I²C Extension Kit Documentation

Windows Embedded Compact

Version 1.02
(2014-10-01)

© F&S Elektronik Systeme GmbH
Untere Waldplätze 23
D-70569 Stuttgart
Fon: +49(0)711-123722-0
Fax: +49(0)711 – 123722-99

History

Date	V	Platform	A,M,R	Chapter	Description	Au
25.05.09	1.0		A, M	*	First version of this documentation	PM
21.05.10	1.1		A, M	*	A lot of adaptions and corrections. New board layout.	MK
31.10.14	1.2		A, M	*	Add information and pictures for efus	MW

V	Version
A,M,R	Added, Modified, Removed
Au	Author

About this document

This documentation is about the hardware and the drivers of the I²C extension kit. It shows how to connect the board, install the drivers and use them in own software applications. The board and drivers are available for most of the boards from F&S for Windows Embedded CE/Compact. The latest version of this document can be found at: http://www.fs-net.de

Table of Contents

1 Introduction

In some applications the number of interfaces and I/O pins natively available is not sufficient. But by using a standard bus interface this connectivity can be extended very easily. For this need F&S has designed an extension-kit that can by connected directly to the corresponding starter kit base board. It uses the I²C bus interface that is available on all modules.

The extension board offers additional connectivity for

- I/O pins
- A/D inputs
- PWM signals

To control these interfaces, this extension kit includes a set of libraries and drivers to access the devices very easily within your application. **The driver interfaces comply with the interfaces already available for native drivers**. So for example accessing an I/O on the extension board works similar to the mechanism you access a digital I/O (DIO) on your F&S embedded board (armStone, efus, NetDCU, PicoMOD, PicoCOM).

Beside the extension adapter board, all schematics are also available, so that parts of these interfaces can be integrated on your baseboard directly.

1.1 Known issues

- Interrupt support is missing in the ext_IO and the ext_keyboard driver.
- PWM driver not available yet.
- PicoCOM2: If there occur some data errors on the PicOCOM2 with the native I²C driver, please try to using the software I²C driver that is available since kernel version V1.13.

2 Hardware

2.1 I²C extension board

The extension board is designed to be used with any F&S board available. But as there are still some basic differences, the board to use the adapter with, must be selected with the solder bridges JP1-JP12. Additionally have to connect an external 5.0V supply voltage.

For providing the additional interfaces on the extension board, following micro-controllers are used:

- I/O chip: PCA9555 from Philips
- A/D chip ADS7828 from Texas Instruments
- PWM chip:PCA9533 from Philips

Following figure shows the I²C extension board with the I²C connector (J1), the Extension Connector (J2), the external power connector (J3) and the 2 dip switches (S1, S2).

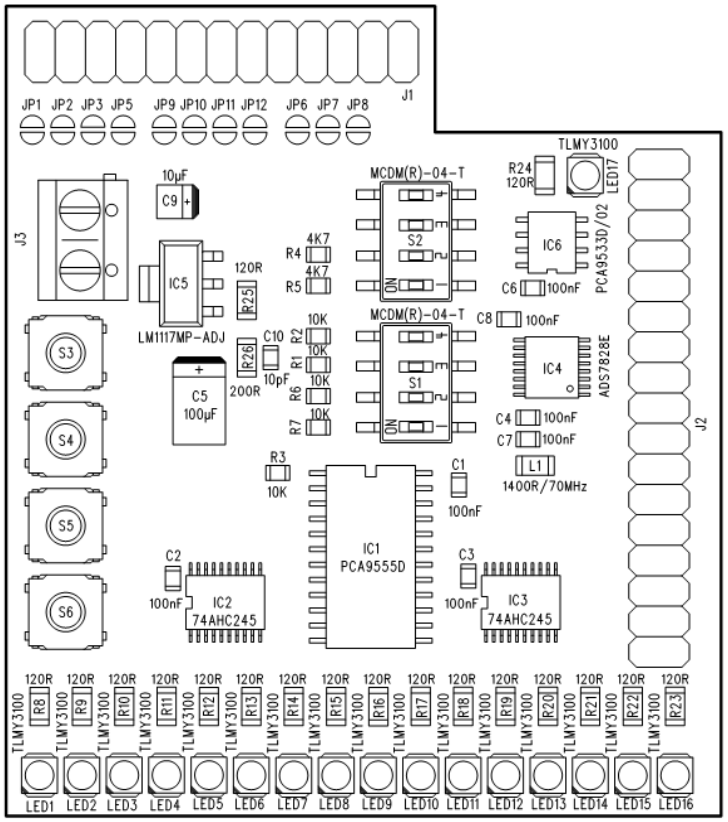


Figure 1: I²C extension board

2.1.1 I2C Slave Address of Devices

Chip	Function	Slave Address
PCA9555	I/O chip	0 1 0 0 0 A1 A0 R/W 0x20 – 0x23
ADS7828	A/D chip	1 0 0 1 0 A1 A0 R/W 0x48 - 0x4B
PCA9533	PWM chip	PCA9553/01 1 1 0 0 0 1 0 R/W 0x62 PCA9553/02 1 1 0 0 0 1 1 R/W 0x63

Table 1: I2C Slave Addresses

2.1.2 DIP switches

The dip switches (S1 and S2) are used to configure the I²C address of the used controller chips. Additionally some board specific configuration must be arranged to get the board working properly. E.g. the NetDCU boards are not equipped with pull-ups on the I²C bus natively. The following table shows functions of the two dip switches:

Dip switch 1:		
1	Address bit A1 of ADS7828 A/D chip. ON: A1 = 0 OFF: A1 = 1	ADS7828 address: 1 0 0 1 0 A1 A0 R/W
2	Address bit A0 of ADS7828 A/D chip. ON: A0 = 0 OFF: A0 = 1	
3	Address bit A0 of PCA9555 I/O chip. ON: A0 = 0 OFF: A0 = 1	PCA9555 address: 0 1 0 0 0 A1 A0 R/W
4	Address bit A1 of PCA9555 I/O chip. ON: A1 = 0 OFF: A1 = 1	

<table>
<tr><th colspan="3">Dip switch 2:</th></tr>
<tr><td>1</td><td colspan="2">I²C pull-up for SDA.
ON: Pull-up enabled
OFF: Pull-up disabled</td></tr>
<tr><td>2</td><td colspan="2">I²C pull-up for SCL.
ON: Pull-up enabled
OFF: Pull-up disabled</td></tr>
<tr><td>3</td><td>VDD used for A/D reference voltage.</td><td rowspan="2"><table>
<tr><th>S2-3</th><th>S2-4</th><th>VREF</th></tr>
<tr><td>OFF</td><td>ON</td><td>VREF-EXT</td></tr>
<tr><td>OFF</td><td>OFF</td><td>invalid</td></tr>
<tr><td>ON</td><td>ON</td><td>invalid</td></tr>
<tr><td>ON</td><td>OFF</td><td>VDD</td></tr>
</table></td></tr>
<tr><td>4</td><td>VREF-EXT used for A/D reference voltage.</td></tr>
</table>

Table 2: DIP switch configuration

2.2 Pin Assignment

2.2.1 Extension Connector (J2)

The next table shows how the pins of the extension connector (J2)

Pin	Function	Pin	Function
1	IO0 *(port0-0)*	2	IO1 *(port0-1)*
3	IO2 *(port0-2)*	4	IO3 *(port0-3)*
5	IO4 *(port0-4)*	6	IO5 *(port0-5)*
7	IO6 *(port0-6)*	8	IO7 *(port0-7)*
9	IO8 *(port1-0)*	10	IO9 *(port1-1)*
11	IO10 *(port1-2)*	12	IO11 *(port1-3)*
13	IO12 *(port1-4)*	14	IO13 *(port1-5)*
15	IO14 *(port1-6)*	16	IO15 *(port1-7)*
17	A/D channel 0	18	A/D channel 1
19	A/D channel 2	20	A/D channel 3

21	A/D channel 4	22	A/D channel 5
23	A/D channel 6	24	A/D channel 7
25	A/D COM	26	A/D VREF
27	PWM0	28	PWM1
29	PWM2	30	PWM3
31	V33	32	V33
33	GND	34	GND

Table 3: Extension Connector J2

Note:

The four push-buttons (S3-S7) are connected to the I/O pins 0..3. **Be careful to configure them as input before using the buttons.**

By default all pins are configured as Input with a internal pull-up (100k) enabled.

2.2.2 I²C Connector (J1)

Depending of the solder bridges JP1-JP12 the pins available on the main connector are routed on the extension board.

Before connecting the extension board please make sure that the jumpers are configured correctly for your board.

Jumper	Connector layout
JP1-JP5	armStone/ NetDCU/PicoMOD family
JP6-JP8	PicoCOM1
JP9-JP12	PicoCOM2/ efus

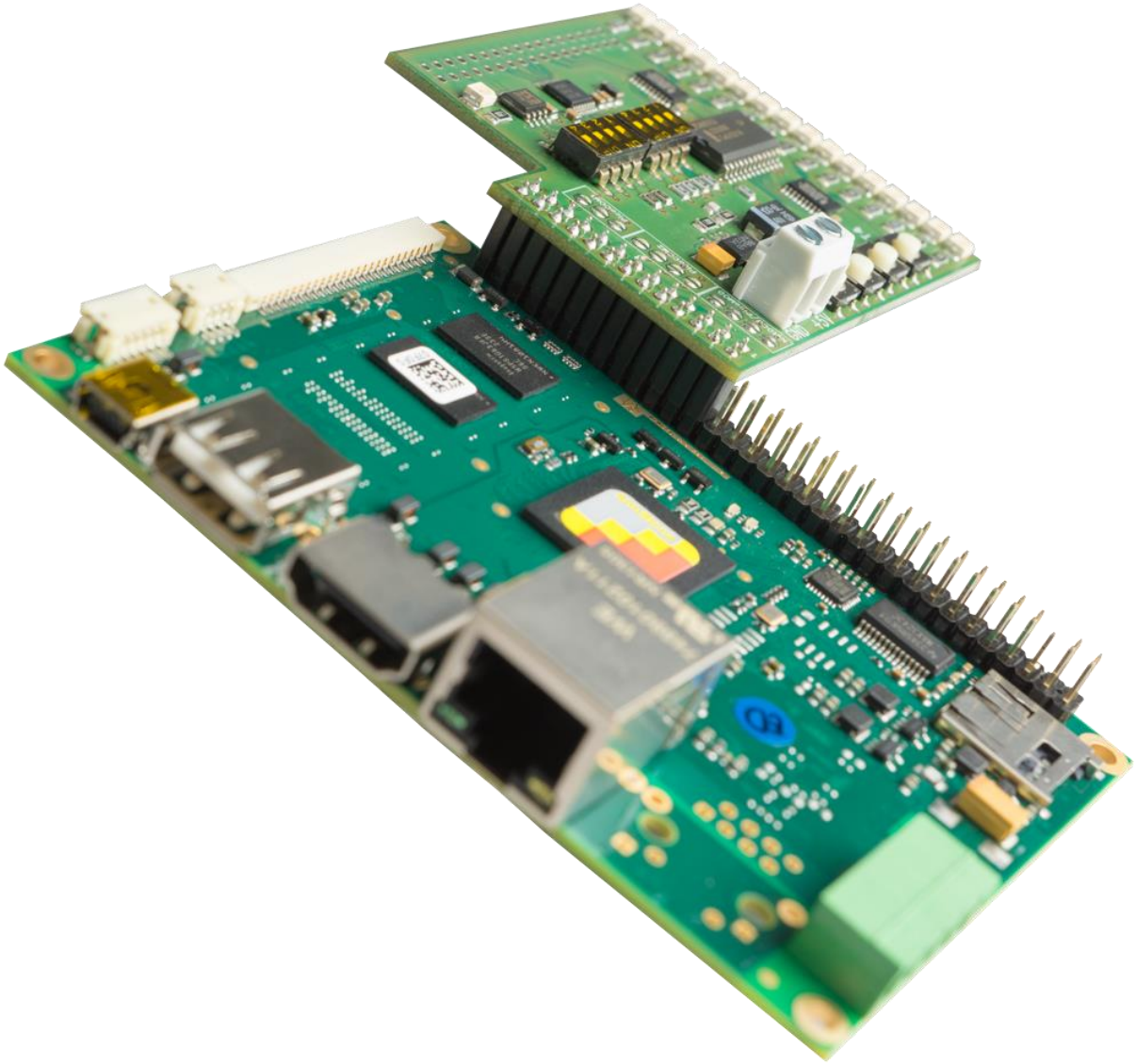
Note:

It is not possible to combine several layouts configurations at the same time.

Following tables shows which pins of connector are used on the extension board according to the current corresponding configuration.

armStone family – feature connector

25	23	21	19	17	15	13	11	9	7	5	3	1
---	---	---	---	I²C SDA	---	---	GND	---	---	---	---	---
26	**24**	**22**	**20**	**18**	**16**	**14**	**12**	**10**	**8**	**6**	**4**	**2**
---	---	---	---	IRQ	I²C SCL	---	---	---	---	---	---	---

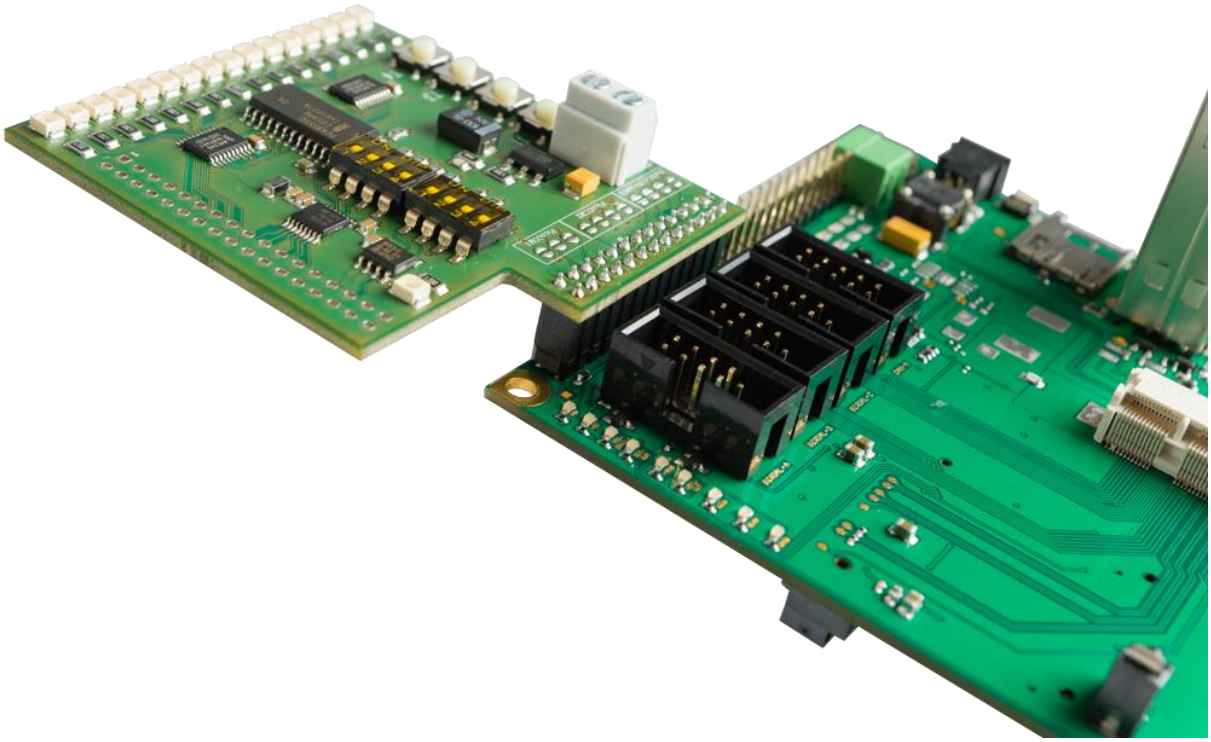


EFUS family

1	3	5	7	9	11	13	15	17	19	21	23	25
---	---	---	---	I²C DAT		---	---	---	---	---	---	---
2	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**	**22**	**24**	**26**
---	---	---	---	I²C SCL	I²C IRQ	---	---	GND	---	---	---	---

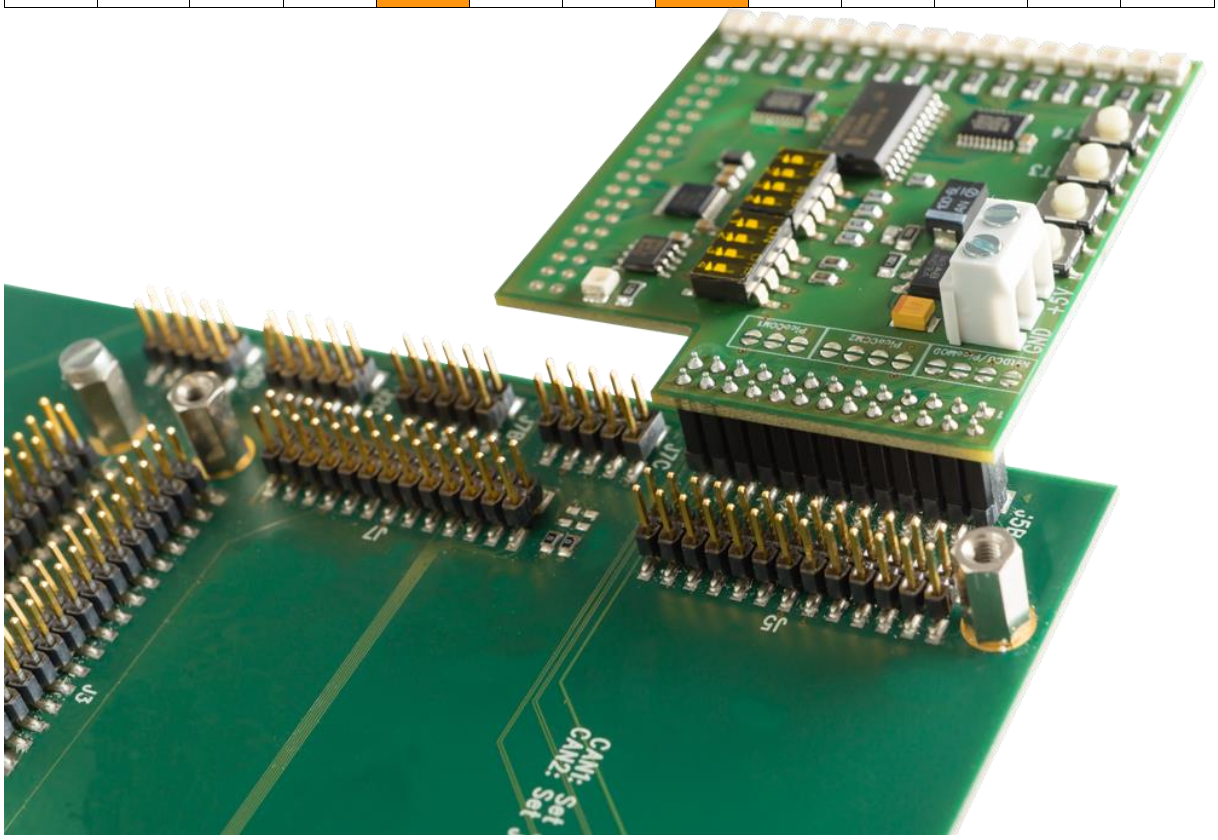
Note:

Pin 1 of the I²C-Extension Board has to be connected to Pin 33 on the efus-SINTF-Feature connector (J22).



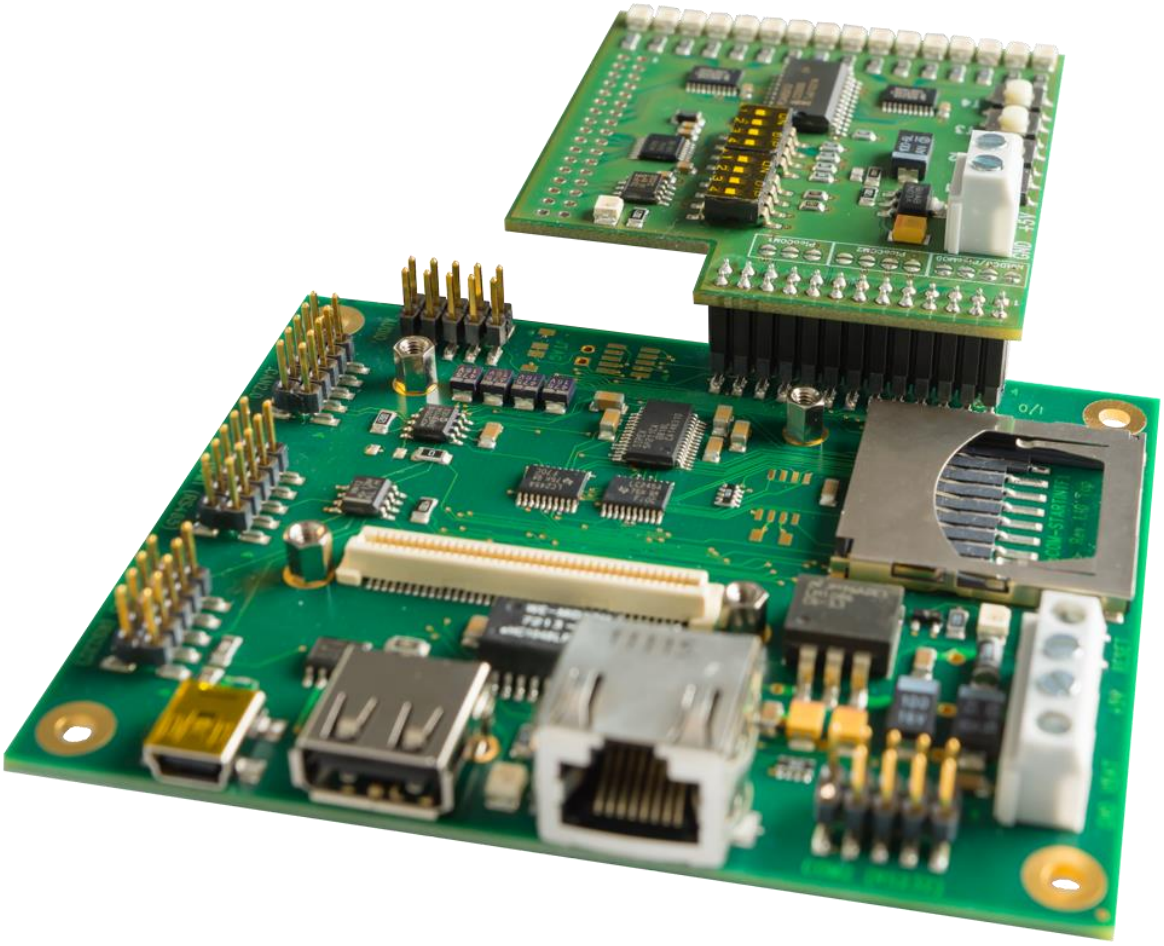
NetDCU / PicoMOD family – connector J5

1	3	5	7	9	11	13	15	17	19	21	23	25
---	---	---	---	IRQ	I²C SCL	---	---	---	---	---	---	---
2	4	6	8	10	12	14	16	18	20	22	24	26
---	---	---	---	I²C SDA	---	---	GND	---	---	---	---	---



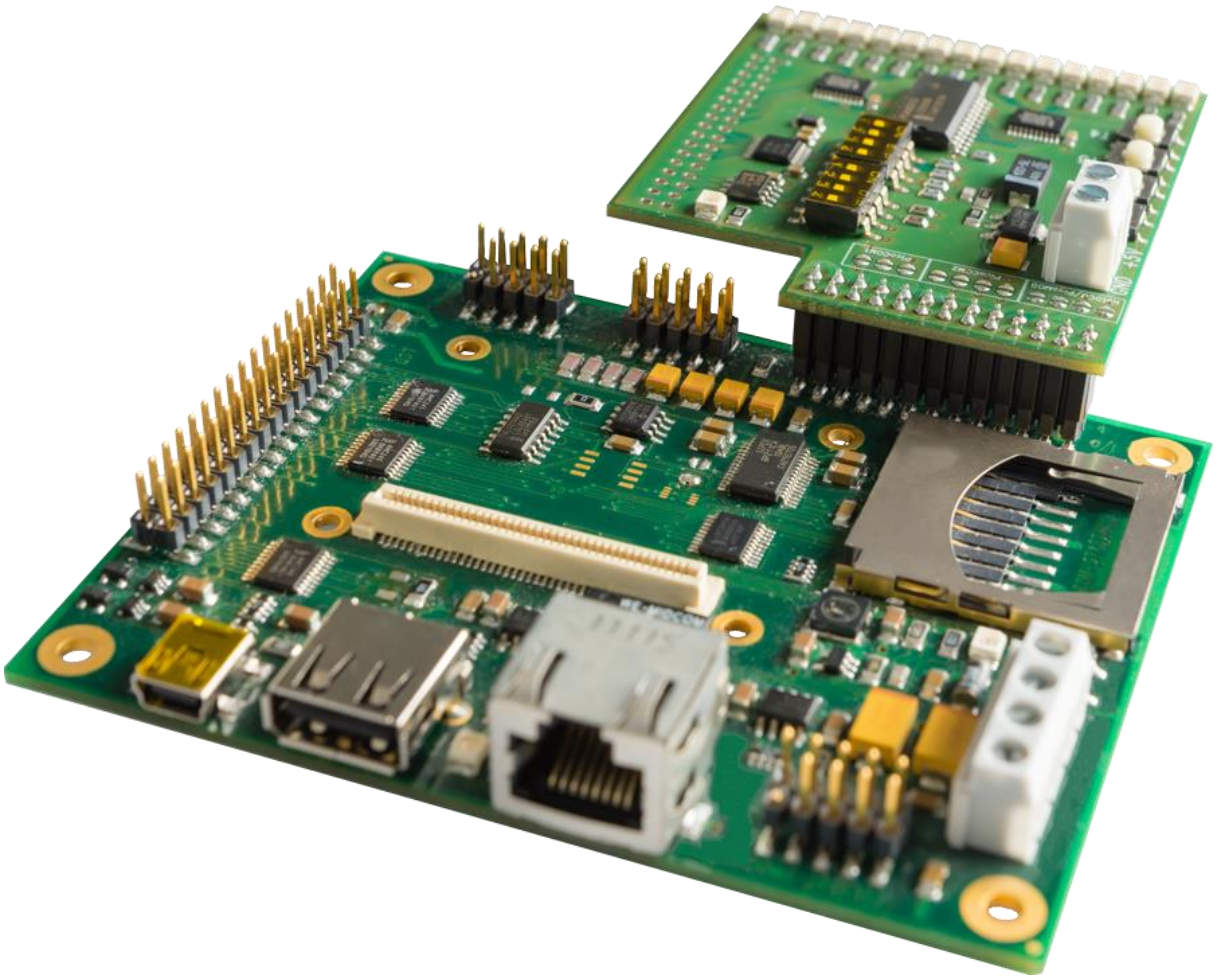
PicoCOM1

1	3	5	7	9	11	13	15	17	19	21	23	25
---	---	---	---	IRQ	---	---	---	---	---	I²C SDA	---	---
2	4	6	8	10	12	14	16	18	20	22	24	26
---	---	---	---	---	---	---	---	---	---	I²C SCL	---	---



PicoCOM2 / PicoCOM4 / PicoCOMA5

1	3	5	7	9	11	13	15	17	19	21	23	25
---	---	---	---	I²C SDA	---	---	---	---	---	---	---	---
2	4	6	8	10	12	14	16	18	20	22	24	26
---	---	---	---	I²C SCL	IRQ	---	---	GND	---	---	---	---



Software Drivers

There are different possibilities to work with the extension board. First of all you can access the board by the I2C driver which is already installed on your F&S board. To inspect the I2C devices at the extension board you can use the tool FS_I2CSCAN.EXE.

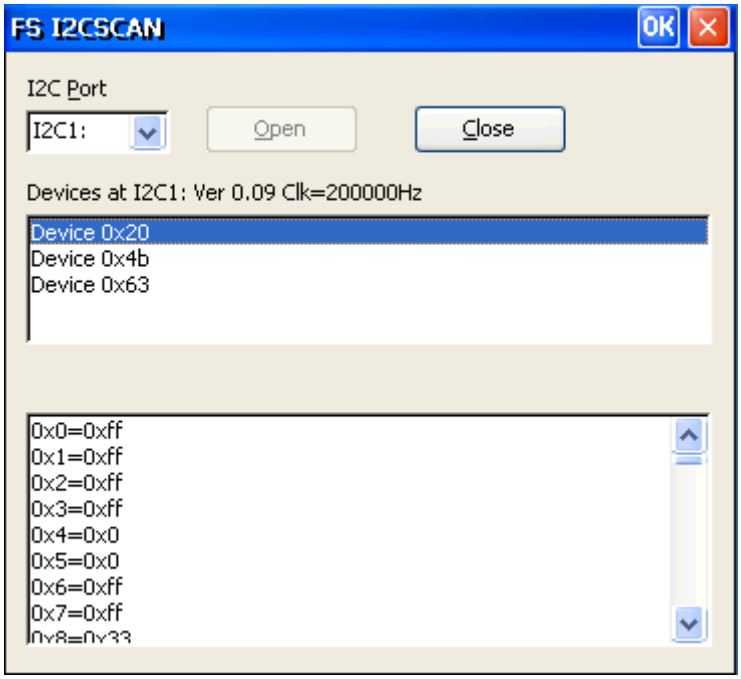


Figure 2: Tool FS_I2CSCAN.EXE

The extension board driver offers the same software interface as the drivers available for your single board computer, so you don't have to rewrite your application if you want to use the external pins. All the functions of the drivers have been transmitted. The drivers **only** use the platform specific DIO driver and (N)I2C driver of your module to access the extension board. This offers the possibility to run the same driver on nearly every board of the F&S board family.

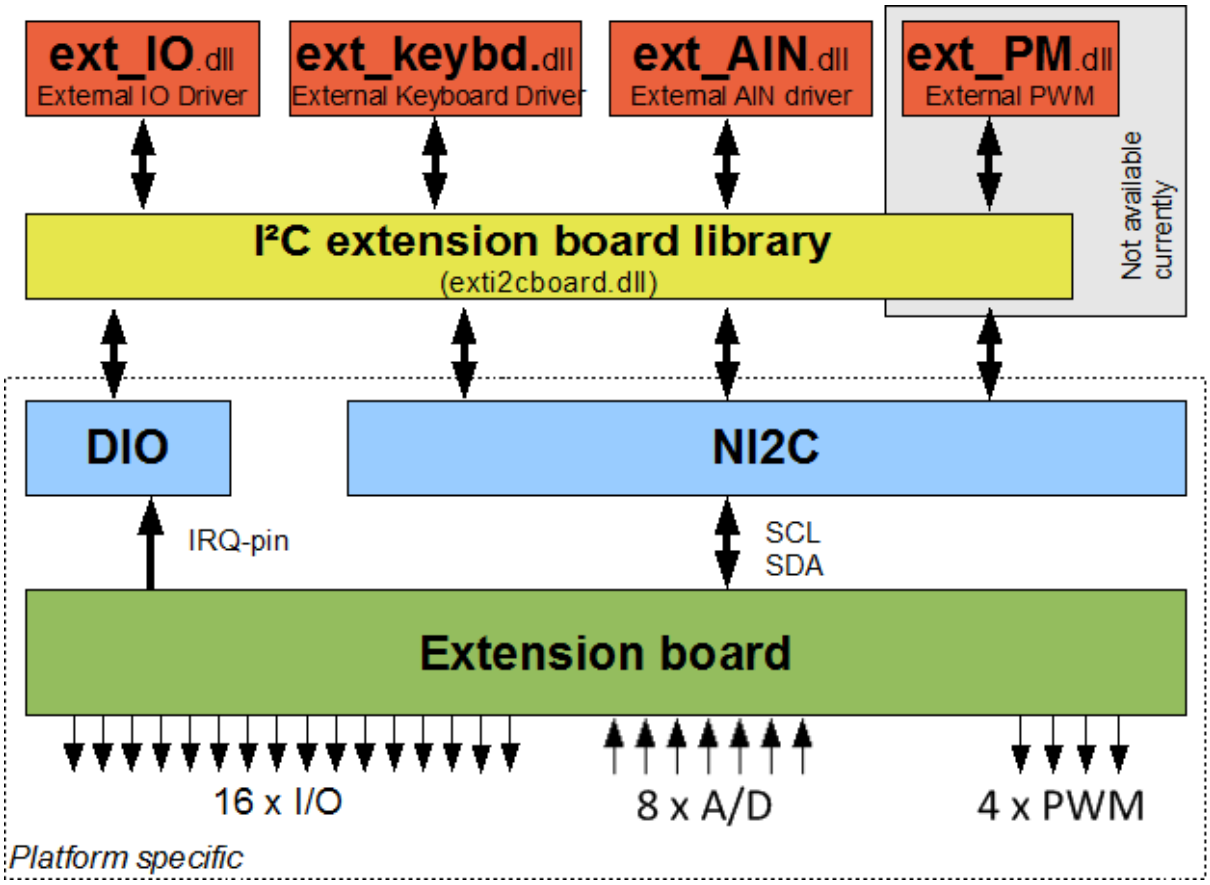


Figure 3: Connection between the drivers and the board

2.3 Concurrent Configuration

The `exti2cboard.dll` library handles access to the extension board and also coordinates concurrent operations. Hence this enables simultaneously usage of the available interfaces. For example it is possible to use some I/O-pins with the ext_IO driver and use some other pins for a matrix keyboard (handled by ext_keybd driver) at the same time.

2.4 Configuration Requirements

Before you start installing the drivers you have to do a few preparations:

1 Plug the extension board with the right connector to your F&S embedded board.

2 If you use NetDCU you have to turn on the I²C pull-up resistors on the dip switch (see table 1).

3 Connect the external power supply (5V) and GND to the power connector on the top of the board.

4 The (N)I2C and the DIGITALIO (DIO) driver must be installed and enabled on your board.

3 The ext_IO Driver

3.1 Installation

The libraries `ext_io.dll` and `exti2cboard.dll` have to be stored in flash memory into directory `\FFSDISK` if it is not pre-loaded in the kernel already.

Note:

The interrupt functionality is not available yet.

3.2 Configuration

Additionally the ext_IO driver requires setting some registry values. Installation of the ext_IO driver takes place in the registry under

`[HKLM\Drivers\BuiltIn\EIO1]`

Entry	Type	Value	Description
`Dll`	String	`ext_io.dll`	Driver DLL
`FriendlyName`	String	`I2C IO driver`	Description
`Prefix`	String	`EIO`	For EIO<index>:
`Index`	DWORD	`1`	For EIO1:
`Order`	DWORD	`301`	Load sequence
`I2cDevAddr`	DWORD	`0x46`	I2C Address of the I/O chip (PCA9555)
`I2CDevName`	String	`I2C1:`	I2C device used to access the extension board.
`Debug`	DWORD	`0`	Debug verbosity
`Port`	DWORD	`0 or 1`	
`DataDir`	HEX	`00,00`	Data Direction. 0 = The corresponding pin is an input. 1 = The corresponding pin is an output. One bit for each I/O pin.
`DataInit`	HEX	`00,00`	Default value of the output pin after driver initialization.
`IRQCfg0`	HEX	`00,00`	Interrupt configuration register 0.
`IRQCfg1`	HEX	`00,00`	Interrupt configuration register 1.
`IRQCfg2`	HEX	`00,00`	Interrupt configuration register 2.

Table 4: ext_IO Registry Values

Most of the values will get meaningful defaults if omitted, only those values highlighted in blue/grey and italics above in the first few rows really have to be given.

Debug

If the `Debug` entry is set to a value different to zero, the driver will output additional information on the debug port. Each bit enables a different category of output. This information is usually not required and only necessary when looking for errors in the driver. Keep this value at zero to have the best possible performance.

Port

Set the default value of port. If you use `WriteFile()` or `ReadFile()` you will write or read from this port by default. You may adjust this port with the `SetFilePointer()` function.

DataDir

Every bit stands for one pin. The first hex byte corresponds to port0 (IO0 to IO7) the second hex byte defines configuration for port1 (IO8 to IO15).

DataInit

Default value of the output pin after driver initialization.

IRQCfg

You can set the interrupt configuration for every pin

IRQCfg2	IRQCfg1	IRQCfg0	Function
0	0	0	Interrupt Disabled
0	0	1	Rising Edge Enabled
0	1	0	Falling Edge Enabled
0	1	1	Rising and Falling Edge Enabled
1	0	0	Interrupts Disabled
1	0	1	High Level Enabled
1	1	0	Low Level Enabled

Table 5: Interrupt Configuration

3.3 Usage in applications

With the ext_IO driver you can write and read the 2 ports on the I²C extension board.

Note:

As the driver interface of the ext_IO driver is identical to the “regular” DIO driver interface, you may still use the `dio_sdk.h` header file that is included in the SDK of your board.

Therefore the samples available for the DIO driver can be used for the ext_IO driver, too. Just make sure that the according digital I/O interface will be opened (`EIO1:` instead of `DIO1:`)

3.4 ext_IO Reference

3.4.1 CreateFile()

Signature:

```
HANDLE CreateFile(
      LPCTSTR lpFileName, DWORD dwAccess, DWORD dwShareMode,
      LPSECURITY_ATTRIBUTES lpSecurity, DWORD dwCreation,
      DWORD dwFlags, HANDLE hTemplate
);
```

Parameters:

`lpFileName`	Device file name, usually "`EIO1:`"
`dwAccess`	Device access (see below)
`dwShareMode`	File share mode (see below)
`lpSecurity`	Ignored, set to `NULL`
`dwCreation`	Set to `OPEN_EXISTING`
`dwFlags`	Set to `FILE_ATTRIBUTE_NORMAL`
`hTemplate`	Ignored, set to `0`

Device access `dwAccess`:

`0`	Device query mode
`GENERIC_READ`	Open device file read-only (receive)
`GENERIC_WRITE`	Open device file write-only (send)
`GENERIC_READ | GENERIC_WRITE`	Open device file in read-write mode

File share mode `dwShareMode`:

`FILE_SHARE_READ`	Subsequent open operations succeed only if read access
`FILE_SHARE_WRITE`	Subsequent open operations succeed only if write access

Return:

`INVALID_HANDLE_VALUE`	Failure, see `GetLastError()` for details
Otherwise	File handle

Description:

Opens the `EIOx:` device file for access. This is required for all other functions using this ext_IO driver.

If the file handle is not required any more, you have to call function `CloseHandle()`.

3.4.2 WriteFile()

Signature:

```
BOOL WriteFile(
      HANDLE hFileHandle, LPCVOID lpBuffer, DWORD dwLen,
      LPDWORD dwActuallySent, LPOVERLAPPED lpOverlapped
);
```

Parameters:

`hFileHandle`	Handle to device file
`lpBuffer`	Pointer to the buffer with data to send
`dwLen`	Number of bytes to send
`dwActuallySent`	Pointer to a `DWORD` where the number of actually sent bytes is returned
`lpOverlapped`	Ignored, set to `NULL`

Return:

`0`	Error, see `GetLastError()` for details
`!=0`	Success

Description:

Sends the `dwLen` bytes that are stored at `lpBuffer` to the actual port of your ext_IO device. It is possible to choose port0 and `dwLen=2`, than the first byte will be sent to `port0` and the second to `port1`. It's not possible to start with `port1` and go on to `port0`.

Example:

Set pin 0 and 2 and clear the rest on actual port

```
DWORD dwBytesWrite = 1;
BYTE data = 5;
WriteFile( hEIO, &data, dwBytesWrite, &dwBytesWrite, NULL );
if( dwBytesWrite != 1 )
{
      //ERROR
}
```

Listing 1: Example WriteFile()

3.4.3 ReadFile()

Signature:

```
BOOL ReadFile(
      HANDLE hFileHandle, LPCVOID lpBuffer, DWORD dwLen,
```

```
      LPDWORD dwRead, LPOVERLAPPED lpOverlapped
);
```

Parameters:

`hFileHandle`	Handle to device file
`lpBuffer`	Pointer to the buffer where the received data is stored
`dwLen`	Number of bytes to receive
`dwRead`	Pointer to a `DWORD` where the number of actually received bytes is returned
`lpOverlapped`	Ignored, set to `NULL`

Return:

`0`	Error, see `GetLastError()` for details
`!=0`	Success

Description:

Receives `dwLen` bytes from the ext_IO device and stores the data at `lpBuffer`. It is possible to choose port0 and `dwLen=2`, than the first byte will be read from `port0` and the second from `port1`. It's not possible to start with `port1` and go on to `port0`.

Example:

Read actual port.

```
DWORD dwBytesRead = 1;
BYTE data;
ReadFile(hDevice, data, dwBytesRead, &dwBytesRead, NULL);
if( dwBytesRead != 1 )
{
      //ERROR
}
```

Listing 2: Example ReadFile()

3.4.4 CloseHandle()

Signature:

```
BOOL CloseHandle(HANDLE hFileHandle);
```

Parameters:

`hFileHandle`	Handle to device file

Return:

`0`	Error, see `GetLastError()` for details
`!=0`	Success

Description:

Closes the device file that was opened with `CreateFile()`.

3.4.5 SetFilePointer()

Signature:

```
DWORD SetFilePointer(HANDLE hFileHandle, long lDistance,
      NULL, DWORD ControlCode);
```

Parameters:

`hFileHandle`	Handle to device file
`lDistance`	the new portnumber
	irrelevant
`ControlCode`	Control code specifying the device specific function to execute

Description:

With this function you can set the actual port. You may need this when you use `WriteFile()` or `ReadFile()`.

`ControlCode` options:

`FILE_BEGIN`	this option can choose the new port with `lDistance`
`FILE_CURRENT`	returns the actual port, `lDistance` has to be 0
`FILE_END`	returns the number of available ports, `lDistance` has to be 0

3.4.6 DeviceIoControl()

Signature:

```
int DeviceIoControl(
      HANDLE hDevice, DWORD dwIoControlCode,
      LPVOID lpInBuffer, DWORD dwInBufferSize,
      LPVOID lpOutBuffer, DWORD dwOutBufferSize,
      LPDWORD lpReturned, LPOVERLAPPED lpOverlapped
);
```

Parameters:

`hDevice`	Handle to already open device file
`dwIoControlCode`	Control code specifying the device specific function to execute
`lpInBuffer`	Pointer to the data going into the function (`IN` data)

dwInBufferSize	Size of the IN data (in bytes)
lpOutBuffer	Pointer to a buffer where data coming out of the function can be stored (OUT data)
dwOutBufferSize	Number of bytes available for the OUT data
lpReturned	Number of bytes actually written to the OUT data buffer
lpOverlapped	Unused, set to NULL

Description:

Executes a device specific function. The type of function is given by a control code in parameter dwIoControlCode. Each function has a specific set of parameters. Usually there is some data going into the function (IN data) and some data is returned out of the function (OUT data).

The following table lists all control codes recognised by the ext_IO driver.

Control Code	Function
IOCTL_DIO_SET_PIN	Set single pin, independent from actual port
IOCTL_DIO_CLR_PIN	Clear single pin, independent from actual port
IOCTL_DIO_GET_PIN	Read single pin, independent from actual port
IOCTL_DIO_REINIT	Reread current settings from registry and a configure the pins accordingly.
IOCTL_DIO_REQUEST_IRQ	Initialize an interrupt input pin
IOCTL_DIO_RELEASE_IRQ	De-Initialze an interrupt input pin
IOCTL_DIO_WAIT_IRQ	Function returns when interrupt is triggered
IOCTL_DIO_DONE_IRQ	Interrupt resets automatically so this does nothing

Table 6: IOCTL command codes

IO-Controls

IOCTL_DIO_REQUEST_IRQ

Parameters:

`hDevice`	Handle to already open device file
`dwIoControlCode`	`IOCTL_DIO_REQUEST_IRQ`
`lpInBuffer`	Pointer to `DWORD` where the pin is defined
`dwInBufferSize`	`sizeof(DWORD)`
`lpOutBuffer`	Unused, set to `NULL`
`dwOutBufferSize`	Unused, set to `NULL`
`lpReturned`	`pointer to return value - DWORD`
`lpOverlapped`	Unused, set to `NULL`

Return:

`0`	Error, see `GetLastError()` for details
`!=0`	Success

Description:

With this command you can request an interrupt for a specific pin. Once you requested an interrupt you can wait for it with the `IOCTL_DIO_WAIT_IRQ` function.

IOCTL_DIO_WAIT_IRQ

Parameters:

`hDevice`	Handle to already open device file
`dwIoControlCode`	`IOCTL_DIO_REQUEST_SYSINTR`
`lpInBuffer`	Pointer to `structure WAITIRQ` – defined in `dio_sdk.h`
`dwInBufferSize`	`sizeof(WAITIRQ)`
`lpOutBuffer`	Unused, set to `NULL`
`dwOutBufferSize`	Unused, set to `NULL`
`lpReturned`	`pointer to return value - DWORD`
`lpOverlapped`	Unused, set to `NULL`

Return:

`0`	Error, see `GetLastError()` for details
`!=0`	Success

Description:

This function behaves like `WaitForSingleObject()`. You can set the pin to wait for and the timeout in the `WAITIRQ` structure.

IOCTL_DIO_RELEASE_IRQ

Parameters:

`hDevice`	Handle to already open device file
`dwIoControlCode`	`IOCTL_DIO_REQUEST_SYSINTR`
`lpInBuffer`	Pointer to `DWORD` where the pin is defined
`dwInBufferSize`	`sizeof(DWORD)`
`lpOutBuffer`	Unused, set to `NULL`
`dwOutBufferSize`	Unused, set to `NULL`
`lpReturned`	`pointer to return value - DWORD`
`lpOverlapped`	Unused, set to `NULL`

Return:

`0`	Error, see `GetLastError()` for details
`!=0`	Success

Description:

With this command you can release an interrupt that you requested before.

IOCTL_DIO_SET_PIN

Parameters:

`hDevice`	Handle to already open device file
`dwIoControlCode`	`IOCTL_DIO_REQUEST_SYSINTR`
`lpInBuffer`	Pointer to `BYTE` where the pin is defined
`dwInBufferSize`	`sizeof(BYTE)`
`lpOutBuffer`	Unused, set to `NULL`
`dwOutBufferSize`	Unused, set to `NULL`
`lpReturned`	`pointer to return value - DWORD`
`lpOverlapped`	Unused, set to `NULL`

Return:

`0`	Error, see `GetLastError()` for details
`!=0`	Success

Description:

With this command you can set a single pin. The pin will be set independent from the actual selected port. You have to set the number of the pin, not a mask of pins.

Example:

Set pin 0;

```
BYTE pin = 0;
if(!DeviceIoControl(hDIO,IOCTL_DIO_SET_PIN, &pin, sizeof(BYTE),
                NULL, 0, &ret, NULL))
{
      //ERROR
}
```

Listing 3: Example set pin

IOCTL_DIO_CLR_PIN

Parameters:

`hDevice`	Handle to already open device file
`dwIoControlCode`	`IOCTL_DIO_REQUEST_SYSINTR`
`lpInBuffer`	Pointer to `BYTE` where the pin is defined
`dwInBufferSize`	`sizeof(BYTE)`
`lpOutBuffer`	Unused, set to `NULL`
`dwOutBufferSize`	Unused, set to `NULL`
`lpReturned`	`pointer to return value - DWORD`
`lpOverlapped`	Unused, set to `NULL`

Return:

`0`	Error, see `GetLastError()` for details
`!=0`	Success

Description:

With this command you can clear a single pin. The pin will be set independent from the actual selected port. You have to set the number of the pin, not a mask of pins.

Example:

Clear pin 7;

```
BYTE pin = 7;
if(!DeviceIoControl(hDIO,IOCTL_DIO_CLR_PIN, &pin, sizeof(BYTE),
                NULL, 0, &ret, NULL))
{
      //ERROR
}
```

Listing 4: Example clear pin

IOCTL_DIO_GET_PIN

Parameters:

`hDevice`	Handle to already open device file
`dwIoControlCode`	`IOCTL_DIO_REQUEST_SYSINTR`
`lpInBuffer`	Pointer to `BYTE` where the pin is defined
`dwInBufferSize`	`sizeof(BYTE)`
`lpOutBuffer`	Pointer to status of the pin - `BYTE`
`dwOutBufferSize`	`sizeof(BYTE)`
`lpReturned`	`pointer to return value - DWORD`
`lpOverlapped`	Unused, set to `NULL`

Return:

`0`	Error, see `GetLastError()` for details
`!=0`	Success

Description:

With this command you can clear a single pin. The pin will be set independent from the actual selected port. You have to set the number of the pin, not a mask of pins.

Example:

Get pin 10;

```
BYTE out;
BYTE pin = 10;

if(!DeviceIoControl(hDIO,IOCTL_DIO_GET_PIN, &pin, sizeof(BYTE),
           &out, sizeof(BYTE), &ret, NULL))
{
      //ERROR
}

printf("pin 10: 0x%x\r\n", out);
```

Listing 5: Example get pin

4 External Keyboard Driver

The organization of the matrix keyboard is very flexible. You can use a maximum of 8 (rows) * 8 (columns) or a maximum 16 (static keys). So you can connect up to 64 keys. You can choose if you want the driver to poll the keyboard every 20 ms or use an interrupt, this requires that you have the interrupt pin from the extension board connected.

In the case a key is pressed, the driver reads the scan code and saves the value. After additional 20 ms it checks the scan code. If the scan code is unchanged the scan code will be transformed with the information stored in the mapping table in a PS2 keyboard scan code. The routing of this keyboard code is the same as the one from a PS2 keyboard. The mapping table for converting a scan code in a PS2 keyboard code is stored in the registry.

The library `ext_keyboard.dll` and `exti2cboard.dll` has to be stored in flash memory into the `\FFSDISK` directory, if it is not already pre-loaded in the kernel.

Note:

Interrupt mode is not supported currently.

4.1 Configuration

The settings which influence the driver are stored under key:

`[HKLM\HARDWARE\DEVICEMAP\KEYBD\MATRIX]`

Entry	Type	Value	Description
`I2CDevAddr`	DWORD	`0x46`	Address of the I/O chip (PCA9555).
`I2CDevName`	String	`I2C1:`	I2C device used to access the extension board.
`Type`	DWORD	`17`	*See Table 8: Matrix Keyboard: Type registry value* .
`RowReverse`	DWORD	`0`	Reverse all bits of the row. Bit 0 to Bit 7, Bit 1 to Bit6.
`ColReverse`	DWORD	`0`	Reverse all bits of the column. Bit 0 to Bit 7, Bit 1 to Bit6.
`ChangeRowCol`	DWORD	`0`	Exchange the scan-value of row and column.
`AutoKeyUp`	DWORD	`0`	If a matrix key is pressed and the previous key is not released, this value sends the KEYUP message to the system.
`OutputScanCode`	DWORD	`0`	Set this value to 1 to output the scan-code of the currently pressed key as a debug message on the serial debug line.
`ExtIrqPin`	DWORD	`0`	I/O pin which is used for the interrupt on the J5 port
`UseIrqPin`	DWORD	`0`	If activated interrupt will be used, else the keyboard polls every 20 ms **Note: Irq mode is not supported currently.**

Table 7: Matrix Keyboard: Registry settings

Type	Function
0	Matrix keyboard driver OFF
1	Matrix keyboard max. 8x8+4, 8 rows, 8 cols, 4 static keys, single key detection
3	Matrix keyboard max. 8x8, 8 rows, 8 cols, 0 static keys, single key detection
17	Matrix keyboard max. 8x8+4, 8 rows, 8 cols, 4 static keys, multiple key detection
19	Matrix keyboard max. 8x8, 8 rows, 8 cols, 0 static keys, multiple key detection

Table 8: Matrix Keyboard: Type registry value

The organization of the columns is done under the following registry key:

`[HKLM\HARDWARE\DEVICEMAP\KEYBD\MATRIX\COLS]`

Key	Value	Comment
IOCol0	DWORD	Number of IO you want use for column 0.
…		
IOCol*n*	DWORD	Number of IO you want use for last column.

Table 9: Matrix Keyboard: Cols registry values

Note:

Please do not add other registry values to this key, because amount of values is directly used for amount of rows.

The organization of the rows is done under the following registry key:

`[HKLM\HARDWARE\DEVICEMAP\KEYBD\MATRIX\ROWS]`

Key	Value	Comment
IORow0	DWORD	Number of IO you want use for row 0.
…		
IORow*n*	DWORD	Number of IO you want use for last row.

Table 10: Matrix Keyboard: Rows registry values

Note:

Please do not add other registry values to this key, because amount of values is directly used for amount of rows.

The organization of the static keys is done under the following registry key:

`[HKLM\HARDWARE\DEVICEMAP\KEYBD\MATRIX\STATIC]`

Key	Value	Comment
`IOStaticKey0`	DWORD	Number of IO you want use for static key 0.
`StaticKey0`	DWORD	PS2 code for static key 0. See Table 13: Matrix Keyboard: PS2 Scan Codes
…		
`IOStaticKey`*`n`*	DWORD	Number of IO you want use for last static key.
`StaticKey`*`n`*	DWORD	PS2 code for last static key. See Table 13: Matrix Keyboard: PS2 Scan Codes

Table 11: Matrix Keyboard: Static registry values

You have to add two registry values for each static key. Please do not add other registry values to this key, because amount of values is directly used for amount of static keys. It's also possible to use this driver without matrix keys. E.g. if you have only a small number of keys you can configure the driver like shown in *Example2*. This could be also a good alternative to using digital IO driver. Especially with .NET framework because you get changes to the IO in the way of key strokes and have not poll to driver.

Mapping of matrix keys to PS2 values are stored under

`[HKLM\HARDWARE\DEVICEMAP\KEYBD\MATRIX\MAP]`

Under `\MAP` you can make settings in the following form:

Key	Value
"1"	Dword:2
"2"	Dword:3
"3"	Dword:4
"4"	Dword:5

Table 12: Matrix Keyboard: Map registry value

The value under Key (string!) is the scan code from the matrix keyboard. The range of this value is from 1 to 127 and must be given in decimal format. The value must be in hexadecimal form. In the above example you send the PS2-Code 2 if you press the matrix key 1.

PS2 Scan Codes:

V-KEY	PS2-Scan-Code
0	// Scan Code 0x0
VK_ESCAPE	// Scan Code 0x1
'1'	// Scan Code 0x2
'2'	// Scan Code 0x3
'3'	// Scan Code 0x4
'4'	// Scan Code 0x5
'5'	// Scan Code 0x6
'6'	// Scan Code 0x7
'7'	// Scan Code 0x8
'8'	// Scan Code 0x9
'9'	// Scan Code 0xA
'0'	// Scan Code 0xB
VK_HYPHEN	// Scan Code 0xC
VK_EQUAL	// Scan Code 0xD
VK_BACK	// Scan Code 0xE
VK_TAB	// Scan Code 0xF
'Q'	// Scan Code 0x10
'W'	// Scan Code 0x11
'E'	// Scan Code 0x12
'R'	// Scan Code 0x13

V-KEY	PS2-Scan-Code
'T'	// Scan Code 0x14
'Y'	// Scan Code 0x15
'U'	// Scan Code 0x16
'I'	// Scan Code 0x17
'O'	// Scan Code 0x18
'P'	// Scan Code 0x19
VK_LBRACKET	// Scan Code 0x1A
VK_RBRACKET	// Scan Code 0x1B
VK_RETURN	// Scan Code 0x1C
VK_LCONTROL	// Scan Code 0x1D
'A'	// Scan Code 0x1E
'S'	// Scan Code 0x1F
'D'	// Scan Code 0x20
'F'	// Scan Code 0x21
'G'	// Scan Code 0x22
'H'	// Scan Code 0x23
'J'	// Scan Code 0x24
'K'	// Scan Code 0x25
'L'	// Scan Code 0x26
VK_SEMICOLON	// Scan Code 0x27
VK_APOSTROPH	// Scan Code 0x28
VK_BACKQUOTE	// Scan Code 0x29
VK_LSHIFT	// Scan Code 0x2A
VK_BACKSLASH	// Scan Code 0x2B
'Z'	// Scan Code 0x2C
'X'	// Scan Code 0x2D
'C'	// Scan Code 0x2E
'V'	// Scan Code 0x2F
'B'	// Scan Code 0x30
'N'	// Scan Code 0x31
'M'	// Scan Code 0x32
VK_COMMA	// Scan Code 0x33
VK_PERIOD	// Scan Code 0x34
VK_SLASH	// Scan Code 0x35
VK_RSHIFT	// Scan Code 0x36
VK_MULTIPLY	// Scan Code 0x37

V-KEY	PS2-Scan-Code
VK_LMENU	// Scan Code 0x38
VK_SPACE	// Scan Code 0x39
VK_CAPITAL	// Scan Code 0x3A
VK_F1	// Scan Code 0x3B
VK_F2	// Scan Code 0x3C
VK_F3	// Scan Code 0x3D
VK_F4	// Scan Code 0x3E
VK_F5	// Scan Code 0x3F
VK_F6	// Scan Code 0x40
VK_F7	// Scan Code 0x41
VK_F8	// Scan Code 0x42
VK_F9	// Scan Code 0x43
VK_F10	// Scan Code 0x44
VK_NUMLOCK	// Scan Code 0x45
VK_SCROLL	// Scan Code 0x46
VK_NUMPAD7	// Scan Code 0x47
VK_NUMPAD8	// Scan Code 0x48
VK_NUMPAD9	// Scan Code 0x49
VK_SUBTRACT	// Scan Code 0x4A
VK_NUMPAD4	// Scan Code 0x4B
VK_NUMPAD5	// Scan Code 0x4C
VK_NUMPAD6	// Scan Code 0x4D
VK_ADD	// Scan Code 0x4E
VK_NUMPAD1	// Scan Code 0x4F
VK_NUMPAD2	// Scan Code 0x50
VK_NUMPAD3	// Scan Code 0x51
VK_NUMPAD0	// Scan Code 0x52
VK_DECIMAL	// Scan Code 0x53
VK_SNAPSHOT	// Scan Code 0x54
VK_F11	// Scan Code 0x57
VK_F12	// Scan Code 0x58
VK_LWIN	// Scan Code 0x5B
VK_RWIN	// Scan Code 0x5C
VK_APPS	// Scan Code 0x5D
VK_HELP	// Scan Code 0x63
VK_F13	// Scan Code 0x64
VK_F14	// Scan Code 0x65
VK_F15	// Scan Code 0x66

V-KEY	PS2-Scan-Code
VK_F16	// Scan Code 0x67
VK_F17	// Scan Code 0x68
VK_F18	// Scan Code 0x69
VK_F19	// Scan Code 0x6A
VK_F20	// Scan Code 0x6B
VK_F21	// Scan Code 0x6C
VK_F22	// Scan Code 0x6D
VK_F23	// Scan Code 0x6E
VK_F24	// Scan Code 0x76
VK_DIVIDE	// Scan Code 0xE035
VK_SNAPSHOT	// Scan Code 0xE037
VK_RMENU	// Scan Code 0xE038
VK_HOME	// Scan Code 0xE047
VK_UP	// Scan Code 0xE048
VK_PRIOR	// Scan Code 0xE049
VK_LEFT	// Scan Code 0xE04B
VK_RIGHT	// Scan Code 0xE04D
VK_END	// Scan Code 0xE04F
VK_DOWN	// Scan Code 0xE050
VK_NEXT	// Scan Code 0xE051
VK_INSERT	// Scan Code 0xE052
VK_DELETE	// Scan Code 0xE053
VK_LWIN	// Scan Code 0xE05B
VK_RWIN	// Scan Code 0xE05C
VK_APPS	// Scan Code 0xE05D

Table 13: Matrix Keyboard: PS2 Scan Codes

Scan codes matrix 8x8:

	C0	C1	C2	C3
R0	0x01	0x02	0x03	0x04
R1	0x11	0x12	0x13	0x14
R2	0x21	0x22	0x23	0x24
R3	0x31	0x32	0x33	0x34
R4	0x41	0x42	0x43	0x44
R5	0x51	0x52	0x53	0x54
R6	0x61	0x62	0x63	0x64
R7	0x71	0x72	0x73	0x74

Table 14: Matrix Keyboard: Scan Codes matrix 8x8 C0 – C3

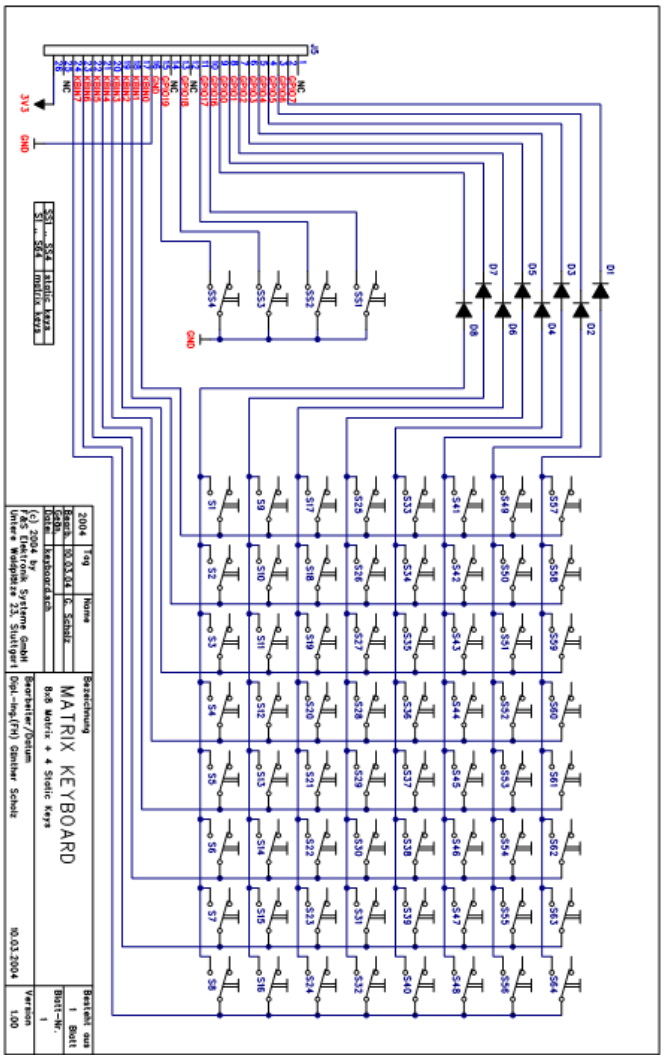
	C4	C5	C6	C7
R0	0x05	0x06	0x07	0x08
R1	0x15	0x16	0x17	0x18
R2	0x25	0x26	0x27	0x28
R3	0x35	0x36	0x37	0x38
R4	0x45	0x46	0x47	0x48
R5	0x55	0x56	0x57	0x58
R6	0x65	0x66	0x67	0x68
R7	0x75	0x76	0x77	0x78

Table 15: Matrix Keyboard: Scan Codes matrix 8x8 C4 – C7

4.2 Configuration Example

4.2.1 Hardware configuration

The following schematic show the connection of a keyboard with a 8x8 matrix and four static keys.



Note:

Please note that this is a sample schematic only. It is desigend for the NetDCU Starterkit. Therefore the pin layout of the connector may not be usable on the extension board.

4.2.2 Registry configuration examples

A. Create matrix keyboard with matrix 2x2 and no static keys.

```
[HKLM\hardware\devicemap\keybd\matrix]
    "Type"=dword:10 ; multi no static keys
    "OutputSCanCode"=dword:1
    "Debug"=dword:0

[HKLM\hardware\devicemap\keybd\matrix\Cols]
    "IOCol0"=dword:4          ; IO4 - J2:pin 5
    "IOCol1"=dword:5          ; IO5 - J2:pin 6

[HKLM\hardware\devicemap\keybd\matrix\Rows]
    "IORow0"=dword:6          ; IO6 - J2:pin 7
    "IORow1"=dword:3          ; IO7 - J2:pin 8

[HKLM\hardware\devicemap\keybd\matrix\map]
    "1"=dword:1E      ; r0,c0 -> 'A'
    "2"=dword:30      ; r0,c1 -> 'B'
    "17"=dword:2E     ; r1,c0 -> 'C'
    "18"=dword:20     ; r1,c1 -> 'D'
```

Listing 6: Matrix keyboard configuration example A.

B. Create keyboard with four static keys and no matrix.

```
[HKLM\hardware\devicemap\keybd\matrix]
    "Type"=dword:11; multi with static keys
    "OutputSCanCode"=dword:1
    "Debug"=dword:0

[HKLM\hardware\devicemap\keybd\matrix\Static]
    "IOStaticKey0"=dword:0    ; IO0 - S3
    "IOStaticKey1"=dword:1    ; IO1 - S4
    "IOStaticKey2"=dword:2    ; IO2 - S5
    "IOStaticKey3"=dword:3    ; IO3 - S6
    "StaticKey0"=dword:E04B   ; VKEY_LEFT
    "StaticKey1"=dword:E048   ; VKEY_UP
    "StaticKey2"=dword:E04D   ; VKEY_RIGHT
    "StaticKey3"=dword:E050   ; VKEY_DOWN

; remove this key or delete all values
[HKLM\hardware\devicemap\keybd\matrix\Cols]

; remove this key or delete all values
[HKLM\hardware\devicemap\keybd\matrix\Rows]

; remove this key or delete all values
[HKLM\hardware\devicemap\keybd\matrix\map]
```

Listing 7: Matrix keyboard configuration example B.

4.2.3 The EKB Driver in Applications

The external keyboard driver behaves like a normal keyboard driver. You can assign a ps2 code for each key of your matrix keyboard. So you can use programs that do normal keyboard requests.

5 Analogue Input

The extension board features 8 analogue inputs, each having a resolution of 12 bit.

This eight inputs can be read with this driver. The selection of the channel can be done statically within registry or dynamically with the `SetFilePointer()` function.

5.1 Configuration

Configuration of the driver is done by setting some registry values under the following registry key:

`[HKLM\Drivers\BuiltIn\ANALOGIN]`

Required settings:

Entry	Type	Value	Description
`Dll`	String	`ext_ain.dll`	Driver DLL
`FriendlyName`	String	`Extension analog in driver`	Description
`Prefix`	String	`AIN`	For AIN<index>:
`Index`	DWORD	`1`	For AIN1:
`Order`	DWORD	`301`	Load sequence.
`I2CDevAddr`	DWORD	`0x96`	I2C address for the analogue input controller (ADS7828).
`I2CDevName`	String	`I2C1:`	I2C device used to access the extension board.
`Debug`	DWORD	`0`	Debug verbosity
`Channel`	DWORD	`0-7`	Default analogue channel

Table 16: Analogue Input: Registry

5.2 Programming Example:

C. Open one analogue channel:

```
HANDLE hAIN;
hAIN = CreateFile( _T("AIN1:"),GENERIC_READ, 0, NULL, OPEN_EXISTING
                              ,FILE_ATTRIBUTE_NORMAL, NULL );
if( hAIN == INVALID_HANDLE_VALUE )
{
  ERRORMSG(1,L"Can not open AIN1. LastError = 0x%x\r\n",GetLastError()));
  return(FALSE);
}
```

Listing 8: Analogue Input: Open channel

D. Read data from previously opened channel:

```
unsigned short data;
DWORD dwSamples = 1;
ReadFile( hAIN, data, dwSamples, &dwSamples, NULL );
if( dwSamples != 1 )
{
  ERRORMSG(1,L"Can not read from AIN1. LE = 0x%x\r\n",GetLastError()));
}
```

Listing 9: Analogue Input: reading samples

E. Select another channel without changing registry:

```
int nChannel = 0x0;
SetFilePointer( hAIN, nChannel, 0, FILE_BEGIN );
```

Listing 10: Analogue Input: changing channel from application

F. Closing the analogue channel:

```
CloseHandle(hAIN);
```

Listing 11: Analogue Input: closing a channel

Appendix

Important Notice

The information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. F&S Elektronik Systeme assumes no responsibility, however, for possible errors or omissions, or for any consequences resulting from the use of the information contained in this documentation.

F&S Elektronik Systeme reserves the right to make changes in its products or product specifications or product documentation with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

F&S Elektronik Systeme makes no warranty or guarantee regarding the suitability of its products for any particular purpose, nor does F&S Elektronik Systeme assume any liability arising out of the documentation or use of any product and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

Specific testing of all parameters of each device is not necessarily performed unless required by law or regulation.

Products are not designed, intended, or authorized for use as components in systems intended for applications intended to support or sustain life, or for any other application in which the failure of the product from F&S Elektronik Systeme could create a situation where personal injury or death may occur. Should the Buyer purchase or use a F&S Elektronik Systeme product for any such unintended or unauthorized application, the Buyer shall indemnify and hold F&S Elektronik Systeme and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorized use, even if such claim alleges that F&S Elektronik Systeme was negligent regarding the design or manufacture of said product.

Specifications are subject to change without notice.

Warranty Terms

Hardware Warranties

F&S guarantees hardware products against defects in workmanship and material for a period of two (2) years from the date of shipment. Your sole remedy and F&S's sole liability shall be for F&S, at its sole discretion, to either repair or replace the defective hardware product at no charge or to refund the purchase price. Shipment costs in both directions are the responsibility of the customer. This warranty is void if the hardware product has been altered or damaged by accident, misuse or abuse.

Software Warranties

Software is provided "AS IS". F&S makes no warranties, either express or implied, with regard to the software object code or software source code either or with respect to any third party materials or intellectual property obtained from third parties. F&S makes no warranty that the software is useable or fit for any particular purpose. This warranty replaces all other warranties written or unwritten. F&S expressly disclaims any such warranties. In no case shall F&S be liable for any consequential damages.

Disclaimer of Warranty

THIS WARRANTY IS MADE IN PLACE OF ANY OTHER WARRANTY, WHETHER EXPRESSED, OR IMPLIED, OF MERCHANTABILITY, FITNESS FOR A SPECIFIC PURPOSE, NON-INFRINGEMENT OR THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION, EXCEPT THE WARRANTY EXPRESSLY STATED HEREIN. THE REMEDIES SET FORTH HEREIN SHALL BE THE SOLE AND EXCLUSIVE REMEDIES OF ANY PURCHASER WITH RESPECT TO ANY DEFECTIVE PRODUCT.

Limitation on Liability

UNDER NO CIRCUMSTANCES SHALL F&S BE LIABLE FOR ANY LOSS, DAMAGE OR EXPENSE SUFFERED OR INCURRED WITH RESPECT TO ANY DEFECTIVE PRODUCT. IN NO EVENT SHALL F&S BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES THAT YOU MAY SUFFER DIRECTLY OR INDIRECTLY FROM USE OF ANY PRODUCT. BY ORDERING THE PRODUCT, THE CUSTOMER APPROVES THAT THE F&S PRODUCT, HARDWARE AND SOFTWARE, WAS THOROUGHLY TESTED AND HAS MET THE CUSTOMER'S REQUIREMETS AND SPECIFICATIONS

Listings

Figures

Tables