Unfortunately page 154 of the Teach-In 2 book has been incorrectly printed. We apologise for the error. This is the correct page which should be used instead of page 154 in the book.

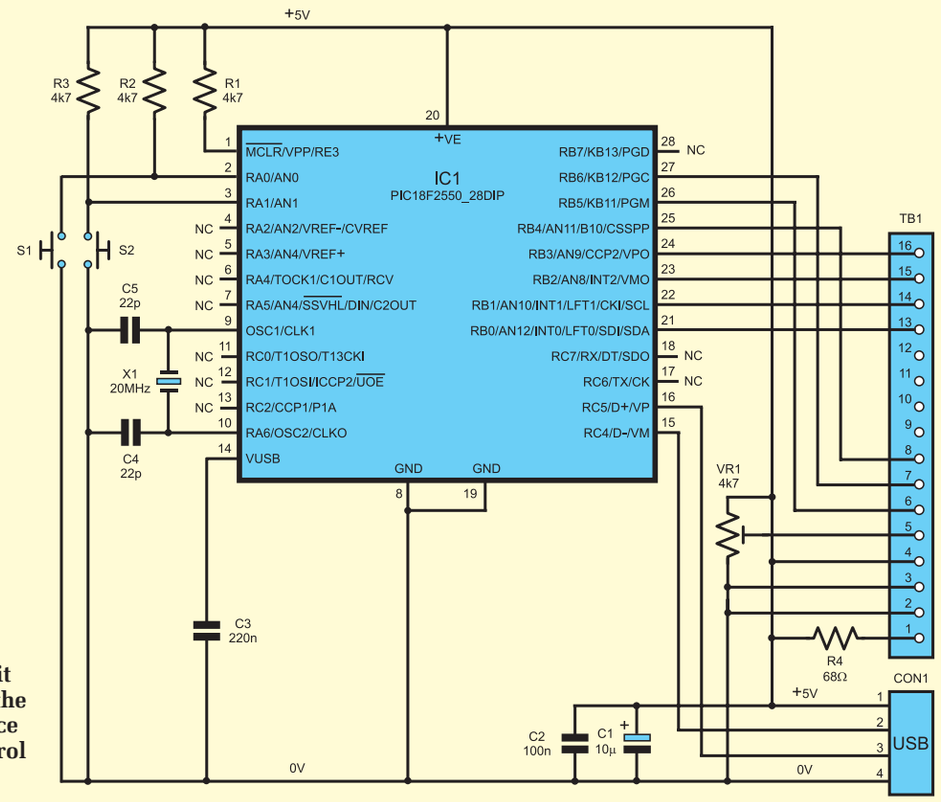

Fig.1: circuit diagram for the USB Interface for LCD control

pins) we will connect to the default port pins: PORTB0-3 for LCD data, PORTB4 for the E signal, PORTB5 for the RS signal and PORTB6 for the RW signal.

Now we know the connections for the LCD we are free to choose the two port pins for our switches. For simplicity, we will use PORTA0 for switch 1, and PORTA1 for switch 2.

Adding a few pull-up resistors and a couple of decoupling capacitors, we quickly arrive at the circuit diagram in Fig.1.

Building the circuit should not be a challenge; the USB connector has four pins on a 0.1 inch pitch, which will fit to stripboard. The connection to the LCD, TB1, is for a CDL4162 2 × 16 LCD that uses a 16-pin 0.1 inch pitch header. It should be easy to connect up to any 2 × 16 LCD based on the popular HD44780 controller.

Software configuration

The example code will have been installed into the directory **C:\MCHPFSUSB**, with several subdirectories. The code to the project can be found in the **fw\cdc** sub directory. You will find the file **MCHPUSB.mcw** MPLAB project file in this directory; if you double-click on it you will open the entire project in the MPLAB program.

So what does the CDC emulation firmware actually do for us? Reading the application note, it explains that the code will enable the PIC, when plugged into a USB port, to appear as a new COM port on the computer. You may then use any serial port code (or HyperTerminal for that matter) to connect to the board as though it were on an RS232 interface.

Obviously, we must take a look at what the software does before we can start to modify it to make it work with our specific hardware. Reading the application note and scanning the main source files (**main.c, user.c**) give some ideas; the 'framework' of code should start-up, wait for a keypress and then display a message on any terminal program that is connected to the virtual com port.

Key press? What key? Searching through the code reveals a couple of macros that decide which port pins 'map' to the switch input signals used by the software. Unsurprisingly, the port pins used by the software do not match ours; therefore, we need to change these macros in the source code. You will find these definitions in **io_cfg.h**.

At the moment they point to ports B4 and B5; we want to use A0 and A1 – so let's change them:

```
#define mInitAllSwitches() TRISAbits.TRISA0=1;TRISAbits.TRISA1=1;
#define mInitSwitch2() TRISAbits.TRISA0=1;
#define mInitSwitch3() TRISAbits.TRISA1=1;
#define sw2 PORTAbits.RA0
#define sw3 PORTAbits.RA1
```

As we mentioned in an earlier article, it is always a good idea to explicitly specify the config register settings in your source code. The example code does not do that, so we will add it, to the beginning of the file **main.c** (anywhere near the top of the file). The code you should add is:

```
#pragma config PLLDIV=5, CPUDIV=OSC1_PLL2, USBDIV=2
#pragma config IESO=OFF, FCMEM=OFF, FOSC=HSPLL_HS
#pragma config PWRT=OFF, BOR=OFF, VREGEN=ON, MCLRE=ON
#pragma config PBADEN=OFF, STVREN=ON, LVP=OFF
#pragma config XINST=OFF, DEBUG=OFF, WDT=OFF
```

The example code has several configuration options for different hardware designs. As we are going