A

ASCII to LED Image Generator

Abstract:

Light arrays are commonplace in today's society, found in everything from Christmas decorations to downtown streets. We decided to spin on the concept of standard light arrays by creating a controlled array of lights that is able to cycle through a series of images via a switch. The images are created through conversion of ASCII text images and are uploaded through a USB connection in the controller and sent to the array.

GROUP E

Matthew Kunze
Yevgeni Rubin
Jim Diep

EECS129B Professor Klefstad

ject Description

When deciding on a topic for a project, the main factor that weighs on the decision process is the ıcept of usefulness. What would we want to use? Christmas first brought about the idea of a light ay. Light arrays have a wide variety of uses; they are not limited to just decorations. The concept ıind our idea is to be able to put a sign of lights outside of the front door of a home and to modify or ınge it from inside the house without having to go outside and fiddle with parts or components. The ht array should be able to display a series of pictures, originating as ASCII files composed of characters d spaces. One clever example of our design at use would be a "Welcome" sign or a sign that could be anged to "BOO!" on Halloween. If the cycling of the images were to be automated, simple animation uld also be shown through a series of pictures. This all seemed like a playful idea at first, but as we ɔught about the details it became more complex; controlling an array of hundreds of lights would not be easy as we first thought.

cking a Microcontroller

The first problem which needed to be addressed was to determine which microcontroller should be ed to drive the signals. We experimented with two different microcontrollers to accomplish this task: e Arduino and a 32 I/O pin microcontroller (Atmega16). Using the Arduino we would route 8 control gnals through two 4-to-16 decoders. One decoder would control rows while the other would control ɔlumns. This approach uses fewer data lines from the microcontroller itself. Using the other icrocontroller we would directly connect all 16 rows and all 16 columns to I/O ports of the icrocontroller, without the use of decoders. In the end, we decided on the Arduino for two reasons: the ɔgramming environment for the Arduino was more reliable and its functionality was more robust. The ɔher microcontroller was slow and unreliable, not always producing quality results.

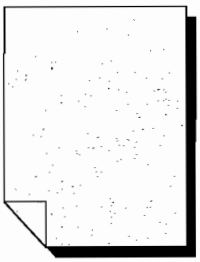
eeping the Lights On

The second problem that we encountered was to determine how to effectively switch lights on and ff as needed and to determine how to form a picture from these lights. The solution to this problem was ɔ form a grid, where the columns are connected to the cathodes of the LEDs and the rows are connected ɔ the anodes. When a light is off, the cathode is set to V_{dd} and the anode is set to ground. To turn a light n, the anode is changed to V_{dd} from ground and the cathode is changed to ground. Pictures will be ɔrmed by flashing lights on at a rapid pace, so that the eye is tricked into seeing them as always on, when ı fact they are really flashing on and off too quickly to for humans to notice (on the order of ıicroseconds).

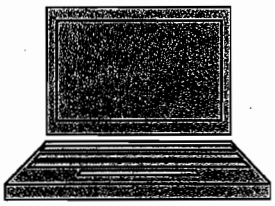
Creating the Image

With the concept for the circuitry in place, the method of transforming an image into control and data signals must be addressed. In relation to light arrays, an optimal choice for a beginning data type is an ASCII text file, where a space would mean light off and any character would mean light on. The file would have a fixed number of characters per row and a fixed length, matching that of the light array. Since the microcontroller does not have sufficient tools to obtain and convert an ASCII text file into the required binary data type we decided to put that conversion program onto a laptop separate from the microcontroller network. The program would be written in Java and would receive a text file as an input and output a pre-formatted binary string which would then be copied and pasted into the microcontroller code. The device will contain a USB port for which to plug in the laptop. When updating the microcontroller with new pictures, all that must be done is to copy the binary string output from the laptop's program and paste it into the microcontroller's code. The new images would then be uploaded to the microcontroller.

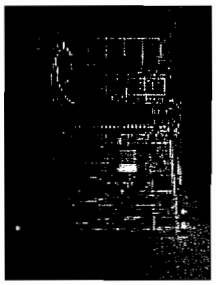
Block Diagrams (Part 1 – Flow)



ASCII text file is created with 16 characters/spaces on each line and a total of 16 lines. The file is an ASCII representation of an image to display



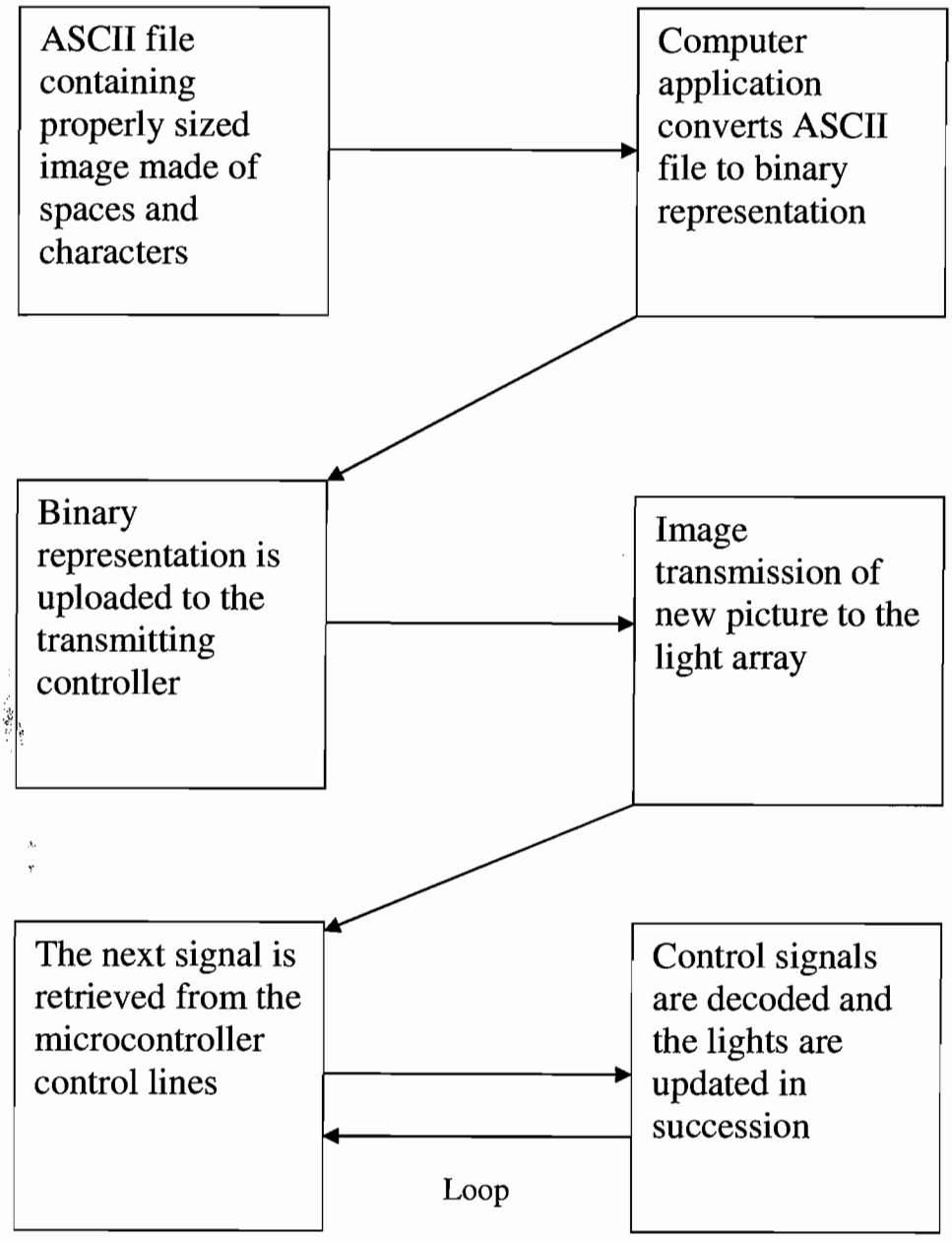
Java program running on a laptop converts the ASCII text file into binary



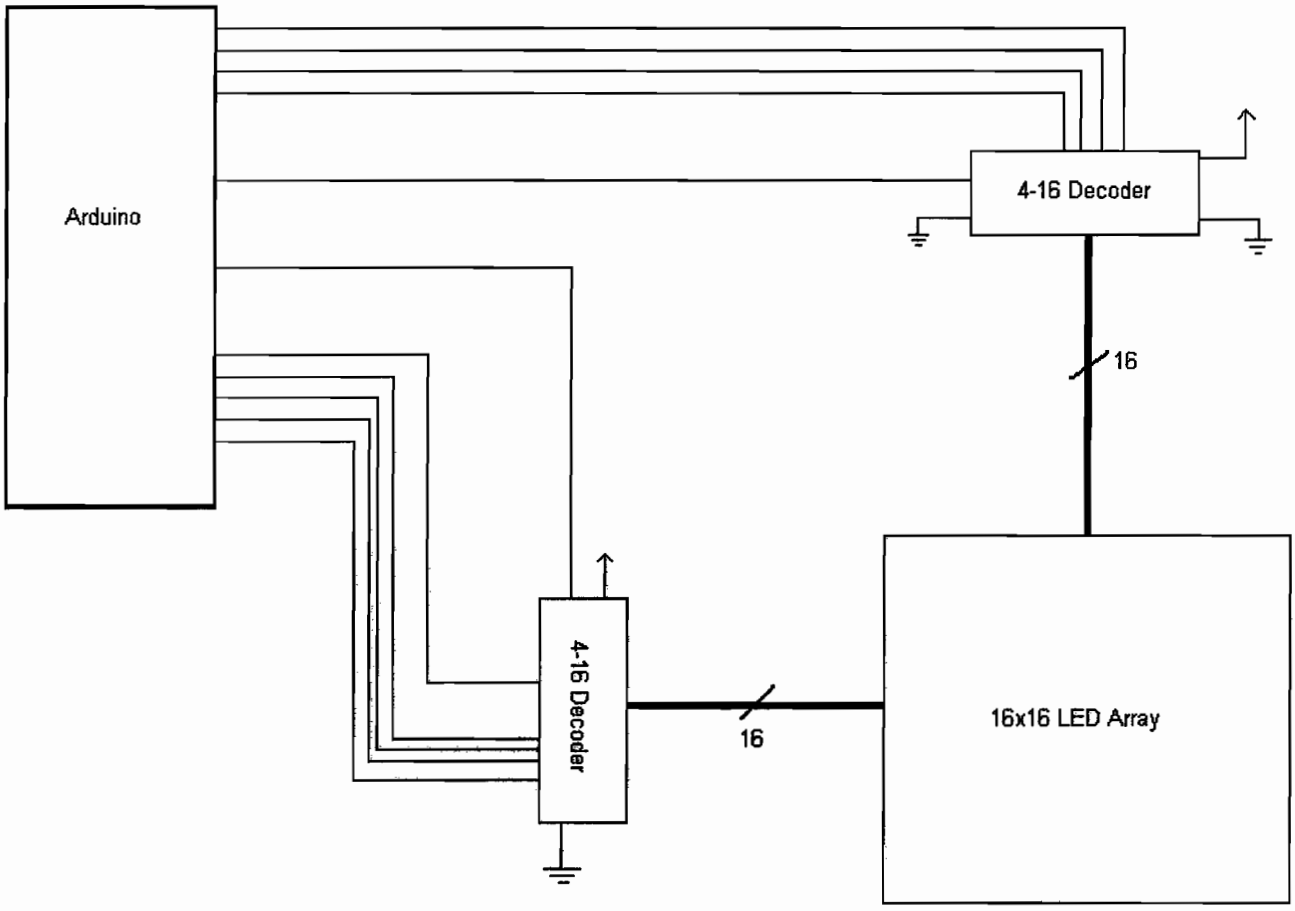
The binary output is inserted into the microcontroller's program picture queue and the image becomes ready to display

The next image is displayed by the LED array connected to the microcontroller. Each light that is displayed in the image is powered on individually, repeated at a rate which is too fast to recognize.

Block Diagrams (Part 2 - Step-by-Step)



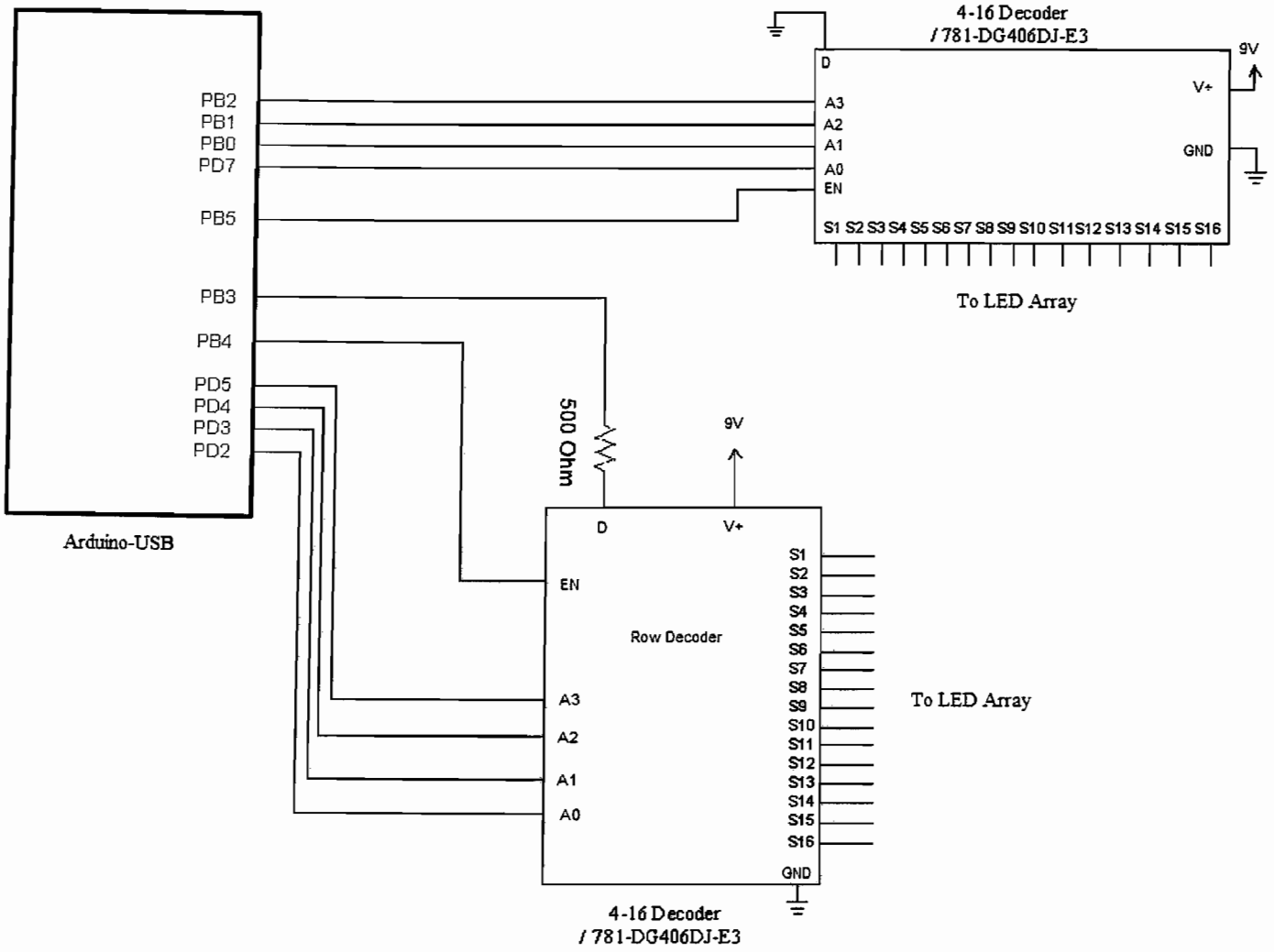
System Level Block Diagram



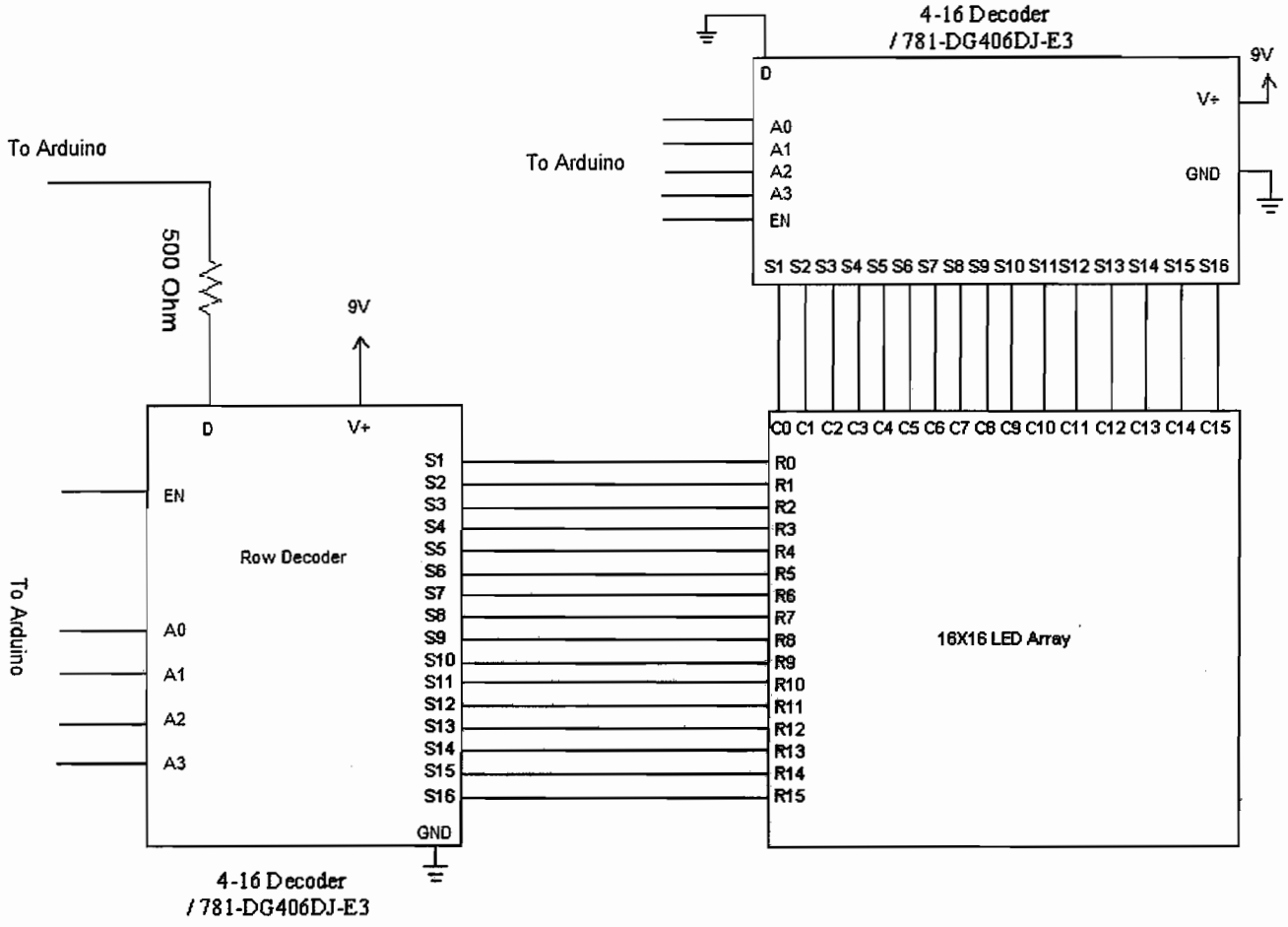
High Level Schematic Diagram

The Arduino is connected to two 4-16 decoders and they are in turn connected to a 16x16 LED array. The decoders take 8 input signals from the microcontroller and converts them into 2 of 32 output signals to light a specific LED in the matrix. This is repeated multiple times per second to give the illusion of a static image, but in reality, multiple LEDs are being flashed repeatedly to trick the eye. This is all made possible by selectively applying VDD or GND to rows and columns via the decoders; this allows us to control 256 LEDs with only 8 control signals. The columns are connected to the cathodes of the LEDs and the rows are connected to the anodes.

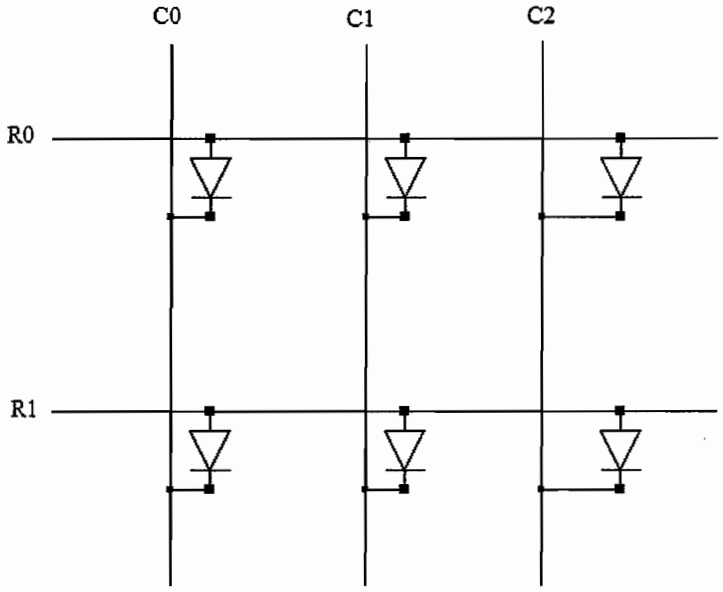
Circuit Level Block Diagram



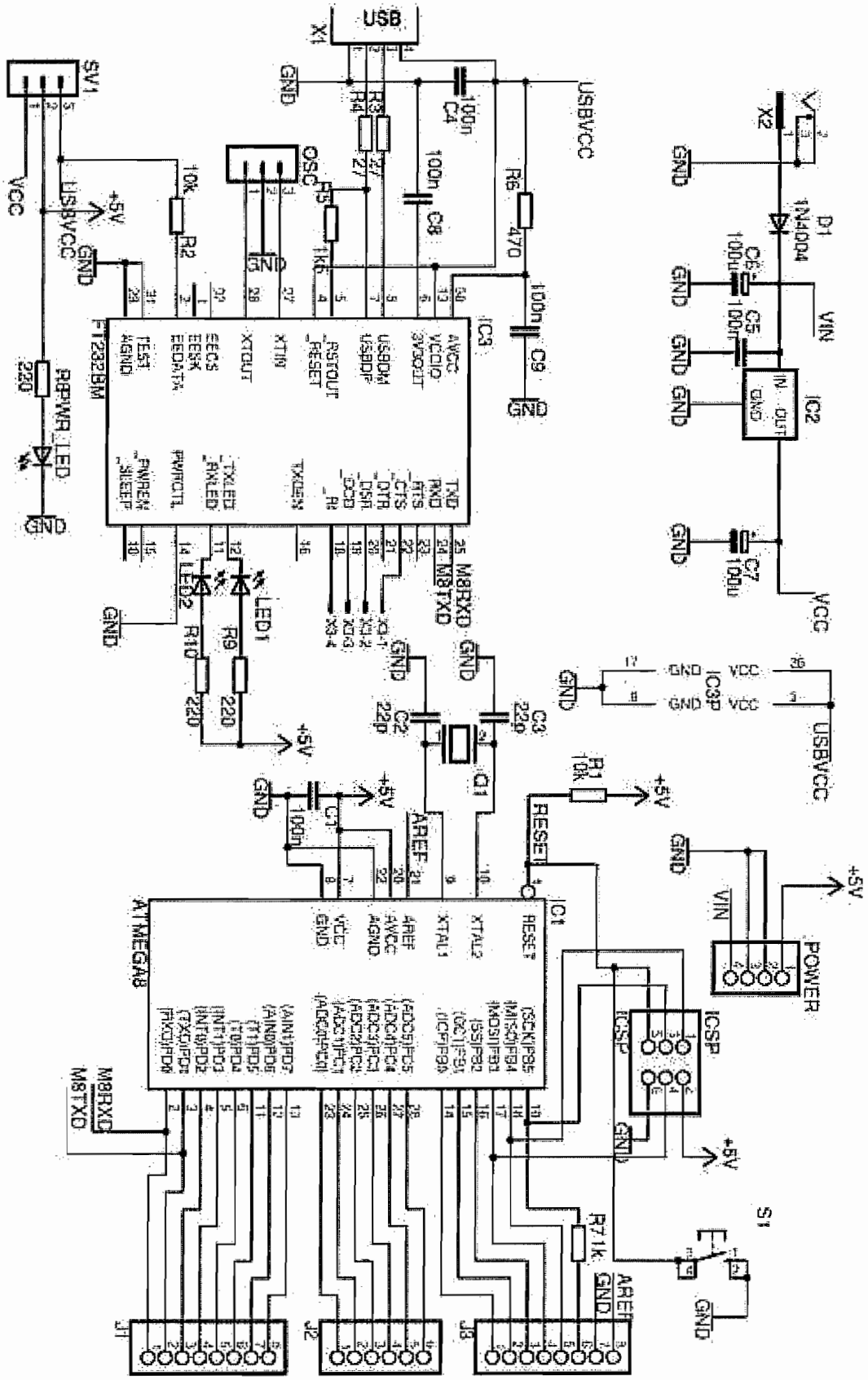
Arduino → 4-16 Decoders



4-16 Decoders → 16x16 LED Array



Close-up View of LED Array on Prototyping Board



Arduino-USB Schematic (Courtesy of SparkFun Electronics)

Software Description

There are 2 pieces of software associated with this project:

1. Java Programming (ASCII to Binary Converter)
2. Microcontroller Programming

ASCII to Binary Converter

The java program runs off of a laptop and performs the function of converting ASCII images composed of characters and spaces into binary strings in an acceptable format to be copy and pasted into the microcontroller code. The program accepts input in the form of an ASCII text file and checks to make sure that all lines in the ASCII file fit the specified format of a combination of 16 characters and spaces for a total of 16 lines. If the input file is invalid, the incorrect line of the file will be noted in the output and the user must correct the file. The output is given in the form of binary strings which can be copied and pasted into the microcontroller's program. The format for the binary strings is as follows:

Ex:

{{ 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1 }
{ 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1 }
... };

The final output is given in the console/command line and from there can be copied to the microcontroller code.

Microcontroller Code

The microcontroller program consists of code running on the microcontroller itself. The program content consists of a series of array variables for the binary images. The array's contents are filled with the output from the ASCII to Binary Converter program. After the code is copied to the proper array location, the image display process begins. A loop cycles through the pictures one at a time in such a fashion to create animation. Multiple frames looped quickly create movement and liveliness to still images. If there is only one image, the single image is held static on the LED array. For each picture, a series of if-statements determine and decode the array into microcontroller signals; these signals control the row and column states — flipping the polarity of the nodes causing LEDs to activate at the appropriate time. The lights are then turned on one at a time, and refreshed at a very high rate (in the order of microseconds). By doing this, the eye of the viewer is tricked into seeing all of the intended lights as being turned on. The translation of the array to lit LED grid is so fast, that the eye is fooled into believing a static image exists.

TEST PROCEDURE

This Document contains a list of test cases, divided into two groups: functional tests and negative tests. Functional tests test the functionality of the machine. Success of all functional tests means that the device is able to perform all of the intended tasks when given proper input. Negative tests test the ability of the system to respond to improper inputs and values. Success of all negative tests means that the device is able to respond to all predicted error conditions as expected.

FUNCTONAL TESTS

Stimulus	Success Criteria	Verified
1. Input a properly sized (16x16) ASCII image into the ASCII to binary conversion program located on an available computer.	Binary string of length 256 is returned. This string should match the expected binary output for the given ASCII image.	
2. Input the binary string generated in the previous test to the microcontroller's program. Upload the modified program.	The microcontroller accepts the revised program without error,	
3. Add an additional picture to the microcontroller's program in the same manner as the previous tests indicated. This test is designed to test that the microcontroller can accept a variable number of inputs. (multiples of 1, 2, and 10 will be tested) Upload the modified program.	The microcontroller accepts the revised program without error.	
4. Add 5 more images to the microcontroller's program. This test is to verify that the microcontroller has sufficient memory to handle a reasonable number of images. Upload the modified program.	The microcontroller accepts the revised program without error.	

NEGATIVE TESTS

Stimulus	Success Criteria	Verified
1. Input an ASCII image that is greater than the allowed size to the binary conversion program (>16x16).	The program should reject the input value and print an accurate error message.	
2. Input an ASCII image that is smaller than the allowed size to the binary conversion program (<16x16).	The program should reject the input value and print an accurate error message.	
3. Attempt to upload the microcontroller's program with no entries in queue.	The microcontroller accepts the revised program without error.	
4. Attempt to upload the microcontroller's program with an excessive number of entries (1000). Upload the modified program.	The program should not upload successfully. An error message should be received while uploading.	
5. Enable the viewing of serial messages on a pc attached to the microcontroller. Upload the program with a series of images in the queue. One of the images should be improperly formatted.	The serial messages should report that there is an inaccuracy. The LED lights may display some or none of the picture, depending on the location and quantity of inaccuracies.	

Parts List / Cost Analysis

Quantity	Price	Name / Part Number	Manufacturer	Supplier Name
1	29.95	Arduino USB board / SKU#: Arduino-USB	SmartProjects	www.sparkfun.com
2	27.58	4-16 Decoder / 781-DG406DJ-E3	Vishay	www.mouser.com
300	16.80	LED (Green) / 211086	LITE-ON	www.jameco.com
6	6.00	9V battery	Any	Any
3	16.47	100' reel 22awg, Solid Wire / 36792	CONSOLIDATED WI	www.jameco.com
100	1.00	1k Resistors / 690865	Various	www.jameco.com
1	9.99	Prototyping Board / 8000-45-LF	Twin Industries	Fry's Electronics
Total	$107.79			

Summary

In summary, the project works as it was intended; however, there are still issues which would need to be improved before this became a marketable product.

First, a microcontroller more specifically designed for this project would allow for more I/O lines while retaining the necessary functionalities. This would allow the light array size to increase drastically. Along with that same idea, larger and faster decoders could be used to further enhance the size and response of the LED array. Ideally, more memory would be necessary as well to allow for additional room for pictures. A larger number of pictures would also mean an increase in the quality of animation that could be done. The more images that we are able to store, the smoother the animation could be. Currently be are restricted to 2 or 3 frames of animation, but increased memory would allow for 8-10 frames of animation.

Second, the inputting of new images would need to be streamlined so that the microcontroller code does not need to be recompiled when pictures are changed. Recompiling is quick, but for most users this would be viewed as an unnecessary hassle. We explored the possibility of direct transfer of the binary image to the microcontroller, but found that it restricted the different ports that we were able to use. In order to have direct access, Pin0 and Pin1 on the microcontroller must not be occupied; otherwise the serial transfer will not work. We found that to be too restrictive; this could be remedied by a different microcontroller.

Third, in some pictures, lights adjacent to the ones intended are partially turned on. They appear very dimly lit; this problem has to do with the refresh rate of the signals from the microcontroller and the decoder. This could be remedied with the use of a higher-end microcontroller with better control over the signals and the speed at which they can be sent; another option is a superior decoder which has greater switching times. Given our limited budget, we ran a series of tests to optimize this problem with the Arduino and were able to produce acceptable results. Although the problem still exists slightly, the pictures are clearly visible. We feel that with a greater budget, we would be able to stamp out these minor issues to create crystal clear images with no leakage lighting on adjacent LEDs.

In conclusion, we feel that our project has accurately demonstrated the concept and understanding required to build a marketable design while being scaled back enough to accommodate our limited budget and time schedule.