UCF Senior Design II

“Spicer” Automatic Spice Dispenser

Department of Engineering and Computer Science
University of Central Florida
Dr. Samuel Richie

Group 10

Adrian Garcia	Computer Engineering	adriangarcia@knights.ucf.edu
Jacob Wood	Electrical Engineering	jwood98@knights.ucf.edu
Marcos Barros	Computer Engineering	marcosbarros@knights.ucf.edu
Nicholas Campbell	Computer Engineering	nickcamp556@knights.ucf.edu

Contents

Executive Summary

Cooking is an art that human beings have been partaking in for centuries, and with this storied history of this art comes a long history of the use of many different types of spices throughout history. Historically, different societies used the herbs, roots and fruit that were local to their territories, which helped to develop the great diversity of cuisines and flavors we see around the world.

The trading of spices around the world has occurred since antiquity, as they had a very high perceived value for use as medicine or for religious rituals. Nowadays, spices are mainly used for cooking, but their trade and availability is unprecedented. We now have access to hundreds of different kinds of kinds of seasoning to flavor our meals, making the possibilities for culinary creative expression innumerable. However, with all this variety comes a slight inconvenience: the organization of these spices.

Using the correct seasoning for a dish is imperative to the proper preparation of a well-crafted meal, but this becomes difficult when one’s spice cabinet contains a great variety of spices. Some spice containers need to be placed behind or on top of another in order to fit some cabinets. Currently, there are small turntables, colloquially referred to as Lazy Susans, that aid in this problem by placing the spices on a rotating platform, meaning that no spices will be hidden behind another and all spices can then be in full view. Our project is to improve on this design by creating an automatic turntable for spices that rotates in such a way that the wanted spice will be rotated toward the user. We plan on bringing the concept of the Lazy Susan to the modern age, while improving on and adding other features

that take advantage of current technologies for an innovative experience for all who want to explore their culinary possibilities.

Project Description

In this section we will be discussing what the Spice Rack will do, and the certain goals and objectives we need to overcome in order to get the Spice Rack as a system to operate to the specifications.

Project Motivation and Goals

As previously mentioned, the organized storage of spices and herbs can be a challenge, but there are some solutions that are currently available that address this issue. However, we see an opportunity to improve upon the designs available currently while also creating a learning experience for our members to develop our skills as computer and electrical engineers.

We propose a system consisting of motorized turntable with numbered containers for placing each spice and a mobile app used for initial setup and wireless control of the turntable. The position of the container relative to the front of the device will be stored in on-board memory in the turntable’s chip, while the app will translate the user’s selected spice into positions that will be sent to the turntable via Bluetooth. During the initial setup of the app, the user would assign each slot in the app to a spice and place the spice in the corresponding container on the turntable. After this initial setup process, the user would be able to select the spice on the app through interacting with the app’s user interface or through voice commands. The turntable would then respond by turning itself so that the spice requested is directly on the front face of the device. Each slot will have a small LED light next to it to indicate that the spice within has been selected. The user will also have the option to cycle through the spices on the board by pressing physical buttons on the turntable or by using hand gestures, which will be captured by the turntable’s on-board image sensor and processed by a specialized controller on the turntable.

With this variety of features included on this turntable, we believe this should provide an optimal level of challenge for us as a team, giving us the ability to display our level of engineering and design ability and our capacity to solve real-world problems in our engineering fields. The features also enhance the user experience for our target user base in significant and useful ways.

Objectives

The main objective of this system is to improve upon current spice organization solutions in order to maximize convenience and flexibility of use. We want our solution to be able to distinguish itself from other current solutions by providing a greater variety of ways to interact with the turntable and adding other convenience features such as alerts when a spice held in the turntable is in low supply.

We also want to be able to deliver this convenience without limiting the user’s ability to have a variety of spices in their cabinet. The user should be able to put in any of their desired spices into the turntable without worrying about whether the turntable can recognize the spice and properly respond to the user’s commands.

Requirements

- *Physical layout*
  - Have at least one level of spices with 8 spice containers per level.
  - Turntable diameter of approximately a foot for both levels.
  - Marking for each spice slot to match in-app turntable representation.
  - Should be able to be carried by one person.
  - Have proprietary measuring containers for dispensing spices.
- *Electrical features*

    - The turntable will be battery powered.
    - LED's will surround each spice container and will be able to light up to indicate the requested spice.
    - Flash all LED's in the event a requested spice is not present in the turn table.
    - A memory device must be present on the turntable, so in the event of a disconnection from the app, a second set up isn't required.
    - The light sensors track the amount of spice remaining.
    - Photovoltaic cells on the device to charge batteries.
    - Motion sensor to sense gestures to choose different spices.
    - Microcontroller with Bluetooth connectivity to manage power input and outputs and process input from the user through:
        - The mobile app
        - Physical buttons on the device
        - Gesture controls
    - Separate chip for computer vision processing, which will be used to gesture controls.
    - Camera or image sensor will be attached to the platform to allow for gesture controls.
- *Mechanical features*
    - Have a maximum speed that allows it to deliver a spice on the other end of the table in at most 5 seconds.
    - Turntable must be able to withstand water splashes.
    - Utilize a pouch to maintain the position of the spice while in motion.
    - Manually and automatically dispense spice.
    - Spice containers must be refillable with any spice the user desires.
- *Mobile app*
    - App must be available for iOS and Android devices.
    - App must be able to perform set up and labeling of spices.
    - App must communicate wirelessly with the turntable.
    - UI representation of turntable on the app for initial setup and ease of use.
    - Through voice command to the app (or to a home assistant), the device will be able to deliver a requested spice.
    - App will allow for reorganization of spices after setup.
    - App will send the user a reminder through a push notification to buy more of a certain spice when the amount of that spice is below a certain threshold.
    - App will retrieve amounts of spices from online recipes chosen by the user and the device will dispense these spices with the desired amounts.
- *Miscellaneous*
    - Components must cost a total of $450.

    - Will be colored a solid black color.

Specifications

- The system shall take less than 4 seconds to reach any spice in the spice rack.
- The system shall weigh less than 25 pounds.
- The system shall cost less than $450.
- The system shall use a solar panel to charge a 6-volt battery to maintain power efficiency.
- The system shall use motors running at no less than 9 rpms in order to maintain quick and precise dispensing.
- The system shall hold at least 6 spices at once.
- The system shall be able to dispense 1 teaspoon of fresh crushed spices under 15 seconds.
- The system shall be able to dispense 1 tablespoon of spices under 15 seconds.
- The system shall be able to convert 6 volts to 3.3 and 3 volts in order to power components and have communication signals between components.
- Must detect hand gestures performed in less than 2 meters of the camera.
- Raspberry Pi camera system must boot and begin operation within 15 seconds of the on switch being flipped.
- After a hand gesture is detected, the chassis must respond to it in under 5 seconds.

Research related to Project

In this Section we will be discussing the various research for the system and how this research will be implemented in the design.

Existing Similar Projects and Products

Currently, there are some commercial projects and products that aim to achieve some of the goals that we set out to achieve with this project. However, our project offers a more comprehensive feature set that is not available in many of the extant solutions when each is considered individually.

One such example is the TasteTro Spice System. This project was very recently released, having launched their crowdfunding campaign for the product on May 8, 2018[1]. This product offers automatic dispensing of spices which are requested through a screen on the front of the device. The user of the device has the option to pour individual spices by selecting the spice on the screen. The amount of the chosen spice that is poured is measured by the device. The device can also pour out spice blends that are stored within the device, which the user can choose on the device's screen.

The spices are stored in proprietary pods with RFID chips. These chips are used to identify each spice, and update which spice blends are available to dispense from the device. The available spice blends can also be updated through Bluetooth connectivity.

There are other solutions in the market currently[2], which have less automatic features. Such solutions offer a convenient storage system for spices by providing the user with a number of canisters on a carousel the user can easily refill. These canisters have a pouring mechanism that allows for pouring in measured amounts. The user can place any of their spices in these canisters, and the canisters usually come with labels that the user can write on to indicate the spices they filled the labeled canister with. Some solutions can also be stacked, so that a single platform can have two or more layers.

Our solution offers a similar feature set as some of these products. Our solution aims to provide the automated features of TasteTro while also giving the user more control and customizability with the spices that can be stored in the rack. This will be because our device's spice capsules will be filled by the user instead of being prepackaged in proprietary pods. The labeling of the spices will be done with the help of a mobile application that will connect with our device and be used as one of the ways to select a spice.

Our solution will also be innovative in the ways to select a spice. We will use our previously mentioned app to have the user select between the spices in our rack using the user interface or voice commands spoken to the phone. The device itself will have physical buttons or hand gestures to scroll through the available spices on the rack. This provides a variety of ways to interact with the device to accommodate for convenience and accessibility.

Relevant Technologies

These technologies will be a detailed account of specific components that were added to the Spice Rack design and their technical information.

Motors

This section will discuss how motors work and which motor would work best for our design. Also, it will talk about the difference between a stepper motor and a gear reduction motor.

A major key to our project is how precise and quick the spice rack spun to the desired spice position. By using a motor with a gear system, we can spin the whole rack to the desired position precise and accurately, while maintaining our budget for the components. When looking at different types of motors, we needed something that is efficient when it is running and when it isn't, can move to different positions during an extended time frame, stays at a given speed so the spice rack maintains stability, and isn't too expensive. While there are quite a few different types of motors, the two that would fit our design best were the gear motor and stepper motor because they meet one or more of the requirements that we are looking for.

The stepper motor allows the user to go to a very precise position, which is a huge must for our design because we need the motors to move the spice rack to the exact position for the spice to be dispensed. Along with the precise positioning, the stepper motor gives great low speed control. This feature would allow us to keep the whole system stable, along with keep the positioning more precise. Also, the stepper motors stay in our price range, even though we would like for them to be a little bit cheaper. Although the stepper motor checks all the boxes, the one fatal flaw is that it has high current consumption when the motor is not running (can consume 2 amps when they aren't even turned on). When a device consumes tons of current when they are not running, it means the power is getting transferred into heat. This causes other devices in our design to heat up and consume more power than we need them to. Since our design used a portable battery and a PV panel that recharges our battery, we need a everything to be as efficient as possible. When adding a device that heats up when it isn't working, it causes a lot more harm that outweighs the upside of the device.

On the other hand, we can use a gear motor or more specifically a gear reduction motor. Looking at Bright Hub Engineering's article called "What are Reduction Gears? What is Gear Reduction?", shows that a gear reduction motor would give us the ability have low speeds and high efficiency because of the gear ratio inside the motor. Along with the making the motor have a high efficiency, the gear ratio also gives the motor higher torque per power ratio. A high torque per power ratio allows for less current to be drawn to turn the spice rack. While the spice rack is charged up using the PV panel, it is good to have a motor that is efficient enough to allow different components to draw more power if needed or if other components run hot. Also, the gear reduction motors are fairly priced and are a fraction of the cost of some stepper motors. We have the possibility in getting multiple gear reduction motors for the price of a stepper motor we would need for our design. However, the one flaw the gear reduction motor has is the fact that we can't directly know the position of the shaft during a given point in time. Yes, we could do the math by hand, but that wouldn't take into account the little changes of speed during its run time. But if we would get a gear reduction motor with an encoder on the end, it would allow for us to know the precise position of the motor and get a more accurate reading on the speed during run time. So, with adding the encoder we were able to have precise positioning of the motor to position the spice in the right position to dispense, accurate speed to make sure the spice rack is stable and gets to a position quick enough, and is efficient when it is running through the torque per power ratio and when the motor is not running it the gear reduction motor takes only a fraction of power that the stepper motor takes when it isn't running.

Encoders

In this section, the main objective is to understand how encoders can be used to give us precise positioning for our spice rack and our dispenser.

The article "WP-2011: The Basics of How an Encoder Works", explains that an encoder is a control system that gets feedback from a mechanical aspect. Most encoders give their feedback from the rotation of a motor, and from this feedback know the location of the shaft of the motor. There are many types of encoders like mechanical, magnetic, resistive, and optical. While there are different types of encoders all send back relatively the same signal that is a series of high and low waves which show the change in position of the motors shaft. These encoders can track the changes in the signal and know which location the motors shaft is in. Then through the signals sent out, a microcontroller can interpret and know the different locations the encoder is seeing and can figure out the speed due to the changes in location.

When encoders send these signals to the microcontroller, the signals are often in a form of a single channel or a duel channel. A single channel encoder can only be used for motors that spin in only one direction (can't use this one). While a dual channel encoder sends out two signals that are phased 90 electrical degrees apart. The combination of these two signals give the position and the direction of the motor at a given point by seeing the leading and lagging feedback signal. Dual channel encoders give great direction and motor positioning but are used for complex motion control applications.

Motor Drivers

In this section, the objective is to shine light on how motor drivers will help the boards microcontroller to communicate to the motors. Along with how the microcontroller was able to control the direction of the motors.

When using a motor driver, it gives the user control over motor through the microcontroller by using a lower current signal to drive the Essentially the motor driver takes in a lower current signal and converts it to a higher current signal by using different current amplifiers, which gave the motor enough power to run at its rated load. All the motor driver is doing is translating the information from the microcontroller to the motors with a signal that allowed the motors to understand and perform the way the microcontroller wants them to ("DC Motor Driver Circuit"). Since the design has two motors rotating the spice rack, we need a motor driver that can drive both of them at the same time and can run around 1 amp of current (each motor is rated at a max of 0.5 amps). The motor driver also has to output a voltage of 6 volts because both of our motors run on 6 volts. While one could run with various amplifiers that transform the signal from the microcontroller to a signal that would run the motors, it would cause a lot more headache than just using a known motor driver that could meet all the requirements needed to control and run the two motors.

Battery/Power Supply

This section will touch on how a battery was selected for our design, and the features a battery needed to run our spice rack.

In order to run the entire system of the spice rack, a power supply was needed to supply power to all the various components. The best option for our system would be a rechargeable battery. While there may not be a perfect battery out there for the spice rack system, the system needs to have a battery that can handle the amount of power the motors drew at their maximum load. When looking at different batteries, they give the voltage of output along with the amps per hour (Ah). The relationship between the voltage and amps per hour are a good way in finding the total power that can be produced by a certain battery. While these are the two keys in find the right battery, a major factor is the drain rate of the system. If the spice rack were to pull a maximum of 4 Watts during a given point, then the battery would need to be larger than 4 Watts per hour because the battery won't be able to supply enough power to keep running the system. A good idea to finding the right battery for the spice rack system, would be to find the total worst-case power consumption of the system and get a battery that can supply slightly more power than that. A huge advantage to the spice rack system, is that the large power consumption of the motors won't be running for a long period of time. So, the battery didn't have to run at full capacity, which allowed for it to maintain its charge and power longer ("How to Select the Right Battery for Your Application? Part 1: Important Battery Metric Considerations").

There are among other factors that go into selecting a battery than just the amount of power that it can give off. These factors are the shelf-life, chemistry, size and shape, and cost. Since our design is mainly small, the cost of our batteries was moderately cheap, but the price of the batteries goes hand and hand with the chemistry and the shelf-life. Most batteries tend to get more expensive the longer it did last, and that is due to its chemistry and shelf-life. While large power batteries tend to last longer, it is good to understand how long the battery would last due to its chemistry.

PV Panel

This section, like the battery section above, will give understanding on how the PV panels will function with our system and how PV panels was selected for design.

The biggest factor when finding the right PV panel would be the time it takes for it to recharge the battery. While the spice rack would only be used during certain times of the day like dinner, lunch, or even breakfast and for maybe 30 minutes maximum, the PV panels won't have to be too large to recharge the battery to full capacity. This makes it easy to put a PV panel on top of the system, and by photons coming from household lights, the PV panel was able to produce energy to recharge the battery. So, for a PV panel that is small enough to put on the top of the spice rack the panels power would only be about 2 Watt. This seems like a stretch since the lowest battery we would use is a 50 Watt per hour battery. From "Estimating Solar Charge Time for Batteries", the time to fully charge this battery up would be the (Battery Capacity (in Watt hours) * 2 / PV panel Power (in Watts). Using this equation, it would take the 6-volt 2 Watt PV panel to take 30 hours to recharge the battery. But that is to recharge the battery fully and let's say that the battery is only used for 1.5 hours a day, then the battery would be recharged for 22.5 hours for the rest of the day. This gives plenty of time for a small 2 Watt PV panel to charge up a 30 Watt per hour battery. But using an 18-volt 4.2-watt PV panel, we would only need to take half as long as the 6-volt 2-Watt PV panel. It would also allow for us to take away the nonlinear characteristics of the panel because the current and power from the panel is correlated with the amount of light that impacts it. While light isn't that constant of a variable.

Unfortunately, due to Covid-19 the PV Panel wasn't added to the final product even though it was procured. But it was able to charge the battery.

LED Lights

There needs to be a way of telling the user the current spice pods the spice holder is on. This function is to both help with manually pouring the spices into the standardized cups. To make this visualization there are multiple ways to do this feature. For instance there can be a voice interface that can indicate to the user which spice is the current spice and level. There could also be a screen interface in which the user looks at the spices on an embedded application. The way that was chosen to be implemented in this project is to use a light to indicate the one that is currently in use. Since all the spices had this light there was a way to tell the user which spice is the one currently being used.

To notify the user which spice is currently being used it had two different modes. Active mode and inactive mode. These modes served as the way to indicate to the user the current spice. Without this function it would be difficult for the user to pinpoint which spice is about to be poured. Moreover, without this function for manual pouring it would be confusing to the user which one is ready to be manually poured.

The way the lights indicated to the user that this is the current spice is by toggling in between red and green. The green light indicates which is the current spice being poured. The red light is used to indicate to the user that these are spices that are not in the place to be poured. There is only one spice being poured at any given time. This means that there is only one green light at a time.

The lights that are used to indicate this function are standard LED bulbs to achieve the goal of indicating to the user which spice is ready to be poured. The reasoning behind doing this for the design is these are cheap and reliable. Having the ability to get bulbs in bulk for a small cost gives the ability to use manage the budget easily. The light-emitting diode (LED) is one of today's most energy-efficient and rapidly-developing lighting technologies. Quality LED light bulbs last longer, are more durable, and offer comparable or better light quality than other types of lighting. (LED Lighting – Department of Energy). This is also good for energy efficiency, which is another benefit for achieving a solar powered system. For each pod there are two LEDs. This means there is one red and a green LED. The LEDs are toggled depending on if it is the spice that is currently being used. Using this small and bright lighting system ensures the user is always able to tell which spice is being used and give the system good power efficiency.

This function was implemented in the design. Unfortunately the chassis did not support the usage of lights. This means that the lights are displayed but it cannot be seen by the user. There is a cap that hides the wiring and embedded systems we implemented in the spice dispenser. If design constraints permitted this we would have just changed the pin to a gpio pin and display to the user this functionality.

Proximity Sensors

This section will explain what we will be using to detect how full each spice pod is currently. Doing this would have given the user the ability to keep track of the amount of remaining spice in each pod simultaneously.

For this project, we are did not implement a Sensor to tell what the specific spice level is. According to Merriam-Webster's definition a sensor is," a device that responds to a physical stimulus (such as heat, light, sound, pressure, magnetism, or a particular motion) and transmits a resulting impulse (as for measurement or operating a control). We are trying to achieve this with the spice levels at any given time. Out of these sensors we used the light sensor. "A Light sensor is a mechanical device sensitive to light, temperature, radiation level, that transmits a signal to a measuring or control instrument", according to Dictionary.reference.com. So, there are multiple ways to measure the light from the sensor. Since we know that this instrument is sensitive to temperature as well, we needed to ensure we have a recommended operating temperature so we alleviated the problem of it not working. With the radiation level, that won't be a problem because we are not working with radiation.

Looking at what the benefits of using a light sensor would be for this project there are a few. The energy consumption benefits are great because we can control the amount of power we need. It was used in an automated format. This sensor could set zones of operation. Moreover, Light sensors do not require large amounts of voltage to operate. This is a sensor that the shape of the output can be manipulated.

There are a few things that should be taken into consideration when it comes to using a light sensor. The higher response times the slower LDR’s are when in use. The resistance was varying constantly. Photodiodes are temperature sensitive which and end up being unidirectional which can sometimes cause problems. Phototransistors will break from a volt input above 100v. It would also need surges, spikes and EM energy protection to run optimally.

Surges can impact any electrical device. This is a disturbance to a power system. This can be both a positive and a negative disturbance. This could create overvoltage. This overvoltage can destroy an electronic system by providing more voltage than the system can manage. This type of problem must be accounted for in any electronic plan. This something, in our project we accounted for by looking at different mediums that offer this protection.

To ensure that our device has protection from surges we ensured that there is a way to implement this. A surge protection device that includes varistors to absorb surge currents and protect a load from an overvoltage condition. (Palma and Mosesian). We know that phototransistors are vulnerable to surges. To protect this there needs to be a counter measure. Surge protection fulfils this need to ensure the device is protected from such voltage abnormalities.

There a few types of light sensors that can be used to identify what when the light level changes. There is the Photovoltaic sensor or the ones that we usually refer to as the solar cells. There is also one of the more commonly used light dependent light sensors which just gauge the different light levels. Photo diodes are the ones we typically use in cameras to tell when to use the flash. Proximity sensor uses infrared light to detect motion or proximity to something else to manage the light levels. Diving into each we saw the one that is the most useful for our project of managing spice levels.

Photovoltaic light sensors are interesting because of the uses of this in the industry. Why these are usually referred to as the “solar cells” is because they are the ones that are typically used in emergency lights. How is works is it detects high light and then generates current or voltage, depending on the implementation. If it is in low light situations it doesn’t generate current. These light sensors save energy in silicon cells to be used later. A downside to this is that it only can look at light that is human visible. This is a viable way to emit light given a specific situation.

Light dependent light sensors are commonly used for assessing light levels. This implementation of light sensors is from the branch called photoresistors. Photoresistors increases resistance proportional to the amount of light is in its surroundings. This means that’s the more light the more resistance is given. These are typically used for outdoor lightings because of the sun. These generate a small amount of current in the slightest of light change. These are hypersensitive light sensors that have a versatile amount of uses.

Proximity light sensors are one of the most interesting of the light sensors. These sensors use inferred light to detect motion and the immediacy of another entity. Proximity light sensors are usually used for robotic applications. To avoid hitting objects and being able to tell when things are in the general vicinity, we use the proximity light sensors. Other great applications for this type of light sensor is vehicles, especially for automatic braking systems. This is because it can warn or act when the proximity reading is small

For our implementation we have be used the proximity sensors because it is the most reliable one when it comes to being able to tell if something is being blocked. We want to know when a threshold is passed. Inferred rays are hard to mistake with other frequencies. This light sensor is ideal for the case of translucent spices that could deflect light or push light into another light sensor.

There are different speeds can be achieved depending on the different type of material used in the project. There are photoresistors offering the slowest response time of the light sensors. Photodiodes have a better bidirectional response time. Phototransistors offers the best time with the addition of generating a higher current than the photo diode. These options offer versatility in the project.

Photo resistor are resistors that depend on light to function. These parts have a semiconductor material that makes it behave like an actual resistor. The resistance is measure by the amount of light that hits this component. Using the amount of light that hits this component we can determine the resistance and use that for other reasons. Photoresistors are highly sensitive to light and will is hard to get a steady reading. When using this part, it should be considered to have a controlled amount of light that hits it.

Photo diodes are devices used to convert light to electric current. This component is made with a semi conductive material. The current that is produces is dependent on the amount of light absorbed. Unfortunately, the higher the surface area the slower the response time that is produced by this component. Knowing this helps to identify what type of implementation this component is good for.

Photo transistors are like phototransistors. This component is having one major difference from the ordinary transistor that is that it uses light to produce the current. This makes it a versatile component because it can achieve better responsiveness because of the format of the transistor. This is helpful to know for the sake of comparing which part will produce the best responsiveness for our project.

To make it easier for the user to know how when a spice is low, we need a way for this to be indicated. By using a sensor, we can accurately track what is going on in each pod. A sensor is the best way to report what is happening in the physical world accurately. We chose the light sensor because it seemed to be the most straightforward way of measuring at what level the light sensor is at. Upon researching we realized there were several advantages to using a light sensor to track our spice rack. By having the ability to control the brightness of the light we can then influence the amount of power being used to operate each light sensor. Being that the light sensor is only one beam of light and a sensor that tracks if the light can be seen, this takes the complexity of trying to figure out what to do or fix to only two components. These two components can be individually checked to see if the function is being achieved. We can also influence the size and shape of the light beam. Having the ability to influence these properties gives us the chance to then too make it more versatile depending on the spice. Since each spice has different physical properties.

Reasons we choose to use this type of sensor. Photodiodes are also cheap to produce, which helps with the overall cost of the project. The light sensor offers a quick response time at a low cost to create digital outputs. There are multiple choices to implement the light sensor for speed. There are photoresistors which are the most basic components to creating a light sensor. A step up is the photodiode which offers a faster response than the resistors. The next level would be Photo Transistors which generate higher current than the photodiode.

To dive deeper into why we are choosing the parts over all the rest, let’s examine the different implementations of the proximity light sensor and the mixture of photodiodes and the phototransistors. It is helpful to note, the decisions made were skewed to cost of production. To make an optimal product we need to see which one works best for the implementation we are trying to fulfil.

Why we chose to use the proximity light sensor. This is because the others couldn’t fulfil the same purpose. With the proximity sensor we wouldn’t need to be concerned as much about the area in which we the device would be placed because of lighting. With a Phot resistor it would be difficult to control the amount of light that could enter the container. This would be a problem because we would need to fix this by implementing a way to control the light. The implementation of a thing to restrict the amount of light would make it difficult for the user to see what was happening with the spices. This meaning, the ability to double check the device would be restricted because of the light restriction. We use light to read, which if restricted we wouldn’t be able to read the articles used.

The combination of both photodiodes and phototransistors helps us to be able to gather the appropriate amount of light in a controlled manner. To be able to work bi-directionally with the light sensor we need a reliable response rate. Though the benefit is the second best, this offers a moderate boost to what is trying to be achieved which is not a real-time system. The phototransistors was used to enhance the speed for multiple sensors checking the status

Looking at the performances between different implementations of light sensor it has been discovered that with detecting if there is a vacancy inferred light sensor work better than the light dependent resistors. Since with inferred the frequency of light is different gives it an edge over the standard light dependent resistors. The performance analysis of the accuracy for detection of vacant parking slots and vehicle detection under different conditions is presented. It is concluded that IR sensor outperforms LDR sensor (Bachani, Qureshi and Qureshi). From the observation of how it works with vehicles it can be determined that this is the best implementation to check for vacancies.

This function was implemented in our design but, not in the way that was originally intended. It instead was used to ensure the mechanical feature of ensuring that the motor was correctly lined up with the wedge was done correctly. This was due to the lack of having a transparent container. This would have been a great feature if design constraint permitted.

Strategic Components

In this section, the research will show the specific components that will be added to the Spice Rack design. Along with showing how each component would function in our design.

DC Gear Reduction Motors

This section will explain what a DC gear reduction motor is and how it operates, along with how it is the best motor selection for our design.

We were able to narrow down that a DC gear reduction motor would be best for our design because when running off a portable battery, it is way easier to step down a voltage than to add a DC to AC converter in our design. Plus, the gear reduction motor with an encoder allows for precise positioning, controlled speed, high efficiency and is fairly priced. Which checks all the boxes for what is needed for our design. Before we could just throw a regular DC gear reduction motor with an encoder, we needed to know how it worked and what affects it would have on our design.

Through research, we were able to find out that the DC gear reduction motor would work like any other DC motor. When current passes through a coil or loop of conductive wire that is inside a magnetic field (between a north pole and a south pole) a mechanical force was applied to the coil of wire (This can be seen in Figure 1 below from Explain That Stuff's article "Induction Motors"). This mathematical principle is known as Flemings left hand rule where the direction of the current and magnetic flux density is perpendicular to the force applied to the coil of conductive wire, and the equation can be seen in Figure 3 below. So, inside the DC motor all this rotation happens inside an Armature, which is where there are multiple coils of conductive wire intwined together to move the shaft when a current is running through them. When applying a direct current through the motor there needs to be something that would cause the current to switch directions, because if the current doesn't switch directions the coil would eventually stall at a given point where the forces acting on it cancel out. Through the Commutator, the DC motor is able to switch the current through the various coils that cause the shaft to spin. A typical Commutator was able to spin with the coils so that the two different sides of the Commutator (one side is applied to the positive node of a voltage source and the other side is connected to the negative side of a voltage source) was able to switch between the positive node to the negative node of the voltage source, thus changing the direction of the current through the various wires. A full diagram of this process can be seen below in Figure 2, where you can see how the Armature and Commutator work together with a voltage source to turn the shaft of a motor ("DC Motor Basic Parts").

While a DC gear reduction motor does is like any other motor, it also has some features that are the reason it fits so well with our design. The main difference between a gear reduction motor and a regular DC motor is the gear reduction that is going inside the motor. So, a gear reduction is where a system of gears is added to slow down the revolutions per second (rpm) of the motor, while keeping the torque the same or increasing it. This is important because in our design we needed something that has a relatively slow rpms but would be able to have enough torque to turn the spice rack as a whole. The gear reduction allows for a high efficiency because the motor was be able to spin at high rpm with a low power consumption, and the gear reduction was stepped down the speed of the motor through a system of gears that are different sizes and maintain the torque because the motor was still running at the high speed. A gear reduction motor allowed for us to make a simple gear assembly using the same size gears from the motor to the spice rack, instead of having to step up or down various gears to make the spice rack spin slower. By using a simple gear assembly with the same size gears allows the encoder to be able to be more precise on the location of each spice and gives us less calculations to make when coding the encoder to certain spice locations.

Crushing/Dispensing DC Motor

This section below will show how a DC motor will be used to crush and dispense whole spices in our design. Along with being able to explain how the motor can crush and dispense the fresh spices.

A specific feature the spice rack is able to perform is crushing fresh pepper, salt, among other spices and herbs for the user. By using a motor to spin a rigid gear system (like a regular pepper mill shaker), the user canl get the desired amount of crushed fresh spices and herbs they desire. For the spice rack design, the motor would need to have low power consumption, have enough torque to crush pepper, and run fast enough to crush enough spice for the user. From the previous research done on the motors that rotated the spice rack, the ideal low power consumption motor would be a regular DC gear motor.

In order to dispense the spice desired by the user, we used a servo motor to dispense due to physical constraints and the lack of space for a dc gear motor.

Bluetooth

This section will talk about how the spice holder will connect to the user, wirelessly. This is to ensure that the user can interface with the spice holder without having to touch it

There are different ways to implement wireless communication. Wireless communication is important for this project because the user needs to know the status of each spice. To know the spice at any given time, there needs to be a way to know without looking at all the spices individually. To also manipulate which spice is in which spice pod there needs to be a way to tell that to the machine without inputting this into an onboard interface. This wireless implementation made the project convenient to use for anyone, hence, making the spice holder accomplish the goal of creating an easier way to hold your spices.

What is needed for this is a low power way to connect to the spice holder. This needs to work within short range, for our implementation. The project scope is limited on the budget this needs to be cut to make room for what is going on in the spice holder. Bluetooth is a standard for short range, low power, and low-cost wireless communication that uses radio technology (McDermott-Wells). With this knowledge Bluetooth is the solution to this problem. To avoid using too much power for the solar charging this is a great way to solve the problem of rationing the power we have too harshly. This also give the user the power to use the phone app to communicate with the spice holder. Since most modern-day phones give the ability to communicate with Bluetooth it makes this an easy implementation. For a wide variety of reasons Bluetooth is the best way to communicate with two devices from short range, this influenced the project to use this technology.

There are many advantages to Bluetooth. Bluetooth technology in addition to its low cost, offers the advantage of using a standard protocol to achieve wireless transmission with low interference and low energy consumption. (Sagahyroon, Eqbal and Khamisi). When considering the ability to go standalone it is necessary to be power efficient. When on a limited budget it is also needed to consider how to cut costs and still maintain efficiency. The unique advantages of Bluetooth such as low power consumption capability, cheap hardware interfaces and easy set-up offer new application areas. (Bohn, Bobek and Golatowski). Being an easy to set up tool makes for a good way to implement on the fly. Given that Bluetooth is easy to access technology gives room to create more features for the spice holder. The invention discloses a Bluetooth device quick pairing method and a Bluetooth device. (杨杰, 傅启洪 and 朱渊). This gives a good reason to use Bluetooth as well with this we can achieve quick connections which is much needed for the user to get updates when it is needed to tell the appropriate spice levels. With the reliability of Bluetooth, it is great for this project.

Possible Architectures

This section will discuss the various architectures of the spice rack system, and give understanding on how the architecture were specifically used in different scenarios.

Spice Rack Rotational Gear System

The Spice Rack Rotation Gear System will discuss the architecture behind how we will be able to move the spice rack to the desired spice, and the gear system involved to do so.

While the spice rack was rotated using 1 Machifit 25GA370 DC 6V micro gear reduction motor with an encoder, a gear system was needed to be added to connect the spice rack to the motors. By using a two-gear system that are the same size in diameter, allows the Machifit low speed of 12 rpms to be in full effect. If the gears aren’t the same size, then the encoder won’t be able to track the different positions of the spices. Depending on the gear ratio the, the 12 rpms could easily become 6 rpms and that would double the time between each spice. The gear ratio would actually be proportional to the speed of the spice rack, so if the spice rack gear was twice the size of the two gears attached to the Machifit motors then the speed of the spice racks rotations would only be 6 rpms. When the spice rack gear is twice as large as the two gears attached to the Machifit motors, the torque would also increase due to the relation to the speed of the two Machifit motors. Then if we would make the do the opposite and make the gears attached to the shaft of the motors, the speed would increase, and the torque would decrease. This would be the worst-case scenario for the design because the 12 rpms would be fast enough to reach a spice that is a half a revolution away. Also, if the gears on the motors were large than the gear that rotated the spice rack, it would add more torque for the motors to rotate as a whole. Which would inevitably decrease the true amount of torque the motors would be able to spin with. To keep everything that is on the datasheet from the Machifit motors, like the speed and the torque. By making the gears that are on the shaft of the Machifit motors and the spice rack, would allow for the speed and torque of the motors to be valid and for less testing as a whole of the system. But if needed, the changing the size of the gears to give more speed or more torque can best be done by changing the size of the gears.

Crushing/Grinding Spice Gear System

This section will discuss the architecture for how our design will crush fresh spices and the gear system used to accomplish the goal.

While the gear system was most likely be the same as the gear system for the spice rack rotation, the gears are a little different. The gear system had one outer gear that is held on an axis vertically that had teeth on the inside to grind and crush the spices and allowed it to fall down once the spice is small enough to pass through the small hole (works just like a regular salt or pepper grind). Then the gear system attached to the MM10 was the same size depending on how the design of the MM10 motor and the L9110S motor driver. But a good solution for the torque issue with the MM10 would be to make the gear on the MM10 motors shaft smaller than the gear on the axis that grinds the spice. By making the gear smaller, it would allow for the MM10 to have more torque but would cause the speed of the motor to decrease proportionally. While the minimum speed of the motor is 12400 rpms, it wouldn’t hurt to add a smaller gear to the MM10 shaft because the maximum torque for the motor is 10g.cm. 10g.cm might be enough to crush certain spices but when testing the system and if the motor had trouble grinding certain spice then the changing of the gear ratio between the two would have to be made. While decreasing the size of the gear to the motors shaft would benefit the system design issues with the torque of the MM10 motor, if we increased the size of the gear to the motors shaft then we should add more torque to the spice rack system (which inadvertently decreases the torque of the motor). When increasing the size of the gear on the motors shaft, it increased the speed of how fast the spices would be grinded. This wouldn’t really have any benefits for us because the MM10 already has a speed of 12400 rpms and adding to this wouldn’t do anything but add more noise to the system as a whole. So, if a gear ratio was added to the MM10 and the grinder gears, the grinder gear would have to be larger than the MM10 gear.

While we couldn’t implement this design due to the physical constraints of the system, we actually used a servo motor to dispense the spiced. The Servo rotates and pulls a paddle that is aligned with another paddle on the dispenser proper. Once this second paddle is pulled, a teaspoon-sized container is pulled forward along with it, until the teaspoon area is aligned with a funnel, allowing spice to be dispensed. After this, the servo returns the dispenser to its original spot, and the process is repeated.

Manufacturing Processes:

Due to the current global situation of the COVID-19 Pandemic, it will be more difficult than usual to acquire and use parts for our project. This is because of a few reasons, chiefly that origin of the outbreak was China, which is the source of most of the world’s manufacturing. This results in parts being more rare and taking longer to arrive, especially more customized components, such as a custom PCB. However, the actions undertaken in the United States to help curb the effects of the outbreak are also impacting our design process. Laboratories and manufacturing equipment that would otherwise be available to use are not available, especially those that would normally be provided for by the University. In lieu of these, it was decided to acquire a consumer grade 3D printer, the Creality Ender 3 Pro 3D Printer. This allowed us to print components for our design with dimensions of 220x220x250 mm. This was significantly less than what would have been used ideally, and was only used because another solution to the manufacturing limitations due to the current global situation was not be found.

The Creality Ender Pro 3D is designed to be used with a large variety of 3D design software, and multiple libraries of pre-designed components exist for it. Additionally, it should be noted that with the exception of the grinder and dispenser mechanisms, our design, in terms of mechanical engineering, was relatively simple. This means that, should our design require it, we could easily import various similar shapes into any freely available 3D design software and modify them slightly so they can fit together, thus then allowing us to create a larger design than what is limited by the Ender Pro’s size constraints. For instance, a circular base larger than the 220x220 mm size constraint is buildable using portions of a circle, such as by printing three thirds of full size sections. This will allow us to better work with the Ender Pro without compromising our original design.

Figure 1: Ender Pro 3D Printer

Additionally, it is necessary to consider the fact that the 3D printed materials can be made to have much more precise perfections and imperfections were necessary. This means it is now possible, if the Ender Pro is used, to tweak our design further than before. For instance, the dispenser and grinding mechanisms can be made to have different colors for maintenance purposes without requiring us to manually paint the components ourselves. Further, it allowed us to improve our shaping of components, since using the Ender Pro means that we can adapt the shape of various components to fit the mechanisms and parts we will use. For instance, suppose that we would need a square shaped container for the dispenser and grinder apparatus, then we could position it near the front of the device, and add a square shape that extrudes out of the circular shape that composes most of the device.

Then, materials were considered. Originally, it was stipulated that the entire device prototype would be constructed out of wood and would be provided with a matte black finish. However, due to lack of accessibility to wood, to wood cutting materials, and to industrial paint, we also must account for how the replacement of these parts will impact the design. For instance, wood is obviously much harder and resilient than plastic, even plastic that is used for 3D printing. Further, things such as the grinding apparatus will not be able to be made out of these softer materials, because then they will be unable to perform their tasks well. For instance, the grinding apparatus cannot grind if it is softer than what it is grinding. Thus, we need to carefully decide what we will print with. Chiefly, we went with two types of material for the components of the device:

- PLA: Polyactic Acid is one of the most popular filaments for 3D printing, which comes in a variety of colors and also can be used for the vast majority of applications. This is the softer of the two materials which we will consider, and thus will likely be used for most of the external sections of the apparatus. This means the apparatus will be less durable than it would be otherwise, but due to cost limitations we will rely on them for the majority of the device.

Figure 2: Polyactic Acid Filament

- PTEG: Glycol modified Polyethylene Terephthalate is a harder material than the Polyactic Acid. It too, can be made in a variety of colors and is to be used in applications wherein the softer materials cannot be used. For instance, it could be used for a container that has to be able to withstand impacts. Additionally, in our case, it will be used for the grinding apparatus, that will be within the device. For easy cleaning, it will be made with a partially transparent version of the material.

Figure 3: Polyethylene Terephthalate Filament

In the end, the grinding apparatus was scrapped from the final design due to time constraints, but the harder material was used for the gear system to allow for rotation, to prevent wear from damaging the system in the long term. This was the case with the original, “soft” gear system that was used.

Central Processing Unit

The central processing unit (CPU), serves as the brains of our design, as it would in any computerized system. In our case, the processor serves as an intermediary between all the peripherals, rather than as a complicated mechanism for input management. It sends data to and from peripherals and attempts to remain offline for as long as possible, to conserve energy.

MSP-430 Processor

We considered using the MSP-430 Texas Instruments Processor. This CPU is primarily designed and aimed at low power consumption embedded applications, making it ideal for our product, which does not require much computational power but does require a long-lasting battery and thus, requires low power consumption. This is because our design’s features are mainly centered around controlling a mechanism that will select the appropriate spice and dispense it at a correct quantity, as determined by a sensor. As a result, the CPU will mostly be serving as a redirector between inputs and outputs, with inputs read from the motor's position and from a Bluetooth input (which will be connected to a mobile application) and from a ML model, and directing commands, as appropriate, to necessary components, such as the spice grounding motor, used to ground pepper and other similar spices, various LED lights, which will mark the selected spice for a visual reference for the user, and will activate the dispenser mechanism. All in all, it can be concluded that power consumption will be minimal if the software is built around software and hardware interrupts, rather than through an infinite loop of constant polling for states.

General CPU Features

Additionally, the MSP-430 contains multiple features which will allow us to add multiple other features to our product in the long term, making it an ideal choice even for potential future growth beyond the context of this project, potentially expanding it into a proper product line. The MSP-430 contains a 16-bit von-Neumann architecture allowing for use of 16 registers to rapidly execute instructions within one clock cycle, as long as the data for the operation is held within registers, eliminating memory access. Of these 16 registers, the first 4 registers are used as a program counter, stack pointer, stack register, and as a constant generator. The remainder of the registers can be used as General Purpose Registers, however.

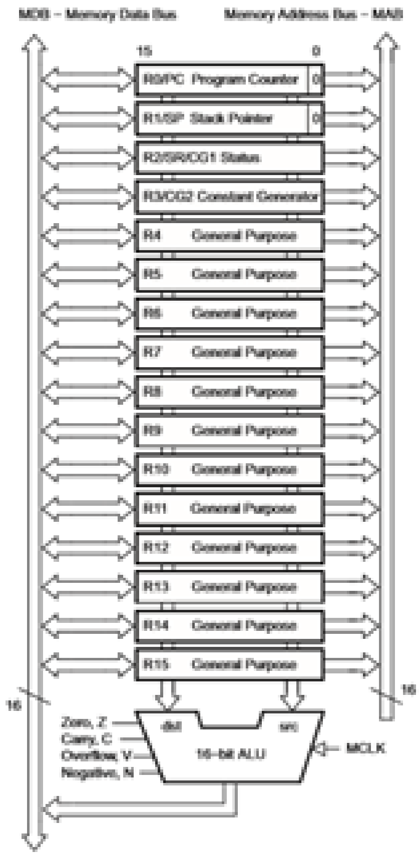


Figure 4: MSP-430 Registers

Special Registers

The registers marked from R0 to R3, have a specialized purpose. In this section, this will be elaborated upon further. The Program Counter (PC) is the first of these, R0, which relies on two-level increments of instructions, that is, increasing the program counter by a value of 2 every clock cycle, and every instruction is read as a Word. This Program Counter's value is also modifiable, allowing for jumping between instructions and thus performing loops and conditional statements. Secondly, there is the Stack Pointer (SP) stores the return addresses of subroutine calls, and interrupts. The Stack Pointer is initialized on the top of the Random-Access Memory and is aligned in such a way as to point to even addresses. Next is the Status Register (SR), which contains an assortment of bits that may be used to identify the current status of the CPU. For instance, there is the Negative (N), Zero (Z), Carry (C), and Overflow (V) bits which would then allow you to check the result of an operation logically, such as by checking the Z bit for equality in a subtraction.

Separate from these operation bits, there is also bits that reflect the status (On or Off) of the CPU, the status of the Oscillator, and the system clocks. Finally, there is the General Interrupt Enable (GIE) bit which allows for interrupts to occur. If this bit is not set, then interrupts may not be performed, and only direct program execution may occur. Finally, there is the Constant Generator, which allows for the creating of generally used constants, such as numbers which elevate 2, in order to control individual bits in a byte, and to simplify word processing.

General Registers

The general-purpose registers may be used to store information needed during program execution, but of note is that these may be accessed in multiple ways, to ensure the correct selection of data in a register such as using offsets with the register to work with arrays. Additionally, it should be noted that the registers work with both word and byte modes of operation, to ensure that they may be used correctly for any kind of operation.

Low Power Modes

In terms of the low power consumption, the MSP-430 Processor provides multiple modes of low power operation. Most of these deal with the shutdown of the Central Processing Unit's cores itself, and the details amongst the different modes are about shutting down different clocks available to the MSP-430. However, there is no need, in our design, to rely on clocks for our product to work. There is no timing required, nor the repetition of code after a given constant amount of time, nor even the need to time actions during operation, as the dispensing mechanism will use sensors to measure dispensed spices, and the motors for rotation and selection will rely on a measurement performed directly by the motors, rather than by measuring distance covered in a given amount of time. In other words, the most extreme of the power-consumption limiting measures may be taken and will allow for full operability of our design. This will allow for the majority of the power consumed by the device to be done so by the motors and sensors, rather than by the processor that will coordinate the operations between them.

Peripheral Information

Peripherals are connected to the Central Processing Unit using data, address, and control buses, and allow for full interaction with the CPU's instructions. This is ideal for us, as it permits full control of all peripherals directly from the CPU and only the CPU. As a result, we would only have to program the processor and would only have to control that to directly control the entirety of the apparatus with one program.

System Clocks

While, as already established, our design does not necessitate multiple clocks, nor timed interactions between peripherals, in the event that they are needed, or that a future decided upon feature requires one, the MSP-430 contains multiple system clocks that may be used, such as an auxiliary clock for general time measurements, a main clock to control CPU instructions, and most importantly for the purposes of our design, a sub main clock used to control peripherals. This all works with a basic clock module, which relies on four clock sources, a digital oscillator, a low frequency oscillator, a high frequency oscillator, and an interchangeable low/high frequency oscillator. Ideally, the low frequency oscillators will be used to ensure low power consumption, especially in an application wherein it is difficult to maintain the power supply, or where the device cannot be recharged once the device is shutdown. In the final iteration of the device, there was a Piezo Buzzer used to indicate a command being received, that triggered for one second using a timer.

Initialization Procedures

Upon performing a system reset of a development board, and thus a total reset of the processor, a few things must be noted that will be relevant to our design, in the event a reset must be performed due to a technical failure, or due to the device being disconnected from the power supply for any reason whatsoever. The things to note, in terms of hardware, are as follows: The General Purpose Input/Output pins will all be reset into the “Input” mode, thus necessitating that they all be reset to the appropriate mode (if necessary) prior to the activation of the peripherals to ensure no damage is done to the PCB board or the processor. Secondly, all peripherals and registers will enter an initialization mode, allowing for this set up to occur. Additionally, the watchdog timer activates, and finally, the Program Counter is loaded with the address contained at a reset vector. However, this device may enter automatically into low-power mode and shut down the CPU if the vector is set to the content of “0FFFFh”. In terms of software, the Stack Pointer must be initialized to the top of the RAM, the watchdog timer must be overwritten based on custom application requirements and peripherals must be manually configured as necessary by each peripheral. Further, if certain actions are to be taken based on the source of the reset, the watchdog timer, the oscillator, and the flash memory flags must be evaluated, to take appropriate action based on the hardware configuration of the CPU. Finally, there was an addition of a motor initialization process. After the MSP-430 is completely initialized, it enters a holding state where it begins sending turn commands to the motor until it detects a specific container, marked with a plastic marking, was found. Then, it will stop after a preset offset, and will be ready for use by the user.

Hardware Resets

The aforementioned initialization procedures must be undertaken when a user initiates a reset, via a reset button (if there is one present on the board), which activates the Reset/Non-Maskable Interrupt pin, or by disconnecting the power supply to the board. However, there may be an automatic reset performed by the hardware in multiple ways. These are as follows: A Flash Memory Access Violation may be detected, activating a specific flag for the CPU to detect. This flag is NOT reset when a system reset is performed, allowing for the identification of an access violation after the fact, and then taking preventative measures to ensure it does not happen again. Next, there is an Oscillator fault, which triggers when the crystal oscillator enters into an error condition. Similar to the Memory Access Violation, this triggers a special flag which will not be reset along with the rest of the board, allowing for identification of the issue and the undertaking of preventative measures. This is also true for the watchdog timer, which, depending on configuration, may set a bit flag upon initiating a system reset.

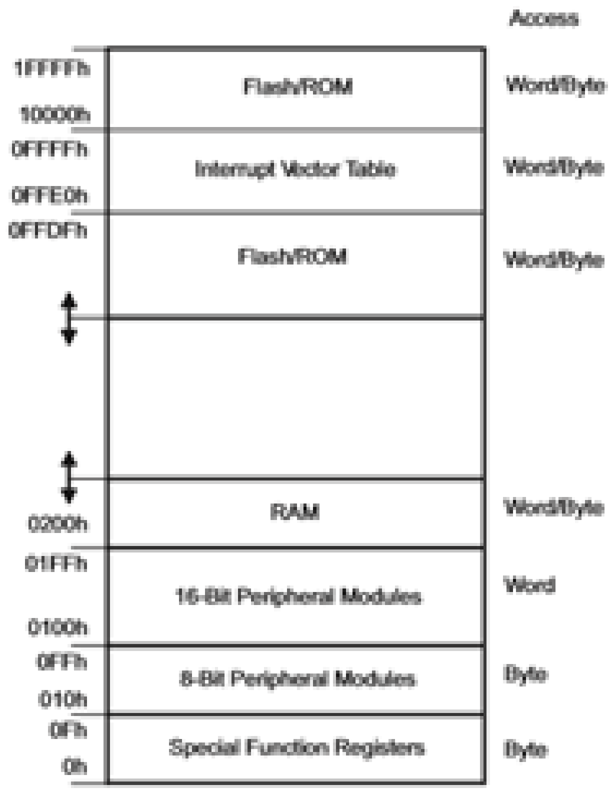


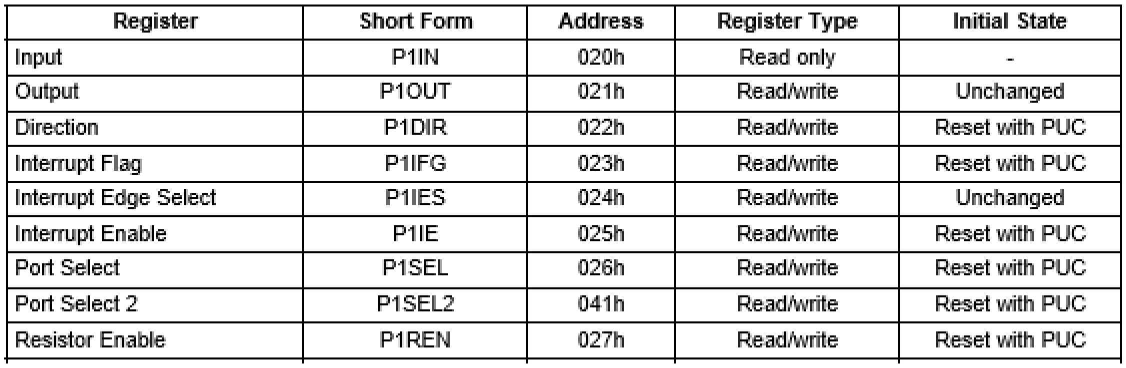
Figure 5: MSP-430 Registers

Memory

The memory of the unit is split into both Flash/Read Only Memory, and Random-Access Memory. Both of these can be used for both storing the code to be executed by the processor, and for the data the processor is working from and generating in-between operations. Further, the memory storage of words is only performed on even addresses, in the interest of providing sufficient space for both bytes of a word. The Flash memory is partitioned into segments, which are the minimum unit that may be erased by software at a time. However, writing may be performed in a single bit, a byte, or a complete word. It is impossible to read from or write to memory while it is being programmed or erased by the processor. Additionally, during an erase procedure, the processor stops executing instructions until the writing to memory is complete. However, this is only the case if the erase procedure is initiated from within the Flash memory. If it is initiated from Random-Access Memory, the CPU does not paralyze execution until the procedure concludes. Should the CPU then attempt to access memory while an erase is being performed, an access violation will occur and possibly, an associated reset or interrupt. Further, the result of the read will be completely unpredictable, thus making it worthless for the program's logic. Finally, a write or an erase can be aborted with an emergency exit bit.

DMA Control Unit

The Direct Memory Access Control Unit allows for transferring the data from one memory address into another memory address without requiring the CPU to intervene. This allows for reduced power consumption, as the CPU may remain off while the DMA remains active, but also increases the throughput of the peripherals connected to the CPU. Additionally, the DMA Unit contains its own flags, and allows for potential interrupts to be called upon in certain conditions. Further, these flags must be reset by software, not by hardware.



Register	Short Form	Address	Register Type	Initial State
Input	P1IN	020h	Read only	-
Output	P1OUT	021h	Read/write	Unchanged
Direction	P1DIR	022h	Read/write	Reset with PUC
Interrupt Flag	P1IFG	023h	Read/write	Reset with PUC
Interrupt Edge Select	P1IES	024h	Read/write	Unchanged
Interrupt Enable	P1IE	025h	Read/write	Reset with PUC
Port Select	P1SEL	026h	Read/write	Reset with PUC
Port Select 2	P1SEL2	041h	Read/write	Reset with PUC
Resistor Enable	P1REN	027h	Read/write	Reset with PUC

Figure 6: Pin States in the MSP-430

Digital I/O

MSP-430 processors have eight Digital I/O ports, and each of these ports contain I/O pins. Each of these I/O pins can be made to work in either an input mode or an output mode and may be made to trigger interrupts on either a rising edge or a falling edge. These interrupts are performed on a vector dependent upon the port of the pins. Also, they may all be configured so as to have individual oscillators.

Additionally, they may have pullup and pulldown resistors to detect the state of the pin in, for instance, a button. Also, the unused pins may be completely disabled to prevent a floating input that may be triggered by noise, and to reduce the power consumption of the device.

Supply Voltage Controller

The Supply Voltage Supervisor can set a flag or generate a full system reset upon detecting a voltage drop below a certain threshold. This threshold is completely customizable and may be set by the user to be any value, allowing for any number of applications and actions to be undertaken in certain conditions regarding the voltage drops. Further, a preset low voltage condition may exist when the input external voltage drops below 1.25V.

Watchdog Timer

The Watchdog Timer on the MSP-430 processor is present in order to induce a system restart after a set amount of time, or if a software problem occurs. Alternatively, should it be desired, it can be completely disabled. Attempting to illegally access this timer also triggers a system reset, to prevent software tampering with it. Aside from a system reset, it may also trigger interrupts, which allow for more customized responses to timers. Additionally, it is possible to link the watchdog timer to another timer, and then it becomes impossible to disable that timer in a low power mode, to ensure the Watchdog timer can keep working. It is also possible to, via software, stop the watchdog timer without disabling the timer itself, to ensure it can still run. When this occurs, the current value of the counter stored in the timer remains and prevents it from continuing until the software allows for it again.

Timer

Timers are present on the MSP-430, chiefly, an asynchronous 16-bit timer with a counter, along with 3 specialized registers for comparison of data. It allows also for performing operations requiring Pulse Width Modulation. There are multiple modes of operation for the timers, such as continuous mode, wherein an interrupt is generated once an interval completes, up/down mode for symmetrical pulse generation, among other mixtures and combinations of these. Attached to each of the comparison registers is an output unit which can generate the PWM signals. This output unit itself contains multiple modes of operation as well. For our project, these would be most useful to work as a customized watchdog timer, wherein rather than trigger a system reset, they may shutdown.

Serial Communication

The Universal Serial Interface Module allows for synchronous serial communication through an 8-bit or 16-bit register attached to an output data stream. This unit also allows for I2C communication and SPI communication. This was not be ultimately important for the prototype, but this feature allows for extensibility of the design. For instance, it would allow us to attach a display apparatus to the device, to ensure an optimal user experience. This screen could be used for spice selection or for modifying dispenser settings, among other things. Additionally, the MSP-430 supports a Universal Serial Communication Interface with a UART mode, which would allow it to connect to a computer. This means we could create a program that would perform troubleshooting and may detect errors on the device, a program that could be run by the customer from their own personal computer. This may also be formatted, in the interest of supporting multiple languages and perhaps even a custom Graphical User Interface based application. None of this is expected for the prototype, but this shows that the MSP-430's Serial Communication allows for plenty of expansion in the long term, should it be desired, of this product, to provide a much better user experience.

In the end, I2C communication turned into a major component of our implementation. All commands, from every source, are sent to the MSP-430 (which is the processor we picked), via an I2C bus. To prevent collisions, every I2C master device, connected to the MSP-430's slave, contains a hardware semaphore to prevent collisions between commands, and allow for a buffer of commands to be received by the MSP-430. Additionally, the MSP-430 is polled by the ESP32 chip as a slave, and sends forward a buffer of commands it has implemented multiple times every second, to keep every source of inputs in sync with each other.

C2000 Processor

Another processor considered for this project was the C2000 line of Texas Instruments processors. Where the MSP-430 is ideal because of its low power consumption, the C2000 is ideal for low latency, high performance operations. Then, it might seem unnecessary to have such a powerful 32-bit processor on this device, but in the interest of ensuring fast responses and a better user experience, the C2000 becomes an option. It would still be unnecessary to perform computationally taxing operations for the general operation of the product, but it might become ideal to have the device awake from sleep faster and ensure faster operation of peripherals.

General CPU Features

The C2000 processor is a 200 MHz 32-bit, Real-Time Operating System Central Processing Unit, designed for low-cost applications that require higher performance than a low-power Microcontroller unit. It contains multiple accelerators to optimize the performance of various operations performed by the CPU.

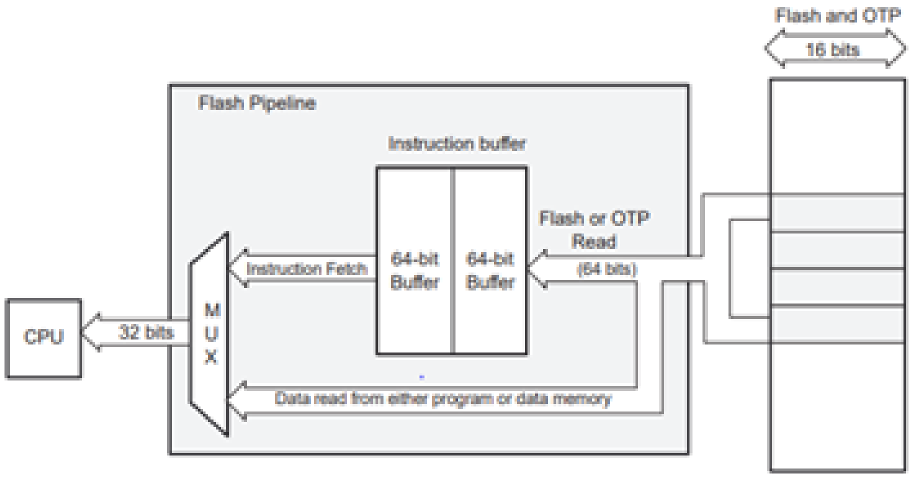


Figure 7: C2000 CPU Pipeline

Memory

The C2000 based microprocessor units, contain both Flash and One-Time Programmable (OTP) Memories. Both of these contain multiple states to optimize the power consumption and ensure that the CPU does not needlessly consume energy it does not require. The Flash memory is always enabled and contains multiple sectors that can be erased as groups, code security to prevent accidental flashing, wait sates, and a specialized pipeline mode that improves performance. These allow for instruction fetching at 32-bits, 16-bit program space read, and data space read in either format. The Flash memory can be accessed in multiple ways, such as random access and paged access. Further there exist registers that store the power and state information of the Flash memory. This means that the Flash may be analyzed and considered without having to actually poll or check the memory itself. There also exist registers for the OTP memory.

Memory Code Security Module

The Flash Memory is protected by the Code Security Module. This prevents access to the on-chip memory to unauthorized users, preventing the reverse engineering of proprietary code. This requires all code to be stored in secure memory, otherwise this allows for insecure code to call upon secured memory, thus revealing the contents of the secured memory. The CSM does NOT impact RAM, thus RAM may allow for code to be executed regardless of the security state of the device. Booting, peripherals, and Vector Tables are also not impacted by the security status of the device. To ensure security, once the CSM is enabled, the unsecured memory must be used as a stack, or security must be unlocked before a function call.

Oscillators/Clocking

Multiple clocks and oscillators are present, thus allowing for multiple timing mechanisms to function in low-power modes. These clocks can also be used to control peripherals. This means we can coordinate various components effectively, more so than with the MSP-430 processor. However, only the necessary clocks must be used to reduce needless power consumption.

Figure 1-37. CPU Watchdog Module

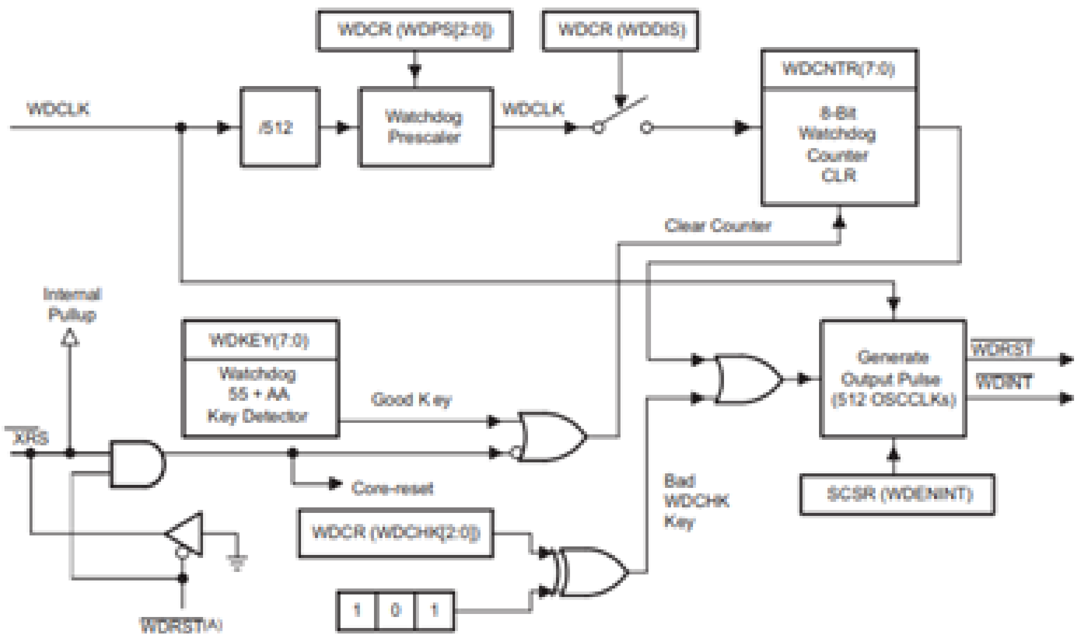


Figure 8: CPU Watchdog Module

Watchdog Timer Controller

Similar to the MSP-430 processor, the C2000 processor contains a Watchdog Timer. This timer allows for triggering interrupts and system resets as necessary by the system and application. This remains available even in low-power modes. It may however be suspended when the processor is suspended, to reduce power consumption, and it will operate as normal otherwise. This means that we can better control infinite motion caused by either a system error, or due to the user pointlessly and potentially damagingly commanding the device to do an endless motion. This is also possible in the MSP-430, but it will be better performed by the C2000, thus ensuring the limitations are reduced. In other words, due to the real time nature of the CPU, we can limit the aborting of the operation to the last possible moments, whereas the MSP-430 would require more time to do so, thus eliminating the possibility of reducing the time to stop the mechanism.

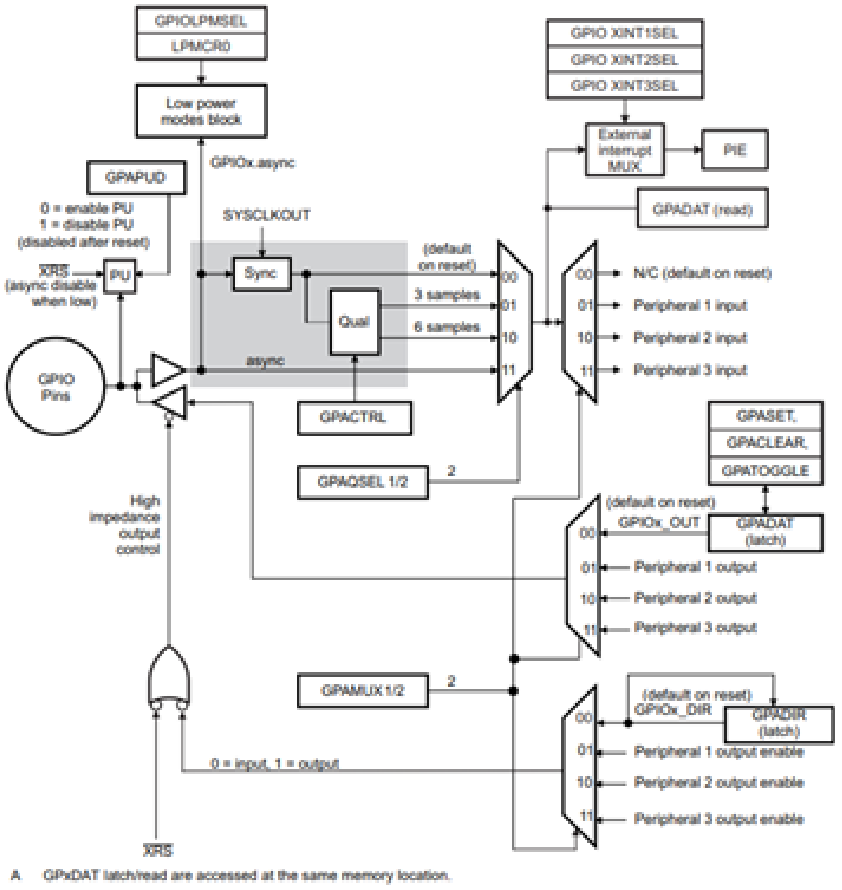


Figure 9: GPIO Pin Schematic

General-Purpose Input/Output

The general-purpose input and output pins, which may be configured for either input or output directions, contain a mode to remove unwanted noise. These pins may be controlled with specialized registers which set the direction of the GPIO pins, control the pullup and pulldown resistors, and access multiplexers. These must be individually customized and set in terms of directions and purpose, with each collection of pins able to throw an interrupt. Further, there are special digital general-purpose input and output control registers, which allow for setting or resetting the pin values. These pins thus would allow us to properly interact with all our peripheral components as necessary and would permit for the construction of our current design effectively.

Peripheral Interrupt Expansion

The Peripheral Interrupt Expansion Block allows for multiplexing of multiple interrupt sources into a reduced amount of interrupt inputs. This is functional because of the fact that the processor fetches the true source of the interrupt, using a table of vectors which contains the possible interrupt sources, and performs the relevant actions accordingly. This means that, despite the CPU’s limitations of handling a large number of interrupt sources, the design may still be assembled and made to function properly.

Pulse Width Modulation

The C2000 processor also contains an enhanced Pulse Width Modulator (ePWM) Module, which can better deal with Pulse Width Modulation problems. Chiefly, it permits PWM problems to be resolved with minimal CPU proper intervention, with a separate module dealing with the generation of the pulse modulation. This means we can, without sacrificing the low-power mode operation that would be normal with an MSP-430, and thus sacrificing the main advantage of it, perform PWM operations. These would be performed to control the rotator motors in such a way as to emulate slower, more precise motions improving the user experience with our product.

Serial Communication

The C2000 processor also contains an enhanced Pulse Width Modulator (ePWM) Module, which can better deal with Pulse Width Modulation problems. Chiefly, it permits PWM problems to be resolved with minimal CPU proper intervention, with a separate module dealing with the generation of the pulse modulation. This means we can, without sacrificing the low-power mode operation that would be normal with an MSP-430, and thus sacrificing the main advantage of it, perform PWM operations. These would be performed to control the rotator motors in such a way as to emulate slower, more precise motions improving the user experience with our product.

I2C Communications

Inter Integrated Circuit communications are also possible via an I^2C module, specifically with those peripherals that support NXP Semiconductor Inter-IC Buses. It however, lacks support for High-speed modes (Hs-mode). Regardless, through the use of I^2C communication, we can expand our microcontroller further, to directly connect expansion mechanisms, or perhaps even offer them to customers as necessary if they should desire them. This of course is a potential long term feature that could be performed once the prototype is done. That is, it would be beyond the scope of this project, but it would remain an attractive possibility for a potential future of this device.

In the end, the C2000 CPU was decided against, but its feature set would have allowed for a similar operation of the device as the MSP-430 albeit at a higher cost.

Raspberry Pi

Additionally, we used the Raspberry Pi 4 minicomputer in our design. The Raspberry Pi serves also as a microcontroller board by itself, and it runs a full Operating System in the form of Raspbian OS, a Linux Distribution. However, we are NOT using the Raspberry Pi as our main microcontroller for a few reasons:

- As per required for this prototype, the main microcontroller must be of a customized design, and a Raspberry Pi cannot be redesigned by a user. Therefore, it would not fulfill our requirements.
- The Raspberry Pi, running a complete OS, cannot be made to shutdown its CPU and still respond to interrupts. Or rather it cannot be made to do so in the limited scope of time for development we have for this prototype. As a result, should it be used as our main microcontroller, it would drastically increase our power consumption.
- The General-Purpose Input/Output pins on the Pi can be accessed primarily using the Python programming language, or the C++ programming language. However, the C++ method of accessing these pins has significantly less support than the Python implementation, thus meaning that our access to peripherals would be significantly slower, as Python is an interpreted language, rather than the compiled language of C++. Still, even if we could use C++, the MSP-430 CPU’s we plan on using for our main design, or even the C2000, are programmed using either Assembly, or the C Programming Language, both of which are noticeably faster than even C++.

However, despite these reasons, the Raspberry Pi still has a place in our design. That place is for the processing of visual data for control of the device using hand gestures. This is where one of the weaknesses of the Pi that was mentioned above becomes a strength. Whilst there are a multitude of drawbacks of Python, especially in performance, it is ideal for use for image recognition, owing to the large number of optimized libraries for such a task. As a result, the Raspberry Pi can be used as a secondary processor that will constantly receive input from an external camera peripheral, and, depending on the input of the camera (that is, if the Raspberry Pi recognizes a hand or any gesture the team decides upon), will send relevant data to the primary processor. Depending on this data, the processor will trigger the pouring mechanism, or the rotation mechanism.

Figure 10: Raspberry Pi

Raspberry Pi General Features:

- Broadcom BC2711 Quad-Core Cortex-A72 64-bit CPU, with a 1.5GHz Frequency
- 1GB-4GB LPDDR4-3200 RAM, depending on model (for optimal image recognition speed, we would opt for the 4GB model)
- Wi-Fi and Bluetooth connectivity
- USB ports (both 3.0 and 2.0)
- 40 – pin General Purpose Input/Output pins
- OpenGL Graphics
- Operating Temperature of 0-50 Degrees Celsius
- 5V supply via GPIO and via USB.

These are not all the features of the Raspberry Pi, but they are the ones that interest us for our design.

Raspberry Pi CPU:

The Broadcom BC2711 Quad-Core Cortex-A72 64-bit CPU will provide the optimal performance per dollar value for a minicomputer of this class. The Raspberry Pi has plenty of competitors, such as the more high-performance Latte Panda miniature computer, which is even capable of running the Microsoft Windows Operating System and thus serving as a complete personal computer for an average user. However, it comes at a price roughly 10 times more than that of the Raspberry Pi 4 at its most expensive configuration. As a result, this CPU will be fast enough to process our visual inputs, while being cheap enough to not drastically increase the cost of the system as a whole.

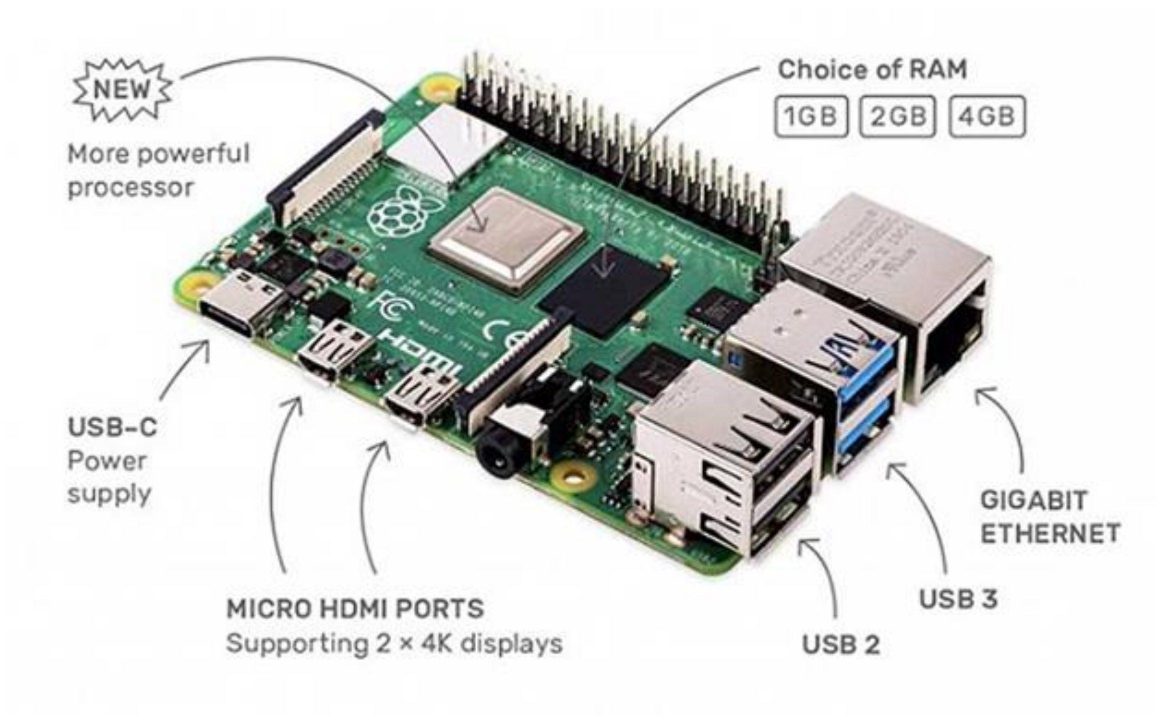


Figure 11: Raspberry Pi 4 Components

Raspberry Pi Random Access Memory:

The Raspberry Pi will run on a complete Linux Distribution, albeit a lightweight one, and as a result it will need to have enough RAM to both control the entire system, and to process the input it receives from the camera the Raspberry Pi will have attached. It should be noted, additionally, that the Raspberry Pi will not have a display attached, and as such will be configured to not render a Graphical User Interface for the Operating System. As a result, the amount of RAM consumed by the Operating System will be at the minimum point it can consume, allowing for the majority of it to be dedicated to the image processing.

Raspberry Pi Wi-Fi and Bluetooth Connectivity:

It is unlikely that the Raspberry Pi's Wi-Fi and Bluetooth connectivity will be used immediately upon activation of the device, but they may still be used in the long term. To begin with, should the image recognition library we rely upon be updated after the construction of our prototype, or of a final product, the Raspberry Pi's in-built Wi-Fi connectivity may be used to download an update, which will then allow for a more optimized performance of the device. Additionally, the Bluetooth connectivity may be used as a fail-safe feature, should our main Bluetooth chip connected to the primary processor fail. Should this occur, the Raspberry Pi's Bluetooth will then take over (after being turned on by a signal by the main processor), and use the Raspberry Pi as a middleman between the Bluetooth Connection and the primary processor, allowing for commands to still be received from the Bluetooth enabled mobile application.

USB 3.0 and 2.0 Ports

Much like the Wi-Fi and Bluetooth connectivity, it is unlikely the USB 3.0 and 2.0 ports will receive immediate use upon the activation of the device, but that does not mean they might not find a use. As previously mentioned, one of the features of both processors in consideration for the position of serving as our main processor for our microcontroller, is Serial UART communication. Because of this, we could configure our design to have the Raspberry Pi serve as a middleman between the main processor and the output the user will read, should it be necessary. Using the example of the previously mentioned diagnostics program, that could be developed after the finalization of this prototype, the UART communication could be connected to the Personal Computer of the user via the USB ports of the Raspberry Pi, which will already have to be near an edge of the device, to allow for the camera to have a clear line of sight towards the user.

Pin General Purpose Input/Output Pins:

The Raspberry Pi contains 40 Input/Output pins which may be used to connect peripherals. These pins would not be used for the selected external camera but would remain available for connectivity with other peripherals that require the full powers of an Operating System, which our primary processor cannot provide. Despite this room for growth, the Input/Output pins will serve as our connector between the main processor and the Raspberry Pi, allowing data to be transferred between one and the other. For instance, it was originally expected there would be a dedicated wire for leftward motion, one for rightward motion, and a pouring mechanism trigger wire, but in our final implementation, we opted for using an extended I2C bus to allow for communication between the Pi and the processor, all the while using prespecified numerical codes to codify the instructions.

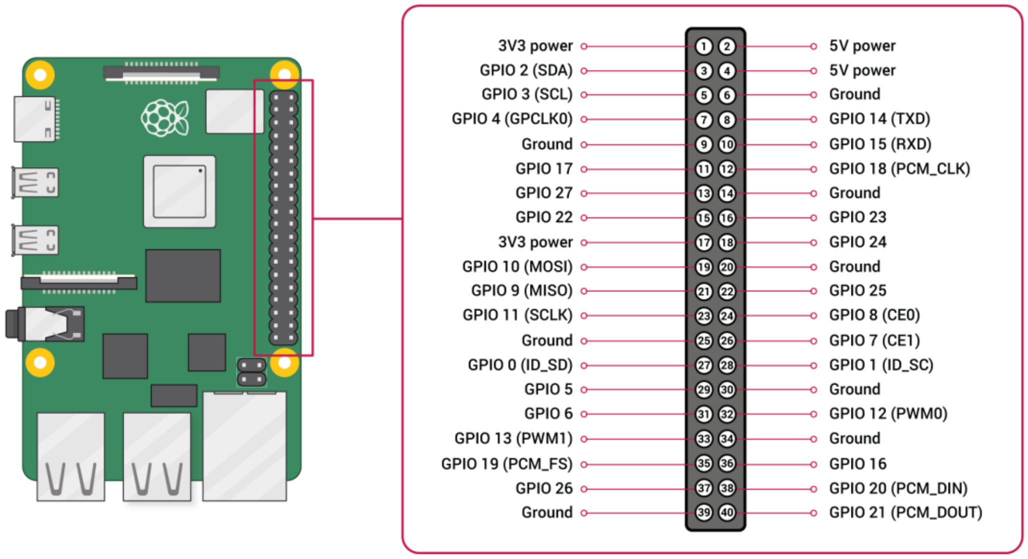


Figure 12: Raspberry Pi GPIO Map

OpenGL Graphics:

The Raspberry Pi's Graphical Processing Unit (GPU) works with Open GL Graphics, and, when the Graphical User Interface is active and there are available connectors, can power two 4K Resolution Displays at 60 frames per second. Of course, as previously stated, we will not be rendering a Graphical User Interface for our design, as there will be no screen wherein to display the output of such a GUI. This having been said, the Graphical Processing Unit and the OpenGL graphics capability can instead be used to help power the image recognition software we will make, to increase the performance of the device, as well as its accuracy. Further, should we later decide to include a display in a second iteration of the device, the Raspberry Pi should be able to power it without many problems.

Unfortunately, in our implementation, the OpenGL graphics, while useful for implementing the image recognition software, did not provide satisfactory performance. This was remedied using a Google Edge Tensor Processing Unit Accelerator, which can be connected via USB, and a TensorFlow Lite model compiled for said Edge TPU Accelerator. This combination gave us sufficient performance.

Operating Temperature 0 – 50 Degrees Celsius

The Raspberry Pi may function at a range of 0 to 50 degrees Celsius, or 32 to 122 degrees Fahrenheit. This means our device will remain operational at these temperatures, as will our visual recognition capabilities. Since it is expected the device will be used in a kitchen inside of a home or an apartment or a similar residence, these temperatures should be a more than sufficient range for most use cases. Additionally, should it somehow become possible to connect the device in an outdoors environment, it will also be sufficient for most conditions.

5V Supply Via GPIO Pins or USB Ports

Since our camera will not be connected via General Purpose Input/Output pins, nor via USB ports, the 5V DC power output will not be relevant for it. However, it will allow us to provide power for other peripherals, allowing us to bypass the requirement to connect potential peripherals that need to be near the outside of the device, without having to provide a wire that connects to the main microcontroller device.

Raspberry Pi Software

The Raspberry Pi works using the Python Programming Language as a building block for our hand recognition software. Chiefly, we used the OpenCV library, a library for Computer Vision applications, as the base of our code. We will also use the matrix-based library, Numpy, to perform analyses upon the data collected using OpenCV. In essence, our code will function as follows:

The camera will be turned on and will constantly be feeding data to the program. This data will be stored only as needed for processing, and will then be discarded, to not necessitate much in storage. The image will be analyzed to detect a hand, in any state (that is, in any position or shape). This hand will then be tracked in an attempt to detect the desired gesture. For instance, an open hand moving left will rotate the machine clockwise, whereas an open hand moving towards the right will rotate the machine counterclockwise. Other than this, a downwards moving gesture of an open hand will trigger the dispenser and will serve spice until the hand is closed or it is moved away from its position. In other words, the code will be centered around understanding the image, and sending the correct signal back to the primary processor for the correct action to be undertaken.

The above paragraph was our original plan, at least. In practice, the hand motion tracking proved too noisy and unstable for practical use, and was replaced with a hand gesture only recognition. Regardless of the state of the movement of the hand, if the appropriate gesture is detected, the relevant command is sent over to be executed by the processor.

The Raspberry Pi will work with the Ubuntu OS Linux Distribution for the ARM CPU architecture, in the Server configuration, and will be kept as minimal as possible. In other words, the Raspberry Pi will be configured as if it were a server, with no GUI and no other functionality, than the ability to execute the Python Program and the installation of the relevant libraries. However, the internet connectivity may be used for potential updates of the system, such as code optimizations, or updates to the libraries that are required for the code to function.

While the Raspberry Pi Ubuntu server configuration was used in early development, it became difficult to use the more Raspberry Pi specific libraries with it, and Raspbian OS, the default OS of the Raspberry Pi, was reinstalled. It was then maintained until the end of the project. It may be reconfigured to have a server mode, and to disable the GUI for better performance, but in practice, with the use of the TPU accelerator, the GUI being active or not has little to no actual impact on the performance of the image recognition software. This is because the GUI is powered by the GPU on the Raspberry Pi, while the image recognition software is powered by the external TPU.

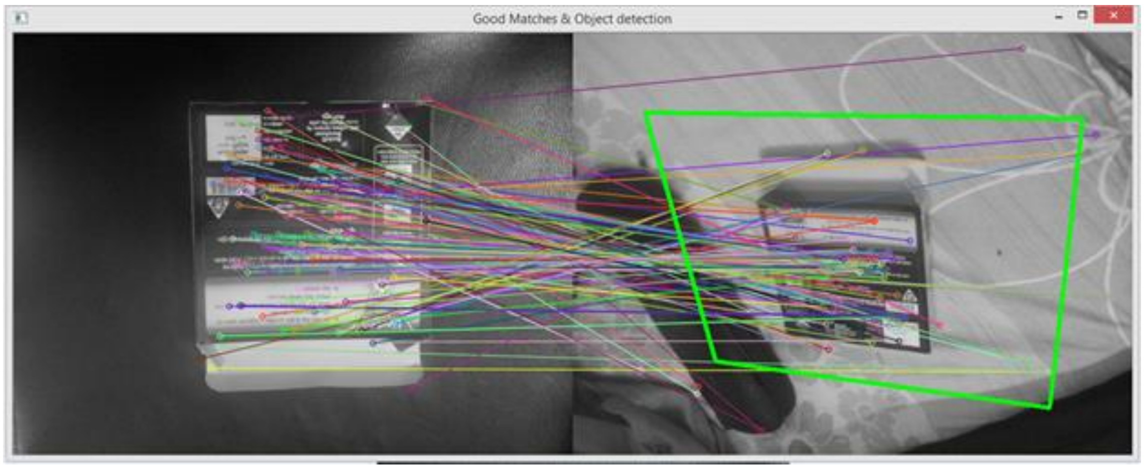


Figure 13: Example of Object Detection Software with OpenCV

Raspberry Pi Camera

For the Camera component, we utilized a Unistorm 1080p resolution webcam camera, with an OV5647 sensor module. This camera will allow us to record video at either 1080p at 30 frames per second, or at 720p at 60 frames per second. However, for performance reasons, it is likely that we will do neither of these options, but should they be required, they are present. Instead, we will likely work at 480p, where the camera may record at 90 frames per second, but we will set it to record at 30 frames per second, or rather, we will only consider a frame after 3 frames are recorded. This has to be done because the Raspberry Pi by itself will not be able to perform many computations at its optimal condition, due to it being inside of a greater structure and thus being unable to dissipate heat as efficiently as it would in normal operating conditions. Additionally, even in optimal conditions, the Raspberry Pi would also struggle to process 90 frames per second of input, and even if it could, it would not be necessary to do so to catch a gesture being performed by a user.

This camera is built specifically for a Raspberry Pi, and it has compatibility with the Raspbian Operating System, which will ensure there is a minimal chance for error. Further, it means there will be no need for the creation of a software solution that will serve as a middleman between our Python program and our camera.

Additionally, the camera contains an Infra-Red sensor and Infra-Red lights, to allow for visibility in low light conditions and high-light conditions. The image wouldn't be of a particularly high quality, but it will be sufficiently good that a hand can be detected.

Figure 14: Unistorm 1080p OV6547 Webcam

Raspberry Pi – Processor Communication

It is crucial for our Raspberry Pi and our custom microcontroller's Central Processing Unit to communicate effectively and with minimal error. This communication will likely be monodirectional, as there will be very little to no data that will have to be sent from the processor to the Raspberry Pi's systems. The Raspberry Pi serves as an input, not as an output, and any output that has to be done to the Raspberry Pi will be self-contained, such as an over the internet software update that will affect the Raspberry Pi's functionality. This is subject to change over the lifetime of the product, should it become more than a prototype, as perhaps the processor might trigger peripherals on the Raspberry Pi that are yet to be determined. Still, for the initial construction of the project, there will be little to no communication between the Raspberry Pi and the main processor, in the direction of the processor to the Pi.

For the monodirectional communication that will be present, however, there is was be dedicated wiring for every input signal that will be sent from the Raspberry Pi to the main processor. These signals include the rotation and dispensing mechanism triggers, and they will cause an interrupt of the main processor's operation. These will be sent as Logical High or Logical Low Voltage outputs and will each trigger unique interrupts. It is likely that the wiring will be placed inside of a secure compartment of the device, allowing for it to be safe from the movement of the device.

In the end, there was no dedicated wiring, instead a single I2C bus was used to send data between the Raspberry Pi and the MSP-430 CPU. This allowed for simpler wiring, and for more software based communication (where the message being sent is more important than the means through which is sent).

Raspberry Pi Implications

The Raspberry Pi will have the implication of increasing our power consumption significantly, as it will not ever be in a standby mode and will always be drawing current from our battery. Additionally, because of the fact that the Raspberry Pi's Central Processing Unit is significantly faster and more powerful than the one of the MSP-430 or the C2000, the power we save from setting either chip in a low power consumption mode will be offset by the Raspberry Pi. Now, we could of course attempt to program the Raspberry Pi to function using interrupts to lower power consumption, but this will not shut down the CPU of the Raspberry Pi and thus still consume a disproportionately large amount of power when compared to our main CPU.

Additionally, it should be noted that there is likely to be a camera cover for the privacy concerns of the user. While there will be no direct recording of a user's actions, should it become necessary to do so, there will be no saving of that recording outside the Raspberry Pi. Still, understandably, users will be likely to distrust our honesty in that statement, so a camera cover would alleviate potential fears in regard to that. In the final implementation, however, due to manufacturing constraints, the camera cover was not implemented.

The Raspberry Pi, it should be noted, lacks a proper "low-power" mode, unlike our main processor. Because of this, it needs a dedicated power source, and will have to depend on a switch to turn it on or off. This is because if we do not do this, then the Raspberry Pi will consume its entire battery in a single use, especially since we are attaching a camera. However, doing this presents a separate problem, that the Raspberry Pi takes roughly 30 seconds to complete its booting process, and the user will likely expect to flip the switch and begin use of the product immediately. Because of this, we will need to perform some customization to the kernel of the Raspberry Pi Ubuntu OS, to ensure a speedy booting process, for easy comfortable use of the device.

In practice, for our model, it became impossible to use this switch or batteries, since, due to COVID-19, we couldn't exchange the parts appropriately. That being said, the circuitry was designed so that it could work without the Raspberry Pi being in use, and with just the main CPU being connected to a power source.

Raspberry Pi Battery

To power the Raspberry Pi, we would have used 3000 mAh 18650 batteries, which would be connected in parallel and power the Raspberry Pi. It is estimated that these would provide around an hour of use per battery. This means that the Raspberry Pi would not be able to remain in use for extended periods of time, without requiring the user to either recharge or replace the batteries, thus requiring us to rely on a user turning the Raspberry Pi on or off manually via a switch that will be present on the chassis of the device.

Another alternative is to use a mobile phone power bank, as both the Raspberry Pi and most modern smartphones work using USB-C charging. One such power bank we considered would be the Charmast 26800 mAh portable power bank. However, this option presents a few drawbacks. Chiefly, it would be significantly more difficult to fit it into the chassis of the device. The 18650 batteries may be covered in a special compartment that only exposes them once opened, but doing the same with a flat power bank would be much more difficult, due to its size, shape, and having to provide the ability to remove such a power bank for recharging. Secondly, in terms of design, it isn't good to use an already existing complete product as a part of our own product, both for ethical reasons, and for practical reasons; if there is an as of yet undiscovered problem in their design, it will be carried over into our design as well, and will likely remain there until their product is updated, at which point the shape might change requiring a redesign of our entire product. Meanwhile, the 18650 batteries would be more consistent, as they are mere batteries and contain no outside design. Additionally, they would be cheaper to replace, should there be any damage done to them, such as if, for instance and purely hypothetically, our water resistance were to fail. Overall, it seems using the batteries, and instructing the user to conserve the energy of the Raspberry Pi is the best approach, even if a power bank would power it for a longer period of time.

In our final implementation, batteries were not used and two dedicated cables were used to plug in both the Raspberry Pi and the main circuitry, although it should be fairly simple to incorporate them in the future.

Figure 15: 18650 Batteries

MSP-430 to Raspberry Pi Communication:

The MSP-430 relies on General Purpose Input and Output pins for communication with peripherals. Luckily, the Raspberry Pi mini computer works in the same way, and also contains GPIO pins. This means, for the most part, the MSP-430 and Raspberry Pi can both communicate with each other without much problem and without requiring much in terms of dedicated I/O.

The Raspberry Pi provides triggers to interrupts in the MSP-430 code, via simple wiring that will connect a GPIO pin on the Raspberry Pi to a GPIO pin on the MSP-430's board. For example, the Raspberry Pi will detect a hand gesture requesting clockwise rotation, and a single GPIO pin will trigger Logical High, and will then send through the wire that to a GPIO pin on the MSP-430 which will trigger the clockwise rotation interrupt. This will also be the same for counterclockwise rotation, albeit on a separate set of GPIO pins and wiring. The GPIO pins could be extended using an adapter, for better compatibility with the MSP-430 and with reduced risk of exposing the Raspberry Pi.

This being our original idea, in practice a single I2C bus was used to achieve the desired result, which simplified the wiring.

Figure 16: GPIO Pin Extender For Raspberry Pi

However, while this is sufficient for the core functionality of rotation and dispensing of spices, it still leaves the Raspberry Pi with much potential. Chiefly, it is considered to use the Raspberry Pi's SD Card for onboard storage of the condiments list, so that the device can be operated when the mobile application is not available, such as in cases where the user's smartphone does not have battery or is too distant from the kitchen to be used. For this, we plan on using a third section of GPIO pins that will be connected to UART, in order to transfer data to the Raspberry Pi for storage, and to trigger special commands on the Raspberry Pi, such as activating the bluetooth receiver in order to replace the MSP-430 peripheral, should it be damaged or necessitate a replacement. Alternatively, as a separate plan, it is also possible to rely on both bluetooth devices to allow for wireless communication. This will allow us to store the data perfectly well without requiring a new set of wires and GPIO pins.

It should also be noted that this will necessitate the creation of a special program to manage the Raspberry Pi’s Filesystem, and it will then control access to this section of the filesystem and will update the relevant file and read from it when the device is first turned on, should the Bluetooth connection fail. This can be accomplished with a Python script to be invoked upon booting the Raspberry Pi, which will parse a file if it already exists, or alternatively will create it should it not. Then, upon changes to the list of spices, the Raspberry Pi will update the file, and keep writing to it for the duration of the operation. Thus, we can accomplish a simplified secondary storage solution using the same communication methods between the MSP-430 and the Raspberry Pi.

In practice, the ESP32’s eeprom was used for storing the information of the spice layout of the device, as this was ideal for updating the mobile application that needed to visually represent the layout, while every other device did not.

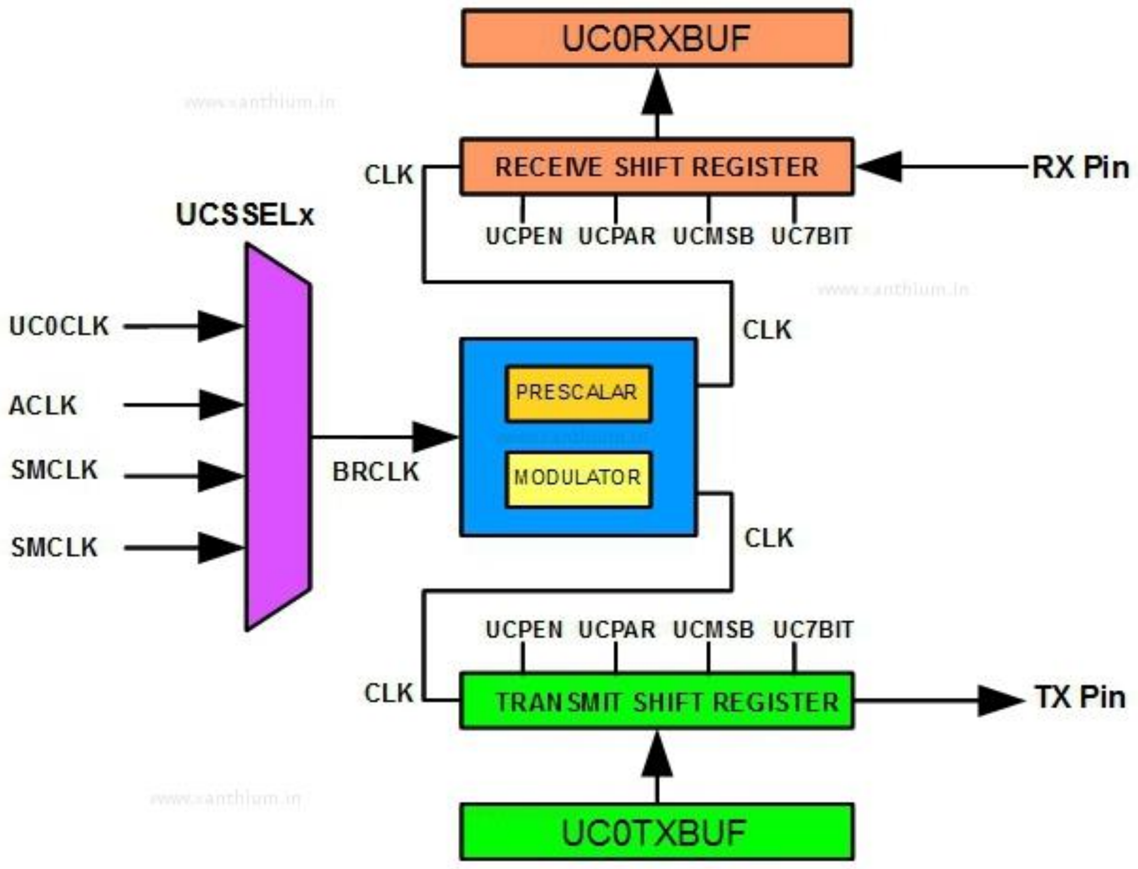


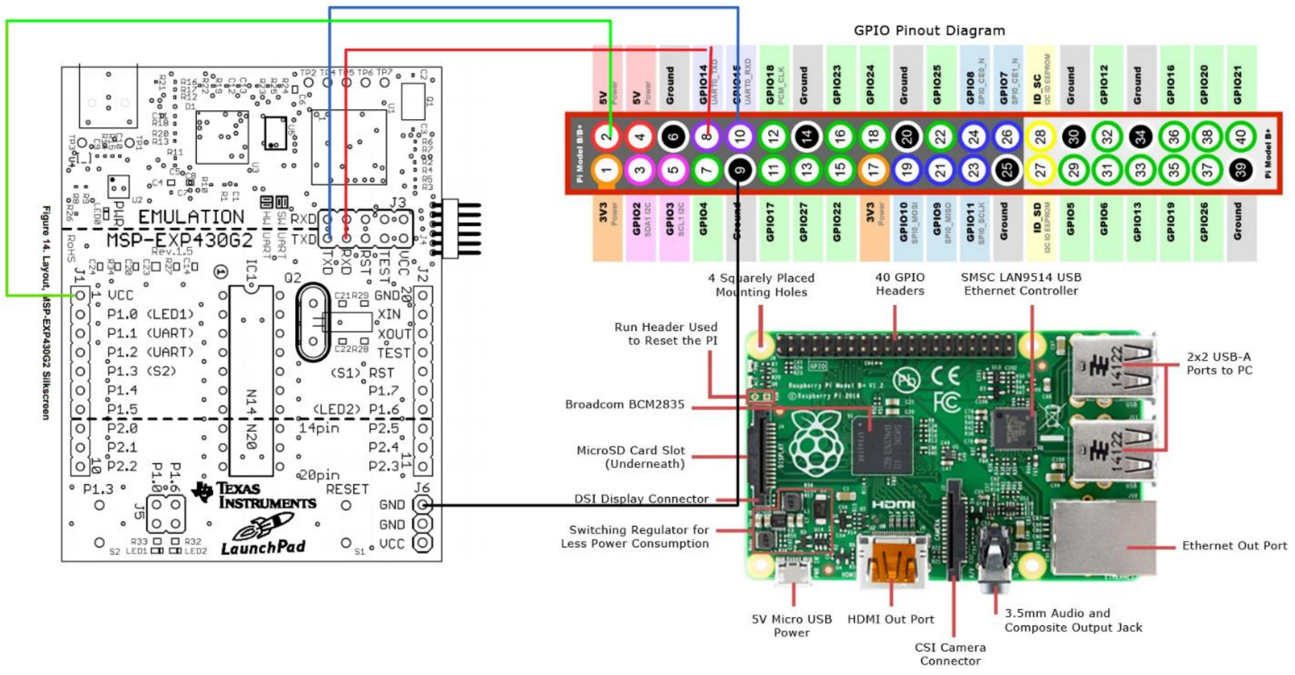
Figure 17: UART Communication in MSP-430

Integrating the Raspberry Pi

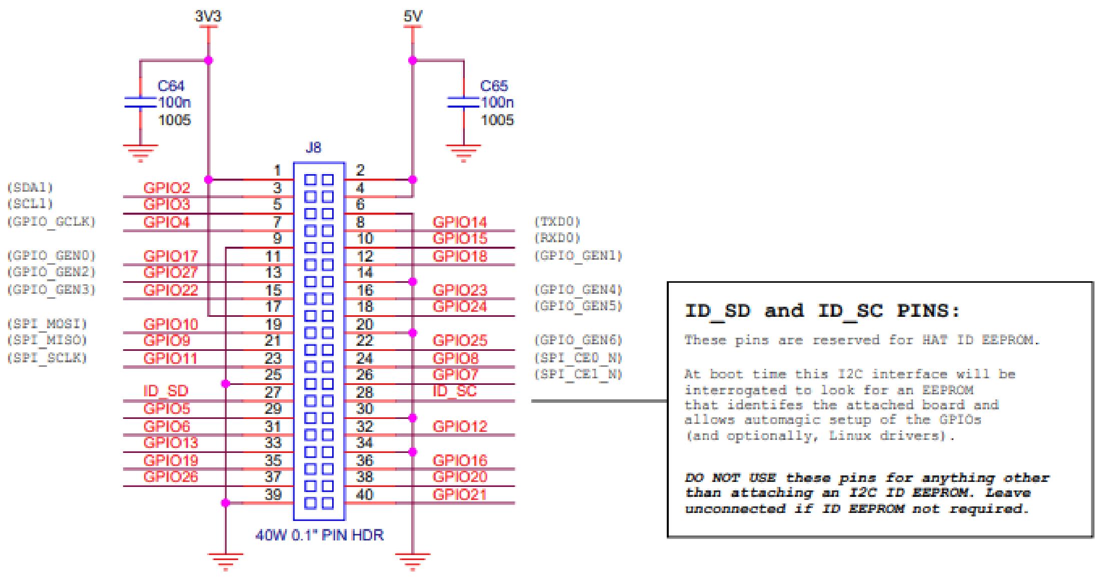
Overview

In this section the integration of the raspberry pi and the microcontroller will be discussed. What data will be transmitted will also be viewed.

Schematic and integration


GPIO Pinout Diagram
Figure 14. Layout, MSP-EXP430G2 Silkscreen
EMULATION
MSP-EXP430G2
RXD
TXD
4 Squarely Placed Mounting Holes
40 GPIO Headers
SMSC LAN9514 USB Ethernet Controller
Run Header Used to Reset the PI
2x2 USB-A Ports to PC
Broadcom BCM2835
MicroSD Card Slot (Underneath)
DSI Display Connector
Switching Regulator for Less Power Consumption
Ethernet Out Port
5V Micro USB Power
HDMI Out Port
CSI Camera Connector
3.5mm Audio and Composite Output Jack

This schematic shows multiple connections to the microcontroller. These are needed to supply power and other information. The Raspberry Pi requires a five-voltage output to run, it is recommended to get to six or seven to account for lost power. The microcontroller outputs around five and a half volts to supply that power requirement. For communication between the Raspberry Pi and the micro controller the UART ports are connected. Note that the transmitter port from the microcontroller is connected to the receiver port. This is so that when the microcontroller transmits a message the Raspberry Pi receives it. If the wires were inverted, then the transmission would fail because none of the connections would be able to read the transmission.



ID_SD and ID_SC PINS:

These pins are reserved for HAT ID EEPROM.

At boot time this I2C interface will be interrogated to look for an EEPROM that identifes the attached board and allows automagic setup of the GPIOs (and optionally, Linux drivers).

DO NOT USE these pins for anything other than attaching an I2C ID EEPROM. Leave unconnected if ID EEPROM not required.

GPIO EXPANSION

The microcontroller is sending signals to the Raspberry Pi to tell this to do different implementations. There is a motion sensor attached to the spice rack. This is for gesture sensing that is monitored through the Raspberry Pi. This was monitored. The microcontroller sent the respective signal to tell the camera to be on. The camera function is through the Raspberry Pi. The microcontroller sends a signal through the UART ports. With these connections, the integration of these devices make the spice rack an innovative device that helps to reduce clutter with spices in the kitchen.

Realistic Design Constraints

All engineering projects face design constraints when being designed and made in the real world. Our product is no exception to this eventuality. We list and consider the relevant design constraints below.

Economic Constraints

Due to the lack of outside funding, the project was fully funded by the individual group members. Thus, the budget for the project was rather limited. This means that the design was greatly affected by our funding, especially when it comes to the selection of the parts for the design and which features that we choose to include in the design.

We also wanted to limit the cost of building the prototype to provide a more affordable solution for the market. While the non-automatic solutions are very affordable, with prices around $25, due to the lower level of technology and cheaper plastic construction with these devices, we likely cannot match this price point as a budget for our prototype.

In contrast, the TasteTro system costs $300[3], so we wanted to have our prototype to cost around this amount to be built. The feature set of this solution and our project is more comparable than the more affordable solutions in the market, so we believe that our prototype should be similarly priced. This makes our project more attractive to our target users if it were put on the market, as it would be an affordable spice organization solution with a rich feature set. Thus, we make it a focus to create our prototype as affordably as possible to achieve this appeal to our target user base.

Time Constraints

This project is done as part of our computer and electrical engineering degree curricula, as mandated by the University of Central Florida and the Accreditation Board of Engineering and Technology (ABET). Therefore, the project's timeline is directly related with the class schedule of the University. This gives our group a definitive deadline to complete the project. At the time of writing, the date for the completion of the project has been published as July 21, 2020, but we estimated that the deadline would be around July 14, 2020, two weeks before the University's commencement ceremony for the summer semester[4]. We plan on finalizing our design and procuring the parts for our project by April 27, the last day of the spring semester.

After procuring the parts, we spent the rest of the available time prototyping, keeping in mind the time we have left in order to adjust our project's feature set and specifications accordingly, if the need arises.

Environmental, Social and Political Constraints

Our project seeks to address an issue with a very small and local scope in terms of environment impact and effects. One’s choice in selection and storage of herbs and spices likely has little impact on the environment at large. Therefore, our project’s design faces little to no challenges in conforming to proper environmental considerations and procedures. The materials we will use are not very polluting, and neither are our design and manufacturing processes, which means we will be able to, should our prototype design be expanded upon and become a complete market product, manufacture a vast quantity of this product without having to have much concern for environmental impact.

Our project also faces some constraints of a social nature. Chiefly, it is unlikely it will be well received by the culinary community, since our product appeals more to home users rather than for a chef in a restaurant. This is important because our audience will have different needs and desires than those of a person cooking in a professional capacity. Simply, they require much more precision than any device is able to give, due to the implicit latency of mechanical components reacting to inputs from the user. That is, if a chef requires a specific amount of spices for a dish, using an automated system to dispense them makes it significantly more likely that they will overuse or underuse the relevant spice, when compared to their use if it was done with human measurements. Then there is the fact that professional cooks use more varied spices than are used in a home environment, and use them more frequently, even when accounting for the extended time they spend cooking. Simply put, our device would not be able to hold all the spices they require, and using multiple devices for all their spices would be inefficient, both for the aforementioned reasons of precision, and for the delay in time they’d have from searching for the appropriate device with the correct spice. Because of this we must be aware that, while improvements in design will allow for a better suited device for these types of individuals, the more beneficial improvements lie in making it easier to use for home users and unprofessional users. Indeed, it is likely the main audience for the product lies in people interested in technological novelty, over those interested in cooking efficiency.

Then, there is the admittedly limited political impact of our product. Being designed for home use and for a limited market, it is unlikely it will have much of a personal political impact, that is, that a politician will take notice of the product itself. The product is inoffensive, and has no political impact whatsoever. However, there is possibly a political impact from the manufacturing processes. The manufacturing is mostly automated in both plans of construction. Should the wood cutter be used to achieve the desired shapes, or should a 3D printed shell be used for the exterior of the apparatus, then there would be minimal need for human input in the assembly. Indeed, it would be mostly limited to assembly, which could even be performed by the end user, should the design be simple enough in practice. This could have a political impact, should the product be successful, since it would have potential to create manufacturing jobs that are being automated from the beginning. Still, given the limited audience for our product, it is unlikely to be singled out for this reason, and would, in the worst case scenario, be lumped in with other products of similar construction instead.

Ethical Constraints

While our project does contain very innovative aspects in its design, it does also share some functionality with existing products in the market. Therefore, as responsible engineers and professionals, we have the ethical responsibility to ensure that our design does not infringe upon the intellectual property of products and solutions that are already available in the market, or that have already been patented or otherwise legally protected.

Another ethical concern that our design must address is safety of use. With the incorporation of electrical components, our design will be made so that no wiring is exposed, and the risk of electrical discharge to the user is non-existent. The mechanical and kinematic aspects of our design, such as the turning mechanism for the capsules when the user selects a spice, will also be made to minimize any risk of injury. The turning mechanism will not be exposed, and the turning speed will be tuned so that it is slow enough that it cannot cause damage to the user.

Lastly, some of the features that we want to implement into this project will require the collection of data that may potentially be personally identifiable. For example, for voice-activated control, the mobile application would need to record the user's voice to recognize and execute the command and select the spice that the user wants. Since our project will involve the handling of potentially sensitive information, we must keep relevant privacy concerns in mind in our project's software design. None of the recorded data (audio for voice commands and video for gesture controls) will be recorded on any non-volatile memory on the project's hardware. This means that none of this data is stored once the device is shut down. The data recorded will only be sent to any third party if it is required to process this data. The data will not be shared to any other party, but will be processed locally to the fullest extent possible while keeping as much functionality as intended for the device.

Health Constraints

Our project deals very closely with food consumption and food preparation. Thus, we must adapt our design to accommodate for these health-related concerns. We will strive to make the parts of our design that make physical contact with the spices, namely the spice capsules and the dispensing mechanism, able to be cleaned. However, this would greatly increase the difficulty of the mechanical design of the device.

Outside of the implicit requirement for hygiene in the cooking environment, it must also be considered that the spices will come into direct contact with the materials from which the product is constructed. In other words, should any of the materials used for the product be revealed in the future to be toxic, then the entire design would have to be redone with different materials, and the existing products would likely have to be recalled, for the health and safety of the end users. Our primary design plan has further constraints in this section, as it is made of wood and painted with a black finish. This is due to two reasons: First, the paint used for the wood may be the most toxic component in the entire product, and as such must be carefully selected to ensure it is able to come into contact with food. Secondly, the wood must be of high quality and carefully worked, to ensure no loose pieces or splinters mix with the food, which could cause much harm to the end user.

Manufacturability Constraints

Our project's design is constrained by the manufacturing facilities and technologies each group member has access to as individuals. Since we are not currently affiliated with a company or industrial-scale manufacturer, we do not have access to many kinds of advanced manufacturing tools we may use for the project. This greatly affects our design, and we respond by making our design simpler and more manageable to be manufactured, especially in regards to the mechanical aspects of our design, as most of our team members have very limited knowledge on mechanical engineering or physical manufacturing. This will manifest practically through the use of cheaper and more commonly available materials for construction, like using plywood for most of the rotating base of the spice rack, and foregoing aesthetic requirements, as they likely will require more advanced manufacturing to create the custom outer components necessary to achieve them.

Sustainability Constraints

With the grave developments of climate change happening in an increasingly frequent fashion, we must be more vigilant and careful with our design's sustainability impact than ever before. We therefore took some of these concerns into consideration when developing our design. The main impact on sustainability that our project would have in use would be its consumption of electricity. We seek to keep this consumption low, for sustainability reasons and for other design purposes, we will seek to design our device to use as little power as possible.

With this in mind, we also included photovoltaic cells, colloquially known as solar cells, in our design to address some of these sustainability concerns. While the photovoltaic cells will likely not provide the necessary power for the operation of the device on their own, they can passively power the device to some extent, thereby lowering the environmental impact the operation of the device has on the environment.

Project Hardware and Software Design

Through this section we will discuss the various components that go into our system, including Hardware and Software designs.

Initial Design Architectures and Related Diagrams

Through this section we will show the architecture of the Microcontroller selection and show our hardware flowchart. The flowchart shows the significant parts needed for the Spice Rack design to meet specifications and objectives.

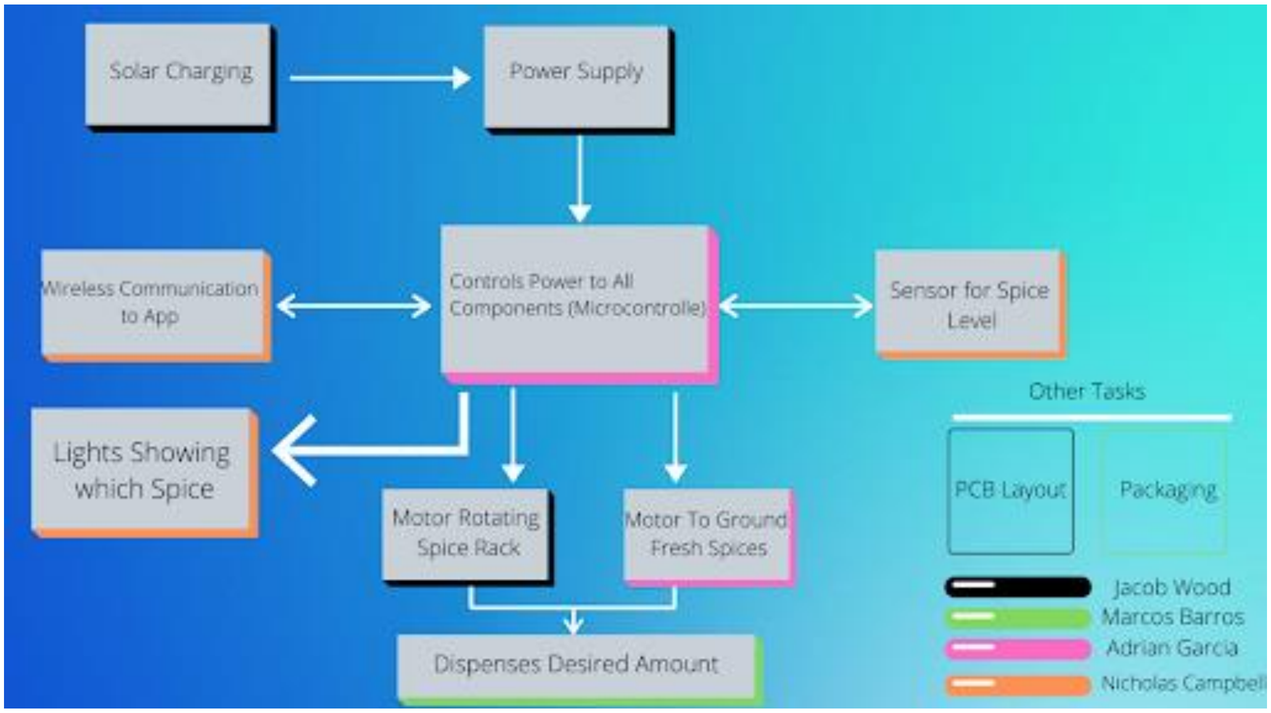

Solar Charging
Power Supply
Wireless Communication to App
Controls Power to All Components (Microcontroller)
Sensor for Spice Level
Other Tasks
Lights Showing which Spice
Motor Rotating Spice Rack
Motor To Ground Fresh Spices
PCB Layout
Packaging
Dispenses Desired Amount
Jacob Wood
Marcos Barros
Adrian Garcia
Nicholas Campbell

Figure 18: Hardware Design Flowchart

IR flowchart

This section will go over how the IR will be implemented. This is the current plan for what happens with the IR and how it works with the Bluetooth.

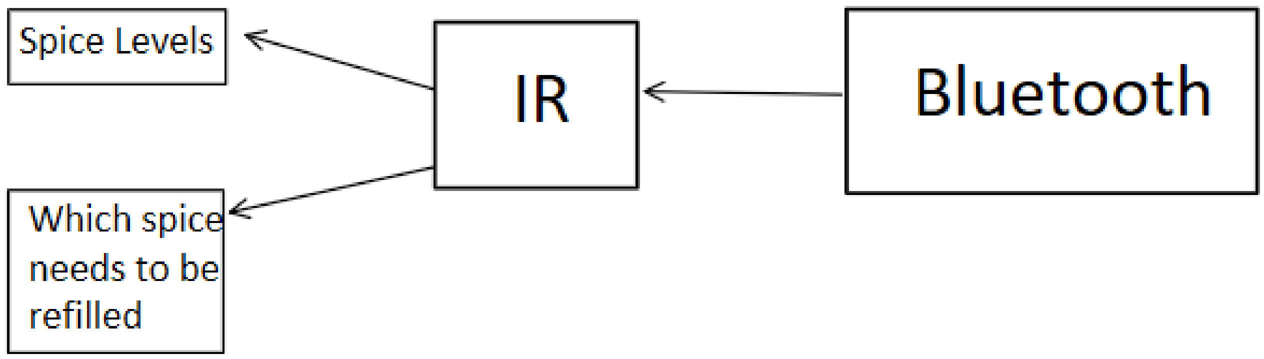


Figure 19: IR Flowchart

This flowchart displays how the IR sensor interacts with the Bluetooth device. There are three light sensors per pod; it is important to keep track of what the current level of each spice. The spices report individually to the Bluetooth device. With this implementation of the IR light sensors it were easy for the Bluetooth device to know which spice at the appropriate level.

Bluetooth flowchart

This section will go over the how the Bluetooth will be implemented. This is how the Bluetooth device on the spice holder would function with the user.

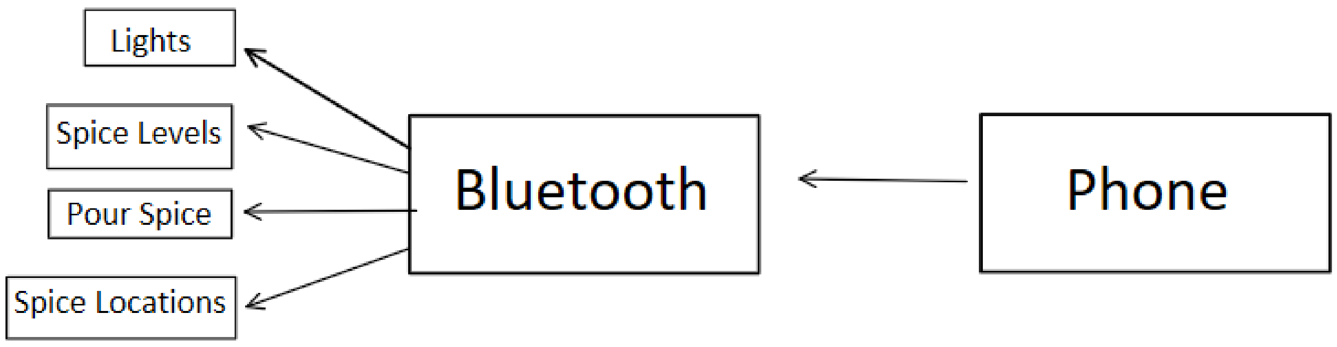


Figure 20: Bluetooth Flowchart

In this flowchart the Bluetooth function is explained. This function is to give the user the ability to communicate with the spice holder wirelessly. It is assumed that the phone being used in this flow chart has Bluetooth. The Bluetooth device onboard the spice holder was called by the phone to give the different functions. The light function is when the indication of current spice to be used. The Bluetooth device also monitors the spice level in each spice. The ability to manually pour the current spice is also done through Bluetooth. Another function that is implemented in Bluetooth is the ability to tell where each spice is located. if need be the use, the user can find the spice that is low and change it. All these implementations contributed to the success of wirelessly connecting to the spice holder.

Microcontroller Selection

Given all considerations, it is believed by the engineering team that the MSP-430 serves as the best candidate for our design. This is for a variety of reasons. The MSP-430’s lower power consumption, while resulting in poorer performance of peripherals, allows for the majority of the power consumption to be done by the peripherals themselves. This means that the device as a whole consumes less power, making it more attractive to potential customers. Personal familiarity with the MSP-430 also is a factor, as this means that it is easier for the programming process to be performed by the programming team, as there are less difficulties with reading and understanding documentation and performing testing. Most of the features of the C2000 that are critical for our design are also present in the MSP-430, albeit at a lesser quality. However, the reduced quality is not drastic enough to entice a change. This is because the difference in reaction time of peripherals is at such a small level, that they are nearly unnoticed by the user. For example, if the emergency abort procedure is performed in the event of an infinite loop, if it is done at the last possible moment, or if it is done after a predetermined amount of seconds that is not as drastic as the last possible moment, the customer wouldn’t notice much of a difference. Further, in some applications, such as the emergency abort system, it might make more sense, for safety reasons, to work with the slower CPU. For instance, there is no point in risking the last possible moment when there can be a safe amount of time in between the two extremes, prior to aborting. The MSP-430, including MSP-430 based microcontroller boards, are outright cheaper than the C2000 based microcontrollers. Because of this, it makes more sense to work with the MSP-430, both in the interest of saving money for the engineering team, as there is a limited budget. Additionally, should it become a viable product, it makes for a better product if the price is cheaper, and if there is no noticeable drawback between a change in the design done to save money and reduce costs.

This means that, all in all, the MSP-430 better serves our design. However, while this is our primary choice, should there be an unexpected issue that makes an MSP-430 insufficient for the final product, the C2000 still remains a viable alternative, albeit a less desirable one.

First Subsystem

The First Subsystem was composed of the Machifit 6V DC Reduction Motors that turn the Spice Rack system, the Hall-Effect encoders that are on the motors, and the L298N Motor Driver. This subsystem was used to rotate the Spice Rack to the desired spice of the user. Instead of using two motors for this system we were able to run only on one motor.

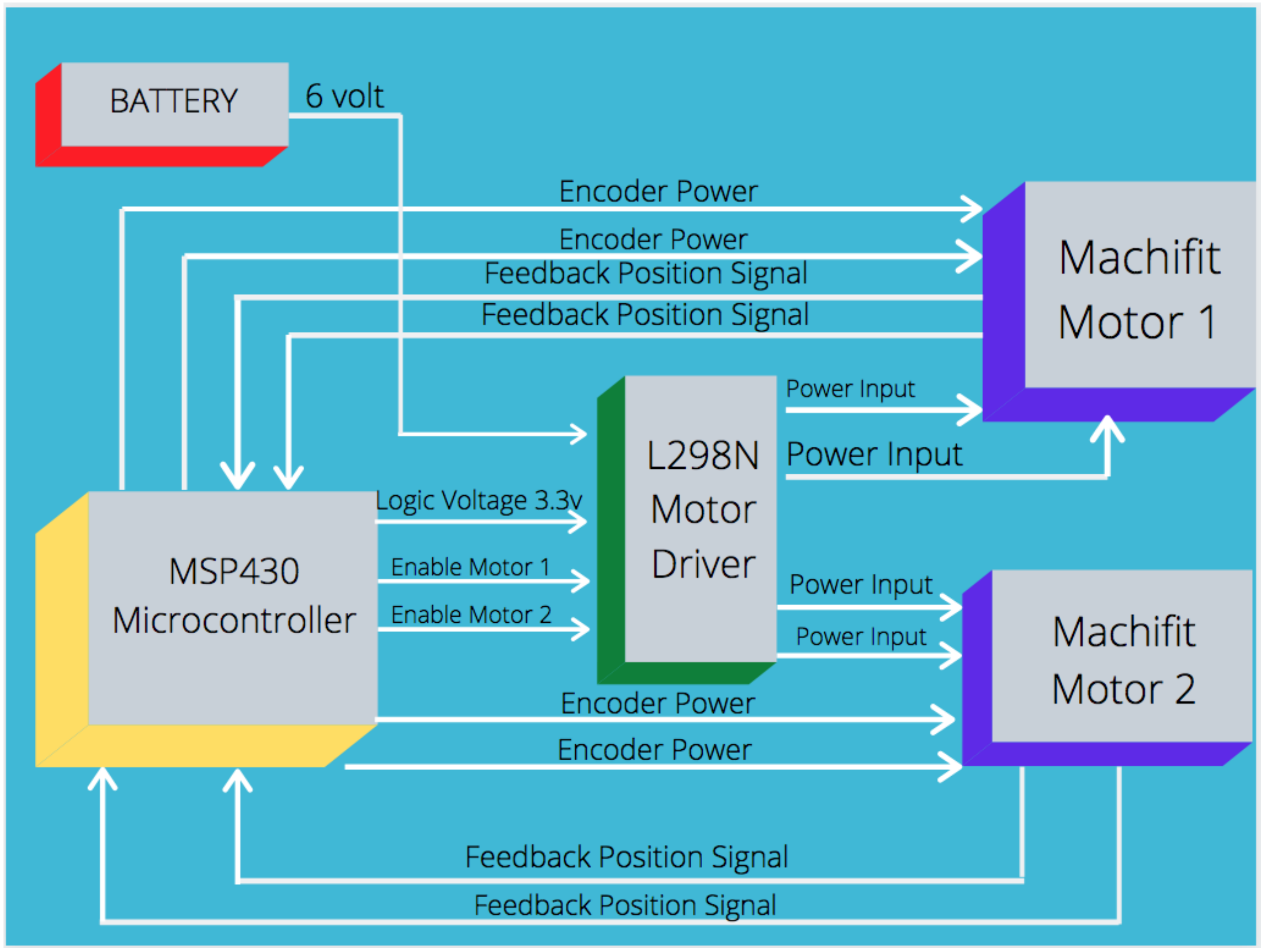


Figure 21: First Subsystem Flowchart

Machifit 6V DC Gear Reduction Motor

This section gives the specifications of the Machifit motor, and reasoning of why this particular motor was best for our design.

Narrowing down the search to a gear reduction motor with an encoder wasn’t the end of our search, but gave us the opportunity to find a gear reduction motor that gave us enough torque to turn our spice rack at a low speed and have a low power consumption. Starting out the search there were a lot of 12-volt DC micro gear reduction motors, but if the motors would run on 12 volts the battery would have to be large enough to supply a voltage that high. So, by looking for a motor that ran on 6 volts, it narrowed our search down to a Machifit 25GA370 DC 6V micro gear reduction motor with encoder. The Machifit gave us a wide variety of speeds (from 12rpm to 130rpm), but the higher the speed the lower the torque (torque is measured with kilogram centimeters) was. So, the right motor for our design was running with a speed of 12 rpms, which gave us a maximum time around 3.33 seconds to move from one spice to another since the motor only had to move the spice rack half a cycle to reach the furthest spice away from the spice dispenser (located in the front of the rack). Along with the fact that at the maximum rated load, the motor runs at a speed of 9 rpms. Also, the motor only had to move at a max of a half a cycle because the motor was driven by a motor driver which allows the motor to move clockwise and counterclockwise. When running with a speed of 12 rpms, the motor had a rated-load torque of 8.3 kg.cm. While this might be enough torque for the spice rack if we made it fit only a small number of spices. Then when analyzing the parameters of the Machifit 25GA370 DC 6V micro gear reduction motor, the maximum power that it did consume is 3 Watts (takes in 0.5 amps of current).

The Machifit 25GA370 DC motor has a 6-pin connector, but only 2 of the pins are used directly for the motor because 4 of the pins are for the encoder signals and power. The 2 pins of the motor connector are used for the positive and negative (red and white wires) power inputs that the motors use to run. By changing the voltages between these two wires, the microcontroller can control the rotation of the motor because the current through the wires stayed the same and the voltage difference between the two inputs was equal to 6 volts.

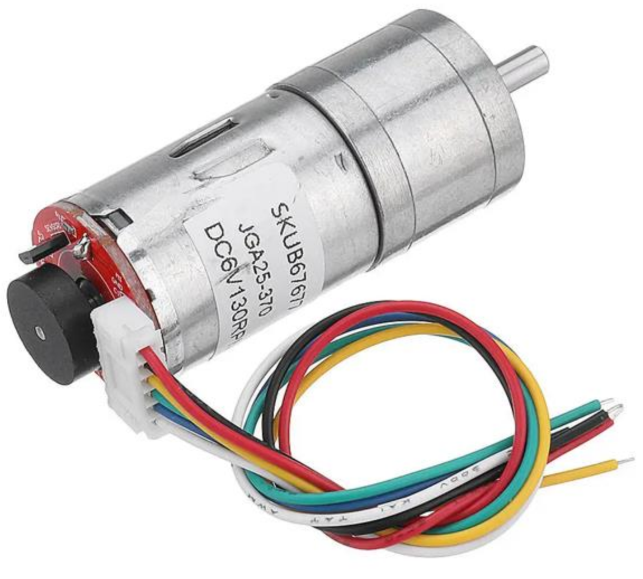


Figure 22: Machifit 6V Gear Reduction Motor

Hall-Effect Encoder

This section will explain the type of encoder that is on the Machifit motor, and how it was used to give precise positioning to our system. Along with the specifications on how the encoder worked with the microcontroller.

Since we used the Machifit 25GA370 DC 6V micro gear reduction motor with an encoder, we needed to know how the encoder operated and how accurate it could measure the position of the motor. The encoder is a rotary encoder that gives a closed loop feedback signal that are tracking the position of the motors shaft. With the feedback signals, the microcontroller can be able to accurately know what the position of the shaft is and anticipate when to stop the motor because if the encoder knows where the position of the shaft is, it knows how fast the shaft is moving. From the Machifit 25GA370 datasheet, we know that the feedback from the encoder to the microcontroller had two digital signals that represents the constantly changing position of the motor. Using these digital signals, the microcontroller has the able to predict when to stop the motor in order for the desired spice to stop at the dispenser.

The rotary encoder on the Machifit 25GA370 is able to find the position of the shaft and the speed by measuring the absolute angle of the encoded shaft because the encoder can see unique positions that appear on the motors shaft. These unique positions on the shaft appear to the encoder through a coded disc that rotates with the shaft. So, as the shaft rotates the disc positioned on the shaft rotates as well and a receiver on the encoder picks up the particular point on the disc. The specific encoder that is on the Machifit 25GA370 is a magnetic encoder, which uses a receiver that detects the change in magnetic field due the shafts rotation of the disc and knows the position of the shaft. This change in the magnetic field is due to the north and south poles on outline each position (can be seen in Figure 4 from “Magnetic Encoder Guide”).

The Specific receiver used for this encoder is a Hall-Effect sensor. A Hall Effect sensor takes into account the Hall Effect Principle, which is the fact that a potential difference (also known as a voltage) is a product of a magnetic field being applied perpendicularly to the current that is passing through a conductor(visualization can be seen in Figure 5 from “Magnetic Encoder Guide). By using this principle, the Hall Effect sensor can detect a change in magnetic field due to the change in voltage. The magnetic field can change based on the north and south poles that are positioned on the outside of the encoder disc. Once the shaft of the motor rotates the encoder disc had a change of magnetic field from the different poles, which deflects less or more electrons. With more or less electrons flowing through the conductor, it can fluctuate the voltage and this change in voltage is seen from the Hall Effect sensor through a differencing amplifier and a set resistor. With a Hall Effect sensor having around a 1% output error, it gave us enough accuracy to be able to have the right spice that the user wants to use. The Hall Effect sensor also has a low power consumption and works best with high current motors. While there may not be a ton of current flowing through our Machifit gear reduction motors, the amount of current running through the Hall Effect sensor should be enough to keep it as accurate as possible. The sensor then would output a sinewave that is produced from the magnetic deflection. This sinewave is then converted into a square wave which can be translated by the microcontroller to display the position of the motors shaft. Thus, by knowing when each position changes, the encoder is able to calculate the speed of the motors shaft through calculations and output it in digital signals (“Understanding and Applying the Hall Effect”).

In the Machifit 25GA370, the encoder had inputs through 4 pins. As stated, before 2 of the input pins are used for the feedback signals from the encoder to the microcontroller. These 2 pins (yellow and green wires) communicated the position of the motors shaft to the microcontroller (which can be used to find speed as well). The other 2 pins (black and blue wires) are the power inputs for the encoder, which supply the power for the encoder to function and monitor the speed and position of the motor shaft. While these pins are the power inputs for the encoder, they can be positive or negative side of the input because as long as the encoder gets the right amount of voltage (3.3 – 5 volts) then it works correctly.

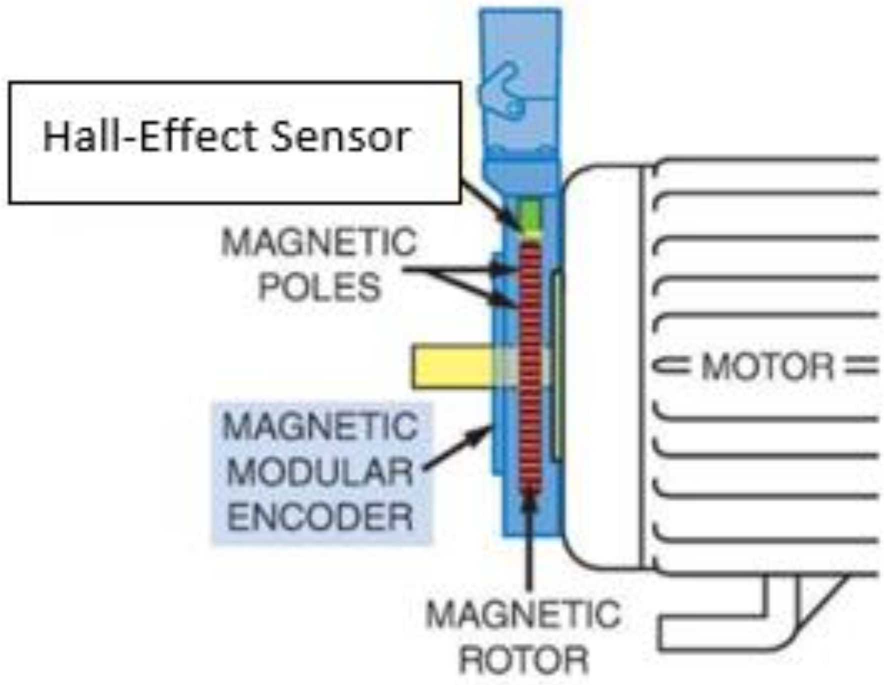


Figure 23: Hall-Effect Sensor

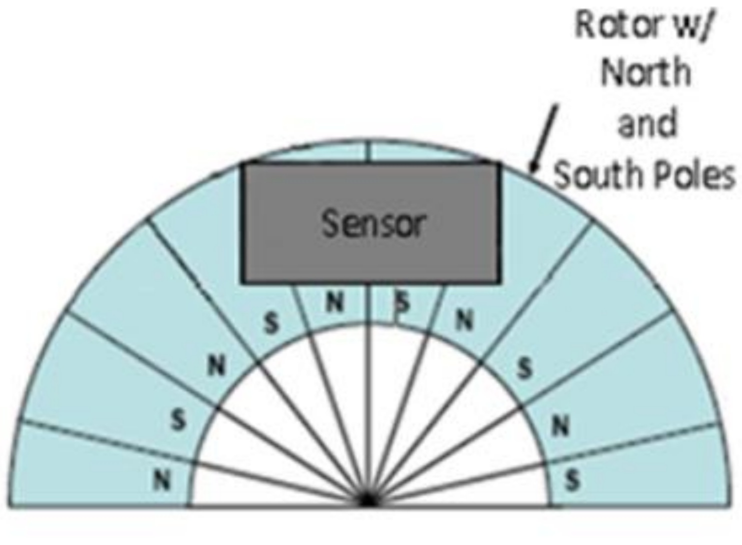


Figure 24: Encoder Rotor

In practice, there is an error in the encoder of about 20 degrees in a full rotation in our implementation. However, this was rectified using a VL53L0X Time-To-Flight sensor to detect when one of the container supports has moved 90 degrees, which allows us to eliminate this error. This time to flight sensor complicated the design of our rotational mechanism, but it allowed for very precise movements.

L298N Motor Driver

This section gives specification details on the L298N and how it will interact between the microcontroller and the Machifit motors.

By communicating from the encoders of the gear reduction motors to the microcontroller, the spice rack was able to spin to the desired spice and dispense it accordingly. But there still was a need for something that can spin the motor clockwise and counterclockwise because the motor would waste power if it had to make one complete cycle in order for a spice desired to the right of the dispenser (assuming the motor is moving clockwise). Not including that with the motor moving at a speed of 12 rpms, the spice to the right of the dispenser won't be reached for at least 4 seconds. This is where a motor driver was able to come in handy when driving the motor and changing the direction of our motors. Through the motor driver, the motors were more efficient and overall consumed less power than if they had to run in one direction the whole time. Besides, no user of the spice rack wants to wait at least 4 seconds just for another spice to be ready to be dispensed, this would defeat the ease and accessibility of the design.

By looking at various motor drivers, the best fit for our design was the L298N component, because it met all the requirements needed for the microcontroller to control and run the two motors that rotates the spice rack. The L298N motor driver is a 15-lead Multiwatt package dual full-bridge driver (means we can run two motors using this driver), that is designed for high voltage and high current use. While we didn't need the high voltage use of the L298N, we do need around 1 amp to run our motors. The L298N is rated for a max of 4 amps, which easily meets our requirement for needing 1 amp to run the two motors. The extra 3 amps gap adds for some realistic leeway in our design through different tolerances and if we max out the motors that the L298N is driving. Plus, the minimum voltage supply for the L298N is 4.8 volts (maximum voltage supply is 46 volts), which means we can run the 6-volt motor using this driver and the 6V servo motor.

Other important features that was needed from the datasheet was the logic supply voltage, which is the low current signal that is coming from the microcontroller to control the motors. The logic supply voltage needed to run the motors is a minimum of 4.5 volts and a maximum of 7 volts. We had an output signal from our microcontroller closer to the minimum voltage rather than wasting power with maximum rating. With the logic supply voltage being at about 4.5 volts the enable voltages which enables the signal from the microcontroller to the motor, had a high enable high voltage from 2.3 to a max of the logic supply voltage (around 4.5 volts). While the enable low voltage was between 0 to 1.5 volts. The enable signal acted like a digital system that turns on and off the motors, so when there is an enable high the motor turns on and when there is an enable low the motor did turn off. The enable signal essentially enables or disables the device so that the logic signal did or didn't reach the motors. Then the inputs shown on the L298N device, which are the output signal from the microcontroller that communicated the direction (clockwise or counterclockwise) the gear reduction motor spins the spice.

By using the L298N motor driver, the microcontroller was able to communicate and control the motor through an input signal with a lower current than the motors needed current level. Along with being able to rotate the motor in unison clockwise or counterclockwise motion and give a signal that would enable or disable the microcontrollers connection to the motor. In all, the L298N gave the microcontroller easy access to the motor and helped translate the signals from the microcontroller to a signal that the motors can understand.

While we used the L298N motor driver we needed to implement an integrated circuit through L297, which generates the signals needed to drive the L298N from the few signals given from the microcontroller. The 298N also has 8 diodes connected to the outputs of the L298N to regulate the current going to the motors so that we don't over drive them or burn them out. Then we needed a couple of capacitors in connected to ground from the power inputs of the driver and I.C so that we won't get any noise that may come in and disrupt the circuit. This circuit is added to the L298N is seen below, with the different components and values are needed (voltages are shown as the max value and is set to 3.3 volts going into the L297 and 6 volts going into the L298N).

But, since the MSP430 had PWM pinouts, we were able to get rid of the L297 that would drive the motor driver.

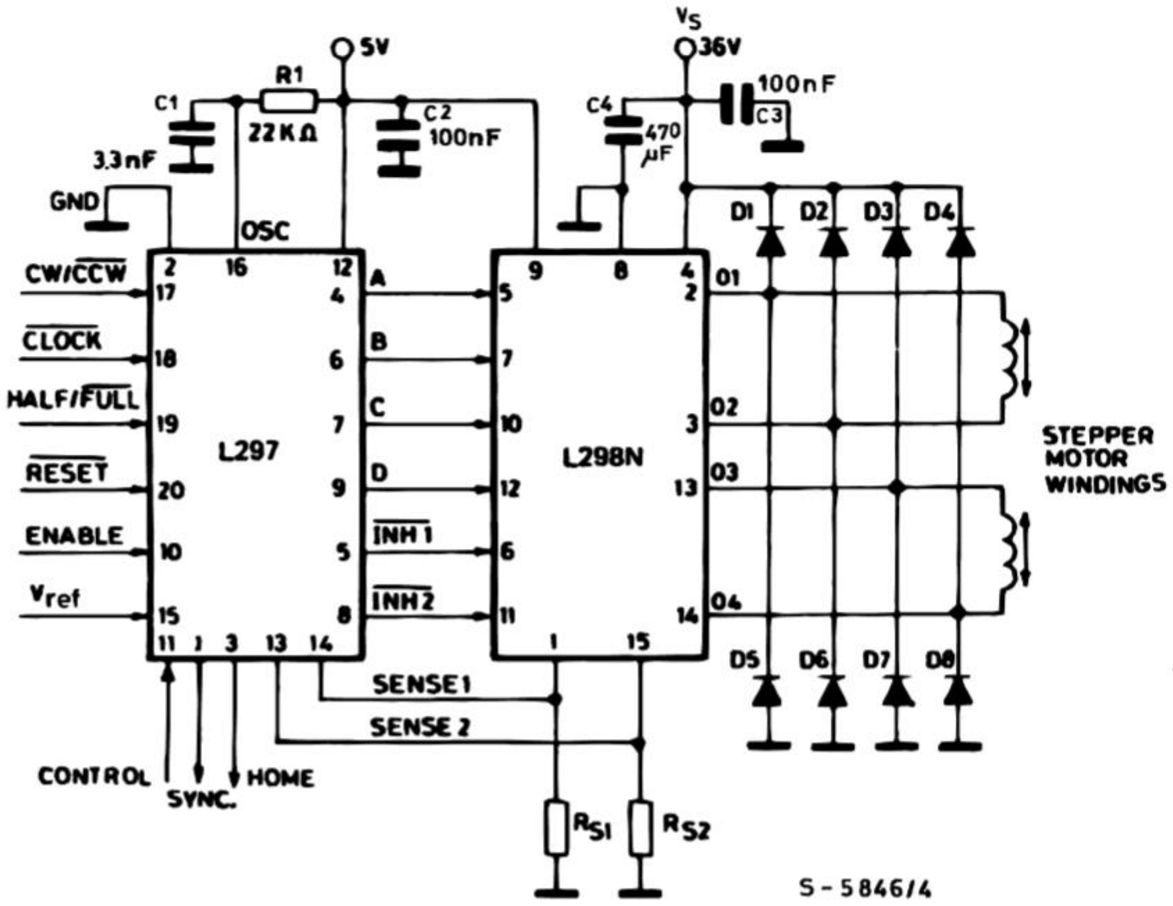


Figure 25: L298N Motor Driver for Machifit Motors Schematic

Second Subsystem

This Subsystem is made up of the MM10 DC Motor that would have been used to crush and grind down the solid spices that the user needs, and the L9110S Motor Driver that practically translates the signal from the microcontroller to the motors to crush and grind the desired spices.

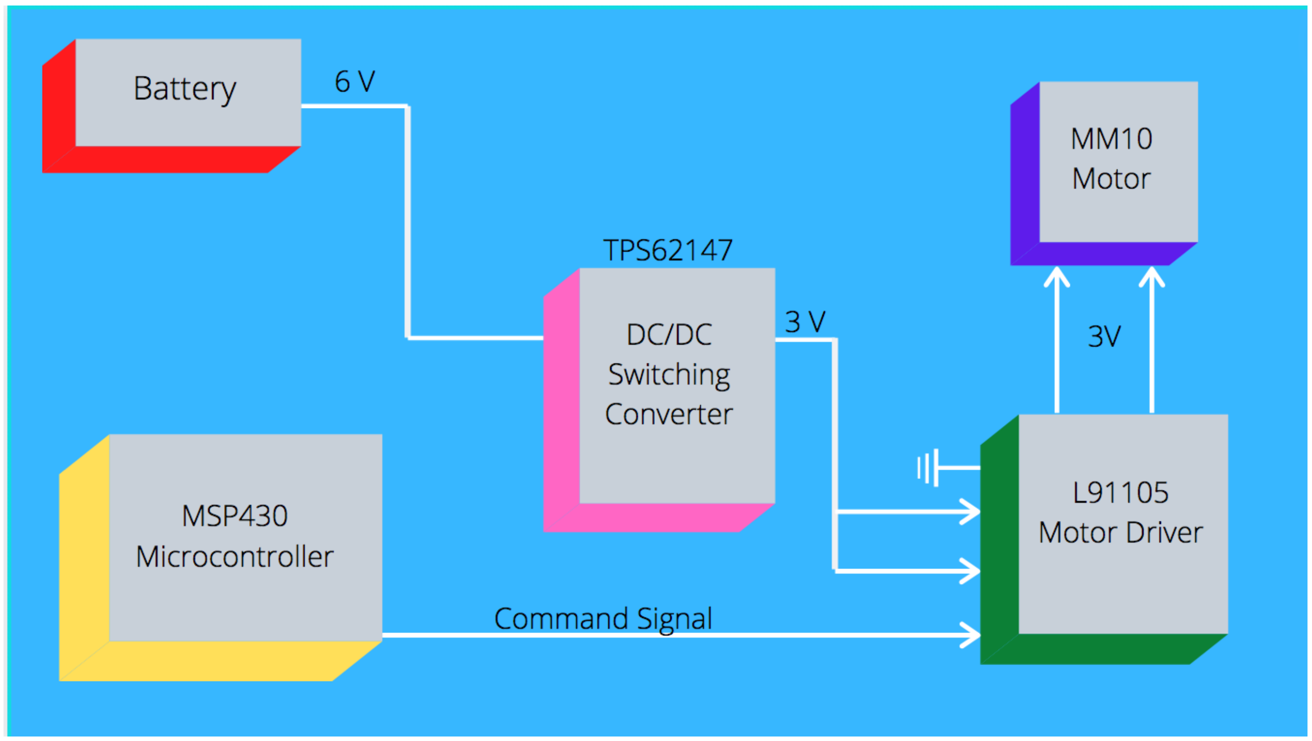


Figure 26: Second Subsystem Flowchart

MM10 DC Motor

This section will give detailed specifications on the MM10 DC motor and why it was chosen.

By searching for a low power, low voltage DC motor, the best option found was a MM10. The MM10 is a cheap ($0.423 per unit), low power consumption motor with a voltage rating of 3-volts. From the datasheet, the MM10 is at maximum load pull a current of 1.5 amps, while having a speed of 12400 rpms (plus or minus 8%). This motor also has a torque of 10g.cm, which may cause some problems for crushing harder spices. However, the concerns of the motor not having enough torque to crush certain spices and having too much noise (30cm<68dB), is outweighed by the price and low the power consumption of the motor. The MM10 is also easy to use because the only inputs required to run the device is a positive and negative voltage that is equivalent to 3 volts. While the motor may need a driver, so that the low torque of the motor won’t be as much of an issue, it is all around easy to use. By adding a motor driver to the MM10, it would be able to rotate the motor clockwise and counterclockwise and allow the microcontroller to enable or disable the motor while using a low current signal. With the motor driver, it may allow us to save money by not getting a motor with a high torque and can allow for better control of the amount of spice the user needed.

Figure 27: MM10 Motor

Unfortunately, due to physical constraints and the lack of space in the spice rack, we weren't able to implement the crushing subsystem into our final design.

L9110S Motor Driver

This section will explain the specification details of the L9110S motor driver and how it will communicate between the microcontroller and the MM10 motor.

The L9110S is a motor driver that works for motors at a voltage rating of 2.5 volts to 12 volts. While allowing a max of 0.8 amps of current through the output to the motor. With the L9110S, the microcontroller was able to control the MM10 motor that grinds the fresh spices and herbs for the user. Since the L9110S is a dual bridge motor driver, we can actually get a max of 1.6 amps coming from its outputs which is large enough to run the MM10 motor. This may not be ideal to run one motor using two different outputs from the L9110S motor driver, but it is cheap enough and gives enough control of the MM10 motor to the microcontroller to manipulate. Especially since the L9110S can take input values from a range of 3.3 volts to 5 volts. This gives a wide range of values the microcontroller can send in order to control the MM10 motor. To control the direction of the motor the L9110S uses a Pulse Width Modulation (PWM), which modulates the signal that is used to power the motor. By changing the pulse of the input power signal, the motor positive and negative terminals can be switched to change the polarity of the motor and make it rotate in the opposite direction. This is a completely different way than the L298N changes the direction of the motors, since it just changes the signal between the positive and negative terminals instead of using a pulse width modulation like the L9110S does to change the rotation of the motors. Like the L298N, the L9110S takes a lower current signal from the microcontroller and output a higher current signal to the MM10 motor. However, the L9110S is made for smaller power consumptions and simpler functions. With the L298N the microcontroller could enable the inputs and disable them, while the L9110S only turns on and off based on the control of the microcontrollers input power. This may cause some issues, but overall, it is an easy to use the L9110S and it won't need as much power to run the MM10 motor.

Another issue we may face, is that the MM10 needs more power than 0.8 amps to grind the different spices (different spices have different loads put on the motor), since the MM10 motor has a maximum rated load current draw of 1.5 amps. A solution to this problem is that two outputs of the L9110S can be tied together to drive the MM10 with 1.6 amps maximum (both outputs drive 0.8 amps maximum). The 1.6 amps was enough to run the MM10 its maximum rating, but this could cause some delays due to the noise differences between the two outputs. Even though the L9110S outputs should run two motors in phase with each other. Another solution to this problem is that the grinder feature could be ran by 2 motors with just drawing 0.8 amps, which may give us more torque than just maxing out one motor. To make sure of the exact design testing had to be done to the L9110S with the MM10 motor to see which design would work best for us.

While we may use the second output of the L9110S system to run the N20 motor would dispense the regular spices from their containers, the L9110S would use the same inputs regardless since both motors ran on 3 volts.

The circuitry with the L9110S would had been constructed as seen below, where the Vcc was set to 3 volts because that's the voltage needed to run the N20 and MM10 motor. There was also be capacitors used to reduce the noise from the inputs and keep the DC signal as constant as possible by dissipating low (10uF capacitor) and high frequencies (0.1uF capacitor). While we won't need to use the LED light shown in the circuit below, we may need it to make sure the motor driver is getting power.

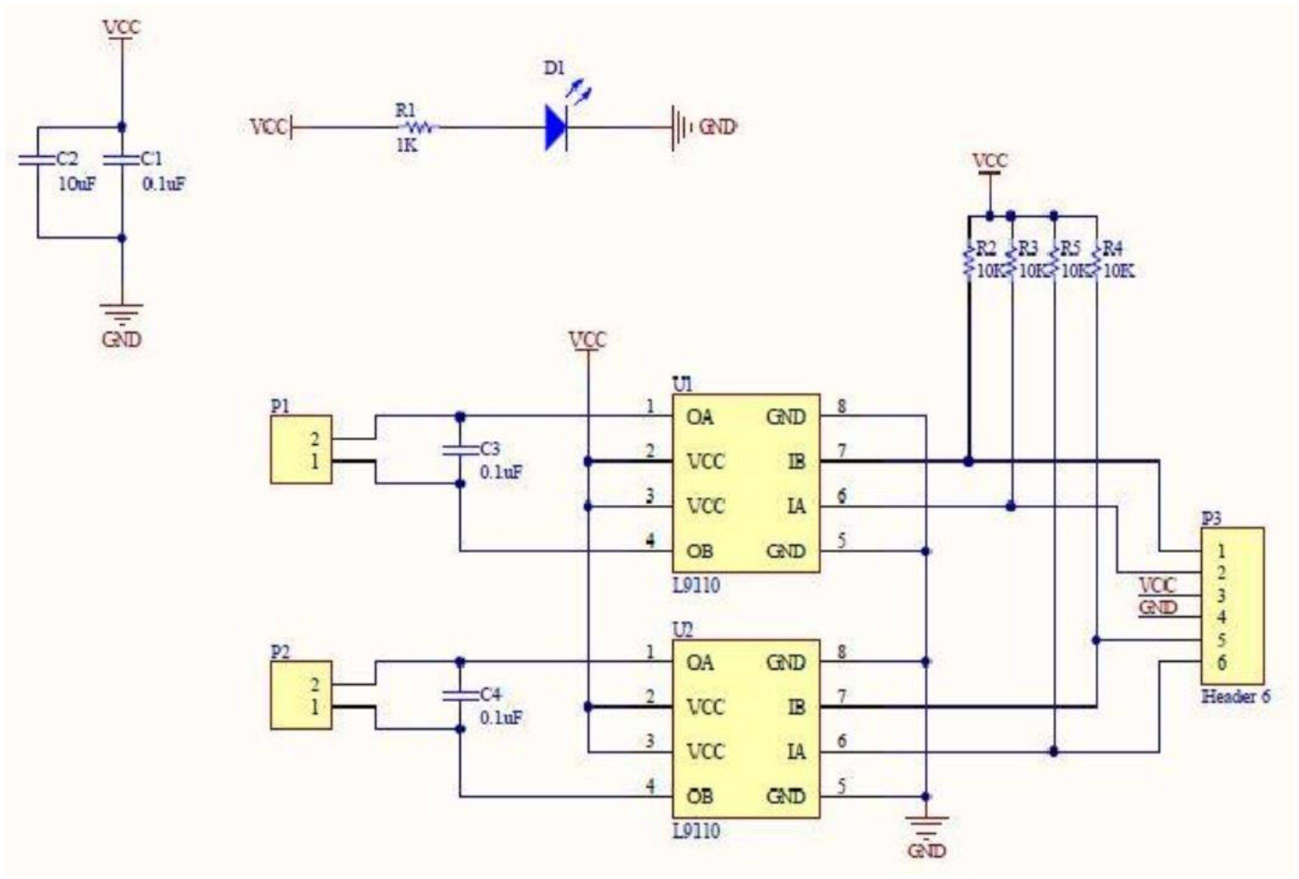


Figure 28: L9110S Motor Drivers Schematic

Again, due to size constraints of the spice rack system, we couldn't implement the motor drivers and crushing and dispensing motors this way. We were able to use the L298N to drive the sero to actually dispense the spice using the dispensing system phsyical design.

TPS62147 DC/DC Converter

This section will discuss how we used the TPS62147 converter and given circuitry to drop the battery voltage of 6 volts to 3 volts, and what the components of the circuit are used for.

The TPS62147 DC/DC Converter was used to drop the battery voltage from 6 volts (VBAT) to 3 volts (VOUT) and maintain a maximum output current of 2 amps, which was needed since we were running the N20 and MM10 motors along with other components on the board using this output. The main use for the converter wwas to supply power to the motors to crush and dispense the various spices on the board. Like previous components, the converter needed capacitors to dissipate the noise that is caused by the switching of the component. Then the resistors needed to have a value so that the DC drop gave an output of 3 volts. When making R1 equal to 510 KW and R2 equal to 180 KW, we were able to get our 3-volt output (it is actually equivalent to 3.06 volts).

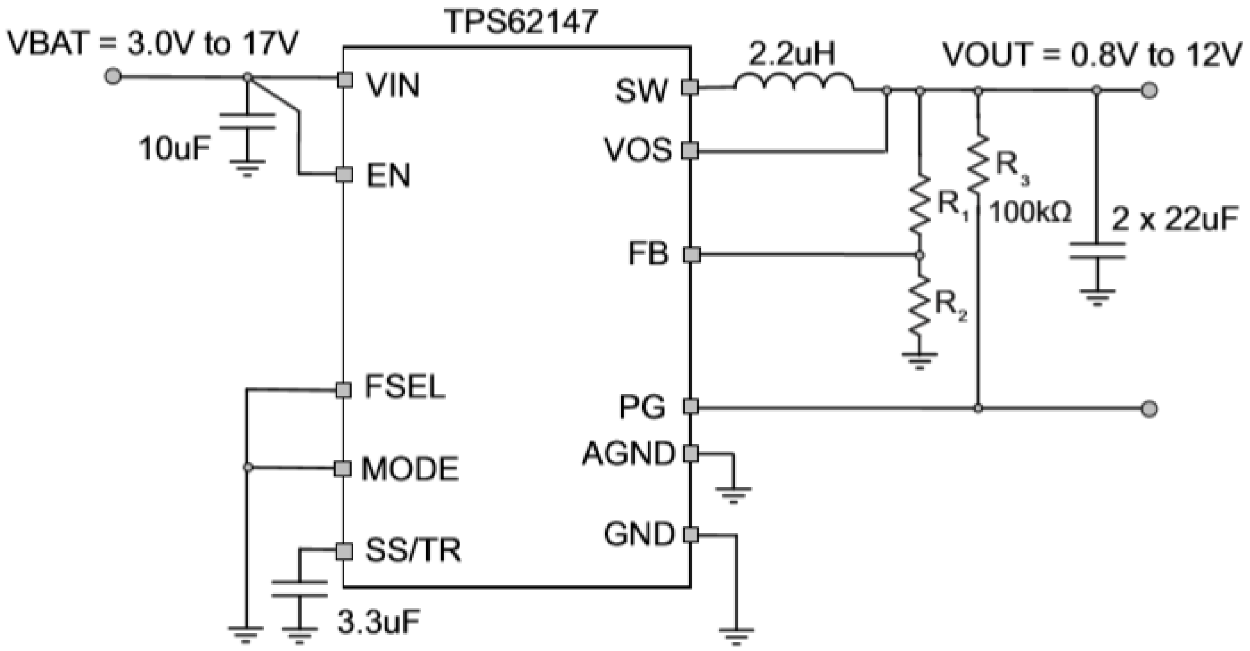


Figure 29: TPS62147 DC/DC Converter Schematic

Due to the fact that we were actually able to run a 6V servo motor with the L298N we didn't need the TPS62147 to output 3V. So this was scrapped and we were able to optimize our design and get rid of a couple of components. But we were still able to use the TPS62147 to get our 3.3V output for our ESP32 and MSP430.

Third Subsystem

The Third Subsystem is the core of the whole Spice Rack system. Using a Rechargeable battery to provide power to the entire board, and a PV panel to recharge the battery when the system is not being used. Then the Microcontroller will take some power to communicate to each component when to operate and how to operate during certain inputs provided by the user.

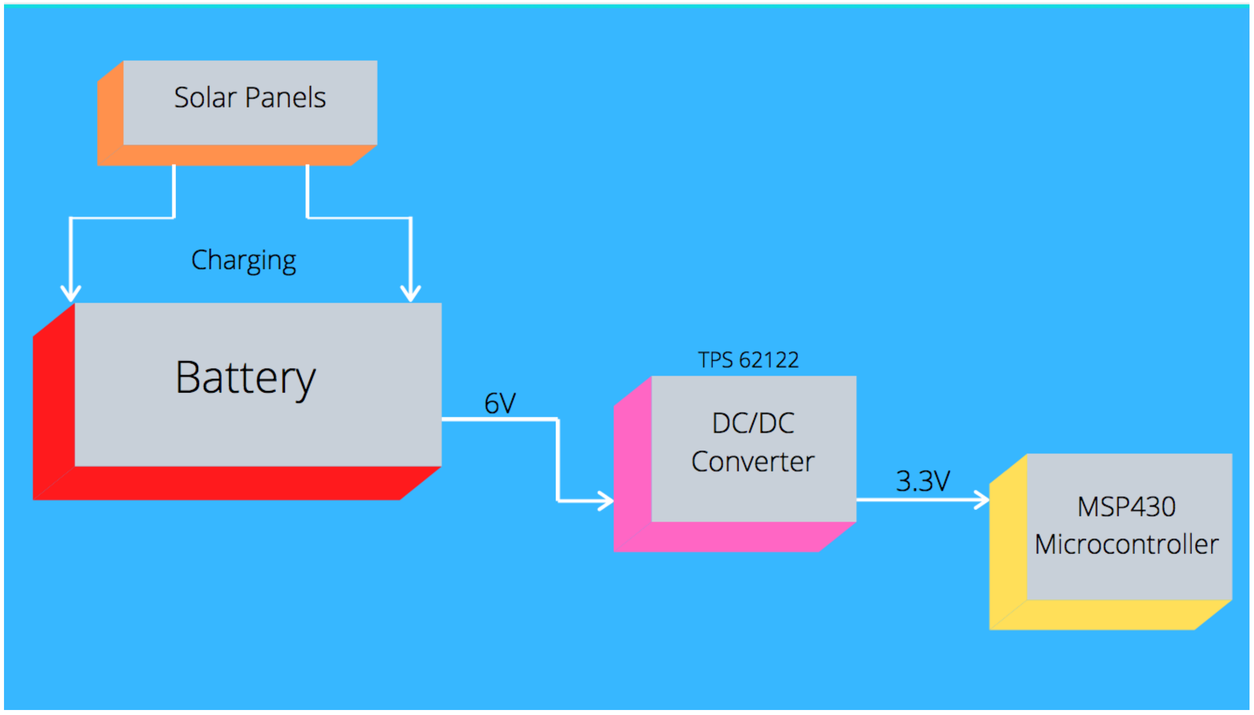


Figure 30: Third Subsystem Flowchart

Rechargeable Battery

This section will go into different types of rechargeable batteries that could be used for our design and which one would give the best results.

Finding the ideal battery, we needed to know the max power consumption of the spice rack at one time and knowing the highest current and voltage needed by one single component. The largest component power consumption at one time was running the two Machifit 6V motors, which consumed around 6 Watts maximum (the current for the maximum load for the motors is 0.5 amps). While Machifit motor wasn't the most power consuming motor, it was used most of the time because the spice rack had to continually spin to various locations back to back for the user (most users have more than one spice dispensed per use). This gave us multiple options of how to know the right amount of power to run our spice rack.

Option 1 was to get a 6-volt 5Ah (5 amps per hour) and directly contact the Machifit 6V motors, but they consumed a lot of the power starting out. With only have 30 Watts of power per hour, the MM10 motor would have to run efficient and not be used often. Along with the other components on the board would have to be as efficient as possible. Thus the 6-volt battery it would have to be rechargeable so the user doesn't have to keep changing batteries out, and it would have to be charged with a PV panel. Which should be manageable, since the battery is very isn't very large.

Option 2 would be to have a 12-volt or 9-volt battery with a 5Ah and use a switching regulator to knock down the voltage to 6 volts. By knocking down the voltage, the system would be able to use the 6 volts for the Machifit motor and could again knock down the 6 volts to the voltage the rest of the system would need to operate. With using a 12-volt or 9-volt battery, the system would have more power to use per hour than just using a 6-volt battery. However, knocking down the voltage efficiently would have been troublesome because most switching regulators aren't made for how much current the battery is putting off (not enough current). Then if the system was to use linear regulators, the current would have to be drastically lower than in order to use a linear regulator. Most linear regulators have a max current capacity of 1 amp because they aren't usually used to step down a large battery like the switching regulators. This option would give more of a challenge for circuit design, while giving the most power.

Both of these options would have been valid but depending on the design of the system and how much power we needed impacted which battery to use. Both can be found at Battery Mate. The 6-volt Battery may be cheaper, but it isn't by too much, and both batteries had recharging capabilities.

Figure 31: 6-Volt Battery

Unfortunately, due to Covid-19 we weren't able to run our design using the battery, even though we had the components to due so.

PV Panel

This section will go over the PV panel that will be used for our design and why it was selected.

Since the PV panel can hold 2 Watts (output of 6 volts and 0.35 amps) and still be able to charge the battery enough because of the frequency of use for the system. When looking for PV panels, the system would need it to fit right on the top to keep it sleek and compressed. So, with a 6V 2W Round Style Polycrystalline Solar Panel Epoxy Board that has a diameter of 80 millimeters (about 3.15 inches), would easily fit on top of the spice rack. This Polycrystalline Solar Panel has a conversion efficiency of 18%, which isn't a great conversion efficiency. But with the Solar Panel being $1.68, it gives some concerns because it might not perform the way it is stated. Though it is cheap enough to get and work with until the problems outweigh the reward.

While a 6-volt rated solar panel would charge our battery accordingly, if we get a large rated PV panel to charge the battery, we can neglect the nonlinear characteristics of the PV panel. By using an 18-volt, 4.2-watt PV panel to charge the battery, it substantially droped the charging time drastically. While it is vastly more money at $11.98, it gave us more versatility for our design and allowed us to take away the nonlinear characteristics of the PV panel.

Figure 32: 18-Volt PV Panel

LM317T Regulator

This section will discuss how the battery will be charged up by the 18-volt solar panel, and the circuitry needed to have the 18-volt panel to charge a 6-volt battery.

In order to connect our 18-volt, 4.2-watt panel to our 6-volt battery we needed to add a circuit to drop the voltage down from the panel to the battery, but keep the amount current the and power the same. So, by adding a LM317T voltage regulator to step down the voltage from 18-volts to 6-volts we needed to use the circuit below. The circuit shows how the resistors set the output voltage to be equal to 6 volts and the capacitor is used to keep the noise coming from the solar panel low. Then the transistor is to modulate the current that was going to the battery from the battery so that the solar panel didn’t jeopardize the integrity of the battery.

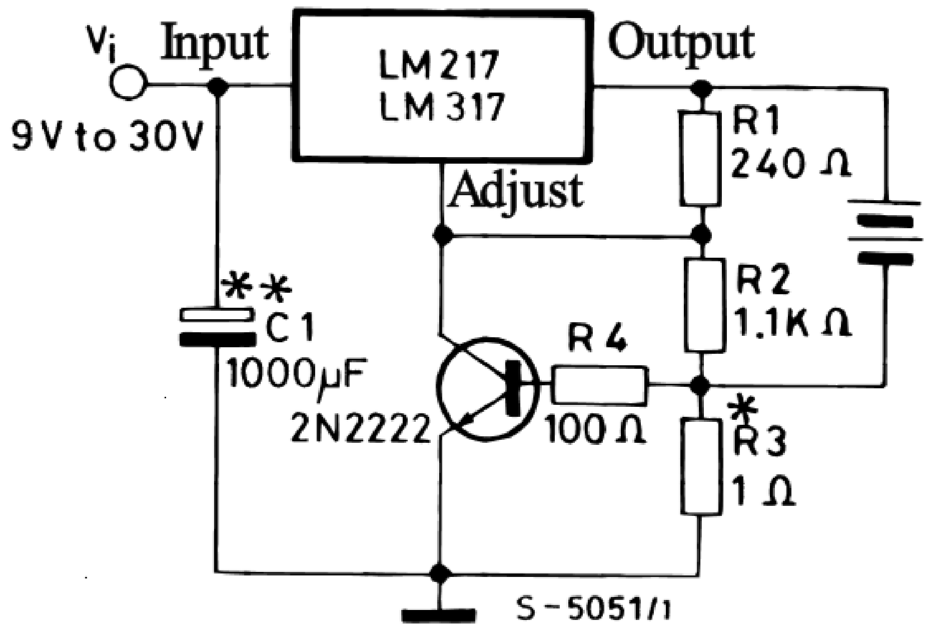


Figure 33: LM317T Regulator Schematic

TPS62122 DC/DC Converter

This section will discuss how the battery voltage will be dropped down to 3.3 volts to run the microcontroller at a max of 78 milliamps.

The TPS62122 DC/DC Converter is used to take the input of the battery at 6 volts and drop it down to 3.3 volts. It also has a max current of 78 milliamps because the microcontroller won't have a current draw over 78 milliamps. The circuitry seen below was used alongside this converter because it needed the capacitors to dissipate noise, since it is a switching converter there was noise due to the fast switching of the component. Then the resistors seen as R1 and R2 are used as voltage dividers to have the output voltage to equal 3.3 volts. The values for R1 and R2 were found using the equation Vout = Vref *(1 + R1/R2), where Vref is 0.8 volts. The TPS62122 datasheet gives valid resistor values R1 and R2 along with the capacitor Cff to make the output voltage to equal 3.3 volts (seen below).

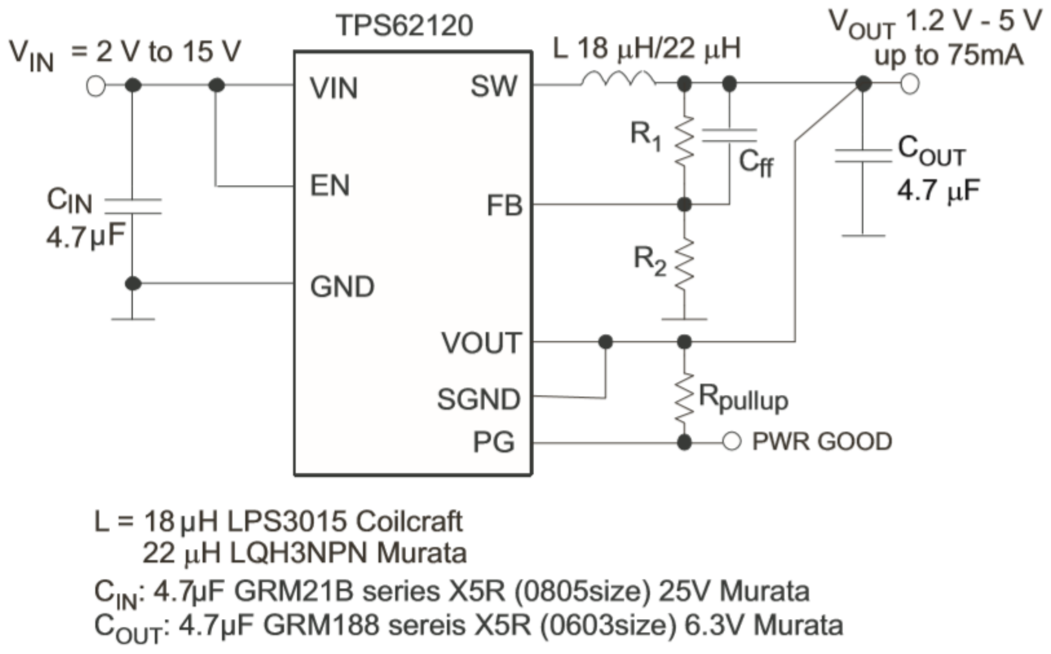

TPS62120
V_{IN} = 2 V to 15 V
VIN
EN
GND
SW
FB
VOUT
SGND
PG
L 18 µH/22 µH
V_{OUT} 1.2 V - 5 V up to 75mA
R_1
C_{ff}
R_2
C_{OUT} 4.7 µF
C_{IN} 4.7µF
R_{pullup}
PWR GOOD
L = 18 µH LPS3015 Coilcraft
22 µH LQH3NPN Murata
C_{IN}: 4.7µF GRM21B series X5R (0805size) 25V Murata
C_{OUT}: 4.7µF GRM188 sereis X5R (0603size) 6.3V Murata

Figure 34: TPS62120 Converter Schematic

VOLTAGE SETTING (V)	3.06	3.29	2.00	1.80	1.20	5.00
R_1 [kΩ]	510	560	360	300	180	430
R_2 [kΩ]	180	180	240	240	360	82
C_{ff} [pF]	15	22	22	22	27	15

Figure 35: R1 and R2 Values for 3.3-Volt Output

Due to its low output current, we decided to use the TPS62147 DC/DC switching regulator had a higher output current and would fit the same circuit as the TPS62120. The TPS62147 gave a max output of 2A, which suites running the MSP430 and ESP32 better.

Microcontroller Unit

The Microcontroller Unit refers to the general PCB board around the Central Processing Unit, which will contain all the pins for wiring multiple components together and to allow communication between the pin-connected components and the CPU. Whilst there are existing, pre-built and already carefully designed MSP-430 based development board MCU's as well as C2000 based development board MCU's, we will construct our own MCU with our own design. This will allow us to directly, via hardware, connect all elements to the processor, and will then allow us to simplify the wiring and construction of our prototype device. For instance, we should be able to directly connect the Bluetooth chip which will provide for the connectivity with the mobile application into the board itself, eliminating the need to deal with General Purpose Input/Output pins, and thus simplifying the process of putting the parts together once it all has been programmed.

This, however, does not mean we cannot look at existing microcontroller boards for inspiration. Already, the prior section on the Central Processing Unit, included some compatibility features that, while not directly available on the CPU, are easy to build using other Texas Instruments components, which means we can use existing microcontroller families as inspiration for our design. Further, it means that we can build a pre-prototype model, one purely for testing the software that does not necessitate the totality of the design, such as an exterior cover for the mechanism, with an existing microcontroller prior to our custom microcontroller design being complete.

Although it has been decided that the MSP-430 will be our main processor, that is, it will be the processor we will want to use as long as possible, the C2000 remains our backup alternative. As a result, we would want to look at the pre-existing microcontrollers for inspiration for our design.

MSP-430: The MSP-430FR6989 Microcontroller Unit is the main board we will use as a reference for the MSP-430 based boards. This is because they are the ones we as an engineering team are the most familiar with, the ones that are more readily accessible to us, and ones that certainly have all the requirements for our design, as well as additional features for extendibility.

C2000: The TMS320F28027 Piccolo Microcontroller Unit is our reference board for the C2000 board. This is because it seems to be a member of the board family with the best documentation, and seems perfectly capable of accomplishing all of our objectives. Further, it is the reference board used on this paper to gauge the C2000 features. As a result, it is the best suited for our reference designs, as it will highlight all the secondary Texas Instruments components that might also be needed.

Figure 36: MSP-430FR6989

Fourth Subsystem

In this subsystem we will discuss the lights that indicate the right spice selected, the light sensors that show the amount of spice left in a container, and the Bluetooth communication with the APP.

Light Sensor

In the definition of a sensor, there are several ways to sense listed. While there are multiple ways to track how much spice is in the current pod, we had to settle on one for our implementation. It has been have decided to use light sensing to decide the amount of spice remaining. This gives us a controlled way to visualize what is going on in each spice pod.

The objective in using a sensor is to notify the user about the remaining amount of spice. This information is important because it helps the user know when there needs to be a refill for whichever spice pod. This makes it easier than trying to look in each spice pod to find the remaining amount, manually. With the proximity sensors it makes this task easier because it is able to sit in the spice pod and update the user about what is left in the spice pod.

To avoid having too many light sensors this implementation limited the number of light sensors that were used in the spice pod. The goal here is to give the user three different states of the spice level. With each state the user was updated via the phone app. There were three light sensors that track and updates the users on what level the spices are currently at. These three sensors gave readings about what the spice level

The full state is when the spice pod is above the first sensor. The pod reports being full until the first light sensor is tripped. This state is to serve as a something that tells the user when the spice has just been refilled and does not require refilling soon. Having this knowledge helps with knowing which was refilled recently or which spice is hardly used. Having this helps with the overall usage of the spices in each spice pod.

Then we have the seventy-five percent full state. This was be after the first sensor is tripped until the other sensor gets tripped. The seventy-five percent state activated immediately after the first twenty-five percent of the spice is used. This state served as a notice the user that there is less spice in the spice the pod and but there is no need to change out the spice yet. With this state it is like a curtesy update, so the user is aware of the status of the spice.

The fifty percent state the halfway reminder state. This state served as an update to either consider replacing soon or keep an eye on this spice usage. This middle ground helps to give the user a semi-frequent update system instead of only when full and empty. From this it gives the user more awareness about the system. With this being implemented it helped to solidify the amount of knowledge is known about each spice, at any given time.

Finally, there is the twenty-five percent state or low, need refilling, state. Just like most devices there is a low battery level. This low state is to warn the user that the spice is about to run out and it should be replaced as soon as it is possible. Without this state the user would never really know when to refill each spice. With this added to the implementation the device can dynamically update the user so that it is known when to replace a spice.

To ensure that the project works test cases need to be developed that validate that the light sensor is functional. To ensure that the light sensor works we are implementing alignments to ensure the sensors aligned. After ensuring that each sensor is well aligned, tests was done to see when a sensor receives the appropriate signals. The thresholds was then be tested to see which level would be the optimal for operation. Different lighting environments were included in the testing to further ensure that no light but the emitted infrared is influencing the light sensors.

The chance of surges is possible, to alleviate this possibility there were be surge protective elements implemented. For instance, a surge protective outlet. This is to minimize the risk of overvoltage. By testing this extensively, the project can be protected from this treat to the system.

There were also heat tests because this is one of the reasons light sensors can fail. This was by raising the temperature. To ensure the components are not broken, the specs sheet was consulted before testing. Heat is an element that must be considered when using a light sensor because it can skew the components into giving false readings.

The other factor that can ruin a light sensing device is radiation. This is something that was not tested because it is unsafe. Radiation is a consideration when dealing with light sensors but is not worth the risk. The assumption can be made that users would not be working around highly radioactive substances. While radiation is a factor that can break a light sensor, it is not a component that is expected to be influencing the usage of the project.

The plan to test the cups is trying to get the light sensors to where the cup has specific values. This is done by getting the place at which the cup's spice dispenser is able to put the amount that is requested. This was achieved by getting standardized instruments of measurement, for example, a teaspoon. After gathering these instruments tests of light sensor placements was conducted. In doing this it can be insured that when the appropriate level is poured into the cup, the light sensor displayed this indication. The indication is done via sending a signal to an LED. This LED lights up upon the appropriate amount of spice being poured into the cup. This is done for each type of measuring size.

Ensuring the light sensor sends the signal to the Spice Holder is something that must be tested as well. This hinged on the LED test working. Given that the LED test works, the signal sent to the LED sent to the spice holder and it was controlled the pouring function that the holder performed. When the appropriate measurement is achieved a signal was sent to tell the spice holder to spot pouring. That was tested by going through each measurement ensuring that the signal is sent, and the pouring function stops appropriately. This test is important for the accuracy of the measurement pouring function of the project.

IR sub-system

This section will go over which IR subsystem will be used for the spice holder. The IR sensor selected needs to comply with the three most important requirements for parts picked on this project. The different elements that need to be considered are power efficiency, cost and usability.

Figure 37: IR Light Sensors

These IR light sensors help us active what is needed for the implementation of our project. These sensors use IR which is optimal do detecting if something is in the proximity. They are also low in power usage which makes the implementation of the spice holder being standalone possible. The cost on these light sensors are low, this is a major factor that needs to be considered when working on a project. It detects the right amount of space between the pods. Looking at all these functions this makes the opb100ez the optimal choice for the implementation of the spice holder.

IR Circuitry

This section will discuss our design on how the optic sensor will be used to let the user know how much spice is left in a given container when it is being dispensed.

For the optic sensor, which involves an LED transmitter and a phototransistor receiver, we need to have an input voltage of 5 volts to power the phototransistor receiver and an input from the microcontroller to get the signal to emit the LED transmitter. The input voltage of 5 volts came from the same 5 voltage that is going to the 74AC11138D to power the LEDs, and that came from the 10K and 50K resistors voltage divider seen in the LED section. The microcontroller input powers the LED to make it turn on and emit to the phototransistor. Once the spice is low enough for the emission to get through the output of the phototransistor connected to another microcontroller pin so that the user gets the information about how full the spice is. The microcontroller input powered the LED, but the LED did not continuously consume power. The LED also be flashing because the signal from the microcontroller has a bit waveform meaning when the wave is at its maximum (a 1 bit) it turns on the LED and when the waveform is at its minimum (a 0 bit) it has the LED off.

The circuit below demonstrates how the optic couplers were connected to the microcontroller, because the microcontroller inputted a digital signal to the LED to flash to the phototransistor to see if it can receive the light. When the phototransistor sees the light, it gives an output back to the microcontroller showing a bit value of 1 in wave form. While the phototransistor only sees the LED light only if the spice is low enough for the light to pass through the container and go to the phototransistor.

This circuit below was used three times so that we can have various amounts of information going back to the user. The design had three LEDs vertically and their phototransistors on the other side vertically to let the user know how much of the spice they had remaining. This updated on the app to show the user for instance if a spice is lower than 75% full, 50% full, and 25% full. From the circuit below we used a 510 ohm resistor for R1 because that gave a voltage across the LED to be around 3 volts, and R2 would be 4.17K so that the output from the phototransistor would be around 5 volts. The outputs from the phototransistor were given to the microcontroller and then from the microcontroller back to the Bluetooth device that connected to the app on the users phone to let them know how much of the spice is left in the container. Once the first phototransistor picks up the signal from the

LED transmitter, the user was able to know it is less than 75% full, and the other two optocouplers follow the same steps.

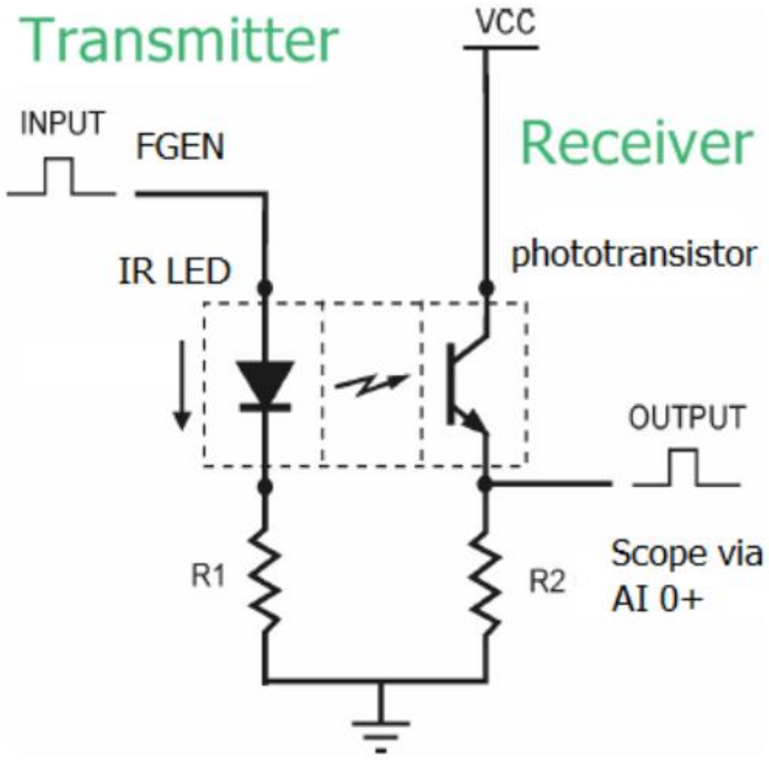


Figure 38: IR Light Sensor Circuitry

Due to the fact that we had to print out our own spice rack with a 3D printer, we couldn't implement the light sensors because the spice holders weren't transparent enough for the light sensors to work. We were also able to add a XC6201P502MR-G 6V to 5V linear regulator with a 200mA output to run the sensors on the board. This would give us enough output and allow for less power to be lost through that voltage divider.

Since the third circuit showed when a spice is less than 25% full we can add a red LED in parallel with the output signal going back to the microcontroller because the output signal back to the microcontroller had a peak of 5 volts, which was high enough to power up the LED and give the microcontroller the information it needs. The circuit below had the same resistors for R1 (R1 in circuit below) and R2 (R11 in circuit below) like the other two optocouplers, while resistor R2 in the circuit below were the exact value as R1, 510 ohms. R2 allowed the voltage across the LED to be around 3 volts which turned on once the optocoupler flashed the phototransistor that the spice is less than 25% full. Then the same line the turns on the LED will send the microcontroller the signal that the spice is ready to be refilled by communicating with through the Bluetooth device to the phone to send them a notification that the spice needs to be refilled soon.

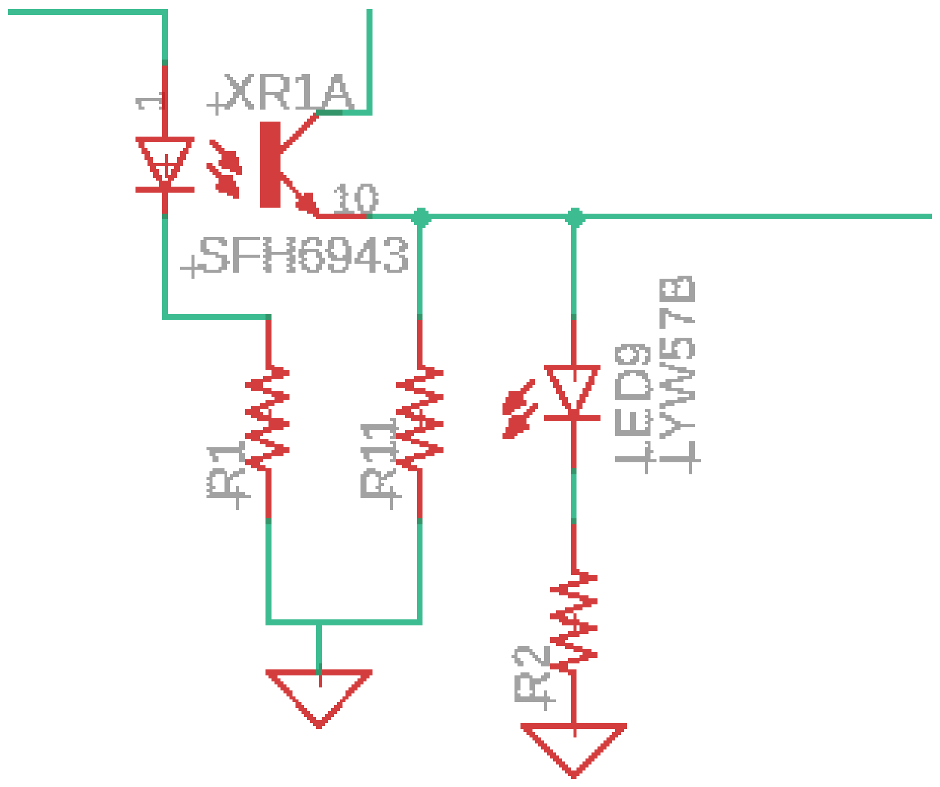


Figure 39: Third IR Sensor Adjustment Schematic

LED Light

To test these lighting fixtures, we went through each light ensuring that they all work. To ensure that it works, we installed this to a circuit. After attaching the LED's to the circuit there is a preset motion that loops through each LED ensuring that they turn on. After ensuring that each LED turns on, there needs to be a test to ensure that they work with the project implementation. This is a sending of a signal to the individual mock pods. This indication makes all but the, mock, current spice be red. The mock, current spice be toggled to display the green LED. By testing this, we are able to see what happens before being put on the project.

LED sub-system

This section will go over which LED subsystem will be utilized in the spice holder. The LED selected requirements must comply with the three most important conditions for parts selected on this design. The different elements that are necessary to be judged are power efficiency, cost and usability.

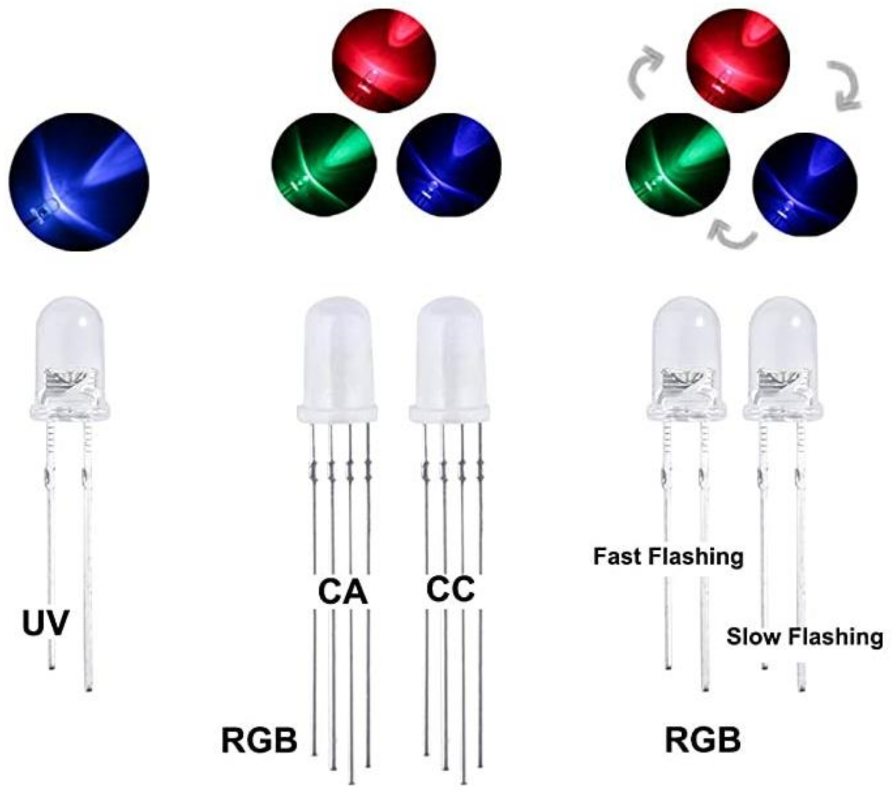


Figure 40: LEDs

These LED’s are a great way to implement the light flashing mechanism we have for the implementation of the spice holder. They are cheap to buy so this falls well in the budget. They are reliable to use and are easy to replace. These LEDs give the ability to switch colors. With the ability to switch colors, it removes the need to have multiple lights pe LED, which is ideal for indicating which spice is being used. Looking at what this LED offers makes it ideal for the spice holder implementation.

74AC11138D 3 to 8 Demultiplexer

This section will discuss how our design will get the various LEDs to light for the spice selected by the user and show them the location of the spice.

These LEDs that are above each spice would light up green when the user specifies in the app which spice, they want to dispense. In order to get the LEDs to light up correctly with the desired spice, the LEDs would have to have communication from the microcontroller and power from the power to run each LED light. The best device to use in order to turn on the LED for a given spice, the design would need a demultiplexer that would take in 4 signals from the microcontroller. These signals would have to be 3-bit signals, meaning they would act as a 1 or 0 in order to signify which LED would light up, and then the last signal would have to be an enable signal. So, the number of bits would relate to the number designated to each spice. These means that if the microcontroller sends 3 signals with the bits 001, then the LED with the first spice position would light up green. While there would only have 3 bits from the signals coming from the microcontroller, the enable allowed us to have 8 different LEDs because if the demultiplexer is enabled the signal 000 from the microcontroller turned on the LED at the spice position 8.

While we used a demultiplexer to designate each LED to turn on during a specific use, the circuit with the demultiplexer would need to allow a voltage across each LED to equal 0.7V that means we had to use a resistor in series to give the LED the correct voltage. The Input voltage has to be 3 volts since the LEDs used in our design needed an input voltage of 3 volts. When searching for a demultiplexer that allowed a voltage minimum of 3 volts, the 3 to 8 74AC11138D allowed a minimum voltage of 3 volts and a maximum voltage of 5.5 volts. While this isn’t a great component with LEDs that run on 3 volts because it only gives a current of 4 mA, but it is cheap and gives a high enough output current for a 5-volt input. This means that we had to use a voltage divider to the input of the demultiplexer from the 6-volt input that comes directly from the battery. Then we would have to add a small voltage divider for each output so that the input voltage to take the voltage down for each LED. While each output had enough current to run each LED, the divider had to have large resistors because the larger the resistor the less current it consumed from the voltage source. However, since the forward voltage of the LEDs is 3 volts, we can add a resistor at the end of the LEDs to dissipate more voltage before we return to the ground. This would be the best option because it cut down on the number of components.

The schematic below shows how the demultiplexer would fit with the LEDs to show the user the spice they have selected. It also shows the 4 signals that would come from the microcontroller that allowed the demultiplexer to be enabled and which register to turn on. The resistor values would be around 510 ohms so that it would dissipate the amount of voltage needed so that the forward voltage across the LED would be 3 volts. The enable signals would be tied together because when G1 is high and G2A and G2B are low the enable can be turned on to get the LEDs power. While signals A, B, and C can turn on the different registers, and an enable with three low signals from A, B, and C can turn on the 8th LED. While the rest of the registers would fit with their bit number.

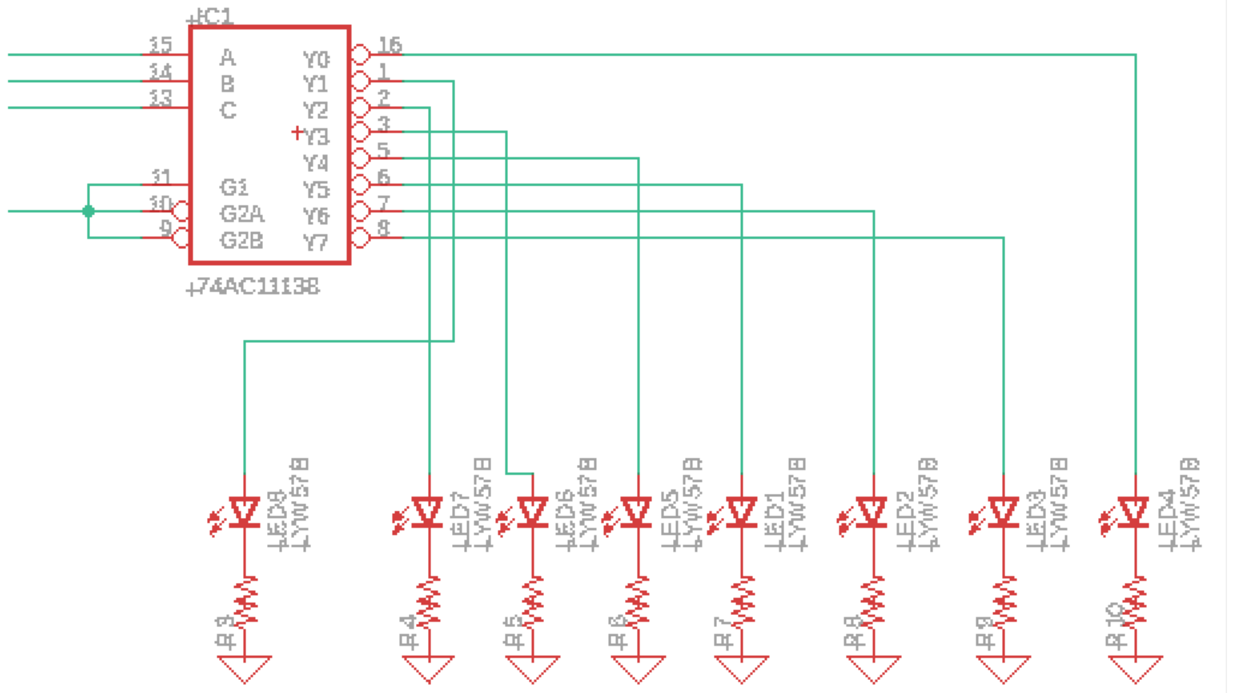


Figure 41: 74AC11138 Demultiplexer Schematic

Due to physical constraints of the spice rack we were able to get rid of the 74AC11138 because we only needed 4 lights to light since we had only 4 spices.

The voltage divider below will show how the 74AC11138D will operate and the amount of voltage that would be sent to the LEDs. This voltage divider shows a 10K resistor and a 50K resistor in series, with the output voltage coming from the node between the 10K and 50K resistor.

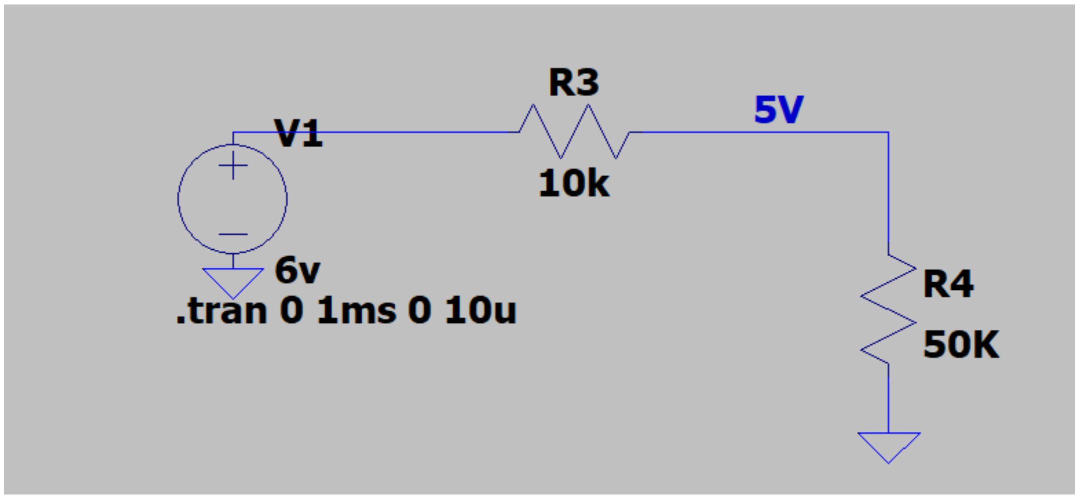


Figure 42: Voltage Divider for 5-Volt Output

This section has been obsolete because we used a 6V to 5V regulator that was implemented to decrease the power consumption of the board. See above sections for the information about the XC6201P502MR-G above.

LEDs in the schematic

In the is section the basic summary of what the different LEDs will be doing in the device will be discussed. Which LED signifies what function will be explained and why this will be done.

There are two added LEDs in the schematic. These LEDs are to indicate to the user what is going on with the spice rack, on a wireless sense. The green LED communicates to the user things about the Wi-fi usage in the device. The blue LED will serve as an indication to the user what is happening with the Bluetooth aboard the device. These two lights will serve as indications of when the device is running. With these LEDs the ability to understand the states of the device, though light indication, will help any user understand what the spice rack is doing.

The Wi-fi LED is to ensure that the user knows what is happening in the system. There needs to be two modes that is indicated to the user by the light. These modes are on and pending connection to a Wi-Fi network and connected to a Wi-fi network. The pending connection encapsulates the default function when the device is turned on. The LED will be blinking at a slow rate to indicate that it isn't connected but the Wi-Fi is on. There will also be a connected state indicator. This will be indicated by the light remaining on constantly. These will be implemented through the general-purpose input output. This will happen by setting the pin to output constant voltage when it is connected to a Wi-Fi network. When the esp32 is not connected to a Wi-Fi network, the general-purpose input's output pin will be set to output voltage in an on and off state. These indications will indicate to the user what is happening with the Wi-fi subsystem.

The Bluetooth LED will ensure that the user is aware of the state of the Bluetooth. Two modes will be available for the user to understand what is happening in with the Bluetooth subsystem. The two modes are on and waiting to pair mode. The waiting to pair mode will be displayed by a blinking light. The paired mode will be displayed by the blue LED being solid. This will be implemented by using the general-purpose input output pin being activated. The pairing mode will be implemented on the general-purpose input output by setting the pin on then exclusive or after a period of a second. The connected state will be implemented by the pin being set to be on constantly. These two states will be a good indication to the sure of what is happening in the system.

Bluetooth Communication

With Bluetooth being embedded in most mobile devices. To test this function the microcontroller chip will have a Bluetooth chip on it. This will allow the Spice holder to connect with the microcontroller. By testing if the devices pair, this is how the testing will occur. There also must be a standardization of what is the best effective distance to operate these devices. The utility model discloses an integrated Bluetooth mobile phone. The Bluetooth mobile phone has simple structure, easy operation, capability of both meeting the portable and charge-up requirements, capability of preventing loss of Bluetooth headset, and the like advantages. (赵昱晴). Knowing this about mobile devices it would be ideal to see where it pairs and using that to gauge where it disconnects this will be in the specifications for the project.

Bluetooth sub-system

This section will go over which Bluetooth chip was chosen and why. The chip chosen needs to fulfil the three most important requirements for parts chosen on this project. The different factors that need to be considered are power efficiency, cost and, usability.

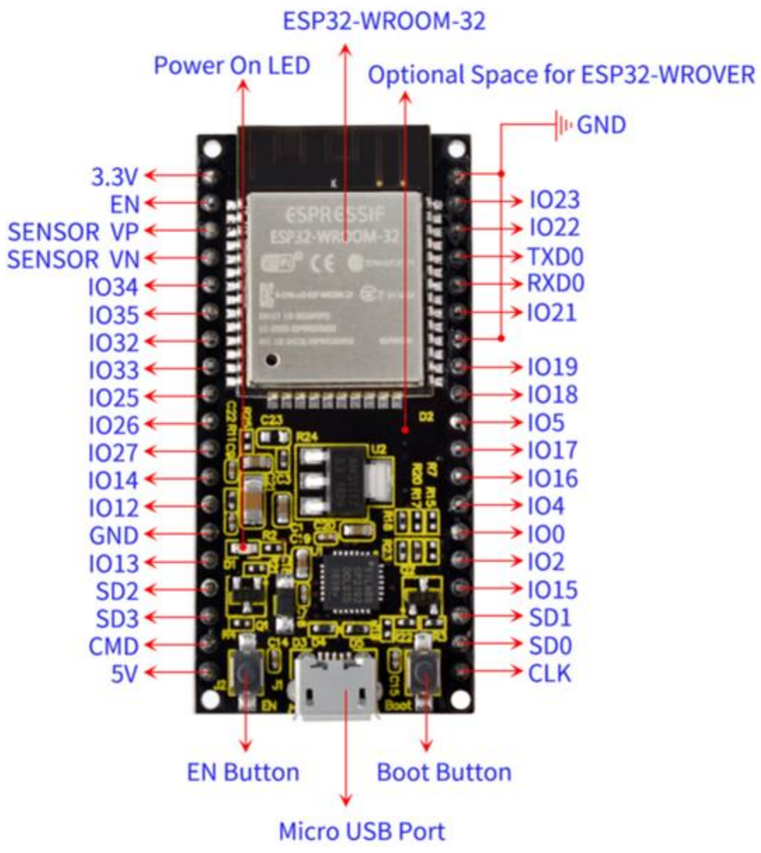


Figure 43: Bluetooth Device

For the spice holder Bluetooth function, the esp8266 is the one of choice. To implement the spice holder there needs to be three core needs. These needs are as follows: cost, power efficiency, usability. All these considerations are met by this device. The cost is within the budget allotted for the project. For power saving, the esp8266 offers an ultra-power saving mode. The ultra-power saving mode allows for the spice holder's Bluetooth standby mode to work efficiently, which is needed if there is to be a standalone solar powered device. Lastly, the usability of the esp8266 is great for the spice holder project. It allows for wireless communication for multiple applications, this is exactly what is needed for the spice holder.

Integrating Bluetooth Device

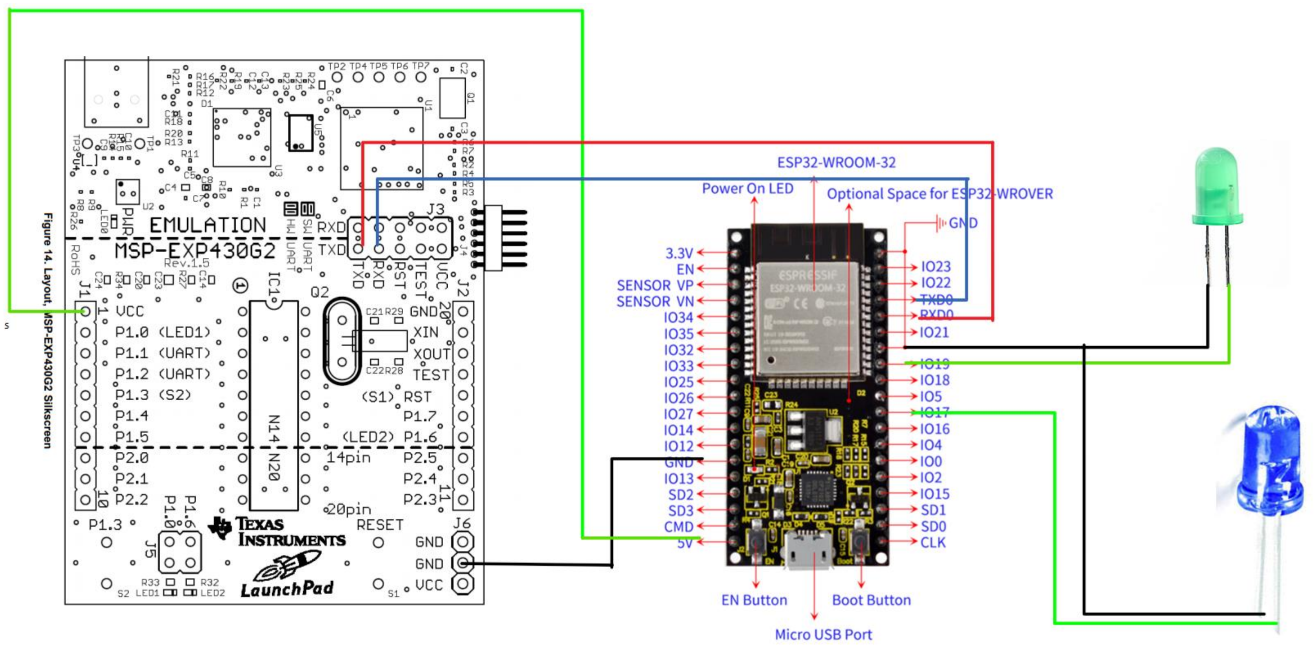


Figure 44: Bluetooth and Microcontroller Connection

Overview

In this section the integration of the wireless chip will be discussed in detail. This pertains to how it will be connected in a hardware sense. The requirements the chip needs to function will be discussed and the addition of a few parts for the user's benefit will be discussed.

Schematic of integration of wireless device

In this section the schematic will be discussed and displayed. This will help to outline the layout of the board.

Basic layout

In this section the basic layout of the board will be discussed in minor details. The overview of what each part's relevance is to the project will be viewed.

To integrate the Bluetooth device to the msp430 there needs to be pins that connect the device. The msp430 come with different pins for this layout Next, the signal was communicated to an android device using Bluetooth protocol by HC-05 Bluetooth module (Tiwari, 2017). For our project we are choosing to implement this device in the android format. This is written in Java an object-oriented language which helps with programing the device.

Integrating the esp32 gives the ability to connect both to a wireless and a Bluetooth device. This works to the advantage of the user to give the ability to talk to the board. The esp32 requires a 3.3v output and this is what is supplied. Given this integration, the esp32 is going to give the ability to program on a different software. This integration schematic shows two LED lights being connected; these are to alert the user about the what is being done. For transmission of data between the PCB and the esp32 there needs to be a Universal asynchronous receiver-transmitter (UART).This transmission of data came from the TxD and the RxD port of the PSB to the RxD0 and the TxD0 on the esp32, respectively. This is a great addition to the spice rack because it gives the ability to connect wirelessly.

This was not the way we implemented the communication. We chose to use i2c instead. This was because it was easier to set up, on our end. After doing this all of the features discussed were implemented accordingly.

General Purpose Input output

Overview

This section will go over general-purpose input output. The usage of general-purpose input output in the project will be explored. The exploration of these pins well expose how it can help with the implementation of the spice rack.

What is General purpose input output

In this section the usage of general-purpose input output will be discussed. What was this made to do and why could it be used in the spice rack will also be explored in a high-level view.

General purpose input output is an axillary part of any system. These pins are able to be controlled by the user at runtime. This meaning that these pins can be turned on and off in code as the user desires. This can be used for things such as LEDs in which the general-purpose input output can be set to on and voltage begins going through it, turning on the light. This is a powerful tool, while not an essential system can be used to create user indications or giving the ability for creativity within a device.

General purpose input output has several applications. This can be used, depending on the voltage constraint, to supply power to smaller devices. So, the general-purpose input output’s significance to the project is being able to control the motors and the LED lights on. This is not limited by just smaller devices though; multiple general-purpose input outputs can be connected to create serial communication. These can be used to communicate signals to bigger devices like temperature sensors and accelerometers. If need be these pins can be multiplexed to give a desired output. These pins are versatile and can be used for a variety of applications it is just up to the user to implement these ideas.

GPIO Role

This section will explain the usage of the general-purpose input output in the spice rack. This includes why this was chosen to be used as an implementation for each subsystem.

The general-purpose input output pins on the esp32 are simple. These pins are meant to read or write high or low voltages. Most of the digital GPIOs can be configured as internal pull-up or pull-down, or set to high impedance. When configured as an input, the input value can be read through the register. … Most of the digital IO pins are bi-directional, (Espressif Systems, 2020). With that being known, these pins can be used in a way that gives the ability to write out a high voltage. This was used for writing to LEDs, as an indication to users what is happening in the device. There are two LEDs connected to the esp32. The general-purpose input output device is also used to power the motors. With the motor being powered this way, it helps with the general function of the spice rack system. The spice rack can then tell the individual motors to run from one central unit. This use helps with making the device work more succinctly in one device rather from multiple devices together.

UART and data transmission

Overview

In this section the way data transmission will be done will be explored. The microcontroller the Bluetooth device will be connected. For the microcontroller to understand and control the Bluetooth device data must be exchanged.

Data Transmission

In this section, how data will be transmitted between the micro controller and the wireless devices will be discussed. What is the importance of this data transmission and the usage will also be explored.

In the schematic, there are wires that run between the Bluetooth system and the microcontroller unit. This is to indicate the transmission of data between the two devices. In the schematic it shows the TxD connected to the RxD0 pin on the Bluetooth device. This connection is showing that when the micro controller transmits data the wireless device receives data. The schematic also displays RxD being connected to TxD0 which displays that when the wireless device transmits data the wireless device receives data. This display shows how data is transmitted between the two devices.

Universal asynchronous receiver and transmitter is an efficient way of sending data between two devices. This is the method that is given by the esp32. UART (universal asynchronous receiver and transmitter) is a serial communication protocol. Basically this protocol is used to permit short distance, low cost and reliable full duplex communication. It is used to exchange data between the processor and peripherals. (Dakua, Hossain, & Ahmed, 2015). With this method of communication, the microcontroller and the wireless device can communicate effectively. This gives the ability to send what is needed for the spice rack to the devices, allowing for the display of the LED to indicate what is happening in the system.

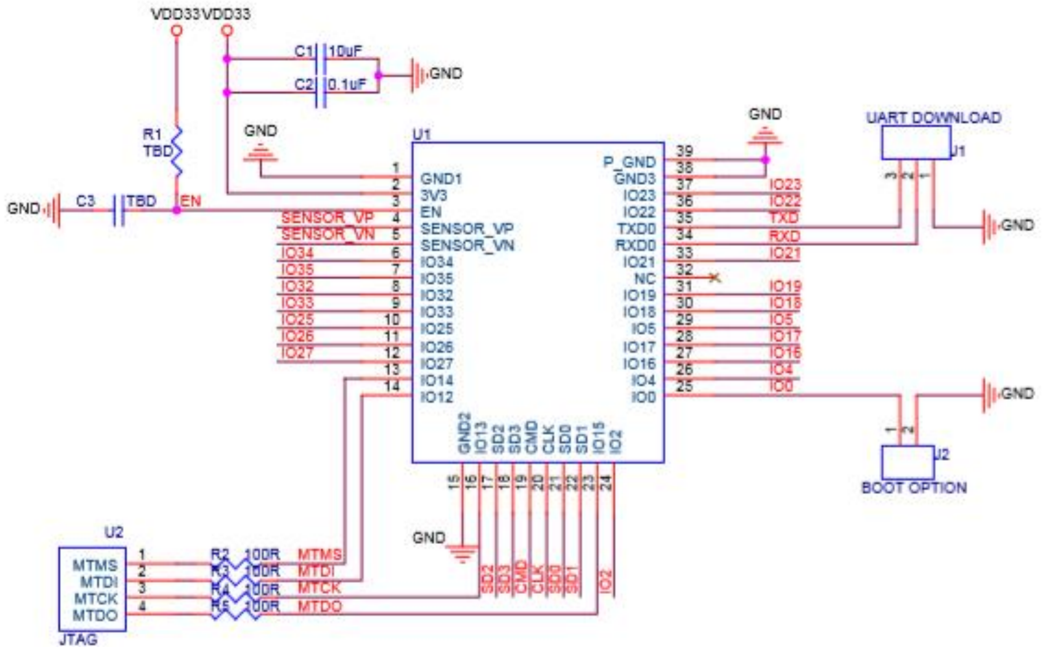


Figure 45: UART Schematic

In the figure above the UART port downloads are displayed. This shows the complete circuit of what happens to when data is transmitted. The internal ground makes it possible to properly receive current to power the system. These ports make it possible to implement what is needed for the spice rack. There needs to be a low power state, this signaled to the esp32 by UART. The UART sends the signal saying to the system that it is time to go to low power state. This makes it possible to conserve power on the spice rack. The pairing signal is also be indicated through that way of transmitting data. This helps to indicate to the user what is happening with the Bluetooth. The communication to signal to the esp32 with the Wi-Fi to connect to a wireless network was prompted by through UART giving the microcontroller power to be over the different parts in this system.

Bluetooth Power

The section will discuss how the Bluetooth device will be powered and incorporated in the circuit design.

The Bluetooth device will be powered from the TPS62147 converter like the two 3-volt motors because the Bluetooth device will need a current around 80 mA, and a 3-volt input. That fits perfectly with the TPS62147 and there doesn’t have to be any other circuitry to get the Bluetooth incorporated in our design.

Dispensing Subsystem

This section will go over the architecture and mechanical details for dispensing subsystem of the turn-table, which will be responsible for dispensing the spice, in the specified amount, held within one of the turn table’s containers as specified by the user through the mobile application or manually by interacting with the turn table.

To dispense the spice from its container, we would have the bottom of the container have 3 different holes. A gear would be attached to the bottom of the container to ensure that there is no undesired leakage of the spice when the spice rack is spinning. Once the spice is in position to be dispensed, a low RPM motor would spin the gear and would line up the holes that are on the gear with the holes on the bottom of the container.

To spin the gear, we would have needed a small lever that would get into the grooves of the motor. The motor would have then spin it to location A, where it would open the holes on the container and then move to location B to close the holes back up. Then, once the motor moves to location B, the small lever would be too short to move the gear and would be able to return to its original position. This position is where the lever can’t hit any of the gears when the spice rack is rotating, but is in a good position to move the gears on the bottom when the desired spice gets into the dispensing area.

The motor that would have be used for moving the dispenser gear with a lever. This motor is a speed reduction motor, like the motors used to move the spice rack, but without the encoder. Since this motor runs around 15 RPM, we would have been able to move the dispensing gear to dispense the spice. After the desired amount of spice is dispensed, the motor would have closed the gear system. Below is a picture of the N20, which drew a current of 50 mA when there is no load.

Figure 46: N20 Dispensing Motor

Due to physical constraints like stated before, we used a 6V Futaba servo motor to move the dispensing mechanism described in the Possible Architecture section.

Mobile Application Subsystem

In this section, we discuss the overall design and functionality of the mobile application that will go along with the spice rack.

React Native

Here, we will discuss how we plan on building our mobile application subsystem prototype using the React Native library.

To fulfill all of our project’s requirements, we needed to develop a mobile application to interface with our device. To build a prototype of the app, we intended to use the React Native framework for JavaScript in order to quickly develop the application. However, due to some complications we ran into during the development of the project, we ultimately decided to use Android Studio to develop our mobile application. We discuss our use of Android Studio in the next section.

React Native is a JavaScript framework that is used for creating cross-platform mobile apps. Because it is based on JavaScript, a language that both of the major mobile operating systems, Android and iOS, support, we can code one app that works on both of these mobile platforms, and that provides the same, or a very similar, user experience and visual design. This would allow us to more easily fulfill our time requirements, since we would only need one code base for both of the platforms.

One caveat of using React Native for both platforms is that we would need access to different development environments to compile and test our application on both of the mobile platforms. Building the Android version of the application requires a personal computer running Android Studio, an application made by Google to develop and test Android applications. Android Studio allows for the packaging of the application into APKs (Android Package, a package file format for Android software) to be run on mobile phones or emulators managed by Android Studio. This development tool can be run on computers with most operating systems available today, making compiling and testing our code for the Android version of the application accessible simple.

The iOS version, however, will be more challenging to deploy. iOS applications must be compiled by XCode, an integrated development environment for building iOS and macOS applications. XCode only runs on macOS personal computers, which is more exclusive. Our access to a macOS personal computer to develop iOS application is limited, as only one of our team members owns a Mac computer. There are some alternatives to this, including running virtual machines to run a macOS instance on other personal computer platforms. These alternatives were investigated further, and we came to the decision to use Android Studio ...

The different development requirements notwithstanding, coding the project would have still taken advantage of React Native's cross-platform support if we decided to use this framework. Most of the application's code base would have been built using React Native and JavaScript, which is accessible in most development environments available today. We therefore chose to use the React Native framework at first. Writing native code (platform-specific software meant to run on one platform or another, as opposed to cross-platform code) would likely be entirely unnecessary for this version of the project, as our application is not expected to make use of features specific to one mobile platform or another. However, it would have been possible for us to run into some instances where writing native code may be necessary. This would have been especially likely during the testing phase, as we encountered some faults that affect one mobile platform and not the other. Such faults would then be assigned to the team member that has the development resources for the respective mobile platform most readily available.

Android Studio

During the development of the project, we decided to change our main development tool from React Native to Android Studio. We came across unexpected difficulties working with React Native, and since more of our members were familiar with Android Studio.

Mobile Application Functionality

With our main development tool for the mobile application being discussed, we will now move on to the application's design. Here, we will discuss how the application was designed to be broken up into discrete pages that the user navigates through, what functionality should be available to the user for each page, and how that functionality will be achieved. The final version of the application only used one of these pages, with the main functionality of the additional page being discarded.

The application opens and starts on the main page. The first UI element on the main page shows the Bluetooth connection status with the turn table, and has a button to establish a connection to the turn table if it is not already connected to the phone.

On the main page, the user will also find a graphical representation of the turn table. This representation will be numbered, representing the spice's position relative to the front of the turn table, with number 1 representing the spice container on the front of the device, and the numbers incrementing clockwise. A mockup of this UI element is shown below.

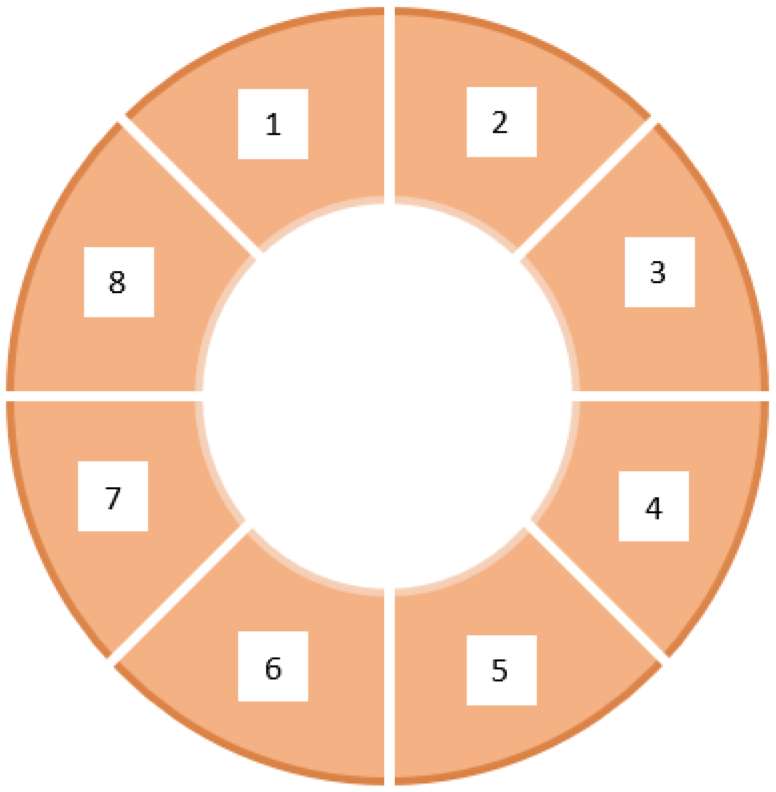


Figure 49: Spice Rack Positions

Tapping on one of the sections in this element will bring a pop-up to either tell the user which spice is in that slot or that the slot is empty. The user will be given the option to enter the name of the spice or replace the name already in this section. The names will be stored in a file on the phone's storage.

The final version of the application instead has 4 circles representing each of the spices that are contained in the device. Tapping on one of these circles shows a menu displaying options to rename spices, dispense, and other options.

Below the graphical representation of the turn table would have had a series of buttons. The first would be for requesting a spice. Here, the user would have the option to select a spice from a drop-down list and the amount of the spice wanted from another drop-down list of predetermined amounts. Once the user inputs this information, the number of the corresponding spice and the amount wanted would be sent to the turn table by Bluetooth. The user would also have the option to input this information through voice. The user would be directed to make a request for a spice by name and amount, in the form of "X teaspoon(s) of Y," where X is the amount, as a whole number, a half, or a quarter fraction, and Y is the name of the spice. The application would then convert this speech to text and process this information in a similar fashion as if the user had used the drop-down lists mentioned above. If the user requests a spice that is not in any of the containers, requests an unsupported amount (e.g., requesting more than what is left in the container), or if the user's speech is not recognized, the application would display an error message to the user and would ask the user to try again.

Lastly, the main page would include a button for searching for recipes online. Tapping the button would open the recipe page of the application, which would give the user a text field to input a search query for recipes online, and an option for the type of meals (e.g., breakfast or lunch, etc.) the user is looking for. This information would be used by the application to make an API (application programming interface) call to the Edamam API. Edamam is a company that offers nutritional information on thousands of recipes online, and the company gives developers access to an API that makes use of this information for their own applications. We would have used Edamam's API to retrieve recipes and the ingredient lists for each recipe. Once the user inputs their search query, the application would show a list of recipes with accompanying pictures (provided by Edamam) and links for the directions for the recipe. The application would also show the ingredients in the recipe that are also available in the turn table. For example, if the user finds a baked chicken recipe that requires oregano, and the user already has oregano in one of the spice containers, then oregano would be displayed as an ingredient for the recipe, along with its amount, in the application. In the final version of the application, this page was removed due to limitations in working with the Edamam API and changing our development platform from React Native to Android Studio.

Mobile Application Design

As far as the organization of the code for the mobile application, we would organize our code modules, our components, into page, UX (user experience) and communication components. This would have taken advantage of React Native's focus on user interface (UI) design and encapsulates UI elements (such as buttons, dialog boxes, etc.) to make prototyping and testing the application easier.

The diagram below displays the organization of the different components within our original design of the application. The MainPage component, for the main page of the application, would have contained the VoiceRequest, GraphicalRequest, BluetoothRequest and SpiceMenu UX components. The MainPage component would also have used the BluetoothComm component to send and receive data between the application and the turn table. The RecipePage component would have contained the RecipeSearchBar and RecipeCard components. The functionality of these components will be explained the following sections.

Since the application was eventually developed with Android Studio and not React Native, the application's UI design is organized differently. In Android studio, each activity's UI elements are organized with an associated XML file that determines the layout of the page and how the content will be displayed. We will discuss the original design below, and detail how the design was changed as the project developed.

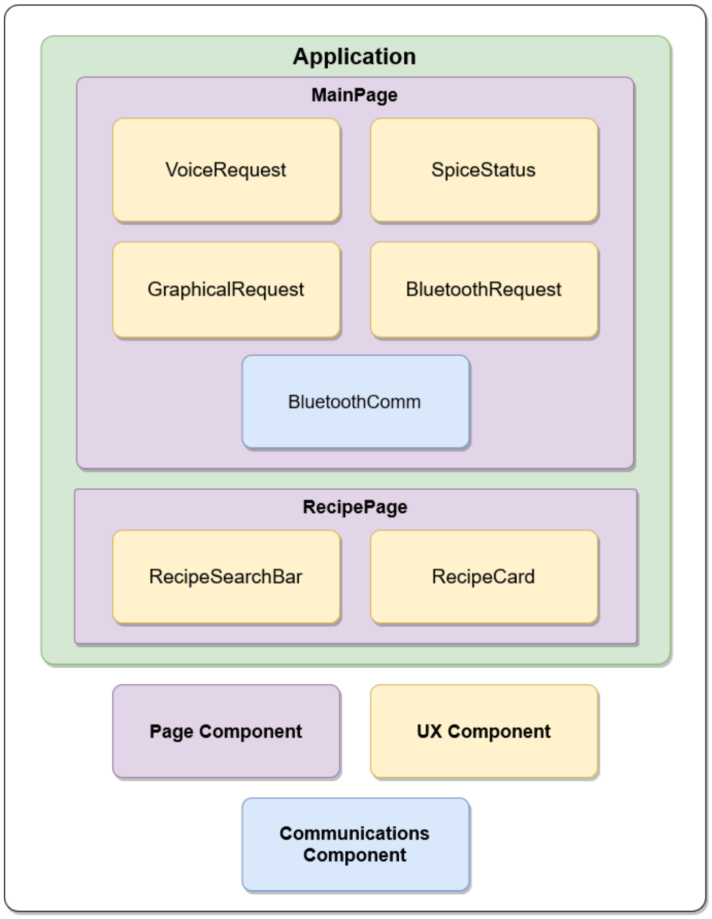


Figure 50: Component Diagram of the Mobile Application

Page Components

The page components would have provided the framework and styling for the UX components. They would have mainly served as the screens that the user interacts with and on which UI elements such as buttons and search bars will be placed. In the code, these pages would have been written as JavaScript classes that import React and React Native packages, along with any other packages necessary to implement the page’s functionality. The mobile application would have consisted of two pages: the main page and the recipe page.

The main page would have been the landing page of the app once it opens, and it would be where the user will spend most of their time when using the application. It would have contained most of the options the user would need to connect with the turntable via Bluetooth, check on the estimated spice levels in each spice container, and request spices in their desired amount. In the current version of the application, there is one main page which accomplishes the functions of selecting and requesting spices and checking the Bluetooth connection status with the device.

The MainPage class would have been mainly responsible for the styling of the UI elements that would trigger and display the UX components and enable the functionality of the application. This component would also have stored the list of spices and their amounts in the turn table, and would have handled communications with the turn table with the BluetoothComm component.

The other page component will be the RecipePage, which would have allowed the user to search for recipes online and display the results. The RecipeSearchBar component would have allowed the user to enter keywords to search for recipes online and allowed the user to refine the results with a few options on the types of recipes they would like to see. The RecipeCard component would have served to display the information for each recipe search result, including the name of the recipe and a URL link to the recipe. There would also have been a small list of some of the ingredients, with any ingredients that match the contents of the turn table being listed first. The results would have been displayed in groups of 10 cards, with the next and/or previous groups of cards being available by pressing the “next” or “previous” buttons (respectively) at the bottom of the page.

This page was eventually removed from the final version of the mobile application. Some of the difficulties that were encountered while developing this part of the application took too much of a toll on our time resources to include in the final project. Switching the development environment from React Native to Android Studio made it so that we would need a specific library to work with the JSON objects we received from the Edamam API, and the library we chose had some unexpected behaviors that we would not remedy in time. The Edamam API itself also presented some issues during development due to inaccurate documentation which made extracting information from the API more difficult than we expected. We, therefore, chose to exclude this part of our mobile application in order to finalize the rest of the application and ensure that more essential functionality, including communication with the device and selection of spices by UI usage and by speech, were included.

UX and Communication Components

The UX and communication components would have provided the UI components with pages that the user interacted with and the functionality that went along with those components. Writing these UI components as separate JavaScript files that would have been imported in by the pages would have allowed for greater encapsulation and therefore would have made testing these components in isolation much easier. These components would have been written as JavaScript classes, like the page components, and instances of these classes would have been created to be displayed within the page components.

The GraphicalRequest component would have been responsible for the creation of the graphical turn table representation in the main menu. As explained in the Mobile Application Functionality section, the user would have been able to tap on one of the sections of the turn table and see what spice is contained within it, check the approximate amount of spice, change the name of the spice and request a measured amount of the spice. All of this would have been done through a menu that pops up once the user taps on one of the sections. This component would also have included a drop-down list of spices currently on the turn table, so that the user would be able to make a request that way instead of using the graphical representation of the turn table, if they so wished.

This menu would have been the SpiceStatus component. This would have contained the popup menu that shows information about the spice in the selected container of the turn table and would have allowed the user to change the name of the spice or request it. If the name of the spice is changed (or it is given a name for the first time as part of the setup process of the turn table), this change would have been reflected on an array stored as part of the MainPage component's state. The request for a spice would have been made by sending a request through the BluetoothComm component. In the final version of the application, the graphical representation of the device's spices, which are circles with the names of each of the spices contained in the device, can be tapped to give the user the option to request a spice in the amount they would like in tablespoons or teaspoons. The amount of spice that is left in the container would not be indicated, as we could not implement a way for the device to detect the amount of spice left in the container.

The VoiceRequest component would have provided the user with a button to press and hold to make a request for a spice through voice. Because we lack the expertise to implement more sophisticated solutions, we would have asked the user to make the voice request in a very specific format (as explained in the Mobile Application Functionality section). We would have used a third-party React Native library that would allow the application to make use of the voice-recording services managed by the mobile operating system and implements a speech-to-text system, giving us a string representation of what the user said. With this string available, we would have parsed the string and made a request for a spice in a similar fashion as the SpiceStatus component. If the user's speech is not understood by the application, or it is processed into an unexpected format, the speech-to-text results would have been shown as a popup notification in the application and it would have asked the user to try again. From there, they could try to make another voice request or go back and try the other methods of requesting spices. The final version of the application works similarly, using Google’s voice recognition library to return a string containing the words that the user spoke. In this version, we do not require the user to speak in a specific format, as we simple search the string for a number, the word “teaspoon” or “tablespoon” and the name of a spice that is contained in the device.

The BluetoothRequest and BluetoothComm components would have been used to send information to and from the turn table via Bluetooth and to manage the connection with the turn table while the application is being used (respectively). We chose to separate these components from one another as a preliminary precaution, as we expected the maintenance and setup of the Bluetooth connection to require more complicated code and dedicated testing. The BluetoothComm component would have been responsible for establishing the connection with the turn table by scanning for the turn table and checking its status. Once the turn table is connected to the application, the BluetoothRequest component would have sent read/write commands to the turn table to retrieve the names and amounts of the current spices on the turn table. It would have then handled the requests for spices from the application. This component would also have provided a UI element that shows the connection status of the turn table at the top of the main page of the application. In the final version of the application, the main page activity uses Android’s Bluetooth connection packages to establish and maintain a connection with the device. Because we are working with native code instead of React Native, establishing and maintaining Bluetooth connections is a much more straightforward process, thus a separate activity for Bluetooth connections are not required.

The RecipeSearchBar component would have been the first UI element present on the recipes page of the application. This component would have included a search bar and an option menu to search for recipes online using the Edamam API. The options menu would have given the user filtering choices for the type of meal desired (breakfast, lunch, etc.) and certain dietary restrictions (vegan, low-carb, etc.) that they may subscribe to. These filtering choices would have been in the form of checklists in the option menu. We would have used the information the user enters to create an API call, retrieved the recipes from the result and display them in a list in the recipe page.

The recipes themselves would each have been displayed using the RecipeCard component. This component would have placed the information retrieved from the API call into distinct UI card elements, which would have been listed as results from the recipe search. The card would have included a small picture of the dish, the name and source of the recipe, some of the ingredients and a URL link to the full recipe. This information would have been retrieved from the JSON object that is returned by the API call made by the RecipeSearchBar component.

Dependencies

The functionality of this mobile application requires the use of third-party libraries to give us the tools to implement this functionality. We will discuss the libraries we expect to use for this project. This list has changed as we developed the application, and we will detail the changes below.

One of the first kind of library we would likely have used is a UI library. This kind of library would have given us many different React Native components that focus on how to display text and data in an organized and consistent way. There are some UI components that come with React Native but using a separate UI library allows for more options without too much more coding. This would have allowed for an easier and faster process in creating a working prototype of the application. We first decided to use the Prime React library, as it provides many UI components that will be relevant to the project. For example, we can use the DataView component in the recipe page to set up the list of RecipeCard components for the recipes. Due to the nature of React Native, we can easily change the library that we use for the UI components or use UI components from multiple libraries. However, due to changing our development platform from React Native to Android Studio, this type of library became unnecessary. All of the UI classes that Android Studio includes fulfilled the functionality we needed for our application.

We also used a third-party voice recognition library for the VoiceRequest component. This library was responsible for implementing the speech-to-text functionality we use when giving the option to use speech to request a spice. We can therefore avoid implementing a speech-to-text algorithm ourselves, which would involve time and research that is outside the scope of this project. This library would have worked across both of the major mobile platforms, but some issues with permissions may occur that would be individual to each platform. Since we focused exclusively on the Android platform, this became an irrelevant issue for our project. Both mobile platforms implement safety precautions that prevent use of certain hardware and software services from the phone (such as access to the microphone or camera) to prevent malicious use of the services and protect the phone user’s privacy. There are some situations when using this library that would require working with these permissions, which are platform-specific. The documentation for this library does address this issue and proposes some potential solutions, so we will consult that if this concern arises during development.

We also would have used a third-party Bluetooth library to implement Bluetooth functionality in the application, since it is not natively supported on React Native. This library works specifically with the Bluetooth Low Energy (BLE) standard, which our ESP32 chip supports. This library would have given us the tools to create the BluetoothComm and BluetoothRequest components, allowing us to maintain a Bluetooth connection and send data to the turn table. However, Bluetooth connection is natively supported by Android Studio, which makes this third party library unnecessary.

Lastly, we would have been using the application programming interface (API) provided by Edamam to retrieve the ingredients from recipes online. We chose this specific API because it gives us free access to this API if we stay within a range of number of API calls. For the free tier, this range is 5000 API calls per month, which is likely more than enough for what we would need to test and run our application. There are some other limits that are in place, such as limited options for filtering by dietary restrictions and no options for filtering by cuisine or dish type (sandwiches, sauces, etc.). However, the functionality that the API does provide will be enough to implement the important feature of the recipe page, namely fetching the names of ingredients from recipes online.

The API is documented online in Edamam’s website. In summary, we would have made an API call by making a GET request with a search keyword from the user and any restrictions the user placed on the recipe results through the filters. The API returns a JSON object with an array of Hit objects, which denote the search hits based on our previous call. These Hit objects contain Recipe objects, which contain the information we will use in the RecipeCard components in the recipe page. This includes an array of Ingredient objects, which have the ingredient name, a floating-point number of the quantity, and the name of the measurement the quantity is in. We would have used this ingredient information to choose which ingredients to display in the RecipeCard component. We would then have compared the list of ingredients in the Ingredients array with the list of spices in the array of spices stored in MainPage that contains the spices currently in the turn table. If any spices match, they would have been displayed first and in bold type, to emphasize that the user can use this recipe with the spices they currently have in the turn table. However, it was determined during testing that the JSON objects returned by the Edamam API were structured in a very different way than as detailed in Edamam’s documentation. This is one of the reasons that factored into our decision to exclude the API functionality from our final mobile application.

Project Prototype Testing Plan

This section will discuss the testing process and how the spice rack system will be tested in its various sections.

Hardware Testing Environment

This section will discuss how we will test each major component and the several different subsystems. It will also discuss the different needs for each test and what we plan to accomplish with the certain tests.

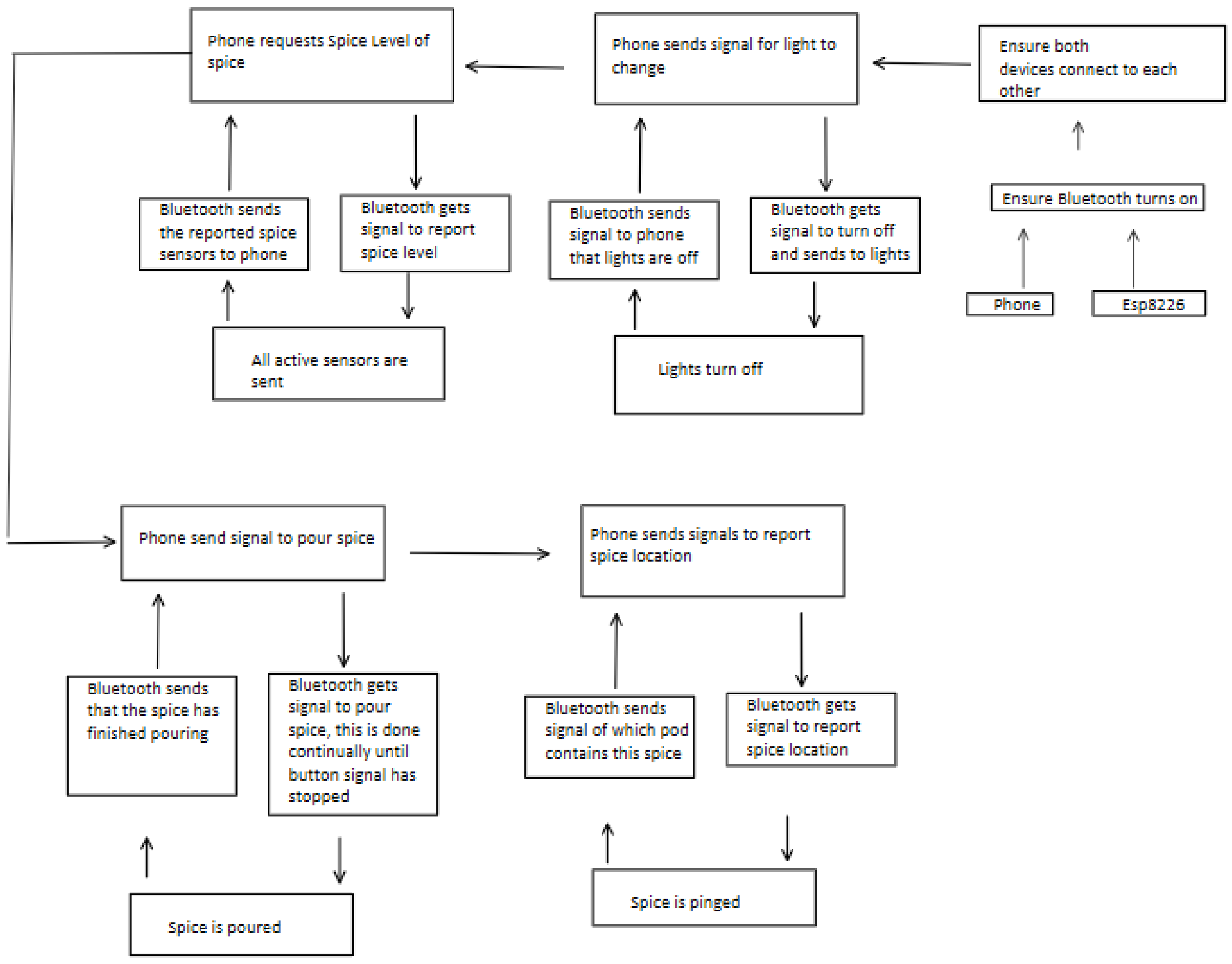
Spice Rack Rotation

In this section, we will discuss how the Machifit motors will be tested to move the spice rack to the correct spice desired and how we will test the communication from the motors to the microcontroller through the motor driver.

In order to test the Machifit motors we would need a power source to supply 6V to the Machifit motors for them to run properly. Then we would need to correctly connect the encoder to the microcontroller where we can use the feedback information to stop or keep the motor running through the enable in the L298N motor driver. Once getting the feedback information from the encoder to the microcontroller, we would need to hook up the enable and disable into the microcontroller through the enable A and enable B pins. Then by setting up the logical voltage on the motor driver to the microcontroller that can supply around 3.3 volts, we are able to communicate from the motor. Once getting everything set up between the motor and the motor driver, and the encoder connected to the microcontroller we would be able to read the feedback from the encoder once we get the motor turning. By understanding the encoder positioning from reading the output of the encoder, we can then code the microcontroller to stop at certain locations.

Bluetooth Testing flowchart

This section will go over the Bluetooth testing that will be implemented. This test is to validate that the Bluetooth works with the spice holder system.



Phone requests Spice Level of spice
Phone sends signal for light to change
Ensure both devices connect to each other
Bluetooth sends the reported spice sensors to phone
Bluetooth gets signal to report spice level
Bluetooth sends signal to phone that lights are off
Bluetooth gets signal to turn off and sends to lights
Ensure Bluetooth turns on
Phone
Esp8226
All active sensors are sent
Lights turn off
Phone send signal to pour spice
Phone sends signals to report spice location
Bluetooth sends that the spice has finished pouring
Bluetooth gets signal to pour spice, this is done continually until button signal has stopped
Bluetooth sends signal of which pod contains this spice
Bluetooth gets signal to report spice location
Spice is poured
Spice is pinged

Figure 51: Bluetooth Testing Flowchart

Crushing/Dispensing Fresh Spice

This section will demonstrate how we will test the crushing and dispensing of the fresh spice application of our design. Along with dispensing regular spices that are already crushed up.

In order to optimize our testing, we needed to use a 3-volt power supply along with a controlling signal from 3.3 volts to 5 volts, since the microcontroller gave commands between that range. Once getting the voltage sources set, we needed to plug the MM10 motor into one of the L9110S motor driver's output. Then plug the 3-volt power supply into the L9110S pins 1 and 2, and the command signal into pin 4 (needs a ground at pin 3 since pins 1 and 2 modulated by the driver to make the motor spin in different directions). Once getting the motor to spin clockwise and counter clockwise by changing the voltages going through pins 1 and 2 (making pin 1 take 3 volts and pin 2 take 0 volts makes the motor spin clockwise, and vice versa when the pins have opposite voltages), we can then implement the microcontroller to input the same command signal to get the motor to run the board.

In order to see if the L9110S can run two motors at a time, since it had to run the MM10 and the N20 to dispense spice that is already ground up and to crush fresh pepper to dispense. Once getting the MM10 motor running correctly with the motor driver and microcontroller, we needed to add the N20 by connecting the microcontroller to pins 5 and 6 and connect the N20 to the L9110S through the output pins 3 and 4.

Mobile Application Testing

In this section, we will discuss our testing for the testing of our mobile application. We followed standard software engineering procedures for testing software projects, taking advantage of the object-oriented programming paradigm implemented by the programming languages used in the development of our mobile application, namely JavaScript for React Native and Java and Swift for development in Android Studio and XCode, respectively.

To follow common software testing procedures, we tested our application by the smallest code unit first, and then testing the groups of code units increasing in size, then testing the application as a system. This system testing ensured that the app functions without any faults occurring, and that all the requirements for the application are fulfilled. Below is a diagram of a typical software testing protocol.

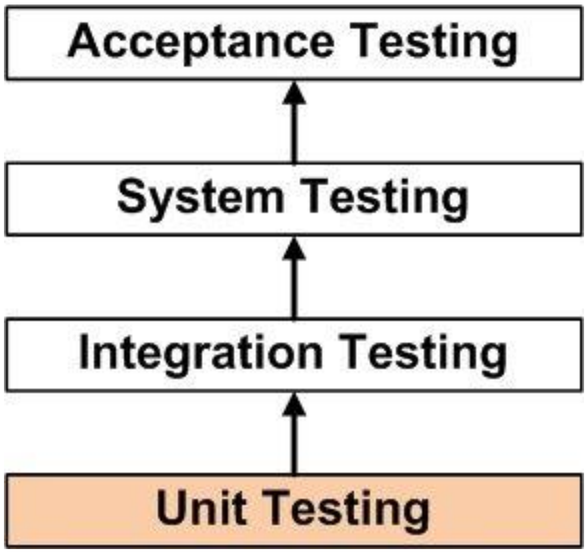


Figure 52: Mobile Application Testing

In JavaScript, the smallest code unit is the function. With the React Native framework, which is built on top of the React library, JavaScript functions and classes are both abstracted into different types of what React refers to as components. Therefore, we would have started our testing process with unit testing by confirming the functionality of each of the components we write in the code. This would entail ensuring that each component returns the proper value and that it is displayed properly on the application's user interface. We would have used available React Native testing libraries to conduct snapshot testing, which tests each React component in isolation with one another.

After ensuring the functionality of the components, we would have tested how these components come together in each page of the application. We would emulate each of the pages separately, checking that each component is rendered correctly on the app and that the interactive elements respond properly to user input. Using a test renderer library, we could have rendered the components using pure JavaScript without needing a mobile environment to run our app. This means we could quickly check the user experience and functionality of the application in an isolated environment without worrying about any platform-specific implementation details or issues. This also ensures that we could isolate and correct as many faults as possible before compiling our application into a test package to be used on an emulator or phone. This could minimize development time that would otherwise be used in compiling the code into application packages multiple times. There are other tools and libraries available that would have been able to assist us on this particular stage in testing of our application.

However, we had to change our testing procedure slightly to fit the use of Android Studio instead of React Native. Java is structured in a much more traditional manner from an object-oriented perspective, with each Java file needing at least one class and with no abstraction that equates functions and classes like what is present with the React framework. Therefore, we tested the functions in Android Studio by ensuring that we receive the correct outputs from the functions in the application. We then ensured proper displaying of the information and content in the application by building test versions of the application and running them on the Android devices that some of our group members had available.

Here, we would also have tested each component's communication with the device. Some components would have handled the Bluetooth communication from the phone to the device, sending and retrieving data. We tested if the user can establish a Bluetooth connection to the device, and if data could be sent to or from the device properly. This involved sending "dummy" messages from the phone and having the device respond in a predetermined way, and vice versa. With the change of development environment, we tested this functionality not as a separate class, but as a function that is contained within the main page activity.

When we received satisfactory results from testing each of the application's pages, we would have moved on to test how the pages interact with each other. This would have mainly consisted of making sure that the transition from one page to another is functional and convenient for the user, and data is properly transferred from one page to another. What this specifically entails will depend on each page. However, this was never tested as we changed the design from two pages to a single main page.

At this point of testing, this should ensure that the application can run without any major failures, and we tested the application as a system to ensure that all the functionality and other types of requirements are fulfilled by the application. This was the acceptance testing portion of our testing plan. We compiled the application and ran it on Android phones to make sure that the application functions in a real-world setting.

Project Prototype Construction

Roughly, the project's design of the project is as follows. There will be 8 or so spice containers attached to a rotating platform. The platform's rotation will powered by two motors, one for each direction, while a third motor will power a grinder on the front face of the turntable, where the spices will be poured out. These motors will be controlled by a microcontroller on the turntable.

The spices will be poured into a clear cylindrical cup that we will provide. This cup will allow for infrared signals to pass through so that we can measure how much of the spice has been poured out. The user will be able to select different amounts using the app.

There will also be light sensors on the spice containers to indicate how much of the spice the user has left. These sensors will be made to approximately measure the amounts of spices in quarters. When the spice level reaches the lowest quarter, an alert is sent to the user through Bluetooth and the app will send push notifications to the user about the spice’s status.

The turntable will be powered by a battery, which will be charged through a solar panel on the top of the device and through a wall plug.

Final Coding Plan

The code that will control the selected CPU, either one that is used, will generally follow the following pattern: First, there will be an initialization phase where the CPU will turn on, and all peripherals will be turned on also. Following this, the CPU will trigger the Wi-Fi connection to communicate with the mobile app, which will then respond with the list of currently loaded spices. This list will be modifiable, and will be updated as it changes by the app. Once this is performed, the main program loop will be entered. From here, the CPU will then listen for inputs coming from a Raspberry Pi, which will be used for visual detection. This visual detection will detect hand gestures that will serve as commands (such as rotation or dispensing), and will then trigger an interrupt on the CPU. Aside from this, the Raspberry Pi will also listen to voice commands, and will similarly trigger an interrupt based upon them to the processor. This interrupt will then control the main mechanism. Aside from this, however, the main mechanism will also be controlled via the app, so there will be a secondary phase to the main loop that listens for an interrupt from the Wi-Fi chip, that will also control this mechanism. Regardless of which triggered the interrupt, the CPU will then trigger the mechanism until the selected spice is aligned with the dispenser mechanism, and will begin dispensing. Alternatively, should there be a button press to the manual dispenser button, whatever spice is currently aligned will be dispensed. After this, a light sensor connected to the spice container will be triggered. If light is visible, it will be assumed that the spice is running low, and thus the CPU will trigger the Wi-Fi chip to communicate with the app. The app will then receive a push notification, and will also trigger, if connected, an Amazon Alexa alert so that the user knows to purchase more of the spice. After this is done, the main loop will be repeated. In other words, the loop will do nothing, and the CPU will remain offline for lower power consumption, unless an interrupt is triggered by the Raspberry Pi or by the mobile application.

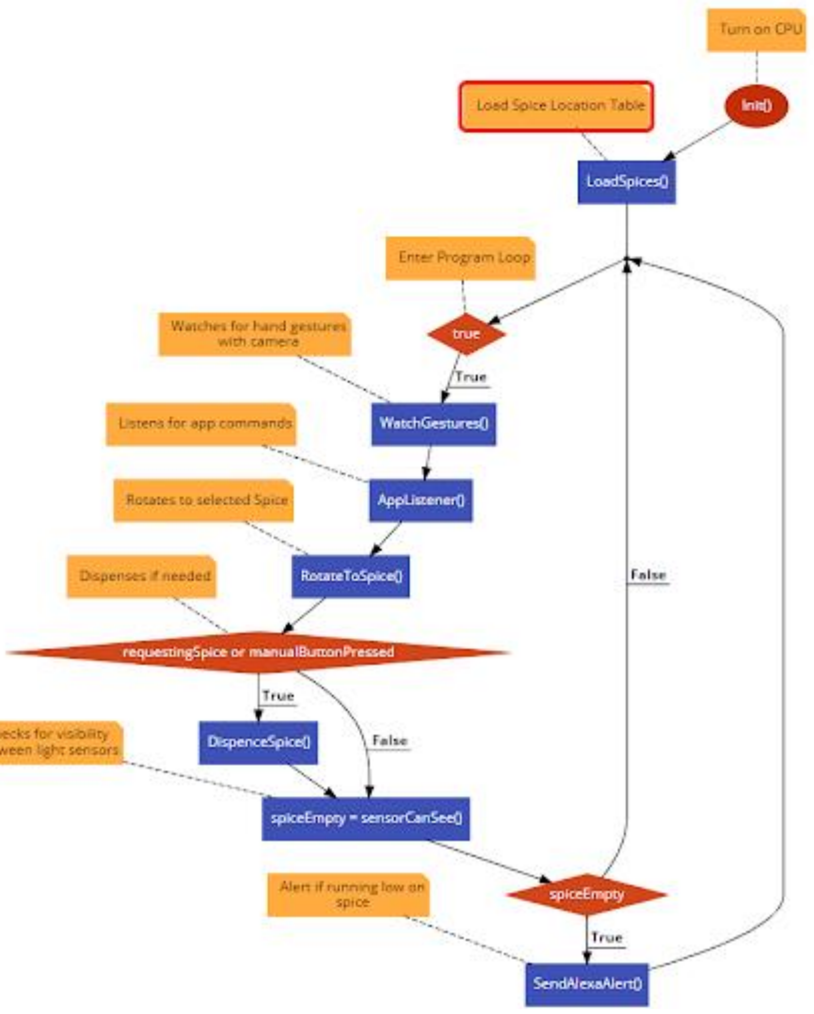

Turn on CPU
Init()
Load Spice Location Table
LoadSpices()
Enter Program Loop
true
True
Watches for hand gestures with camera
WatchGestures()
Listens for app commands
AppListener()
Rotates to selected Spice
RotateToSpice()
Dispenses if needed
requestingSpice or manualButtonPressed
True
False
DispenceSpice()
checks for visibility between light sensors
spiceEmpty = sensorCanSee()
False
Alert if running low on spice
spiceEmpty
True
SendAlexaAlert()

Figure 53: Software Flowchart

Administrative Content

Through the Administrative Content, we will discuss the Business side of our design. This means deadlines and expense will be discussed so that the design process doesn't take too much of the time away from implementing a design and producing an actual project.

Milestone Discussion

This section will discuss some check points for our design process and certain things that need to be done in order to create our design.

While most of our various milestones are TBD, to be disclosed, due to the fact that the majority of them are going to have to be pushed back, changed or just depend on the day given by the professor. The key milestone dates that we have kept to and have accomplished can be seen in the table below. These milestones serve to give the team goals to look towards completing. Given that the goals on the dates that will be outlined are accomplished in a timely fashion, the team had an edge with ensuring the spice rack is a success. There is a checklist seen below that we used in order to start our planning process and understand our needs going forward.

Timeline				
21	Create Prototype	3/13/20	3/20/20	Yes
22	Test and test	5/14/20	6/1/20	Yes
23	Final Modification	7/4/20	7/6/20	Yes
24	Peer Presentation	6/12/20	6/12/20	Yes
25	Final Report	4/21/20	4/21/20	Yes
26	Final Presentation	7/21/20	7/21/20	Yes

Figure 55: Milestone Discussion

1. General creation of the concept product
2. Part selection and general planification
3. Finalization of the dispensing mechanism design
4. Finalization of the app design
5. Completion of the main CPU programming
6. Creation of the mobile application
7. Addition of Alexa reminder functionality
8. Design and creation of the custom PCB Board
9. Adaptation of CPU Program to work on custom PCB Board as well as development board
10. Construction of mechanisms and parts for outer shell

11. Assembly and wiring of internal components
12. Assembly of external shell.
13. Final testing

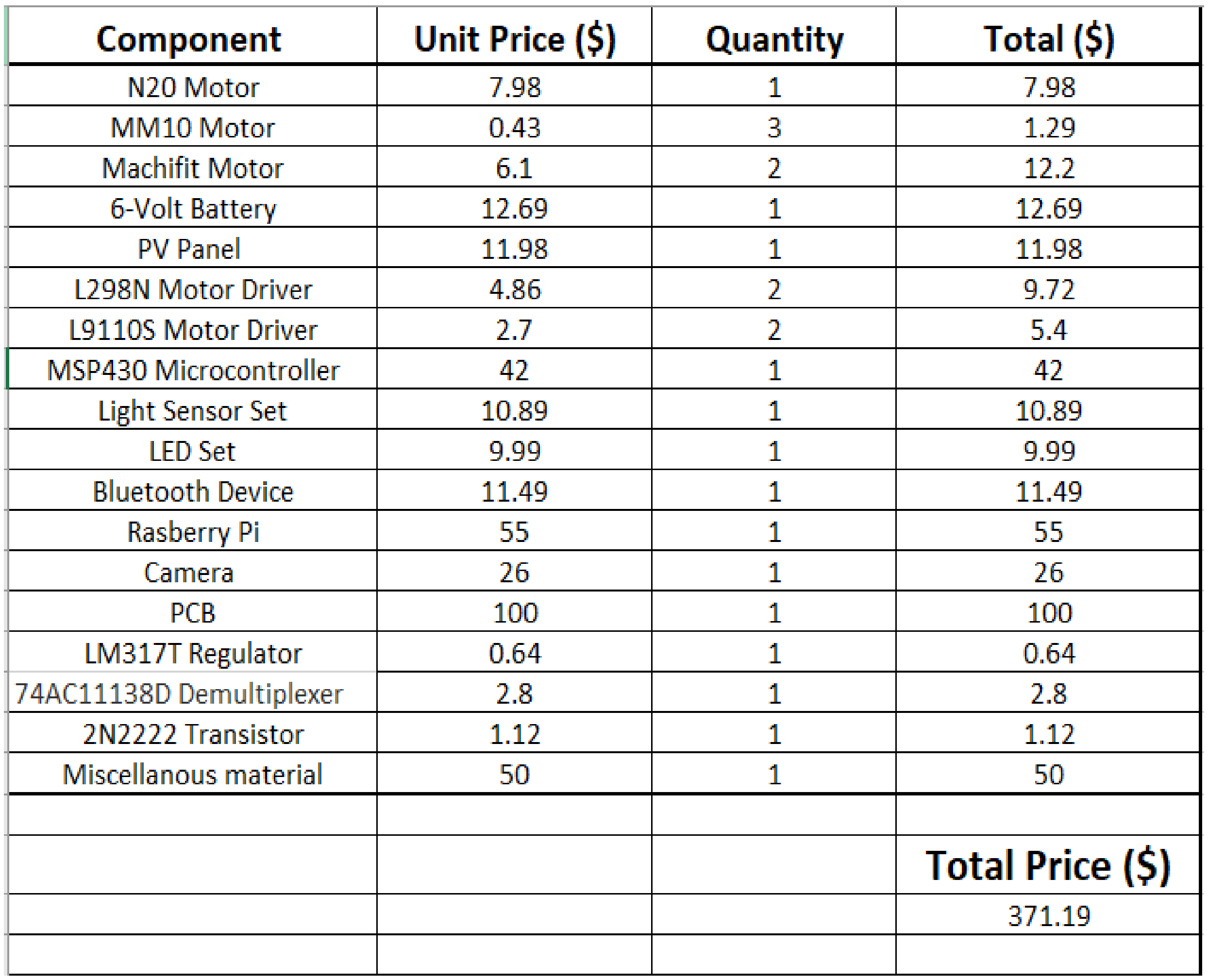
Budget and Finance

This section will discuss the budget of the spice rack system and what the major expenses will be coming from. Also, the budget will give a good depiction on if we are spending too much on the product.

Going through the budget of our design, the most expensive items are the Raspberry Pi and the camera for the visual specifications for our design. These take up 38% of the budget seen below. These components might cost a lot, but they are essential for what makes our design so different from the rest that are in the market right now. If we didn't have these components, we would have a device similar to the devices seen in the market right now. The next pricey component on our board is the MSP430 microcontroller, which facilitated the communication of the board and work as the central processing unit. It is the most important feature on the board since it communicates from the App to the system and communicates to the Raspberry Pi. The MSP40 only takes up about 20% of the budget, which is a lot but necessary for the design because it communicated to all components of the system.

The cheapest stuff in the system is the MM10 motors because they are just regular 3-volt motors that run at a fast, constant speed. Then the motor drivers that drive the various motors on the system are also pretty inexpensive, even though they have a large part in the system. The motors and the motor drivers, including the extras take up about 18% of the budget. This is a good number since the motors aren't as essential as the power supply or the communication on the board.

While all the budget of the spice rack system can be seen below, it shows that our design may be expensive compared to its competitors, it has vastly more features and quality that make up for the price.



Component	Unit Price ($)	Quantity	Total ($)
N20 Motor	7.98	1	7.98
MM10 Motor	0.43	3	1.29
Machifit Motor	6.1	2	12.2
6-Volt Battery	12.69	1	12.69
PV Panel	11.98	1	11.98
L298N Motor Driver	4.86	2	9.72
L9110S Motor Driver	2.7	2	5.4
MSP430 Microcontroller	42	1	42
Light Sensor Set	10.89	1	10.89
LED Set	9.99	1	9.99
Bluetooth Device	11.49	1	11.49
Rasberry Pi	55	1	55
Camera	26	1	26
PCB	100	1	100
LM317T Regulator	0.64	1	0.64
74AC11138D Demultiplexer	2.8	1	2.8
2N2222 Transistor	1.12	1	1.12
Miscellanous material	50	1	50
			Total Price ($)
			371.19

Figure 56: Budget Spreadsheet

Bill of Materials

In this section, we list all of the materials that will be bought for the projects and their components. This list is displayed in the table below.

Title	**Quantity**	**Title**	**Quantity**	**Title**	**Quantity**
0.0Ω Resistor	11	10 pF Ceramic Capacitor	4	L9110 Motor Driver	2
1Ω Resistor	1	33 pF Ceramic Capacitor	3	Diode	8

27Ω Resistor	2	1nF Ceramic Capacitor	1	40V 120mA Diode	2
100Ω Resistor	1	1.1nF Ceramic Capacitor	1	TPS62120 DC/DC Converter	1
240Ω Resistor	1	3.3nF Capacitor	1	TPS62147 DC/DC Converter	1
390Ω Resistor	2	100nF Capacitor	2	2N2222 Transistor	1
470Ω Resistor	4	0.1µF Capacitor	3	45V 100ma Transistor	1
510Ω Resistor	12	0.1 µF Ceramic Capacitor	14	8KB Flash	1
820Ω Resistor	1	0.22 µF Ceramic Capacitor	2	128KB Flash	1
1.1kΩ Resistor	1	0.47 µF Ceramic Capacitor	1	0.02A 15V Tactile Switch	3
1.4kΩ Resistor	1	3.3µF Capacitor	1	128KB FRAM	1
2.2kΩ Resistor	1	4.7µF Capacitor	2	20V 820 mA MOSFET P-Channel	1
3.30kΩ Resistor	1	4.7 µF Ceramic Capacitor	3	3.3V Regulator Board	1
4.17kΩ Resistor	3	4.7 µF Tantalum Capacitor	2	32.768 KHz Micro Crystal	1
4.7kΩ Resistor	2	10µF Capacitor	2	4 channel IC Array	1
6.80kΩ Resistor	1	10 µF Tantalum Capacitor	2	4.00 MHz Resonator	1
10kΩ Resistor	5	22µF Capacitor	3	74AC11138D Demultiplexer	1
22kΩ Resistor	1	470µF Capacitor	1	ESP32	1
33.0kΩ Resistor	1	1mF capacitor	1	Ferrite Core 241 Ω	1

47.0kΩ Resistor	3	LED	8	IC Switch	1
50kΩ Resistor	1	Red LED	2	LCD Screen	1
100kΩ Resistor	1	Greed LED	2	LM317 DC/DC Converter	1
150kΩ Resistor	1	Connection Header 2Pos	2	Male/Female Connection Header	2
180kΩ Resistor	2	Connection Header 3Pos	2	Nylon Standoff	4
220kΩ Resistor	6	Connection Header 14Pos	1	OPB100Z Optic Coupler	3
240kΩ Resistor	1	2.2µH Inductor	2	Optical Emitter/Sensor Pair	1
510kΩ Resistor	1	18µH Inductor	1	Test Point	9
560kΩ Resistor	1	L297 Motor Driver	1	USB A to Micro B Cable	1
1.00MΩ Resistor	1	L298N Motor Driver	1	USB B Connection Receiver	1

Conclusion

We believe our design is not only capable of meeting the market demand for smart kitchen appliances, but is built in such a way as to be safe, and to be able to be extended in the future, leaving much room for growth. Our system should be constructable in many circumstances, thanks to our open design, which means we should be able to achieve this project's objectives even with the limitations of lacking dedicated laboratories in which to work on. Further, our software is also made to be updateable, leaving room for growth in the future which might be achieved in the lifetime of the product. Additionally, the low cost aspect of the design is likely to increase market interest. Overall, we believe our design to be intelligent, powerful, and reliable.

Appendices

This section will show the citations and datasheets from all the components and research done to get our design to function properly.

Appendix A – Citations

Alwayslevel, and Guido. “6 Volt 5 Ah Sealed Lead Acid Rechargeable Battery with F1 Terminal.” *BatteryMart.com*, www.batterymart.com/p-6v-5ah-sealed-lead-acid-batteryf1.html.

Bachani, Mamta, et al. "Performance Analysis of Proximity and Light Sensors for Smart Parking." *Procedia Computer Science* 83 (2016): 385-392. 18 3 2020. <https://sciencedirect.com/science/article/pii/s1877050916302332>.

Banggood.com. “Φ80MM 6V 2W Round Style Polycrystalline Solar Panel Epoxy Board Electronic Components & DIY Kits from Electronics on Banggood.com.” *Www.banggood.com*, usa.banggood.com/80MM-6V-2W-Round-Style-Polycrystalline-Solar-Panel-Epoxy-Board-p-1379265.html?gmcCountry=US¤cy&createTmp=1&utm_source=googleshopping&utm_medium=cpc_bgcs&utm_content=frank&utm_campaign=pla-usg-rm-all-purchase-pc&gclid=CjwKCAjwsMzzBRACEiwAx4lLG-08L7cmp8wEfJEx52I7cyNsSP00cHYmDvVbZh4stmNUJFcBMLUh2RoCsrYQAvD_BwE&cur_warehouse=CN.

Banggood.com. “Machifit 25GA370 DC 6V Micro Gear Reduction Motor with Encoder Speed Dial Reducer Mechanical Parts from Tools, Industrial & Scientific on Banggood.com.” *Www.banggood.com*,

Bohn, H., A. Bobek and Frank Golatowski. "Bluetooth device manager connecting a large number of resource-constraint devices in a service-oriented bluetooth network." *Lecture Notes in Computer Science* (2005): 430-437. 19 3 2020. <https://link.springer.com/chapter/10.1007/978-3-540-31956-6_51>.

Borenstain, Shmuel I. *Phototransistors For Long-Wavelength Infrared*. 1991. 18 3 2020. <https://ntrs.nasa.gov/search.jsp?r=19910000193>.

Challa, Vidyu. “How to Select the Right Battery for Your Application? Part 1: Important Battery Metric Considerations.” *PCB Design Software, Design Analysis Software, Reliability Testing*, 20 Jan. 2017, www.dfrsolutions.com/blog/how-to-select-the-right-battery-for-your-application-part-1-battery-metric-considerations.

Dakua, B. R., Hossain, M. D., & Ahmed, F. (2015). *Design and Implementation of UART Serial Communication Module Based on FPGA*. Retrieved 4 20, 2020, from https://researchgate.net/profile/biswajit_dakua/publication/296485808_design_and_implementation_of_uart_serial_communication_module_based_on_fpga/links/56d5e68008aebabdb4005193.pdf

Faragher, Ramsey M. and Robert Harle. "Location Fingerprinting With Bluetooth Low Energy Beacons." *IEEE Journal on Selected Areas in Communications* 33.11 (2015): 2418-2428. 19 3 2020. <https://ieeexplore.ieee.org/document/7103024>.

Jeff. “Estimating Solar Charge Time for Batteries: Voltaic Systems.” *Voltaic Systems Blog*, 17 Feb. 2020, blog.voltaicsystems.com/estimating-battery-charge-time-from-solar/.

John. “DC Motor Basic Parts.” *Instrumentation Forum*, 8 June 2018, instrumentationforum.com/t/dc-motor-basic-parts/4583.

LED Lighting – Department of Energy. n.d. 20 3 2020. <http://energy.gov/energysaver/articles/led-lighting>.

“Magnetic Encoder Guide.” *Anaheim Automation - Your Source for Stepper Motor, Brushless DC Motor, DC Motor, and Planetary Gearbox Products*, www.anaheimautomation.com/manuals/forms/magnetic-encoder-guide.php.

McDermott-Wells, P. "What is Bluetooth." *IEEE Potentials* 23.5 (2005): 33-35. 19 3 2020. <https://ieeexplore.ieee.org/document/1368913>.

“MM10 - DC Motor, Miniature, 3 V, 1.23 W, 12000 Rpm, 10 g-Cm.” *Newark*, www.newark.com/multicomp/mm10/motor-miniature-3v-12000rpm/dp/07WX1229?CMP=KNC-BUSA-GEN-SHOPPING-07WX1229.

“Motor Driver.” *Mepits*, 15 June 2015, www.mepits.com/tutorial/379/electrical/motor-driver.

Palma, Jean-Francois de and Jerry Mosesian. *Static surge protection device*. 2012. 18 3 2020. <https://patents.google.com/patent/wo2012166374a1/en>.

Sagahyroon, Assim, Mohammed Eqbal and Farshad Khamisi. *Drawing on the benefits of RFID and bluetooth technologies*. 2010. 19 3 2020. <http://yadda.icm.edu.pl/yadda/element/bwmeta1.element.ieee-000005774850>.

Schmitt, Helmut. United States: Patent 662482. 1970.

Tiwari, B. (2017). *Developing wireless ECG device using Bluetooth protocol for interfacing with Android based applications*. Retrieved 4 19, 2020, from http://ethesis.nitrkl.ac.in/8694

“The Basics of How an Encoder Works.” *Encoder*, 2011, www.encoder.com.

"Understanding and Applying the Hall Effect - Technical Articles." *All About Circuits*, 3 July 2015, www.allaboutcircuits.com/technical-articles/understanding-and-applying-the-hall-effect/.

Woodford, Chris. "AC Induction Motors: How AC Motors Work." *Explain That Stuff*, 6 Jan. 2019, www.explainthatstuff.com/induction-motors.html.

杨杰, et al. *Bluetooth device quick pairing method and Bluetooth device*. 2012. 19 3 2020. <https://patents.google.com/patent/cn103209007a/en>.

赵昱晴. *Integral Bluetooth mobile telephone*. 2010. 19 3 2020. <https://patents.google.com/patent/cn201577121u/en>.

Edamam. (n.d.). Recipe Search API Documentation. Retrieved April 21, 2020, https://developer.edamam.com/edamam-docs-recipe-api

React-Native-Community. (2020, March 19). react-native-community/voice. Retrieved April 21, 2020, https://github.com/react-native-community/voice

Appendix B – Datasheets

"74AC11138 3-LINE TO 8-LINE DECODER/DEMULTIPLEXER." *Texas Instrument*, www.ti.com/lit/ds/symlink/74ac11138.pdf.

"DUAL FULL-BRIDGE DRIVER." *STMicroelectronics*, www.st.com/content/ccc/resource/technical/document/datasheet/82/cc/3f/39/0a/29/4d/f0/CD00000240.pdf/files/CD00000240.pdf/jcr:content/translations/en.CD00000240.pdf.

L9110 2-CHANNEL MOTOR DRIVER . me.web2.ncut.edu.tw/ezfiles/39/1039/img/617/L9110_2_CHANNEL_MOTOR_DRIVER.pdf.

"LM317 3-Terminal Adjustable Regulator." *Texas Instrument*, www.ti.com/lit/ds/slvs044x/slvs044x.pdf.

"MSP430FR58xx, MSP430FR59xx, And MSP430FR6xx Family." *MSP430FR58xx, MSP430FR59xx, And MSP430FR6xx Family*, Texas Instruments, Dec. 2017, www.ti.com/lit/ug/slau367o/slau367o.pdf.

GPIO - Raspberry Pi Documentation. (n.d.). Retrieved 4 20, 2020, from Raspberry Pi Foundation:

https://www.raspberrypi.org/documentation/hardware/raspberrypi/gpio/README.md

Espressif Systems. (2020). *ESP32 Datasheet V3.3.* Retrieved from www.espressif.com: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

“Slotted Optical Switch .” *TT Electronics*, www.ttelectronics.com/TTElectronics/media/ProductFiles/Optoelectronics/Datasheets/OPB804.pdf.

“TMS320F2802x,TMS320F2802xx Piccolo Technical Reference Manual.” *TMS320F2802x,TMS320F2802xx Piccolo Technical Reference Manual*, Texas Instruments, Dec. 2018, www.ti.com/lit/ug/sprui09/sprui09.pdf.

“TPS6212x 15-V, 75-MA Highly Efficient Buck Converter.” *Texas Instrument*, www.ti.com/lit/ds/symlink/tps62122.pdf.

“TPS62147, TPS62148 High Accuracy 3-V to 17-V 2-A Step-Down Converter With DCS-Control™.” *Texas Instrument*.