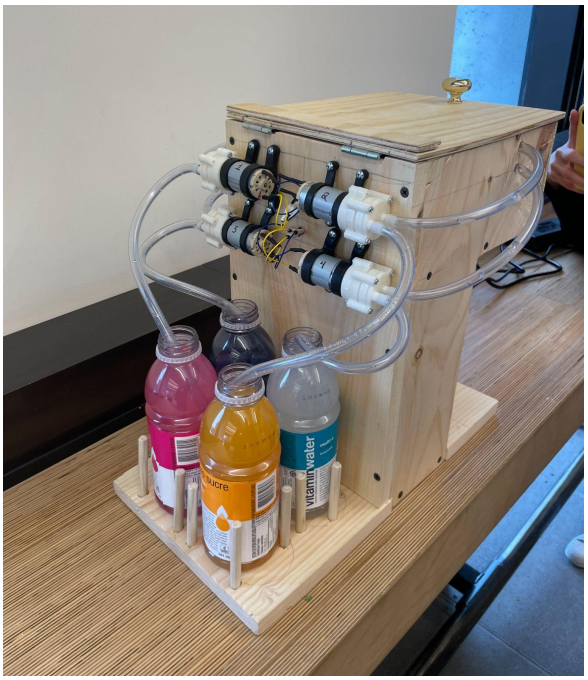
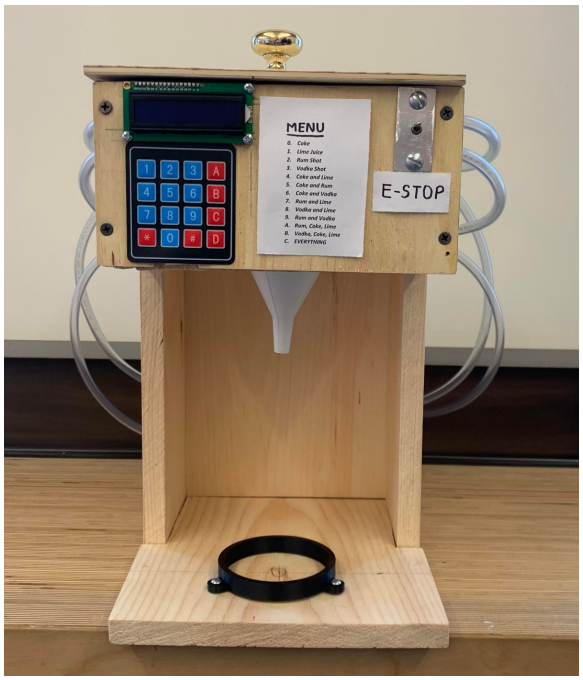
MIE438 Project Report

Team 29

Adam Cavallo - 1004881979
Andres Cervera Rozo - 1005469217
Liam Toner - 1005178754
Selin Ertan - 1004766096





1.0 Project Summary

1.1 Initial Project Description

Our team created a drink mixing device that is capable of accepting a user specified drink order from a preset menu, and autonomously creating the desired beverage. We had 4 different beverages that can be mixed into 13 different combinations of the 4 source beverages. Users can make a selection from the drink menu using the keypad on the front of the machine and press the associated key, at which point the machine employs diaphragm pumps to mix pre-defined volumes of each component beverage into a cup for the user while communicating the machine states with the user through the use of the LCD screen. The machine also features an emergency stop switch which can be flipped at any time to halt operation and return to an initial idle state where it awaits user input. The drink presets can be customized in any number of ways with only small changes to the code, provided the correct 4 source beverages are provided. For the video included in our submission, we used 4 different flavors of vitamin water.

1.2 Project Changes

Originally, we planned to collect user input utilizing speech recognition machine learning models for orders, and optical character recognition (OCR) on user IDs to ensure they were of sufficient age to order certain drinks. These technologies were to be implemented using a Raspberry Pi which would be capable of machine learning for speech recognition, GPIO for pump driving, and OCR for ID checking. Unfortunately, the Pi was not approved after our proposal, so we decided to switch to an Arduino to implement an LCD screen and keypad interface for user input collection. We originally decided to use an Arduino Uno and implement i2C communication with the LCD screen. However, the LCD screen we ended up with did not have i2C capabilities as we expected, so we used digital I/O to communicate with the screen instead. The additional digital pins required for this functionality encouraged us to switch to an Arduino Mega which is what our final design ended up using. For drink creation, we initially planned to use solenoid valves to control the flow of each liquid, however, after evaluating their cost vs. diaphragm pumps which could achieve the same goal, we decided to pivot to using pumps instead. Finally, we had initially discussed the use of Arduino push buttons for drink selection and/or emergency stop. However, after more fully considering the potential issues with mechanical bounce that we had experienced in the past, we decided to use more stable switches and the keypad. As we had hoped, we experienced no issues with mechanical bounce using these more robust manual input methods.

1.3 Decision Evaluation

After completing MIE438, we are content with our choice of microcontroller for the task we chose to complete. The Arduino Mega was a good match in terms of production (availability/cost), interfacing (number of GPIO pins, supported protocols, interrupt functionality), and speed (low speed required for this project). The only thing we might change if we were to do this project again

is ensure that the LCD screen we obtain has i2C capability as we originally planned so we can use fewer wires to operate the screen. For further discussion on future improvements, see section 3.2.

1.4 Final Design Summary

1.4.1 Power Electronics

A 5V, 15A wall power supply was used as the main source of system power. This was chosen as the supply as our team already had it on hand, so we could use it without having to purchase a dedicated power supply. The output of this power supply is the input to an adjustable boost converter which has been set to output 10V. 10V is the output because the pumps can run on anywhere from 6-12V, and the arduino takes 9-12V as its preferred voltage input when powered from an external supply, so 10V was chosen as a level that satisfies both conditions and provides a desirable flow rate from the pumps. The output of the buck converter is split into three channels: two go to pump driver circuits (L298Ns), and the last one goes to the Arduino Mega. The Arduino then outputs 5V to a breadboard which is used as a power source for all the remaining low power devices such as the keypad, and pump driver control signals. Below is a high level block diagram, illustrating the structure of the power electronics of our system.

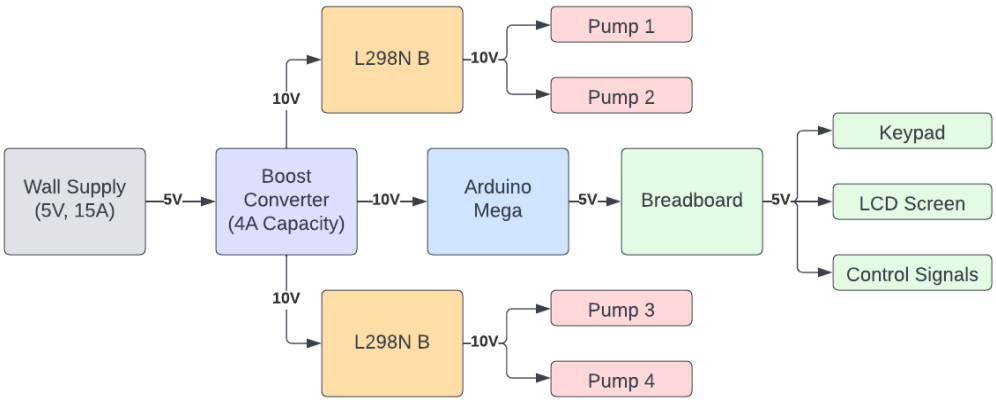


Figure 1: Power Electronics Block Diagram

1.4.2 Software

The code was partitioned into individual functions to streamline the main loop and facilitate debugging. Detailed descriptions of each user-defined function and the libraries used are available in Appendix 5.1. An overall outline of the main loop's flow is provided in the block diagram below.

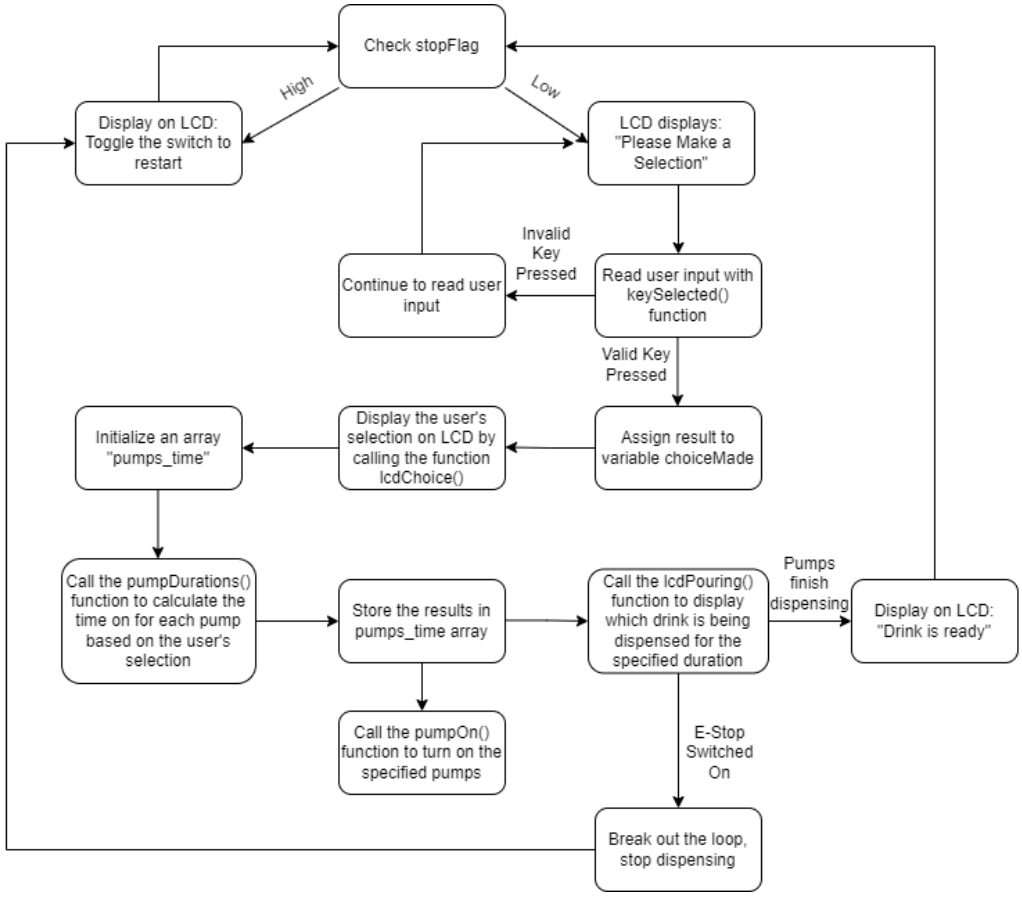


Figure 2: Flow Diagram of the Main Loop

1.4.3 Mechanical

The housing of the design was constructed using ¾ inch Pine wood sheets held together with coarse threaded M3 screws. A bottom wooden plate 42 cm x 23 cm provides a platform to hold the operator's drink cup with a 3D printed cup holder, and to support the top structure and drink containers at the rear of the housing. Located at the rear are 4x 16 oz gatorade bottles used to contain the four drink types. Several wooden dowels are used to constrain these containers during operation. The top structure of the design contains a 25cm x 23 cm container area to hold our circuit components. This area was chosen based on the overall footprint of the design, and to allow close proximity between the microcontrollers and external peripherals. At the back of this structure, 4x 12V DC pumps are mounted with silicon hoses connecting the inlets to the drink containers and the outlets to a central funnel above the operator's drink holder. The pumps are oriented towards the outside edges of the housing, exposing the positive and negative connection pins at the centerpoint of the pump layout where they can be more easily accessed. To enclose the electronics compartment, a thin plywood sheet at the top of the housing is paired with dual hinges, providing convenient access. At the front of the electronics compartment, the LCD screen, keypad, menu, and emergency stop switches are mounted with close proximity to their controlling components. At the

left side of the housing, a power supply connection piece is secured to the side sheet with a 3D-printed mounting piece. The housing components are shown in figure 3 below.

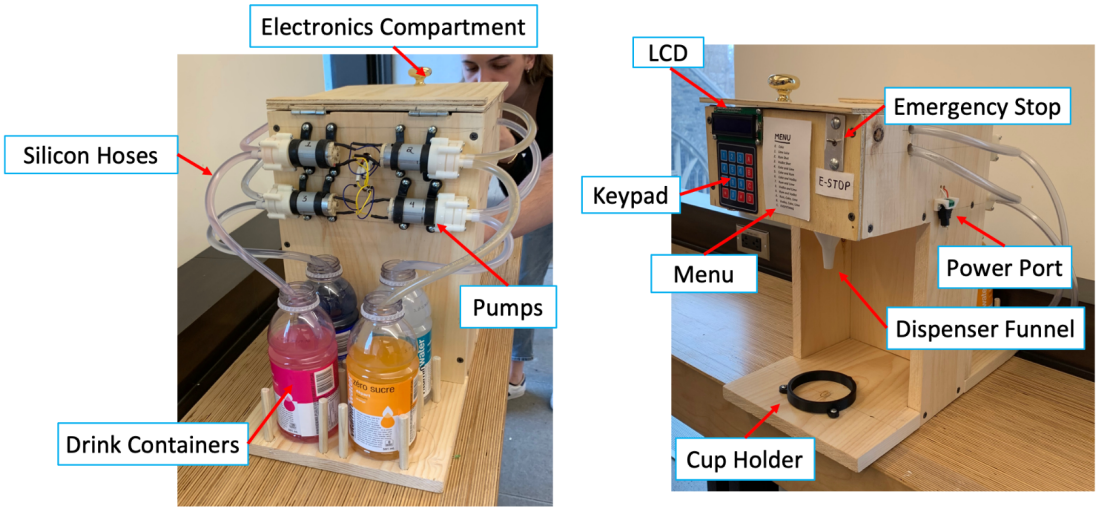


Figure 3: Housing Design Components

1.4.4 Microcontroller

The microcontroller chosen for the design was the Arduino Mega 2560 (ATMega2560). Due to the iterative nature of the design process involved, this controller was chosen to ensure that there would be no hardware limitations as the scope evolved and more digital/analog pins became necessary. The internal 16MHz crystal oscillator proved to be more than sufficient for this application due to the relatively low need for time-sensitive operations. Furthermore, there was no need for any bluetooth/WiFi connectivity, so this board met and exceeded all the project requirements for the microcontroller. The Mega is also equipped with 256 KB of flash memory, of which 11 KB were required for this application.

Many of the Mega's functionalities were not required for the purpose of this project, such as the 3 pairs of UART pins, the full 54 digital pins, and the full extent of the onboard flash memory. If an LCD screen with I2C capabilities was obtained, then many of the required digital pins would also be made available. Additionally, the motors embedded in the pumps are only ever required to turn in 1 direction, so only 1 digital signal is required per motor (given that the enable is pulled high). The resulting pins required would be an I2C channel (SDA and SCL) for the LCD screen and 4 digital output pins for the pumps, 1 for the switch, and 8 for the keypad. A more appropriate choice for microcontroller may have been the Arduino Nano Every which has much fewer unnecessary features, sufficient pins for the required application, is equipped with a 48 KB flash size, 16 MHz crystal oscillator, and is the most affordable Arduino board available ($12.50, a quarter of the price of the Arduino Mega [1]). It is preferred to stay within the Arduino family of boards due to its reliability and design for hobbyist applications.

2.0 Application of Course Material

2.1 Interrupts

The emergency stop feature of this system employed a hardware interrupt as a safety feature to stop all fluid flow at the flip of a switch. This single pole double throw (SPDT) switch toggles an interrupt-enabled pin on the arduino between HIGH and LOW. Any change in the state of this pin triggers an interrupt service routine. There is only one line of code in this routine, and it changes the state of a global flag, stored in volatile memory so it can be accessed in and outside of the main program loop. When this flag is raised, the program resets back to the initial idle state, stopping all fluid flow and affording the user an 'emergency stop' option should one ever be needed.

2.2 Code Optimization

The code has been optimized in several ways. First of all, the LCD screen display function incorporates a check to ensure that the input string is less than or equal to 32 characters, the maximum length that the LCD can display. Additionally, the user's input, obtained via the keypad, is subjected to a while loop with validity conditions. This validity scheme ensures that only valid inputs are accepted, such as those correspond to available drink options, and excludes invalid inputs such as the characters 'D', '*', or '#'.

2.3 Sub-Functions

To optimize the codebase, a series of user-defined sub-function were created that were called upon in the main loop when required. This modular approach promotes code reusability and saves time, as sub-functions can be leveraged across different sections of the program without requiring extensive rewrites. Furthermore, the use of sub-functions facilitates the debugging process by simplifying the isolation and resolution of issues.

2.4 Timers

The pumpOn() function that turns on a specific pump for a certain amount of time (see Appendix 5.1.6) utilizes a timer. The digital pin of the corresponding pump is set to HIGH for a specified duration in a while loop, which uses the millis() function to track time. The previousMillis parameter is initialized when the pump is turned on, and the time elapsed is measured by subtracting the current time from previousMillis. When elapsed time reaches the specified duration, the while loop terminates.

Using millis() or timer-based code in programming has several advantages. It allows for non-blocking code, meaning that the program can perform other tasks while waiting for a specific time to elapse. It avoids the use of delay() functions, which can cause the program to freeze.

2.5 Function Input Protection

To ensure that the input values from the user are valid, the concept of input protection was applied to the output of the keySelected() function. This was done such that if no button was pressed or if an invalid button was pressed, then the state machine would not advance to the next stage. This protected the remainder of the code from being subject to an unexpected/invalid drink choice.

For the remainder of the functions, the inputs and outputs variable types were specifically defined to be robust and adaptable to potential changes in the code. This included setting the dispense time of a pump as a float rather than an int, to avoid any potential inaccuracies in the calculations of these times. For variables where there may only be integers within a small range, these were established to be uint8_t data types to minimize the required memory to be allocated. This was the case for variables such as *pump* where the potential values were integers from 1-4.

3.0 Testing and Future Improvements

3.1 Testing

To test our design, our team initially split the code into several individual files which would be connected to the main file through the inclusion of header files. This allowed for both an organized coding structure, and an easy way to test individual functions. The pumps were individually connected to a separate circuit composed of the microcontroller, power supply, drivers and boost convertors. Using a multimeter, we were able to adjust the boost convertor output and determine the optimal input voltage to the pumps. We started with lower voltages, iterating and measuring the resulting flow rates from the pumps until an appropriate speed was achieved. This allowed us to update the duration that each pump needed to run at in order to produce the correct drink portions. Simultaneously, we tested the LCD and keypad functionality in a separate circuit to ensure user inputs could be accurately registered and displayed on the LCD. Once satisfied, we began to combine all the sub components onto a single arduino mega and update the pin numbers. Using the multimeter, we were able to ensure that connections were correct by measuring the voltage readings at critical locations in our circuit.

One issue that occurred was with regards to custom drink selections. Originally, we planned on having functionality to allow the user to pour custom drinks and proportions by holding buttons for a period of time on the keypad. However, we found that the pumps could not be accurately controlled due to delays between user input registrations and pump outputs. Often, this would lead to undesired pump actuations after the buttons were released, adding the risk of overflowing the drink cup. Thus, the team decided to remove this feature and constrain the user inputs to several different preset options that could be more easily controlled.

3.2 Future Improvements

3.2.1 Functionality Improvements

Several design changes can be implemented in the future to expand the functionality of our drink mixer. For instance, we can increase the number of options available to users by allowing for a custom drink dispensing mode. This could be achieved by adding a more reliable input method such as a toggle switch that would dispense the desired drink while in its 'On' position. This would eliminate input delays caused by keypad usage. In addition, further user input can be collected to dispense different drink sizes - Large, medium or small.

To create a better user experience, we could also add voice activation as an option to replace the manual keypad. This can be done through the use of a raspberry PI with a usb microphone device. Due to redundancies from having two microcontrollers, this option was omitted.

Aside from direct functionality, to improve machine maintenance we could implement a 'cleaning mode'. This mode could instruct the user through the LCD to connect water supplies to each of the pumps and flush the tubes to clear excess residue from prior use. This could be useful when switching to different drink types to avoid contamination.

3.1.2 Optimization Improvements

To optimize our design and reduce memory, several changes can be made to our code structure and variable assignments. First, we could re-evaluate our variable size assignments. Some variables do not change, or may not require an 'int' designation of 16 bits. Rather, they can be defined as a 'byte' variable and use values ranging from 0-255 (8 bits). For example the function 'int keySelected()' will currently not exceed an 8 bit value as it is constrained to numbers ranging from -10 to 12. However, future user inputs may include larger number combinations that could exceed 8 bits, and thus it was designated as type 'int'. There is risk of overflow associated with variable sizing limitations, and so many variables were kept as integers. However, with additional functionality in the future, additional space may be needed and can be achieved through these variable size re-assignments to 'byte' types..

Furthermore, we implemented a for-loop to operate each of the 4 pumps based on the user's input. To decrease the computation time for this loop and reduce memory, the loop can be unraveled. Instead of 4 loop iterations, the pump code can be called in a consecutive manner line by line without needing to constantly check the loop conditions. This structure was omitted in favor of a more organized, concise, and readable program.

3.1.3 Control Improvements

Several control loops can be implemented with our design in conjunction with additional devices. For instance, we currently have no control for the flow rate of the pumps. This means that variances in pressures, fluid viscosity, residue buildup/flow resistance, and electrical noise can all affect the

flow rate. By implementing a flow rate sensor we could continuously monitor and adjust to the desired flow rate to ensure the correct volume of fluid is provided to the user. Currently, fluid is dispensed based on a tested flow rate value that we implement into the code as a global variable. By doing this, we make the assumption that the flow rate will not vary by large amounts from this value. However, having a control loop can account for inconsistencies in flow rate and respond by adjusting the voltage to the pumps.

Additionally, there is currently no control for the drink container volume levels. If a container is empty, our program will still attempt to dispense the fluid. This can be resolved by implementing a control loop with a water level sensor, that can either interrupt the program when a certain minimum level is reached, or reject user inputs if there is insufficient fluid volume.

4.0 References

[1] “Arduino Nano every,” *Arduino Online Shop*. [Online]. Available: https://store-usa.arduino.cc/products/arduino-nano-every. [Accessed: 12-Apr-2023].

5.0 Appendix

5.1 User Defined Functions

5.1.1 Int keySelected()

This function is designed to read user input from a custom keypad and return a numerical value corresponding to the key pressed. Each character corresponds to a drink option. A Positive numerical return value indicates valid key pressed while a negative numerical return value indicates an invalid key pressed.

5.1.2 Void lcdPrint(String message)

This function takes a string message as input and displays it on a two-line LCD screen. If the length of the message is greater than 16 characters, the function searches for the last space before the 16th character and splits the message into two lines at that point. If a space is found, the first line is displayed on the top row of the LCD and the second line is displayed on the bottom row. If a space is not found, the message is displayed on the top row of the LCD.

5.1.3 Void lcdChoice(int choice)

This function takes an integer choice as input, which represents a drink selection, and displays the corresponding drink name on the LCD screen using the aforementioned lcdPrint() function. The

drink name is retrieved from the drink_names array based on the index indicated by the choice integer.

5.1.4 Void lcdPouring(int pump_number)

This function displays the message “Pouring” on the first line of the LCD screen and displays the name of the corresponding drink being poured on the second line based on the value of the pump_number variable.

5.1.5 Void pumpDurations(int choice, float durations[])

This function takes a drink choice and an array of float values as input. It assigns the float values corresponding to drink choice from a 2D array to the elements of the input array. The input array is then modified and is used to store the duration of time for which each pump should run when making the drink.

5.1.6 Void pumpOn(int pump_number, float duration[])

This function turns on a specific pump (based on the pump_number parameter) by setting the corresponding pin to HIGH, and then starts a timer for the duration of the pour (specified in the duration parameter). It then loops until the timer has expired, checking the time interval with the current time using the millis() function. Once the timer has expired, it turns off the pump by setting the corresponding pin to LOW.