A NTP DRIVEN NIXIE CLOCK

Over the last few years, clocks based on Nixie Tubes have become popular and attractive kits are now widely available.

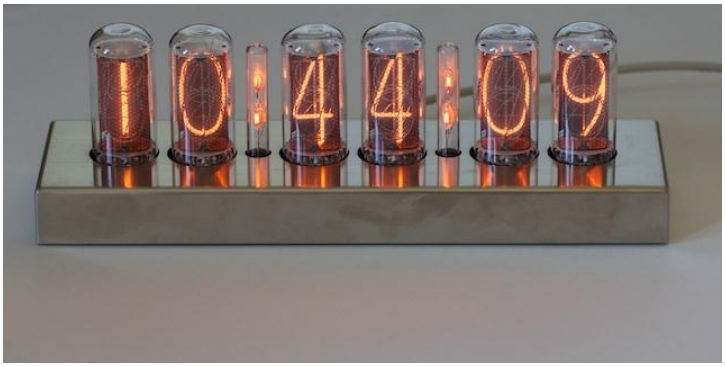
I thought it would be fun to build a Nixie Clock, but I wanted one which got the time from the Internet using NTP. That way, it would always be accurate, even jumping forwards and back to accommodate summer time.

My initial plan was to design my own clock, but it seemed unlikely that I'd make anything which looked as good as the kits, even were I to spend a lot of time on the process. Perhaps I could adapt a kit: most of the clocks have a microcontroller inside, so my next idea was to take an existing clock kit and hack the firmware.

That seemed easier, but then a better idea struck me: some of the clock designs accept NMEA data from a GPS receiver. I was sure I could replace the GPSr with a computer and fake the signal.

The ИН-18 Blue Dream

The ИН-18 Blue Dream was the nicest looking clock kit I could find, so I bought one.



The kit is almost too easy to build, because most of the components are SMD parts which are pre-soldered to the PCB.

Although the documentation includes the pin-out for the GPS, it is silent about which signals are expected. Happily Dieter at Nocrotec quickly supplied the answers. The Blue Dream:

- Expects a normal 4800 baud, 8N1 NMEA serial signal.
- Uses RS-232 levels. This implies that a ‘1’ is represented by a negative voltage and a ‘0’ by a positive one.
- Extracts the time from the GPRMC sentence.

Raspberry Pi

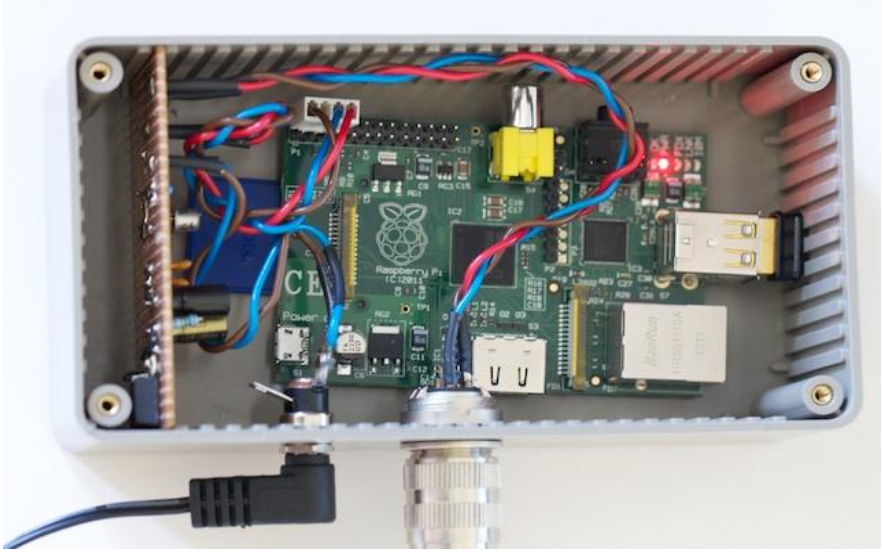
To synthesize the fake GPS signal, a Raspberry Pi seemed the obvious choice: it’s cheap and small, yet runs a full Linux distribution.

To avoid running an Ethernet cable, I installed one of those dinky USB WiFi adapters.

The software is pretty trivial. In fact, the most difficult part wasn't making the serial port say the right things, but telling the kernel to keep its hands off! Happily Clayton Smith provided an excellent recipe to do this.

By contrast, the software itself is just a few lines of Perl. You can grab the code from github

Hardware



Interfacing the Raspberry Pi's serial output to the Nixie clock proved equally easy. A simple NPN-transistor inverter converts the levels into something the clock will accept:

Logic Level	RS-232 (typical)	Raspberry Pi GPIO port	'Inverter' Output
0	+5V	0V	+5V
1	-5V	3.3V	0V

As you'll see the conversion isn't faithful, but it works.

Power

Obviously it would be nice to minimize the number of power supplies and cables we need. The clock needs a 12V supply, and draws about 500mA in normal operation. The power supply supplied with the clock is rated at 1.5A though, so obviously the best solution would be to power the Raspberry Pi from that too.

The Pi wants 5V, so I built a simple switching regulator around a LMZ12002 to drop the voltage. It's true that something like a 7805 would work, but you'd have to dissipate about 3W of heat which is a bit of a bore. Needless to say the regulator was the most significant part of the electronics!

Both the inverter and the regulator fitted easily on a small piece of stripboard.

Finally, we obviously want to run only one cable to the clock, so I hacked the clock to accept the +12V from one of the spare pins on the GPS input connection (pin 6 in case I forget).

Observations

I've wanted to build an NTP driven clock for ages, but using an Arduino or a PIC seemed to be quite a lot of hassle and fairly expensive. By contrast the Raspberry Pi is both cheaper and easier to use. In fact, its biggest downside is the complete lack of any mounting holes on the PCB!

The cost difference seems significant. As of August 2012, here are some representative prices:

Arduino Ethernet	
Arduino Uno	€20.00
Ethernet Shield	€29.00
	€49.00

Raspberry Pi Ethernet	
Raspberry Pi	€30.00
	€30.00

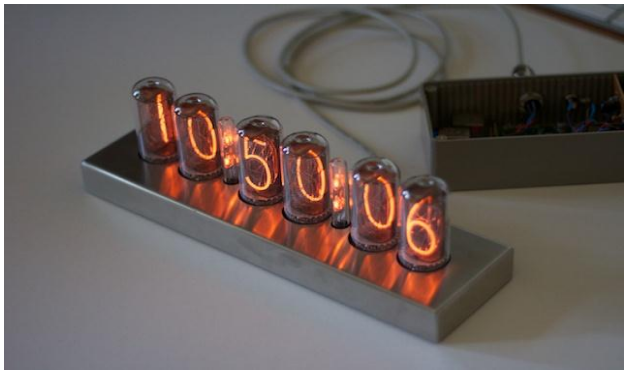
Arduino WiFi	
Arduino Uno	€20.00
WiFi Shield	€69.00
	€89.00

Raspberry Pi WiFi	
Raspberry Pi	€30.00
USB WiFi adapter	€15.00
	€45.00

Despite being cheaper, software development is easier on the Raspberry Pi. Given that it's a full Linux environment, we get a full networking stack as standard which extends all the way up to clients for NTP and any other protocol you want.

For the clock the only software we actually needed to write was a noddy little loop which just formats a message and sends it to the serial port. Neither performance nor memory matter much, so it's an obvious task for something like Perl or Python.

I'm sure that we'll see many projects, particularly those with a network connection, migrate from the Arduino and its ilk to the Raspberry Pi.



Source: http://www.mjoldfield.com/atelier/2012/08/ntp-nixie.html