The Beer Grid

Edgar Alastre, Jonathan Chang, Colton Myers, and Ashish Naik

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

***Abstract* — This project consists of an electronic table filled with RGB LEDs and infrared sensors that work together to create a display on the surface of the table that is compatible with games of "Beer Pong". The table makes use of several TLC5940 LED drivers to control about 150 LEDs and multiple SN74HC165 Shift Registers to receive information from 100 infrared sensors. This is combined with dynamic memory allocation techniques to make "LED Nodes" to manage the information of each LED for programming patterns on the table. Meanwhile, the table has Bluetooth connectivity to update game information on player's phones.**

***Index Terms* — SN74HC165 Shift Registers, TLC5940 LED Drivers, RGB LEDs, Dynamic Memory Allocation, Bluetooth Communication.**

I. INTRODUCTION

"The Beer Grid" is an electronic table that makes use of a system of 100 RGB LEDS and an equal number of 100 infrared sensors to create a display on the top surface of the table which can display patterns and respond to surface impacts during a game of "Beer Pong". This information is then processed and sent to an app, which can be downloaded onto a player's smartphone, to help them keep track of the information of each game. Bluetooth connectivity is used for transferring information to and from the table, and all of the circuitry for this design is housed inside of a wooden framework.

The software of our design involves using a system of "LED Nodes" to hold the current state information of each of the individual LEDs on the surface of our table. The table's program updates the information on each of these "nodes" as the games progress and various patterns are used on the display. This way patterns can be programmed in the table based on this information and easily manage all 100 of the LEDs to respond to each of their corresponding sensors. These same techniques will also be used to manage a smaller system which makes spots on the table light up when cups are placed on them for play. It will help with keeping track of when someone scores a point because the cups will be removed when this happens.

Meanwhile, changes in the table's display are tracked by the program in response to in-game occurrences. This lets players know when one team or the other has scored points and how many turns have passed since the beginning of play. This information is sent to a database and that database is referenced by the app we've built to let people know how the game is proceeding.

II. COMPONENTS

A. Atmel SAM3X8E ARM Cortex-M3 CPU

The microcontroller is the part controls most of the subsystems used in The Beer Grid. Out of all the microcontrollers considered the Atmel SAM3X8E was the preferred choice given its versatility and the available resources. The Atmel SAM3X8E is responsible for sending data to the Cup Display System, Sensor Array, and RGB LED powered by the LED Drivers, as well as sending Bluetooth information about the game to a smartphone device via a Bluetooth adapter. The project required a device that was more powerful than an 8-bit 16Kbytes as the amount of information handled was extensive and as research shown, similar projects required to move to a more powerful 32-bit processor. In the case of the SAM3X8E it features 512kB of memory and runs on 84MHz as well as its architecture being 32-bits. Additionally, several libraries were available online of the other components used in the assembly of The Beer Grid therefore it was convenient for the team to make this the microcontroller of choice. The Beer Grid utilizes several communication protocols such as SPI, serial, and UART to communicate with every subsystem. The LED Drivers required an SPI port as well as the shift registers which utilized a more basic serial protocol that still required a clock pin and a "serial in/out" pin in order to fully communicate. Bluetooth on the other hand required UART. The microcontroller required individual and ample ports as the best configuration possible required each subsystem its own port. The SAM3X8E allows for every subsystem to communicate without the need to share ports between the devices. The Atmel SAM3X8E is housed on its own separate printed circuit board as the circuitry required for the microcontroller is of higher complexity than any other subsystem. Additionally, having the microcontroller isolated decreased price given a smaller board and minimized errors. The choice of the SAM3X8E also allowed the creation of the subsystems to be less complicated due to its many libraries provided by the community which streamlined the software development of the other subsystems with specific software libraries for every component used as well as example codes. The SAM3X8E also proves to be convenient with its

programming as it also houses many ways to communicate with a programming device such as computer as it also includes its own native USB port as well as programming and JTAG ports.

B. TLC5940 LED Drivers

This component is used to take the limited number of output pins we have on our PCB and use them to control multiple LEDs simultaneously. This chip allows us to control the PWM values individually on each of its 16 output channels. We can use this to let us control the brightness on each of the bulbs in each LED with very accurate results, which in turn lets us produce a variety of colors in a fairly simple manner. Each chip has the capacity to be cascaded from one to another so that we can control multiple chips, and therefore multiple LEDs, at once. They are fairly cheap and easy to wire, and our design is very dependent on their use.

C. RGB Common Anode LEDs

We needed to make use of RGB LEDs because we wanted to have multiple colors at our disposal for the display we had in mind. These LEDs specifically had to be Common Anode LEDs because these are the only type compatible with the afore-mentioned LED drivers. These are an easily obtained part and cheap to purchase, which is important since we expect to use at least 100 LEDs on our table's grid.

D. SN74HC165 Shift Registers

These chips serve a similar purpose to that of the LED Drivers in that they allow us to increase the number of output pins we can control using a limited number from the PCB. Unlike the LED Drivers, however, they do not have any PWM settings for their information channels. This is appropriate because we are using them as a means of controlling our array of 100 infrared sensors. We only need to know when these sensors have been triggered, and these Shift Registers are design to detect when their individual channels have been affected and send that information to the main CPU as a series of ones and zeros. We can then use that information to determine which infrared sensors were triggered and which ones weren't. The rest of our code can they focus on processing those signals to create the patterns on the surface of our table.

E. TCRT5000L Infrared Sensors

These sensors are used together with the LEDs to create our impact sensitive display. They can be used to detect objects reflecting infrared light to about 15 mm. This is sufficient for our purposes since we only intend to have them detect when a ball strikes the surface of our table as a trigger for the rest of the system to respond to. Each one is meant to set off a specific LED in our system, so we need 100 of them to match each of our LEDs.

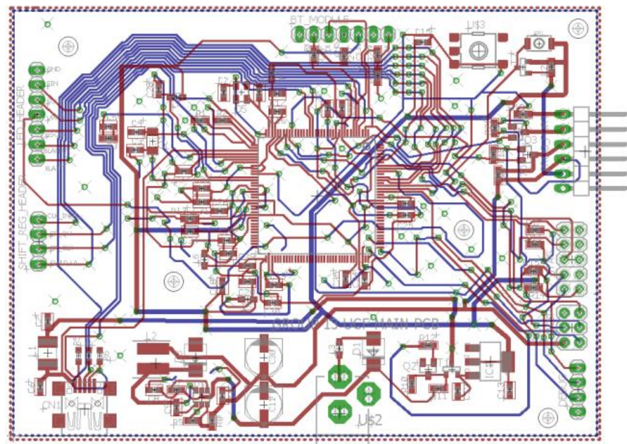
F. HC-05 ZS-040 Bluetooth Module

This module allows wireless communication with the table so that we can send information from the table to the smartphone app so that players can see the game information updated during play. We opted to use Bluetooth so that we could ensure communication with the table would exist, even if an internet connection wasn't available, and because it was easier to include in our design.

III. PCB DESIGN

At first it was agreed by the team to make a single PCB that would house all of the components. However, it was later proven that it would be problematic therefore the final design was to have a modular design which consists of different PCBs for different tasks. The main PCB houses the main microcontroller and it has the required ports to connect different PCBs that can be connected in cascading configuration. The creation of the main microcontroller board proved to be a part of high complexity as the components required to power the microcontroller and the overall schematic involved many components to achieve basic functionality. Fully explaining the functionality of the main PCB would go out of the scope of this document. SMD components were required otherwise the size of the board would increase exponentially.

Fig. 1 shows the main PCB on its final version. The main PCB has ports on the left and top and right side. On the left



of the board there are two ports. On the top we have the LED Driver port and on the bottom we have the Shift Register port which both connect to a PCB that has these components connected in series or "cascade" configuration. To the top we have the Bluetooth port where the NC-05 Bluetooth adapter is housed. To the right we have all the

programming ports given the case any port fails there is still the possibility to upload any code to the microcontroller. Finally, on the bottom we have the power and USB port.

The next PCB involves the LED Drivers and the Shift Registers. As it can be seen in Fig X. The board has two sets of ports. The first set connects servers as the input ports. In other words, these ports will receive the information from the main microcontroller or any other equal board and the second set will send the information to the next set of boards. Each board is capable of handling up to 25 RGB LEDs and sensors. Each board has 25 4-pin ports that (R, G, B, and touch) are used to send/receive information from and to the board. Additionally, this PCB has 25 VCC and GND ports to provide power to the aforementioned components. The same PCB and its ports are also used for the Cup Display subsystem that allows to keep track of the game score.

Lastly, an additional PCB to house the Ball Cleaner System. This one houses the microcontroller required to program the behavior of the components attached to it. This one has two voltage rails. One being 5V and the other being 12V. This board is considered as its own isolated system as it does not communicate with any other subsystem.

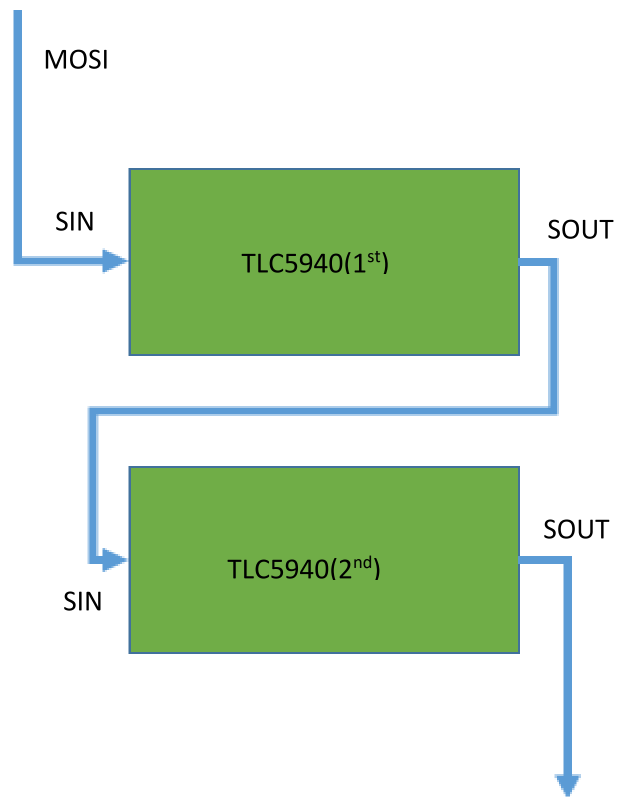
IV. LED ARRAY

This is the most obvious system in our design. It makes use of 100 LEDs that display various patterns across the surface of the table during gameplay, depending on the currently running game mode. Each of the 100 LEDs takes 3 output channels to control, where the output channels being used to control them come from the TLC5940 LED Drivers. Each of the output channels on these drivers has 4096 different PWM rates that can be programmed to adjust the brightness on the bulb connected to that specific channel. Since each LED has a 3 bulbs, and each channel can be programmed to input a different PWM rate into that bulb, we can produce a variety of colors for our display.

A. *Wiring*

In terms of wiring, the SIN port of the first TLC5940 chip is wired to the PCB where the main MCU has been set. We use a specific "Master Out Slave In" (MOSI) pin on the PCB to connect to the SIN port of the LED Driver. This is the specific pin from the PCB that is responsible for sending the serial information that is used to control the PWM rates on the LED Drivers. Meanwhile, the SOUT Port of the first chip is connected to the SIN port of the next chip in the array. This allows signals from one chip to travel to the next so that multiple chips can be connected in series and controlled via a single MCU. Other than the first chip in the series, the remaining chips are connected with their SIN port linked to the SOUT of the previous chip, and their SOUT port linked to the next chip's SIN port. The following is a small diagram of the setup:

Fig. 2. Diagram illustrating the way that TLC5940 chips are linked together in order to communicate serial information.



The remainder of our pins connecting between the PCB and the first TLC5940 are connected in parallel with the same ports on the remaining TLC5940s. This is because the information sent to those ports is not dependent on the number of LED Drivers in use. Thus, we are able to connect them in parallel without any complications. Below is a diagram of how these connections are made:

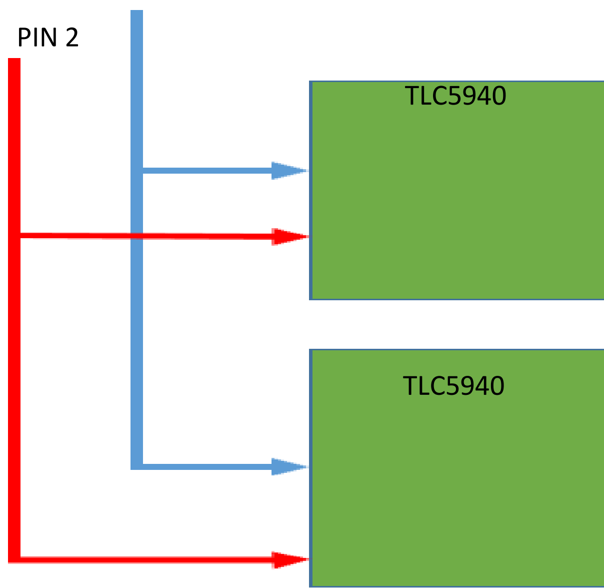


Fig. 3. Illustration of parallel connection between two TLC5940 chips and two pins from the PCB.

B. Programming

This wiring allows us to send signals to the LEDs through the LED Drivers. The exact code that we use to do this is a combination of functions obtained from a pre-written library about controlling the TLC5940 LED Drivers that is specifically compatible with processor we are using because it was built into an Arduino Due experimenter board. The functions of this library include the ability to pulse any output channel of the TLC5940 chip with a specific rate between 0 – 4095 baud. The library also includes several configuration options that allow us to set the number of LED Driver chips that we expect to use so that it will automatically know to move from the output channels on one chip to the next automatically. With these basic tools at our disposal it is possible to write more complex patterns for the LEDs in our display to execute.

With the basic means of sending signals to the TLC5940 chips accounted for, we moved on to building a more complex method of keeping track of the current state information of each LED. Using Dynamic Memory Allocation we built “LED Nodes” in the table’s program. Dynamic Memory Allocation is a technique used for several different purposes in code. In this particular case, it was used for making new variables that could have multiple characteristics. This way the table program could keep track of multiple objects with several different qualities in an organized and efficient manner. The following is how the structure has been written into our code:

PIN 1

```
typedef struct LedNode {
        int idNumber;
        COLOR current;
        int blinking;
        int off;
        struct LedNode *north;
        struct LedNode *east;
        struct LedNode *south;
        struct LedNode *west;

}LedNode;
```

Fig. 4. Copy of the code used to build the “LED Node” structures in our code.

By constructing our “LED Node” with these qualities we can effectively track the changes in the LEDs as a game mode is running and several different patterns are working on the table at once. Each node is assigned the variable “idNumber” so that we can distinguish each one in a lineup. This number is also used in a crucial equation for determining the corresponding channel to output signals on from the TLC5940 chips. The channels from all chips are numbered from the first channel on the first chip, the last channel on the last chip. In order to find the channels corresponding to a particular LED, we use the following equation:

$$\text{idNumber} \times 3 = \text{first channel} \tag{1}$$

$$(\text{idNumber} \times 3) + 1 = \text{second channel} \tag{2}$$

$$(\text{idNumber} \times 3) + 2 = \text{third channel} \tag{3}$$

This provides us with the channels corresponding to each LED in a simple manner. Meanwhile, there is also a “blinking” variable to indicate to the rest of the code that this LED should have a steady blinking action rather than remain a solid and constant color. The variable “current” is based on an enumerated type we’ve built called “COLOR” which holds the names of each of the different colors we will be making use of in our program. There are a total of 8 different values of COLOR including the following: BLUE, GREEN, RED, YELLOW, PINK, PURPLE, ORANGE, WHITE, and BLACK. Whenever the current color displayed by an LED on the table matches one of these specific values of COLOR, our code will update the information on the corresponding “LED Node” to reflect that change. Getting the program to track this information is much easier than trying to do it any other way, so it was always intended that our code make use of a structure like this.

Take note that there is also a simple “off” variable in the node as well. Sometimes patterns in our code will cause an LED to deactivate, but not change its color. Other times we

will want an LED to remain dark regardless of whether they have been triggered or not. The “off” variable lets us know when we have our LED deactivated because a pattern has indicated that it should be that way, and when it needs to light up again, it already knows what color it is supposed to have because that information is stored in “current. However if we want an LED to completely ignore any of the conditions that would cause it to light up ordinarily, we would set that LED’s “current” value to BLACK and it would always appear as a single dark spot on the grid until another pattern in our program changes that value.

In addition to keeping track of the current state information of each LED, these nodes are designed to keep track of which LEDs in our grid are adjacent to one another using the following four pointers: north, east, south, and west. Some of the patterns we implemented use LEDs that are adjacent to one another to create an effect on the table. One such effect is the “ripple” effect where a single LED lights up in response to impact on the surface of the table, followed by the LEDs surrounding it lighting up afterward to create an effect similar to a drop of water falling into a pond. The simplest way to do this is to have each LED keep a pointer that remembers which LED in the grid is next to it on all sides. This way we only need to tell a single LED to light up, followed by the LEDs that it is connected to through its nodes to create effects like the one we described.

As a direct consequence of the building of these nodes, making patterns on the surface of the table is much easier. All nodes can be referenced from a single double pointer that has been built to hold enough memory for all 100 nodes in our LED array. Any time there is need to access a particular node, they can be identified by the unique “idNumber” of that specific node. Then the program can modify the properties of it according the patterns running on the table. With this in place, the table will function as desired so long as the sensor array is functioning properly.

V. SENSOR ARRAY

In order to make the table responsive to impacts from a ball used during play, it was armed with an equivalent number of infrared sensors to match the number of LEDs in the LED Array. With 100 different infrared sensors being used at once, there needed to be a means of connecting and receiving information from each one individually. To do this, each of the infrared sensors was wired through an SN74HC165 Shift Register. Doing so allowed for the main MCU to receive accurate information about which sensor was triggered and which wasn’t using only a few output pins to connect to the first Shift Register.

A. Wiring

The Shift Registers used in the design of this sensor array aren’t that different from the LED Drivers mentioned earlier. Shift Registers can be used to receive signals from multiple inputs simultaneously and have those signals organized as a set of ones and zeros kept in order from first channel to last channel. Knowing this, the infrared sensors were wired to the Shift Registers so that any time an object came within range of the sensors it would be able to send a signal through a specific channel on the Shift Register that would indicate the status of that particular sensor had changed.

In order to connect the PCB holding the main MCU to multiple Shift Registers, we connected one pin from the PCB to the “Q7” port on the Shift Register so that the MCU could receive serial input information. This, however, would only allow it to receive information from the first chip in the series. The Sensor Array has to be able to manage at least 100 separate infrared sensors, and that specifically requires that we have more than one Shift Register. To obtain this additional information, the Shift Registers are cascaded into each other so that they can pass information from the farthest channel on the last channel of the last chip, back through the wire connecting the PCB to the first chip. This serial communication of data through multiple chips allows for communication with multiple Shift Registers, and therefore multiple infrared sensors. Meanwhile, the other wires between the PCB and the first Shift Register in the series are also connected in parallel with all other

B. Programming

Each infrared sensor only takes a single channel from a Shift Register to send information, and each Shift Register has 8 channels to receive information from. When the main MCU is told to read information from the shift register, it receives that information as a string of ones and zeros that indicate if any of the input channels have changed status. The rest of the program then reacts to these signal changes and triggers any patterns that the table has been set to show. The following is a diagram of how the infrared signal information is received:

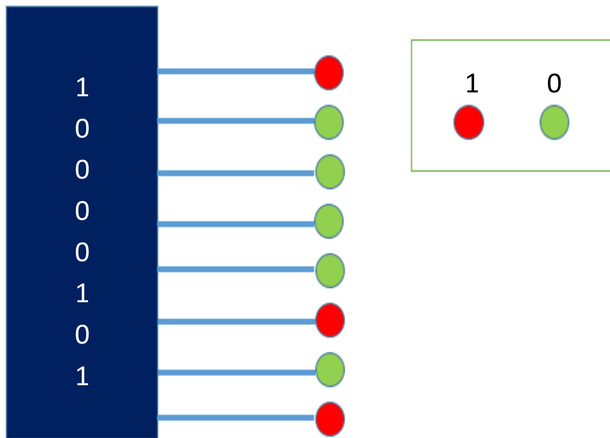


Fig. 5. Illustration of how data from infrared sensors is read by the shift registers.

The above diagram is useful for understanding how the signals are gathered from each of the sensors, but to actually distinguish which sensor was activated and which wasn't requires more mathematical analysis. The strings of ones and zeros are a binary code, and this code is translated into a base 10 number in the main program in order for the rest the table's program to function. When all sensors are inactive, the binary code sent to the main program from all of the sensors translates to about

$$(2^{101})-1 = \text{total sensor data} \quad (4)$$

However, when one or more of the sensors are activated, this number is reduced by a specific sum depending on which of the sensors was activated. For example, if the very first sensor in the array was activated, the new value read from the Shift Registers would be the equivalent of

$$((2^{101})-1) – (2^{0}) \quad (5)$$

This is because the first sensor adds the sum of (2^0) to the overall sum. The next sensor adds (2^1) followed by (2^2) and (2^3) and so on. As such, the main program for the table is designed to calculate the difference in the sum and use this information to deduce which of the sensors in the array was triggered. That information is then coordinated with the LED Array to make patterns appear on the table's surface.

VI. CUP DISPLAY SYSTEM

This system combines technology from both the LED Array and the Sensor Array to create a system that is able to use the infrared sensors to detect when cups have been put into proper positioning for play in a game of Beer Pong. When cups have been set for play, the system lights up the surface of the table on either end in the specific spots where the cups are supposed to be positioned. Cups will also be removed from play as part of the game, which helps to indicate when one team has scored certain points against another. As such, this particular display system will also detect when cups are removed as part of the game and send the corresponding information to update the app that tracks in-game scores.

VII. GAME-TRACKING APP

A. Android Application

The Beer Grid will be making use of an Android based application to keep track of player's statistics and the current state of any game being played on the table. To develop this application, the project group will be making use of the Android Studio IDE.

The smartphone app will allow for players to sign in and create their own username. To store this information, the application will be connected to an online MySQL database. This database will not only hold the players' usernames and passwords, but it will store the amount of wins and losses a player has. To get this information, the smartphone application will be connected to the table via Bluetooth. The number of cups remaining on both sides of the table is the data that table will send to the application. The app will have a game in progress page which will show the data received from the table indicating the progress of the game being played. Once one score equals to zero, the game ends. A winner (the player who still have points left) and loser (the player whose score reaches zero) is determined, and the smartphone app will then send the outcome of the match for both players to the online database that holds the status information of the players.

The app will also be handling a queuing mechanic for the project. While a game is currently in progress, the application will be able to allow other users to enter in a queue which will show which users are next to use The Beer Grid. This queue will also be handled by an online database. This database just store the username and the position in which they queued up for playing. To enter this database, a game will have to be in progress, and once a user hits the queue up button in the app, their username gets added to the database. Once that game finishes, the first person added to the database will get put into the game. Once in the game, the table will erase that user's information.

B. Database

For the application database, the Beer Grid uses two MySQL databases. One database is used to store player/user information such as username, password, wins, and losses. The other database will be used to store a queue

on the order of who is up next to play. So this database will only have information stored in it when there are multiple players trying to play on the table at the same time.

C. Overall Communications Setup

Although the app will have all features available for any android operated mobile device, only one device at a time will be connected to the Beer Grid via Bluetooth. So the device that is connected to the Bluetooth will be the one updating the wins and losses in the player's stats databases. All other devices that uses the app will not connect to the table and all the information of the application will be updated through the databases. The figure below illustrates how all of this works together.

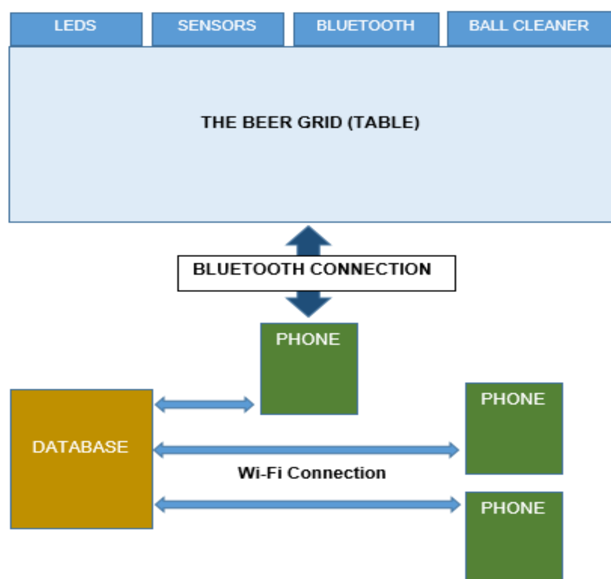


Fig. 6. Illustration of how all the subsystems are connected to one another.

VIII. POWER SUPPLY

The power supply required to provide enough power to drive all LEDs turned on at the same time in order to have a safe operation. Each LED would require at least 20mA per channel, therefore 60mA per LED. 100 LEDs would require at least 6A of current at any given time on a 3.3V rail that would power the microcontroller, LED Driver, Shift Registers, RGB LEDs and, Proximity Sensors. During the design phase it was determined that it would be beneficial if all the circuits involving all of those subsystems be running on that voltage as the main microcontroller runs on the same voltage and rule out any incapability issue given by voltage. An additional 5V rail is required for an additional microcontroller that will control another set of infrared sensors and blower fans that run on 12V which would require additional components for successful communication.

IX. BALL CLEANER

The ball cleaner is a device that is used to remove any debris in case a player drops the ball to the ground. This tends to be a common issue, so it made sense to include functionality in the table that would account for it. Additionally the ball cleaner will dry the ball from normal usage in order to provide players an optimal game experience and to remove the need to clean the ball manually with the player's shirt or a nearby rag. This is also a frequent occurrence in games of Beer Pong. The functionality of this subsystem is completely isolated from the main microcontroller. This decision was mainly done in order to avoid increasing the complexity of the main microcontroller board and to avoid any communication issues given by voltage. The microcontroller chosen is the Atmel ATMEGA328P as the team acquired coding experience with it and the overall components requirement to have the microcontroller running on a circuit board are much lower than many of the other available choices. The additional components utilized on this subsystem are the blower fans and proximity sensors. The proximity sensor utilized is the Q12AB6FF50 from Banner and the blower fan is the 41851 Bilge Blower from Seachoice. Both of these devices require 12V of power to run, but since the microcontroller runs at 5V of power then some additional devices were required in order to achieve communication between the two components. Two selected relays including the LM2-12D-R from Rayex Elec. and R10S-E1-Y1-J5.0K from Potter Brumfield were chosen to achieve this task. The set up consists of a PBC tubing that will go from one end of the table to the next for each player. The PBC tubing has an opening on one side where the proximity sensor is located and the blower fan is placed in a junction. The microcontroller will constantly check any change in the infrared spectra emitted by the sensor, once the ball passes through and a change is made then it activates the blower fan which runs on PWM at a speed that will send the ball to the exit which is located at the other end of the table. A second sensor will then sense the ball at the exit and once the ball is removed from the second sensors view, the microcontroller will then turn the blower off. The figure shown below is a sense of where the sensors are located and the one side with the blower.

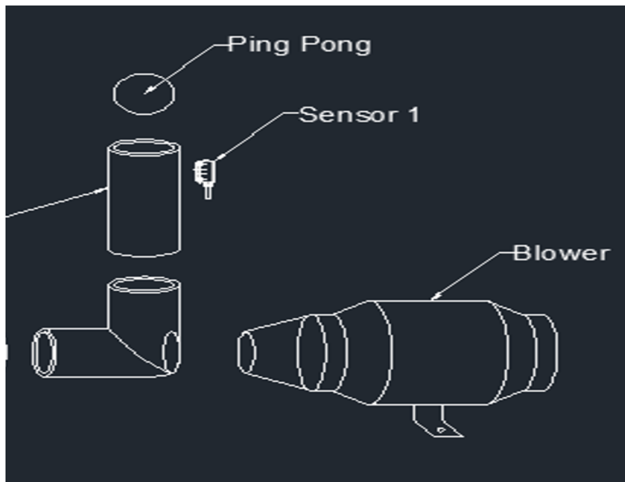


Fig. 7. Illustration of components used in building the Ball Cleaner System.

X. PROJECT MEMBERS

Edgar Alastre is a 25-year old international student from Venezuela pursuing a Bachelor's of Science in Electrical Engineering at the University of Central Florida. He has a keen interest in computer hardware such as CPUs and GPUs and will seek work experience on these fields.

Jonathan Chang is a 22 year old graduating senior pursuing a Bachelor's of Science in Computer Engineering at the University of Central Florida. Has the ability to work with a variety of computer programming languages such as, but not limited to, Java and C. Currently searching for a job in fields related to software design.

Colton Myers is a 24-year old student pursuing a Bachelor's of Science in Electrical Engineering at University of Central Florida. His interests are working in military simulation, specifically on military aircraft weapon systems at Lockheed.

Ashish Naik is a 23-year old student raised here in the United States. He is attempting to obtain a Bachelor's of Science in Computer Engineering at the University of Central Florida. Work specialties are coding in C, Java, and MSP430 Assembly Code. Will be pursuing a job related to Software Design after graduation.

ACKNOWLEDGMENT

The authors wish to express their gratitude towards their families for their support throughout this project. None of this would have been possible without them.