D*rink*W*izard:* Automated Drink Dispensing System

Michael Amaral, John Brushwood, Reginald Fergerson, and Zachary Kirby

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450, USA

***Abstract* — The objective of this project is to design and build an automated drink dispensing system. This system will utilize our groups electrical and computer engineering background to design an Android application that will control and dispense the user's choice of adult beverage. The group chose this idea for our senior design project because we felt we could design an automated system at a lower price point than the current solutions available on the market.**

***Index Terms* — Android application, Bluetooth, breathalyzer, LCD screen, microswitches, peristaltic pump, and ultrasonic sensor.**

I. Introduction

The American population consumes alcohol, as a whole, in vast quantities. The consumption of alcohol can survive through a depression, good times, bad times, and it can even prosper through a prohibition. Needless to say, Americans love their alcoholic beverages. We created the D*rink*W*izard* to capitalize on the love the American population has for drinking.

To give some insight on how much alcohol the American population consumes, let's look at some statistical data. Thirty percent of the population does not drink at all, another thirty percent drink, on average, less than one drink per week. On the other hand, the top ten percent of American drinkers, about twenty-four million, consume on average about seventy-four drinks per week, or a little more than ten drinks per day. [1]

The D*rink*W*izard* is not the first drink-dispensing machine available on the market. There are already several systems available to the consumer. Some of these systems include: Smartender ($25,000), Monsieur ($2,699), and Bar2D2 ($2,000 & DIY). All of these have a price point above $2,000 and will be out of the price range of most Americans.

Our goal with the D*rink*W*izard* is to design the system at a price point lower than the current competitors. If we can get the price point down to $1,000 dollars per unit, we become very competitive in the current market. If we can capture one percent of the twenty-four million Americans that heavily drink, we will have 240,000 customers. At a price point of $1,000 per unit this leaves us at $240 million gross, which is our major driving factor for this project.

Some of the key features of the D*rink*W*izard* are that we have a nine-bottle capacity, four mixers and five alcohols. We will be running an Android based tablet and phone application to allow the customer to order drinks while within Bluetooth range. This application will allow the customer to create custom drinks based on their taste preference as well as select from over thirty classic drink choices. A Breathalyzer will be incorporated to aid the user in determining their intoxication level. The D*rink*W*izard* team wants the customer to have a dependable experience where their drink of choice is made correctly and consistently every time. We have incorporated peristaltic pumps to achieve this accuracy. We also do not want wasted product so ultrasonic sensors are incorporated to determine when a cup is present and when the cup is full of the correctly made beverage.

All of the funding for the D*rink*W*izard* came from the four group members. For this reason we wanted to keep the cost as low as possible. We set a price point of $500 as our goal that also allows us to achieve a greater net profit once we bring the D*rink*W*izard* to market.

II. D*RINK*W*IZARD* Profile

A. Vending Unit Design

The D*rink*W*izard* is a vending unit based off of a soda-dispensing machine. The vending unit is a rectangular unit containing two drawers. The bottom drawer will house space for the battery storage as well as contain storage for whatever the customer would like to store there. Our intention was for it to be used to store ice.

B. Liquid Dispensing

The upper drawer will contain the nine containers, four of which will contain the mixers and five will contain the alcohols. The liquid in the containers is fed up through two different diameters of tubing using nine peristaltic pumps. We used two different pumps: one set for the mixers and the other for the alcohols. These nine tubes are fed to a hockey puck that has nine holes for the tubes, which is located right above where the cup is to be placed.

C. Sensors

Before dispensing any liquid from the nine containers, an ultrasonic sensor will determine if a cup is present. Once a cup is present the liquid will dispense, and a second ultrasonic sensor will ensure that the cup is not overfilled.

D. LCD and Microswitches

Also incorporated into the D*rink*W*izard,* is an LCD display to inform the user when one of the containers is empty. In order to detect that a container is empty, nine microswitches will be used. Once the microswitch is triggered it will turn on an LED beneath the bottle as well as trigger the LCD display to inform the customer that a bottle is out of liquid. With power turned on and all nine containers filled, users will enjoy a delicious adult beverage made the same way every time.

III. HEART OF THE D*RINK*W*IZARD*: MSP430

A. Selection of microcontroller

During the design of this vending unit several microcontrollers were evaluated before a final decision was made. The first microcontroller our group reviewed was the Atmel Atmega16, which is a very popular and versatile microcontroller. The Atmega16 has 32 I/O pins, 16 KB flash memory, 1024 bytes of ram and a current draw of 1.1 mA. The Atmega16 meets all the needs of this project, however this chipset was the most expensive out of the ones we reviewed. Since the cost is of concern, this microcontroller was ruled out.

The second microcontroller reviewed was the Microchip Pic24FJ16MC101. The Pic24 was very similar to the Atmega16; it has a 16-bit architecture, 15 I/O pins, 1024 bytes ram, 16 KB flash memory, and 1 mA current draw. An interesting feature of the Pic24, which was considered for future use, is the chipset feature to support capacitive touch sensing. The Pic24 also has three comparators on board and each can have up to four inputs that are a great advantage for comparing the liquid levels of our containers. The Pic24 is an ideal microcontroller for this project and is the cheapest microcontroller reviewed. However, due to the unfamiliarity of this chipset and the additional cost of purchasing a programmer this microcontroller was ruled out.

The last microcontroller reviewed was the MSP430G2553. The MSP430 has the lowest instruction count, coming in at 51 instructions. This makes coding up the microcontroller, for this project, much easier. The MSP430 has 16 I/O pins, 512 bytes, 16 KB flash memory, and 230 µA current draw. The familiarity with this microcontroller is ultimately why it was selected.

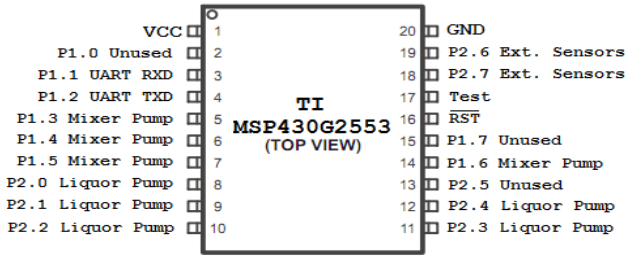
B. MSP430

The MSP430 was ultimately chosen for its familiarity, however the MSP430 has a few drawbacks. One of the drawbacks of the MSP430 is that the maximum voltage that the microcontroller can safely handle at an I/O pin is Vcc to 0.3 V. This becomes a problem due to the fact that our sensors output a maximum voltage of 5 V. This will require additional hardware to interface these sensors with the MSP430. However, the additional hardware required was as easy as a simple voltage divider circuit. The voltage divider circuit will reduce the input voltage to the MSP430 to a manageable voltage level of around 2.5 V. The second drawback of the MSP430 is that it contains the least amount of RAM and using a lot of registers could present a huge problem. Luckily, this negative characteristic will not be an issue with this project.

The *DrinkWizard* will use three MSP430s. The first MSP430(1) is to control the peristaltic pumps and Bluetooth communication, the second MSP430(2) will control the microswitches and LCD display, and the third MSP430(3) will control the ultrasonic sensors. Since all three of the MSP430s were already owned by each one of the group members, it will not add to the cost of the project. If we were to purchase the MSP430, we could purchase three MSP430s for about the same cost as one Atmega16.

The first MSP430(1) will use the 16 KB of built in storage to hold the code for all of the different drink recipes. The 16 KB is more than enough storage to hold the recipe list. The MSP430(1) will use nine of the 16 I/O pins to turn on the individual liquid pumps. The I/O Pins 1.3 to 1.6 are used for the four mixer bottle pumps; the I/O pins 2.0 to 2.4 are used for the five liquor pumps. Pins 1.1 and 1.2 are used for UART communication. The UART communication will communicate via Bluetooth with the Android device. The MSP430(1) will use pins 2.6 and 2.7 for the external sensor circuits. The external sensor circuit communication is for liquid level sensors and cup sensors. The pins can be seen below in Figure 1.

Fig. 1. MSP430(1) Pin Assignment



The second MSP430(2) is monitoring the nine microswitches using nine of the sixteen I/O pins. When the MSP430(2) receives a 5-volt high signal on one of the

nine I/O pins, indicating that one of the liquid containers is empty, this will trigger an LED to illuminate under the bottle that triggered the 5-volt high signal. This 5-volt high signal will also trigger the second MSP430(2) to tell the first MSP430(1), the one controlling the pumps, to stop all pumps and production of the current drink. At the time of a triggered event the MSP430(2) will also signal the LCD to display to the user that one of the bottles is empty.

The third MSP430(3) is monitoring the three ultrasonic sensors. The MSP430(3) will detect the duration of the logic high-level on a general purpose I/O pin. The MSP430(3) will monitor this high-level I/O pin for a duration of time for the x-axis. If this time interval is within the tolerance of the D*rink*W*izard,* the MSP430(3) will then move on to the y-axis and then finally the z-axis. This process will cycle and continue to cycle until the MSP430(3) detects the proper tolerance on all three axes. Once all three axes are within tolerance, the MSP430(3) will use another general purpose I/O pin to send a high-level one out to the MSP430(1). This will signal to turn on the pumps.

The market had other solutions available that could have incorporated all the functions of the three MSP430s into one board. However, we chose not to do that for several reasons. The first reason is the cost; we had three free MSP430s and there was no need to add additional cost. The second reason we chose to use three MSP430s is that we wanted a MSP430 for each of the different subsystems. Having an individual MSP430 for each subsystem allowed the troubleshooting of any issues to be completed in a more efficient manor. Also the use of multiple microcontrollers allowed us to expand the capabilities of the D*rink*W*izard* down the road.

IV. LCD, Pumps, Sensors, and Switches

A. LCD

The D*rink*W*izard* uses a New Haven 4x20 LCD serial display. The LCD is used by the DrinkWizard to inform the user that all bottle levels are good, “All bottles good” or it will inform the user that bottle levels are low, “Check bottle level”.

The MSP430(2) is monitoring the nine microswitches. When the MSP430(2) receives a high-level signal, VCC 5+ volts, on one of its I/O pins this will trigger the UART to the ASCII code for “Check bottle level” to the LCD controller. If the MSP430(2) does not detect any high-level signals on the I/O pins then it will signal the LCD controller to display the “All bottles good” on the screen.

These messages will stay on the screen until an event occurs. One possible event would be that one of the bottles is empty, so the screen changes. Once this bottle is filled up, as long as no other bottles are empty, the screen will update to “All bottles good”. This updating process will continue until the wall power cord is disconnected and the power switch is switched off.

B. Peristaltic Pump

The D*rink*W*izard* uses two different types of peristaltic pumps. Peristaltic pumps were selected because of their accuracy. The pumps turn and pinch the tube; the rotation and amount of liquid released will be the same every time. The same amount of liquid is dispensed every time, so to perform our recipes we will determine the length of time each pump needs to be on to exactly make these recipes.

The two different peristaltic pumps will both operate at 12 volts. The reason for two different types of pumps is that one type is for the liquors and the other type is for the mixers. The main difference between the two types is the rate at which they pump. The pumps for the mixers can pump at a rate of 500-mL per minute and the pumps for the liquors can pump at a rate of 100-mL per minute. The majority of recipes for mixed drink have more of one mixer than actual liquor, by having a faster pump on the mixers we can reduce the time it takes to make the cocktails.

The first time the pump is used, or after a line is cleared, the pump will need to be primed. Nine push button switches will be installed to individually prime each pump. Pushing the priming button will turn on the pump which then begins rotating and pinching the tube. This is in effect the same thing as putting your finger over your straw and drawing liquid out of a cup. Once primed the pump will never loose pressure unless there is a hole in the tubing.

Additional tubing is needed to bring the liquid up from the bottle storage area to the individual pumps. The tubing used for the liquor bottles is Tygon E-3603 which is approved for food and beverage use. The diameter for this tubing is 3/32 for the inner diameter and the outer diameter is 5/32. The tubing for the mixer pumps is Watts’s 3/8 inch clear vinyl tubing. The inner diameter is ¼ inch and 3/8 inch is the outer diameter. The individual tubing will also run from the pumps to the dispensing nozzle.

C. Breathalyzer Sensor

The D*rink*W*izard* uses Micro4you Studio MQ-3 ethanol sensor to monitor users intoxication level. The ethanol sensor interfaces with the android tablet through the tablets USB port. Two of the pins receive power and

ground from the tablet, and the other two pins were analog out and digital out. The VCC is 5+ volts and the analog out is 0 to 3 volts.

The digital out has an adjustable potentiometer that would send out a digital high based on the sensitivity of the sensor. The D*rink*W*izard* takes advantage of the digital out. The ethanol sensor is calibrated to send out a digital high when a reading is equivalent to a .08 intoxication level. This calibration is determined by adjusting the potentiometer as ethanol is blown across the sensor. A Breathalyzer that can be purchased at any electronic store was used to confirm our calibration specs of .08.

Once the android tablet receives the digital high, a window pops up in the D*rink*W*izard* app to inform the user that they are at an intoxication level that is not safe to drive. They are then required to check a box saying they are going against the recommendations to stop drinking. After checking the box the responsibility is put on them.

D. Ultrasonic Sensors

The D*rink*W*izard* uses three MicroPic HC-SR04 ultrasonic sensors. All three sensors are used to detect when a cup is present and one of the sensors is also used to detect if the cup is full of liquid. There are four pins on the sensor; two are VCC 5+ volts and ground. The last two are trigger pulse input and echo pulse output. The ultrasonic ranging module receives I/O trigger for at least 10 mS on the trigger pulse input. Then the module automatically sends 8 40 kHz pulses and detects if there is a pulse signal back.

If there is a received pulse signal, the echo pulse output terminal goes high for an interval of time. This interval of time is from when the pulse was sent to the time when the pulse was received. This interval of time can be used to determine the distance the object is from the ultrasonic module. The distance is determined by the time duration of the high-level times the velocity of sound (340 meters/second) divided by 2. This will give the distance in meters.

If the distance is within the D*rink*W*izard's* tolerance for a cup present, the other sensor will then detect if the cup is already full. If this ultrasonic sensor is within the tolerance of the D*rink*W*izard* the MSP430(3) will send out a high level 1 to the MSP430(1) to signal the peristaltic pumps to turn on. If one of the ultrasonic sensors goes out of tolerance the MSP430(3) turns off the high level 1 out and this signals the MSP430(1) to shut off the peristaltic pumps.

E. Microswitches

The microswitch used for the D*rink*W*izard* was the Mulon M8 Precision switch, microswitch. This microswitch contains three pins, N.C (normally closed), N.O (normally open), and ground. Conventionally you would use ground on the ground pin and then when the switch is pressed down the ground connects to the normally open pin. Our needs needed this to be wired up differently.

The MSP430(2) is looking for a 5-volt high signal, so for our needs we connected a 5-volt high signal to the ground pin of the microswitch. The microswitch at rest has the normally closed pin connected to the ground pin; In our project we are using the microswitch to monitor if a container is full or empty so normally closed for our scenario would be a full bottle and the microswitch lever pressed down.

The lever pressed down has the ground pin connected to normally open pin. For our project, the bottle is empty which then puts the microswitch in a normal state and has the normally closed pin connected to the ground pin, which is our 5-volt high signal. This then triggers the MSP430(2) to send out its appropriate signals and the LED under the empty bottle to illuminate.

V. D*RINK*W*IZARD* ANDROID APPLICATION

The Android application has three major activities in which the user can interact with. They are labeled Main Activity, Order Activity, and Custom Drink Activity. These three govern the entire application and make simplify the processes of connecting to Bluetooth, ordering a drink, and creating a custom drink. There is another activity called Paired List Activity but it is only used for pairing with devices, breaking bonds with devices, and connecting to bonded devices and shouldn't be accessed as often. The user interface was designed with the layout editor in Android Studio. The following diagram outlines how the three major activities interact with each other.

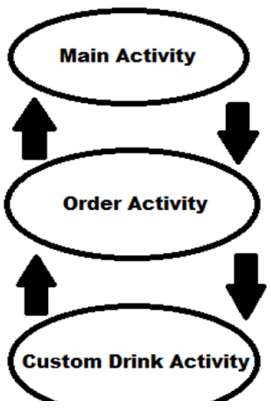


Fig. 2. Three Activities of the Application

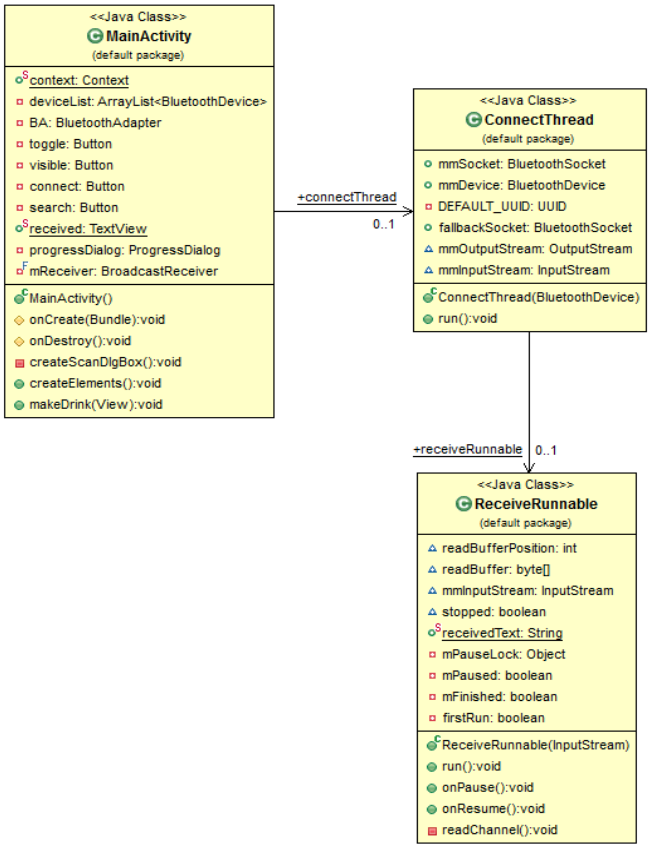
Main Activity handles administrative activities such as connecting to Bluetooth and building the database of drinks. It is able to turn Bluetooth on and off, pair with a visible device, show the currently connected devices, and take the user to Order Activity. If Bluetooth is turned off when the user attempts to show the connected devices, the application will alert the user that Bluetooth is not enabled and present the option to turn it on.

Order Activity provides the major functionality the Drink Wizard application was designed for. It is familiar, intuitive and allows the user to order a drink. It also updates the user when their drink is being created and when it is finished. It will also have the option to go back to Main Activity in case an administrative action needs to be performed. This may happen if the Android device loses the connection to the mixer for any reason. Another point about Order Activity is that it keeps a private instance of the Drink List class, which has an Array List of objects, which hold the information about the ingredients, which compose a drink. This ensures that the Drink Wizard Android application will be able to store over two billion drink combinations. The number of combinations that can be created defines a more practical limit. Currently the application will only allow combinations of ingredients, which will create an 8oz drink.

Finally, Custom Drink Activity is used to create new drinks. The user is allowed to add ingredients to a beverage of their choice. Sliders are displayed and can be moved back and forth to add or remove a liquid. Should the user craft a drink that the developer's feel is not safe, a warning will be displayed, which will tell the user why it is unsafe and ask if they wish to proceed. The Drink Wizard application will never disallow a drink from being made and the responsibility is left to the user to continue creation or go back and edit the drink. This activity will also allow the user to return to Order Activity without creating a drink, completing the cyclical structure of the application.

A. Bluetooth Functionality

Android requires permissions to keep applications from accessing unnecessary features on the device. When the application installed, the user is shown a list of the included permissions. This serves as a security measure and keeps applications from gaining access to personal information. Bluetooth also requires permissions. There are two permissions, which must be written into the Android Manifest called Bluetooth and Bluetooth Admin. The Bluetooth permission is needed for all Bluetooth actions such as data transmission and connection management. Bluetooth Admin handles administrative capabilities such as allowing the device to be discovered and discovering devices in the area. Without these two permissions, the application would not be able to communicate through with the D*rink*W*izard* mixer and would likely crash.



B. Communication from the Android Perspective

Fig. 3. Bluetooth Communication Class Diagram

Threads are used for connecting and sending and receiving data via Bluetooth. Connect Thread is the link between the application and the serial port through which data travels. When the thread is started, the application attempts to create a radio frequency communication socket with the device the user requests to connect to. To do this, a Universal Unique Identifier (UUID) is used. For safety, a fallback socket is also created. A connection is established with this socket only if an exception is thrown by when the application attempts to connect to a remote device. Without the fallback socket, if a connection is not made, the application cannot communicate with the mixer. The following class diagram shows the relationship between Main Activity and the threads for sending and receiving data via a Bluetooth connection.

After a successful connection has been made, the application will begin listening for data from the mixer. Constantly listening will cause the central processing unit (CPU) usage to remain at relatively high levels for the duration of the application's run time. To remedy this, the device will only begin listening to the channel once an order for a drink has been submitted. This is possible because the mixer should not be transmitting any data to the application when it is idle (when no drinks are being made). A class, which extends Runnable called Receive Runnable, handles the incoming data from the mixer. Upon running this class, a method called readChannel() is called. This method attempts to get the length (the number of the bytes) of the incoming data then iterate through the bytes one by one and add each one to an array, which will later be turned into a human-readable string. If a delimiter is found, the bytes are copied from the buffer they were being appended to and decoded to a readable string. The data is now ready to shown to the user.

A class handles sending data, which is also a child of Runnable. When the thread is started the data needed to make a beverage will be prepared. Once it is ready to be interpreted by the MCU, it will be sent using the open socket's output stream. This completes the Bluetooth-related functionality of the Drink Wizard Android Application. The application is now connected to the mixer and data to make a drink can be sent to the mixer. Likewise, information can also be received by the Dink Wizard application.

C. Ordering from the Android Perspective

The Drink List class contains an Array List object that holds all the drinks in the application. A getter method for the list of drinks has been included for convenience and so has a method to get an individual drink by name. The most used method in this class is the addDrink() method which takes in all the necessary information for a drink, creates it in the application, and adds it to the master list. The addDrink() method creates Drink objects. This class Drink holds all the information about a given drink that is sent to the mixer when an order is placed. It includes how much of each ingredient goes into the drink. Possible ingredients are vodka, tequila, rum, whiskey, peach schnapps, orange juice, cranberry juice, pineapple juice, and sour mix. For convenience a method has also been included to get the index of a drink. Getter methods for how much of each ingredient goes into a drink have also been included. Setters are also in the class and are used by the application's database. The database is used for storing all custom drinks so they are saved when the application is stopped.

D. Storing Custom Drinks

Without a database, custom drinks would immediately be deleted whenever the device was shut off or even when the application was closed. Because new data can be actively generated at almost any point in Order Activity's lifetime, developers decided to either write new drink data to a file or keep a dedicated database for the data. Instead of keeping track of a file, which could easily be deleted by going through the file system of the device, it was decided that a true database should be used. For this reason, an Android SQLite database is used to keep track of all custom drinks.

VI. Power Supply

The D*rink*W*izard* will connect to a normal 15 amp duplex receptacle, three prong 120-volt outlet. The wall connection will run through a 6.4-amp 12-volt step down transformer. After the step down transformer, the power will run through a bridge rectifier capable of handling 12 amps. The rectification stage is a full wave rectifier so it will need to be smoothed out, so the power will run through two 6800-μF capacitors capable of voltages up to 63 volts. These capacitors will smooth out the final output to 12-volt DC. From the 12-volt DC input the voltage will be stepped down to 5-volts DC and 3.3-volts DC.

A. 3.3-volt Load

The lowest step-down from the 12-volt input is the 3.3-volt DC load. This 3.3-volt load will be used to power up the microprocessor's as well as the Bluetooth module. To step-down the voltage from the 5-volt supply the Texas Instruments LM21215, 15-amp high efficiency point of load synchronous buck regulator, is used. The LM21215 has an adjustable output voltage from 0.6-volts to Vin. The input voltage or Vin can range from 2.95-volts to 5.5 volts. The voltage can be adjusted by using a simple

voltage divider circuit with the middle node connected to the Vfb, which is pin 19 of the HTSSOP-20 package.

B. 5-volt Load

The second to lowest step-down from the 12-volt input is the 5-volt DC load. This 5-volt load will be used to power the cup sensors, liquid level sensors, and LCD module while also feeding the 3.3-volt load. For the D*rink*W*izard* to step-down the voltage from the 12-volt DC input the Texas Instruments LM22678-ADJ, 5-amp SIMPLE SWITCHER® step-down voltage regulator is used. The LM22678 has a wide voltage input range, 4.5 to 42-volts. The LM22678 has an adjustable output voltage with outputs as low as 1.285 volts. The voltage can be adjusted by using a simple voltage divider circuit with the middle node connected to the Vfb, which is pin 6 of the TO-263 package.

C. 12-volt Load

The last voltage needed is 12-volts DC to power the peristaltic pumps used to dispense the liquid from the nine bottles. The D*rink*W*izard* is using a 12-volt lead acid battery to power the D*rink*W*izard* when an A/C output is not available, such as when the user is at a tailgating event. The 12-volt lead acid battery will always be connected in parallel with the system so this will be used to regulate the 12-volt supply needed for the peristaltic pumps.

D. 12-volt Battery

The D*rink*W*izard* is designed to run off a 12-volt, 7 Ah, lead acid battery. The max current draw is 2.5 amps with everything running at once. We could run the pumps straight for about 2.5 hours before we would have to recharge the battery. The only reason everything would be on at the same time for 2.5 hours would be under a stress test condition, so this was ruled out for normal operation. A single lead acid battery was used to save cost as well as weight. The lead acid battery was the battery of choice for the low initial cost, low maintenance cost and the size and weight were perfect for our application. The only disadvantage for the lead acid battery is the long charging times.

To counteract the long charging times of the lead acid battery, the power supply circuit was designed to continuously charge the battery while plugged into the 120-volt wall outlet.

E. Battery Charging Circuit

To properly charge a lead acid battery we need to monitor overcharge and undercharge. One conventional way to charge the lead acid battery is to use a current-limited power supply circuit that maintains a constant voltage across the battery, on average 2.4-volts per cell. Once the charging current drops below a current tolerance, defined by the capacity of the battery, the battery charger is placed in a trickle-charge mode. The proper charging voltage is a give and take between cell lives versus charging time. While we could charge at high voltage this minimizes the time required, however at a full charge this produces a large overcharging current. This overcharging current shortens the battery's life. Lowering the current via decreasing the charging voltage can save the battery's life. This decreasing of the charging voltage lengthens the time it takes to charge the battery back up.

To get the best of both worlds we can charge the battery at high voltage until the current drops to .12 amps or below this point. At that time the voltage can be lowered to maintain a low trickle charge current of about 120 mA and this trickle-charge will continue until the current rises back above the .12 amps.

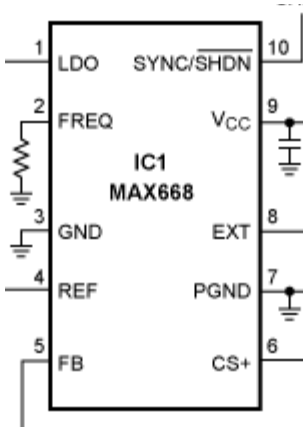
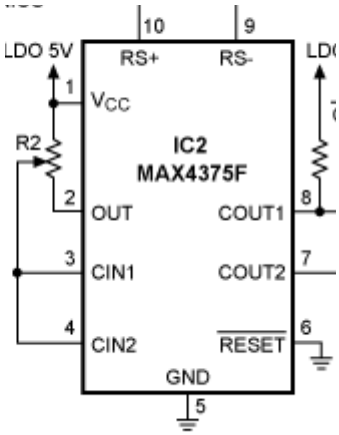


Fig. 4. MAXIM MAX668

To get to the proper voltage to charge the lead acid battery a boost converter, Maxim MAX668 Step-Up Controller, applies a constant voltage of nominal 15.4-volts to the lead acid battery. Once the lead acid battery is fully charged the voltage is lowered to about 13.4-volts to maintain a trickle charge. We decided to use a flyback transformer instead of an inductor to isolate the battery away from Vin. This will allow Vin to range above and below the charging voltage.



. MAXIM MAX4375F

The, Maxim MAX4375F will do monitoring of the battery charging current. The, MAX4375F, measures the current by gernerating a proportional voltage at the OUT terminal, pin 2. The drop across our output resistor produces a voltage at pins 3 and 4. At the point that the charging current drops below .12 amps, the voltage crosses the internal comparator threshold and drives COUT1, pin 8, low and sets COUT2, pin 7, to high impedance. Disconnecting COUT2, pin 7, the feedback level is shifted and thus changes the charging voltage to roughly 13.4-volts. Our maximum available charging current depends on our voltage, Vin, and our current-sense resistor R1. This will allow us to charge our lead acid battery continuously without overcharging the lead acid battery, which ultimately will allow us a greater life span of our battery.

VII. Conclusion

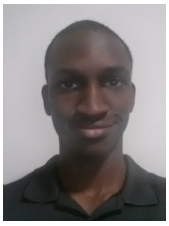
All of the aforementioned systems comprise a collection of components needed for the design and implementation of the D*rink*W*izard* vending unit. This reading covers multiple components and subsystems that when in synchronization allows the flawless execution of the D*rink*W*izard* and the user or users can have the ultimate experience and enjoyment from our system. We look forward to seeing a D*rink*W*izard* in your home or at a tailgating event soon.

Acknowledgement

Michael Amaral, a senior student of the electrical engineering department at the University of Central Florida Florida. Michael is currently an employee at Mtron PTI, specializing in RF filter design. Michael plans on continuing his employment at Mtron PTI while pursing and developing his career in the electrical engineering field.

John Brushwood is a senior student of the electrical engineering department at the University of Central Florida. John is looking forward to pursing a career in the electrical engineering profession. John would like to specialize in the RF filter design field.

Reginald Fergerson is a senior student of the computer engineering department at the University of Central Florida. Reginald is looking forward to pursing a career in the computer engineering profession. Reginald would like to specialize in the defense contract sector.



Zachary Kirby is a senior student of the computer engineering department at the University of Central Florida. Zachary is currently a Master certified car audio installer. Zachary looks forward to pursing his career in the computer engineering profession and specializing in the car audio sector.

References

[1] Ingraham, Christopher. “Think you Drink a Lot? This Chart Will Tell You.” Web log post. Washingtonpost. N.p., 25 Sept. 2014. Web. 30 May 2015. http://www.washingtonpost.com/blogs/wonkblog/wp/2014/09/25/think-you-drink-a-lot-this-chart-will-tell-you/