SPI to 4 x UART Bridge (MULTIUART)

by assasinsareus on August 21, 2015

Table of Contents

Author:assasinsareus Rowland Technology
Thingiverse Account: http://www.thingiverse.com/BenRowland/about YouTube Channel: http://www.youtube.com/channel/UCPlKfDCmsEMXslHv3z2Jg_A

Intro: SPI to 4 x UART Bridge (MULTIUART)

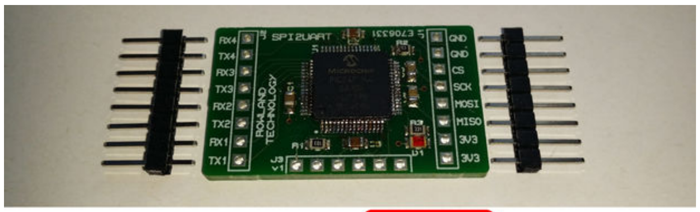
If your a fan of electronics then you like me will often find it annoying on the lack of hardware serial ports on modern devices. Many modules like the Wifi ESP8266 and the Bluetooth HC-06 are available for peanuts but they each require a UART based serial peripheral on your controller to work effectively. In fact a huge range of external electronics can be added to your system via a serial UART connection: GPS, GSM (mobile phone), RFID, RS232, LIN, Ethernet, Zigbee, Modbus, DMX, 4D systems graphical LCDs to name a few more.

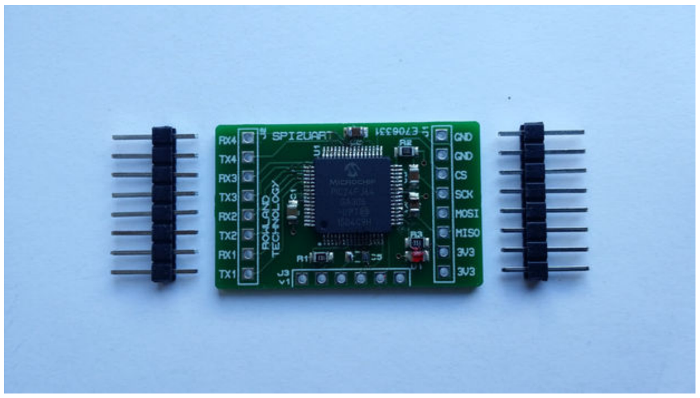
Most modern microcontrollers and devices like the Raspberry Pi have at least one serial UART peripheral so you can do a lot with these devices. However now and then you need to combine several communications style modules together into a single design. A recent project I undertook was a mobile alarm system which used Bluetooth proximity to arm / disarm the system, GPS to track the location, Accelerometer to track movement and GSM based SMS messages to inform the owner where their item is.

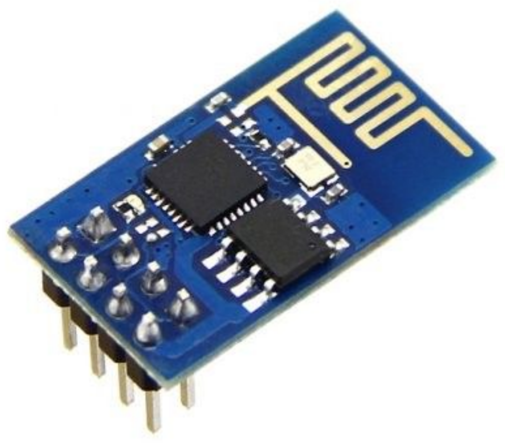
The Arduino Mega 2560 offers two serial UART peripherals but what if that is not enough or you need something more affordable for mass production. To move to a different chip may mean rewriting your entire code so is there an easier way?

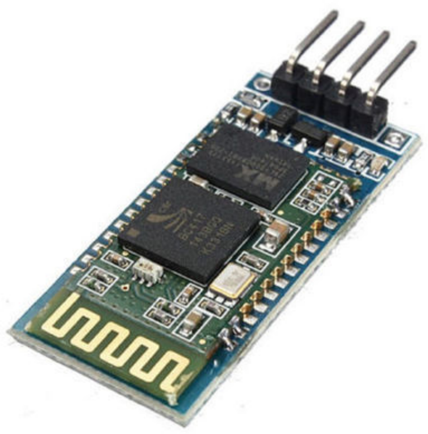
These modern microcontrollers commonly also feature a peripheral named SPI which is typically a lot faster then a UART based serial peripheral and can be used to talk to multiple devices by use of individual chip select signals from the controller. If the controller does not have an SPI peripheral then it can simply be driven using a bit banged software approach using standard I/O pins with no major downfalls. By using the SPI interface and my design you can communicate with up to four serial UART peripherals simultaneously.

Here is a guide to recreate and build my SPI to 4 x UART bridge for use in your own projects. If you don't want to make your own then I also have a limited number of assembled boards available.









Step 1: The schematic

The circuit for the board is pretty basic as we are simply using a larger 16-bit PIC micro controller to do most of the work for us.

U1 is the PIC24FJ64GA306 micro controller, I chose this chip because it has four UART peripherals and was the cheapest I could find by shopping around on Farnell and microchipdirect. The chip also features several 5V tolerant I/O pins meaning that even though the device will need to be powered at 3.3V it will work as is with 5V systems.

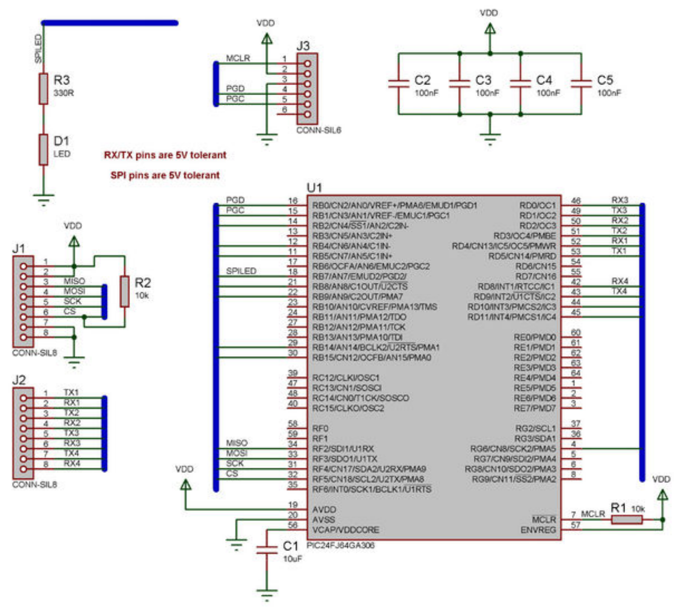
C1 is a 10uF ceramic capacitor and is required by the micro controller to maintain the internal processor voltage levels. This capacitor need to be placed close to the VCAP pin.

C2, C3, C4 and C5 are 100nF ceramic capacitors to provide some voltage smoothing and noise rejection to allow the micro controller to run reliably. These capacitor need to be placed close to the VSS and VDD pins of the micro controller.

R1 provides the MCLR reset signal allowing the micro controller to run. R2 pulls the chip select signal high by default so the SPI interface is automatically disabled until driven low by the host. R3 is a current limiting diode to drive the communications indicator LED.

J1 is the interface to the host controller. It provides connections such as power (3.3V) and ground as well as the four standard SPI pins (MOSI, MISO, SCK and CS). J2 is the interface to the four serial UART peripherals. J3 is the in circuit serial programming (ICSP) header designed to work with the PICkit 3 programming tool.

Note that if your designing your own board then you might want to include other features to be accessed via the SPI interface.



Step 2: The bill of materials

Here is a list of the materials I used in the project.

- 1 x PIC24FJ64GA306
- 2 x 8 way SIL headers 2.54mm pitch
- 2 x 10K Resistor
- 1 x 330R Resistor
- 1 x 10uF Ceramic Cap
- 4 x 100nF Ceramic Cap
- 1 x LED
- 1 x PCB

Also required for the build

- 1 x PICkit 3 programmer
- 1 x 6 way SIL header 2.54mm pitch
- Host controller with SPI interface for testing (I used an ECIO28P from MatrixTSL, an Arduino would also be fine)

Step 3: The PCB

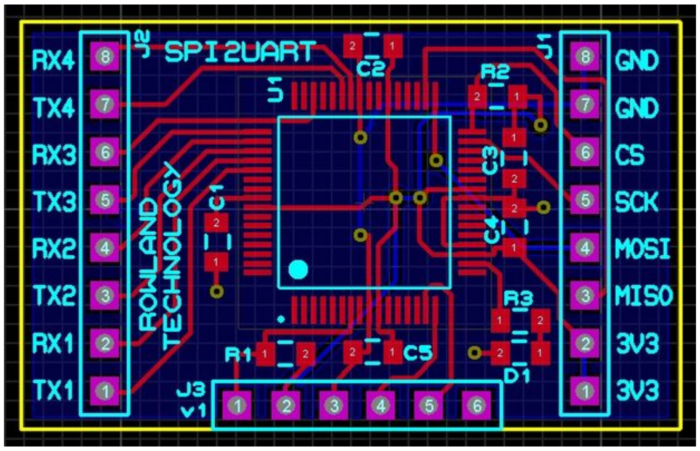
The PCB was created using Proteus and standard 0805 sized surface mount components. Eagle would work just as well if you are familiar with that.

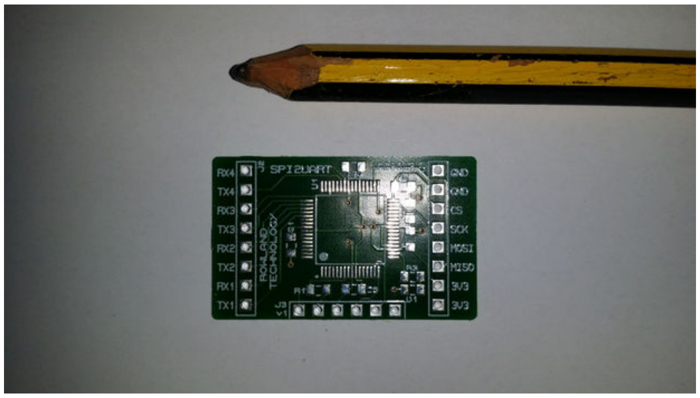
The PCB design was exported as a series of Gerber files and manufactured using Eurocircuits. The Gerber files are attached in the zip file.

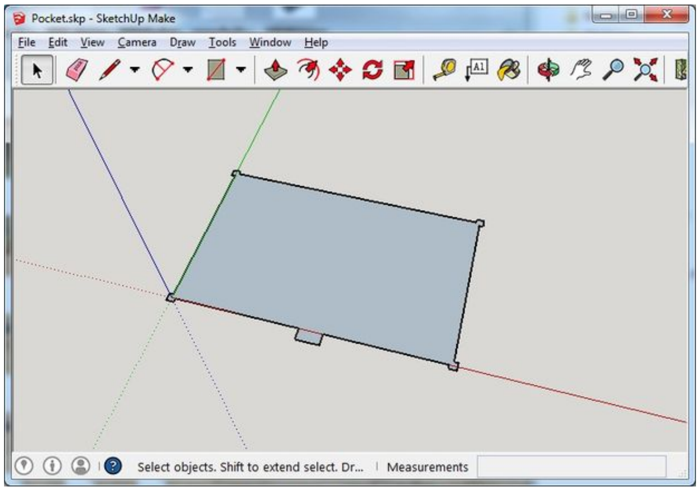
The trick with Eurocircuits is to order in bulk to bring down the price per board. So make sure your circuitry is correct before submitting your design. Of course any other PCB manufacturer should also be able to work with the Gerber files.

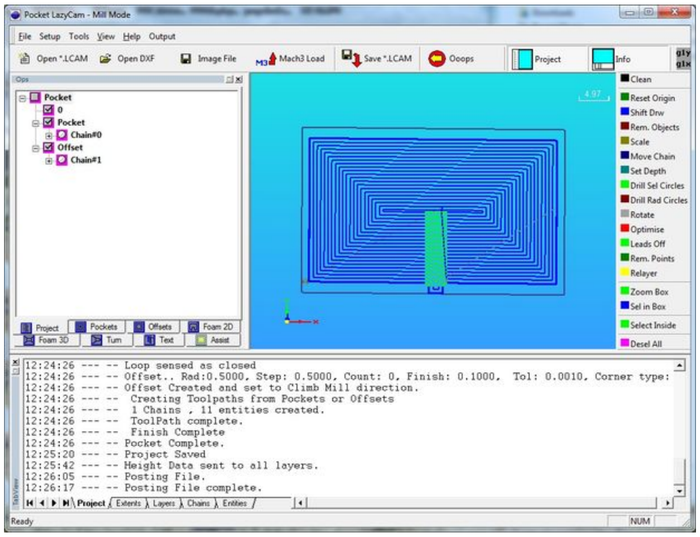
I ordered my PCBs with a top layer surface mount mask to make the manufacturing process easier. I created a jig for the PCBs to sit in by using my trusty CNC machine to create a pocket in a piece of MDF. The pocket needs a small cut out to ensure you can get it out of the hole easily. I placed a PCB into the pocket and used this to line up the solder paste mask. Once the mask is in place use tape to secure one side of the mask to the MDF. This creates a hinge so you can easily lift up the mask to add or remove the PCBs.

To create the code to drive the CNC I used SketchUp to draw a square the size of my PCB. I then added a small slot to make removing the PCB from the pocket easier. The file was then saved and exported as a .dxf CAD file. I then used the LazyCAM software to convert the .dxf file into G-code which will drive my CNC machine.

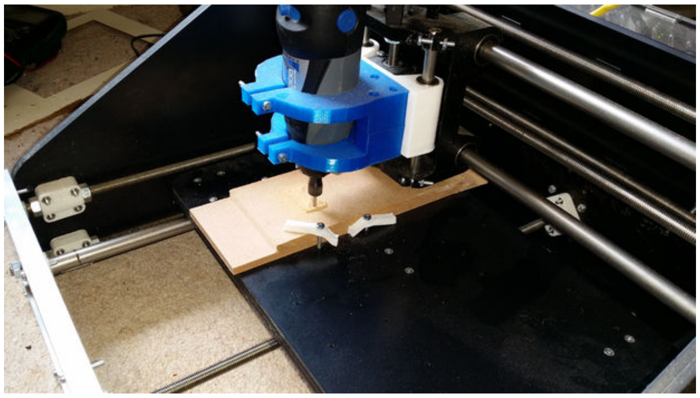


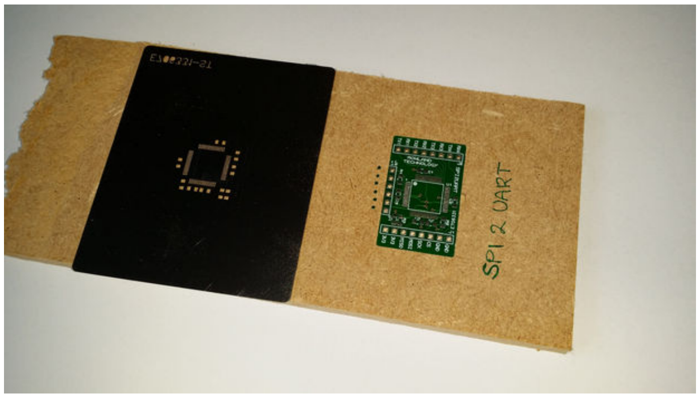



Pocket.skp - SketchUp Make
File
Edit
View
Camera
Draw
Tools
Window
Help
Select objects. Shift to extend select. Dr...
Measurements



```
12:24:26 --- -- Loop sensed as closed
12:24:26 --- -- Offset.. Rad:0.5000, Step: 0.5000, Count: 0, Finish: 0.1000,  Tol: 0.0010, Corner type:
12:24:26 --- -- Offset Created and set to Climb Mill direction.
12:24:26 --- --  Creating Toolpaths from Pockets or Offsets
12:24:26 --- --  1 Chains , 11 entities created.
12:24:26 --- --  ToolPath complete.
12:24:26 --- --  Finish Complete
12:24:26 --- -- Pocket Complete.
12:25:20 --- -- Project Saved
12:25:42 --- -- Height Data sent to all layers.
12:26:05 --- -- Posting File.
12:26:17 --- -- Posting File complete.
```





File Downloads

MultiUART - CADCAM.ZIP (19 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'MultiUART - CADCAM.ZIP']

CNCJIG.zip (9 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'CNCJIG.zip']

Step 4: The software

The software was created using MPLAB X and the XC8 compiler. Flowcode does not natively support SPI slave just yet so I had to go back to basics with my C compiler.

Flowcode 6 does provide some nice features out of the box such as multiple UART support and circular buffers so this really helped to speed up the development in terms of generating C code i could copy and paste into my program.

I go through the process of obtaining a completely unlocked 30 day version of Flowcode for all chip types in my Two wheel robot Instructable.

To develop the software I created a simple protocol for the SPI connection to allow me to access each of the UART channels functionality. Each value is sent via the SPI as a byte with the CS signal pulled low. When the command is complete the CS signal is pulled high.

The command code and UART channel are packed together into a single byte to increase efficiency. The command code resides in the top 4 bits and the channel resides in the bottom 2 bits.

Here are the bits in the byte and their representation.

cccc00uu

where cccc is the command code and uu is the UART channel.

Command Code - Parameters - Returns - Description

0x10 - N/A - NumBytes (0-255) - Reads the number of bytes in the channel receive buffer.

0x30 - N/A - NumBytes (0-255) - Reads the number of bytes in the channel transmit buffer.

0x20 - NumBytes (0-255) - DataByte (0-255) - Reads data byte(s) from the channel receive buffer.

0x40 - NumBytes (0-255), DataByte (0-255) - N/A - Puts a data byte into the channel transmit buffer.

0x80 - Baud (0-7) - N/A - Sets the channel baud rate

Here are the options for the baud rate parameter.

0=1200, 1=2400, 2=4800, 3=9600, 4=19200, 5=38400, 6=57600, 7=115200

So to read the number of bytes in UART channel 2 receive buffer we send the following command code and then perform a read.

CS Low

SPISend (0x12)

NumBytes = SPIReceive (0xFF)

CS High

To read 5 bytes from UART channel 2 receive buffer we send the following command code, the number of bytes and then perform enough reads to pull out all the data we want.

CS Low
SPISend (0x22)

SPISend (0x05)

Byte[0] = SPIReceive (0xFF)

Byte[1] = SPIReceive (0xFF)

Byte[2] = SPIReceive (0xFF)

Byte[3] = SPIReceive (0xFF)

Byte[4] = SPIReceive (0xFF)

CS High

To write 5 bytes to UART channel 2 transmit buffer we send the following command code, the number of bytes and then perform enough writes to send out all the data we want.

CS Low

SPISend (0x42)

SPISend (0x05)

SPISend (Byte[0])

SPISend (Byte[1])

SPISend (Byte[2])

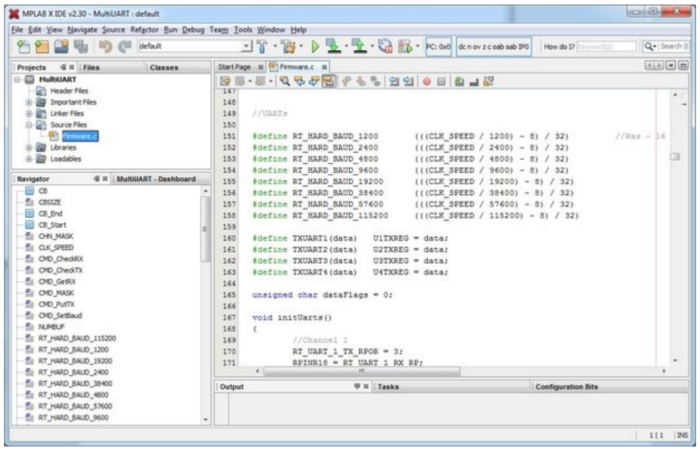
SPISend (Byte[3])

SPISend (Byte[4])

CS High

Each UART peripheral has a software 512 byte circular buffer allowing a large amount of data to be sent and received which should hopefully mean that you never loose a byte.

The LED glows dimly when the board is powered, glows at half power when transmitting or receiving via a UART and glows at full power when communicating via SPI. I thought about having a LED for each UART channel but decided against it in the end, just more components to place by hand.



```
//UARTs

#define RT_HARD_BAUD_1200       (((CLK_SPEED / 1200) - 8) / 32)        //Was - 16
#define RT_HARD_BAUD_2400       (((CLK_SPEED / 2400) - 8) / 32)
#define RT_HARD_BAUD_4800       (((CLK_SPEED / 4800) - 8) / 32)
#define RT_HARD_BAUD_9600       (((CLK_SPEED / 9600) - 8) / 32)
#define RT_HARD_BAUD_19200      (((CLK_SPEED / 19200) - 8) / 32)
#define RT_HARD_BAUD_38400      (((CLK_SPEED / 38400) - 8) / 32)
#define RT_HARD_BAUD_57600      (((CLK_SPEED / 57600) - 8) / 32)
#define RT_HARD_BAUD_115200     (((CLK_SPEED / 115200) - 8) / 32)

#define TXUART1(data)   U1TXREG = data;
#define TXUART2(data)   U2TXREG = data;
#define TXUART3(data)   U3TXREG = data;
#define TXUART4(data)   U4TXREG = data;

unsigned char dataFlags = 0;

void initUarts()
{
        //Channel 1
        RT_UART_1_TX_RPOR = 3;
        RPINR18 = RT_UART_1_RX_RP;
```

File Downloads

Firmware.zip (72 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'Firmware.zip']

Firmware.c (17 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'Firmware.c']

Step 5: Programming

Programming was done by inserting a 6 way SIL header into the PICkit 3 and then placing this header into the 6 way socket on the PCB. The arrow on the PICkit should go to pin 1 of the ICSP header which is the side marked J3 on my PCB design.

The board can be powered externally or via the PICkit 3. Note the boards needs some method of powering before the PICkit will see the device.

Step 6: Testing

To test the board I used the 5V PIC based ECIO28. I created a simple test jig using veroboard with sockets for both the ECIO and the MULTIUART bridge. The ECIO28 is a 5V device so I used a 3.3V low dropout (LDO) voltage regulator to provide the voltage to power the MULTIUART.

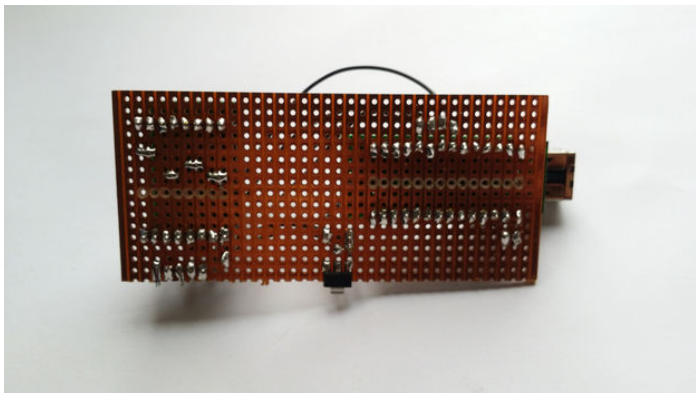
Each of the UART channels has their TX to RX pins shorted together so that anything sent is automatically echoed back. This allows us to automatically test the send and receive functionality of each of the UARTs to ensure everything is working as it should.

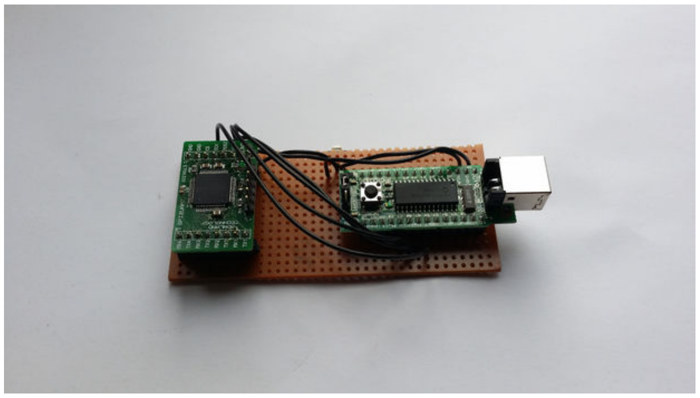
The ECIO features USB communications so I can use the Flowcode 6 simulation to communicate with the device and see how the testing is going. Arduino's also have a USB interface so they could also do this type of automated testing using the Flowcode software. Alternatively you could use an LCD or other display to show if the test has passed or failed without the need of a computer.

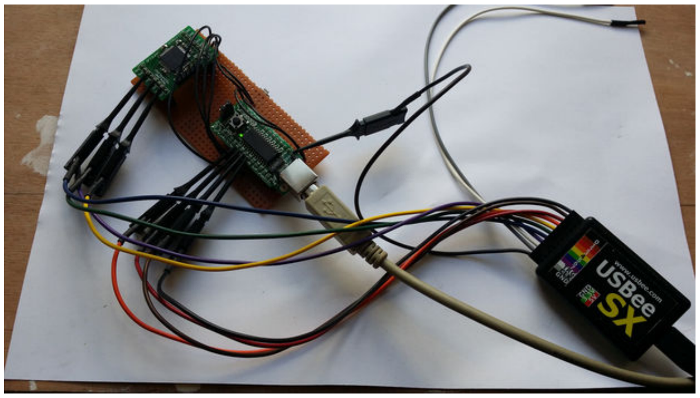
The nice thing about using the Flowcode software is once you past the testing stage you then have effectively an interface to your four serial devices via USB to SPI to UART and back again. You could even use multiple MULTIUART boards to add in as many serial devices as you need. The same thing could be done using Python if you wanted to do the testing or interface using a Raspberry Pi as the controller.

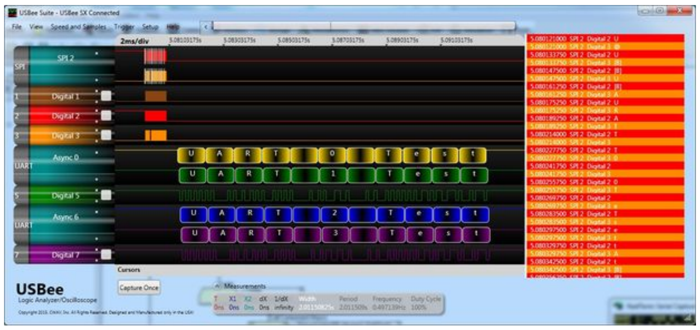
As a fall back I also have access to a USBee SX which is a very nice piece of hardware for diagnosing problems with serial interfaces. It can decode serial, SPI, I2C as well as others and has helped me out time and time again to resolve problems with digital communications. Shown in the images is my initial debugging rig and screenshots of the software decoding the communications busses. You can really see the difference between the speed of the SPI and the speed of the serial data at 9600 baud.

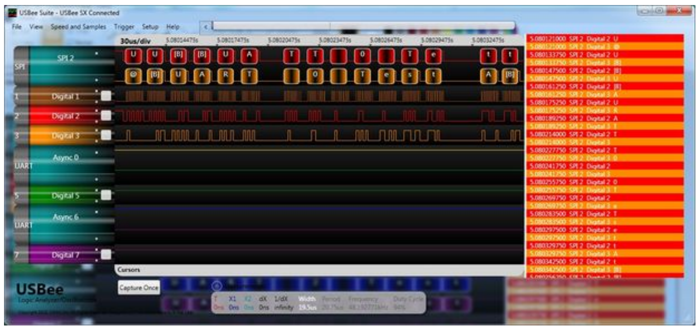
RealTerm is another handy tool. In the image I show the data coming in via the ECIO28P USB connection. The Flowcode test project file and hex are also included. To convert this for use with an Arduino simply swap the USB component to a RS232 component and change the target in the project options.

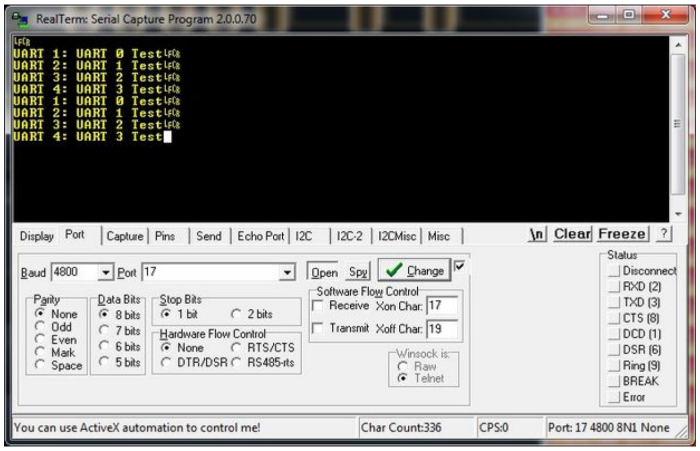











RealTerm: Serial Capture Program 2.0.0.70

```
LFCR
UART 1: UART 0 TestLFCR
UART 2: UART 1 TestLFCR
UART 3: UART 2 TestLFCR
UART 4: UART 3 TestLFCR
UART 1: UART 0 TestLFCR
UART 2: UART 1 TestLFCR
UART 3: UART 2 TestLFCR
UART 4: UART 3 Test
```

Baud 4800 Port 17
Parity: [✓ None] Odd Even Mark Space
Data Bits: [✓ 8 bits] 7 bits 6 bits 5 bits
Stop Bits: [✓ 1 bit] 2 bits
Hardware Flow Control: [✓ None] RTS/CTS DTR/DSR RS485-rts
Software Flow Control: Receive Xon Char: 17 Transmit Xoff Char: 19
Winsock is: Raw [✓ Telnet]

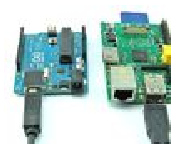
You can use ActiveX automation to control me! | Char Count:336 | CPS:0 | Port: 17 4800 8N1 None

File Downloads

ECIO28P Test.zip (9 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'ECIO28P Test.zip']

Related Instructables

Raspberry Pi & the Neo 6M GPS by sspence


Turn Raspberry Pi ON w/ Remote Control by Dalton63841

SmartBox powered by BeagleBone by mdemirst

Smart home automation webserver on OpenWRT router WR703N interfaced to Arduino, compared to Raspberry Pi and Ubuntu by janisalnis

smart geo-location tracker by rajasekher.mutukui

Solar Powered GPS Hiking Logger by spyclub

Comments