Using Cheap Asian Electronic Modules Part 3

Computer Interface Modules

Want to connect a microcontroller to your PC? How about interfacing with a microSD memory card? These low-cost modules make life really easy! Jim Rowe shows you how.

THE FIRST MODULE we're looking at this month has been used in a number of SILICON CHIP's recent projects. It's a serial USB-UART (universal asynchronous receiver/transmitter) bridge which allows just about any microcomputer or peripheral module to exchange data with a PC, via a standard USB port.

Let's start by explaining what is meant by the rather clumsy term "serial USB-UART bridge". Firstly, a UART is an interface which can operate in one of several different common serial protocols. The serial protocol we're most interested in (and which is most widely used) is 3.3V "TTL" RS-232. The term "bridge" simply refers to the fact that this module allows data to pass between the USB interface and UART interface unchanged.

In fact, we've already described a device with essentially the same purpose, the Microchip MCP2200 "protocol converter" used in the USB/RS-232C serial interface which was published in the April 2014 issue.

Another very similar device is the FT232 chip from the British firm FTDI, which was used in the Elexol USBMOD3 USB interface module in the USB Electrocardiograph project (SILICON CHIP, February 2005).

Note that for a UART to provide a fully compatible RS-232 serial port, as used in many now obsolete PCs, it's necessary to provide level shifting from the UART's 3.3V (TTL) signalling levels to the RS-232 bipolar logic levels of ±3-15V.

But these days, RS-232 is commonly used for short-range communications between microcontrollers and bridges and in this case, the TTL signal levels are all you really need.

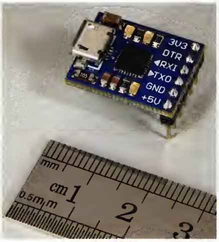
The first serial USB-UART bridge modules to become popular were based around FTDI's improved FT232R converter chips. However, these chips became so popular that some Asian firms made "clones" of them, even going so far as copying the package markings.

Understandably, this upset FTDI and as a result they released a new version of their Windows VCP driver which was able to identify when a clone chip was being used and disable it. This "clone killer" driver was included in an automatic update that Microsoft unwittingly provided to Windows users.

As a result, thousands of people found that their low-cost USB-UART converter modules, some inside commercial products, suddenly stopped working and became worthless. Naturally, this made many people cautious of buying any converter based on the FTDI FT232R chip, because of the difficulty in ensuring that you are buying a genuine FTDI chip rather than a clone chip that would stop working as soon as you tried to use it with Windows.

As a result of this, CP2102-based USB-UART bridges have become very popular. These are not only less expensive than FT232-based modules but are (currently) free from such driver issues.

A good example of this type of module is the tiny one shown in the photo below. This same module has been used in quite a few of our recent projects, such as the Micromite LCD BackPack (SILICON CHIP, February 2016) and Touchscreen Appliance Energy Meter



A CP2102 module, measuring only 20 x 16mm. Two of the indicator LEDs glow when data is being transmitted.

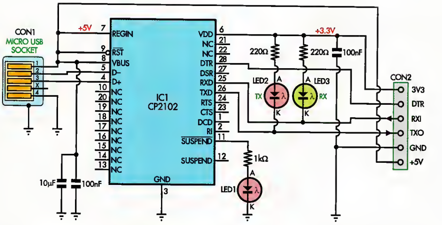


Fig.1: complete circuit diagram for the CP2102-based serial USB-UART bridge. The CP2102 can be powered directly from the USB V_{BUS} line and it contains a low drop-out voltage regulator to provide 3.3-3.45V (V_{DD}) from 4-5.25V (REGIN).

(SILICON CHIP, August-October 2016). In fact, it can be used with virtually any Maximite or Micromite, to program the micro as well as debug the software or load data into or out of the micro's RAM.

The CP2102-based bridge

As you can see from the photo and circuit diagram Fig.1, there's very little in this module apart from the CP2102 chip itself (IC1), three indicator LEDs and half a dozen passive components.

The internals of IC1's tiny (5 x 5mm) 28-pin QFN SMD package are shown in the internal block diagram, Fig.2. It's conceptually quite simple but involves tens of thousands of logic gates and memory cells as well as carefully-designed analog circuitry.

The main functional blocks are the USB transceiver at lower left, the USB function controller at lower centre and the UART block at lower right with its full range of data and handshaking inputs and outputs. Notice that there's also an internal 1024-byte EEPROM used to store the USB ID information: the vendor ID, the product ID, the serial number, the power descriptor, the release number and product description strings.

In addition, there are two RAM buffers, one 640 byte USB transmit buffer and one 576 byte USB receive buffer.

Since the CP2102 has a calibrated 48MHz oscillator, it needs no external crystal to operate at the USB 2.0 full-speed rate of 12Mbps. Finally, it contains its own low drop-out (LDO) voltage regulator, to give an output of 3.3-3.45V from an input (REGIN) within the range 4.0-5.25V. This means that it can be powered directly from the USB V_{BUS} line.

Circuit details

While this regulator can supply up to 100mA, the circuitry within the chip itself draws only a little over 26mA (maximum) even in normal operation and only 100µA when suspended. This means it can supply up to 70mA or so for external circuitry needing a 3.3V supply.

In short, the CP2102 is a very impressive chip. Now turn your attention back to the module's circuit of Fig.1. There's a micro-USB socket at the left (CON1) to connect to a PC's USB port via a standard cable and also to power the module itself. So the V_{BUS} line from pin 1 of the socket connects to pins 7, 8 and 9 of the CP2102, with 10µF and 100nF bypass capacitors.

Note that the module does not provide connections to any of the CP2102 UART's handshaking lines, except for DTR ("data terminal ready"). However this is unlikely to pose a problem for most applications nowadays, since even the DTR line is rarely used.

On the right-hand side there's a 6-way pin header (CON2) for the UART input, output and handshaking (DTR) connections, plus the ground, +5V and +3.3V power connections for use by external circuitry. There's also a 100nF bypass capacitor on the +3.3V line, plus three small indicator LEDs, each with its own series resistor for current limiting.

LED1 is driven from pin 11 of the CP2102, the SUSPEND-bar output, so it only glows when the device is not suspended by the host PC, ie, when it's communicating with the PC normally via USB. On the other hand, LED2 and LED3 are connected between the +3.3V supply (pin 6) and pins 26 (TXD) and 25 (RXD) respectively, to indicate when data is being sent and received via the bridge.

LED1 draws a little over 1mA when it's operating while LED2 and LED3 will each draw about 5mA. Thus the LEDs could draw up to 11mA from the 3.3V supply (with full duplex serial communications, allowing LED2 and LED3 to light simultaneously) and this should be taken into account when figuring out how much reserve current is available for external circuitry.

How to use it

Using the CP2102 based USB-UART bridge module is very straightforward. But before you can do so, you may need to install a virtual COM port (VCP) driver on your PC. This is the software which takes care of buffering data to and from the bridge and setting up the UART. In Windows, it makes the UART appear as if it were a legacy COM port.

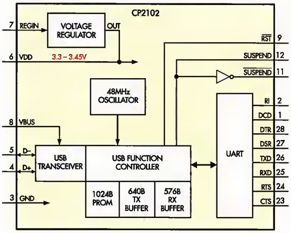


Fig.2: block diagram for the CP2102. This UART interface implements all RS-232 signals, including those for control and handshaking, although an external level shifter is required for full RS-232 compatibility.

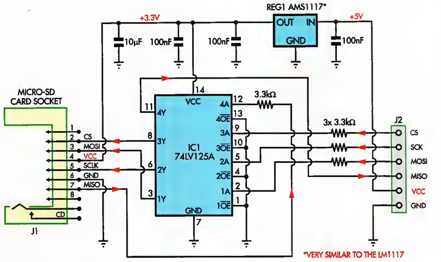


Fig.3: full circuit of the SPI/microSD adaptor module. REG1 reduces the 5V (V_{CC}) input supply from the host module to 3.3V, as required by microSD cards while IC1 similarly reduces signal levels from the micro (which may run off 5V) to the 3.3V signal levels used by the SD card's I/Os.

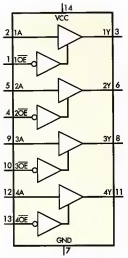


Fig.4: internal block diagram of the SN74LV125A IC. When an $\overline{OE}$ input is pulled high, the corresponding output is disabled and has a high impedance.

You can get the right VCP driver from the Silicon Labs website: www.silabs.com/products/interface/Pages/interface-software.aspx

You can also download the latest version of the CP2102 data sheet from: www.silabs.com/support/Pages/document-library.aspx

When you go there you'll find they can provide VCP drivers for not only Windows 7-10, but also for Windows 2000/XP/Vista/Server 2003, WinCE, Mac OS 9 and X, Linux (3.x.x and 2.6.x) and Android. They can also provide drivers for direct "USB-Xpress" interfacing to the PC, as an alternative to using the VCP approach.

Note that most modern operating systems, including Windows 10 and the latest versions of Mac OS X and Linux, should already have a suitable VCP driver installed. In this case, all you need to do is plug the bridge into a USB port and check that it has been recognised (eg, in Windows, check that a new COM port appears).

Once the driver is installed and working, you can set up your applications to communicate with the module via the new COM port. That includes setting the correct baud rate and other options.

Of course, your circuitry on the UART side of the module needs to be connected to the appropriate pins on header CON2. These will usually be just the RXI, TXO and GND pins, although you might also want to make use of one of the power supply pins as well.

If you aren't sure whether the bridge is working properly, the simplest way to test it is to wire up the RXI pin to the TXO pin. You can then open a terminal emulator, connect to that port and type on your keyboard. The typed characters should be sent back to you and appear in the terminal. If that works, but you still can't communicate with your target device, check that the connections to its TX/RX pins are not swapped and also that you have set the right baud rate.

microSD card interface

There are many different adaptors for accessing an SD memory card from a microcontroller or embedded module but they generally function in the same manner. The main differences are in terms of the card socket they provide and the chip(s) they use for interfacing.

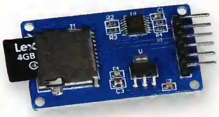
The full circuit for this module is shown in Fig.3. Note that all SD cards can communicate via either serial peripheral interface (SPI) or a faster method, which consists of either a 4-bit parallel bus (older cards) or a high-speed differential interface (UHS-compatible cards). The SPI method is by far the simplest to implement with a microcontroller, unless it has a built-in SD card interface.

The other important thing to note is that all SD memory cards are intended to run from a 3.3V power supply and expect logic signals no higher than +3.3V. Some cards can only accept signals swinging over a smaller range, like 0-1.8V (UHS-I) or 0-0.4V (UHS-II).

Just because a chip has an SPI interface doesn't mean it can necessarily interface directly with an SD card. If the micro operates from a 5V supply, its SPI port(s) may well provide and expect logic high signals above +3.3V. This means that the adaptor is needed both to drop the supply voltage down to 3.3V (assuming a suitable rail is not already available elsewhere) and also to act as a logic level translator for the SPI signals.

The module shown here incorporates LDO regulator REG1 to drop the +5V supply voltage from the micro (via J2) down to the +3.3V needed by both the microSD card at J1, and the single chip (IC1) on the module itself.

IC1 is an SN74LV125A tri-state buffer, to interface between the 5V logic levels (TTL) used on the micro side (via J2) and the low-voltage (0-3.3V) logic levels used on the SD card side (via J1). IC1 operates as a quad non-inverting buffer with tri-state outputs, ie, each output has its own $\overline{OE}$ (out-



This microSD module on a 43 x 24mm PCB is available from the SILICON CHIP online shop at: www.siliconchip.com.au/Shop/7/4019

put enable low) input; see the internal block diagram of Fig.4. The $\overline{OE}$ inputs are not used, they are all tied to ground to enable the buffers permanently.

If you trace the signal paths through the circuit, you'll see that the three outgoing signal lines from the micro's SPI port at J2 (CS [card select], SCK [serial clock] and MOSI [data; master out, slave in]) each pass through a 3.3kΩ isolating resistor (to reduce ringing and provide some static electricity protection) and then through one of the buffers in IC1 to reach the corresponding pin on SD card socket J1.

For example, the 5V MOSI signal enters via J2, passes through its 3.3kΩ resistor and then goes to buffer input 1A (pin 2). The low-voltage logic version of this signal then emerges from the 1Y output (pin 3) and runs to the MOSI pin of J1, the microSD card socket.

The SCK and CS signals are processed via IC1 buffers 2 and 3 in the same way. The path followed by the MISO (data; master in, slave out) signal is similar, the only difference being that in this case the signal is travelling from the microSD card at J1 back to the micro at J2. Note though that the circuit does not level-shift this signal to 5V, so the micro will have to cope with a data input signal that only swings up to around 3.3V; most 5V micros are capable of this.

So the hardware side of the module is quite simple. Having said that, the SD card control protocol is quite complicated and so the software required to drive it is far from trivial.

Putting it to use

Since the module simply provides a transparent bridge linking the microSD card to the SPI port of your microcomputer, the software or firmware in the micro can exchange data with the card using the standard SPI commands. So with an Arduino, you can use commands like:

```
SPI.beginTransaction(SPISettings());
receivedVal = SPI.transfer(val);
SPI.end();
```

There's also an Arduino code library built into recent versions of the Arduino IDE, designed especially for reading from and writing to SD cards. It offers commands like begin(), mkdir(), open(), remove(), rmdir(), available(), close(), write() and read().

With a Micromite it's also fairly straightforward, using commands like:

```
SPI OPEN speed, mode, bits
received_data = SPI(data_to_send)
SPI CLOSE
```

However, the Micromite Plus has built-in library commands specifically intended for reading and writing to SD cards; see the article on Micromite programming on page 58.

Useful links

Information on using standard SPI commands with an Arduino, including some short examples, can be found at: www.arduino.cc/en/Reference/SD

Details on using SPI communications with a Micromite begin on page 92 of the Micromite manual: http://geoffg.net/Downloads/Micromite/Micromite%20Manual.pdf

An article on the SPI bus is available at: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

Wikipedia also has a very informative article on the many kinds of SD cards, at: http://en.wikipedia.org/wiki/Secure_Digital **SC**

Glossary

COM Port: PC communications port, normally sending and receiving data using the RS-232 serial protocol.
CS (Card/Chip Select): used in an SPI bus to indicate when the master wants to communicate with a slave (pulled low).
DTR (Data Terminal Ready): a "flow control" signal which is used to indicate when the serial port is ready to receive data. Other, related flow-control signals include DSR (Data Set Ready), CTS (Clear To Send) and RTS (Ready To Send).
EEPROM (Electrically Eraseable, Programmable Read-Only Memory): non-volatile memory that can be erased and rewritten by applying a higher voltage than is used to read data back. EEPROM is normally more robust than flash.
LDO (low drop-out [regulator]): a regulator which can maintain regulation with less than 2V between its input and output.
Micromite: a Microchip PIC32 programmed with the MMBasic interpreter.
MISO (master in, slave out): the serial data line used to transmit data from the selected slave to the master in an SPI bus.
MOSI (master out, slave in): the serial data line used to transmit data from the master to the selected slave in an SPI bus.
QFN (Quad Flat No-lead): a standard series of surface-mount integrated circuit packages. As the name suggests, it is attached to a PCB without through-holes via lands (pads) on the bottom and sides of the package (ie, without leads).
RS-232 or EIA-232: one of the most common standards for serial communications. Used by the serial ports on older PCs. Uses one wire for self-clocked data in each direction plus optionally, several flow control signals.
RX or RXD: serial data receive line. Normally connected to TX or TXD on the other device.
Serial Communication: the process of transferring data one bit at a time over a communication channel or bus.
SCK (Serial Clock): the shared clock line in an SPI bus, driven by the master, typically up to 20MHz.
SD (Secure Digital): a non-volatile portable storage device utilising flash memory. Successor to MMC (MultiMedia Card).
SPI (Serial Peripherial Interface): a standard serial interface bus, commonly used between a microcontroller and peripherals such as SD cards. Unlike RS-232, SPI has a separate clock line, ie, three wires for bidirectional communications.
TTL (Transistor-Transistor Logic): refers to digital signals with a 5V or (later) 3.3V amplitude, as used in early digital circuits.
TX or TXD: data transmission line. Normally connected to RX or RXD on the other device.
UART (Universal Asynchronous Receiver/Transmitter): circuitry which handles sending and receiving of serial data using one of several different serial protocols or variations thereof.
USB (Universal Serial Bus): high speed serial bus with power (initially using four conductors) which replaced RS-232 and parallel ports for interfacing a PC to pluggable peripherals; from 1.5Mbps up to 5Gbps in the latest version.
UHS (Ultra High Speed): transfer speed for the latest SD cards; up to 104MB/s for UHS-I, and 312MB/s for UHS-II.
VCP (Virtual COM Port): a device driver that emulates an RS-232 serial port over a different protocol such as USB.