# How to configure Raspberry Pi as a microcontroller

14 Mar 2016 | Seth Kenlon (Red Hat) (/users/seth) | 219 | 4 comments

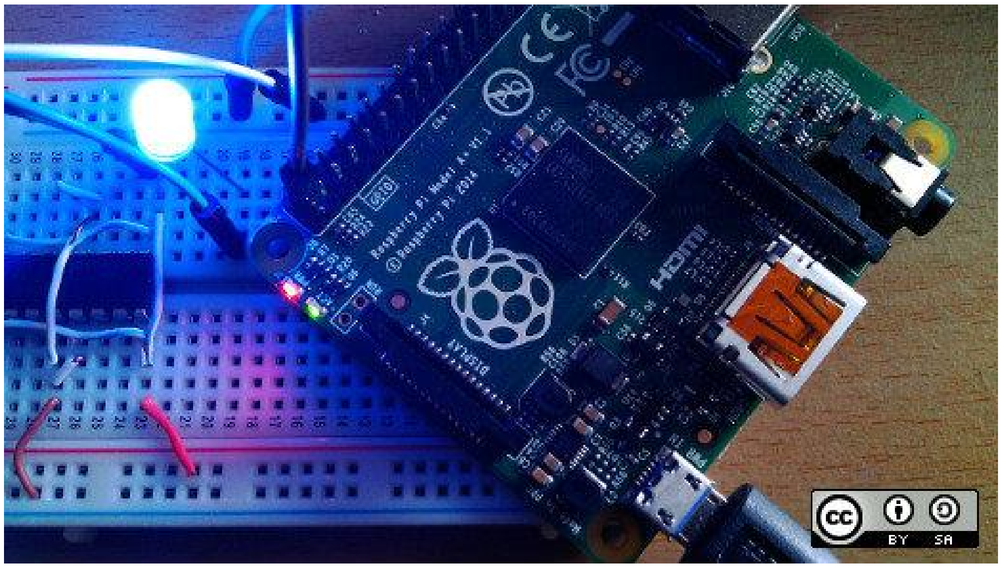


***Image by :*** *opensource.com*

Being small and low-powered, the Raspberry Pi (/resources/what-raspberry-pi) is as popular with electronic hobbyists as it is with educators. As such, it gets associated with the "maker" scene, and sometimes the line between the Arduino (/resources/what-arduino) (and Arduino-style embedded microcontrollers) and the Pi gets blurred.

Fact is, Raspberry Pi and Arduino are very different devices, even though they are often seen on the same workbench together. The Arduino makes a lot of sense for some projects, but for others the Pi may actually be the better choice (if you're triggering complex multimedia events, for example). But if you expect the Pi to be a drop-in replacement for an Arduino, you're in for a few surprises. I've used both the Arduino and the Pi in different craft projects, sometimes because one is clearly the right choice over the other, and other times simply because I had one on hand and not the other. If you're used to the Arduino (or if you've just read up on it a lot and aren't clear on how it and the Pi are different), then here are the extra steps that I have found when getting the Pi to act more like an embedded microcontroller than a mini PC.

### More on Raspberry Pi

- Our latest on Raspberry Pi (https://opensource.com/tags/raspberry-pi?src=raspberry_pi_resource_menu1)
- What is Raspberry Pi? (https://opensource.com/resources/what-raspberry-pi?src=raspberry_pi_resource_menu2)
- Getting started with Raspberry Pi (https://opensource.com/article/16/12/getting-started-raspberry-pi?src=raspberry_pi_resource_menu3)
- Send us your Raspberry Pi projects and tutorials (https://opensource.com/article/17/2/raspberry-pi-submit-your-article?src=raspberry_pi_resource_menu4)

## GPIO

One of the first things people learn about the Arduino is that unless you have stuff to hook up to it, it's not terribly useful; you need input sensors and output devices or else you're programming a microcontroller without anything to control. The Pi is the polar opposite; much of what it does can be entirely software-based, so the tendency to connect outboard electronics is a little less common. But it doesn't have to be.

The Raspberry Pi has a number of GPIO pins (the exact number depends on the model), which can detect input or send output signals. Unlike the Arduino, all of these pins are *digital* IO, meaning that they know two states: on and off. Most Arduino boards also have analog pins, which can detect *degrees* of states, the practical application of which is the ability to connect things like temperature sensors, light sensors, or potentiometers for arbitrary input.

As long as you can work with binary in and out, the GPIO pins of the Pi are very useful. But unlike the Arduino, access to those pins through software is not presumed. To talk to the GPIO pins, you must install GPIO libraries. There are two development libraries on offer:

### RPi.GPIO Python

RPi.GPIO (https://pypi.python.org/pypi/RPi.GPIO) is a Python library that provides you with the ability to read from and output to GPIO pins. It's a fairly high-level interface in terms of what an Arduino user might expect, so there's no SPI or serial functionality. But for simple projects, it works quite well. You can turn an LED on and off, activate a motor, and so on.

### WiringPi C

WiringPi (http://wiringpi.com) is a GPIO access library written in C. Its syntax and general design will feel familiar to anyone who has used an Arduino system, and being a low-level library, it even contains modules that allow you to use external boards (like an Arduino, or similar device) as analogue inputs.

Contributors to the original WiringPi project have written wrappers for Python, Ruby, Perl, Java, and more.

## Real-time performance

Unlike the Arduino, using the GPIO library does not provide real-time performance. The Python libraries have the overhead of Python, which periodically needs to do memory management tasks behind the scenes, and even WiringPi is subject to the Linux kernel's scheduling decisions. In practice, this probably won't actually matter to you unless you are building a device that depends vitally on real-time performance. When preparing for a multimedia project that was to be Pi-based, I did do a few tests in an attempt to boost performance using security **limits**, but they all failed (I never went so far as to re-compile the kernel, so that didn't help). Ultimately I found that the project—a musical interface that was subject to unpredictable input and did need some sense of instantaneous response—never missed a beat, but your mileage may vary depending on your project.

## Launch on boot

Since the Arduino is an embedded system, the program you flash onto its chip is always running. You power the Arduino on and your program runs, looping constantly, until it is powered off. The Pi doesn't do that. When you power the Pi, it loads an OS and then sits patiently waiting for you to log in or perform some action.

If you need embedded behavior, you need to essentially tell the Pi to auto-launch an application upon boot. Better still, you can strip away the processes you don't need so that the boot time is faster. You don't *have* to do it this way (you could just set your Pi's desktop to auto-login, and then set up a startup application), but if you're treating the Pi as an pseudo-embedded device and don't need a desktop, why use the resources to run one? And why introduce that as a variable, knowing that in terms of demo-ing your cool new robot (or whatever you are building), whatever *can* go wrong *will* go wrong?

To make a Pi behave like an embedded OS, first strip away the services you don't need. What those services are depends on your project. If your project has no need for a network, then disable the network service. If your project has no need for a graphical interface, lose the X11 packages from your system. Look through the services being run on boot and disable as many as you can.

You also want the Pi to auto-login. How you implement this depends on what Pi distro you are running.

### Raspbian

To activate auto-login on Raspbian, edit (with **sudo** permissions) the **/etc/inittab** file and change this line:

```
1:2345:respawn:/sbin/getty 38400 tty1
```

to this:

```
1:2345:respawn:/sbin/getty --autologin your-username-here 38400 tty1
```

Replace **your-username-here** with the username of the user you want to auto login. Your Pi probably shipped with **pi** as the default user, but you may have created a new user; if you're unsure, use **whoami** to find your username.

After you make that change, reboot to make sure that it works. The Pi should boot to a shell prompt (not a login prompt).

To run your application immediately after a successful boot, add your application as a command in the file **/etc/rc.local**, above the last line (which should read **exit 0**). For example, here are the last two lines of my sample project:

```
$ sudo tail -n /etc/rc.local
/usr/local/bin/robotnix/robotnix.py
exit 0
```

Make sure that **rc.local** is executable, and that the application you want to launch is executable:

```
$ sudo chmod a+x /etc/rc.local
$ sudo chmod a+x /usr/local/bin/robotnix/robotnix.py
```

Reboot again to test that your application gets launched after a successful auto-login.

## Systemd on Pidora

On Pidora (or any distribution using Systemd), the process is similar, but takes place in a different set of config files.

For the auto-login process, edit the file **>/usr/lib/systemd/system/getty@.service** and comment out this line:

```
ExecStart=-/sbin/agetty --noclear %I 38400
```

And replace it with this variation:

```
## old line ## ExecStart=-/sbin/agetty --noclear %I 38400
## new command
ExecStart=/sbin/agetty --autologin your-username-here --noclear %I 38400
```

It's the exact same thing Debian users would do, except the Systemd file spawns the process instead of **inittab**.

Reboot your Pi to test the auto-login.

Assuming auto-login works as expected (the Pi should boot to a shell prompt, not a login prompt), the next step is to create an auto-launch rule for your application. The internet at large suggests that placing a command in **rc.local** should work (because, in theory, Systemd does a backwards compatibility execution of rc.local), but I've had mixed success with that. I have had nothing but success with a custom Systemd rule.

To write a Systemd rule, create a file, with sudo privileges, called **/etc/systemd/system/robotnix.service** (you can call it whatever you like; I'm using **robotnix** because that was the name of the sample project I generated for this article).

The file contents:

```
[Unit]
Description=Robot Script Startup

[Service]
Type=idle
ExecStart=/usr/local/bin/robotnix/robotnix.py

[Install]
WantedBy=multi-user.target
```

Now enable the service to happen upon boot:

```
$ sudo systemctl enable robotnix.service
```

Reboot the Pi one more time to make sure that your application is being auto-launched.

## Pi or Arduino?

They're both small, they're both powerful, and they both get used in electronics hobby projects, but the Raspberry Pi is not an Arduino. However, the Pi can do a pretty good job of acting the part in a pinch. That said, don't be afraid of trying the Arduino if you haven't and are curious about it. They're both great tools to have in your kit. Using the tips in this article, you can use the Pi as a testbed for future Arduino projects, or you can have the best of both worlds all in one device.

Whatever you end up using on your next hardware project, keep it open, and keep it fun.

Topics:
**Raspberry Pi** (/tags/raspberry-pi) | **Raspberry Pi Week 2016** (/tags/raspberry-pi-week-2016)

(/users/seth)

## About the author

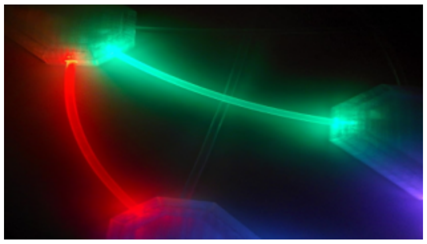
**Seth Kenlon** - Seth Kenlon is an independent multimedia artist, free culture advocate, and UNIX geek. He has worked in the film (http://www.imdb.com/name/nm1244992) and computing (http://people.redhat.com/skenlon) industry, often at the same time. He is one of the maintainers of the Slackware-based multimedia production project, http://slackermedia.info (http://slackermedia.info)

• More about me (/users/seth)

• Learn how you can contribute (/participate)

## Recommended reading

**5 ways to invigorate education with Raspberry Pi**

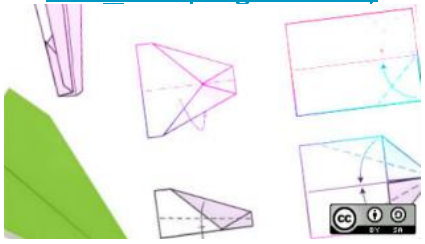

**How to use an Arduino and Raspberry Pi to turn a fiber optic neural**

**How to set up a Postgres database on a Raspberry Pi** (/article/17/10/set-

(/article/17/10/raspberry-pi-marketing-education?utm_campaign=intrel)

**network into wall art** (/article/17/10/fiber-optic-neural-network-art?

postgres-database-your-raspberry-pi?utm_campaign=intrel)



**Give old electronics new life with Linux and Raspberry Pi** (/article/17/10/giving-retro-electronics-new-life?utm_campaign=intrel)

**Make your own Twitter bot with Python and Raspberry Pi** (/article/17/8/raspberry-pi-twitter-bot?utm_campaign=intrel)

**5 ways to use Raspberry Pi in the classroom** (/article/17/8/raspberry-pi-teachers?utm_campaign=intrel)

## 4 Comments

ardweebno on 28 Mar 2016 1

You can run a RTOS on Raspberry Pi:ChibiOS

http://www.stevebate.net/chibios-rpi/GettingStarted.html (http://www.stevebate.net/chibios-rpi/GettingStarted.html)

Seth Kenlon (/users/seth) on 29 Mar 2016 1

Very cool! I'd never heard of this, but it looks like a really nice little GPL OS for the Pi. Amazing; I can't wait to try it out, even though I have no need for actual realtime accuracy in a Pi...yet. I typically run my payload application as a systemd service on the Pi at boot time, and strip away most blatantly unecessary services (dhcp, avahi, plymouth, a few others , I think), and that basically works for me, plus I get the benefit of the familiarity of Linux.

But I'll definitely check out ChibiOS just for fun, because it looks really cool and I imagine I'll eventually have a need for it. Thanks again for the tip, ardweebno!

Dodutils on 08 Jun 2016 0

You told about Arduino but forgot to tell about the ESP8266 that is a 80Mhz microcontroller, include WiFi chipset for IoT that can communicate as Internet client but also as a nano server ! smaller than an

Arduino, come with onboard memory storage, can be programmed in C, LUA, BASIC, very cheap 3$, can go sleep or deepsleep can be updated OTA.

Seth Kenlon (/users/seth) on 09 Jun 2016 0

Great to know about! thanks. It's exciting that there are so many options out there.

Sign up for Opensource.com news

Continue

Find us:

Privacy Policy | Terms of Use | Contact | Meet the Team | Visit opensource.org