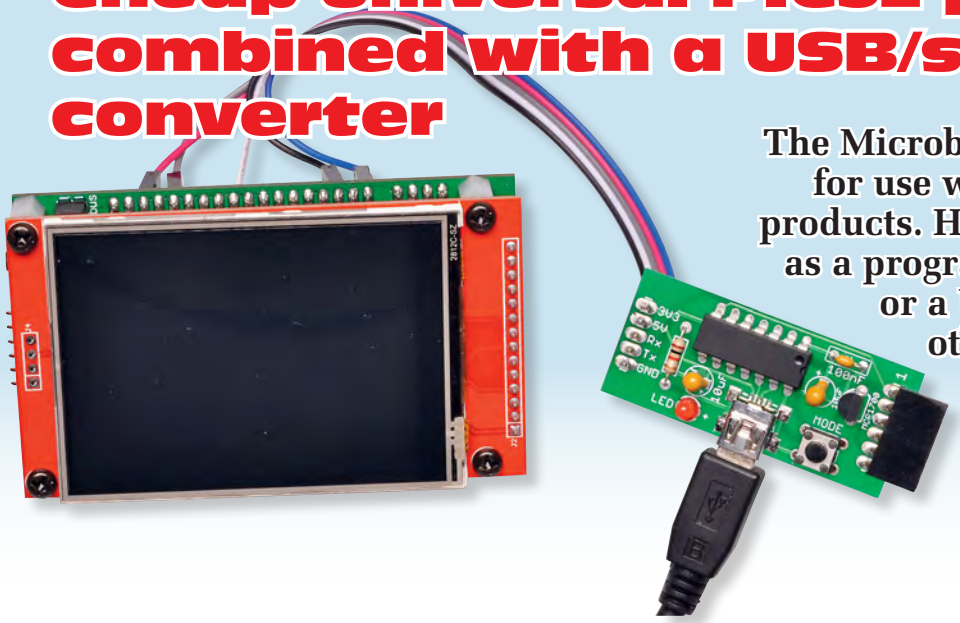
Microbridge

Cheap universal PIC32 programmer combined with a USB/serial converter



The Microbridge was created as a tool for use with the Micromite range of products. However, it can also be used as a programmer for any PIC32, and/or a USB-to-serial converter for other processors, such as the Arduino or Raspberry Pi.

By Geoff Graham

The Micromite microcontroller, which has featured many times on our pages, requires a USB/serial converter to load, edit and run the program (unless you purchased a pre-programmed chip).

We previously recommended devices based on the CP2102, or FTDI FT232 for this job. They are cheap and convenient; however, you still require a PIC32 programmer if you need to update the Micromite firmware.

Firmware updates for the Micromite are released regularly and usually provide worthwhile new features and bug fixes, so it is definitely an advantage having access to a PIC32 programmer.

But now you don't need a dedicated PIC32 programmer. Instead, the *Microbridge* combines the USB/serial interface and PIC32 programming features in a single package. It is easy to build and uses a low-cost 14-pin chip.

In fact, the Microbridge is so economical and convenient that it makes sense to permanently attach it to your Micromite. With that in mind, we have designed a new version of the *Micromite LCD Backpack* with the *Microbridge* integrated which is featured on page 22 of this issue.

The development of the *Microbridge* and the associated software was truly an international effort, with contributions from New Zealand, Thailand, the US and UK (see the side box for the details).

Circuit details

Referring to Fig.1, you can see that the *Microbridge* consists of just a Microchip PIC16F1455 microcontroller, a voltage regulator and a few passive components.

Microbridge credits

The Microbridge is the result of an international collaboration.

- Peter Mather in the UK wrote the firmware for the PIC16F1455 and the BASIC program for programming a PIC16F1455 using a Micromite (see panel on programming)
- Serge Vakulenko in the USA wrote pic32prog
- Robert Rozee in New Zealand wrote the ASCII ICSP interface for pic32prog
- MicroBlocks (a company in Thailand) developed the original concept of using the PIC16F1455 as both a USB/serial converter and programmer, but did not publish their code for copyright reasons.

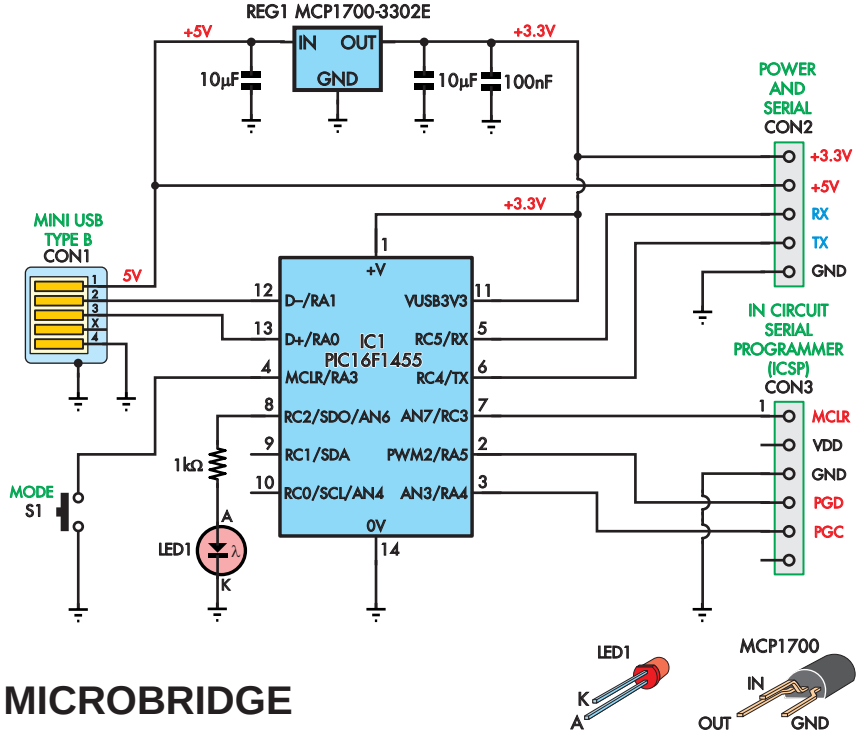
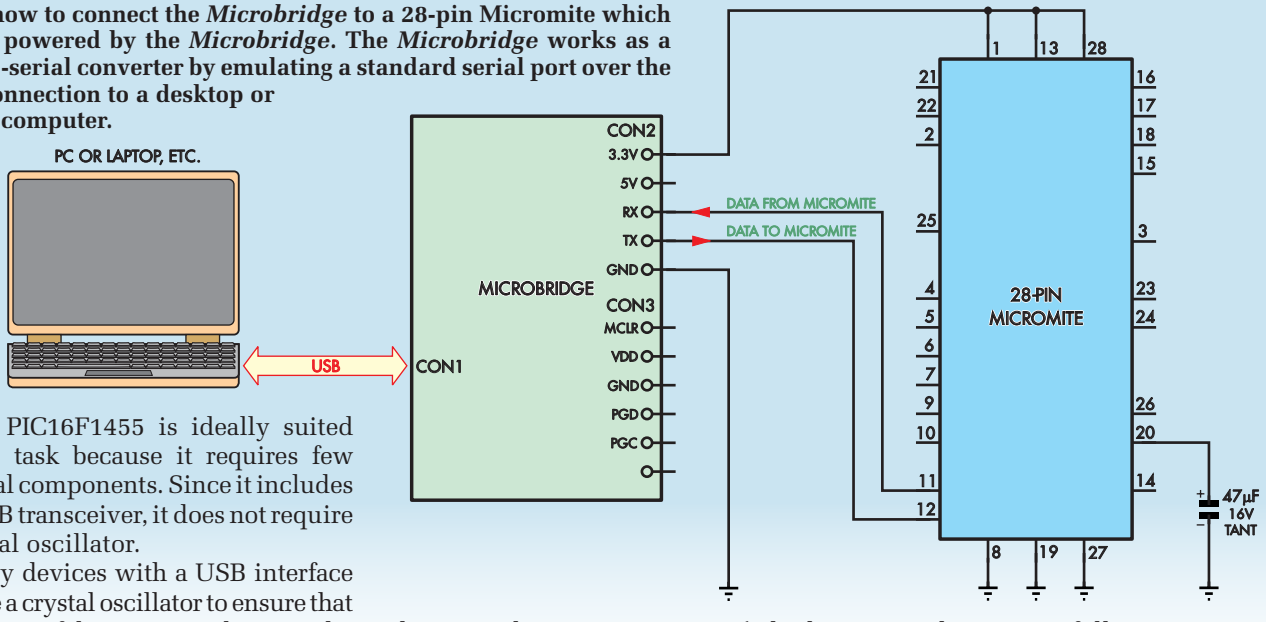


Fig.1: the *Microbridge* consists of a Microchip PIC16F1455 microcontroller, a voltage regulator and a few passive components. The PIC16F1455 is ideally suited to this task because it requires few external components and can automatically tune its internal clock to the host's USB signal timing.

Fig.2: how to connect the *Microbridge* to a 28-pin Micromite which is also powered by the *Microbridge*. The *Microbridge* works as a USB-to-serial converter by emulating a standard serial port over the USB connection to a desktop or laptop computer.



The PIC16F1455 is ideally suited to this task because it requires few external components. Since it includes the USB transceiver, it does not require a crystal oscillator.

Many devices with a USB interface require a crystal oscillator to ensure that the timing of the USB signals meets the strict timing requirements of the USB standard. However, the PIC16F1455 has a feature that Microchip calls 'active clock tuning'.

This allows the PIC16F1455 to use the host's USB signals (which presumably are derived from a crystal oscillator) to automatically tune its internal RC oscillator to the precision required by the standard. Hence, a crystal is not required and this helps keep the circuit simple and the cost down.

The PIC16F1455 can run on a supply voltage of 2.3-5.5V and also includes its own 3.3V regulator for powering its USB transceiver (USB uses 3.3V signal levels).

This means that we could directly power the PIC16F1455 from the USB 5V supply, but then we would need level converters for the signal lines that go to the PIC32 processor (which runs from 3.3V).

For that reason, we've included a low-cost 3.3V regulator (REG1, MCP1700) for powering the PIC16F1455 and we are ignoring its internal regulator. A side benefit of this approach is that this 3.3V supply has spare current capacity so it can also be used to power an attached Micromite chip.

The serial interface is made available on CON2 and includes the 5V USB power and the 3.3V from our on-board regulator.

By default, the serial interface runs at 38400 baud, which is also the default used by the Micromite's console interface.

The programming interface is on CON3 and this provides the standard I/O pins used for In-Circuit Serial Programming (ICSP) on Microchip products. These are as follows:

Pin 1: MCLR/V_{PP} – this is the reset pin for the PIC32 chip and is driven low by the *Microbridge*. It is also used to force the PIC32 into programming mode. On other PICs, this pin is also used as a programming voltage source of around 15V, but the PIC32 generates this internally.

Pin 2: V_{DD} – normally, this is used to detect the power supply voltage for the PIC32, but on the *Microbridge* it is not used.

Pin 3: GND – the ground connection which must go to V_{SS} (ground) on the PIC32.

Pin 4: PGD – the programming data pin, which is bidirectional so that data can be sent to the PIC32 then read back by the *Microbridge's* firmware to verify that programming has been successful and no errors have been introduced.

Pin 5: PGC – the programming clock signal, generated by the *Microbridge* to synchronise the transfer of data on the PGD line.

Pin 6: NC – not connected in most ICSP devices.

The *Microbridge* is switched into programming mode by using pushbutton switch S1. LED1 flashes to indicate serial traffic or it lights up continuously when in programming mode.

USB/serial mode

USB/serial mode is the default when power is applied. In this mode, the *Microbridge* works as a USB-to-serial converter – it emulates a standard serial port over USB and converts the signal to a standard TTL-level serial interface for the Micromite (or other processor).

From an operating system viewpoint, the *Microbridge* imitates the Microchip MCP2200 USB/serial converter. Windows 10 is delivered with the correct driver for this device

Parts List

1 double-sided PCB available from the *EPE PCB Service*, coded 24104171, 50mm × 22.5mm
1 Mini Type-B USB socket, horizontal SMD USB 2.0
1 PCB-mount SPST momentary tactile switch (S1)
1 14-pin DIL IC socket (for IC1)
1 6-pin 90° female socket, 2.54mm pitch OR
1 6-pin female socket, 2.54mm pitch, with pins bent 90°
1 5-pin vertical header, 2.54mm pitch

Semiconductors
1 PIC16F1455-I/P* microcontroller programmed with 2410417A.HEX (IC1)
1 MCP1700-3302E/TO 3.3V linear regulator (REG1)
1 3mm red LED (LED1)

Resistors (5%, ¼W)
1 1kW

Capacitors
2 10µF 16V tantalum or X5R SMD ceramic (3216/1206 size)
1 100nF 50V multi-layer ceramic

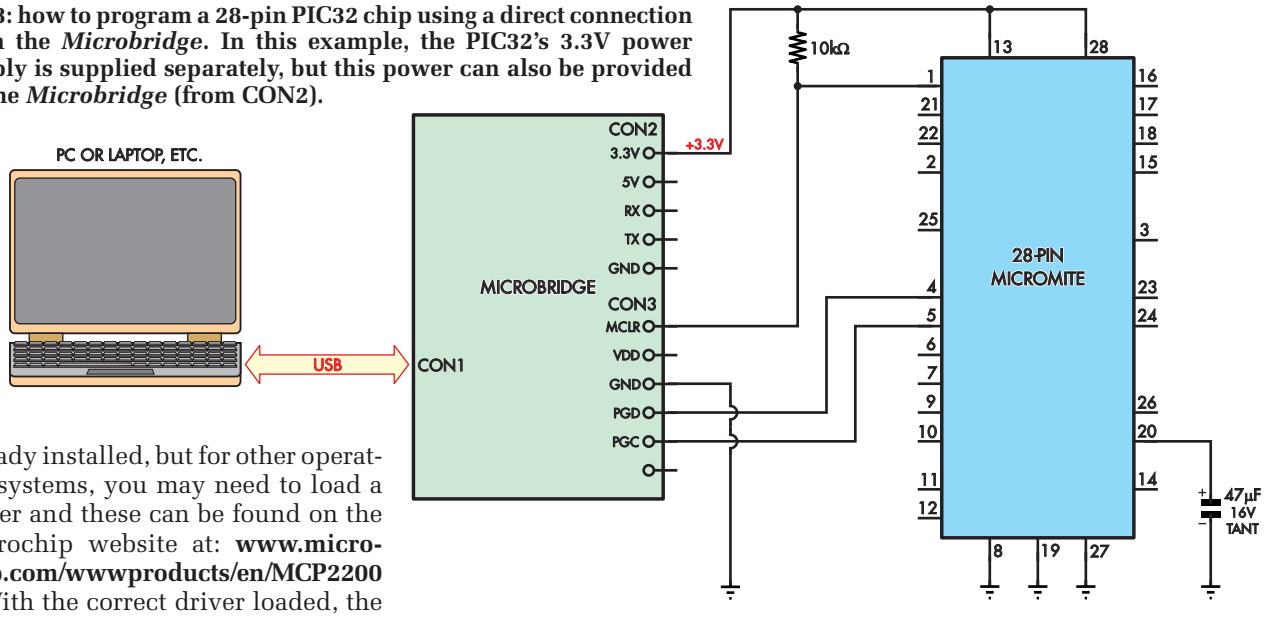
* PIC16LF1455-I/P or PIC16(L)F1454-I/P are also suitable

Win a Microbridge!

EPE is running a competition to win a fully-assembled Microbridge thanks to the generous sponsorship of Micromite online shop **micromite.org**

For entry details, please turn to page 27

Fig.3: how to program a 28-pin PIC32 chip using a direct connection from the *Microbridge*. In this example, the PIC32's 3.3V power supply is supplied separately, but this power can also be provided by the *Microbridge* (from CON2).



already installed, but for other operating systems, you may need to load a driver and these can be found on the Microchip website at: **www.microchip.com/wwwproducts/en/MCP2200**

With the correct driver loaded, the *Microbridge* appears as a standard serial port on your computer. For example, in Windows it will appear as COMxx where xx is some number allocated by Windows.

To discover this number you can use Device Manager and look under 'Ports (COM and LPT)' for the *Microbridge*, which will be labelled 'USB Serial Port (COMxx)', where xx is the serial port number (eg, COM6). You can then start your terminal emulator (eg, Tera Term) and specify this COM number in the setup menus.

By default, the *Microbridge* operates at 38400 baud with 8-bit data, one stop bit and no parity, which are the standard settings used by the Micromite's console. However, you can change the baud rate to any standard speed from 300 to 230400 (ie, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 76800, 115200 or 230400 baud) in the terminal emulator.

Fig.2 shows how to connect the *Microbridge* to a 28-pin Micromite, which is also powered by the *Microbridge*. When a character is sent or received by the *Microbridge*, LED1 flashes briefly. This is a handy visual clue that the device is working correctly.

Note that TX (transmit) from the *Microbridge* must go to the RX (receive) on the Micromite; likewise, the TX on the Micromite must connect to RX on the *Microbridge*. This is logical when you think about it because signals transmitted by one device must be received by the other.

If you connect pin 1 of CON3 (the programming connector) to the MCLR (reset) pin of the Micromite, you can also use the *Microbridge* to remotely reset the Micromite. This is done by sending a serial break signal to the *Microbridge*. In Tera Term this is accomplished by pressing ALT-B or via the Tera Term menu.

Another way of generating a reset is to press and hold the mode switch on the *Microbridge* for two or more seconds. LED1 will flash and the MCLR line will be briefly driven low to effect the reset.

Programming mode

CON3 on the *Microbridge* (the ICSP socket) is compatible with the connector used on the Microchip PICkit 3 programmer so the *Microbridge* can plug into any programming connector intended for the PICkit 3. For example, the *Microbridge* can plug directly onto the programming connector on the original *Micromite LCD Backpack* (see the accompanying photograph on the next spread).

Alternatively, to program a 28-pin PIC32 chip using direct connections, Fig.3 shows how to do this. The PIC32's 3.3V power supply can be supplied separately or this power can be provided by the *Microbridge* via CON2.

To enter programming mode, momentarily press and release mode switch S1 and LED1 will illuminate to indicate that programming mode is active.

If you accidently pressed this switch and did not want to enter programming mode, cycle the power on the *Microbridge*, or, press and hold down S1 for two seconds; either way, this will return you to the default USB/serial mode.

To program a PIC32 via the *Microbridge*, use a program called **pic32prog** written by Serge Vakulenko in California.

This is a Windows progra and it can be downloaded from th from GitHub: **https://github.com/se gev/pic32prog**

pic32prog must be run from th command prompt in Windows using th the command line:

pic32prog -d ascii:comxx yyyy.he

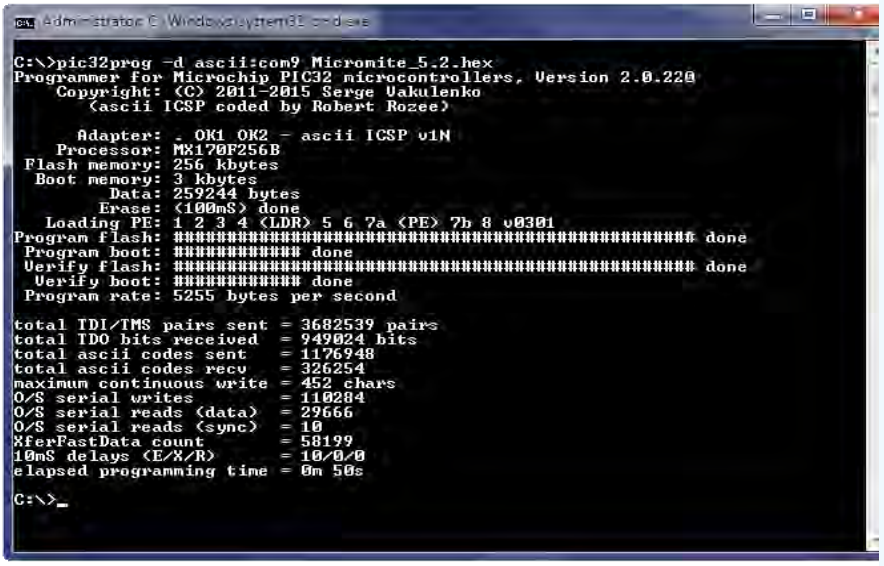


Fig.4: This screenshot shows the complete operation of pic32prog**. It uploads the he file to the *Microbridge*, which programs it into the PIC32 and subsequently reads bac the programmed data to verify that the programming operation completed correctl**

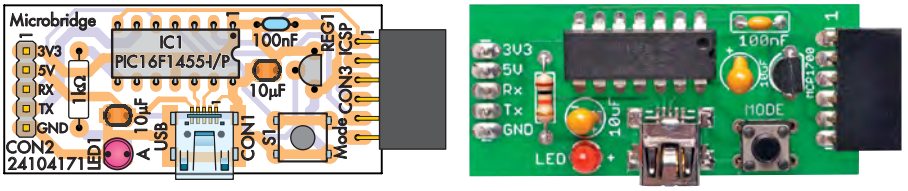


Fig.5: PCB component overlay diagram for the *Microbridge*. The USB socket is the only SMD component. IC1 may be mounted in a socket. We prefer SMD ceramic capacitors to tantalum due to their longer life, however you can use through-hole tantalum capacitors.

Where xx is the COM port number created by Windows for the *Microbridge* and **yyyy.hex** is the file containing the firmware that you want to program into the PIC32. For example, if your *Microbridge* was allocated the virtual serial port of COM12 and the file that you wanted to program was **firm.hex**, the command line that you should use would be:

```
pic32prog -d ascii:com12 firm.hex
```

When you press enter, **pic32prog** will automatically upload the hex file to the *Microbridge*, program it into the PIC32 then read back the programmed data to verify that the programming operation was executed correctly. Fig.4 shows a typical output of this operation.

At the completion of the programming operation, LED1 switches off and the *Microbridge* will revert to operating as a USB/serial converter. You can then start up your terminal emulator, connect to the *Microbridge* and run your program.

A common cause of programming errors is that **pic32prog** cannot access the serial port on your computer because you have not closed the terminal emulator that you were previously using to access the *Microbridge*. So, make sure that you close your terminal emulator before you run **pic32prog**.

Construction

The *Microbridge* uses fewer than a dozen components and all except the USB socket are through-hole types, so construction should take less than half an hour. The component overlay diagram is shown in Fig.5.

Start with the USB socket as this is the only surface-mount component. On the underside of the socket, there should be two small plastic pegs which match corresponding holes on the PCB and these will correctly locate the socket.

Once it is in place, solder the connector's mounting lugs first using plenty of solder for strength then, using a fine point soldering iron tip, solder the signal pins. Carefully check the pin soldering under a good light and with magnification and clean up any solder bridges using solder wick with a little added flux paste to make it easier.

The remaining components are easy to fit and should be soldered starting with the low-profile items such as resistors and ending with the high profile components such as the connectors.

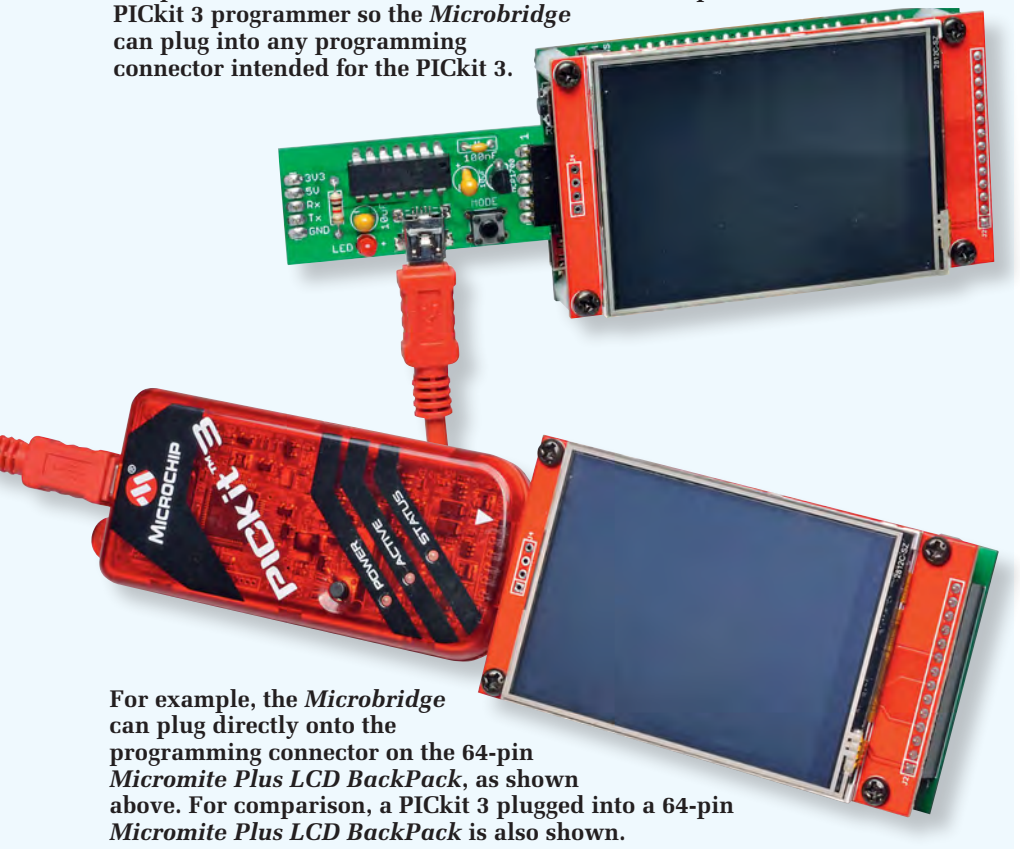
Two of the capacitors and the LED are polarised, so pay attention to their mounting orientation. We did not use an IC socket for IC1 because we had programmed and tested it beforehand, but a socket is recommended and is handy if you suspect a fault and want to swap out the IC for testing.

For CON2 (the serial I/O and power) connector, we mounted a five-pin header on the underside of the board so that it could easily plug into a solderless breadboard for prototyping with the Micromite, but you could use a different arrangement, for example, flying leads.

The right-angle six-pin socket used for the ICSP programmer output (CON3) can be difficult to find so you can do what we did and purchase a straight six-pin socket intended for Arduino boards and bend the pins to 90° so that the socket can mount flush to the PCB. See the parts list for suitable components.

(**Note: although not shown on the circuit diagram, a 10kΩ resistor between PIC pins 1 and 5 will improve S1 response when the Microbridge is not connected to a target device.**)

CON3 on the *Microbridge* (the ICSP socket) is compatible with the connector used on the Microchip PICkit 3 programmer so the *Microbridge* can plug into any programming connector intended for the PICkit 3.



For example, the *Microbridge* can plug directly onto the programming connector on the 64-pin *Micromite Plus LCD BackPack*, as shown above. For comparison, a PICkit 3 plugged into a 64-pin *Micromite Plus LCD BackPack* is also shown.

Testing

There is not much to go wrong with the *Microbridge*, so if it does not work the first time you should first re-check the driver installation on your PC. Do you have the right driver, is it installed correctly and do you have the right COM port number? In normal USB/serial mode the *Microbridge* will draw about 8mA and any reading substantially different from this indicates an assembly error.

A handy test feature is that when you press a key in your terminal emulator, LED1 on the *Microbridge* should flash. Another test that you can make is to short the TX and RX pins on CON2, and as you type characters into the terminal emulator, you should see them echoed back to the terminal emulator.

The *Microbridge* uses a PIC16F1455 which acts as a PIC32 programmer to load the firmware into your blank PIC32 microcontroller; for example, to make it into a Micromite. This sounds great because now you do not need a PIC programmer. Or do you?

Firmware transfer

The problem now is getting the *Microbridge's* firmware into the PIC16F1455. One option is to purchase a pre-programmed PIC16F1455 from **micromite.org**. But if you already have at least one Micromite, you can program the PIC16F1455 yourself using just the Micromite and a standard 9V battery.

It is easy to do and will only take 30 seconds. Then, once you have the PIC16F1455 programmed, you can use it to program as many other Micromites as you want!

To get started, wire up the PIC16F1455, the Micromite and the 9V battery as shown in Fig.6.

The best way to do this is on a solderless breadboard or a strip of perforated prototyping board. The battery can be a standard PP3 9V battery and this is used to provide the programming voltage for the PIC16F1455. Only a few milliamps will be drawn from it, and as long as its terminal voltage is 8V or greater it will do the job. The switch used to connect the battery can be as simple as a lead with an alligator clip that can be clipped onto the battery's positive terminal.

The Micromite used for the programming operation can be any version of the Micromite family (ie, a 28-pin Micromite to a 100-pin Micromite Plus) so long as it is running version 5.0 or later of MMBasic. Pins 4 and 5 on the Micromite are used to load the firmware into the PIC16F1455, and all versions have these two pins free.

If for some reason your one does not, you can edit the BASIC program to change the pin assignments (they are defined at the very start of the program).

With everything connected, load the BASIC program **MicrobridgeProg.bas** into the Micromite. This program can be downloaded for free from the author's website (**geoffg.net/microbridge.html**). It will work with all chips that are supported by the *Microbridge* firmware (16F1455, 16F1454, 16LF1454 or 16LF1455). This program was written by Peter Mather of the UK, who also developed the *Microbridge's* firmware.

Make sure that the 9V battery is disconnected and run the BASIC program on the Micromite. From there, it is just a case of following the program's on-screen instructions which will tell you when to connect and disconnect the battery.

The programming time is under 30 seconds and the software will report its progress as it goes. Fig.7 shows a typical programming session. When the programming operation has finished, you can disconnect the battery, remove the PIC16F1455 and install it in your *Microbridge* board. Then, you can use the *Microbridge* to program further PIC32 chips.

The firmware loaded into the PIC16F1455 will be version 1.18 and this contains a bootloader which allows another Micromite to update it via the serial console interface.

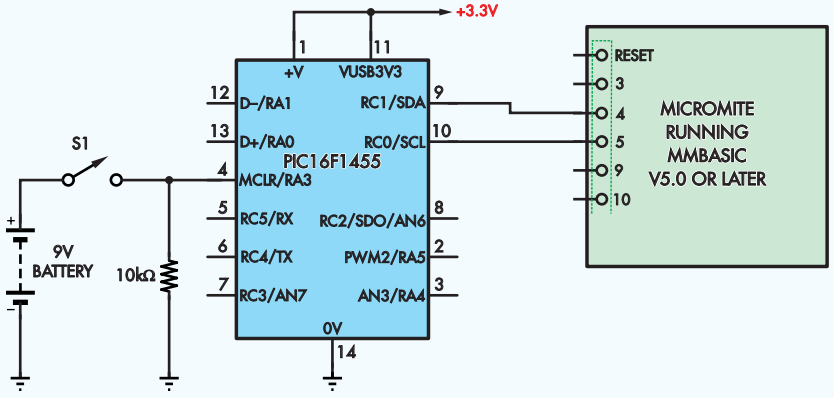


Fig.6: if you already have a Micromite, you can use it to program a blank PIC16F1455 (for use as a *Microbridge*). All you need is a standard 9V battery and a 10kΩ resistor. Connect everything as shown in the circuit above. The MicrobridgeProg.bas **running on the Micromite will prompt you when to connect and disconnect the battery.**

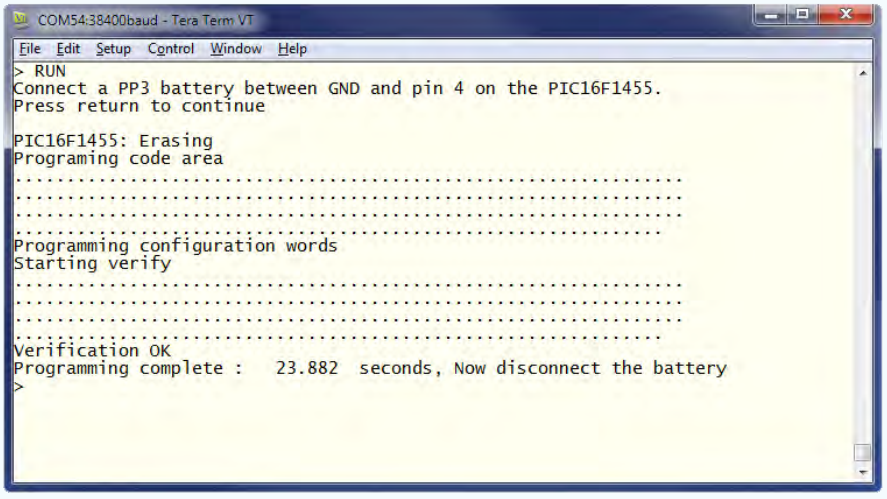


Fig.7: this screenshot shows the complete programming operation for a PIC16F1455 using a Micromite and a standard 9V battery. The program running on the Micromite is MicrobridgeProg.bas.

Updates

This updating is even easier than the initial programming described above and can be done with the *Microbridge* permanently connected to the Micromite. There will likely be no need to update the *Microbridge's* firmware but, if there is, the current firmware can do it.

Reproduced by arrangement with SILICON CHIP magazine 2018. www.siliconchip.com.au