Final Project Report - Automated Spice Mixer

Embedded Systems II

Fall 2024

Group 5 - Ethan Sprinkle, Peter Nguyen, Evan Guinn

Table of Contents

[1. Abstract 3](#_Toc184045101)

[2. System Design & Architecture 3](#_Toc184045102)

[3. Implementation 5](#_Toc184045103)

[3.1 Hardware 5](#_Toc184045104)

[3.1.1 Components and Peripherals 5](#_Toc184045105)

[3.1.2 Driver Board 6](#_Toc184045106)

[3.2 Software 7](#_Toc184045107)

[3.2.1 UART 7](#_Toc184045108)

[3.2.2 User-Interface and Main Program 7](#_Toc184045109)

[3.2.3 Motor Control 8](#_Toc184045110)

[3.2.4 EEPROM 14](#_Toc184045111)

[4. Results & Testing 17](#_Toc184045112)

[4.1 Integration Findings 17](#_Toc184045113)

[5. Conclusions & Future Work 18](#_Toc184045114)

[6. Appendices 20](#_Toc184045115)

[6.1 Source Code Repository 20](#_Toc184045116)

[6.2 Demo Video 20](#_Toc184045117)

# Abstract

The purpose of this project was to design, code, and build an embedded project using a TM4C (TIVA C) microcontroller and external peripherals. For our project, we produced an automated spice mixer capable of dispensing spice recipes containing up to 8 different spices. This is useful for cooking applications allowing a user to quickly dispense and measure multiple spices for a given recipe. Future capabilities of the project were kept in mind during the design process so the project could be added to and improved upon in future iterations.

The concepts and techniques obtained from this project can also be expanded to apply to a wider range of dispenser applications such as pill dispensers.

# System Design & Architecture

The spice mixer consists of the following major components: eight 3D printed spice containers with individual augers, a stepper-motor driven turnstile base, and a servo-actuated motor controlled dispensing mechanism. The mixer utilizes two stepper motors, a servo actuator, and two hall effect sensors to properly rotate and dispense the various spices based on a command from a user. The basic features of the system include the ability for a user to select and dispense a single spice, add/delete recipes, and save recipes. A user can perform any of these commands using a UART interface (such as Putty) to both send and receive information from the microcontroller. Spice recipes are saved using the EEPROM, allowing recipes and spice information to be retrieved upon power loss to the unit.

A drawing of a mechanical device

Description automatically generated

Figure . Spice Mixer CAD Design



Figure . Spice Mixer

A diagram of a power supply system

Description automatically generated

Figure . System Block Diagram

# Implementation

## Hardware

### Physical Components

#### Auger and Auger Shaft

Physical components of the spice mixer were modeled using SolidWorks. Shown below are the designs of the Auger and Auger Shaft mechanism which is used for dispensing spices. The auger is designed to attach to the inside of the spice holder via a small magnet placed on the end of it. The triangular points are designed to allow connection with the auger motor shaft. The opening is designed to hold approximately a volume of half teaspoon.

A black and silver object

Description automatically generated with medium confidence

Figure . 3D Model of Auger

The auger shaft that is connected to the auger motor is designed to slide along the motor shaft using the servo. The servo controls a pinion gear which will slide the rack that the shaft is attached to.



Figure . Auger Shaft and Servo Connection

#### Rack

The entire spice table sits on a turntable which is turned via gears attached to the turntable and the rack motor.

Close-up of a mechanical device

Description automatically generated

Figure . Rack Motor Gear Mechanism

### Electrical Components and Peripherals

A diagram of a machine

Description automatically generated

Figure . System Electrical Schematic

The figure shown above shows the overall electrical schematic of the system. The motors chosen for controlling the spice mixer are the 42BYGH NEMA 17 stepper motors. These operate at a nominal voltage of 12V and can be driven up to 1.2A max allowing for the spice mixer to be supplied with power using widely available 12 DC power supplies (commonly used by laptops and other household devices). The motors have a resolution of 1.8 degrees per step or 200 steps per revolution. The stepper motor drivers chosen for the spice mixer are the TMC2209 Silent Step-stick drivers. These are known for their silent operation and include a variety of features including a UART interface for configuring the driver. The drivers also come with on-board current sensing which is important for controlling a stepper motor and avoiding damage to the motors. Additionally, the drivers also allow for the micro-stepping of stepper motors allowing for an increased resolution per step from the base resolution. The TMC2209 allows up to a 256 microstep resolution. For the spice mixer, the configured resolution is 16 microsteps. The extended features of the driver via the UART interface were not used for the current implementation. Each driver requires a minimum of 3 pins for control, these being the step (PWM) pin, the enable pin, and the direction pin. As such at least 2 PWM pins and 4 output pins on the TIVA C are required for driving. Any of the capable PWM pins on the TIVA C can be used for driving. The pins chosen for the spice mixer were originally chosen given that each of the pins had PWM capability. During initial integration it was found that the original chosen drivers (which required 4 PWM pins each) were not suitable for the project application and thus these were replaced with the TMC2209. In consideration of time constraints and to avoid a full re-design, the original pins were re-programmed to work with the TMC2209.

The hall sensors are the A3144 Hall-Effect Sensors. Unlike linear hall sensors which have an analog output, these sensors have a digital output which operate based on an open-drain configuration, where the output pin is pulled to ground upon detection of a magnetic field and open when no magnetic field is detected. The output pins of the hall sensor are connected to input pins on the TIVA C. These input pins are configured to use the internal 20kΩ pull-up resistor to pull the sensor output high while there is no magnetic field detected. The servo used is a generic S51 servo which can be operated off a 5V supply and allows for up to 180 degrees of movement. The servo requires a 50Hz PWM signal with a duty cycle varying from 5 to 10% (pulse of 1ms to 2ms) for varying the angle. Due to the slow PWM rate required for the servo, the PWM pins of the TIVA C are not capable of driving at 50Hz without requiring a clock divisor. However, as this would impact the configuration of the stepper motor drivers, it was decided to use the wide-timers (32-bit) and its PWM output capability for controlling the servo. Communication between the TIVA C and the user is performed via UART. Module UART0 was chosen as this is directly connected to the micro-USB/JTAG output of the TIVA C board. The UART is configured to operate at the standard baud-rate of 115200 bits/s. The user may communicate using any serial terminal such as Putty or Real-Term.

### Driver Board

A PCB (shown in figure 5) was made to minimize the wiring required for the spice mixer. The PCB has header pins on the bottom of the board allowing it to interface with the TIVA C as a “hat.” The driver board includes a screw terminal for the 12V motor supply and optional 5V terminal for the hall sensors and servo. If an external 5V supply is used, the jumper pin connecting the 5V rail to the TIVA C VBUS rail must be removed. In addition to this the driver board also contains jumper pins for the TMC2209 driver configuration. The pins can be moved to either apply 3.3V or GND to the MS1 and MS2 pins of the TMC2209. These pins determine the microstep resolution the driver will operate at. As stated in the previous section, the 16 microstep resolution (MS1=MS2=3.3V) is used. Refer to the datasheet of the TMC2209 driver for further configuration. The step motor drivers interface to the driver-board via header pins allowing them to easily be removed in the event it needs to be replaced. The PCB shown in figure 4 was produced in-house at UTA by the EE Makerspace Lab. As it was a prototype board, the board does not contain a silk screen or any labeling to indicate the pin connections and polarity. Connections to this board should be verified against the schematic (figure 4).

A close-up of a circuit board

Description automatically generated

Figure . Driver Board PCB

## Software

The software is broken up into 3 primary modules consisting of the User-Interface module, the Motor Control module, and the EEPROM module. Each of the primary modules are designed specifically for the control and program flow of the Spice Mixer. The primary modules may each contain smaller secondary modules which are designed for the purpose of performing a specific action such as moving a stepper motor, reading/writing to the EEPROM, and parsing for user commands. These secondary modules can be readapted for other purposes.

### UART

User commands are sent over the UART protocol from an input device, such as a computer (in the case of the prototype) or touchscreen. For the prototype, the baud rate used is 115,200, with 8N1 and a 16-entry FIFO. These commands are sent as plain text, which is then parsed into a sequence of ‘fields’ delineated by spaces and stored in a USER\_DATA structure. During parsing, the positions and number of fields are stored alongside the original plain text for use by the command functions in UIControl.

### User-Interface and Main Program

The user-interface/UART is designed to continuously poll for user commands in the main loop of the program. When a command is received from the UART, the user-input is parsed (as described in the UART section) and the first argument is compared and validated against a list of known commands. Any command that is not recognized by the UART interface will result in a notification error to the end user and the display of the “help page” which will list all known commands and their usage/syntax. In addition to being a known command, some commands require additional initial arguments (fields). These required minimum fields will also be validated at the time of processing the user command. An error message will also be displayed for the following conditions:

* Incorrect parameters used when entering a UART command
* The amount of spice remaining in a container is less than the amount required by a recipe
* A Recipe Name does not exist in EEPROM
* A Spice Name does not exist in EEPROM

Figure 9 shows the basic flow of the main loop. The valid system commands are discussed in the sections below.

A diagram of a system

Description automatically generated

Figure . Main Loop Flow Diagram

#### Spice Command

The spice command is the basic command to dispense a single selected spice. The syntax of the spice command is as follows:

* spice <spice name> <quantity>

The user must provide the name of an existing spice followed by the quantity. If the name provided does not match any of the known spices, an error will be indicated, and the command will abort. The quantity will be checked to verify there is enough spice remaining for the requested quantity. This is done by reading the remaining quantity from the EEPROM and then comparing it. If there is an insufficient amount, an error message will be indicated to the user and the system will then request additional user action to determine if the user would like to proceed with the dispense or cancel the action. The user may respond by pressing any key to continue with the dispense or typing in “cancel” to abort the command. On completion of the dispense sequence (discussed in the Motor Control section), the remaining quantity will be updated in the EEPROM.

Note that, on start-up, this command will not be accessible to the user until the home command is performed. (See Home command below). This is to ensure the system has accurately determined the home status before attempting to dispense. Any attempts to use this command before homing is performed will result in a warning message to be indicated to the user. The command will not be performed.

#### Recipe Command

The recipe command is an advanced command to dispense multiple spices. The syntax of the recipe command is as follows:

* recipe <recipe name>

The user must provide the name of an existing recipe. No other arguments are required. If the name provided does not match any of the known spices, an error will be indicated, and the command will abort. If the name exists, the contents of the recipe will be extracted from the EEPROM. Prior to dispensing, each spice element of the recipe will be verified to ensure there is enough remaining. Like the spice command, if there is not enough, an error message will be indicated to the user and the system will prompt the user to either continue or cancel the command. This is repeated for every element in the recipe. The end of a recipe is determined either when the requested quantity is 0 (indicating an empty slot) or the max slots (8) have been dispensed. On completion of the dispense sequence, the remaining quantity of each spice is updated in the EEPROM.

Note, on start-up, this command will not be accessible to the user until the home command is performed. (See Home command below). This is to ensure the system has accurately determined the home status before attempting to dispense. Any attempts to use this command before homing is performed will result in a warning message to be indicated to the user. The command will not be performed.

#### View Command

The view command allows for the user to view a list of existing spices and their quantities or a list of stored recipes. The user can access either by indicating “spices” or “recipes” as the second argument. The syntax of the view command is as follows:

* view <items>

#### Check Command

The check command allows for the user to check the contents of an existing recipe. The user must provide the name of an existing recipe. The command will extract the contents of the recipe from the EEPROM and display each element and their quantities (in teaspoons). The syntax of the check command is as follows:

* check <name>

#### Save Command

The save command allows for the user to save a new recipe or update an existing recipe. If the name of the recipe already exists, the system will notify the user that the recipe already exists and whether they would like to continue or not. If the user chooses to cancel (by typing in cancel) then the command will abort. If the user chooses to continue, the command will begin prompting the user for entries in the recipe. If the name does not exist, the command will validate that the name entered is less than 16 characters. If it is not, the command will be aborted. If the name is within the required length, the system will begin prompting the user for entries in the recipe. The syntax of the initial command is as follows:

* save <recipe name>

For each entry, the user must provide the name of an existing spice along with a quantity less than 1 cup (48 teaspoons). The system will validate that the user enters a valid name and quantity. If either is invalid, an error message will be indicated to the user. Unlike other errors, at this stage, the error message will not abort the command. Instead, it will loop and prompt the user to try again and enter a valid entry. This will continue until either a valid entry is provided or the user cancels. The syntax of entries is as follows:

* <spice name> <quantity>

At any time, the user may type “done” instead of an entry to indicate that this is the end of the recipe. Alternatively, the user can type “cancel” to abort the command. If the user enters a spice name that they had previously already entered, the user will be notified and prompted if they would like to overwrite their previous entry. If the user chooses to continue, then their newest entry will be stored. The system will continue to prompt the user until either “done” is entered or the max number of elements (8 spices) is entered. On completion, the recipe is stored into the EEPROM. If the recipe is an update of an existing recipe, the number of stored recipes (in the EEPROM) will not be incremented. Otherwise, it is incremented. Additionally, the name of the new recipe is copied to the local list/dictionary of stored recipes.

#### Delete Recipe Command

The delete recipe command allows for a user to delete a stored recipe from the EEPROM. The syntax of the delete recipe command is as follows:

* delete <recipe name>

The user must provide the name of an existing recipe. If the name does not exist, an error will be indicated to the user and the command will be aborted. If the name does exist, the system will then ask the user to confirm that they want to delete the recipe. The user must type the keyword “delete” to continue with the removal. If any other word is entered, the system will abort the command. On entering the keyword, the system will remove the recipe from the EEPROM. Due to the direct indexing and accessing of the recipes, along with deleting the recipe, the system must subsequently copy each existing recipe below the deleted recipe and move them up one index if the recipe is in the middle of the block of existing recipes. This method is not ideal as it requires multiple read and writes of the EEPROM thus decreasing the overall usage life of the EEPROM. However, this was the most “efficient method” and as such was used. Future work may improve upon this method.

#### Refill Command

The refill command allows for the user to refill an existing spice with a specified quantity. The syntax of the refill command is as follows:

* refill <spice name> <quantity>

The user must provide the name of an existing spice followed by the quantity. If the name provided does not match any of the known spices, an error will be indicated, and the command will abort. The quantity will be checked to verify if it is less than the max allowed quantity. If the quantity is greater than the max allowed quantity, then a message will be indicated to the user indicating that the quantity they have entered is greater than the max and the system is assuming that they meant to fill the spice to the max quantity. After validation, the command will then subsequently update the remaining quantity of the spice in the EEPROM with the requested quantity.

#### Change Spice Command

The change spice command allows for the user to change or update the name of an existing spice. This is mainly used if the user has filled a spice slot with a different spice from the one currently stored in the system. The syntax of the command is as follows:

* change

On entering the command, the user is prompted for the slot value they would like to update. This slot must be between 0 and 7 (max of 8 slots). If an invalid slot number is provided, an error message will be indicated to the user and the system will prompt the user to try again and enter a valid slot number. This will repeat until either a valid slot number is entered, or the user chooses to cancel. On a valid slot number, the system will repeat the name of the spice that is currently stored in the slot number the user had entered and ask the user to confirm that they would like to continue with changing the name. If the user chooses to continue, the command will then prompt the user to enter the name of the new spice. This name must be less than 16 characters and must be one continuous word. If the entered name is invalid, an error message will be indicated to the user and the system will prompt the user again to enter a valid name. On a valid name entry, the system will write the new name to the corresponding slot in the EEPROM as well as update the local list/dictionary of known spices. The system will then ask if they would also like to update the quantity of the slot they had chosen. The user must type “done” if they wish not to continue. Otherwise, the system will begin prompting the user to enter the quantity of the spice. The user may enter cancel if they change their mind, and the command will then abort. Otherwise, if a valid quantity is entered the remaining quantity of the spice in the EEPROM will be updated with the new requested quantity.

#### Home Command

The home command performs the homing of the rack. The syntax of the home command is as follows:

* home

The command will simply perform a call to the home function in the motor control module to perform the homing. Details of the homing method are discussed in the Motor Control section below. On failure of homing the system will indicate an error message to the user to verify there are no obstructions and to either try again or power reset the system.

#### Reset Command

This command allows for the user to perform a system reset. The syntax of the reset command is as follows:

* reset

The reset command is to be used in the event of an EEPROM failure or some other system type failure. This command will reset all spices to the default first start-up values and will also remove all stored recipes. On entry to the command the user will be asked to confirm that they want to proceed with the system reset. The user must type in the reset keyword “RESET\_SYSTEM\_X342” to proceed with the reset. Any other value entered will result in the command to be aborted. On the entering of the keyword, the system will re-initialize the EEPROM with the default spice data (like the initialization that is performed when the system is powered on for the first time). As this reset is destructive, once the reset is complete, the system will prompt the user to restart the system. The program will then enter an infinite loop, thus forcing the user to power-cycle/restart the system.

#### Stop Command

The stop command allows for the user to remove power from all the primary stepper motors. The syntax of the stop command is as follows:

* stop

On entry of the command, power to both the Rack and Auger motors will be removed. Note, homing must be performed following the usage of this command before spice or recipe commands can be used as once the motors are turned off, the position of the rack will effectively be unknown.

### Motor Control

This section discusses the Motor Control module of the spice mixer. The motor control module includes all functions necessary for controlling the stepper motors and the servo, as well as reading the inputs from the hall-effect sensors.

#### Dispense Sequence

When a recipe or spice is selected to be dispensed, there are 4 major aspects of the dispense sequence. These are: positioning the rack, engaging the servo, commanding the auger dispense, and the disengaging of the servo.

The microcontroller first checks its current position and compares it to the desired position of the selected spice. The program will perform a delta calculation to obtain the shortest distance and determine if it should turn clockwise or counterclockwise (whichever is closer to the desired position). Once the delta has been calculated, the required angle is converted into microsteps to be used by the PWM Interrupt handler for commanding the rack motor. Note, the chosen microstep resolution for the Spice Mixer was 16. The conversion from an angle to microsteps would normally be performed using the following equation:

Where 200 as recalled is the number of full-sized steps for a full rotation of the motor shaft. However, since the rack motor is connected to the rack via a gear, the conversion must also account for the gear ratio. As designed, the gear ratio is 56:15 where the rack motor must make 3.5 turns for a full rotation of the rack motor. The previous equation now becomes:

Once all rotation steps have been completed (and the auger is aligned with the shaft of the motor dispenser), the servo actuates the auger motor shaft into place with the auger of the spice dispenser. The auger will then be commanded to turn the desired number of rotations. Note, a full rotation of the auger motor equates to a ½ teaspoon of the selected spice. The rotation command is like the rack motor except that the auger motor only needs to rotate one direction, so the delta calculation is simply calculating the total number of microsteps per requested rotations. Equation 1 can be used for this conversion. Once rotation is complete, the servo will then disengage the motor shaft from the auger and the process will repeat for any additional spices in a recipe. The figure below shows the overview of the dispense sequence. Note, due to the auger shaft sometimes not being properly aligned with the auger, an auger offset was added in the software. The auger motor will complete the requested number of rotations and then will rotate an additional 20 degrees to offset the auger. The servo will disengage at this point. After disengagement, the auger motor will rotate the shaft back 20 degrees. This placement positions the auger approximately in the middle of the opening of the auger shaft thus increasing the likelihood of a successful connection.

A diagram of a diagram

Description automatically generated

Figure . Dispensing Sequence Flowchart

#### Stepper Motor Control

The stepper motors are controlled by the TMC2209 stepper motor drivers. These require at least 3 inputs which are: STEP, EN, and DIR. The step pin is the PWM input which is used to control the speed and step sequencing of the stepper motors. Speed itself is not controlled by varying the duty cycle. Instead, speed is controlled by increasing or decreasing the PWM frequency since for each pulse detected the driver will increment the step index to move the motor. The duty cycle output from the TIVA C is fixed at a 50% duty cycle. The EN pin is the enable pin and is used to enable/disable the stepper driver output. The DIR pin is the direction pin and is used to control the direction of the motor (CW = 0, CCW = 1).

The commanding and controlling of the PWM outputs are performed via a PWM Interrupt. The beginning of a motor command starts with the calculation of microsteps required by the command. For the instance of the rack motor, it will be provided a slot position (0-7) that it needs to move to. The slot position is then converted to the angle on the rack with respect to the home position. The difference between the requested angle and the current position is taken. The delta is used to determine the direction of rotation (CW or CCW, whichever is the shortest distance). For the instance of the auger motor, it will be provided the number of rotations. The angle/number of rotations is then converted to microsteps of which are provided to the PWM Interrupt Handler for sequencing. Once calculation is complete, the EN pin is cleared to enable the stepper driver output and the PWM Interrupt is enabled to begin the driving of the motor. The main program at this point enters a while loop and will poll the status of the motor while it waits for the step command to be completed by the interrupt handler.

A diagram of a motor and running

Description automatically generated

Figure Stepper Motor Control Flowchart

The PWM Interrupts are configured to interrupt when the generator value reaches 0. This is used since the duty cycle is always fixed at 50%. On an interrupt, the handler will determine if it needs to run based on the step count (previously calculated by the main program). If the step count is greater than 0, this indicates the interrupt should continue to run and sequence the step pulse to the driver. With each interrupt, the step count is decremented until it reaches 0, at which point the interrupt handler will disable further interrupts and disable the PWM output to the stepper driver. In addition to decrementing the step count, the interrupt handler is also responsible for updating the speed of the motor (by setting the LOAD and CMPX registers of the PWM module to vary the frequency). This ensures a consistent PWM output as the interrupt handler will sync the output. The interrupt handlers are mostly the same for both the stepper motors with the exception being that the rack motor has acceleration control. The choice of using interrupts vs actively managing the motor control in the main program was made with the intention of performing other tasks such as active homing or failure handling cases, such as when the user may issue a “stop’ command.

A diagram of a software process

Description automatically generated

Figure . PWM Interrupt Handler Flowchart

To allow for a common architecture and provide the potential for expansion, all necessary data for motor control is stored in a structure type. A motor can be “registered” into the system by adding it to the Motor Data array. Each motor has the following available properties:

* Run Status – (OFF, HALTED, RUNNING, FAILED)
* Direction – (CW, CCW)
* Home Status – (NOTHOME, NEARHOME, HOME)
* Steps - (The number of steps the motor needs to move)
* Position – (Not used in current implementation. Was kept with the intention of using for active position feedback).
* Speed – (speed in rpm)

Not all properties are used by a motor (such as the Home Status and position for the auger motor). This motor data is only accessible by the stepper motor control module. This was done to limit access to important motor control data. “Get” functions were developed to allow external modules to read the status.

Acceleration control of the rack motor was added to improve accuracy and prevent skipping of steps during start and stop maneuvers due to the inertia of the rack. A simplified flowchart of the acceleration control algorithm is shown below.

A diagram of a diagram

Description automatically generated

Figure . Acceleration Control Flowchart

The algorithm begins by determining the previous status of the motor. If the motor status was previously halted, then the acceleration/deceleration steps are calculated. This is simply the amount of command steps divided by 2 for both (i.e. the motor will accelerate for half of the steps and then decelerate for the remaining half). The speed will also be set to the minimum RPM (6 RPM). In the subsequent iterations, the speed of the motor will slowly be increased by a constant factor of 0.03125 RPM (corresponding to 1 RPM increase for every 1.8 degrees). The speed will continue to increase until either the motor has reached the full commanded speed, the total number of acceleration steps has been performed, or the remaining commanded steps is less than 600. The 600 remaining commanded steps condition was added to improve control for smaller moves (such as that performed during the homing offset) as the motor has less time to decelerate in these moves. Once acceleration is completed, the motor will then begin to decelerate as it approaches the midpoint between the final position. Like the acceleration component, deceleration will decrease the speed by a constant factor. This factor is determined based on the final speed of the motor during acceleration divided by the total deceleration steps. This is done to prevent the motor from reaching the minimum speed too early in the deceleration, resulting in the motor moving into position extremely slowly during longer commands.

#### Rack Homing

On startup and/or after a stop command was issued, the Spice Mixer will require the user to initialize the calibrated “home” position before any dispense action can be performed. The homing method utilizes the mid-point between two hall effect sensors to determine the “home” position for additional accuracy. Using a single sensor may incur directional error since the hall sensor may indicate detection even if the magnet is not directly over the sensor, so by using two sensors the directional error is cancelled out since the mid-point between the two sensors will always be the mid-point regardless of the direction the rack was moving. A magnet placed under the turn table is used to trigger the sensors.

A close up of a green device

Description automatically generated

Figure . Hall Sensors and the Homing Magnet

Due to the placement of the sensors, the mid-point of the sensors is not the true home-point. Once the hall sensors have both detected the magnet, the rack motor must be commanded some additional steps to put it in the true home position.

The home state has 3 possible states: NOT\_HOME, NEAR\_HOME, and HOME. When one of the sensors detects the magnet, the home state will transition to NEAR\_HOME indicating to the system to prepare to stop. When both sensors detect the magnet, the home state will then transition to the HOME state indicating to the system to halt the rack motor. If the home state has transitioned to NEAR\_HOME and one or more sensors stop detecting the magnet, the home state will transition back to NOT\_HOME, indicating to the system to speed up and continue rotation.

The checking of the hall sensors is performed via a GPIO interrupt. Since the time at which the home point is detected is critical to obtaining an accurate home position, the usage of interrupts for this is required. The interrupt is configured to trigger on both rising and falling edges. Recall from the Hardware implementation section that the Hall sensors are active low and thus their logic states are inverted. The detection of the falling edge allows us to determine when a sensor has detected the homing magnet. The rising edge allows us to determine the opposite. This is mainly used for the scenario that the hall sensor stopped detecting the magnet and reset the home status to indicate from NEAR\_HOME to back to NOT\_HOME. Note, this interrupt will continue to run even if a homing command is not currently being performed. This has minimal impact on the system resources as the magnet is placed in a specific position and thus it will not constantly interrupt (unless there is a failure). The usage of the interrupt may be useful in future work to perform active homing which may be used to provide positioning feedback to adjust for any error that may have incurred during normal operation.

On the issuance of a home command, the program will command the rack motor to perform 3 full rotations at a speed of 10 RPM. It will then wait approximately 1ms for the motor to begin moving to ensure an accurate motor status is obtained. The function will then continue to periodically poll the home status (that is set by interrupt) and the run status of the rack motor to determine if homing was successful or failed. If the run status of the motor indicates HALTED and the home status indicates NOT\_HOME, then this indicates that the rack performed 3 full rotations before home was found. In the event of a home failure, an error code of 0xDEAF is returned to the main program for processing and failure management. If the home status indicates HOME, the function will then command the rack to rotate an additional number of steps to align with the true home position. Note, the number of steps is largely dependent on the placement of the hall sensors. For the current implementation this offset was found to be approximately 32 degrees counterclockwise.

#### Servo Control

The servo used to actuate the auger motor shaft is controlled using the Wide-Timer 3 PWM Output of the TIVA C. Unlike the Stepper motors which used an interrupt routine for controlling and maintaining the step count, the servo only requires us to set the necessary duty cycle with no further action needed. The servo operates at a rate of 50Hz and requires a pulse lasting between 1ms to 2ms (0 to 180 degrees). The conversion from desired angle to load value to be written to the TxMATCHR register of the wide timer is calculated as follows:

SYSCLOCK is simply the frequency of the system (40Mhz for our case). The 0.025 factor is derived from the 2.5% duty cycle bias for a 0-degree command.

### EEPROM

This section discusses the EEPROM Control module of the spice mixer. The EEPROM module is used to store system data for the spice mixer and most importantly recipes. The EEPROM is a non-volatile type of memory, i.e., it retains its data even when power is removed.

The EEPROM of the TIVA C is distributed into “blocks” where each block contains a specified number of 32-bit words. The TIVA C contains 16 total blocks with each containing eight 32-bit words. This equates to a total of 128 32-bit words. Read and write access of the EEPROM is performed in 32-bit words. The data used by the spice mixer, however, does not need to be 32-bits. To maximize memory storage, the spice mixer data is stored using smaller 16-bit words and bitfields. The memory layout of the EEPROM is shown below.

A screenshot of a computer

Description automatically generated

Figure . EEPROM Memory Layout

#### System Information Blocks

The system information blocks (blocks 0-2) contain data needed for operation of the spice mixer. This includes spice names, spice quantities, and the total number of stored recipes. Each of the 8 spice names are allotted four 32-bit words of memory, allowing for up to 15 characters plus the null character to be stored. The spice data is stored in a single 16-bit word. The spice data word is further split into the position and quantity bits with position being the first 4 bits of the word and the remaining 12 bits being the quantity. As the max slot capacity of the current implementation is 8, it was decided to only use 4 bits. However, this can be increased to 8-bits to support an increase in slot capacity. The quantity bits allow up to a quantity of 4096 to be stored. This is more than sufficient for the needs of the system as the max capacity is only 96 (half teaspoons). The number of stored recipes is also stored as a single 16-bit word. The full 16-bit word is not needed to store the number. However, it was decided to use the full word to simplify the reading of the number. In addition to this data, there is also a First Power-On Key that is stored in the system information block. This key is used to initialize the EEPROM with default spice data/names for the case of when it is the first time powering on the system as the EEPROM will not have anything stored in it yet. After this initial power-on, a keyword of 0xBEEF is written to the EEPROM to indicate for the next start-up that it is already initialized and should use the data currently stored in the EEPROM. The remainder of block 2 is unused. This is not only to maintain separation of system information and the stored recipes but also allows for future expansion if additional slots are added to the mixer.

#### Recipe Information Blocks

Recipe Information is stored in blocks 3 to the end of the EEPROM. As seen from the memory map, each recipe data contains the name (15 characters plus null) of the recipe, and 8 16-bit spice data words (one for each capable slot). This results in each recipe requiring a total of eight 32-bit words. Given this and the max EEPROM size of 128 32-bit words, the system has the capability to store up to 26 recipes. Not all 8 spice data words need to be used in a recipe. Unused words will be written to 0 allowing for the simple detection of the end of a recipe (when the quantity is 0). Additionally, spice data does not necessarily need to be stored in order of slots as the position bits will contain the necessary value to determine which spice to dispense.

#### Reading and Writing to EEPROM

Since the EEPROM is only accessible in 32-bit words, special considerations must be made when reading and writing to the EEPROM.

Reading from the EEPROM is simpler than a write request in that we do not need to worry about data loss. For instance, say we want to read the spice data for slot position 0. From the memory map this is stored at offset 0x0020. To initiate a read, we must write to the EEBLOCK and EEOFFSET registers. The EEBLOCK is provided the desired block number our data is in (block 2 in this instance). The EEOFFSET register is the 32-bit word offset we would like to read from (Word 0 in this instance). The read data will be returned as a 32-bit word. Per the memory map, we know that spice data for position 0 is stored in the lower half of the 32-bit word so we only need to read the data from that half. The upper half of the 32-bit word (which contains spice data for position 1) is simply ignored.

Writing data to the EEPROM requires the additional step of reading the data currently in the EEPROM. Since writing to the EEPROM is performed in 32-bits, we must ensure that the lower or upper half of the word we are not writing to is not overwritten with zeroes. By first reading the full 32-bit word and then making a local copy, we can then write data to the desired upper or lower half and then write the full 32-bit word back, thus maintaining the data. Additionally, all functions that write to the EEPROM will also return an error code from the EEPROM if the EEPROM failed to write. This error code may be used for notifying the user of a system issue and have them attempt to restart the Spice Mixer to attempt to reset the EEPROM.

# Results & Testing

The physical components of the project were first modeled in Solidworks3D to ensure proper design dimensions and functionality. Parts were then 3D printed using PLA material and then tested for form, fit, and function with the rest of the assembly. Parts that failed were adjusted in the CAD software and reprinted. A wooden frame was used to elevate and hold the Spice Mixer assembly.

Software development was shared across the team using a git repository. Branches were created, tested, and merged by individual group members for the development of three main functionalities: EEPROM, UART Interface, and Motor Control. The Motor Control function was used to test and calibrate the physical components to ensure they would work properly with the software. Prior to validating the motor control with the hardware, the debugger was used to validate proper commands and outputs and ensure no hazardous faults (such as a short to ground) were being done. This was performed by verifying the state of the output registers. Additionally, appropriate timing/frequency were verified using an oscilloscope.

Once the software was validated, the motor control functions were integrated with the hardware. Electrical components and wiring were verified using a breadboard. The prototype PCB was developed afterwards once all connections were verified. The EEPROM storage was validated by first developing and testing the write functions by writing test data to the EEPROM. The read functions were then developed and tested by reading the data from the EEPROM. Proper functionality is confirmed when the value that was written to the EEPROM is repeated back by the read functions. The UART interface commands were tested individually to ensure validation of commands and ensure consistent user operation.

## Integration Findings

Integrated testing revealed some design flaws in the system. Some of these issues were corrected while others are documented here with the intention of improving it in future work.

During initial integration it was found that the stepper motor drivers initially chosen were not sufficient for the application. The drivers lacked current sensing/limiting which is critical for stepper motor control and preventing damage to both the motor and the system. This issue was corrected by choosing proper drivers (the TMC2209). Due to the difference in requirements for driving the motor control code was completely redesigned to work with the TMC2209.

Initially, the rack motor had no acceleration control. Instead, it was immediately commanded to the requested speed (like the auger motor). This was found to cause the rack motor to skip steps during start and stops resulting in inaccurate positioning during both homing and regular operation. This issue was corrected by implementing the acceleration control (described in the Motor Control section).

Variances in the 3D components led to inaccuracies in positioning. For example, it was noted that at certain positions in the rack rotation, the rack would “rub” against the outer wall. This indicated that the rack was not entirely circular. Due to time constraints, this issue was not corrected. It was also noted that the stepper motor provided enough torque to overcome this resistance and as such no further action was taken.

The original design of the auger shaft and auger required that the positioning of the shaft was perfectly aligned. This mostly resulted in the auger shaft failing to engage the auger. This issue was mitigated through both hardware and software. The hardware mitigation was to increase the area of the auger shaft. The size of the auger shaft was slightly increased thus in turn increasing the opening for the auger to connect to. The software mitigation would offset the auger screw, and then rotate the auger shaft back in the opposite direction by the same distance to position the opening of the shaft approximately in the middle of the auger screw, thus increasing the chances of a successful connection.

The flat bottom of the spice holders results in some of the contents not being fully dispensed as there is nothing to push it into the auger opening. As re-designing this would require significant redesign of the current structure, it was decided to defer this issue for future work as the purpose of this project was to develop a prototype.

# Conclusions & Future Work

The main goal of this project was to design and build a functional prototype spice mixer, where additional capabilities could be added in the future to improve upon the original design. To reduce printing material costs and simplify testing, the current iteration of the mixer is limited to 8 different spices. This amount could easily be doubled to 16 containers by simply reducing the size of the current containers and updating the source code, which was designed to be easily configurable. Furthermore, the current augers were designed to dispense in ½ teaspoon amounts, but this value could be lowered by either reprinting the augers with smaller buckets, or by adding blocks to the augers to reduce their overall capacity. Additionally, with the proven functionality, focus can also be placed on improving the design of the spice holder to ensure all spices are dispensed. This may include designing so that the inside sharply slopes inward towards the auger opening thus allowing for the contents of the spice holder to naturally fall into the opening via gravity.

Currently, there is not a precise method to detect the amount of spice remaining in a container beyond a rudimentary decrement method for each auger rotation. By using this method, the system assumes that a full amount is dispensed with each auger rotation and cannot account for scenarios where the full amount is not dispensed, such as with clumping of powdered spices. Conversely, if another peripheral such as a weight scale were added, there could be a check between the expected and actual weights and the mixer could account for the calculated difference.

The UART interface provides a cheap and quick method to communicate with the spice mixer; however, it is not very user friendly. An ideal solution would be to replace the UART interface with an LCD touchscreen equipped with a GUI. This would allow easier parsing of data such as remaining spice levels, current recipes, and recipe modification. A more advanced interface could also include voice activation capability, allowing the user to speak commands to the mixer.

As previously discussed, the drivers used for the project contain advanced capabilities which can be accessed via the UART. These capabilities include stall detection and step indexing for positioning feedback. As these drivers were selected late in the project, the full capabilities of the drivers were not utilized. These features may be beneficial in providing enhanced positioning accuracy for the rack as well as for safety/failure management.

Overall, the project was designed specifically for dispensing spices. However, the concepts as well as the code baseline can easily be re-adapted for any number of dispenser type applications.

# Appendices

## Source Code Repository

The following is the link to the GitHub repository containing the work described in this report. This includes all source code, documentation, and 3D CAD and CAM files.

<https://github.com/Pvhn/CSE5342_SpiceDispenser_Proj.git>

## Schematics and PCB Files

Development of the electrical schematic and PCB were done in Fusion 360. The following link provides copies of these files. NOTE: Fusion 360 is required for proper viewing and editing. In addition to these links, these files can also be found in the repository

Fusion 360 Project:

<https://a360.co/3CVfpxP>

Electrical Schematic:

<https://a360.co/3AWSpOs>

PCB Design:

<https://a360.co/3ViIjOJ>

## Demo Video

Attached is a video demonstrating the different commands. This video can also be found in the Source Code Repository (See Appendix 6.1).

Commands in the video are shown in the following order:

1. home
2. view spices
3. spice xxx
4. view recipes
5. check CAJUN
6. recipe CAJUN
7. view spices
8. save CHICKEN
   1. Saving a new recipe. For demo purposes the recipe is a random selection of spices with quantities 1,2,3, and 4.
9. view recipe
10. check recipe
11. recipe CHICKEN
12. delete CHICKEN
13. view recipes
14. change
    1. Shown in the video we are swapping out slot 2 (previously GARLIC) for our new spice “PICKLE”
15. refill SAGE 20

Link to demo video: [EE5342\_SpiceMix\_Demo.mp4](https://mavsuta-my.sharepoint.com/:v:/g/personal/peter_nguyen3_mavs_uta_edu/EVefTSrf9KpFvghp_Y4DllgBElhwOAp0LO3sR-mjClhxzw?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=4Yvl9b)