Atmel AVR Control of an Optrex Graphics LCD

THIS PROJECT WAS STARTED WHEN WE ORIGINALLY NEEDED A WAY TO GRAPHICALLY DISPLAY A TEMPERATURE PROFILE FOR A REFLOW OVEN. WE DETERMINED THAT OPTREX HAD A REASONABLY SIZED, VALUE PRICED LCD THAT COULD BE USED FOR A PROJECT LIKE THIS. WE ALSO HAD EXPERIENCE WITH USING ATMEL AVRS FOR SOLVING DESIGN PROBLEMS THAT REQUIRED THE USE OF A MICRO. IT SEEMED THAT THE COMBINATION OF THE OPTREX DISPLAY WITH AN ATMEL AVR WAS A VIABLE, COST EFFECTIVE SOLUTION FOR SHOWING TEMPERATURE ON A DISPLAY.

By: Bob Proctor, Tim Davis, Alan Helfinstine

Project History

We have done four different revisions of PCBs for this project as we have constantly refined what we have. A picture showing the display mounted on the PCB is given in Figure 1. The goal has been to build something that not only we will use but others could use as well. The project initially was started by the three authors emailing each other and refining what we thought would be needed for this design. Each of us works remotely from each other so we try and be concise with our design recommendations. After several email iterations, a set of requirements was written and we were ready to start. We went ahead and ran the first revision of the PCB. The system worked on the first revision. The following revisions were used to go after the form factor by making the component side low profile and to use less expensive components in the design.

A 9-pin RS232 connector and serial communication level translator IC were used to allow the LCD to be used as an appliance where graphic patterns could be built and downloaded to the display. In our initial application for the reflow oven, the key temperature points along the profile versus time graph could be downloaded.

The schematic shown in figure 2 shows that a lab supply is used for the LCD biasing. A set of externally generated on board power supplies can also be built or one can use the charge pumps built into the LCD. Our pc board left vacant component pads for the charge pump caps to be installed if we ever wanted to use this power mode. PCB pad arrangement allows for both 1/7 and 1/9 display contrast options. The part of the circuit design that provides the 1/7 or 1/9 bias is the voltage divider between +5V and V- as shown in the schematic. The divider taps supply bias to the LCD. Our design used the 1/9 contrast option.

Figure 1: LCD controller in operation

Details in the Hardware

Some of the key features of the hardware include being able to drive the LCD display in either parallel or serial communication. Work done for this article used parallel communication. Options were left in the circuit in order to allow for communicating with the display using serial communication.

Two switches were used as input in order to change between different display screens that were designed. A red reset switch was also provided. A 6-pin 0.1" connector using the Atmel programming tools standard pinout was used for ISP access. The intensity of the backlight can be controlled by using a PWM signal

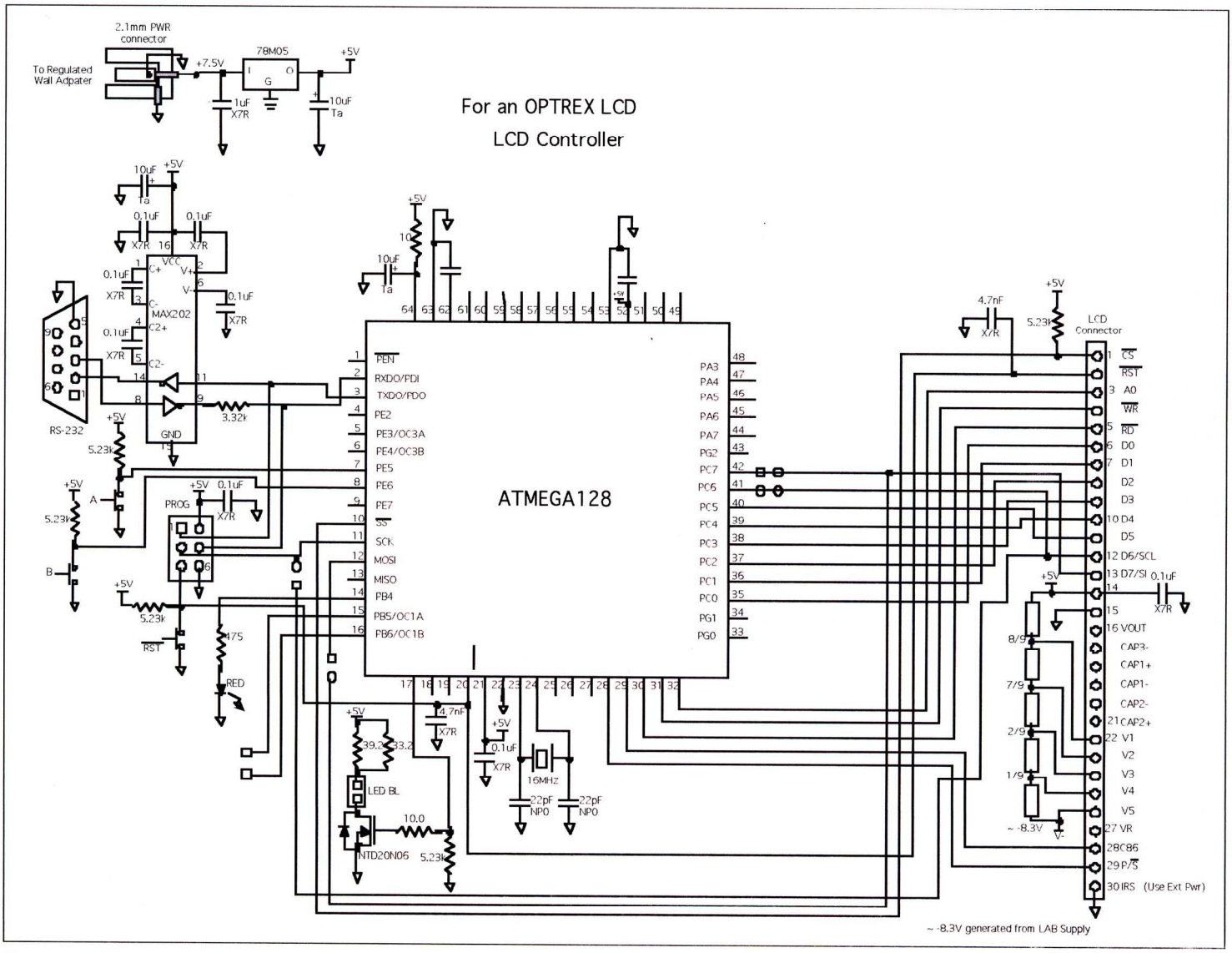

Figure 2: Schematic of LCD driver PCB

controlling the gate of a n channel mosfet which turns on the LED backlight for the LCD.

The power supply for the system was derived from a 7.5V wall adapter that powers a 5V regulator. An external 16 MHz crystal was used for the system clock. A red LED was also provided that allows for either a "power on" or a "microprocessor is alive" indicator.

The system used a rotary backlock 30-pin zif socket to plug in the flex cable from the LCD. This allowed for both Optrex models F-515X and F-518X graphic LCDs to be tested and shown to work with our system.

PCB pads were provided for an optional breakout connector to allow for easy direct connection to the cable of the LCD. This allowed for monitoring or for driving the LCD signals. This breakout connector can be used for logic analyzer or oscilloscope access.

Atmel AVR Studio was used to design the code for this project. Both a STK500 and an AVR ISP MKII were used to program the controller project through the ISP connector. Boards were built with either the Mega128 or the Mega64 AVR installed on them.

Details in the Software

We planned our software graphics functions to fit the way the Optrex module handles the data. It takes data one byte at a time, to represent dots in rows that are 8 pixels high. 128 bytes are needed to fill in a horizontal line of 8 dots high by 128 dots wide. Optrex refers to this as a page. There are 8 pages to be written to the screen to fill in all 64 vertical dots. As it turns out, the number of bytes to represent all the pixels is 128 horizontal dots X 8 pages X 8 dots = 1024 bytes of data. We used SRAM in the AVR as a two-dimensional array to store all the screen data as a screen (full-page) buffer. The idea is that we could take our time filling in the screen data into the buffer, then quickly update the screen from the buffer, keeping screen transitions very clean.

Once we had our screen buffer in place, we wrote routines to be able to draw individual dots on the screen, addressed by dot rows and columns. Once that worked, it was easy to add another function to draw lines, using the dot drawing routine. All the graphics functions build on the basic dot drawing routine.

To allow fancy overlapping, we allowed the dot drawing routine to do either set a pixel, clear a pixel, or exclusive OR a pixel. That functionality was passed upwards into the other functions that used the dot drawing routine.

In order to show text, we had to build a font and find a way to convey that to the screen in different rotations. Since the Optrex version of a page is 8 pixels high, we chose a font that is also the same height. Actually, we

successfully did two different fonts, 5x8 and 8x8. The classic 5x8 font is actually 5 pixels wide, plus a space between characters. And the true height of each character is only 7 pixels, leaving a blank line at the bottom to prevent smearing in the character below. Still, the pixels were created in a 5 byte form, each byte being a vertical 8-bit slice of the character. We wrote a character drawing routine which fit the font into places that are 6 pixels wide. The routine allows us to draw the 5 bytes anywhere on the screen according to a character position rather than a pixel position. This makes it fit much better into the displays 8-bit high format. Once the character drawing function was created, we built on other functions to print strings.

To draw text rotated by 90 degrees required some tricks as to how we read the data from the font file and wrote it into the buffer. Once we solved this, it worked just as well as the standard font. We only did this for the 5x8 font, but the same pattern could be applied to others.

As we mentioned, the screen buffer was 1 kbytes long; we needed a routine to takes these bytes and move then onto the screen quickly. Since the data in the array was two-dimensional, and formatted to "look" like the screen data, it was very easy to simply read each row of 128 bytes, send each byte to the screen, updated the screen "page", then continue with the next row of bytes. After 8 rows, the screen update was finished.

Another function we needed was to fill the buffer with data. Typically used for erasing a page by writing all zeros, we gave it the capability to fill the buffer with any byte value. This function is also very straightforward, and moves very quickly in SRAM, so it allowed us to erase a screen, fill in new data, and then update the screen dozens of times per second (faster than the liquid crystal is able to respond).

Another advantage to having our screen data in the buffer is to allow shifting in either horizontal or vertical directions, or both. In those cases, a small buffer was used to copy some of the screen buffer and write it into a new position in the screen buffer. Animated graphics become possible whether the whole screen, or part of it. We tried it for whole screen moving, but we realize that it would work for small areas just as well, for moving icons or sprites.

Having high-level graphics functions that allowed us to draw lines, based on simple begin points and end points, gave us the foundation to do other specialized graphics functions. Line graphs, including smooth sinusoidal lines, can be made as a combination of many small straight lines.

Cool Things That We Found to Do with it

We began to write routines to graph basic functions, and finally to read in A/D data and show it on the screen as an oscillograph. The important factor here was to buffer our samples, then to draw them into the buffer, and finally to update the screen. Doing this, we were able to get very fast samples and nice analog data capture up on the screen quickly. The important thing was to use a consistent sampling clock. If we tried to draw the lines between samples, some samples were delayed, and that messed up our time reference.

Conclusions - Where to Go from Here

As we move forward, we are looking for more applications for this LCD controller. We are also interested in trying other Optrex display sizes besides the one described in this article.

If we move to larger displays and AVRs with many resources, we should be able to do real-time measurements of voltage and enable post processing of these waveforms to display information such as can be found with an oscilloscope or spectrum analyzer.

We enjoy the challenge that these types of projects offer. In the not too distant past, projects such as these weren't possible without the capabilities that the Atmel AVRs now offer as well as the lower cost LCDs that have become available.