Part 60
-
Power Off Switch

Version: 2020-02-23

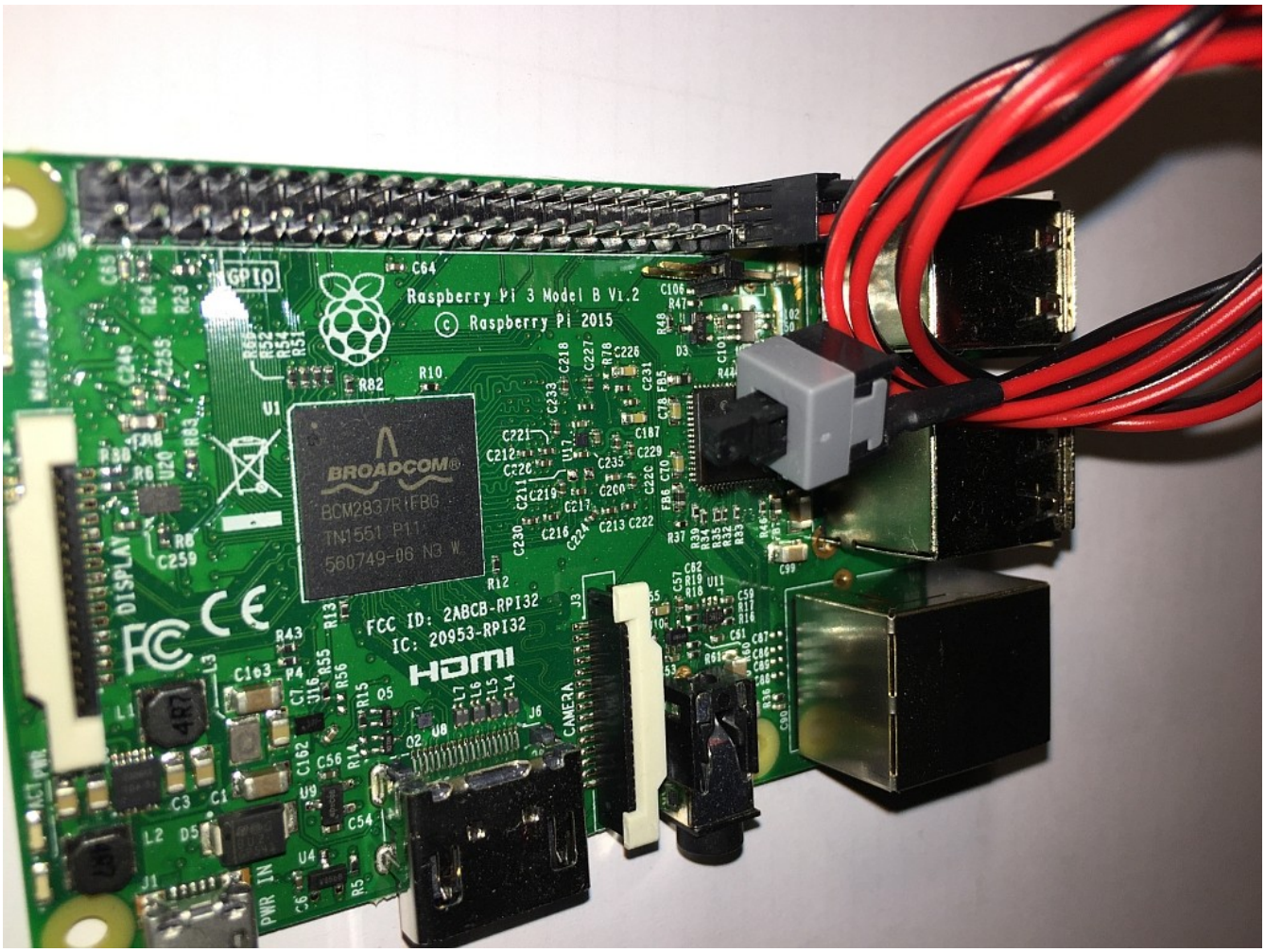
Add a Safe Off Switch to Power Down Your Raspberry Pi

Intro

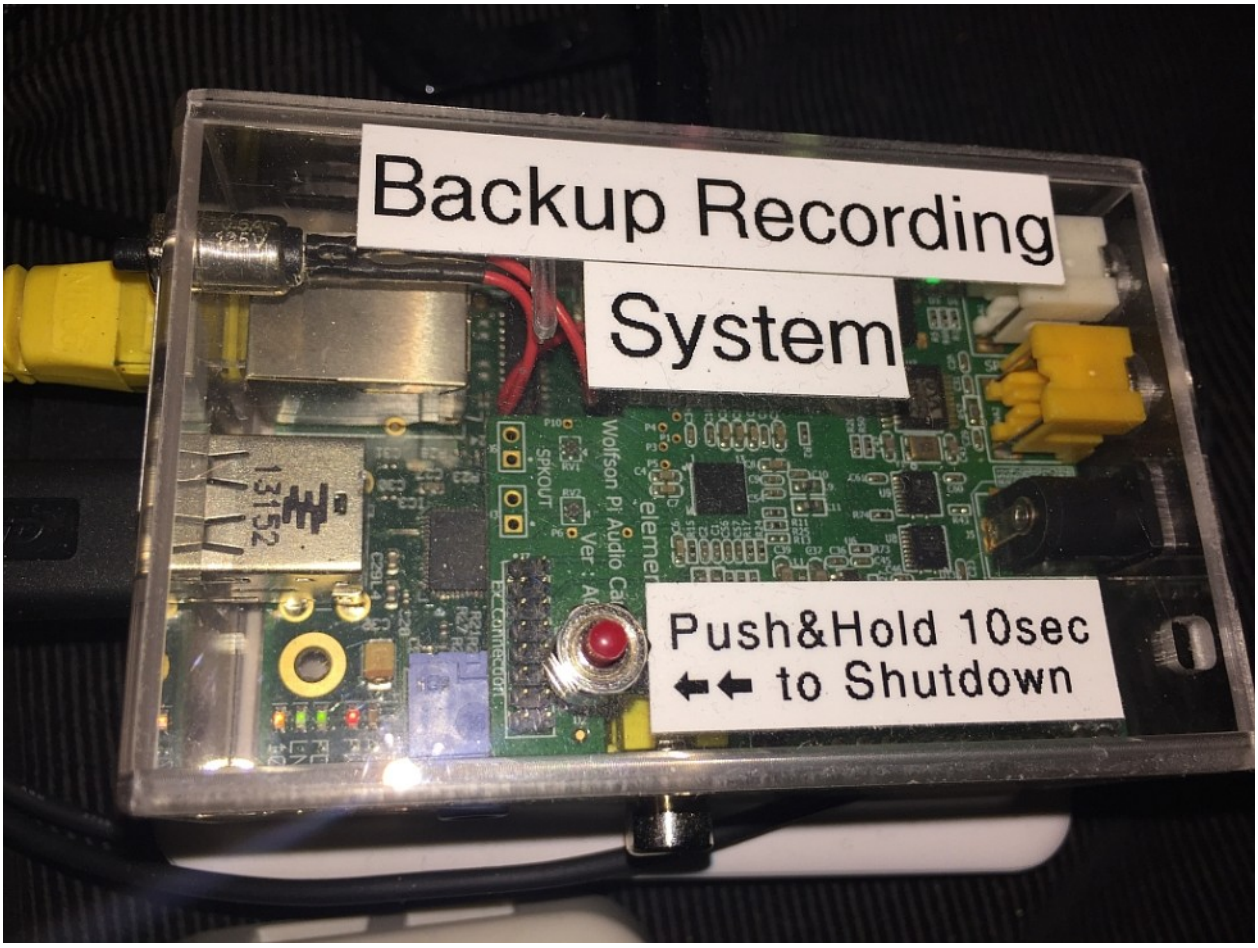
To keep prices down, the Raspberry Pi is missing something that most electronic devices come with: a switch to turn it on or off. That's okay, you say, we'll just pull the plug to turn it off. Unfortunately, this can lead to corruption problems with the SD card. All the instructions say you should run 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.

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.

Reset Buttons
The safe off switch is complementary to a reset switch, Issue 52 of TheMagPi had an excellent article on how to connect a Reset Button.
A reset button has several uses. The primary purpose is to start the Raspberry Pi up again from a powered-off state.
It is preferred over pulling the power connector out and putting it back in.

Depending on which connector is being used for the reset switch, the reset button can ALSO be pressed while the system is running and the system will restart immediately. However, this is fraught with peril -- your SD card can become corrupted or damaged. DON'T DO THIS.
Instead, use the safe off switch described in the rest of this article.

The connector to use for the reset switch will vary between Raspberry Pi models. Most models have a connector that is marked as RUN on board. Connecting that connector to ground will usually perform the reset action.

On the Raspberry Pi 4, you need to use the connector marked GLOBAL_EN instead of RUN. You most likely will need to solder on a pair of headers if you don't want to solder your button directly to the connector.

On Raspberry Pis prior to the Raspberry Pi 4, you can also use GPIO 3 (pin 5) as a reset connector, but ONLY from the powered-off state. Unfortunately this also means losing your I2C connectivity on those models.

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 :

```
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.

If you don't mind losing your I2C connectivity, an alternative choice would be GPIO 3, situated on pin 5, right next to the ground connection on pin 6. What is particularly nice about GPIO 3 is that it **also** acts as a reset pin when the computer is powered down. By using GPIO 3, you can use a single button for **both** an ON and OFF switch. Create the script on your Raspberry Pi using your favorite text editor (e.g., nano, vim or emacs), and make certain that it's executable, as in

```
$ nano shutdown-press-simple.py
$ chmod a+x shutdown-press-simple.py
```

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

```
~pi/shutdown-press-simple.py &
```

If your /etc/rc.local ends with an `exit` statement, make certain that the new line is before the exit statement. Use your favorite editor to add the line.

```
$ 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.

```
#!/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.

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.

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.)

```
#!/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:

```
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 `
` system command executed.

```
#!/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
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", 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" and power down.

```
#!/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?