RTArduLink

A General Purpose Subsystem Interconnect

Introduction

RTArduLink is a system that makes it easy to connect Arduinos to host systems such as a PC, Raspberry Pi, BeagleBoard, PandaBoard etc. Why do this? Well in many cases there's a need to interface to hardware that is not compatible with the host system or else there's a convenient Arduino shield available. Arduinos can connect to host systems using a USB (FTDI style) interface or in some cases a real serial interface, perhaps via a radio link like Zigbee. RTArduLink provides a convenient way of connecting one or more of these subsystems with minimal programming, both on the subsystem and the host.

RTArduLink Features

- Allows one or more subsystems (Arduinos) to be connected to the host processor and provides a unified software interface to ease application development.

- The RTArduLink API for the host can handle up to 8 subsystems directly connected via USB or serial ports.

- Each of those 8 subsystems may have daisy-chained subsystems connected to them indirectly. 16 bit address allow for potentially 65534 total subsystems.

- The specific topology of the subsystem connection is transparent to the host application as it just addresses subsystems using identity strings configured in to the subsystems.

- The host API uses the Qt system and has been tested on Ubuntu and Windows.

- The subsystem API is available for the Arduino and is easily ported to other subsystems by providing a custom Hardware Abstraction Layer (HAL) for the new subsystem.

- The RTArduLink Arduino HAL has been tested on the UNO and the Mega 2560.

RTArduLink Communications

Frame Structure

RTArduLink uses a proprietary and highly efficient frame structure to transfer arbitrary messages between the host and the subsystem. The frames are a maximum of 64 bytes in length and consist of a four byte frame header that's managed by RTArduLink and a four byte message header followed by up to 56 bytes of user data.

The message structure (defined in RTArduLinkDefs.h) has the following fields:

- **messageAddress**. This is a two byte field that contains the address of the target

subsystem. For host to subsystem messages, this is the address of the target subsystem. For subsystem to host messages, this is the address of the subsystem that generated the message.

- **messageType**. This is a one byte field that contains the message type. Valid types are defined in RTArduLinkDefs.h.

- **data**. This is the user data field and can contain up to 56 bytes.

Addressing Subsystems

Subsystem addresses and dynamically assigned by RTArduLink based on the topology of connection. In fact, the host uses a port/address pair to identify a specific subsystem. This is because there can be up to 8 separate serial ports in use, each port identified by a port number. On each port, there can be up to 65534 subsystems, identified by an address.

For example, if there are 8 directly connected subsystems, these will be identified as 0/0, 1/0, 2/0, 3/0, 4/0, 5/0, 6/0 and 7/0. If there were three subsystems daisy-chained on port 0 and two daisy-chain on port 1, these would be identified as 0/0, 0/1, 0/2, 1/0 and 1/1.

Host applications cannot assume that the address or port is fixed as subsystems maybe connected in different ways at different times. Instead, hosts use "identity" strings to identify specific subsystems. The host application can look up an identity string and obtain the port number and address of the subsystem before sending a message.

There is a special broadcast address (all 1s) that allows a host to send the same message to all connected subsystems. This is used by the Poll and Identity mechanisms and can be used by custom messages also.

The Poll System

The host RTArduLink library sends out a Poll message once per second. It is a broadcast message and so goes to all subsystems. Subsystems respond with a poll message. The host library uses this to check that the subsystem is present and working.

The Identity System

This is similar to the Poll system except that it is sent out every 5 seconds. Subsystems respond with their programmed identity string. The host library records this so that the host application can look up known identity string and map them to port/address pairs.

Message Rate Management

There isn't any! Typically, the host will send a message and a subsystem will respond. Once the subsystem has responded, the host is free to send another one, either then or later (this is how the echo test works). Alternatively, the subsystem can regularly send a message – something like a status update. RTArduLink does not impose any flow control so it is up to the application to consider things like buffer sizes and link speeds in their design.

Subsystem Daisy-Chain Configurations

Daisy-chained subsystems can only be built with subsystems that have more than one hardware port. Port index 0 is always the upstream port, any other ports can be daisy-chain ports. In addition, port index 1 can have a subsystem connected that itself has daisy-chained systems. Any

other ports cannot support a subsystem that itself has daisy-chained ports.

In practice it is most efficient to directly connect subsystems to the host wherever possible. However, if the host runs out of USB ports (or else the connection is serial over a radio link like Zigbee), daisy-chaining can be a great solution.

Installing RTArduLink

Download the Source

RTArduLink is available from GitHub and can be obtained using the git command:

```
git clone git://github.com/richards-tech/RTArduLink.git
```

The download consists of the following subdirectories:

- **Host**. This contains the host side programs.

    - **HostTest**. This is the host side of the echo test system. It is able to send continuous echo requests with full size data payloads to all detected subsystems. It demonstrates how to use the host RTArduLink API, specifically HostTestEcho.cpp and HostTestEcho.h. These subclass the RTArduLinkHost class and implement the required overrides.

    - **HostDemo**. This is the host side of the PWM, servo and GPIO demo. The demo provides GUI controls that allow PWM, servo and GPIO on the subsystem to be controlled. The demo needs the corresponding code loaded onto the subsystem.

    - **RTArduLinkHost**. This contains essential host libraries for the RTArduLink system.

    - **qextserialport**. This is a 3rd party library that provides serial port interfacing for Qt. It is required by the RTArduLinkHost library.

- **Arduino**. This contains the subsystem (Arduino) programs and HAL.

    - **libraries**. This contains the Arduino and HAL library.

    - **RTArduLinkDemo**. This is the subsystem side of the GPIO, servo and PWM demo code and works with both HostTest and HostDemo apps.

    - **RTArduLinkConfig**. This is a stand-alone program for the Arduino that allows its RTArduLink configuration to be defined and stored in EEPROM. It manages the subsystem's identity string and serial port configuration (including daisy chain port configurations).

Compiling the Host Example and Libraries on Windows using Visual Studio

First of all the Qt must be installed. This can be obtained here:

```
http://qt-project.org/downloads
```

Then the Qt plug-in for Visual Studio can be installed the same location.

Once these have been installed, the solution files for HostTest and HostDemo can be use to build the programs.

Compiling on Ubuntu 14.04

To install on Ubuntu, Qt5 should be installed. For example, using a terminal window, enter:

```
sudo apt-get install qt5-default
```

To compile HostTest or HostDemo, just navigate to the relevant directory and execute:

```
qmake
make
sudo make install
```

Compiling on Raspberry Pi/Raspbian

To install on the Raspberry Pi, Qt4 should be installed. For example, using a terminal window, enter:

```
sudo apt-get install libqt4-dev
```

To compile HostTest or HostDemo, just navigate to the relevant directory and execute:

```
qmake
make
sudo make install
```

Preparing a Subsystem

Which Board?

RTArduLink has been tested on the Arduino UNO and Mega 2560 Arduinos. For direct connection to the host via USB and with no dasiy-chaining, any of these Arduinos can be used. RTArduLink can only work with hardware serial ports so UNO Arduinos can only be used either as directly connected subsystems or the end of a daisy-chain as they only have one hardware serial port. Mega 2560s can have three daisy-chained subsystems connected as they have a total of four hardware ports (one port is used for the upstream connection).

The Arduino IDE

Arduino IDE 1.0.6 should be used. It can be downloaded from here:

http://arduino.cc/en/Main/Software

The best thing is to set the sketchbook location to be the root directory to be <path to RTArduLink>/Arduino. This way, the libraries will be recognized without further configuration.

If it is intended to use daisy-chaining, it is advisable to increase the size of the HardwareSerial transmit buffer to prevent message loss under high loading (such as the echo test). The file is:

```
<path to Arduino>/hardware/arduino/cores/arduino/HardwareSerial.cpp
```

The code defining the size of the buffer should be changed to that below:

```
#if (RAMEND < 1000)
#define SERIAL_BUFFER_SIZE 16
#elif (RAMEND < 2400)
   #define SERIAL_BUFFER_SIZE 64
#else
   #define SERIAL_BUFFER_SIZE 256
#endif
```

This will change the size to 256 bytes on Arduino Mega 2560s for example. 256 bytes should be enough in most cases but the actual size is dependent on the application.

Configuring the Subsystem

(Note that the screenshots here were made with an earlier version of this software but the details are the same except for name changes.)

The RTArduLinkConfig program should be compiled and uploaded to the subsystem. Use the serial monitor in the IDE to communicate with the subsystem:
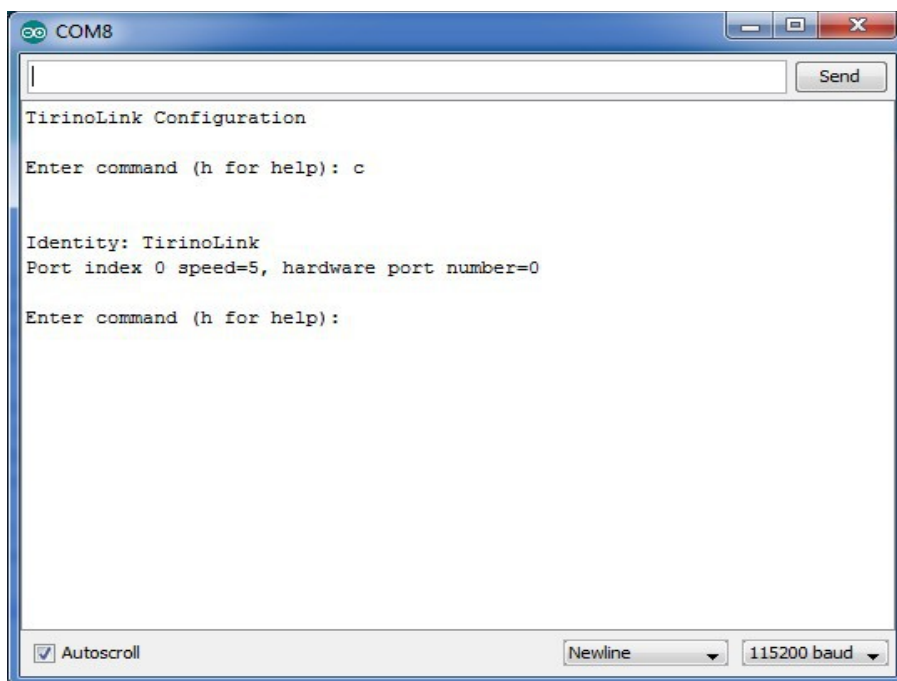
Note that "Newline" has been set and the speed is 115200 baud. Typing "c" followed by return results in the screenshot above. This shows the default configuration. In many cases, this is exactly what's required. Port index 0 is assigned to hardware port number 0 (the USB serial port) and set to speed 5 (115200).

Enter "h" and return to see the help message:
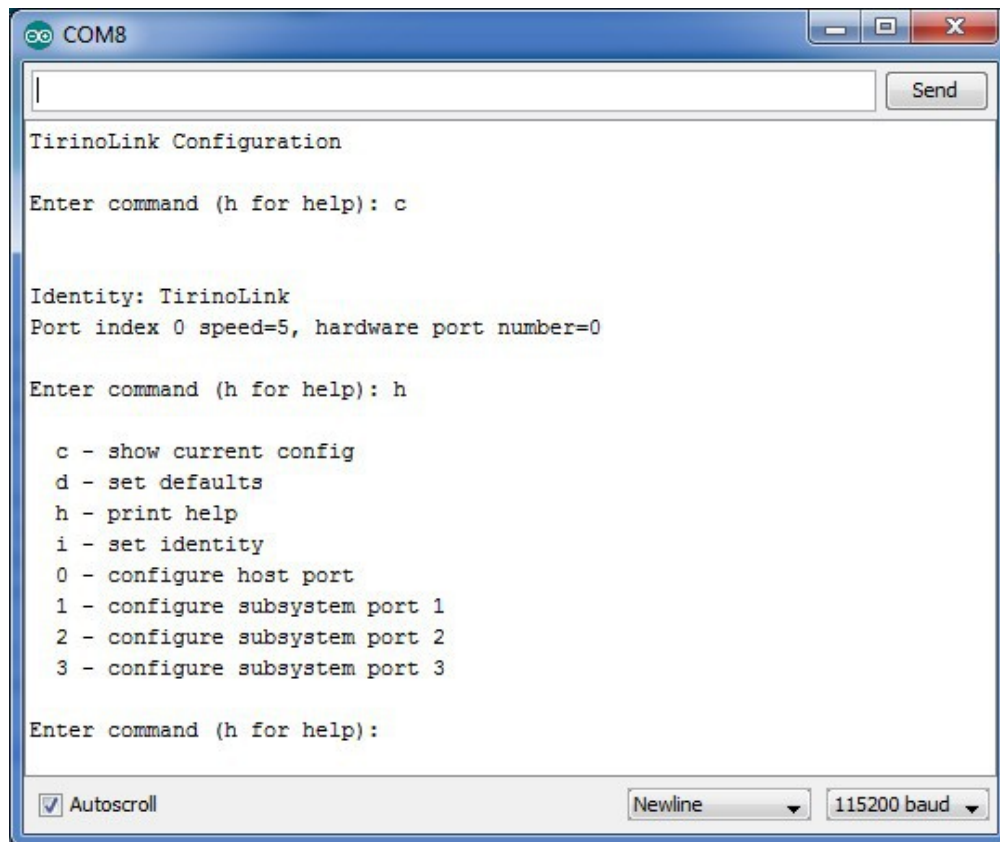
Entering "0" and return results in the following screen:
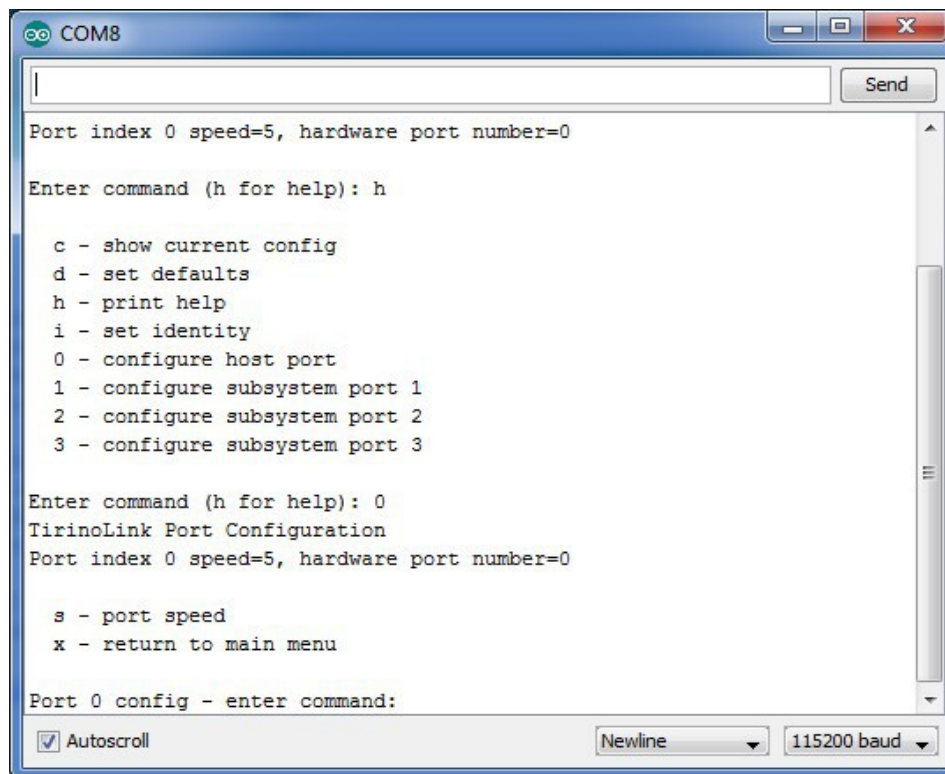
The speed of port 0 can be configured here. Type "x" followed by return and then "1" followed by

return to get the port 1 config screen:
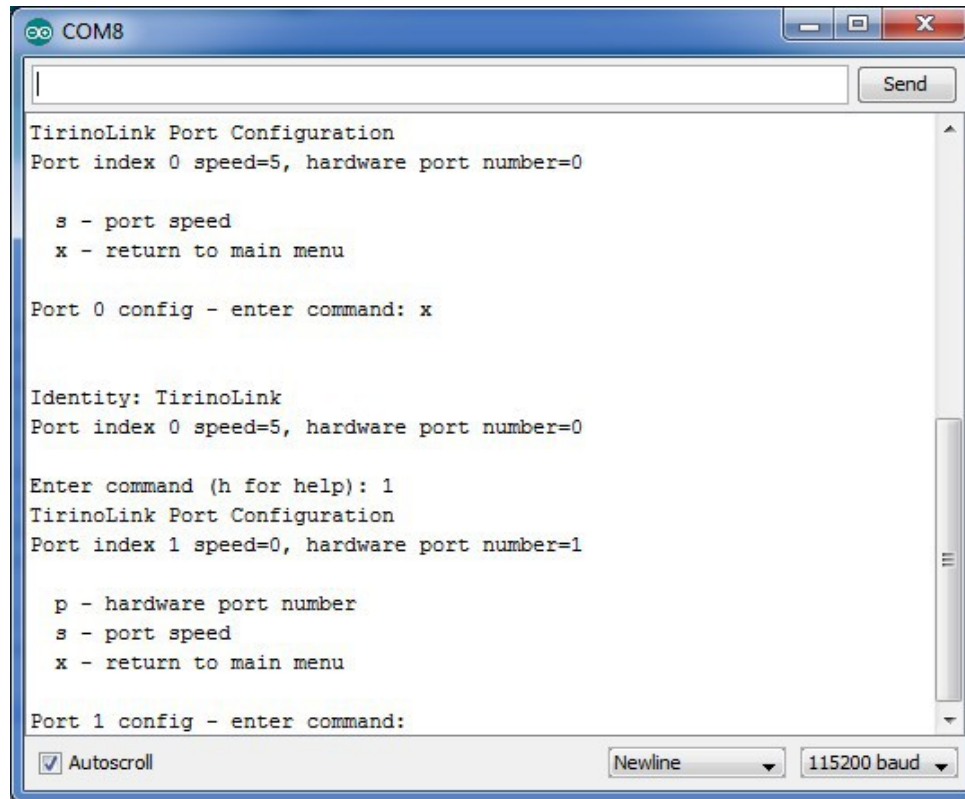
Here, the hardware port number assigned to port index 1 can be selected, as can the port speed.
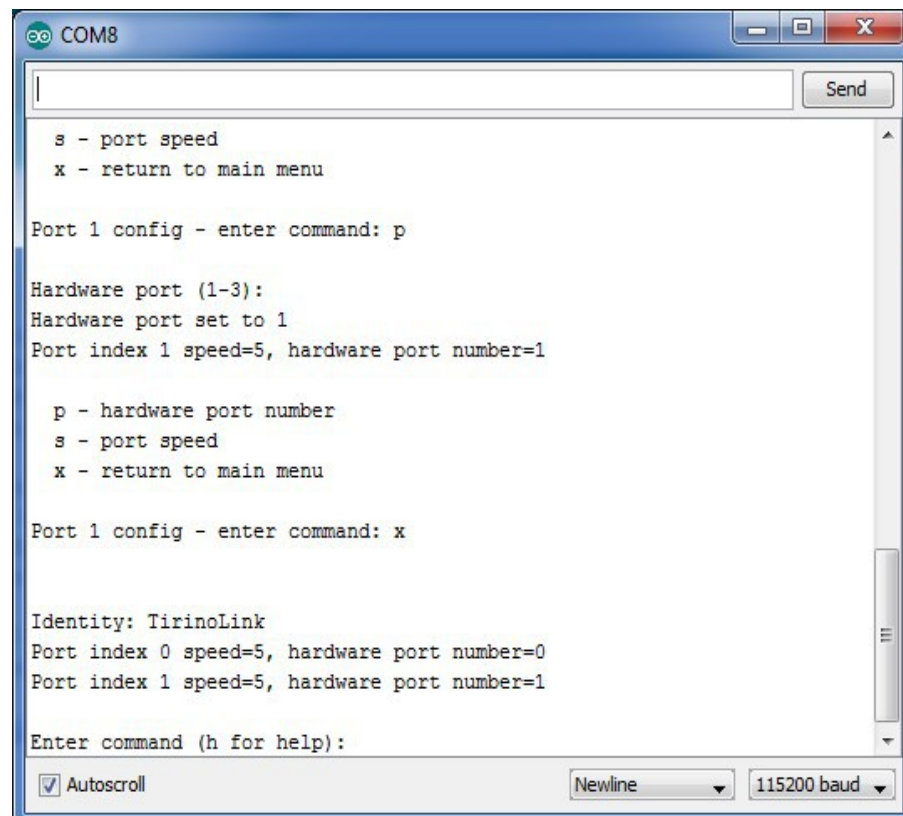
In this screenshot, port index 1 has been configured to be hardware port 1 running at 115200 baud. Port indices 2 and 3 can be configured this way if required. Ports can be turned off by changing their speed to code 0. Finally, it is important to configure the identity to be a unique string:
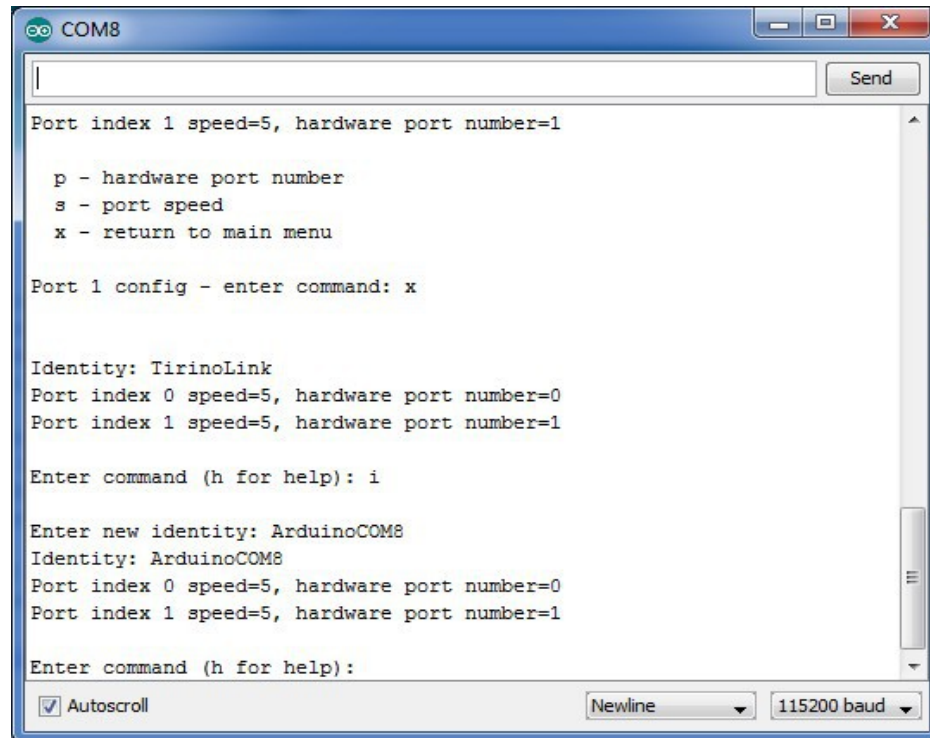
Enter the identity string followed by return.

RTArduLinkConfig saves configuration data in EEPROM as the changes are made. So, everything is now ready for the test program.

Running HostTest

Compile and upload the RTArduLinkDemo program to the subsystem. On the host, run HostTest. The screen will look like:
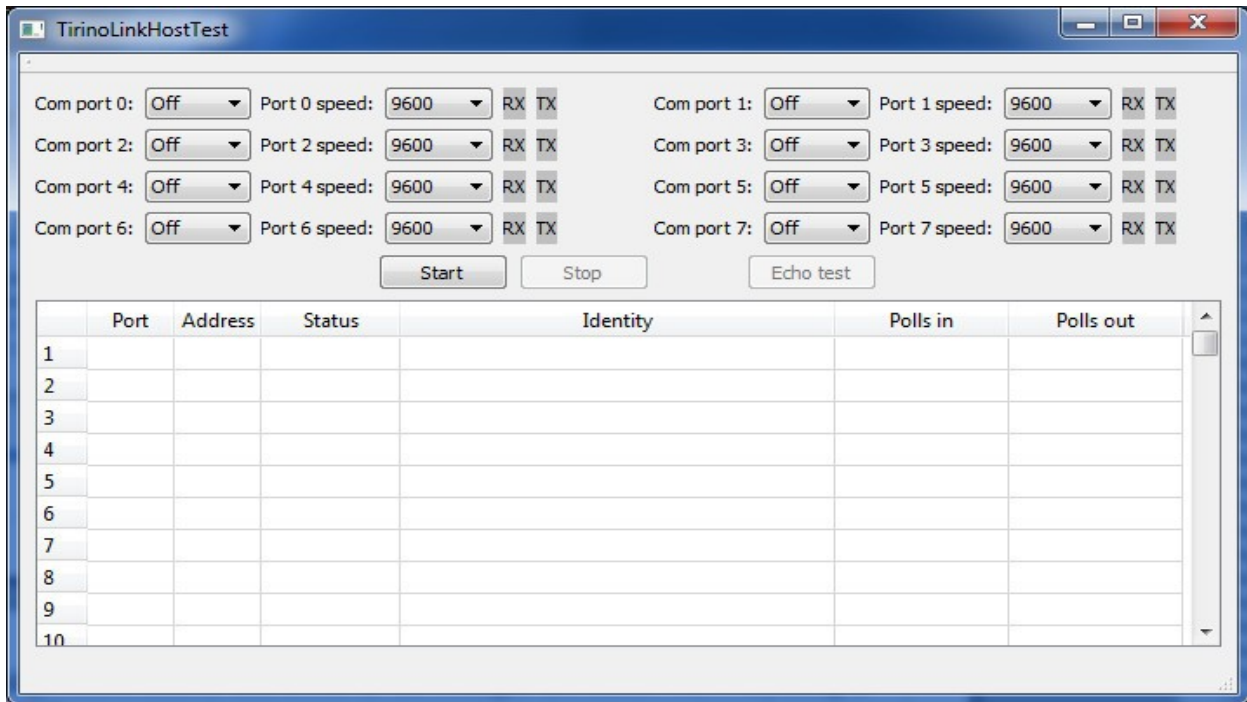
Using the drop downs, select the correct port for Com port 0 and the appropriate speed (115200 by default). Then press the start button. You should see something like this:
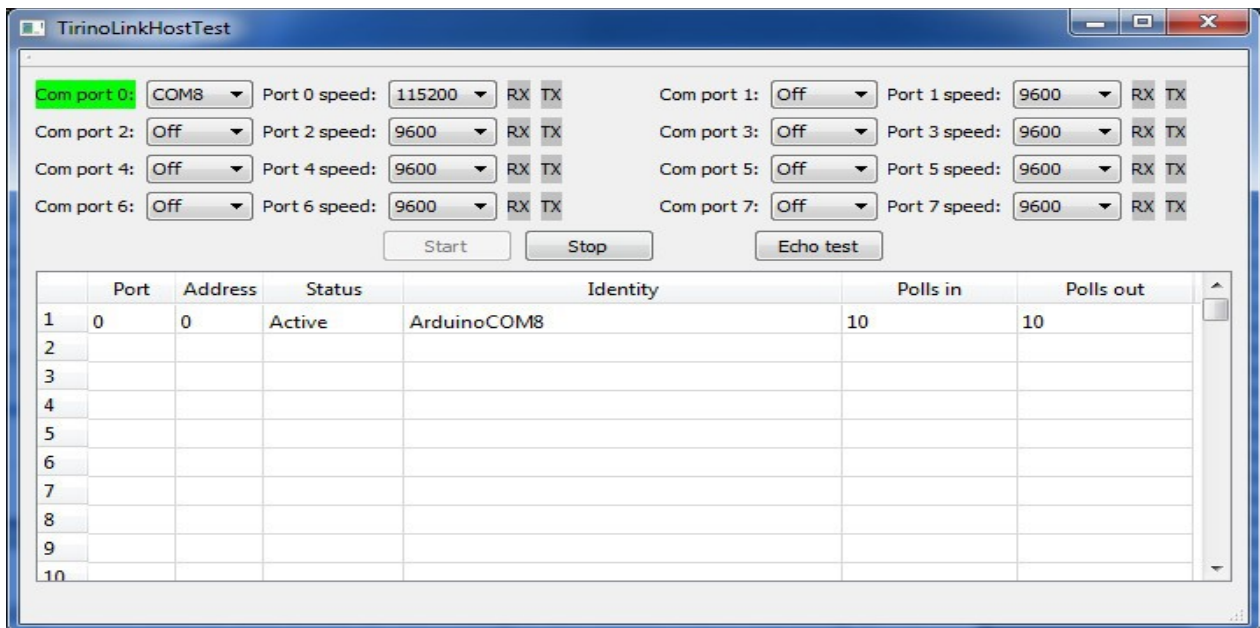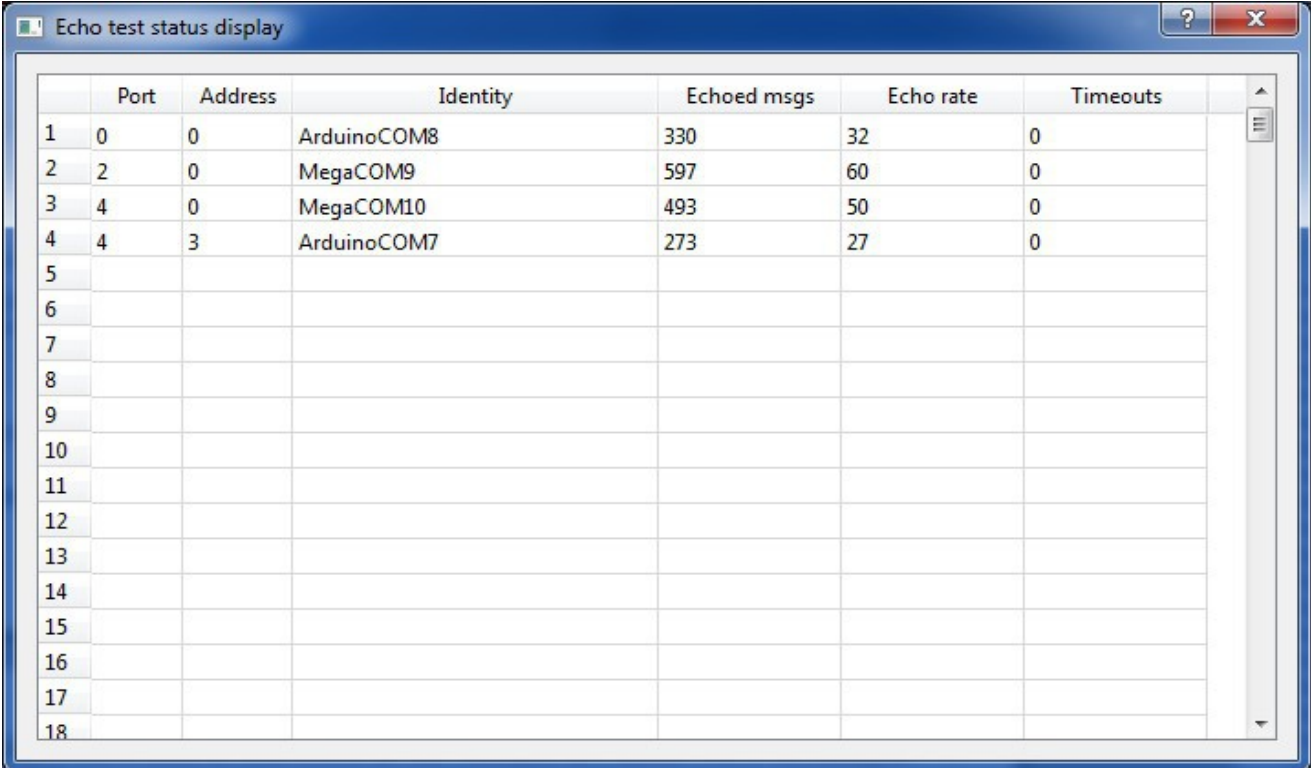

If there is another subsystem running RTArduLinkTest connected, configure up another subsystem port, press Stop and then Start again and you should see something like this:
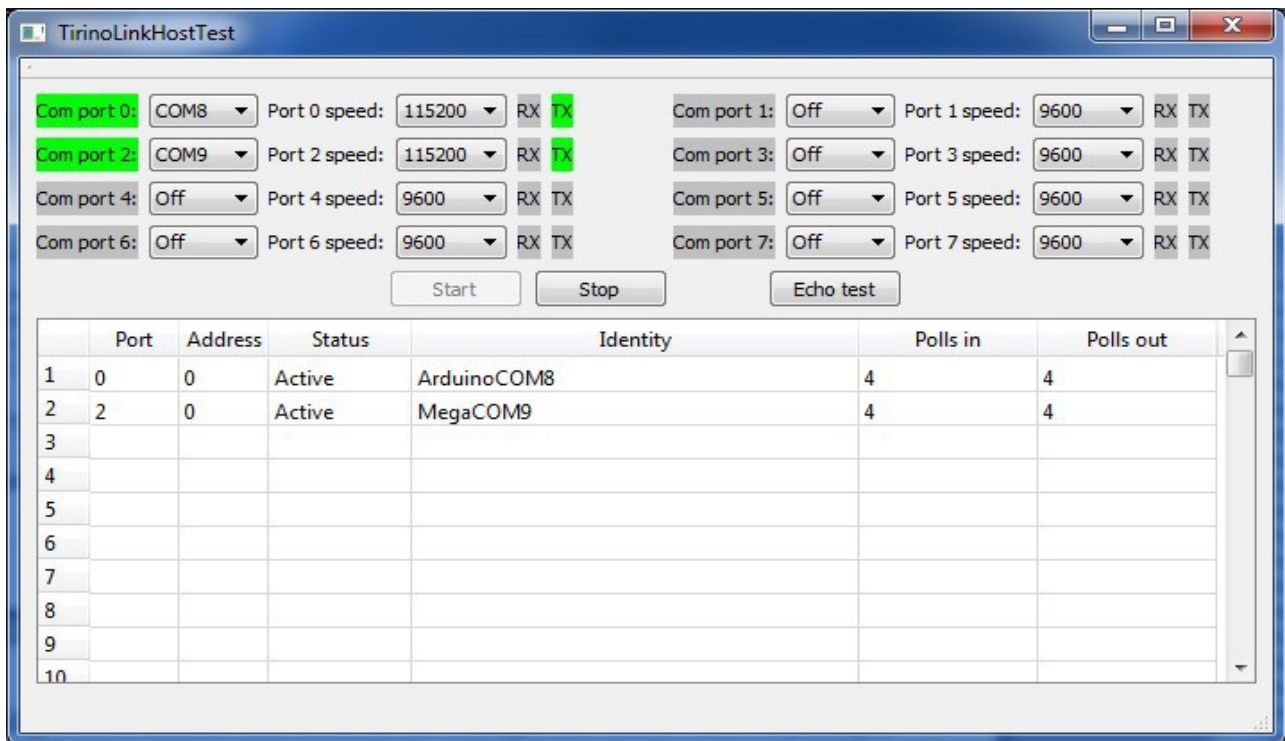
The screenshot below shows a third subsystem connected which also has another subsystem connected to its daisy-chain port 3:
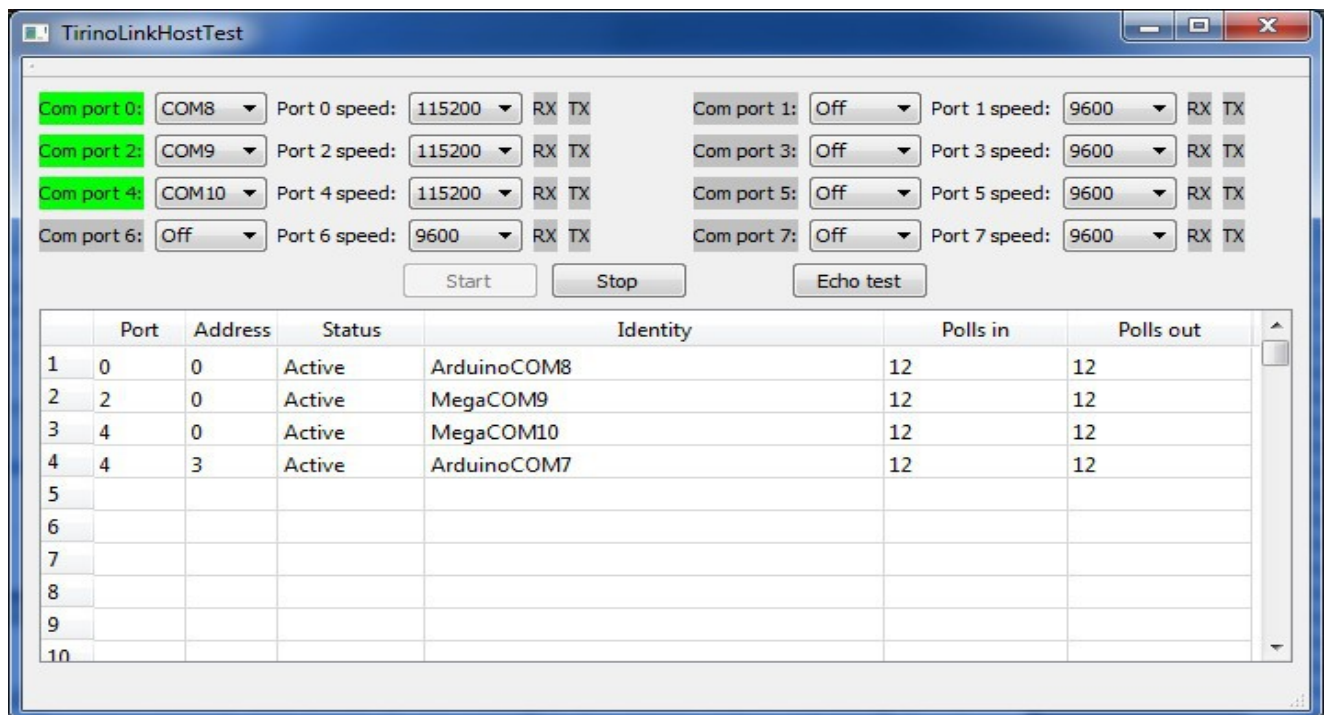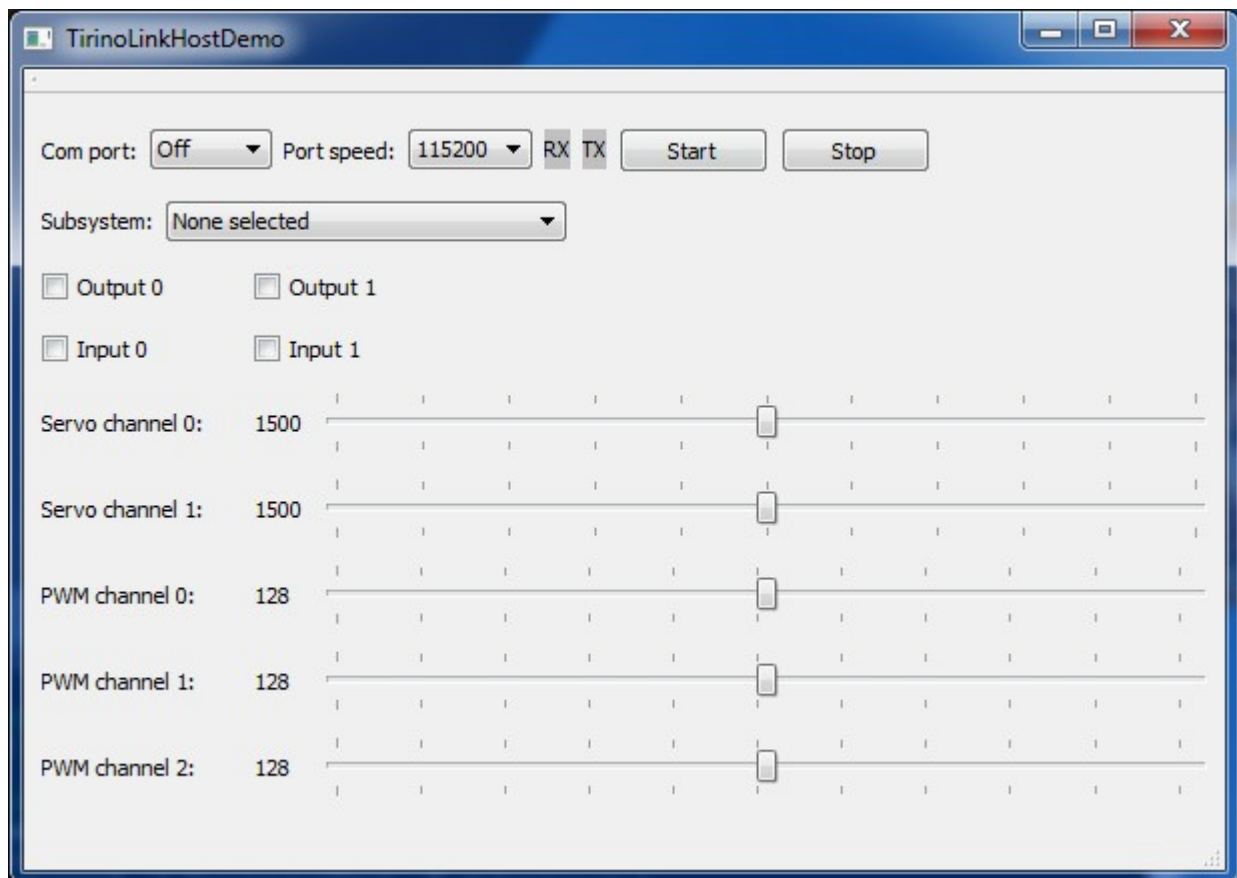

Finally, running the echo test again results in this display:

	Port	Address	Identity	Echoed msgs	Echo rate	Timeouts
1	0	0	ArduinoCOM8	330	32	0
2	2	0	MegaCOM9	597	60	0
3	4	0	MegaCOM10	493	50	0
4	4	3	ArduinoCOM7	273	27	0
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						

Echo test status display

Running HostDemo

TirinoLinkHostDemo

Com port: Off ▼ Port speed: 115200 ▼ RX TX [Start] [Stop]

Subsystem: None selected ▼

☐ Output 0   ☐ Output 1

☐ Input 0   ☐ Input 1

Servo channel 0: 1500

Servo channel 1: 1500

PWM channel 0: 128

PWM channel 1: 128

PWM channel 2: 128

HostDemo shows how RTArduLink can be used to control hardware functions on the subsystem, specifically PWM, servo and GPIO in this case.

To run this demo, the target subsystem needs to be loaded with the subsystem RTArduLinkDemo software. The default subsystem configuration is fine but any configuration should work. Use RTArduLinkConfig to check the configuration. Alternatively, RTArduLinkDemo will display the current configuration on the USB port when starting up.

The correct com port to be used for the test should be selected using the dropdown list and then the Start button pressed. After a few seconds, the subsystem dropdown should be populated with available subsystem identity strings and the appropriate target system should be selected. This is what it should look like:
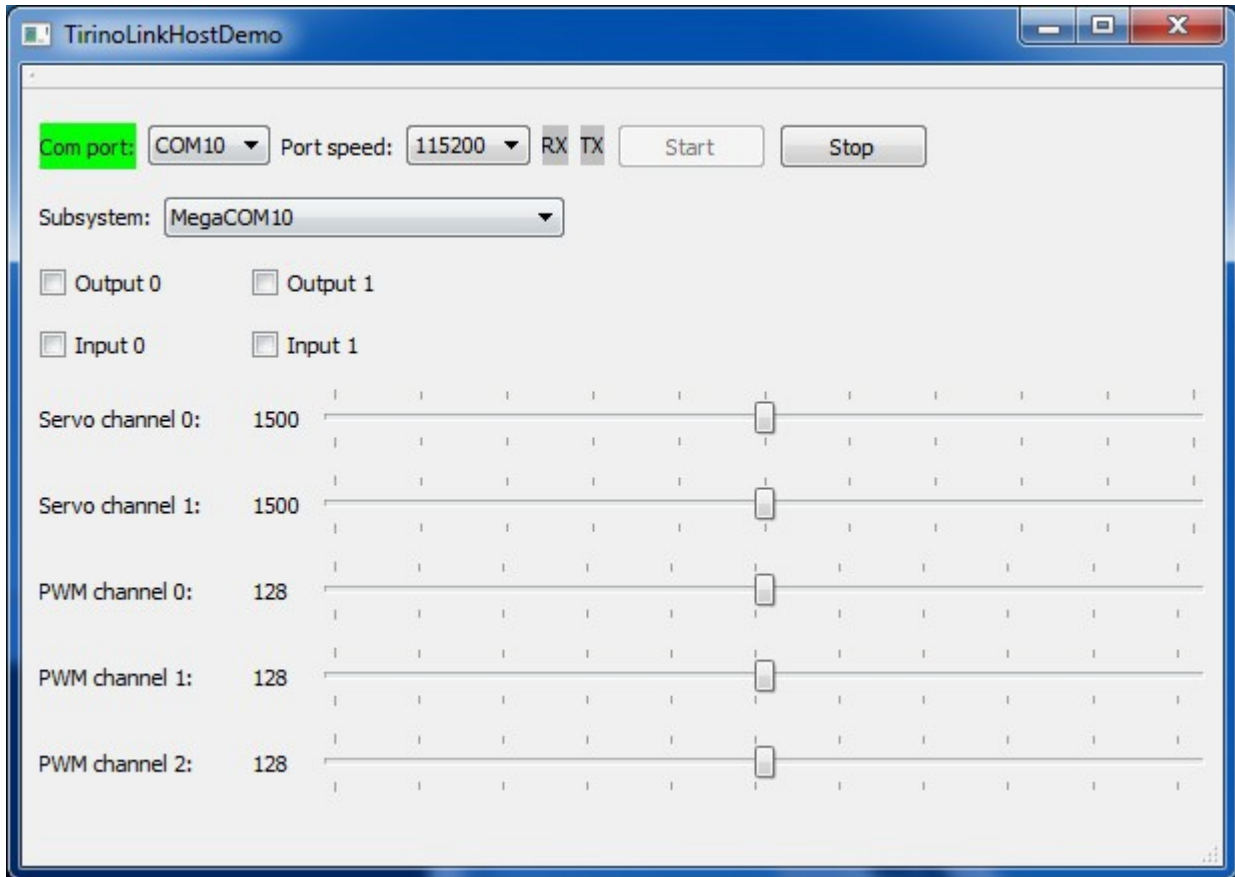
The RX and TX blocks should be flashing green occasionally. If the port shows red, it means that the software cannot open the port for some reason.

The sliders can be used to control the PWM and servo outputs while the Output tick boxes control the output pins. The Input tick boxes show the state of the input pins.

The PWM and servo outputs can be checked in various ways but an oscilloscope is probably the best. To check that the GPIO is working, connect the inputs to the outputs and any change in the Output tick boxes should be reflected in the Input check boxes.

The Arduino pins used are:

- Digital pin 2: Servo channel 0.

- Digital pin 3: Servo channel 1.

- Digital pin 4: GPIO Input 0.
- Digital pin 5: GPIO Input 1.
- Digital pin 6: GPIO Output 0.
- Digital pin 7: GPIO Output 1.
- Digital pin 9: PWM channel 0.
- Digital pin 10: PWM channel 1.
- Digital pin 11: PWM channel 2.