Arduino XPressNet DCC
Simple Throttle Version 1

John A. Stewart

December 13 2010

Introduction

The Arduino (http://www.arduino.cc) is an open-source hardware platform that contains a simple micro-controller, that is widely used in a variety of fields.

XPressNet is an open-source command-bus protocol presented by Lenz (http://www.lenz.com) that is used by a variety of manufacturers.

Presented here is an experiment in open-source XPressNet devices, using the Arduino platform. It is hoped that other projects, from Railcom detection triggering semi-automation, to wireless using the ZigBee standards will eventually be produced.

Why?

Why not? Some people like to see how things work. Sometimes, this translates into "how does this clock work?" to "what happens if you add or remove this control?". The following presents the results of my experiments in XPressNet understanding, and in trying to see what happens if you do something a little bit different.

Simple Throttle - version 1.

I'm new to DCC, but not to model railroading, nor to operating full size rail stuff. I have been out of the model railroading "scene" since the 1980s when the CTC-16 was first presented.

What I have noticed since re-joining the fray is:
- throttles now have many, buttons;
- my eyesight is not as good as it was;
- lots of the buttons are "useless" - at least to me.

Lets go through these in more detail:

"throttles have many buttons".

Which means that not only are they relatively large, they take your attention away from the task at hand. On the full sized stuff, especially when within yard limits, ones head should be out looking around, not focusing on buttons and dials and gauges.

"my eyesight is not as good as it was".

Which means, that I need to put glasses on to see what button does what. Hmmm - full sized stuff, you get to instinctively know where the controls are, and each control feels different, so by tactile feedback, you know without looking if you are on the right control or not.

"lots of the buttons are useless".

On the model operating sessions that I have either watched, or have participated in, layouts don't have DCC controlled turnouts; people only control one train (ie, one locomotive), and they generally don't worry about lots of functions and lights and sounds.

Design

Given the above, I designed a throttle that harks back to the simplicity of the pre-DCC days, to see how it works. The requirements are:

- operation without looking at it;
- can not "take over" another train by accident;
- can not stop the entire layout by accident;
- easy to build;
- inexpensive.

My first design has 2 input controls, and, one status light. No other displays. The input buttons are "speed" and a forward-neutral-reverse switch. The locomotive headlight is turned on automatically, and turned off after a period of inactivity on the locomotive.

Because the locomotive address is programmed into it, you can not take over the wrong locomotive by accident; because there is no "absolute stop layout" button, you can't stop everything (for better or worse). Finally, the XPressNet address is hard-programmed into each throttle, so if your bookkeeping is ok, you will not have any network conflicts.

Design

You need an Arduino board, or similar. Any board will do; if you can find one of the older "ATmega8" boards for a bargain basement price, the code will fit on it.

XPressNet is a serial bus, and uses RS-485 for it. For my first 2 throttles, I prototyped the throttle on an Arduino board that had more than 1 serial port (one port for diagnostic messages, another for XPressNet communication), but the simpler boards have only 1 serial port.

For the real throttles, I used:

- "Arduino Pro Mini 328, 5v, 16mhz" (http://www.sparkfun.com/products/9218)
- "RS-485 breakout module" (http://www.sparkfun.com/products/9505)

There is nothing special about these boards; they were just small and convenient. If you purchase the RS-485 chips from, say, digikey, you'll find that they come in much cheaper than the RS-485 breakout board shown above.

NOTE: to put the Throttle program down to the Arduino, you need an FTDI USB connector for the "Pro Mini", most other Arduino boards come with the USB FTDI connector built in.

I also needed the following:

- 5-pin DIN plug;
- 1 diode;
- one LED, and a 220 ohm (red,red,brown) or close resistor;
- a linear taper potentiometer; value not really important;
- a toggle switch (SPDT - single pole, "on-off-on" type of switch);
- a case;
- old bits of copper PC board;
- some wire.

Although the pictures make it out to be a bit of a rats-nest, the wiring is pretty simple. If you are good at scrounging, building a throttle for, say, $20.00 would not be out of line.

Power:

- 12v power comes in from the XPressNet "L" and "M" pins - the "L" goes through the diode and into the "raw" input of the Arduino. The "M" connects directly to a bit of old PC board that I have arbitrarily labelled "ground". (Check your LZV100 manuals for the correct pins for the DIN connector, or go and see the excellent XnTCP website - http://www.terdina.net/rails/)

- The Arduino board supplies 5v regulated supply, "vcc", so connect this to another bit of old PC board. Also ensure that the Arduino is connected to the "ground" bit of old PC board.

So, now we have 12v coming in, regulated 5v and ground connections sitting waiting there. The Arduino is now powered, and you can play with some simple Arduino programs; if you want - hint - look for the "Blink" program to blink the onboard LED light.

Input devices:

The *forward, neutral, reverse switch* - centre pin connects to the ground, the two outer pins connect to the Arduino digital pins 2 and 3.

The *potentiometer*; the two outer connections connect to ground and 5v, the middle one (the "wiper") goes to the Arduino Analog input pin A0. Try and connect it so that when the throttle is at "zero" the wiper output is 0v.

If you get the potentiometer backwards, you can always change one line in the Arduino program to reverse it in software. Same with the forward/reverse switches; you can always "flip" these easily by changing the software, but it is best to keep things as written here.

Output devices:

LED display. Most Arduino boards have an LED connected to digital pin 13; try the "blinking LED" program in the Arduino examples to see. We just bring this out to the outside, so another LED is connected (with resistor in series - much like wiring a LED in a DCC decoder) so that one side goes to digital pin 13, and the other to ground. The Blinking LED program is great for testing to ensure that this goes ok.

RS-485 interface. The little guy on the board has 8 pins; the board wires 2 of these together, so we have 7 connections. Just to make sure of what is what, here are the actual connections to the RS-485 chip, in case you purchase the chips separately:

Power:
pin 8: to the "vcc" bit of PC board, as outlined above.
pin 5: to the "ground" bit of PC board.

You now have power to this chip.

pin 1: RO (receive output) to Arduino serial receive input;
pin 2: connect to pin 3, connect these to Arduino digital pin 12;
pin 3: connect to pin 2, see instructions on the line above;
pin 4: DI (data in) to Arduino serial transmit output;

You now have the Arduino connected so it can send and receive, AND enable the transmitter to actually send to the XPressNet bus;

finally:

pin 6: "A" on the XPressNet DIN plug connector;
pin 7: "B" on the XPressnet DIN plug connector.

Again Georgio Terdina's XnTCP website goes into details on how to wire a device very similar to this one - http://www.terdina.net/rails/)

Testing:

Presuming that you can program your Arduino, (see http://www.arduino.cc for more info), here are some ways of testing this configuration.

I'll let you in on a little secret. When I made the throttle in the pictures, I just put the final program on it and it worked. Of course, I prototyped the throttle before hand, and I have done lots with Arduinos, etc, so I kind of had a head start. To verify these instructions, for the second throttle, I followed them, or so I thought. Please read carefully, especially the bits in bold… (smile)

LED Blinking.

Find the Blink example in the Arduino examples. (hint, on my mac, in the Arduino environment, "File"->"Examples"->"Basics"->"Blink").

This should blink both the LED on the Arduino board, and our connected LED. If it does not work, check your wiring, especially the polarity of the LED; it works one way, not the other.

Try this powered by the USB connector to your computer (disconnected from the XPressNEet); then try it with power coming from the XPressNet bus (disconnected from the computer). This tests both the LED, and the power coming from the XPressNet bus.

Switches to LED:

Try the "Button" Arduino sketch. (found in "Examples"->"Digital"->"Button"). Two notes of mention:

Note 1) The sketch assumes digital input 2; we use both this and input 3 for the forward-neutral-reverse switch.

Note 2) We use a method that reduces the parts count, so the switches are wired differently. **You WILL have to put the following lines in**:

```
pinMode(buttonPin, INPUT);
digitalWrite(buttonPin, HIGH); // pullup resistors ON
```

Verify and download the sketch, try it with input 2; if it works, edit the sketch, verify, and download for digital input 3 and ensure that it works. You can do this with power coming from the USB connector.

POT to LED:

Again, find the sketch "AnalogInput". On my Mac, it is under the Arduino environment "File"->"Examples"->"Analog". This sketch reads analog pin A0; which we use for our throttle. The comments in the sketch should give you enough to go on to ensure that this pot is connected correctly.

XPressNet stopper;

Ok - now to test the Arduino's ability to control your XPressNet connection. Here we have to enable/disable the RS-485 transmit control, and send serial data.

Take the LED Blinking example, but change the digital pin line from 13 to 12. This will enable transmission to the XPressNet bus; but as we will not be transmitting anything, it will momentarily disrupt the bus. A Lenz controller should give you an error indication; my Lenz LH90 gives me the "---" display.

Running Verification:

Choose a locomotive ID and empty XPressNet slot. Put these into the constants "LOCO_ID" and "MY_ADDRESS" in the PotSwitchThrottle Arduino program. "Verify" and "Save" your changes.

Download the verified PotSwitchThrottle program; disconnect from your computer.

Ensure that the Lenz system is **not in power off mode** - and power the throttle from a working XPressNet system.

The LED should come on and should be stable. (if it flashes an error code; see the PotSwitchThrottle code, near the top, for a complete listing of error messages). If it does not come on, it MIGHT be because it does not get the right initialization steps complete, which is the reason for ensuring that the Lenz system is not in "off" mode!

When you get the LED so that it is on, and is stable, turn off track power from your LH90 or LH100. The LED on our throttle should blink when track power is turned "off". The LED should go back to being solidly on when the "off" is turned off (you know what I mean!)

Now, put a locomotive that matches the address in "LOCO_ID". Put the throttle into gear, hit the gas, and have fun! Hopefully it will work flawlessly for you.

My Pictures

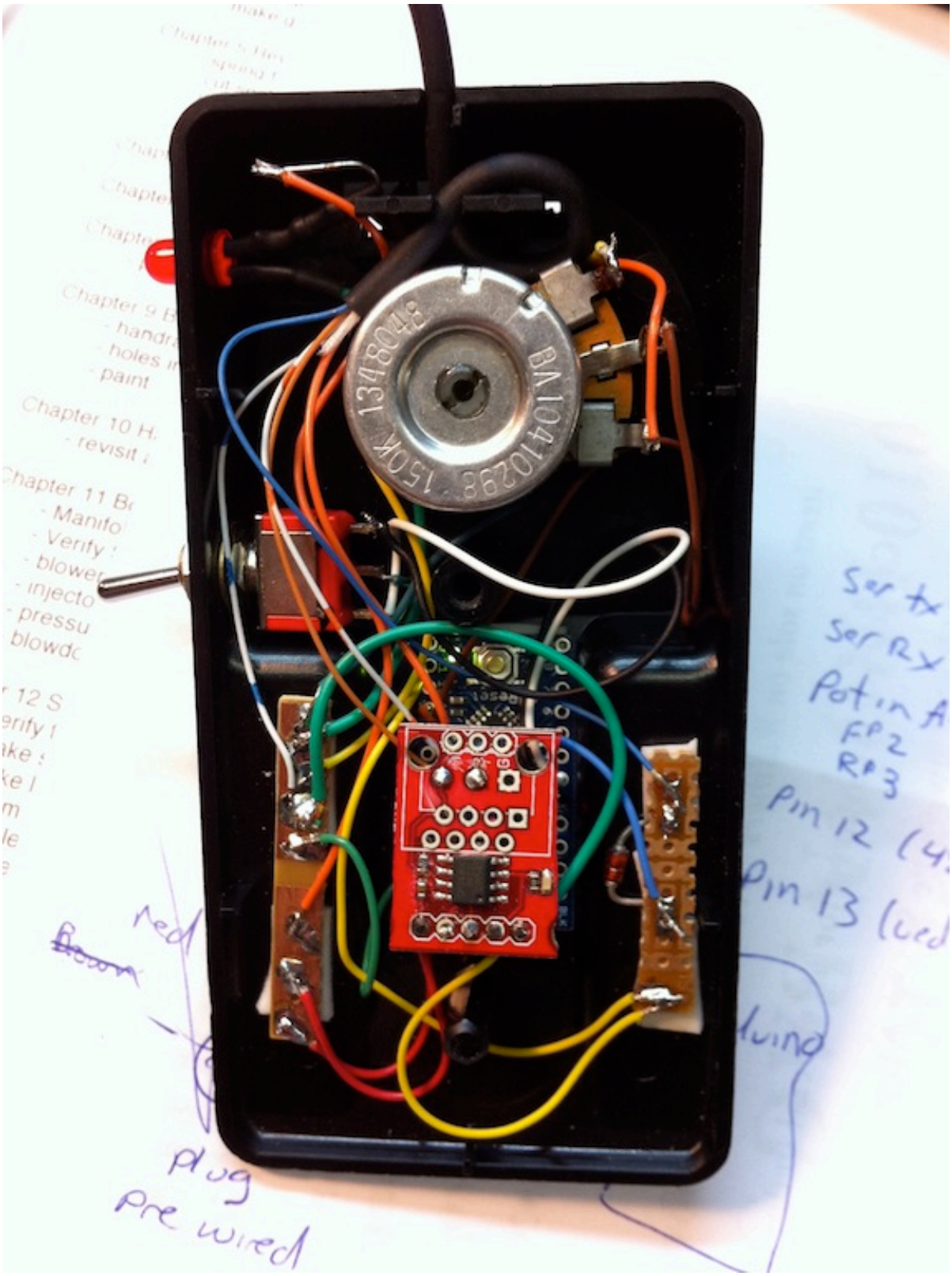
First picture - large knob, you can see the XPressNet cord leaving the top; LED is lit, and locomotive is in reverse. This fits nicely in the hand; the container is actually from an old computer video game.


Ser tx
Ser Rx
Pot in A0
FP 2
RP 3
Pin 12 (4
Pin 13 (LE
Arduino
pre wired
blue -

Second picture - you can see the bits and pieces, and how I put it together. The red board is the RS485 board; it actually is kept in place by two "pin sockets" that connect the serial lines from the Arduino. Power, and control, and the XPressNet "A" and "B" lines are wired directly to it. The Arduino is behind this, double sided taped to the case.

You can see my copper "bus bars" - the one on the bottom left is divided in two; ground and +5v. The one on the right is divided in 3; first is the "L" from the XPressNet interface, this +12v line goes through a diode, then via the blue line, into the "raw" input of the Arduino. The yellow lines are just tied together and the solder pad makes a nice place to do this. That line is actually the control for the transmitter for the RS-485 chip.

To program, I pull off the RS-485 breakout board, and stick on an FTDI 5v USB adapter.


Chapter 10 H
- revisit
Chapter 11 B
- Manifo
- Verify
- blowe
- injecto
- pressu
blowdc
Ser tx
Ser Rx
Pot in A
FP2
RP3
Pin 12 (4
Pin 13 (led
Red
Boom
Arduino
Plug
Pre wired

Copyright

/*

Simple XPressNet Throttle

Version 1.1

*/