Comment allumer le Raspberry, et qu'il commande son extinction ?

Contenu

LTC2951 (ze best solution?)

https://www.astuces-pratiques.fr/electronique/bistable-a-deux-transistors-schema

http://cds.linear.com/docs/en/datasheet/295112fb.pdf**Erreur ! Référence de lien hypertexte non valide.**

http://www.mosaic-industries.com/embedded-systems/microcontroller-projects/raspberry-pi/on-off-power-controller

https://www.raspberrypi.org/forums/viewtopic.php?f=107&t=113789**Erreur ! Référence de lien hypertexte non valide.**

https://www.raspberrypi.org/forums/viewtopic.php?t=128019**Erreur ! Référence de lien hypertexte non valide.**

https://www.raspberrypi.org/forums/viewtopic.php?t=117863**Erreur ! Référence de lien hypertexte non valide.**

https://www.raspberrypi.org/forums/viewtopic.php?t=102015

https://lowpowerlab.com/guide/atxraspi/

Simple (ze KISS solution?)

https://raspberrypi.stackexchange.com/questions/4719/how-to-shut-down-rpi-when-running-headless**Erreur ! Référence de lien hypertexte non valide.**

http://www.recantha.co.uk/blog/?p=13999**Erreur ! Référence de lien hypertexte non valide.**

https://github.com/TonyLHansen/raspberry-pi-safe-off-switch**Erreur ! Référence de lien hypertexte non valide.**

How to shut down RPi when running headless

I know it's 3 years after the original question. But I just got my Raspberry Pi and I'm having trouble shutting it down if I forgot to connect it to a monitor screen and it doesn't have any network connection.

I've written a small Python script to automatically shut it down within 60 seconds by plugging in a thumbdrive containing file named "pi_auto_shutdown".

Just call this script from rc.local.

```
shutdown_loop_delay = 60
shutdown_flag_file = 'pi_auto_shutdown'

def poll_shutdown_flag():
    """check whether a shutdown flag file in a usb drive exists"""

    ## run mount command
    ## sample mount output: "/dev/sda1 on /media/path/"
    output, error = subprocess.Popen('mount', shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE).communicate()
    if len(error) > 0:
        log('mount error: {}'.format(error))
        return False

    ## parse mount output
    for output_line in output.split('\n'):
        output_words = output_line.split(' ')

        if len(output_words) < 3:
            continue

        if output_words[0].startswith('/dev/sd'):
            flag_file_path = os.path.join(output_words[2],
shutdown_flag_file)
```

```
            if os.path.isfile(flag_file_path):
                return True

    return False

def shutdown():
    """shutdown the system immediately"""
    subprocess.Popen('sudo shutdown -h now', shell=True).communicate()

def loop_shutdown():
    while True:
        time.sleep(shutdown_loop_delay)
        if poll_shutdown_flag():
            shutdown()

loop_shutdown()
```

Shutdown your Raspberry Pi using a paper clip

When you're running your Raspberry Pi headless, one of the first things you should concern yourself with is how to shut it down safely. Shutting it down by simply pulling the power out can result in a corrupt SD card or damaged files. If you had it connected to a network, you could of course just SSH into it and issue the 'halt' command. However, what if you couldn't connect to it? Or what if you just wanted a way to simply and easily shut it down without touching a keyboard at all?

Here's where Adafruit steps in and gives us a simple script that will monitor a GPIO pin and shut down if it detects a change in state. What use is that? Well, it means you just simply need to connect two pins on the GPIO together with a paper clip (or anything conductive) to issue the halt command. I've taken the Adafruit tutorial and some other guides online and put together the following instructions.

Installation

First of all, you will need a Pi with Git installed on it.

```
sudo apt-get install git
```

Next, you download the script from Adafruit's Github:

```
git clone https://github.com/adafruit/Adafruit-GPIO-Halt
```

If this doesn't work, try:

```
git clone git://github.com/adafruit/Adafruit-GPIO-Halt
```

An aside: The script is written in C, so is able to be modified if you want to, say, light up an LED when the halt command is issued. You *could* re-write it as a Python program, of course, which might make it easier for some to adapt.

Change into the new directory, compile and install:

```
cd Adafruit-GPIO-Halt
make
sudo make install
```

This installs the script to /usr/local/bin/gpio-halt. You then need to run it as a service.

Please note

In the following sections, if you're using an older Pi with a 26-pin GPIO header, use GPIO pin 7 instead of 21.

Run it automatically on Jessie

If you're running Raspbian Jessie, you will need to do it via systemd:

```
sudo nano /lib/systemd/system/gpio-halt.service
```

This will create a file and open it. The following is the contents of that file:

```
[Unit]
Description=Short pins 21 and ground to shutdown the Pi
After=multi-user.target

[Service]
Type=idle
ExecStart=/usr/local/bin/gpio-halt 21 &

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

Then, make the script executable by the right users:

```
sudo chmod 644 /lib/systemd/system/gpio-halt.service
```

Then, tell systemd to use the script:

```
sudo systemctl daemon-reload
sudo systemctl enable gpio-halt.service
```

And reboot your Pi

```
sudo reboot
```

When the Pi comes back up, you can check the status of the service by doing:

```
sudo systemctl status gpio-halt.service
```

Run it automatically on Wheezy

If you're running the older Raspbian Wheezy, you will need to do it via rc.local:

```
sudo nano /etc/rc.local
```

Before 'exit 0', add the line:

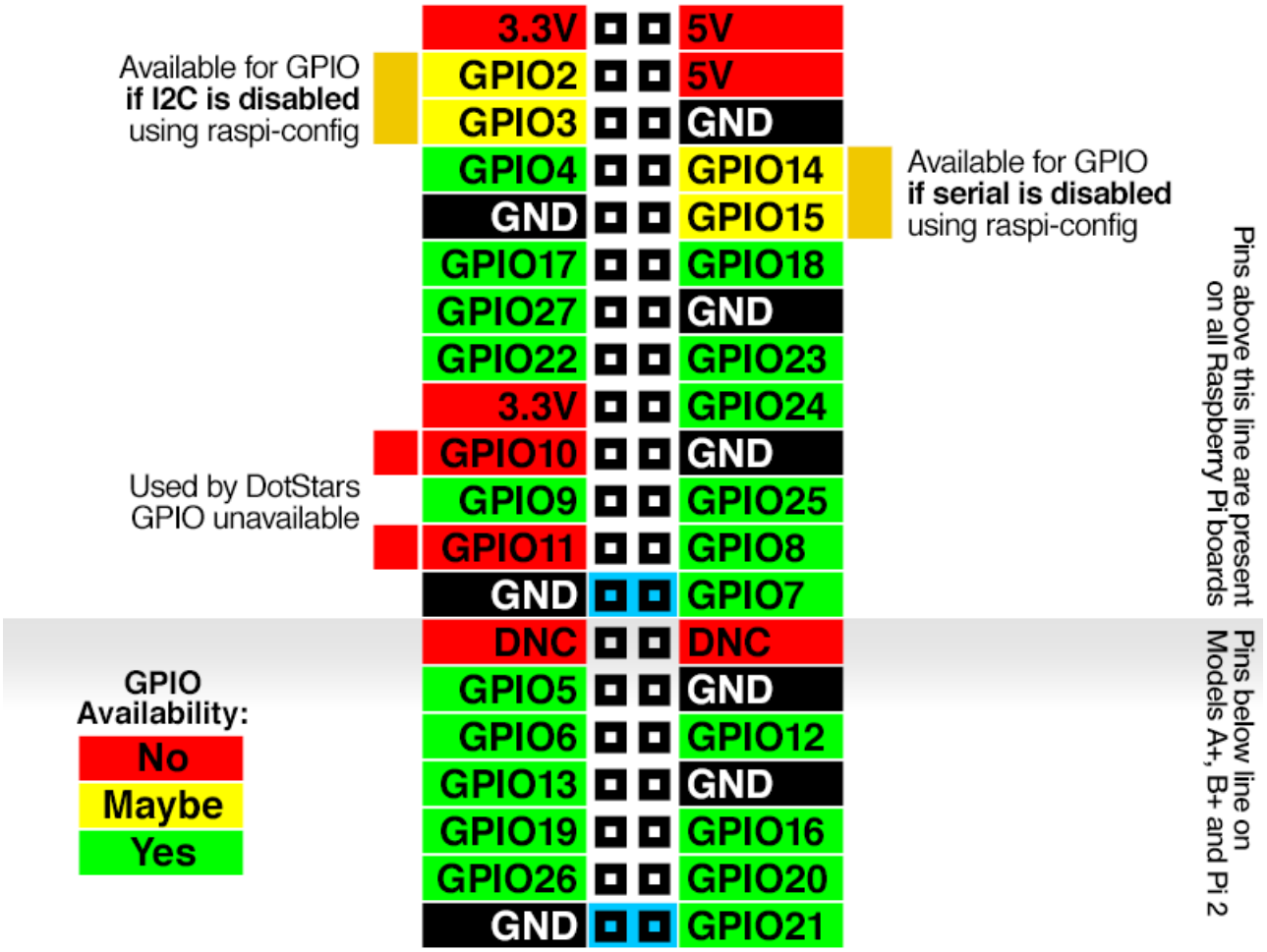
```
/usr/local/bin/gpio-halt 21 &
```

Now reboot your Pi:

```
sudo reboot
```

Now use it

Take a paper clip (or other conductive object) and touch the last two vertical GPIO pins at the same time. If you're using a 26-pin Pi, it will be GND and GPIO7. If you're using a 40-pin Pi, it will be GND and GPIO21. See the diagram below. The first time you do this, it might be worth having a monitor connected so that you can see it happening. Wait about 15 seconds for the halt procedure to complete and then unplug your Pi safely.



Acknowledgements

Thanks to Alex Eames for spotting the Adafruit shutdown script. Thanks to Adafruit, obviously. Thanks to Matt Hawkins for the systemd instructions.

The Adafruit shutdown script

Let's set up the Pi to run fully headless now.

I recommend installing our *gpio-halt* utility. This tiny program lets you connect a button that performs an orderly system shutdown. (Just pulling the plug on a Linux system is a bad idea and can corrupt the SD card, so you'd have to start over.)

Download file
Copy Code

```
1. git clone https://github.com/adafruit/Adafruit-GPIO-Halt
2. cd Adafruit-GPIO-Halt
3. make
4. sudo make install
```

Then edit the rc.local file to start up our code automatically at boot time:

Download file
Copy Code

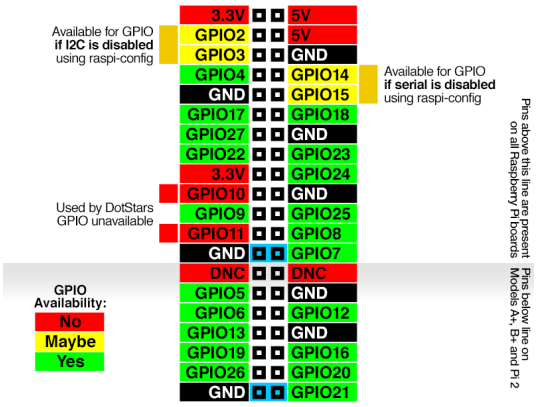
```
1. sudo nano /etc/rc.local
```

Just before final "exit 0" line, insert this line:

Download file
Copy Code

```
1. /usr/local/bin/gpio-halt 21 &
```

Change the "21" to whatever GPIO pin your shutdown button is connected to (the other leg of the button should connect to a ground pin). On Pi models with the **40-pin header, GPIO 21** is at the very end (nearest the USB ports) with an adjacent ground pin, so it's very convenient to connect a button across these two pins. On older Pi models with a **28-pin header, GPIO 7** is similarly situated at the end of the header:



This will take effect on the next reboot. Then, to shutdown, tap the button (or jumper across the pins with a screwdriver if no button installed yet) then wait at least 15 seconds before disconnecting power. Test it once with a monitor connected to confirm that it's working before running the system headless.

Add a Safe Off Switch to Power Down Your Raspberry Pi

Add a switch to your Raspberry Pi to safely shut it down without pulling the power. (A version of this article appeared in Issue 57 of The MagPi.)

Intro

To keep prices down, the Raspberry Pi is missing something that most electronic devices come with: a switch to turn it on or off. That's okay, you say, we'll just pull the plug to turn it off. Unfortunately, this can lead to corruption problems with the SD card. All the instructions say you should run the shutdown command before pulling the plug, but this is not always possible, particularly when your Raspberry Pi is running headless without a connected keyboard and monitor, and possibly even without any network connection. So, what CAN a self-respecting DIY-er do? The answer, of course, is: add your own switch!

Lots of articles are available to tell you how to use a breadboard to connect a button or LED to a Raspberry Pi's GPIO pins. This article focuses on doing something useful with those switches and LEDs.

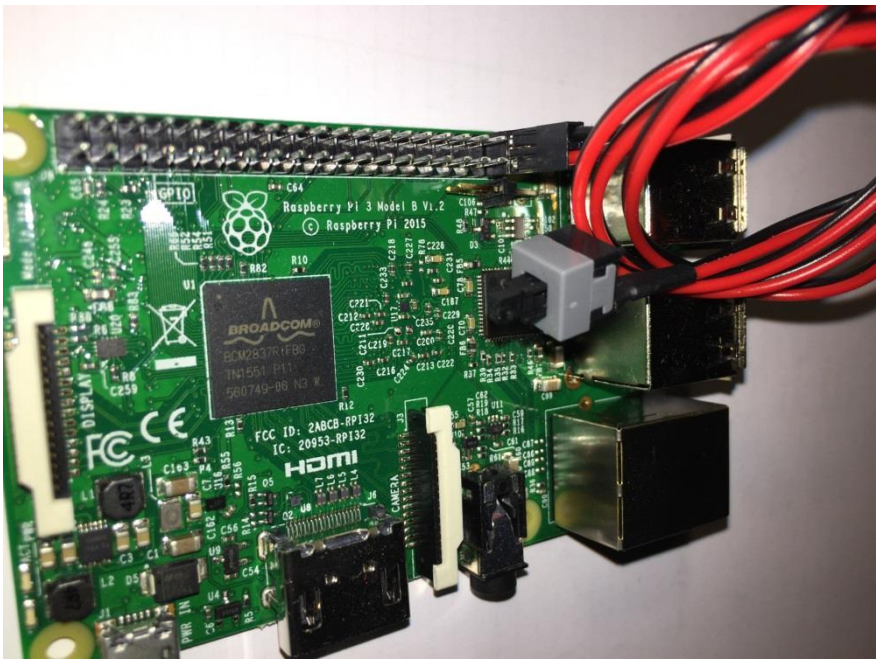
The safe off switch is complementary to a reset switch, which is the best method to start the Raspberry Pi back up again. Issue 42 of TheMagPi had an excellent article (Pi Zero Reset Switch) on how to connect a Reset Button.

You'll Need

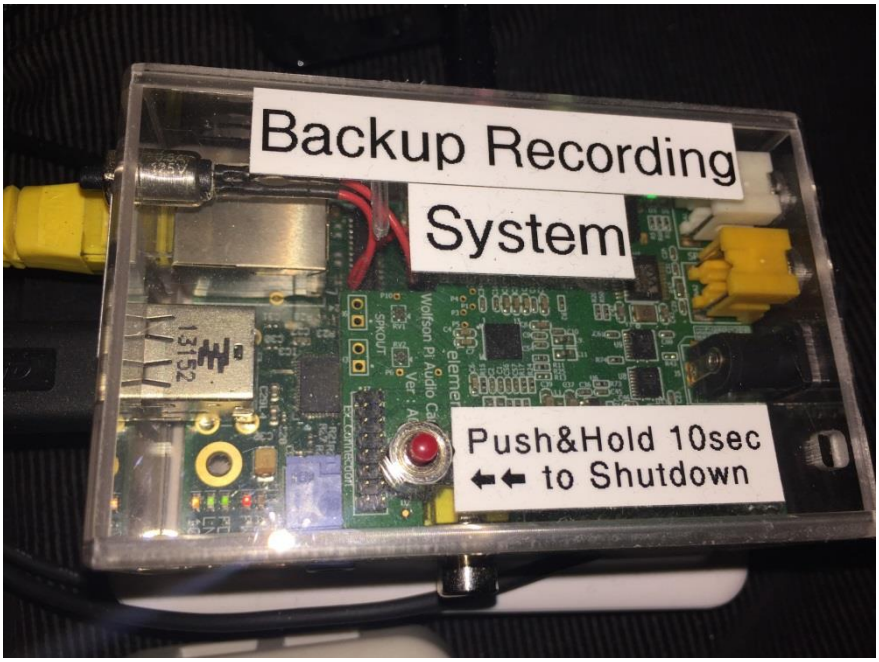
- Raspberry Pi (any model)
- Momentary push button switches, such as these

Pictures

Momentary switch connected to pins 39 and 40

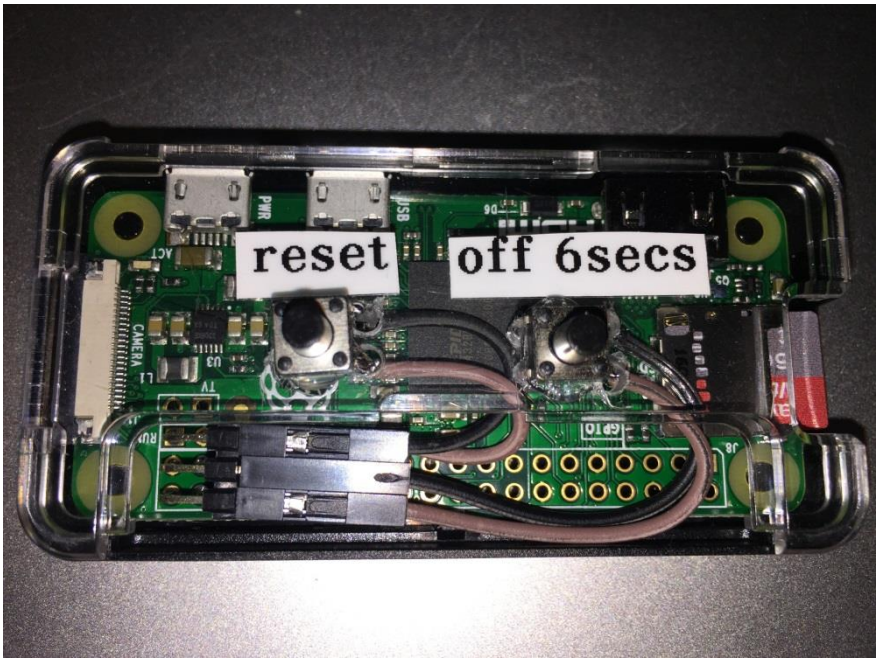


A Safe Off Switch



A reset button on the same system

Example of Both an Off and Reset Switch on a Raspberry Pi Zero. Right angle headers are used for a compact connection. The switches are mounted directly onto an Adafruit case.



Using GPIO Zero

With the GPIO Zero library module, the Python code to deal with a button press becomes extremely simple. Assuming your button is connected between GPIO 21 and Ground, the code can look like as easy as this:

```
#!/usr/bin/env python3
from gpiozero import Button
import os
Button(21).wait_for_press()
os.system("sudo poweroff")
```

This code creates a button on GPIO 21, waits for it to be pressed, and then executes the system command to power down the Raspberry Pi. GPIO 21 is nice because it's on pin 40 of the 40-pin header and sits right next to a ground connection on pin 39. This combination makes it difficult for an off switch to get plugged in incorrectly. On a 26-pin header, GPIO 7 is similarly situated at the bottom there on pin 26, next to a ground connection on pin 25.

Create the script on your Raspberry Pi using your favorite text editor (e.g., nano, vim or emacs), as in

```
$ nano shutdown-press-simple.py
```

Then add a line to the end of /etc/rc.local to run it at boot time:

```
$ sudo su
# echo '~pi/shutdown-press-simple.py &'  >> /etc/rc.local
```

Note: With sone versions of Linux, there may be an "`exit 0`" at the end of `/etc/rc.local`. If yours does, the call to your shutdown scipt will not be done. You might want to verify this by editing the file and adjusting the lines accordingly.

```
$ sudo nano /etc/rc.local
```

Now after rebooting, your script will be running and listening for a button (connected between GPIO 21 on pin 40 and ground) to be pushed.

Preventing Accidental Button Pushes

One major drawback of the previous code is that any accidental push of the button will shut your Raspberry Pi down. It would be better if you needed to hold the button down for several seconds before everything powers down.

shutdown-with-hold.py

```
#!/usr/bin/env python3
from gpiozero import Button
from signal import pause
import os, sys

offGPIO = int(sys.argv[1]) if len(sys.argv) >= 2 else 21
holdTime = int(sys.argv[2]) if len(sys.argv) >= 3 else 6
```

```
# the function called to shut down the RPI
def shutdown():
    os.system("sudo poweroff")

btn = Button(offGPIO, hold_time=holdTime)
btn.when_held = shutdown
pause()     # handle the button presses in the background
```

Instead of hard-coding the GPIO number 21 and the hold time, this code does a few things differently. First, it defines variables to hold these numbers at the top of the code. For a program this small, declaring the values at the top is not necessary, but it is good practice to declare any configurable variables near the top of the code. When making changes later, you won't have to hunt through the code to find these variables. Secondly, it allows the GPIO number and hold time to be overridden on the command line, so that you can change them later without modifying the program. We then define a function named `shutdown()` to execute the `poweroff` system command. The button is also assigned to a variable for use in the next statement. This time, we are also specifying that the button must be held down, and when the hold time (6 seconds) has passed, any function assigned to the `when_held` event will be executed. We then assign that event to the `shutdown()` function we defined earlier. The call to `pause()` is needed to cause the script to wait for the button presses.

If you look at the examples that come with the GPIO Zero source, you'll find a script very similar to this one.

Feedback While Pressing the Button

But, we can do better. The major thing lacking with the above code is any sort of feedback -- it's hard to tell that anything is really happening while you have the button pressed down. Fortunately, GPIO Zero allows us to do much more with a button press, as well as controlling other devices. For example, we can turn an LED on and off, or set it blinking, when the button is first pressed by attaching to the button's `when_pressed` event. We need to ensure that the LED is turned off if the button is not held down for the entire length of time. This can be accomplished by attaching to the when_released event. As before, the important work has been moved into functions named `when_pressed()`, `when_released()` and the same `shutdown()` function we used before. These are assigned to their corresponding button events.

Using the on-board LEDs

Instead of wiring in your own LED, many versions of the Raspberry Pi come with several LEDs already on them that can be controlled. The Raspberry Pi A+, B+ and Pi2 boards have an Activity Status LED and a Power LED that can be accessed through the GPIO numbers 47 and 35, respectively. (Early versions of the Raspberry Pi used GPIO 16 for the Activity Status LED, but did not provide access to the Power LED.) The Raspberry Pi Zero and Computation Modules have the Activity Status LED on GPIO 47. (The GPIO Zero library does not yet have a way to control the LEDs on the Pi3.)

shutdown-led-simple.py

```
#!/usr/bin/env python3
from gpiozero import Button, LED
```

```
from signal import pause
import os, sys

offGPIO = int(sys.argv[1]) if len(sys.argv) >= 2 else 21
holdTime = int(sys.argv[2]) if len(sys.argv) >= 3 else 6
ledGPIO = int(sys.argv[3]) if len(sys.argv) >= 4 else 2

def when_pressed():
    # start blinking with 1/2 second rate
    led.blink(on_time=0.5, off_time=0.5)

def when_released():
    # be sure to turn the LEDs off if we release early
    led.off()

def shutdown():
    os.system("sudo poweroff")

led = LED(ledGPIO)
btn = Button(offGPIO, hold_time=holdTime)
btn.when_held = shutdown
btn.when_pressed = when_pressed
btn.when_released = when_released
pause()
```

The GPIO Zero library will print a warning message if you try using either of these LEDs. The workaround for now is to turn off the warning message temporarily. With current versions of the GPIO Zero library you are using, you can use:

turn-off-power-led-warnings.py

```
import warnings
...
ledGPIO = 47
...
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    led = LED(ledGPIO)
```

If you cannot use the on-board LEDs, you can connect an LED (with a small resistor) to the GPIO of your choice. See numerous other articles on how to connect an LED to a GPIO pin.

Progressive Blinking

To make it more obvious what is happening, it is possible to be more dynamic in your feedback. For example, how about starting with a slow blink, but progressively blink faster and faster? The GPIO Zero makes this easy because it passes information to the event functions that lets you make changes during the button press, and can be set up to call the button press event function repeatedly instead of just once. In this code, we switched to using GPIO Zero's LEDBoard() class to blink multiple LEDs together. (Here, we're passing in the Activity Status and Power LED GPIO numbers.) The constructor for `Button()` has a new parameter (`hold_repeat=True`) and we've set the hold time to 1 second instead of the full hold time. The `when_pressed()` and `when_released()` functions remain the same, but the `shutdown()` function now declares its button parameter, and asks the button how long it's

been pressed so far. The blink rate is then updated accordingly. When the maximum hold time is finally passed, only at that time is the `poweroff` system command executed.

shutdown-ledboard.py

```
#!/usr/bin/env python3
from gpiozero import Button, LEDBoard
from signal import pause
import warnings, os, sys

offGPIO = int(sys.argv[1]) if len(sys.argv) >= 2 else 21
offtime = int(sys.argv[2]) if len(sys.argv) >= 3 else 6
offtime = 6       # shut down after offtime seconds
mintime = 1       # notice switch after mintime seconds
actledGPIO = 47   # activity LED
powerledGPIO = 35 # power LED

def shutdown(b):
    # find how long the button has been held
    p = b.pressed_time
    # blink rate will increase the longer we hold
    # the button down. E.g., at 2 seconds, use 1/4 second rate.
    leds.blink(on_time=0.5/p, off_time=0.5/p)
    if p > offtime:
        os.system("sudo poweroff")

def when_pressed():
    # start blinking with 1/2 second rate
    leds.blink(on_time=0.5, off_time=0.5)

def when_released():
    # be sure to turn the LEDs off if we release early
    leds.off()

with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    leds = LEDBoard(actledGPIO, powerledGPIO)

btn = Button(offGPIO, hold_time=mintime, hold_repeat=True)
btn.when_held = shutdown
btn.when_pressed = when_pressed
btn.when_released = when_released
pause()
```

Playing Sounds?

If you have a speaker connected to your Raspberry Pi, you could play audio clips instead of blinking LEDs. An easy way to do this is to use the `pygame.mixer.Sound` class to play your audio clips. For example, you could repeatedly play the audio clip that says "I'm melting, melting" from the Wizard of Oz, and then play the audio clip that says "There's no place like home" right before powering down the Raspberry Pi.

The basic structure of the code remains the same. At the beginning, we need to initialize the sound system, and then create the sound clips. When the button is pressed, we start playing the "I'm melting" clip, looping it enough times for it to last the hold time. If the button is released early or the hold time has elapsed, we need to stop that clip. When the hold time has elapsed, we then start playing the "There's no place like home" clip and power down.

shutdown-melting.py

```
#!/usr/bin/env python3
from gpiozero import Button
from signal import pause
import pygame.mixer
from pygame.mixer import Sound
import time, os

holdTime = 10
offGPIO = 21

pygame.mixer.init()
melting = Sound("ImMeltingMelting.ogg")
nohome = Sound("NoPlaceLikeHome.ogg")

def shutdown():
    # stop playing one sound and switch to another
    melting.stop()
    nohome.play()
    # give it a chance to play
    time.sleep(1)
    # and shutdown for real
    os.system("sudo poweroff")

def when_pressed():
    # start playing
    melting.play(loops=holdTime/melting.get_length())

def when_released():
    # stop playing if released early
    melting.stop()

btn = Button(offGPIO, hold_time=holdTime)
btn.when_held = shutdown
btn.when_pressed = when_pressed
btn.when_released = when_released
pause()
```

Going Further

Can you think of other ways to provide feedback while pressing the hold button? How about using a buzzer, or popping up a message on a screen? Let your imagination run wild. :)

Can you think of other ways to be signaled that it is time to turn off? How about watching the "low battery" signal from a battery pack? What else can be used to trigger a shutdown?

Now, which of your projects are you going to add shutdown and reset buttons to?

MOS-FET comme interrupteur

http://www.locoduino.org/spip.php?article120

Piloter un moteur

La PWM : Qu'est-ce que c'est ?

- La PWM : Qu'est-ce que c'est ? (1) - Application aux diodes électroluminescentes
- La PWM : Qu'est-ce que c'est ? (2) - Piloter un moteur
- La PWM : Qu'est ce que c'est ? (3) - Changer la fréquence de la PWM
- La PWM : Qu'est ce que c'est ? (4) - Monter en fréquence

Le 2 février 2016. Par : Dominique, Guillaume, Jean-Luc

Dans le premier article concernant la PWM, « La PWM : Qu'est-ce que c'est ? (1) », nous avons abordé l'alimentation d'une DEL par une tension hachée. Nous allons maintenant aborder la commande d'un moteur à courant continu, c'est à dire le type de moteur qui équipe nos locomotives, par une PWM.

La manière de s'y prendre diffère pour plusieurs raisons :

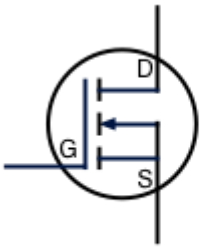
- les moteurs employés en modélisme ferroviaire ont une tension d'alimentation qui est plus élevée que la tension de l'Arduino, généralement 12V ;
- le courant nécessaire est également plus élevé, environ 10 fois, que ce que peut fournir une sortie de l'Arduino ;
- contrairement à une DEL, un moteur est une charge dite *inductive*. C'est à dire que lorsque l'alimentation du moteur est coupée, le courant ne s'arrête pas instantanément de circuler. Il diminue progressivement.

La conséquence directe est qu'**une sortie PWM de l'Arduino ne peut pas commander directement un moteur**. Il est nécessaire d'amplifier le signal, à la fois en tension et en courant. L'amplificateur doit être adaptée au courant consommé par le moteur et doit fournir la tension nécessaire. Elle doit aussi être capable de suivre la fréquence de la PWM.

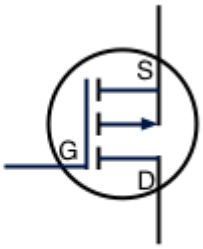
Deux possibilités s'offrent à nous : utiliser un transistor MOSFET de puissance ou bien un pont en H. Nous allons tout d'abord examiner la première solution.

Amplification via un transistor MOSFET

Il existe 2 types de MOSFET, ceux à canal N et ceux à canal P. Les MOSFET sont des composants à 3 broches : le Drain (D), la Source (S) et la Grille (G). Voici le symbole des deux types de MOSFET.



MOSFET canal N

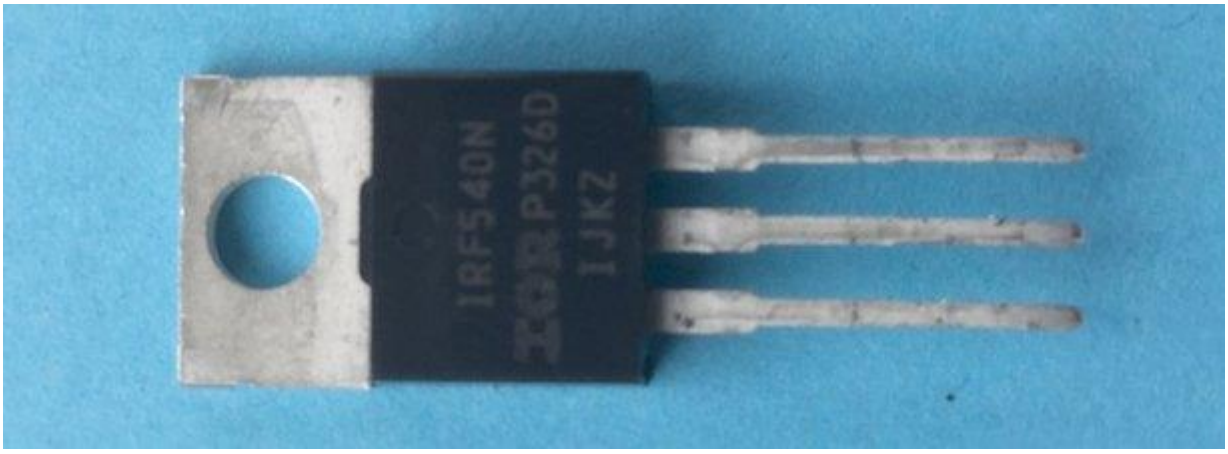


MOSFET canal P

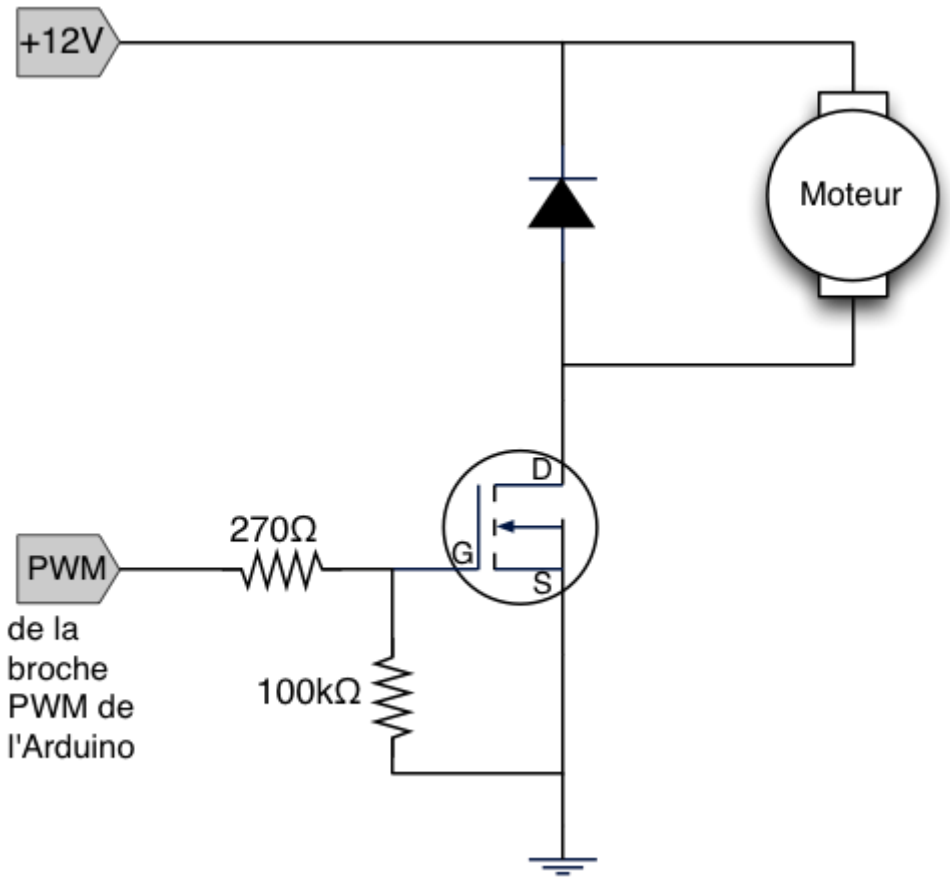
Dans l'emploi que nous allons en faire, on peut voir ces transistors comme des interrupteurs. D et S sont les deux bornes de l'interrupteur et la tension de G détermine la position de l'interrupteur. Contrairement à un interrupteur mécanique, les MOSFET on un sens. Pour un MOSFET canal N, la tension de S doit être plus faible que la tension de D. On connectera donc S côté masse et D côté +12V. Pour un MOSFET canal P, c'est l'inverse. S doit être connecté côté +12V et D côté masse. La tension qui est appliquée entre G et S détermine si l'interrupteur est fermé ou non. Un MOSFET possède une tension de seuil. Pour un MOSFET canal N si la tension entre G et S est plus grande que la tension de seuil, l'interrupteur est fermé (le MOSFET est passant) et le courant circule sinon l'interrupteur est ouvert (le MOSFET est bloqué). Pour un MOSFET canal P, si la tension entre S et G est plus petite que la tension de seuil, l'interrupteur est fermé et le courant circule.

Les transistors MOSFET sont extrêmement sensibles à l'électricité statique. Les manipuler sans précaution peut aboutir à les endommager voire à les détruire. Il est impératif de s'équiper d'un bracelet antistatique qui sera relié à la terre. Les stations de soudage possèdent une prise de terre destinée à recevoir le câble du bracelet antistatique. De manière générale, il est préférable de procéder de même pour les circuits intégrés. Si la plupart des composants à base de transistor MOSFET sont protégés contre l'électricité statique, le risque de casse existe.

Commençons par un MOSFET canal N. Un des modèles les plus répandus est l'IRF540N. Le courant maximum est de 33A et la tension maximum de 100V. très largement suffisant pour nos usages. Sa tension de seuil est comprise entre 2 et 4V. On peut donc en mettant 5V sur la grille fermer l'interrupteur de manière fiable.



On trouve de nombreux exemple de connexion pour l'Arduino, par exemple celui-ci :

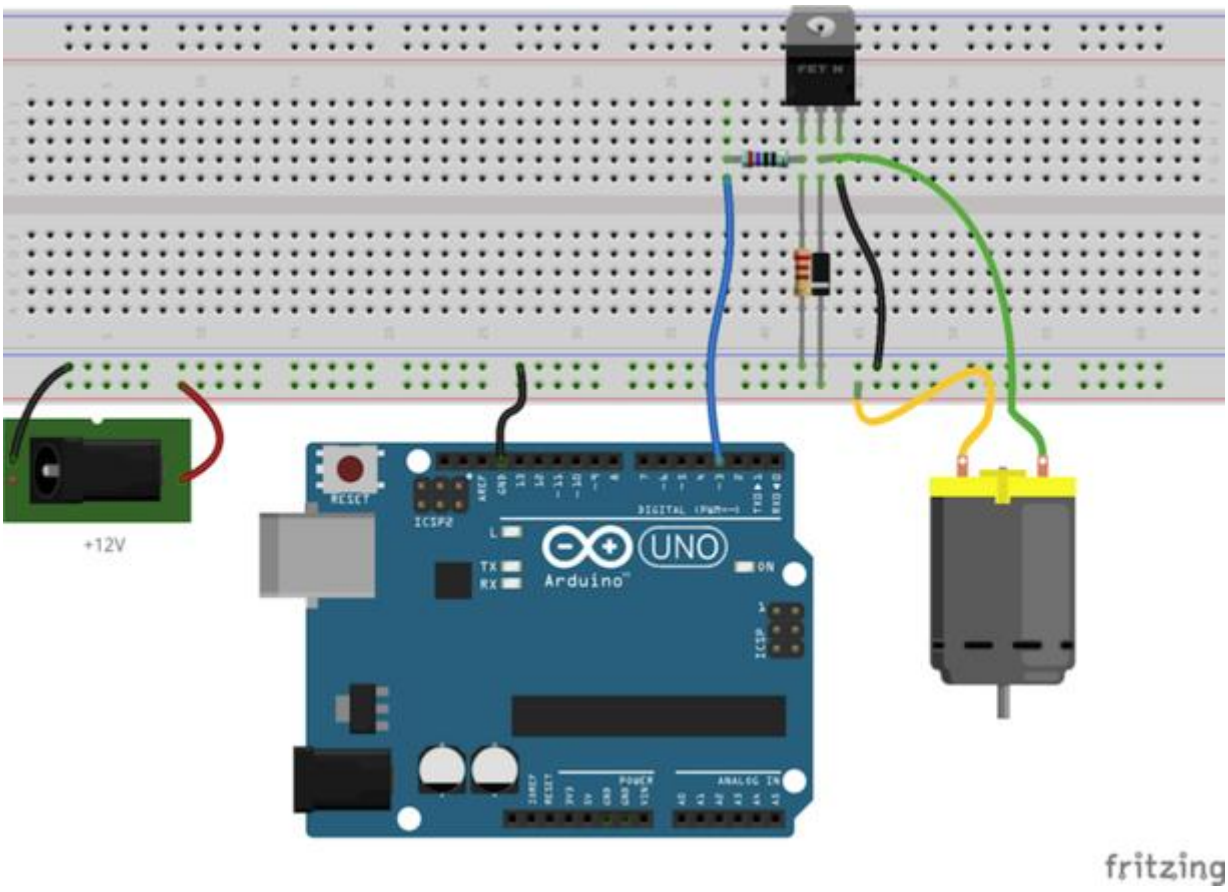


Vous pouvez remarquer la diode en parallèle avec le moteur, il s'agit d'une diode roue libre. Ce type de montage a déjà été présenté dans « Les diodes classiques » pour une commande de relais. Comme cela a été dit plus haut le moteur est une charge inductive et lorsque le MOSFET cesse de conduire le courant ne s'arrête pas instantanément de circuler. Comme il doit aller quelque part, la diode lui offre un chemin. Le courant circule donc en boucle dans le moteur via la diode jusqu'à ce que la résistance du moteur le dissipe sous forme de chaleur. **En l'absence de diode roue libre ce courant provoquerait une augmentation de tension importante sur le Drain du MOSFET avec le risque de l'endommager**.

La résistance de 100kΩ a pour rôle de maintenir le MOSFET bloqué en tirant G à la masse tant que la broche de l'Arduino n'est pas programmée. En son absence G *flotterait* et selon la charge électrique présente le MOSFET pourrait devenir passant avec un démarrage intempestif du moteur.

La résistance de 270Ω évite que trop de courant ne soit tiré de la broche de l'Arduino. En effet, les MOSFET ont sur G l'équivalent d'un condensateur de valeur assez importante. Sans cette résistance le courant serait ponctuellement important et dépasserait les capacités de l'Arduino. Avec 270Ω, le courant instantané ne dépasse pas 19mA.

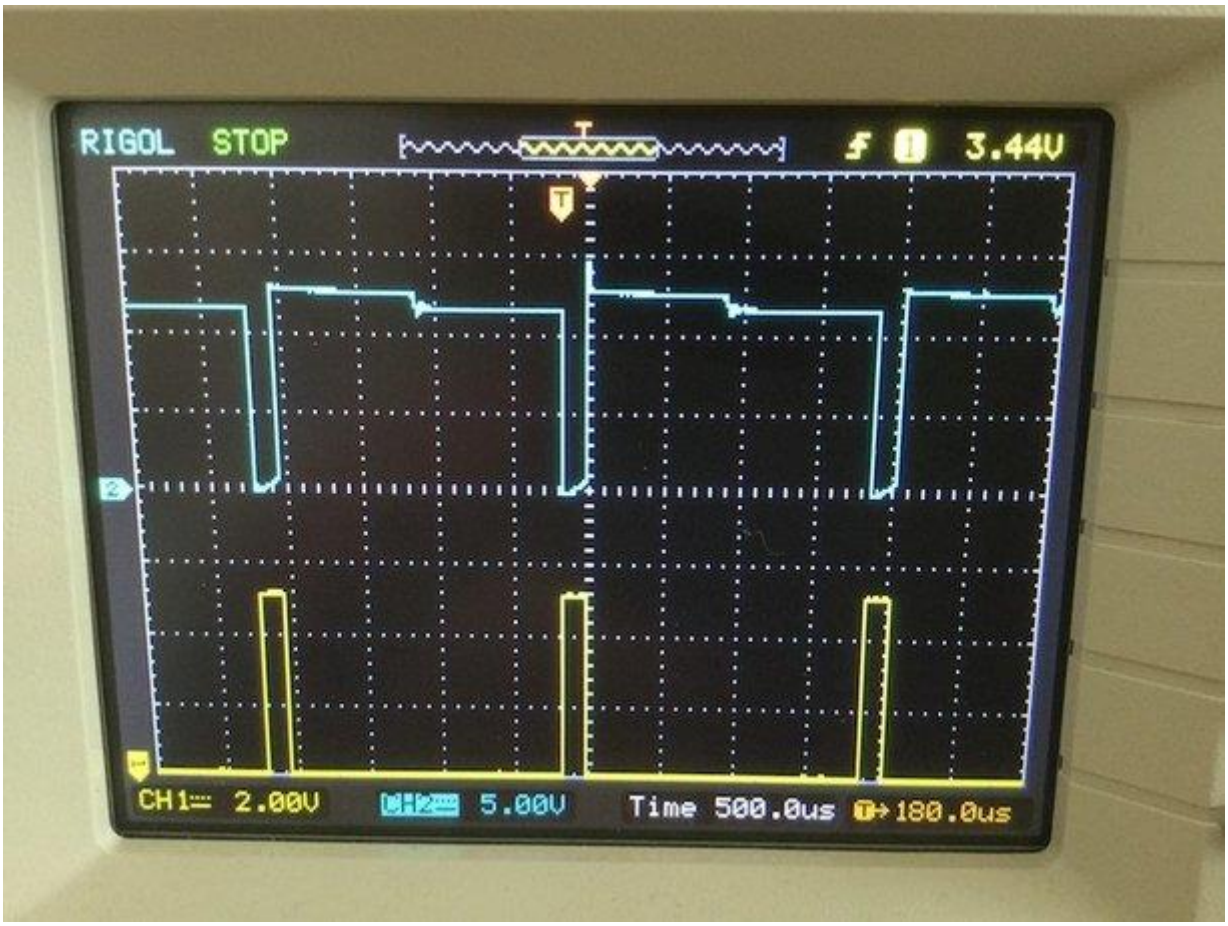
On choisit la broche 3 comme PWM. Le montage sur breadboard est le suivant :



Tout se passe très bien avec la fréquence par défaut de la PWM de l'Arduino, soit 490Hz sur la broche 3. Ici la PWM est réglée à 20, soit un rapport cyclique d'un peu moins de 8%. Le programme de test est réduit à sa plus simple expression.

```
1. void setup()
2. {
3.   analogWrite(3, 20);
4. }
5.
6. void loop()
7. {
8. }
```

Le signal jaune est pris sur la broche 3 de l'Arduino et le signal bleu entre le MOSFET et le moteur, c'est à dire au niveau de drain. Quand le signal jaune est à l'état haut, le MOSFET est passant et le drain est tiré à 0. Le moteur est alimenté. Quand le signal jaune est à l'état bas, le drain flotte et le moteur n'est pas alimenté. Le MOSFET agit donc comme un interrupteur commandé par la PWM de l'Arduino.



Dans un prochain article, nous augmenterons la fréquence de la PWM afin de remédier à deux problèmes :

- **le bruit**. À 490Hz, le moteur émet un Si légèrement faux et assez agaçant ;
- **le problème de dissipation thermique** dans le moteur à cette fréquence. Ce problème est exposé sur mon blog : Tension hachée et pertes par effet Joule. Si vous exploitez en N ou en H0e, ce problème vous concerne et n'est pas à prendre à la légère.

Bistable simple 2 transistors

https://www.astuces-pratiques.fr/electronique/bistable-a-deux-transistors-schema

Un montage bistable permet de conserver un état après un appui sur l'un ou l'autre des boutons poussoirs sur le montage. Le schéma présenté ici comprend deux boutons poussoirs et deux transistors astucieusement montés. Un appui sur un bouton déclenche l'état haut (qui restera après avoir relâché le bouton) et un appui sur l'autre bouton déclenche l'état bas. Les deux états sont "stables" (subsistent sans aucun appui), d'où le nom du montage : le bistable (= deux états stables possibles).

Abordons maintenant le schéma du montage bistable.

Montage bistable : le schéma

Voici le schéma du bistable à deux transistors, basé sur un principe proche du thyristor :

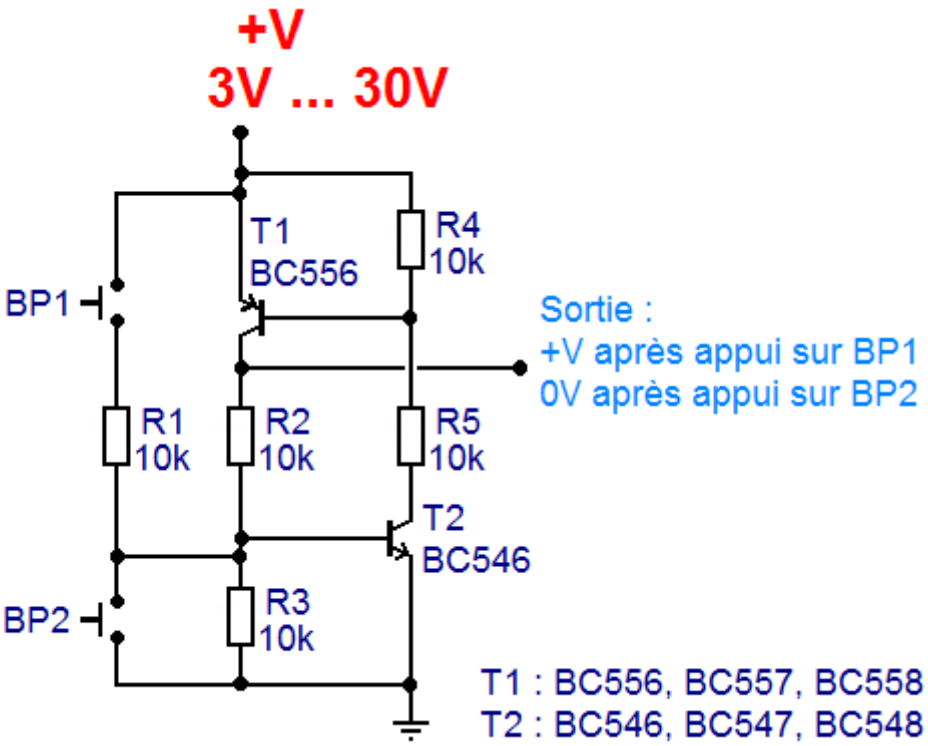
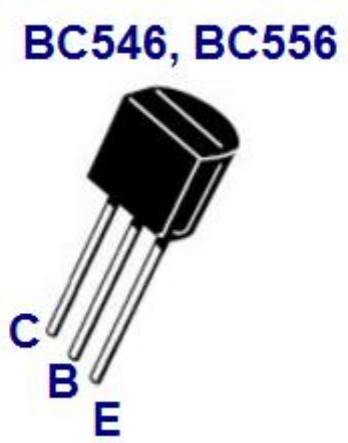


Schéma du bistable à deux transistors (NPN et PNP)

La tension d'alimentation n'a pas grande importance. T1 et T2 peuvent être des transistors assez quelquonques, comme les classiques BC547 et BC557, ou encore 2N3904 et 2N3906, etc.



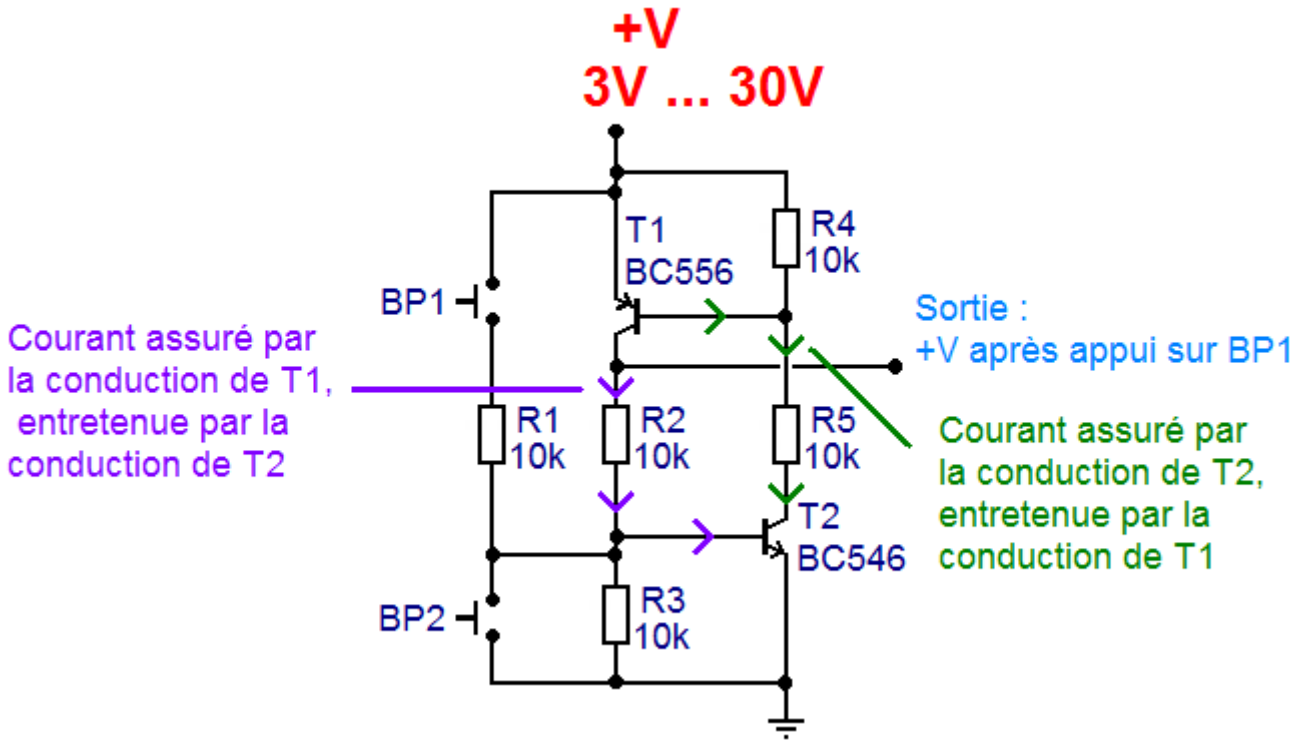
Transistors BC546, BC547 et BC556, BC557

Bistable : principe de fonctionnement

Voyons le fonctionnement de ce montage bistable. Pour comprendre le schéma du bistable, imaginons que les deux transistors (T1 et T2) soient bloqués. T1 et T2 se comportent comme des interrupteurs ouverts. Aucun courant ne circule dans aucune branche du circuit. Cet état subsiste puisque rien ne vient déclencher la conduction de l'un ou l'autre des transistors. La sortie est donc à l'état bas (0V). Si on souhaite le niveau logique (binaire) complémentaire, on prendra le collecteur de T2 qui lui, affiche une tension haute (tension d'alimentation) par rapport à la masse.

Le rôle de R3 et R4, entre base et émetteur, peut être négligé. Ces résistances servent juste à assurer un blocage fiable des transistors.

Imaginons un appui sur BP1, même bref. Lorsque BP1 est appuyé, le transistor T2 est passant : sa base reçoit un courant qui traverse R1. R3 évite un déclenchement intempestif du transistor T2. Comme T2 est passant (saturé vu l'ordre de grandeur des courants de base et de collecteur), du courant circule dans R4 et R5, ainsi que dans la base de T1. T2 forme un quasi court-circuit entre son émetteur et son collecteur. T1 est ainsi passant (lui aussi saturé), ce qui fait qu'il fournit un courant de base pour T2 par l'intermédiaire de R2. Lorsqu'on relâche le bouton poussoir BP1, T1 continue d'être passant grâce à T2 qui continue d'être passant grâce à T1. La sortie est donc au niveau de l'alimentation. Si on souhaite le niveau logique inverse, le collecteur de T2 offre un niveau logique bas (0V). Les deux transistors restent ainsi passants aussi longtemps qu'on ne coupe pas l'alimentation du circuit.



Conduction entretenue réciproquement des deux transistors : état stable du bistable

Imaginons maintenant un nouvel appui sur BP2. BP2 est en parallèle avec la base du transistor T2. Si on appuie sur BP2, on court-circuite la base de T2 et on force ainsi le blocage de T2. Le blocage de T2 coupe le courant dans R5 et ainsi le courant de base de T1. Lorsqu'on relâche l'appui sur BP2, comme T1 est bloqué, plus rien ne permet d'alimenter en courant la base de T2. Le bistable est revenu à l'état décrit au début (T1 et T2 = interrupteurs ouverts). La boucle est bouclée. Et ainsi de suite...

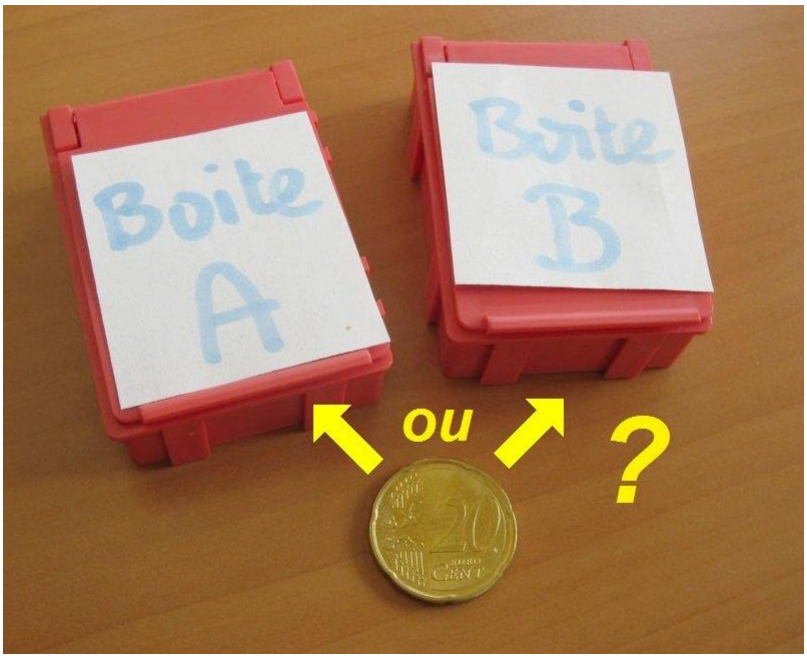
Si on appuie sur les deux boutons poussoirs à la fois, c'est BP2 qui est "prioritaire" puisqu'il force le court-circuit entre la base et l'émetteur de T2, obligeant ainsi son blocage.

Le bistable : une idée, un concept

Le bistable présenté ici du côté de l'électronique est un circuit qui maintient deux états stables possibles qui subsistent après l'action manuelle (ou électronique). Le changement ne peut être réalisé que par une action extérieure (ici, un appui sur bouton poussoir).

Le bistable est la base des mémoires vives en électronique puisque la "donnée" binaire subsiste et peut être consultée (lecture de l'état de sortie) autant de fois qu'on le souhaite.

Imaginons qu'on puisse stocker une pièce de monnaie dans une boite au choix (la boite A ou la boite B).



Bistable : 2 boites et une pièce de monnaie

Une fois placée dans une boite, la pièce de monnaie reste dans sa boite tant que personne n'y touche. L'état du système "boite A, boite B et pièce de monnaie" reste stable dans l'état : "la pièce est dans la boite A". De même si on déplace la pièce dans la boite B, le système restera stable dans l'état : "la pièce est dans la boite B". **Cela illustre le concept du bistable**. A la différence du montage électronique, le bistable à deux boites n'a pas besoin d'alimentation électrique pour conserver sa donnée. C'est une forme de mémoire binaire.

Autres montages bistables

Il existe de nombreux autres montages bistables à transistors. Certains utilisent deux transistors identiques (des NPN par exemples). On trouve encore des montages bistables basés sur des portes logiques, des bascules, des ampli op, des relais etc.

Le montage présenté ici repose sur un transistor NPN et un transistor PNP dont les conductions s'entretiennent une fois qu'elles ont été amorcées. Ce fonctionnement est assez proche du thyristor dont le schéma est proche de deux transistors (un NPN et un PNP) imbriqués l'un à l'autre en terme de jonctions PN.

Alim du pi dans la voiture – extinction

http://www.magdiblog.fr/boa-pi-carjukebox/9-alimentation-du-pi-dans-la-voiture-33-extinction/

http://aws-cf.caradisiac.com/prod/photos/2/8/1/377281/4990718/big-49907186f8.jpg?v=6

http://aws-cf.caradisiac.com/prod/photos/2/8/1/377281/4990717/big-4990717cdc.jpg?v=6

Donc, premier schéma, simplifié, pour bien comprendre le principe :

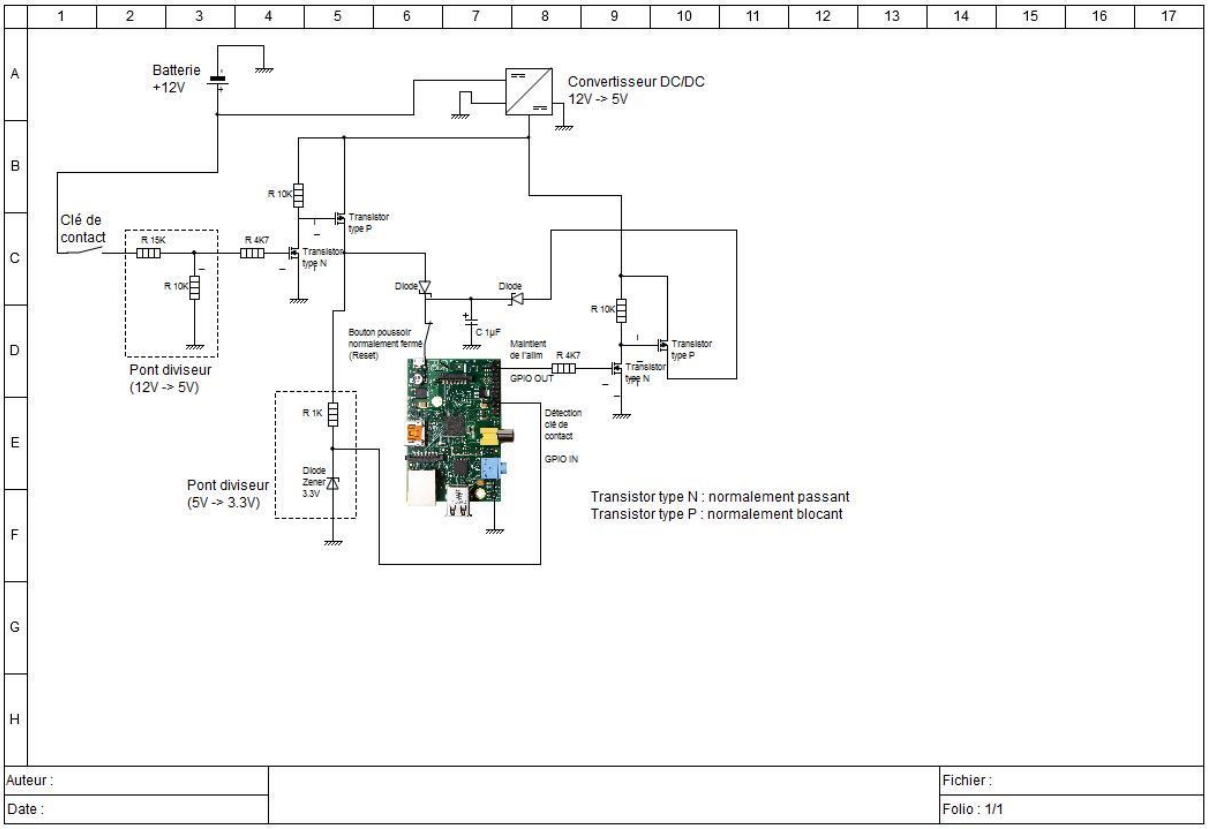
…/…

Le GPIO en mode IN détecte si la clé de contact et bien enclenchée. Lorsque ce n'est plus le cas, son entrée tombe à 0. On envoie alors la commande HALT (après une petite tempo) pour éteindre le PI

Le GPIO en mode OUT a été mis à 1 au démarrage du PI. Il restera dans cet état jusqu'à son extinction. Une fois le PI éteint, le GPIO tombe alors à 0, coupant ainsi l'alim du PI

Mais vous vous en doutez, ce n'est pas si simple. Au moins deux choses ne vont pas : on ne peut pas envoyer du 5V sur une entrée GPIO et on ne sait pas commander directement un relais avec une sortie 3.3V. Il faut donc un peu ruser ….

J'ai donc remplacé les deux relais par 2 couples de transistors qui ont la même fonction. Avantage, ça consomme moins et ça coûte moins cher !
J'ai ajouté des diviseurs de tension pour adapter le voltage au PI.
Voici donc le schéma détaillé, le vrai cette fois-ci !


Batterie
+12V
Convertisseur DC/DC
12V -> 5V
Clé de
contact
R 15K
R 10K
Pont diviseur
(12V -> 5V)
R 4K7
R 10K
Transistor
type P
Transistor
type N
Diode
Diode
C 1µF
Bouton poussoir
normalement fermé
(Reset)
Maintient
de l'alim
GPIO OUT
R 10K
Transistor
type P
R 4K7
Transistor
type N
Détection
clé de
contact
GPIO IN
R 1K
Diode
Zener
3.3V
Pont diviseur
(5V -> 3.3V)
Transistor type N : normalement passant
Transistor type P : normalement blocant
Auteur :
Date :
Fichier :
Folio : 1/1

Pilotage alimentation à l'extinction

https://www.raspberrypi.org/forums/viewtopic.php?f=65&t=141302

mon RPI sera branché sur une batterie de voiture, l'alimentation se fera par le +12V, mais la commande d'alimentation par le +12V "après contact" (APC)
lorsque que je coupe l'APC, l'alimentation +12V du RPI se coupe, logique.
J'ai donc prévu un auto maintien, que le RPI vient couper lui-même lorsqu'il s'éteint. Je m'attendais à ce qu'il tombe son reset (RUN) à 0, mais il apparait que non, le signal RUN reste à 1 (3V3)
existe-il un moyen de commuter ce reset à l'extinction, ou bien de commuter un autre signal ?

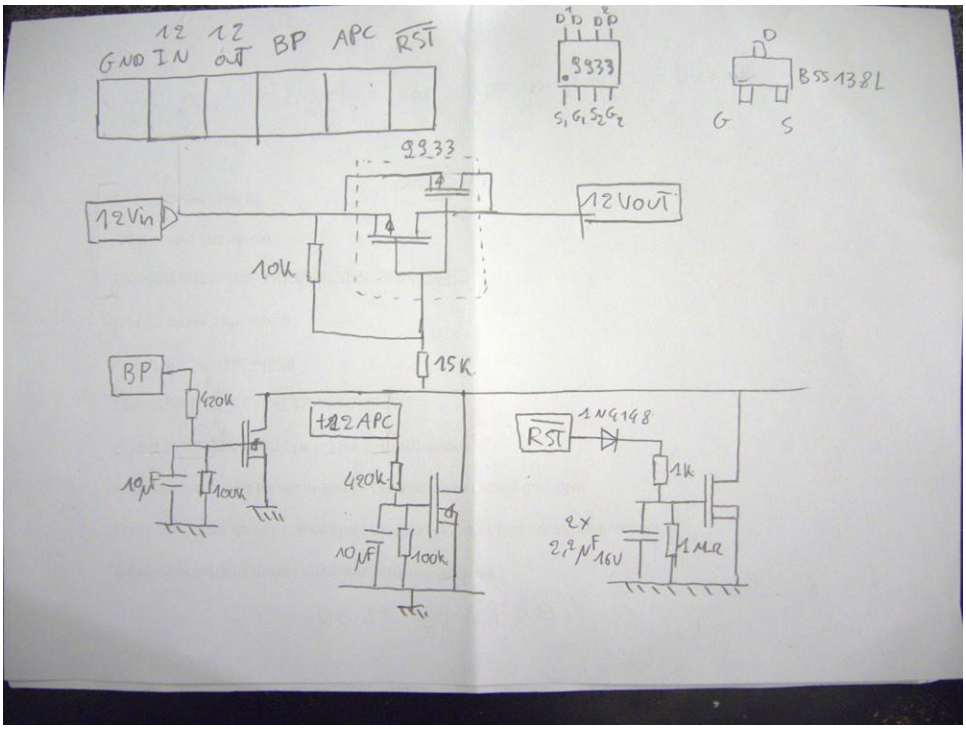
note : j'ai vu passer des choses pour activer le RPI avec un bouton, mais ce n'est pas ce que je veux. Je veux que le RPI coupe son alimentation lorsqu'il s'éteint, l'alimentation reviendra avec l'APC.
je précise que je parle Python, mais très peu Linux.

détails de mon projet : je compte monitorer une voiture de collection avec un ordinateur de bord discret (conso, vitesse, t°, etc) en installant un réseau CAN et des cartes d'IO CAN. J'ai donc une alimentation avec commande pilotée par plusieurs entrées, dont le RPI, l'APC et un bouton pour pouvoir démarrer sans mettre l'APC.

…/…

merci pour ta réponse, c'est exactement ce que je cherchais.
Paradoxalement, cette manip à planté mon RPI, j'ai cru que j'allais pouvoir formater ma carte mémoire qui contient des infos non encore sauvegardées :-/ (je sais c'est pas bien)
impossible de redémarrer avec le fichier dt-blob.bin généré, mais par contre la pate que je voulais lever passait bien à 1 dés la mise sous tension !
Mais entre lignes se trouvait une solution alternative que je choisis pour le moment, et au pire si ça me pose problème j'y reviendrai plus tard : utiliser le TXD, qui monte à 1 dés que le RPI démarre, et tombe à 0 quand il fait un shutdown.

Voici le schéma de mon gestionnaire d'alim HARD, je n'ai pas de scanner, j'ai fait un photo à l'ancienne

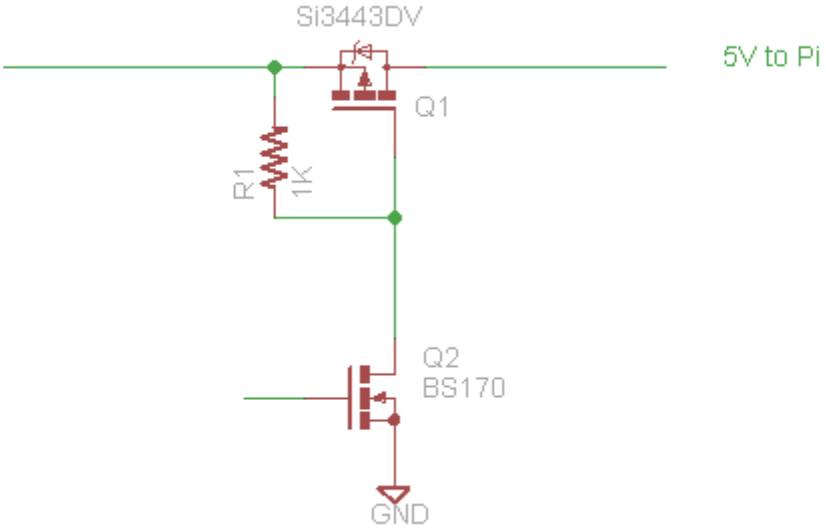


LE 9933 est un double MOS dans un boitier SO8. j'ai mis les 2 MOS en //. Il est donné pour 20V de VGS et 3,5A environ par MOS. j'ose espérer que sans plage de dissipation, il tienne suffisamment pour mon usage (il doit alimenter aussi les modules d'IO au bout du bus CAN)
le BSS138 est un petit MOS en SOT23 qui tient 50V, je ne tire pas de courant dessus.
l'entrée /RST a une diode pour éviter le retours sur la commutation de la pate de commande, en l'occurrence TX.
le 2x 2,2µF x 1MΩ font une constante de temps de 4,4 secondes, ce qui me fait un retard à l'extinction de 5 secondes environ.
j'ai oublié de dessiner la diode de protection SMT 20V en entrée pour limiter les surtensions, fréquentes sur les (vieilles ?) voitures

Infos à extraire !

https://www.raspberrypi.org/forums/viewtopic.php?f=37&t=102015

Power-Switch
We need a power switch to disconnect the Pi from the power. Real men use relays, but were going to use an electronic switch using a P-channel MOSFET, driven by an N-channel MOSFET.



Power Switch

+++ Overlay for on/off push-button controller.

https://www.raspberrypi.org/forums/viewtopic.php?f=107&t=113789

https://www.raspberrypi.org/forums/viewtopic.php?f=41&t=114975

Pas simple, mais excellent !

https://www.raspberrypi.org/forums/viewtopic.php?f=65&t=141302