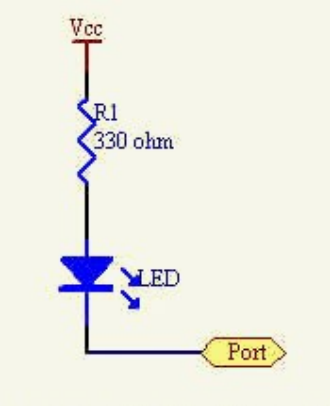
User Interface Ideas

A common task that computers and Microcontrollers need to handle is communicating with a human. On a PC, this is fairly easy to do by typing in some characters at the keyboard, and printing some statements to the screen. On a robot, or other small system, it isn't usually so easy. This article is going to discuss a few ideas about communicating with the user of your project.

Output

There are many different options for outputting information to the user, from blinking an LED to displaying a graphic representation of the data on a CRT. For most of us, the LED approach is more common since we often don't have the capability of carrying or powering a video display. At first glance, the LED approach seems limited to telling us a binary condition. The lights on, or its off.



Interface for an LED

However, if you give the idea some extra thought, you will quickly realize that that single light can actually convey more than the single bit of information. For example, the LED might blink to indicate a third condition. Changing the frequency of the blink can also increase the amount of information. Slow blinking vs. fast blink adds a 4th bit of information. In fact, you can actually increase the information flow quite a bit by blinking patterns. A long blink followed by a short blink, for example, may indicate yet another state. You could even get really extreme and communicate in Morse code. Hey, good enough for Marconi; good enough for me!

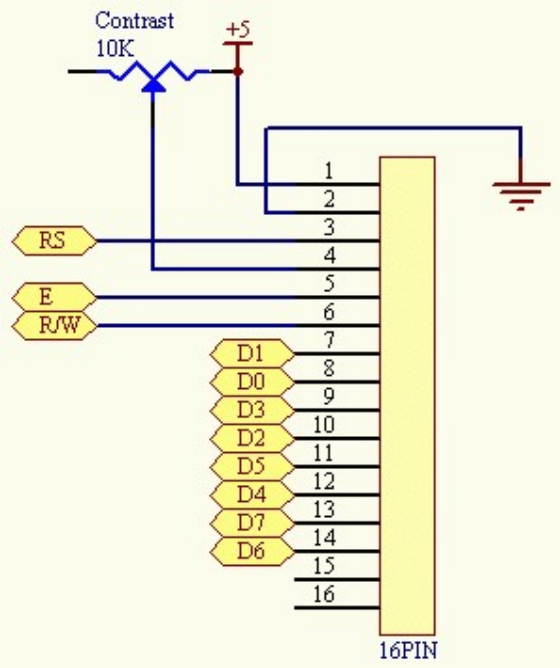
A very nice property of the LED approach is that the circuit is extremely simple, requires very little power, and only one port on the Microcontroller is required. If needed, more than on LED can be used in combination to provide even more output.

Lets assume we want to send some text messages to the user. This is a valuable aid in the debugging process, since you can output messages with data directly to the user. An option here is to use the serial port on the Microcontroller to output the data via an RS-232 port. Upside is its easy. Downside is you need to lug around a PC, laptop, or some other terminal device. It is also difficult to display in the information in a nicely formatted way without getting into specific terminal emulation codes.

Another easy, but often overlooked option, is to output the data to an LCD module. A good LCD module is self contained, has a simple 4 or 8 bit interface, and a decent command set for the onboard processor. You can buy these modules surplus for prices ranging from $8.00 to $25.00, depending on the size. (Check out a recent TIMELINE INC add in Nuts & Volts or other surplus dealers). I have had good luck with Optrex model LCD modules. Specifically, the DMC-40128 (40 characters x 2 lines) and the DMC-20261 (20 chars x 2 lines). I would suggest getting the widest module you can fit into your design and budget.

Most of these modules are based on the Hitachi HD44780 Dot Matrix Liquid Crystal Display Controller and Driver. This driver is basically a integrated controller that accepts input over a 4

or 8 bit parallel interface, and controls the display of graphic characters on the display. There is a data book available from Hitachi (LCD Controller/Driver LSI Data Book) that describes all of the commands and pin outs on how to use these modules. The pinouts of most LCD modules are the same, though you should be sure to get a wiring diagram whenever you order a module (just in case!).



Interfacing an LCD requires only a few lines

As shown in the schematic, the interface circuitry for an LCD is really quite straight forward. The LCD module can be memory mapped directly to the address space of the processor. There are two memory locations on the LCD module. One is the command register, the other is the data register.

The E line is the address strobe. RS stands for Register Select, and determines if the operation is intended as a command or data. The R/W line is for Read/Write.

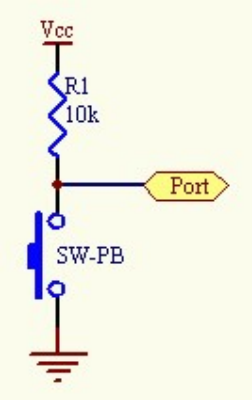
The LCD module can be wired for a 4 bit or a 8 bit mode. In 4 bit mode, D0-D3 are not used. A byte is sent as two successive 4 bit writes on lines D4-D7. This is a good way to save control lines. The entire LCD module can be controlled from as few as 6 control lines. The initialization commands are constructed in such a way that you can tell the LCD module which interface is to be used just after reset.

The best of both worlds can be had by using a Serial Display Module from Scott Edwards Electronics. You can find them advertised in most electronic rags. The idea is that the module accepts serial data in, and does all of the needed translation to drive the 8 bit interface. Quick and easy, though a little more expensive.

An even less expensive route was introduced by Karl Lunt in one of his Nuts & Volts articles. Specifically, the June 1994 issue of Nuts and Volts, Karl explains in great detail what is required to drive an LCD with the SPI of the MC68HC11. It involves using a 74HC595 serial input/parallel output shift register. This reduces the number of control lines to 3 with the addition of a $2.50 part, and a little more software overhead.

Input

The other big task is user input. There are several ways to implement user input. The simple approach is a single push button switch. This allows the user to give a binary value to the project. As with the LED, you can increase the information given by how long the switch is depressed. Depressing it for less than a second might have one meaning, while greater than a second has a different meaning. Unlike the LED example, managing this input can actually be a little complex.



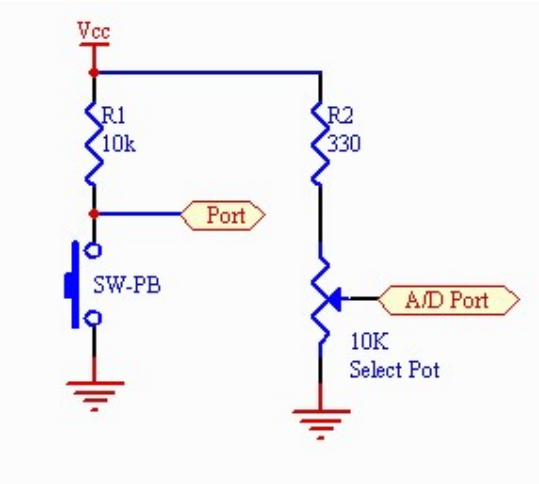
Interfacing a switch

Accepting input from the physical world is always a challenge for the engineer. The world is a truly noisy place. For example, the switch will need to be debounced for it to operate correctly. As the switch makes physical contact, it may electrically bounce, giving more than one state transition. Any switch has this problem. Its easy to deal with in software by using multiple readings of the switch state taken over time (say 100 milliseconds), and deciding that a press occurs only if the switch state remains relatively constant.

Insuring that timing values are adhered to by the human can also be a task. If there are only a couple states, it works out OK. Its easy for a human to press for 1 second or 3 seconds, and get it right. However, even pressing for 1 second verses 2 seconds can be problematic. If you have more than a couple of inputs, you can use multiple switches or even DIP switches to process your input. However, there is a better way.

A Menu System

By combining the output function with an input function, you can put together a more flexible input system. One possible approach that I have used with success is to use a LCD display, a push button, and a single POT connected to an A/D port.



A selection POT and a select switch

When the system wants input from the user, it displays a message on the LCD. It then determines what the current selection is by reading the value of the A/D port. The current selection is printed to the LCD. This is done in a loop so that as the POT is turned, the A/D port value changes, and the selection changes. The exit condition for the loop is the user pressing the select push button.

Note in the schematic a that the A/D port is connected to the sweep on the POT. I also have added R2 as a backup current limiting resistor. This protects the chip against having the A/D port damaged by forcing too much current through it. Some chips allow the A/D port to be a bi-directional. If the port were selected to be output mode, and the pot was turned too far one direction, then the chip could get a fatal dose of current.

Using a linear pot in this circuit is pretty important, or you will find that selecting some menu items is extremely difficult.

Depending on your language of choice, a single function can be written to handle a list of menu options. Basically, the idea is that the function takes as input an array of strings, a count of the number of strings, and returns the numeric value of the string selected. The function is implemented by dividing the number of possible readings on the A/D port by the number of selections in the menu. This yields a value that indicates the range of each selection. Each A/D reading is divided by this range value to yield a zero based selection number. Easy enough! Here it is in 'C':

```
int SelectMenu(char *pszMenu[],unsigned int uCount)
{
        unsigned int uEachSel;
        unsigned int uPrevSel = 257;
        unsigned int uCurSelection;
        uEachSel = 255 / uCount;
        do
        {
                uCurSelection = SampleADPort() / uEachSel;
                if(uPrevSel != uCurSelection)
                {       LCD_CLEAR();    // Clears screen
                        printf("%s",pszMenu[uCurSelection]);
                        uPrevSel = uCurSelection;
                }
        } while ( ! ButtonPressed());
        LCD_CLEAR();    // No longer in menu mode.
        return uCurSelection;
}
```

In Conclusion

Taking the time to put a decent interface on your project can really enhance your experiences. It can also drastically reduce the amount of time you spend debugging problems with your code.