PIXIE BOARD
ADC8-DAC2 ANALOGUE I/O 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

1. Contents

1. **Contents** 3

2. **Introduction** 6
   - 2.1 Caution 6
   - 2.2 Forward note 6

3. **Common concept** 7
   - 3.1 Control overview 7
   - 3.2 More SPI devices 8
   - 3.3 Configurable 8
   - 3.4 Stackable 8
   - 3.5 Updatable 8

4. **Hardware details** 10
   - 4.1 Specification. 10
   - 4.2 Input pin overview. 10
   - 4.3 I/O connections. 11

5. **Getting Started** 13
   - 5.1 Insulate USB connector housing on Raspberry Pi 4 13
   - 5.2 Mounting the boards. 14
   - 5.3 Set the boards identity. 15
   - 5.4 Power up and LED status. 16
   - 5.5 Principle of operation. 16
   - 5.6 Data sampling principle of operation. 16
   - 5.7 Input and output data calibrations. 17
   - 5.8 Configuration and test functions. 17
     - 5.8.1 Board specific configuration. 17
     - 5.8.2 Board specific test utilities. 19
     - 5.8.3 Board calibrations. 19

6. **Software support** 20
   - 6.1 ADC8-DAC2 Analogue I/O board ‘C’ library support functions 20
     - 6.1.1 PixieAdc8Dac2AdcRead() 20
     - 6.1.2 PixieAdc8Dac2AdcToEng() 21
     - 6.1.3 PixieAdc8Dac2Construct() 21
     - 6.1.4 PixieAdc8Dac2DefaultCalibration() 22
     - 6.1.5 PixieAdc8Dac2DefaultCalibrations() 22
     - 6.1.6 PixieAdc8Dac2Destroy() 22
     - 6.1.7 PixieAdc8Dac2EngToDac() 23
     - 6.1.8 PixieAdc8Dac2GetAdcConvstSource() 23
     - 6.1.9 PixieAdc8Dac2GetAdcOsBits() 24
     - 6.1.10 PixieAdc8Dac2GetAdcRange() 24
     - 6.1.11 PixieAdc8Dac2GetAdcReady() 25
     - 6.1.12 PixieAdc8Dac2GetCalibrations() 25
     - 6.1.13 PixieAdc8Dac2GetDacRange() 25
     - 6.1.14 PixieAdc8Dac2GetDefaultCalibration() 26
     - 6.1.15 PixieAdc8Dac2GetTimedAdc() 26
     - 6.1.16 PixieAdc8Dac2Initialise() 27
     - 6.1.17 PixieAdc8Dac2SaveSettings() 27
     - 6.1.18 PixieAdc8Dac2SetAdcConvstSource() 28

6.1.19 PixieAdc8Dac2SetAdcOsBits() 28
6.1.20 PixieAdc8Dac2SetAdcRange() 29
6.1.21 PixieAdc8Dac2SetAdcRunStop() 29
6.1.22 PixieAdc8Dac2SetCalibration() 30
6.1.23 PixieAdc8Dac2SetCeDecode() 31
6.1.24 PixieAdc8Dac2DacLoadA() 32
6.1.25 PixieAdc8Dac2DacLoadAB() 32
6.1.26 PixieAdc8Dac2DacLoadB() 32
6.1.27 PixieAdc8Dac2SetDacRange() 33
6.1.28 PixieAdc8Dac2SetIrq() 34
6.1.29 PixieAdc8Dac2SetSamplingMode() 35
6.1.30 PixieAdc8Dac2SetTimedAdc() 36
6.2 ADC8-DAC2 Analogue I/O board ‘C++’ library support functions 37
6.2.1 PixieBoardAdc8Dac2() 37
6.2.2 Initialise() 37
6.2.3 Close() 38
6.2.4 AdcRead() 38
6.2.5 DefaultCalibration() 38
6.2.6 DefaultCalibrations() 39
6.2.7 EngToDac() 39
6.2.8 GetAdcConvstSource() 39
6.2.9 GetAdcOsBits() 40
6.2.10 GetCalibration() 40
6.2.11 GetCalibrations() 41
6.2.12 GetCalibrationsPtr() 41
6.2.13 GetInputRange() 41
6.2.14 GetOutputRange() 42
6.2.15 GetTimedAdc() 42
6.2.16 GetValue() 42
6.2.17 GetValues() 43
6.2.18 GetValuesPtr() 43
6.2.19 IsAdcReady() 43
6.2.20 SaveSettings() 44
6.2.21 SetAdcConvstSource() 44
6.2.22 SetAdcOsBits() 44
6.2.23 SetCalibration() 45
6.2.24 SetCeDecode() 46
6.2.25 SetInputRange() 47
6.2.26 SetIrq() 47
6.2.27 SetOutputRange() 48
6.2.28 SetSamplingMode () 48
6.2.29 SetTimedAdc() 49
6.2.30 SetValue() 49
6.2.31 SetValues() 49
6.3 ADC8-DAC2 Analogue I/O board ‘Python’ library support functions 50
6.3.1 PyPixieBoardAdc8Dac2() 50
6.3.2 Initialise() 50
6.3.3 Close() 50
6.3.4 AdcRead() 51
6.3.5 DefaultCalibration() 51
6.3.6 DefaultCalibrations() 51
6.3.7 GetAdcConvstSource() 52
6.3.8 GetAdcOsBits() 52

6.3.9 GetCalibration() 53
6.3.10 GetCalibrations() 53
6.3.11 GetInputRange() 53
6.3.12 GetOutputRange() 54
6.3.13 GetTimedAdc() 54
6.3.14 GetValue() 54
6.3.15 GetValues() 55
6.3.16 IsAdcReady() 55
6.3.17 IsAdcReady() 55
6.3.18 SaveSettings() 56
6.3.19 SetAdcConvstSource() 56
6.3.20 SetAdcOsBits() 56
6.3.21 SetCalibration() 57
6.3.22 SetCeDecode() 58
6.3.23 SetInputRange() 58
6.3.24 SetIrq() 59
6.3.25 SetOutputRange() 59
6.3.26 SetSamplingMode() 60
6.3.27 SetTimedAdc() 60
6.3.28 SetValue() 61
6.3.29 SetValues() 61

7. Warranty conditions 63

8. Notes 64

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

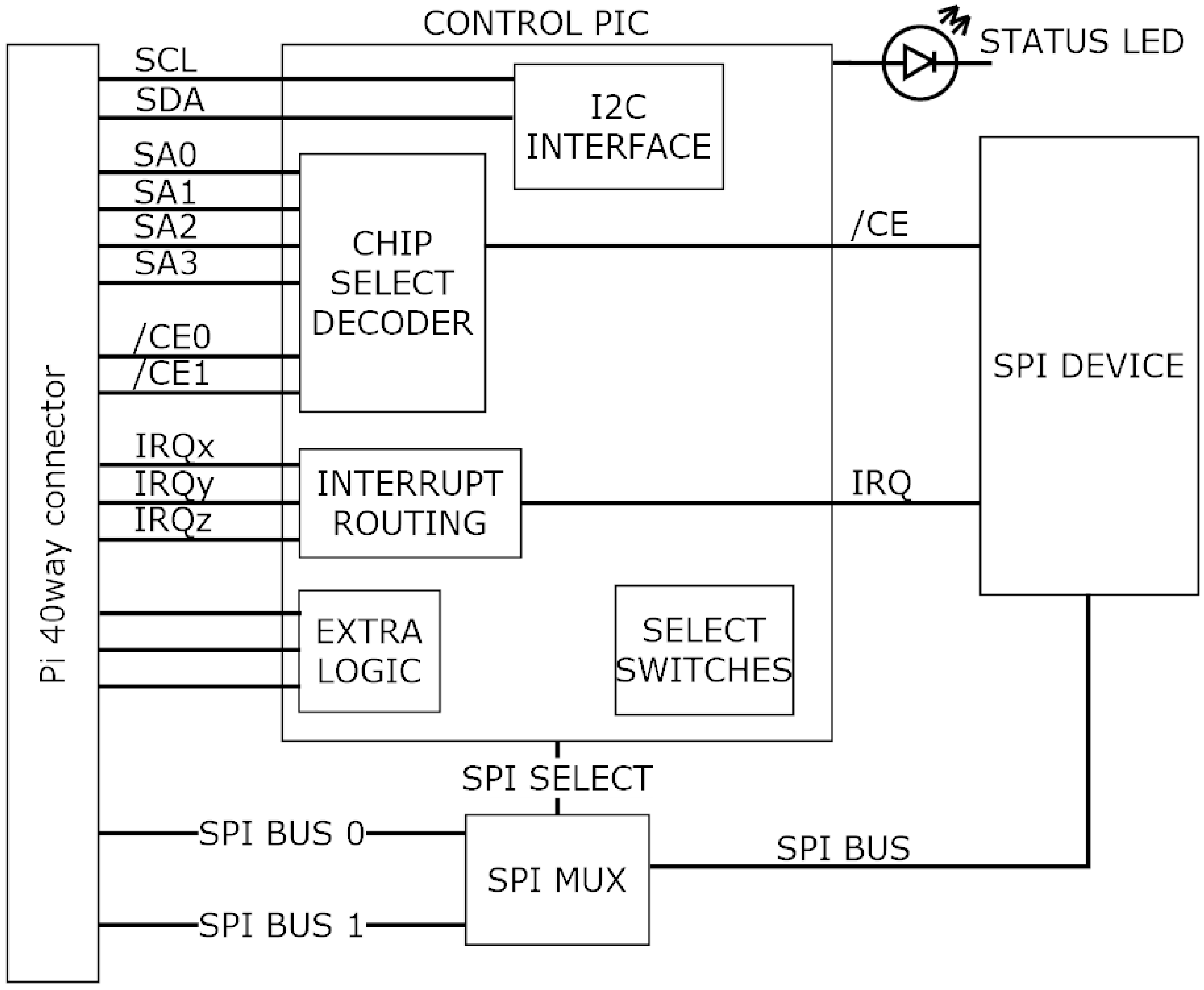
This shows the control overview of the **PIXIE** board.

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 **ADC8-DAC2** is a **PIXIE** board that can input and output analogue signals. The inputs can convert a signal from -5V to +5V or -10V to +10V and the outputs can provide a signal from 0 to +5V or 0 to +10V.
The inputs and outputs are protected against transient voltage and short circuit protected.

The ADC is a Texas Instruments ADS8588 device with 16bit resolution and has 8 differential inputs that are converted all at the same time so no need for channel multiplexing. As they are differential the -Ain signal can be connected remotely from the board to reduce line drop signal errors and improve measurement.
The ADC can be set to convert over the range of either ±5V or ±10V, it can also be set to perform oversampling of each signal up to 64times and also a 3rd order low pass filter.
For full details of the device its datasheet should be consulted.

The DAC is a Burr Brown DAC8522 device whit provides 2 independent channels of 16bit resolution.
Using an external amplifier two ranges of +5V and +10V are provided.
For full details of the device its datasheet should be consulted.

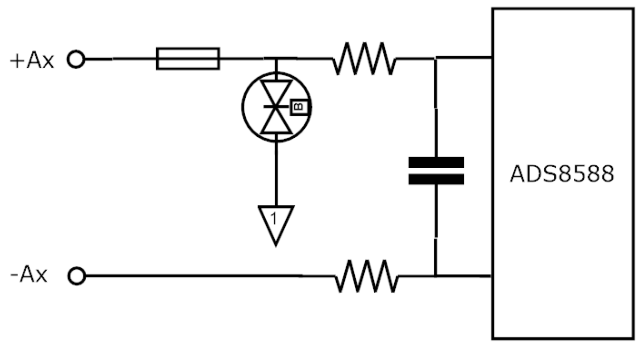
4.1 Specification.

Number of inputs	8
Voltage range	±5V or ±10V
Input impedance	1Mohms
Input protection	±12V, includes PTC fuse

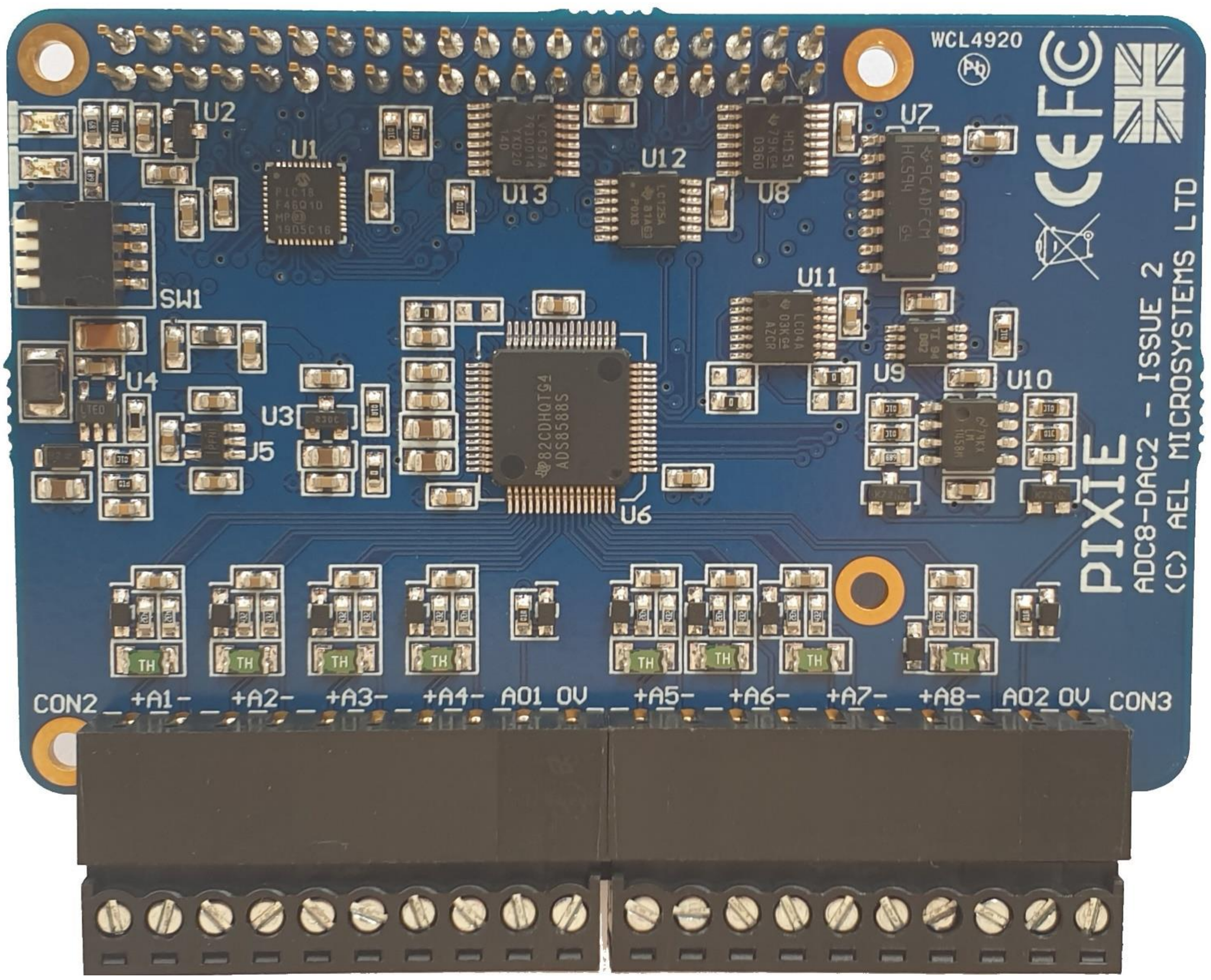
Number of outputs	2
Voltage range	+5V or +10V
Output impedance	100ohms
Output protection	+12V
Power consumption	50mA
I2C speed	<=100kHz
SPI speed	<=10MHz
Temperature	0-70C

4.2 Input pin overview.

This shows the input circuit.



4.3 I/O connections.



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

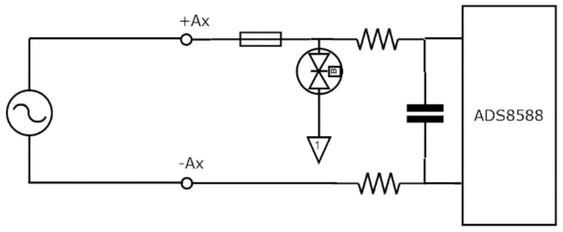
CON2 is a 10-way connector that covers inputs 1 thru 4, and output 1
CON3 is a 10-way connector that covers inputs 5 thru 8, and output 2

Signals:
+Ax- of the connector are each of the 8 analogue inputs.

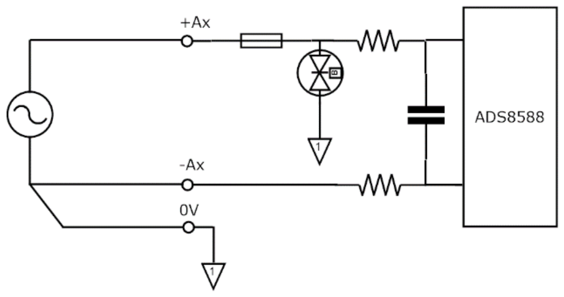
AOx of the connector is the analogue output.

0V of the connector needs to be connected to the common 0V of the system.

Connecting of a differential voltage source.



Connecting of a single ended voltage source.

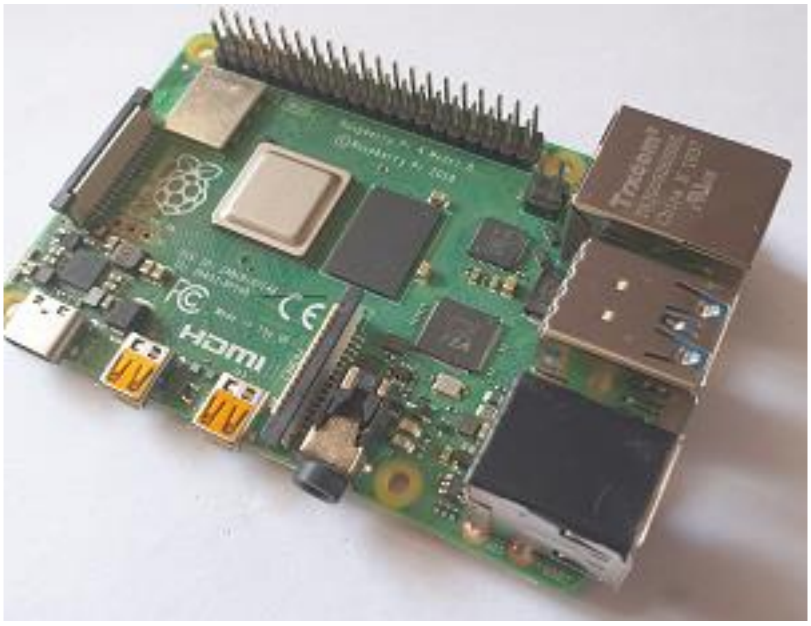


5. Getting Started

5.1 Insulate USB connector housing on Raspberry Pi 4

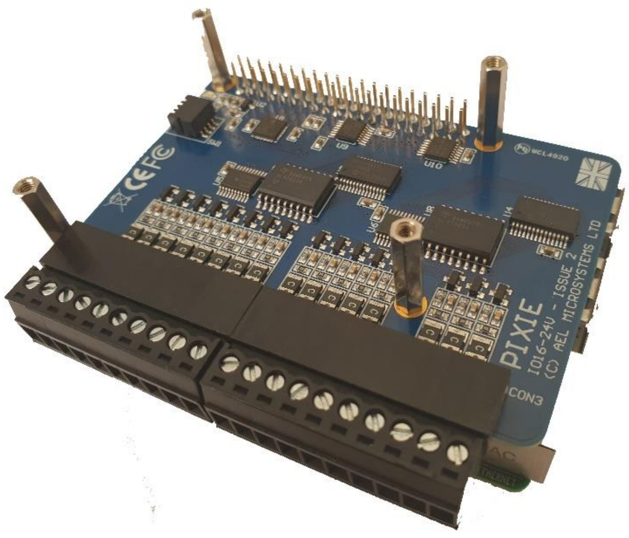
WARNING

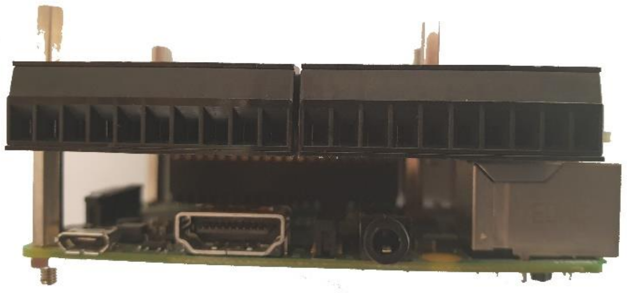
The Ethernet and USB connectors were swapped on the Raspberry Pi 4 which means the clearance between the terminal connectors on the PIXIE board and the metal body of the USB connector is very close and, in some circumstances, could short out the terminals, which is not so good.
To prevent this, add a 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.

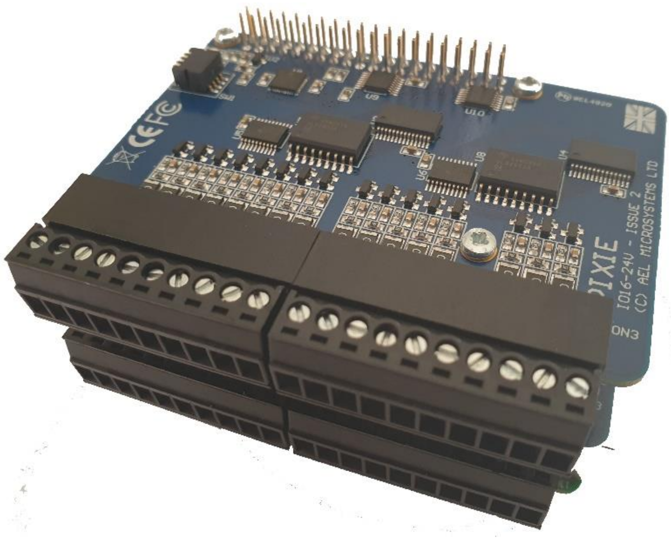


5.2 Mounting the boards.

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.

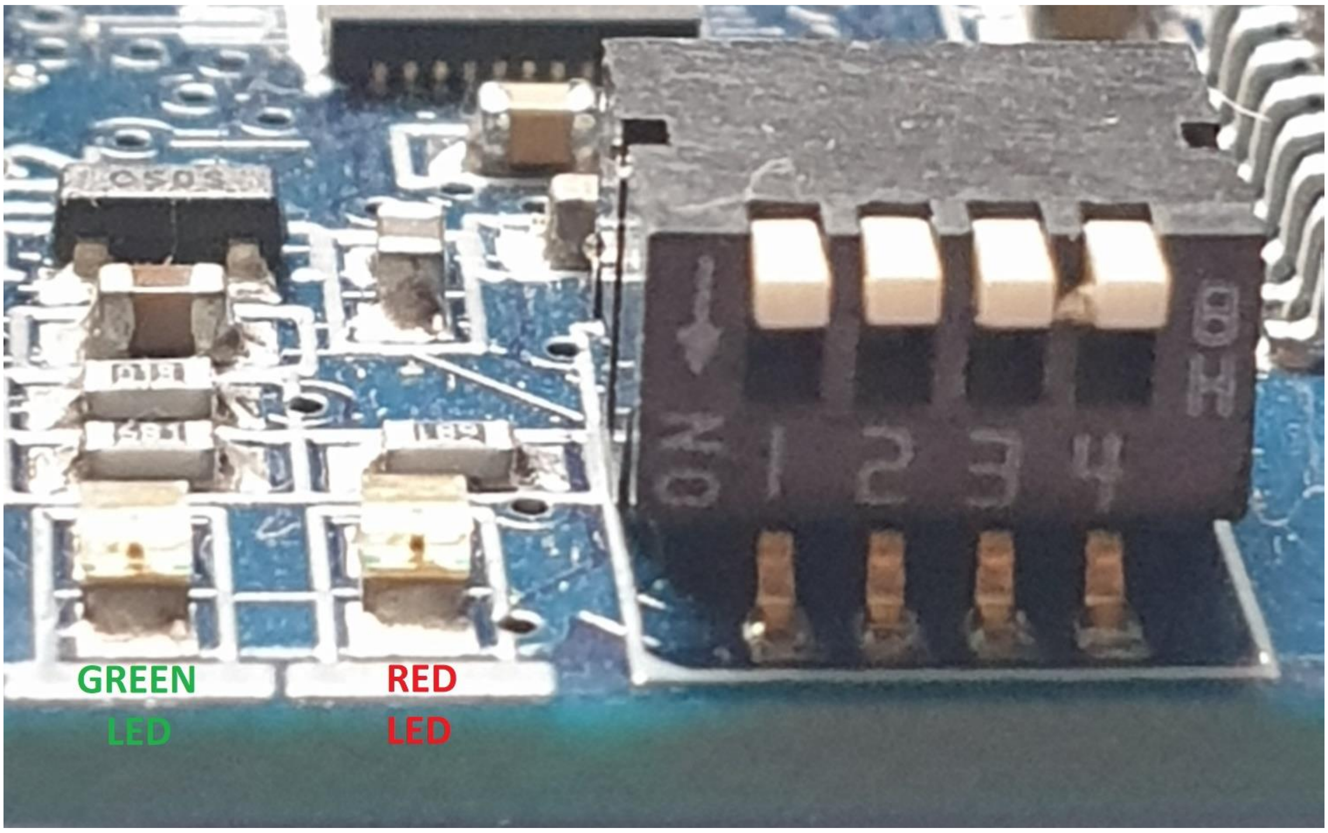






5.3 Set the boards identity.

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 Identifier”.

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,	the board is not configured.
2 Flashes,	the board has an invalid identity, contact the manufacturer 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.

5.5 Principle of operation.

The board uses a ADS8588 8 channel ADC converter and a DAC8552 two channel DAC.
Onboard logic controls the start and end of ADC conversions, the DAC is a straightforward write to update it.
Both devices are accessed using the SPI and a full set of supporting functions are available, see later in this manual.
These devices use the board references /CE0, /IRQ0 for the ADC and /CE1 for the DAC.

5.6 Data sampling principle of operation.

To sample input data the ADC conversion needs to be started and when it completes a GPIO signal is used to indicate the end of conversion or to generate an interrupt so the data can read out of the device.
First configure to interrupt of the board to select the GPIO you want to use for end of conversion polling or interrupt.

There are three types of start conversion available, manual, hardware or timed.

Manual trigger
Call the function ***PixieAdc8Dac2SetConvstSource()*** using the conversion source ***PXBC_ADC8_DAC2_CONVST_SW***, this indicates the trigger is software.
Then call the function ***PixieAdc8Dac2StartAdc()*** using the action ***PXAD_ADC_START***, then poll or wait for the end of conversion interrupt, reading the values out of the device will reset the end of conversion output.
Repeat the start process for the next sample.

Hardware trigger
Call the function ***PixieAdc8Dac2SetConvstSource()*** using the conversion source as an external hardware GPIO signal to start the conversion which is a high to low level transition.
The following GPIO pins values can be used, 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 25, 26.

There are two ways of driving the GPIO pin, one is using your application to toggle the pin or by another external source which is driving the GPIO pin.

To drive the GPIO pin from your application call the function to configure the GPIO pin as an output then call the ***PixieAdc8Dac2StartAdc(),*** using the action ***PXAD_ADC_START***, along with the GPIO control structure and the pin you are using.

Synchronous triggering of multiple boards can be achieved by setting their conversion start pin to the same GPIO pin then toggling the pin.

Timed trigger
A time base timer in the PIC is used to repeatedly start conversions. The output of the timer can also be routed to one of the GPIO pins to signal to other boards to synchronously start conversions at the same time.

Call the function ***PixieAdc8Dac2SetConvstSource()*** using the conversion source ***PXBC_ADC8_DAC2_CONVST_TIMED***, this indicates the trigger is timed.

Call the function ***SetTimedAdc()*** to set the time base value in uS, along with an optional GPIO pin to output the trigger signal to for other boards to use.

Call the function ***PixieAdc8Dac2StartAdc()*** using the action ***PXAD_ADC_START***, this will start the time base timer. Your application can then poll or wait for the end of conversion interrupt, reading the values out of the device will reset the end of conversion output.

To stop the time base call the function ***PixieAdc8Dac2StartAdc()*** using the action ***PXAD_ADC_STOP.***

5.7 Input and output data calibrations.

The board supports calibration values for both the ADC inputs and the DAC outputs which allow you to scale the raw data into sensible engineered for inputs and engineered to raw values for outputs.
These calibrations are merely 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.

5.8 Configuration and test functions.

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

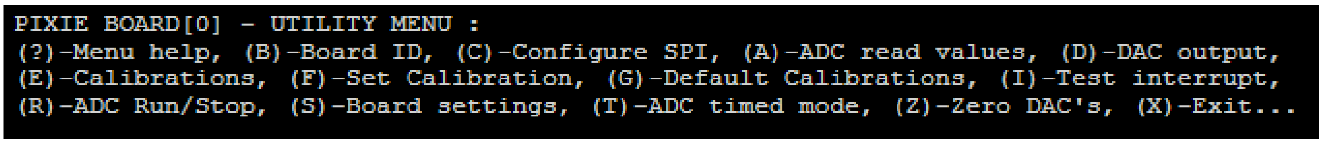
5.8.1 Board specific configuration.

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

They are:

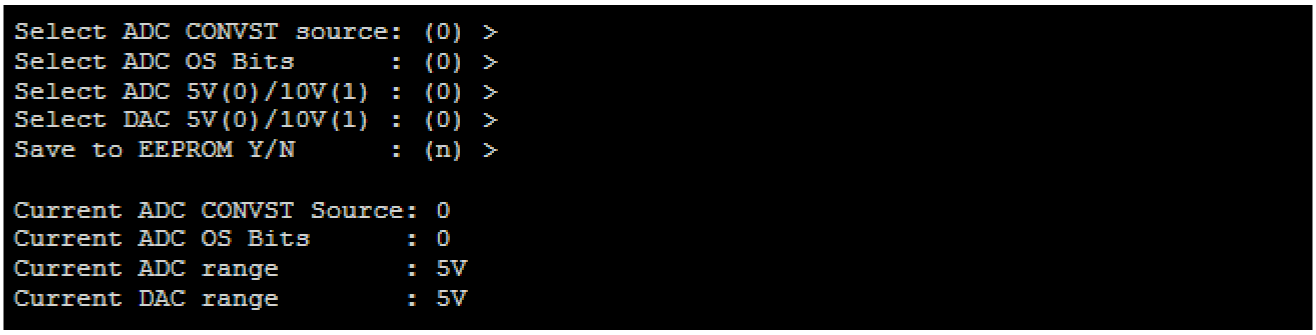
- Set the number of oversampling bits for the ADC.
- Set the input range of the ADC, either $\pm$5V or $\pm$10V.
- Set a GPIO pin to act as an external ADC start conversion signal.
- Set the DAC output range either +5V or +10V.
- Set and recover input and output calibrations.

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, (C)-Configure SPI, (A)-ADC read values, (D)-DAC output,
(E)-Calibrations, (F)-Set Calibration, (G)-Default Calibrations, (I)-Test interrupt,
(R)-ADC Run/Stop, (S)-Board settings, (T)-ADC timed mode, (Z)-Zero DAC's, (X)-Exit...
```

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



```
Select ADC CONVST source: (0) >
Select ADC OS Bits      : (0) >
Select ADC 5V(0)/10V(1) : (0) >
Select DAC 5V(0)/10V(1) : (0) >
Save to EEPROM Y/N      : (n) >

Current ADC CONVST Source: 0
Current ADC OS Bits      : 0
Current ADC range        : 5V
Current DAC range        : 5V
```

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

Select ADC CONVST source
This sets the GPIO pin used as a trigger source for externally starting an ADC conversion.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 25, 26 can be used

Select ADC OS Bits This sets the ADC over sampling bits.
0 = None.
1 = x2.
2 = x4.
3 = x8.
4 = x16.
5 = x32.
6 = x64.

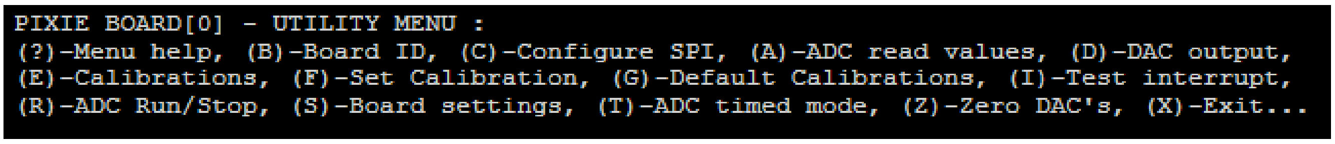
Select ADC 5V(0)/10V(1)
This sets the input range of the ADC conversions.
0 = +5V
1 = +10V

Select DAC 5V(0)/10V(1)
This sets the output range of the DAC's.
0 = +5V
1 = +10V

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.



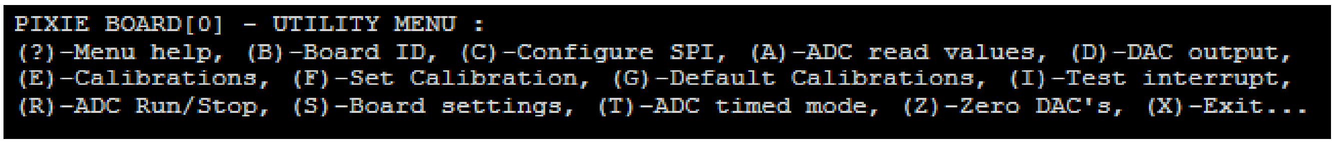
```
PIXIE BOARD[0] - UTILITY MENU :
(?)-Menu help, (B)-Board ID, (C)-Configure SPI, (A)-ADC read values, (D)-DAC output,
(E)-Calibrations, (F)-Set Calibration, (G)-Default Calibrations, (I)-Test interrupt,
(R)-ADC Run/Stop, (S)-Board settings, (T)-ADC timed mode, (Z)-Zero DAC's, (X)-Exit...
```

A full description of these are given by the **(?)-Menu help**, however the most important on which needs to be set before any of these will work is the **(C)-Configure SPI**. This sets the SPI bus and the SPI channels that the board has been configured to use so that the test utilities know which bus and channel to use in the test function.

(B)-Board ID	Change the current board.
(C)-Configure SPI	Set the SPI bus and chip enable channels to use for the board utility menu functions.
(A)-ADC read values	Displays the current values of the ADC.
(D)-DAC output	Set a DAC's output level, value = 0 to 65535.
(I)-Test interrupt	Test the completion of ADC conversion interrupt.
(R)-ADC Run/Stop	Start and stop the ADC conversions.
(S)-Board settings	Change the board settings. See previous section.
(T)-ADC timed mode	Change the ADC timed mode settings. These are the pin used as a master output to other boards for sync'd conversion starts and the scan time in uS.
(Z)-Zero DAC's	Zero both DAC outputs.

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, (C)-Configure SPI, (A)-ADC read values, (D)-DAC output,
(E)-Calibrations, (F)-Set Calibration, (G)-Default Calibrations, (I)-Test interrupt,
(R)-ADC Run/Stop, (S)-Board settings, (T)-ADC timed mode, (Z)-Zero DAC's, (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. In this case for the inputs the values would scale the raw ADC value to +-5V. In this case for the outputs the values would scale the 0-5V DAC value to its raw value.

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 ADC8-DAC2 Analogue I/O board 'C' library support functions

This group contains 'C' functions for supporting the ADC8-DAC2 analogue I/O 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 ***PixieAdc8Dac2Ctrl_t*** which is declared one for each board used and contains all the control parameters used for the board, it is constructed using the ***PixieAdc8Dac2Construct()*** function and can be optionally destroyed using the ***PixieAdc8Dac2Destroy()*** function.

6.1.1 PixieAdc8Dac2AdcRead()

This function is used to read all 8 raw converted inputs of the ADC and stores them in the ***readings.rawReading[]*** of the ***PixieAdc8Dac2Ctrl_t*** structure.
Precede this function with a call to ***PixieAdc8Dac2StartAdc()*** to trigger a conversion.
The SPI channel for the ADC needs to have been opened and initialised using the ***PixieAdc8Dac2Initialise()*** function to access the ADC via the SPI bus.

Syntax:
int_t PixieAdc8Dac2AdcRead(PixieAdc8Dac2Ctrl _t* pCtrl);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use. Ensure this structure is initialised and the channel is open before calling the function.

Returns:

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

6.1.2 PixieAdc8Dac2AdcToEng()

This function is used to scale the raw ***readings.rawReading[]*** into engineered values ***readings.engReading[]*** in the ***PixieAdc8Dac2Ctrl_t*** structure.
The **PixieAdc8Dac2GetCalibrations()** needs to be called prior to this function to load in the calibrations from the board otherwise ***EINVAL*** is returned.

Syntax:

Int_t PixieAdc8Dac2AdcToEng(PixieAdc8Dac2Ctrl _t* pCtrl);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.

Returns:

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

6.1.3 PixieAdc8Dac2Construct()

This function is used to initialise the ***PixieAdc8Dac2Ctrl_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 PixieAdc8Dac2Construct(PixieAdc8Dac2Ctrl _t* pCtrl, uint_t i2cChannel, uint16_t i2cAddress);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
i2cChannel	I2C channel to use for i2C access, default = 1, i.e. i2c-1.
i2cAddress	I2C address for the board to access.

Returns:

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

6.1.4 PixieAdc8Dac2DefaultCalibration()

This function is used to set default calibration values for a full scale of 5.0V for both the ADC inputs and DAC outputs when used with the raw to engineered conversion functions.

Syntax:

int_t PixieAdc8Dac2DefaultCalibration(PixieAdc8Dac2Ctrl _t* pCtrl, uint8_t calibNumber);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
calibNumber	The number of the calibration to default.

Returns:

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

6.1.5 PixieAdc8Dac2DefaultCalibrations()

This function is used to default all the boards calibrations.

Syntax:

int_t PixieAdc8Dac2DefaultCalibrations(PixieAdc8Dac2Ctrl _t pCtrl);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.

Returns:

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

6.1.6 PixieAdc8Dac2Destroy()

This function is used to destroy the ***PixieAdc8Dac2Ctrl _t*** structure when the board is no longer required.

Syntax:

int_t PixieAdc8Dac2Destroy(PixieAdc8Dac2Ctrl _t* pCtrl);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.

Returns:

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

6.1.7 PixieAdc8Dac2EngToDac()

This function is used to convert an engineered value into a DAC value.

Syntax:

```
int_t PixieAdc8Dac2EngToDac(
                 PixieAdc8Dac2Ctrl _t* pCtrl,
                 uint8_t dacChannel,
                 double_t engValue,
                 uint32_t* pValue);
```

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
dacChannel	The number of the DAC output channel.
engValue	Is the engineered value.
pValue	Is a pointer to the variable to put the converted result into.

Returns:

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

6.1.8 PixieAdc8Dac2GetAdcConvstSource()

This function is used to read the current setting of the ADC external trigger GPIO pin source.

Syntax:

```
int_t PixieAdc8Dac2GetConvstSource(PixieAdc8Dac2Ctrl _t* pCtrl, uint8_t* pSrcValue);
```

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
pSrcValue	Is a pointer to return the current value in. ***PXBC_ADC8_DAC2_CONVST_SW*** = Software trigger. ***PXBC_ADC8_DAC2_CONVST_TIME*** = Timed trigger. External trigger GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 25, 26 can be used.

Returns:

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

6.1.9 PixieAdc8Dac2GetAdcOsBits()

This function is used to read the current setting of the ADC oversampling control bits.

Syntax:

int_t PixieAdc8Dac2GetAdcOsBits(PixieAdc8Dac2Ctrl _t* pCtrl, uint8_t* pOsValue);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
pOsValue	Is a pointer to return the current value in. 0 = No oversampling. 1 = x2 oversampling. 2 = x4 oversampling. 3 = x8 oversampling. 4 = x16 oversampling. 5 = x32 oversampling. 6 = x64 oversampling.

Returns:

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

6.1.10 PixieAdc8Dac2GetAdcRange()

This function is used to read the current setting of the ADC range control.

Syntax:

int_t PixieAdc8Dac2GetAdcRange(PixieAdc8Dac2Ctrl _t* pCtrl, PixieBcRange_t* pRange);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
pRange	Is a pointer to return the range in. ***PXBC_RNG_5V*** or ***PXBC_RNG_10V***.

Returns:

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

6.1.11 PixieAdc8Dac2GetAdcReady()

This function is used to read the current ADC ready status.

Syntax:

int_t PixieAdc8Dac2GetAdcReady(PixieAdc8Dac2Ctrl _t* pCtrl, uint8_t* pReady);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
pReady	Is a pointer to return the ready status in, TRUE = Ready, FALSE = busy.

Returns:

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

6.1.12 PixieAdc8Dac2GetCalibrations()

This function is used to read all the calibrations from the board and save in the ***PixieAdc8Dac2Ctrl _t*** structure ***calibrations***.

Syntax:

int_t PixieAdc8Dac2GetCalibrations(PixieAdc8Dac2Ctrl _t* pCtrl);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.

Returns:

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

6.1.13 PixieAdc8Dac2GetDacRange()

This function is used to read the current setting of the DAC range control.

Syntax:

int_t PixieAdc8Dac2GetDacRange(PixieAdc8Dac2Ctrl _t pCtrl*, PixieBcRange_t * pRange);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
pRange	Is a pointer to return the range in. ***PXBC_RNG_5V*** or ***PXBC_RNG_10V***.

Returns:

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

6.1.14 PixieAdc8Dac2GetDefaultCalibration()

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

Syntax:

void PixieAdc8Dac2GetDefaultCalibration(uint8_t calibNumber, PixieAnlCalib_t* pCalibration);

Arguments:

calibNumber	The number of the calibration to default.
pCalibration	Is a pointer to the ***PixieAnlCalib _t*** structure to fill in.

Returns:

Nothing.

6.1.15 PixieAdc8Dac2GetTimedAdc()

This function is used to read the current setting of the ADC timed conversion generator and master trigger GPIO pin source.

Syntax:

```
int_t PixieAdc8Dac2GetTimedAdc(
                PixieAdc8Dac2Ctrl _t* pCtrl ,
                uint32_t* pTimeBetweenUs,
                PixieGpioPin_t* pGpioPin)
```

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
pTimeBetweenUs	Is a pointer to return the time between sample in.
pGpioPin	Is a pointer to return the current master trigger pin in. GPIO pins 5, 6, 12, 13, 19, 20, 21, 25 can be used

Returns:

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

6.1.16 PixieAdc8Dac2Initialise()

This function is used to initialise the board using the SPI interfaces provided.

Syntax:

```
int_t PixieAdc8Dac2Initialise(
                    PixieAdc8Dac2Ctrl _t* pCtrl ,
                    PixieSpiCtrl_t* pSpiAdc,
                    PixieSpiCtrl_t* pSpiDac);
```

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
pSpiAdc	Is a pointer to the opened SPI channel to be used by the ADC functions. Ensure the channel is initialised and the channel is open. It can be ***NULL*** if no ADC parts of the board are to be accessed.
pSpiDac	Is a pointer to the opened SPI channel to be used by the DAC functions. Ensure the channel is initialised and the channel is open. It can be ***NULL*** if no DAC parts of the board are to be accessed.

Returns:

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

6.1.17 PixieAdc8Dac2SaveSettings()

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 PixieAdc8Dac2SaveSettings(PixieAdc8Dac2Ctrl _t* pCtrl);
```

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.

Returns:

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

6.1.18 PixieAdc8Dac2SetAdcConvstSource()

This function is used to set the current value of the ADC external trigger GPIO pin source.

Syntax:

int_t PixieAdc8Dac2SetConvstSource(PixieAdc8Dac2Ctrl _t* pCtrl, uint8_t srcValue);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
srcValue	The value of the external trigger source pin. ***PXBC_ADC8_DAC2_CONVST_SW*** = Software trigger. ***PXBC_ADC8_DAC2_CONVST_TIME*** = Timed trigger. External trigger GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 25, 26 can be used.

Returns:

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

6.1.19 PixieAdc8Dac2SetAdcOsBits()

This function is used to set the current value of the ADC oversampling control bits.

Syntax:

int_t PixieAdc8Dac2SetAdcOsBits(PixieAdc8Dac2Ctrl _t* pCtrl, uint8_t osValue);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
osValue	The value of the oversampling bits. 0 = No oversampling. 1 = x2 oversampling. 2 = x4 oversampling. 3 = x8 oversampling. 4 = x16 oversampling. 5 = x32 oversampling. 6 = x64 oversampling.

Returns:

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

6.1.20 PixieAdc8Dac2SetAdcRange()

This function is used to set the current value of the ADC range control.

Syntax:

int_t PixieAdc8Dac2SetAdcRange(PixieAdc8Dac2Ctrl _t* pCtrl, PixieBcRange_t range);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
range	The value of the range to set. ***PXBC_RNG_5V*** or ***PXBC_RNG_10V***.

Returns:

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

6.1.21 PixieAdc8Dac2SetAdcRunStop()

This function is used to set the ADC running or stopped.

Syntax:

int_t PixieAdc8Dac2SetAdcRange(PixieAdc8Dac2Ctrl _t* pCtrl, uint8_t runFlg);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
runFlg	***TRUE*** = Start timed or trigger software sample, ***FALSE*** = stop.

Returns:

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

6.1.22 PixieAdc8Dac2SetCalibration()

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

Syntax:

```
int_t PixieAdc8Dac2SetCalibrations(
                        PixieAdc8Dac2Ctrl _t* pCtrl,
                        uint8_t calibNumber,
                        float_t zeroValue,
                        float_t spanValue);
```

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
calibNumber	The number of the calibration to set.
zeroValue	The zero value.
spanValue	The span value.

Returns:

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

6.1.23 PixieAdc8Dac2SetCeDecode()

This function is used to set the board chip enable decoding.

Syntax:

```
int_t PixieAdc8Dac2SetCeDecode(
                    PixieAdc8Dac2Ctrl _t* pCtrl,
                    uint8_t usedAdcMask,
                    uint8_t polAdcMask,
                    uint8_t usedDacMask,
                    uint8_t polDacMask,
                    uint8_t* pAdrIds);
```

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
usedAdcMask	This is the used mask for the ADC and is made up of the OR of the following masks: ***PXBC_CEx_A0_M, PXBC_CEx_A1_M, PXBC_CEx_A2_M, PXBC_CEx_A3_M*** use of one or more to show which sub address lines to include. ***PXBC_CEx_CE0_M, PXBC_CEx_CE1_M, PXBC_CEx_CE2_M*** use only one of these to show which SPI CE to use. ***PXBC_CEx_SPI_0_M, PXBC_CEx_SPI_1_M*** use only one of these to show which SPI bus to use. NOTE: the board can only use one or the other for all devices.
polAdcMask	This is the polarity mask for the ADC and is made up of the OR of the following masks: ***PXBC_CEx_A0_M, PXBC_CEx_A1_M, PXBC_CEx_A2_M, PXBC_CEx_A3_M*** use of one or more to show which sub address lines to decode as active high
usedDacMask	Same as ***usedAdcMask*** shown above for this is for the DAC.
polDacMask	Same as ***polAdcMask*** shown above for this is for the DAC.
pAdrIds	Is a pointer to an array of 4 address identifiers used for the sub address decode. If using SPI bus 0 = ***PXBC_PI_GPIO22, PXBC_PI_GPIO23, PXBC_PI_GPIO24, PXBC_PI_GPIO27*** If using SPI bus 1 = ***PXBC_PI_GPIO5, PXBC_PI_GPIO6, PXBC_PI_GPIO12, PXBC_PI_GPIO13***

Returns:

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

6.1.24 PixieAdc8Dac2DacLoadA()

This function is used to set the value of the DAC output A (AO1).

Syntax:

int_t PixieAdc8Dac2DacLoadA(PixieAdc8Dac2Ctrl_t* pCtrl, uint16_t value);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
value	The value to set.

Returns:

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

6.1.25 PixieAdc8Dac2DacLoadAB()

This function is used to set the value of the DAC output A (AO1) and B (AO2) at the same time.

Syntax:

int_t PixieAdc8Dac2DacLoadAB(PixieAdc8Dac2Ctrl_t* pCtrl, uint16_t valueA, uint16_t valueB);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
valueA	The value to set for output A (AO1).
valueB	The value to set for output B (AO2).

Returns:

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

6.1.26 PixieAdc8Dac2DacLoadB()

This function is used to set the value of the DAC output B (AO2).

Syntax:

int_t PixieAdc8Dac2DacLoadB(PixieAdc8Dac2Ctrl_t* pCtrl, uint16_t value);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
value	The value to set.

Returns:

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

6.1.27 PixieAdc8Dac2SetDacRange()

This function is used to set the current value of the DAC range control.

Syntax:

int_t PixieAdc8Dac2SetDacRange(PixieAdc8Dac2Ctrl_t* pCtrl, PixieBcRange_t range);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
range	The value of the range to set. ***PXBC_RNG_5V*** or ***PXBC_RNG_10V***.

Returns:

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

6.1.28 PixieAdc8Dac2SetIrq()

This function is used to set the ADC device interrupt mapping.

Syntax:

int_t PixieAdc8Dac2SetIrq(
PixieAdc8Dac2Ctrl_t* pCtrl,
uint16_t usedMask,
bool_t activeLowFlg,
bool_t openDrainFlg);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
usedMask	This is the PI pin to use, select only one of the following masks: ***PXBC_IRQ_EN_GPIO5*** ***PXBC_IRQ_EN_GPIO6*** ***PXBC_IRQ_EN_GPIO12*** ***PXBC_IRQ_EN_GPIO13*** ***PXBC_IRQ_EN_GPIO19*** ***PXBC_IRQ_EN_GPIO20*** ***PXBC_IRQ_EN_GPIO21*** ***PXBC_IRQ_EN_GPIO25*** ***PXBC_IRQ_EN_GPIO18*** ***PXBC_IRQ_EN_GPIO17*** ***PXBC_IRQ_EN_GPIO16*** ***PXBC_IRQ_EN_GPIO26*** ***PXBC_IRQ_EN_GPIO22*** ***PXBC_IRQ_EN_GPIO23*** ***PXBC_IRQ_EN_GPIO24*** ***PXBC_IRQ_EN_GPIO27***
activeLowFlg	TRUE = IRQ to Pi is active low.
openDrainFlg	TRUE = IRQ to Pi is open drain.

Returns:

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

6.1.29 PixieAdc8Dac2SetSamplingMode()

This function is used to start the ADC conversion, also start and stop timed mode.

Syntax:

int_t PixieAdc8Dac2SetSamplingMode(
PixieAdc8Dac2Ctrl_t* pCtrl,
TePixieAdcStartAction action
PixieGpioCtrl_t* pGpioCtrl,
PixieGpioPin_t gpioPin);

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
action	Type of action. ***PXAD_ADC_START*** This will trigger a conversion if ***PixieAdc8Dac2SetConvstSource()*** is set to ***PXBC_ADC8_DAC2_CONVST_SW*** or ***PXBC_ADC8_DAC2_CONVST_TIMED*** ***PXAD_ADC_STOP*** This will stop any timed trigger conversions if ***PixieAdc8Dac2SetConvstSource()*** is set to ***PXBC_ADC8_DAC2_CONVST_TIMED*** ***PXAD_ADC_START_GPIO*** This will trigger a conversion using the following 2 parameters
pGpioCtrl	Is a pointer to the ***PixieGpioCtrl_t*** structure to use when triggering via a pin. Ensure this structure has been initialised before use. Used for ***PXAD_ADC_START_GPIO*** only otherwise ***NULL***
gpioPin	The GPIO pin to use as the conversion trigger. GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 25, 26 can be used Used for ***PXAD_ADC_START_GPIO*** only otherwise 0.

Returns:

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

6.1.30 PixieAdc8Dac2SetTimedAdc()

This function is used to set the current values of the ADC timed conversion generator and master trigger GPIO pin source.

Syntax:

```
int_t PixieAdc8Dac2SetTimedAdc(
                PixieAdc8Dac2Ctrl_t* pCtrl,
                uint32_t timeBetweenUs,
                PixieGpioPin_t gpioPin,);
```

Arguments:

pCtrl	Is a pointer to the ***PixieAdc8Dac2Ctrl _t*** structure to use.
timeBetweenUs	Time between sample conversions in uS.
gpioPin	The GPIO pin to set as the master conversion trigger output. GPIO pins 5, 6, 12, 13, 19, 20, 21, 25 can be used

Returns:

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

6.2 ADC8-DAC2 Analogue I/O board ‘C++’ library support functions

This group contains ‘C++’ functions for supporting the ADC8-DAC2 analogue I/O 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 ***PixieBoardAdc8Dac2***
They all require the ***#include <PixieLib.hpp>*** header file.

6.2.1 PixieBoardAdc8Dac2()

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

Syntax:
PixieBoardAdc8Dac2(uint_t i2cChannel, uint_t i2cAddress, uint8_t busId);

Arguments:

i2cChannel	I2C channel to use for i2C access, default = 1, i.e. i2c-1.
i2cAddress	I2C address for the board to access.
busId	The SPI bus to use, ***0, 1, …***

6.2.2 Initialise()

This function is used to open a path to an SPI device given its channel, clock speed for the ADC and DAC to communicate over.

Syntax:
int_t Initialise(uint_t channel0, uint_t channel 1, uint32_t speedHz);

Arguments:

channel0	The SPI channel to use for the ADC, ***PIXIE_SPI_CHAN_0, PIXIE_SPI_CHAN_1***…
channel1	The SPI channel to use for the DAC, ***PIXIE_SPI_CHAN_0, PIXIE_SPI_CHAN_1***…
speedHz	The SPI clocking speed Hz, e.g. 100000 = 100kHz. (Optional, default 10000000)

Returns:

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

6.2.3 Close()

This function is used to close any open paths to the ADC and DAC devices.

Syntax:
int_t Close(void);

Arguments:

Returns:

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

6.2.4 AdcRead()

Read the ADC values and optionally return the values.
Precede this function with a call to ***SetSamplingMode ()*** to trigger a conversion then wait for the nominated GPIO data ready or interrupt to be triggered.

Syntax:
Int_t **AdcRead** ***(int32_t* pValues = NULL, double_t* pValuesEng = NULL);***

Arguments:

pValues	Pointer for the raw values (optional).
pValuesEng	Pointer for the Engineered values (optional).

Returns:

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

6.2.5 DefaultCalibration()

This function is used to set default calibration values for a full scale of 5.0V for both the ADC inputs and DAC outputs when used with the raw to engineered conversion functions.

Syntax:
int_t DefaultCalibration(uint8_t calibNumber);

Arguments:

calibNumber	Number of the calibration to default.

Returns:

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

6.2.6 DefaultCalibrations()

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

Syntax:
int_t DefaultCalibrations(void);

Arguments:

Returns:

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

6.2.7 EngToDac()

Convert an engineered value into a DAC value.

Syntax:
uint32_t EngToDac(uint8_t dacChannel, float_t engValue);

Arguments:

dacValue	The DAC channel to convert.
engValue	Engineered value to convert.

Returns:
The converted value.

6.2.8 GetAdcConvstSource()

This function is used to read the current setting of the ADC external trigger GPIO pin source.

Syntax:
int_t GetAdcConvstSource(uint8_t* pSrcValue);

Arguments:

pSrcValue	Is a pointer to return the current value in. ***PXBC_ADC8_DAC2_CONVST_SW*** = Software trigger. ***PXBC_ADC8_DAC2_CONVST_TIME*** = Timed trigger. External trigger GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 25, 26 can be used.

Returns:

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

6.2.9 GetAdcOsBits()

This function is used to read the current setting of the ADC oversampling control bits.

Syntax:

int_t GetAdcOsBits(uint8_t* pOsValue);

Arguments:

pOsValue	Is a pointer to return the current value in. 0 = No oversampling. 1 = x2 oversampling. 2 = x4 oversampling. 3 = x8 oversampling. 4 = x16 oversampling. 5 = x32 oversampling. 6 = x64 oversampling.

Returns:

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

6.2.10 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.
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.

6.2.11 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.

6.2.12 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.

6.2.13 GetInputRange()

This function is used to read the current setting of the ADC range control.

Syntax:
int_t GetInputRange(PixieBcRange_t* pRange);
int_t GetInputRange(int_t* pRange);

Arguments:

pRange	Is a pointer to return the value in. ***PXBC_RNG_5V*** or ***PXBC_RNG_10V***.

Returns:

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

6.2.14 GetOutputRange()

This function is used to read the current setting of the DAC range control.

Syntax:

int_t GetOutputRange(PixieBcRange_t* pRange);
int_t GetOutputRange(int_t* pRange);

Arguments:

pRange	Is a pointer to return the value in. ***PXBC_RNG_5V*** or ***PXBC_RNG_10V***.

Returns:

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

6.2.15 GetTimedAdc()

This function is used to read the current setting of the ADC timed conversion generator and master trigger GPIO pin source.

Syntax:

int_t GetTimedAdc(uint32_t* pTimeBetweenUs, uint8_t* pGpioPin = NULL);

Arguments:

pTimeBetweenUs	Is a pointer to return the time between sample in.
pGpioPin	Is a pointer to return the current master trigger pin in. GPIO pins 5, 6, 12, 13, 19, 20, 21, 25 can be used

Returns:

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

6.2.16 GetValue()

This function is used to read a single input value either raw or scaled values using calibrations from the internal store.
Precede this function with a call to ***AdcRead()*** to ensure the latest values are available.

Syntax:

int_t GetValue(uint8_t input, int32_t* pValue);
int_t GetValue(uint8_t input, double_t* pValues);

Arguments:

input	The input number to get the value for.
pValues	Is a pointer to a 32bit signed or double value to return the read data in.

Returns:

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

6.2.17 GetValues()

This function is used to read all 8 raw converted inputs of the ADC or scaled values using calibrations.
Precede this function with a call to ***SetSamplingMode()*** to trigger a conversion.

Syntax:
int_t GetValues(int32_t* pValues);
int_t GetValues(double_t* pValues);

Arguments:

pValues	Is a pointer to an array of 8 x 32bit signed values to return the read data in, or an array of 8 x double floating point engineered values

Returns:

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

6.2.18 GetValuesPtr()

This function is used to return a pointer to the raw and engineered values.

Syntax:
PixieAdc8Dac2Readings_t* GetValuesPtr(void);

Arguments:

Returns:
Pointer to the values structure.

6.2.19 IsAdcReady()

This function is used to get the status of the ADC ready.
The 1st function used an I2C polled method to read the BUSY signal, the 2nd function can poll one of the GPIO pins which has been setup for interrupting.

Syntax:
int_t IsAdcReady(uint8_t* pReady);
int_t IsAdcReady(uint8_t* pReady, PixieGpioPin_t gpioPin);

Arguments:

pReady	Is a pointer to return the status in.
gpioPin	The GPIO pin to poll.

Returns:

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

6.2.20 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.

6.2.21 SetAdcConvstSource()

This function is used to set the current value of the ADC external trigger GPIO pin source.

Syntax:

int_t SetAdcConvstSource(uint8_t srcValue);

Arguments:

srcValue	The value of the external trigger source pin. ***PXBC_ADC8_DAC2_CONVST_SW*** = Software trigger. ***PXBC_ADC8_DAC2_CONVST_TIME*** = Timed trigger. External trigger GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 25, 26 can be used.

Returns:

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

6.2.22 SetAdcOsBits()

This function is used to set the current value of the ADC oversampling control bits.

Syntax:

int_t SetAdcOsBits(uint8_t osValue);

Arguments:

osValue	The value of the oversampling bits. 0 = No oversampling. 1 = x2 oversampling. 2 = x4 oversampling. 3 = x8 oversampling. 4 = x16 oversampling. 5 = x32 oversampling. 6 = x64 oversampling.

Returns:

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

6.2.23 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.
zeroValue	The zero calibration value.
spanValue	The span calibration value.

Returns:

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

6.2.24 SetCeDecode()

This function is used to set the board chip enable decoding.

Syntax:

```
int_t SetCeDecode(
        uint8_t usedAdcMask,
        uint8_t polAdcMask,
        uint8_t usedDacMask,
        uint8_t polDacMask,
        uint8_t* pAdrIds);
```

Arguments:

usedAdcMask	This is the used mask for the ADC and is made up of the OR of the following masks: ***PXBC_CEx_A0_M, PXBC_CEx_A1_M, PXBC_CEx_A2_M, PXBC_CEx_A3_M*** use of one or more to show which sub address lines to include. ***PXBC_CEx_CE0_M, PXBC_CEx_CE1_M, PXBC_CEx_CE2_M*** use only one of these to show which SPI CE to use. ***PXBC_CEx_SPI_0_M, PXBC_CEx_SPI_1_M*** use only one of these to show which SPI bus to use. NOTE: the board can only use one or the other for all devices.
polAdcMask	This is the polarity mask for the ADC and is made up of the OR of the following masks: ***PXBC_CEx_A0_M, PXBC_CEx_A1_M, PXBC_CEx_A2_M, PXBC_CEx_A3_M*** use of one or more to show which sub address lines to decode as active high
usedDacMask	Same as ***usedAdcMask*** shown above for this is for the DAC.
polDacMask	Same as ***polAdcMask*** shown above for this is for the DAC.
pAdrIds	Is a pointer to an array of 4 address identifiers used for the sub address decode. If using SPI bus 0 = ***PXBC_PI_GPIO22, PXBC_PI_GPIO23, PXBC_PI_GPIO24, PXBC_PI_GPIO27*** If using SPI bus 1 = ***PXBC_PI_GPIO5, PXBC_PI_GPIO6, PXBC_PI_GPIO12, PXBC_PI_GPIO13***

Returns:

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

6.2.25 SetInputRange()

This function is used to set the current value of the ADC range control.

Syntax:

int_t SetInputRange(PixieBcRange_t range);

Arguments:

range	The value of the range to set. ***PXBC_RNG_5V*** or ***PXBC_RNG_10V***.

Returns:

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

6.2.26 SetIrq()

This function is used to set the ADC device interrupt mapping.

Syntax:

int_t SetIrq(
uint16_t usedMask,
bool_t activeLowFlg,
bool_t openDrainFlg);

Arguments:

usedMask	This is the PI pin to use, select only one of the following masks: ***PXBC_IRQ_EN_GPIO5*** ***PXBC_IRQ_EN_GPIO6*** ***PXBC_IRQ_EN_GPIO12*** ***PXBC_IRQ_EN_GPIO13*** ***PXBC_IRQ_EN_GPIO19*** ***PXBC_IRQ_EN_GPIO20*** ***PXBC_IRQ_EN_GPIO21*** ***PXBC_IRQ_EN_GPIO25*** ***PXBC_IRQ_EN_GPIO18*** ***PXBC_IRQ_EN_GPIO17*** ***PXBC_IRQ_EN_GPIO16*** ***PXBC_IRQ_EN_GPIO26*** ***PXBC_IRQ_EN_GPIO22*** ***PXBC_IRQ_EN_GPIO23*** ***PXBC_IRQ_EN_GPIO24*** ***PXBC_IRQ_EN_GPIO27***
activeLowFlg	TRUE = IRQ to Pi is active low.
openDrainFlg	TRUE = IRQ to Pi is open drain.

Returns:

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

6.2.27 SetOutputRange()

This function is used to set the current value of the DAC range control.

Syntax:

int_t SetOutputRange(PixieBcRange_t range);

Arguments:

range	The value of the range to set. ***PXBC_RNG_5V*** or ***PXBC_RNG_10V***.

Returns:

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

6.2.28 SetSamplingMode ()

This function is used to start the ADC conversion, also start and stop timed mode.

Syntax:

int_t SetSamplingMode(PixieAdcStartAction_t action, PixieGpioPin_t gpioPin = PIXIE_GPIO_PIN_NONE);

Arguments:

action	Type of action. ***PXAD_ADC_START*** This will trigger a conversion if ***SetAdcConvstSource()*** is set to ***PXBC_ADC8_DAC2_CONVST_SW*** or ***PXBC_ADC8_DAC2_CONVST_TIMED*** ***PXAD_ADC_STOP*** This will stop any timed trigger conversions if ***SetAdcConvstSource()*** is set to ***PXBC_ADC8_DAC2_CONVST_TIMED*** ***PXAD_ADC_START_GPIO*** This will trigger a conversion using the parameters set using ***SetAdcTimed()***
gpioPin	The GPIO pin to use as the conversion trigger. GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 25, 26 can be used Optional used for ***PXAD_ADC_START_GPIO***

Returns:

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

6.2.29 SetTimedAdc()

This function is used to set the current values of the ADC timed conversion generator and master trigger GPIO pin source.

Syntax:

int_t SetTimedAdc(uint32_t timeBetweenUs, uint8_t gpioPin = PIXIE_GPIO_PIN_NONE);

Arguments:

timeBetweenUs	Time between sample conversions in uS.
gpioPin	The GPIO pin to set as the master conversion trigger output. GPIO pins 5, 6, 12, 13, 19, 20, 21, 25 can be used

Returns:

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

6.2.30 SetValue()

This function is used to set the value of the DAC output A or B.
The floating point function will scale to a raw value using the calibrations before setting the DAC.

Syntax:

int_t SetValue(uint8_t outputId, int32_t value);
int_t SetValue(uint8_t outputId, double_t value);

Arguments:

outputId	Output to set, 0 = DAC A (AO1), 1 = DAC B (AO2).
value	The value to set.

Returns:

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

6.2.31 SetValues()

This function is used to set the value of the DAC output A (AO1) and B (AO2) at the same time.
The floating point function will scale to a raw value using the calibrations before setting the DAC’s.

Syntax:

int_t SetValues(int32_t* pValues);
int_t SetValues(double_t* pValues);

Arguments:

pValues	Pointer to the values to set.

Returns:

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

6.3 ADC8-DAC2 Analogue I/O board ‘Python’ library support functions

This group contains ‘Python’ functions for supporting the ADC8-DAC2 analogue I/O board.

The class is ***PyPixieBoardAdc8Dac2***
They all require the ***import PixiePy*** module.

6.3.1 PyPixieBoardAdc8Dac2()

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

Syntax:
object = PixiePy.PyPixieBoardAdc8Dac2(i2cChannel, i2cAddress, spiBus)

Arguments:

i2cChannel	I2C channel to use for i2C access, default = 1, i.e. i2c-1.
i2cAddress	I2C address for the board to access.
spiBus	The SPI bus to use, 0, 1, …

6.3.2 Initialise()

This function is used to open a path to an SPI device given its channel, clock speed for the ADC and DAC to communicate over.

Syntax:
result = object.Initialise(channel0,channel1, speedHz)

Arguments:

channel0	The SPI channel to use for the ADC, **0, 1, …**
channel1	The SPI channel to use for the DAC, **0, 1, …**
speedHz	The SPI clocking speed Hz, e.g. 100000 = 100kHz. (Optional, default = 10000000)

Returns:

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

6.3.3 Close()

This function is used to close any open paths to the ADC and DAC devices.

Syntax:
result = object.Close()

Arguments:

Returns:

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

6.3.4 AdcRead()

Read the ADC values.
Precede this function with a call to ***SetSamplingMode ()*** to trigger a conversion then wait for the nominated GPIO data ready or interrupt to be triggered.

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

Arguments:

Returns:

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

6.3.5 DefaultCalibration()

This function is used to set default calibration values for a full scale of 5.0V for both the ADC inputs and DAC outputs when used with the raw to engineered conversion functions.

Syntax:
Result = object.DefaultCalibration(calibNumber)

Arguments:

calibNumber	Number of the calibration to default.

Returns:

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

6.3.6 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.

6.3.7 GetAdcConvstSource()

This function is used to read the current setting of the ADC external trigger GPIO pin source.

Syntax:

result, srcValue = object.GetAdcConvstSource()

Arguments:

Returns:

result	***0*** or ***E...*** Linux error code.
srcValue	The current source value. ***0*** = Software trigger. ***1*** = Timed trigger. External trigger GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 25, 26.

6.3.8 GetAdcOsBits()

This function is used to read the current setting of the ADC oversampling control bits.

Syntax:

result, osValue = object.GetAdcOsBits()

Arguments:

Returns:

result	***0*** or ***E...*** Linux error code.
osValue	The current oversampling value. 0 = No oversampling. 1 = x2 oversampling. 2 = x4 oversampling. 3 = x8 oversampling. 4 = x16 oversampling. 5 = x32 oversampling. 6 = x64 oversampling.

6.3.9 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.

6.3.10 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.

6.3.11 GetInputRange()

This function is used to read the current setting of the ADC range control.

Syntax:
result, range = object.GetInputRange()

Arguments:

Returns:

result	***0*** or ***E...*** Linux error code.
range	***0*** = 5V, ***1*** = 10V range.

6.3.12 GetOutputRange()

This function is used to read the current setting of the DAC range control.

Syntax:

result, range = object.GetOutputRange()

Arguments:

Returns:

result	***0*** or ***E...*** Linux error code.
range	***0*** = 5V, ***1*** = 10V range.

6.3.13 GetTimedAdc()

This function is used to read the current setting of the ADC timed conversion generator and master trigger GPIO pin source.

Syntax:

result, timeBetweenUs, gpioPin = object.GetTimedAdc()

Arguments:

Returns:

result	***0*** or ***E...*** Linux error code.
timeBetweenUs	Sample time in uS.
gpioPin	Master trigger pin. GPIO pins 5, 6, 12, 13, 19, 20, 21, 25 can be used

6.3.14 GetValue()

This function is used to read a single input value either raw or scaled values using calibrations from the internal store.
Precede this function with a call to ***AdcRead()*** to ensure the latest values are available.

Syntax:

result, reading, readingEng = object.GetValues(input)

Arguments:

input	The input number to get the value for.

Returns:

result	***0*** or ***E...*** Linux error code.
reading	Raw values.
readingEng	Engineered value.

6.3.15 GetValues()

This function is used to read all 8 raw and converted engineered inputs.
Precede this function with a call to ***AdcRead()*** to ensure the latest values are available.

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

Arguments:

Returns:

result	***0*** or ***E...*** Linux error code.
readings	8 converted raw ADC values, inputs 0 through 7 in that order.
readingsEng	8 converted and calibration scaled floating point values, inputs 0 through 7 in that order.

6.3.16 IsAdcReady()

This function is used to get the ADC ready status.

Syntax:
result, ready = object.IsAdcReady()

Arguments:

Returns:

result	***0*** or ***E...*** Linux error code.
ready	TRUE = ready, FALSE = busy.
readingEng	Engineered value.

6.3.17 IsAdcReady()

This function is used to get the ADC ready status by polling an interrupt setup ADC's ready GPIO signal.

Syntax:
result, ready = object.IsAdcReady(gpio)

Arguments:

gpio	The GPIO pin to poll.

Returns:

result	***0*** or ***E...*** Linux error code.
ready	TRUE = ready, FALSE = busy.

6.3.18 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.

6.3.19 SetAdcConvstSource()

This function is used to set the current value of the ADC external trigger GPIO pin source.

Syntax:
result = object.SetAdcConvstSource(srcValue)

Arguments:

srcValue	The value of the external trigger source pin. 0 = Software trigger. 1 = Timed trigger. External trigger GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 25, 26 can be used.

Returns:

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

6.3.20 SetAdcOsBits()

This function is used to set the current value of the ADC oversampling control bits.

Syntax:
result = object.SetAdcOsBits(osValue)

Arguments:

osValue	The value of the oversampling bits. 0 = No oversampling. 1 = x2 oversampling. 2 = x4 oversampling. 3 = x8 oversampling. 4 = x16 oversampling. 5 = x32 oversampling. 6 = x64 oversampling.

Returns:

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

6.3.21 SetCalibration()

This function is used to set a calibration.

Syntax:

result = object.SetCalibration(calibNumber, zeroValue, spanValue)

Arguments:

calibNumber	Number of the calibration to set.
zeroValue	Zero calibration value.
spanValue	Span calibration value.

Returns:

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

6.3.22 SetCeDecode()

This function is used to set the board chip enable decoding.

Syntax:

Result = object.SetCeDecode(usedAdcMask, polAdcMask, usedDacMask, polDacMask, adrIds)

Arguments:

usedAdcMask See "PixieBoardCommon.h" for the value of the masks to use.
This is the used mask and is made up of the OR of the following masks:
PXBC_CEx_A0_M, PXBC_CEx_A1_M, PXBC_CEx_A2_M, PXBC_CEx_A3_M
use of one or more to show which sub address lines to include.

PXBC_CEx_CE0_M, PXBC_CEx_CE1_M, PXBC_CEx_CE2_M
use only one of these to show which SPI CE to use.

PXBC_CEx_SPI_0_M, PXBC_CEx_SPI_1_M
use only one of these to show which SPI bus to use.
NOTE: the board can only use one or the other for all devices.

polAdcMask See "PixieBoardCommon.h" for the value of the masks to use.
This is the polarity mask and is made up of the OR of the following masks:
PXBC_CEx_A0_M, PXBC_CEx_A1_M, PXBC_CEx_A2_M, PXBC_CEx_A3_M
use of one or more to show which sub address lines to decode as active high

usedDacMask Same as ***usedAdcMask*** shown above for this is for the DAC.

polDacMask Same as ***polAdcMask*** shown above for this is for the DAC.

adrIds See "PixieBoardCommon.h" for the value of the masks to use.
Is an array of 4 address identifiers used for the sub address decode.
If using SPI bus 0
= ***PXBC_PI_GPIO22, PXBC_PI_GPIO23, PXBC_PI_GPIO24, PXBC_PI_GPIO27***
If using SPI bus 1
= ***PXBC_PI_GPIO5, PXBC_PI_GPIO6, PXBC_PI_GPIO12, PXBC_PI_GPIO13***

Returns:

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

6.3.23 SetInputRange()

This function is used to set the current value of the ADC range control.

Syntax:

result = object.SetInputRange(range)

Arguments:

range The value of the range to set.
0 = 5V, ***1*** = 10V.

Returns:

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

6.3.24 SetIrq()

This function is used to set the ADC device interrupt mapping.

Syntax:

Result = object.SetIrq(usedMask, activeLowFlg, openDrainFlg)

Arguments:

usedMask	See "PixieBoardCommon.h" for the value of the masks to use. This is the PI pin to use, select only one of the following masks: ***PXBC_IRQ_EN_GPIO5*** ***PXBC_IRQ_EN_GPIO6*** ***PXBC_IRQ_EN_GPIO12*** ***PXBC_IRQ_EN_GPIO13*** ***PXBC_IRQ_EN_GPIO19*** ***PXBC_IRQ_EN_GPIO20*** ***PXBC_IRQ_EN_GPIO21*** ***PXBC_IRQ_EN_GPIO25*** ***PXBC_IRQ_EN_GPIO18*** ***PXBC_IRQ_EN_GPIO17*** ***PXBC_IRQ_EN_GPIO16*** ***PXBC_IRQ_EN_GPIO26*** ***PXBC_IRQ_EN_GPIO22*** ***PXBC_IRQ_EN_GPIO23*** ***PXBC_IRQ_EN_GPIO24*** ***PXBC_IRQ_EN_GPIO27***
activeLowFlg	1 = IRQ to Pi is active low.
openDrainFlg	1 = IRQ to Pi is open drain.

Returns:

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

6.3.25 SetOutputRange()

This function is used to set the current value of the DAC range control.

Syntax:

result = object.SetOutputRange(range)

Arguments:

range	The value of the range to set. ***0*** = 5V, ***1*** = 10V.

Returns:

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

6.3.26 SetSamplingMode()

This function is used to start the ADC conversion, also start, and stop timed mode.

Syntax:

result = object.SetsamplingMode(action, gpioPin)

Arguments:

action	Type of action. ***0*** This will trigger a conversion if ***SetAdcConvstSource()*** is set to ***0*** or ***1*** ***1*** This will trigger a conversion using the toggle of the ***gpioPin***. 2 This will stop any timed trigger conversions if ***SetAdcConvstSource()*** is set to ***1***
gpioPin	The GPIO pin to use as the conversion trigger. GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 25, 26 can be used Optional used for action = ***1*** (Optional, default = 0)

Returns:

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

6.3.27 SetTimedAdc()

This function is used to set the current values of the ADC timed conversion generator and master trigger GPIO pin source.

Syntax:

result = object.SetTimedAdc(timeBetweenUs, gpioPin)

Arguments:

timeBetweenUs	Time between sample conversions in uS.
gpioPin	The GPIO pin to set as the master conversion trigger output. GPIO pins 5, 6, 12, 13, 19, 20, 21, 25 can be used. (Optional, default = 0)

Returns:

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

6.3.28 SetValue()

This function is used to set the value of the DAC output A (AO1), or output B (AO2).
If ***value*** is a floating point then the function will scale to a raw value using the calibrations before setting the DAC.

Syntax:

Result = object.SetValue(value, output)

Arguments:

value	The value to set.
output	Output to set, 0 = DAC A (AO1), 1 = DAC B (AO2).

Returns:

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

6.3.29 SetValues()

This function is used to set the value of the DAC output A (AO1) and output B (AO2) at the same time.
If ***values*** are floating point values, then the function will scale to a raw value using the calibrations before setting the DAC.

Syntax:

Result = object.SetValues(values)

Arguments:

value	The values to set.

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