Adruino Uno Controlling the Sparkfun Color LCD Shield

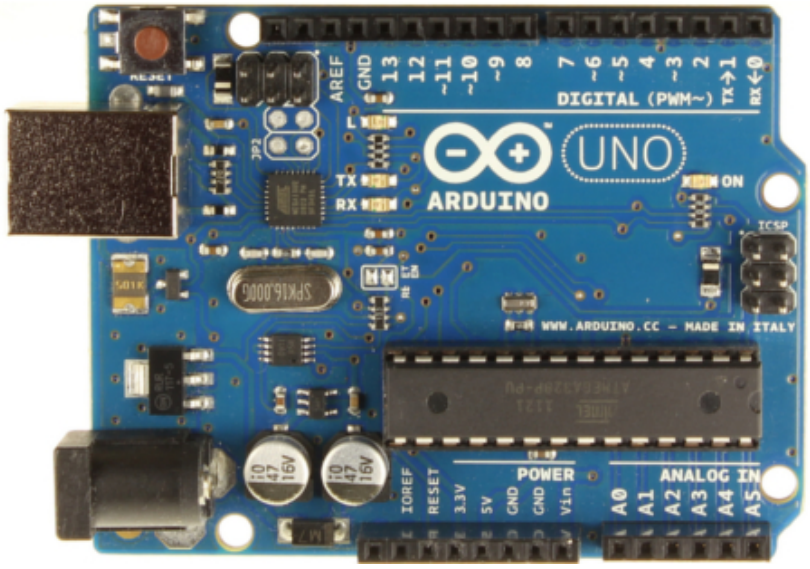
by Heather Briggs, Dan Lewis, Endurance Idehen, and Lauren O'Keefe

Table of Contents

I. Introduction:

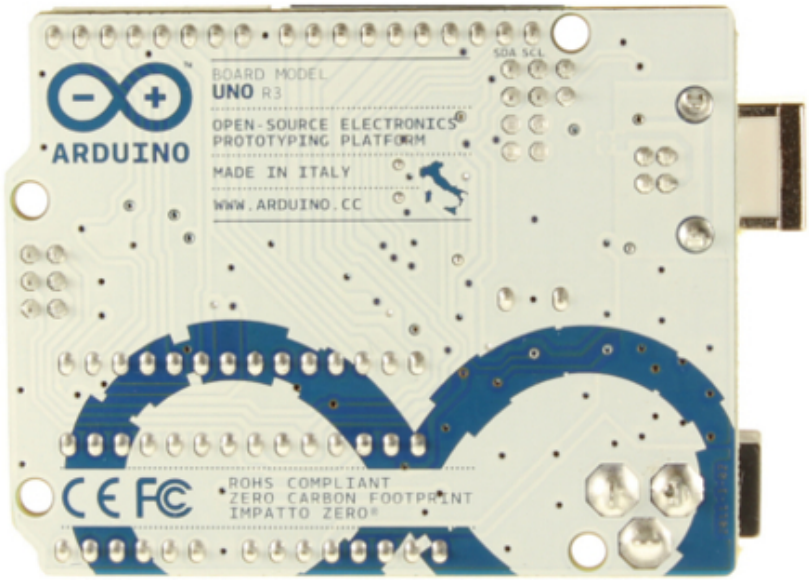
Our group worked with an Arduino Uno with an AtMega328 and a Color LCD Shield. We used jumper cables connected to potentiometers in order to control the movements on the LCD screen. The group tested the capabilities of this hardware by loading Micro-Pong onto the Arduino chip. The screen then displays the pong ball and paddles and the potentiometers allow the paddles to move up and down along the sides of the screen.

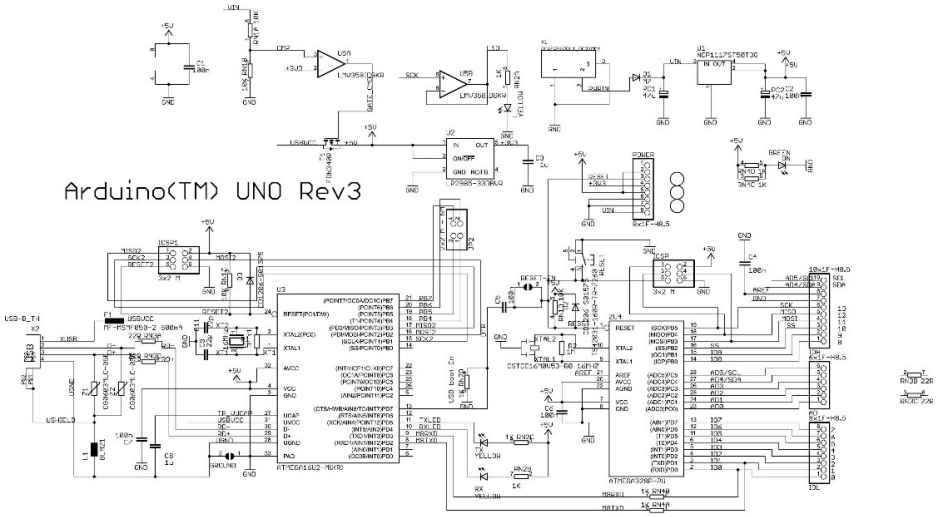
II. Microcontroller Platform:



In our experiment, we are looking at the Arduino Uno R3 board with the AtMega328. Basic specs include 14 digital input/output pins, 6 analog inputs, a 16 MHz crystal oscillator, which connections for a USB or Power jack, and a basic reset button. The AtMega itself has

about 32 KB of flash memory (.5 KB used by the boot loader), 2 KB of SRAM, 1 KB of EEPROM, and runs at a blazing 16 MHz. For our experiments that we played with, we noticed couple limitations in the way we used memory. Trying to store large arrays full of data, even bit arrays, can result in crashing the program we upload into the Arduino. We had to take extra precautions with our usage of memory while coding in C/C++ and make sure we were not being wasteful with such limited resources.





III. The Testing Device:

The widget we chose to tackle was the Sparkfun Color LCD Shield, a common Nokia 6100 LCD panel affixed to a special PCB designed to interface nicely with the Arduino. The panel sports a 132x132 resolution and the board included three momentary push buttons. Using a helper library maintained by Sparkfun, the panel can be made to display text, draw shapes, or display images for any number of projects, anywhere one might need to display color graphics.

The device communicates via a 9-bit SPI interface and comes preloaded with one or two possible drivers, one made by Phillips and the other by Epson. The driver defines a number of commands that can be performed on the panel, most important of which being the ability to designate the position and size of a rectangle one wants to draw and then feeding in a series of 12-bit values corresponding to the 12-bit color of each pixel in the

rectangle specified, with 4 bits dedicated to each color channel: Red, Green, and Blue. The following diagram demonstrates how two 12-bit color pixel values can be represented by three 8-bit bytes.

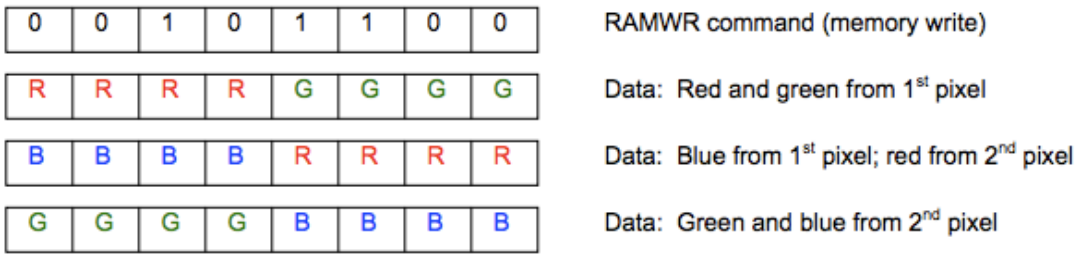


Figure 7. Color encoding for 12 bits / pixel - example illustrates sending 2 pixels

At it's core, the driver can only draw rectangles, from 1x1 rectangles to 132x132 ones. It is through this rectangle drawing that libraries, such as the one maintained by Sparkfun, can render more complicated images like circles, lines, or text.

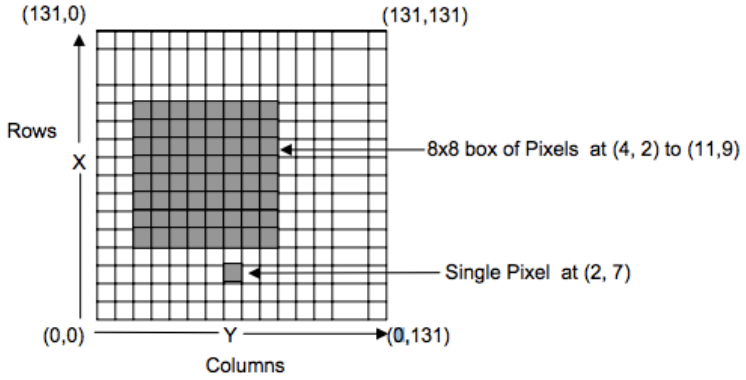


Figure 6. Philips PCF8833 Pixel Memory

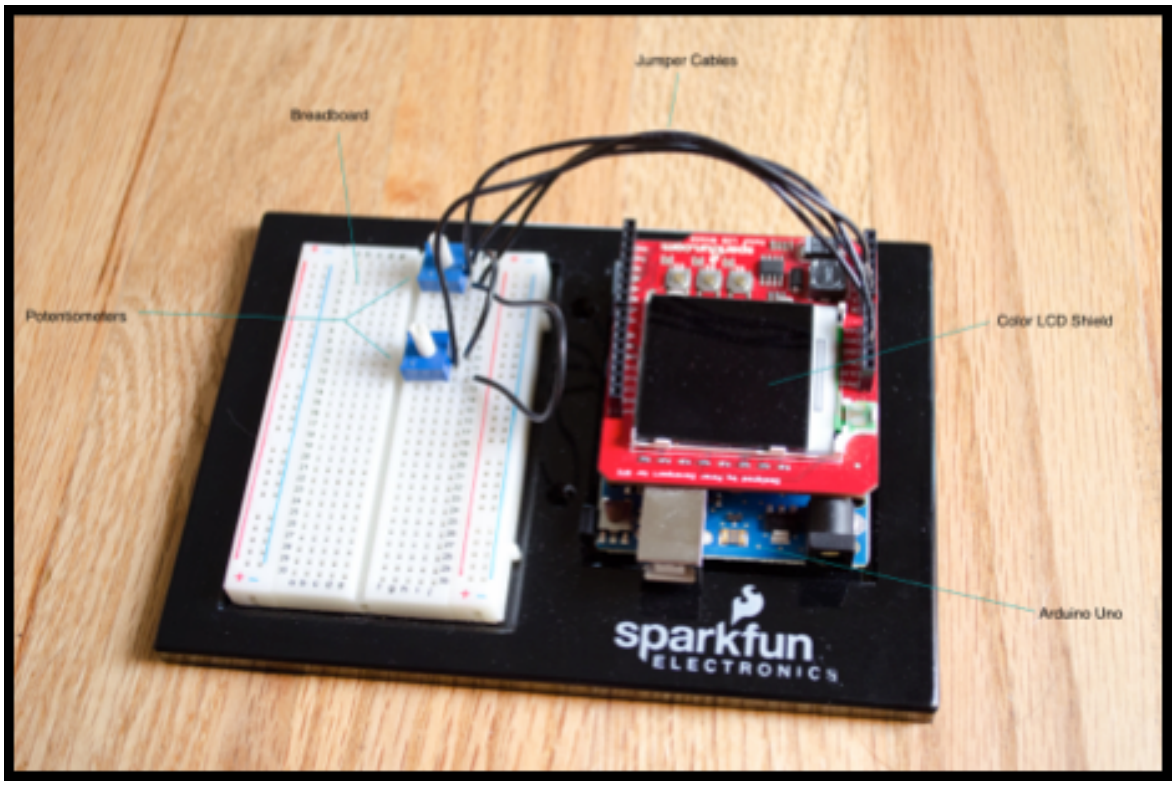
IV. Development Tools:

The hardware that we were given to use was the Color LCD Shield, and the Arduino Uno board. The board itself needed to be controlled along with the LCD shield that is mounted on top of it to produce the output we wanted. The Arduino comes with the basic Arduino IDE which is a

C/C++ compiler designed to compile and subsequently upload the “schematic” (what Arduino calls source code) to the board itself. To control the LCD shield, we had to download additional libraries from Sparkfun, and reference the libraries in our schematic. Afterwards, we read the README files, and the Github resources for the Color LCD shield libraries to be able to control the device. With these resources and software handy, we were able to create our end result: Micro-Pong.

V. Our Experiment:

We used a mixture of sample code from Peter Davenport’s ColorLCDShield libraries and group written code to implement the game Micro-Pong onto the Arduino microcontroller and test the Color LCD Shield’s capabilities. In order to perform this task, the LCD Shield was attached to the Arduino with the microcontroller on it’s board and then jumper cables hook the potentiometers into the LCD shield using a breadboard.



We had to first decipher how to load code onto the LCD which included figuring out the maker of the screen (EPSON) and the contrast settings (40). The group then used sample code from the "Zany Circles" sample and altered the code to set up the ball and paddles. In order to program collisions, we had to write new code to allow the paddle and ball objects to interact creating the proper effects. The ball is initialized in the center of the screen and travels at a set velocity towards the origin, the lower lefthand corner. If it collides with a paddle or the top or bottom of the screen it bounces at the reverse angle to its last trajectory. If it instead collides with the left or right sides of the screen, the ball is reset to the middle of the screen. The buttons on the LCD Shield are used to control the speed of the ball; S1 is "SLOW," S2 is "MED," and S3 is "SLOW." Each time one of these buttons is hit the ball resets to the center.

We wanted to be able to use the LCD screen in order to have a visual that could be interacted with. Originally we wanted to use a joystick, but the jumper cables with

potentiometers proved to be a better way to implement Pong. The idea for Pong was inspired by a throwback to the earliest arcade game. Its seemed like a fun way to test the system while staying within its capabilities. We added in color to show the ability of LCD screen to display colored pixels.

VI. Conclusion:

While working on the project, the group ran into several surprise issues along the way. Since Dan was the most experienced with graphics and game programming, he decided test attempt to find the easiest and most efficient ways for manipulating the screen. He did not anticipate, however, that the refresh rate of the screen would be so low that clearing the screen each frame update and then redrawing the entire scene (a technique very commonly used in game programming for the PC and current day game consoles) would take so long as to make it infeasible for anything that requires movement on the screen. This fact necessitates a much more complex program that not only draws objects to the screen but cleans up its old drawn objects when they are no longer needed.

Another problem that no one in the group was anticipating was that of space. After seeing that both the group before us as well as well as the test pattern example code supplied with the library managed to display some kind of bitmap image, we figured that it would be simple enough to implement. First, we determined that since no standard image format known to us stored its data in the same 12-bit format the Nokia 6100 required, we would have to find or create a program to convert a standard, easily produced format such as BMP to the special one we needed. Dan produced a C++ program that would take a BMP image and produce a

formatted .h file containing an array of 12-bit values corresponding to the pixels of the original. The program worked and produced the file we needed, but upon first loading the code comprising of the pixel array of a 130x130 pixel image as well as the code needed to display it, it would simply not respond, displaying a blank screen. After experimenting with the image size we surmised that the array was somehow overflowing the part of the Arduino where the program was stored, due to the size of the array. We are still unsure how the last group was able to display their image.

The greatest lesson we learned in this project was that if you have expectations going into a project with hardware you do not understand yet, there may be a requirement for reevaluation of how to approach the problem. We all learned some of the ramifications of not having enough space or speed and will no doubt take them into consideration when working with embedded systems in the future.

VII. Contributions:

Heather contributed the potentiometer circuitry, ball and paddle code, reference and code documentation. Lauren contributed the integration of the ball and paddle code, cleaned up the code a little, wrote the “our experiment” and introduction sections, and compiled the report. Endurance did a lot of the initial work on figuring out how the Arduino functioned, where the libraries were located, and the settings that had to be configured for it to work. He also programmed the speed into the buttons and wrote the Microcontroller and Development tools

portions of the group report. Dan contributed the Test Device and Conclusion portions of the paper. He also did a fair amount of figuring out how the device worked and figuring out its graphical and memory limitations. This was very crucial to the experiment we ended up choosing.

VIII. Project Code:

The source code for this project is located at the following link:

https://github.com/scubagirl/Pong

The main file that is deployed to the Arduino is Pong.ino. Like all Arduino code, this file contains initializations, the setup() function, and the loop() function. The setup() function initializes the screen. The loop() function handles the game play features such as paddle control, screen redrawing, collision detection, and pong physics. It utilizes functions from ColorLCDShield.cpp and pong_support.cpp to perform these tasks.

ColorLCDShield.cpp and ColorLCDShield.h are support files that were available for free online. These files provide all of the necessary functions to control the LCD screen. We used the init() function to initialize the screen, the contrast() function to set and adjust the contrast, the clear() function to clear the screen, and the setRect() function to draw the paddles. One function that ColorLCDShield.h did not provide was the ability to draw a solid circle. Therefore, we wrote a function called setSolidCircle() so that we could represent a pong ball.

The pong_support.cpp and pong_support.h files define a paddle and a ball class. The paddle class is used to store the dimensions of the paddle, paddle color, paddle position, and

previous paddle position. It also has a function to compute the new paddle position based on the analog input. The ball class is used to store the radius, position, and the vector velocity of the ball. It also has a function that computes the updated position of the ball. This function does collision detection as well.

IX. References:

Provides the library to use the LCD screen:
https://github.com/sparkfun/ColorLCDShield/tree/master/firmware/Arduino/libraries/ColorLCDShield

Pinout for LCD shield(used to verify that the analog inputs are unused by LCD shield):

http://www.sparkfun.com/datasheets/DevTools/Arduino/Color-LCD-Shield-v12.pdf

Arduino for Dummies (getting started):

http://arduino.cc/en/Guide/Windows

Analog Specs:

http://arduino.cc/en/Main/arduinoBoardUno (pictures in the microcontroller section)