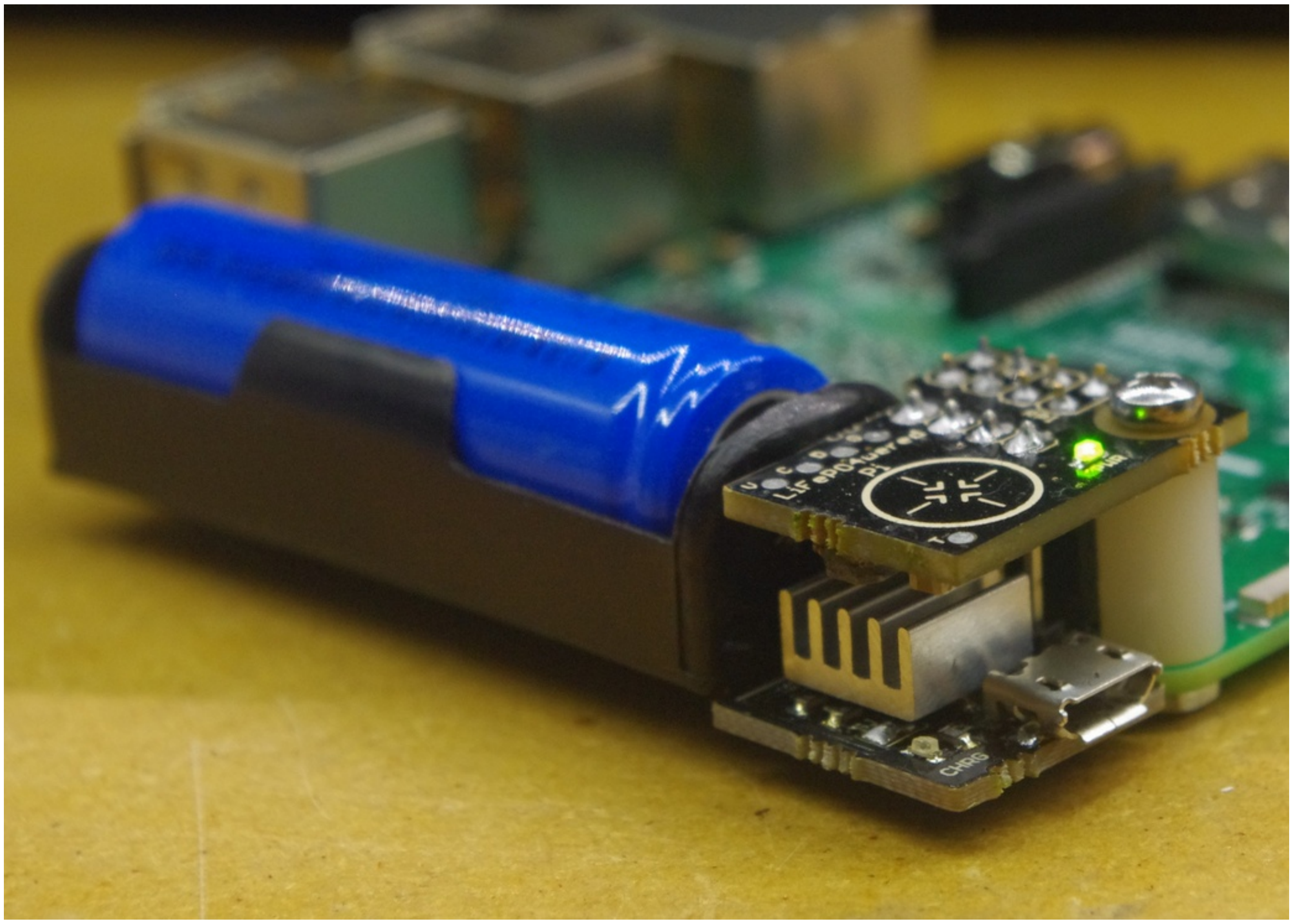


Introduction

The LiFePO$_4$wered/Pi™ is a high performance battery power system for the Raspberry Pi. It can power a Raspberry Pi from 20 minutes to 3 hours from the battery (depending on Raspberry Pi model, attached peripherals and system load) and can be left plugged in continuously. It features:

- A single 3.2 V, 550 mAh LiFePO$_4$ (lithium iron phosphate) cell, providing high power density, extended cycle life (2000+ cycles), and safety from fire and explosions.
- A smart USB charge controller with over-charge protection, allowing the device to stay plugged in and provide UPS (Uninterruptible Power Supply) functionality for low load systems based on the Model A+, Model B+ and Pi Zero.
- Two-way communication between the power system and the Raspberry Pi over I^{2}C bus (to

monitor voltages and customize settings) and Raspberry Pi shutdown detection.

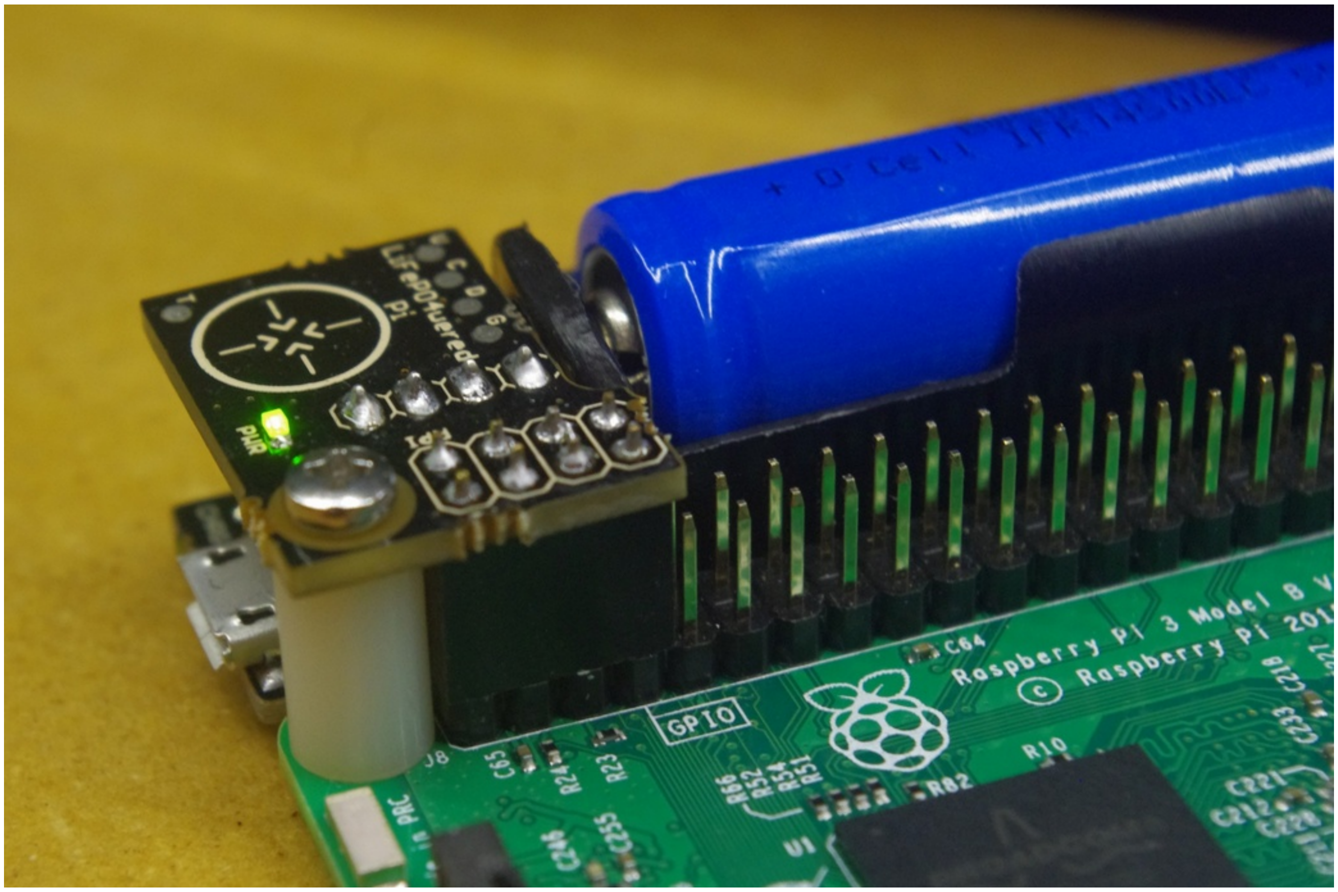
- A smart power manager controller and open source daemon that work in tandem to provide clean shutdown functionality and over-discharge protection.
- Continuous measurement of battery voltage and load voltage with user programmable thresholds for boot, shutdown and hard power down.
- A touch pad with programmable parameters that provides clean boot/shutdown capability even in headless setups.
- Press-and-hold touch pad for protection against accidental activation and with the ability to also monitor the touch pad in user software.
- A green PWR LED that indicates the Raspberry Pi power state, gives user feedback and can even be controlled by user code.
- A separate red CHRG LED that indicates charging state.
- A wake timer allowing the Raspberry Pi to be off most of the time for low duty cycle applications.
- An auto-boot feature to maximize uptime by making the Raspberry Pi run whenever there is sufficient battery power.
- A flexible watchdog timer that can alert a user by flashing the PWR LED or even trigger a shutdown if the user application fails to service the timer within a configurable amount of time. In combination with the auto-boot feature, this can greatly improve reliability by allowing the system to reboot to a known state if the user application crashes or becomes unresponsive.
- Out of the box compatibility as both UPS and battery power supply with Raspberry Pi Model A+, Model B+ and Raspberry Pi Zero.
- Compatibility with Raspberry Pi 2 and 3 as a battery power supply only. For UPS functionality with Raspberry Pi 2 and 3 use a LiFePO$_4$wered/Pi3™ instead.
- Compatibility with original Raspberry Pi Model A and Model B as a battery power supply only, and either with additional wiring or by removing the composite video RCA connector.
- Compatibility with other Raspberry Pi style SBCs using time-based power off after shutdown instead of actual shutdown detection.
- A host-side command line tool, shared library and Python and Node.js bindings to allow easy configuration and integration into user programs and scripts.

Hardware installation

The LiFePO$_4$wered/Pi™ is designed to connect to the first 8 pins of the Raspberry Pi GPIO header. In case of the Pi Zero (which doesn't come with a header populated), it is necessary to first install a header for at least the first 8 GPIO pin locations.

The LiFePO$_4$wered/Pi™ has a single mounting hole which lines up with the mounting hole on the Raspberry Pi. For mechanical stability, it is recommended to mount the LiFePO$_4$wered/Pi™ to the Raspberry Pi using a 16 mm minimum length M2.5 machine screw, M2.5 nut and a 7/16" length, number 4 screw size nylon spacer which maintains the correct distance without putting stress on the GPIO header connections.

The image below shows a correctly installed LiFePO$_4$wered/Pi™.



There is also a version available with stackable 40-pin header, which allows the user to install the LiFePO$_4$wered/Pi™ in combination with other HATs.

Software installation

The LiFePO$_4$wered/Pi™ requires software to be running on the Raspberry Pi to operate correctly. This software provides a daemon that automatically manages the power state and shutdown of the Raspberry Pi, a shared library that allows integration of LiFePO$_4$wered/Pi™ functionality in user programs, a Python and Node.js library to provide easy access to the LiFePO$_4$wered/Pi™ from user scripts, and a CLI (command line interface) program that allows the user to easily configure the LiFePO$_4$wered/Pi™ from the command line or control it from shell scripts.

It used to be possible to do the initial install of the host software while the LiFePO$_4$wered/Pi™ was powering the Raspberry Pi, but this is not recommended for devices purchased in 2018 and later. These devices ship with updated firmware that includes a timeout for the boot sequence with a default length of 5 minutes. If the LiFePO$_4$wered/Pi™ is turned on and doesn't get notified by the host daemon that the system has booted within 5 minutes, the power will be turned off. While it may be possible to finish the host software installation outlined below within that time, it will be safer to do this step with the Raspberry Pi powered directly and not risk losing power during installation.

The LiFePO$_4$wered/Pi™ host software package can be found on Github at:

https://github.com/xorbit/LiFePO4wered-Pi

The "Clone or download" button provides the option to download a ZIP file or clone the software using Git. It is recommended to use Git since this makes updating the software easier. This can be done by opening a terminal window on the Raspberry Pi (using a local interface or over SSH), and running the following command to first ensure the build tools and Git are installed:

```
sudo apt-get -y install build-essential git
```

Then clone the software package to a location where you keep source software packages:

```
git clone https://github.com/xorbit/LiFePO4wered-Pi.git
```

Now go into the newly created LiFePO4wered-Pi directory by running:

```
cd LiFePO4wered-Pi
```

In the source project directory, you can now build the software by running:

```
python build.py
```

This will create the binaries to be installed on the system. These can be installed by running:

```
sudo ./INSTALL.sh
```

This will not only install the software to the Raspberry Pi system, but also perform any necessary configuration changes such as enabling the I^{2}C bus and enabling the GPIO UART.

When installation is complete, the Raspberry Pi can be shut down, and the LiFePO$_4$wered/Pi™ can now be used to power the system. The LiFePO$_4$wered/Pi™ daemon will automatically be started on boot and manage shutdown requests.

Basic usage

In the basic use case, the user does not need to interact with the LiFePO$_4$wered/Pi™ software on the Raspberry Pi at all once it is installed. The only necessary user interaction is with the touch button, with feedback provided by the green PWR LED.

To use the system as a basic power manager, just keep a 5 V USB charger connected to the LiFePO$_4$wered/Pi™ micro USB, like you normally would have it connected to the Raspberry Pi's own micro USB. The Raspberry Pi's micro USB should remain unconnected (the LiFePO$_4$wered/Pi™ would not be able to control the Raspberry Pi's power state if it was).

The LiFePO$_4$wered/Pi™ touch button can be used to turn the Raspberry Pi on and off. The touch button needs to be pressed and held to take effect. During this press-and-hold delay, the PWR LED glow will ramp up. The press-and-hold delay is implemented to prevent accidental activation when handling the system.

While the system is booting or shutting down, the LiFePO$_4$wered/Pi™ cannot respond to more touch input until the Raspberry Pi reaches the desired state (on or off). The changing of state (booting or shutting down) is indicated by the slow pulsing of the PWR LED, which indicates the system is “busy”. If the user touches the button during this time, the PWR LED will do a quick flashing sequence to indicate it cannot comply with the user request at that time. Once the Raspberry Pi reaches a steady state (on or off), the user can interact with the touch button to change power state again.

If the power going to the LiFePO$_4$wered/Pi™ is disconnected or fails, the system will keep running from the battery for 20 minutes to 3 hours, depending on the Raspberry Pi model, attached peripherals and system load. If the power returns during that time, the battery will be recharged and the system will not experience any down time. If the battery power runs out before the power returns, the LiFePO$_4$wered/Pi™ will instruct the Raspberry Pi to do a shutdown, and once the system is shut down properly, the power will be turned off.

In the default configuration, the system will not be automatically booted when power returns, but the user is expected to turn the system back on using the touch button. It is possible to make the Raspberry Pi boot again automatically when power returns by configuring the AUTO_BOOT setting in the configuration. This is ideal for unattended systems that need to provide maximum uptime.

If the user attempts to turn on the Raspberry Pi by touching the button when the battery is depleted, the PWR LED will do a quick flashing sequence to inform the user that the system cannot comply with the request. The same happened if the LiFePO$_4$wered/Pi™ detects that Raspberry Pi is already powered from another source.

Since the LiFePO$_4$wered/Pi™ ensures that the Raspberry Pi is always shut down in a proper way before power is removed, no matter what the reason for the shutdown is, the file systems are always properly unmounted and left in a clean state. This will go a long way in preserving reliable system operation and preventing SD card corruption, which often is a result of removing power while the system is running.

If for whatever reason the host system is unresponsive (for instance no Raspberry Pi present, no SD card present, kernel panic), the LiFePO$_4$wered/Pi™ can be forced off by pressing and holding the touch button for approximately 10 seconds. This feature is only available in firmware of units shipped in 2018 and later.

Limitations

UPS functionality

The LiFePO$_4$wered/Pi™ is designed as a UPS for the Raspberry Pi Model A+, Model B+ and Pi Zero, and stays within the power limits of the USB specification, using a charge current of 480 mA typical. While many USB chargers can provide more current, the LiFePO$_4$wered/Pi™ is thermally and chemically limited in charge current due to its compact size and will not use any extra available current. With a maximum input power of 2.4 W, it is obvious that if the load takes more energy out than the charger puts in over time, the battery will eventually become depleted.

So while the LiFePO$_4$wered/Pi™ is capable of providing reliable and stable power to any Raspberry Pi model under high load conditions, it may not always be able to provide continuous UPS functionality under such conditions, because the battery will eventually become depleted even when plugged in. This happens with high power consumption Raspberry Pi models such as the original Model B and Raspberry Pi 2 and 3, or when high power USB peripherals are connected.

A Raspberry Pi Model A+, Model B+ and Pi Zero can be powered in UPS mode indefinitely, even at 100% CPU load, not including external loads. Using a LiFePO$_4$wered/Pi™ as a UPS (simultaneous charge and discharge) for an original Model B, or a Raspberry Pi 2 or 3 is not supported, since the battery will become depleted even with minimal system load. For such use cases, the LiFePO$_4$wered/Pi3™ is available. Using a LiFePO$_4$wered/Pi™ as a battery power supply with a Raspberry Pi 2 or 3 (charge and discharge not simultaneous) is fully supported.

Battery run time

The run time on battery power depends on many factors, so only general guidelines can be given to help set expectations. In general a LiFePO$_4$ cell will have more capacity when discharged at a lower rate. A cell will also lose some capacity as it ages, but this effect is small in LiFePO$_4$ cells compared to most other lithium chemistries. The cell manufacturer specifies that the cell should still have 80% of its original capacity after 2000 cycles. Also note that the cell will have increased self-discharge at higher temperatures. When deployed in high temperature environments, make sure the cell is charged regularly so it doesn't discharge far enough to cause permanent damage (< 2 V).

The following scenarios can be used as a reference to estimate battery run times for the LiFePO$_4$wered/Pi™:

- Raspberry Pi 3 with 4 cores @ 100% load + Ethernet: 20 minutes
- Raspberry Pi 3 idle + WiFi: 1 hour
- Pi Zero + USB WiFi: 2 hours
- Pi Zero idle: 3 hours

Electrostatic discharge

In dry conditions, electrostatic charge can build up in the human body and this charge will be discharged into conductive systems such as the LiFePO$_4$wered/Pi™ when the user touches them.

While no reports of permanent damage due to electrostatic discharge have been received, it is possible that such a discharge will reset the microcontroller on the LiFePO$_4$wered/Pi™, cutting power to the Raspberry Pi abruptly without doing a proper shutdown first. In dry climates and during dry seasons, it is therefor recommended that the user first discharge before interacting with the LiFePO$_4$wered/Pi™.

Bidirectional load switch

As was mentioned, the LiFePO$_4$wered/Pi™ incorporates a bidirectional load switch which will protect the LiFePO$_4$wered/Pi™ from damage in case the Raspberry Pi is powered from another source such as its own micro USB power connector. However, this switch only works correctly if the LiFePO$_4$wered/Pi™ is powered (the battery is present). Applying power to the Raspberry Pi with the LiFePO$_4$wered/Pi™ connected but the battery removed will expose the LiFePO$_4$wered/Pi™ to voltages that can cause permanent damage.

Another scenario that causes damage to the LiFePO$_4$wered/Pi™ is powering a Pi Zero and connecting a back-powering powered hub to the Pi Zero's USB port. The Pi Zero accepts power input through the data USB and has no protection circuitry to prevent this back-power from damaging the LiFePO$_4$wered/Pi™.

In general, the safest thing is to *avoid powering the Raspberry Pi from any other source when the LiFePO$_4$wered/Pi™ is connected*. This will also ensure the LiFePO$_4$wered/Pi™ has full control over the power to the Raspberry Pi and properly can do its job as a UPS.

Hardware connections

This section describes the hardware connections available on the LiFePO$_4$wered/Pi™.

Micro-B USB connector

This is the main external power input to the system when using the LiFePO$_4$wered/Pi™. Power should not be applied to the Raspberry Pi from another source.

GPIO connector

The following table describes if and how each pin of the Raspberry Pi GPIO connector is used by the LiFePO$_4$wered/Pi™:

Pin	Name	Use
1	3V3	Unconnected.
2	5V	Power output of the LiFePO$_4$wered/Pi™ to provide power to the Raspberry Pi.
3	GPIO2 (SDA1)	I^{2}C bus data signal, used by the Raspberry Pi to communicate with the LiFePO$_4$wered/Pi™. Since it is a bus, it can be shared with other devices. Note that some HATs erroneously add pull-up resistors to the I^{2}C bus which can cause issues with logic levels. The Raspberry Pi as I^{2}C master already has the required pull-up resistors on board, so slave devices are not supposed to add any. If you experience communication issues when the LiFePO$_4$wered/Pi™ is used in combination with other HATs, please check whether they have this issue.
4	5V	Power output of the LiFePO$_4$wered/Pi™ to provide power to the Raspberry Pi.
5	GPIO3 (SCL1)	I^{2}C bus clock signal, used by the Raspberry Pi to communicate with the LiFePO$_4$wered/Pi™. Since it is a bus, it can be shared with other devices. Note that some HATs erroneously add pull-up resistors to the I^{2}C bus which can cause issues with logic levels. The Raspberry Pi as I^{2}C master already has the required pull-up resistors on board, so slave devices are not supposed to add any. If you experience communication issues when the LiFePO$_4$wered/Pi™ is used in combination with other HATs, please check whether they have this issue.

6	GND	Ground reference for power output and all signals.
7	GPIO4	Unconnected.
8	GPIO14 (TX0)	During the shutdown state, the TX signal from the Raspberry Pi is monitored to check if the Raspberry Pi has completed its shutdown. The Raspberry Pi will stop driving this signal when shutdown is completed. The LiFePO$_4$wered/Pi™ does not use this signal as a UART, it only monitors the logic level. The UART functionality is completely available to the user just as if the LiFePO$_4$wered/Pi™ wasn't present. Note that some UART connected modules erroneously add a pull-up resistor to this signal, usually as part of a scheme to provide legacy 5V support. This is not an approved method to terminate a UART signal and as it keeps the TX high when the Raspberry Pi has stopped driving it, it will prevent the LiFePO$_4$wered/Pi™ from correctly detecting that the system has finished shutdown.

The short story is that the presence of the LiFePO$_4$wered/Pi™ is pretty much transparent to the system, as long as it is the only device providing power. All the Raspberry Pi's GPIOs are still available to the user application.

Test pads / solder pads

The LiFePO$_4$wered/Pi™ has the following pads on the top circuit board:

Name	Use
V	Battery voltage. Used during production test.
C	Microcontroller programming clock. Used during production.
D	Microcontroller programming data. Used during production.
G	Ground reference for all signals.
T	Touch pad signal. Not available on very early hardware revisions. Can be used in combination with the G pad to add a remote on/off button that duplicates the functionality of the touch pad. For devices shipped before 2018, a momentary switch in series with a 100pF capacitor can be connected between the T and G pads to simulate a touch. Keep the wiring as short as possible. For devices shipped in 2018 and later, this still works, but the device can also be switched to a more robust "mechanical switch" mode that is less sensitive to the length of the wiring. This is accomplished by setting the `TOUCH_CAP_CYCLES` register to 0 and connecting a momentary switch between the T and G pads (without series capacitor).

Software interface

The LiFePO4wered/Pi™ exposes a set of registers that can be accessed from the Raspberry Pi through the I²C bus. By default, the 7-bit device address is 0x43. This can be changed in case of a conflict, but keep in mind that the shared library, daemon and CLI will need to be adjusted and recompiled to access the LiFePO4wered/Pi™ at any other address. To change the address in the host software, adjust the value of the `I2C_ADDRESS` definition in the `lifepo4wered-access.c` source file, recompile and reinstall.

This section provides a lot of low level detail necessary to implement direct access to the hardware, but please note that when using the provided host side software such as the CLI tool, the shared library or the provided Python or Node.js bindings, a lot of this complexity is hidden. These tools automatically convert low level values into convenient units such as millivolts and seconds, take care of mapping the correct register addresses based on register version, and provide definitions that clarify what is happening instead of having to use magic values in your code.

The tools also take care of access control in case multiple processes are trying to access the LiFePO4wered/Pi™ at the same time, data consistency checks when reading, write unlock code required since the 2018 firmware release, and retries in case of access contention. So it is recommended to use these tools if at all possible, instead of rolling your own.

Low level I²C access

The LiFePO4wered/Pi™ supports I²C access from the host at the standard speed of 100 kHz maximum. Faster speeds are not supported, since the controller handles a good portion of the I²C protocol in software and the Raspberry Pi does not support proper clock stretching.

I²C write access

The first byte of a write access is always the register address to be read or written. If the write access is only used to specify a register address for a read, this is all that is needed.

If the write access intends to write data, and the LiFePO4wered/Pi™ is running the 2018 or later firmware release (register version 5), an unlock code needs to be send next. Units running older firmware do not use or support the unlock code. The unlock code is calculated as:

```
(I2C_ADDRESS << 1) .xor. 0xC9 .xor. REG_ADDRESS
```

It is specifically intended to prevent erroneous writes in case of bus contention when other processes may be trying to use the I²C bus. If the unlock code is incorrect, the LiFePO4wered/Pi™ will respond with a NACK and the access fails.

After the unlock code, data bytes can be sent to be written. The internal register address pointer is automatically incremented after each byte, allowing for bulk data writes. Writes outside the valid range of registers that can be written will trigger a NACK and the write will fail.

Since the 2018 firmware release, the LiFePO$_4$wered/Pi™ uses a shadow buffer to cache and then atomically write complete registers. This measure can prevent race conditions when writing multi-byte registers that are in use by the controller.

I^{2}C read access

To read data from the LiFePO$_4$wered/Pi™, the host first needs to do a write with the desired register address, followed by one or more read accesses to read out the data. The internal register address pointer is automatically incremented after each byte, allowing for bulk data reads. Data reads outside the defined register range return 0xFF.

Low level I^{2}C register specification

The following I^{2}C registers are available in the LiFePO$_4$wered/Pi™:

I2C_REG_VER

1 byte, register address 0x00, read only access

Value: 0x01 (early prototypes), 0x02 (production prior to 11/11/16), 0x03 (production prior to 1/1/18), 0x05 (current production)

This value specifies an I^{2}C register set version. It allows the client software to choose the correct register addresses.

I2C_ADDRESS

1 byte, register address 0x01, read/write access, saved to flash

Default value: 0x43

7-bit bus address of the LiFePO$_4$wered/Pi™ device. If this is changed, the host software on the Raspberry Pi needs to be changed and recompiled to match the new value.

LED_STATE

1 byte, register address 0x02, read/write access, saved to flash

Default value: 0x01

This byte can be used to set the PWR LED state when the Raspberry Pi is on. The LED is under control of the LiFePO$_4$wered/Pi™ when the system is off (LED off), booting (LED pulsing) or shutting down (LED pulsing). When the Raspberry Pi is on, by default the LED is on solid, but this can be changed. Possible reasons to do so are to save power for maximum run time or to indicate the state of a user program. Possible values are: 0x00 (LED off), 0x01 (LED on), 0x02 (LED pulsing) or 0x03 (LED fast flash).

TOUCH_STATE

1 byte, register address 0x19 (register version 1), 0x1B (register version 2) or 0x1D (register version 3), 0x1F (register version 5), read only

Value is 0 if touch pad is not currently touched. Value is nonzero when indicating touch states. Press and hold of the touch button will make the Raspberry Pi turn off, but short touch events can be interpreted by user code. The 4 lowest bits of this byte indicate the last 4 touch button samples, shifting from the low to the high bit. For instance, a value of 0x01 indicates the user just started touching the touch pad, while 0x0E indicates the touch pad was just released after it had been held for at least 3 system ticks.

TOUCH_CAP_CYCLES

1 byte, register address 0x03, read/write access, saved to flash

Default value: 20

The total number of charge and discharge cycles generated and measured by the touch detection subsystem. This is one of the touch parameters that can be customized in case sensitivity needs to be adjusted.

In devices shipped in 2018 and later (register version 5), setting this register to 0 switches the touch detection to “mechanical switch” mode. To make this work, a momentary switch can be added between the T and G pads on the device.

TOUCH_THRESHOLD

1 byte, register address 0x04, read/write access, saved to flash

Default value: 12

Internally, a low pass filtered baseline is maintained that follows the average touch reading level. For a touch to be detected, the current touch reading has to exceed the baseline plus the touch threshold plus the touch hysteresis (see below). For the touch detection to become inactive, the current touch reading has to fall below the baseline plus the touch threshold minus the touch hysteresis. This is one of the touch parameters that can be customized in case sensitivity needs to be adjusted.

TOUCH_HYSTERESIS

1 byte, register address 0x05, read/write access, saved to flash

Default value: 2

The touch detection system has a hysteresis to ensure reliable touch detection performance. The hysteresis is added to and subtracted from the touch threshold, depending on whether an active touch is detected. This is one of the touch parameters that can be customized in case sensitivity needs to be adjusted.

DCO_RSEL

1 byte, register address 0x06, read/write, saved to flash

Default value: factory calibrated

This value is factory calibrated so the microcontroller clock runs at 12 MHz. Refer to the MSP430G2231 datasheet for more details. The user should not need to change this value.

DCO_DCOMOD

1 byte, register address 0x07, read/write, saved to flash

Default value: factory calibrated

This value is factory calibrated so the microcontroller clock runs at 12 MHz. Refer to the MSP430G2231 datasheet for more details. The user should not need to change this value.

VBAT (called VIN prior to 11/10/2016)

2 bytes little endian, register address 0x15 (register version 1), 0x17 (register version 2) or 0x19 (register version 3), 0x1E (register version 5), read only

Value (register version 3 and below): battery voltage, 10-bit value, 5 V full scale, resolution of 4.88 mV per LSB

Value (register version 5): battery voltage, 13-bit value, 5 V full scale, resolution of 0.61 mV per LSB

This value represents the battery voltage. The Raspberry Pi host software package contains scaling code so the value is converted to mV for convenience.

VOUT

2 bytes little endian, register address 0x17 (register version 1), 0x19 (register version 2) or 0x1B (register version 3), 0x20 (register version 5), read only

Value (register version 3 and below): output (Raspberry Pi supply) voltage, 10-bit value, 5.548 V full scale, resolution of 5.42 mV per LSB

Value (register version 5): output (Raspberry Pi supply) voltage, 13-bit value, 5.548 V full scale, resolution of 0.68 mV per LSB

This value represents the output (Raspberry Pi supply) voltage. The Raspberry Pi host software package contains scaling code so the value is converted to mV for convenience.

VBAT_MIN (VIN_MIN prior to 11/10/2016)

2 bytes little endian, register address 0x08, read/write, saved to flash

Default value (register version 3 and below): 584 (corresponding to 2.85 V, resolution of 4.88 mV per LSB)

Default value (register version 5): 4669 (corresponding to 2.85 V, resolution of 0.61 mV per LSB)

This value determines the minimum battery voltage. If the input voltage falls below this value, the LiFePO$_4$wered/Pi™ will immediately shut the Raspberry Pi power off so no damage occurs to the battery. Note that this is an emergency procedure which normally doesn't occur, the Raspberry Pi should have been given a command to shut down at a higher battery voltage, but in case the Raspberry Pi fails to shut down, this is provided as a safety feature.

The Raspberry Pi host software package contains scaling code so the value can be read and set in mV for convenience.

VBAT_SHDN (VIN_SHDN prior to 11/10/2016)

2 bytes little endian, register address 0x0A, read/write, saved to flash

Default value (register version 3 and below): 604 (corresponding to 2.95 V, resolution of 4.88 mV per LSB)

Default value (register version 5): 4833 (corresponding to 2.95 V, resolution of 0.61 mV per LSB)

This value determines the battery voltage at which the Raspberry Pi will be instructed to shut down. The Raspberry Pi host software package contains scaling code so the value can be read and set in mV for convenience.

VBAT_BOOT (VIN_BOOT prior to 11/10/2016)

2 bytes little endian, register address 0x0C, read/write, saved to flash

Default value (register version 3 and below): 645 (corresponding to 3.15 V, resolution of 4.88 mV per LSB)

Default value (register version 5): 5156 (corresponding to 3.15 V, resolution of 0.61 mV per LSB)

This value determines the battery voltage level at which the Raspberry Pi is allowed to boot. Note that this value is higher than VBAT_SHDN to provide hysteresis. This will ensure that the system will not oscillate between boot and shutdown when the battery is nearly empty, but then the voltage recovers when the load is turned off. Under heavy load, it may be necessary to increase this value to prevent continuous boot / shutdown cycling.

The Raspberry Pi host software package contains scaling code so the value can be read and set in mV for convenience.

VOUT_MAX

2 bytes little endian, register address 0x0E, read/write, saved to flash

Default value (register version 3 and below): 717 (corresponding to 3.88 V, resolution of 5.42 mV per LSB)

Default value (register version 5): 5722 (corresponding to 3.88 V, resolution of 0.68 mV per LSB)

This value determines the minimum output voltage present for which the LiFePO$_4$wered/Pi™ will refuse to boot the Raspberry Pi when it is supposed to be off (according to the LiFePO$_4$wered/Pi™). The LiFePO$_4$wered/Pi™ power supply employs a bidirectional load switch that makes it possible to power the Raspberry Pi from a different source (such as its own micro USB power connector) with the LiFePO$_4$wered/Pi™ attached without causing any damage (NOTE: this only works if the battery is present, damage *will* occur if the Raspberry Pi is powered from another power source and the battery has been removed from the LiFePO$_4$wered/Pi™). Because the LiFePO$_4$wered/Pi™ should not be allowed to turn on when the Raspberry Pi is powered from a different source, this voltage check provides a safety feature that prevents this from happening.

The Raspberry Pi host software package contains scaling code so the value can be read and set in mV for convenience.

VOFFSET_ADC

2 bytes little endian, register address 0x10 (register version 3), read/write, saved to flash

Default value: 0 (corresponding to 0 V, resolution of 4.88 mV per LSB)

This register provides a calibration value for the ADC voltage measurements. It is a simple 1 point offset calibration that provides compensation for inaccuracy of the internal reference voltage.

The Raspberry Pi host software package contains scaling code so the value can be read and set in mV for convenience.

AUTO_BOOT

1 byte, register address 0x10 (register version 1), 0x12 (register version 2) or 0x14 (register version 3 and 5), read/write, saved to flash

Default value: 0x00

When this register is 0 (AUTO_BOOT_OFF), the LiFePO4wered/Pi™ will stay off until the user touches the on/off touch pad to turn the Raspberry Pi on.

Setting this register to 1 (AUTO_BOOT_VBAT) will make the Raspberry Pi boot immediately when sufficient battery voltage is available (VBAT >= VBAT_BOOT threshold). This is useful when using the LiFePO4wered/Pi™ as a UPS to maximize uptime.

On units shipped after 12/1/2016, setting this register to 2 (AUTO_BOOT_VBAT_SMART) will make the Raspberry Pi boot immediately when sufficient battery voltage is available, but only if the unit was previously not shut down by the user, but shut down due to a low voltage condition or watchdog timeout. This makes it so the user can still choose to turn the Raspberry Pi off with the touch button or from a user program.

WAKE_TIME

2 bytes little endian, register address 0x12 (register version 1), 0x14 (register version 2) or 0x16 (register version 3), 0x1A (register version 5), read/write, not saved to flash

Default value: 0

This register allows the user to set a time in minutes that determines how long the Raspberry Pi will stay off before the LiFePO4wered/Pi™ will automatically boot it again. It is implemented using RC oscillators in the microcontroller and as such has limited accuracy (expect +/- 10%). If the value is 0, the wake timer is off.

This value cannot be saved in flash, but needs to be set by a user program every time before the Raspberry Pi shuts down. It allows extended run time on battery power for tasks that have low duty cycles. The LiFePO4wered/Pi™ will still respond to touch button presses and AUTO_BOOT as usual when the wake timer is set.

A user program can check this value after boot, the value will reflect the number of minutes remaining in the wake timer when the system was booted. If there is still time remaining, this indicates the system boot was not triggered by the wake timer, but from another source.

SHDN_DELAY

2 bytes little endian, register address 0x10 (register version 2) or 0x12 (register version 3 and 5), read/write, saved to flash

Default value: 65

This sets the number of LiFePO$_4$wered/Pi™ system ticks that elapse between when the Raspberry Pi is shut down (detected by the UART TX line going low) and when the power to it is turned off. The system ticks are not at all accurate (they are implemented with a low power oscillator), and can vary from 2.6 to 13 ticks per second. The default value is chosen to allow plenty of time between shutdown and power off, even with the fastest system tick. To attain maximum run time on battery power in low duty cycle systems using the wake timer, the user can reduce this value.

Another possible use for changing this value is when the Raspberry Pi 3 is configured to disable the UART on the GPIO header. This is actually the default state on the Raspberry Pi 3, however the LiFePO$_4$wered/Pi™ software installer will change the system configuration to turn the UART back on. If this conflicts with what the user wants to do, it is possible to keep the UART disabled and set this register to a large value. This can make the system work correctly for shutdown and reboot even when UART TX line detection is not available. The delay in that case has to be long enough to last through a reboot from the time the LiFePO$_4$wered/Pi™ daemon is unloaded until it's loaded again.

PI_BOOT_TO

1 byte, register address 0x15 (register version 5), read/write, saved to flash

Default value: 30 (corresponding to 300 seconds or 5 minutes, resolution of 10 seconds per LSB)

Since the 2018 firmware update (register version 5), the LiFePO$_4$wered/Pi™ will not indefinitely stay in the boot state anymore if there is no response from the Raspberry Pi. Instead, power will be turned back off after the boot timeout expires. The boot timeout can be set in 10 second increments up to 2550 seconds or 42.5 minutes. Setting this register to 0 turns the boot timeout off.

The Raspberry Pi host software package contains scaling code so the value can be read and set in seconds for convenience.

PI_SHDN_TO

1 byte, register address 0x16 (register version 5), read/write, saved to flash

Default value: 12 (corresponding to 120 seconds or 2 minutes, resolution of 10 seconds per LSB)

Since the 2018 firmware update (register version 5), the LiFePO$_4$wered/Pi™ will not indefinitely stay in the shutdown state anymore if the UART TX line (which is monitored to detect kernel shutdown) stays high. Instead, power will be turned off after the shutdown timeout expires. The shutdown timeout can be set in 10 second increments up to 2550 seconds or 42.5 minutes. Setting this register to 0 turns the shutdown timeout off.

This feature makes the LiFePO$_4$wered/Pi™ compatible with many more single board computers that use the Raspberry Pi form factor and compatible GPIO header. One common problem when trying to use the LiFePO$_4$wered/Pi™ with these boards was that they do not bring the UART TX line low on shutdown, causing the power to stay on until the battery ran out. Using this shutdown timeout, the power can now be turned off after giving the board a reasonable amount of time to complete the shutdown sequence.

The Raspberry Pi host software package contains scaling code so the value can be read and set in seconds for convenience.

WATCHDOG_CFG

1 byte, register address 0x17 (register version 5), read/write, saved to flash

Default value: 0x00

Since the 2018 firmware update, the LiFePO$_4$wered/Pi™ supports a watchdog feature to ensure the user application keeps running correctly. It consists of a timer that counts down every 10 seconds, and when it reaches zero, an action is taken. Which action is taken is determined by this watchdog configuration register.

When the value is 0x00 (WATCHDOG_OFF), no action is taken and the watchdog feature is turned off. In this state the watchdog timer will also not count down.

When the value is 0x01 (WATCHDOG_ALERT), the LED will start producing the fast error flash instead of the normal steady on (or whatever pattern the user has activated using the LED_STATE register) when the watchdog timer reaches zero. This can alert the user that something is wrong with their application.

When the value is 0x02 (WATCHDOG_SHDN), the LiFePO$_4$wered/Pi™ will trigger a Raspberry Pi shutdown when the watchdog timer reaches zero. All the normal shutdown behavior is active: the Raspberry Pi will be told to shut down, but if it fails to do so, the shutdown timeout or the battery voltage falling below the VBAT_MIN threshold will turn the Raspberry Pi off if it fails to perform a clean shutdown. In combination with the AUTO_BOOT feature, this can be used to restart the

Raspberry Pi if the user application locks up and recover the application from a clean boot.

WATCHDOG_GRACE

1 byte, register address 0x18 (register version 5), read/write, saved to flash

Default value: 2 (corresponding to 20 seconds, resolution of 10 seconds per LSB)

This sets the watchdog grace period after the host daemon informs the LiFePO$_4$wered/Pi™ that the system has booted, until the user application has the chance to write to the watchdog timer. It essentially is the initial value written to the watchdog timer on boot.

A user application may need some time after boot to start up, initialize, connect to WiFi, establish a server connection, etc. before it is ready to service the watchdog timer. A full stack watchdog may require a lot of things to happen before it is ready to declare success by writing to the watchdog, or may even require a remote heartbeat to service the watchdog. All of this needs time.

The watchdog grace period can be set to something that makes sense for the user's application, depending on how much time is required. It can be set in 10 second increments up to 2550 seconds or 42.5 minutes.

The Raspberry Pi host software package contains scaling code so the value can be read and set in seconds for convenience.

WATCHDOG_TIMER

1 byte, register address 0x1C (register version 5), read/write, not saved to flash

Default value: WATCHDOG_GRACE on boot

This is the watchdog timer register that needs to be written with a time value in 10 second increments when the watchdog is enabled with WATCHDOG_CFG. When the watchdog is enabled, from the moment the timer is written it will count down in 10 second steps, until it reaches zero. When it reaches zero, the action configured in WATCHDOG_CFG will take place.

The watchdog is intended as a user application watchdog and is *not* serviced by the LiFePO$_4$wered/Pi™ daemon. The idea is that if the user application is working correctly it will reset the timer value to a sufficient value at regular intervals, but if the user application suffers a failure, this will not happen in time and the LiFePO$_4$wered/Pi™ can take an appropriate action to try and recover.

The Raspberry Pi host software package contains scaling code so the value can be read and set in seconds for convenience.

PI_RUNNING

1 byte, register address 0x14 (register version 1), 0x16 (register version 2), 0x18 (register version 3), 0x1D (register version 5), read/write, not saved to flash

Default value: 1 once the Raspberry Pi is booted

This is an important register that determines the state of the Raspberry Pi power. It is normally managed by the LiFePO$_4$wered/Pi™ itself and the LiFePO$_4$wered/Pi™ daemon on the host. When the power to the Raspberry Pi is off or when the Raspberry Pi is booting, the value of this flag is 0. When the LiFePO$_4$wered/Pi™ daemon starts, it sets this flag to 1 to indicate the Raspberry Pi has booted. This will change the state of the LiFePO$_4$wered/Pi™, the PWR LED will go from pulsing to on state, and will be ready for user input (touching the button to turn the Raspberry Pi off again). This flag can be cleared by various sources, such as a user pressing the touch button, the battery voltage falling below VBAT_SHDN, or the daemon being shut down when a user shuts down the Raspberry Pi. On the other hand, the daemon will also *trigger* a shutdown if this flag goes low from another source. In either case, the system will be shutting down, the LiFePO$_4$wered/Pi™ will show this by pulsating the PWR LED and the power will be turned off.

As mentioned, the user does not need to worry about manually controlling this flag, the LiFePO$_4$wered/Pi™ daemon takes care of it. If the user sets this flag to 0, this will trigger a system shutdown.

CFG_WRITE

1 byte, register address 0x11 (register version 1), 0x13 (register version 2), 0x15 (register version 3), 0x19 (register version 5), read/write

Default value: 0

This register makes it possible to make configuration changes permanent by writing the values to flash memory (only those marked by “saved to flash”). It may not be necessary to use this, since the LiFePO$_4$wered/Pi™ microcontroller stays powered even when the Raspberry Pi is off. However, when the battery is removed, configuration changes will be lost. This can be a good thing, it allows the user to experiment with changing configuration values, and if they cause a problem, they can be undone by removing the battery and putting it back, with power disconnected. Only if the user is very sure about their configuration changes should they be written to flash. Writing bad configurations to flash can **MAKE THE DEVICE UNUSABLE**.

To write the current configuration to flash, the user has to write the “magic value” 0x46 (70) to the CFG_WRITE register. Any other value is ignored, and the register is always read as 0.

Command line tool specification

To make it convenient to interact with the LiFePO$_4$wered/Pi™, the software package installed on the Raspberry Pi provides a command line tool. Help is provided when you run it without parameters:

```
lifepo4wered-cli
```

The tool can be used to get and set the values of the LiFePO$_4$wered/Pi™ I^{2}C registers described in the previous section without having to know implementation details such as register addresses and unit scaling. For instance, to get the current battery voltage, run:

```
lifepo4wered-cli get vbat
```

This will return the battery voltage converted to millivolts. If no register is specified, the values of all available registers are dumped:

```
lifepo4wered-cli get
```

You can use `hex` instead of `get` if you want to display values in hexadecimal notation.

Setting values works similarly. Values can be provided in decimal notation or hexadecimal by using the `0x` prefix. For instance, to set the wake time to an hour, run:

```
lifepo4wered-cli set wake_time 60
```

When you shut down the Raspberry Pi, it will wake up again in about 60 minutes. Or if you want the Raspberry Pi to boot when there is enough battery power to do so, you can run:

```
lifepo4wered-cli set auto_boot 0x01
```

Please refer to the I^{2}C register specification for a complete reference of available options, but note that the command line tool will convert many registers from their low level values to more useful units such as millivolts and seconds for convenience.

Electrical characteristics

Unless otherwise indicated, all characteristics apply for V_{USB} = 4.5 V to 5.5V and T_A = 0 ºC to 50 ºC. Typical values are at 25 ºC and V_{USB} = 5 V.

Parameter	Sym	Min	Typ	Max	Unit	Conditions
USB charge voltage	V_{USB}	4.2	5.0	6.5	V	
Battery leakage current	$I_{DISCHARGE}$		4		µA	USB voltage absent, Raspberry Pi powered off

Battery charge current	I_{CHARGE}		480		mA	With sufficient cooling to prevent thermal limiting
Continuous load current	I_{LOAD}		0.3		A	
Output voltage	V_{OUT}	4.8	5.0	5.2	V	
Wake timer accuracy	t_{WAKE}	-20		+20	%	Units shipped prior to 12/2/2016
Wake timer accuracy	t_{WAKE}	-10		+10	%	Units shipped after 12/2/2016
Default minimum battery voltage (power forced off)	$VBAT_{MIN}$		2.85		V	
Default shutdown battery voltage (Pi shutdown triggered)	$VBAT_{SHDN}$		2.95		V	
Default minimum boot battery voltage	$VBAT_{BOOT}$		3.15		V	
Default output voltage preventing boot	$VOUT_{MAX}$		3.88		V	Raspberry Pi is powered from another source

Sales and support

To buy the LiFePO$_4$wered/Pi™, please visit http://lifepo4wered.com. To order in quantity and for volume discounts, please contact sales@lifepo4wered.com.

For technical support, please contact support@lifepo4wered.com.