Controlling a Dot Matrix LED Display with a Microcontroller

By Matt Stabile

University of California Santa Barbara

Media Arts and Technology

MAT 200C Winter 2008

Abstract

A tutorial on the basics of choosing and setting up a microcontroller to control an LED matrix is provided. Fundamentals of LED matrices and microcontrollers are first covered, followed by tips on making the necessary interconnections between the two. Relevant software for development is surveyed and finally some example code is provided for testing the LED matrix.

Introduction

This paper will provide a comprehensive tutorial on how to drive and control a dot matrix Light-Emitting Diode (LED) display with a microcontroller. The display used is a commercially available PCB mount 8 x 8 dot matrix RGB LED display, with a total of 192 individual LEDs that are controlled by 32 control signals. The microcontroller used is an Atmel ATMega128; however, the control and programming will be explained in general terms as well to allow for adaptation to any comparable microcontroller or LED matrix.

Figure 1: 8x8 Dot Matrix LED Display

The Dot Matrix LED Display

An LED Matrix consists of an array of LED' s which are interconnected such that the positive terminal (anode) of each LED in the same column are connected together and the negative terminal (cathode) of each LED in the same row are connected together. Note that this could be the other way around as well, with the positive terminals connected to the rows and the negative terminals connected to the columns. Figure 2 shows a schematic for a typical 8 x 8 matrix with single color LEDs.

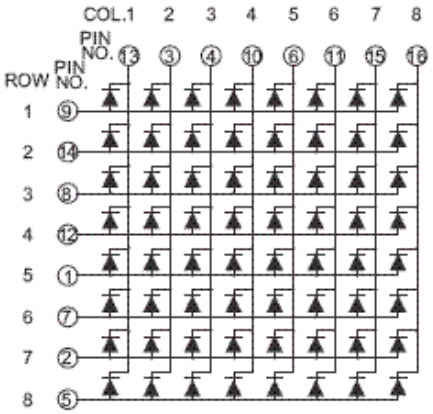

Figure 2: 8x8 LED Matrix Schematic

An LED dot matrix display ("dot" refers to the circular lenses in front of the LEDs) can also come with multiple LEDs of varying colors behind each dot in the matrix. For example, the matrix used in this project has a Red, Green and Blue LED behind each dot in the 8x8 grid. A configuration with multiple LEDs behind each dot adds another control pin to every column (positive terminal) for each additional color of LED, while the rows (negative terminals) are still all connected together. Therefore an RGB matrix has 32 control pins compared to the 16 pins seen in Figure 2.

Controlling the LED Matrix

Since all of the individual LED's in a matrix share their negative and positive terminals in each row and column, it is not possible to control each individual LED at the same time. Instead, the matrix is controlled by cycling through each row very quickly while triggering the correct column pins to light the desired LED's for that particular row. If the switching is done at a quick enough rate, there will be no visible flicker and the LED matrix display will appear to have each LED turned on at the same time. This works because of the principle known as Persistence of Vision, which is the theory that the retina of the human eye retains an image for about a tenth of a second. Thus an LED matrix must be very precisely controlled, with the Rows being scanned through sequentially at a rate greater than about 40Hz (to be safe) while sending out the column data at the exact same rate. This kind of control is most easily accomplished with the aid of a microcontroller, plus some additional components.

Choosing a microcontroller

When choosing a microcontroller that will meet the requirements necessary for controlling an LED matrix, the specifications that must be met are those of the number

of I/O pins available, the amount of current that each pin can source and sink and the speed at which the microcontroller can send out control signals. Another option that one might desire is that of an onboard Analog to Digital Converter for sampling an input signal (such as an audio waveform) which can then influence the control signals being sent to the LEDs.

Modern day microcontroller technology has made the issue of control signal speed and available I/O pins almost trivial for this application. Almost every chip on the market now has upwards of 32 I/O pins available and almost all operate at speeds greater than 10MHz, which allows for control signals to be sent at rates much greater than the 40Hz that is required to satisfy the Persistence of Vision principle. For example, the microprocessor that I chose ran at an operating speed of 16MHz and in the loop of code that cycled through the rows I was still able to put a 2ms delay between each iteration, meaning that there was a guaranteed 16ms of delay in my code (excluding the actual code execution time) and still no visible flicker on the matrix display. Up until recent years, 1-8 decoder chips (which allow the 8 rows pins to be controlled by just 3 I/O pins) were almost always used to minimize the number of I/O pins required on the microcontroller; however, with the large amount of I/O pins available on almost all micros now, these chips seem like an unnecessary complication to the design.

The final specification that must be met is that of current consumption, meaning that the amount of current that each I/O pin can source and sink is greater than the amount of current needed to illuminate a row of LEDs. These numbers must be determined from the specifications sheet of the LED matrix in conjunction with the microcontroller's maximum current values. This specification is also easily met in most situations though, due to the way in which we are switching the LEDs off and on in such a rapid manner. When a typical current specification is given on the LED matrix data sheet (usually around 20mA per LED), the spec is for steady-state consumption (continuously "on"). Since the LEDs are being switched on and off so quickly, our current consumption is actually much lower due to the current draw's continuously transient state. With the LED's fully switching, I observed there to be only

about a third of the current being drawn as compared to the LEDs consumption at a steady-state. To be safe, a Darlington transistor array is almost always used in between the cathodes of the LEDs and the microcontroller I/O Pins. The Darlington transistor array is an IC that is an array of 8 grounded NPN transistors which serve as a current sink, enabling more current to be drawn through the LEDs safely. The array has a dual purpose in that it also inverts the signal on the pins, so when we send a positive control signal, it is actually completing the Ground connection as is needed to turn on the LEDs.

Bias resistors must also be calculated and inserted in between each I/O pin on the microcontroller and the anode pins on the LED matrix. Bias resistors are calculated by referring to the Electrical Characteristics table on the LED matrix's data sheet. Each differently colored LED has a voltage that it prefers to be biased at for correct operation, somewhere usually between 2 – 5 Volts. The voltage on the I/O pins of most microcontrollers is a standard 5 Volts, so using Ohms law with the desired voltage and current specs will yield a correct bias resistor value for each column of LEDs.

Putting it all together

With all of the specifications met, now comes the fun part of wiring the whole matrix up. This involves fabricating many cables as the RGB matrix requires 32 control signals, which first have to go from the microcontroller to a board with the bias resistors and Darlington transistor array, and then from there to the 32 pins on the back of the matrix.

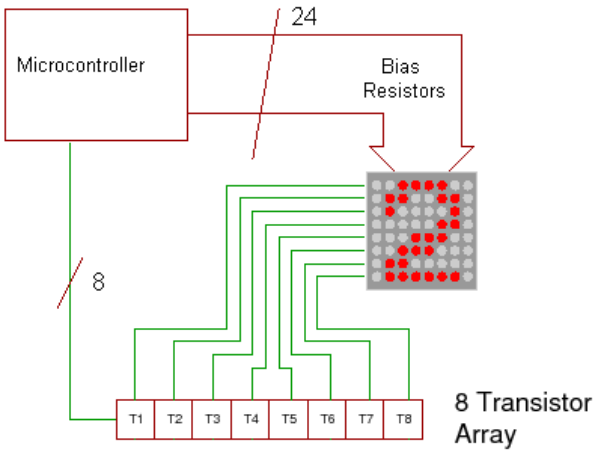

Figure 3: Block Diagram of Connections

The Atmel microcontroller used in this project has its I/O pins grouped in Ports with 8 I/O pins each plus a Vcc pin and Ground pin. Thus I chose to assign one port for controlling the switching of the 8 row pins, one port for the 8 Red Column pins, one for the 8 Green pins and one for the 8 Blue pins. This separation of colors, rows, and columns also makes the programming

more simple and intuitive. The most time consuming part of wiring up the matrix is mapping the pins from the microcontroller ports to the PCB mount pins on the back of the matrix. This turns out to be quite complex because the 32 pins on the matrix are in a seemingly completely random order and must be correctly assigned to pins 0-7 on each port of the microcontroller. Table 1 provides the pin mappings with the matrix pin numbers in bold and the corresponding microcontroller port letter and number directly below.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
A0	F0	E1	A2	F2	E3	B7	B6
9	**10**	**11**	**12**	**13**	**14**	**15**	**16**
B5	B4	A4	E4	F5	A6	E6	F7
17	**18**	**19**	**20**	**21**	**22**	**23**	**24**
E7	A7	F6	E5	A5	F4	B3	B2
25	**26**	**27**	**28**	**29**	**30**	**31**	**32**
B1	B0	F3	A3	E2	F1	A1	E0

Table 1: Matrix Pin Numbers in Bold, corresponding microcontroller Port Letter and number directly below.

These can be used in conjunction with the matrix data sheet information (links to this found in the References section), which shows the pin orientation on the actual matrix and which pins correspond to which LEDs in each row and column. After correctly making all of the connections, we are then ready to write a test program and begin testing the board.

Programming the Microcontroller

The Atmel ATMEGA128 can be programmed using BASIC, C, or Assembly language. After the code is written, it is compiled using the WinAVR compiler which outputs a Hex file that can then be directly downloaded to the microcontroller. The Hex file is downloaded via an In-Circuit Programmer that connects the microcontroller to the parallel port of a PC.

Figure 4: In-Circuit program download tool

The Hex file must be downloaded through use of special Serial Device Programming software such as the freeware program PonyProg2000. The Atmel evaluation board I purchased from futurlec.com came with extremely easy to follow tutorials and example programs that showed step by step how to compile and download the files

to the microcontroller. Any series of microcontrollers no doubt comes with extensive documents on setting up and programming one for the first time, so I will not into further detail of those processes.

There are a variety of IDEs available for developing microcontroller programs depending on the language that is desired and the chip architecture being used. For Atmel's AVR family of chips, Atmel provides a free Assembly Language IDE called AVR Studio 4 which can be downloaded from www.atmel.com. For programming in BASIC, a demo version of BASCOM-AVR is available from MCS Electronics at www.mcselec.com. The demo version's only limitation is that of a 4kB maximum source code file size. For programming in C there is CodeVisionAVR, available in a demo version with a 2kB file size limitation at www.codevision.be. A good free C/C++ IDE is called Code::Blocks and is available at www.codeblocks.org. Code::Blocks already has preconfigured settings for using the WinAVR compiler, so it is a very good choice for a free, fully functional IDE.

The first program used to test the setup should be very simple and test every LED in the array. Figure 5 shows an example program that demonstrates how to turn on every LED in the matrix, changing which color LEDs are lit at even intervals. Note the infinite while loop, since we just want the microcontroller to run this routine for as long as it is powered up. This section of code would appear in the main() of your C source file. The only other code that would be necessary would be the #include libraries for your specific microcontroller and whatever lines of initialization may be required depending on the application. The code is extrememly simple, with the main 'for' loop simply iterating down the rows of the matrix while simultaneously updating the column data on the Ports of the microcontroller. The delay in between each row must be calibrated depending on the speed of the microcontroller being used. The target value is to use the longest delay possible without being able to see any flickering of the LEDs. The delay is desired to ensure that the LEDs are getting enough current each time to achieve full brightness. The counter variable outside of the for loop allows states to change according to the length of the duration variable.

```
unsigned char ROW;      //Used to hold current ROW values in loop
unsigned int Counter, Color, Duration;
Color = 0;              //Colors: 0 = RED, 1 = GREEN, 2 = BLUE
Counter = 0;
Duration = 50;

while(1)                                        // Infinite Loop
 {
    for (ROW = 0x80; ROW > 0x00; ROW >>= 1)  // Shift Right (Iterates down the Rows)
    {
         PORTB = ROW;           //PORTB is connected to the 8 row pins

         if(Color == 0) {
           PORTA = 0xFF;        //PORTA is connected to the 8 Red pins
           PORTE = 0x00;
           PORTF = 0x00;
         }
         else if(Color == 1){
           PORTA = 0x00;
           PORTE = 0xFF;        //PORTE is connected to the 8 Green pins
           PORTF = 0x00;
         }
         else if(Color == 2){
           PORTA = 0x00;
           PORTE = 0x00;
           PORTF = 0xFF;        //PORTF is connected to the 8 Blue pins
         }

         delay_ms(2);       //Add delay up until flicker is seen for added brightness
    }

    Counter++;

    if(Counter > Duration){      //The Duration decides how long each color is displayed
       Counter = 0;

       if(Color == 0)           //Switches Color Variable to next Color
               Color = 1;
       else if(Color == 1)
               Color = 2;
       else if(Color == 2)
               Color = 0;
    }
 }
}
```

Figure 5: Example Control Program in C

Conclusion

As is apparent, there are many different options when it comes to choosing a microcontroller and LED matrix to work with. It is easiest to choose an LED matrix first and then to select a microcontroller that meets the demands of the LEDs to be controlled. Once the basic setup is complete, the real challenge lies in programming the LED matrix to display interesting patterns. A good expansion is to capture an input signal using an ADC and then modify the display based on the input signal information. With a digitized input signal, the FFT could then be taken and DSP algorithms could be used to map the data to the LED matrix in interesting ways.

On-Line References

Arduino - www.arduino.cc

Open-source microcontroller project based on Atmel AVR family of chips

Atmel - www.atmel.com

Manufacturer of the AVR series of microcontrollers

Best Microcontroller Projects –

www.best-microcontroller-projects.com

Many example projects with PIC micros

Code::Blocks – www.codeblocks.org

Free C++ IDE with built in support for many different compilers

Futurlec - www.futurlec.com

Sells microcontroller evaluation boards and starter kits, as well as LED matrices

Microchip - www.microchip.com

Manufacturer of PIC family of microprocessors

Parallax - www.parallax.com

Sells microcontroller starter and advanced kits

SparkFun Electronics – www.sparkfun.com

Sells microcontrollers and LED matrices

The LED Light - www.theledlight.com

Sells all types of LEDs, has good beginner tutorials on setting up LEDs.

RGB LED Dot Matrix Display

www.foryard.com

FYM-23881ABxxx Manufactured by Foryard Optoelectronics

Data Sheet

http://www.sparkfun.com/datasheets/Components/FYM-23881ABxxx.pdf