TRAFFIC SIGNALS AND THE LINUX REVOLUTION

We can make anything!

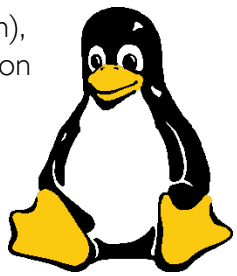
Through the 1980s and 90s a huge change rolled slowly through the UK traffic signal industry. Gradually, soldering irons and wiring diagrams became abandoned in desk drawers. In their place came handsets, such as Psions and Oysters. A new generation of microprocessor-based controllers meant that, for the first time, traffic lights could be PROGRAMMED, not just wired. Every office soon had a computer (and a Computer Science Graduate) in the corner.

The value of the new generation of controllers could be seen and new manufacturers began to replace, or merge with the old establishment companies, as new technology overtook the old. Old names such as Plessey, Phillips and GEC were gradually replaced by Siemens, PEEK, Microsense (and a few others on the way). Over 20 years, these companies continued to develop their controllers, until they became immensely powerful and sophisticated. The law of diminishing returns however meant that each new version saw smaller improvements, and development meant refinement.

There is now another new generation of equipment. Although it has been in existence almost as long as microprocessor-controllers, Linux is the "new big thing" in Traffic Signals. It has been with us in the industry for some years, however this year, for the first time, all four of the largest manufacturers in the industry are offering Linux based controllers (and cue three new names; Imtech, Telent and Motus, alongside Siemens). The benefits of this are not immediately apparent; in the same way that the first micro' controllers were not always better than the refined electro-mechanical versions they replaced. As we all get used to this new wave though, the benefits will start to become apparent.

Linux, combined with the powerful, small and cheap processors available, offers a paradigm shift in traffic signal design and operation. It is admittedly difficult to think of a single revolutionary benefit that Linux provides "out of the box". Instead, it is the power and flexibility that it provides, which will be exploited over the coming years.

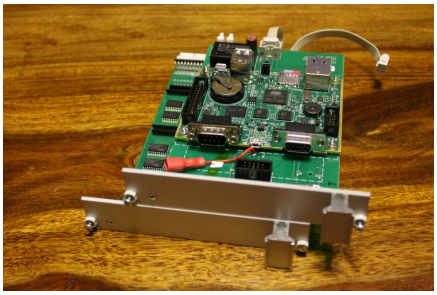
So what is Linux?



Linux is an open-source computer operating system, (based on the Unix system), first released in 1991. It has since become ubiquitous in industry and is widely used on all types of computer, other than PCs. Being open-source means that it is open to anybody to play with or develop, and it is often tailored extensively to suit a particular purpose. It is this flexibility which makes it so powerful. It can be parred back and hardened to make it suitable for safety-critical uses, while on the same hardware platform, another version can be running several fast and complex applications at the same time.

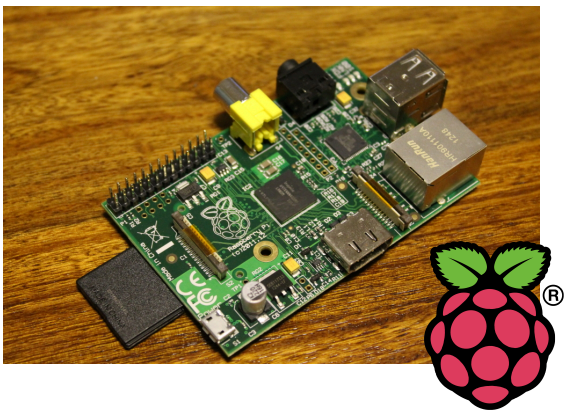
Although the different versions of Linux, and indeed different hardware, mean that the applications cannot be moved seamlessly between equipment, it does make it much simpler. We are already seeing this in practice. Two UK manufacturers, Telent and MOTUS already share much of their hardware with the European Swarco ITC2, although all three run distinctly different controller-applications. All the manufacturers are offering either newly integrated facilities, or stand-alone devices based on Linux processors. This has only come about because of the ease with which the Linux-based applications can be ported between hardware.

There are also an increasing number of small, stand-alone hardware platforms pre-installed with Linux, such as the Local Intelligence Unit (LIU) from Pleydell Technology Consulting Ltd. These are arriving from a number of different sources, both inside the traffic signal industry and outside. In much the same way as companies such as Ferranti saw an opportunity with the arrival of micro-processors, companies outside the industry are now looking at what they can do for us.



Why now?

Until recently, the focus of the computing industries has largely been on increasing processor speed. This has been used to improve and increase the ability to generate sophisticated graphics. Neither of these things are particularly useful in a traffic signal controller, beyond a certain point. With the increase in use of smartphones however, the focus has changed. Greater attention is now paid to reducing the size and power consumption of processors. These devices are also expected to run many separate applications simultaneously, carefully managing the limited processor resources available. This is of much interest to a company looking to reduce the underlying cost of production, particularly in an industry as competitive as ours.



All these developments have come at the same time as an explosion in interest in Linux home computing, with the arrival of the Raspberry Pi. This small cheap computer, devoid of even a box to keep it in, has become a sensation. Sold at around £25 and designed as an educational tool, it has begun to introduce a new generation of people to both programming and Linux. While it is not suitable for use in the 'wild' (although used extensively on the BBC's *Spingwatch*), a quick search of Google or youtube will show the amateur traffic signal projects underway. It is not hard to see how almost anyone could use it as a starting point for a new idea. With the ease with which programs can be ported between Linux platforms, the best of these ideas could easily end up running on industrial devices such as the LIU, or even in controllers.

That's all very nice. So what?

Here's the real revolutionary bit. We can now program WHATEVER WE WANT into our controllers. By this I don't just mean configuration or special conditioning. With an extra card such as the LIU in the controller, we have a fully programmable computer sat there, waiting to be told what to do. And with Linux and a simple programming language, we can program it ourselves.

Still can't quite see the point? Perhaps you want to measure headway between vehicles and record the average by time of day? Or monitor your detectors for chatter by comparing counts from different loops. Or provide a remotely accessible log book, which logs the controller faults, door opening, actions taken and the Engineer's comments, all through a simple web-page. Some are more difficult or complicated to do, but all are possible and the list is endless.

Here's an example.

I have a problem with high-speed, rural sites running MOVA. Up to 20% of vehicles are HGVs or agricultural, with a very low cruise speed. The rest are fast-moving cars and light vehicles, with very high cruise speeds. This causes a problem in setting the MOVA cruise speed, as the normal value is not appropriate for the majority of traffic. A simple solution would be to measure the longest detector inputs and to feed these back into MOVA on a separate link. This link could be tailored to the slow moving vehicles, while the normal traffic link handles the majority of traffic.

Identifying slow, large vehicles is easy, as we are not trying to actively classify them. We can very simply measure the length of each detector input and record it to a rolling log. From this we can measure an average and compare each new input to it. Thresholds, data sample sizes and processor speeds can all be easily adjusted. As a long input tips over the threshold, an output turns on.

Of course, while the theory is simple, in practice it is a bit more difficult. For a start, you need hardware - in this example an LIU provided by Pleydell Technology Consulting Ltd. Secondly, you need to be able to write computer programs in a language supported by the hardware. This is more difficult but even with no previous programming experience, there are enough self-learn books and support for it to be quite possible, within just a few weeks. Of course, the easier option is to find a friend with the right skills, or even hire a professional. Finally, you need to be able to test your results in a safe environment.

Starter Shopping List

Raspberry PI, £25

SD Card £5

RPI IO, box and power supply, £35

Mixed bags of resistors, switches, transistors and LEDs, £15

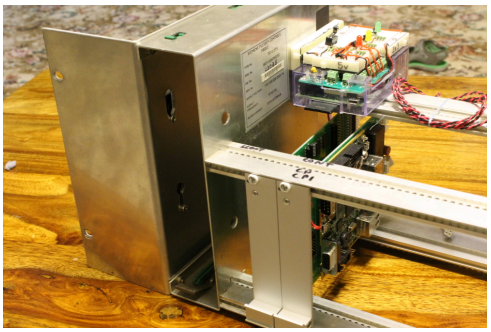
Wire and breadboard, £10

Screw driver and wire cutters, £10

Monitor and keyboard, just plug your own in

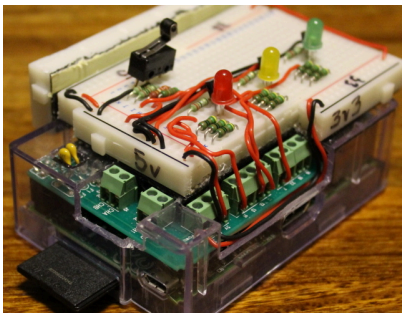
Writing your own programs; priceless!

In my example I've used a Raspberry Pi (RPi) to both teach myself the programming language (in this case Perl and Lua), and to create a test rig. The RPi is wired to the LIU and can either send a variety of inputs to the LIU, or monitor the outputs. A small selection of switches and LEDs can be used to manually driver or monitor the IO as well.



There are obviously a few other things that need to be done as well. A power supply and rack has been liberated from a redundant Tele12 outstaion, which helpfully provides both a 24v supply for the LIU and 5v for the RPI. Some breadboard, a few basic electronic components and a basic knowledge of electronics is all that is needed to wire the IO together. There is a weekend's work to learn everything you need, and pocket money to buy the parts, if you can't already find them.

Even buying everything from scratch, only around £100 is needed for enough parts to build a working proof of principle. Developing this idea further, onto a hardware platform suitable for use in the field is admittedly more expensive. An environmentally hardened linux-based platform is likely to cost around £1000 and many suppliers are able to provide development expertise on a consultancy basis. Although times



are tight, even this modest sum is within the reach of most employers, if you can demonstrate that you can save money, time or even possibly, lives.

But why should I bother?

This has all been done with very limited resources, borrowing favours and equipment, but it does show what is possible. Most companies and Local Authorities have the resources and expertise (often in IT departments) to be able to do the same with ease; the hardest part is likely to be justifying why you need to do it yourself, rather than just trying to buy something similar ready made. This misses the point though. If we don't take the opportunities to develop new things, we will never get exactly what we want. Moreover, the good ideas that don't get developed will be lost to the industry. It is these good ideas, developed on hardware like the LIU, proven on our own sites, that will gradually become standard features in new controllers.

This of course is not ideal for many companies, who would like to see a return on their (albeit small) investment. Perhaps as an industry we should start looking at licensing our ideas and software, making them free, or at a peppercorn rate, to use as long as we are credited? Similar schemes are already widely used in software and computer language development. By devolving development and sharing ideas (and the credit) in this way, our industry could develop better solutions faster than ever before, building companies' reputations. Rather than product development, it could be seen as advertising.

Increasingly, the same reasons for developing Linux controllers in the first place - reduction in cost and hardware, will lead to a reduction in the amount of bespoke or specialist hardware needed. It will not mean the end of it though; there will always be a place for accurate, dedicated and sophisticated systems. But a much greater selection of tools will slowly become more widely available to us; for an example, just look at the range of UG405 OTUs now available.

Where do we go from here?

We, as an industry, and as employers need to think about what we will need in the future. We will increasing need people with programming skills, as well as people with computer networking skills, to support the new communications that are already here. This points to us all moving much closer to the world of I.T.! We need to consider whether I.T. qualifications and training are once again needed for our graduates and trainees.

We also need to go back-to-basics, teaching electronics and valuing basic engineering knowledge and skills. While the idea of employing only managers and outsourcing technical skills entirely has some attraction, Employers must begin to realise that this is the expensive option. Employing or training people with relevant technical ability, as well as management expertise, can save thousands of pounds in consultancy fees and greatly improve decision making.

So, lets start arguing for test equipment to be provided in design offices. Soldering irons may attract worried glances from those who manage the fire alarms, but 'breadboard' and 'chocolate blocks' will do instead. Wire, simple components and basic Linux computers with no internet access, are fine. We don't need post-grad degrees in electronic design either; a basic day release course or even on line learning would be enough. We are willing to do this for learning drawing, design, modeling or configuration skills so why not electronics and programming?

To support this, perhaps our professional bodies could help? If the IHE and CIHT publicly recognise these skills and courses as being beneficial in traffic signals applications, we would be halfway

to persuading our bosses. If our training providers would then offer courses, intended for designers, but teaching basic principles of electronics and programming, teaching how controllers are built and work, we would be almost there.

As I've mentioned above, we also need to look at how we can share our ideas while preserving intellectual rights and being acknowledged for them. A new regime of open 'democratised' development, with support from suppliers and manufacturers, could massively increase the number of practical solutions available. With a wide variety of applications available, albeit for free, manufacturers will be able to sell a greater number of units. This in turn provides the turnover and profits to recoup the initial time spent in development. This is only possible though if manufacturers continue to facilitate development of the software, alongside their future customers.

About me

I have been working for Cambridgeshire County Council for the last ten years and I am currently the Lead Engineer for Traffic Signals. Prior to this, I was an Apprentice with the M.O.D. learning electronics and telecommunications. Before this project I had no practical knowledge of computer programming; this is genuinely a skill that anyone can learn.

For further information about this project, or about Linux processors, you can contact me (Chris Kennett) at kennettc@o2.co.uk and I will do my best to help.

Acknowledgements

The name "Raspberry Pi" and associated Raspberry logo are trademarks of the Raspberry Pi Foundation, a not-for profit organisation. The Raspberry Pi Logo is used with permission.

The mane "Linux" and the logo of Tux, the Linux Penguin is a Trademark of the Linux Foundation, another not-for profit organisation.

Many thanks must go to Dr Mark Pleydell, Director of Pleydell Technology Consulting Ltd, and his colleagues and associates, Chris Smith and Rajesh Krishnan. PTC Ltd for providing the LIU used in the example and Chris and Rajesh for their help in developing, checking and porting the computer scripts.

Finally, thank you to Mark Crabtree of TRL, who unwittingly started the long chain of events leading to this paper, during a brief conversation at last years Symposium.