some sort of graphical user interface (GUI) for the touchscreen on the Raspberry Pi to communicate the sensor data in the most user-friendly way possible. Extensive research for the design of this GUI was done through both online sources and interviews. The interviews were conducted with individuals with varying disabilities, and those who work with people with disabilities. The idea was to get as many different perspectives and opinions as possible in order to be as inclusive as possible with the design choices. The design process consisted of a preliminary design done after strictly online research, and presenting said design to people during interviews for their feedback. Upon request, all interviewees have been left anonymous.

The preliminary design included a printed output of the code interpreting the sensor data, and a pop up window asking if the user wants to turn on the watering system or heating system. It was determined that this design would not work for a multitude of reasons. The first point made by interviewees was to reduce the usage of language to accommodate those who either did not speak English or had limited or no language processing skills. Instead, communication of information could be done with universally recognized images. For example, the watering system could be represented by a button with a watering can, and the heating system could be represented by a thermometer. Additional feedback indicated that the buttons should be as large as the screen should allow, in order to accommodate those with motor control issues or visual

impairments. This meant that the user interactive portion went from a pop up window to two buttons, one for each subsystem, taking up the entire screen. What remained was the issue of how to communicate sensor data without using language or blocking the two buttons in any way. Consultation with those who work with people with disabilities revealed that a traffic color code system is often used for this type of problem. Specifically, for this system, the buttons would change color based on the readings for their related sensor readings. Ranges upon consultation with Donna Reeds would indicate what sensor readings should be red, for dangerous readings, which should be yellow, or warning levels of readings, and which should be green for safe readings. For example, if the sensor data indicated low moisture readings, the watering can button would turn red.

Design

1. Overview

As Figure 1 shows, the automated watering and heating system for the greenhouse has multiple parts. Our focus is on improving the subsystems outlined in red as was seen in Figure 2.

The controller is an XBee 3 configured to be a coordinator so that it will be able to receive data from the moisture and temperature sensors. To send data, the remaining XBee 3s will be configured to be end devices. The sensors would then connect to the end device. Figure 3 below shows an overview of how this will work.

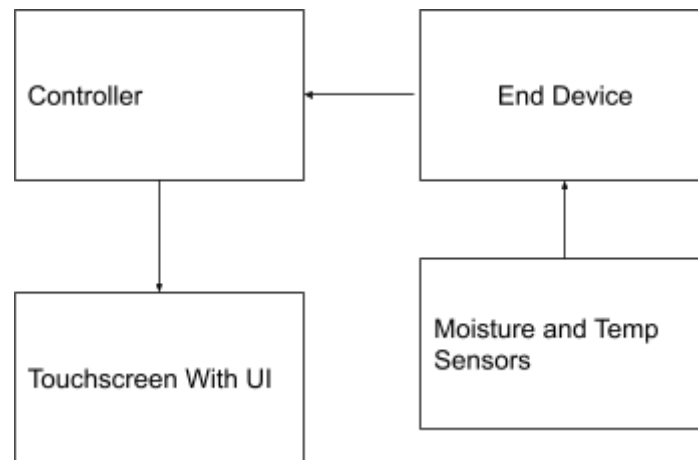


Figure 3. Subsystem with further detail about implementation

Before going into how exactly the moisture and temperature sensor would connect to an end device, an overview of how an end device will connect to a coordinator will also be needed.

Figure 4 below depicts how this would work.

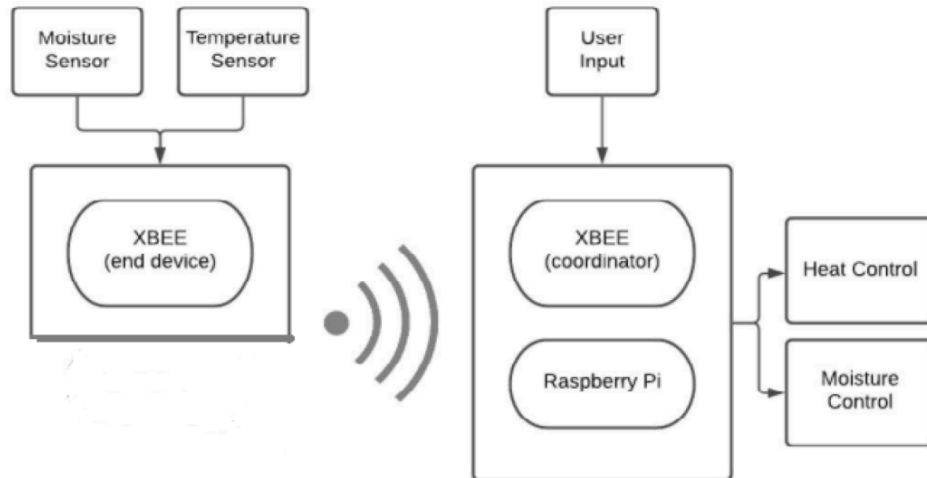


Figure 4. Diagram of Controller and End Device system

As Figure 3 had shown, a moisture and temperature sensor would connect to the end device. This end device will communicate with the coordinator through RF communication and be able to send the data that it reads from the sensors. The coordinator would work by being connected to a microcontroller, which will be a Raspberry Pi 3 B+ in our design. This Raspberry Pi is supporting the code that allows this coordinator Xbee to read from the end devices. Additionally, it provides the necessary computational power to run a touchscreen with the UI that will be implemented. This coordinator hub will have user input through the touchscreen and will have the functionality to control the heat and moisture control systems through a manual or automatic mode. This portion is independent of our current project work.

End Devices

Through the XCTU application, the Digimesh protocol stack was selected, as discussed in the Design Alternatives section. The transition from the original 802 protocol stack to Digimesh proved difficult not just from the extra work needed, but also for determining how to construct a star network topology with Digimesh. With the 802 firmware, there were options with the XCTU software to manually select which devices were end devices and which was the

controller. For Digimesh, the majority of the break between Fall term and Winter term was spent researching the details of the new firmware. Eventually, after consulting a section on addressing in Bell's book [9], the correct method to implement a star network was found. The coordinator XBee needed to be identified, and the 16-bit MAC address of said XBee split between each end device's destination address high and low. Therefore, the end devices are selected by simply using the top 8 bits of the MAC address for the DH (destination address high) and the bottom 8 bits for the DL (destination address low). Once the end devices are configured, it is established that the device with the MAC address is the coordinator, and only this device can be the coordinator. It would not be possible for other coordinators to interrupt the network as its MAC address would have to be configured on the end devices. Appendix Figure 14 shows a screenshot of where the MAC address can be found through the XCTU.

These controller and end device assignments create the star network topology chosen for this system's wireless communication. This topology also reflects the sleep mode configuration; the coordinator is also the sleep coordinator and the end devices are the same for both. Each end device will be placed in a plant bed, where its respective circuit will collect data via sensors. These readings will be done through the operations of Micropython code, which will be discussed in the following sections. When the end devices wake up from sleep, they send the data read from the sensors to the coordinator.

Temperature Sensor

The circuitry had to be changed from the previous team's version of the system as the temperature readings were not correct. The design was initially trusted to have been correct until testing resulted in temperature values that were not correct in Celsius, Fahrenheit or Kelvin. The values being read were raw ADC values, so we had to convert them to Celsius and then

Fahrenheit for better interpretation. However, the values were still incorrect after applying these conversions. Therefore a thorough analysis of both the conversion calculations and circuit were completed. Eventually, this circuit design was dropped in favor of one that was found through research. This updated circuit design resulted in temperature readings that were relatively accurate. Figure 5 below shows the new circuit design.

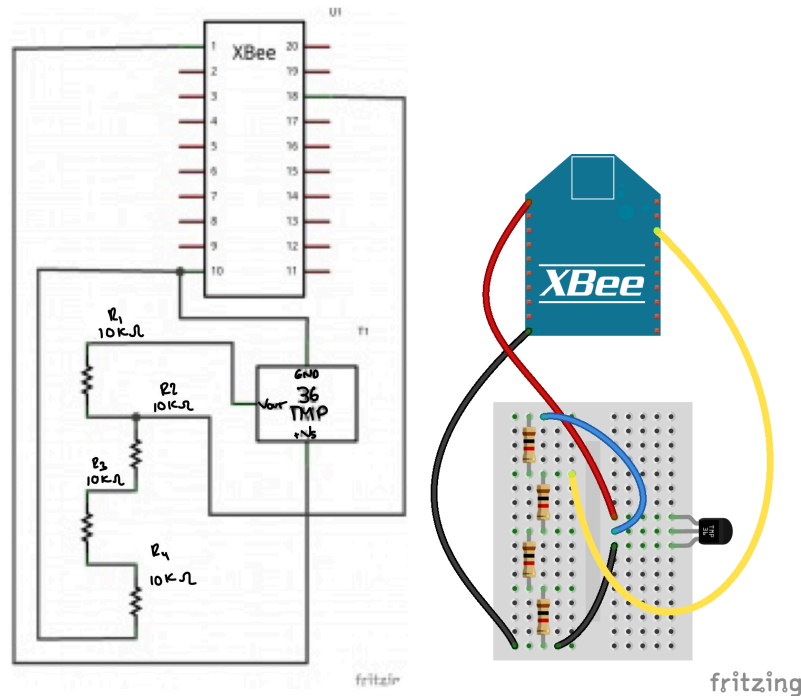


Figure 5. Circuit diagrams of end device connected to TMP36 sensor

The circuit requires the following: 4 10kΩ resistors in series, connection to ground, voltage coming from the XBee, TMP36 sensor, and wire connections to the pin of the TMP36 that outputs the temperature readings.

Four 10kΩ resistors are needed because they are going to scale the input down from 3.3V (output from XBee) to the 2.5V maximum of the XBee Wi-Fi's ADC input pin (pin 18 on the XBee). Something to note about pin 18 on the XBee is that it is configurable via XCTU and its default setting has no purpose. Appendix Figure 10 shows that DI01/AD1/SPI_nATIN is set as

ADC. This has to be configured on XCTU to operate as ADC. If this isn't configured, errors will arise and no readings will occur when running the code.

Now that the circuit is functional and the TMP36 can output values, these values need to be converted from ADC to an understandable value. The XBee has 4 10-bit ADC pins and we are making use of this feature.

Since a pin supports ADC values of up to 10 bits, this means that the readings can vary from 0-1023. This is calculated by $2^{10} = 1024$, meaning there are 1024 possible values that can be read from the ADC pin. Furthermore, the TMP36 is outputting a value based on the voltage that it is receiving which is 3.3V which comes from the XBee. Thus we can calculate an analog voltage from the ADC values by the following equation:

$$analog_V = adc_value * (3.3 / 1024)$$

Now that the analog voltage has been obtained, we can use this value to convert to degrees Celsius. In Appendix Figure 11, the graph shows TMP36 Output Voltage vs Temperature (°C). From this table we can determine a relationship between the analog voltage and celsius. Based on the documentation for the TMP36 and the table, we know that there is an offset voltage of .5V. Based on the slope, we found that it was .01 V/C. This was calculated by starting at 0°C and heading towards 50°C because the analog voltage was 1V at that point. Thus $\frac{0.5V}{50C} \cdot 0.01V/C$. From this we could make the following equation from the relationships found:

$$analog_V = 0.01t + .5 \text{ where } t \text{ is any temperature in } ^\circ C.$$

The above equation can be generalized into its units for better understanding and will help with deriving the equation for getting a temperature in Celcius.

$$V = \frac{V}{C} C + 0.5$$

$$\frac{C}{V} (V - 0.5) = (\frac{V}{C} C) \frac{C}{V}$$

$$C = \frac{C}{V} (V - 0.5)$$

$$C = \frac{50C}{0.5V} (analog_V - 0.5)$$

$$celcius = (analog_V - .5) * 100$$

Now that the formulas' derivation have been explained, the implementation needs to be discussed. Throughout ECE 498, the XBees relied on an external microcontroller to run the necessary code to convert the ADC values to an understandable value by a user. The external microcontroller was no longer used in favor of loading micropython code onto the ROM of the XBee. By doing this, it was found that in micropython, ADC operates from 0-4096, where $\log(4096) = 12\text{bits}$. To accommodate this range, the analog_V equation was changed to:

$$analog_V = adc_value * (3.3 / 4096)$$

None of the other equations needed any adjustment, so they were left as is.

Thus, line 34 is now justified from our code found in Appendix Figure 1. Line 35 shows the celsius equation and line 36 shows the conversion to fahrenheit, but that conversion is trivial and doesn't require an explanation.

Moisture Sensor

The moisture sensor will be left as it was implemented by the previous team's system. Connecting the sensor to the XBee posed no problems, as it doesn't require an involved circuit design like the TMP36 to output the correct ADC values. The moisture sensor requires a 3.3 VDC, a ground connection, and the analog output wire would connect to one of the ADC input pins on the end device XBee.

There are no calculations required to get an appropriate ADC value; the ADC pin it is connected to can just simply be read. When first reading values, the moisture sensor was tested in very wet conditions to see if the ADC values changed, and then placing it back in the dry environment. Through this, it was found that the moisture sensor was not calibrated correctly, and so the appropriate calculations had to be done to get accurate readings.

To calibrate, we tested what the moisture sensor outputs when it is fully exposed to water. This was done across multiple moisture sensors to both determine the average reading, and to make sure that this calibration could be universal for all the moisture sensors. This value was found to be 2370. When the moisture sensor was in completely dry conditions, the value was 4095. This calibration also helped show the moisture sensor readings in a percentage range, which is more understandable when measuring moisture levels.. So the new max output value would be 100, representing complete water moisture, and the new minimum output value would be 0, representing no moisture. This was not hard to convert, as this was something previously achieved with the aforementioned PIC24H/Microstick II used in ECE-218 Embedded Microcontroller Projects. The equation is as follows:

$$\frac{(moistureADC\ value - OldMin) * NewRange}{OldRange} = NewValue$$

where:

OldRange = OldMax - OldMin

NewRange = NewMax - NewMin

OldRange and NewRange simply calculate the difference between the desired ADC output range and the new readable range. By using this equation, any moisture ADC output value can be

converted to tell if the moisture is between 0-100%. Lines 11-19 and line 30 in Appendix Figure 1 shows how it is implemented on the XBee end devices.

Sleeping

To minimize power consumption, the previously discussed synchronous cyclical sleep cycle was implemented for the XBees. In our implementation, we are going to have the end devices sleep for thirty minutes and wake up for one minute. Figure 6 below shows the overview of how an end device operates with this sleep cycle.

The end device XBee sleep configuration is done through the XCTU software and the configuration can be seen in the Appendix Figure 10. For this to work, both the coordinator and the end device need to be able to communicate with each other. This will be discussed when going in depth on the Coordinator device.

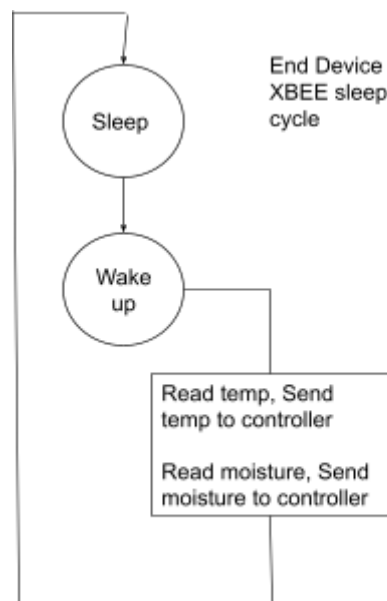


Figure 6. Diagram of End Device activity cycle

Coordinator Device

As Figure 4 showed, the coordinator receives the sensor data from all the end devices. It is also connected to a Raspberry Pi 3 B+ which makes it possible to have the touchscreen. Additionally, the coordinator will be able to control the heating and water delivery systems.

XBee Communication

To make communication between devices possible, both the end devices and coordinator needed to be configured via the XCTU software. The most important settings were under the Networking, Addressing, and API Configuration sections of the XCTU.

Under Networking, the PAN ID needed to be configured such that both end devices and coordinator have the same matching ID number. This enables a network and by having all the devices have the same unique PAN ID, they are set to only operate with each other.

Addressing was the most difficult section to work with. The previous team had an implementation of this working, but only between a single end device and a coordinator, which wasn't optimal. Additionally, the previous team's implementation was for the 802 protocol stack. The Digimesh firmware's implementation is a different process. As discussed in the End Device section, the controller is determined by splitting the 16-bit MAC address between each end device's DH and DL.

Before configuring the XBee end devices to operate without an external microcontroller, the API configuration was set to API Mode without Escapes(1). This means that there are no escaped characters in the transmission of data. By using API mode, data frames include the source of the message so it is easy to identify where data is coming from, which is useful as there are multiple end devices communicating with the coordinator.

Now that the XBee has moved to run code from its ROM, the API configuration changed such that the API Enable was set to MicroPython REPL (4). Now the data frames are configured to be dealt with by the code loaded onto the device.

Under UART Interface, UART Baud Rate for the End Devices were configured to be 115200. This is changed from the default value of 9600. The reason for this is because the XBees are now going to operate with Micropython code running and transmitting data. This baud rate helps to prevent data loss.

Appendix Figure 2 shows how the coordinator receives messages from the End Devices. Appendix Figure 1 line 56 shows that a message with the necessary information transmits to the coordinator by making use of the coordinator's MAC address. The message is transmitted, and is received by the coordinator as a dictionary object. Appendix Figure 2 shows how it takes the dictionary and extracts the necessary information to print to the display for the user to read temperature and moisture. On top of that, the message also tells the user which End Device the message is coming from.

GUI

As the ARC is an organization to help individuals with disabilities, accessibility improvement is essential. The implementation of graphical user interfaces on the Raspberry Pi 7" touchscreen is meant to improve interactions between users and the system. As outlined before, thorough research involving those with disabilities was conducted. Additionally, consultations with professionals who are trained to help those with disabilities was conducted to assure that the design is as universally accessible as possible. It is worth noting that this design is subject to change upon consultation with the ARC, as it is possible that certain individual needs are still not being met. Open communication with stakeholders is essential for this aspect of the project.

The layout of the design is two large buttons displayed on the Raspberry Pi 7" touchscreen for the users to interact with. One button will be used to monitor moisture levels via moisture sensor readings, and will turn on the water system when pushed. The other button will be used to monitor temperature levels via temperature sensor readings, and will turn on the heating pads when pushed. The design will also have the ability to switch between automatic and manual modes so the system will be capable of taking care of itself when users are unavailable. A prototype of the design is shown in Figure 7 below.

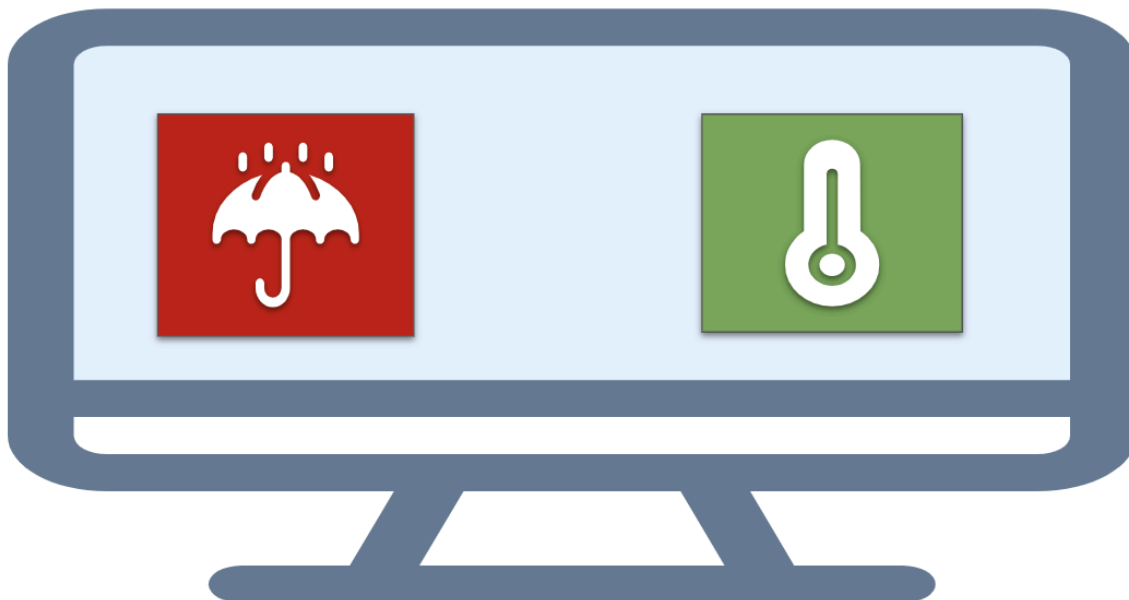


Figure 7: Basic Representation of GUI Display Design

Outside of the design prototype above and the research that was needed to develop it, code in C using GTK has been drafted. It is considered drafted because it has not yet been run on the Raspberry Pi 7" touchscreen and therefore may need to be changed to fit the dimensions of the screen and debugged for implementation. The code also does not yet connect to the other portions of the system, meaning that pressing the buttons does not do anything and that the color of the buttons do not change.

Parts

The parts that will be used in the portion of our project will be:

- Digi Xbee3
- Raspberry Pi 3 B+
- Raspberry Pi 7" Touch Screen LCD
- Capacitor Moisture Sensor
- TMP36 Sensor

Final Results and Design Evaluation

Results

Certain aspects of this project have been only partially developed due to both time constraints and prioritization of the XBee communication and sleep cycles. It is important to note that the GUI is not implemented at all. Ideally, it would display on the Raspberry Pi 7” touchscreen constantly whenever the system is put in manual mode. No display or user interfaces would be available when the system is in automatic mode. Therefore some sort of if statement is needed to launch this portion of the code when the system is in manual mode. Additionally, the Raspberry Pi needs to have GTK installed in order to run the code. Unfortunately due to timing and prioritization, all of the aspects of this portion of the system outside of the design and drafted code are being left to future teams for this project.

Testing

Testing our design was a multi-step process. It was done modularly, meaning the system was broken down into small manageable portions and each of these portions was tested.

As was discussed in this project’s ECE-498 paper, the TMP36 was extensively tested. The tests included making sure the output values were correct by measuring the voltage and comparing it to what was being calculated through our code. Part of this also included making sure that the epoxy applied to provide the waterproofing didn’t affect the readings. This made integrating the sensor into our project much easier. The only problem that was encountered was when it was discovered that ADC runs on 12 bits instead of 10 on MicroPython. As was discussed in the Design, Temperature Sensor section, the formula only needed a simple change to support the 12-bit ADC.

Testing the Moisture sensor has been discussed in the Design, Moisture Sensor section. The most important part was that the calibration had to be done to get good output values. Once the calibration had been completed, the moisture sensor was exposed to different conditions to test the output values. This was successful, but it was only tested on a small amount of sensors. As was explained previously, the calibration was done using the average of the original output values, and so this gives us the confidence that the conversion will be functional across all the moisture sensors. Something to note from this process is that we were initially working with a defective moisture sensor. It took thorough troubleshooting of the entire circuit and code before narrowing it down to the hardware of the moisture sensor. The reason it was overlooked is because when using a multimeter, there was an output voltage which was interpreted to mean that the sensor worked correctly. This ended up not being the case, and it was found that the sensor was shorting out the entire circuit. The sensor was subsequently discarded and all other moisture sensors were tested with the circuit to prevent this same display of behavior.

As was discussed in the Design section, establishing communication between the controller and end devices was not an easy task. After changing the network protocol to DigiMesh, getting the correct configuration settings was important to make the network possible. This has been discussed in the End Devices, XBee Communication, and Microcontroller subsections of the Design section. Similar to when the XBees were using the 802 protocol, simple testing was done by using the XCTU to send small packets of data to confirm that the controller was receiving anything. This initially took copious amounts of debugging because of incorrect configurations, but upon the resolution it was then possible to move on to running code without an external microcontroller.

Learning how to load MicroPython was a much different process that was found through [9] and [11]. Once we understood how to load the code onto the XBee's ROM, it was found that our code could not compile. This was because we were initially running the code, with the aid of a Raspberry Pi, that made use of libraries that the XBee didn't have. To avoid any further errors, the initial code was modified to be compilable on the XBee's ROM. The code that had to be modified mostly had to do with how ADC values were being read, and so it wasn't a drastic change. The method to send messages to the coordinator also had to change because it made use of the transmit function which requires a 64 bit address to identify the recipient. This 64 bit address was found to be the coordinator's MAC address. Once this was implemented, messages were being received by the coordinator as references to objects. To remedy this, the read code was adjusted such that the object was converted to a dictionary, which had many different values. This was later adjusted so that it would only print the message displaying the current temperature and moisture value. To be able to differentiate where the messages were coming from, the message also shows which End Device it is coming from through a number naming system.

When testing if the messages were being delivered, there were two methods to accomplish this. The XBee could fully run standalone by being connected to a power source or using XCTU. When using XCTU, more useful information was printed so this was a go-to. By connecting it to the computer via USB, a power supply was provided, so this was initially the easiest method to test. The XCTU provides a MicroPython terminal which is also where the code could be loaded onto. If the code had already been loaded, and the XBee has been configured to autostart the MicroPython it has loaded onto it, the code would start running and printing the information we had coded to display. Appendix Figure 20 shows what this looks like.

Getting the sleep cycles to work between the controller and end devices was successfully implemented. This was a very crucial aspect of the project, and this feature became more attainable once the communication between the coordinator and end devices was successful. Something that helped accomplish this was that the coordinator's sleep mode was set to Synchronous Sleep Support (Sleep Mode 7). Digi's resources was helpful in setting up this configuration and was the reason for choosing Sleep Mode 7 as it mentioned that "Sleep support nodes are especially useful when used as preferred sleep coordinator nodes and as aids in adding new modules to a sleeping network" [12] Once this was implemented, testing if the sleeping was working according to the set sleep and wake time was done. This was very hard to do because it was hard to tell what was happening. We were also using a short amount of time for testing purposes since we didn't want to wait a long time. This led to errors explained later on. To remedy the problem of not knowing if the XBee was awake or not, an LED was introduced. This LED indicated that the XBee was sleeping by turning on and vice versa for when the XBee was awake. The LED made testing the sleep and wake amounts much easier and confirmed that the sleep cycle was working accordingly.

As was mentioned above, using shortened sleep times brought unforeseen challenges. When the XBee wanted to be changed to another sleep or awake time, the XBee couldn't be registered by the XCTU. Since it was falling asleep fast, it was either not being detected or the settings couldn't be accessed. It was very hard to remove this problem, because to change the settings, the only method was through the XCTU. There was even the case where the XBee would be detected as corrupted, but it actually wasn't. The following solution is something that was found through many efforts and was not found online. Using the coordinator, you scan for devices on the network. They do not have to be connected to the XCTU at all. Since the

connection already existed between the coordinator and the end device, this was possible. Once the end device was detected, it would appear under the coordinator. Appendix Figure 21 shows how this would look. Now it was possible to access the End Device settings and it was also possible to remotely edit them. The sleep mode would be disabled, and the problem was resolved. The XBee with the problem could now be accessed through the XCTU.

When using the XBee as a standalone, meaning there was no external microcontroller, it needed a power source. The power source options used were a USB connection to the computer, connecting to a wall outlet via USB, and a 4.5V AAA battery pack. As discussed throughout, wire usage was to be reduced whenever possible in order to reduce shock hazards. The battery pack was thus chosen as the source of power for the XBee. Appendix Figure 18 shows how this was implemented.

Discussion, Conclusions and Recommendations

This was the sixth installment to this project that seeks to automate the Schenectady ARC's greenhouse watering and heating systems. As such, the project is one step closer to being automated. The wired connections between the sensors in each plant bed and the controller were successfully replaced with XBees. These XBees were set up with the sensors in a circuit and configured in a star network topology. They also operate on a synchronous cyclical sleep cycle. All in all, the power consumption has been reduced for the battery pack powering the end device circuits from approximately twelve hours per battery life to thirty days. This estimate is conservatively estimated, and not yet tested on site.

This point also leads to the largest issue with this portion of the project; nothing new for the system has been implemented on site due to COVID-19. Therefore, debugging for onsite implementation will be necessary for the next installment. Additionally, the GUI design has been developed but not fully debugged or integrated into the system. A huge job for the next team will be to optimize this code that has been developed, and set up some form of multi-threading that prevents interleavings between this and the sending and receiving of information within the system. All details necessary for how to continue the project are included in the legacy letter and are to be presented to the next team during their ECE-497 course.

References

- [1] “Making A Difference Every Day: (518) 372-1160.” Schenectady ARC, 17 Feb. 2021, www.arcschenectady.org/.
- [2] Guo, Qian Yue, Automated Watering System for Greenhouse. 2016. https://cpb-us-w2.wpmucdn.com/muse.union.edu/dist/3/343/files/2015/11/498Report_Guoq.pdf
- [3] Stengel, Kyle, Automated Greenhouse Heating System, 2018.
- [4] Gu, Lisa, Automated Greenhouse for Schenectady ARC, 2019. <https://digitalworks.union.edu/cgi/viewcontent.cgi?article=3309&context=theses>
- [5] Umulinga, Larissa, Automated Greenhouse Watering System for the Schenectady ARC, 2020. https://cpbusw2.wpmucdn.com/muse.union.edu/dist/c/602/files/2020/03/LarissaUmulinga_ECE4_99DesignReport-2.pdf
- [6] Petridou, Aikaterini and Meng, Jason, Automated Greenhouse Watering and Heating System for the Schenectady ARC, 2021
- [7] “PIC24F Mcus - 16 MIPS - Microchip Technology.” *PIC24F Microcontrollers*, <https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/16-bit-mcus/pic24f-mcus-16-mips>.
- [8] *Single Chip 2.4 GHz Transceiver NRF24L01*. 2007, https://www.sparkfun.com/datasheets/Components/nRF24L01_prelim_prod_spec_1_2.pdf.
- [9] Bell, Charles. *Beginning Sensor Networks with XBee, Raspberry Pi, and Arduino: Sensing the World with Python and MicroPython*. Apress, 2020.
- [10] “Compare Products.” *Compare Products | Digi International*, 2022, <https://www.digi.com/products-compare/56578/56577/56576>.
- [11] Arwensadler. “How to Download MicroPython Code onto Xbee 3.” *Instructables*, Instructables, 25 Sept. 2019, <https://www.instructables.com/How-to-Download-MicroPython-Code-onto-XBee-3/>.
- [12] “Synchronous Sleep.” *Digi Resources*, Digi, https://www.digi.com/resources/documentation/Digidocs/90001496/containers/cont_synchronous_sleep.htm?TocPath=Low+power+and+battery+life%7CSynchronous+sleep%7C_____0.

Appendices

Send Temperature and Moisture

```
1. from machine import ADC
2. from time import sleep
3. import xbee
4. import time
5.
6. # Target address to send data. This is the Coordinator's MAC
   address
7. TARGET_64BIT_ADDR = b'\x00\x13\xA2\x00\x41\xD5\x87\x5A'
8. wait_time = 3 # seconds between measurements
9.
10.
11. def remapMoisture(moistureADCval):
12.     OldMax = 2370
13.     OldMin = 4095
14.     NewMax = 100
15.     NewMin = 0
16.     OldRange = (OldMax - OldMin)
17.     NewRange = (NewMax - NewMin)
18.     NewValue = ((moistureADCval - OldMin) * NewRange) /
        OldRange) + NewMin
19.     return NewValue
20.
21.
22. while True:
23.     # Read temp value & print to debug
24.     temp_pin = ADC("D1")
25.     adc_val = temp_pin.read() # equal to adc_value
26.     print("Temp ADC pin reading: %d" % adc_val)
27.
28.     # Read moisture value & print to debug
29.     moisture_pin = ADC("D3")
30.     moisture_val = moisture_pin.read()
31.     print("Moisture ADC pin reading: %d" % moisture_val)
32.
33.     # Converting temp to proper units
34.     analog_V = adc_val * (3.3 / 4096) # 3.3 V output
35.     temp_c = (float(analog_V) - .5) * 100
36.     temp_f = (temp_c * (9.0 / 5.0)) + 32.0
37.
38.     print("Temperature: %.2f Celcius" % temp_c)
39.
```

```

40.     print("Temperature: %.2f Fahrenheit" % temp_f)
41.
42.     # Converting moistureADC to analog
43.     updated_moist = remapMoisture(moisture_val)
44.     print("Moisture Analog value: ", updated_moist, "%")
45.
46.     # Send data to coordinator
47.     temp_f_str = "{:.2f}".format(temp_f)
48.     moisture_str = "{:.2f}".format(updated_moist)
49.     message_array = ["End Device 1, Temperature (F): ",
        temp_f_str, " degrees, Moisture: ", moisture_str, " %"]
50.     message = "".join(message_array)
51.     print("Sending: %s" % message)
52.
53.     try:
54.         xbee.transmit(TARGET_64BIT_ADDR, message)
55.         print("Data sent successfully")
56.     except Exception as e:
57.         print("Transmit failure: %s" % str(e))
58.
59.     # Wait between cycles
60.     sleep(wait_time)

```

Appendix Figure 1. Send Temperature and Moisture Code

Read

```

1. from digi.xbee.devices import XBeeDevice
2. import time
3. device = XBeeDevice('/dev/ttyUSB0', 9600)
4. device.open()
5. while 1:
6.     xbee_message = device.read_data()
7.     if xbee_message != None:
8.         xbee_msg_dict = xbee_message.to_dict() # Turns received
            message into dict
9.         msg_bytearray = xbee_msg_dict['Data: '] # We just want data
10.        msg_string = msg_bytearray.decode() # Turns message to
            string and so that it doesn't have b in front
11.        print(msg_string)
12.

```

Appendix Figure 2. Read Code

Coordinator XCTU Configuration Screenshots using DigiMesh

Product family: XB3-24 **Function set:** Digi XBee3 DigiMesh 2.4 TH **Firmware version:** 300D

▼ **Networking**
Parameters which affect the DigiMesh network

CH Channel	C	
ID Network PAN ID	3332	
CE Device Role	Standard Router [0]	
C8 Compatibility Options	0	Bitfield

▼ **Discovery Options**
Configuration of network discovery options

NI Node Identifier	sleep_coord	
DD Device Type Identifier	140000	
NT Network Discovery Back-off	82	* 100 ms
N? Network Discovery Timeout	3A06	
NO Network Discovery Options	0	Bitfield

▼ **Addressing**
Source and destination addressing settings

SH Serial Number High	13A200	
SL Serial Number Low	41D5875A	
DH Destination Address High	0	
DL Destination Address Low	0	
RR Unicast Retries	A	Retries
MT Broadcast Multi-Transmits	3	
TO Transmit Options	C0	Bitfield
NP Maximum Packet Payload Length	49	

Appendix Figure 3. Coordinator Networking and Addressing XCTU Configuration

▼ **Sleep Settings**
Configure low power options

SM Sleep Mode	Synchronous Cyclic Sleep [8]	
SP Sleep Time	C8	x 10 ms
ST Wake Time	EA60	x 1 ms
SN Number of Cyclic Sleep Periods	1	
WH Wake Host Delay	0	x 1 ms
SO Sleep Options	1	Bitfield

▼ **Diagnostics - Sync Sleep Status/Timing**
Sleep diagnostics and timing for synchronously sleeping devices










SS Synchronous Sleep status	600F	
OS Operating Sleep Time	C8	
OW Operating Wake Time	EA60	
MS Missed Sync Messages	0	
SQ Missed Sleep Sync Count	0	

Appendix Figure 4. Coordinator Sleep XCTU Configuration












End Device XCTU Configuration Screenshots using DigiMesh

Product family: XB3-24 **Function set:** Digi XBee3 DigiMesh 2.4 TH **Firmware version:** 300D

▼ **Networking**
Parameters which affect the DigiMesh network

i	CH Channel	<input type="text" value="C"/>		 
i	ID Network PAN ID	<input type="text" value="3332"/>		 
i	CE Device Role	<input type="text" value="Standard Router [0]"/>	▼	 
i	C8 Compatibility Options	<input type="text" value="0"/>	Bitfield 	 


▼ **Discovery Options**
Configuration of network discovery options

i	NI Node Identifier	<input type="text" value="end3"/>		 
i	DD Device Type Identifier	<input type="text" value="140000"/>		 
i	NT Network Discovery Back-off	<input type="text" value="82"/>	* 100 ms 	 
i	N? Network Discovery Timeout	<input type="text" value="3A06"/>		
i	NO Network Discovery Options	<input type="text" value="0"/>	Bitfield 	 

Appendix Figure 6. End Device Networking Configuration

▼ Addressing

Source and destination addressing settings

i	SH Serial Number High	13A200	
i	SL Serial Number Low	41C8FB14	
i	DH Destination Address High	<input type="text" value="13A200"/>	
i	DL Destination Address Low	<input type="text" value="41D5875A"/>	
i	RR Unicast Retries	<input type="text" value="A"/> Retries	
i	MT Broadcast Multi-Transmits	<input type="text" value="3"/>	
i	TO Transmit Options	<input type="text" value="C0"/> Bitfield	
i	NP Maximum Packet Payload Length	49	



▼ DigiMesh Configuration

Parameters which affect outgoing transmissions in a DigiMesh network

i	NH Network Hops	<input type="text" value="7"/> Hops	
i	BH Broadcast Hops	<input type="text" value="0"/>	
i	MR Mesh Unicast Retries	<input type="text" value="1"/> Mesh Unicast Retries	
i	NN Network Delay Slots	<input type="text" value="3"/> Network Delay Slots	
i	SE Source Endpoint	<input type="text" value="E8"/>	
i	DE Destination Endpoint	<input type="text" value="E8"/>	
i	CI Cluster ID	<input type="text" value="11"/>	



Appendix Figure 7. End Device Addressing and DigiMesh XCTU Configuration

▼ **Diagnostics - Addressing Timeouts**
Transmission timeout values






i	%H MAC Unicast One Hop Time	1E	
i	%B MAC Broadcast One Hop Time	1B	

▼ **Security**
Change security parameters














i	EE AES Encryption Enable	Disabled [0]		
i	KY AES Encryption Key	<input type="text"/>		
i	DM Disable Device Functionality	0 <input type="text"/> Bitfield 		
i	US OTA Update Server	0 <input type="text"/>		
i	SA Secure Access Options	0 <input type="text"/> Bitfield 		
i	Secure Session Authentication	<input type="button" value="Configure"/>		

Appendix Figure 8. End Device Diagnostics and Security XCTU Configuration

▼ **RF Interfacing**
Change RF interface options for 2.4 GHz DigiMesh traffic

i	PL TX Power Level	Highest [4]		
i	PP Output power in dBm	8		
i	CA CCA Threshold	0 <input type="text"/> -dBm		

▼ **MAC Diagnostics**
Media Access Control diagnostic information

i	DB Last Packet RSSI	0		
i	EA MAC ACK Failure Count	0 <input type="text"/>		
i	EC CCA Failure Count	0 <input type="text"/>		
i	BC Bytes Transmitted	698 <input type="text"/>		
i	GD Good Packets Received	0 <input type="text"/>		
i	TR Transmission Failure Count	0 <input type="text"/>		
i	UA Unicasts Attempted Count	0 <input type="text"/>		

Appendix Figure 9. End Device RF Interfacing XCTU Configuration

▼ Sleep Settings

Configure low power options

i	SM Sleep Mode	No Sleep [0]					
i	SP Sleep Time	3A98	x 10 ms				
i	ST Wake Time	4E20	x 1 ms				
i	SN Number of Cyclic Sleep Periods	1					
i	WH Wake Host Delay	0	x 1 ms				
i	SO Sleep Options	0	Bitfield				

▼ Diagnostics - Sync Sleep Status/Timing

Sleep diagnostics and timing for synchronously sleeping devices

i	SS Synchronous Sleep status	2041				
i	OS Operating Sleep Time	3A98				
i	OW Operating Wake Time	4E20				
i	MS Missed Sync Messages	1				
i	SQ Missed Sleep Sync Count	1				

▼ MicroPython Options

Change MicroPython behavior

i	PS MicroPython Auto Start	Enabled [1]			
---	---------------------------	-------------	--	--	--

Appendix Figure 10. End Device Sleep Settings and MicroPython Options XCTU Configuration

API Configuration
Change API mode configuration

AP API Enable	MicroPython REPL [4]	
AO API Output Mode	API Rx Indicator - 0x90 [0]	
AZ Extended API Options	0	Bitfield

UART Interface
Configuration options for UART

BD UART Baud Rate	115200 [7]	
NB UART Parity	No Parity [0]	
SB UART Stop Bits	One stop bit [0]	
FT Flow Control Threshold	D9	Bytes
RO Transparent Packetization Timeout	3	* character times

Appendix Figure 11. End Device API and UART XCTU Configurations

AT Command Options
Change AT Command Mode Behavior. Command mode is only accessible via the UART.

CC Command Sequence Character	2B	Recommend...F (ASCII)
CT Command Mode Timeout	64	x 100 ms
GT Guard Times	3E8	x 1 ms

UART Pin Configuration
Pin configuration for the UART interface

D6 DIO6/RTS Configuration	Disabled [0]
D7 DIO7/CTS Configuration	CTS flow control [1]
P3 DIO13/UART_DOUT Configuration	UART_DOUT [1]
P4 DIO14/UART_DIN Configuration	UART_DIN [1]

Appendix Figure 12. End Device AT Command Options and UART Pin XCTU Configurations

I/O Settings
Modify DIO and ADC options

D0 DIO0/AD0/Commissioning Button Configuration	Commissioning Button [1]	
D1 DIO1/AD1/SPI_nATTN Configuration	ADC [2]	
D2 DIO2/AD2/SPI_CLK Configuration	ADC [2]	
D3 DIO3/AD3/SPI_nSSEL Configuration	ADC [2]	
D4 DIO4/SPI_MOSI Configuration	Disabled [0]	
D5 DIO5/Association LED Configuration	Associated indicator [1]	
D8 DIO8/DTR/Sleep_Rq Configuration	DTR/Sleep_Rq [1]	
D9 DIO9/Sleep Indicator Configuration	Awake/Asleep indicator [1]	
P0 DIO10/PWM0 Configuration	RSSI PWM Output [1]	
P1 DIO11 Configuration	Disabled [0]	
P2 DIO12/SPI_MISO Configuration	Disabled [0]	
PR Pull-up Resistor Enable	FFFF	Bitfield
PD Pull-up/down Direction	FFFF	Bitfield
M0 PWM0 Duty Cycle	0	
M1 PWM1 Duty Cycle	0	
LT Associate LED Blink Time	0	x 10 ms
RP RSSI PWM Timer	28	x 100 ms

Appendix Figure 13. End Device I/O Settings XCTU Configurations















































I/O Sampling
Configure IO sampling parameters

IR IO Sampling Rate	0	x 1 ms
IC Digital IO Change Detection	0	Bitfield
AV Analog Voltage Reference	2.5v reference [1]	
IF Sleep Sample Rate	1	

Appendix Figure 14. End Device I/O Sampling XCTU Configurations

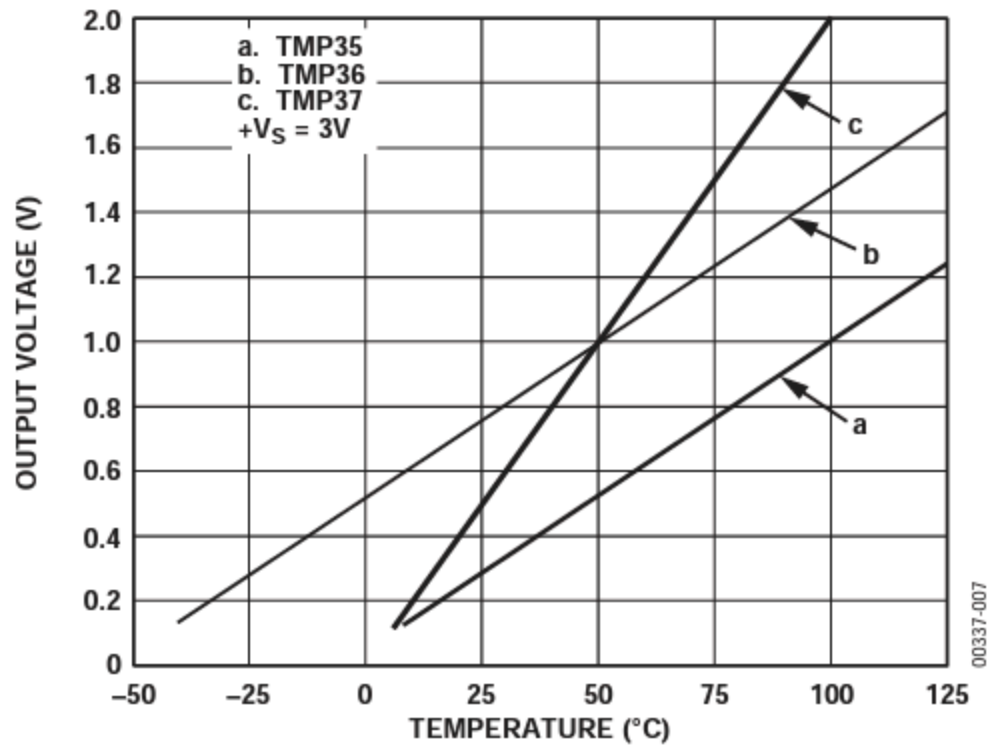
▼ I/O line Passing

Configure IO passing parameters

i	IA I/O Input Address	FFFFFFFFFFFFFFFF		 
i	IU Enable Serial Output For I/O Line Passing	Enabled [1]	▼	 
i	T0 D0 Output Timeout	0	x 100 ms 	 
i	T1 D1 Output Timeout	0	x 100 ms 	 
i	T2 D2 Output Timeout	0	x 100 ms 	 
i	T3 D3 Output Timeout	0	x 100 ms 	 
i	T4 D4 Output Timeout	0	x 100 ms 	 
i	T5 D5 Output Timeout	0	x 100 ms 	 
i	T6 D6 Output Timeout	0	x 100 ms 	 
i	T7 D7 Output Timeout	0	x 100 ms 	 
i	T8 D8 Output Timeout	0	x 100 ms 	 
i	T9 D9 Output Timeout	0	x 100 ms 	 
i	Q0 P0 Output Timeout	0	x 100 ms 	 
i	Q1 P1 Output Timeout	0	x 100 ms 	 
i	Q2 P2 Output Timeout	0	x 100 ms 	 
i	PT PWM Output Timeout	FF	x 100 ms 	 

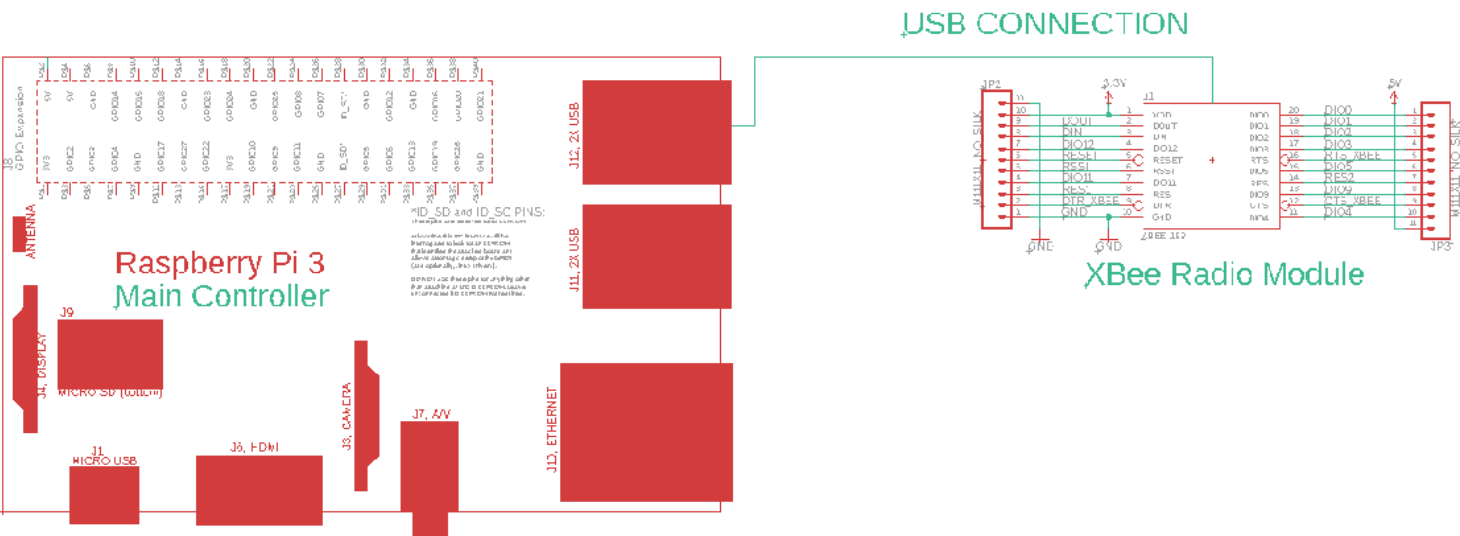
Appendix Figure 15. End Device I/O Line Passing XCTU Configurations

Table of TMP36 Output Voltage vs Temperature



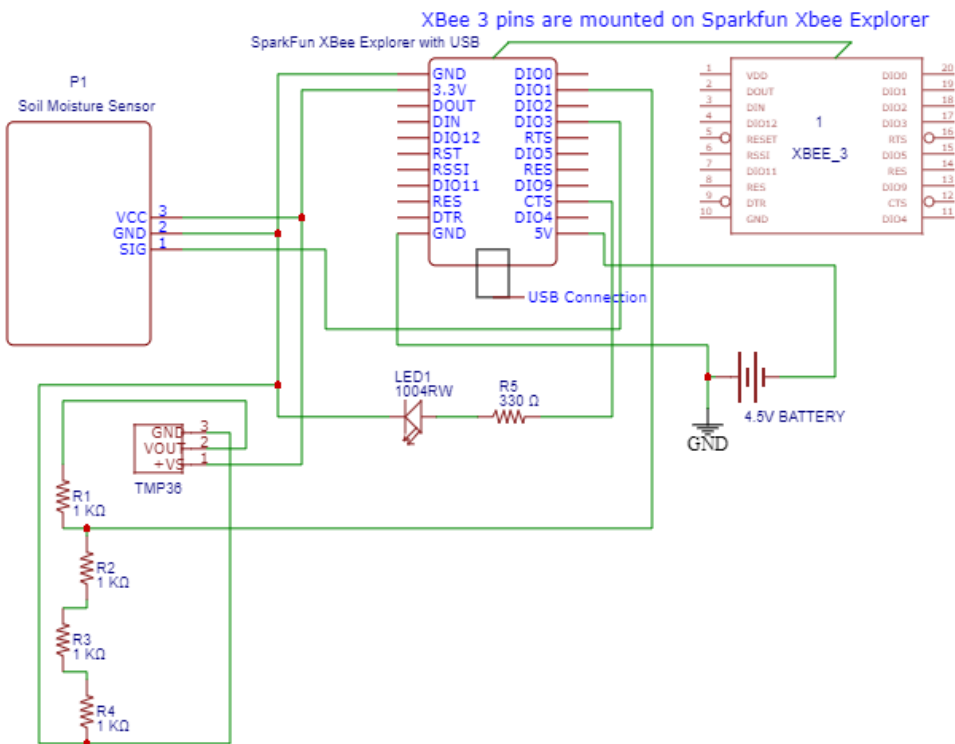
Appendix Figure 16. Output Voltage vs Temperature

Diagram of Coordinator connections



Appendix Figure 17. Diagram of Raspberry Pi with Coordinator

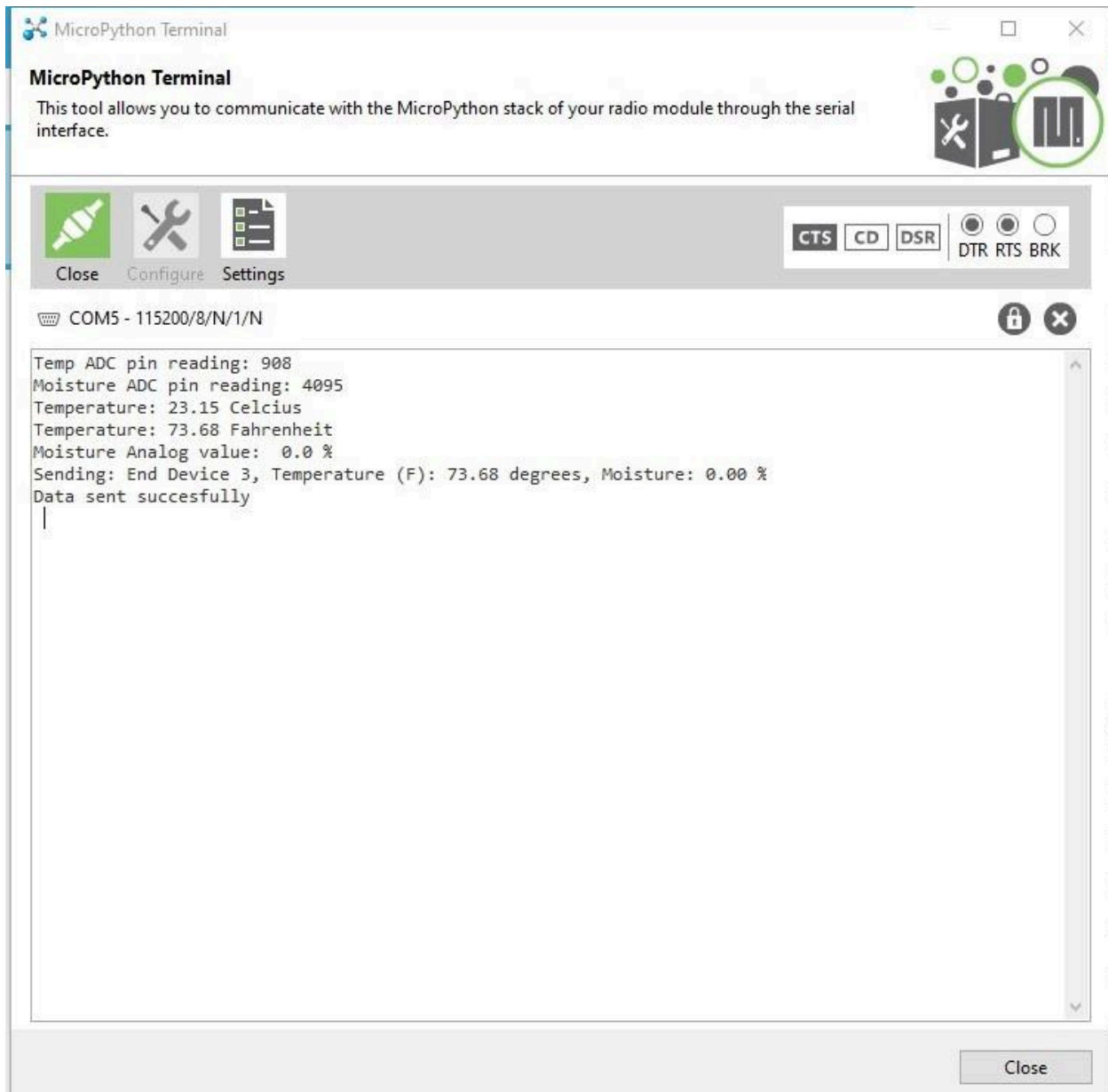
Diagram of End Device connections



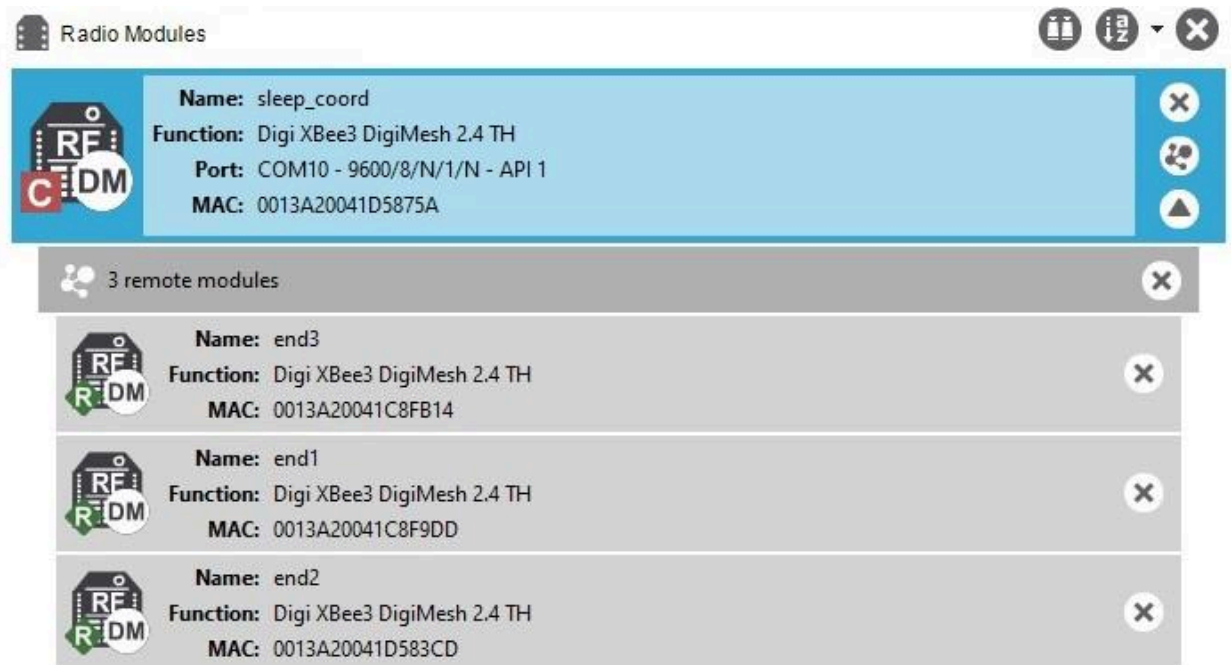
Appendix Figure 18. Diagram of standalone End Device with sensors



Appendix Figure 19. Coordinator MAC Address



Appendix Figure 20. MicroPython terminal



Appendix Figure 21. End Devices under Coordinator