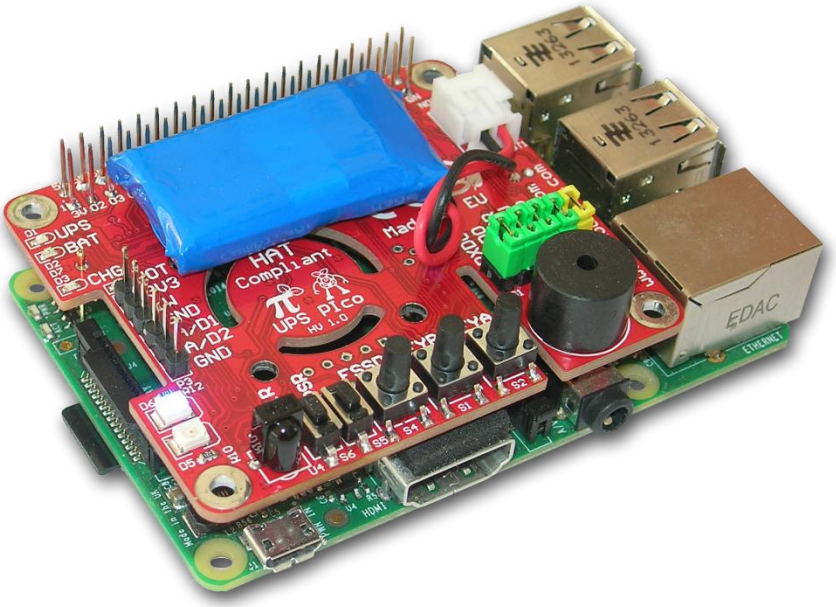
UPS PIco

Uninterruptible Power Supply

with Peripherals and I²C control Interface

for use with

Raspberry Pi® B+, A+, B, and A



HAT Compliant

Bootloader and File Safe Shutdown Functionality

Version 1.0

Intelligent Modules for your Raspberry Pi®

Designed and Manufactured by PiModules and ModMyPi
www.pimodules.com www.modmypi.com

Document Revisions

Version	Date	Modified Pages	Modified Sections	Comments
1.0	18/01/2015	none	none	First Public Document Release

Table of Figures

Credits

Our Company would like to thank the following people that reviewed and, many times, commented and corrected this document before we released it to the public domain.

Marcello Antonucci from Italy

Vit Safar from Slovakia - who provide the initial version of python bootloader script

System Overview

Introduction

The **UPS PIco** is an advanced uninterruptible power supply for the Raspberry Pi® that adds a wealth of innovative power back-up functionality and development features to the innovative microcomputer!

The standard **UPS PIco** is equipped with a 300mAh LiPO battery specially designed to enable safe shutdown during a power cut. Additionally, this can be easily upgraded to the extended 3000mAh version, which enables prolonged use of a Raspberry Pi for **up to 8 hours** without a power supply connected!

The **UPS PIco** features an embedded measurement system that continuously checks the powering voltage of the Raspberry Pi®. When the cable power on the Raspberry Pi® is absent, insufficient, or the device detects a power failure, the **UPS Pico** automatically switches to the unit's battery source. The module then continues to check the voltage on the Pi and switches automatically back to the regular cable supply when power is once again available.

The **UPS PIco** is powered and the battery pack intelligently charged via the GPIO pins on the Raspberry Pi®, so no additional cabling or power supply is required.

The **UPS PIco** is designed to be 100% compliant with HAT standards for the Raspberry Pi® B+ and A+, and is mechanically compatible with the original Raspberry Pi® models A and B when an extension header is used. In addition to this, because the **UPS PIco** requires no external powering and fits within the footprint of the Raspberry Pi®, it is compatible with most cases.

The **UPS PIco** can also be equipped with an optional **Infra-Red Receiver** which is routed directly to GPIO18 via the PCB. This opens the door for remote operation of the Raspberry Pi®and **UPS Pico**!

Finally, the **UPS Pico** features an implemented Automatic Temperature Control **PWM fan controller**, and can be equipped with a micro fan kit, which enables the use of the Raspberry Pi® in extreme conditions including very high temperature environments.

Applications

UPS PIco is equipped with plenty of features which make it an extremely useful tool for Raspberry Pi® project development. It not only provides powering continuity, but also offers extra user programmable LEDs, sensors, buttons and I/O's. The unit also features a dedicated **10-bit analogue to digital converter** with two channels making it the perfect board for remote and unmanned sensor deployment. These extra features result in the **UPS Pico** being a superior all-in-one device, perfect for many innovative projects and embedded applications.

Features

The list of features of the **UPS PIco** is as follows:

- Raspberry Pi B+ **HAT Compliant**
- **Plug and Play**
- **Smart Uninterruptible Power Supply (UPS)**
- **Integrated LiPO Battery** (8-10 Minutes of Power Back-Up)
- **Intelligent Automatic Charger**
- **No Additional External Power Required**
- **Optional 3000 mAh** Battery for 8 Hours Run-Time (Not Included)
- **5V 2A Power Backup (Peak Output 5V 3A)**
- Integrated Software Simulated **Real Time Clock (RTC)** with Battery Back-Up
- **File Safe Shutdown** Functionality
- Raspberry Pi B+ **Activity Pin**
- **PWM fan control** (Fan Not Included)
- **2 User Defined LEDs**
- **2 User Defined Buttons**
- **Integrated Buzzer** for UPS and User Applications
- **Status Monitoring** - Powering Voltage, UPS Battery Voltage and Temperature
- **I2C PICo Interface** for Control and Monitoring
- **RS232 Raspberry Pi** Interface for Control and Monitoring
- **XTEA Based** Cryptography User Software Protection
- 2 Level **Watch-dog Functionality** with **FSSD and Hardware Reset**
- **Raspberry Pi B+ Hardware Reset Button via Spring Test Pin** (Not Included)
- **Jumpers for Raspberry Pi B+ Pin** Functionality Selection
- **Stackable Header** for Add-On Boards
- **Boot Loader** for Live Firmware Update
- Compatible with **Intelligent IR Remote Power ON/OFF (PowerMyPi)**
- **Integrated ESD-Protected 2 Channel A/D 10 Bit Converters 0-5.2V**
- **Integrated ESD-Protected 1-Wire Interface**
- **Labeled J8 Raspberry Pi B+ GPIO Pins** for Easy Plug & Play
- **Infra Red Receiver** Sensor Interface (IR Not Included)
- **Upgradable** with **PIco Add-on Boards**
- **Fits Inside Most Existing Cases**

UPS Pico Bootloader

UPS PIco is equipped with bootloader functionality. The bootloader is a functionality that allows the user to keep the firmware up-to-date by downloading newer versions from the company website. This ensures that the **UPS PIco** is flexible and always with the latest version of firmware. New versions of the firmware are announced on the www.pimodules.com and can be downloaded by the user. The **UPS PIco** firmware is smart enough and automatically recognizes which hardware model of **UPS PIco** it is running on, adjusting the available functionality set to it. The boot loading procedure uses the Raspberry Pi® serial interface. Therefore the user need to take care to keep this inteface free from any other applications during the firmware uploading process. A few simple steps are need to make the **UPS Pico** and Raspberry Pi® cooperative in order to upload the newer firmware. A detailed description how to do this is provided in the next sections.

Setting Up the RaspberryPi® Serial Port for other applications (i.e. Bootloader)

By default Raspberry Pi®'s serial port is configured to be used for console input/output. While this is useful if you want to login using the serial port, it means you can't use the Serial Port in your programs. To be able to use the serial port to connect and talk to other devices, the serial port console login needs to be disabled.

Needless to say you will need some other way to login to the Raspberry Pi®, it is suggested doing this over the network using an SSH connection.

Disable Serial Port Login

To enable the serial port for your own use you need to disable login on the serial port. There are two files that need to be edited

The first and main one is /etc/inittab. You can edit it by issuing this command[1]:

$sudo nano /etc/inittab

This file has the command to enable the login prompt and this need to be disabled. Edit the file and move to the end of the file. You will see a line similar to:

T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100

Disable it by adding a # character to the beginning. Save the file.

#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100

[1] throughout this manual we will assume that text files are edited with nano and that the user knows how to save the file, after editing. However, the user is free to use any other text editor that he or she feels comfortable with.

Disable Boot-up Info

When the Raspberry Pi® boots-up, all the boot-up information is sent to the serial port. Disabling this boot-up information is optional and you may want to leave this enabled as it is sometimes useful to see what is happening at boot-up. If you have a device connected (i.e. Arduino) at boot-up, it will receive this information over the serial port, so it is up to you to decide whether or not this is a problem.

You can disable it by editing the file /boot/cmdline.txt:

sudo nano /boot/cmdline.txt

The contents of the file look like this

dwc_otg.lpm_enable=0 console=ttyAMA0,115200 kgdboc=ttyAMA0,115200 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait

Remove all references to ttyAMA0 (which is the name of the serial port). The file will now look like this:

dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait

Reboot

In order to enable the changes you have made, you will need to reboot the Raspberry Pi

$sudo shutdown -r now

Test the Serial Port

A great way to test out the serial port is to use the ***minicom*** program. If you don't have this one installed, run

$sudo apt-get install minicom

Run up ***minicom*** on the Raspberry Pi® using

minicom -b 38400 -o -D /dev/ttyAMA0

Make sure that proper jumpers are installed (**RXD0** and **TXD0**) and tht no other boards using the serial port are placed on the top of the **UPS PIco**.

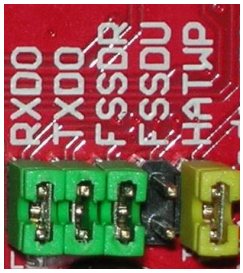


Figure 1 UPS PIco Jumpers

By pressing the **UPSR** (the UPS Reset Button) you should see on the ***minicom*** screen the **UPS PIco** welcome message after reset. This will ensure you that the **UPS PIco** is cooperating properly with your Raspberry Pi ®.

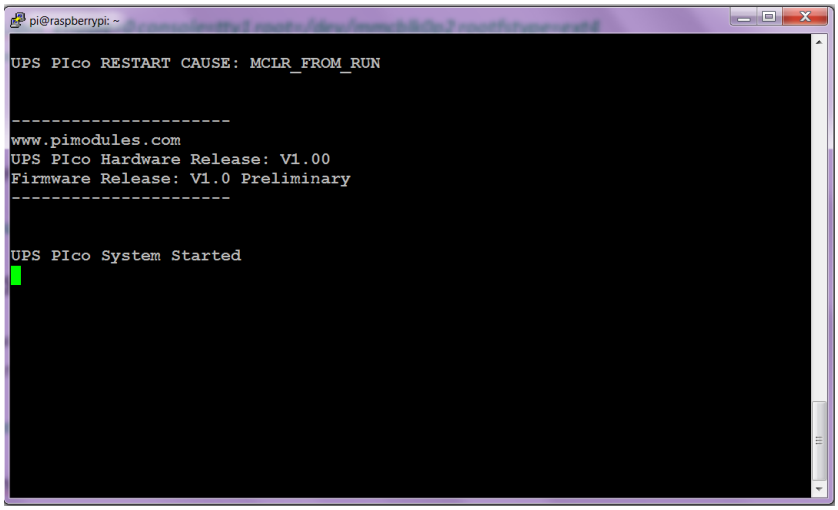


Figure 2 Minicom screenshot while the UPS PIco restarts

Be careful when pressing the **UPSR** (the UPS Reset Button) to avoid pressing of the **RPIR** (the Raspberry Pi® Reset Button), because it will make also a reset to the Raspberry Pi® , and cause immediately cutting of the communication between **UPS PIco** and Raspberry Pi® .

NOTE1: Resetting of the **UPS PIco** does not reset the Raspberry Pi®

NOTE2: Resetting of the **UPS PIco** does reset the simulated RTC to default values

NOTE3: Resetting of the Raspberry Pi® does not reset the **UPS PIco** (the RTC is still working with the proper values)

NOTE4: Resetting of the Raspberry Pi® is possible only if the **Reset Gold Plated Pin** is installed (soldered)

Setting-up the I2C interface and RTC

The **I²C** Ports on the Raspberry Pi® are not enabled by default. Follow these steps to enable the **I²C** port and then the **RTC** communicating trough **I²C** with RaspberryPi®.

First it is needed to edit the config file that disables the **I²C** port by default. This setting is stored in **/etc/modprobe.d/raspi-blacklist.conf**.

sudo nano /etc/modprobe.d/raspi-blacklist.conf

Once this file is open find this line **blacklist i2c-bcm2708** and comment it out by adding **#** to the front of it.

#blacklist i2c-bcm2708

Edit /etc/modules

sudo nano /etc/modules

And add the following:

i2c-bcm2708
i2c-dev
rtc-ds1307

Add the modules to the kernel (they will automatically be added on subsequent boots from /etc/modules):

sudo modprobe i2c-bcm2708
sudo modprobe i2c-dev
sudo modprobe rtc-ds1307

Reboot the system

sudo reboot

Install **I²C** tools

sudo apt-get install i2c-tools

Look for ID #68 with i2cdetect. This must be done in two alternative ways:

- On a 256MB Raspberry Pi Model A+:

  sudo i2cdetect –y 0

- On a 512MB Raspberry Pi Model B+:

  sudo i2cdetect –y 1

The result should look like:

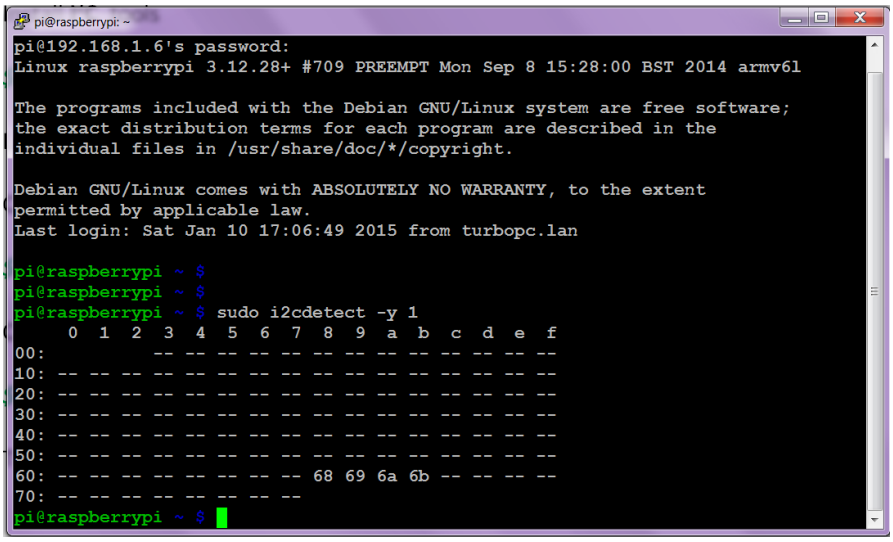


Figure 3 I2C UPS PIco Interface and Simulated DS1307 Clock detection

Then, running as root, do the following for model of RaspberryPi® you have

- On a 256MB Raspberry Pi Model A+:

  sudo bash
  # echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-0/new_device
  # exit

- On a 512MB Raspberry Pi Model B+:

  sudo bash
  # echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
  # exit

The result should look like:

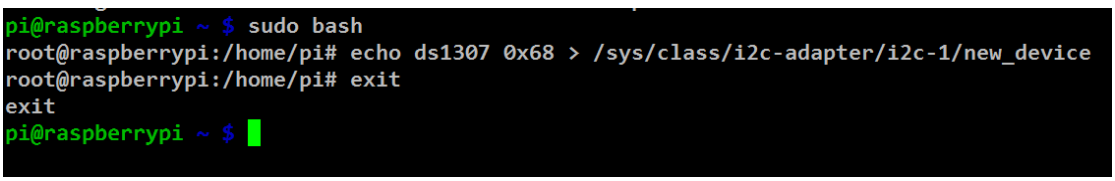


Figure 4 UPS PIco Simulated DS1307 Clock sudo bash commands execution

Then check for time from the clock (which will show Sat 01 Jan 2000 if it is the first time that it is used):

sudo hwclock -r

Then write the current system time to the clock:

sudo hwclock -w

Then edit /etc/rc.local:

sudo nano /etc/rc.local

and add the following before exit 0:

- On a 256MB Raspberry Pi Model A+:

  echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-0/new_device
  hwclock -s

- On a 512MB Raspberry Pi Model B+:

  echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
  hwclock -s

Running the UPS Pico Bootloader

In order to keep **UPS PIco** module firmware up-to-date, an embedded serial bootloader has been implemented. In order to upload the new firmware to the **UPS PIco** a dedicated bootloader ***python*** software needs to be running on the Raspberry Pi®. It is mandatory to have previously installed the ***python*** and ***I2Ctools***. The activation of **RTC** is not mandatory for the new firmware uploading.

There are two ways to invoke the bootloader mode and to upload the new firmware:

1. The manually initiated one:

   It is invoked when the **UPS PIco** module starts from **UPS PIco** RESET **UPSR** key, when pressed the **KEYA** button. The user must press and hold the **UPSR** button, then press the **KEYA** button while the **UPSR** button is still pressed, then relase the **KEYA** button and finally release the **UPSR** button. As a result the **UPS PIco** module enters the bootloader mode, informing you about it, by lighting the red LED. It will keep waiting for the start of the new firmware uploading as long as needed. The following command needs to be executed in order to start new firmware uploading procedure. Please make sure that the new firmware has been already downloaded and stored onto the Raspberry Pi® mass storage.

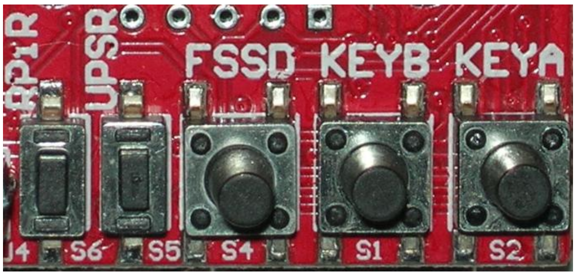


Figure 5 UPS PIco Keys

sudo python picofu.py -f UPS_PIco.hex

Just after the new firmware starts to upload, the red LED lights off and the blue LED starts flashing, and keeps flashing until the firmware is completly uploaded. The following screen will be visible.

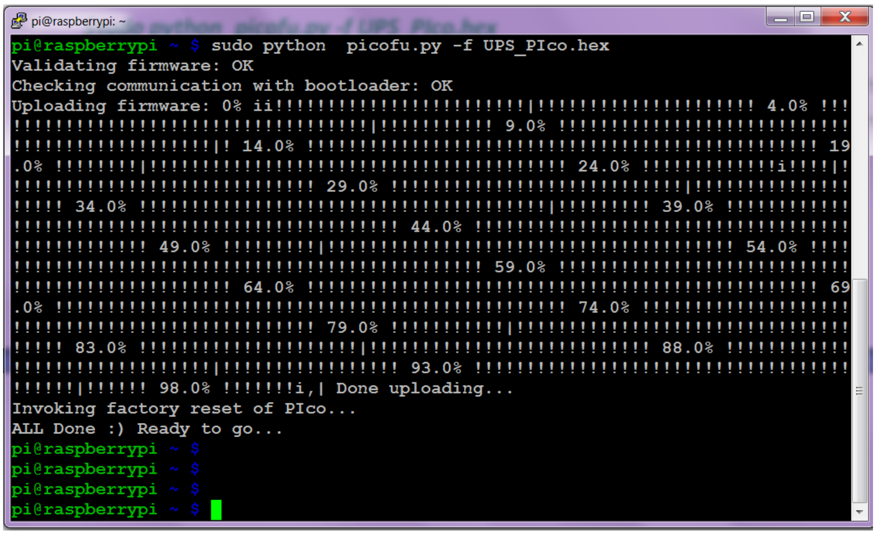


Figure 6 UPS PIco uploading new firmware screenshot

2. The automatically initiated one:

   It is invoked by running the following command line

   sudo i2cset -y 1 0x6b 0x00 0xff && python picofu.py -f UPS_PIco.hex

   Execution of this command will light red LED for a short time, and after that the BLUE LED starts flashing and the new firmware is uploaded. This automatic firmware upload can be used when the Raspberry Pi® is placed in remote place and need to be uploaded remotely. This automatic firmware uploading uses the **PICo** interface (**P**eripherals **I**2C **Co**ntrol – PICo – Interface) which is described in another document.

RaspberryPi® File Safe Shutdown Procedure and RaspberryPi® RUN

The **File Safe Shutdown** feature guarantees to the user that a proper shutdown of the RaspberryPi® will be executed when switching off the system. The **FSSD** (**F**iles **S**afe **S**hut **D**own) can be executed automatically when some events happen, or on user request by pressing the **FSSD** Button. The proper usage of the **FSSD** needs a Python script running on the RaspberryPi®. This FSSD script covers also one additional functionality (the RaspberryPi® RUN) that informs the **UPS PIco** if the RaspberryPi® is running.

In order to support the **File Safe Shutdown** procedure, a simple script should be stored on the RaspberryPi®. There are many simple scripts concerning this matter, which can be easily found over the internet; however, we provide one example that can be easily implemented. Scripts could be divided into two basic categories:

- Interrupt based
- Loop based.

The user of the **UPS PIco** module is basically free to use his own script; however, the user should always keep in mind some of the basics of the implemented circuit on the **UPS PIco** board:

- There are no Pull-Up resistors on the **UPS PIco** board therefore the user needs to setup the RaspberryPi® resistors
- The Pin which has been dedicated for the **FSSD** is the pin **GPIO.27**
- The Pin which has been dedicated for the **RUN** is the pin **GPIO.22**
- Before this functionality will be used, the user needs to put a proper jumper on the **UPS PIco** Board otherwise it will not work.
- If the user does not need this functionality (**which however is highly recommended**), or needs to use these pins for other applications, the **GPIO.27** pin and **GPIO.22** pin could be used for other applications provided that the associated jumpers are open (removed). However some important functionalities will become unavailable, as the **UPS PIco** interacts with RaspberryPi® via these pins.

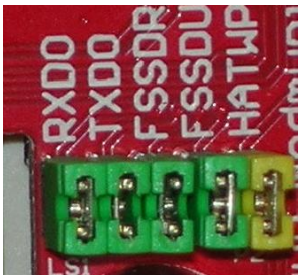


Figure 7 UPS PIco Jumpers

The **File Safe Shutdown** Functionality saves files from any corruption by executing a proper shutdown of the RaspberryPi®. This is executed when the **FSSD Button** is pressed for a longer time than 2 seconds.

If you have initiated the minicom on the RaspberryPi® will also see the following messages on the screen:

UPS PIco System Started File Safe Shutdown Procedure

Here below describes the simple procedure on how to implement the simplest Python script used for the **Safe File Shutdown**.

You have to add some code to enable the Python script created to run when the RaspberryPi® boots up. Type in:

sudo nano /etc/rc.local

and then add in the following code:

sudo python /home/pi/picofssd.py

just before the line that says:

exit 0

save and exit, then create a new file with name picofssd.py

sudo nano /home/pi/picofssd.py

The screen below shows the script that need to be entered using nano.

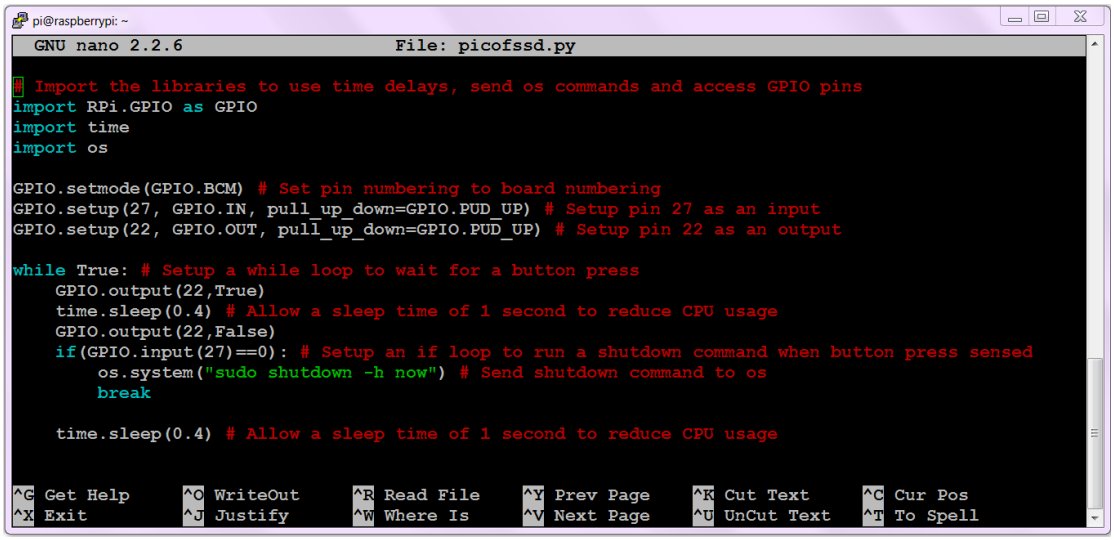


Figure 8 File Safe Shutdown and RUN Python Script - picofssd.py

This python script is also available for download from the website www.pimodules.com . You can easily check if your script is running by just writing on the command line

sudo python /home/pi/picofssd.py

and then pressing the **FSSD button** for more than 2 seconds. If you have properly performed the above tasks, the computer should print on the screen the following message and then shutdown.

The system is going down for system halt NOW!

After the **Safe File Shutdown** you can restart your RaspberryPi® using the **Reset Functionality** (using the **RPiR** button) or remove and eter again the power supply. If the system is running from battery power back-up, then it goes automatically to Low Power Mode.