Phone Microwave

By
Khanh Hoang
Emily Zhou

Project Advisor
Dr. Bridget Benson

Senior Project
Computer Engineering Department
California Polytechnic State University
San Luis Obispo
March 2023
# Table of Contents

List of Figures .......................................................... 3
List of Tables .......................................................... 3
Abstract ........................................................................ 4
Acknowledgments .......................................................... 5
Chapter 1: Introduction ..................................................... 6
Chapter 2: Requirements, and Specifications ......................... 7
Chapter 3: Functional Decomposition ................................... 9
Chapter 4: Project Planning ............................................... 10
Chapter 5: Project Design ................................................ 12
   Hardware Design ....................................................... 12
      Microcontroller ...................................................... 12
      Cellular Communications Module ............................. 12
      Microwave Keypad .................................................. 12
      Microwave Microcontroller .................................... 13
   Software Design ....................................................... 14
Chapter 6: Testing ........................................................ 18
Chapter 7: Conclusion and Future Work ............................... 20
References ...................................................................... 21
Appendices ..................................................................... 22
   Appendix A: Senior Project Analysis .............................. 22
   Appendix B: Datasheets and Resources ......................... 26
   Appendix C: Images of Phone Microwave ...................... 27
   Appendix D: Schematic of PCB for Microwave Control .... 29
   Appendix E: Code ....................................................... 30
      ArduinoCode.ino ..................................................... 30
      Keypad.h ................................................................ 37
      Keypad.cpp .......................................................... 39
      MicrowaveControl.h .............................................. 42
      MicrowaveControl.cpp ......................................... 43
      SimCom.h ................................................................ 45
      SimCom.cpp ........................................................ 49
      PresetFoods.h ....................................................... 53
      PresetFoods.cpp .................................................... 53
List of Figures

Fig. 1. Level 0 Functional Decomposition 9
Fig. 2. Level 1 Functional Decomposition 9
Fig. 3. Fall 2022 Gantt Chart 10
Fig. 4. Winter 2023 Gantt Chart 10
Fig. 5. Arduino to Microwave Interface Connected via CD4051B CMOS 13
Fig. 6. Main Software Architecture of the Phone Microwave 15
Fig. 7. Function Diagram for Processing Calls & DTMF Key Press Input 16
Fig. 8. The handleSMS Function Diagram for Controlling Additional Microwave Features 17

List of Tables

Table I. Engineering Specifications 8
Table II. Bill of Materials 11
Table III. Testing Requirements 18
Abstract

This project involves the installation of remote-control capabilities in an antique 1980s microwave, effectively turning the microwave into a “smart” device. While preserving the original functionality of the microwave, a combination of software and hardware components allows for remote microwave operations. The microwave can be remotely operated by calling the built-in number, and more advanced settings and options can be utilized by texting. The microwave is also secured against unauthorized use with the addition of a PIN code that is required to operate the device.
Acknowledgments

We would like to first thank Dr. Bridget Benson for advising us throughout this project and supporting its development. We are also thankful to the creators of Steins;Gate, a work of fiction, for providing the inspiration to create this project. We would also like to thank the Cal Poly Computer Engineering Department for funding this project. Finally, we would like to thank the numerous professors at Cal Poly for their hard work in teaching us and providing us with the knowledge and skills to make the creation of this device possible.
Chapter 1: Introduction

When ‘Master Chef’ winner Christine Ha was first diagnosed with Neuromyelitis Optica, an autoimmune disease known to severely impair vision and movement, she was distraught [1]. She encountered many difficulties performing what others deemed as basic tasks. Making a PB&J sandwich, for example, was one incident that was met with a flurry of cuts and messes in the kitchen. Despite her initial struggles in cooking, Ha, much like other visually-impaired individuals, employed a combination of sensory cues and technology to better navigate the kitchen.

Commonplace in visually-impaired households are gadgets like talking scales, talking thermometers, and microwaves with audible capabilities or braille labeling. Individuals furthermore frequent phone applications to purchase groceries online, reducing the hazards associated with in-person shopping. Despite these accommodations, a study of blind and visually-impaired Canadians revealed that participants prefer to prepare simple meals using microwaves [2]. The reduced involvement and preparation time in microwaving meant minimizing the risk of cuts and burns. Thereby, due to its safety and convenience, microwave cooking became the third most popular heating method in western kitchens [3].

While visually-impaired individuals can have microwaves equipped with audio functionality and braille-labeled texts, there are also microwaves marketed toward the general audience with similar capabilities. For example, the Smart Microwave by Whirlpool connects to the smartphone using wi-fi. The user then remotely operates the microwave by setting the cook time, adjusting the power level, and more [4]. Additionally, the microwave functions with voice commands, with some brands using Alexa as a method of controlling smart devices. Through the usage of phones and voice communications, modern microwaves are equipped with accessible technology for the visually impaired.

Our project aims to achieve remote-control functionality with a microwave through texting and phone calls. Unlike the above-mentioned smart microwaves, our microwave can operate without an internet connection. A text interface allows the user to configure power and defrost settings, while phone calls allow users to set the time and start cooking. As phones become an increasingly prevalent appliance in society, most individuals would be familiar with phone-calling and texting. Provided that older generations are less proficient with wi-fi communications and internet usage, our goal is to make the remote-control microwave an accessible experience for a wide range of people.
Chapter 2: Requirements, and Specifications

The primary purpose of the remote-control phone microwave is to provide additional convenience and accessibility features that a standard microwave would not normally provide. For that reason, the feature requirements and corresponding engineering requirements are described below in Table I.

The microwave’s physical keypad and other pre-existing functions should remain fully operational to ensure ease of use. It is moreover important for the microwave to have standard functions that are familiar and universal to many other microwaves, while also providing simple and clear instructions that are accessible to a diverse range of people. Therefore, the microwave must be easy to use, fully functional, and accessible.

Safety is a top priority for all kitchen appliances, including the microwave. Exposed wires or circuitry should be avoided to prevent any electrical hazards. The microwave cover must therefore continue to hide any additional electronic components included in the final design.

Finally, the microwave should remain affordable for it to be accessible to a wider range of people. Especially for lower-income individuals with disabilities, it is necessary that these users are able to afford and purchase accessible technology.
<table>
<thead>
<tr>
<th>#</th>
<th>Engineering Specification</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Microwave is capable of remote control via DTMF touch-tone interface.</td>
<td>Users are able to remotely control the microwave with a familiar interface that doesn’t require an extra application to set up.</td>
</tr>
<tr>
<td>2</td>
<td>Microwave cook time can be set via remote control using phone call.</td>
<td>Basic functionality of the microwave should have remote control capabilities.</td>
</tr>
<tr>
<td>3</td>
<td>Microwave can be started and stopped via remote control through a phone call.</td>
<td>Basic functionality of the microwave should have remote control capabilities.</td>
</tr>
<tr>
<td>4</td>
<td>Microwave remains operational using physical buttons.</td>
<td>Users should still be able to easily operate the microwave without relying on its remote interface.</td>
</tr>
<tr>
<td>5</td>
<td>Operating in frequency range bands of 700MHz, 1900MHz, or 1700MHz.</td>
<td>These bands are the supported frequency bands used for LTE communications in the AT&amp;T network.</td>
</tr>
<tr>
<td>6</td>
<td>Product has no exposed electronic components.</td>
<td>Since a microwave will be in close proximity to food and beverages, exposed electronics can put the device at risk of damage or harm to the user.</td>
</tr>
<tr>
<td>7</td>
<td>Cost of the microwave is less than $100.</td>
<td>Microwave should match market price values to keep up with competition.</td>
</tr>
<tr>
<td>8</td>
<td>Additional circuitry can fit within two areas of 8in x 3.75in.</td>
<td>Additional circuitry needs to be small enough to fit within the confines of the microwave.</td>
</tr>
<tr>
<td>9</td>
<td>~5V voltage for Arduino and LTE module.</td>
<td>Voltage required by the additional modules used in Phone Microwave system.</td>
</tr>
<tr>
<td>10</td>
<td>Power consumption of the additional circuitry does not exceed 5W.</td>
<td>Limits on the power consumption of additional circuitry lowers the energy cost of using the system and improves sustainability.</td>
</tr>
<tr>
<td>11</td>
<td>Microwave’s additional features such as reheat, defrost, and power levels can be controlled remotely.</td>
<td>Additional features added to the phone microwave system.</td>
</tr>
<tr>
<td>12</td>
<td>Prestored recipes for certain microwavable foods can be activated.</td>
<td>Convenient feature to avoid having to remember recipe times for different foods.</td>
</tr>
<tr>
<td>13</td>
<td>A PIN code is required to operate the microwave remotely.</td>
<td>Security feature to prevent unauthorized use of the microwave.</td>
</tr>
</tbody>
</table>
Chapter 3: Functional Decomposition

Figure 1 demonstrates a Level 0 Functional Decomposition of the remote-control phone microwave system. The inputs for running the microwaves are through physical microwave keypad presses, SMS messages and through DTMF (dual tone multi-frequency) presses on a call.

![Level 0 Functional Decomposition Diagram](image1)

**Fig. 1. Level 0 Functional Decomposition**

The Level 1 Functional Decomposition shown in Figure 2 describes the specific components handling the inputs and processing logic. Most notably, inputs to the microwave originating from the cell phone are processed through the SIM7600A. This module handles the wireless communication by acting as a secondary phone. It receives and transfers information from the phone caller or texter to the Arduino MEGA microcontroller. The Arduino converts the instructions sent by phone or keypad to a set of CD4051B CMOS chips. This circuit then translates the information and simulates them into physical key presses, allowing the microwave to finally read and act upon the inputs.

![Level 1 Functional Decomposition Diagram](image2)

**Fig. 2. Level 1 Functional Decomposition**
Chapter 4: Project Planning

Figures 3 and 4 showcases the activities and objectives planned for each week.

The first quarter consists of brainstorming, extracting information from schematics and datasheets, and designing the skeleton of the final project. These activities include testing compatibility between modules, determining the electrical components that needed modifications, and identifying potential safety risks. During this time period, the prototypes for each subsystem that would eventually be integrated as part of the Phone Microwave were also iteratively built.

![Fall 2022 Gantt Chart](image)

Fig. 3. Fall 2022 Gantt Chart

The time used during the second quarter included testing, completing the Phone Microwave's functions, as well as completing the final report. Basic remote control functionality of the Phone Microwave was completed during this time. The prototypes developed during the first quarter were polished up and integrated into the final system. Afterwards, some additional features were added, such as the inclusion of preset recipes through SMS, and a security PIN code.

![Winter 2023 Gantt Chart](image)

Fig. 4. Winter 2023 Gantt Chart
In addition to a proposed schedule detailed for two quarters, Table II describes the materials, and their corresponding costs, required to fulfill the project objectives.

**TABLE II. BILL OF MATERIALS**

<table>
<thead>
<tr>
<th>Item</th>
<th>Unit Cost</th>
<th>Count</th>
<th>Total Cost</th>
<th>Description</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microwave (Montgomery Ward Model KSA-8068T)</td>
<td>$10</td>
<td>1</td>
<td>$10</td>
<td>A working microwave</td>
<td>Facebook Marketplace</td>
</tr>
<tr>
<td>Prepaid AT&amp;T Phone Plan</td>
<td>$46.82</td>
<td>1</td>
<td>$46.82</td>
<td>Initial payment for prepaid phone plan.</td>
<td>AT&amp;T</td>
</tr>
<tr>
<td>CD4051B CMOS IC</td>
<td>$0.799</td>
<td>10</td>
<td>$7.99</td>
<td>8-channel multiplexor ICs.</td>
<td>Digikey</td>
</tr>
<tr>
<td>SIM7600A-H LTE Module</td>
<td>$78.39</td>
<td>1</td>
<td>$78.39</td>
<td>LTE module</td>
<td>Waveshare</td>
</tr>
<tr>
<td>Arduino MEGA</td>
<td>$20.99</td>
<td>1</td>
<td>$20.99</td>
<td>Microcontroller</td>
<td>Amazon</td>
</tr>
<tr>
<td>PCB</td>
<td>$6.933</td>
<td>3</td>
<td>$20.80</td>
<td>PCBs of the additional circuitry.</td>
<td>OSH Park</td>
</tr>
<tr>
<td>Power Strip</td>
<td>$6.57</td>
<td>1</td>
<td>$6.57</td>
<td>For powering the microwave, LTE module, and Arduino all at once.</td>
<td>Amazon</td>
</tr>
<tr>
<td>Flux Paste</td>
<td>$12.45</td>
<td>1</td>
<td>$12.45</td>
<td>Used for soldering.</td>
<td>Amazon</td>
</tr>
</tbody>
</table>
Chapter 5: Project Design

The design of the phone microwave requires a synthesis of both hardware and software working together to operate properly. The description for each hardware component and software architecture for the system is described below.

Hardware Design

Microcontroller
At the core of the phone microwave is a microcontroller controlling the microwave and communicating with the peripherals needed for cellular communication. The microcontroller selected for this project is the Arduino Mega. The Mega has enough pins to support the connections required for connecting to both the microwave’s internal microcontroller and the microwave’s keypad, supports the serial UART protocol needed to communicate with the cellular module, and is also capable of concurrent serial communications with a computer for debugging purposes.

Cellular Communications Module
The remote interface for the phone microwave is through the cellular capabilities of a phone, through phone calls or texting. This would require a way for the system to access a cellular network and receive phone calls and SMS, hence a cellular communications module is required. The module selected for this purpose is SIMCOM’s SIM7600A-H. This specific model had to be selected due to its support for LTE communications, which is necessary as the major carriers in the United States no longer have support for 2G or 3G cellular service. This specific model also supports the frequency bandwidths that most carriers in the United States support, making it a clear choice for this project.

The SIM7600A-H supports communication with an external microcontroller via a serial UART line. It is capable of 3.3V and 5V logic levels, for communication with the Arduino Mega, the logic level will be at 5V.

Microwave Keypad
The microwave's keypad is a membrane keypad with 22 buttons and originally connects to the microwave’s internal microcontroller with a Flexible Printed Circuit (FPC) cable. The keypad registers a key press by shorting a corresponding row and column pin matching the appropriate button press. A microcontroller could detect the key pressed by outputting a signal on each of the row pins and scanning each column pin to see if they match that voltage. The microwave schematic in Appendix B shows the matching row and pin connections for each button. To connect this keypad to the Arduino, the FPC connector was desoldered from the
microwave’s PCB, and the pins attached to this connector were then wired to the Arduino. This would allow the Arduino to read in the key press of the microwave’s keypad.

Fig. 5. Arduino to Microwave Interface Connected via CD4051B CMOS

Microwave Microcontroller

To operate the microwave’s features using the Arduino, key presses of the microwave’s keypad will need to be simulated. The microwave uses the process described above in the “Microwave Keypad” to read in a button press. This means to simulate a button press, the corresponding row and button pins connected to the microwave’s microcontroller must be shorted. To best simulate this, multiple 8-channel MUXes were used to allow a connection between row pins and column pins. The channel selection and enabling of the MUX would be handled by the Arduino Mega. This circuit would allow the microwave microcontroller’s keypad scanning
process to remain as close as possible to the original microwave, and isolate the Arduino circuitry from the microwave’s circuitry as much as possible. The diagram detailing this circuit is shown in Figure 5. The printed circuit board used for the final iteration of the Phone Microwave is shown in Appendix D.

Software Design

The software architecture of this system running on the Arduino Mega is primarily focusing on two things: checking for data from the SIM7600, and checking if there has been a microwave keypad press. The main software flow will first initialize all the pins and set up the serial connection to the SIM7600. To communicate with the SIM module, AT commands are used and sent through the serial line. The list of AT commands supported by the SIM7600 is listed in a document written by the manufacturer of the SIMCom module [6]. The Arduino will continuously loop checking if there’s any data from the SIM module; if there has been a call or a text, it will call the corresponding functions. Otherwise, the software will check the physical keypad for any key presses and relay them to the microwave if there have been any. To avoid conflicts between inputs, the system will disable the physical keypad if there is a call connected to the phone system. Figure 6 shows the software flow of the system.
Fig. 6. Main Software Architecture of the Phone Microwave

The handle call function shown in Figure 7 will be called whenever the system is on a current call. This function will process data coming from the SIM7600 and check if there have been any key presses on DTMF interface or if the call has ended. If the call has ended, the function will set the state of onCall to false and reset the state of the PIN lock. During the processing of the DTMF key presses, if the state of the system is currently locked, the key presses will be compared against the four-digit PIN code; a mismatch will end the call, while a matching code will unlock the use of remote-controlled microwave functionality. Successive key presses in this unlocked state will control the microwave’s number keys, cancel/stop and start buttons.
Texting inputs were added as an additional feature of the Phone Microwave to allow for the use of preset recipes, defrosting, and reheating options. The SMS input splits the message into tokens separated by spaces. The first option will select what feature of the microwave will be used: DEFROST, REHEAT, POWER\(^1\), or PRESET. The handleSMS function will then check the second argument, if none is provided or it is incorrect, then a list of correct arguments will be sent as a response. Typing the correct feature followed by a valid argument will activate that feature. For the presets feature, there is a saved list of a few food recipes, and each food item has a list of microwave keypresses saved. When activated, the microwave will simulate those

\(^1\) The use of POWER requires the microwave be in operation, or has a cooking time imputed.
key presses in the order given in the saved recipe. If an invalid first argument is provided, the system will reply with a valid list of arguments. A user could also START or CANCEL operations on the microwave by entering those respective phrases into SMS. Finally, a user could change the PIN code through SMS, but security for SMS controls has not been added yet, so currently, any user could modify the Phone Microwave PIN or activate the microwave through SMS texts.

Fig. 8. The handleSMS Function Diagram for Controlling Additional Microwave Features.
Chapter 6: Testing

The Phone Microwave system was tested repeatedly throughout the entirety of its design and manufacturing process. During the initial stages of the Phone Microwave’s manufacturing process, each component of the microwave was tested independently and repeatedly to ensure that it functioned correctly. The microwave keypad, cellular component, and microwave control systems were all tested separately with many prototype iterations, software, and hardware. The microwave control systems initially used a breadboard system before it was migrated to a PCB board, which ensured microwave control functionality before being finalized. While most test cases did pass, there were a few cases with undesired results. Table III below shows the tests corresponding to each requirement and whether they passed or not, as well as some additional notes for each test.

The Phone Microwave’s remote control functionality is fully functional, there is a noticeable delay that occurs when using the remote control functions. There is approximately a 2-4 seconds delay between pressing a button on the phone dial pad or SMS being sent and the input being registered on the microwave. The inputs are buffered, so this doesn’t mean every key press will take seconds to complete. This does mean that attempting to stop microwave cooking might take a few seconds to occur, which might be problematic if there is an emergency. This delay is likely due to the latency of communicating data from a cell phone through a cellular network before finally reaching the microwave’s SIM module.

<table>
<thead>
<tr>
<th>#</th>
<th>Requirement</th>
<th>Met?</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Microwave is capable of remote control via DTMF touch-tone interface.</td>
<td>✅</td>
<td>Fully working. There is a noticeable 2-4s delay between remote control input and microwave input being registered.</td>
</tr>
<tr>
<td>2</td>
<td>Microwave cook time can be set via remote control using a phone call.</td>
<td>✅</td>
<td>Fully working. Phone call connects to system and phone dial presses corresponds to respective numbers on microwave keypad.</td>
</tr>
<tr>
<td>3</td>
<td>Microwave can be started and stopped via remote control through a phone call.</td>
<td>✅</td>
<td>Fully working, ‘*’ corresponds to stop/cancel, and ‘#’ corresponds with start. Microwave was successfully stopped, started, and paused repeatedly.</td>
</tr>
<tr>
<td>4</td>
<td>Microwave remains operational using physical buttons.</td>
<td>⚠</td>
<td>Inconsistent. The physical connection of the FPC cable is loose, and Arduino often loses connection with the keypad. When the connection works, every button is recognized by Arduino and forwarded to the microwave correctly.</td>
</tr>
<tr>
<td></td>
<td>Operating in frequency range bands of 700MHz, 1900MHz, or 1700MHz.</td>
<td>AT&amp;T successfully registered our device onto their network. The device is capable of receiving calls and SMS. Tested by calling and texting from different numbers across different carriers.</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Product has no exposed electronic components.</td>
<td>The cover of the microwave was successfully placed back on, and the additional circuitry is not exposed.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Cost of the microwave is less than $100.</td>
<td>The final cost of the microwave exceeds the estimated cost, ending up at $204 including the phone plan, and $157 without.</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Additional circuitry can fit within two areas of 8in x 3.75in.</td>
<td>The dimensions of the SIM module are slightly less than 2.75in x 2.6in. The Arduino and the PCB fit on a board of dimensions 6.5in x 3in, which means it could easily fit within the areas on top of the microwave.</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>~5V voltage for Arduino and LTE module.</td>
<td>The Arduino and SIM module is powered by a USB cable connected to a wall adapter, which supplies 5V.</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Power consumption of the additional circuitry does not exceed 5W.</td>
<td>The USB 2.0 standard specifies a maximum of 0.5A current drawn. The two devices total to 5V@1A, which is a maximum of 5W. This power limit is not likely to be reached.</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Microwave’s additional features such as reheat, defrost, and power levels can be controlled remotely.</td>
<td>Reheat, defrost, and power can be accessed through SMS controls. This has been tested repeatedly and does activate their corresponding microwave functions.</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Prestored recipes for certain microwavable foods can be activated.</td>
<td>SMS controls can activate a predefined set of button presses for the microwave. Currently popcorn, rice and potato presets have been added. When activated, the microwave repeats a set of predetermined keypresses matching the recipe.</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>A PIN code is required to operate the microwave remotely.</td>
<td>Phone calls require a PIN code to operate the microwave, however, SMS does not require a PIN code. The PIN code can be modified from SMS controls.</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 7: Conclusion and Future Work

The Phone Microwave was able to accomplish its purpose by being a remote-controlled smart system. The remote capabilities of the system through the use of phone calling and SMS texting allow the system to be controlled from effectively any phone, without requiring a separate application or website. This system provides a universal control system that almost everyone can access. However, because this system does rely on a cellular network for its communication, the costs for this system are very high. The equipment for communicating with an LTE network has a high upfront cost, and a cellular plan would also cost extra as well. An alternative version of this system that uses a LAN network and a microcomputer such as a Raspberry Pi running a backend server might cost less but also require higher computational resources for running the backend.

One significant issue with this system is that the physical keypads aren’t always functional because of the loose connection between the keypad’s FPC connector and the jumper cables. For future work, the FPC cable could be soldered directly to wires, but efforts will need to be made to ensure that these wires are secured and won’t disconnect due to stress on the cables as a result of their positioning. An additional improvement would be to the security of the system. While the calling functionality is locked with a PIN code, as it stands, anyone with the phone number of the microwave system could send a text for changing the PIN code or control the microwave remotely, as SMS is unlocked. Securing SMS would require the state of the lock to be kept, the phone number for that messaging session, and perhaps time tracking to relock the controls after a certain amount of time. This would require quite some more processing that might affect the performance of the Phone Microwave system since this system is only running on an Arduino Mega.

As an educational endeavor, the design and development of the Phone Microwave system exemplified many of the skills and knowledge gained through Cal Poly’s “Learn By Doing” philosophy. This project modified an existing commercial microwave and required knowledge from embedded systems courses, electronics manufacturing, and digital computer design classes to create a design and eventually a final product. This project demonstrates that it is feasible to convert a general appliance at home into a modern “smart” product that can be controlled remotely. While improvements to the system could be made, we were overall very satisfied with how the Phone Microwave turned out.
References


Appendices

Appendix A: Senior Project Analysis

https://cpe.calpoly.edu/courses/senior-project/senior-project-report-completion-and-filing/senior-project-analysis-submittal/

Summary of Functional Requirements

Our project is an alteration of an existing microwave to add additional features to it. The altered microwave can be remote-controlled through phone calling and text messaging. A phone call could be made to the Phone Microwave’s phone number, and dialing the numbers on the keypad would allow setting the cooking time and starting and stopping the microwave. Texting the number allows additional microwave features, like defrosting and reheating. The microwave will also have additional features such as a simple database of foods memorized. These additional features should not affect the original functionality of the microwave, such as preventing the use of the physical buttons on the microwave.

Primary Constraints

One major challenge we faced with this project was the lack of official documentation regarding the microwave. Microwaves tend to not have official documentation posted about their microcontroller, or the voltages they use when scanning for keypad inputs. Another challenge we faced was obtaining cellular service for our system. The major carriers in the U.S. recently closed down their 2G and 3G services, which limited our options for the cellular module to only those that can support 4G LTE. AT&T is also selective about the devices they support on their cellular network, so we had to call in support to get the LTE module’s IMEI approved for use on their network. We also faced another challenge due to the nonstandard connection between the microwave’s keypad and the microwave’s microcontroller. We had to desolder this connector, but its connections were loose with jumper cables, and the flexible cable often struggled to stay in place within the connector. This made our goal of maintaining the original keypad functionality very difficult to achieve.

Economic

<table>
<thead>
<tr>
<th>Item</th>
<th>Unit Cost</th>
<th>Count</th>
<th>Total Cost</th>
<th>Description</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microwave (Montgomery Ward Model KSA-8068T)</td>
<td>$10</td>
<td>1</td>
<td>$10</td>
<td>A working microwave</td>
<td>Facebook Marketplace</td>
</tr>
</tbody>
</table>
The original estimated cost of components prior to the project was expected to be around $100. However, the requirement for an LTE module instead of a GSM module has dramatically increased the price, and the cellular phone price was higher than expected. This resulted in a final cost of $204. The additional equipment required is a Phillips screwdriver (~$9), a multimeter ($15.50), a soldering set ($20), and a drill with large drill bits (~$45). This totals to $89.50 in equipment costs, but we already had this equipment prior to the project. The total amount of time spent on the project is difficult to estimate, since a significant amount of time was spent designing or researching information, and the time spent on development was non-structured. The estimated time spent on this project is likely around 140 hours.

**If Manufactured on a Commercial Basis:**
Due to the niche nature of this product and the lack of brand recognition, this product is unlikely to achieve a high number of sales. A generous estimate would be around 50 products sold a year. Building a microwave from scratch would likely increase manufacturing costs by a significant amount (requiring upfront costs, specific microwave components like a magnetron), since the resale of a modified microwave will likely violate some regulations. It is unlikely that the production of this product would be profitable, without the existing infrastructure of a microwave company.
Environmental
Soldering was heavily involved during the manufacturing of this system. If the solder was lead-based, this might produce hazardous fumes for the environment. A typical microwave is also rated at 1000W when heating up food. This does consume quite a significant amount of energy, but since a microwave is only active for minutes at a time and is mostly on standby, its power consumption overall is not very high. The additional circuitry added to the microwave systems also uses very little energy, limited to 5W at most or even lower if on standby.

Manufacturability
The connections to the microwave keypad caused many issues during the manufacturing process. The keypad's flexible printed circuit board is flat and needs to be twisted in a certain orientation to connect to our additional circuitry, which causes stress on the cables. The connection between the jumper wires and the FPC was also loose. These reasons contributed to the unreliability of the microwave keypad component of our system. Also, since this specific microwave is old and not mass manufactured anymore, the procedure for manufacturing this system not be replicable and be slightly different with a different microwave.

Sustainability
Maintainance of this system will be difficult. It is easy to make software changes due to the USB cable connecting to the Arduino being easily accessible. However, physical changes to this system would require opening up the microwave again. The current system requires a cellular phone plan, which usually requires a monthly payment, making it difficult to sustain. This project relies on PCB manufacturing, which might raise some sustainability concerns, as PCB manufacturing makes use of mined copper and silicon, which impacts Earth’s natural resources. Upgrades to the system might improve the wiring, and use custom wiring that is more flexible or flat. Currently, the use of jumper wires makes the internals of the system messy and tangled, making it difficult to upgrade.

Ethical
Since the Phone Microwave allows users to remotely control a microwave, there could be ethical concerns with unauthorized use of the system. An attacker might gain control of the microwave, running it constantly, which might damage the microwave and pose a hazard to others, or they could raise the energy costs for the owner of that microwave. Therefore, it is important to properly secure this system against unauthorized use.

Health and Safety
There were several health and safety concerns associated with the manufacturing of the system. Since a microwave contains uses very high voltage capacitors (typically thousands of volts), it is essential to avoid operating on the microwave when it is powered or in use.
Soldering was involved in the manufacturing of the system, and the person soldering must also take care to wear safety glasses, avoid inhaling fumes from soldering, and avoid burning themselves during the process. Concerns about a product such as this might also be raised during its use. Many people have concerns about using a microwave due to its use of radiation. While a microwave does use radiation to heat up its food, the radiation is non-ionizing and contained inside the oven, which makes it safe for use. However, the microwave should not be run empty, and the use of metal should be avoided as it could damage the microwave electronics.

**Social and Political**

This microwave system is primarily for personal use, and microwaves and smart devices are already common household appliances. This means there is negligible social or political impact with the Phone Microwave.

**Development**

Describe any new tools or techniques used for either development or analysis that you learned independently during your project.

Some new tools we learned for this project were the use of AT commands for communicating with cellular devices. We also learned how to design and create a circuit that could create a low current “short”, in order to simulate a keypad press that a microcontroller would detect. We had already learned how to use UART to communicate between embedded devices, and that skill was further developed for this project. Altium was the tool that we learned how to use in order to design a custom PCB for this project.
Appendix B: Datasheets and Resources
Montgomery Ward Model KSA-8068T Microwave Schematic

CD4051B 8-channel MUX Datasheet

Arduino Libraries
https://docs.arduino.cc/learn/built-in-libraries/eeprom

Arduino Mega Pinout
https://docs.arduino.cc/static/2de2c8ff00fc05065634e3823a9266c4/A000067-pinout.png

Arduino Mega Datasheet
https://docs.arduino.cc/static/ce41a5c2e7640463424a84ee52205224/A000067-datasheet.pdf

Waveshare SIM Module Wiki
https://www.waveshare.com/wiki/SIM7600G-H_4G_HAT

Waveshare SIM Module Hardware Schematic
https://www.waveshare.com/w/upload/f/fd/SIM7600_Series_Hardware_Design_V1.02.pdf

Waveshare SIM Module AT Commands
Appendix C: Images of Phone Microwave
Appendix D: Schematic of PCB for Microwave Control
Appendix E: Code

The most up to date version of the code for the Phone Microwave system can be found in this GitHub repository, but the code will also be pasted here.

https://github.com/KKhanhH/PhoneMicrowave

ArduinoCode.ino

```cpp
#include <math.h>
#include <EEPROM.h>
#include "src/SimCom/SimCom.h"
#include "src/Keypad/Keypad.h"
#include "src/MicrowaveControl/MicrowaveControl.h"
#include "src/PresetFoods/PresetFoods.h"

#define KEYPAD_COL_START 22
#define KEYPAD_ROW_START 28

#define INH_ROW_8 34
#define INH_ROW_9 35
#define INH_ROW_10 36
#define INH_ROW_11 37

#define CH_SELECTOR_0 40
#define CH_SELECTOR_1 41
#define CH_SELECTOR_2 42

bool onCall = false;
bool isMicrowaving = false;

SIM7600 simModule(Serial1);

Keypad keypad = Keypad(KEYPAD_ROW_START, KEYPAD_COL_START);

MicrowaveControl mcu = MicrowaveControl(INH_ROW_8, INH_ROW_9, INH_ROW_10, INH_ROW_11,
CH_SELECTOR_0, CH_SELECTOR_1, CH_SELECTOR_2);

bool setPin(char *pinStr) {
    int i = 0;
    for(i = 0; pinStr[i] != '\0' && i < 4; ++i) {
        Serial.println(pinStr[i]);
        if(i > 4 || pinStr[i] < '0' || pinStr[i] > '9') {
            return false;
        }
    }
    for(i = 0; i < 4; ++i) {
        EEPROM.update(i,pinStr[i]);
    }
    return true;
}

void getPin() {
    for(int i = 0; i < 4; ++i) {
```
pinCode[i] = EEPROM.read(i);
}

// Unused function for texting the location of the microwave with a Google Maps link
void textGPS(const char* phone_number) {
    char latStr[16];
    char longStr[16];
    char replyStr[128];

    SIM7600::GPSStruct gpsData = simModule.getGPSLocation(45000);
    if (!gpsData.status) {
        simModule.sendSMS(phone_number, "Failed to get GPS data");
    }

dtostrf(gpsData.latitude, 3, 8, latStr);
dtostrf(gpsData.longitude, 4, 8, longStr);
sprintf(replyStr,
    "https://www.google.com/maps/search/?api=1&query=%s%%2C%s", latStr,
    longStr);
simModule.sendSMS(phone_number, replyStr);
}

void initCall(char* dataBuffer, int bufferSize) {
    char phone_number[30] = "\0";
    int number_length = 0;
    char* token;

    // Get the phone number of the phone calling
    simModule.sendATCommand("AT+CLCC", 1000, dataBuffer, bufferSize);
    token = strtok(dataBuffer, ",");
    for (int i = 0; i < 5; i++) token = strtok(NULL, ",");
    strcpy(phone_number, token + 1);

    // Null terminate the number string
    number_length = strlen(phone_number);
    phone_number[--number_length] = '\0';

    // Print phone number
    Serial.print("Call from: ");
    Serial.println(phone_number);

    // Answer Phone Call
    simModule.sendATCompare("ATA", 500, 0);

getPin();

    // Set up TTS settings and play welcome message
    simModule.sendATCommand("AT+CDTAM=1", 500, 0);
    simModule.sendATCompare("AT+CTTSPARAM=2,3,0,1,2", 500, 0);
    simModule.sendTTS("Please enter pin code");
onCall = true;
}

void handleCall(char* dataBuffer, int bufferSize) {
    char* strPtr = dataBuffer;
}
static bool isUnlocked = false;
static int lockIndex = 0;

// Check if call has ended
if (strstr(dataBuffer, "VOICE CALL: END:")) ||
    strstr(dataBuffer, "NO CARRIER")) {
    Serial.print("Call Ended");
    onCall = false;
    lockIndex = 0;
    isUnlocked = false;
}

// Search for DTMF data
strPtr = strstr(strPtr, "+RXDTMF: ");
while (strPtr) {
    char keyPressed = strPtr[9];
    if(isUnlocked == false) {
        if(keyPressed == ' *') {
            lockIndex = 0;
        } else if(keyPressed != pinCode[lockIndex]) {
            onCall = false;
            lockIndex = 0;
            // Hang up call
            simModule.sendATCompare("AT+CHUP", 500, 0);
        } else {
            ++lockIndex;
            if(lockIndex == 4) {
                simModule.sendTTS("Welcome to the Phone Microwave");
                isUnlocked = true;
                lockIndex = 0;
            }
        }
    } else {
        // Match key pressed to microwave button
        Keypad::readPin button;
        bool success = true;
        char *response;
        switch(powerLevel) {
            case 1:
                response = "SET POWER LEVEL TO LOW. TYPE START TO CONFIRM, CANCEL OTHERWISE."
                button = Keypad::BTN_LOW;
                break;
            case 2:
                response = "SET POWER LEVEL TO MEDIUM. TYPE START TO CONFIRM, CANCEL OTHERWISE."
                button = Keypad::BTN_LOW;
                break;
            case 3:
                response = "SET POWER LEVEL TO HIGH. TYPE START TO CONFIRM, CANCEL OTHERWISE."
                button = Keypad::BTN_LOW;
                break;
            default:
                response = "INVALID POWER LEVEL REQUESTED."
                button = Keypad::BTN_LOW;
                break;
        }
        // Simulate the microwave button press
        mcu.simulateButton(button.rowPin, button.colPin);
    }
    // Scan for next DTMF key press
    strPtr++;
    strPtr = strstr(strPtr, "+RXDTMF: ");
}

void powerLvlSms(int powerLevel, char *responseBuffer, size_t len) {
    Keypad::readPin button;
    bool success = true;
    char *response;
    switch(powerLevel) {
        case 1:
            response = "SET POWER LEVEL TO LOW. TYPE START TO CONFIRM, CANCEL OTHERWISE.");
            button = Keypad::BTN_LOW;
            break;
        case 2:
            response = "SET POWER LEVEL TO MEDIUM. TYPE START TO CONFIRM, CANCEL OTHERWISE.");
            button = Keypad::BTN_LOW;
            break;
        case 3:
            response = "SET POWER LEVEL TO HIGH. TYPE START TO CONFIRM, CANCEL OTHERWISE.");
            button = Keypad::BTN_LOW;
            break;
        default:
            response = "INVALID POWER LEVEL REQUESTED.");
            button = Keypad::BTN_LOW;
            break;
    }
}
button = Keypad::BTN_MED_LOW_DEFROST;
break;
case 3:
    response = "SET POWER LEVEL TO MEDIUM. TYPE START TO CONFIRM, CANCEL OTHERWISE."
    button = Keypad::BTN_MEDIUM;
    break;
case 4:
    response = "SET POWER LEVEL TO MEDIUM HIGH. TYPE START TO CONFIRM, CANCEL OTHERWISE."
    button = Keypad::BTN_MED_HIGH;
    break;
case 5:
    response = "SET POWER LEVEL TO HIGH. TYPE START TO CONFIRM, CANCEL OTHERWISE."
    button = Keypad::BTN_HIGH;
    break;
default:
    response = "\"POWER <LEVEL>\", WHERE <LEVEL> IS A NUMBER FROM 1-5, WHERE 5 IS HIGHEST."
    success = false;
}
strncpy(responseBuffer, response, len);
if(success) {
    Serial.print(button.rowPin);
    Serial.print(button.colPin);
    mcu.simulateButton(button.rowPin, button.colPin);
}
}

void defrostSms(int defrostOption, char *responseBuffer, size_t len) {
    bool success = true;
    char *response;
    switch(defrostOption) {
    case 1:
        response = "DEFROSTING 1LB GROUND MEAT. TYPE START TO CONFIRM, CANCEL OTHERWISE."
        break;
    case 2:
        response = "DEFROSTING 2LB PORK CHOP. TYPE START TO CONFIRM, CANCEL OTHERWISE."
        break;
    case 3:
        response = "DEFROSTING 2LB STEAKS. TYPE START TO CONFIRM, CANCEL OTHERWISE."
        break;
    case 4:
        response = "DEFROSTING 2LB CHICKEN PIECES. TYPE START TO CONFIRM, CANCEL OTHERWISE."
        break;
    case 5:
        response = "DEFROSTING 3LB WHOLE CHICKEN. TYPE START TO CONFIRM, CANCEL OTHERWISE."
        break;
    default:
        response = "\"DEFROST <OPT>\", 1: 1LB GROUND MEAT, 2: 2LB PORK CHOP, 3: 2LB STEAKS, 4: 2LB CHICKEN PIECES, 5: 3LB WHOLE CHICKEN."
        success = false;
    }
    strncpy(responseBuffer, response, len);
    if(success) {
        Keypad::readPin button = keypad.dtmfLookup('0'+defrostOption);
        mcu.simulateButton(Keypad::BTN_EASY_DEFROST.rowPin, Keypad::BTN_EASY_DEFROST.colPin);
        mcu.simulateButton(button.rowPin, button.colPin);
    }
}
void reheatSms(int reheatOption, char *responseBuffer, size_t len) {
    bool success = true;
    char *response;
    switch(reheatOption) {
        case 1:
            response = "REHEATING 1CUP CASSEROLE. TYPE START TO CONFIRM, CANCEL OTHERWISE.";
            break;
        case 2:
            response = "REHEATING 1 DINNER PLATE. TYPE START TO CONFIRM, CANCEL OTHERWISE.";
            break;
        case 3:
            response = "REHEATING 10-12oz FROZEN ENTREE. TYPE START TO CONFIRM, CANCEL OTHERWISE.";
            break;
        case 4:
            response = "REHEATING 1CUP SOUP. TYPE START TO CONFIRM, CANCEL OTHERWISE.";
            break;
        case 5:
            response = "REHEATING 1CUP VEGETABLES. TYPE START TO CONFIRM, CANCEL OTHERWISE.";
            break;
        default:
            response = "REHEAT <OPT>, 1: 1CUP CASSEROLE, 2: 1 DINNER PLATE, 3: 10-12oz FROZEN ENTREE, 4: 1CUP SOUP, 5: 1CUP VEGETABLES.";
            success = false;
    }
    strncpy(responseBuffer, response, len);
    if(success) {
        Keypad::readPin button = keypad.dtmfLookup('0'+reheatOption);
        mcu.simulateButton(Keypad::BTN_EASY_REHEAT.rowPin, Keypad::BTN_EASY_REHEAT.colPin);
        mcu.simulateButton(button.rowPin, button.colPin);
    }
}

void handleSMS(SIM7600::SMSStruct smsInput) {
    char messageCpy[200] = "";
    strncpy(messageCpy, smsInput.message, sizeof(smsInput.message));

    char *token = strtok(messageCpy, " ");
    char response[180] = "";
    char *postConvert = NULL;
    if(strncmp(token , "PIN", 3) == 0) {
        token = strtok(NULL, " ");
        if(token == NULL || setPin(token) == false) {
            strncpy(response,"PIN <4 DIGIT CODE>\", 180);
        } else {
            strncpy(response,"NEW PIN CODE SET", 180);
        }
    } else if(strncmp(token , "POWER", 5) == 0) {
        token = strtok(NULL, " ");
        int powerVal = (token == NULL)? 0: strtol(token, &postConvert, 10);
        powerLv1Sms(powerVal, response, 180);
    } else if(strncmp(token , "DEFROST", 7) == 0) {
        token = strtok(NULL, " ");
        int defrostVal = (token == NULL)? 0: strtol(token, &postConvert, 10);
    }
defrostSms(defrostVal, response, 180);
} else if(strcmp(token, "REHEAT", 6) == 0) {
    token = strtok(NULL, ", ");
    int reheatVal = (token == NULL) ? 0 : strtol(token, &postConvert, 10);
    reheatSms(reheatVal, response, 180);
} else if(strcmp(token, "PRESET", 6) == 0) {
    token = strtok(NULL, ", ");
    int presetVal = (token == NULL) ? 0 : strtol(token, &postConvert, 10);
    handleFenetFood(mcu, presetVal, response, 180);
} else if(strcmp(token, "CANCEL", 6) == 0) {
    mcu.simulateButton(Keypad::BTN_STOP_CANCEL.rowPin, Keypad::BTN_STOP_CANCEL.colPin);
    mcu.simulateButton(Keypad::BTN_STOP_CANCEL.rowPin, Keypad::BTN_STOP_CANCEL.colPin);
    strncpy(response,"CANCELLING", 180);
} else if(strcmp(token, "START", 5) == 0) {
    mcu.simulateButton(Keypad::BTN_START.rowPin, Keypad::BTN_START.colPin);
    strncpy(response,"STARTING OPERATION.", 180);
} else {
    strncpy(response,"AVAILABLE COMMANDS:\n\nDEFROST\nREHEAT\nPRESET", 180);
}

simModule.sendSMS(smsInput.number, response);

}

void setup() {
    Serial.begin(115200);
    Serial.println("Initializing");
   Serial.begin(9600);
    keypad.initializePins();
    mcu.initializePins();
    delay(500);
    mcu.simulateButton(Keypad::BTN_STOP_CANCEL.rowPin, Keypad::BTN_STOP_CANCEL.colPin);
    simModule.initConfig(15000);
    Serial.println("READY");
    delay(500);
}

void loop() {
    static char dataBuffer[256] = {0};
    static SIM7600::SMSStruct smsData = {};
    static boolean btnPressed = false;
    char* index;
    if (simModule.readToBuffer(dataBuffer, sizeof(dataBuffer)) > 0) {
        if (((index = strstr(dataBuffer, "+CMTI: \"",\") ) != NULL) {
            index += 12;
            smsData = {}; 
            simModule.readSMS(atoi(index), smsData);
            handleSMS(smsData);
        }
        if (onCall == false && (index = strstr(dataBuffer, "RING") ) != NULL) {
            index += 4;
            initCall(dataBuffer, sizeof(dataBuffer));
        }
        if (onCall == true) {
            handleCall(dataBuffer, sizeof(dataBuffer));
        }
    }
}
if (onCall == false) {
    Keypad::readPin keypadRead = keypad.readKeypad();
    if (keypadRead != Keypad::BTN_UNPRESSED && !btnPressed) {
        Serial.println(keypad.buttonStr(keypadRead));
        mcu.simulateButton(keypadRead.rowPin, keypadRead.colPin);
        btnPressed = true;
    } else if (keypadRead == Keypad::BTN_UNPRESSED) {
        btnPressed = false;
    }
}
Keypad.h

/**
 * @file Keypad.h
 * @brief This file contains the declarations for the Keypad class and button structs.
 *
 * The Keypad class provides an interface for reading in button inputs from the microwave keypad.
 * Constant static structs are declared for easier comparisons to buttons.
 */

#ifndef KEYPAD_H
#define KEYPAD_H

#include <Arduino.h>

class Keypad {
    private:
        int _rowStart;
        int _colStart;

    public:
        /**
         * @brief Structure representing an individual microwave button function.
         * Contains pin number matching microwave diagram and has overloaded
         * comparison functions.
         */
        struct readPin {
            int colPin;
            int rowPin;

            bool operator==(const readPin &other) const {
                return (colPin == other.colPin && rowPin == other.rowPin);
            }

            bool operator!=(const readPin &other) const {
                return (*this == other) == false;
            }
        };

    /*----------------------BUTTON MAPPING------------------------*/
    const static readPin BTN_TIME_MINDER;
    const static readPin BTN_CLOCK;
    const static readPin BTN_EASY_REHEAT;
    const static readPin BTN_START;
    const static readPin BTN_STOP_CANCEL;
    const static readPin BTN_INSTANT_MINUTE;
    const static readPin BTN_EASY_DEFROST;
    const static readPin BTN_HIGH;
    const static readPin BTN_MED_HIGH;
    const static readPin BTN_MEDIUM;
    const static readPin BTN_MED_LOW_DEFROST;
    const static readPin BTN_ONE;
    const static readPin BTN_TWO;
    const static readPin BTN_THREE;
    const static readPin BTN_LOW;
    const static readPin BTN_FOUR;
    const static readPin BTN_FIVE;
    const static readPin BTN_SIX;
    const static readPin BTN_SEVEN;

};

#endif
const static readPin BTN_EIGHT;
const static readPin BTN_NINE;
const static readPin BTN_ZERO;
const static readPin BTN_UNPRESSED;

/**
 * @brief Construct a new Keypad object as a physical input for the
 * PhoneMicrowave System. Row pins and column pins must be consecutive.
 * Row pins are set to inputs with pullup, while col pins are set to output.
 * @param rowStart Pin number corresponding to start of row pins for keypad
 * @param colStart Pin number corresponding to start of column pins for
 * keypad
 */
Keypad(int rowStart, int colStart);

/**
 * @brief Sets column pin output initial state to HIGH.
 */
void initializePins();

/**
 * @brief Scans keypad for a press and if pressed, return a struct with matching pins.
 * @return readPin Struct containing pins matching button that was pressed.
 * Can be compared to constant structs that match the pressed button.
 */
readPin readKeypad();

/**
 * @brief Looks up the readPin struct for a regular number button on keypad (not function button)
 * @param buttonNumber A single digit positive integer
 * @return readPin Struct containing pins matching number button
 */
readPin dtmfLookup(int buttonNumber);

/**
 * @brief Get string representation of button pressed for debugging purposes
 * @return const char* Button string.
 */
const char* buttonStr(readPin);
};
Keypad.cpp

#include "Keypad.h"

/*-----------------------------------*/

extern const Keypad::readPin Keypad::BTN_TIME_MINDER = {2, 10};
extern const Keypad::readPin Keypad::BTN_CLOCK = {2, 9};
extern const Keypad::readPin Keypad::BTN_EASY_REHEAT = {2, 8};
extern const Keypad::readPin Keypad::BTN_START = {3, 11};
extern const Keypad::readPin Keypad::BTN_STOP_CANCEL = {3, 10};
extern const Keypad::readPin Keypad::BTN_INSTANT_MINUTE = {3, 9};
extern const Keypad::readPin Keypad::BTN_EASY_DEFROST = {3, 8};
extern const Keypad::readPin Keypad::BTN_HIGH = {4, 11};
extern const Keypad::readPin Keypad::BTN_MED_HIGH = {4, 10};
extern const Keypad::readPin Keypad::BTN_MEDIUM = {4, 9};
extern const Keypad::readPin Keypad::BTN_MED_LOW_DEFROST = {4, 8};
extern const Keypad::readPin Keypad::BTN_ONE = {5, 11};
extern const Keypad::readPin Keypad::BTN_TWO = {5, 10};
extern const Keypad::readPin Keypad::BTN_THREE = {5, 9};
extern const Keypad::readPin Keypad::BTN_LOW = {5, 8};
extern const Keypad::readPin Keypad::BTN_FOUR = {6, 11};
extern const Keypad::readPin Keypad::BTN_FIVE = {6, 10};
extern const Keypad::readPin Keypad::BTN_SIX = {6, 9};
extern const Keypad::readPin Keypad::BTN_SEVEN = {7, 11};
extern const Keypad::readPin Keypad::BTN_EIGHT = {7, 10};
extern const Keypad::readPin Keypad::BTN_NINE = {7, 9};
extern const Keypad::readPin Keypad::BTN_ZERO = {7, 8};
extern const Keypad::readPin Keypad::BTN_UNPRESSED = {0, 0};

Keypad::Keypad(int rowStart, int colStart)
  : _rowStart(rowStart), _colStart(colStart) {
    // Initialize row pins to input, with a pullup to avoid floating inputs
    for (int r = _rowStart; r < _rowStart + 4; r++) {
      pinMode(r, INPUT_PULLUP);
    }
    // Initialize col pins to output, with active low
    for (int c = _colStart; c < _colStart + 6; c++) {
      pinMode(c, OUTPUT);
    }
  }

void Keypad::initializePins() {
  // Initialize col pins to high output
  for (int c = _colStart; c < _colStart + 6; c++) {
    digitalWrite(c, HIGH);
  }
}

Keypad::readPin Keypad::readKeypad() {
  readPin result = {0, 0};
  // cycle through each column pin
  for (int c = _colStart; c < _colStart + 6; c++) {
    digitalWrite(c, LOW);
    // evaluate which row pin is low
  }
}
for (int r = _rowStart; r < _rowStart + 4; r++) {
    int roweval = digitalRead(r);
    if (roweval != HIGH) {
        result.rowPin = r - _rowStart + 8;
        result.colPin = c - _colStart + 2;
        digitalWrite(c, HIGH);
        return result;
    }
}

digitalWrite(c, HIGH);
}
return result;
}

const char* Keypad::buttonStr(readPin input) {
    if (input == BTN_TIME_MINDER)  
        return "BTN_TIME_MINDER";
    else if (input == BTN_CLOCK) 
        return "BTN_CLOCK";
    else if (input == BTN_EASY_REHEAT) 
        return "BTN_EASY_REHEAT";
    else if (input == BTN_START) 
        return "BTN_START";
    else if (input == BTN_STOP_CANCEL) 
        return "BTN_STOP_CANCEL";
    else if (input == BTN_INSTANT_MINUTE) 
        return "BTN_INSTANT_MINUTE";
    else if (input == BTN_EASY_DEFROST) 
        return "BTN_EASY_DEFROST";
    else if (input == BTN_HIGH) 
        return "BTN_HIGH";
    else if (input == BTN_MED_HIGH) 
        return "BTN_MED_HIGH";
    else if (input == BTN_MEDIUM) 
        return "BTN_MEDIUM";
    else if (input == BTN_MED_LOW_DEFROST) 
        return "BTN_MED_LOW_DEFROST";
    else if (input == BTN_LOW) 
        return "BTN_LOW";
    else if (input == BTN_ONE) 
        return "BTN_ONE";
    else if (input == BTN_TWO) 
        return "BTN_TWO";
    else if (input == BTN_THREE) 
        return "BTN_THREE";
    else if (input == BTN_FOUR) 
        return "BTN_FOUR";
    else if (input == BTN_FIVE) 
        return "BTN_FIVE";
    else if (input == BTN_SIX) 
        return "BTN_SIX";
    else if (input == BTN_SEVEN) 
        return "BTN_SEVEN";
    else if (input == BTN_EIGHT) 
        return "BTN_EIGHT";
}
return "BTN_EIGHT";
else if (input == BTN_NINE)
    return "BTN_NINE";
else if (input == BTN_ZERO)
    return "BTN_ZERO";
else if (input == BTN_UNPRESSED)
    return "BTN_UNPRESSED";
}

Keypad::readPin Keypad::dtmfLookup(int buttonNumber) {
    switch (buttonNumber) {
        case '0':
            return Keypad::BTN_ZERO;
        case '1':
            return Keypad::BTN_ONE;
        case '2':
            return Keypad::BTN_TWO;
        case '3':
            return Keypad::BTN_THREE;
        case '4':
            return Keypad::BTN_FOUR;
        case '5':
            return Keypad::BTN_FIVE;
        case '6':
            return Keypad::BTN_SIX;
        case '7':
            return Keypad::BTN_SEVEN;
        case '8':
            return Keypad::BTN_EIGHT;
        case '9':
            return Keypad::BTN_NINE;
        case '*':
            return Keypad::BTN_STOP_CANCEL;
        case '#':
            return Keypad::BTN_START;
        default:
            return Keypad::BTN_UNPRESSED;
    }
}
# Microw ave Control.h

/**
 * @file MicrowaveControl.h
 * @brief Header file for the MicrowaveControl class.
 * This file declares the interface for simulating button presses on a microwave.
 */

#ifndef MICROWAVECONTROL_H
#define MICROWAVECONTROL_H

#include <Arduino.h>

class MicrowaveControl {
private:
  int _inhPin0;
  int _inhPin1;
  int _inhPin2;
  int _inhPin3;
  int _chSelPin0;
  int _chSelPin1;
  int _chSelPin2;
public:
  /**
   * @brief Constructor.
   * @param inhPin0 The input enable pin for the mux corresponding to 1st row pin.
   * @param inhPin1 The input enable pin for the mux corresponding to 2nd row pin.
   * @param inhPin2 The input enable pin for the mux corresponding to 3rd row pin.
   * @param inhPin3 The input enable pin for the mux corresponding to 4th row pin.
   * @param chSelPin0 The channel select pin for the 1st bit of the muxes.
   * @param chSelPin1 The channel select pin for the 2nd bit of the muxes.
   * @param chSelPin2 The channel select pin for the 3rd bit of the muxes.
   */
  MicrowaveControl(int inhPin0, int inhPin1, int inhPin2, int inhPin3,
                   int chSelPin0, int chSelPin1, int chSelPin2);

  /**
   * @brief Simulates a button press on the microwave keypad.
   * @param rowNum The row number of the button to be pressed (8-11).
   * @param colNum The column number of the button to be pressed (2-7).
   */
  void simulateButton(int rowNum, int colNum);

  /**
   * @brief Initializes the pins used to control the keypad.
   * This function should be called before any button presses are simulated.
   */
  void initializePins();
};
#endif // MICROWAVECONTROL_H
```cpp
#include "MicrowaveControl.h"

MicrowaveControl::MicrowaveControl(int inhPin0, int inhPin1, int inhPin2,
                                   int inhPin3, int chSelPin0, int chSelPin1,
                                   int chSelPin2)
    : _inhPin0(inhPin0),
      _inhPin1(inhPin1),
      _inhPin2(inhPin2),
      _inhPin3(inhPin3),
      _chSelPin0(chSelPin0),
      _chSelPin1(chSelPin1),
      _chSelPin2(chSelPin2) {
    pinMode(_inhPin0, OUTPUT);
    pinMode(_inhPin1, OUTPUT);
    pinMode(_inhPin2, OUTPUT);
    pinMode(_inhPin3, OUTPUT);
    pinMode(_chSelPin0, OUTPUT);
    pinMode(_chSelPin1, OUTPUT);
    pinMode(_chSelPin2, OUTPUT);
}

void MicrowaveControl::initializePins() {
    digitalWrite(_inhPin0, HIGH);
    digitalWrite(_inhPin1, HIGH);
    digitalWrite(_inhPin2, HIGH);
    digitalWrite(_inhPin3, HIGH);
    digitalWrite(_chSelPin0, LOW);
    digitalWrite(_chSelPin1, LOW);
    digitalWrite(_chSelPin2, LOW);
}

void MicrowaveControl::simulateButton(int rowNum, int colNum) {
    // Row num 8-11, col num 2-7
    int rowOffsetted = rowNum - 8;
    // No button pressed
    if(rowNum == 0 && colNum == 0) {
        return;
    }
    digitalWrite(_chSelPin0, colNum & 0x1);
    digitalWrite(_chSelPin1, colNum & 0x2);
    digitalWrite(_chSelPin2, colNum & 0x4);
    // Simulate button press
    switch(rowOffsetted) {
    case 0:
        digitalWrite(_inhPin0, LOW);
        break;
    case 1:
        digitalWrite(_inhPin1, LOW);
        break;
    case 2:
        digitalWrite(_inhPin2, LOW);
        break;
    case 3:
    ```
digitalWrite(_inhPin3, LOW);
    break;
    default:
    break;
}

// Keep button held down
delay(20);
// Unpress
digitalWrite(_inhPin0, HIGH);
digitalWrite(_inhPin1, HIGH);
digitalWrite(_inhPin2, HIGH);
digitalWrite(_inhPin3, HIGH);
delay(120);
}
SimCom.h
/**
 * @file SimCom.h
 * @brief This file contains the declarations of the SIM7600 class.
 * @see
 * The SIM7600 class provides an interface for working with the SIMCOM SIM7600
 * cellular module. It provides functions to allow the user to send and receive
 * SMS messages, and read GPS data.
 *
 */

#ifndef SIMCOM_H
#define SIMCOM_H

#include <Arduino.h>

class SIM7600 {
    private:
    Stream* _simSerial;

    public:
    /**
     * @brief Overloaded constructor for SIM7600 object
     *
     * @param simSerial The serial port to communicate with the SIM7600 module.
     *
     * SIM7600(Stream* simSerial);
     *
     */
    SIM7600(Stream* simSerial);

    /**
     * @brief Overloaded constructor for SIM7600 object
     *
     * @param simSerial The serial port to communicate with the SIM7600 module.
     *
     */
    SIM7600(Stream& simSerial);

    /**
     * @brief A struct to store data & metadata about an SMS message.
     *
     * The phone number is limited to 29 chars, message is limited to 199 chars,
     * and the time string is limited to 31 chars.
     *
     */
    struct SMSStruct {
        char number[30];
        char message[200];
        char timeStr[32];
    };

    /**
     * @brief A struct to store GPS coordinates
     *
     */
    struct GPSStruct {
        double latitude;
        double longitude;
        bool status;
    };
    /**
     */
void emptyBuffer();

/**
 * @brief Reads data in serial buffer into a character buffer and null terminates it.
 * @param result The character buffer to write data into.
 * @param maxChars The maximum number of characters to write into the buffer.
 * @return Number of characters read, or 0 if nothing was read.
 */
int readToBuffer(char result[], int maxChars);

/**
 * @brief Sends an AT command to the sim module.
 * @param cmdStr The AT command string to send.
 * @param timeout An integer specifying how long to wait for a response.
 * @param result The character buffer storing the response.
 * @param maxChars The maximum characters to write into the response buffer.
 * @return A boolean representing whether the response was received in time.
 */
bool sendATCommand(const char* cmdStr, int timeout, char* result, int maxChars);

/**
 * @brief Sends an AT command to the sim module and compares the response to an expected number of strings.
 * @param cmdStr The AT command string to send.
 * @param timeout An integer specifying how long to wait for a response.
 * @param expectedCount An integer specifying the number of expected responses.
 * @param ... The expected response strings.
 * @return An integer representing the index of the matching string, starting from 1, or 0 if there is no match.
 */
int sendATCompare(const char* cmdStr, int timeout, int expectedCount, ...);
This function sends an AT command to the SIM7600 without waiting for a response. The response is not buffered, so it should be used for simple, one-off commands only.

@param cmdStr The AT command string to send.

/**
 * @brief Initializes the SIM7600 configuration by setting various parameters.
 * The function initializes the configuration of the SIM7600 module by setting various parameters such as the SMS text mode, GPS power state, etc. During the initialization, the function sends various AT commands to the module and expects certain responses.
 * @param timeout The time in milliseconds to wait for the SIM7600 to connect to the carrier.
 */
void initConfig(unsigned long timeout);

/**
 * @brief Sends an SMS message to the specified phone number.
 * @param number The phone number to send the message to.
 * @param msg The message to send.
 * @return A boolean representing whether the message was sent successfully.
 */
bool sendSMS(const char* number, const char* msg);

/**
 * @brief Read an SMS message from the SIM module's message storage.
 * Reads an SMS message from the SIM module's message storage into the given SMSStruct. The function blocks until either the message is read or the timeout expires.
 * @param index The index of the message to read. The oldest message has an index of 1, while the most recent message has an index of N, where N is the total number of messages.
 * @param smsData A reference to a SMSStruct that will be filled with the contents of the message.
 * @return True if the message was read successfully, false otherwise.
 */
bool readSMS(int index, SMSStruct& smsData);

/**
 * @brief Get the GPS location of the SIM module.
 * Requests the GPS location of the SIM module from the module's GPS subsystem and returns the latitude and longitude as a GPSStruct.
 * The function blocks until a response is received from the module or the timeout expires.
 */
* @param timeout The maximum amount of time to wait for a response from the module.
* @return A GPSStruct representing the current location of the SIM module.
* /
GPSStruct getGPSLocation(unsigned long timeout);

/**
* @brief Parse a GPS location string and return the latitude and longitude as a GPSStruct.
* Takes a NMEA GPS location string and returns the latitude and longitude as a GPSStruct. If the GPS subsystem is not able to provide a valid fix, the returned GPSStruct will have status set to false.
* @param GPSBuffer The NMEA GPS location string to parse.
* @return A GPSStruct representing the latitude and longitude in the GPS location string.
* /
GPSStruct formatGPS(char* GPSBuffer);

/**
* @brief Send a text-to-speech message to a connected phone.
* Sends a text-to-speech message to a connected phone by calling the SIM module's text-to-speech API. The message is synthesized on the module and then played through the phone's earpiece.
* @param message The message to be spoken.
* /
void sendTTS(const char* message);

/**
* @brief Stop a text-to-speech message that is currently being played.
* Sends an AT command to the SIM module to stop any text-to-speech message that is currently being played.
* /
void stopTTS();

void handleCall();

};
```cpp
// SimCom.cpp

#include "SimCom.h"

SIM7600::SIM7600(Stream* simSerial) : _simSerial(simSerial) {}
SIM7600::SIM7600(Stream& simSerial) : _simSerial(&simSerial) {}

void SIM7600::emptyBuffer() {
    while (_simSerial->available() > 0) _simSerial->read();
}

int SIM7600::readToBuffer(char result[], int maxChars) {
    int index = 0;
    if (_simSerial->available() > 0) {
        index = _simSerial->readBytes(result, maxChars - 1);
    }
    result[index] = '\0'; // Length of input, or 0 if nothing read
    return index;
}

void SIM7600::sendImmediate(const char* cmdStr) {
    _simSerial->println(cmdStr);
}

bool SIM7600::sendATCommand(const char* cmdStr, int timeout, char* result, int maxChars) {
    bool receivedResponse = false;
    // Clean the input buffer
    emptyBuffer();
    // Send the AT command
    _simSerial->println(cmdStr);
    unsigned long startTime = millis();
    do {
        if (readToBuffer(result, 256) != 0) {
            receivedResponse = true;
        }
    } while (receivedResponse == false && (millis() - startTime) < timeout);
    return receivedResponse;
}

int SIM7600::sendATCompare(const char* cmdStr, int timeout, int expectedCount, ...) {
    char responseBuffer[256] = {0};
    va_list expectedStrs;
    va_start(expectedStrs, expectedCount);
    int result = sendATCommand(cmdStr, timeout, responseBuffer, 256);
    int answer = 0;
    if (result != 0) {
        for (int i = 0; i < expectedCount; i++) {
            char* arg = va_arg(expectedStrs, char*);
            if (strstr(responseBuffer, arg) != NULL) {
                answer = i + 1;
                break;
            }
        }
    }
    va_end(expectedStrs);
    return answer;
}
```

49
void SIM7600::initConfig(unsigned long timeout) {
    Serial.println("Initiating Sim module");
    _simSerial->setTimeout(1000);

    int answer = 0;
    while (answer == 0) { // Send AT every 0.5 seconds and wait for the answer
        Serial.println("Sending AT");
        answer = sendATCompare("AT", 2000, 1, "OK");
        delay(500);
    }
    Serial.println("Setting SMS Mode to txt");
    sendATCompare("AT+CMGF=1", 1000, 1, "OK"); // sets the SMS mode to text
    sendATCompare("AT+CPMS="MT", "SM", "ME"", 1000, 1, "OK"); // Read & store msgs in flash, write with SIM
    sendATCompare("AT+CMGD=0,2", 1000, 1, "OK"); // delete already read messages

    // Check if the network has been registered, timeout after 15 seconds
    unsigned long startTime = millis();
    while (sendATCompare("AT+CREG?", 2000, 2, "+CREG: 0,1", "+CREG: 0,5") == 0 &&
           millis() - startTime < timeout) {
        Serial.println("Checking network registration");
        delay(500);
    }
    if (millis() - startTime > timeout)
        Serial.println("Cellular Network Registration timed out");
}

bool SIM7600::sendSMS(const char* number, const char* msg) {
    char smsCmd[30];

    sendATCompare("AT+CMGF=1", 1000, 1, "OK"); // sets the SMS mode to text
    snprintf(smsCmd, sizeof(smsCmd), "AT+CMGS="%s", number); // sets the SMS mode to text
    bool result = sendATCompare(smsCmd, 3000, 1, ">"); // sets the SMS mode to text
    char buffer[256] = {0};
    if (result) {
        _simSerial->print(msg);
        return sendATCommand("\x1A", 20000, buffer, sizeof(buffer));
    }
    return result;
}

bool SIM7600::readSMS(int index, SMSStruct& smsData) {
    char buffer[256] = {0};
    char metadata[96] = {0};
    char cmd[10] = {0};
    sprintf(cmd, "AT+CMGR="%d", index); // read and delete message at index
    if (!sendATCommand(cmd, 1000, buffer, sizeof(buffer))) return false;
    char* context1;
char* context2;
char* bufferToken = strtok_r(buffer, "\n", &context1);
bufferToken = strtok_r(NULL, "\n", &context1);

strcpy(metadata, bufferToken, sizeof(metadata));
char* msgDataToken = strtok_r(metadata, ",", &context2);
msgDataToken = strtok_r(NULL, ",", &context2);
strcpy(smsData.number, msgDataToken + 1, sizeof(smsData.number));
size_t len = strlen(smsData.number);
smsData.number[len - 1] = 0;

msgDataToken = strtok_r(NULL, ",", &context2);
msgDataToken = strtok_r(NULL, "\n", &context2);
strcpy(smsData.timeStr, msgDataToken);
strcpy(smsData.message, bufferToken, sizeof(smsData.message));

return true;
}

SIM7600::GPSStruct SIM7600::getGPSLocation(unsigned long timeout) {
    int i = 0;
    char responseBuffer[256] = {0};
    sendATCompare("AT+CGPS=1,1", 1000, 1, "OK");
    unsigned long startTime = millis();
    while ((millis() - startTime) < timeout) {
        bool answer = sendATCommand("AT+CGPSINFO", 1000, responseBuffer, 256);
        if (!answer) {
            Serial.println("Error occurred");
            return (GPSStruct){0, 0, false};
        }
        answer = false;
        if (char* location = strstr(responseBuffer, "+CGPSINFO: "+1)) {
            memmove(responseBuffer, location, strlen(responseBuffer) - (location - responseBuffer) + 1);
        }
        if (strstr(responseBuffer, ",,,,,,,") != NULL) {
            // Reset buffer and redo
            memset(responseBuffer, '\0', sizeof(responseBuffer));
            i = 0;
            answer = 0;
        } else {
            sendATCompare("AT+CGPS=0", 1000, 1, "OK:");
            break;
        }
        delay(1000);
    }
    if ((millis() - startTime) > timeout) {
        return (GPSStruct){0, 0, false};
    }

    return formatGPS(responseBuffer);
}

SIM7600::GPSStruct SIM7600::formatGPS(char* GPSBuffer) {


GPSStruct gpsData = {};
char* token;
token = strtok(GPSBuffer, ",");
gpsData.latitude = atof(token);
gpsData.latitude = ((int)gpsData.latitude / 100) + (fmod(gpsData.latitude, 100) / 60);

token = strtok(NULL, ",");
if (strcmp(token, "S") == 0) gpsData.latitude = gpsData.latitude * -1;

token = strtok(NULL, ",");
gpsData.longitude = atof(token);
gpsData.longitude = ((int)gpsData.longitude / 100) + (fmod(gpsData.longitude, 100) / 60);

token = strtok(NULL, ",");
if (strcmp(token, "W") == 0) gpsData.longitude = gpsData.longitude * -1;

gpsData.status = true;
return gpsData;
}

void SIM7600::sendTTS(const char* message) {
char cmd[256] = "";
sprintf(cmd, "AT+CTTS=2,\"%s\", message);
sendImmediate(cmd);
}

void SIM7600::stopTTS() { sendImmediate("AT+CTTS=0"); }
PresetFoods.h

```c
#ifndef PRESETFOODS_H
#define PRESETFOODS_H

#include "../MicrowaveControl/MicrowaveControl.h"
#include "../Keypad/Keypad.h"

void handlePresetFood(MicrowaveControl &mcu, int presetIndex, char *responseBuffer, size_t len);

#endif
PRESETFOODS_H
```

PresetFoods.cpp

```c
#include "PresetFoods.h"

struct PresetFood {
    char index[2];
    char name[16];
    char description[256];
    unsigned char stepCount;
    Keypad::readPin buttonSteps[8];
};

static PresetFood foods[3] = {
    {"1", "POPCORN", "1 Bag.", 3, {Keypad::BTN_TWO, Keypad::BTN_THREE, Keypad::BTN_ZERO}},
    {"2", "RICE", "1 Cup.", 5, {Keypad::BTN_ONE, Keypad::BTN_TWO, Keypad::BTN_ZERO, Keypad::BTN_ZERO, Keypad::BTN_MED_LOW_DEFROST}},
    {"3", "POTATO", "1 Baked Potato. Flip Half Way Through.", 3, {Keypad::BTN_ONE, Keypad::BTN_ZERO, Keypad::BTN_ZERO}}
};

void handlePresetFood(MicrowaveControl &mcu, int presetIndex, char *responseBuffer, size_t len) {
    int foodCount = sizeof(foods) / sizeof(PresetFood);
    int responseCounter = 0;
    if(presetIndex <= 0 || presetIndex > foodCount) {
        strcpy(responseBuffer, "PRESET <OPT>\n");
        responseCounter += 14;
        Serial.print("CRASH CHECK2\n");
        for(int i = 0; i < foodCount; ++i) {
            int size = snprintf(NULL, 0, "\n%s: %s - %s", foods[i].index, foods[i].name, foods[i].description);
            snprintf(responseBuffer + responseCounter, len - responseCounter, "\n%s: %s - %s", foods[i].index, foods[i].name, foods[i].description);
            responseCounter += size - 1;
        }
    } else {
        PresetFood &currentFood = foods[presetIndex - 1];
        snprintf(responseBuffer, len, "Cooking preset %s: %s - %s", currentFood.index, currentFood.name, currentFood.description);
        for(int i = 0; i < currentFood.stepCount; ++i) {
            mcu.simulateButton(currentFood.buttonSteps[i].rowPin, currentFood.buttonSteps[i].colPin);
        }
        mcu.simulateButton(Keypad::BTN_START.rowPin, Keypad::BTN_START.colPin);
    }
}
```