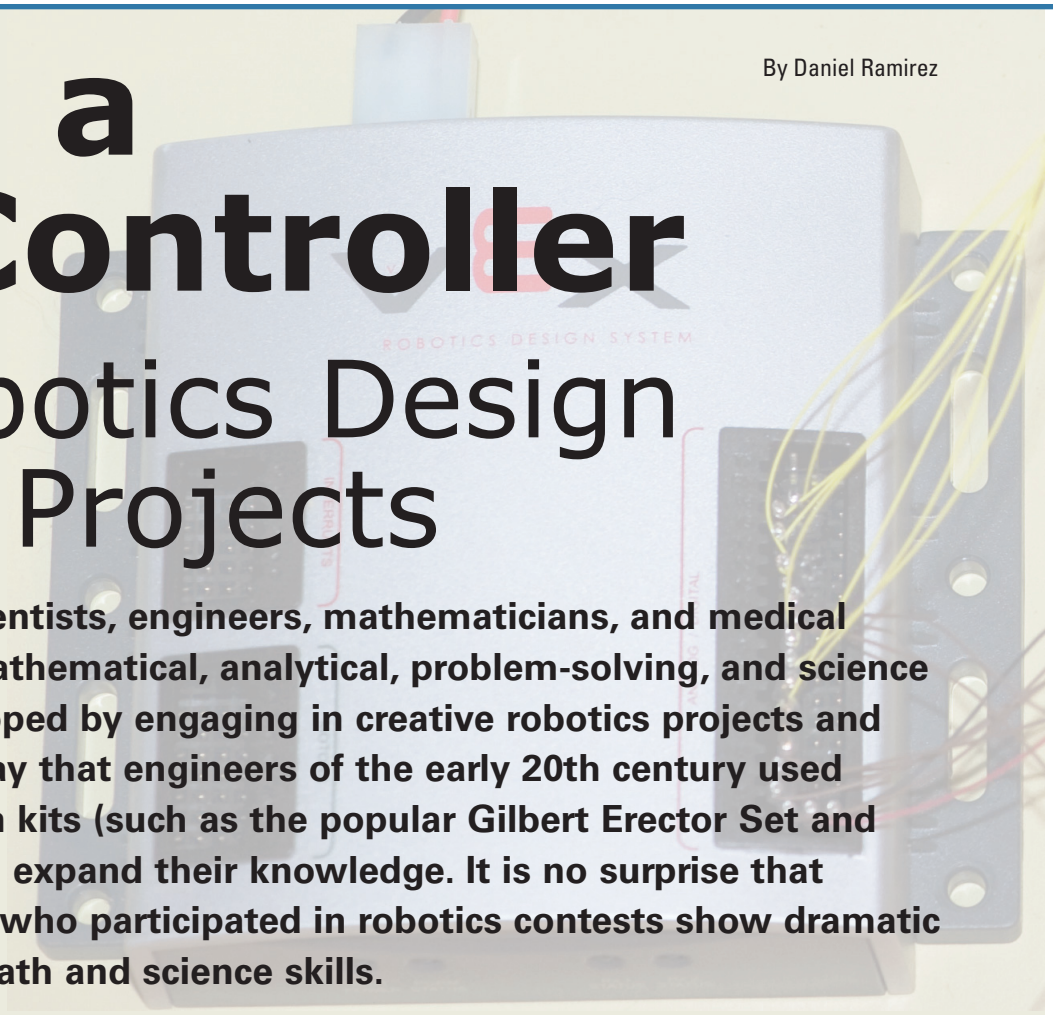
Using a VEX Controller

VEX Robotics Design System Projects

By Daniel Ramirez



Twenty-first century scientists, engineers, mathematicians, and medical researchers need the mathematical, analytical, problem-solving, and science skills that can be developed by engaging in creative robotics projects and contests in the same way that engineers of the early 20th century used mechanical construction kits (such as the popular Gilbert Erector Set and Meccano Erector Set) to expand their knowledge. It is no surprise that students in high school who participated in robotics contests show dramatic improvement in their math and science skills.

One area that has been neglected over the years is the electronics field that once helped to make the US the leader in electronics, computers, and consumer devices such as telephones, TVs, and radios in the 20th century. Now that lead has dramatically eroded due in part to globalization but also due to the lack of course curriculum provided by public and private schools in the US. Electronics courses are generally not available to secondary and high school students.

About the only exposure to electrical theory is in Physics classes. It's really at the college or university level that students can study electronics if they choose. Most students who learn electronics from an early age are either self-taught or have parents in the electronic field that provide assistance to them. Other resources include DIY electronics projects published in magazines such as *Nuts & Volts* and *SERVO Magazine*.

The VEX construction system (which is similar to the Gilbert and Meccano erector sets) is now widely used by both elementary and high school level students who compete regularly in national comptetions like the VEX Robotics Contest and FIRST. These contests have brought together international students from Canada, China, Mexico, Brazil, and other countries from around the world. The VEX microcontroller provides the motor control and also provides feedback from various sensors including bumper switches, limit switches, sonar rangers, IR rangers, and quadrature optical encoders for VEX-based robots. One only has to go to the VEX forum (**www.vexforum.com**) and the VEX Gallery (**www.vexforum.com/gallery/index.php**) to see the hundreds of models featured. Other independent VEX forums exist such as **www.vexfan.com**, which I highly recommend since they also have many models as examples.

The VEX microcontroller is the "silicon brain" that makes an excellent, low cost learning platform that can be used for carrying out both analog and digital electronic experiments, as well as ones for science, robotics, and animatronics (as I have tried to demonstrate in the last three articles). It can be used with the original VEX starter kit, the Vexplorer kit, and additional new kits that IFI now sells. These include educational classroom bundles, along with various robot accessories and contest props.

The VEX Keypad Experiment

In this article, I will show you how to develop a user interface for the VEX microcontroller using the DIY LCD

display information described in my previous article (June '10 issue) with one additional component: an external 4 x 4 numeric keypad. I will detail how you can create your own custom keypad using standard pushbutton switches or even bumper switches and limit switches. I will also describe how I developed a practical user interface written in PIC18 C that demonstrates how you can enter the speed for a motor and special robot commands. It may be used as a terminal or menu-driven user interface for the VEX controller that you can use to enter numeric data necessary to run your applications in a portable manner, while being unplugged from the PC or laptop, using only the VEX microcontroller, a VEX motor, an LCD, and a keypad.

You will be pleasantly surprised that you will not need any other external components other than the keypad, LCD, and pin headers since the VEX controller already has pull-ups and resistors in series to protect the digital input ports even though the schematic shows them. The keypad works together with an LCD display by assigning the I/O pins in such a manner that it works as few pins as possible.

With the information presented here, you should be able to write C applications to make menu selections using a simple user interface so that you can select various robot behaviors without the need for a laptop or PC. Imagine all the VEX applications that could use a keypad as an input device when a laptop or PC is not available. It also opens up the VEX field to rapid prototyping machines such as calculators, vending machines, appliances, etc.

Pushbutton and toggle switches can be used as a convenient way to select various autonomous modes that run with the microcontroller mounted on a robot or prop (without having to use a PC). Pushbutton switches can also be used for menu selections for an embedded user interface. They can be wired to emulate a 4 x 4 keypad, providing up to 16 independent inputs using the same firmware provided with this article.

The pushbutton switches can be replaced with bumper switches or limits switches which is handy to sense objects and to check that mechanical stops or limits are not exceeded when running in autonomous modes.

There are many kinds of switches and pushbuttons. The normally open pushbutton (NO), normally closed pushbutton (NC), the single pole single throw toggle switch (SPST), the single pole double throw switch (SPDT), and others handle low voltages and currents to high voltage home appliance switches (120 volts AC). These switches include momentary switches.

You should also be able to connect up to 16 pushbutton switches and scan their states using only eight VEX I/O pins. If you think of each key as a pushbutton switch and wire it in a similar manner to the keypad matrix, you will be able to connect up to 16 individual pushbutton switches and use our firmware to scan and read their states using only eight I/O pins. Think of all the bumper switches and limit switches that you could sense for your next robot or prop.

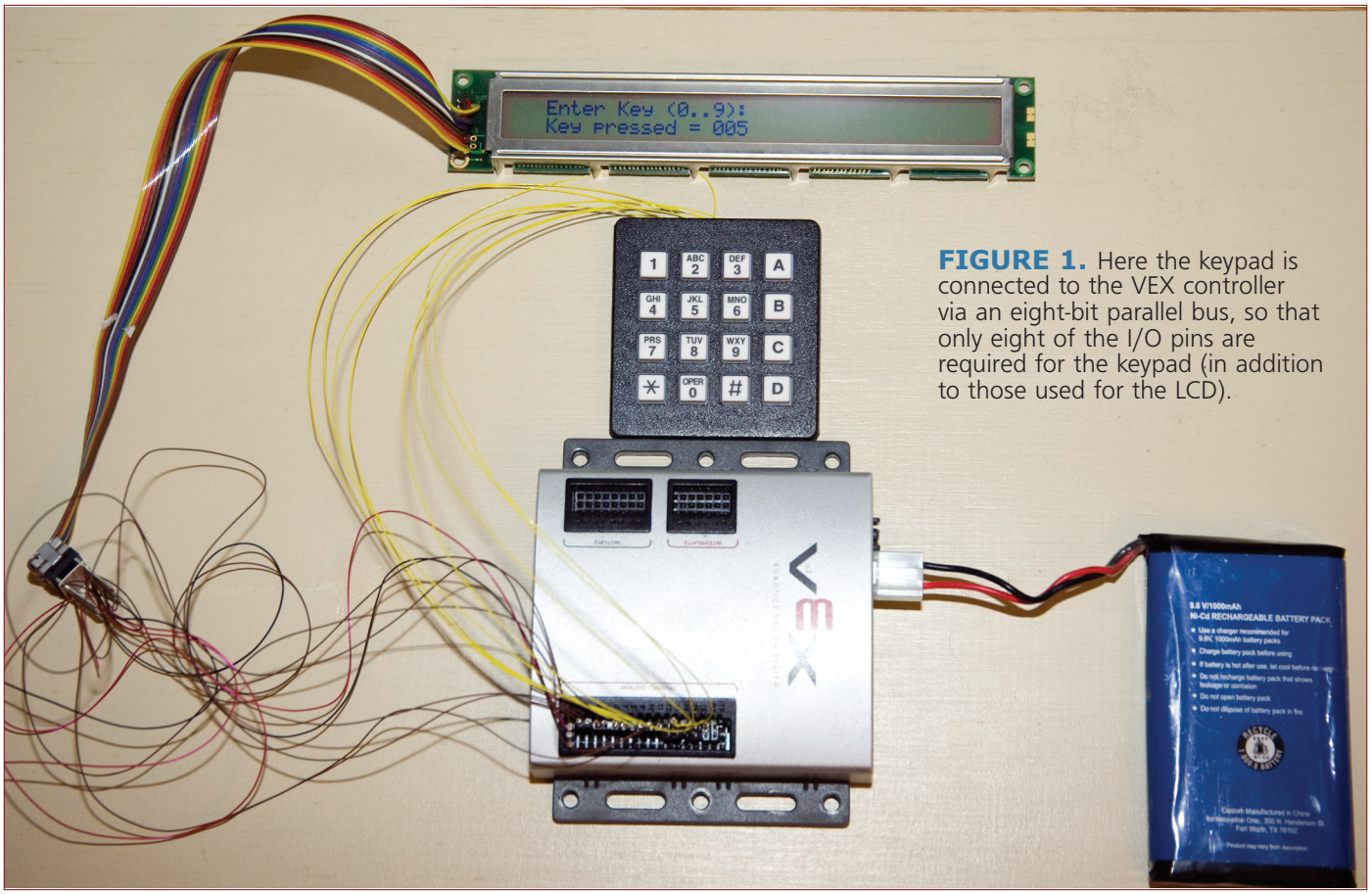


FIGURE 1. Here the keypad is connected to the VEX controller via an eight-bit parallel bus, so that only eight of the I/O pins are required for the keypad (in addition to those used for the LCD).

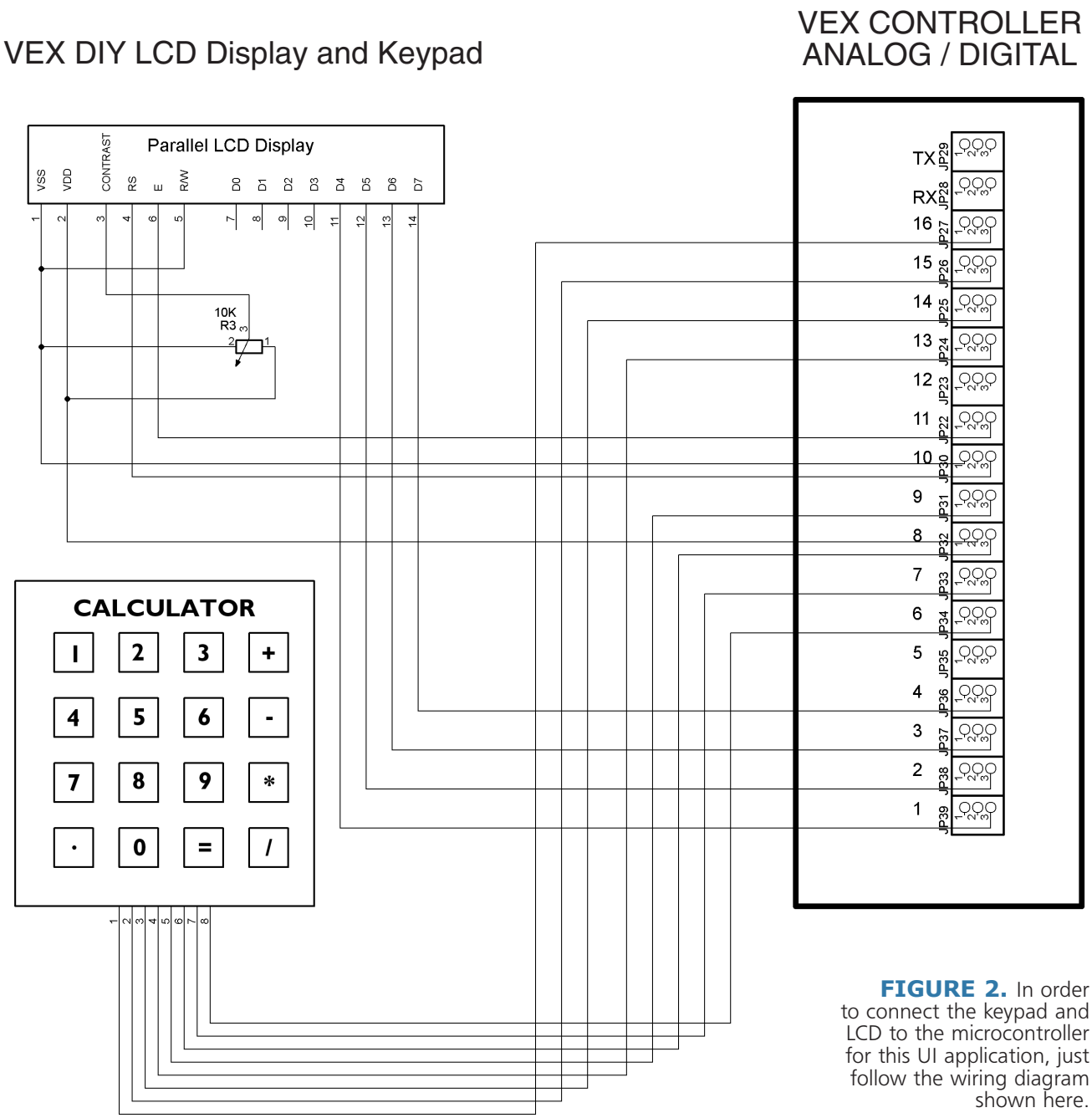


FIGURE 2. In order to connect the keypad and LCD to the microcontroller for this UI application, just follow the wiring diagram shown here.

Keen on Keypads

A keypad allows the user to enter data and commands to an embedded controller or the data entry for the user interface as shown in **Figure 1**. The keypad may also be used to enter text and numeric data like in a calculator application. The keypad and LCD combination makes an excellent portable data input device so robot behavior can be changed on the fly. The keypad is connected to the controller using an eight-bit bus — similar to the four-bit bus used on the LCD display.

Connecting a 4 x 4 Keypad

Connecting the keypad is very simple. The only parts required are the keypad, some pin headers, and wire-wrap wire as shown in the **figure**. Connections to other kinds of microcontrollers usually require four 10K to 470K pull-ups and eight 100 ohm resistors in series (or having weak pull-ups enabled on their input pins if this feature is available). The VEX controller has these components already connected to it internally. Again, our keypad is connected to the controller via an eight-bit parallel bus, so that only eight additional I/O pins are required (for the keypad) in addition

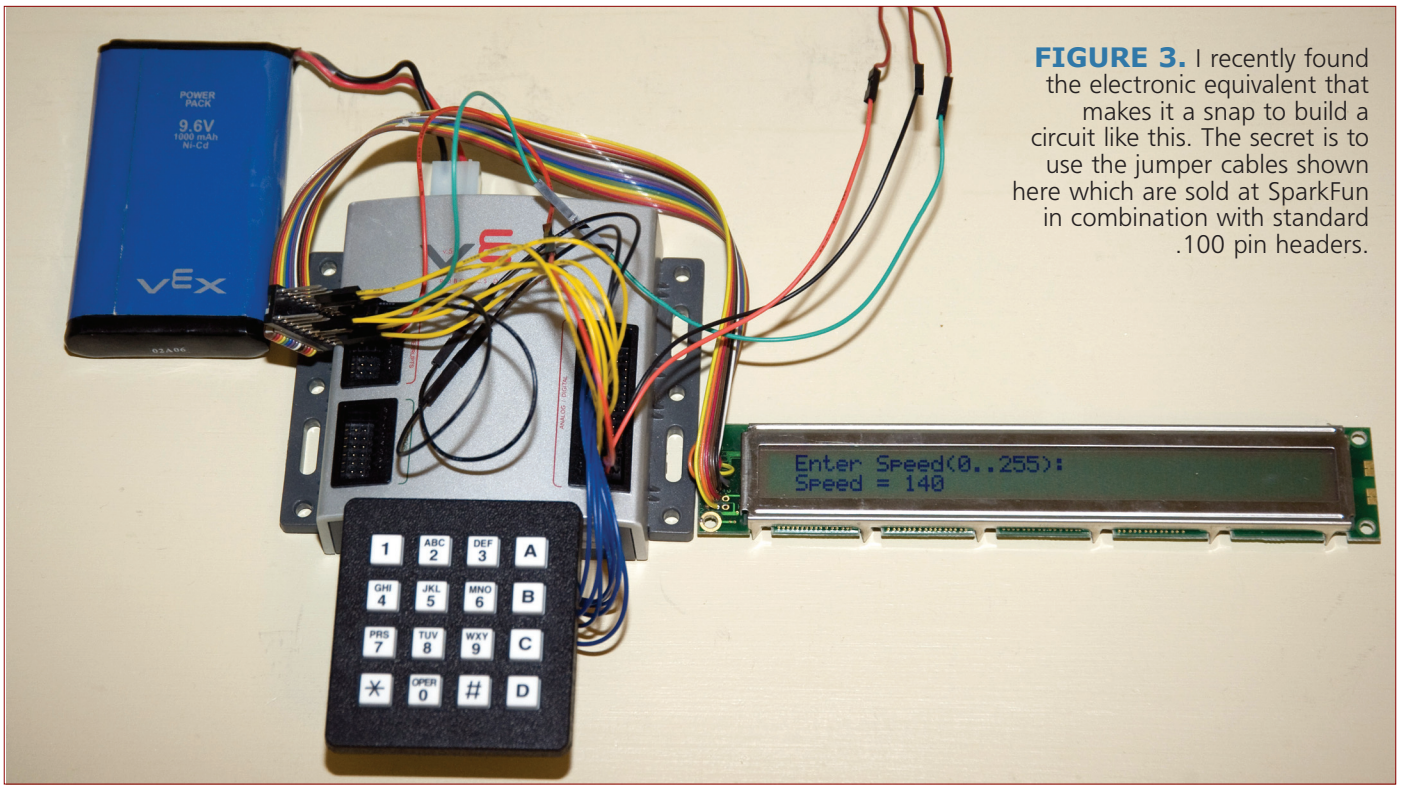


FIGURE 3. I recently found the electronic equivalent that makes it a snap to build a circuit like this. The secret is to use the jumper cables shown here which are sold at SparkFun in combination with standard .100 pin headers.

to those used for the LCD as shown in **Figure 2**.

A VEX User Interface

Data entry can be either in decimal, floating point, or hexadecimal format since the keypad has 16 keys. In fact, we can edit the values by assigning the unused keys from the keypad as a backspace, insert or delete key, cursor position, or tab key. Our PIC18 C User Interface (UI) application uses the keypad scan routine to scan for the user's keystrokes and uses the digits entered to build an ASCII representation of the motor speed entered. We'll slightly alter the firmware from the June '10 article to work with the keypad.

This application also uses the WPILIB routines to drive the VEX motor to the selected speed which prevented me from using a modified version of printf to display data since the WPILIB could not be modified. Instead, I had to resort to making specific function calls to position the cursor and format the data for the LCD.

Although the example provided is very rudimentary, the reader can build on this to develop custom user interfaces using our source code as a starting point.

Some Assembly Required

In order to connect the keypad and LCD to the microcontroller, just follow the wiring diagram shown in the schematic in **Figure 2**. The motor is connected to motor block, pin 1, located on the microcontroller. Please note that two of the LCD signals (RS and E pins) have been relocated from IO7 and IO6 to IO10 and IO11 on the microcontroller (this is different from the schematic in the May issue). I had to change the LCD wiring to accommodate the keypad firmware. The 10K potentiometer controls the contrast to the LCD and should be adjusted for ambient lighting conditions.

Locating pin 1 on the keypad (see **Figure 2**), check the orientation. If it is connected backwards, then the key scan routine will return the values incorrectly when pressed. Care must be taken when connecting the keypad since there are many signals and wires used. The resistors in series with the keypad wires are there to protect the microcontroller from Electro-Static Discharge (ESD). Check for shorts using a

TABLE 1. Bill of materials required to build the User Interface using a keypad and LCD. One component not shown in the schematic is a VEX motor. I used one to demonstrate how to change the motor speed using this interface.

ITEM	QTY	DESCRIPTION	SOURCE
1	1	VEX Controller	Innovation First, Inc. **www.vexforum.com**
2	1	4 x 4 Keypad	RadioShack **www.radioshack.com**
3	1	16 x 2 LCD Display	SparkFun **www.sparkfun.com**
4	40	.100 Pin Headers	Digi-Key **www.Digi-Key.com**
5	1	10K ohm Trim Potentiometer	Digi-Key **www.Digi-Key.com**
6	1	Wire-wrap Cable	RadioShack **www.radioshack.com**
7	1	Package of Jumper Cables	SparkFun **www.sparkfun.com**
8	1	VEX Motor	Innovation First, Inc. **www.vexforum.com**

DVM. Connections should be tested using a continuity tester (or DVM).

Rapid Prototyping the Keypad

I recently found something that makes it a snap to build our circuit. The secret is to use the jumper cables shown in **Figure 3** which are available from SparkFun (**www.sparkfun.com**) in combination with standard .100 pin headers. I was able to build the complete UI (including the keypad, LCD, and motor) in less than two hours using the parts shown in **Table 1**. I took my time to check the connections against the schematic, but one problem I encountered was that I accidently connected the wire leading from the 10K potentiometer wiper to a ground pin which caused smoke to come from the POT when it was first powered up. Fortunately, I was able to disconnect power quickly so no apparent damage was done. It is also always a good idea to check to see if the jumper cables are plugged into the correct pin sockets. These jumper cables are ideal electronic rapid prototyping materials since no soldering (other than pin headers) is required and the parts are easily taken apart and reused. The cables include assortments of various lengths and colors that just plug into the I/O blocks. It is faster than wire-wrapping but not as permanent since the jumper cables can be pulled out (unless they are fastened with tape or hot glue). I do not suggest using these for moving robots or props, or for making permanent circuits. Instead, consider using wire-wrap or point-to-point wiring for those applications. One idea I had is to build a three-row pin header connector so that the combined keypad and LCD modules could just be plugged into the microcontroller for a plug-and-play solution.

VEX Firmware

The PIC18 C example shown in **Listing 1** demonstrates how to scan the keypad for a particular key that has been pressed by the user and then displays it on the LCD. De-bouncing each keystroke is necessary so that repetitive key entries are not accidently entered as valid data.

Listing 1 provides all these necessary functions. To use it, just compile and link it using the PIC18 C tools, and then download it with the IFI loader.

```
// Display each keystroke entered from the
// keypad to the LCD.  Works!!!
// Clear the LCD and move the cursor to
// home position
lcd_clear();
Wait(2);

//Format the Message to be displayed on the LCD
Buffer[0] = ' ';
Buffer[1] = 'E';
Buffer[2] = 'n';
Buffer[3] = 't';
Buffer[4] = 'e';
Buffer[5] = 'r';
Buffer[6] = ' ';
Buffer[7] = 'K';
Buffer[8] = 'e';
Buffer[9] = 'y';
Buffer[10] = ' ';
Buffer[11] = '(';
Buffer[12] = '0';
Buffer[13] = '.';
Buffer[14] = '.';
Buffer[15] = '9';
Buffer[16] = ')';
Buffer[17] = ':';
Buffer[18] = 0;      // String terminator (null
                     // character)

// Position the cursor to the first line
lcd_goto(0,2);

Wait(1);

// Send message to the LCD Display
lcd_puts((char *) Buffer);

// Send message to the Serial Terminal
// if available
   printf("Enter Key ('0'..'9') \r\n",
   KeyValue);

//Format the Message to be displayed on the LCD
Buffer[0] = ' ';
Buffer[1] = 'K';
Buffer[2] = 'e';
Buffer[3] = 'y';
Buffer[4] = ' ';
Buffer[5] = 'p';
Buffer[6] = 'r';
Buffer[7] = 'e';
Buffer[8] = 's';
Buffer[9] = 's';
Buffer[10] = 'e';
Buffer[11] = 'd';
Buffer[12] = ' ';
Buffer[13] = '=';
Buffer[14] = ' ';
Buffer[15] = 0;      // String terminator
                     // (null character)

// Demo loop which reads values from keypad and
// sends them to the LCD. It Waits for press to
// indicate a keypress, then displays the key
// on the LCD Display.  Note that this code
// clears the press bit when done in order to
// prepare for the next press.
while(1)
{
   KeyValue = GetKey();     // Convert the key
                            // value from ASCII
                            // to binary


    // Check the range of the Key Value to make
    // sure is a digit between 0 and 9
    if ((KeyValue >=0) && (KeyValue <=9))
    {
       // Position the cursor
       lcd_goto(1,2);
       Wait(1);

       // Display formatted text to the LCD
      lcd_puts((char *) Buffer);
       lcd_printdec(KeyValue);
    }
}
```

LISTING 1. The PIC18 C example shown here demonstrates how to scan the keypad for a particular key that has been pressed by the user and then displays it on the LCD.

Using the WPILIB Library

WPILib was initially developed as a framework for programming robots used in the *FIRST* competition. There is a version that also works with standard VEX microcontrollers. This is an expanded version of the library that is used by easyC Pro; another version is available for PIC18 C and MPLAB. For more information on WPILib, go to **http://users.wpi.edu/~bamiller/WPILib/**.

There are PIC18 C, Easy C, and Easy C Professional functions that enable the VEX user to read digital and analog inputs which are common to all three C compilers. They all share the WPILIB framework that provides these functions. In Easy C and Easy C Professional, they are automatically included but in PIC18 C applications you need to include the C header file API.h and specify the WPILibVex.lib in the MPLAB project as shown in the screen capture in **Figure 4**. Programming details will be clarified in future articles with C examples, but it's a good idea to get familiar with the programming statements that relate to digital and analog inputs and outputs. (I used these functions to set the motor speed entered from the UI as demonstrated in **Listing 2**.) The PIC18 C example shows how you can change the speed of a motor and display the current speed value on the LCD. It also shows you how to use the WPILIB functions to set the actual motor speed.

Figure 4 shows all the necessary C modules including

```
// VEX User Interface (UI)
// This is a practical example that allows you
to enter the speed for a specific VEX Motor
// Display and display it on the LCD.

// Define a robot two wheels using WPILIB
TwoWheelDrive(1, 2);  // This code works!!!

// Clear the LCD and move the cursor to home
// position
lcd_clear();

//Format the Message to be displayed on the LCD
Buffer[0] = 'E';
Buffer[1] = 'n';
Buffer[2] = 't';
Buffer[3] = 'e';
Buffer[4] = 'r';
Buffer[5] = ' ';
Buffer[6] = 'S';
Buffer[7] = 'p';
Buffer[8] = 'e';
Buffer[9] = 'e';
Buffer[10] = 'd';
Buffer[11] = '(';
Buffer[12] = '0';
Buffer[13] = '.';
Buffer[14] = '.';
Buffer[15] = '2';
Buffer[16] = '5';
Buffer[17] = '5';
Buffer[18] = ')';
Buffer[19] = ':';
Buffer[20] = 0;       // String terminator
                      // (null character)

// Position the cursor to the first line
lcd_goto(0,2);

// Send message to the LCD Display
lcd_puts((char *) Buffer);

// Send message to the Serial Terminal
// if available
   printf("Enter Speed (0..255) \r\n",
KeyValue);

//Format the Message to be displayed on the LCD
Buffer[0] = 'S ';
Buffer[1] = 'p';
Buffer[2] = 'e';
Buffer[3] = 'e';
Buffer[4] = 'd';
Buffer[5] = ' ';
Buffer[6] = '=';
Buffer[7] = ' ';
Buffer[8] = 0;        // String terminator
                      // (null character)

Speed = 0;
i = 0;
while(1)
{
       KeyValue = GetKey();
              // Convert the key value from
              // ASCII to binary

    // Check the range of the Key Value to make
    // sure is a digit between 0 and 9
    if ((KeyValue >=0) && (KeyValue <=9))
    {
        // Enter the Speed one digit at a time

        if (i<3)
        {
            Speed += (int)KeyValue*PowerOfTen[i];
            i++;

            // Position the cursor
            lcd_goto(1,2);

            // Display formatted text to the LCD
           lcd_puts((char *) Buffer);
             lcd_printdec(Speed);
        }
        else
        {
            // Set the VEX Motor # 1 Speed
               SetPWM (1,Speed);

            // Reset digit count and Speed
            i = 0;
            Speed = 0;
        }
    }
    else if (KeyValue == ENTER)
    {
        // Process the speed value entered
              // Send the scanned key to the
              // host serial terminal also
    printf("Current Motor Speed = %d \r\n",
Speed);

        // Set the VEX Motor # 1 Speed
           SetPWM (1,Speed);

        // Reset digit count and Speed
        i = 0;
        Speed = 0;
    }
}
```

LISTING 2. This PIC18 C example shows how you can use a VEX UI to change the speed of a motor and display the current speed value on the LCD. It also shows you how to use the WPILIB functions to set the actual motor speed.

keypad.c and lcd.c that are required to implement a simple VEX UI. These are conveniently provided as a download from the *SERVO* website at **www.servomagazine.com**. To test the keypad and LCD interface, you only need to download the keypad.hex application. To customize it, you will need to modify the code in the main function of the WPIkeypad.c routine and re-make the project using MPLAB. The functions used in this application include the following:

- The Get_Key function scans the keypad and returns the keystroke that was pressed from the user after de-bouncing it.
- The lcd_goto function positions the LCD cursor to the specified position.
- The lcd_puts function sends the text contained in the buffer to the LCD.
- The lcd_printdec displays numeric data to the LCD.
- The TwoWheelDrive function called from the WPILIB initializes the microcontroller to run the VEX motors in two wheel drive mode (2WD).
- The SetPWM function called from the WPILIB is used to set motor #1 speed to the value entered from the keypad.

To run the application, download the WPIKeypad.hex file to the microcontroller using the IFI bootloader and start entering a speed for the motor with up to three digits for a speed value between 000 and 255. Notice that the first digit entered is the most significant digit (unlike calculators which enter the least significant digit first). This can easily be modified in the WPIKeypad.c application by storing the digits into a small buffer. The motor will start turning at the selected speed once an extra digit between 0 and 9 is entered from the keypad; the cycle repeats, overwriting the current motor command.

I will be referring to more WPILIB functions in future articles and showing you some advanced coding examples that would be much more difficult if done without WPILIB since controlling motors requires waveform generation (PWM) and reading sensors may require interrupt support. **Figure 4** also shows where I added my own routines to scan the keypad. This MPLAB project contains all the necessary modules to compile and link the keypad application that generates the keypad.hex file that is used to program the controller once the keypad has been wired to it. As a side note, I do prefer using the DDT tools instead of WPILIB when interrupts are required (such as scanning pushbutton switches or bumper switches since it allows access to most of the PIC18F8520 registers, timers, and peripherals).

Going further, an interesting experiment is to use the keypad to enter floating point or hex numbers, and display

them using the numeric LED or LCD display, or develop custom menus for selecting various robot behaviors while disconnected from a PC.

Until next time, when I show you how to run stepper motors from the microcontroller. **SV**

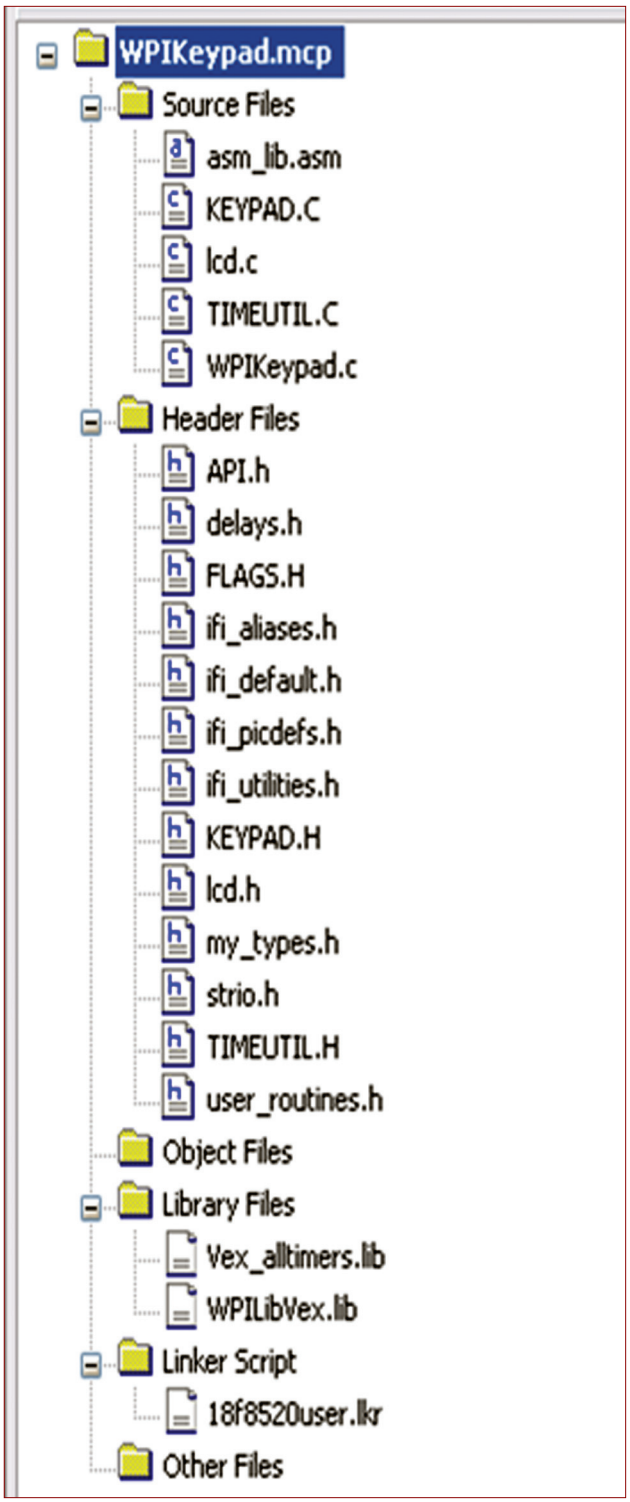


FIGURE 4. The MPLAB project file shows where I added my own routines to scan the keypad. This contains all the necessary modules to compile and link the keypad application that generates the keypad.hex file that is used to program the VEX controller once the keypad has been wired to it.