PIXIE BOARD
IO16-24V DIGITAL 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

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

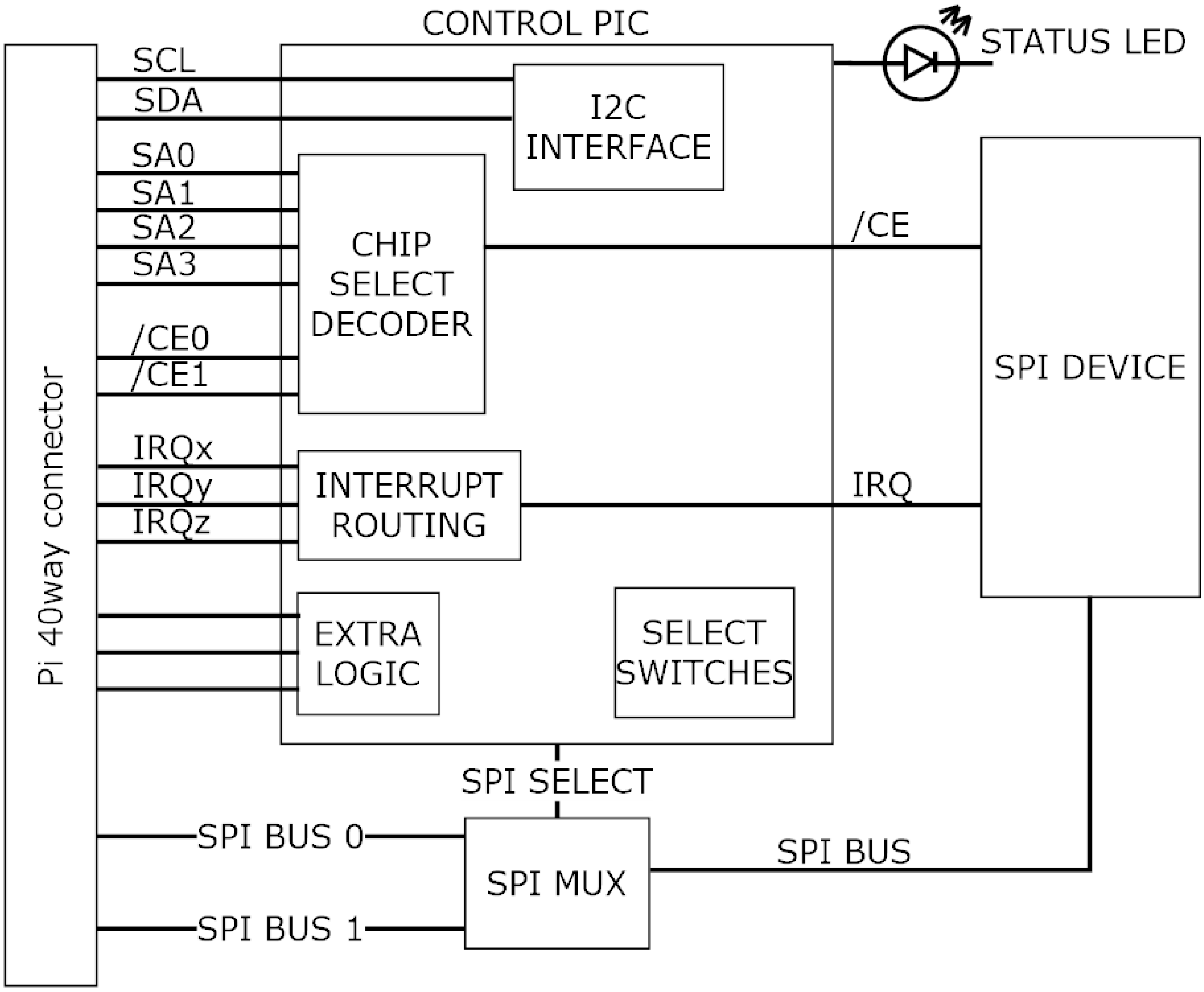
This shows the control overview of the **PIXIE** board.

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

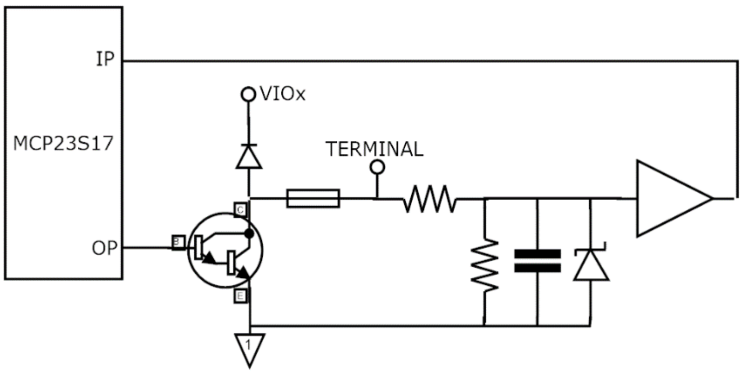
The **IO16-24V** is a **PIXIE** board that can input and output digital signals with a working range of 5-24 DC, this is ideal for a range of interface uses. There are 16 dual purpose I/O connections used for both input and output, the outputs are open collector Darlington driven capable of driving inductive loads directly such as relay coils etc. they all have fly back diodes. The outputs are also PTC fuse protected so that if a fault condition is presented to the output it will safely be disconnected until the power is cycled to reset it.

4.1 Specification.

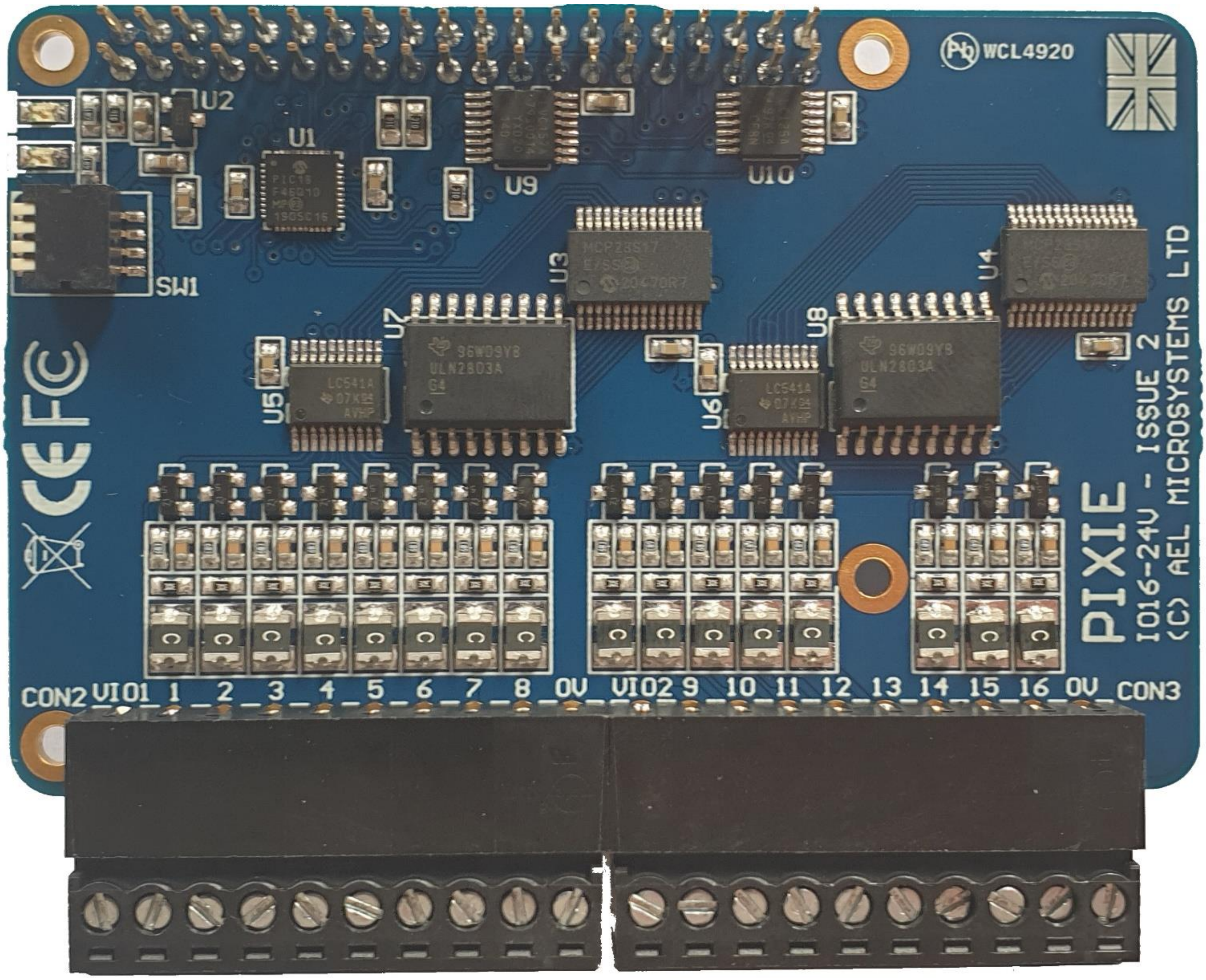
Number of I/O	16
Voltage range	+5 to +24V DC
Input impedance	> 50Kohms
Output drive	200mA, Resettable fuse and flywheel diode protected.
Supply voltage	+5V
Power consumption	50mA
I2C speed	<=100kHz
SPI speed	<=10MHz
Temperature	0-70C

4.2 I/O pin overview.

This shows the input/output 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 digital I/O's 1 thru 8.
CON3 is a 10-way connector that covers digital I/O's 9 thru 16.

Signals:
VIO1 & **VIO2** of the connector are connected to the supply of the system usually +24V and is only required if the fly-back diodes are required.

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

The remaining signals are for the 16 I/O's

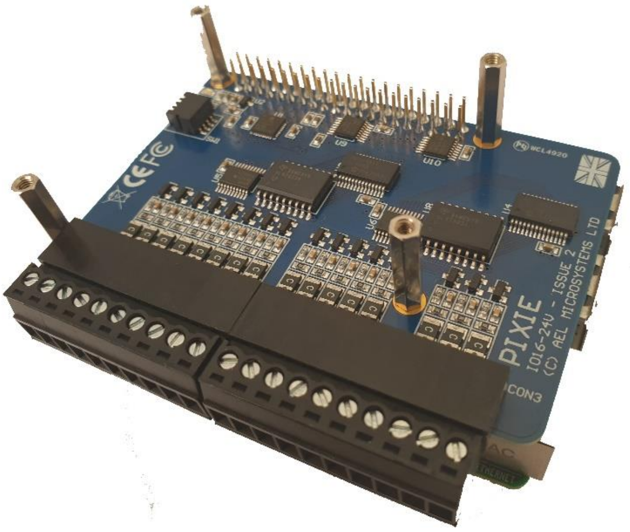
5.1 Insulate USB connector housing on Raspberry Pi 4

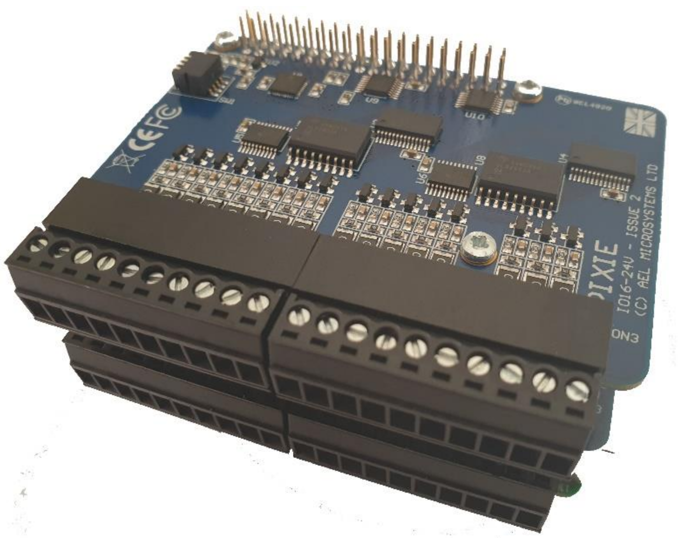
WARNING

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 the other onto the USB connector as shown below before assembling the board onto the Raspberry Pi.

5.3 Mounting the boards.

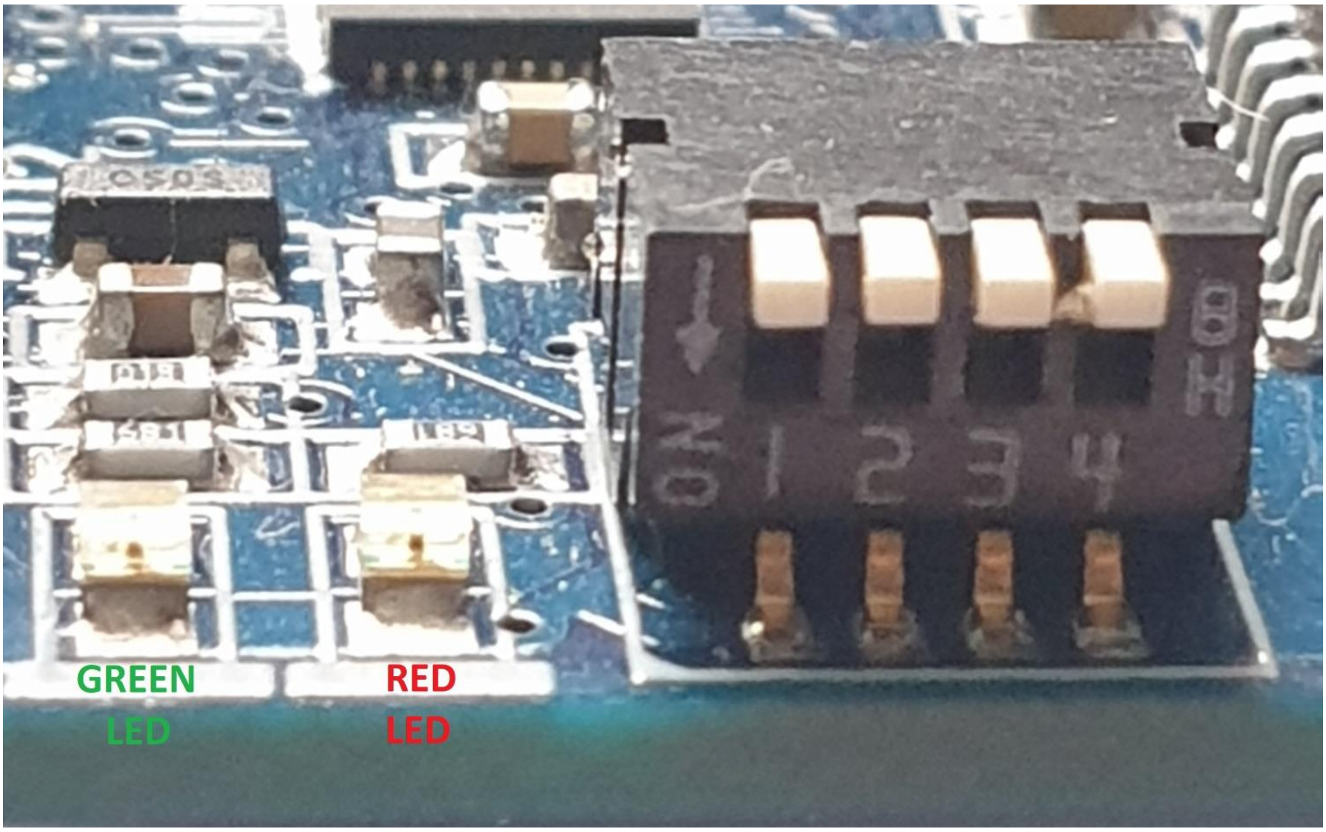
Using M2.5mm - 4mmAF – 17mm long hex standoff's mount the PIXIE boards onto the Raspberry Pi as shown. On some Pixie board's the connector CON3 obscures one of the mounting holes to the Raspberry Pi board when mounted directly to it ,so only 3 pillars are used which is sufficient to securely mount the 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.





5.5 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”

5.6 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 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.

5.7 Principle of operation.

The board uses two MCP23S17 I/O devices, one is used for the lower 8 I/O and the other for the upper 8 I/O. A set of support functions is available to access all the registers of the devices and treats the two devices as one single 16 bit one. The board only needs one chip enable to function and can provide separate interrupts for the lower (IRQ0) and upper (IRQ1) 8 I/O or combine both into one single interrupt for the board.
This device uses the board references /CE0, /IRQ0 and /IRQ1.

5.8 Configuration.

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

5.8.1 Board specific configuration.

There are no additional board specific configuration requirements for this board.

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[8] - UTILITY MENU :
(?)-Menu help, (B)-Board ID, (C)-Configure SPI, (E)-Enable IRQ's, (F)-On Change IRQ's,
(G)-Get inputs, (I)-Invert inputs, (O)-Get outputs, (P)-Pattern test, (R)-Get Irq requests,
(S)-Set outputs, (T)-Interrupt test, (Z)-Clear outputs, (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.
(E)-Enable IRQ's	Enable/Disable the digital pin interrupts.
(F)-On Change IRQ's	Change the on-change digital pin interrupts.
(G)-Get inputs	Display the current inputs.
(I)-Invert inputs	Change the polarity of the inputs.
(O)-Get outputs	Display the current outputs.
(P)-Pattern test	Perform outputs toggle pattern test.
(R)-Get Irq requests	Display the input interrupt requests.
(S)-Set outputs	Set/reset a digital output pin.
(T)-Interrupt test	Test an interrupt triggered from an input pin.
(Z)-Clear outputs	Zero the digital outputs.

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 DIO16-24V Digital I/O board ‘C’ library support functions

This group contains ‘C’ functions for supporting the DIO16-24V digital 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.

Most functions are masks 16bits wide where,
Bit0 = I/O 1, through to Bit15 = I/O 16.
This board used a Microchip Mcp23s17 device and this should be consulted when using these functions.

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

All functions make use of a board control structure ***PixieDio16_24vCtrl_t*** which is declared one for each board used and contains all the control parameters used for the board, it is constructed using the ***PixieDio16_24vConstruct()*** function and can be optionally destroyed using the ***PixieDio16_24vDestroy()*** function.

6.1.1 PixieDio16_v24Construct()

This function is used to initialise the ***PixieDio16_24vCtrl_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 PixieDio16_v24Construct(PixieDio16_24vCtrl_t* pCtrl, uint16_t I2cAddress);

Arguments:

pCtrl	Is a pointer to the ***PixieDio16_24vCtrl_t*** structure to use.
I2cAddress	I2C address for the board to access.

Returns:

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

6.1.2 PixieDio16_v24Destroy()

This function is used to destroy the ***PixieDio16_24vCtrl_t*** structure when the board is no longer required.

Syntax:

int_t PixieDio16_v24Destroy(PixieDio16_24vCtrl_t * pCtrl);

Arguments:

pCtrl	Is a pointer to the ***PixieDio16_24vCtrl_t*** structure to use.

Returns:

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

6.1.3 PixieDio16_24vGetInputPolarity()

This function is used to read the input polarity register.

Syntax:

int_t PixieDio16_24vGetInputPolarity(PixieDio16_24vCtrl_t* pCtrl, uint16_t* pPolarity);

Arguments:

pCtrl	Is a pointer to the ***PixieDio16_24vCtrl_t*** structure to use. Ensure this structure is constructed and the channel is open before calling this function.
pPolarity	Is a pointer to return the current polarity mask in. 0 = no invert, 1 = inverted.

Returns:

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

6.1.4 PixieDio16_24vGetInputs()

This function is used to read the current state of the inputs.

Syntax:

int_t PixieDio16_24vGetInputs(PixieDio16_24vCtrl_t* pCtrl, uint16_t* pInputs);

Arguments:

pCtrl	Is a pointer to the ***PixieDio16_24vCtrl_t*** structure to use. Ensure this structure is constructed and the channel is open before calling this function.
pInputs	Is a pointer to return the current inputs mask in. 0 = Low, 1 = High, unless inverted by the input polarity

Returns:

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

6.1.5 PixieDio16_24vGetInterruptEnable()

This function is used to read the current state of the input interrupt enables.

Syntax:
int_t PixieDio16_24vGetInterruptEnable(PixieDio16_24vCtrl_t* pCtrl, uint16_t* pEnables);

Arguments:

pCtrl	Is a pointer to the ***PixieDio16_24vCtrl_t*** structure to use. Ensure this structure is constructed and the channel is open before calling this function.
pEnables	Is a pointer to return the current interrupt enables mask in. 0 = Disabled, 1 = Enabled.

Returns:

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

6.1.6 PixieDio16_24vGetInterruptFlags()

This function is used to read the current state of the input interrupt pending flags.

Syntax:
int_t PixieDio16_24vGetInterruptFlags(PixieDio16_24vCtrl_t* pCtrl, uint16_t* pFlags);

Arguments:

pCtrl	Is a pointer to the ***PixieDio16_24vCtrl_t*** structure to use. Ensure this structure is constructed and the channel is open before calling this function.
pFlags	Is a pointer to return the current interrupt pending flags mask in. 0 = No interrupt, 1 = Interrupt pending.

Returns:

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

6.1.7 PixieDio16_24vGetIntOnChange()

This function is used to read the current state of the input interrupt on change control values.

Syntax:

int_t PixieDio16_24vGetIntOnChange(
PixieDio16_24vCtrl_t * pCtrl,
uint16_t* pControl,
uint16_t* pDefValue);

Arguments:

pCtrl	Is a pointer to the ***PixieDio16_24vCtrl_t*** structure to use. Ensure this structure is constructed and the channel is open before calling this function.
pControl	Is a pointer to return the current INTCON register mask in.
pDefValue	Is a pointer to return the current DEFVAL register mask in.

Returns:

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

6.1.8 PixieDio16_24vGetOutputs()

This function is used to read the current state of the outputs.

Syntax:

int_t PixieDio16_24vGetOutputs(PixieDio16_24vCtrl_t* pCtrl, uint16_t* pOutputs);

Arguments:

pCtrl	Is a pointer to the ***PixieDio16_24vCtrl_t*** structure to use. Ensure this structure is constructed and the channel is open before calling this function.
pOutputs	Is a pointer to return the current outputs mask in.

Returns:

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

6.1.9 PixieDio16_v24Initialise()

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

Syntax:

int_t PixieDio16_v24Initialise(PixieDio16_24vCtrl_t* pCtrl ,   PixieSpiCtrl_t* pSpi);

Arguments:

pCtrl	Is a pointer to the ***PixieDio16_24vCtrl_t*** structure to use.
pSpiAdc	Is a pointer to the opened SPI channel to be used by the access functions. Ensure the channel is initialised and the channel is open.

Returns:

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

6.1.10 PixieDio16_24vSetCeDecode()

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

Syntax:

int_t PixieDio16_24vSetCeDecode(
PixieDio16_24vCtrl_t* pCtrl,
uint8_t usedMask,
uint8_t polMask,
uint8_t* pAdrIds);

Arguments:

pCtrl Is a pointer to the ***PixieDio16_24vCtrl_t*** structure to use.
Ensure this structure is constructed before calling this function.

usedMask 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.

polMask 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

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.

6.1.11 PixieDio16_24vSetInputPolarity()

This function is used to set the value of the input polarity register.

Syntax:

int_t PixieDio16_24vSetInputPolarity(PixieDio16_24vCtrl_t* pCtrl,   uint16_t polarity);

Arguments:

pCtrl	Is a pointer to the ***PixieDio16_24vCtrl_t*** structure to use. Ensure this structure is constructed and the channel is open before calling this function.
polarity	The polarity mask to set. 0 = no invert, 1 = inverted.

Returns:

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

6.1.12 PixieDio16_24vSetInterruptEnable()

This function is used to set the current input interrupt enables register.

Syntax:

int_t PixieDio16_24vSetInterruptEnable(PixieDio16_24vCtrl_t* pCtrl, uint16_t enables);

Arguments:

pCtrl	Is a pointer to the ***PixieDio16_24vCtrl_t*** structure to use. Ensure this structure is constructed and the channel is open before calling this function.
enables	The enables to set. 0 = Disabled, 1 = Enabled.

Returns:

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

6.1.13 PixieDio16_24vSetIntOnChange()

This function is used to set the input interrupt on change control values.

Syntax:

int_t PixieDio16_24vSetIntOnChange(PixieDio16_24vCtrl_t* pCtrl, uint16_t control, uint16_t defValue);

Arguments:

pCtrl	Is a pointer to the ***PixieDio16_24vCtrl_t*** structure to use. Ensure this structure is constructed and the channel is open before calling this function.
control	INTCON register value.
defValue	DEFVAL register value.

Returns:

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

6.1.14 PixieDio16_24vSetIrq()

This function is used to set the interrupt mapping for the low and high I/O devices.

Syntax:

```
int_t PixieDio16_24vSetIrq(
        PixieDio16_24vCtrl_t* pCtrl,
        uint16_t used0Mask,
        bool_t activeLow0Flg,
        uint16_t used1Mask,
        bool_t activeLow1Flg);
```

Arguments:

pCtrl Is a pointer to the ***PixieDio16_24vCtrl_t*** structure to use.
Ensure this structure is constructed before calling this function.

used0Mask 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

activeLow0Flg TRUE = IRQ to Pi is active low for low 8 I/O device interrupt.

used1Mask This is the PI pin to use, select only one of the masks shown above.

activeLow1Flg TRUE = IRQ to Pi is active low for the high 8 I/O device interrupt.

Returns:

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

6.1.15 PixieDio16_24vSetOutputs()

This function is used to set the outputs.

Syntax:

int_t PixieDio16_24vSetOutputs(PixieDio16_24vCtrl_t* pCtrl, uint16_t outputs);

Arguments:

pCtrl	Is a pointer to the ***PixieDio16_24vCtrl_t*** structure to use. Ensure this structure is constructed and the channel is open before calling this function.
outputs	The outputs to set. Note: as open collector outputs are used, a logic ‘1’ = On which pulls the output low.

Returns:

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

6.2 DIO16-24V Digital I/O board 'C++' library support functions

This group contains 'C++' functions for supporting the DIO16-24V digital 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.

Most functions are masks 16bits wide where,
Bit0 = I/O 1, through to Bit15 = I/O 16.
This board used a Microchip Mcp23s17 device and this should be consulted when using these functions.

The class is ***PixieBoardDio16_24V***
They all require the ***#include <PixieLib.hpp>*** header file.

6.2.1 PixieBoardDio16_24V()

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

Syntax:
PixieBoardDio16_24V(uint_t i2cAddress, uint8_t busId);

Arguments:

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

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

Arguments:

channel	The SPI channel to use, ***PIXIE_SPI_CHAN_0, PIXIE_SPI_CHAN_1***…
speedHz	The SPI clocking speed Hz, e.g. 100000 = 100kHz

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 GetInputPolarity()

This function is used to read the input polarity register.

Syntax:
int_t GetInputPolarity(uint16_t pPolarity);*

Arguments:

pPolarity	Is a pointer to return the current polarity mask in. 0 = no invert, 1 = inverted.

Returns:

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

6.2.5 GetInputs()

This function is used to read the current state of the inputs.

Syntax:
int_t GetInputs(uint16_t pInputs);*

Arguments:

pInputs	Is a pointer to return the current inputs mask in. 0 = Low, 1 = High, unless inverted by the input polarity

Returns:

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

6.2.6 GetInterruptEnable()

This function is used to read the current state of the input interrupt enables.

Syntax:

int_t GetInterruptEnable(uint16_t* pEnables);

Arguments:

pEnables	Is a pointer to return the current interrupt enables mask in. 0 = Disabled, 1 = Enabled.

Returns:

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

6.2.7 GetInterruptFlags()

This function is used to read the current state of the input interrupt pending flags.

Syntax:

int_t GetInterruptFlags(uint16_t* pFlags);

Arguments:

pFlags	Is a pointer to return the current interrupt pending flags mask in. 0 = No interrupt, 1 = Interrupt pending.

Returns:

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

6.2.8 GetIntOnChange()

This function is used to read the current state of the input interrupt on change control values.

Syntax:

int_t GetIntOnChange(uint16_t* pControl, uint16_t* pDefValue);

Arguments:

pControl	Is a pointer to return the current INTCON register mask in.
pDefValue	Is a pointer to return the current DEFVAL register mask in.

Returns:

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

6.2.9 GetOutputs()

This function is used to read the current state of the outputs.

Syntax:

int_t GetOutputs(uint16_t* pOutputs);

Arguments:

pOutputs	Is a pointer to return the current outputs mask in.

Returns:

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

6.2.10 SetCeDecode()

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

Syntax:

```
int_t SetCeDecode(
        uint8_t usedMask,
        uint8_t polMask,
        uint8_t* pAdrIds);
```

Arguments:

usedMask

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.

polMask

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

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.

6.2.11 SetInputPolarity()

This function is used to set the value of the input polarity register.

Syntax:

int_t SetInputPolarity(uint16_t polarity);

Arguments:

polarity	The polarity mask to set. 0 = no invert, 1 = inverted.

Returns:

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

6.2.12 SetInterruptEnable()

This function is used to set the current input interrupt enables register.

Syntax:

int_t SetInterruptEnable(uint16_t enables);

Arguments:

enables	The enables to set. 0 = Disabled, 1 = Enabled.

Returns:

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

6.2.13 SetIntOnChange()

This function is used to set the input interrupt on change control values.

Syntax:

int_t SetIntOnChange(uint16_t control, uint16_t defValue);

Arguments:

control	INTCON register value.
defValue	DEFVAL register value.

Returns:

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

6.2.14 SetIrq()

This function is used to set the interrupt mapping for the low and high I/O devices.

Syntax:

int_t SetIrq(
uint16_t used0Mask,
bool_t activeLow0Flg,
uint16_t used1Mask,
bool_t activeLow1Flg);

Arguments:

used0Mask 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

activeLow0Flg TRUE = IRQ to Pi is active low for low 8 I/O device interrupt.

used1Mask This is the PI pin to use, select only one of the masks shown above.

activeLow1Flg TRUE = IRQ to Pi is active low for the high 8 I/O device interrupt.

Returns:

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

6.2.15 SetOutputs()

This function is used to set the outputs.

Syntax:

int_t SetOutputs(uint16_t outputs);

Arguments:

outputs The outputs to set.
Note: as open collector outputs are used, a logic ‘1’ = On which pulls the output low.

Returns:

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

6.3 DIO16-24V Digital I/O board 'Python' library support functions

This group contains 'Python' functions for supporting the DIO16-24V digital I/O board.
Most functions are masks 16bits wide where,
Bit0 = I/O 1, through to Bit15 = I/O 16.
This board used a Microchip Mcp23s17 device and this should be consulted when using these functions.

The class is ***PyPixieBoardDio16_24V***
They all require the ***import PixiePy*** module.

6.3.1 PyPixieBoardDio16_24V()

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

Syntax:
object = PixiePy.PyPixieBoardDio16_24V(i2cAddress, busId)

Arguments:

i2cAddress	I2C address for the board to access.
busId	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

Syntax:
result = object.Initialise(channel, speedHz)

Arguments:

channel	The SPI channel to use, ***PIXIE_SPI_CHAN_0, PIXIE_SPI_CHAN_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 GetInputPolarity()

This function is used to read the input polarity register.

Syntax:
result, polarity = object.GetInputPolarity()

Arguments:

Returns:

result	***0*** or ***E...*** Linux error code.
polarity	The current polarity mask. 0 = no invert, 1 = inverted.

6.3.5 GetInputs()

This function is used to read the current state of the inputs.

Syntax:
result, inputs = object.GetInputs()

Arguments:

Returns:

result	***0*** or ***E...*** Linux error code.
inputs	The current inputs mask. 0 = Low, 1 = High, unless inverted by the input polarity

6.3.6 GetInterruptEnable()

This function is used to read the current state of the input interrupt enables.

Syntax:
result, enables = object.GetInterruptEnable()

Arguments:

Returns:

result	***0*** or ***E...*** Linux error code.
enables	Current interrupt enables mask 0 = Disabled, 1 = Enabled.

6.3.7 GetInterruptFlags()

This function is used to read the current state of the input interrupt pending flags.

Syntax:
result, flags = object.GetInterruptFlags()

Arguments:

Returns:

result	***0*** or ***E...*** Linux error code.
flags	The current interrupt pending flags mask. 0 = No interrupt, 1 = Interrupt pending.

6.3.8 GetIntOnChange()

This function is used to read the current state of the input interrupt on change control values.

Syntax:
result, control, defValue = object.GetIntOnChange()

Arguments:

Returns:

result	***0*** or ***E...*** Linux error code.
control	The current INTCON register mask.
defValue	The current DEFVAL register mask.

6.3.9 GetOutputs()

This function is used to read the current state of the outputs.

Syntax:
result, outputs = object.GetOutputs()

Arguments:

pOutputs	Is a pointer to return the current outputs mask in.

Returns:

result	***0*** or ***E...*** Linux error code.
outputs	The current outputs mask.

6.3.10 SetCeDecode()

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

Syntax:

Result = object.SetCeDecode(usedMask, polMask, adrIds)

Arguments:

usedMask 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.

polMask 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

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.

6.3.11 SetInputPolarity()

This function is used to set the value of the input polarity register.

Syntax:

result = object.SetInputPolarity(polarity)

Arguments:

polarity The polarity mask to set.
0 = no invert, 1 = inverted.

Returns:

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

6.3.12 SetInterruptEnable()

This function is used to set the current input interrupt enables register.

Syntax:
result = object.SetInterruptEnable(enables)

Arguments:

enables	The enables to set. 0 = Disabled, 1 = Enabled.

Returns:

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

6.3.13 SetIntOnChange()

This function is used to set the input interrupt on change control values.

Syntax:
result = object.SetIntOnChange(control, defValue)

Arguments:

control	INTCON register value.
defValue	DEFVAL register value.

Returns:

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

6.3.14 SetIrq()

This function is used to set the interrupt mapping for the low and high I/O devices.

Syntax:

Result = object.SetIrq(used0Mask, activeLow0Flg, used1Mask, activeLow1Flg)

Arguments:

used0Mask	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***
activeLow0Flg	1 = IRQ to Pi is active low for low 8 I/O device interrupt.
used1Mask	This is the PI pin to use, select only one of the masks shown above.
activeLow1Flg	1 = IRQ to Pi is active low for the high 8 I/O device interrupt.

Returns:

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

6.3.15 SetOutputs()

This function is used to set the outputs.

Syntax:

result = object.SetOutputs(outputs)

Arguments:

outputs	The outputs to set. Note: as open collector outputs are used, a logic ‘1’ = On which pulls the output low.

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