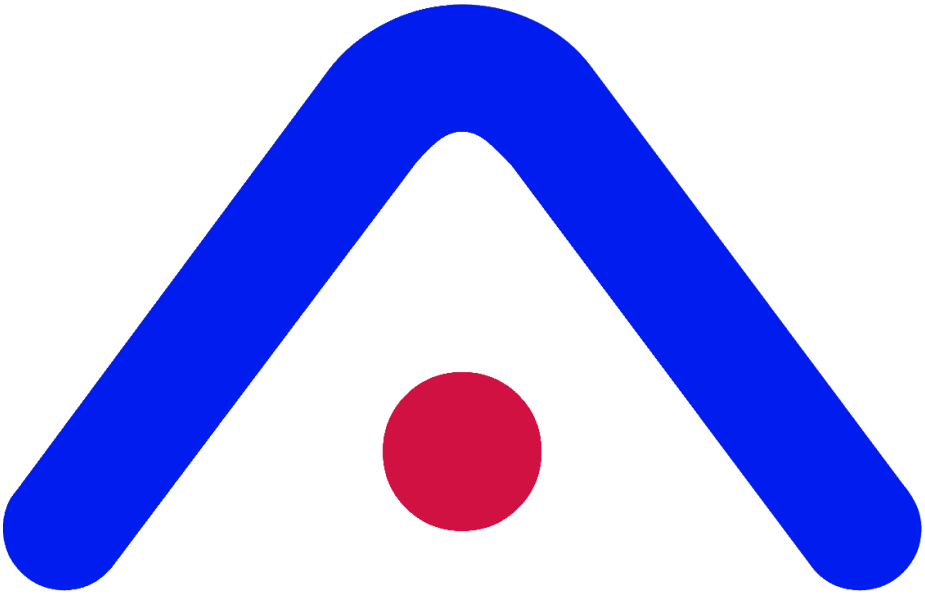


PIXIE BOARD
CONFIGURATION USER MANUAL

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 4
2.1 Caution 4
2.2 Forward note 4

3. Common concept 5
3.1 Control overview 5
3.2 More SPI devices 6
3.3 Configurable 6
3.4 Stackable 6
3.5 Updatable 6

4. Getting Started 6
4.1 Insulate USB connector housing on Raspberry Pi 4 6
4.2 Mounting the boards. 8
4.3 Set the boards identity. 9
4.4 Power up and LED status. 10
4.5 Multiple SPI addresses and chip selects overview 11
4.6 Additional device tree objects 11
4.7 Configure the SPI device tree 12
4.7.1 Configure the config.txt for the new SPI devices 12
4.7.2 Configure the config.txt for additional devices 13
4.8 Install the “PixieBoard” application 14
4.9 Enable I2C & the dreaded I2C clock stretching bug 14
4.10 Board detection and configuration 14
4.11 Configuration 17
4.11.1 Automatically assign SPI chip select decoding 17
4.11.2 Manual edit SPI bus and chip select decoding 20
4.11.3 Edit board to Raspberry Pi interrupt assignments 22
4.11.4 Board defaults 23
4.11.5 Verify overall configuration 24

5. Firmware updates 25
5.1 Set the I2C bus speed 25
5.2 Firmware update 25
5.2.1 Invalidate 26
5.2.2 Update 26
5.2.3 Verify 27
5.2.4 Forced boot mode. 27

6. Linux changes 28
6.1 Getting the kernel 28
6.2 Additional device tree objects 30
6.3 Building the kernel, modules, and device tree objects 31

7. Warranty conditions 34

8. Notes 35

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

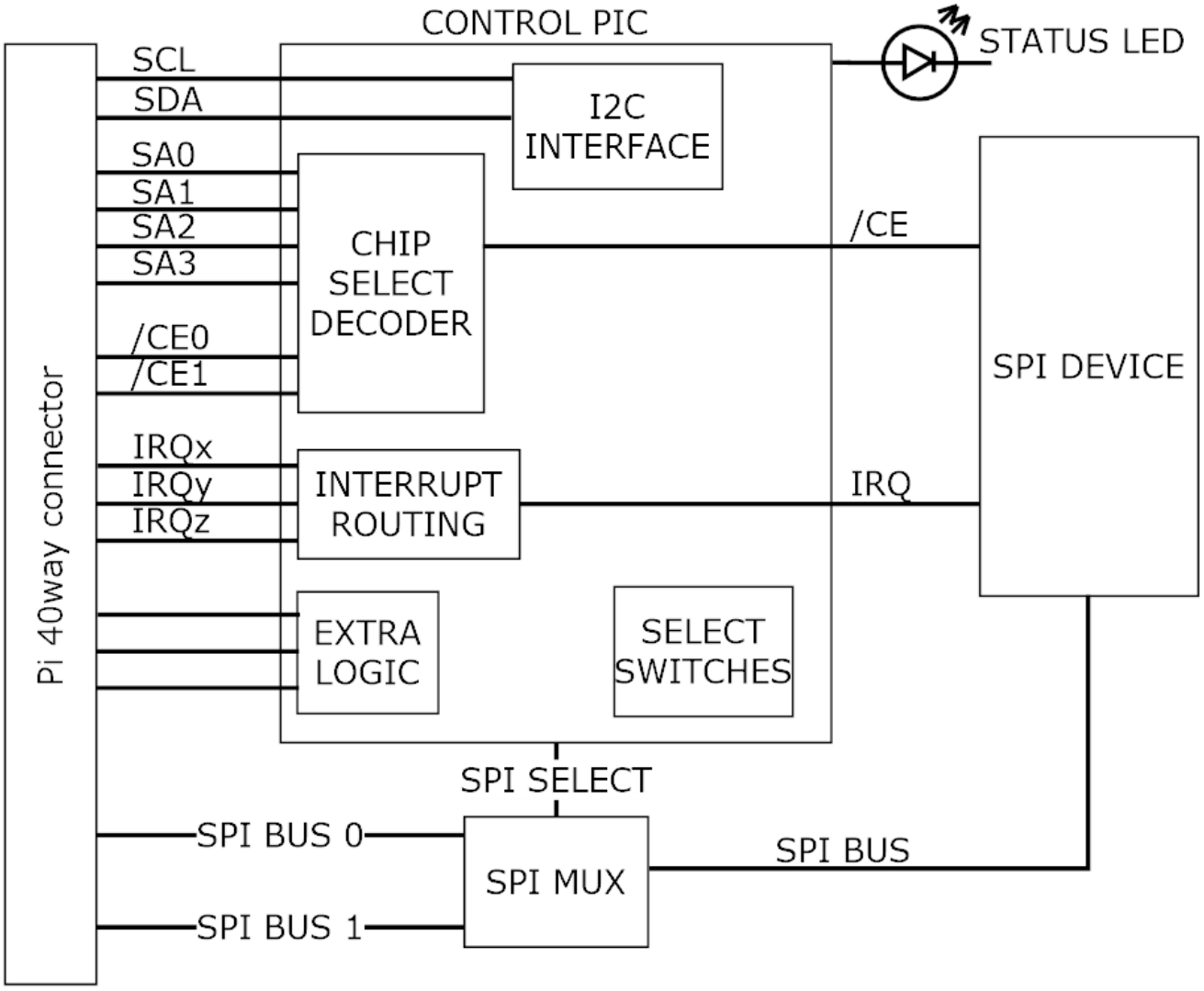
This shows the control overview of the **PIXIE** board.

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. Getting Started

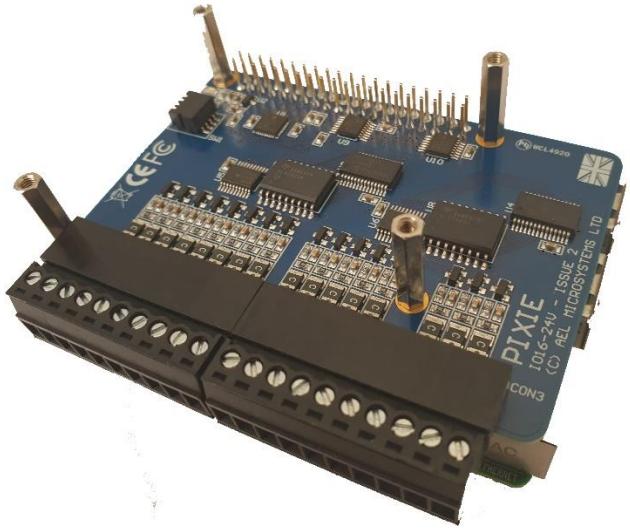
4.1 Insulate USB connector housing on Raspberry Pi 4

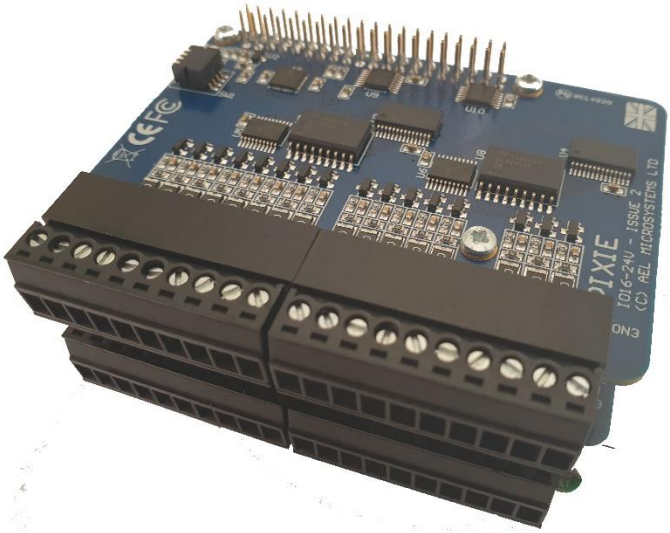
WARNING

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.

4.2 Mounting the boards.

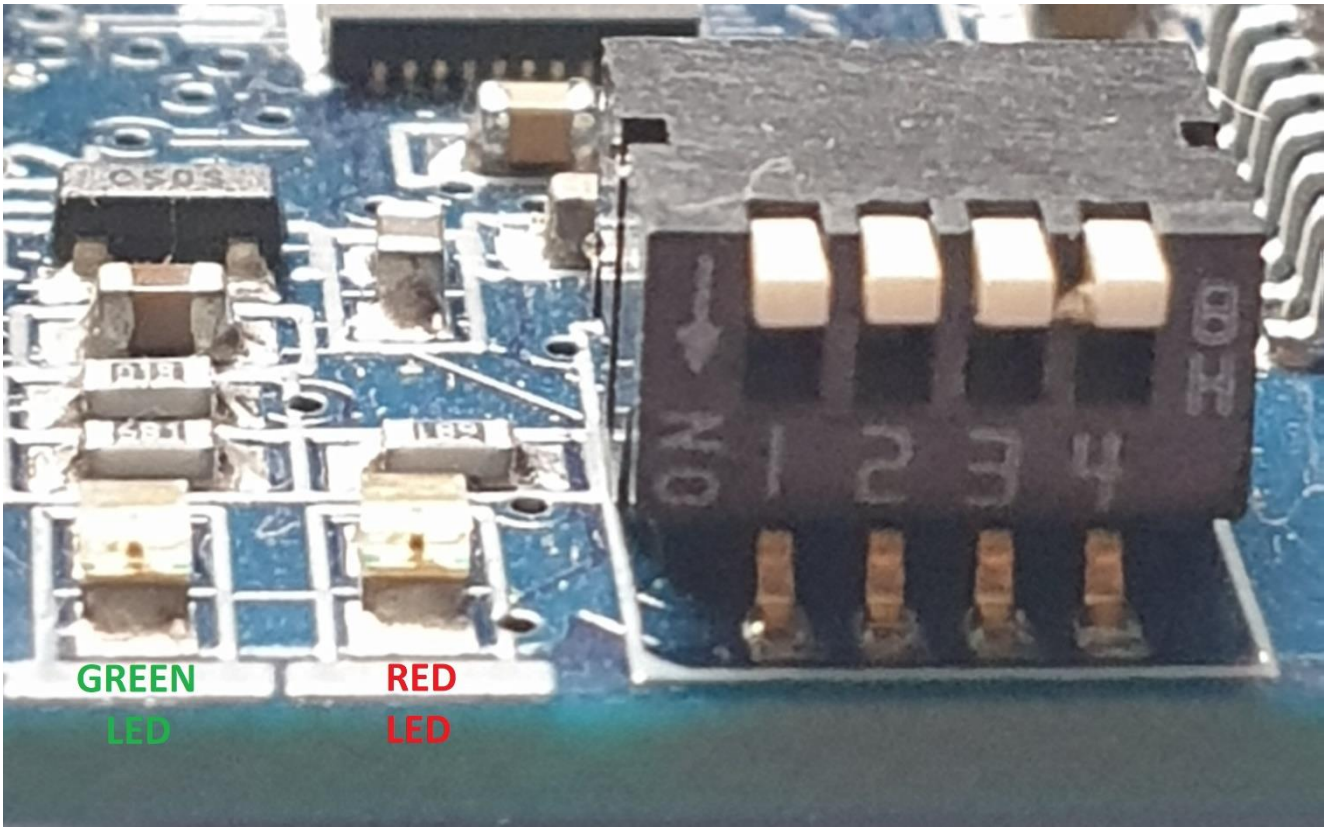
Using M2.5mm - 4mmAF – 17mm long hex standoff's mount the PIXIE boards onto the Raspberry Pi as shown. On some Pixie board's the connector CON3 obscures one of the mounting holes to the Raspberry Pi board when mounted directly to it ,so only 3 pillars are used which is sufficient to securely mount the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





4.3 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”.

4.4 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.

4.5 Multiple SPI addresses and chip selects overview

The method used to create multiple SPI devices and control the additional address lines required for operation of multiple Pixie boards is to make use of the GPIO finite state machine, which is configured using device tree overlays.
This is a very configurable way of assigning the sub address lines and also the main chip selects to a specific SPI device found in the **/dev** directory, e.g. **/dev/spidev0.x,** where 'x' is the device mapped to a sequence of sub address lines and chip selects.
When the SPI device is accessed the finite state machine puts the address lines and the chip selects into the correct enabled state at the start of the SPI access and then into the disabled state at the end.
There is very little delay in processing this in addition to the SPI access.

4.6 Additional device tree objects

This describes the changes needed to add the additional overlays to the Raspberry Pi in order to support the sub addresses used by the **PIXIE** boards.

1) Download the device tree source files **pixie-overlays-x.y.tgz** found at
   aelmicro.com > Support > Pixie Boards > Software > Pixie linux overlays

2) Extract the source files:
   cd /boot/overlays
   sudo tar xzf ~/Downloads/pixie-overlays-x.y.tgz

   The following pre-compiled overlay files are extracted:
   spi0-hw-sa4.dtbo, creates **spidev0.0** thru **spidev0.3**, uses 1 sub address and both CS0 & 1.
   spi0-hw-sa8.dtbo, creates **spidev0.0** thru **spidev0.7**, uses 2 sub address and both CS0 & 1.
   spi0-hw-sa16.dtbo, creates **spidev0.0** thru **spidev0.15**, uses 3 sub address and both CS0 & 1.
   spi0-hw-sa32.dtbo, creates **spidev0.0** thru **spidev0.31**, uses 4 sub address and both CS0 & 1.
   spi1-hw-sa6.dtbo, creates **spidev1.0** thru **spidev1.5**, uses 1 sub address and CS0, 1 & 2.
   spi1-hw-sa12.dtbo, creates **spidev1.0** thru **spidev1.11**, uses 2 sub address and CS0, 1 & 2.
   spi1-hw-sa24.dtbo, creates **spidev1.0** thru **spidev1.23**, uses 3 sub address and CS0, 1 & 2.
   mcp251xfd.dtbo, this is the can0 for the mc2517fd device using the SPI0 or SPI1 bus.

4.7 Configure the SPI device tree

On the Raspberry Pi run the **Preferences->Raspberry Pi Configuration**, then under the **Interfaces** tab enable SPI and I2C interfaces.
To add the additional SPI devices and make them available there are additional device tree overlay files that have been added and the **/boot/config.txt** file needs to be edited to activate them.

4.7.1 Configure the config.txt for the new SPI devices

Now the overlay objects are available they need to be activated in the **/boot/config.txt** file.
This example enables 4 possible selections **spidev0.0** thru **spidev0.3** are activated.

cd /boot
sudo nano config.txt

Find the entry **dtparam=spi0=on**
After this add the following entry:
dtoverlay=spi0-hw-sa4

Save the file and exit.
Reboot the Pi.

Using the file manager check the **/dev** directory for the entries **spidev0.0** thru **spidev0.3**

You are now in a position to fully utilise the **PIXIE** boards over the SPI interface.

4.7.2 Configure the config.txt for additional devices

To use the boards with special devices that use the SPI bus the following changes need to be made to the **/boot/config.txt** file, then the rebooted.

4.7.2.1 Sc16is762 UART used on the Serial/CAN PIXIE boards

These use the **sc16is762-spi0.dtbo** & **sc16is762-spi1.dtbo** overlays for the **/dev/ttySx** devices using the SPI0 and SPI1 bus.

In the config.txt file add the following entries at the end of the file if these are required:

dtoverlay=sc16is762-spi0,intpin=5,regnum=2,spispeed=8000000

This will create two **/dev/ttySx** device ports that are used on the SPI0 bus.

Set the **intpin=** parameter to the GPIO that is to be used for the interrupt from the UART which will be the same as that set when configuring the board.

Set the **regnum=** parameter to the spidev0.x channel you with to use will be the same as that set when configuring the board, e.g. 2 = **/dev/spidev0.2**

Set the **spispeed=** parameter to the SPI clocking speed in Hz, do not exceed the device maximum clock speed.

4.7.2.2 Mcp2517fd CAN bus controller used on the Serial/CAN PIXIE boards

These use the **mc251xfd-overlay.dts** overlay for the can0 and can1 network interfaces.

In the config.txt file add the following entries at the end of the file if these are required:

dtoverlay=mcp251xfd-sp0-0,oscillator=4000001,interrupt=6,regnum=3,speed=16000000

This will create the **can0** network interface using the **/dev/spidev0.3** device used on the SPI0 bus.

Set the **interrupt=** parameter to the GPIO that is to be used for the interrupt from the CAN controller which will be the same as that set when configuring the board.

Set the **oscillator=** parameter to the board oscillator clock in Hz.
If using the updated PLL driver this is 4000000, otherwise it 4000001 for the supplied driver.

Set the **regnum=** parameter to the spidev0.x channel you wish to use will be the same as that set when configuring the board, e.g. 3 = **/dev/spidev0.3**
This parameter is **not available in the standard file** set so the prebuilt **mcp251xfd.dtbo** needs to be copied into the **/boot/overlays/** directory so that the mapping can be moved from /dev/spidev0.0

Set the **speed=** parameter to the SPI clocking speed in Hz, do not exceed the device maximum clock speed.

4.8 Install the "PixieBoard" application

Using the Raspberry Pi web browser download the "**PixieBoard**" application **pixie-board-x.y.tgz** found at **aelmicro.com > Support > Pixie Boards > Software > "PixieBoard" Application**
which will usually end up in the **/home/pi/Downloads** directory on the Raspberry Pi.

Open a command line window and enter the following:

sudo tar xzf ~/Downloads/pixie-board-x.y.tgz -C /usr/bin/

4.9 Enable I2C & the dreaded I2C clock stretching bug

Ensure the I2C driver is enabled, go to **Preferences->Raspberry Pi Configuration**.
Under the interfaces tab, enable the I2C interface.
Press **OK** and then reboot the Pi.

For a long time, there has been an issue with the Raspberry Pi not implementing the i2c clock stretching correctly resulting in access to the PIXIE boards not been 100% accurate. This is shown up usually when trying to update the firmware of the board when the baud rate of the i2c bus is set to its default of 100kHz, which results in verification errors.
There are various workaround and claims that it has been fixed in more recent silicon, but this may not always be the case.
Some of the workaround methods are listed below:

1) Slow down the baud rate to about 50kHz, this usually does the trick.
   Edit the **/boot/config.txt** file and add the parameter **dtparam=i2c_arm_baudrate=50000**
   Then save & reboot.

2) Change the i2c bus driver to the **i2c-gpio** type, this is guaranteed at all speeds but as it's a software implementation it is higher on CPU load and has a maximum i2c clock speed of about 150kHz.
   Edit the **/boot/config.txt** file and comment out the parameter **dtparam=i2c_arm_on**, then add **dtoverlay=i2c-gpio,i2c_gpio_sda=2, i2c_gpio_scl=3, i2c_gpio_delay_us=2,bus=1**
   Then save & reboot.

Note that all the baud rates can be affected by the core clock speed of the CPU.

4.10 Board detection and configuration

Ensure the SPI driver is enabled, go to **Preferences->Raspberry Pi Configuration**.
Under the interfaces tab, enable the SPI interface.
Press **OK** and then reboot the Pi.

The PIXIE configuration tool can be either run as single command line action or use a hierarchical menu which has additional test functions available to check the configured boards. The single line action command is useful in a script file to be run to quickly configure a system with multiple boards installed.

The following instructions are described primarily when using the menu system, but where applicable the equivalent single line command action is also described.
The basic syntax for command line actions is **sudo PixieBoard [<boardId>] [<opts>] [<CEx and settings>]**
The **<boardId>** is a value 0 to 15 and is the board identified by the setting if the board switches.
e.g. **sudo PixieBoard 3** will display the settings of board 3.

For the menus system open a command line window and enter the following:

sudo PixieBoard

You are first presented with a menu.

```
pi@raspberrypi:~ $ PixieBoard

PIXIE BOARD[0] - MAIN MENU :
(?)-Menu help, (B)-Board ID, (C)-Configuration, (F)-Firmware, (S)-Scan for boards,
(U)-Board Utilities, (X)-Exit...
```

Menu items are selected by pressing the key letter shown in the brackets,
e.g. **(?)** for the "Menu Help" or **(B)** for the "Board ID".

Letter case is not normally important but on some of the selections a more verbose output may be provided if an upper-case letter is used, the **Menu help** will show these differences.

Using a lower-case '**s**' will scan for fitted boards and display those found:

```
PIXIE BOARD[0] - MAIN MENU :
(?)-Menu help, (B)-Board ID, (C)-Configuration, (F)-Firmware, (S)-Scan for boards,
(U)-Board Utilities, (X)-Exit...

 [** PIXIE BOARD 0 **] - Found.
 [** PIXIE BOARD 8 **] - Found.
```

The equivalent for single command action is:

sudo PixieBoard <boardId> -s

Using an upper-case '**S**' will scan for fitted boards and display more details:

```
PIXIE BOARD[0] - MAIN MENU :
(?)-Menu help, (B)-Board ID, (C)-Configuration, (F)-Firmware, (S)-Scan for boards,
(U)-Board Utilities, (X)-Exit...

 [** PIXIE BOARD 0 **]
 Serial Number     : "12345"
 Board Revision    : "1"
 Board Type        : "ADC8-DAC2"
 Board Mode        : "Application"
 Software Version  : "1.0"
 Software Date     : "01 May 2020"

 [** PIXIE BOARD 8 **]
 Serial Number     : "12345"
 Board Revision    : "1"
 Board Type        : "DIG16-24V"
 Board Mode        : "Application"
 Software Version  : "1.0"
 Software Date     : "01 May 2020"
```

The equivalent for single command action is:

sudo PixieBoard <boardId> -S

In our examples above there are 2 boards fitted, an Analogue I/O board with its board identifier set to 0, and a Digital I/O board with the board identifier set to 8.

All the menus will normally show the **(?)-Menu help** and **(B)-Board ID** options:

(?)-Menu help will show the help details for the current menu.

```
PIXIE BOARD[0] - MAIN MENU :
(?)-Menu help, (B)-Board ID, (C)-Configuration, (F)-Firmware, (S)-Scan for boards,
(U)-Board Utilities, (X)-Exit...

MENU HELP...
(B)-Board ID
  Change the board you wish to use, values = 0 to 15.

(C)-Configuration
  Change the configuration of the target board.

(F)-Firmware
  Firmware update and verification of the target board.

(S)-Scan for boards
  Scan for fitted boards.
  Use 's' for shortform, or 'S' for verbose.

(U)-Board Utilities
  Various board test utility verification and diagnostic tests.

(X)-Exit
  Exit the application.
```

(B)-Board ID allows the current board selection to be changed and will also display the newly selected board details. The currently selected board is shown in the square brackets in the menu title.

```
PIXIE BOARD[0] - MAIN MENU :
(?)-Menu help, (B)-Board ID, (C)-Configuration, (F)-Firmware, (S)-Scan for boards,
(U)-Board Utilities, (X)-Exit...

 Enter board ID 0..15 (0) > 8

 [** PIXIE BOARD 8 **]
 Serial Number     : "12345"
 Board Revision    : "1"
 Board Type        : "DIG16-24V"
 Board Mode        : "Application"
 Software Version  : "1.0"
 Software Date     : "01 May 2020"
```

The equivalent for single command action is:

sudo PixieBoard <boardId> -i

<boardId> = 0,1,2…15

4.11 Configuration

Having scanned for the boards and obtained the list of those fitted it is now time to configure each board for use.

Select **(C)-Configuration**, the menu changes to:

```
PIXIE BOARD[0] - CONFIG MENU :
(?)-Menu help, (A)-Auto assign, (B)-Board ID, (C)-Chip selects, (D)-Apply defaults,
(I)-Interrupts, (M)-SPI mapping, (S)-Show settings, (V)-Verify boards, (X)-Exit...
```

Before you start your configuration, you will need to decide how you are going to map each board to the SPI device driver or you can take the easy option and use the **(A)-Auto assign** option.

4.11.1Automatically assign SPI chip select decoding

Using **(A)-Auto assign** is ideal if all the boards are going to be using the SPI0 bus only, if you require anything more specific then this will give you a good starting point and then use the **(C)-Chip selects** and **(M)-SPI Mapping** options to change the settings to suit specific requirements.

Select **(A)-Auto assign** and the two boards found will be mapped as show:

```
PIXIE BOARD[0] - CONFIG MENU :
(?)-Menu help, (A)-Auto assign, (B)-Board ID, (C)-Chip selects, (D)-Apply defaults,
(I)-Interrupts, (M)-SPI mapping, (S)-Show settings, (V)-Verify boards, (X)-Exit...

Scanning...

Number of boards to configure = 2

Board[0]-CE0 set to SPI0, CE0.0 and sub address[0], "/dev/spidev0.0"
Board[0]-CE1 set to SPI0, CE0.1 and sub address[0], "/dev/spidev0.1"
Board[8]-CE0 set to SPI0, CE0.0 and sub address[1], "/dev/spidev0.2"
```

To show the current settings for the currently selected board use the **(S)-Show settings** option.
This shows how the SPI, chip selects, and sub address signals have been allocated to the board.

```
[** PIXIE BOARD 0 **]
Serial Number     : "12345"
Board Revision    : "1"
Board Type        : "ADC8-DAC2"
Board Mode        : "Application"
Software Version  : "1.0"
Software Date     : "01 May 2020"

(Board CE0, SPI0 Decoding)
 Board CE0 Signal: Enabled, Uses Pi SPI0 signal CE0.0, (GPIO8)
 Board CE0 Level : Active Low
 SPI0 Decode A0  : Active Low , Uses Pi signal (GPIO22)
 SPI0 Decode A1  : Unused
 SPI0 Decode A2  : Unused
 SPI0 Decode A3  : Unused
 SPIDEV path     : "/dev/spidev0.0"

(Board CE1, SPI0 Decoding)
 Board CE1 Signal: Enabled, Uses Pi SPI0 signal CE0.1, (GPIO7)
 Board CE1 Level : Active Low
 SPI0 Decode A0  : Active Low , Uses Pi signal (GPIO22)
 SPI0 Decode A1  : Unused
 SPI0 Decode A2  : Unused
 SPI0 Decode A3  : Unused
 SPIDEV path     : "/dev/spidev0.1"

(Board IRQ's Mapping)
 Board IRQ0 Signal: Disabled
 Board IRQ1 Signal: Unused
```

The equivalent for single command action is:

sudo PixieBoard 0

The output shows the boards main detail, serial number, revision, etc.

Under the **(Board CE0, SPI0 Decoding)** title, the boards CE0 signal which is used to access the ADC device is enabled and is using the SPI0 signal CE0.0 on GPIO8. It is also using SPI0 sub address decode A0 which is decoded active low and is connected to the GPIO22 signal, all the other address lines are unused.
The path on which this device will be found is /dev/spidev0.0

Under the **(Board CE1, SPI0 Decoding)** title, the boards CE1 signal which is used to access the DAC device is enabled and is using the SPI0 signal CE0.1 on GPIO7. It is also using SPI0 sub address decode A0 which is decoded active low and is connected to the GPIO22 signal, all the other address lines are unused.
The path on which this device will be found is /dev/spidev0.1

Under the **(Board IRQ Mapping)** title, it shows that there are no interrupts routed from this board to the Raspberry Pi GPIO pins.

Select the other board by changing the Board ID to 8 and show the settings for that.

```
[** PIXIE BOARD 8 **]
Serial Number     : "12345"
Board Revision    : "1"
Board Type        : "DIG16-24V"
Board Mode        : "Application"
Software Version  : "1.0"
Software Date     : "01 May 2020"

(Board CE0, SPI0 Decoding)
 Board CE0 Signal: Enabled, Uses Pi SPI0 signal CE0.0, (GPIO8)
 Board CE0 Level : Active Low
 SPI0 Decode A0  : Active High, Uses Pi signal (GPIO22)
 SPI0 Decode A1  : Unused
 SPI0 Decode A2  : Unused
 SPI0 Decode A3  : Unused
 SPIDEV path     : "/dev/spidev0.2"

(Board CE1, SPI0 Decoding)
 Board CE1 Signal: Unused
 Board CE1 Level : Unused
 SPI0 Decode A0  : Unused
 SPI0 Decode A1  : Unused
 SPI0 Decode A2  : Unused
 SPI0 Decode A3  : Unused
 SPIDEV path     : Unused

(Board IRQ's Mapping)
 Board IRQ0 Signal: Disabled
 Board IRQ1 Signal: Disabled
```

The equivalent for single command action is:

sudo PixieBoard 8

Two differences are,
1) the Digital I/O board only used 1 chip select (CE0) which is the same as that for the ADC on the other board,
2) the SPI0 Decode A0 is now active high so this will decode this boards device to /dev/spidev0.2

4.11.2Manual edit SPI bus and chip select decoding

To manually edit the chip select decoding use the **(C)-Chip selects**, and to edit the SPI bus and sub address source signals use the **(M)-SPI Mapping**.

To edit the SPI bus and sub address source signals select **(M)-SPI Mapping**, then at the prompts enter the source SPI to use, either 0 or 1, followed by the GPIO source pin number for sub address A0,1,2,3.
Pressing carriage return at each prompt without entering a value will retain the original value.

```
Select SPI source BUS: (0) >
Select A0 source: (22) >
Select A1 source: (23) >
Select A2 source: (24) >
Select A3 source: (27) >

 [** PIXIE BOARD 0 **] - Set values
```

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

Select SPI source BUS This takes a value of 0 or 1 and selects the routing of the SPI signals to/from the appropriate GPIO pins.
The selection of the bus is for the board as a whole so if there are two SPI devices on the board they will both use the same bus but different SPI chip selects.

Select A0 source This sets the GPIO pin used as A0 in the SPI chip select decoder.
Its default is GPIO22 for SPI0 or GPIO5 for SPI1.

Select A1 source This sets the GPIO pin used as A1 in the SPI chip select decoder.
Its default is GPIO23 for SPI0 or GPIO6 for SPI1.

Select A2 source This sets the GPIO pin used as A2 in the SPI chip select decoder.
Its default is GPIO24 for SPI0 or GPIO12 for SPI1.

Select A3 source This sets the GPIO pin used as A3 in the SPI chip select decoder.
Its default is GPIO27 for SPI0 or GPIO13 for SPI1.

The equivalent for single command action to display the current values is:

sudo PixieBoard <boardId> spi=0

The equivalent for single command action to change the values is:

sudo PixieBoard <boardId> spi=0 a0=22 a1=23 a2=24 a3=27

To edit the chip selects, sub addresses to use and their levels, select **(C)-Chip selects**, then at the prompts enter the boards target chip select if prompted, followed by the SPI CE source signal to use, and the use and the level of the sub addresses.
The sub address values are 'X' signal not used, '0' signal needs to be low, '1' signal needs to be high.
Pressing carriage return at each prompt without entering a value will retain the original value.

```
Select CE to edit: (0) >
Select CEx source for CE0: (0) >
Select decode A0: (0) >
Select decode A1: (X) >
Select decode A2: (X) >
Select decode A3: (X) >

 [** PIXIE BOARD 0 **] - Set values
```

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

Select CE to edit This will only be shown if the board requires more than 1 chip select.
A value of 0 or 1 sets the board chip enable signal to change.

Select CEx source for CE0
Select CEx source for CE1
Only one of these messages are shown depending on the board chip select entered at the previous prompt.
Values 'X', '0', '1', select SPI0 bus, disabled or CE0.0 (GPIO8) or CE0.1 (GPIO7)
Values 'X', '0', '1', '2', select SPI1 bus, disabled, CE1.0 (GPIO18), CE1.1 (GPIO17) or CE1.2 (GPIO16)
For an individual chip select that does not use the address decodes the following GPIO numbers can be used:
5, 6, 7, 8, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27
and the prompts for the following address decodes do not appear.

Select decode A0 This determines if the A0 sub address signal is used and what level is expected.
'X' signal not used, '0' signal needs to be low, '1' signal needs to be high.

Select decode A1 This determines if the A1 sub address signal is used and what level is expected.
'X' signal not used, '0' signal needs to be low, '1' signal needs to be high.

Select decode A2 This determines if the A2 sub address signal is used and what level is expected.
'X' signal not used, '0' signal needs to be low, '1' signal needs to be high.

Select decode A3 This determines if the A3 sub address signal is used and what level is expected.
'X' signal not used, '0' signal needs to be low, '1' signal needs to be high.

The equivalent for single command action to change the values is:

sudo PixieBoard <boardId> ce0=a a0=b a1=b a2=b a3=b For CE0
sudo PixieBoard <boardId> ce1=a a0=b a1=b a2=b a3=b For CE1

Where 'a' is the same as the value entered in '**Select CEx source for CEx'** above.
Where 'b' is the same as the value entered in '**Select decode Ax'** above.

4.11.3Edit board to Raspberry Pi interrupt assignments

By default, if a board can generate an interrupt to the Raspberry Pi, it will normally be disabled as this is usually an optional requirement by the program.
The board can assign its interrupt outputs to a GPIO pin which is setup by the program to service the interrupt whenever it is active, the interrupt handler is setup using normal Linux interrupt handling techniques.
Each board may have up to two device sourced interrupts IRQ0 and IRQ1 which can either route to separate GPIO pins or be combined on the board into just one GPIO pin, the interrupt handler then determines which device on the board is requiring service.
The interrupt output can be set to either active low or high when used in bipolar mode, or active low when used in open drain wire OR'd mode.
When configured as open drain it automatically enables the interrupt output pullup on the GPIO pin, so no additional resistor is required. This mode also allows other boards configured in the same way to use the same GPIO pin as a common interrupt, it is then up to the software to determine the interrupting board.

To edit the interrupt outputs and levels use the **(I)-Interrupts**.

```
Select IRQ to edit: (0) >
Select GPIO signal for IRQ0: (X) >
Select Polarity  : (0) >
Select Open Drain: (0) >

 [** PIXIE BOARD 0 **] - Set values
```

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

Select IRQ to edit This will only be shown if the board can source more than 1 interrupt.
A value of 0 or 1 sets the board interrupt signal to change.

Select GPIO signal for IRQ0
Select GPIO signal for IRQ1
Only one of these messages are shown depending on the board interrupt entered at the previous prompt.
A value of 'X' signals its disabled or a GPIO pin to output the interrupt on.
GPIO pins 5, 6, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 can be used provided these pins are not being used for another function.

Select polarity This sets the active level of the interrupt output from the board.
A value of 0 = active low, or 1 = active high.

Select Open Drain This sets the output type of the interrupt output from the board.
A value of 0 = bipolar, or 1 = open drain.

The equivalent for single command action to change the values is:

sudo PixieBoard <boardId> irq0=a irqpol=b irqoc=c	For IRQ0
sudo PixieBoard <boardId> irq1=a irqpol=b irqoc=c	For IRQ1

Where 'a' is the same as the value entered in '**Select GPIO signal for IRQx'** above.
Where 'b' is the same as the value entered in '**Select polarity'** above.
Where 'c' is the same as the value entered in '**Select Open Drain'** above.

4.11.4Board defaults

Board defaults can be applied to the selected board using the **(D)-Apply defaults**. This will apply basic defaults to the board, the SPI bus is set to SPI0, the chip selects are set to CE0 (GPIO8) and CE1 (GPIO7), the sub addresses are disabled and so are any interrupts.

The equivalent for single command action is:

sudo PixieBoard <boardId> -d

4.11.5Verify overall configuration

The overall configuration of the fitted boards is checked using the **(V)-Verify boards**. This scans all the boards in the system and checks and report duplicate addressing, chip select usage and interrupts.

```
Scanning...

Checking chip selects...
Board[0]-CE0 address CLASH with Board[8]-CE0

Checking interrupts...
Board[0]-IRQ0 address CLASH with Board[8]-IRQ0
```

The output above shows board 0 CE0 is decoded the same as board 8 CE0, it also shows the interrupt outputs are the same GPIO pins as well, below shows there are no clashes.

```
Scanning...

Checking chip selects...

Checking interrupts...
```

For now, exit the **PixieBoard** application back to the command line prompt.

The equivalent for single command action is:

sudo PixieBoard -v

5. Firmware updates

All **PIXIE** boards are supplied with the latest firmware but there may be the occasion where new features are added to a board or the firmware just needs to be re-installed which is very unlikely, to allow for this the boards contain a reload and verify feature which will allow this to be carried out.

5.1 Set the I2C bus speed

Due to the lack of I2C clock stretching support it is necessary to lower the I2C bus speed below 100kHz otherwise you may get firmware update verification errors, to do this make the following change:

cd /boot
sudo nano config.txt

Find the entry **dtparam=i2c_arm=on**
then append a comma and the following to the line, **i2c_arm_baudrate=90000**

If should now say: **dtparam=i2c_arm=on, i2c_arm_baudrate=90000**

Save the file and exit.
Reboot the Pi.

5.2 Firmware update

Download the current firmware hex files **pixie-firmwares-x.y.tgz** found at
aelmicro.com > Support > Pixie Boards > Software > Pixie board firmware

Extract the firmware files.

cd ~
tar xzf ~/Downloads/pixie-firmware-x.y.tgz

Run the **PixieBoard** application.

Select **(F)-Firmware**, from the main menu and the menu changes to:

```
PIXIE BOARD[0] - FIRMWARE MENU :
(?)-Menu help, (B)-Board ID, (I)-Invalidate, (U)-Update, (V)-Verify, (X)-Exit...
```

There are primarily 3 options, Invalidate, Update and Verify.

The **(I)-Invalidate** is used to invalidate the current PIC application and return to the boot mode.

The **(U)-Update** is used to update the current PIC application run this as normal.

The **(V)-Verify** is used to verify the current PIC application with its hex file.

Note: If a board has a main and slave PIC fitted which is the case for the PSU-POE power PIXIE board then additional menu options are shown to allow these additional devices to be programmed.

5.2.1 Invalidate

Select the board you want to invalidate using **(B)-Board ID**

Select **(I)-Invalidate** to invalidate the current PIC application and return to the boot mode which will be signalled by the Red LED flashing on for 100mS every 1 second.

Once invalidated an update is required to reinstate normal operation.

sudo PixieBoard <boardId> -fim	Invalidate the configuration PIC firmware.
sudo PixieBoard <boardId> -fis	Invalidate the slave PIC firmware if board has one.

5.2.2 Update

Select **(U)—Update** and the firmware will be updated and verified as shown from the firmware file.

```
[** PIXIE BOARD 0 **]
Serial Number     : "12345"
Board Revision    : "1"
Board Type        : "ADC8-DAC2"
Board Mode        : "Application"
Software Version  : "1.0"
Software Date     : "20 Aug 2020"

[Pixie Board Application, Version "1.0", "20 Aug 2020"]
    Bytes written : 11995
    Bytes read    : 11995
    Bytes verified: 11995
```

sudo PixieBoard <boardId> -fum	Update the configuration PIC firmware. Default file is ConfigPic.hex Append =<Filename> to the -fum switch to change the file. e.g. **-fum=DifferentFilename.hex**
sudo PixieBoard <boardId> -fus	Update the slave PIC firmware if board has one. Default file is board dependent Append =<Filename> to the -fus switch to change the file. e.g. **-fus=DifferentSlaveFilename.hex**

5.2.3 Verify

Select **(V)—Verify** and the firmware will be verified as shown against the firmware file.

```
[** PIXIE BOARD 0 **]
Serial Number     : "12345"
Board Revision    : "1"
Board Type        : "ADC8-DAC2"
Board Mode        : "Application"
Software Version  : "1.0"
Software Date     : "20 Aug 2020"

[Pixie Board Application, Version "1.0", "20 Aug 2020"]
    Bytes read    : 11995
    Bytes verified: 11995
```

sudo PixieBoard <boardId> -fvm	Update the configuration PIC firmware. Default file is ConfigPic.hex Append =<Filename> to the -fvm switch to change the file. e.g. **-fum=DifferentFilename.hex**
sudo PixieBoard <boardId> -fvs	Update the slave PIC firmware if board has one. Default file is board dependent Append =<Filename> to the -fvs switch to change the file. e.g. **-fus=DifferentSlaveFilename.hex**

5.2.4 Forced boot mode.

If there is a situation where a program update fails, and the board becomes unresponsive there is a hardware method that can be followed to force the board into it boot mode state.

1. Remove the board so it is not connected to the Raspberry Pi.
2. Make the following connections to the 40way connector.
3. Using a suitable +5V power supply connect the 0V to pin 6, and +5V to pin 4.
4. Connect pin 14 to pin 17.
5. Connect pin 3 to pin 5.
6. Power on the board and the Red LED will start flashing on for 100mS every 1 second which indicates the board has been invalidated and is now running in boot mode.
7. Power off, remove the connections, refit the board, and update the firmware as described.

6. Linux changes

This is a useful procedure to rebuild the Raspberry Pi kernel if you wish to edit the device tree overlays used on the Pixie boards, this is just one method but there are many others posted on the Raspberry Pi and other websites.

6.1 Getting the kernel

Setup a virtual machine on your PC and load your favourite Linux distribution, the authors is **Kubuntu.**

<user> is your username in the **/home** directory.

1) In **/home/<user>** create a directory "**raspberrypi**".

2) Register an account on github.com if you do not already have one as you will need those details to get the source code and tools.

3) Add a file in the **/home/<user>** directory called **.netrc** then add the following to it

   machine github.com
   username YOUR USERNAME
   password YOUR PASSWORD

   e.g.
   machine github.com
   username bob
   password abc1234

   Save file.

4) Add to the **/home/<user>/.bashrc** file

   export CC32PREFIX=arm-linux-gnueabihf-
   export CC64PREFIX=aarch64-linux-gnu-

5) Update the shell environment, at the prompt enter:
   source ~/.bashrc

6) Install support tools.
   sudo apt install git bc bison flex libssl-dev make libc6-dev libncurses5-dev

   For 32 bit
   sudo apt install crossbuild-essential-armhf

   For 64 bit
   sudo apt install crossbuild-essential-arm64

7) **cd** into the **raspberrypi** directory

8) Get the latest mainline source code.
git clone –depth=1 https://github.com/raspberrypi/linux

For the real time patched version
git clone https://github.com/raspberrypi/linux **-b rpi-4.19.y-rt** or latest

9) Get the **.config** file from the raspberry Pi or use the default **.config** file (**recommended**)

cd linux directory

scp pi@raspberrypi:/proc/config.gz .

If it does not exist on the Pi do
sudo modprobe configs
Then try again!

You can use the default (recommended) **.config** by running,
For pi 2, 3, 3+:
KERNEL=kernel7
make ARCH=arm CROSS_COMPILE=${CC32PREFIX} bcm2709_defconfig
For pi 4 32 bit:
KERNEL=kernel7l
make ARCH=arm CROSS_COMPILE=${CC32PREFIX} bcm2711_defconfig
For pi 4 64 bit:
KERNEL=kernel8
make ARCH=arm64 CROSS_COMPILE=${CC64PREFIX} bcm2711_defconfig

To create a **DEBUG** build:
grep -v DEBUG_INFO < .config > newconfig
mv newconfig .config ARCH=arm
CROSS_COMPILE=${CC32PREFIX} make oldconfig
make oldconfig
Set DEBUG_INFO=y
Set DEBUG_INFO_REDUCED=n
Enter <return> for any other questions.

6.2 Additional device tree objects

This describes the files and changes needed to implement the sub addressing support for the SPI driver and other drivers used by the **PIXIE** boards.

3) Download the device tree source files **pixie-overlays-source-x.y.tgz** found at
aelmicro.com > Support > Pixie Boards > Software > Pixie linux overlays source

4) Extract the source files:
cd /home/<user>/raspberrypi/linux/arch/arm/boot/dts/overlays
tar xzf ~/Downloads/pixie-overlays-source-x.y.tgz

The following files are extracted:
spi0-hw-sa4-overlay.dts, creates **spidev0.0** thru **spidev0.3**, uses 1 sub address and both CS0 & 1.
spi0-hw-sa8-overlay.dts, creates **spidev0.0** thru **spidev0.7**, uses 2 sub address and both CS0 & 1.
spi0-hw-sa16-overlay.dts, creates **spidev0.0** thru **spidev0.15**, uses 3 sub address and both CS0 & 1.
spi0-hw-sa32-overlay.dts, creates **spidev0.0** thru **spidev0.31**, uses 4 sub address and both CS0 & 1.
spi1-hw-sa6-overlay.dts, creates **spidev1.0** thru **spidev1.5**, uses 1 sub address and CS0, 1 & 2.
spi1-hw-sa12-overlay.dts, creates **spidev1.0** thru **spidev1.11**, uses 2 sub address and CS0, 1 & 2.
spi1-hw-sa24-overlay.dts, creates **spidev1.0** thru **spidev1.23**, uses 3 sub address and CS0, 1 & 2.
mcp251xfd-overlay.dts, this is the can0 for the mc2517fd device using the SPI0 or SPI1 bus.

5) Update **overlays/makefile** in overlays to include all the files shown above by adding to the end of the **dtbo-$(CONFIG_ARCH_BCM2835)** += list of objects to make.

wittypi.dtbo \
wm8960_soundcard \
**spi0-hw-sa4.dtbo **
**spi0-hw-sa8.dtbo **
**spi0-hw-sa16.dtbo **
**spi0-hw-sa32.dtbo **
**spi1-hw-sa6.dtbo **
**spi1-hw-sa12.dtbo **
**spi1-hw-sa24.dtbo **

6.3 Building the kernel, modules, and device tree objects

1) Build the kernel which are all 32bit versions
   For pi 2, 3, 3+:
   KERNEL=kernel7
   For pi 4: 32 bit
   KERNEL=kernel7l 32 bit

   cd into the **raspberrypi/linux** directory
   make clean
   make ARCH=arm CROSS_COMPILE=${CC32PREFIX} zImage modules dtbs

2) For pi 4: 64 bit
   KERNEL=kernel8 64 bit

   cd into the **raspberrypi/linux** directory
   make clean
   make ARCH=arm64 CROSS_COMPILE=${CC64PREFIX} Image modules dtbs

3) Create the modules
   For pi 4: 32 bit
   ARCH=arm CROSS_COMPILE=${CC32PREFIX} INSTALL_MOD_PATH=../modules make modules_install

   For pi 4: 64 bit
   ARCH=arm64 CROSS_COMPILE=${CC64PREFIX} INSTALL_MOD_PATH=../modules make modules_install

   Resultant modules found in **raspberrypi/modules/lib/modules/**

4) Create the kernel image
   For pi 4: 32 bit
   sudo cp arch/arm/boot/zImage $KERNEL.img
   For pi 4: 64 bit
   sudo cp arch/arm64/boot/Image $KERNEL.img

5) Copy the new image to the Pi
   Ensure the Pi is powered up and on the network.
   scp $KERNEL.img pi@raspberrypi:/tmp

6) Compress the modules
   cd ../modules/lib/modules
   tar czf modules.tgz *

7) Copy the modules to the Pi
   scp modules.tgz pi@raspberrypi:/tmp

8) Copy the device tree files to the Pi

cd ../../../linux/arch/arm/boot/dts
scp *.dtb pi@raspberrypi:/tmp
cd overlays
scp *.dtb* pi@raspberrypi:/tmp

9) Now go to the Pi and open a terminal window.
For pi 2, 3, 3+:

KERNEL=kernel7

For pi 4 32bit:

KERNEL=kernel7l

For pi 4 64bit:

KERNEL=kernel8

cd /
sudo mv /tmp/$KERNEL.img /boot/ *May get ownership message which can be ignored
sudo cp /tmp/*.dtb /boot/
sudo cp /tmp/*.dtb* /boot/overlays
cd /lib/modules
sudo tar xzf /tmp/modules.tgz

10) Reboot the Pi

sudo reboot

11) Check latest kernel is loaded using

uname –a
and check the time and date it was built.

Congratulations you have now successfully rebuilt and installed the Raspberry Pi kernel.

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