PiZ-UpTime PLUS

Technical details and User Guide

PiZ-UpTime PLUS interacts with the Raspberry Pi on Raspbian OS via a Python Program. This program can be downloaded from the alchemypower.com web site. This program is supported on a Raspberry Pi 3 or Pi 4 or alternate Raspberry Pi with a 40 in header, running on 32-bit or 64-bit Raspbian OS (always tested with latest version of Raspbian OS). The interaction between Raspbian and PiZ-UpTime PLUS is via the I^2C bus and uses the default I^2C address 0x32 (decimal 50).

Parameters from the power-bus, battery, and the Battery Management System (BMS) are collected by the microcontroller and reported back to the Raspberry Pi via I^2C. This is shown in Figure 1 below.

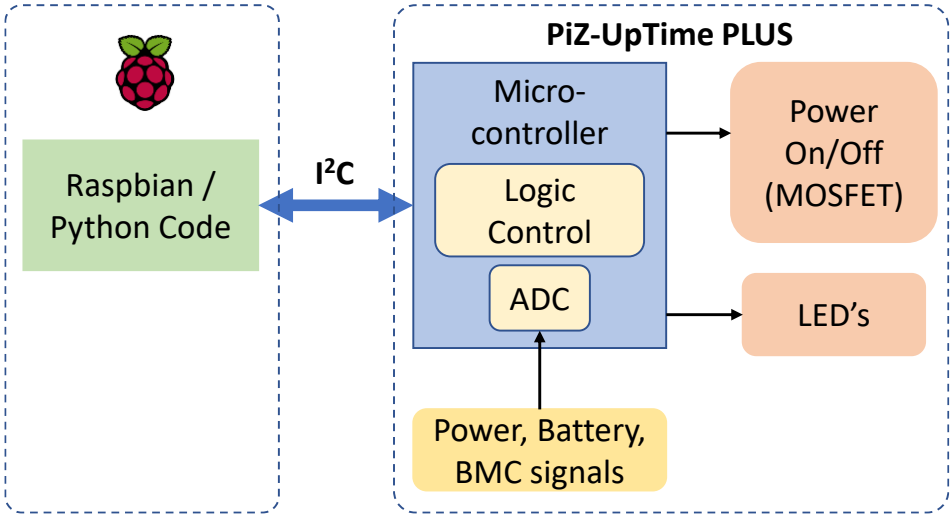

***Figure 1** – Interaction of Raspberry Pi and PiZ-UpTime PLUS*

The Microcontroller (a PSoC chip from Cypress Semiconductor, now Infineon) monitors the necessary signals and controls the logic to turn things on/off. For example, the PSoC controller turns on/off the shutdown LED (blue LED) and Power on/off MOSFET on PiZ-UpTime PLUS board.

Since there are a lot of system parameters to track and monitor, variables are maintained in the Python code and the values are exchanged with the PSoC controller. The PSoC controller also measures the analog values of Voltage, current etc. and converts them to digital values which are reported back and displayed by the Python code.

Running the Python Code

The Python code provided can be run at boot up time by inserting a line in /etc/rc.local file on Raspbian. All print statements should be commented out from the sample code. It is important to run the code as sudo in /etc/rc.local as well as to run the code in the background. Before enabling the code via /etc/rc.local, please ensure to check if the I^2C address is visible and the interaction with the controller is enabled. This can be done using the "i2cdetect -y 1" on the Raspbian OS.

An example of how to start the code at every boot up is shown below.

Edit /etc/rc.local file and insert the lines below at the bottom of the file. The location of the file plus.rc.local.py is in the home directory of account "pi" at /home/pi/ - if a different location is used, please edit the path to reflect that.

```
# For Pi-UpTime PLUS
#
sudo python /home/pi/plus/plus.rc.local.py &
#
```

NOTE: The I^2C may not be enabled at boot up time. It is possible that the code may not be enabled or running via the rc.local statement. If it does not run, some other method should be used to start the process after I^2C processes have started.

The Python code plus.rc.local.py, running at bootup time, disables all print statements. If a log is needed, please use the uptime-plus.py file and redirect stdout (output) to a file. An example of doing that is below. We assume the OS is the latest version of Raspbian OS and the login runs the bash interface as a default.

```
Raspbian OS bash> python uptime-plus.py >& log-file &
Raspbian OS bash> tail -f log-file
```

The code can also be run interactively to monitor system variables.

Please be aware that two instances of the code running (one in the background initiated from /etc/rc.local file and one interactively) may cause some readings to return erroneous results. These erroneous results can be ignored.

The details

The Python code:

1. Initializes the variables to set up the monitoring environment and instructing PiZ-UpTime PLUS as to operating conditions desired.
2. Start monitoring the system in an endless loop, sleeping between each reading.

Each section is explored in mode details below.

Step 1: Initialize the variables

There are several variables which are initialized and depends on the operating conditions desired by the user.

i) Timer: how long should the UPS power remain on for shutdown to complete.

Variable name: timer. Located appx. line 102 in the sample script.
Default value: 30 seconds
Maximum value: Max theoretical value is 65535 seconds or ($2^{16} - 1$) seconds. Limited by MAX_TIMER constant defined in the script to be 480 seconds (8 minutes).
Minimum value:10 seconds.

When a shutdown command is issued, the Raspbian OS needs between 10 seconds to several minutes before the shutdown is complete the shutdown properly. During this time several processes are shut down and any files used by the system are closed as well. Networked files and files on the cloud may take longer to process, sync and close properly. Depending on the process, the time needed for the sequence of events may vary anywhere from a few seconds to a few minutes.
The "timer" variable in the Python code can be from 0 to 65535 ($2^{16} - 1$) seconds. This is approximately 18 hours and is too long. 255 (8 bit timer variable) or appx 4 minutes may be too short for some Raspberry Pi systems. The maximum value is controlled by MAX_TIMER variable in the Python code. The value of MAX_TIMER is set to 480 or 8 minutes, which is plenty of time for most Raspbian OS systems to shut down.
Please make sure the UPS can operate for the duration the shutdown needs the power i.e., the battery has enough capacity (mAh) to power the Pi for that duration.
Other variables which may need to be adjusted to make PiZ-UpTime PLUS operate properly is battery_min (minimum battery V at which shutdown is triggered).
When the shutdown sequence is started, the blue LED blinks for "timer" seconds, on for one second, off for one second. The last five seconds, the LED blinks faster, indicating only 5 seconds remain. The fast blink is when the LED is on for ½ second and off for ½ second.

ii) Don't care about battery level i.e., start shutdown as soon as power fails.

Variable name: Battery_dont_care. Located appx line 112 in the script.
Default value: 0 i.e., run the UPS till the battery runs low. Do not shut down on power failure. A value > 0, will cause the shutdown command to be issued once the power failure is detected. Power to the Pi is provided for "timer" seconds so that the shutdown can complete successfully.

Certain situations require the Pi to be shutdown as soon as input power fails. An example of such a use-case is when Pi's are used in automobiles for infotainment. When

the automobile is turned off, the Pi should shut down. When the automobile starts, the Pi should reboot. The battery is used to provide power to the Pi so that the shutdown is completed properly.
To use PiZ-UpTime PLUS in this mode, set the value of the "Battery_dont_care" variable to "1". If the value is set to "0", PiZ-UpTime PLUS is used as a UPS, till the battery is drained or till power returns.
Other variables which may need to be adjusted as well:

- Power_on_TO: how long to wait when power returns before power is supplied to the Pi. For the use case of the automobiles described above, the delay could be useful to ensure the automobile electronics stabilize before power to the Pi is applied.
- V_battery_min: Define the battery V at which the shutdown is triggered. This is the lowest value of the battery V at which all systems should be shutdown. This value will vary based on the load (external peripherals) on the Pi. Without any external peripherals, it is recommended to keep this value to 3.1 V (31 in the script). At appx 2.8V most Lithium Ion batteries shut down and the external V is cut off.

iii) Battery Minimum: How low can the battery go to before a shutdown is initiated.

Variable name: V_battery_min. Located appx line 89 of the sample script.
Default value: 31 (or 3.1V)
Minimum Value: 29 (or 2.9V)
Maximum Value: 42 (or 4.2V)

This value is a critical value for proper operation of the system. Most Lithium-Ion batteries operate from 4.1V to 2.8 V. Each battery is different and the operating paraments are set by the manufacturer. It is not a good idea to drain the battery below 2.8V as it can permenantely damage the Li-Ion chemistry.
The battery power at the low end of the operating range is needed to ensure the Pi remains on for the duration of the shutdown process. For example, if high current devices (e.g. SSD drives or disk drives) are used with the Pi, the current demand is higher and the battery will drain very fast at the low end of the operating range. For those situations, it may be advisable to initiate the shutdown when the battery power is 3.2V or even 3.3V. You may have to experiment with this value for your operating environment.

iv) Power-on Time Out: After power comes back, how long should the system wait before power is provided to the Pi.

Variable name: Power_on_TO. Located appx. line 140 in the sample script.
Default value: 0 i.e., power on the Pi once the power is on.

Minimum Value: 0 seconds i.e., don't wait.
Maximum Value: 255 seconds (4 minutes, 15 seconds), defined by MAX_TO constant in the Python script.

Usually when the power fails and comes back on, the power may cycle on and off a few times before it stabilizes.
Another possible condition where this capability becomes critical is when motors or inductive load / devices are powered on creating a current spike on the electrical system when power returns. A few minutes delay would stabilize the power.
Power on Time out variable blocks power going to the Pi. The Pi remains off for that duration. Battery charging continues as Power is on.
Should power fail during the count down, the value of the counter is reset. For example, if Power_on_TO variable is set to 30, PiZ-UpTime PLUS will wait for 30 seconds before power is supplied to the Pi. If say in 25 seconds the power fails again, the system will reset and wait 30 more seconds before power is supplied to the Pi. In this situation, PiZ-UpTime PLUS will wait for 55 seconds before the Pi receives the power and reboots.
During the Power Time Out, the blue LED blink on for two seconds and off for 2 seconds. After the timeout is completed, the LED turns off.

NOTE: When this mode is used with Battery minimum value (described in iii above), the blue LED will blink for the duration set, when power returns while the UPS continues to operate. This is normal expected behavior. It is a reminder that there is a delay to restart when power returns.

v) I^2C address: Change the I^2C address.

Variable name: address. Default value is 0x32 (decimal 50). Located appx. line 61 in the sample script.
Default value variable: DEFAULT_ADDRESS. Located appx. line 37 in the sample script.

NOTE: change the I^2C address value with extreme caution. Please test the capabilities before deploying the system, especially if used with /etc/rc.local to start up automatically.

The default address used for I^2C communications is 0x32 (or decimal 50). Sometimes, this address is in use by another device on the Pi and cannot be changed on the other device. PiZ-UpTime PLUS offers the capability to change the I^2C address. Please use this capability with caution. It is recommended to test this capability before automating the program start capability. In many situations, the I^2C address change does not go into effect till a few seconds later. It is recommended to pause (sleep) while the address change goes into effect.

Step 2: Monitor the system

Once all the variables are set, it is time to monitor the system. The system uses a built in ADC to monitor the operating parameters such as Input Voltage, Output Voltage, Battery Voltage etc.

A block of 31 bytes is read in at a time using I^2C. The block includes the variables set (described above) as well as values updated by the system. The bytes read in; are parsed; and converted into human readable format. Once these values are available, the values are printed, or a decision is made whether to shut down the operating system.

Temperature monitoring is for informational purposes only. The temperature monitoring and whether the battery should be charged, or the charging be disabled (or suspended) is made in hardware by the Battery Management System (BMS). The code can be supplemented by adding an alert via email or a text message.

i) How often to gather the values i.e., loop around in the infinite loop:

Variable name: zeit – German for time. Located appx. on line 191 in the sample code.
Default value: 30 seconds

This loop is repeated "zeit" seconds. The variable "zeit" can be changed in the code by the user.

ii) Has shutdown started?

Variable name: shutdown_start. Located appx. on line 157 in the script.
Default value: 0 i.e., shutdown has not started. A value of "1" (> 0) indicates shutdown has started.

It is important for PiZ-UpTime PLUS to know when system shutdown has started. If the system has shutdown and the battery runs low, the Power to the Pi can be disabled. When the battery runs dangerously low, PiZ-UpTime PLUS can indicate to the Pi that shutdown should begin. Power to the Pi will be maintained for "timer" seconds for the shutdown to complete properly.

Note: if the battery falls below 2.8V, the output V from the battery falls below the threshold value for the electronics to function properly. In this situation, the power to the Pi cannot be sustained for the shutdown to complete properly. So please make sure that the battery V is set to the level that the whole system can operate properly. This is done by setting the value of "V_batt_min" described earlier.

iii) I^2C block size.

Variable name: lange. Located appx. on line 55 in the script.
Default value: 31 i.e., read 32 bytes of data on one read command.

To make the communications efficient, a block of 32 bytes is read from the microcontroller on the PiZ-UpTime PLUS in one read command. The layout of the variable is described below. Note that some of the variables define the operating parameters. The other variables hold the operating values used by PiZ-UpTime PLUS.

I²C Block and Variables

The variables used by PiZ-UpTime PLUS is read in as a block of 32 bytes and is shown below in Figure 2. The cells shown in green are used to initialize the variables used to set the operating parameters discussed above. These variables are read-write variables. The other variables are read-only variables. I²C will generate an error if an attempt is made to write to those variables.

Decimal	Hex	Variable		Decimal	Hex	Variable
0	0x00	Wakeup *(future use)*		14	0x0E	Output V - High Byte
1	0x01	Shutdown start (0=no, 1=yes)		15	0x0F	Output V - Low Byte
2	0x02	Timer - High Byte (seconds)		16	0x10	NTC V - High Byte
3	0x03	Timer - Low Byte (seconds)		17	0x11	NTC V - Low Byte
4	0x04	Battery Low V (Value * 10)		18	0x12	Battery Current - High Byte
5	0x05	I²C address		19	0x13	Battery Current - Low Byte
6	0x06	UPS/batt ignore - shutdown on power fail (0=no, 1=yes)		20	0x14	System Current - High Byte
				21	0x15	System Current - Low Byte
7	0x07	Power on Time-Out (seconds)		22	0x16	*Future Use*
8	0x08	*Future Use*		23	0x17	*Future Use*
9	0x09	*Future Use*		24	0x18	*Future Use*
10	0x0A	Input V - High Byte		25	0x19	*Future Use*
11	0x0B	Input V - Low Byte		26	0x1A	*Future Use*
12	0x0C	Battery V - High Byte		*till*		
13	0x0D	Battery V - Low Byte		31	0x1F	*Future Use*

***Figure 2** – Variables exchanged between the Raspberry Pi and PiZ-UpTime PLUS over I²C. Green colored cells are used for initialization of the system operation.*

Variables used in the Python sample code, which sets these variables are discussed earlier in this document.

Legal Stuff

The sample script is provided as a sample to you. It has not been tested in your environment nor has it been tested for robustness in your environment or other operating conditions.

Feel free to modify, use, distribute etc. the code. No implicit or explicit warranties are offered or implied with the sample code. We don't bear any responsibilities for the code operating properly. Please use the code with caution.