We probably all know the TV show «Pimp my Ride» where cool guys take an ordinary car and transform it into something special.

In my last mailbag, I got an ordinary Raspberry Pi Model 3 with an Acrylic housing, a fan, and a battery power supply. All nice things, but not good enough for me. That is what we will change now: I will pimp my Raspi!

Without a fan, even with the standard small heat sinks, the Raspbi 3, runs hot. With the CPU test program used by others, my Pi 3 reaches 73 degrees, even in Switzerland, where we still have glaciers in our mountains… The very small fan which came with the acrylic case is quite effective. Even under heavy load, it keeps the temperature below 45 degrees. You hear the fan if it runs, and I am pretty sure, it will not survive endless hours. So, I want, that it only runs when the CPU temperature is above a particular value, in my case 55 degrees centigrade.

A Raspi battery pack is a nice thing, because you can run the Raspi independently from a power source. But Linux systems should not just be switched off. This can lead to data loss, or, in worst case, problems for the next boot. That is, why I want, that my Raspi performs an “official” shutdown before the battery is too low.

If you look at a battery you do not see how much it is charged. This is why I want also a nice “fuel gauge ” for my battery pack.

And the pimping should not cost an arm and a leg. We should be able to stay well below 10 dollars additional cost (the fan, the battery pack, and the case not included).

So, let’s start with the automatic fan control. It consists of two parts: Hardware and Software. The hardware, fortunately, it is simple and cheap. I use a logic level N-channel FET in a TO-220 housing and connect some wires to it. Basically, it acts as a switch between the fan and ground. The gate is controlled by a GPIO, in my case, GPIO17. If the pin is high, the fan is on, and if the pin is low, the fan is off. We can test this on a breadboard before we solder the things together. But why do I use such a beefy transistor which is capable to switch 50 amperes? The fan only needs about 100 mA. This has a simple reason: Such a transistor costs only a few cents and you can use it as a base to build your device, protect the bare wires with heat shrink tubes, and stick it into the housing. With a smaller case, we would be obliged to use a PCB.

The wires are connected to the Raspi using female Dupont connectors. Of course, you can use any other logic level N channel FET. BTW: What does logic level mean? N-channel FETs are switched on by a positive voltage between gate and source. It is called VGS. If you look at the data sheet of your desired FET, you usually find more than one value. Each line shows the resistance between Source and Drain at a specific VGS. voltage. The ILRZ44 for example, has a resistance of 0.028 Ohm if you use a Arduino with 5 volt logic. If you use a ESP8266 with 3.3 volt, we find no value, but a chart. In this chart, we see, that it still is capable to conduct about 20A at this voltage. If we look at at a similar, non logic level FET, the IRF44, we see, that we need 10 volt to get it fully on. This transistor would be completely off at 3.3 volt and nearly off at 5 volts. So, you see, for the IRF types, the “L” stands for logic level and the “F” for the rest.

So, the hardware is ready and we can go-on to the software. I use python to do this job. There are many other ways, but I want to concentrate my efforts on python anyway. So, this is a good training for me.

I suggest you create a directory for your scripts by typing cd /mkdir Scripts and then get the file called “fan\_shutdown.py” from my link in the comments.

I will not go through the code in detail. The principle is, that we measure the CPU temperature using the command “vcgencmd measure\_temp”. This can also be used as a stand-alone command. Based on this temperature, we decide to switch the fan on or off. The “off” point is 5 degrees below the “on” point because we want to avoid frequent on- and offs.

The main loop handles the fan and afterwards sleeps for 5 seconds. If you remove the comment sign in the print statement and start your code in your terminal, it should show the CPU temperature every 5 seconds. Now, we know, that the code is ok. If you set the temperature to the right level, the fan should start to run after a while…

Great. But now, we want to start this code automatically and run it in the background. How can we do this? We have to create an additional file, called laucher.sh. You also can wget it from my github.

This file has only one active line: sudo python /home/pi/Scripts/fan\_shutdown.py &. The first part calls our program, as we did before. The & tells the computer, that it should run the command in the background. Now, we could execute laucher.sh using the bash command. But we want to make it executable using the command “chmod 755 launcher.sh”. Now, we can execute it by issuing the command “sh /.launcher.sh”. Make sure, you commented the print statement in your python program before you do that!

The next step is to create a directory for the log file. In this directory, you will find any error messages if something goes wrong with our experiment. Not a bad idea…

We do this with the command cd and mkdir logs. Now, we are just one step before the finish line: We have to add one line in a file called “crontab”: So, you type: sudo crontab -e” and enter the line “@reboot sh /home/pi/bbt/launcher.sh >/home/pi/logs/cronlog 2>&1” at the very end of the file. Save the file and reboot. This statement will add all error messages at the end of our log file. If you now reboot, the fan should start after a while. If not, you have to issue some commands to “heat” your Raspi a bit.

If you had a close look at the Python file, you saw, that the software part for our next feature is already implemented. So, we only have to build the hardware. As said at the beginning, I want a nice fuel gauge for my battery and an automatic shutdown of the Raspi before the battery is dead. Viewers who watched my last mailbag saw, that I got a nice-looking fuel gauge for a 3.7-volt battery. It looks nice if mounted on my raspy. We have to connect it to the battery, not to the output of the battery pack. And now, it nicely shows us the state of the battery. Simple and cheap. Now, we see when the battery is low and we have to shut the Raspi down. But I want that feature automatic. We now already have a Python program which executes automatically during boot. We only need a signal if the battery is low.

To create such a signal, we need two parts: A constant voltage source and a comparator. For that, we could use a Zener diode and an op amp. Maybe there are even specialized ICs around for exactly this purpose. But we are hackers! Look at what we already have: This fuel gauge has exactly what we need: If we could shut our Pi safely down as soon as the last bar disappears, we would reach our goal. But as usual with these Chinese parts, the IC has no text on it. But I “reverse-re-engineered” it for you:

Out of the 14 pins, 4 are responsible for the bars. An pin 14 if responsible for the last bar. Unfortunately, the signal at this pin is only very small. It changes between 0.2 and 1.6 volt. This signal cannot be used for the GPIOs of the Raspi because we need more than 2 volt to safely switch the pin on. So, what to do? As with the FAN, we could use a N-channel FET to switch the pin. Unfortunately, we discovered, that 1.6 volt is not enough to switch a FET on. But fortunately, we have other weapons in our shelf: A normal NPN transistor. These transistors are switched by current, not by voltage as the FETs. And they start to conduct if the base voltage is about 0.7 volt above the emitter voltage. Exactly what we need. Recently I purchased an assortment of transistors. So, I use one of these and connect the base through a 1k current limiting resistor to pin 14 of the “fuel gauge”. Just to be sure, I add a 100nF capacitor to reduce noise. Then I connect the collector via a 4.7 k resistor to 3.3 volt. I do not connect it to the battery, because the battery, fully loaded, is about 4.2 volt and well over the 3.3 volts of our precious Raspi. Now we can test the whole thing. This time, I mount it on a small PCB, cover the whole device with a heat-shrink tube, and fill it with hot glue. This protects it and I can glue it wherever I want. Again, I connect it with Dupont wires to the Pi pins.

If we look at our Python program, we already see the function “handleBattery”. This function checks GPIO18 and if high, shuts the Pi down. Here with the option 1 minute. This is just for testing that you see, that everything works. In production, you can replace the 1 with “now” and it shuts immediately down.

So, summarized, we:

Learned, that a small fan is sufficient to keep the Raspi 3 cool

We created a simple hardware to switch a fan on and off using a Raspi GPIO

Created a Python program to read the CPU temperature and, based on this temperature, switches the fan on- and off. So, we have now a cool Raspi without too much noise.

We also created an environment to auto-launch applications during boot including an error log if something goes wrong

We connected our nice “battery fuel gauge” to our battery pack and

Hacked it to provide a signal if the battery level is low

During this process, we learned how to use N-channel FET transistors and the difference between logic level and “normal” FETs

At the end, a normal NPN transistor rescued us and made a simple automatic shutdown possible

So, we achieved all goals and have now a really “pimped” Raspi.

By the way, my battery pack always reset the Raspi if I disconnected and re-connected the charging cable during operation of the Raspberry. After adding this beefy capacitor at the output pins, it runs ok now.

Recently, I got a second UPS hat in the mail, but so far, did not test it.

I hope, this video was useful or at least interesting for you. Bye.

<http://www.instructables.com/id/Raspberry-Pi-Launch-Python-script-on-startup/?ALLSTEPS>

<https://www.element14.com/community/docs/DOC-78055/l/adding-a-shutdown-button-to-the-raspberry-pi-b>

<http://www.instructables.com/id/RPi-3-Cooling-Tests/step1/Test-1/>

https://www.raspberrypi.org/forums/viewtopic.php?f=63&t=147420&p=970279