# **Programmable LED Signs for Campus Buses**

**Benjamin Cizdziel** 

University of California, Santa Cruz Inter-Networking Research Group Faculty Advisor: Katia Obraczka Graduate Advisor: Kerry Veenstra

### I. Abstract

Our goal with this project is to design programmable LED route signs for UCSC campus buses. The sign design involves an LED matrix used to display the route characters and a microcontroller that controls which rows and columns of the matrix are turned on. Additionally, we processed font files to generate data that is used to drive the correct LEDs to display the text. We have built a working miniature prototype, which validates our design approach. Our next step is to develop a full-size prototype. In an effort to reduce the power consumption of the sign, we will use DC step-down converters to power the LED display. This project is part of the UCSC Inter-Networking Research Group's Bus Tracking System project, whose major goals are to provide real-time campus bus location maps on the web along with an ad-hoc networking test bed. In the future, the microcontroller used to control the LED sign will also wirelessly transmit GPS route data to campus base stations.

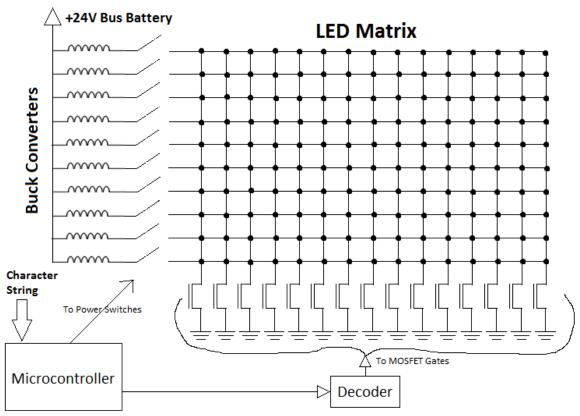


Figure 1: LED bus sign design diagram.

## II. General Design

Figure 1 depicts the overall design of the LED route sign. The LED matrix displays the text of the route name. Appropriate LEDs are turned on to show each character on the matrix. Locations on the matrix can be identified by its rows and columns.

The entire system is powered by the bus battery, at either 12 or 24 volts (choice between the two). This battery powers each LED in the matrix, the microcontroller, and the decoder.

Each column of the LED matrix is connected to the drain of an NMOS transistor whose source is grounded, to be used as a switch. Additionally, the bus battery is connected to a buck converter, which is a DC step-down converter (explained in Section V), and through a switch to each row. These row switches will most likely be MOSFETs, but this is still being deliberated. To power an LED, its row switch must be closed and its column transistor must be turned on. This completes the circuit: from power to the buck converter and through the LED to ground.

A microcontroller is used to control the row switches and column transistors, thus controlling which LEDs are turned on. It determines which LEDs will be displayed based on a character string input.

A set of pins from the microcontroller that counts from 0 to n bits is connected to a digital decoder, which outputs one "high" voltage signal at a time and shifts it among its  $2^n$  outputs. Each decoder output is connected to the gate of its corresponding column transistor. Whenever an NMOS gate gets a high signal, it turns on, giving its column a path to ground. Therefore, due to the decoder output, the columns turn on one at a time in order, continuously cycling.

As the columns are cycling on one at a time, microcontroller pins close the appropriate row switches to turn on the proper LEDs for displaying the characters. Each column gets its own set of row data. A major reason for cycling the columns instead of having each column always on is to reduce the amount of row switching pins needed from the microcontroller. Since the columns are cycled, the same row pins can be used with their data changing for each column. The columns will be cycled fast enough so that each column appears to be on at all times, with no flickering of text.

The actual sign will use multiple modules of these cycling column sets, which will all be working at the same time. This is to keep the LEDs bright enough to see clearly. The more columns there are in a cycle, the dimmer each LED is, since its percentage of on time (duty cycle) decreases.

## **III. Font Processing**

A font is necessary to display the text on the LED sign. We found a Python program called "dewinfont.py" that we used to help process the necessary font data. This program requires a .fon file (a common bitmapped font format) input and outputs general information about the font as well as data for each ASCII character. Figure 2 shows the data that is output for each character of the .fon file: the character number, width in bits, and outline (with 1's filled in and 0's unfilled). The font that we have chosen to use thus far is System Bold, which is the same one illustrated in Figure 2.

		i
char 65	char 66	char 67
width 8	width 10	width 9
00000000	0000000000	000000000
00000000	0000000000	000000000
00000000	0000000000	000000000
00011000	0111111100	000111100
00011000	0110000110	001100110
00111100	0110000110	011000010
00111100	0110000110	011000000
00100100	0111111100	011000000
01100110	0110000110	011000000
01100110	0110000110	011000000
01111110	0110000110	011000010
11000011	0110000110	001100110
11000011	0111111100	000111100
00000000	0000000000	000000000
00000000	0000000000	000000000
00000000	0000000000	000000000

Figure 2: System Bold font data to be processed.

For this output data to be useful to the LED sign project, it needed to be processed and formatted in a way that the data could be easily accessed by the main microcontroller program that controls the sign. To do so, I wrote a C program called "fontproc.c". This program requires an input of the text file output of "dewinfont.py" (from some .fon file) and automatically creates the "fontdata.c" file when run, which contains relevant data of the processed font. This data includes a constant for the height of the characters in bits (same for all characters in a font), an array containing each column of character outline data converted into binary integers, an array of structs containing the index of each character in the column data array and the width of each character, and more. The number and set of characters that are processed can be specified in "fontproc.c".

Each part of a character's outline from the Python program's output data directly corresponds to an LED on the sign. A '1' represents an illuminated LED while a '0' represents an LED that is turned off. This functionality is created in the microcontroller program, which is explained in the next section (IV). The "fontdata.c" file is listed as an included file for the preprocessor in the microcontroller program so that its data can be readily accessed for use in controlling the sign.

## **IV. Microcontroller Programming**

We are using the Atmel AT32UC3A0512 32-bit AVR microcontroller for this project. The development board that has been used with it is the Atmel STK600 (Figure 3). However, for the final product, a custom board will be designed. As noted in the previous section, the "fontdata.c" file is included in the main microcontroller C program, which is called "LEDsign0512.c". The microcontroller program was developed and runs with AVR Studio 5. The current general tasks of this program will be discussed.

The STK600 GPIO (general purpose input/output) is used to control which columns and rows of the sign are turned on. One set of pins is

sent to the decoder to cycle the columns and many other pins are given row data for each column from the column data array in "fontdata.c". The flow of the main LED sign program loop follows. First, the GPIO is cleared. Then, a column is turned on and the row GPIO is set based on which column is turned on and what text needs to be displayed. Finally, the column count is incremented and the process repeats.



Figure 3: Atmel STK600 development board with mounted Atmel AT32UC3A0512 microcontroller.

The sign displays whatever the 'route' string is set to. If the string changes, the sign will change to display the text from the new string. This interaction is controlled by the USART (universal synchronous / asynchronous receiver / transmitter) and the INTC (interrupt controller) of the microcontroller. When the USART receives characters, the INTC activates and interrupts the main loop to process and store the characters in a string. When the string is completed, the 'route' string is replaced by the new string and the sign begins displaying the new text.

Each time the route string changes, calculations are made for the new text to be displayed. These calculations include finding the size of the string in LED columns and the starting and ending column positions on the text, which are used to center the text on the display. Data for each character of the 'route' string is copied from the column data array into another array that is used to set the GPIO controlling the rows on the LED matrix. All of the data used for these calculations comes from "fontdata.c".

## V. Buck Converters

Buck converters are high efficiency DC to DC step-down converters and will be used to drive each row of LEDs on the LED matrix. They are being used to reduce the power consumption of the LED sign. The simple alternative to using converters to drive the LEDs is to place a resistor in series with the bus battery (power supply) and the LED load. However, this would waste a large amount of power, as there would be a large voltage drop across the resistor.

The circuit diagram for a buck converter is shown in Figure 4. It consists of a power supply, an inductor, a diode, and the load. In the LED sign, the supply is the bus battery (12 or 24 volts DC) and the load is one LED, since the columns turn on one at a time. Approximately 2 volts are needed to drive (turn on) the amber LEDs that we are planning to use. Consequently, the buck converter will be stepping down the bus battery voltage (12 or 24 volts) to the 2 volts needed for the LED load.

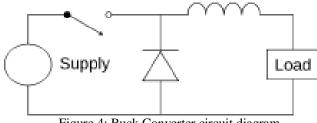


Figure 4: Buck Converter circuit diagram.

Simply put, the buck converter is a switching-mode power supply. It is controlled by opening and closing the switch, which charges and discharges the inductor while providing power to the load. When the switch is closed, the power supply is connected in series to the inductor and the load, and the inductor charges up. When the switch is opened, the inductor's stored energy (in its magnetic field) discharges through the load and the diode. The load voltage and current depend on the duration of time that the switch is open and closed, as well as the inductor's inductance.

## VI. Results

A working miniature prototype of the LED bus route sign has been built, and is shown in Figure 5. This prototype follows the same design as explained in Section II, but with a few differences. Instead of the microcontroller controlling the row switches and the LEDs being powered by the bus battery through the buck converters, the mini prototype is powered directly from the STK600 GPIO ports which are connected through resistors to the rows. This is the major difference from the general design, but the full-size prototype and completed signs will indeed use the buck converters and bus battery to power the LED display.

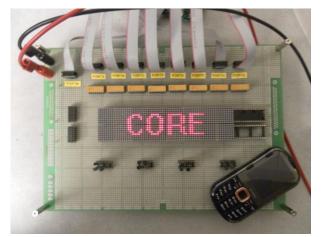


Figure 5: Miniature LED sign prototype running with System Bold font. Cell phone shown for scale.

The mini prototype contains all other facets of the general design, including the column transistors and decoder. Its LED matrix contains 16 rows by 64 columns, but the row and column count of the final sign design is subject to change. 10-pin connector cables are used to connect the STK600 GPIO ports to the prototype board, which can be seen at the top of Figure 5. Ports A to H are row drivers for the LEDs. Port M is connected to the decoder to cycle the columns.

Instead of cycling through each of the 64 columns one at a time, the columns were divided up into four groups of 16 columns. The columns in each of these four groups are cycled at the same time. We chose to split up the columns into groups so that the LEDs on the prototype would be bright enough, since the more columns there are in a cycle, the dimmer the LEDs appear. This was explained at the end of Section II. The final sign design may consist of even more modules of column groups so that it is bright enough to be used on the campus buses.

The USART was configured with the mini sign prototype, and connects by serial port from a PC to the RS232 port on the STK600. To set up the USART, a terminal for the appropriate serial port on the PC should be configured to the correct baud rate (which is set in "LEDsign0512.c"). When the microcontroller program is run, a welcome message is printed to the terminal. Typing characters into the terminal and pressing return will display that text on the mini prototype, with lowercase letters converted to uppercase.

STK600 switches 0 to 6 were also configured with the mini prototype to perform different tasks. These include cycling between preset route names and enabling delays to slow down the sign display, allowing the column cycling to be seen by eye.

## VII. Future Work

Our next immediate step in development is to implement a full-size LED bus sign prototype that includes the buck converters. This will involve choosing the parts and LEDs, as well as designing the printed circuit board to be used. The LED count, sign dimensions, and font type also must be finalized.

Additionally, we will build a control panel and sign programmer. The control panel will allow the bus driver to cycle through preset route names and also select when the bus is out of service. Two buttons (one for cycling routes and one for out of service) and a small screen will be on the control panel. The sign programmer will attach to the control panel and will allow any route name to be programmed into the control panel.

The microcontroller that is used to control the sign will also be integrated with the research

CPU of the Bus Tracking System, which communicates with the GPS and serves as a mobile ad-hoc wireless testbed node for i-NRG's research.



Figure 6: UCSC i-NRG's Bus Tracking System logo.

Furthermore, the microcontroller will be set up with a 900-MHz radio to transmit the route name, bus ID, and GPS location coordinates to campus base stations every few seconds. It will automatically transmit whatever route name the LED sign is displaying. EEPROM memory will also be utilized with the microcontroller to store the route name when the bus is turned off, so that the LED sign will display the same route name when the bus is turned back on.

#### VIII. Acknowledgements

I would like to thank the UCSC Inter-Networking Research Group (<u>inrg.cse.ucsc.edu</u>), specifically Kerry Veenstra and Professor Katia Obraczka, for their contributions to this project.

This research was sponsored by the UCSC SURF-IT (<u>surf-it.soe.ucsc.edu</u>) Research Experience for Undergraduates Program, NSF grant No. CNS-0852099.

dewinfont.py:

http://www.chiark.greenend.org.uk/~sgtatham/fonts

Buck Converter Wiki (image used): http://en.wikipedia.org/wiki/Buck\_converter