Sunlight Foundation

Sunlight Labs

Search

- Blog
- Projects
- Events
- About
- APIs
- Contact

Our Door Opener (A Science Project)

Written by
Tom Lee
Date
02/16/2010 10 a.m.

Life in the labs has been pretty good since we moved into our current offices. Before, we were spread out over two floors: my team was upstairs in a stuffy law office sublet, and the rest of our colleagues were stuck in a homey but increasingly cramped and run-down space four floors below. Since moving everyone to the third floor we've found ourselves with plenty of room, lots more light and a nicer kitchen. It's just a more pleasant working environment in general.

But there's always room for improvement. For one thing, the new space came with new locks -- ones with really expensive keys. Issuing keys to the entire staff wasn't practical, and coordinating door-opening responsibilities in a way that accommodated team members' occasionally odd schedules was inconvenient. Fortunately, the space also came with the button you see to the right. This button, with a few hacks, now makes it so we can open up our office with our Nexus Ones, iPhones or even SMS.

Located near the reception desk, this button opens an electronic latch on the front door. Pulling the assembly out of the wall revealed the system to be about as simple as possible: the button simply connects two wires. Bridging them with a screwdriver fired the latch (from their small gauge and uninsulated connections, it was obvious we weren't dealing with dangerous voltages, but please don't start pulling cable from your walls unless you know what you're doing).

Connecting two low-voltage wires electronically isn't a particularly hard trick, so I decided it'd be fun to spend some evenings building a system to expose that switch to our network. I've built a few projects in the past that make use of custom router firmwares and Arduino microcontrollers, so I decided to use those tools for the job. I'm a big fan of this approach -- it's a great, inexpensive way to add a scriptable network interface to your microcontroller projects (if you're interested, I talk more about this technique here).

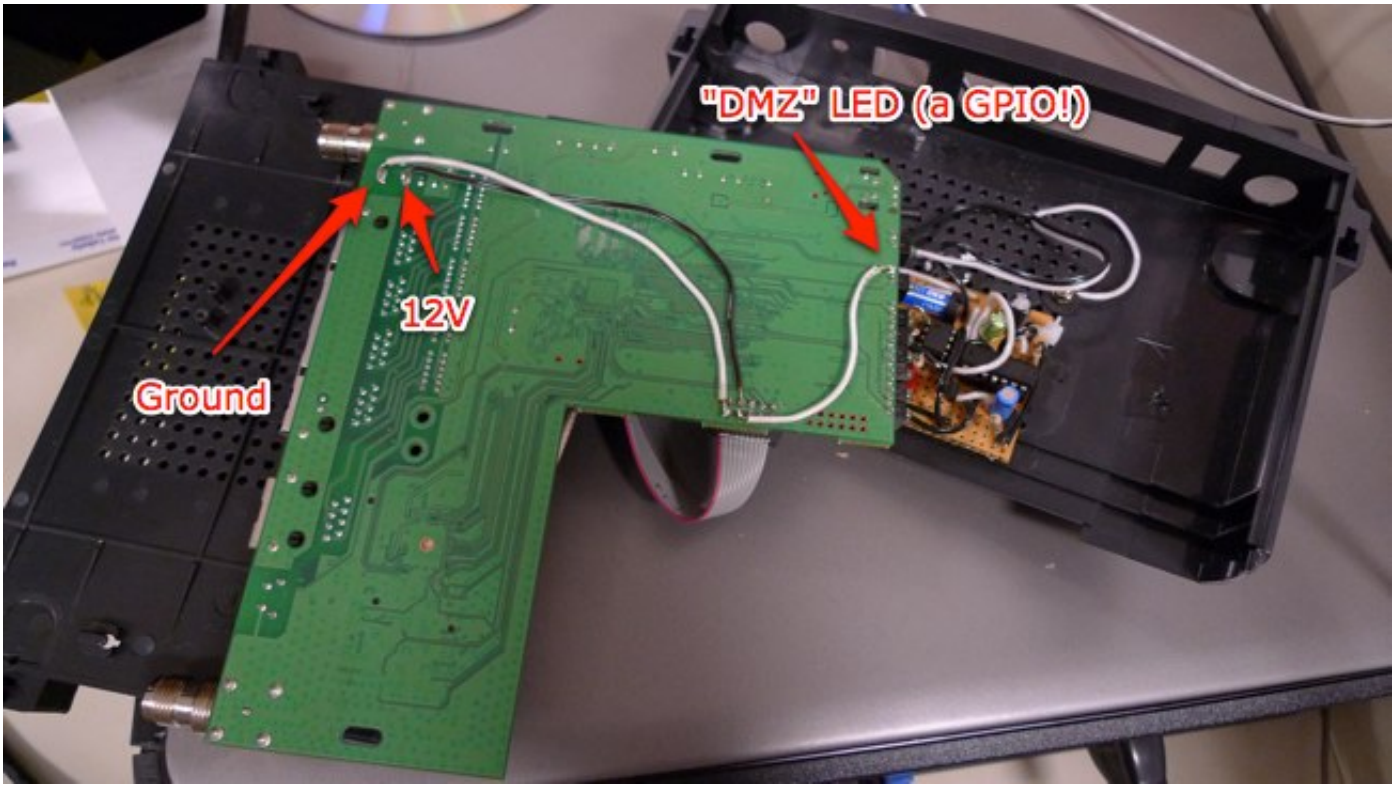
Ultimately, I ran into difficulty getting this particular router's serial connection to work, so I abandoned the Arduino -- it was overkill anyway. Inspired by the excellent new *MAKE: Electronics* book, I decided to build a circuit to do the job. Here's the finished franken-router, hooked up to the switch:

So how does this all work? Well, let's start with the router. It's a Linksys WRT54GL, a still-Linux-friendly descendant of the WRT54G, the router which jump-started the custom firmware scene. There are a lot of custom router firmwares available -- DD-WRT, Tomato, Gargoyle -- and they can all make your consumer-grade router do some professional-grade things. For this project I used OpenWRT, the system on which many of those other projects are based. It's got a much steeper learning curve than those other distros -- you definitely need to be comfortable with the command line -- but it lets you build a custom firmware with exactly the components you want. With some tweaking I was able to create a firmware that included a

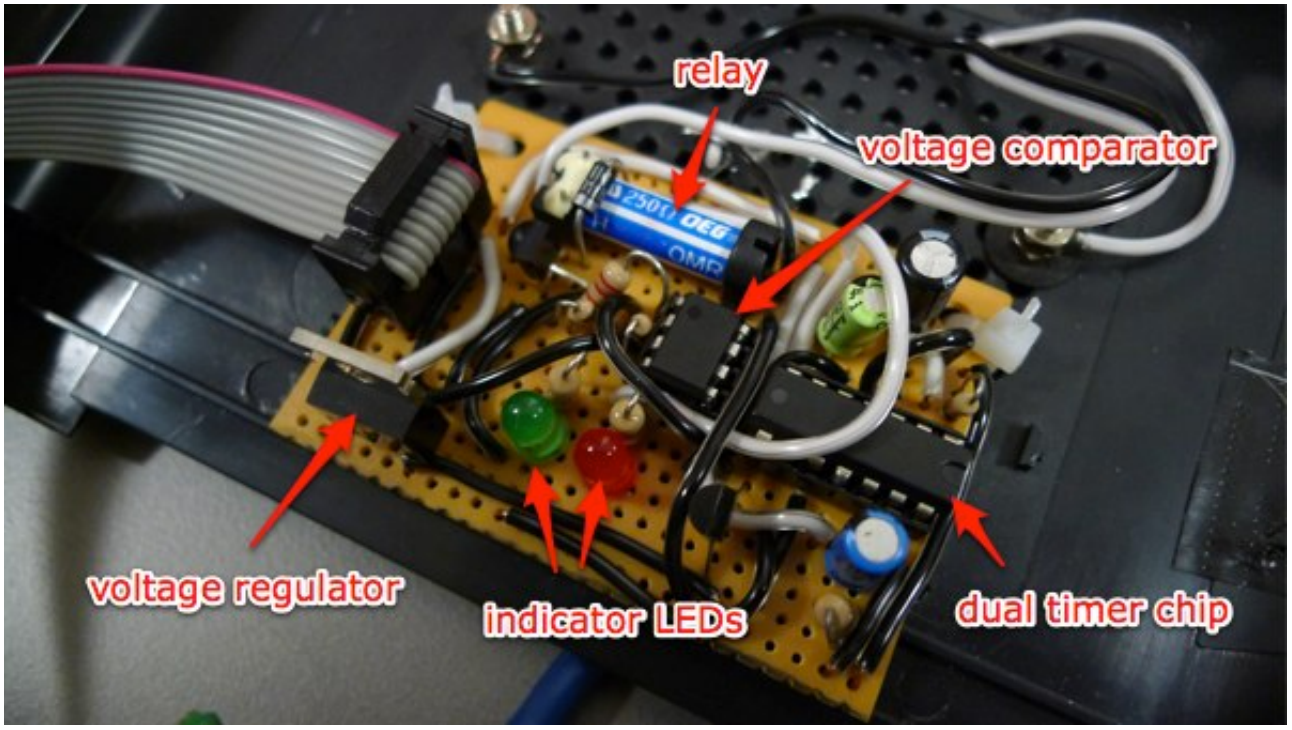
stripped-down version of Python, but which still fit into the WRT54GL's meager 2MB flash memory. (Like the Arduino, Python turned out to be overkill, but at the time I expected to need it -- and it's always nice to have it handy.)

With the firmware installed, I was able to SSH into the router and perform some simple manipulations of the system's GPIOs -- General Purpose Input/Outputs. These connect to things like the system's LEDs and switches, and can be controlled in software. I selected a GPIO that didn't seem to be used by OpenWRT -- it illuminates the "DMZ" LED on the front panel -- and wrote a very simple script to control it. I could now flip a tiny light on and off from a network connection. The hard part was over.

Next came the hardware. It was easy to add a pin header to the router's PCB -- it's got holes drilled and ready to go (some of them are connected to router systems -- see here). To this I hooked up ground, power, and a connection to one of the DMZ LED's pins.



A cable then connects those three pins to this breakout board:



There are a few things going on here. That blue cylinder is a relay, and it's the heart of the system. A relay is basically just a switch that can be closed by the electromagnet that's attached to it. They're useful for employing a small voltage to turn a larger voltage on and off. In this case, the switched inputs of the relay are wired up to the connection posts that I added to the outside of the router: when the relay is energized, those posts are electrically connected. When it's not, they aren't. The relay runs on 5 volts, which is why there's a voltage regulator on the board: to reduce the router's 12 volts to 5 (the other components on the board can run on a range of voltages; they're perfectly happy at 5).

The router's GPIO isn't up to the task of controlling the relay directly, so there's some additional circuitry that's needed to turn the GPIO signal into a firing relay. First up: the 556 chip, which contains two 555 timer chips in a single package. When triggered, one of these fires the relay for a set amount of time -- about half a second. The other timer starts up as soon as the circuit gets power, and stays on for almost half a minute. It's responsible for disabling the first timer during the router's boot sequence, when various scripts cycle the LEDs (and would therefore accidentally fire the relay and open the door).

You trigger a 555 by connecting one of its pins to ground, and at first I thought I'd be able to simply hook the GPIO up to the 555's trigger. No such luck: this GPIO, at least, doesn't go to ground when you switch it. Instead it changes between 2 and 3.3 volts (in between is the threshold where the LED happens to light up). The solution to this was to use a voltage comparator chip. This compares its input -- the GPIO -- to a reference value that's supplied on another pin. In this case the reference voltage is 2.5 volts, made with an extremely simple voltage divider (basically just two equal-value resistors in series -- the junction will be at half of the total voltage across them, which in this case is 5. Science!). If the input value is above the reference, it outputs a high value. If it's below the reference, it connects its output to ground. So the comparator's output gets connected to the 555, and voila! We're done.

This is a pretty simple project, but I'm no electrical engineer, so it took me a while to get it working. But eventually I was able to use my SSH client to open the door latch. That's not exactly an ideal interface, though: it's both hard to use and hard to administer. I turned the problem over to my coworkers in the labs, and they built the rest of the system with amazing speed (and, it should be noted, in the middle of a blizzard). They've come up with some pretty impressive solutions -- more on that in the next post!

UPDATE: I've been asked for a schematic of the door-opening circuit; here it is as an EAGLE .sch and as a simple PNG. If you do build this, a word of warning: the initial timer, which is meant to disable the system during boot, behaves somewhat differently in-system than it does on a breadboard. I'm still trying to address this issue -- and have added a bypass capacitor to the schematic in an attempt to do so -- but for us, it's not a particularly pressing issue, as the vulnerability it represents is still substantially harder to take advantage of than it would be to simply smash down the door. Still, if you plan to use this circuit, please be aware of its limitations and understand that you use it at your own risk.

Share |

16 comments

0

Comments for this thread are now closed.

Discussion | Community | Share

Tom • 3 years ago

@vintermark @greg yup, that's exactly right. Just remember that there are two 555s in that chip.

555s are extremely useful circuits, and can do a lot of different things. In this case, I'm using them in "monostable" mode. I'd suggest reading through this page to get an idea of how they work:

http://www.kpsec.freeuk.com/55...

The Wikipedia page for the 555 may also be of help. And, as I mentioned, the MAKE: Electronics book is an excellent resource for learning how to use this chip, and a digital version can be had quite cheaply.

1 • Share ›

Anusha Ramegowda • a year ago

I am actually working on an application of a Linksys WRT54GL wireless router, in which I have to get a web page with doors opened and closed when the SES button on the wireless is hit. I have created the webpage using Java Scripts, wherein the doors get opened and closed on a mouse click. But this is not serving the purpose. I need the doors to open/close on a SES button hit. Can anyone please suggest me how this could be achieved???

0 • Share ›

good themes for parties • a year ago

Must agree that you are one of the coolest blogger I ever saw. Thanks for posting this useful information. This was just what I was on looking for. I'll come back to this blog for sure! I bookmarked this blog a while ago because of the useful content and I am never being disappointed. Keep up the good work.

0 • Share ›

Vintermark • 3 years ago

@Greg

The 555 circuit timing depends on the size of Capacitor and resistors you connect to it. Same as

you it was a while I wrecked..uh I mean worked with these circuits so I do not remember the exact pin numbers. But in the schematic provided I belive that R2 and C3 would control the timing of the relay.

0 · Share ›

Greg • 3 years ago

This project is exactly what I am looking for. I am working on a project that I need to use the lighting of an LED to trigger another circuit to fire. Are there any tips you can give me regarding building the secondary circuit here? Can you adjust the amount of time that the circuit "fires" to keep the door switch activated? I used to mess with this stuff in high school. It has been a while and I am trying to get back into it. Embarrassing how you forget over time! Thanks in advance.

0 · Share ›

Tom • 3 years ago

@lee: not a dumb question at all. The answer is that the LED on the router doesn't provide enough voltage or current to power a relay. Also, it *does* provide *some* of both of these, even when powered off. So at a minimum, the comparator is necessary, and so is a transistor to switch the current to the relay, and so is the voltage regulator to supply that current. The timer is necessary in order to disable the circuit during the router's boot cycle, when the LED flips on and off (we can't control it then).

But! You're right that the other timer is a bit redundant. It would be possible to manage the timing in software, rather than having a brief pulse trigger a slightly less-brief pulse. But I was more comfortable moving that operation out of software.

0 · Share ›

lee • 3 years ago

maybe im just retarded...but couldnt have you just set the LED to turn on for a specific period of time (in this case, 30 seconds), and run the LED wires to the relay, which would trip the wires going to the button? Why are all of the other components necessary?

0 · Share ›

Tom • 3 years ago

@Fercho: I'm afraid I haven't got the compatible firmware handy -- I've lost track of which image is which, embarrassingly. But as I mentioned, you don't actually need the one that I used, which has no characteristics to distinguish it from the vanilla Broadcom 2.4 fw that you can get straight from the OpenWRT project -- other than a working minipython install, which is not strictly necessary for manipulating the GPIOs (you could just as easily use a shell script with the commands listed in the pastebin code from the post).

0 · Share ›

Fercho • 3 years ago

Just amazing! I could SO use this! Would you like to explain in a bit more detail how you did your firmware install?

Would you like to share the custom firmware and explain how we can get the iphone to open the

door?

0 · Share ›

Tom • 3 years ago

@Randy: really? Cheaper? I've priced out some of those options before. The XPort and Charon both cost around $50, which is about the same as the router -- except the router can run Python, do wireless, etc. Believe me, I've examined this problem in some detail. To each their own, and of course if you wanted to do this at scale the solutions you list would be more appropriate, but I still think this is a pretty good technology stack to work with. I'm used to it, anyway.

0 · Share ›

Randy • 3 years ago

I do applaud and admire your efforts, but there are so many cheaper, much easier ways to do this! To name a few solutions, the Digi ME, Lantronix Xport, Web51 Charon, Ethernut, Rabbit, etc.

0 · Share ›

Tom • 3 years ago

@Paul Asadoorian: ah, I see that you're right. Either way, space is in short supply:

> Filesystem Size Used Available Use% Mounted on
> rootfs 2.6M 2.6M 0 100% /
> /dev/root 2.6M 2.6M 0 100% /rom
> tmpfs 7.0M 28.0k 7.0M 0% /tmp
> /dev/mtdblock/4 640.0k 388.0k 252.0k 61% /jffs
> mini_fo:/jffs 2.6M 2.6M 0 100% /

@peter: yeah, OpenWRT is not really designed to be compiled on the router. Its toolchain counts on cross-compilation, which sounds terrifying, but the OWRT team has made it quite pleasant. It insists on a case-sensitive filesystem, though, so don't bother trying on Windows or OS X. I'm on the latter, so I got it done by using a VirtualBox Ubuntu image (both VB and the images are freely available).

0 · Share ›

peter • 3 years ago

Great project. I have the same router laying around and I set about installing OpenWRT on it over the holidays. That went well.

Instead of tailoring the build to include differerent packages, I tried compiling my own source to run on the router, what a nightmare to try and get even a helloworld to compile, but I am trying another route similar to yours.

0 · Share ›

James Heath • 3 years ago

Very very neat.

I have one of the original versions of this router lying around - now all I need to do is think of something to do with it!

one possible tip on the 556 timer at start up - I've had similar issues. it seems they behave oddly at power on sometimes - the trick seems to be to tie the trigger (pin 2) to the threshold (pin 6) rather than just tieing it high.

Also note that the 1st pulse is longer than subsequent ones apparently.

Cheers,

James

0 · Share ›

Paul Asadoorian • 3 years ago

Awesome project! One of the most interesting usages for a WRT54GL I've seen in some time.

Minor correction though, the WRT54GL has 4MB of FLASH, not 2 :)

Great work!

Cheers,
Paul

Labs Members

- Tom
- Ali
- Alison
- Andrew
- Caitlin
- Dan
- Daniel
- Drew
- Eric
- Ethan
- James
- Jeremy
- Kaitlin
- Lee
- Tim

Categories

- data.gov [29]
- openstates [27]
- design [22]
- hackathon [22]
- appsforamerica2 [20]

- congress [18]
- data [18]
- opensource [18]
- recovery.gov [17]
- api [16]
- update [14]
- appsforamerica [13]
- contest [13]
- redesigning the gov [13]
- python [12]

Track Us Down

Join the Sunlight Labs Google Group

Follow The Labs And See What We're Up To

- we just squashed a couple of bugs in the @openstates iOS app -- get the updated version here: https://t.co/rw6SFDsv

- our Influence Explorer API just got even easier to use--we think it's a great source for money in politics data http://t.co/zZmZoQGd

Sunlight Foundation
1818 N Street NW, Suite 300
Washington, DC 20036
202.742.1520