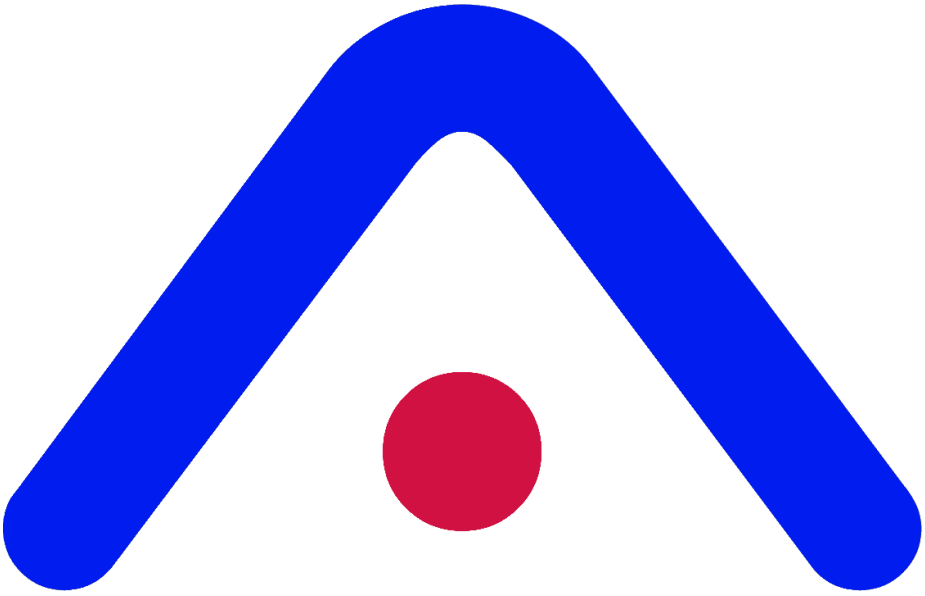


PIXIE BOARD
ISOLATED CANBUS/RS232/RS485 USER MANUAL

The information contained herein is believed to be accurate as of the date of this publication. AEL Microsystems Ltd assumes no liability for errors, or for any incidental, consequential, indirect, or special damages, including, without limitation, loss of use, loss or alteration of data, delays or lost profits or savings, arising from the use of this document, or use of any product, circuit or software described herein or the product which it accompanies.

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

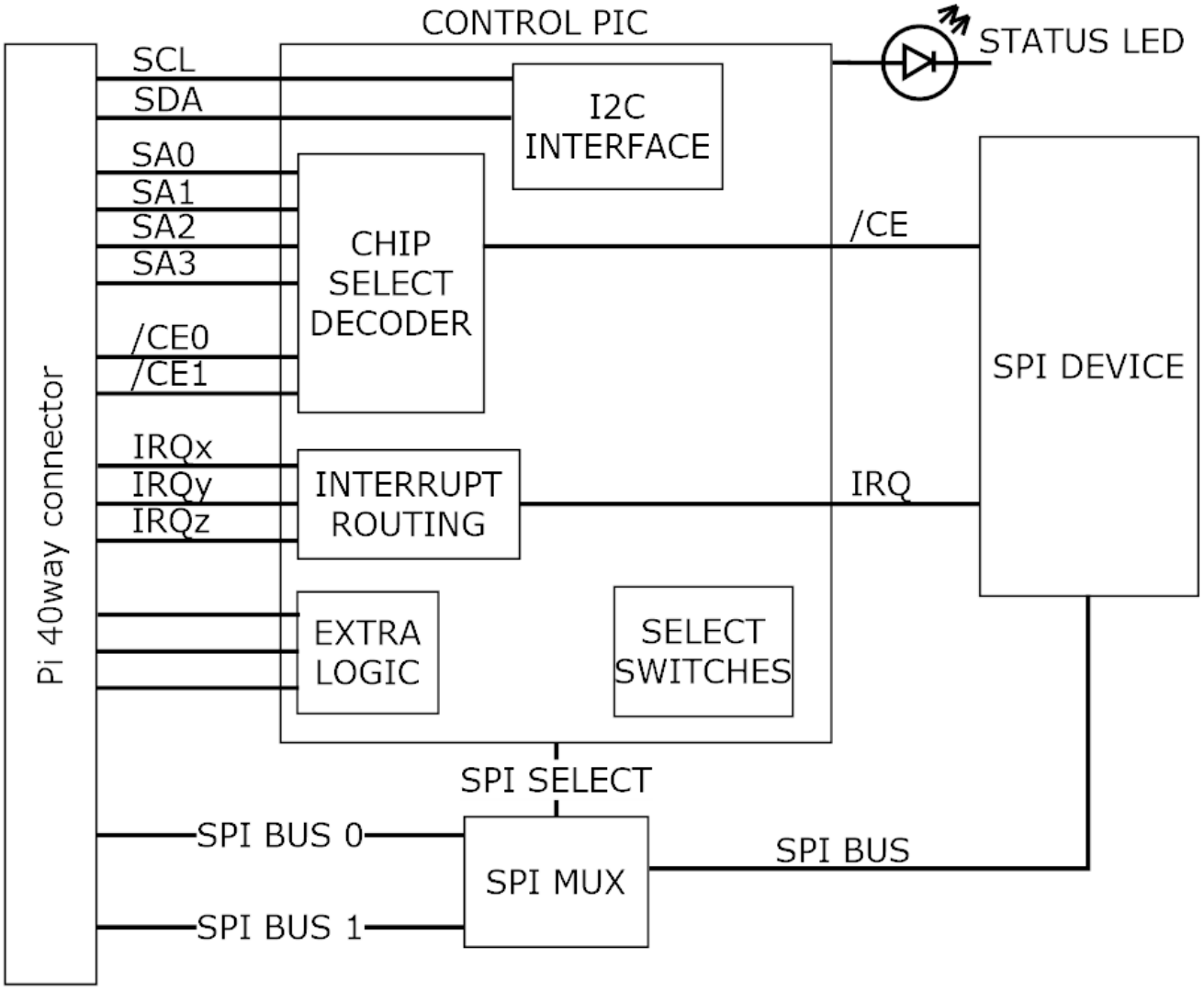
This shows the control overview of the **PIXIE** board.

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 **CAN-SERIAL-ISO** is a **PIXIE** board which contains an isolated CANFD bus interface, an RS232 isolated interface and an RS485 full/half duplex isolated interface.

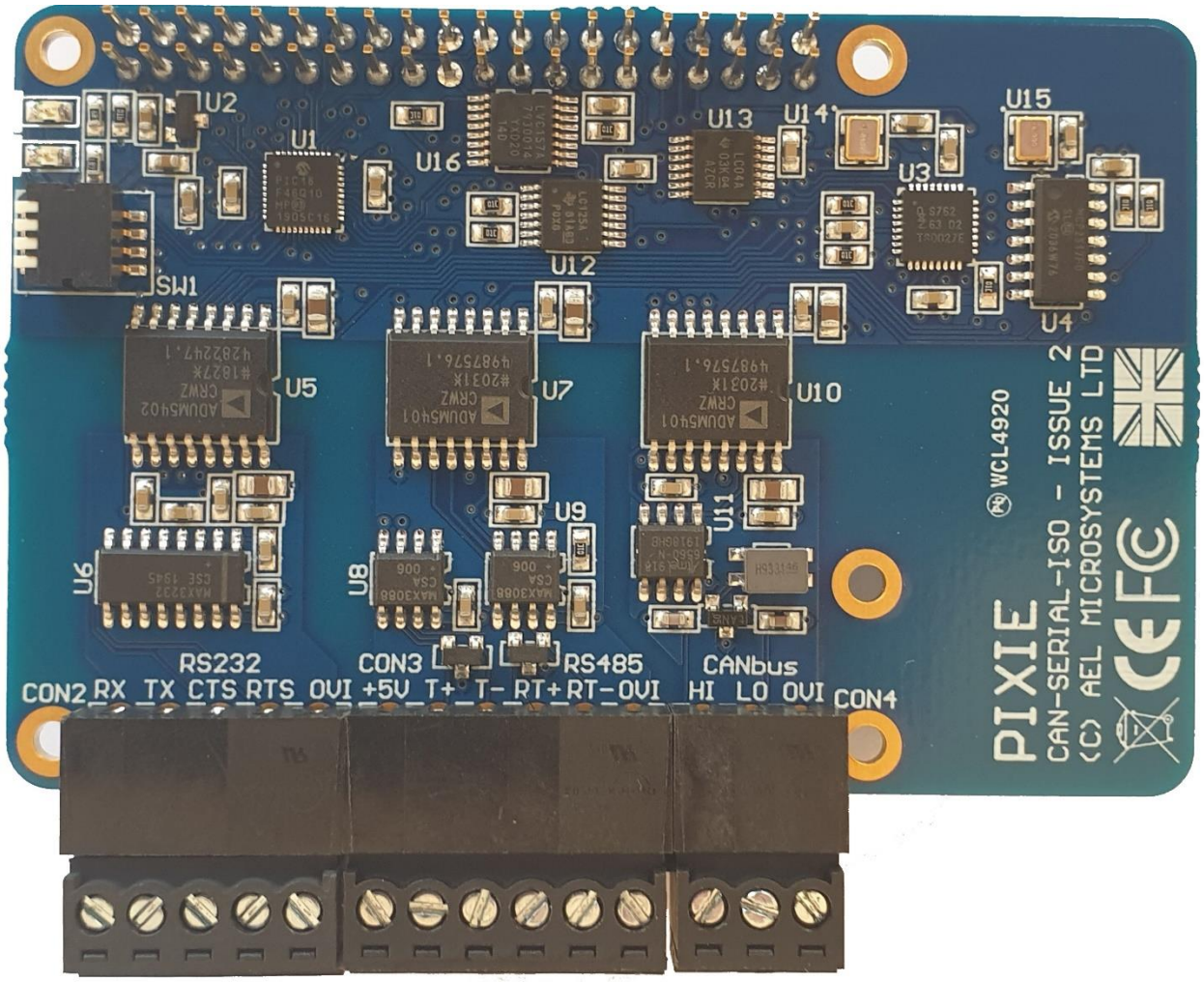
The CANFD controller is a MCP2517FD device and is isolated from the Raspberry Pi circuit by a galvanically isolated device.
It supports CAN V2.0 and FD with speeds up to 8mHz.
For full details of the device its datasheet should be consulted.

The RS232 and RS485 are supported by a Sc16is762 dual UART controller and are isolated from the Raspberry Pi circuit by a galvanically isolated devices.
For full details of the device its datasheet should be consulted.

4.1 Specification.

CAN bus	CAN V2.0 and CANFD
Speeds	Up to 8MHz
Galvanically isolated	<= 1000V
RS232	All standard speeds supported
HW handshake	RTS and CTS
Galvanically isolated	<= 1000V
RS485	All standard speeds supported
Duplex	Half and Full
Galvanically isolated	<= 1000V
Power consumption	50mA
I2C speed	<=100kHz
SPI speed	<=15MHz
Temperature	0-70C

4.2 I/O connections.



Above shows the connector positions and identifiers for the communication interfaces.

CON2 is a 5-way connector for the RS232 interface.
CON3 is a 6-way connector for the RS485 interface.
CON4 is a 3-way connector for the CAN bus interface.

Signals for **CON2**, RS232:

RX The data receive input.
TX The data transmit output.
CTS The Clear To Send input.
RTS The Ready To Send output.
0VI Isolated 0V common for the RS232 only.

Signals for **CON3**, RS485:

+5V Isolated +5V for the RS485 only, this is for reference only and not powering external circuits.
T+ Transmit + output (Full duplex).
T- Transmit - output (Full duplex).
RT+ Receive/Transmit + (Half duplex), Receive + input (Full duplex).
RT- Receive/Transmit - (Half duplex), Receive - input (Full duplex).
0VI Isolated 0V common for the RS485 only.

Signals for **CON4**, CANbus:

HI CAN high signal.

LO CAN low signal.

0VI Isolated 0V common for the CANbus only.

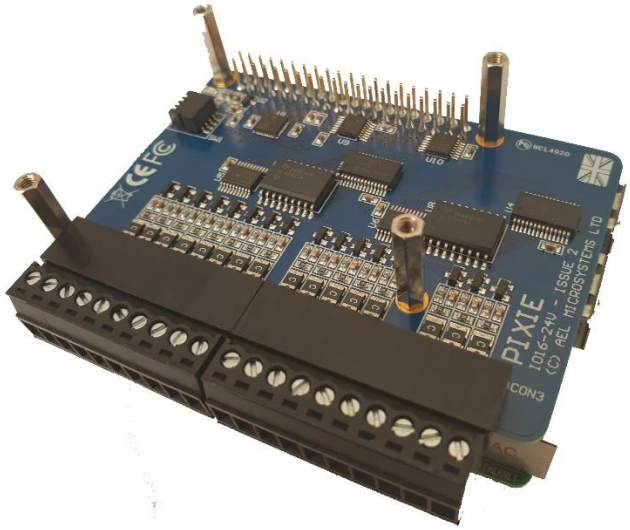
5.1 Insulate USB connector housing on Raspberry Pi 4

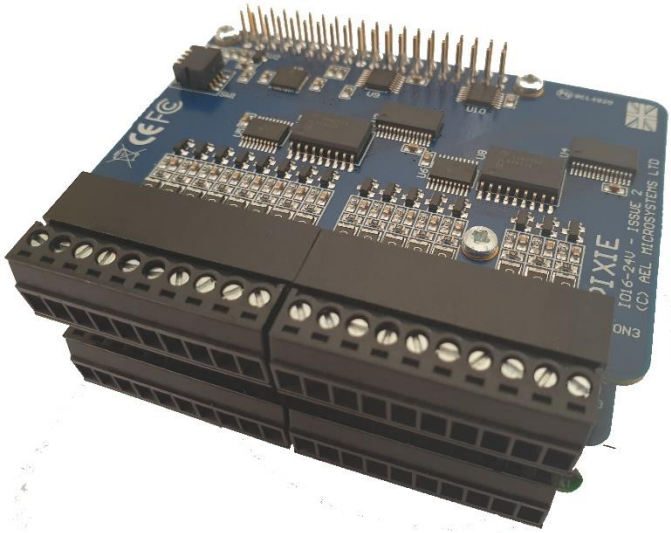
WARNING

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.

5.2 Mounting the boards.

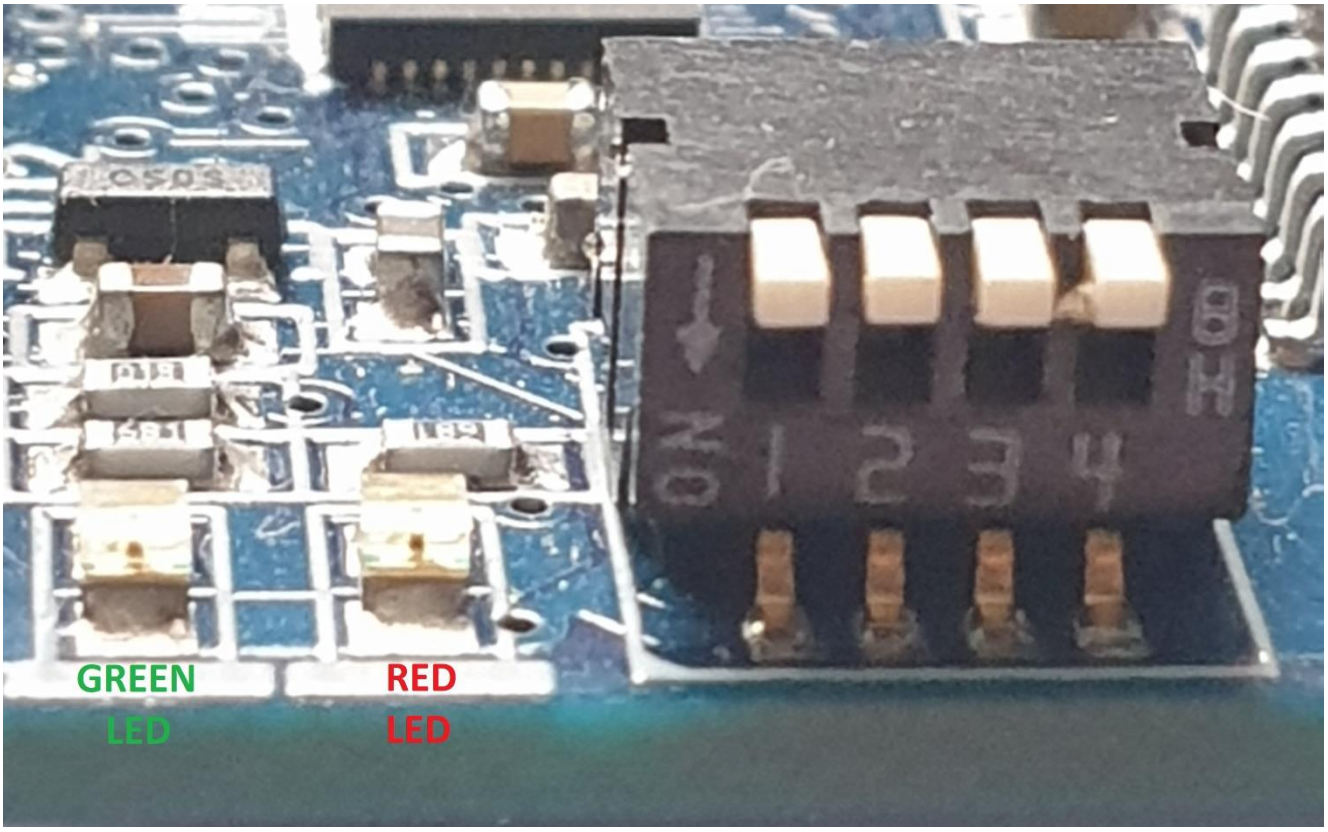
Using M2.5mm - 4mmAF – 17mm long hex standoff's mount the PIXIE boards onto the Raspberry Pi as shown. On some Pixie board's the connector CON3 obscures one of the mounting holes to the Raspberry Pi board when mounted directly to it ,so only 3 pillars are used which is sufficient to securely mount the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





5.4 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”

5.5 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.

5.6 Principle of operation.

The board uses a dual UART SC16IS762, and a CAN-FD device MCP2517 device.

Channel A of the UART is used for the RS232 interface and channel B for the RS485 interface. Linux uses standard drivers for this device and creates two TTY devices, /dev/ttyS0 for the RS232, and /dev/ttyS1 for the RS485 interface. If more than one of this type of board is fitted the additional /dev/ttSx devices will appear.
This device uses the board references /CE0 and /IRQ0.

The CAN bus device uses a driver that is used with the standard socket layering for CAN network messages.
See the section 5.7.2 for details for configuring the driver and linking it to the socket layering.
This device uses the board references /CE1 and /IRQ1.

5.7 Configuration and test functions.

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

5.7.1 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 full or half duplex operation of the RS485 interface.

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[1] - UTILITY MENU :
(?)-Menu help, (B)-Board ID, (L)- UART loopback, (R)- CAN RX, (S)- Board settings, (T)- CAN TX,
(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 /F-H, F(0), H(1): (0) >
Save to EEPROM Y/N     : (n) >

Current /F-H bit       : 0
```

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

Select /F-H, F(0), H(1) Sets either full or half duplex of the RS485 interface.

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

5.7.2 Board specific device driver configuration.

This board is supported by standard Linux device drivers but these need to be configured for the board to work correctly.
The driver for the sc16is762 is already contained in the standard Raspberry Pi kernel however the Mcp2517FD is not and needs to be built and installed separately either by building it standalone or including it in the Raspberry Pi kernel build. The procedures to do this are contained in the "PIXIE board configuration user guide" in the section **Linux changes.**

The device tree and config.txt file also needs to be setup so the same manual should also be consulted in the section **Configure the config.txt for additional devices** which describes this in greater detail, this will create network interfaces **can0, can1,…** and the serial devices **ttyS0, ttyS1,…**

For the network interface to work it needs to be brought up by entering:

sudo ip link set can0 up type can bitrate 100000

5.7.3 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[1] - UTILITY MENU :
(?)-Menu help, (B)-Board ID, (L)- UART loopback, (R)- CAN RX, (S)- Board settings, (T)- CAN TX,
(X)-Exit...
```

A full description of these are given by the **(?)-Menu help**
Ensure the correct device drivers and device trees are setup for these tests to work, see previous section.

(B)-Board ID Change the current board.

(L)-UART Loopback Performs a loopback test of the RS232 or RS485 UART's.
Using suitable wire links for the RS232, connect RX to TX and CTS to RTS.
For the RS485 connect T+ to RT+ and T- to RT- and set it to work as full duplex.

(R)-CAN RX Displays any message received on the CAN bus.
This requires an external tool to generate suitable messages.
Ensure the CAN network interface is up by entering:
sudo ip link set can0 up type can bitrate 100000

(S)-Board settings Change the board settings.
See previous section.

(T)-CAN TX Transmits various messages on the CAN bus.
This requires an external tool to monitor the transmitted messages.
Ensure the CAN network interface is up by entering:
sudo ip link set can0 up type can bitrate 100000

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 CAN-SERIAL-ISO board 'C' library support functions

This group contains 'C' functions for supporting the CAN-SERIAL-ISO communications 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 main software drivers for this board are supported by Linux, the functions described here are to configure the board.

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

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

6.1.1 PixieCanSerialConstruct()

This function is used to initialise the ***PixieCanSerialCtrl_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 PixieCanSerialConstruct(PixieCanSerialCtrl_t* pCtrl, uint16_t I2cAddress);

Arguments:

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

Returns:

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

6.1.2 PixieCanSerialDestroy()

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

Syntax:

int_t PixieCanSerialDestroy(PixieCanSerialCtrl_t* pCtrl);

Arguments:

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

Returns:

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

6.1.3 PixieCanSerialGetFullHalfDuplex()

This function is used to read the state of the full/half duplex control signal used for the RS485 interface.

Syntax:

int_t PixieCanSerialGetFullHalfDuplex(PixieCanSerialCtrl_t* pCtrl, uint8_t* pFhFlg);

Arguments:

pCtrl	Is a pointer to the ***PixieCanSerialCtrl_t*** structure to use.
pFhFlg	Is a pointer to return the signal state in. ***FALSE*** = Full duplex, ***TRUE*** = Half duplex.

Returns:

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

6.1.4 PixieCanSerialSaveSettings()

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 PixieCanSerialSaveSettings(PixieCanSerialCtrl_t* pCtrl);

Arguments:

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

Returns:

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

6.1.5 PixieCanSerialSetCeDecode()

This function is used to set the UART and CANbus chip enable decoding.

Syntax:

int_t PixieCanSerialSetCanCeDecode(
PixieCanSerialCtrl_t* pCtrl,
uint8_t usedUartMask,
uint8_t polUartMask,
uint8_t usedCanMask,
uint8_t polCanMask,
uint8_t* pAdrIds);

Arguments:

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

usedUartMask 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.

polUartMask 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

usedCanMask Same as ***usedUartMask*** shown above for this is for the DAC.

polCanMask Same as ***polUartMask*** 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.

6.1.6 PixieCanSerialSetFullHalfDuplex()

This function is used to set the state of the full/half duplex control signal used for the RS485 interface.

Syntax:

int_t PixieCanSerialSetFullHalfDuplex(PixieCanSerialCtrl_t* pCtrl, uint8_t fhFlg);

Arguments:

pCtrl	Is a pointer to the ***PixieCanSerialCtrl_t*** structure to use.
fhFlg	The state of the signal. ***FALSE*** = Full duplex, ***TRUE*** = Half duplex.

Returns:

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

6.1.7 PixieCanSerialSetIrq()

This function is used to set the interrupt mapping for the CANbus and UART devices.

Syntax:

int_t PixieCanSerialSetIrq(
PixieCanSerialCtrl_t* pCtrl,
uint16_t usedUartMask,
uint16_t usedCanMask,
bool_t activeLowUartFlg,
bool_t activeLowCanFlg);

Arguments:

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

usedUartMask 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

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

activeLowUartFlg TRUE = IRQ to Pi is active low for UART interrupt.

activeLowCanFlg TRUE = IRQ to Pi is active low for CANbus interrupt.

Returns:

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

6.2 CAN-SERIAL-ISO board 'C++' library support functions

This group contains 'C++' functions for supporting the CAN-SERIAL-ISO communications 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 main software drivers for this board are supported by Linux, the functions described here are to configure the board.

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

6.2.1 PixieBoardCanSerial()

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

Syntax:
PixieBoardCanSerial(uint16_t i2cAddress);

Arguments:
i2cAddress I2C address for the board to access.

6.2.2 GetFullHalfDuplex()

This function is used to read the state of the full/half duplex control signal used for the RS485 interface.

Syntax:
int_t GetFullHalfDuplex(uint8_t* pFhFlg);

Arguments:
pFhFlg Is a pointer to return the signal state in.
FALSE = Full duplex, ***TRUE*** = Half duplex.
Returns:
PIXIE_OK Completed OK.
E... Linux error code.

6.2.3 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.

6.2.4 SetCeDecode()

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

Syntax:

int_t SetCeDecode(
uint8_t usedUartMask,
uint8_t polUartMask,
uint8_t usedCanMask,
uint8_t polCanMask,
uint8_t* pAdrIds);

Arguments:

usedUartMask This is the used mask for the UART 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.

polUartMask This is the polarity mask for the UART 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

usedCanMask Same as ***usedUartMask*** shown above for this is for the CAN.

polCanMask Same as ***polUartMask*** shown above for this is for the CAN.

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.

6.2.5 SetIrq()

This function is used to set the interrupt mapping for the CANbus and UART devices.

Syntax:

int_t SetIrq(
uint16_t usedUartMask,
bool_t activeLowUartFlg,
uint16_t usedCanMask,
bool_t activeLowCanFlg);

Arguments:

usedUartMask 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

activeLowUartFlg 1 = IRQ to Pi is active low for UART interrupt.

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

activeLowCanFlg 1 = IRQ to Pi is active low for CANbus interrupt.

Returns:

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

6.2.6 SetFullHalfDuplex()

This function is used to set the state of the full/half duplex control signal used for the RS485 interface.

Syntax:

int_t SetFullHalfDuplex(uint8_t fhFlg);

Arguments:

fhFlg	The state of the signal. ***FALSE*** = Full duplex, ***TRUE*** = Half duplex.

Returns:

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

6.3 CAN-SERIAL-ISO board 'Python' library support functions

This group contains 'Python' functions for supporting the CAN-SERIAL-ISO communications board.
The main software drivers for this board are supported by Linux, the functions described here are to configure the board.

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

6.3.1 PyPixieBoardCanSerial()

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

Syntax:
object = PixierPy.PyPixieBoardCanSerial(i2cAddress)

Arguments:
i2cAddress I2C address for the board to access.

6.3.2 GetFullHalfDuplex()

This function is used to read the state of the full/half duplex control signal used for the RS485 interface.

Syntax:
Result, fhFlg = object.GetFullHalfDuplex()

Arguments:
pFhFlg Is a pointer to return the signal state in.
FALSE = Full duplex, ***TRUE*** = Half duplex.
Returns:
result ***0*** or ***E...*** Linux error code.

fhFlg ***0*** = Full duplex, ***1*** = Half duplex.

6.3.3 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.

6.3.4 SetCeDecode()

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

Syntax:

Result = object.SetCeDecode(usedUartMask, polUartMask, usedCanMask, polCanMask, adrIds)

Arguments:

usedUartMask 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.

polUartMask 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

usedCanMask Same as ***usedCanMask*** shown above for this is for the CAN.

polCanMask Same as ***polCanMask*** shown above for this is for the CAN.

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.

6.3.5 SetFullHalfDuplex()

This function is used to set the state of the full/half duplex control signal used for the RS485 interface.

Syntax:

result = object.SetFullHalfDuplex(fhFlg)

Arguments:

fhFlg The state of the signal setting.
0 = Full duplex, ***1*** = Half duplex.

Returns:

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

6.3.6 SetIrq()

This function is used to set the interrupt mapping for the CANbus and UART devices.

Syntax:

Result = object.SetIrq(usedUartMask, activeLowUartFlg, usedCanMask, activeLowCanFlg)

Arguments:

usedUartMask 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

activeLowUartFlg 1 = IRQ to Pi is active low for UART interrupt.

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

activeLowCanFlg 1 = IRQ to Pi is active low for CANbus interrupt.

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