ECE 453 Senior Design Final Report
Electronic Beirut Table

Group 8
Paul Dickey
David Hoese
Christopher Wilcox

CONTENTS

Executive Summary

The Electronic Beirut Table (EBT) uses the provided hardware platform as well as custom hardware components to create an interactive playing surface that improves an already entertaining activity. Similar to an average Beirut table, or beer pong table, there are 10 cups per side for a total of 20 cup positions. On the EBT, each cup position can detect a cup that is placed on its location, which makes it possible to keep score during a game. Each cup location can be illuminated to any RGB color and can be configured by the user.

The EBT also has multiple demo modes. The various demo modes light up the cup pods with different patterns and timings. These demo modes are meant to act as entertaining light shows while a game is not being played.

Once the table is turned on, a demo mode is automatically started. The GUI can be connected to the table at any time and allows the user to enter different demo modes or the game mode. The GUI provides the user with a digital representation of the table's current state. This can be used for a live scoreboard during the game, or a visualization of the board during demo modes. The GUI also allows the two playing teams to choose the colors displayed on their side when cups are present and when cups are not present. Along with the current state of the table, the GUI also displays the score for the currently played game.

The pulse width modulation of the LED colors is controlled by the FPGA on the provided TLL5000 development board. The FPGA sends serial control data to each TLC5940 Texas Instrument chip. The Electronic Beirut Table uses four TLC5940 chips that pulse width modulates five RGB LEDs each, for total control of all twenty RGB LEDs. Cup detection is achieved via one photoresistor per cup position that detects how much light is visible above each cup.

The non-electronic portion of the table is built out of plywood. There are two sheets that are kept apart by dowels and bolts in the corners. Each end of the table has ten 3.5-inch cup holes. These create room for the reflective cones that are placed underneath. Siliconized paper was used as a diffusing agent, and a sheet of acrylic plastic was placed over the top to make everything stable and water resistant.

The Electronic Beirut Table will provide enjoyment and an improved game experience to all participants.

Expectations Met and Test Plan

Hardware Requirements

Cup Detection

The initial cup detection requirements were defined as being able to detect a cup in a binary manner, with no bouncing. Due to the unreliable and inconsistent nature of photoresistors, cup detection is not as perfect as originally hoped. Where the original plan was to have one threshold for all photo-resistors, individual thresholds for photoresistors were created instead to account for the inconsistencies. Some photoresistors were of a poor enough quality (the valid threshold range was small) that the bias resistor in the photoresistor circuit had to be changed to another value to allow for better cup detection.

LED Color and Brightness

The original project plan required that the LEDs of the Electronic Beirut Table would operate with little issue. The LEDs were expected to have bright and solid (not flickering) colors and include all colors in the RGB spectrum. These expectations became difficult to meet after a couple of the TLC5940 chips were found to be malfunctioning. After some investigation, it was discovered that the datasheet was read incorrectly and the chips were not able to sink as much current as originally anticipated. We also found that a start sequence was necessary to minimize the risk of breaking the TLC5940 integrated circuit. After some hardware and Verilog changes, less current was required to achieve a desirable result. This also stopped the chips from malfunctioning. Additionally, we were unable to get *all* colors, particularly colors that could be described as 'near white.' While these are indeed different colors, the white factor overpowers the unique color making it hard to differentiate in a well lit room.

PCB Design

A PCB board was designed to contain the TLC5940 PWM Chips as well as the ten photoresistor-capacitor circuits needed to detect cup presence. The board had a 60-pin header to attach the LEDs (four pins each) and the photoresistor (two pins) to the board. A 32-pin header ran from the PCB to the ARM Development Board. This allowed the Verilog to sample and write to the hardware and provided a clean package for all components. On the pages following you will find a figure showing the PCB board. For a PCB schematic, please see Appendix B.

Wiring

The largest unexpected hardware problem was an issue with wiring from the TLL5000 board to the individual Beirut table ends. The first indicator that something was wrong with the wiring was that in certain demo modes, and with certain colors, cups would either flicker a lot, display the wrong color, or both. This problem seemed to be helped by placing a hand on the ribbon cable; it was also helped when shorter cables were used. It was first thought that this was due to noise in the ribbon cables because of the fast clocks (four total clocks per ribbon cable) in the cable. The main reason for these problems was found to be the 3.3V supplied signal being attenuated on the lines. Thus, a 5.0V supplied signal provided a strong enough signal for correct colors and very little flickering.

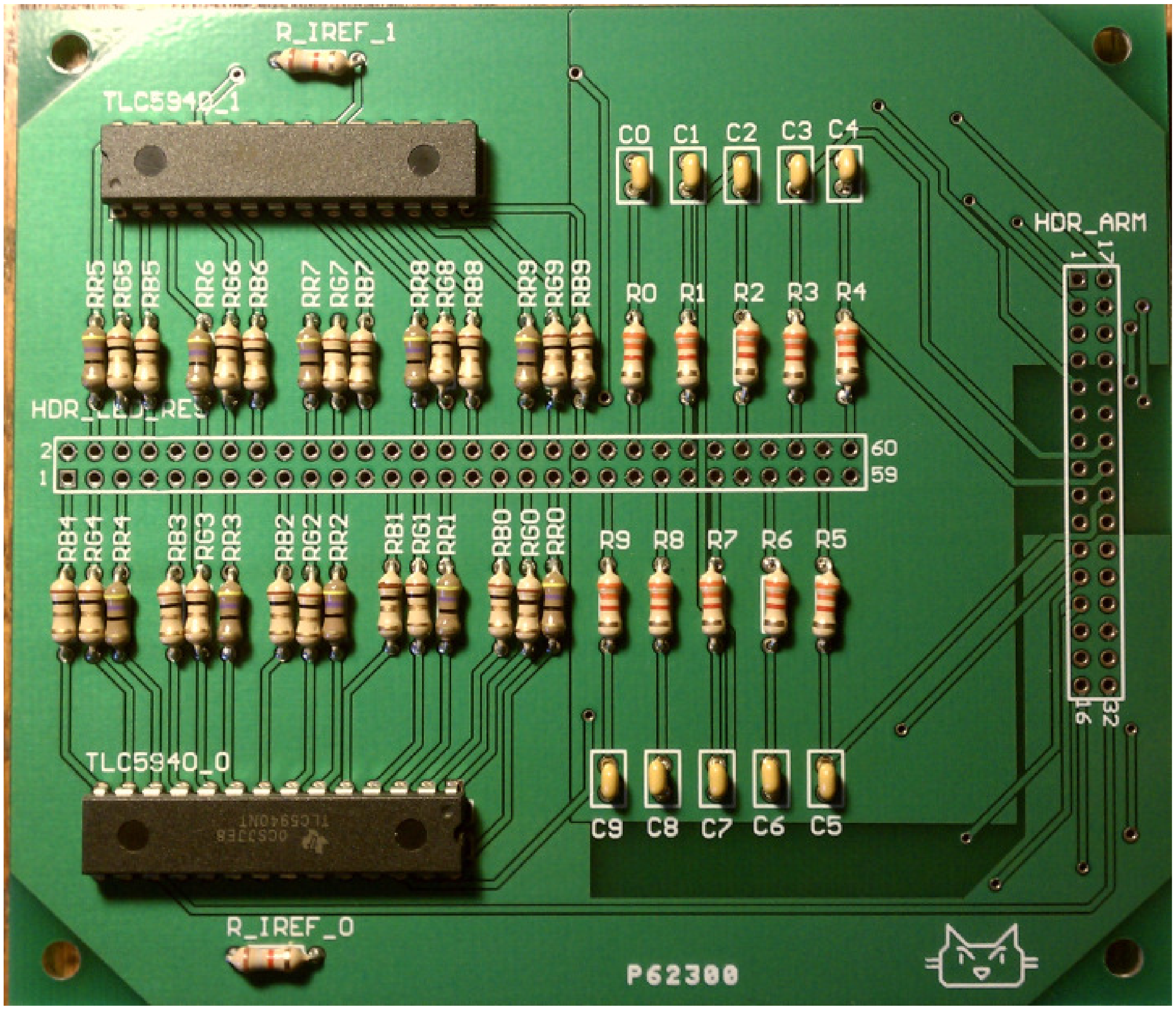


Figure 1. Assembled PCB Board. The board contains 2 TLC5940 PWM chips as well as 10 Photoresistor control circuits

Software Requirements

Five major software components were required to meet the fully functioning design: C Library, Game Mode, Demo Modes, Graphical User Interface, and Socket Layer.

C Library

The C library was responsible for accessing Verilog memory space and any other direct functioning with the hardware. By having this library as a separation layer, it allowed us to each work independently. The GUI only worried about the data reported by the C library and the C worried about how to communicate with the Verilog. We took care to make sure the protocol was well defined so that all the pieces would come together with ease.

Game Mode

The Game Mode was responsible for controlling the on and off colors for each team, reading sensor values using the C library, and reporting a win of the game. The creation of this was greatly simplified due to putting in time on the C library. We did change our minds on one design aspect of the Game Mode. We originally intended to have the Game Mode control covered and uncovered cup colors for the teams, but realized that any implementation of this would prove complicated for a user to utilize effectively. For this reason, the GUI was made responsible for allowing custom color selection.

Demo Modes

The Demo Modes, like the Game Mode, were greatly simplified by offloading the calls to the hardware with a library. The Demo Modes were used to make the table the center of attention, even when game was not in play. In the end, we designed 8 distinct demonstration patterns that added to the overall project: Cycle Rainbow, Cycle Rainbow Blend, Rainbow Wave, Smooth Rainbow Wave, Motion Detect Red, Breathing, Snake, and Counting Colors.

Socket Layer

The Socket Layer was designed as an interface between the C program and the GUI. The purpose of the Socket Layer was to make sure that the C program did not have to get bogged down with the details of whom it was communicating with, or if a client was even connected. We accomplished this by creating two processes that communicate with each other via pipes. The main process runs the C program, while the forked child sets up a socket and handles accepting subsequent requests on that socket. Once a client connects, the communications manager takes all input from the socket and sends it through the pipe to the main process. Similarly, it takes all output from the main process and sends it out on the socket. If a client disconnects, the connection manager simply stops passing data to/from the main program. This allowed us to develop the C application without knowledge of the client state, which greatly simplified the logic.

Graphical User Interface

The Graphical User Interface (GUI) was designed in C# and WPF and met all the requirements we had of it. It displays the current table state, team scores, current game mode, and grants the user the ability to changed current mode and modify settings of each mode. One distinct feature of the GUI is the ability for the user to pick any RGB color value to represent on and off colors. While originally a part of the Game Mode, by allowing the GUI control over this, we can grant the user the ability to choose any color on the spectrum, as opposed to ten distinct values as they would have been displayed on the table. We believe that, while this does not meet our original design, it is a change that both maintains and enhances all original functionality while simplifying the user experience.

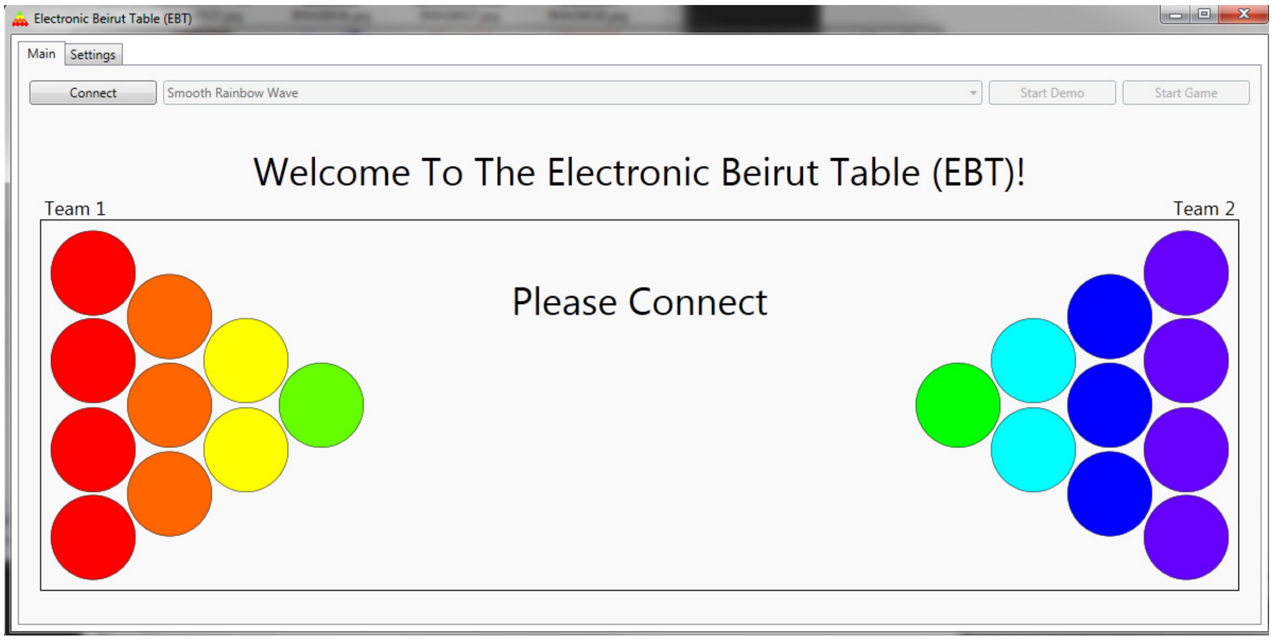
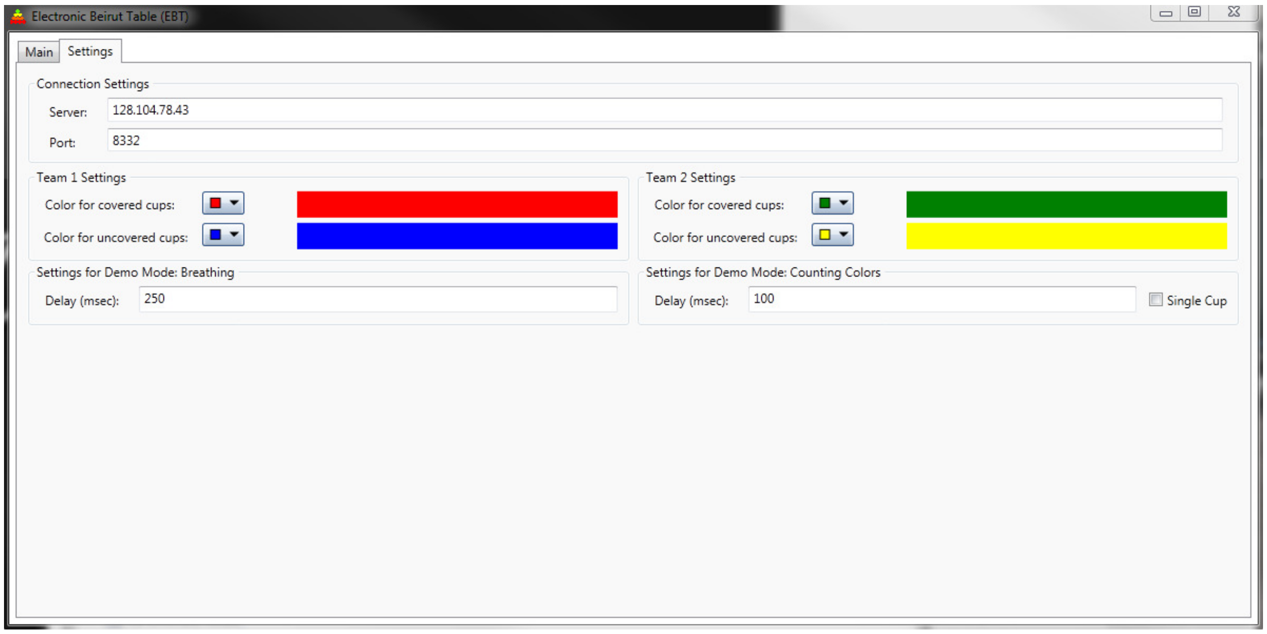


Figure 2. GUI Main Tab. This tab contains all command sending buttons and a display which shows full state of the table. This tab can present a score as well as tell the user what mode they are currently running in.



Connection Settings
Server: 128.104.78.43
Port: 8332

Team 1 Settings
Color for covered cups: 🟥
Color for uncovered cups: 🟦

Team 2 Settings
Color for covered cups: 🟩
Color for uncovered cups: 🟨

Settings for Demo Mode: Breathing
Delay (msec): 250

Settings for Demo Mode: Counting Colors
Delay (msec): 100
☐ Single Cup

Figure 3. GUI Settings Tab. This tab allows the user to set the server address and port to connect to, demo parameters, as well as color choice for game mode.

Test Plan Effectiveness

The original test plan involved many progressive tests; for example, start with one LED with one color, then try more colors, then try multiple LEDs with one color, etc. This test plan was designed with the expectation of a limited number of components and a lot of testing time. Due to having multiple "backup" parts and little time, the majority of the test plan was slimmed down to fewer tests per component. For example, instead of testing one LED with one color and moving on from there, we started with all the LEDs with one color and then went to all LEDs with multiple colors.

The test plan would have been more effective if wiring problems had been anticipated. We did not foresee the level of difficulty we would have diagnosing issues involving signal integrity. Since we did not account for this, a test plan for the wiring was developed on a need be basis.

Final Cost Report

We were originally hoping to make a device with a cost of around $200 (before the cost of the ARM development board). The end cost for the device is $215.01. It did end up being slightly more than anticipated, but is within a 10% deviation. For this reason, we believe that we hit our target pricing.

Note that this budget does not include costs for time or miscellaneous equipment used. Actual production of such a device would cost more to account for the additional cost of a microcontroller, as well as construction cost and profit.

On the following page you will find the Bill of Materials for our project (Figure 4).

PAGE FOR Figure 4. Bill of Materials

Final Time Report

On the following page(s), you will find a Gantt chart (Figure 6) showing the schedule that we followed to complete this project. This schedule breaks tasks down to days as opposed to hours. For this reason, debugging time is accumulated into the larger categories.

Below we have also included a table showing labor division. This project had multiple tasks that needed to be accomplished. For most tasks, an individual was designated to lead the feature.

Task/Job	Dave	Chris	Paul
Design	34%	33%	33%
Verilog	97%	1%	2%
GUI	0%	50%	50%
C Library	13%	80%	7%
Demo Modes (all)	33%	34%	33%
Game Mode	15%	85%	0%
Socket Layer	0%	10%	90%
Table Construction	1%	47%	52%
PCB Design	1%	98%	1%
Hardware Debugging	100%	100%	100%
Software Debugging	20%	75%	75%

Figure 5. Time Division Table. Note: Row values are approximate and may not sum to 100%.

PAGE FOR FIGURE 6 (GANTT CHARTS)

PAGE FOR FIGURE 6 (GANTT CHARTS)

PAGE FOR FIGURE 6 (GANTT CHARTS)

PAGE FOR FIGURE 6 (GANTT CHARTS)

VI. Appendix A: Lessons Learned

1. Don't send clock signals over long cables.
One of our most unexpected problems came from the wiring from the development board to the table ends. We first suspected that it was the clock signals causing noise on the serial data lines. Our fastest clocks that we were running were 25 MHz to start with. We slowed the clock down and this seemed to help the noise, but did not fix all of it. The next two lessons continue to describe the problems we had with wiring.

2. Use higher voltages for longer cable runs/use shortest cable lengths possible.
After talking with other students and instructors, we decided to change our signal supply voltage from 3.3V to 5.0V. This caused the largest reduction in signal noise in our wiring. When doing wire runs longer than a foot, use a higher signal voltage. If a lower voltage is used, the signal is likely to get attenuated and the system ends up not working as intended. In our case, the LEDs flickered or were the wrong color. The other solution we found was to make the cables as short as possible. Less cable means less attenuation, which means better results.

3. Design PCB to fit microcontroller connectors.
Due to our inexperience in PCB design, we did not create the best header layout to connect to our development boards. When designing the PCB, we decided to take all the signals we needed to output and put them on one 32-pin header. We did not think about wiring or signal interference. This resulted in a lot of wiring work to connect one 32-pin header to two 40-pin headers on the development board. To fix this issue, we created a wire wrapped board to connect the two different sets of headers.

4. If possible, do not use photoresistors for object detection.
Photoresistors are very inconsistent and unreliable. We used the photoresistors to detect light levels and detect if a cup was present based on a threshold. The inconsistency of the photoresistors caused some bouncing to occur when a cup was placed on the table. Some photoresistors would detect a cup, then not detect a cup, then detect a cup again, because the range of its resistance was very small.

5. Read the datasheet, then `while(true) { read it again }`.
After reading the datasheet for the TLC5940, writing the Verilog, and getting the LEDs to turn on, we started burning chips up. If we had read the datasheet more closely (in terms of current limits) we would have been more cautious with the total amount of constant current the chip can handle. We also would have noticed that a power sequence is needed when the chip is powered to 5.0V Vcc.

6. For time-sensitive projects, choose the simpler solution.
Our first attempt at making a cup “pod” was to use polyester resin to hold the LED and photo-resistor. In short, this was a huge failure. The resin cracked and was a dark (non-transparent) hardened material which did not work well for displaying RGB colors. The simpler solution was to hot glue paper cones to the wood. These cones held the components in a way that made it “easy” to replace damaged components without having to replace the entire pod.

VII. Appendix B: Diagrams

a. EXTERNAL CIRCUITRY SCHEMATIC

   i. PAGE FOR PCB SCHEMATIC

      Figure B.1

b. BLOCK DIAGRAM OF FPGA LOGIC

i. PAGE FOR BLOCK DIAGRAM

Figure B.2

c. FUNCTIONAL BLOCK DIAGRAM

i. PAGE FOR BLOCK DIAGRAM

Figure B.3

VIII. ATTACHED: Compact Disc of Software and Verilog Code