Introduction

The LiFePO4wered/Pi+™ is a high performance battery power system for the Raspberry Pi. It can power a Raspberry Pi for up to 9 hours from the battery (depending on installed battery size, Raspberry Pi model, attached peripherals and system load) and can be left plugged in continuously. It features:

- A single 3.2 V, 18650 size 1500 mAh $LiFePO_4$ (lithium iron phosphate) cell or optionally a 14500 size 600 mAh $LiFePO_4$ cell, providing high power density, extended cycle life (2000+ cycles), and safety from fire and explosions.
- Continuous output load current of 2 A with the 18650 size battery, both when connected to external power and when on battery power.
- A smart charge controller with over-charge protection, allowing the device to stay plugged in continuously and provide UPS (Uninterruptible Power Supply) functionality.
- Smart auto-adjusting charge current allowing charge currents up to 1.5 A when used with high

power chargers, but automatically reducing current as needed to not overload lower power sources such as a PC USB port.

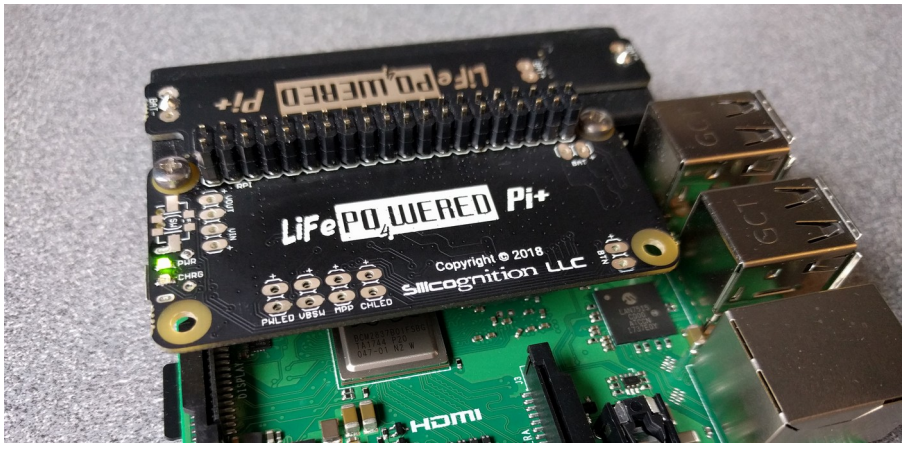
- Customizable MPPT functionality making it possible to charge the LiFePO4wered/Pi+™ from suitably sized 5 – 18 V nominal solar panels.
- Two-way communication between the power system and the Raspberry Pi over I^2C bus (to monitor voltages and customize settings) and Raspberry Pi shutdown detection.
- A smart power manager controller and open source daemon working in tandem to provide clean shutdown functionality and over-discharge protection, resulting in long life and high reliability.
- Continuous measurement of input voltage, battery voltage, output voltage and load current with smart user programmable thresholds for boot, shutdown and hard power down.
- An on-board mechanical on/off button providing clean boot/shutdown capability even in headless setups, with convenient connection points to add an external button if desired.
- Press and hold button functionality protecting against accidental button activation and enabling the possibility to use short button presses to control user software.
- A green PWR LED indicating the Raspberry Pi power state, providing feedback to the user and controllable by user software, plus convenient connection points to add an external power LED if desired.
- A separate red CHRG LED indicating charging state, plus convenient connection points to add an external charge LED if desired.
- A wake timer allowing the Raspberry Pi to be off most of the time and wake up as needed for low duty cycle applications.
- Real-time clock functionality, keeping track of time when the Raspberry Pi is off and restoring system time on boot, with the ability to wake up at an absolute time.
- An auto-boot feature maximizing uptime by making the Raspberry Pi run whenever there is sufficient battery power, or when the input power is present or absent.
- An auto shutdown feature making it possible to automatically shut down the Raspberry Pi immediately when input power is been removed or after a programmable amount of time.
- A flexible watchdog timer able to alert a user by flashing the PWR LED or 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 with Raspberry Pi Model A+, Model B+, Raspberry Pi 2 and 3, Raspberry Pi 3 Model A+ and B+ and Raspberry Pi 4 (with 18650 battery), Raspberry Pi Zero and Raspberry Pi Zero W.
- Compatibility with original Raspberry Pi Model A and Model B with additional wiring, or by removing the composite video RCA connector.
- Compatibility with other Raspberry Pi style SBCs using time or load current based power off after shutdown, if shutdown detection is not available.
- Convenient connection points for input power, 5 V output power and switched battery power.
- A host-side command line tool, shared library and Python and Node.js bindings allowing easy configuration and integration into user programs and scripts.

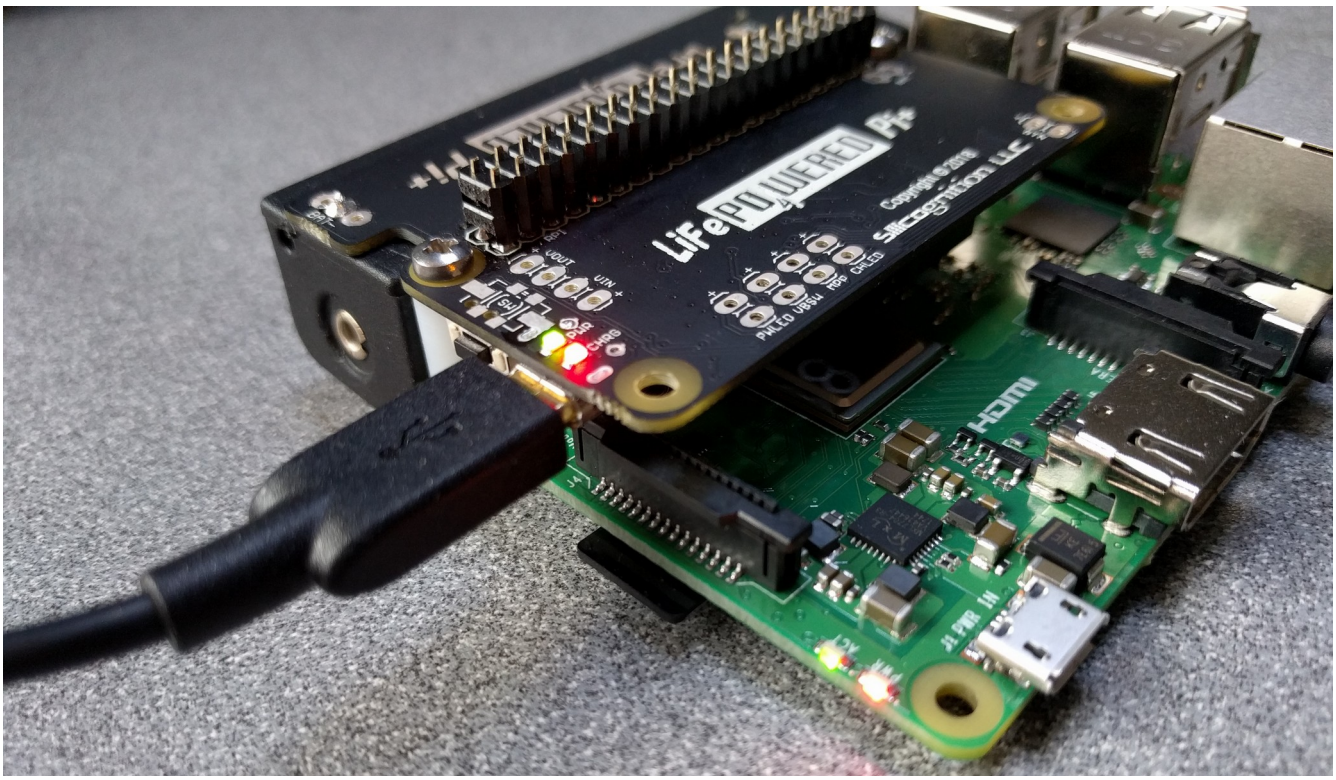
Hardware installation

The LiFePO4wered/Pi+™ is designed to plug in to 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. To ensure compatibility with all Raspberry Pi models, the LiFePO4wered/Pi+™ has the same form factor as a Raspberry Pi Zero, but with a battery holder that is located to the side of the Raspberry Pi.

The LiFePO4wered/Pi+™ has four mounting holes with the same layout as a Raspberry Pi Zero. When used with a Pi Zero, all four holes can be used to mount the LiFePO4wered/Pi+™ to the Pi Zero. When used with other Raspberry Pi models, only the two mounting holes next to the GPIO header are used. For mechanical stability, it is recommended to mount the LiFePO4wered/Pi+™ to the Raspberry Pi using two 16 mm minimum length M2.5 machine screws, M2.5 nuts and a 7/16” length, number 4 screw size nylon spacers which maintains the correct distance without putting stress on the GPIO header connections.



External power has to be connected to the LiFePO4wered/Pi+™ instead of the Raspberry Pi's own power input. This is necessary to allow battery charging and to ensure that the LiFePO4wered/Pi+™ has complete control over the Raspberry Pi's power.



Available hardware options

Battery size

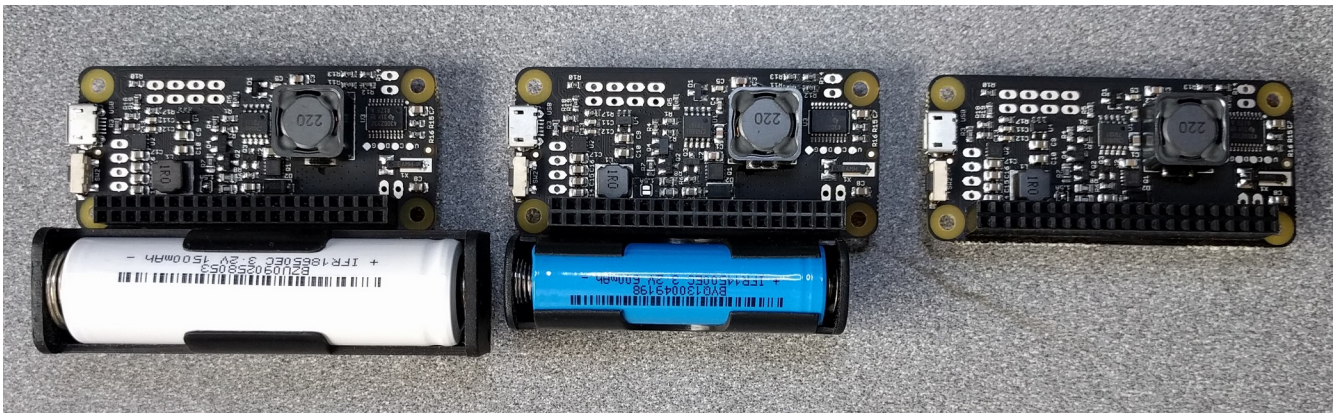
Three standard battery options are available:

- O'Cell IFR18650EC 3.2 V 1.5 Ah mounted in 18650 battery holder. This option is recommended for heavily loaded systems with many peripherals.
  It supports a continuous load current of 2 A, both when plugged in and when running from battery. If charging below 0°C is required, a cell from A123 or K2 Energy can be used instead.
- O'Cell IFR14500EC 3.2 V 600 mAh mounted in 14500 (AA) battery holder. This option reduces cost, size and weight, and is ideal for lower power systems when used as a battery supply, or in some higher power systems when used as a UPS requiring only enough power to

perform a shutdown when external power is removed. It does not support the Raspberry Pi Model 3 A+ and B+ or the Raspberry Pi 4.
It supports a continuous load current of 2 A when plugged in to a quality high power (3 A) supply, less if the source power supply voltage sags under high load. When running from battery, it supports a continuous load current of 0.75 A.

- No on-board battery, and battery compartment removed. This option requires the user to obtain and connect their own 1S $LiFePO_4$ battery pack to the BAT terminals. This is the lowest cost and smallest solution, making the footprint equal to that of a Raspberry Pi Zero, and allows the user the freedom to use large packs to obtain very long run times.
  Performance will depend on the cells used and the wiring. Keeping the wiring short and using thick gauge wire will provide the best performance, similar to that of the IFR18650EC option.
  Note that only 1S (3.2 V) $LiFePO_4$ chemistry battery packs are supported, and that a battery has to be connected for proper operation! The LiFePO4wered/Pi+™ will not function correctly without a battery present.

Aside of these standard options, other battery options are available by request for bulk orders. The picture below shows the different standard battery options.

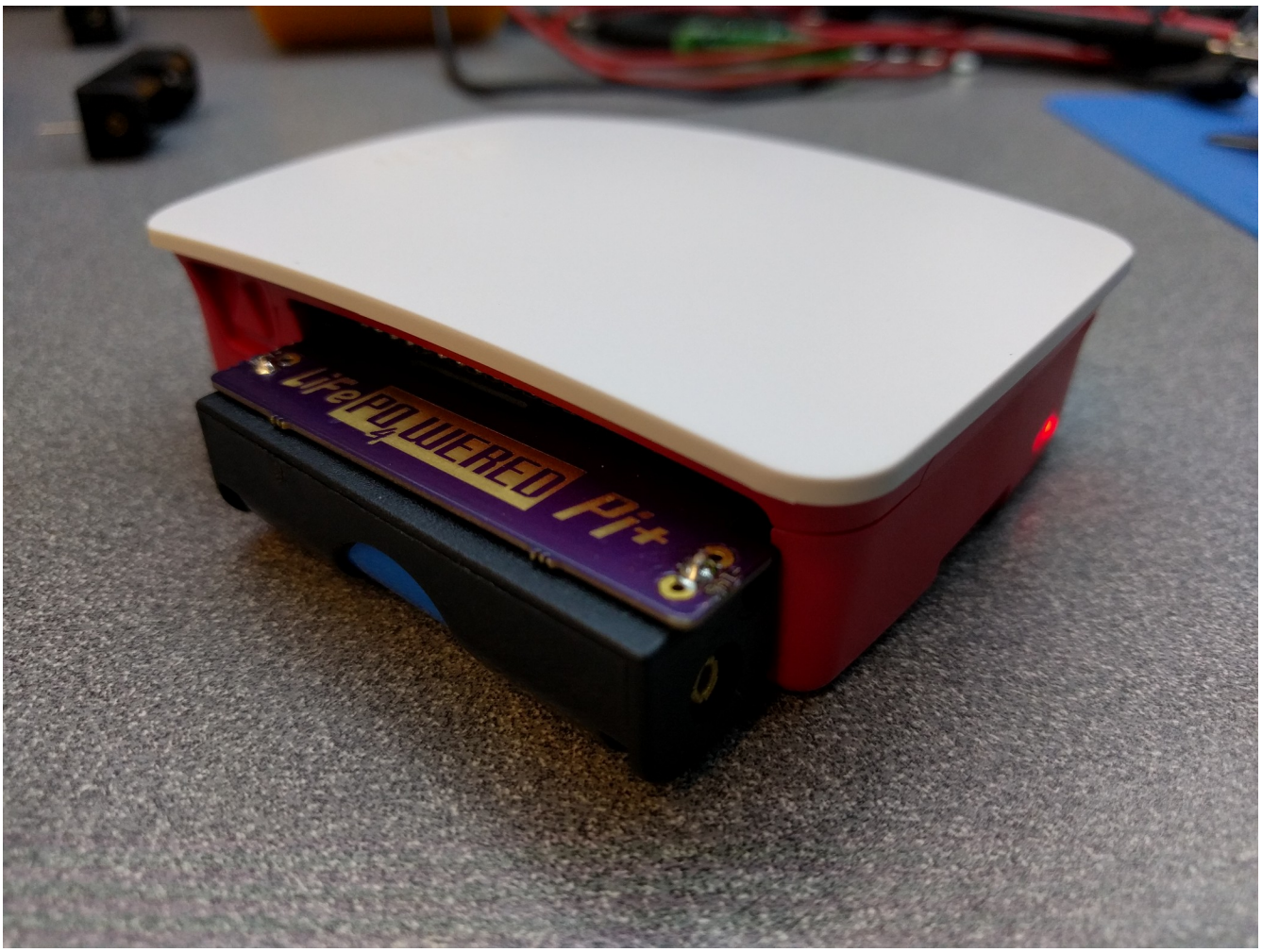


Battery spacing

By default, the battery is installed so it sits right next to the Raspberry Pi board. This is the “compact” option, and is the one to be used in Raspberry Pi cases that are specifically designed to accommodate the LiFePO4wered/Pi+™. A “spaced” option is available that provides 2.5 mm of space between the battery and the Raspberry Pi, allowing the use of some standard Raspberry Pi cases (possibly needing some modification) while the battery is outside the case.

The picture below shows an example of a prototype LiFePO4wered/Pi+™ in a modified official

Raspberry Pi case. While the case allows no access to the LiFePO4wered/Pi+™ micro USB power port or the power button without making extra holes, by using the auto-boot/auto-shutdown features and connecting power wiring directly to the VIN connections, this would be a perfectly usable setup.



GPIO header

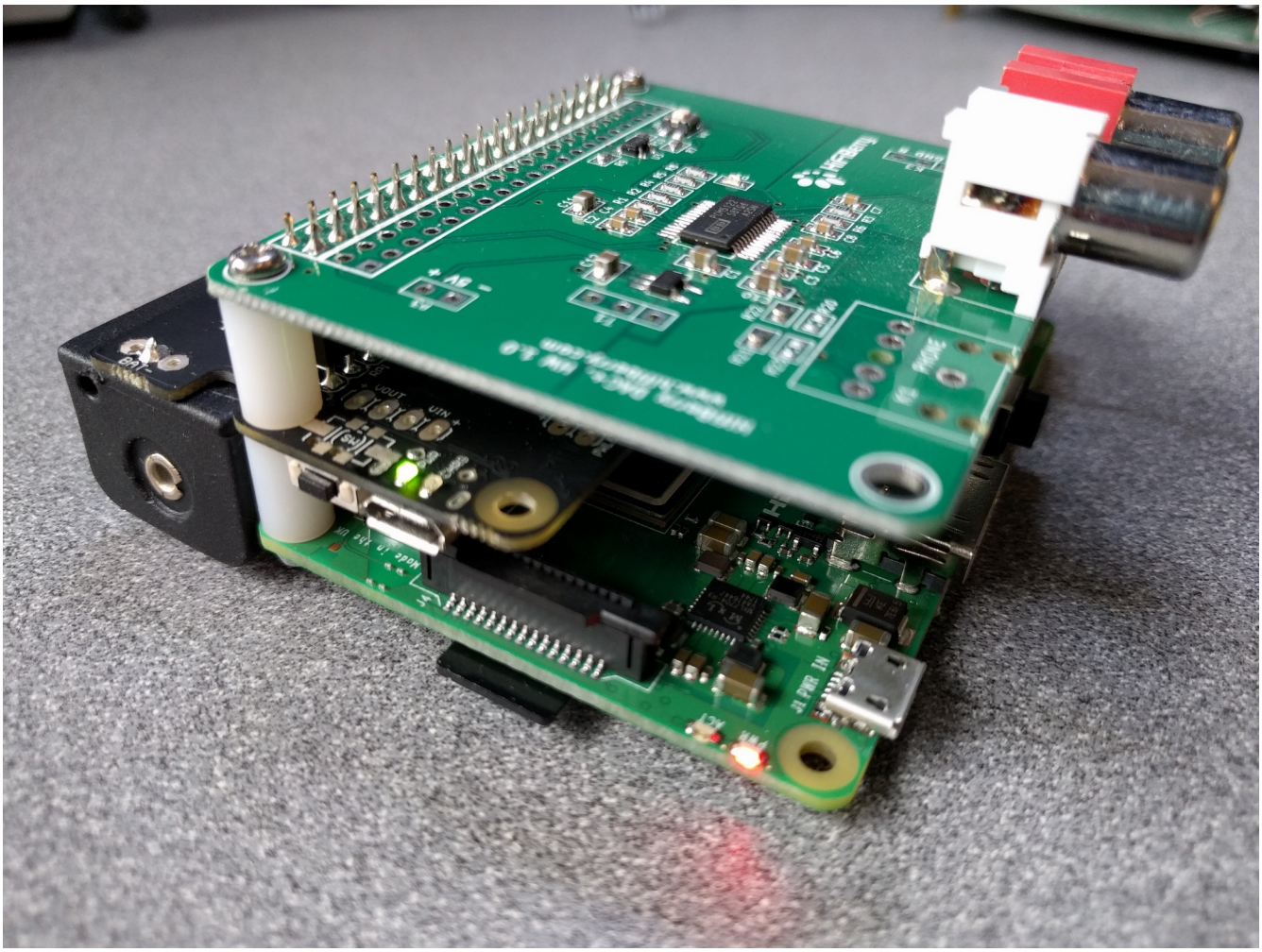
Two options are available for the GPIO header:

- A normal short 40-pin female header.
- A stackable 40-pin header with extended pins to allow installation of Raspberry Pi HATs on top.

The stackable header option is shipped with spacers covering the pins for protection during shipping. To install a HAT, the spacers need to be removed to make the pins available. Note that the extended

pins are not square like the Raspberry Pi's own GPIO pins but flatter in one direction. This causes no issues for 40-pin female headers plugged in on top, because the orientation of the female contacts is ensured to mate correctly with the larger pin dimension. However, these pins don't work well with fly leads, because the orientation of the contacts is not enforced.

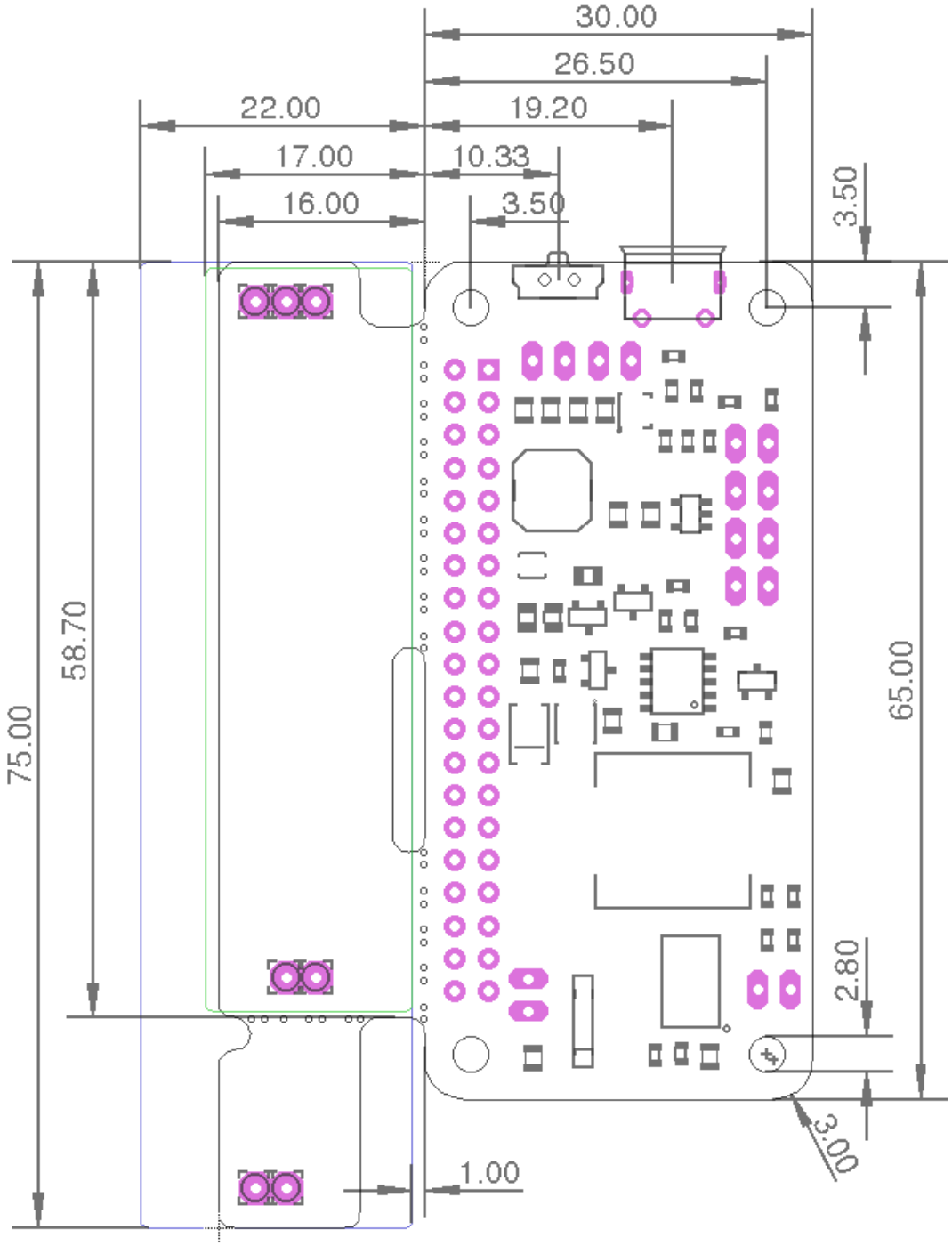
As an example, the picture below shows a setup with a Raspberry Pi 3 B+, a LiFePO4wered/Pi+™ and a HifiBerry DAC+ powered from the battery.



Mechanical dimensions

The LiFePO4wered/Pi+™ provides much functionality in a compact form factor, with most of the circuitry confined within the same footprint as a Raspberry Pi Zero. The on-board battery is located outside this footprint to limit vertical space requirements and prevent issues when stacking HATs on

top. The large power inductor location is optimized to place it between the Raspberry Pi's CPU and USB / Ethernet hub, allowing the user to add heat sinks to these chips if desired. To assist the user with the mechanical design of their system, below is a mechanical drawing with the most pertinent dimensions.



Note that this drawing is a view of the component side or bottom of the PCB.

The drawing contains dimensions for both battery sizes. The 14500 battery holder outline is marked in green while the 18650 battery holder outline is blue. When the 14500 battery is used, the small breakaway tab in the bottom left corner is removed. When there is no on-board battery holder but an external battery is used, the whole left side breakaway area is removed.

When the LiFePO4wered/Pi+™ is installed on a Raspberry Pi, the total height from the top of the LiFePO4wered/Pi+™ PCB to the bottom of the Raspberry Pi PCB is about 14 mm. The total height from the top of the LiFePO4wered/Pi+™ PCB to the bottom of the 14500 battery is 16.5 mm, meaning that this battery extends 2.5 mm below the bottom of the Raspberry Pi PCB. The total height from the top of the LiFePO4wered/Pi+™ PCB to the bottom of the 18650 battery is 21.5 mm, meaning that this battery extends 7.5 mm below the bottom of the Raspberry Pi PCB. When installed in a case, the Raspberry Pi should be mounted on standoffs to allow enough clearance so the battery does not get pushed against the bottom of the case.

Software installation

The LiFePO4wered/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 LiFePO4wered/Pi+™ functionality in user programs, a Python and Node.js library to provide easy access to the LiFePO4wered/Pi+™ from user scripts, and a CLI (command line interface) program that allows the user to easily configure the LiFePO4wered/Pi+™ from the command line or control it from shell scripts.

Initial installation of the host software should preferably be done before the LiFePO4wered/Pi+™ hardware is installed. The reason is that the LiFePO4wered/Pi+™ firmware includes a timeout for the boot sequence with a default length of 5 minutes. If the LiFePO4wered/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 LiFePO4wered/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, Git and the systemd support library are installed:

```
# sudo apt-get -y install build-essential git libsystemd-dev
```

The systemd support library is optional and does not need to be installed if systemd support for the daemon is not required or desired.

Next 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:

```
# make all
```

Or alternatively, if systemd support is not required or desired, run:

```
# make all USE_SYSTEMD=0
```

After building the binaries, they need to be installed on the system. This can be done by running:

```
# sudo make user-install
```

This will not only install the software to the Raspberry Pi system, but also perform any necessary configuration changes such as enabling the I^2C bus and enabling the GPIO UART (UART pin TX is used for shutdown detection).

When installation is complete, the Raspberry Pi can be shut down, and the LiFePO4wered/Pi+™ can now be used to power the system. The LiFePO4wered/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 LiFePO4wered/Pi+™ software on the Raspberry Pi at all once it is installed. The only necessary user interaction is with the on/off 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 LiFePO4wered/Pi+™ micro USB, like you normally would have it connected to the Raspberry Pi's own power input. The Raspberry Pi's own power input should remain unconnected (otherwise the LiFePO4wered/Pi+™ will not be able to control the Raspberry Pi's power state and will in fact refuse to

power the Raspberry Pi).

The LiFePO4wered/Pi+™ button can be used to turn the Raspberry Pi on and off. The button needs to be pressed and held for 2 seconds 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 LiFePO4wered/Pi+™ cannot respond to button pushes 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 LiFePO4wered/Pi+™ is disconnected or fails, the system will keep running from the battery for up to 9 hours, depending on the LiFePO4wered/Pi+™ battery size, 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 LiFePO4wered/Pi+™ will instruct the Raspberry Pi to do a shutdown, and once the system is shut down properly, the power will be turned off.

If the user attempts to turn on the Raspberry Pi using 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 happens if the LiFePO4wered/Pi+™ detects that Raspberry Pi is already powered from another source (like its own power input).

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 button. It is possible to change this behavior and make the Raspberry Pi boot automatically when power returns by configuring the AUTO_BOOT setting. This is ideal for unattended systems that need to provide maximum uptime.

Safe boot and shutdown behavior based on the presence of external power is also possible using the AUTO_BOOT and AUTO_SHDN_TIME settings. This is a great way to make the LiFePO4wered/Pi+™ ensure the Pi always does clean shutdowns in systems where the user just wants to be able to flip a main power switch or unplug the system without having to worry about manually shutting down first.

Since the LiFePO4wered/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 LiFePO4wered/Pi+™ can be forced off by pressing and holding the touch button for approximately 10 seconds.

Limitations

Power source dependence

The LiFePO4wered/Pi+™ was designed to provide continuous UPS functionality for a Raspberry Pi 3 Model B+ under high load and with peripherals attached. The default 18650 battery option is not dependent on the input source for providing continuous load current up to 2 A to the Pi and peripherals, but since it isn't 100% efficient, it still needs to have more energy coming in than going out to the Pi to maintain the battery level.

In addition to this, the 14500 battery option *is* dependent on the power source for instantaneous load current, because of the higher internal resistance of the small cell and higher current sense resistor used in the circuit. At high loads, most power needs to be provided directly from the power source through the LiFePO4wered/Pi+™ power chain for this option.

The charger automatically adjusts its charge current to the power available from the connected source. This means you can power the LiFePO4wered/Pi+™ from a PC USB port for instance, but the charge current will be reduced to 0.5 A typically and the system won't be able to keep the battery charged under the high load. It is the user's responsibility to provide a power source sufficient to keep up with the expected system load. Many USB power supplies available on the market will *not* be able to keep up with the load presented by the LiFePO4wered/Pi+™ if it is under the maximum 2 A load, due to losses in the circuit and typical voltage sag of these supplies. Even many 2.5 A chargers on the market will show significant voltage drop at high current levels, causing the LiFePO4wered/Pi+™ to limit the input current so as not to damage the power source. For this reason, using a quality 3 A rated power supply is recommended.

Another matter of concern is the USB cable used. There are unfortunately many low-quality micro USB cables on the market, often using very thin wire. When drawing high currents through such a cable, a substantial amount of voltage will be dropped and the LiFePO4wered/Pi+™ will again limit the current to not make the input voltage drop below the MPPT threshold. This can significantly limit the charge current even when a good power supply is used. It is the user's responsibility to use a high quality cable for maximum performance. At high system loads, it is recommended to connect power directly to the VIN pads with large gauge wire instead of using the micro USB port, to avoid wiring

and contact losses.

The LiFePO4wered/Pi+™ can be powered from a wide input voltage range of 5 V to 20 V. Some of the issues with losses in wiring can be alleviated by powering the LiFePO4wered/Pi+™ from a higher input voltage. Note however that performance without active cooling will decrease at high input voltages, due to the extra power dissipation and resulting heat at these high voltages.

Heat management

The LiFePO4wered/Pi+™ is characterized to be able to provide a continuous load current of 2 A at 5 V nominal output, without active cooling. This testing was done in open air at room temperature. If the LiFePO4wered/Pi+™ is built in to a case with the Raspberry Pi and other loads consuming 10 W of power, it should be obvious that the temperature inside the case will rise quickly. It is the user's responsibility to work out an appropriate thermal design for their system. If the LiFePO4wered/Pi+™ is heated by other sources and enclosed in a case, active cooling is likely necessary at high power levels.

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 LiFePO4wered/Pi+™:

- Raspberry Pi 3 with 4 cores @ 100% load + Ethernet: 1 hour
- Raspberry Pi 3 idle + WiFi: 3 hours
- Pi Zero idle: 9 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 LiFePO4wered/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 LiFePO4wered/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 by touching grounded metal before interacting with the LiFePO4wered/Pi+™.

Bidirectional load switch

The LiFePO4wered/Pi+™ incorporates a bidirectional load switch which will protect it 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 LiFePO4wered/Pi+™ is powered (the battery is present). Applying power to the Raspberry Pi with the LiFePO4wered/Pi+™ connected but the battery removed will expose the LiFePO4wered/Pi+™ to voltages that can cause permanent damage.

Another scenario that causes damage to the LiFePO4wered/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.

In general, the safest thing is to *avoid powering the Raspberry Pi from any other source when the LiFePO4wered/Pi+™ is connected*. This will also ensure the LiFePO4wered/Pi+™ has full control over the power to the Raspberry Pi and can properly do its job as a UPS.

Raspberry Pi 4 compatibility

The LiFePO4wered/Pi+™ with 18650 battery is fully compatible with the Raspberry Pi 4, as long as the continuous total system load is 2 A or less (higher peak currents can usually be handled). Since the Raspberry Pi 4 under full system load takes a good amount of this available power, it is up to the user to evaluate whether it will work in their system, keeping in mind added loads such as a fan or USB peripherals.

Hardware connections

This section describes the hardware connections available on the LiFePO4wered/Pi+™.

Micro-B USB connector

This is the default external power input to the system when using the LiFePO4wered/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 LiFePO4wered/Pi+™. Pins not listed are unconnected:

Pin	Name	Use
2, 4	5V	Power output of the LiFePO4wered/Pi+™ to provide power to the Raspberry Pi.
3	GPIO2 (SDA1)	I2C bus data signal, used by the Raspberry Pi to communicate with the LiFePO4wered/Pi+™. Since it is a bus, it can be shared with other devices. Note that some HATs erroneously add pull-up resistors to the I2C bus which can cause issues with logic levels. The Raspberry Pi as I2C 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 LiFePO4wered/Pi+™ is used in combination with other HATs, please check whether they have this issue.
5	GPIO3 (SCL1)	I2C bus clock signal, used by the Raspberry Pi to communicate with the LiFePO4wered/Pi+™. Since it is a bus, it can be shared with other devices. Note that some HATs erroneously add pull-up resistors to the I2C bus which can cause issues with logic levels. The Raspberry Pi as I2C 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 LiFePO4wered/Pi+™ is used in combination with other HATs, please check whether they have this issue.
6, 9, 14, 20, 25, 30, 34, 39	GND	Ground reference for power output and all signals.
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 LiFePO4wered/Pi+™ does not use this signal as a UART, it only monitors the logic level during the shutdown phase. The UART functionality is completely available to the user just as if the LiFePO4wered/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 LiFePO4wered/Pi+™ from correctly detecting that the system has finished shutdown. In that case, power will be turned off by timeout.

In short, the presence of the LiFePO4wered/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 available to the user application.

User through-hole solder pads

The LiFePO4wered/Pi+™ was designed to make it easy to customize its functionality and use it as a component in a larger system. To facilitate this, large through-hole pads are available for making external connections. These are the connection pads available to the user:

Name	Use
BAT	Battery voltage. This is where the user connects their external 1S LiFePO4 battery pack when the "no on-board battery" option was chosen. The "+" mark indicates the positive terminal. It is recommended to have a high-current connector in the battery leads, so the wires can be soldered to the board without the actual battery connected. If this is not possible, be very careful to not short the battery terminals while soldering the wires to the board. If the LiFePO4wered/Pi+™ does come with a battery on board, these connections provide direct access to the battery voltage. It is not recommended to connect any load to these pads because they are directly connected to the battery and as such provide no short circuit or over-discharge protection.
VOUT	Output voltage. This is the switched 5 V output voltage to the Raspberry Pi. The "+" mark indicates the positive terminal. A user can power other 5 V loads from these pads and they will be switched on and off together with the Raspberry Pi. For example, a case fan can be connected here.
VBSW	Switched battery voltage. This is an auxiliary switched output of the battery voltage (3.2 V nominal). The "+" mark indicates the positive terminal. External 3.3 V circuitry can be powered from these pads and it will be switched on and off together with the Raspberry Pi. Unlike the BAT connections, this output has short circuit and over-discharge protection.
VIN	Input voltage. This is where the user can permanently connect the input power source instead of using the micro USB connector. The "+" mark indicates the positive terminal. Note that these are directly connected to the micro USB connector pins, so the USB input voltage can be measured here if the micro USB is used as power input. You should only ever use one of them at a time: either the micro USB or these pads. It is recommended to use these pads instead of the micro USB in high power systems, since micro USB connectors were not designed to handle high currents.
BTN	External button. This is where the user can connect an external momentary switch to duplicate the functionality of the on-board push button. The "+" mark indicates the positive terminal (polarity is important if using a transistor instead of mechanical switch). If the LiFePO4wered/Pi+™ is switched to touch pad mode, a capacitive touch pad can be connected to the pad marked with "+" and the other pad is not used. Keep wiring as short

	as possible in touch pad mode.
PWLED	Power LED. The user can connect an external LED to these terminals to duplicate the functionality of the on-board PWR LED. The "+" mark indicates the positive terminal. It is necessary to add a current limiting resistor in series with the external LED. The current limiting LED is not integrated on the board because it needs to be customized to give the external LED the desired brightness.
CHLED	Charge LED. The user can connect an external LED to these terminals to duplicate the functionality of the on-board CHRG LED. The "+" mark indicates the positive terminal. It is necessary to add a current limiting resistor in series with the external LED. The current limiting LED is not integrated on the board because it needs to be customized to give the external LED the desired brightness.
MPP	Maximum Power Point setting. This connection will allow the user to customize the maximum power point (MPP) voltage by adding a resistor across the pads. The charger will reduce the charge current as needed to ensure the input voltage does not drop below the MPP voltage. The "+" mark indicates the positive terminal. By default the MPP voltage is 4.66 V nominal. This is what limits the input current so a weak USB port does not get overloaded. If the user connects a solar panel as input source, a resistor needs to be added to optimize for the panel's MPP voltage. The MPP resistor value can be calculated with the following formula: $R_{MPP} = 51815 / (V_{MPP} - 4.66)$

User surface mount pads

Name	Use
SW	Switch. The LiFePO4wered/Pi+™ by default is shipped with a side-facing on/off button mounted on the bottom of the board. However on the top of the board are two footprints for commonly available surface mount momentary switches that can be added by the user if an up-facing button is desired.

Test pads and configuration solder jumpers

The LiFePO4wered/Pi+™ has the following test pads on the bottom of the circuit board:

Name	Use
V	Battery voltage. Used during production.
C	Microcontroller programming clock. Used during production.
D	Microcontroller programming data. Used during production.
G	Ground reference for all signals.
SDA	I^2C bus data. Used during production.

SCL	I²C bus clock. Used during production.
TX	UART TX signal detect. Used during production.
1.5A	1.5A charge mode. This solder jumper sets the maximum charge current. For the 14500 size battery, the maximum nominal charge current is 0.57 A and the solder jumper should be open. For the 18650 size battery, the maximum nominal charge current is 1.5 A and the solder jumper should be bridged with solder. If no battery is installed and the user adds their own pack, it is the user's responsibility to configure this jumper to select a charge current appropriate for the connected battery pack.

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 raw register values into convenient units such as millivolts, milliampères 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 calculation, 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 I²C 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, an unlock code needs to be send next. 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.

The LiFePO4wered/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.

I²C read access

To read data from the LiFePO4wered/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 I²C register specification

The following I²C registers are available in the LiFePO4wered/Pi+™:

I2C_REG_VER

1 byte, register address 0x00, read only access

Value: 0x07

This value specifies an I²C register set version. It allows the client software to choose the correct register addresses, since the same software is used for various LiFePO4wered products.

I2C_ADDRESS

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

Default value: 0x43

7-bit bus address of the LiFePO4wered/Pi+™ device. If this is changed, the host software on the

Raspberry Pi needs to be changed (`I2C_ADDRESS` definition in the `lifepo4wered-access.c` source file) 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 in the on state. The LED is under control of the LiFePO4wered/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 0x3A, read only

This register indicates the touch/press state of the button. It is called TOUCH_STATE to maintain compatibility with older LiFePO4wered products. The value is 0 if the button is not currently pressed. The value is nonzero when indicating button press states. Press and hold of the button for 2 seconds will make the Raspberry Pi turn off, but short button press events can be interpreted by user code. The 4 lowest bits of this byte indicate the last 4 button samples, shifting from the low to the high bit. For instance, a value of 0x01 indicates the user just started pressing the button, while 0x0E indicates the button 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: 0

By default, with a register value of 0, the button handler is configured in “mechanical switch” mode to use the on-board (or an added external) mechanical switch as on/off button. However, the button handler can be switched to “capacitive touch” mode by setting this register in case the user chooses to connect an external capacitive touch area.

To configure capacitive touch, this register must be set to a non-zero value that indicates the total number of charge and discharge cycles generated and measured by the touch detection subsystem. A good starting value for this register is 20, but the value can be customized as needed to optimize the sensitivity of the connected touch pad.

TOUCH_THRESHOLD

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

Default value: 12

When the button handler is in capacitive touch mode, 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.

When the button handler is in mechanical switch mode, this register should be left at the default value to ensure correct button operation.

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.

This register may only be customized when the button handler is in capacitive touch mode. When the button handler is in mechanical switch mode, this register should be left at the default value to ensure correct button operation.

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 MSP430G2332 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

MSP430G2332 datasheet for more details. The user should not need to change this value.

VIN

2 bytes little endian, register address 0x36, read only

Value: input voltage, 13-bit value, 25.37 V full scale, resolution of 3.1 mV per LSB

This value represents the input (USB) voltage. The Raspberry Pi host software package contains scaling code so the value is converted to mV for convenience. For instance, the following command reads the input voltage as 5004 mV or 5.004 V.

```
# lifepo4wered-cli get VIN
5004
```

VBAT

2 bytes little endian, register address 0x32, read only

Value: 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. For instance, the following command reads the battery voltage as 3606 mV or 3.606 V.

```
# lifepo4wered-cli get VBAT
3606
```

VOUT

2 bytes little endian, register address 0x34, read only

Value: output voltage, 13-bit value, 5.256 V full scale, resolution of 0.64 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. For instance, the following command reads the output voltage as 5036 mV or 5.036 V.

```
# lifepo4wered-cli get VOUT
5036
```

IOUT

2 bytes little endian, register address 0x38, read only

Value: output (load) current, 13-bit value, 5.814 A full scale, resolution of 0.71 mA per LSB

This value represents the output (load) current of the 5V output powering the Raspberry Pi. The

Raspberry Pi host software package contains scaling code so the value is converted to mA for convenience. For instance, the following command reads the output current as 675 mA.

```
# lifepo4wered-cli get IOUT
675
```

Note that this measurement is approximate and not linear. For more information, see “CC Pin Voltage” curves in the TI TPS61236P datasheet.

VBAT_MIN

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

Default value: 4665 (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 LiFePO4wered/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 (VBAT_SHDN), but in case the Raspberry Pi is unresponsive and 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. For instance, the following command reads the minimum battery voltage as 2850 mV or 2.85 V.

```
# lifepo4wered-cli get VBAT_MIN
2850
```

VBAT_SHDN

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

Default value: 4829 (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. For instance, the following command reads the shutdown battery voltage as 2950 mV or 2.95 V.

```
# lifepo4wered-cli get VBAT_SHDN
2950
```

VBAT_BOOT

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

Default value: 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 once the load is removed. Under heavy load, it may be necessary to increase this value to prevent continuous boot / shutdown cycling when the battery starts to be depleted.

The Raspberry Pi host software package contains scaling code so the value can be read and set in mV for convenience. For instance, the following command reads the boot battery voltage as 3150 mV or 3.15 V.

```
# lifepo4wered-cli get VBAT_BOOT
3150
```

VOUT_MAX

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

Default value: 5449 (corresponding to 3.5 V, resolution of 0.64 mV per LSB)

This value determines the minimum output voltage present for which the LiFePO4wered/Pi+™ will refuse to boot the Raspberry Pi when it is supposed to be off (according to the state maintained in the LiFePO4wered/Pi+™). The LiFePO4wered/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 power connector) with the LiFePO4wered/Pi+™ attached without causing 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 LiFePO4wered/Pi+™). Because the LiFePO4wered/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. For instance, the following command reads the maximum output voltage to allow boot as 3500 mV or 3.5 V.

```
# lifepo4wered-cli get VOUT_MAX
3500
```

VIN_THRESHOLD

2 bytes little endian, register address 0x10, read/write, saved to flash

Default value: 1451 (corresponding to ~4.5 V, resolution of 3.1 mV per LSB)

This value determines the input (USB) voltage level at which the Raspberry Pi will be booted, if the battery voltage is also high enough (VBAT_BOOT threshold), and the AUTO_BOOT register is set to option 3 or 4. It also determines the input voltage level at which the input is considered to be missing and the Raspberry Pi will be shut down after AUTO_SHDN_TIME minutes if this value is set.

The Raspberry Pi host software package contains scaling code so the value can be read and set in mV for convenience. For instance, the following command reads the input voltage threshold as 4498 mV or 4.5 V (rounded).

```
# lifepo4wered-cli get VIN_THRESHOLD
4498
```

IOUT_SHDN_THRESHOLD

2 bytes little endian, register address 0x1A, read/write, saved to flash

Default value: 0 (magic value to turn the feature off, other values resolution of 0.71 mA per LSB)

When this register is set to the default value of 0, the LiFePO4wered/Pi+™ determines whether the Raspberry Pi has finished shutdown by monitoring the UART TX line, which will go from a high to low logic level and stay there after shutdown has finished.

However, when using the LiFePO4wered/Pi+™ with other SBCs, this option is sometimes not available because the monitored GPIO pin doesn't behave the same on different hardware. Or the user may have reason to disable the UART on the Raspberry Pi. In these cases, shutdown detection by GPIO TX pin will not work, and by default the power will only be turned off after the PI_SHDN_TO time expires.

However, this register provides another option. If the power consumption of the board and peripherals when running is different enough compared to when shutdown has finished, the current level can be used to detect that shutdown has occurred. In that case this register can be set to a threshold current level between running and shutdown current.

The Raspberry Pi host software package contains scaling code so the value can be read and set in mA for convenience. For instance, the following command reads the output current threshold as 100 mA.

```
# lifepo4wered-cli get IOUT_SHDN_THRESHOLD
100
```

VBAT_OFFSET

2 bytes little endian, register address 0x12, read/write, saved to flash

Default value: factory calibrated (resolution of 4.88 mV per LSB)

This register provides a calibration value for the battery voltage measurements. It is a simple 1 point

offset calibration at 3.2V nominal that provides compensation for inaccuracy of the internal reference voltage and resistive divider.

The Raspberry Pi host software package contains scaling code so the value can be read and set in mV for convenience. The user should not have to use this register, it is for factory calibration only.

VOUT_OFFSET

2 bytes little endian, register address 0x14, read/write, saved to flash

Default value: factory calibrated (resolution of 5.14 mV per LSB)

This register provides a calibration value for the output voltage measurements. It is a simple 1 point offset calibration at the zero load output voltage that provides compensation for inaccuracy of the internal reference voltage and resistive divider.

The Raspberry Pi host software package contains scaling code so the value can be read and set in mV for convenience. The user should not have to use this register, it is for factory calibration only.

VIN_OFFSET

2 bytes little endian, register address 0x16, read/write, saved to flash

Default value: factory calibrated (resolution of 24.8 mV per LSB)

This register provides a calibration value for the input voltage measurements. It is a simple 1 point offset calibration at 5V nominal that provides compensation for inaccuracy of the internal reference voltage and resistive divider.

The Raspberry Pi host software package contains scaling code so the value can be read and set in mV for convenience. The user should not have to use this register, it is for factory calibration only.

IOUT_OFFSET

2 bytes little endian, register address 0x18, read/write, saved to flash

Default value: 0 (corresponding to 0 mA, resolution of 0.71 mA per LSB)

This register provides a calibration value for the output current measurements. It is a simple 1 point offset calibration value that by default is 0. It is not factory calibrated because, due to the non-linearity of the IOUT reading, calibration should be performed in the region of interest. The user can use this register to perform a custom calibration value suitable for their application.

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

AUTO_BOOT

1 byte, register address 0x20, 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 presses the on/off button 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, since the Raspberry Pi will run whenever possible.

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 button or from a user program.

Setting this register to 3 (AUTO_BOOT_VIN) will make the Raspberry Pi boot if sufficient battery voltage is available (VBAT >= VBAT_BOOT threshold) and the input voltage is also present (VIN >= VIN_THRESHOLD). This can be useful in conjunction with the auto shutdown feature to boot and properly shut down the Raspberry Pi by using a flip switch in series with the input power. It can also boot a solar powered system when the sun comes up.

Setting the register to 4 (AUTO_BOOT_VIN_SMART) enables the smart version of the previous setting, which still allows the user to shut down the unit manually as described above.

Setting this register to 5 (AUTO_BOOT_NO_VIN) will make the Raspberry Pi boot if sufficient battery voltage is available (VBAT >= VBAT_BOOT threshold) and there is no input voltage present (VIN < VIN_THRESHOLD). This is useful in cases where the Raspberry Pi needs to take action when external power goes away, for instance, when monitoring the health of another wall powered system.

Setting this register to 6 (AUTO_BOOT_NO_VIN_SMART) enables the smart version of the previous setting. This allows the Raspberry Pi to shut down and stay off while external power is still missing, after it has taken care of what it needs to do. Only after external power comes back and then goes away again will it automatically boot the system next.

WAKE_TIME

2 bytes little endian, register address 0x26, 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 in the off state before the LiFePO4wered/Pi+™ will automatically boot it again. The timing is based on a 32kHz watch crystal. 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 button presses and the AUTO_BOOT setting as usual when the wake timer is set.

A user program can check the value of this register 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, such as AUTO_BOOT or a button press. For instance, the following command reads the wake time remaining as 7 minutes.

```
# lifepo4wered-cli get WAKE_TIME
7
```

SHDN_DELAY

2 bytes little endian, register address 0x1C, read/write, saved to flash

Default value: 40

This sets the number of LiFePO4wered/Pi+™ system ticks that elapse between when the Raspberry Pi is shut down (detected by the UART TX line going low or output current dropping below the set threshold) and when the power to it is turned off. There are 8 ticks per second so the default delay is around 8 seconds, but this is not accurate because the counter is maintained in software in the task manager. The default value is chosen to allow plenty of time between shutdown and power off for the Raspberry Pi's LED light blink sequence to happen so users don't become concerned that shutdown didn't finish correctly. The user can reduce this value to attain maximum run time on battery power in low power systems using the wake timer.

Another possible use for changing this value is when the Raspberry Pi 3 and newer is configured to disable the UART on the GPIO header. This is actually the default state on recent Raspberry Pis, however the LiFePO4wered/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 LiFePO4wered/Pi+™ daemon is unloaded until it's loaded again.

It has been observed that when using the NOOBS distribution, the recovery mode screen on boot can cause a long enough delay to prevent reboots from working correctly. It prevents the kernel from running quickly enough to pull the UART TX line high again before the shutdown delay timer runs out and the system is shut off instead of rebooted. When using NOOBS, it is recommended to increase this register value to 100 so reboots work correctly.

Since this is a deeply technical register, no scaling is provided by the host tools. It's assumed the user who changes it knows what they're doing.

AUTO_SHDN_TIME

2 bytes little endian, register address 0x1E, read/write, saved to flash

Default value: 0xFFFF

Auto shutdown is a feature that will shut down the Raspberry Pi if the input (USB) voltage falls below the VIN_THRESHOLD level. The shutdown will happen after a delay in minutes specified in this register. The default value of 0xFFFF disables auto shutdown, which will allow the system to run on battery power for as long as battery power is available. When this value is set to 0, immediate shutdown is triggered when the input power is removed. Since in some situations where auto shutdown is used it may be more desirable to keep the system running through short power interruptions, a run time on battery power can be set in minutes. For instance, the following command can be used to keep the system running on battery power for 10 minutes after external power fails.

```
# lifepo4wered-cli set AUTO_SHDN_TIME 10
10
```

Note that if the battery is too depleted, or if this value is set too high, shutdown may still occur sooner than indicated by this register since VBAT check against the VBAT_SHDN threshold will take precedence.

PI_BOOT_TO

1 byte, register address 0x21, read/write, saved to flash

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

The LiFePO4wered/Pi+™ will not indefinitely stay in the boot state 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. For instance, the following command reads the boot timeout as 300 s.

```
# lifepo4wered-cli get PI_BOOT_TO
300
```

PI_SHDN_TO

1 byte, register address 0x22, read/write, saved to flash

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

The LiFePO4wered/Pi+™ will not indefinitely stay in the shutdown state 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 LiFePO4wered/Pi+™ compatible with more single board computers that use the Raspberry Pi form factor and compatible GPIO header. Many of these boards do not bring the UART TX line low on shutdown, preventing shutdown detection (unless IOUT_SHDN_THRESH can be used). 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. For instance, the following command reads the shutdown timeout as 120 s.

```
# lifepo4wered-cli get PI_SHDN_TO
120
```

RTC_TIME

4 bytes, register address 0x28, read/write, not saved to flash

Default value: counting up every second

The LiFePO4wered/Pi+™ uses a 32 kHz watch crystal as a time base, and implements this 32-bit register to store and track real time in the form of a Unix time stamp (seconds since midnight Jan 1 1970), even when the host system is off. When first powered, this register will start counting from zero. The daemon automatically manages saving the host system time to this register on shutdown, and restoring system time from this register on boot if it makes sense to do so. The logic implemented in the daemon is designed to make the RTC functionality automatic and compatible with other (possibly more accurate) time sources such as network time and temperature compensated RTC modules. There is no need for the user to manage this register directly.

RTC_WAKE_TIME

4 bytes, register address 0x2C, read/write, not saved to flash

Default value: 0

With this register the user can specify an absolute time for the host system to be woken up. The wake time is specified as a 32-bit Unix time stamp (seconds since midnight Jan 1 1970).

The LiFePO4wered/Pi+™ implements smart logic to ensure wake up requests are not missed, which could otherwise cause the system to stay off indefinitely in certain race conditions. For instance, if the wake time was set to a time in the past, or of a wake time was set and then the system didn't shut down in time before the wake time had passed, a check that would just compare whether the wake time is equal to the RTC time would never wake the host up.

Instead, the LiFePO4wered/Pi+™ checks if the current RTC time is greater than or equal to the wake time. This way, the host will reboot immediately after shutdown if the wake time was missed. The wake time register will automatically be reset to 0 if it causes the system to be booted. This prevents erroneous repeated wake ups, yet preserves the ability to set a wake time in the future while other events may cause shutdowns and boots before the wake time.

For command line use, the date utility can be used in combination with the lifepo4wered-cli tool to conveniently set wake times. For instance, the following command will wake the Raspberry Pi at 5 PM next Friday.

```
# lifepo4wered-cli set RTC_WAKE_TIME `date +%s -d "5 pm next friday"`
1599865200
```

WATCHDOG_CFG

1 byte, register address 0x23, read/write, saved to flash

Default value: 0x00

The LiFePO4wered/Pi+™ implements a watchdog feature that can be used to ensure the user's 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 LiFePO4wered/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 0x24, read/write, saved to flash

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

This sets the watchdog grace period from when the host daemon informs the LiFePO4wered/Pi+™ that the system has booted, until the user application finally has its first 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. For instance, the following command reads the watchdog grace period as 20 s.

```
# lifepo4wered-cli get WATCHDOG_GRACE
20
```

WATCHDOG_TIMER

1 byte, register address 0x30, 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

LiFePO4wered/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 LiFePO4wered/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. For instance, the following command sets the watchdog timeout to 45 s.

```
# lifepo4wered-cli set WATCHDOG_TIMER 45
45
```

PI_RUNNING

1 byte, register address 0x31, 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 LiFePO4wered/Pi+™ itself and the LiFePO4wered/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 LiFePO4wered/Pi+™ daemon starts, it sets this flag to 1 to indicate the Raspberry Pi has booted. This will change the state of the LiFePO4wered/Pi+™, the PWR LED will go from pulsing to on state, and will be ready for user input (pressing the button to turn the Raspberry Pi off again). This flag can be cleared by various sources, such as a user pressing the 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 LiFePO4wered/Pi+™ will show this by pulsating the PWR LED and the power will be turned off when shutdown is finished.

As mentioned, the user does not need to worry about manually controlling this flag, the LiFePO4wered/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 0x25, 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 LiFePO4wered/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 LiFePO4wered/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 LiFePO4wered/Pi+™ I2C 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. For instance, to set the wake up timer to an hour, run:

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

When you shut down the Raspberry Pi, it will wake up again in 60 minutes. Using the RTC, you can also cause the Pi to wake up at an absolute time. Instead of using the Unix time stamp directly, we can combine the lifepo4wered-cli command with other command line tools to make things easier. The following will wake the Pi up at 10:00 pm today.

```
# lifepo4wered-cli set rtc_wake_time `date -d "10:00 pm" +%s`
```

Values can be provided in decimal notation or hexadecimal notation by using the `0x` prefix. For instance, if you want the Raspberry Pi to boot whenever the USB input voltage is applied, you can run:

```
# lifepo4wered-cli set auto_boot 0x03
```

Please refer to the I2C 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.

The user running the tool needs to have sufficient permissions to access the I²C bus. On Raspbian, the `pi` user by default can access the bus because it is in the `i2c` group. If you run as a different user, you either need to add this user to the `i2c` group or run the tool with `sudo`. On other distributions, a different group name may be used. You can check the owner and group of the I²C device with:

```
# ls -l /dev/i2c-1
```

The command line tool returns the following negative values to indicate error conditions:

Return value	Condition
-1	Could not access the LiFePO4wered/Pi+™ to perform the specified operation. Usually this condition is caused by insufficient privileges when trying to access the I²C bus. Trying to run the command as `root` or with `sudo` to fix the problem. When writing settings, this value is also returned if a register is not writable.
-2	The I²C bus could be accessed and the operation is valid, but communication with the LiFePO4wered/Pi+™ failed. After trying several times (20 by default), the LiFePO4wered/Pi+™ I²C bus transaction could not be completed successfully. This happens if the LiFePO4wered/Pi+™ is not physically present or if something (possibly another HAT) is preventing the I²C bus from operating correctly.

Electrical characteristics

Unless otherwise indicated, all characteristics apply for V_{IN} = 4.85 V to 5.15 V and T_A = 0 ºC to 50 ºC. Typical values are at 25 ºC and V_{IN} = 5 V.

Parameter	Sym	Min	Typ	Max	Unit	Conditions
Input voltage	V_{IN}	4.85	5.0	20.0	V	
Battery leakage current	$I_{DISCHARGE}$		4		µA	Input voltage absent, Raspberry Pi powered off
Battery charge current	I_{CHARGE}		1.5		A	1.5A solder bridge closed, for use with 18650 or bigger battery
Battery charge current	I_{CHARGE}		0.57		A	1.5A solder bridge open, for use with 14500 battery
Continuous output current externally powered	I_{OUT_EXT}		2.0		A	If using 14500 battery option, power source needs to be solid (not sag under

						load)
Continuous output current on battery	I_{OUT_18650}		2.0		A	18650 battery installed, battery fully charged
Continuous output current on battery	I_{OUT_14500}		0.75		A	14500 battery installed, battery fully charged
Output voltage	V_{OUT}	4.7	5.0	5.2	V	
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	$VOUT_{MAX}$		3.50		V	Raspberry Pi is powered from another source
Default input threshold voltage	VIN_{THRS}		4.50		V	For auto boot and auto shutdown
Maximum Power Point threshold voltage (default)	VIN_{MPP}		4.65	4.8	V	Charge current will be limited to prevent V_{IN} from dropping below threshold

Safety information

The LiFePO4wered/Pi+™ uses $LiFePO_4$ (lithium iron phosphate) batteries which are known for their reliability, long life and safety compared to most other lithium-ion chemistries. They also have very high power density, which is required for a high performance product like this, but which also requires precautions to safely handle and use the product. Here are some things to keep in mind:

- While the LiFePO4wered/Pi+™ provides overload and short circuit protection on its VOUT and VBSW outputs, the BAT connections are directly connected to the battery without any protection. The BAT connections are *only* intended for connecting an external battery, please do not use them to connect loads to an on-board battery, but use the protected VBSW output for this instead.
- In order to be able to offer the highest performance in normal operation, the battery connections (BAT pads and on-board battery holder) do not include reverse battery protection. When connecting an external battery or inserting a battery into the on-board holder, please take care to

ensure correct battery polarity. Connecting the battery in reverse will cause high current flow and destroy the device.

- The LiFePO4wered/Pi+™ is only designed to work with 1S LiFePO4 cells. Do not connect higher voltage battery packs. Do not use any other type of battery with the device. The maximum voltage across the battery terminals is 3.65V.
- The LiFePO4wered/Pi+™ is always powered when a battery is connected. If it is necessary for the user to perform work on the unit such as soldering external wiring to any of the pads, it is highly recommended to remove / disconnect the battery first.
- Never place the unit on a conductive or metal surface. As mentioned, the unit is powered from a high power cell and damage can occur if any parts of the device get shorted. Take special care to avoid anything that could cause a short across the battery terminals, as the battery is capable of sustaining very high currents, leading to excessive heat production or possibly fire.
- Never expose the unit to water, fire, excessive heat or chemicals.

We have taken every precaution to provide a high performance, safe product but do not accept any liability or responsibility for your use of it. It is your responsibility as a customer to use the device in a proper and sensible way, and show proper respect for the high amount of energy present. By using the device you agree that Silicognition LLC will not be held liable for any damages you may incur due to your use of the device.

Sales and support

To buy the LiFePO4wered/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.

X-ON Electronics

Largest Supplier of Electrical and Electronic Components

Click to view similar products for Power Management IC Development Tools ***category:***

Click to view products by Crowd Supply ***manufacturer:***

Other Similar products are found below :

EVB-EP5348UI MIC23451-AAAYFL EV MIC5281YMME EV 124352-HMC860LP3E DA9063-EVAL ADP122-3.3-EVALZ ADP130-0.8-EVALZ ADP130-1.8-EVALZ ADP1740-1.5-EVALZ ADP1870-0.3-EVALZ ADP1874-0.3-EVALZ ADP199CB-EVALZ ADP2102-1.25-EVALZ ADP2102-1.875EVALZ ADP2102-1.8-EVALZ ADP2102-2-EVALZ ADP2102-3-EVALZ ADP2102-4-EVALZ AS3606-DB BQ25010EVM BQ3055EVM ISLUSBI2CKIT1Z LM2734YEVAL LP38512TS-1.8EV EVAL-ADM1186-1MBZ EVAL-ADM1186-2MBZ ADP122UJZ-REDYKIT ADP166Z-REDYKIT ADP170-1.8-EVALZ ADP171-EVALZ ADP1853-EVALZ ADP1873-0.3-EVALZ ADP198CP-EVALZ ADP2102-1.0-EVALZ ADP2102-1-EVALZ ADP2107-1.8-EVALZ ADP5020CP-EVALZ CC-ACC-DBMX-51 ATPL230A-EK MIC23250-S4YMT EV MIC26603YJL EV MIC33050-SYHL EV TPS60100EVM-131 TPS65010EVM-230 TPS71933-28EVM-213 TPS72728YFFEVM-407 TPS79318YEQEVM ISL85033EVAL2Z UCC28810EVM-002 XILINXPWR-083