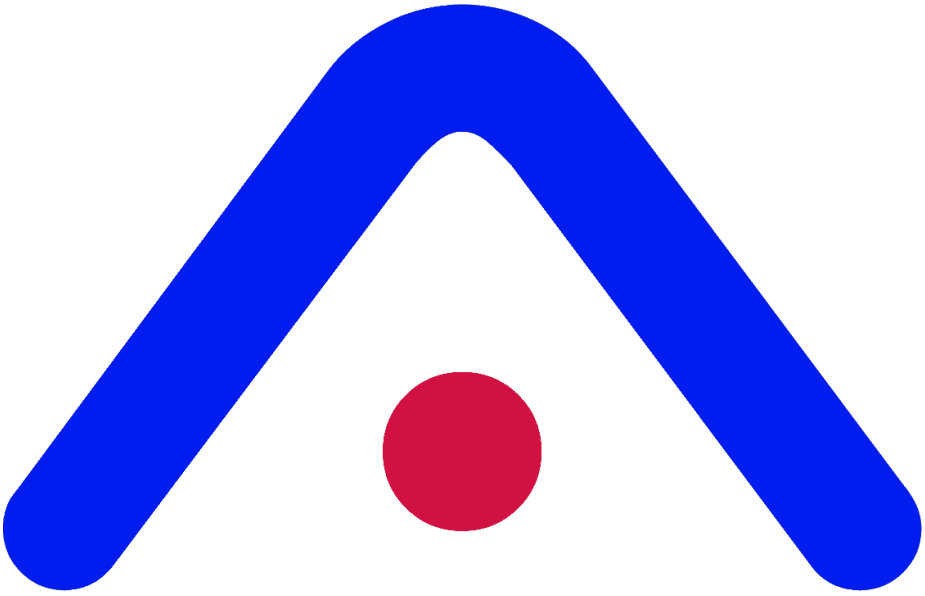



# PIXIE BOARD
# POE-PSU POWER SUPPLY USER MANUAL

AEL Microsystems Ltd
Malvern
UK

Acknowledgements:

AEL Microsystems Ltd acknowledges the trademarks of other organisations for their respective products and services mentioned in this document.

This contact information is subject to change, for the latest details go to www.aelmicro.com

Web: https://www.aelmicro.com
Email: pixie.support@aelmicro.com

Phone: +44 (0)1886 881059

# 1. Contents

# 2. Introduction

## 2.1 Caution

This board is designed using modern CMOS devices, observe standard anti-static procedures when handling this board otherwise permanent damage may result.

**You have been warned !**

## 2.2 Forward note

Thank you for choosing one of the **PI** e**X**pansion **I**ndustrial **E**lectronic boards, **"PIXIE"**.

The range of **PIXIE** boards has been developed to allow you to expand the hardware functionality of your Raspberry Pi, by adding one or more **PIXIE** boards gives you a wider range of interface solutions. The **PIXIE** range of boards has been developed to allow for the Raspberry Pi to be used in harsher industrial and real-world environments.

The key objective of a PIXIE board is:

- Provide an expansion board to allow the use of a Raspberry Pi in industrial environments.
- Allow for more than one expansion board to be stacked onto an existing Raspberry Pi unlike a HAT.
- 16 boards can be stacked and given a unique logical address using the board selector switches.
- Provides a low-cost industrial control solution.
- Standard board profile which is the same as the Raspberry Pi.
- Optional enclosure to allow mounting direct to industrial DIN rail.
- Fully software configurable, i.e. no links to set.
- Can use either SPI devices 0 or 1.
- Supported is provided for National Instruments LabVIEW.
- Comes with fully supported **PIXIE** software API and libraries, for 'C', 'C++' and Python

The **PIXIE** boards have not only been developed for use solely with the Raspberry Pi but can easily be interfaced to other microcontrollers and CPU modules allowing your project to be based on alternative platforms and operating systems.

All **PIXIE** boards use a standard size board and are connected to the Raspberry Pi board using the 40-way IDC connector.

Multiple **PIXIE** boards can be stacked on to the Raspberry Pi and once assembled can be configured using the **PIXIE** board configuration and update utility eliminating the need to dismantle the board stack to change the settings.

# 3. Common concept

## 3.1 Control overview

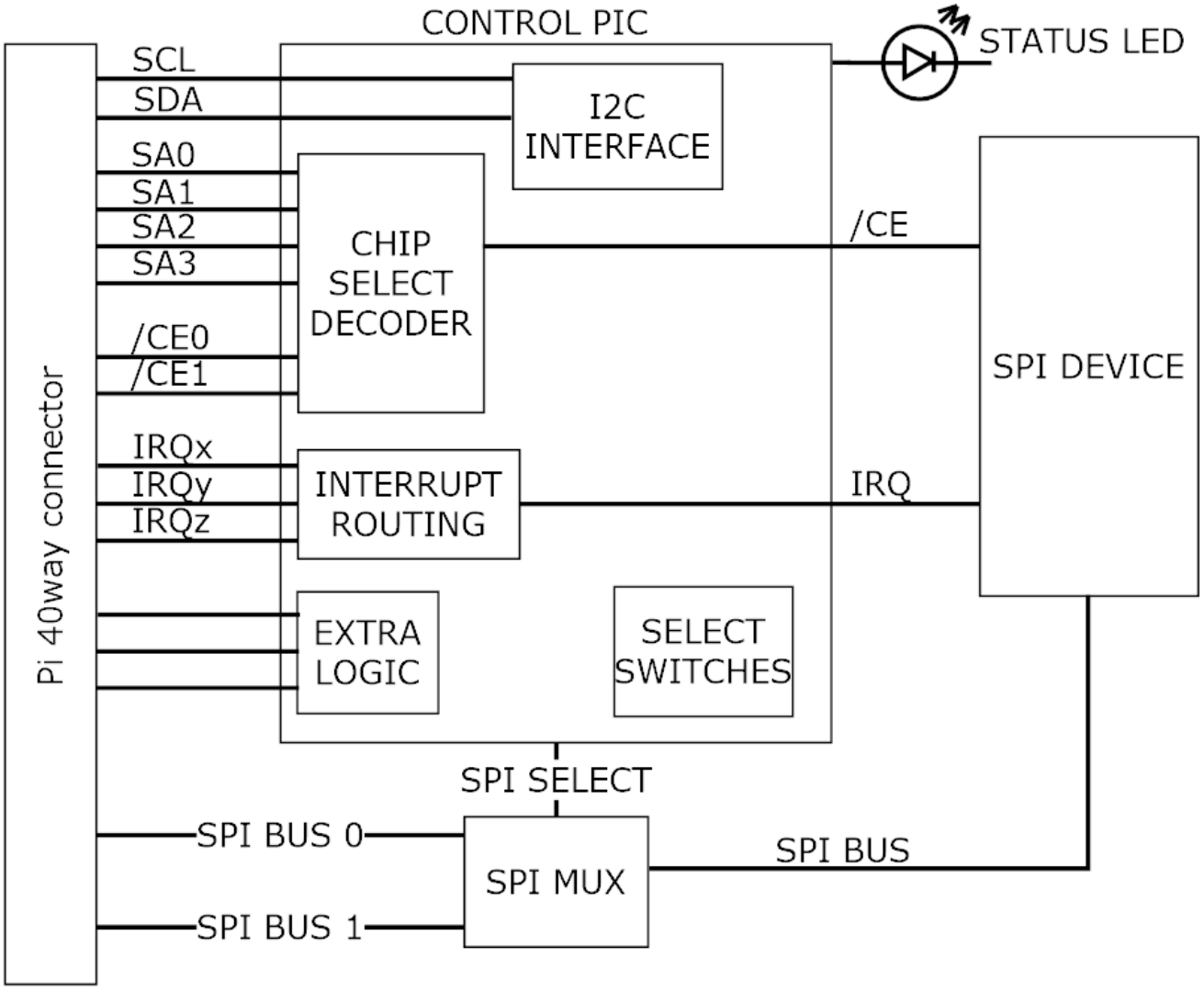
This shows the control overview of the **PIXIE** board.

The select switches give each **PIXIE** board a unique identity.

The Raspberry Pi communicates using the I2C bus to configure the **PIXIE** board.

The sub address (SAx) signals, interrupt signals (IRQx) and extra logic signals are connected to the GPIO pins of the Raspberry Pi.

Either SPI0 or SPI1 bus is routed to the board devices




The design goal of the **PIXIE** board range is to provide the user with a board that has a common footprint, uses no configuration links, and once assembled into a stack with the Raspberry Pi, can be configured and used without the need to make any further physical changes except for wiring in the connectors. All boards use 3.5mm two-part pluggable terminal blocks which in the event of a board change or other upgrade, can be simply unplugged without the need for a screwdriver.

**IT IS NOT A HAT**
The boards are not HAT's, their biggest difference is that you can stack up to 16 onto the Raspberry Pi and they all use the SPI busses for maximum software access, nor do they use the HAT configuration memory.

## 3.2 More SPI devices

This concept is achieved by the use of some of the GPIO signals to provide additional address signals used for decoding the SPI chip selects found on the 40-way connector. You can have up to 4 additional SPI address signals per SPI bus, these are qualified by the onboard hardware decoding to give you up to 16 possible decode addresses for each SPI bus chip select. So, for SPI0 it has 2 chip selects, that is 32 possible devices, for SPI1 it has 3 chip selects, that is 48 possible devices, 80 in total which is more than most needs.
Each board can be configured to use as many of the address signals as it requires as well as which chip select the board will use.
To facilitate this sub address system requires changes to the SPI device driver and rebuild the kernel or use the precompiled SPI device driver and install it on the Raspberry Pi to replace the current one.

## 3.3 Configurable

Each board is software configurable from the Raspberry Pi using a simple command line application called "**PixieBoard**". Each board is given a unique identity which is set by the small piano key switch allowing each board to be numbered 0 to 15. The board is configured over the I2C bus using a base address of 0x10 plus the value of the piano switch giving a range of unique I2C addresses from 0x10 to 0x1F. Each board can then be accessed individually and configured as required.
All the configuration values for each board are stored in EEPROM memory on the board so once it has been configured it does not have to be reloaded whenever the board is power cycled.

The key configurable items of each board are:

- Which SPI to use, SPI0 or SPI1
- Which chip SPI selects to use, CE0, CE1 or CE2
- Which SPI sub address signals to use and the sub address value to decode.
- Which GPIO will receive an interrupt if required from the board.
- Additional board specific settings.

## 3.4 Stackable

Each board can be stacked on top of each other and the Raspberry Pi using 17mm spacers or enclosed in one of the plastic housings which allows for the use in a more robust environment as well as mounting to a standard industrial DIN rail.
As previously mentioned up to 16 boards can be stacked together.

## 3.5 Updatable

All boards make use of a small microcontroller to interface to the Raspberry Pi over the I2C bus, and provide the real time hardware decoding logic and other board support functionality. If at any point new firmware is made available, the "**PixieBoard**" application can be used to update the boards firmware without the need to dismantle the board from the stack or use any external programme

# 4. Hardware details

The **POE-PSU** is a **PIXIE** board that can power of up to 4.5A @ +5V to the Raspberry Pi and other Pixie boards from either a POE+ power source or DC supply ranging from +9V to +48V.

In addition, it also includes a battery backed real time clock, a CPU cooling fan, current and voltage monitoring, control logic for using an additional external battery to provide a managed UPS solution, as well as other power management control capabilities.

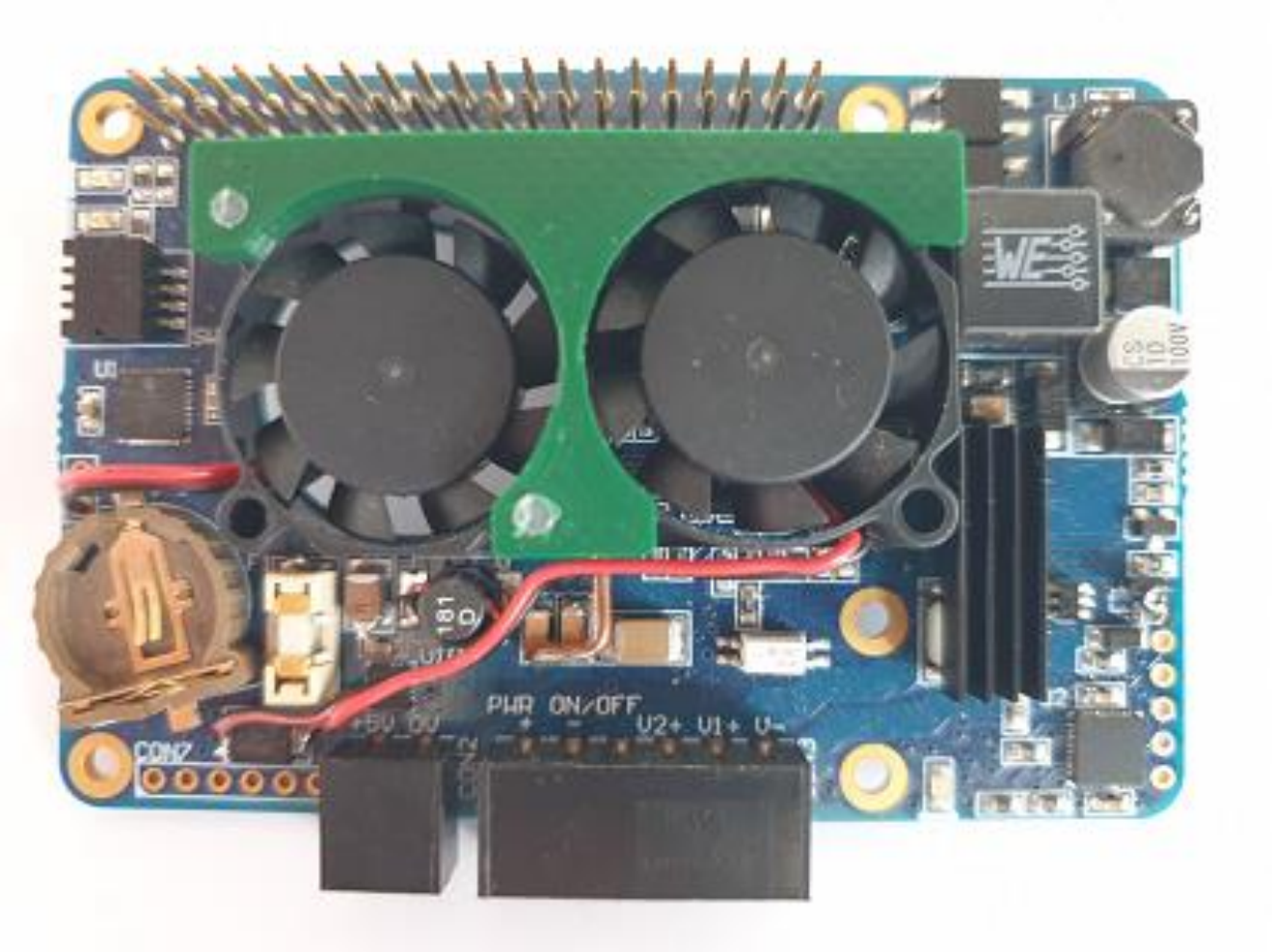
Key features:

- POE+ input supply.
- Auxiliary +9V to +48V DC supply.
- Output voltage protection with replaceable fuse.
- Software controlled secondary auxiliary +9V to +48V DC supply, (may be a battery).
- Supply monitoring of input and output voltages and output current.
- Enter or leave standby mode using external input.
- Enter or leave standby mode using software.
- Enter or leave standby mode using timed off and on periods of 1 second to 49710 days (136 years).
- Power cycle watchdog, will power cycle the output whenever a programmed watchdog timer is not reset, 1 to 255 seconds.
- Power fail detection interrupt to a GPIO pin.
- Board temperature sensor for controlling board cooling fan.
- CPU controlled fan, can be controlled using software or a selectable GPIO pin from the Raspberry Pi.
- Battery backed real time clock, I2C accessed.
- Board ID selectable switch

## 4.1 Specification.

| | |
|---|---|
| POE+ Voltage | +48V |
| DC Voltage range | +9V or +48V |
| Output protection | +5V and fused with replaceable fuse. |
| No load power consumption | 50mA |
| I2C speed | <=100kHz |
| Temperature | 0-70C |

## 4.2 I/O connections.




Above shows the connector positions and identifiers for the I/O signals.

**CON2** is a 2-way connector which can be used to output the +5V to other modules.
**CON3** is a 6-way connector which is used for the external on/off signal as well as the auxiliary V1 & V2 inputs.

Signals CON2:
**+5V** Output supply 5V.
**0V** Output supply common.

Signals CON3:
**PWR ON/OFF+** Link both pins to a volt free contact.
**PWR ON/OFF-**

**V2+** Secondary auxiliary supply input for battery supply.
**V1+** Auxiliary supply input.
**V-** Common for V1+ and V2+ inputs.

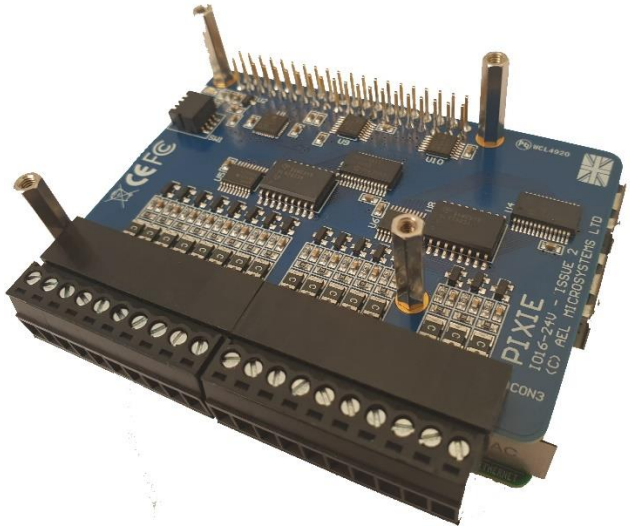
## 5.1 Insulate USB connector housing on Raspberry Pi 4

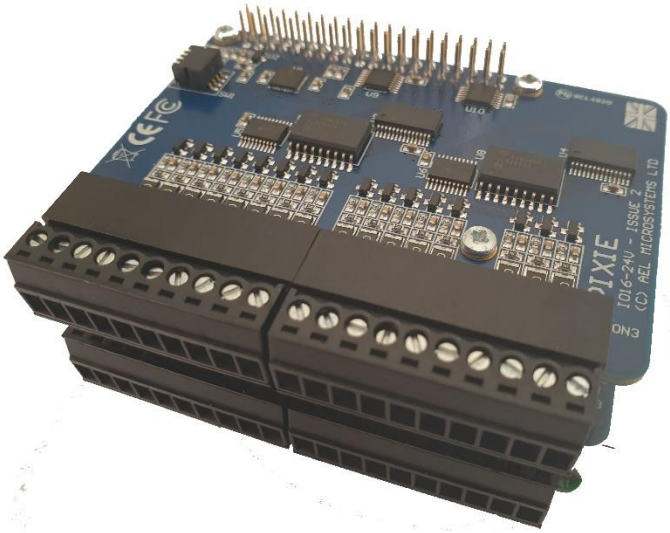
WARNING

The Ethernet and USB connectors were swapped on the Raspberry Pi 4 which means the clearance between the terminal connectors on the PIXIE board and the metal body of the USB connector is very close and, in some circumstances, could short out the terminals, which is not so good.
To prevent this, add a couple of pieces of electrical tape one on top of each other onto the USB connector as shown below before assembling the board onto the Raspberry Pi.

## 5.2 Mounting the boards.

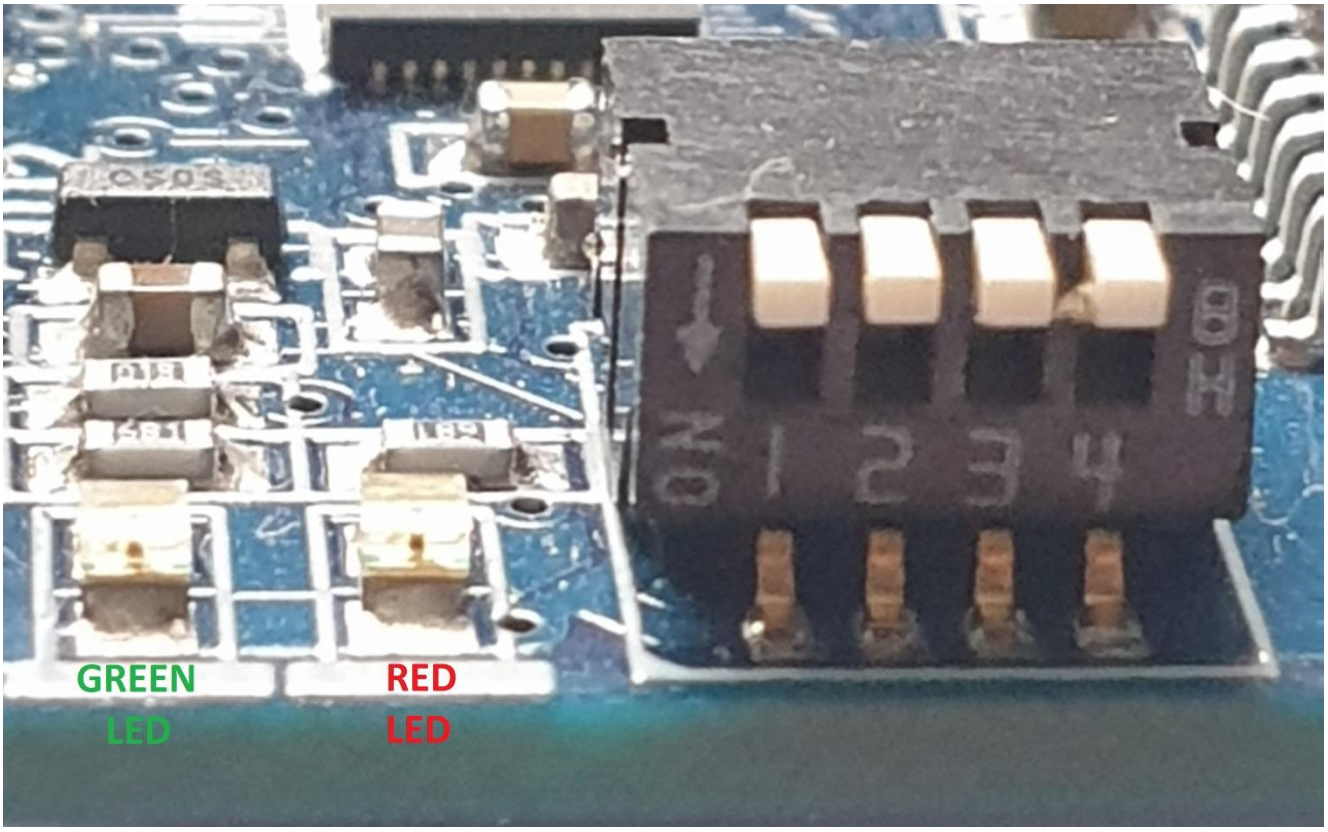
Using M2.5mm - 4mmAF – 17mm long hex standoff's mount the PIXIE boards onto the Raspberry Pi as shown. On some Pixie board's the connector CON3 obscures one of the mounting holes to the Raspberry Pi board when mounted directly to it ,so only 3 pillars are used which is sufficient to securely mount the boards together. Boards mounted on top of this one can use all 4 pillars by using the offset hole.
Mount the pillars to the Raspberry Pi using the nuts provided, the screws provide are used to secure top board to the pillars when multiple Pixie boards are used.



## 5.3 Set the boards identity.

Set the select switches shown to give each stacked PIXIE board a unique identity.




Use the following truth table to set the switches for the correct address.

| Board Id | SW1 | SW2 | SW3 | SW4 |
|---|---|---|---|---|
| **0** | OFF | OFF | OFF | OFF |
| **1** | ON | OFF | OFF | OFF |
| **2** | OFF | ON | OFF | OFF |
| **3** | ON | ON | OFF | OFF |
| **4** | OFF | OFF | ON | OFF |
| **5** | ON | OFF | ON | OFF |
| **6** | OFF | ON | ON | OFF |
| **7** | ON | ON | ON | OFF |
| **8** | OFF | OFF | OFF | ON |
| **9** | ON | OFF | OFF | ON |
| **10** | OFF | ON | OFF | ON |
| **11** | ON | ON | OFF | ON |
| **12** | OFF | OFF | ON | ON |
| **13** | ON | OFF | ON | ON |
| **14** | OFF | ON | ON | ON |
| **15** | ON | ON | ON | ON |

The default I2C address for each board will be 0x10 + “Board Id”

## 5.4 Power up and LED status.

Power on the Raspberry PI and PIXIE board's combination.

The **GREEN LED** on each of the PIXIE boards will illuminate indicating power present.

If the **RED LED** is flashing ON for 200mS with a 2 second OFF pause in between flashes, this indicates the board required configuration.

List of **RED LED** flashing states.

| | |
|---|---|
| Off, | the board is configured and fully functional. |
| 1 Flash, | not used for this board. |
| 2 Flashes, | the board has an invalid identity, contact the manufacture for advice. |
| 3 Flashes, | the board has a corrupt EEPROM, power cycle to correct the issue & defaults have been applied. |

If the speed of the LED flashes is only 100mS with 1 second pause, this means that the board is working in its boot mode and needs the firmware to be reloaded.
See the Firmware section in the PIXIE board configuration guide to resolve this problem.

The **ORANGE LED** flashing ON for 200mS with a 5 second OFF pause in between flashes, this indicates a mode as shown below.

List of **ORANGE LED** flashing states.

| | |
|---|---|
| Off, | the board is working OK. |
| 1 Flash, | V2 is currently active and supplying the input voltage. |
| 2 Flashes, | the board is in standby mode with the power output turned off. |
| 3 Flashes, | the board is in standby mode with the power turned off due to an over temperature. |

If the speed of the LED flashes is only 100mS with 1 second pause, this means that the board is working in its boot mode and needs the firmware to be reloaded.
See the Firmware section in the PIXIE board configuration guide to resolve this problem.

## 5.5 Principle of operation.

The board uses a monolithic device which is used to provide the DC-DC functionality, the on board PIC microcontroller provides the control logic for the fans, standby mode, and other power switching functionality.

In addition, there is a separate battery backed real time I2C accessible clock device, DS1307+, which can be accessed using the host software.

## 5.6 Power input and control options.

### 5.6.1 POE+ direct from the Raspberry Pi board.

To use the power over ethernet (POE) this board needs to be fitted to the Raspberry Pi so the 4 way POE connector mates with the CON4 connector on the rear of the board. Power is supplied to the Raspberry Pi using

a POE power module connected using an Ethernet cable, and the POE+ module should be 24W with a DC voltage of 48V.

### 5.6.2 Auxiliary power supply.

Auxiliary power can be used to supply the board via CON3.
There are two inputs V1+ and V2+ which are referenced to V-, they are diode OR' ed together and V2+ can also be switched on or off using software.
The input range is +9V to +48V DC.

A use of the two inputs is to provide a UPS solution, V1+ is supplied by a regular main's driven PSU or alternative supply such as a solar panel and V2+ could be connected to a battery. The software can turn on the V2 switch which means that if the power fails the battery seamlessly continues to provide power. The voltages at V1 and V2 are available to monitoring software so it can a) detect power failure and restoration, and b) monitor the battery voltage at which point if it begins to fall to low because of a prolonged power outage it can shut down the application and operating system and also switch off the battery before it damages the cells.
When power is restored to V1 then the system will start up again.
An interrupt can be generated using one of the GPIO pins whenever V1 is lost and used to manage a controlled shutdown of the operating system and the board set.

NOTE: and external means of charging the battery would be required.

### 5.6.3 External Power ON/OFF signal.

To allow for an external means of turning the output on or off a volt free signal can be provided across pins 1 & 2 of CON3 usually in the form of a switch or pushbutton.

NOTE: DO NOT REFERENCE THIS SIGNAL TO ANY OTHER 0V IN THE SYSTEM AS THIS CAN AFFECT THE OPERATION OF THE BOARD.

To turn the supply off and put the board into standby mode, close the switch or hold the pushbutton for greater than 3 seconds after which the output is turned off and enters standby mode, the Orange LED will flash twice every 5 seconds indicating standby mode, after this the switch can be opened of the pushbutton released.

To turn the output back on just toggle the switch or push button and the board exits standby mode and turns on the output supply.

### 5.6.4 V2+ input control.

The V2+ input can be turned on or off using software.
It is off when the board powers on for the first time but will maintain its state during normal and standby modes.
See the function ***PixiePoePsuSetV2()*** later in this manual to control its operation.

It also has the ability to turn the output power on if it's in standby mode and V1 becomes greater than V2, i.e. power has been restored so the boards can now be repowered and the Raspberry Pi rebooted.

### 5.6.5 Software power on/off control.

The board can be powered on/off using software commands which puts the board into a standby mode with the +5V output turned off, it can also be configured to turn the board off after a delay time and then turned back on again after a delay time, each settable between 1 second and 136 years.
See the function ***PixiePoePsuSetPowerOnOff()*** later in this manual to control its operation.

### 5.6.6 Power cycle watchdog control.

The board provides a power cycle watchdog which will turn off then back on the +5V output if the power cycle watchdog is not reset withing a pre-set time, this is useful in case the Raspberry Pi application crashes or get stuck.
Timeouts of between 1 and 255 seconds can be set.
See the functions ***PixiePoePsuSetWatchdog()*** and **PixiePoePsuWatchdogReset()** later in this manual to control its operation.

### 5.6.7 +5V Output fuse.

The +5V output is protected against short circuits with a replaceable fuse.
The fuse is a Littlefuse NANO2 448 series, 5A fast blow, part number 0448005.MR

### 5.6.8 Voltage, current and temperature monitoring.

The input voltages for V1 & V2, the +5V output and current and the board temperature along with the number of power cycles and power cycles due to watchdog reset can also be read.
The values are raw in nature and can be converted to engineering units of volts, current and degrees using the calibration values and conversion functions.
See the functions ***PixiePoePsuGetReadings()***, ***PixiePoePsuGetCalibrations(), PixiePoePsuRawToEng()*** later in this manual.

If the board temperature exceeds 60°C the board will automatically go into a standby mode by turning off the +5V output and indicating the condition by flashing the Orange LED 3 times every 5 seconds. A bit in the boards status indicates the presents of the overtemperature and a power cycles due to overtemperature counter is incremented.

The board will remain in standby mode until the temperature falls back below 40°C or the power on/off is toggled.

### 5.6.9 CPU fan control.

The CPU fan can be controlled using software or via a selectable GPIO pin
See the functions ***PixiePoePsuGetFanState(), PixiePoePsuSetFanState()*** and ***PixiePoePsuGetGpioPins() PixiePoePsuSetGpioPins()*** later in this manual.

### 5.6.10 GPIO pin control.

There are 5 controllable GPIO pins which can be individually configured, they are:

**V1 level**, this pin will change to logic 1 whenever the V1 voltage is < 5V, can be used to signal an early power fail.
It is an output at the board and input to the Raspberry Pi.

**Power on/Off**, this pin is a direct copy of the input Power On/off pin of CON3.
It is an output at the board and input to the Raspberry Pi.

**Watchdog reset**, this pin can be used to trigger a power cycle watchdog reset.
It is an input at the board and output from the Raspberry Pi.

**CPU fan**, this pin can be used to turn on/off the CPU fan. CON3.
It is an input at the board and output from the Raspberry Pi.

**Power Off**, this pin can be used to turn off the output and put the board into a standby mode.
It is an input at the board and output from the Raspberry Pi and is always pulled high and active low when configured.

See the functions ***PixiePoePsuGetGpioPins()*** and ***PixiePoePsuSetGpioPins()*** later in this manual.

### 5.6.11 Board cooling fan.

There is an additional fan used to cool aspects of the DC-DC converter which can get hot whenever full current is being drawn and the input supply voltage is low. This is fully automatic and generally will come on if the board goes above 40C and turns off below 35C.

### 5.6.12 Real time battery backed clock.

The real time clock is a DS1307+ device and can be driven directly using the appropriate operating system driver and support functions, there are many examples of these methods available.

To enable the driver, edit the **/boot/config.txt** file and add the following:

**dtparam=i2c1=on**
**dtoverlay=i2c-rtc, ds1307**

The save and reboot.

## 5.7 Input voltages and current calibrations.

The board supports calibration values for the input voltages and current which allow you to scale the raw data into sensible engineered units of volts and amps.
These calibrations are stored in the memory on the board, so the calibration values stay with the board and not in some application file somewhere.
Two values are available for each input and output, a zero and a span, which are both “float” 32bit values. Functions are provided to set and retrieve these values as well as support routines to apply these values to the data.

See the functions ***PixiePoePsuDefaultCalibrations(), PixiePoePsuGetCalibrations()*** ***PixiePoePsuSetCalibrations()*** and ***PixiePoePsuGetReadings ()*** later in this manual.

## 5.8 Configuration and test functions.

See the PIXIE configuration manual for information to configure the board.

### 5.8.1 Board specific configuration.

This board has some additional software and configuration settings to allow the full features of the board to be used.

They are:

- Set a GPIO pin to act as a V1 power loss signal.
- Set a GPIO pin to act as a direct copy of the POWER ON/OFF signal on CON3.
- Set a GPIO pin to act as a power cycle watchdog reset.
- Set a GPIO pin to act as a CPU fan on/off signal.
- Set a GPIO pin to act as a power off signal.

All these items can either be set and stored in the onboard EEPROM or are activated and changes using the software support libraries.
To set them in hardware as well as run some optional board test utilities which are available as a sanity or diagnostic check of the board, from the main **PixieBoard** menu select the **(U)-Board Utilities** option.

```
PIXIE BOARD[0] - UTILITY MENU :
(?)-Menu help, (B)-Board ID, (2)-V2 Control, (C)-CPU Fan ON/OFF, (D)-Set on/off delay,
(E)-Calibrations, (F)-Set Calibration, (G)-Default Calibrations, (P)-Reset Watchdog,
(R)-Display readings, (S)-Edit GPIO settings, (W)-Watchdog timeouts, (X)-Exit...
```

To set the additional configuration settings select the **(S)-Edit GPIO Settings** option and edit the values required with an option to save them to EEPROM so they always take effect on power up.

```
Select V1 Power loss GPIO: (0) >
Select V1 Power loss POL : (L) >
Select power on/off signal GPIO: (0) >
Select power on/off signal POL : (L) >
Select watchdog reset GPIO: (0) >
Select watchdog reset POL : (L) >
Select CPU fan GPIO: (0) >
Select CPU fan POL : (L) >
Select power off signal GPIO: (0) >
```

A complete display of the board specific current settings will be shown to display any changes.

**Select V1 Power loss GPIO**

This sets the GPIO pin used as an indication of V1 power loss.
It is an output at the board and input to the Raspberry Pi.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 can be used

**Select V1 Power loss POL**

This sets the GPIO pin polarity to active ‘L’ low or ‘H’ high.

**Select power on/off signal GPIO**

This sets the GPIO pin used as an indication of the POWER ON/OFF signal of CON3.
It is an output at the board and input to the Raspberry Pi.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 can be used

**Select power on/off signal POL**

This sets the GPIO pin polarity to active ‘L’ low or ‘H’ high.

**Select watchdog reset GPIO**

This sets the GPIO pin used as a power cycle watchdog reset.
It is an input at the board and output from the Raspberry Pi.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 can be used

**Select watchdog reset POL**

This sets the GPIO pin polarity to active ‘L’ low or ‘H’ high.

**Select CPU fan GPIO**

This sets the GPIO pin used to control the CPU fan.
It is an input at the board and output from the Raspberry Pi.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 can be used

**Select CPU fan POL**

This sets the GPIO pin polarity to active ‘L’ low or ‘H’ high.

**Select power off signal GPIO**

This sets the GPIO pin used to power down the board and go to standby mode.
It is an input at the board and output from the Raspberry Pi.
It is always active low and adds a pullup to the GPIO line whenever configured.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 can be used

**Save to EEPROM Y/N** This saves any changes in the EEPROM and will be used whenever the board is power cycled.

### 5.8.2 Board specific test utilities.

On the board utilities menu are some sanity diagnostic tests that can be invoked to test the functionality of the board.

```
PIXIE BOARD[0] - UTILITY MENU :
(?)-Menu help, (B)-Board ID, (2)-V2 Control, (C)-CPU Fan ON/OFF, (D)-Set on/off delay,
(E)-Calibrations, (F)-Set Calibration, (G)-Default Calibrations, (P)-Reset Watchdog,
(R)-Display readings, (S)-Edit GPIO settings, (W)-Watchdog timeouts, (X)-Exit...
```

A full description of these is given by the **(?)-Menu help.**

**(B)-Board ID** Change the current board.

**(2)-V2 Control** Sets the power V2 controls

**(C)-CPU Fan ON/OFF** Turn the CPU cooling fan on or off.

**(D)-Set on/off delay** Sets the power on and off delays.

**(P)-Reset Watchdog** Reset the watchdog.

**(R)-Display readings** Display the measured input voltages and counters.

**(S)-Edit GPIO settings** Change the board settings.
See previous section.

**(W)-Watchdog timeouts**
Sets the watchdog timeout value.

### 5.8.3 Board calibrations.

On the board utilities menu are some calibration storage and retrieval functions.

```
PIXIE BOARD[0] - UTILITY MENU :
(?)-Menu help, (B)-Board ID, (2)-V2 Control, (C)-CPU Fan ON/OFF, (D)-Set on/off delay,
(E)-Calibrations, (F)-Set Calibration, (G)-Default Calibrations, (P)-Reset Watchdog,
(R)-Display readings, (S)-Edit GPIO settings, (W)-Watchdog timeouts, (X)-Exit...
```

**(E)-Calibrations** Displays the current calibration values.

**(F)-Set Calibration** Allows the editing of the zero and span value for the input or output.

**(G)-Default Calibrations** Default the calibrations to those applicable to the board.
These set calibrations to scale the voltage and current inputs to volts and amps and the temperature to degrees C.

# 6. Software support.

This board is fully supported by the PIXIE 'C', C++, Python and LabVIEW libraries, details of these functions follows.
For a more detailed overview of the software syntax and common supporting functionality used by these functions the PIXIE software manual should be consulted.

## 6.1 POE8-PSU Power supply board 'C' library support functions

This group contains 'C' functions for supporting the POE-PSU power supply board.
All of the Pixie support functions are contained in a compiled static library **libpixiepistatic.a** which can be used at link time or the individual source files can be compiled along with your application.

They all require the ***#include <Pixie.h>*** header file.

All functions make use of a board control structure ***PixiePoePsuCtrl_t*** which is declared for the board used and contains all the control parameters used for the board, it is constructed using the ***PixiePoePsuConstruct()*** function and can be optionally destroyed using the ***PixiePoePsuDestroy()*** function.

### 6.1.1 PixiePoePsuConstruct()

This function is used to initialise the ***PixiePoePsuCtrl_t*** structure for use by all the other board support functions. Failure to construct the structure will result in returned errors when all the other board support functions are called, so this is the first board support function to be called.

**Syntax:**

***int_t PixiePoePsuConstruct(PixiePoePsuCtrl _t* pCtrl, uint16_t I2cAddress);***

**Arguments:**

| | |
|---|---|
| ***pCtrl*** | Is a pointer to the ***PixiePoePsuCtrl _t*** structure to use. |
| ***I2cAddress*** | I2C address for the board to access. |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

### 6.1.2 PixiePoePsuDefaultCalibration()

This function is used to set default calibration values for the measured volts, current and temperature measurements when used with the raw to engineered conversion functions.

**Syntax:**

***int_t PixiePoePsuDefaultCalibration(PixiePoePsuCtrl _t* pCtrl, uint8_t calibNumber);***

**Arguments:**

| | |
|---|---|
| ***pCtrl*** | Is a pointer to the ***PixiePoePsuCtrl _t*** structure to use. |
| ***calibNumber*** | The number of the calibration to default, use ***PX_POE_PSU_MEASURE_...*** |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

### 6.1.3 PixiePoePsuDefaultCalibrations()

This function is used to default all the boards calibrations.

**Syntax:**

***int_t PixiePoePsuDefaultCalibrations(PixiePoePsuCtrl _t pCtrl);***

**Arguments:**

| | |
|---|---|
| ***pCtrl*** | Is a pointer to the ***PixiePoePsuCtrl _t*** structure to use. |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

### 6.1.4 PixiePoePsuDestroy()

This function is used to destroy the ***PixiePoePsuCtrl _t*** structure when the board is no longer required.

**Syntax:**

***int_t PixiePoePsuDestroy(PixiePoePsuCtrl _t* pCtrl);***

**Arguments:**

| | |
|---|---|
| ***pCtrl*** | Is a pointer to the ***PixiePoePsuCtrl _t*** structure to use. |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

### 6.1.5 PixiePoePsuGetCalibrations()

This function is used to read all the calibrations from the board and save in the ***PixiePoePsuCtrl _t*** structure ***calibrations***.

**Syntax:**

***uint16_t PixiePoePsuGetCalibrations(PixiePoePsuCtrl _t* pCtrl);***

**Arguments:**

***pCtrl*** Is a pointer to the ***PixiePoePsuCtrl _t*** structure to use.

**Returns:**

***PIXIE_OK*** Completed OK.
***E...*** Linux error code.

### 6.1.6 PixiePoePsuGetDefaultCalibration()

This function is used return the default calibration values for the calibration number requested.

**Syntax:**

***void PixiePoePsuDefaultCalibration(uint8_t calibNumber, PixieAnlCalib_t* pCalibration);***

**Arguments:**

***calibNumber*** The number of the calibration to default, use ***PX_POE_PSU_MEASURE_...***

***pCalibibration*** Is a pointer to the ***PixieAnlCalib _t*** structure to fill in.

**Returns:**

Nothing.

### 6.1.7 PixiePoePsuGetFanState()

This function is used to read the current state of the CPU and board cooling fans.

**Syntax:**

***int_t PixiePoePsuGetFanState(***
***PixiePoePsuCtrl _t* pCtrl ,***
***uint8_t* pCpuStateFlg,***
***uint8_t* pBoardStateFlg)***

**Arguments:**

***pCtrl*** Is a pointer to the ***PixiePoePsuCtrl _t*** structure to use.

***pCpuStateFlg*** Is a pointer to return the CPU fan state in.

***pBoardStateFlg***
Is a pointer to return the board fan state in.

**Returns:**

***PIXIE_OK*** Completed OK.
***E...*** Linux error code.

### 6.1.8 PixiePoePsuGetGpioPins()

This function is used to read the current setting of the GPIO pins.

**Syntax:**

***int_t PixiePoePsuGetGpioPins(***
***PixiePoePsuCtrl _t* pCtrl ,***
***uint8_t* pV1Gpio,***
***uint8_t* pPwrOnOffGpio,***
***uint8_t* pWdogGpio,***
***uint8_t* pCpuFanGpio,***
***uint8_t* pPwrOffGpio)***

**Arguments:**

***pCtrl*** Is a pointer to the ***PixiePoePsuCtrl _t*** structure to use.

***pV1Gpio*** Is a pointer to return the V1 power fail signal pin in.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***PXBC_GPIO_POLARITY*** which indicated the polarity.

***pPwrOnOffGpio***
Is a pointer to return the Power On/Off signal pin in.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***PXBC_GPIO_POLARITY*** which indicated the polarity.

***pWdogGpio*** Is a pointer to return the watchdog trigger signal pin in.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***PXBC_GPIO_POLARITY*** which indicated the polarity.

***pCpuFanGpio*** Is a pointer to return the CPU fan control signal pin in.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***PXBC_GPIO_POLARITY*** which indicated the polarity.

***pPwrOffGpio*** Is a pointer to return the power off signal pin in.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated

**Returns:**

***PIXIE_OK*** Completed OK.
***E...*** Linux error code.

### 6.1.9 PixiePoePsuGetReadings()

This function is used to read all the measured values from the board and store them in the ***readings.rawReading[]*** of the ***PixieAdc8Dac2Ctrl_t*** structure.

**Syntax:**

***int_t PixiePoePsuGetReadings(PixiePoePsuCtrl _t* pCtrl);***

**Arguments:**

| | |
|---|---|
| ***pCtrl*** | Is a pointer to the ***PixiPoePsuCtrl _t*** structure to use. |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

### 6.1.10 PixiePoePsuRawToEng()

This function is used to scale the raw ***readings.rawReading[]*** into engineered values ***readings.engReading[]*** in the ***PixiePoePsuCtrl_t*** structure.
The **PixiePoePsuGetCalibrations()** needs to be called prior to this function to load in the calibrations from the board otherwise ***EINVAL*** is returned.

**Syntax:**

***Int_t PixiePoePsuRawToEng(PixiePoePsuCtrl _t* pCtrl);***

**Arguments:**

| | |
|---|---|
| ***pCtrl*** | Is a pointer to the ***PixiePoePsuCtrl _t*** structure to use. |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

### 6.1.11 PixiePoePsuSaveSettings()

This function is used to save any settings made to this board into its EEPROM for use next time the board powers up.

**Syntax:**

***int_t PixiePoePsuSaveSettings(PixiePoePsuCtrl _t* pCtrl);***

**Arguments:**

| | |
|---|---|
| ***pCtrl*** | Is a pointer to the ***PixiePoePsuCtrl _t*** structure to use. |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

### 6.1.12 PixiePoePsuSetCalibration()

This function is used to set a calibration on the target board.

**Syntax:**

***int_t PixiePoePsuSetCalibrations(***
***PixiePoePsuCtrl _t* pCtrl,***
***uint8_t calibNumber,***
***float_t zeroValue,***
***float_t spanValue);***

**Arguments:**

| | |
|---|---|
| ***pCtrl*** | Is a pointer to the ***PixiePoePsuCtrl _t*** structure to use. |
| ***calibNumber*** | The number of the calibration to set, use ***PX_POE_PSU_MEASURE_...*** |
| ***zeroValue*** | The zero value. |
| ***spanValue*** | The span value. |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

### 6.1.13 PixiePoePsuSetFanState()

This function is used to set the current state of the CPU cooling fan.

**Syntax:**

***int_t PixiePoePsuSetFanState(PixiePoePsuCtrl _t* pCtrl, uint8_t stateFlg);***

**Arguments:**

| | |
|---|---|
| ***pCtrl*** | Is a pointer to the ***PixiePoePsuCtrl _t*** structure to use. |
| ***stateFlg*** | ***TRUE*** = On.<br>***FALSE*** = off. |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

## 6.1.14 PixiePoePsuSetGpioPins()

This function is used to set the setting of the GPIO pins.

**Syntax:**

***int_t PixiePoePsuSetGpioPins(***
***PixiePoePsuCtrl _t* pCtrl ,***
***uint8_t v1Gpio,***
***uint8_t pwrOnOffGpio,***
***uint8_t wdogGpio,***
***uint8_t cpuFanGpio,***
***uint8_t pwrOffGpio)***

**Arguments:**

***pCtrl*** Is a pointer to the ***PixiePoePsuCtrl _t*** structure to use.

***v1Gpio*** The V1 power fail signal pin mapping and polarity.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***PXBC_GPIO_POLARITY*** which indicated the polarity.

***pwrOnOffGpio***
The Power On/Off signal pin mapping and polarity.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***PXBC_GPIO_POLARITY*** which indicated the polarity.

***wdogGpio*** The watchdog trigger signal pin mapping and polarity.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***PXBC_GPIO_POLARITY*** which indicated the polarity.

***cpuFanGpio*** The CPU fan control signal pin mapping and polarity.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***PXBC_GPIO_POLARITY*** which indicated the polarity.

***pwrOffGpio*** The power off signal pin mapping.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated

**Returns:**

***PIXIE_OK*** Completed OK.
***E...*** Linux error code.

## 6.1.15 PixiePoePsuSetPowerOnOff()

This function is used to set the power off and on delays.

**Syntax:**

***int_t PixiePoePsuSetPowerOnOff(***
***PixiePoePsuCtrl _t* pCtrl ,***
***uint32_t onDelaySecs,***
***uint32_t offDelaySecs)***

**Arguments:**

***pCtrl*** Is a pointer to the ***PixiePoePsuCtrl _t*** structure to use.

***onDelaySecs*** The number of seconds to delay before turning back on after entry to standby mode. Set to 0 if it's to stay in standby.

***offDelaySecs*** The number of seconds to delay before turning off the output and going to standby mode.

**Returns:**

***PIXIE_OK*** Completed OK.
***E...*** Linux error code.

## 6.1.16 PixiePoePsuSetV2()

This function is used to set the V2 power control.

**Syntax:**

***int_t PixiePoePsuSetV2(***
***PixiePoePsuCtrl _t* pCtrl ,***
***uint8_t v2OnFlg,***
***uint8_t v1GtV2OnFlg)***

**Arguments:**

***pCtrl*** Is a pointer to the ***PixiePoePsuCtrl _t*** structure to use.

***v2OnFlg*** The state of the V2 switch, ***TRUE*** = on, ***FALSE*** = Off.

***v1GtV2OnFlg*** The state of the V1 greater than V2 condition to exit standby mode.
***FALSE*** = Ignore.
***TRUE*** = Exit standby mode when V1 > V2.

**Returns:**

***PIXIE_OK*** Completed OK.
***E...*** Linux error code.

## 6.1.17 PixiePoePsuSetWatchdog()

This function is used to set the power cycle watchdog timer.

**Syntax:**

***int_t PixiePoePsuSetWatchdog(PixiePoePsuCtrl _t* pCtrl , uint8_t time)***

**Arguments:**

| | |
|---|---|
| ***pCtrl*** | Is a pointer to the ***PixiePoePsuCtrl _t*** structure to use. |
| ***time*** | The time in seconds before power cycling if a reset trigger has not been received.<br>0 = Watchdog disabled.<br>1-255 seconds |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

## 6.1.18 PixiePoePsuWatchdogReset()

This function is used to reset the power cycle watchdog.

**Syntax:**

***int_t PixiePoePsuWatchdogReset(PixiePoePsuCtrl _t* pCtrl)***

**Arguments:**

| | |
|---|---|
| ***pCtrl*** | Is a pointer to the ***PixiePoePsuCtrl _t*** structure to use. |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

## 6.2 POE-PSU Power supply board 'C++' library support functions

This group contains 'C++' functions for supporting the POE-PSU power supply board.
All of the Pixie support functions are contained in a compiled static library **libpixiepistatic.a** which can be used at link time or the individual source files can be compiled along with your application.

The class is ***PixieBoardPoePsu***
They all require the ***#include <PixieLib.hpp>*** header file.

### 6.2.1 PixieBoardPoePsu()

This is the class constructor used to create a board object.

**Syntax:**
***PixieBoardPoePsu(uint_t i2cAddress);***

**Arguments:**
***i2cAddress*** I2C address for the board to access.

### 6.2.2 DefaultCalibration()

This function is used to set default calibration values for the measured volts, current and temperature measurements when used with the raw to engineered conversion functions.

**Syntax:**
***int_t DefaultCalibration(uint8_t calibNumber);***

**Arguments:**
***calibNumber*** Number of the calibration to default, use ***PX_POE_PSU_MEASURE_...***

**Returns:**
***PIXIE_OK*** Completed OK.
***E...*** Linux error code.

### 6.2.3 DefaultCalibrations()

This function is used to set default calibration values for all board readings.

**Syntax:**
***int_t DefaultCalibrations(void);***

**Arguments:**

**Returns:**
***PIXIE_OK*** Completed OK.
***E...*** Linux error code.

### 6.2.4 GetCalibration()

This function is used to read the current calibration from the board.

**Syntax:**

***int_t GetCalibration(uint8_t calibNumber, float_t* pZeroValue, float_t* pSpanValue);***

**Arguments:**

***calibNumber*** The number of the calibration to get, use ***PX_POE_PSU_MEASURE_...***

***pZeroValue*** Pointer to return the zero value in.

***pSpanValue*** Pointer to return the span value in.

**Returns:**

***PIXIE_OK*** Completed OK.
***E...*** Linux error code.

### 6.2.5 GetCalibrations()

This function is used to read the current calibrations from the board.

**Syntax:**

***int_t GetCalibrations(void);***

**Arguments:**

**Returns:**

***PIXIE_OK*** Completed OK.
***E...*** Linux error code.

### 6.2.6 GetCalibrationsPtr()

This function is used to return a pointer to the calibrations of the board.

**Syntax:**

***PixieAnlCalibs_t* GetCalibrationsPtr(void);***

**Arguments:**

**Returns:**

Pointer to the calibrations.

### 6.2.7 GetFanState()

This function is used to read the current state of the CPU and board cooling fans.

**Syntax:**

***int_t GetFanState(uint8_t* pCpuStateFlg, uint8_t* pBoardStateFlg)***

**Arguments:**

***pCpuStateFlg*** Is a pointer to return the CPU fan state in.

***pBoardStateFlg***
Is a pointer to return the board fan state in.

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

## 6.2.8 GetGpioPins()

This function is used to read the current setting of the GPIO pins.

**Syntax:**

***int_t GetGpioPins(***
***uint8_t* pV1Gpio,***
***uint8_t* pPwrOnOffGpio,***
***uint8_t* pWdogGpio,***
***uint8_t* pCpuFanGpio,***
***uint8_t* pPwrOffGpio)***

**Arguments:**

***pV1Gpio*** Is a pointer to return the V1 power fail signal pin in.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***PXBC_GPIO_POLARITY*** which indicated the polarity.

***pPwrOnOffGpio***
Is a pointer to return the Power On/Off signal pin in.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***PXBC_GPIO_POLARITY*** which indicated the polarity.

***pWdogGpio*** Is a pointer to return the watchdog trigger signal pin in.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***PXBC_GPIO_POLARITY*** which indicated the polarity.

***pCpuFanGpio*** Is a pointer to return the CPU fan control signal pin in.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***PXBC_GPIO_POLARITY*** which indicated the polarity.

***pPwrOffGpio*** Is a pointer to return the power off signal pin in.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated

**Returns:**

***PIXIE_OK*** Completed OK.
***E...*** Linux error code.

### 6.2.9 GetStatus()

This function is used to read the current board status.

**Syntax:**

***int_t GetStatus(***
***uint8_t* pPwrOnOffSig,***
***uint8_t* pT2Sig,***
***uint16_t* pPowerCycles,***
***uint8_t* pWdogCycles***
***uint8_t* pOverTempCycles)***

**Arguments:**

***pPwrOnOffSig*** Is a pointer to return the Power On/Off signal pin in.

***pT2Sig*** Is a pointer to return the T2 signal pin in.

***pPowerCycles*** Is a pointer to return the power cycles count in.

***pWdogCycles*** Is a pointer to return the watchdog power cycles in.

***pOverTempCycles*** Is a pointer to return the over temperature power cycles in.

**Returns:**

***PIXIE_OK*** Completed OK.
***E...*** Linux error code.

### 6.2.10 GetValues()

This function is used to get the PSU values and convert to engineered as well.

**Syntax:**

***int_t GetValues();***

**Arguments:**

**Returns:**

***PIXIE_OK*** Completed OK.
***E...*** Linux error code.

### 6.2.11 GetValues()

This function is used to get the ADC raw readings and copy target array.

**Syntax:**

***int_t GetValues(int32_t* pValues);***

**Arguments:**

***pValues*** Is a pointer to return the values.

**Returns:**

***PIXIE_OK*** Completed OK.

***E...*** Linux error code.

## 6.2.12 GetValues()

This function is used to get the ADC engineered values and copy target array.

**Syntax:**
***int_t GetValues(double_t* pValues);***

**Arguments:**
***pValues*** Is a pointer to return the values.

**Returns:**
***PIXIE_OK*** Completed OK.
***E...*** Linux error code.

## 6.2.13 GetValuesPtr()

This function is used to get a pointer to the input values.

**Syntax:**
***PixiePoePsuReadings_t GetValues();***

**Arguments:**

**Returns:**
Pointer to the board readings.

## 6.2.14 SaveSettings()

This function is used to save any settings made to this board into its EEPROM for use next time the board powers up.

**Syntax:**
***int_t SaveSettings(void);***

**Arguments:**

**Returns:**
***PIXIE_OK*** Completed OK.
***E...*** Linux error code.

## 6.2.15 SetCalibration()

This function is used to set a calibration.

**Syntax:**

***int_t SetCalibration(uint8_t calibNumber, float_t zeroValue, float_t spanValue);***

**Arguments:**

| | |
|---|---|
| ***calibNumber*** | The number of the calibration to set, use ***PX_POE_PSU_MEASURE_...*** |
| ***zeroValue*** | The zero calibration value. |
| ***spanValue*** | The span calibration value. |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

## 6.2.16 SetFanState()

This function is used to set the current state of the CPU cooling fan.

**Syntax:**

***int_t SetFanState(uint8_t stateFlg);***

**Arguments:**

| | |
|---|---|
| ***stateFlg*** | ***TRUE*** = On.<br>***FALSE*** = off. |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

## 6.2.17 SetGpioPins()

This function is used to set the setting of the GPIO pins.

**Syntax:**

***int_t SetGpioPins(***
***uint8_t v1Gpio,***
***uint8_t pwrOnOffGpio,***
***uint8_t wdogGpio,***
***uint8_t cpuFanGpio,***
***uint8_t pwrOffGpio)***

**Arguments:**

***v1Gpio*** The V1 power fail signal pin mapping and polarity.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***PXBC_GPIO_POLARITY*** which indicated the polarity.

***pwrOnOffGpio***
The Power On/Off signal pin mapping and polarity.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***PXBC_GPIO_POLARITY*** which indicated the polarity.

***wdogGpio*** The watchdog trigger signal pin mapping and polarity.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***PXBC_GPIO_POLARITY*** which indicated the polarity.

***cpuFanGpio*** The CPU fan control signal pin mapping and polarity.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***PXBC_GPIO_POLARITY*** which indicated the polarity.

***pwrOffGpio*** The power off signal pin mapping.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated

**Returns:**

***PIXIE_OK*** Completed OK.
***E...*** Linux error code.

## 6.2.18 SetGpioPinCpuFan()

This function is used to set the CPU fan control input GPIO pin.

**Syntax:**

***int_t SetGpioPinCpuFan(uint8_t gpioPin, uint8_t invertFlg);***

**Arguments:**

| | |
|---|---|
| ***gpioPin*** | GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.<br>GPIO pin 0 = Not allocated |
| ***invertFlg*** | 1 = Invert the signal. |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

## 6.2.19 SetGpioPinPowerOff()

This function is used to set the power off input GPIO pin.

**Syntax:**

***int_t SetGpioPinPowerOff uint8_t gpioPin);***

**Arguments:**

| | |
|---|---|
| ***gpioPin*** | GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.<br>GPIO pin 0 = Not allocated |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

## 6.2.20 SetGpioPinPowerOnOff()

This function is used to set the power on/off input to output GPIO pin.

**Syntax:**

***int_t SetGpioPinPowerOnOff(uint8_t gpioPin, uint8_t invertFlg);***

**Arguments:**

| | |
|---|---|
| ***gpioPin*** | GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.<br>GPIO pin 0 = Not allocated |
| ***invertFlg*** | 1 = Invert the signal. |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

## 6.2.21 SetGpioPinV1()

This function is used to set the V1 present output GPIO pin.

**Syntax:**

***int_t SetGpioPinV1(uint8_t gpioPin, uint8_t invertFlg);***

**Arguments:**

| | |
|---|---|
| ***gpioPin*** | GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.<br>GPIO pin 0 = Not allocated |
| ***invertFlg*** | 1 = Invert the signal. |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

## 6.2.22 SetGpioPinWatchdog()

This function is used to set the watchdog reset input GPIO pin.

**Syntax:**

***int_t SetGpioPinWatchdog(uint8_t gpioPin, uint8_t invertFlg);***

**Arguments:**

| | |
|---|---|
| ***gpioPin*** | GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.<br>GPIO pin 0 = Not allocated |
| ***invertFlg*** | 1 = Invert the signal. |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

### 6.2.23 SetPowerOnOff()

This function is used to set the power off and on delays.

**Syntax:**

***int_t SetPowerOnOff(uint32_t onDelaySecs, uint32_t offDelaySecs)***

**Arguments:**

| | |
|---|---|
| ***onDelaySecs*** | The number of seconds to delay before turning back on after entry to standby mode.<br>Set to 0 if it's to stay in standby. |
| ***offDelaySecs*** | The number of seconds to delay before turning off the output and going to standby mode. |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

### 6.2.24 SetV2()

This function is used to set the V2 power control.

**Syntax:**

***int_t SetV2(uint8_t v2OnFlg, uint8_t v1GtV2OnFlg)***

**Arguments:**

| | |
|---|---|
| ***v2OnFlg*** | The state of the V2 switch, ***TRUE*** = on, ***FALSE*** = Off. |
| ***v1GtV2OnFlg*** | The state of the V1 greater than V2 condition to exit standby mode.<br>***FALSE*** = Ignore.<br>***TRUE*** = Exit standby mode when V1 > V2. |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

### 6.2.25 SetWatchdog()

This function is used to set the power cycle watchdog timer.

**Syntax:**

***int_t SetWatchdog( uint8_t time)***

**Arguments:**

| | |
|---|---|
| ***time*** | The time in seconds before power cycling if a reset trigger has not been received.<br>0 = Watchdog disabled.<br>1-255 seconds |

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

## 6.2.26 WatchdogReset()

This function is used to reset the power cycle watchdog.

**Syntax:**

***int_t WatchdogReset()***

**Arguments:**

**Returns:**

| | |
|---|---|
| ***PIXIE_OK*** | Completed OK. |
| ***E...*** | Linux error code. |

## 6.3 POE-PSU power supply board ‘Python’ library support functions

This group contains ‘Python’ functions for supporting the POE-PSU power supply board.

The class is ***PyPixieBoardPoePsu***
They all require the ***import PixiePy*** module.

### 6.3.1 PyPixieBoardPoePsu()

This is the class constructor used to create a board object.

**Syntax:**
***object = PixiePy.PyPixieBoardPoePsu(i2cAddress)***

**Arguments:**
***i2cAddress*** I2C address for the board to access.

### 6.3.2 DefaultCalibration()

This function is used to set default calibration values for the measured volts, current and temperature measurements when used with the raw to engineered conversion functions.

**Syntax:**
***Result = object.DefaultCalibration(calibNumber)***

**Arguments:**
***calibNumber*** Number of the calibration to default.
0 = V1+,1 = V2+, 2 = V+, 3 = +5V, 4 = 5VI, 5 = temperature

**Returns:**
***result*** ***0*** or ***E...*** Linux error code.

### 6.3.3 DefaultCalibrations()

This function is used to set default calibration values for all of the board’s inputs and outputs.

**Syntax:**
***Result = object.DefaultCalibrations()***

**Arguments:**

**Returns:**
***result*** ***0*** or ***E...*** Linux error code.

### 6.3.4 GetCalibration()

This function is used to read the current calibrations from the board.

**Syntax:**
***result = object.GetCalibration(calibNumber)***

**Arguments:**
***calibNumber*** Number of the calibration to default.

**Returns:**
***result*** ***0*** or ***E...*** Linux error code.
***zero*** Zero value.
***span*** Span value.

### 6.3.5 GetCalibrations()

This function is used to read all the calibrations from the board.

**Syntax:**
***result = object.GetCalibrations()***

**Arguments:**

**Returns:**
***result*** ***0*** or ***E...*** Linux error code.

### 6.3.6 GetFanState()

This function is used to read the current state of the CPU and board cooling fans.

**Syntax:**
***result, cpuStateFlg , boardStateFlg = object. GetFanState()***

**Arguments:**
Is a pointer to return the board fan state in.
**Returns:**
***result*** ***0*** or ***E...*** Linux error code.

***cpuStateFlg*** The current state of the CPU fan.

***boardStateFlg*** The current state of the board fan.

### 6.3.7 GetGpioPins()

This function is used to read the current setting of the GPIO pins.

**Syntax:**

***result, v1Gpio ,pwrOnOffGpio, wdogGpio, cpuFanGpio, pwrOffGpio = object. GetFanState()***

**Arguments:**

**Returns:**

***result*** ***0*** or ***E...*** Linux error code.

***v1Gpio*** V1 power fail signal GPIO pin and invert state.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***0x80*** which indicated the polarity.

***pwrOnOffGpio***
Power On/Off signal GPIO pin and invert state.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***0x80*** which indicated the polarity.

***wdogGpio*** Watchdog trigger signal GPIO pin and invert state.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***0x80*** which indicated the polarity.

***cpuFanGpio*** CPU fan control signal GPIO pin and invert state
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***0x80*** which indicated the polarity.

***pwrOffGpio*** Power off signal GPIO pin.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used
GPIO pin 0 = Not allocated.

### 6.3.8 GetStatus()

This function is used to read the current board status.

**Syntax:**
***result, pwrOnOffSig, t2Sig, powerCycles, wdogCycles, overTempCycles = object.GetStatus()***

**Arguments:**

**Returns:**
***result*** ***0*** or ***E...*** Linux error code.

***pwrOnOffSig*** State of the Power On/Off signal.

***t2Sig*** State of the T2 signal pin.

***powerCycles*** Power cycles count.

***wdogCycles*** Watchdog power cycles.

***overTempCycles*** Over temperature power cycles.

### 6.3.9 GetValues()

This function is used to read the board values, raw and engineered.

**Syntax:**
***result, readings, readingsEng = object.GetValues()***

**Arguments:**

**Returns:**
***result*** ***0*** or ***E...*** Linux error code.

***readings*** ***6*** converted raw board values, V1+, V2+, V+, +5V, 5VI, temperature in that order.

***readingsEng*** ***6*** converted and calibration scaled floating point values,
V1+, V2+, V+, +5V, 5VI, temperature in that order.

### 6.3.10 SaveSettings()

This function is used to save any settings made to this board into its EEPROM for use next time the board powers up.

**Syntax:**
***result = object.SaveSettings()***

**Arguments:**

**Returns:**
***result*** ***0*** or ***E...*** Linux error code.

### 6.3.11 SetCalibration()

This function is used to set a calibration.

**Syntax:**
***result = object.SetCalibration(calibNumber, zeroValue, spanValue)***

**Arguments:**

***calibNumber*** Number of the calibration to set.
0 = V1+,1 = V2+, 2 = V+, 3 = +5V, 4 = 5VI, 5 = temperature

***zeroValue*** Zero calibration value.

***spanValue*** Span calibration value.

**Returns:**

***result*** ***0*** or ***E...*** Linux error code.

### 6.3.12 SetFanState()

This function is used to set the state of the CPU fan.

**Syntax:**
***result = object.SetFanState(state)***

**Arguments:**

***state*** The value to set.
***0*** = off, ***1*** = on.

**Returns:**

***result*** ***0*** or ***E...*** Linux error code.

## 6.3.13 SetGpioPins()

This function is used to set the setting of the GPIO pins.

**Syntax:**

***result = object. SetGpioPins(v1Gpio, pwrOnOffGpio, wdogGpio, cpuFanGpio, pwrOffGpio )***

**Arguments:**

***v1Gpio*** The V1 power fail signal pin mapping and polarity.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***0x80*** which indicated the polarity.

***pwrOnOffGpio***
The Power On/Off signal pin mapping and polarity.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***0x80*** which indicated the polarity.

***wdogGpio*** The watchdog trigger signal pin mapping and polarity.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***0x80*** which indicated the polarity.

***cpuFanGpio*** The CPU fan control signal pin mapping and polarity.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated
The value may be masked with ***0x80*** which indicated the polarity.

***pwrOffGpio*** The power off signal pin mapping.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated

**Returns:**

***result*** ***0*** or ***E...*** Linux error code.

## 6.3.14 SetGpioPinCpuFan()

This function is used to set the CPU fan control input GPIO pin.

**Syntax:**

***result = object. SetGpioPinCpuFan(gpioPin, invertFlg)***

**Arguments:**

***gpioPin*** GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated

***invertFlg*** 1 = Invert the signal.

**Returns:**

***result*** ***0*** or ***E...*** Linux error code.

## 6.3.15 SetGpioPinPowerOff()

This function is used to set the power off input GPIO pin.

**Syntax:**

***result = object. SetGpioPinPowerOff(gpioPin)***

**Arguments:**

***gpioPin*** GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated

**Returns:**

***result*** ***0*** or ***E...*** Linux error code.

## 6.3.16 SetGpioPinPowerOnOff()

This function is used to set the power on/off input to output GPIO pin.

**Syntax:**

***result = object. SetGpioPinPowerOnOff(gpioPin, invertFlg)***

**Arguments:**

***gpioPin*** GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.
GPIO pin 0 = Not allocated

***invertFlg*** 1 = Invert the signal.

**Returns:**

***result*** ***0*** or ***E...*** Linux error code.

## 6.3.17 SetGpioPinV1()

This function is used to set the V1 present output GPIO pin.

**Syntax:**

***result = object. SetGpioPinV1 (gpioPin, invertFlg)***

**Arguments:**

| | |
|---|---|
| ***gpioPin*** | GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.<br>GPIO pin 0 = Not allocated |
| ***invertFlg*** | 1 = Invert the signal. |

**Returns:**

| | |
|---|---|
| ***result*** | ***0*** or ***E...*** Linux error code. |

## 6.3.18 SetGpioPinWatchdog()

This function is used to set the watchdog reset input GPIO pin.

**Syntax:**

***result = object. SetGpioPinWatchdog(gpioPin, invertFlg)***

**Arguments:**

| | |
|---|---|
| ***gpioPin*** | GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 may be used.<br>GPIO pin 0 = Not allocated |
| ***invertFlg*** | 1 = Invert the signal. |

**Returns:**

| | |
|---|---|
| ***result*** | ***0*** or ***E...*** Linux error code. |

## 6.3.19 SetPowerOnOff()

This function is used to set the power off and on delays.

**Syntax:**

***result = object. SetPowerOnOff(onDelaySecs, offDelaySecs)***

**Arguments:**

| | |
|---|---|
| ***onDelaySecs*** | The number of seconds to delay before turning back on after entry to standby mode.<br>Set to 0 if it's to stay in standby. |
| ***offDelaySecs*** | The number of seconds to delay before turning off the output and going to standby mode. |

**Returns:**

| | |
|---|---|
| ***result*** | ***0*** or ***E...*** Linux error code. |

## 6.3.20 SetV2()

This function is used to set the V2 power control.

**Syntax:**

***result = object. SetV2(v2OnFlg, v1GtV2OnFlg)***

**Arguments:**

***v2OnFlg*** The state of the V2 switch, ***1*** = on, ***0*** = Off.

***v1GtV2OnFlg*** The state of the V1 greater than V2 condition to exit standby mode.
***0*** = Ignore.
***1*** = Exit standby mode when V1 > V2.

**Returns:**

***result*** ***0*** or ***E...*** Linux error code.

## 6.3.21 SetWatchdog()

This function is used to set the power cycle watchdog timer.

**Syntax:**

***result = object. SetWatchdog (time)***

**Arguments:**

***time*** The time in seconds before power cycling if a reset trigger has not been received.
0 = Watchdog disabled.
1-255 seconds

**Returns:**

***result*** ***0*** or ***E...*** Linux error code.

## 6.3.22 WatchdogReset()

This function is used to reset the power cycle watchdog.

**Syntax:**

***result = object. WatchdogReset(time)***

**Arguments:**

**Returns:**

***result*** ***0*** or ***E...*** Linux error code.

# 7. Warranty conditions

All fully assembled & tested products of AEL Microsystems Ltd are guaranteed for one year from the date of shipment against defects in materials & workmanship and perform in accordance with applicable specifications. AEL Microsystems Ltd warrants that the application support SOFTWARE will perform substantially with the accompanying written materials for a period of ninety (90) days from the date of receipt.

This warranty does not extend to products which have been altered or repaired by persons other than persons authorised by AEL Microsystems Ltd, or to products that have been subjected to misuse, abuse, neglect, improper installation or application, accident, disaster, or modification not approved by written instructions from AEL Microsystems Ltd.

Final determination of the suitability of this product for the use contemplated by the buyer is the sole responsibility of the buyer and AEL Microsystems Ltd shall not be responsible for its suitability and assumes no liability arising out of the use or application of the device described herein.

In the event that this product fails to operate as warranted, the buyer shall obtain a return number from AEL Microsystems Ltd and forward the product in suitable packaging with a detailed failure report to AEL Microsystems Ltd, the cost of transportation being the responsibly of the buyer. The returned product will be repaired or replaced at the discretion of AEL Microsystems Ltd.

While every effort is made to repair or replace any item as quickly as possible, no guarantees can be made for the time taken, & AEL Microsystems Ltd cannot be held responsible for any loss or inconvenience caused.

# 8. Notes