Make your own IIC interface to a parallel character LCD

Jon Gabay - August 15, 2012

We have gotten accustomed to seeing Thin Film Transistor (TFT) displays and touch screens on everything from pads, to phones to tablets. While captivating and rightly so, not every application can bear the cost or complexity of color and graphics displays. More times than not, a simple alphanumeric display is all that is needed for applications like lab equipment, dedicated machines, field devices, rack equipment, and low power remote equipments.

Even more, there are times when you just can't use a sexy display. TFT's for the most part still wash out in sunlight. There are super high bright backlights, but they consume a lot of power and generate a lot of heat, even with LEDs and CCFLs.

As a result, character displays which are mostly based on Twisted Nematic and Super Twist Nematic technologies offer a distinct advantage because they can operate in direct sunlight or total darkness. They can operate reflectively (just with ambient or external light), or transflective (can be backlit or use ambient or external light).

To minimize microcontroller I/O needs, more costly serial UART, SPI, and IIC based character displays are available from a few manufacturers. But, by far, the most popular and lowest cost interface uses a parallel bus and control signals to access the display's data and issue commands. With so many manufacturers of parallel modules, you can find very reasonably priced displays and turn them into serial displays.

Here's how to connect the parallel bus alphanumeric displays using a simple IIC serial scheme from your microcontroller. I will also show how you can structure your firmware and drivers to create very efficient data storage of canned messages and display modes.

The parallel interface

Because of a few early interface chips, a pseudo standard has emerged that uses a common electronic interface allowing logic levels to control command modes, data modes, clearing, shifting, and read, and write functions to name a few.

For the most part, two connector styles have emerged, a 14 to 16 pin 100 mil spaced SIP, or a 14 to 16 pin 100 mil spaced DIN (See **Fig. 1 A** and **B**). Note, display manufacturers may switch a pin or pin orientations here and there or change things a bit just to make their displays different. This usually means it becomes a sole source part. Clever pinnouts on your PCB can accommodate different manufacturers.

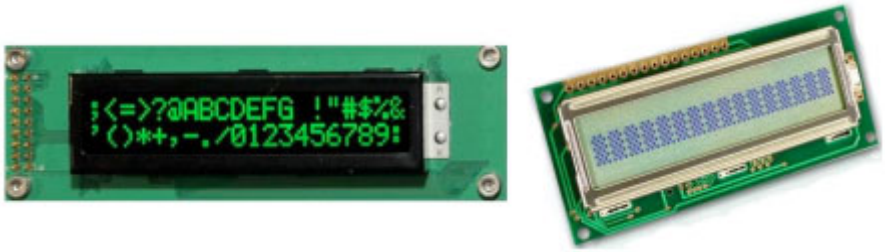

Figure 1 Although the data and control signals are the same, the two pseudo standard parallel interface styles for character displays are in a DIN style (left) or SIP style (right).

The parallel interface uses a combined Enable and Clock function pin (E) that strobes data in or out of the display on a falling edge. A control bit (RS) determines if a command is being written to the display (L), or if data is transferred to the display(H). A R/W signal determines if you are reading (H) or writing (L) to the display (See **Fig. 2** below). There is an 8 bit data bus, as well. Note, an external trimmer or resistance divider can be used to set or adjust the display's contrast.

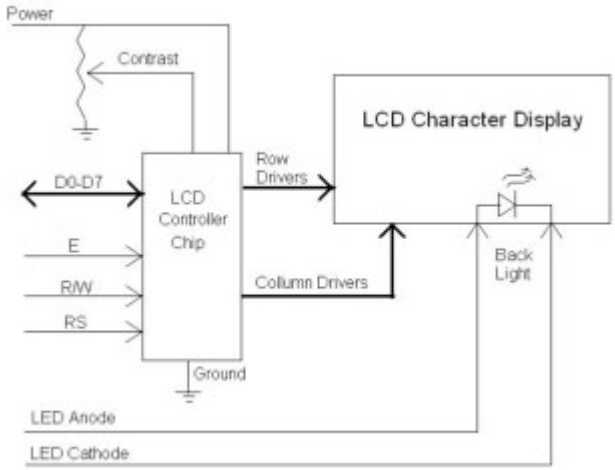

Figure 2 The electronic signals to the controller chip embedded in most character displays includes a data bus, read/write, clock, backlight control, contrast control, and a signal to put the display into command mode.

Because ASCII characters map well into an 8-bit wide space, the 8-bit parallel bus has become the most widespread interface when connecting alphanumeric displays. But, using 8 data bits and 3 control bits and another bit for backlight control means 2 ports are needed from your microcontroller, which is typically I/O limited.

Nevertheless, the parallel timing is very straightforward and easy to understand and implement. The operation is set up using the RS signal to determine if you are issuing a command or transferring display data. The R/W signal determines if you are reading or writing. The falling edge of the E signals strobes in or out the data which is on the data bus (See **Fig. 3**).

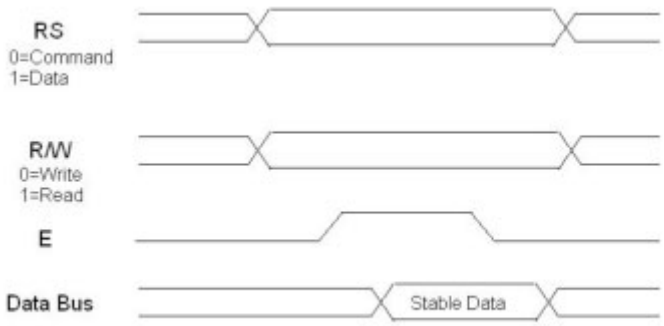

Figure 3 Electronic signal timing sets up the mode and clocks data into or out of the display.

While the 8-bit parallel bus is commonly used, there is another mode which that quite useful. Sort of hidden in the data sheets of most displays is information on the 4-bit mode.

This lets a single 8-bit port completely control and drive a character display including backlight control (See **Fig. 4**). The D4-D7 data bits on the display are multiplexed with the D0-D3 bits. It takes two data transfers to transfer the entire 8 bits, but the display gets the data and is happy.

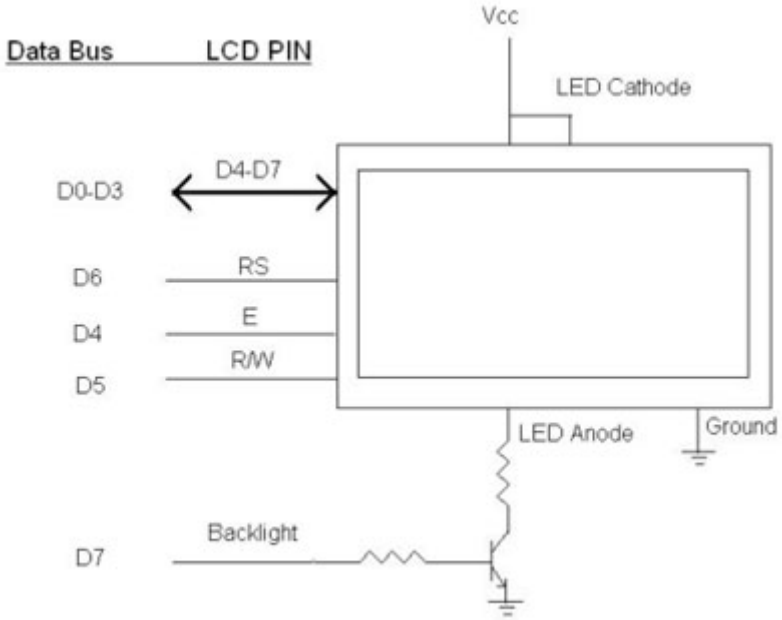

Figure 4 Using the 4 bit mode, I was able to map the entire control of the parallel interface into a single 8 bit port including backlight control through the use of an NPN transistor.

Making it serial
Making it serial

A common serial interface is the Inter-Integrated-Circuit (IIC) protocol, which uses only two wires (Clock and Data) allow chip to chip communications. Unlike a full duplex UART, the IIC interface uses a master and slave approach where a master (the microcontroller) initiates transfers by creating a start bit condition the slave devices listen for. When the clock line is held high and the data line transits low, this signals all the slave devices that an IIC operation is about to take place.

From this point, the master clocks out an address of the chip it wants to communicate with along with a bit indicating if it wants to write to the chip or read from it. Then commands/status or data can be sent back and forth. An acknowledge bit from the slave chip verifies to the master that the transfer has taken place and a stop bit condition from the master ends the cycle (Clock held high while data line transits from low to high).

Most microcontrollers have at least one IIC port implemented in hardware. If not, it is relatively easy to implement a bit bang routine to communicate over any two general purpose I/O lines that can be used to create an IIC port.

To make the display IIC driven, I use the oldie but goodie; the PCF8574. This part is available from TI and from NXP and is an 8 bit parallel to IIC converter chip (See **Fig. 5**). Note, the three address bits (A0-A2) determine what IIC address the device will respond to. This address is encoded into the data stream so that up to 8 PCF8574’s can be connected to the same two wire IIC bus and be

individually addressable and controllable.

There is also an ‘A’ version of the chip which uses a different address. The Non ‘A’ version responds to address 4x. The ‘A’ version responds to address 7x. This allows up to 16 of these devices to connect on the same IIC bus.

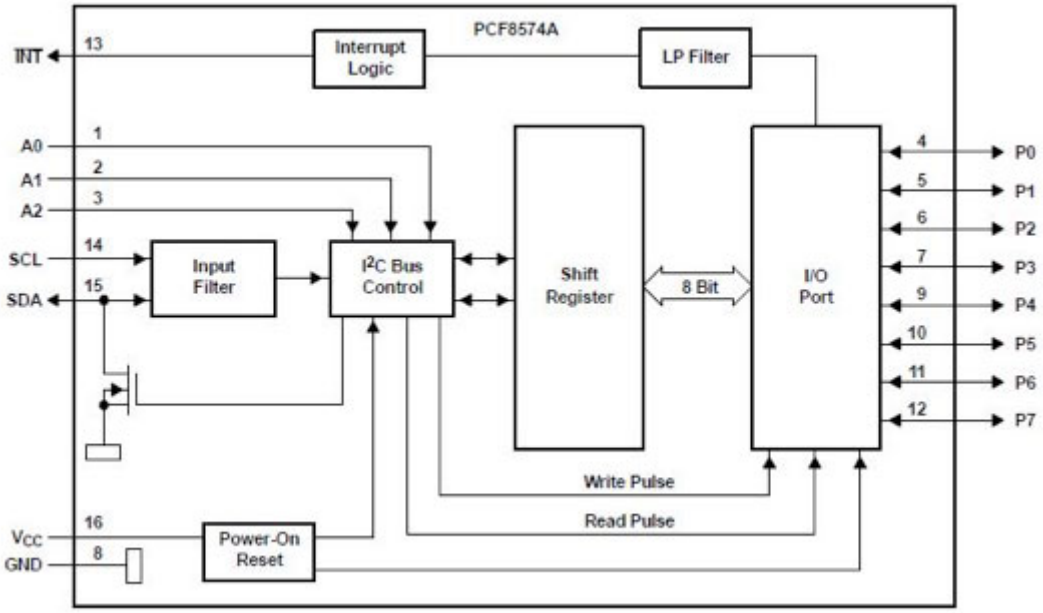

Figure 5 Internally, the PCF8574 converts the serial bit stream to parallel when writing, and visa versa when reading. Note, this part is capable of generating an interrupt when any input signal changes state. We are not using that feature on this project.

By using the PCF8574 as a data pump, it is relatively easy to pump out initialization commands, display mode commands, and write to the display simply by transferring blocks of data. These blocks can be canned prompts and messages, or be constructed in RAM and sent out in one fell swoop.

To do this, I created a notation and data structure that can easily be implemented in firmware (See **Fig. 6**). Note, every IIC device has its own registers and functional map. The PCF8574 is no exception. Refer to the data sheets to see the specifics.

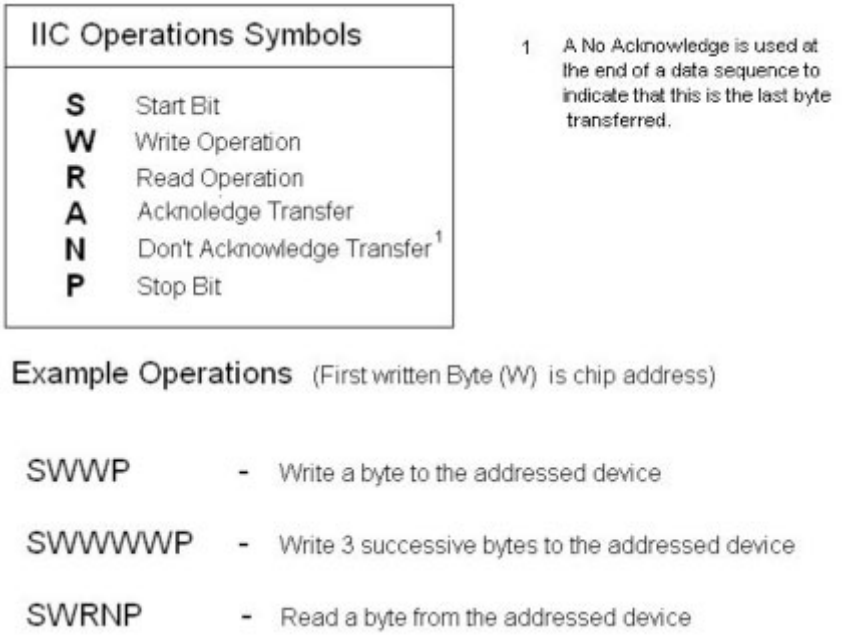

Figure 6 Using abbreviations for the IIC bus states, I came up with a convention which allows my microcontroller to define virtually any data structure that any IIC chip may need. Each chip has its own transfer sequence and commands, but they all use the same states.

To implement this, I used a microcontroller board I designed with a BASIC like language. It runs interpretively so I can store programs in a serial EEPROM (also IIC) and run them through commands over the serial (RS-232) port. The operator syntax is shown in **Fig. 6**.

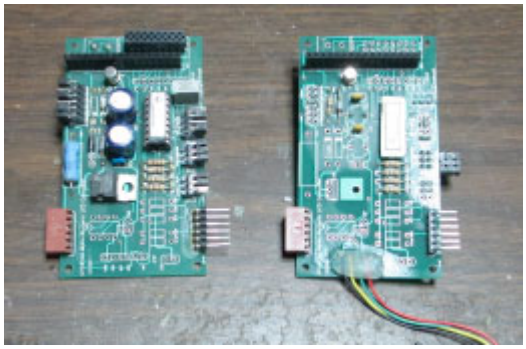

Figure 7 I spun a small PC board to be able to handle virtually any type of display and situation. It has both SIP and DIN display connectors, a contrast timer, address jumpers, pass through expansion wiring, and an optional on board power supply.

I spun a small PC board with both the SIP and DIN connectors so I could use this board for virtually any type of character display. I also added the trimpot for contrast adjustments (See **Fig. 7**).

For hobbyist and do-it-yourself projects, I made everything through hole to make it easier to assemble and debug. I also added redundant connectors on three sides so that these boards could snap together allowing more than one display to connect without the need for an additional cable.

You will note that even though only four wires are needed (clock, data, power, and ground), I made my connectors six pin. One if for interrupts because I may want to have digital I/O like push buttons sharing the same four wire bus. This approach makes a great universal front panel creation

technique. Another pin is a spare -- Always a good idea.

You may also notice that I added the footprints for a local power supply regulator. This way, any one of these boards can hook up to low voltage AC and supply all the others with 5 volts. Jumpers enable supplying power if needed. **Boards tested and ready to use**
With the boards tested and ready to use, I wrote two data pump routines, one for initialization and mode settings, the other for transferring data (See Listing 1). The language is a BASIC interpreter written into the firmware of my microcontroller, here an 8052. Syntax is hexadecimal and operators are basically arithmetic (See **Fig. 8**).

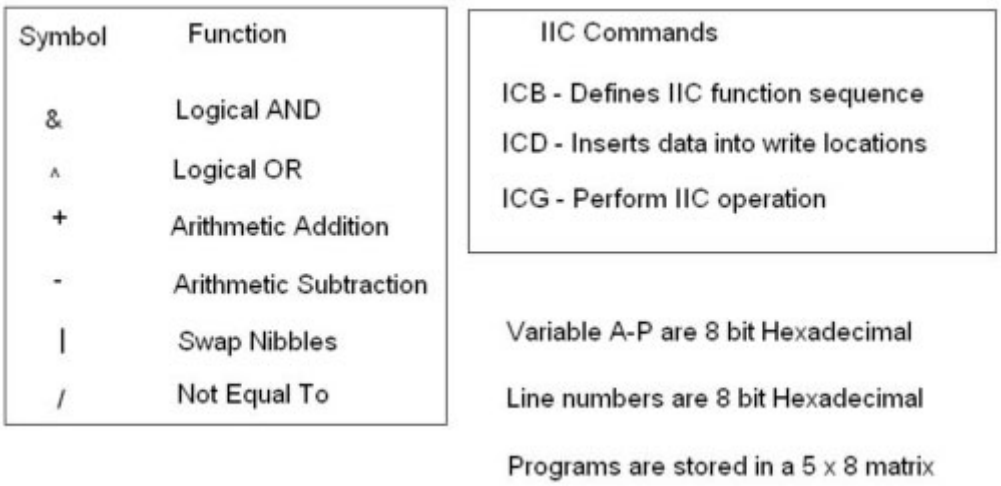

Figure 8 The simple BASIC like language uses standard operators and allows structuring custom IIC command and sequence operations. It made it easy to write data pump routines to control the display and send it data.

The data blocks are stored in a serial IIC EEPROM and a pointer into memory indicates the start address. Note, because values are in bytes but data transfers to the display are in nibbles, the routine breaks each byte up into two transfers. The initialization and mode driver calculates and address from the pointer, reads the value from memory, and pumps out the data to the LCD.

Also note that different displays map their internal RAM differently to the display cursor position. I added initialization routines to set the pointers for a variety of display types like 2x16, 2x20, 4x40, 2x80, etc.

Using this technique, a single four wire bus (Clock, Data, Power, Ground) is all that is needed to drive multiple displays simultaneously, or individually. I first hooked up two similar displays from different manufacturers to two of the LCD interface boards. I set both to have an address of 40, wired them together, and sent them an initialization string and a line of data (See Fig. 9).

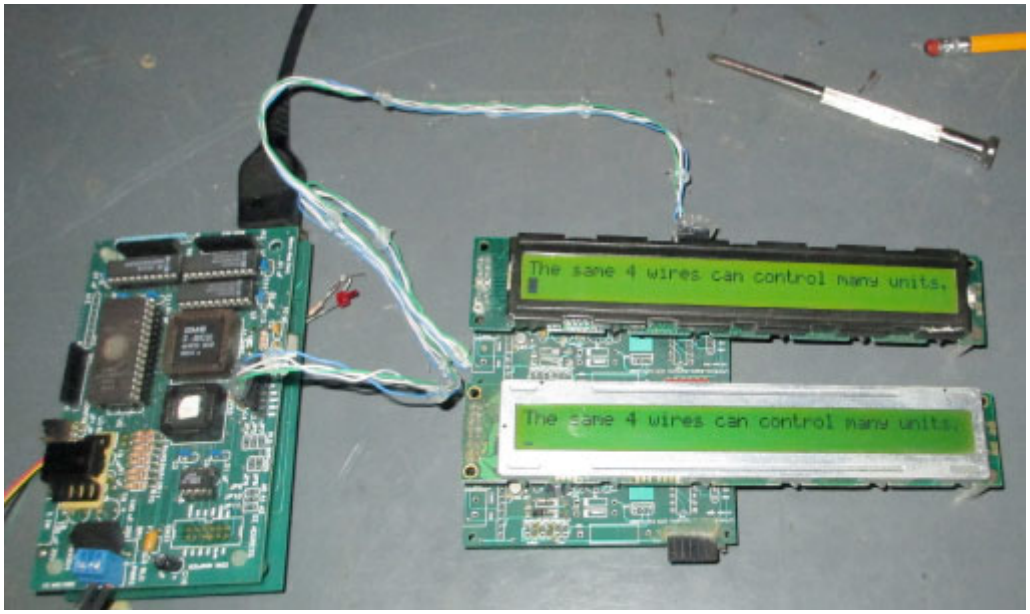

Figure 9 With both LCDs set to the same address, they both responded to a single command/data sequence and updated simultaneously.

I then decided to string four of these together, each with different addresses. I again used four different manufacturers displays of the same configuration (2x40) and sent each address a different data string (See **Fig. 10**).

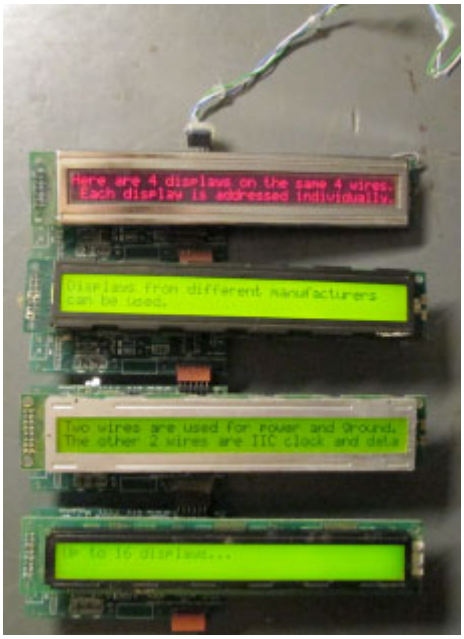

Figure 10 With each board set at a different address, I could send each display an individual and unique message. Not the different qualities and contrasts between different manufacturers. This is why you need to see a display before committing to it in a production design. Note, with the pass through wiring, each board plugs into each other board. No additional cables are needed.

Conclusions

Parallel displays are available from more manufacturers than serial displays. Parallel displays typically cost less and can be second sourced with virtually identical displays from different manufacturers.

Serial displays cost more but simplify cabling, testing, and expansion. Using this technique of driving a parallel display as a serial display gives you the best of both worlds.

Next month, we will add LEDs and Pushbuttons to our serial control bus.

If you liked this feature, and would like to see a weekly or bi-weekly collection of related features delivered directly to your inbox, sign up for the DIY newsletter here.