FTDI SmartBasic Hookup Guide

Introduction

The FTDI SmartBasic is a variation on our classic FTDI Basic which allows you to connect a second device to the hardware serial port on an Arduino Pro, Pro Mini, or other USB-less board without compromising the ability to bootload code from the Arduino IDE.

Normally, to use a device which requires a serial port resource on an Arduino board, one must either use a software serial port or plug and unplug the device during programming. The SmartBasic board adds a multiplexer to the serial port pins coming from the Arduino, which allows the application code to switch the serial signals from the USB port to another device. No special code is required to enable programming, either!

Suggested Reading

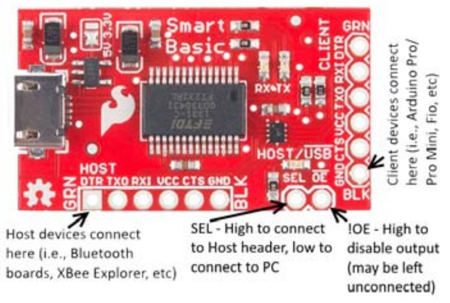
Before we get started, you might want to review these other tutorials:

- Logic levels- Setting the jumper on the FTDI SmartBasic to the wrong voltage level may cause damage to one or more of the boards hooked up to it.
- Serial communications - The FTDI SmartBasic is a device for multiplexing serial signals; learn a bit more about serial data and how it works with this tutorial.
- Installing FTDI Drivers

Hardware Tour

The FTDI SmartBasic hardware is pretty simple. It routes the serial signals from an Arduino Pro, Pro Mini, Fio, or LilyPad board (along with any other board which uses the standard FTDI header footprint) either to the programming PC via a USB-to-serial bridge or to any other device with the FTDI Basic-type header. It uses the venerable FT232RL chip used on the original FTDI Basic boards and the TS3USB221A signal multiplexer from TI to make connecting to multiple serial devices easy.

The Board


Smart Basic
5V 3.3V
CLIENT
GRN DTR RXI TXO VCC CTS GND BLK
RX TX
HOST/USB
SEL OE
HOST
DTR TXO RXI VCC CTS GND
GRN
BLK
Client devices connect here (i.e., Arduino Pro/ Pro Mini, Fio, etc)
Host devices connect here (i.e., Bluetooth boards, XBee Explorer, etc)
SEL - High to connect to Host header, low to connect to PC
!OE - High to disable output (may be left unconnected)

The actual board design is fairly compact. We've left the headers off, so you can choose the header most appropriate for your application.

The header labeled "CLIENT" is basically the same as the output header on a standard FTDI Basic board. If you connect that header to the Arduino as you would with a normal Basic, you can program the Arduino exactly as you would normally, and never notice a difference.

The benefit comes in with the "HOST" header. That header can be connected to any host-type device (such as another FTDI board, or any of our Bluetooth Mate type boards). You can then wire the !OE and SEL lines to pins on the Arduino to enable the application code to route serial data from the hardware port to either the USB serial bridge *or* the device connected to the HOST header.

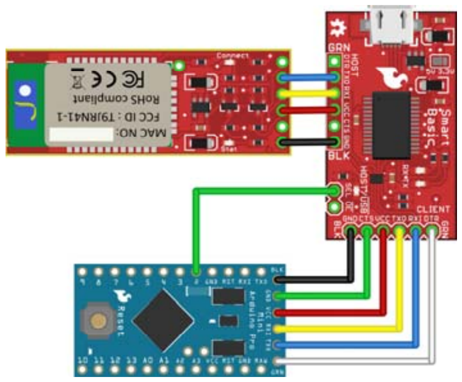
Because of the pull-down resistor on the SEL line, if that pin is left floating, the default destination for the traffic is the USB serial bridge. When in bootloader mode, all non-serial pins will be high impedance inputs, so after the Arduino IDE resets the Arduino board the bootloader and the PC will be able to communicate until the application loaded changes the level of that pin.

The AUX/!USB LED will be lit when the HOST port is selected and off when the data is being routed to the USB serial bridge. There is a solder jumper which can be adjusted to change the voltage on the VCC pins on the two headers (and the IO voltage on the FT232RL chip) from 3.3V to 5V; if that jumper is cleared completely, a supply on one of the two serial headers can be used to power both devices and the VDDIO, should you need a voltage other than 5V or 3.3V.

Connecting the SmartBasic

One of the common problems encountered when developing with a serial-connected Bluetooth dongle is the inadequacy of the Arduino SoftwareSerial library. Transmissions with software serial are resource intensive, blocking the processor for the duration of the transmission. Long receives easily overrun the buffer, and can throw off the internal clocks used for millis() and micros().

Obviously, it's desirable to use the hardware serial port, if possible, as it bypasses most of these issues. However, connecting anything other than an FTDI-type serial port to the hardware serial port header prevents it being used for loading code unless the other device is removed. Here's a diagram showing how to connect the FTDI SmartBasic in a way that removes that problem.



You can see that the connections from the Bluetooth Mate and the Arduino Pro Mini are straight-through; no need to cross wires, so you can plug them right in. Also note the connection of digital I/O pin 2 to the SEL line on the FTDI basic. This is what allows the multiplexing of the serial data: when the board is in bootloader mode, that pin will be a digital input and the SEL line will be pulled low by a pull down resistor on the SmartBasic. That will route the data to the FTDI chip to be sent to the PC, and data from the FTDI chip will be routed to the Pro Mini board, and bootloading of a sketch can occur normally.

Note, as well, that the CTS and DTR pins between the SmartBasic and the Bluetooth Mate are not connected. Since the multiplexer on the SmartBasic only has two channels, only the data channels can be swapped. That's important, though, because DTR is needed to reset the Arduino at bootload time. If it were being re-routed, that would defeat the purpose of this board.

After the application sketch has loaded, the user can switch between the two data endpoints (the PC and the Bluetooth Mate) by asserting pin 2 high (for the Bluetooth Mate) or low (for the PC). Here's a simple Arduino sketch showing that in action.

```
#define SEL 2  // When the SEL pin is held low, the data will
be
               //  routed to the PC via the USB-serial bridge.
               //  That port is also the port used for program
ming
               //  by the Arduino IDE. When in bootloading mod
e, a
               //  pull-down resistor on the SmartBasic cause
s it
               //  to remain in programming mode.

#define ARDUINO_IDE   LOW  // Constants to make our routing ch
ange
#define AUX_TERMINAL  HIGH //  more obvious. When the SEL pin
is
                           //  LOW, data is routed to the
                           //  programming port.

void setup()
{
  Serial.begin(115200);    // Set up the hardware serial port.

  pinMode(SEL, OUTPUT);    // Make the select line an outpu
t...
  digitalWrite(SEL, ARDUINO_IDE); // ...and connect the board
to
                           //  the Arduino IDE's terminal.
}

void loop()
{
  // The loop just says "Hello" to the two terminals, over and
  //  over, forever. Note the use of the "flush()" function. I
f
  //  omitted, the Arduino will re-route the serial data befor
e
  //  the transmission has been completed; flush() causes the
  //  Arduino to block until the serial data output buffer is
  //  empty. Failure to use flush() will result in data being
  //  sent to the wrong device, or to multiplexer changes duri
ng
  //  transmission which may cause framing errors or data
  //  corruption. *Always put in a flush() before you change
  //  destination devices or disable the output.*
  Serial.flush();
  digitalWrite(SEL, ARDUINO_IDE);
  Serial.println("Hello, Arduino IDE!");

  // Swap to the non-Arduino terminal and say hello.
  Serial.flush();
  digitalWrite(SEL, AUX_TERMINAL);
  Serial.println("Hello, auxilliary terminal!");

  Serial.flush();

  delay(500); // This is a rate-limiter only. The temptation t
o use
              //  delay() instead of flush() is strong, but fi
ght it.
              //  If you use delay, you will *certainly* make
a change
              //  to the code which makes the original delay t
ime too
              //  short for the new serial data stream, result
ing in
              //  data corruption. flush() will *always* be th
e right
              //  length.
}
```

Finally, I’ve omitted discussion of the OE pin. It can be left unconnected during normal use; however, if for some reason it becomes useful to disconnect the TX and RX pins on the SmartBasic from the client board, that pin can be asserted HIGH, which will put the client-side pins on the multiplexer into a high-impedance mode.

Further Reading

- Installing the bootloader - This tutorial will help you install, or re-install, the bootloader on an Arduino board (or a bare Atmega328P).
- Bootloader development - More information on how the bootloader works and how to add or remove features from it.
- How the bootloader works - Arduino forum post regarding how the bootloader does its thing.