

POWERFIP Library

User's Guide

Exoligent's Team

Version v0.7.2, 2021-09-21

Table of Contents

1. Disclaimer	1
2. Introduction	2
3. Installation	3
3.1. Linux	3
3.1.1. Kernel Module	4
3.1.2. Firmware/Library	4
3.2. Windows	6
3.2.1. Driver	7
3.2.2. Firmware/Library	9
3.3. Examples	10
3.3.1. Simple Test - pwrfig_2sta	10
3.3.2. Performance Test - pwrfig_performance	13
4. Functions	15
4.1. General	15
4.1.1. init	15
4.1.2. exit	16
4.1.3. version_get	17
4.1.4. strerror	18
4.2. Device	19
4.2.1. device_list_get	19
4.2.2. device_open	20
4.2.3. device_reset	21
4.2.4. device_close	22
4.2.5. device_infos_get	23
4.2.6. device_report_get	24
4.3. AE/LE	25
4.3.1. aele_create	25
4.3.2. aele_delete	26
4.3.3. var_create	27
4.3.4. var_delete	30
4.4. Bus Arbiter	31
4.4.1. ba_mcycle_create	31
4.4.2. ba_mcycle_delete	35
4.4.3. ba_startup_calculate	36
4.4.4. ba_start	37
4.4.5. ba_stop	38
4.4.6. ba_commute	39

4.4.7. ba_status_get	40
4.5. Node	41
4.5.1. node_init	41
4.5.2. node_exit	45
4.5.3. node_status_get	46
4.5.4. node_report_get	48
4.5.5. node_start	52
4.5.6. node_stop	53
4.6. Variables	54
4.6.1. var_write	54
4.6.2. var_read	57
4.6.3. var_evt_set	60
4.7. Medium	61
4.7.1. medium_status_get	61
4.7.2. medium_cmd_set	63
4.8. Events	64
4.8.1. evt_process	64
5. Structures	65
5.1. General	65
5.1.1. date	65
5.1.2. version	66
5.2. Device	67
5.2.1. dev_infos	67
5.2.2. dev_report	68
5.3. Configuration	69
5.3.1. ba_request	69
5.3.2. ba_aper_msg_wind_cfg	70
5.3.3. ba_aper_var_wind_cfg	71
5.3.4. ba_per_wind_cfg	72
5.3.5. ba_wait_wind_cfg	73
5.3.6. ba_wind_cfg	74
5.3.7. ba_mcycle_cfg	75
5.3.8. ba_startup_cfg	76
5.3.9. node_ba_cfg	77
5.3.10. node_cfg	80
5.3.11. node_msg_cfg	86
5.3.12. var_cons_cfg	87
5.3.13. var_prod_cfg	88
5.3.14. var_sync_cfg	89

5.3.15. var_cfg	90
5.4. Objects	92
5.4.1. node	92
5.4.2. var	93
5.5. Infos/Status/Report	94
5.5.1. ba_status	94
5.5.2. medium_status	95
5.5.3. node_infos	96
5.5.4. node_report	97
5.5.5. node_status	98
5.5.6. rx_err	99
5.5.7. tx_err	101
5.6. Event	102
5.6.1. event	102
6. Enumerations	103
6.1. aper_var_channel_type	103
6.2. ba_id_type	104
6.3. ba_startup_mode	105
6.4. ba_state	107
6.5. ba_wind_type	108
6.6. bitrate	109
6.7. error_code	110
6.8. event_code	114
6.9. evt_type	117
6.10. frame_type	118
6.11. medium_cmd_flag	119
6.12. medium_state	121
6.13. msg_rx_seg_cap	122
6.14. node_operation	123
6.15. node_state	124
6.16. var_err_code	125
6.17. var_flags	127
6.18. var_type	130
Appendix A: Revision History	131

Chapter 1. Disclaimer

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. EXOLIGENT MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Exoligent disclaims all liability arising from this information and its use. Use of Exoligent devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Exoligent from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Exoligent intellectual property rights unless otherwise stated.

Exoligent SARL

390 rue d'Estienne d'Orves, 92700 Colombes - France

Tel: +33(0) 1 42 42 42 00

Web Site: <https://www.exoligent.com/>

Copyright © 2021 Exoligent SARL. All rights reserved.

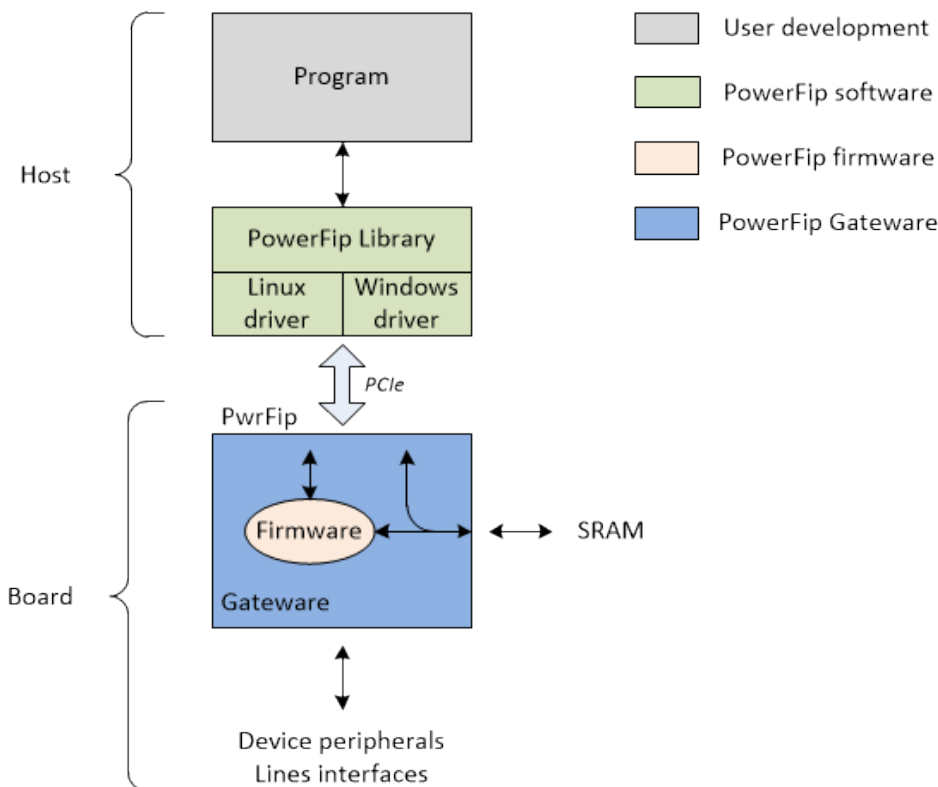
Chapter 2. Introduction

PowerFIP library is a C API providing a programming interface to the Exoligent's FIP/WorldFIP coprocessor.

It offers a set of functions for the FIP/WorldFIP network control:

- Conduct FIP Exchanges:
 - Periodic Variables
- Master/Slave capability:
 - Macrocycle creation
 - Bus arbiter management
- Channel Control
 - Medium redundancy
- Process Events from the network

PowerFIP overview (Principle Diagram)



Chapter 3. Installation

Let's start by downloading the archive of the latest version of the library from the Exoligent website: [Download section](#)

3.1. Linux

The linux archive has the following tree structure

- **[docs]**
 - PowerFIP Library - User's Guide (*.pdf) :
The User's Guide in PDF format.
 - powerfip.html :
The User's Guide in HTML format.
- **[drivers]**
 - **[linux]**
The source code of the Linux kernel module for Exoligent PowerFIP PCI/PCIe devices:
 - **[udev.rules.d]**
 - 10-powerfip.rules
 - install.sh
 - Makefile
 - powerfip-pci.c
 - powerfip-pci.h
 - uninstall.sh
- **[firmware]**
The PowerFIP coprocessor firmware for RISC-V Soft-CPU target :
 - powerfip-firmware.bin
- **[include]**
Header files to include in your projects to use the library:
 - libpowerfip.h
 - mbox-common.h
 - powerfip-common.h
 - powerfip-drv.h
 - powerfip-mbox-common.h
- **[lib]**
The library precompiled according to the architecture of the archive package:
 - libpowerfip.a

- libpowerfip.so.0.7.2
- **[tools]** :
Turnkey examples to get started as soon as possible!
 - **[pwrfip_2sta]**
 - **[pwrfip_performance]**
- install.sh
- uninstall.sh
- release-notes.txt
- readme.txt

3.1.1. Kernel Module

Open a terminal, and go to the linux driver directory:

```
$ cd driver/linux
```

Execute the following commands to build and install the kernel module:

```
$ make  
$ sudo ./install.sh
```



Two files will be copied to your system

1. **powerfip.ko** file to the path:
/lib/modules/\$(uname -r)/kernel/drivers/fip
2. **10-powerfip.rules** file to the path:
/etc/udev/rules.d

To remove the kernel module, enter the following command :

```
$ sudo ./uninstall.sh
```

3.1.2. Firmware/Library

Open a terminal, and go to the archive package root.

Then, enter the following command to install the PowerFIP firmware and library on your machine:

```
$ sudo ./install.sh
```


**Files will be copied to your system**

1. **powerfip-firmware.bin** file to the path:
/usr/local/lib/firmware
2. **libpowerfip.a** and **libpowerfip.so.0.7.2** files to the path:
/usr/local/lib
3. **header (*.h)** files to the path:
/usr/local/include/powerfip

To remove these files, enter the following command:

```
$ sudo ./uninstall.sh
```

3.2. Windows

Run the latest installer (.exe), and follow the wizard steps. All the files will be copied to the folder:

```
C:\Program Files (x86)\Exoligent\PowerFIP\
```

This directory has the following tree structure

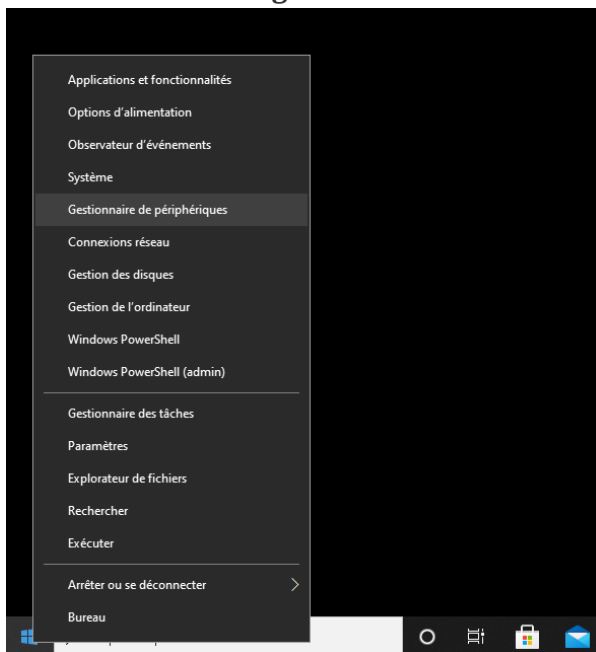
- **[docs]**
 - PowerFIP Library - User's Guide (*.pdf) :
The User's Guide in PDF format.
 - powerfip.html :
The User's Guide in HTML format.
- **[drivers]**
 - **[win]**
The binaries of the Windows driver for Exoligent PowerFIP PCI/PCIe devices:
 - **[Driver]**
 - **[x86]**
 - pwrfig.sys
 - **[x86-64]**
 - pwrfig64.sys
 - pwrfig.cat
 - pwrfig.inf
 - pwrfig64.cat
- **[firmware]**
The PowerFIP coprocessor firmware for RISC-V Soft-CPU target :
 - powerfip-firmware.bin
- **[include]**
Header files to include in your projects to use the library:
 - libpowerfip.h
 - mbox-common.h
 - powerfip-common.h
 - powerfip-driv.h
 - powerfip-mbox-common.h
- **[lib]**
The library precompiled according to the architecture of the archive package:
 - libpowerfip.a [static lib: x86_64-w64-mingw32 compiler (x86_64-8.1.0-posix-seh-rt_v6-

- rev0)]
- libpowerfip.dll [shared lib]
- libwinpthread-1.dll
- **[tools] :**
Turnkey examples to get started as soon as possible!
 - **[pwrfip_2sta]**
 - **[pwrfip_performance]**
- release-notes.txt
- readme.txt

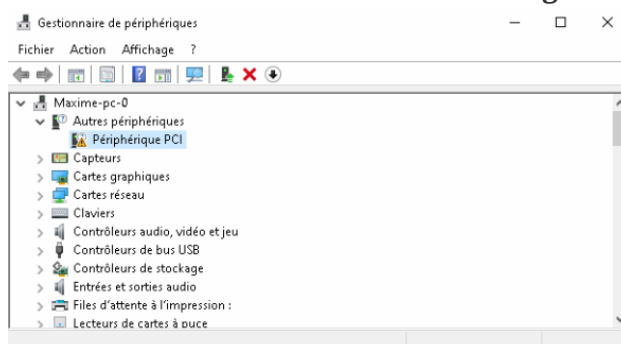
3.2.1. Driver

Installation

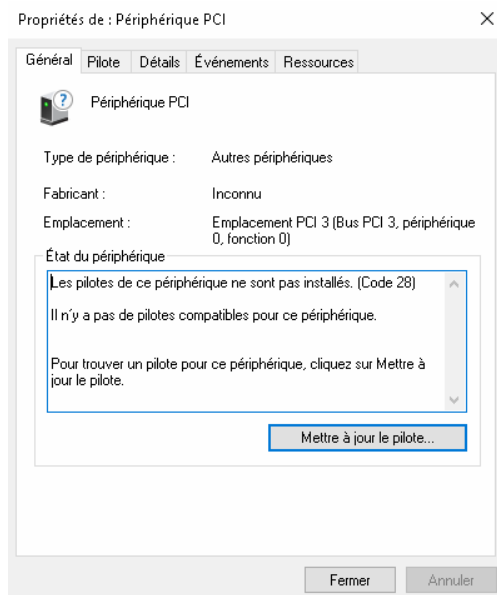
- Open the **Device Manager** window
 - Press **Windows+X** or right-click to **Start** button, then a menu will appear
 - Select **Device Manager** from the list



- Double-click on the new **Other PCI bridge device** detected



- Click on **Update the driver**

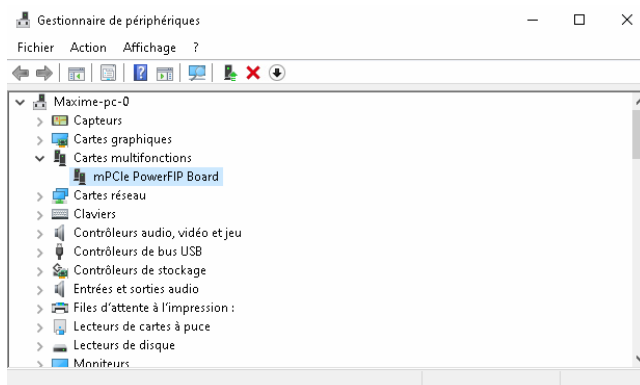


- Select the last package driver, and click on **Next** button

Target directory for driver

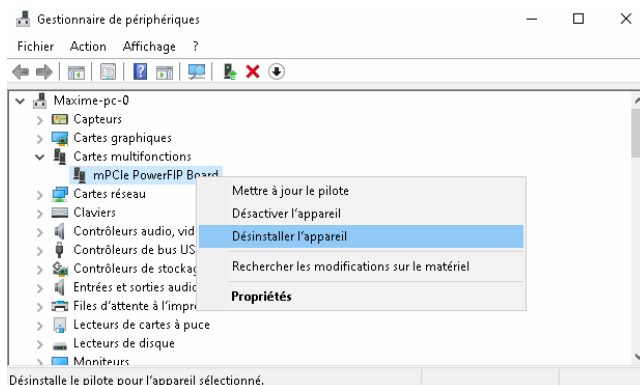
C:\Program Files (x86)\Exoligent\PowerFIP\drivers\win

- PowerFIP driver is now installed !

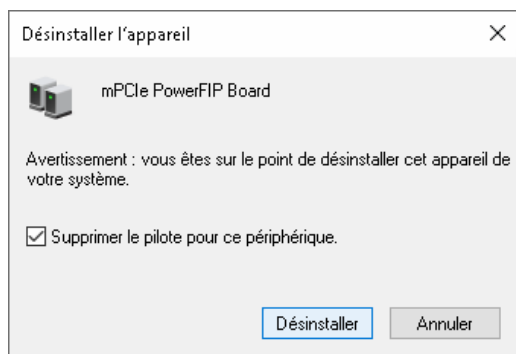


Uninstallation

- Go to the **Device Manager** window
- Right-click on the PowerFIP device to be removed, and click on **Uninstall** button



- Check **Remove the driver for this device**, and confirm the device uninstall



3.2.2. Firmware/Library

The PowerFIP firmware and library will be automatically installed by the package wizard.



Files will be copied to your system

1. **powerfip-firmware.bin** file to the path:
C:\Program Files (x86)\Exoligent\PowerFIP\firmware
2. **libpowerfip.dll** file to the path:
C:\Program Files (x86)\Exoligent\PowerFIP\lib
3. **header (*.h)** files to the path:
C:\Program Files (x86)\Exoligent\PowerFIP\include

To remove the package, execute the uninstaller wizard:

```
C:\Program Files (x86)\Exoligent\PowerFIP\Uninstall.exe
```

3.3. Examples

To quickly get into the swing of things, the source code of some examples is provided with the package.

To access to the examples, open a terminal and enter the following commands:

Linux

```
# From extracted archive directory
$ cd tools
```

Windows

```
$ cd C:\Program Files (x86)\Exoligent\PowerFIP\tools
```



If you do not have administrator rights, it is advisable to copy/paste the following directories into your own workspace:

- C:\Program Files (x86)\Exoligent\PowerFIP\tools
- C:\Program Files (x86)\Exoligent\PowerFIP\lib
- C:\Program Files (x86)\Exoligent\PowerFIP\include

For example paste it to a new directory in 'My Documents':

- C:\Users\%USERNAME%\Documents\PowerFIP\tools
- C:\Users\%USERNAME%\Documents\PowerFIP\lib
- C:\Users\%USERNAME%\Documents\PowerFIP\include

We will now briefly describe the examples:

3.3.1. Simple Test - pwrfig_2sta

This example aims at making two FIP nodes communicate with each other through the FIP network with exchange of two periodic variables.

The configuration of FIP nodes is done via static structures (see [tools/pwrfig_2sta/sta.h](#)), and tries to cover all available FIP services.



The example will therefore only work completely if you have two PCI/PCIe PowerFIP devices connected together with a FIP cable.

However, it is still possible to start a single station to observe the FIP traffic emitted by the device. Indeed the FIP node will try to start by default in *master* mode (with an active bus arbiter).

Build the example:*Linux*

```
$ cd tools/pwrfip_2sta  
$ make
```

Windows

```
# Build the example with static method way.  
# Compiler used is: x86_64-w64-mingw32-gcc (x86_64-8.1.0-posix-seh-rt_v6-rev0)  
$ set SYS_NAME=windows  
$ cd tools/pwrfip_2sta  
$ make
```

Get the help:*Linux*

```
$ sudo ./pwrfip_2sta -h
```

Windows

```
$ pwrfip_2sta.exe -h
```

Usage: pwrfip_2sta [OPTION]...

It tests FIP board communication with PowerFIP library.
By default, if no option is added, the app opens the first PCI/PCIe device with index 1 [-i 1], and starts FIP node 0 (addr=0) [-s 0].

Options:

- i device index [default=1]
- s FIP stations to start [default=0]
 - 0: station0
 - 1: station1
- l list the FIP boards present on the host machine
- h show this help and exit
- v, show version and exit

Examples:

```
pwrfip_2sta -i 1 -s 0
```

Launch FIP node 0:

Linux

```
$ sudo ./pwrfig_2sta -i 1 -s 0
```

Windows

```
$ pwrfig_2sta.exe -i 1 -s 0
```

```
[07-01 17:01:39.841216] app => [info] [fip] device info
[07-01 17:01:39.841250] app => [info]     index      : 1
[07-01 17:01:39.841251] app => [info]     vid        : 0x11aa
[07-01 17:01:39.841254] app => [info]     did        : 0x1556
[07-01 17:01:39.841256] app => [info]     ssvd       : 0x0000
[07-01 17:01:39.841258] app => [info]     ssdid      : 0x5811
[07-01 17:01:39.841260] app => [info]     bar_cnt    : 2
[07-01 17:01:39.841262] app => [info]     bar_bsz[0] : 4096
[07-01 17:01:39.841264] app => [info]     bar_base[0] : 0xa2000000
[07-01 17:01:39.841267] app => [info]     bar_bsz[1] : 33554432
[07-01 17:01:39.841269] app => [info]     bar_base[1] : 0xa0000000
[07-01 17:01:39.841271] app => [info]     irq_number  : 19
[07-01 17:01:39.841273] app => [info]     drv_version : 1.0.0
[07-01 17:01:39.841275] app => [info] test: v1.0.0 - pwrfig lib: v0.7.2
[07-01 17:01:39.841278] sta0 => [info] [fip] node configuration
[07-01 17:01:39.841280] sta0 => [info] [fip] bus arbiter infos
[07-01 17:01:39.841282] sta0 => [info]     start      : 1306800us
[07-01 17:01:39.841285] sta0 => [info]     election    : 77700us
[07-01 17:01:39.841287] sta0 => [info] [fip] node init
[07-01 17:01:39.841298] sta0 => [event] reset component (powerfip)
[...]
```

The example sequence is as follows:

- Open PCI/PCIe device (index 1)
- Load Configuration (fip node 0)
- Start the Bus Arbiter (start-up time: 1306800us, election time: 77700us)
- Infinite Loop



During this loop, an interrupt is raised by the FIP coprocessor each time an ID_DAT(0x9003) is transmitted on the FIP network (i.e. at the macrocycle frequency: here 40ms).

This interrupt triggers the user handler linked to this synchronization variable (see: `tst_pwrfig_end_wind_per_handler` function in the file: `tools/pwrfig_2sta/sta.c`).

As this function is clocked on the macrocycle, we take the opportunity to perform some actions within it:

- Produce a variable on the network (variable ID: 0x8426)
 - Consume a variable from the network (variable ID: 0x8427).
-
- Break the loop on an user keyboard press
 - Stop Bus Arbiter
 - Unload Configuration
 - Close PCI/PCIe device

3.3.2. Performance Test - pwrfig_performance

This example is very similar in structure to the previous example.

However, here the goal of the test is to measure the performance of the user read/write operation of the FIP variables from/to the database embedded in the FIP coprocessor.

Thanks to this we can evaluate the min/avg/max access times to the coprocessor depending on the amount of useful data read or written.

At the end of the test, a diagnostic report is generated and gives the access times (minimum, average, maximum) according to the length of the FIP user data produced or consumed: `tools/pwrfig_performance/report_idx1.txt`.

In the same way as for the previous test, to have a complete performance report, the two FIP nodes in the example must be connected to each other.

Example of two PCI/PCIe PowerFIP devices on the same PC with two terminals:

Linux - Test launch

```
# => Terminal 1
# starts PCI/PCIe device index 1 with FIP node configuration 0
$ sudo ./pwrfig_2sta -i 1 -s 0
```



```
# => Terminal 2
# start PCI/PCIe device index 2 with FIP node configuration 1
$ sudo ./pwrfig_2sta -i 2 -s 1
```

Windows - Test launch

```
# => Terminal 1
# starts PCI/PCIe device index 1 with FIP node configuration 0
$ pwrfig_2sta.exe -i 1 -s 0
```

```
# => Terminal 2
# start PCI/PCIe device index 2 with FIP node configuration 1
$ pwrfig_2sta.exe -i 2 -s 1
```

This makes it possible to generate performance curves according to the execution context (PC architecture, real-time OS, etc.).

Chapter 4. Functions

In this chapter, we will discover and describe the whole API (Application Programming Interface) of *POWERFIP*.

4.1. General

4.1.1. init

Description

Library internal initialization

Prototype

```
int pwrfig_init()
```

Parameters

- *IN* - None
- *OUT* - None

Return Value

If successful, `pwrfig_init()` returns 0.

If unsuccessful, `pwrfig_init()` returns -1 and sets `errno` value.

Remarks



This function must always be called before using any other function of the library.

4.1.2. exit

Description

Free all internal ressources used by the PowerFIP library

Prototype

```
void pwrfig_exit()
```

Parameters

- *IN* - None
- *OUT* - None

Return Value

NONE

Remarks



This function must always be called at the end of the use of the library.

4.1.3. version_get

Description

Gets the software library version.

Prototype

```
const struct pwrfig_version *pwrfig_version_get()
```

Parameters

- *IN* - None
- *OUT* - None

Return Value

Pointer to a `struct pwrfig_version`.

Example

```
int main(int argc, char *argv[])
{
    const struct pwrfig_version *lib_version;

    pwrfig_init();

    /* pwrfig - get lib version */
    lib_version = pwrfig_version_get();

    printf("pwrfig lib: v%d.%d.%d [build date: %02d/%02d/%02d]\n",
        lib_version->info.major,
        lib_version->info.minor,
        lib_version->info.patch,
        lib_version->date.info.month,
        lib_version->date.info.day,
        lib_version->date.info.year);

    pwrfig_exit();

    return 0;
}
```

4.1.4. strerror

Description

Gets the error string specified by its error code.



The library provides this error code via the *lvalue*: ***errno***

Prototype

```
const char *pwrfip_strerror(int err)
```

Parameters

- *IN*
 - **err**:
Error code.
- *OUT* - None

Return Value

Error in string format.

Remarks

See `enum pwrfip_error_code` to get the list of the specific library errors codes.

4.2. Device

4.2.1. device_list_get

Description

Gets the list of PCI/PCIe powerfip devices present on the system.



The PowerFIP driver supports up to **16** devices.

Prototype

```
int pwrfig_device_list_get(struct pwrfig_dev_infos *dev_infos,  
                           int *dev_cnt)
```

Parameters

- *IN* - None
- *OUT*
 - **dev_infos**:
Information list of detected devices.
See `struct pwrfig_dev_infos`.
 - **dev_cnt**:
Count of detected devices.

Return Value

If successful, `pwrfig_device_list_get()` returns 0.

If unsuccessful, `pwrfig_device_list_get()` returns -1 and sets `errno` to one of the following values:

- **EINVAL**:
Invalid input/output parameter.

4.2.2. device_open

Description

Opens a PCI/PCIe powerfip device.

Prototype

```
struct pwrfig_dev *pwrfig_device_open(uint8_t dev_id)
```

Parameters

- *IN*
 - **dev_id:**
Device index on the system.
- *OUT* - None

Return Value

If successful, `pwrfig_device_open()` returns a new `struct pwrfig_dev` pointer (opaque structure).

If unsuccessful, `pwrfig_device_open()` returns NULL and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input parameter.
- **ENOMEM:**
Memory allocation error.
- **[..]:**
Other posix errors related to a file opening error

4.2.3. device_reset

Description

Resets PCI/PCIe PowerFIP device.

- FIP Coprocessor reset
- Logic interface reset (of carrier board)

Prototype

```
int pwrfig_device_reset(struct pwrfig_dev *dev)
```

Parameters

- *IN*
 - **dev:**
Pointer to the device to reset.
- *OUT* - None

Return Value

If successful, `pwrfig_device_reset()` returns 0.

If unsuccessful, `pwrfig_device_reset()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input parameter.

4.2.4. device_close

Description

Closes a PCI/PCIe powerfip device.

Prototype

```
int pwrfig_device_close(struct pwrfig_dev *dev)
```

Parameters

- *IN*
 - **dev:**
Pointer to the device to close.
- *OUT* - None

Return Value

If successful, `pwrfig_device_close()` returns 0.

If unsuccessful, `pwrfig_device_close()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid device pointer.

4.2.5. device_infos_get

Description

Gets information relative to the PCI/PCIe device.

Prototype

```
int pwrfig_device_infos_get(struct pwrfig_dev *dev,  
                           struct pwrfig_dev_infos *info)
```

Parameters

- *IN*
 - **dev:**
Pointer to the device to query.
- *OUT*
 - **info:**
Info structure.
See `struct pwrfig_dev_infos`.

Return Value

If successful, `pwrfig_device_infos_get()` returns 0.

If unsuccessful, `pwrfig_device_infos_get()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input/output parameter.

4.2.6. device_report_get

Description

Gets various information about the PowerFIP board.

Prototype

```
int pwrfig_device_report_get(struct pwrfig_dev *dev,  
                             struct pwrfig_dev_report *report)
```

Parameters

- *IN*
 - **dev:**
Pointer to the device to query.
- *OUT*
 - **report:**
Device report.
See `struct pwrfig_dev_report`.

Return Value

If successful, `pwrfig_device_report_get()` returns 0.

If unsuccessful, `pwrfig_device_report_get()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input/output parameter.

4.3. AE/LE

4.3.1. aele_create

Description

The function `pwrfig_aele_create()` creates an Application/Layer entity attached to a FIP node. This container will gather all the production/consumptions variables and messages related to the local application.

Prototype

```
struct pwrfig_aele *pwrfig_aele_create(struct pwrfig_node *node);
```

Parameters

- *IN*
 - **node:**
Pointer to a `struct pwrfig_node`.
- *OUT* - None

Return Value

Pointer to a `struct pwrfig_aele` (opaque structure).

4.3.2. aele_delete

Description

The function `pwrfig_aele_delete()` deallocates a specific application/layer entity; and removes all items (variable, messages) attached to it.

Prototype

```
int pwrfig_aele_delete(struct pwrfig_aele *aele)
```

Parameters

- *IN*
 - **aele:**
Application entity to delete.
- *OUT* - None

Return Value

If successful, `pwrfig_aele_delete()` returns 0.

If unsuccessful, `pwrfig_aele_delete()` returns -1 and sets `errno` to one of the following values:

- **PWRFIP_ERR_AELE_NOT_STOP:**
AE/LE is currently running. Stop it before try to delete it.

4.3.3. var_create

Description

Creates a FIP variable inside an user's application context.

Prototype

```
struct pwrfig_var *pwrfig_var_create(struct pwrfig_aele *aele,  
                                     struct pwrfig_var_cfg *cfg)
```

Parameters

- *IN*
 - **aele:**
Pointer to a user's application context (AE/LE).
 - **cfg:**
Pointer to a variable's configuration structure.
See `struct pwrfig_var_cfg`.
- *OUT* - None

Return Value

If successful, `pwrfig_var_create()` returns a new `struct pwrfig_var` pointer.

If unsuccessful, `pwrfig_var_create()` returns NULL and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input parameters.
- **ENOMEM:**
Memory allocation error.
- **PWRFIP_ERR_AELE_NOT_STOP:**
AE/LE is currently running. Stop it before trying to create new objects.
- **PWRFIP_ERR_CFG_VAR_EXIST:**
Impossible to create this variable. The AE/LE context already contains a variable with this FIP identifier.
- **PWRFIP_ERR_CFG_VAR_DIR:**
An update of the variable tried to be applied; but it's impossible to change the direction of the variable (prod/cons) for this FIP identifier.
- **PWRFIP_ERR_CFG_MSG_PROD:**
Impossible to link a produced message on this FIP identifier. A consumed variable is already attached to it.
- **PWRFIP_ERR_CFG_MSG_DIR:**
A FIP production message is already attached to this FIP identifier; so it is impossible to change the production channel.

Example

```

void sync_var_handler(struct pwrfig_node *node,
    struct pwrfig_var *var, struct pwrfig_event *evt);

static struct pwrfig_var_cfg prod_var_cfg = {
    .type = PWRFIG_VAR_TYPE_PROD,
    .id = 0x8426,
    .prod.payload_bsz = 12,
    .prod.flags = \
        /* enable prod status */
        PWRFIG_VAR_FLAGS_REFRESH | \
        /* enable aper msg request */
        PWRFIG_VAR_FLAGS_APER_MSG_REQ,
    .prod.refreshment_ustime = 16000, /* 16ms */
    .prod.evt_type = PWRFIG_EVT_TYPE_NONE,
    .pwrfig_var_handler = NULL,
};

static struct pwrfig_var_cfg cons_var_cfg = {
    .type = PWRFIG_VAR_TYPE_CONS,
    .id = 0x8427,
    .cons.payload_bsz = 12,
    .cons.flags = \
        /* enable prod status */
        PWRFIG_VAR_FLAGS_REFRESH | \
        /* enable promptness checking */
        PWRFIG_VAR_FLAGS_PROMPT | \
        /* enable pdu + len bytes checking */
        PWRFIG_VAR_FLAGS_CHK_PDU_LEN,
    .cons.promptness_ustime = 16000, /* 16ms */
    .cons.evt_type = PWRFIG_EVT_TYPE_NONE,
    .pwrfig_var_handler = NULL,
};

static struct pwrfig_var_cfg sync_var_cfg = {
    .type = PWRFIG_VAR_TYPE_SYNC,
    .id = 0x9003,
    .pwrfig_var_handler = sync_var_handler,
};

int main(int argc, char *argv[])
{
    int err = 0;
    struct pwrfig_node *node;
    struct pwrfig_aele *al;
    struct pwrfig_var *prod_var, *cons_var, *sync_var;

    /**

```



```

    * Node initialization
    */
    /*...*/

    /* create an aele context */
    al = pwrfig_aele_create(node);
    if (!al) {
        printf("aele creation failed: %s\n", pwrfig_strerror(errno));
        err = -1;
        goto end;
    }

    /* create a production variable */
    prod_var = pwrfig_var_create(al, &prod_var_cfg);
    if (!prod_var) {
        printf("production variable creation failed: %s\n", pwrfig_strerror(errno));
        err = -1;
        goto end;
    }

    /* create a consumption variable */
    cons_var = pwrfig_var_create(al, &cons_var_cfg);
    if (!cons_var) {
        printf("consumption variable creation failed: %s\n", pwrfig_strerror(errno));
        err = -1;
        goto end;
    }

    /* create a synchronization variable */
    sync_var = pwrfig_var_create(al, &sync_var_cfg);
    if (!sync_var) {
        printf("synchronization variable creation failed: %s\n", pwrfig_strerror(
errno));
        err = -1;
        goto end;
    }

    /**
     * Other tasks
     */
    /* ... */

end:
    /**
     * Node exit
     */
    /*...*/
    return err;
}

```

4.3.4. var_delete

Description

Deallocates a specific variable from an application entity (AE/LE).

Prototype

```
int pwrfig_var_delete(struct pwrfig_var *var)
```

Parameters

- *IN*
 - **var:**
Pointer to the variable to delete.
See `struct pwrfig_var`.
- *OUT* - None

Return Value

If successful, `pwrfig_var_delete()` returns 0.

If unsuccessful, `pwrfig_var_delete()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input parameter.
- **PWRFIG_ERR_AELE_NOT_STOP:**
AE/LE attached to this variable is currently running. Stop it before try to delete it.

4.4. Bus Arbiter

4.4.1. ba_mcycle_create

Description

This function creates a new macrocycle (or Bus Arbiter table) attached to a FIP node.



To support this function, the FIP node has to set *Master* capability.

Prototype

```
struct pwrfip_ba_mcycle *pwrfip_ba_mcycle_create(  
    struct pwrfip_node *node,  
    struct pwrfip_ba_mcycle_cfg *cfg)
```

Parameters

- *IN*
 - **node:**
Pointer to the FIP node where to create the new macrocycle.
See `struct pwrfip_node`.
 - **cfg:**
Pointer to a macrocycle configuration structure.
See `struct pwrfip_ba_mcycle_cfg`.
- *OUT* - None

Return Value

If successful, `pwrfip_ba_mcycle_create()` returns a new `struct pwrfip_ba_mcycle` pointer (opaque structure).

If unsuccessful, `pwrfip_ba_mcycle_create()` returns NULL and sets `errno` to one of the following values:

- **EINVAL:**
Invalid node or macrocycle configuration structure.
- **ENOMEM:**
Memory allocation error.
- **PWRFIP_ERR_MCYCLE_WIND_UNKNOWN:**
Invalid macrocycle window type (see `enum pwrfip_ba_wind_type`).
- **PWRFIP_ERR_MCYCLE_WIND_COUNT:**
Macrocycle window count has to be greater than 0.
- **PWRFIP_ERR_MCYCLE_PER_WIND_REQ_COUNT:**
Requests count inside a macrocycle periodic window has to be greater than 0.
- **PWRFIP_ERR_MCYCLE_PER_WIND_REQ_UNKNOWN:**

Invalid request type inside a macrocycle periodic window.

Requests allowed are only *ID_DAT* and *ID_MSG* type.

- **PWRFIP_ERR_MCYCLE_WIND_TIME_INC:**

Overlap on macrocycle windows end times.

The *.end_ustime* field present in the configuration of certain types of bus arbiter window is a time relative to the beginning of the macrocycle.

This time must therefore be increasing as you go through the configuration list.

- **PWRFIP_ERR_MCYCLE_WIND_END:**

The macrocycle configuration must end with a time window (wait).

Example

```
#define TST_BA_PER_REQ_COUNT 4
static struct pwrfig_ba_request tst_ba_per_req[TST_BA_PER_REQ_COUNT] = {
    {
        .type = PWRFIP_BA_ID_DAT,
        .id = 0x8426,
    },
    {
        .type = PWRFIP_BA_ID_DAT,
        .id = 0x8427,
    },
    {
        .type = PWRFIP_BA_ID_DAT,
        .id = 0x0001,
    },
    {
        .type = PWRFIP_BA_ID_DAT,
        .id = 0x9003,
    },
};

static struct pwrfig_ba_wind_cfg tst_ba_wind_cfg[2][4] = {
    /* ba mcycle 1 */
    {
        /* 1 - periodic variable window */
        {
            .type = PWRFIP_BA_WIND_PER,
            .per.req_cnt = TST_BA_PER_REQ_COUNT,
            .per.req_list = &tst_ba_per_req[0],
        },
        /* 2 - aperiodic message window */
        {
            .type = PWRFIP_BA_WIND_APER_MSG,
            .aper_msg.end_ustime = 5000,
        },
        /* 3 - aperiodic variable window */
        {
```

```

        .type = PWRFIP_BA_WIND_APER_VAR,
        .aper_var.end_ustime = 10000,
    },
    /* 4 - resync wait window */
    {
        .type = PWRFIP_BA_WIND_WAIT,
        .wait.end_ustime = 15000,
        .wait.is_silent = 0, /* fill window with padding frames */
        .wait.is_ext_resync = 0, /* internal resynchronization */
    },
},
/* ba mcycle 2 */
{
    /* 1 - periodic variable window */
    {
        .type = PWRFIP_BA_WIND_PER,
        .per.req_cnt = TST_BA_PER_REQ_COUNT,
        .per.req_list = &tst_ba_per_req[0],
    },
    /* 2 - aperiodic message window */
    {
        .type = PWRFIP_BA_WIND_APER_MSG,
        .aper_msg.end_ustime = 5000,
    },
    /* 3 - resync wait window */
    {
        .type = PWRFIP_BA_WIND_WAIT,
        .wait.end_ustime = 30000,
        .wait.is_silent = 0, /* fill window with padding frames */
        .wait.is_ext_resync = 0, /* internal resynchronization */
    },
    /* no other window */
    {
        .type = 0,
    }
},
};

static struct pwrfig_ba_mcycle_cfg tst_ba_mcycle_cfg[2] = {
    /* ba mcycle 1 */
    {
        .wind_cnt = 4,
        .wind_list = &tst_ba_wind_cfg[0][0],
    },
    /* ba mcycle 2 */
    {
        .wind_cnt = 3,
        .wind_list = &tst_ba_wind_cfg[1][0],
    },
};

```

```

};

int main(int argc, char *argv[])
{
    int err = 0;
    struct pwrfig_node *node;
    struct pwrfig_ba_mcycle *mcycle[2];

    /**
     * Node initialization
     */
    /*...*/

    /* create two bus arbiter macrocycles */
    for (i = 0; i < 2; ++i) {
        struct pwrfig_ba_mcycle_cfg *mcycle_cfg = &tst_ba_mcycle_cfg[i];

        mcycle[i] = pwrfig_ba_mcycle_create(node, mcycle_cfg);
        if (!mcycle[i]) {
            /* get error */
            printf("macrocycle creation failed: %s\n", pwrfig_strerror(errno));
            err = -1;
            goto end;
        }
    }

    /**
     * Other tasks
     */
    /* ... */

end:
    /**
     * Node exit
     */
    /*...*/
    return err;
}

```

4.4.2. ba_mcycle_delete

Description

The function `pwrfig_ba_mcycle_delete()` deallocates a specific macrocycle.

Prototype

```
int pwrfig_ba_mcycle_delete(struct pwrfig_ba_mcycle *mcycle)
```

Parameters

- *IN*
 - **mcycle:**
Macrocycle to delete.
- *OUT* - None

Return Value

If successful, `pwrfig_ba_mcycle_delete()` returns 0.

If unsuccessful, `pwrfig_ba_mcycle_delete()` returns -1 and sets `errno` to one of the following values:

- **PWRFIP_ERR_BA_NOT_STOP:**
The macrocycle is currently running. Stop it before try to delete it.

4.4.3. ba_startup_calculate

Description

Tool function to calculate compliant start-up and election times for a bus arbiter in an environment where multiple FIP nodes are competing to become *Master*.

Prototype

```
int pwrfig_ba_startup_calculate(uint32_t *stup_ustime, uint32_t *elec_ustime,
                                struct pwrfig_ba_startup_cfg *cfg)
```

Parameters

- *IN*
 - **cfg:**
Input parameters for time calculation.
See `struct pwrfig_ba_startup_cfg`.
- *OUT*
 - **stup_ustime:**
BA start-up time calculated in microseconds.
 - **elec_ustime:**
BA election time calculated in microseconds.

Return Value

If successful, `pwrfig_ba_startup_calculate()` returns 0.

If unsuccessful, `pwrfig_ba_startup_calculate()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input/output parameters.
- **PWRFIG_ERR_BA_STUP_PHY_ADDR_INVALID:**
Local physical address exceeds maximum physical address given.
- **PWRFIG_ERR_BA_STUP_PRIO_INVALID:**
Local priority is higher than maximum one given.



BA Priority

Priority range is between [0;15], with 0 the highest priority.

- **PWRFIG_ERR_BA_STUP_TS_INVALID:**
Silence Time input parameter should not be 0.

4.4.4. ba_start

Description

Starts a specific macrocycle for the master node attached.

Prototype

```
int pwrfig_ba_start(struct pwrfig_ba_mcycle *mcycle)
```

Parameters

- *IN*
 - **mcycle:**
Macrocycle to start.
- *OUT* - None

Return Value

If successful, `pwrfig_ba_start()` returns 0.

If unsuccessful, `pwrfig_ba_start()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid macrocycle pointer.
- **PWRFIP_ERR_BA_NOT_STOP:**
The node has already a running macrocycle.
Use `pwrfig_ba_commute()` function instead.
- **PWRFIP_ERR_COM_XXX:**
Communication error with the coprocessor.

4.4.5. ba_stop

Description

Stops the macrocycle of a specific FIP node.

Prototype

```
int pwrfig_ba_stop(struct pwrfig_node *node)
```

Parameters

- *IN*
 - **node:**
FIP node to query.
See `struct pwrfig_node`.
- *OUT* - None

Return Value

If successful, `pwrfig_ba_stop()` returns 0.

If unsuccessful, `pwrfig_ba_stop()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid node pointer.
- **PWRFIP_ERR_COM_XXX:**
Communication error with the coprocessor.

4.4.6. ba_commute

Description

Switch to the execution of another FIP macrocycle.

Prototype

```
int pwrfig_ba_commute(struct pwrfig_ba_mcycle *mcycle)
```

Parameters

- *IN*
 - **mcycle:**
Pointer to the new macrocycle to switch to.
- *OUT* - None

Return Value

If successful, `pwrfig_ba_commute()` returns 0.

If unsuccessful, `pwrfig_ba_commute()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid macrocycle pointer.
- **PWRFIP_ERR_BA_NOT_RUN:**
The node does not have a running macrocycle.
Use `pwrfig_ba_start()` function instead.
- **PWRFIP_ERR_COM_XXX:**
Communication error with the coprocessor.

4.4.7. ba_status_get

Description

Gets the bus arbiter status for a given FIP node.

Prototype

```
int pwrfig_ba_status_get(struct pwrfig_node *node, struct pwrfig_ba_status *status)
```

Parameters

- *IN*
 - **node:**
FIP node to query.
See `struct pwrfig_node`.
- *OUT*
 - **status:**
Pointer to an output `struct pwrfig_ba_status`.

Return Value

If successful, `pwrfig_ba_status_get()` returns 0.

If unsuccessful, `pwrfig_ba_status_get()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input/output parameters.
- **PWRFIP_ERR_COM_XXX:**
Communication error with the coprocessor.

4.5. Node

4.5.1. node_init

Description

Creates a new FIP node context inside the library and load it to the local database of a binded coprocessor.

This is the main step in creating a FIP node.

Prototype

```
struct pwrfig_node *pwrfig_node_init(struct pwrfig_node_cfg *cfg)
```

Parameters

- *IN*
 - **cfg:**
Pointer to the FIP node configuration.
See `struct pwrfig_node_cfg`.
- *OUT* - None

Return Value

If successful, `pwrfig_node_init()` returns a new `struct pwrfig_node` pointer.

If unsuccessful, `pwrfig_node_init()` returns NULL and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input configuration.
- **ENOMEM:**
Memory allocation error.
- **PWRFIP_ERR_NODE_HANDLER_MISSING:**
Some user handlers are mandatory to continue the node creation:
 - `pwrfig_reset_handler`
 - `pwrfig_error_handler`
- **PWRFIP_ERR_NODE_FRM_TYPE_INVALID:**
Invalid frame type configuration. Should be:
 - `PWRFIP_FRM_FIP`
 - `PWRFIP_FRM_WORLDFIP`
- **PWRFIP_ERR_NODE_BITRATE_INVALID:**
Invalid FIP bitrate configuration. Should be:
 - `PWRFIP_BITRATE_31K25`
 - `PWRFIP_BITRATE_1M`

- PWRFIP_BITRATE_2M5
- PWRFIP_BITRATE_5M
- **PWRFIP_ERR_NODE_TR_INVALID:**
Invalid FIP turnaround time. Range should be:
 - @31.25Kbps : min=424us, max=2240us, default=424us
 - @1Mbps : min=10us, max=70us, default=30us
 - @2.5Mbps : min=14us, max=40us, default=14us
 - @5Mbps : min=32us, max=48us, default=32us
- **PWRFIP_ERR_NODE_TS_INVALID:**
Invalid FIP silence time. Range should be:
 - @31.25Kbps : min=4096us, max=64512us, default=4096us
 - @1Mbps : min=70us, max=2056us, default=150us
 - @2.5Mbps : min=96us, max=838us, default=96us
 - @5Mbps : min=92us, max=252us, default=92us
- **PWRFIP_ERR_NODE_RX_MSG_FIFO_SZ:**
Invalid queue size for message consumption.
Range value should be: [1..64].
- **PWRFIP_ERR_NODE_RX_MSG_SEG_CAP:**
Invalid segment capability for consumption message. Should be:
 - PWRFIP_MSG_SEG_IGNORE
 - PWRFIP_MSG_SEG_ACCEPT_ALL
 - PWRFIP_MSG_SEG_ACCEPT_LTD
- **PWRFIP_ERR_NODE_TX_MSG_FIFO_SZ:**
Invalid queue size for message transmission.
Range value should be: [1..64].
- **PWRFIP_ERR_NODE_TX_MSG_REPEAT:**
Invalid maximum repeats for acknowledged message transmission.
Range value should be: [0..3].
- **PWRFIP_ERR_NODE_BA_STUP_TIMES:**
The bus arbiter election time must be shorter than the start-up time.
- **PWRFIP_ERR_NODE_BA_REQ_FIFO_SZ:**
Invalid queue size for BA requests.
Range value should be: [1..64].
- **PWRFIP_ERR_DEV_ALREADY_BIND:**
The provided device is already bound to another FIP node session.
- **[..]:**
Other posix errors related to a file opening error

Example

```

void usr_rst_handler(struct pwrfig_node *node)
{
    /* calling the reset function for a pci/pcie device */
    if (pwrfig_device_reset(node->infos->cfg.dev)) {
        printf("reset failed: %s\n", pwrfig_strerror(errno));
        return;
    }
    _print(src, evt, "reset coprocessor component (powerfip) done\n");
}

void usr_err_handler(struct pwrfig_node *node,
                    enum pwrfig_error_code code)
{
    printf("error_handler: %s (err_code=%d)\n",
          pwrfig_strerror(code), code);
}

int main(int argc, char *argv[])
{
    int err = 0;
    struct pwrfig_dev *dev;
    struct pwrfig_node *node;
    struct pwrfig_node_cfg cfg;

    /* pwrfig lib initialization [mandatory] */
    if (pwrfig_init()) {
        printf("cannot init pwrfig library: %s\n",
              pwrfig_strerror(errno));
        return -1;
    }

    /* open a pci/pcie device (1st index) */
    dev = pwrfig_device_open(1);
    if (!dev) {
        printf("cannot open device: %s\n",
              pwrfig_strerror(errno));
        pwrfig_exit();
        return -1;
    }

    /**
     * Node initialization:
     * Minimal set-up for a PCI/PCIe device
     */
    cfg.fip_phy_addr = 1; /* fip node addr = 0x01 */
    cfg.fip_seg_num = 0; /* fip node belongs to fip segment 0 */
    cfg.fip_frm_type = PWRFIG_FRM_WORLDFIP; /* WorldFIP frame (IEC) */

```

```
cfg.fip_bitrate = PWRFIP_BITRATE_1M; /* 1Mbps */
cfg.turn_around_ustime = 0; /* default TR time for 1Mbps (30us) */
cfg.silence_ustime = 0; /* default TS time for 1Mbps (150us) */
cfg.enable_bimedium = 0; /* mono-medium topology (just one channel) */
cfg.msg.enable = 0; /* fip messaging not supported */
cfg.ba.enable = 0; /* no master capability for this node */
cfg.dev = dev; /* coprocessor attachment: pci/pcie device to bind */
cfg.pwrfip_error_handler = usr_err_handler; /* local handler to notify
                                             the internal errors of
                                             the library */
cfg.pwrfip_reset_handler = usr_rst_handler; /* local handler to reset
                                             the bound device */

node = pwrfip_node_init(&cfg);
if (!node) {
    printf("node initialization failed: %s\n",
        pwrfip_strerror(errno));
    err = -1;
    goto end;
}

/**
 * Other tasks
 */
/* ... */

end:
/**
 * Node exit
 */
/*...*/

/* close device */
pwrfip_device_close(dev);

/* pwrfip lib exit [mandatory] */
pwrfip_exit();
return err;
}
```


4.5.2. node_exit

Description

Stops the coprocessor and deallocates all resources attached to the FIP node inside the library.

Prototype

```
int pwrfig_node_exit(struct pwrfig_node *node)
```

Parameters

- *IN*
 - **node:**
Pointer to the FIP node to exit.
See `struct pwrfig_node`.
- *OUT* - None

Return Value

If successful, `pwrfig_node_exit()` returns 0.

If unsuccessful, `pwrfig_node_exit()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input parameter.
- **PWRFIP_ERR_DEV_IRQ_HANDLER_STOPPED:**
Internal IRQ handler for PCI/PCIe device is already stopped.
- **PWRFIP_ERR_AELE_NOT_STOP:**
Cannot stop the Application/Layer entity. The exit procedure has therefore failed.
- **PWRFIP_ERR_BA_NOT_STOP:**
Cannot stop the bus arbiter FSM. The exit procedure has therefore failed.
- **PWRFIP_ERR_COM_XXX:**
Communication error with the coprocessor.

4.5.3. node_status_get

Description

Get the FSM status of the FIP node.

Prototype

```
int pwrfig_node_status_get(struct pwrfig_node *node,
                          struct pwrfig_node_status *status)
```

Parameters

- *IN*
 - **node:**
Pointer to the target FIP node.
See `struct pwrfig_node`.
- *OUT*
 - **status:**
Pointer to an output `struct pwrfig_node_status`.

Return Value

If successful, `pwrfig_node_status_get()` returns 0.

If unsuccessful, `pwrfig_node_status_get()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input/output parameters.
- **PWRFIP_ERR_COM_XXX:**
Communication error with the coprocessor.

Example

```
static const char *pwrfig_node_state_str[_PWRFIP_NODE_STATE_MAX] = {
    /* PWRFIP_NODE_STATE_INITIAL */
    "intial",
    /* PWRFIP_NODE_STATE_LOADED */
    "loaded",
    /* PWRFIP_NODE_STATE_READY */
    "ready",
    /* PWRFIP_NODE_STATE_RUNNING */
    "running",
};

static const char *pwrfig_node_op_str[_PWRFIP_NODE_OP_MAX] = {
    /* _PWRFIP_NODE_OP_UNKNOWN */
    "unknown",
    /* PWRFIP_NODE_OP_WAIT_RX_RP_FRM */

```

```

    "rx rp frame",
    /* PWRFIP_NODE_OP_WAIT_TX_RP_FRM */
    "tx rp frame",
    /* PWRFIP_NODE_OP_WAIT_RX_ID_FRM */
    "rx id frame",
    /* PWRFIP_NODE_OP_WAIT_TX_ID_FRM */
    "tx id frame",
};

int main(int argc, char *argv[])
{
    int err = 0;
    struct pwrfig_node *node;
    struct pwrfig_node_status n_status;

    /**
     * Node initialization
     */
    /*...*/

    /* get node status */
    err = pwrfig_node_status_get(node, &n_status);
    if (err) {
        printf("node status getter failed: %s\n", pwrfig_strerror(errno));
        goto end;
    }

    printf("node_status\n");
    printf("  node_state      : %d (%s)\n", n_status.state,
        pwrfig_node_state_str[n_status.state]);
    printf("  node_op         : %d (%s)\n", n_status.op,
        pwrfig_node_op_str[n_status.op]);

    /**
     * Other tasks
     */
    /* ... */

end:
    /**
     * Node exit
     */
    /*...*/
    return err;
}

```

4.5.4. node_report_get

Description

Gets the full diagnostic report of the FIP node.

Prototype

```
int pwrfig_node_report_get(struct pwrfig_node *node,
                          struct pwrfig_node_report *report)
```

Parameters

- *IN*
 - **node:**
Pointer to the target FIP node.
See `struct pwrfig_node`.
- *OUT*
 - **report:**
Pointer to an output `struct pwrfig_node_report`.

Return Value

If successful, `pwrfig_node_report_get()` returns 0.

If unsuccessful, `pwrfig_node_report_get()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input/output parameters.
- **PWRFIP_ERR_COM_XXX:**
Communication error with the coprocessor.

Example

```
static const char *pwrfig_node_state_str[_PWRFIP_NODE_STATE_MAX] = {
    /* PWRFIP_NODE_STATE_INITIAL */
    "intial",
    /* PWRFIP_NODE_STATE_LOADED */
    "loaded",
    /* PWRFIP_NODE_STATE_READY */
    "ready",
    /* PWRFIP_NODE_STATE_RUNNING */
    "running",
};

static const char *pwrfig_ba_state_str[_PWRFIP_BA_STATE_MAX] = {
    /* PWRFIP_BA_STATE_INITIAL */
    "intial",
    /* PWRFIP_BA_STATE_READY */
    "ready",
};
```

```

    "ready",
    /* PWRFIP_BA_STATE_STARTING */
    "starting",
    /* PWRFIP_BA_STATE_IDLE */
    "idle",
    /* PWRFIP_BA_STATE_RUNNING */
    "running",
};

static const char *pwrfig_node_op_str[_PWRFIP_NODE_OP_MAX] = {
    /* _PWRFIP_NODE_OP_UNKNOWN */
    "unknown",
    /* PWRFIP_NODE_OP_WAIT_RX_RP_FRM */
    "rx rp frame",
    /* PWRFIP_NODE_OP_WAIT_TX_RP_FRM */
    "tx rp frame",
    /* PWRFIP_NODE_OP_WAIT_RX_ID_FRM */
    "rx id frame",
    /* PWRFIP_NODE_OP_WAIT_TX_ID_FRM */
    "tx id frame",
};

static const char *pwrfig_ba_wind_str[_PWRFIP_BA_WIND_TYPE_MAX] = {
    /* _PWRFIP_BA_WIND_TYPE_NONE */
    "none",
    /* PWRFIP_BA_WIND_PER */
    "periodic",
    /* PWRFIP_BA_WIND_APER_VAR */
    "aper. var",
    /* PWRFIP_BA_WIND_APER_MSG */
    "aper. msg",
    /* PWRFIP_BA_WIND_WAIT */
    "wait",
};

int main(int argc, char *argv[])
{
    int err = 0;
    struct pwrfig_node *node;
    struct pwrfig_node_report n_report; /* full report */
    uint16_t m_state; /* medium state */

    /**
     * Node initialization
     */
    /**...*/

    /**
     * Node startup

```

```

*/
/*...*/

/* get node report */
err = pwrfig_node_report_get(node, &n_report);
if (err) {
    printf("node report getter failed: %s\n", pwrfig_strerror(errno));
    goto end;
}

m_state = n_report.medium_status.state;
printf("*** coprocessor report:\n");
printf("  node_state      : %d (%s)\n", n_report.node_status.state,
    pwrfig_node_state_str[n_report.node_status.state]);
printf("  node_op          : %d (%s)\n", n_report.node_status.op,
    pwrfig_node_op_str[n_report.node_status.op]);
printf("  ba_state         : %d (%s)\n", n_report.ba_status.state,
    pwrfig_ba_state_str[n_report.ba_status.state]);
printf("  ba_window        : %d (%s)\n", n_report.ba_status.window,
    pwrfig_ba_wind_str[n_report.ba_status.window]);
printf("  medium_state     : 0x%04x\n", m_state);
printf("    [channel 1]\n");
printf("      enable       : %s\n",
    (m_state & PWRFIG_MEDIUM_STATE_CH1_VALID) ? "yes": "no");
printf("      tx_err       : %s\n",
    (m_state & PWRFIG_MEDIUM_STATE_CH1_TX_ERROR) ? "yes": "no");
printf("      watchdog     : %s\n",
    (m_state & PWRFIG_MEDIUM_STATE_CH1_WATCHDOG) ? "yes": "no");
printf("    [channel 2]\n");
printf("      enable       : %s\n",
    (m_state & PWRFIG_MEDIUM_STATE_CH2_VALID) ? "yes": "no");
printf("      tx_err       : %s\n",
    (m_state & PWRFIG_MEDIUM_STATE_CH2_TX_ERROR) ? "yes": "no");
printf("      watchdog     : %s\n",
    (m_state & PWRFIG_MEDIUM_STATE_CH2_WATCHDOG) ? "yes": "no");
printf("  tx_err           :\n");
printf("    ok              : %d\n", n_report.tx_err.ok);
printf("    collision        : %d\n", n_report.tx_err.collusion);
printf("    consistency     : %d\n", n_report.tx_err.consistency);
printf("    not_fresh       : %d\n", n_report.tx_err.not_fresh);
printf("  rx_err           :\n");
printf("    ok              : %d\n", n_report.rx_err.ok);
printf("    pre_mis         : %d\n", n_report.rx_err.pre_mis);
printf("    fsd_mis         : %d\n", n_report.rx_err.fsd_mis);
printf("    fsd_unk         : %d\n", n_report.rx_err.fsd_unk);
printf("    fed_mis         : %d\n", n_report.rx_err.fed_mis);
printf("    crc_bad         : %d\n", n_report.rx_err.crc_bad);
printf("    pdu_bad         : %d\n", n_report.rx_err.pdu_bad);
printf("    len_bad         : %d\n", n_report.rx_err.len_bad);

```

```
printf("    not_fresh    : %d\n", n_report.rx_err.not_fresh);
printf("    not_prompt   : %d\n", n_report.rx_err.not_prompt);

/**
 * Other tasks
 */
/* ... */

end:
/**
 * Node exit
 */
/*...*/
return err;
}
```

4.5.5. node_start

Description

Starts the FIP node.

The user's application data (AE/LE) as well as the desired macrocycles - if the node has a master capability - are loaded into the local coprocessor database. Then, the node connects to the FIP network in passive mode (slave agent).



To switch the node to active mode (master agent), use `pwrfig_ba_start()` function after this call.

Prototype

```
int pwrfig_node_start(struct pwrfig_aele *aele,
                     struct pwrfig_ba_mcycle **mcycle_list, int mcycle_cnt)
```

Parameters

- *IN*
 - **aele:**
Pointer to the application context (AE/LE: FIP variables/messages) to be loaded to the coprocessor (opaque structure).
 - **mcycle_list:**
List of pointers to the macrocycles to be loaded (opaque structures).
 - **mcycle_cnt:**
Number of macrocycles to be loaded in the FIP coprocessor.
- *OUT* - None

Return Value

If successful, `pwrfig_node_start()` returns 0.

If unsuccessful, `pwrfig_node_start()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input parameters.
- **PWRFIP_ERR_AELE_NOT_STOP:**
The application context to be loaded is already active elsewhere.
- **PWRFIP_ERR_INVALID_CTX:**
A macrocycle to be loaded do not belong to the same FIP node as the application context (AE/LE).
- **PWRFIP_ERR_COM_XXX:**
Communication error with the coprocessor.

4.5.6. node_stop

Description

Disconnects the FIP node from the network.



Use the `pwrfig_node_start()` function to start a new app session.

No need to reinitialize the node with `pwrfig_node_init()`.

Prototype

```
int pwrfig_node_stop(struct pwrfig_node *node)
```

Parameters

- *IN*
 - **node:**
Pointer to the target FIP node.
See `struct pwrfig_node`.
- *OUT* - None

Return Value

If successful, `pwrfig_node_stop()` returns 0.

If unsuccessful, `pwrfig_node_stop()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input/output parameters.
- **PWRFIP_ERR_COM_XXX:**
Communication error with the coprocessor.

4.6. Variables

4.6.1. var_write

Description

Writes a FIP variable to the local coprocessor database.

Prototype

```
int pwrfig_var_write(struct pwrfig_var *var)
```

Parameters

- *IN*
 - **var:**
Pointer to the target variable to write.
See `struct pwrfig_var`.
- *OUT*
 - **var:**
Variable's updated info.
See `struct pwrfig_var (.error` field).

Return Value

If successful, `pwrfig_var_write()` returns 0.

If unsuccessful, `pwrfig_var_write()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input parameters.
- **PWRFIP_ERR_AELE_NOT_RUN:**
FIP node is not running. It is therefore impossible to query the coprocessor database.
- **PWRFIP_ERR_COM_XXX:**
Communication error with the coprocessor.

Remarks

There are two possible error levels for this operation:

1. The error returned by the `pwrfig_var_write()` function is related to a context error inside the library, or a communication error with the coprocessor.
This type of error is quite critical since it indicates a malfunction of the library.
2. The `.error` field returned in the `struct pwrfig_var` (see `enum pwrfig_var_err_code`) relates directly to the FIP frame state written to the network.
It is an indicator of the quality of the frame written (good or bad freshness).

Example

```
static struct pwrfig_var_cfg prod_var_cfg = {
    .type = PWRFIG_VAR_TYPE_PROD,
    .id = 0x8426,
    .prod.payload_bsz = 12,
    .prod.flags = \
        /* enable prod status */
        PWRFIG_VAR_FLAGS_REFRESH,
    .prod.refreshment_ustime = 16000, /* 16ms */
    .prod.evt_type = PWRFIG_EVT_TYPE_NONE,
    .pwrfig_var_handler = NULL,
};

int main(int argc, char *argv[])
{
    int i, err = 0;
    struct pwrfig_node *node;
    struct pwrfig_aele *al;
    struct pwrfig_var *prod_var;
    uint8_t w_byte = 0;

    /**
     * Node initialization
     */
    /*...*/

    /* create an aele context */
    al = pwrfig_aele_create(node);
    if (!al) {
        printf("aele creation failed: %s\n", pwrfig_strerror(errno));
        err = -1;
        goto end;
    }

    /* create a production variable */
    prod_var = pwrfig_var_create(al, &prod_var_cfg);
    if (!prod_var) {
        printf("production variable creation failed: %s\n", pwrfig_strerror(errno));
        err = -1;
        goto end;
    }

    /* node startup (slave) */
    err = pwrfig_node_start(al, NULL, 0);
    if (err) {
        printf("node startup failed: %s\n", pwrfig_strerror(errno));
        err = -1;
        goto end;
    }
}
```

```
}

/* writing loop */
for(;;) {
    /* update var payload */
    /* for the example, we increment all the bytes of the
       frame by 1 at each write */
    w_byte++;
    memset(prod_var->buffer, w_byte, prod_var->bsz);

    /* write it to fip network */
    if (pwrfig_var_write(prod_var)) {
        printf("w_var[0x%04x] failed: %s\n", prod_var->id,
            pwrfig_strerror(errno));
        /* we consider this error as fatal error; so we
           * stop the test */
        break;
    }

    /* check var state errors */
    if (prod_var->error) {
        printf("w_var[0x%04x] state error: %d\n",
            prod_var->id, prod_var->error);
    }

    usleep(5000); /* 5ms */
}

end:
/**
 * Node exit
 */
/*...*/
return err;
}
```

4.6.2. var_read

Description

Reads a FIP variable from the local coprocessor database.

Prototype

```
int pwrfig_var_read(struct pwrfig_var *var)
```

Parameters

- *IN*
 - **var:**
Pointer to the target variable to read.
See `struct pwrfig_var`.
- *OUT*
 - **var:**
Variable's updated content.

Return Value

If successful, `pwrfig_var_read()` returns 0.

If unsuccessful, `pwrfig_var_read()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input parameters.
- **PWRFIP_ERR_AELE_NOT_RUN:**
FIP node is not running. It is therefore impossible to query the coprocessor database.
- **PWRFIP_ERR_COM_XXX:**
Communication error with the coprocessor.

Remarks

There are two possible error levels for this operation:

1. The error returned by the `pwrfig_var_read()` function is related to a context error inside the library, or a communication error with the coprocessor.
This type of error is quite critical since it indicates a malfunction of the library.
2. The `.error` field returned in the `struct pwrfig_var` (see `enum pwrfig_var_err_code`) relates directly to the FIP frame state read from the network.
It is an indicator of the quality of the frame read (good or bad freshness/promptness), but also an indicator to know if the user configuration matches with the real frame read from the FIP network (length, PDU etc).

Example

```
static struct pwrfig_var_cfg cons_var_cfg = {
```

```
.type = PWRFIP_VAR_TYPE_CONS,
.id = 0x8427,
.cons.payload_bsz = 12,
.cons.flags = \
    /* enable prod status */
    PWRFIP_VAR_FLAGS_REFRESH | \
    /* enable promptness checking */
    PWRFIP_VAR_FLAGS_PROMPT | \
    /* enable pdu + len bytes checking */
    PWRFIP_VAR_FLAGS_CHK_PDU_LEN,
.cons.promptness_ustime = 16000, /* 16ms */
.cons.evt_type = PWRFIP_EVT_TYPE_NONE,
.pwrfip_var_handler = NULL,
};

int main(int argc, char *argv[])
{
    int i, err = 0;
    struct pwrfip_node *node;
    struct pwrfip_aele *al;
    struct pwrfip_var *cons_var;

    /**
     * Node initialization
     */
    /*...*/

    /* create an aele context */
    al = pwrfip_aele_create(node);
    if (!al) {
        printf("aele creation failed: %s\n", pwrfip_strerror(errno));
        err = -1;
        goto end;
    }

    /* create a consumption variable */
    cons_var = pwrfip_var_create(al, &cons_var_cfg);
    if (!cons_var) {
        printf("consumption variable creation failed: %s\n", pwrfip_strerror(errno));
        err = -1;
        goto end;
    }

    /* node startup (slave) */
    err = pwrfip_node_start(al, NULL, 0);
    if (err) {
        printf("node startup failed: %s\n", pwrfip_strerror(errno));
        err = -1;
        goto end;
    }
}
```

```
}

/* reading loop */
for(;;) {
    if (pwrfig_var_read(cons_var)) {
        printf("r_var[0x%04x] failed: %s\n", cons_var->id,
            pwrfig_strerror(errno));
        /* we consider this error as fatal error; so we
         * stop the test */
        break;
    }
    /* print var payload */
    printf("r_var[0x%04x]: ", cons_var->id);
    for (i = 0; i < cons_var->bsz; ++i)
        printf("%02x ", cons_var->buffer[i]);
    printf("\n");

    usleep(5000); /* 5ms */
}

end:
/**
 * Node exit
 */
/*...*/
return err;
}
```

4.6.3. var_evt_set

Description

Dynamically changes - inside the coprocessor database - the event sensitivity for the pointed variable.

Prototype

```
int pwrfig_var_evt_set(struct pwrfig_var *var,  
                      enum pwrfig_evt_type type)
```

Parameters

- *IN*
 - **var:**
Pointer to the variable on which to change the sensitivity to events.
See `struct pwrfig_var`.
 - **type:**
Type of event sensitivity (see `enum pwrfig_evt_type`):
 - `PWRFIG_EVT_TYPE_NONE`:
Never report the transmission/reception of this variable to the user space.
 - `PWRFIG_EVT_TYPE_PERMANENT`:
Always report the transmission/reception of this variable to the user space.
 - `PWRFIG_EVT_TYPE_TEMPORARY`:
Report just once the transmission/reception of this variable to the user space.
- *OUT* - None

Return Value

If successful, `pwrfig_var_evt_set()` returns 0.

If unsuccessful, `pwrfig_var_evt_set()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input parameters.
- **PWRFIG_ERR_AELE_NOT_RUN:**
FIP node is not running. It is therefore impossible to change the variable's event on the fly.
- **PWRFIG_ERR_COM_XXX:**
Communication error with the coprocessor.

4.7. Medium

4.7.1. medium_status_get

Description

Get the status of the FIP channels.

Prototype

```
int pwrfig_medium_status_get(struct pwrfig_node *node,
                             struct pwrfig_medium_status *status)
```

Parameters

- *IN*
 - **node:**
Pointer to the target FIP node.
See `struct pwrfig_node`.
- *OUT*
 - **status:**
Pointer to an output `struct pwrfig_medium_status`.

Return Value

If successful, `pwrfig_medium_status_get()` returns 0.

If unsuccessful, `pwrfig_medium_status_get()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input/output parameters.
- **PWRFIP_ERR_COM_XXX:**
Communication error with the coprocessor.

Example

```
int main(int argc, char *argv[])
{
    int err = 0;
    struct pwrfig_node *node;
    struct pwrfig_medium_status m_status;
    uint16_t m_state;

    /**
     * Node initialization
     */
    /*...*/
```

```
/**
 * Node startup
 */
/*...*/

/* get medium state */
err = pwrfig_medium_status_get(node, &m_status);
if (err) {
    printf("medium status getter failed: %s\n", pwrfig_strerror(errno));
    goto end;
}

m_state = m_status.state;
printf("medium_state : 0x%04x\n", m_state);
printf(" [channel 1]\n");
printf("    enable    : %s\n",
       (m_state & PWRFIG_MEDIUM_STATE_CH1_VALID) ? "yes": "no");
printf("    tx_err     : %s\n",
       (m_state & PWRFIG_MEDIUM_STATE_CH1_TX_ERROR) ? "yes": "no");
printf("    watchdog    : %s\n",
       (m_state & PWRFIG_MEDIUM_STATE_CH1_WATCHDOG) ? "yes": "no");
printf(" [channel 2]\n");
printf("    enable    : %s\n",
       (m_state & PWRFIG_MEDIUM_STATE_CH2_VALID) ? "yes": "no");
printf("    tx_err     : %s\n",
       (m_state & PWRFIG_MEDIUM_STATE_CH2_TX_ERROR) ? "yes": "no");
printf("    watchdog    : %s\n",
       (m_state & PWRFIG_MEDIUM_STATE_CH2_WATCHDOG) ? "yes": "no");

/**
 * Other tasks
 */
/* ... */

end:
/**
 * Node exit
 */
/*...*/
return err;
}
```

4.7.2. medium_cmd_set

Description

Sends a command to the coprocessor to control the FIP channels.
The following operations are allowed for each channel:

- Enable/Disable
- Reset
- Clear error

Prototype

```
int pwrfig_medium_cmd_set(struct pwrfig_node *node, uint16_t flags)
```

Parameters

- *IN*
 - **node:**
Pointer to the target FIP node.
See `struct pwrfig_node`.
 - **flags:**
Command to send to the FIP channels manager.



The available commands are described by the `enum pwrfig_medium_cmd_flag`.

- *OUT* - None

Return Value

If successful, `pwrfig_medium_cmd_set()` returns 0.

If unsuccessful, `pwrfig_medium_cmd_set()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input parameters.
- **PWRFIP_ERR_COM_XXX:**
Communication error with the coprocessor.

4.8. Events

4.8.1. evt_process

Description

Reads and processes all FIP asynchronous events notified by the coprocessor.

Prototype

```
int pwrfig_evt_process(struct pwrfig_node *node)
```

Parameters

- *IN*
 - **node:**
Pointer to the node to treat.
See `struct pwrfig_node`.
- *OUT* - None

Return Value

If successful, `pwrfig_evt_process()` returns 0.

If unsuccessful, `pwrfig_evt_process()` returns -1 and sets `errno` to one of the following values:

- **EINVAL:**
Invalid input parameter.
- **PWRFIP_ERR_COM_XXX:**
Communication error with the coprocessor.

Remarks



Case of PCI/PCIe devices

If a PCI/PCIe PowerFIP device is bound to the node (see `struct pwrfig_node_cfg`, `.dev` field), this function must not be called by the user.

It will be called automatically by an internal interrupt handler of the library.



User handlers called inside this function

- `pwrfig_var_handler`
- `pwrfig_msg_rcv_handler`
- `pwrfig_msg_send_handler`
- `pwrfig_urg_req_handler`
- `pwrfig_nor_req_handler`
- `pwrfig_ba_state_handler`

Chapter 5. Structures

5.1. General

5.1.1. date

Description

Date structure.

Definition

```
struct pwrfig_date {  
    union {  
        uint32_t raw;  
        struct {  
            uint8_t day;  
            uint8_t month;  
            uint8_t year;  
            uint8_t reserved;  
        } info;  
    };  
};
```

Members

Name	Type	Description
day	uint8_t	Day of the date.
month	uint8_t	Month of the date.
year	uint8_t	Year of the date.

5.1.2. version

Description

Version structure.

Definition

```
struct pwrfig_version {  
    union {  
        uint32_t raw;  
        struct {  
            uint8_t patch;  
            uint8_t minor;  
            uint8_t major;  
            uint8_t reserved;  
        } info;  
    };  
    /* build date */  
    struct pwrfig_date date;  
};
```

Members

Name	Type	Description
patch	uint8_t	Patch version Marks bug fixes
minor	uint8_t	Minor version New features without break of compatibility
major	uint8_t	Major version Break of compatibility (ex: API changes)
date	struct pwrfig_date	Build date See struct pwrfig_date

5.2. Device

5.2.1. dev_infos

Description

System information about the PowerFIP board.

Definition

```
struct pwrfig_dev_infos {
    uint32_t    index;
    /* device general info */
    uint16_t    vid;
    uint16_t    did;
    uint16_t    ssvid;
    uint16_t    ssdid;
    /* physical BAR addresses */
    int         bar_cnt;
    uintptr_t   bar_bsz[BAR_MAX];
    uintptr_t   bar_base[BAR_MAX];
    /* irq infos */
    int         irq_number;
    /* driver infos */
    uint32_t    drv_version;
    uint32_t    drv_date;
};
```

Members

Name	Type	Description
index	uint32_t	Device index
vid	uint16_t	Vendor ID
did	uint16_t	Device ID
ssvid	uint16_t	Subsystem Vendor ID
ssdid	uint16_t	Subsystem Device ID
bar_cnt	int	Number of BAR (Base Address Register) for this device
bar_bsz	uintptr_t[BAR_MAX]	Size of the BAR areas
bar_base	uintptr_t[BAR_MAX]	Memory base start address for each area.
irq_number	int	IRQ number
drv_version	uint32_t	Driver version
drv_date	uint32_t	Driver built date

5.2.2. dev_report

Description

Global report of the PowerFIP board.

Definition

```
struct pwr_fip_dev_report {  
    float temperature;  
};
```

Members

Name	Type	Description
temperature	float	Temperature sensor (°C)

5.3. Configuration

5.3.1. ba_request

Description

Structure used to set-up an ID_DAT/ID_MSG request during a macrocycle periodic window.

Definition

```
struct pwrfip_ba_request {  
    enum pwrfip_ba_id_type type;  
    uint16_t id;  
};
```

Members

Name	Type	Description
type	enum pwrfip_ba_id_type	Frame control code for the request See enum pwrfip_ba_id_type
id	uint16_t	FIP identifier to request

5.3.2. ba_aper_msg_wind_cfg

Description

Structure used to set-up a macrocycle aperiodic message window.

Definition

```
struct pwrfig_ba_aper_msg_wind_cfg {  
    uint32_t end_ustime;  
};
```

Members

Name	Type	Description
end_ustime	uint32_t	End of aperiodic message window in microseconds (relative to macrocycle start)

5.3.3. ba_aper_var_wind_cfg

Description

Structure used to set-up a macrocycle aperiodic variable window.

Definition

```
struct pwrfig_ba_aper_var_wind_cfg {  
    uint32_t end_ustime;  
    int enable_testp;  
};
```

Members

Name	Type	Description
end_ustime	uint32_t	End of aperiodic variable window in microseconds (relative to macrocycle start)
enable_testp	int	Enable the presence test if the bus arbiter doesn't have other tasks to perform during this window

5.3.4. ba_per_wind_cfg

Description

Structure used to set-up a macrocycle periodic window.

Definition

```
struct pwrfig_ba_per_wind_cfg {  
    int req_cnt;  
    struct pwrfig_ba_request *req_list;  
};
```

Members

Name	Type	Description
req_cnt	int	Number of periodic request
req_list	struct pwrfig_ba_request *	List of periodic requests See struct pwrfig_ba_request

5.3.5. ba_wait_wind_cfg


Description

Structure used to set-up a macrocycle wait window.

Definition

```
struct pwrfip_ba_wait_wind_cfg {
    uint32_t end_ustime;
    int is_silent;
    int is_ext_resync;
};
```

Members

Name	Type	Description
end_ustime	<code>uint32_t</code>	End of waiting window in microseconds (relative to the macrocycle start time)
is_silent	<code>int</code>	<p>Waiting type</p> <ul style="list-style-type: none"> 0: Padding frame emission 1: No frame during waiting <div>  <p>In a context where several bus arbiters are waiting on the FIP network, it is not recommended to use the <i>silent</i> mode to avoid conflicts between potential masters</p> </div>
is_ext_resync	<code>int</code>	<p>Resynchronization type</p> <ul style="list-style-type: none"> 0: internal The coprocessor itself initiates the looping of the macrocycle. To do this, it uses its own internal timer 1: external The coprocessor listens to an external trigger before looping the macrocycle

5.3.6. ba_wind_cfg

Description

Structure used to set-up a macrocycle window for a master FIP node.

Definition

```
struct pwrfig_ba_wind_cfg {
    enum pwrfig_ba_wind_type type;
    union {
        struct pwrfig_ba_per_wind_cfg per;
        struct pwrfig_ba_aper_var_wind_cfg aper_var;
        struct pwrfig_ba_aper_msg_wind_cfg aper_msg;
        struct pwrfig_ba_wait_wind_cfg wait;
    };
};
```

Members

Name	Type	Description
type	enum pwrfig_ba_wind_type	Macrocycle window type See enum pwrfig_ba_wind_type
<ul style="list-style-type: none"> per aper_var aper_msg wait 	union { struct pwrfig_ba_per_wind_cfg per; struct pwrfig_ba_aper_var_wind_cfg aper_var; struct pwrfig_ba_aper_msg_wind_cfg aper_msg; struct pwrfig_ba_wait_wind_cfg wait; }	Specialized configuration structure according to the .type field of the macrocycle window See: struct pwrfig_ba_per_wind_cfg struct pwrfig_ba_aper_var_wind_cfg struct pwrfig_ba_aper_msg_wind_cfg struct pwrfig_ba_wait_wind_cfg

5.3.7. ba_mcycle_cfg

Description

Structure used to set-up a macrocycle for a master FIP node.

Definition

```
struct pwrfig_ba_mcycle_cfg {  
    int wind_cnt;  
    struct pwrfig_ba_wind_cfg *wind_list;  
};
```

Members

Name	Type	Description
wind_cnt	int	Number of macrocycle window configurations
req_list	struct pwrfig_ba_wind_cfg *	List of window configurations See struct pwrfig_ba_wind_cfg

5.3.8. ba_startup_cfg


Description

This structure is used to set start-up and election times of a master node (Bus Arbiter).

Definition

```
struct pwrfig_ba_startup_cfg {
    enum pwrfig_ba_startup_mode mode;
    uint32_t silence_ustime;
    uint8_t max_phy_addr;
    uint8_t max_prio;
    uint8_t my_phy_addr;
    uint8_t my_prio;
};
```

Members

Name	Type	Description
mode	<code>enum pwrfig_ba_startup_mode</code>	Calculation method for bus arbiter startup and election times. See <code>enum pwrfig_ba_startup_mode</code>
silence_ustime	<code>uint32_t</code>	Silence time in microseconds.
max_phy_addr	<code>uint8_t</code>	Last FIP agent address with master capability on network.
max_prio	<code>uint8_t</code>	Highest BA priority on the network. Should be in [0..15] range. <div>  0 is the highest priority. </div>
my_phy_addr	<code>uint8_t</code>	Physical address of this BA.
my_prio	<code>uint8_t</code>	Priority of this BA.

5.3.9. node_ba_cfg

Description


Bus Arbiter configuration structure for a FIP node.


Definition

```
struct pwrfip_node_ba_cfg {
    int enable;
    /**
     * => BA: StartUp/Election settings
     */
    uint32_t start_ustime;
    uint32_t election_ustime;
    /**
     * => BA: Aperiodic requests FIFOs settings
     */
    uint32_t msg_req_fifo_size;
    uint32_t urgent_var_req_fifo_size;
    uint32_t normal_var_req_fifo_size;
};
```

Members

Name	Type	Description
enable	int	Enable/Disable bus arbiter (master) capability. If disabled, all other structure members are non significant.

Name	Type	Description
start_ustime	uint32_t	<p>Bus arbiter start-up time in microseconds.</p> <div><p>When the user calls the <code>pwrfig_ba_start()</code> function, the node initiates its bus arbiter start procedure with settings of the start-up timeout. Before the end of the start-up countdown, if an activity is detected on the network, the bus arbiter enter in <i>idle</i> mode and considers itself as potentially eligible. At the start-up timeout, having detected no activity on the network, the bus arbiter sends three padding identifiers on the line to check its ability to transmit. If faults are detected, the BA reports the anomaly to the user (event code: PWRFIG_EVT_BA_STOP_ER) and switches to <i>stopped</i> status. Otherwise it enters <i>idle</i> status, and starts its election countdown.</p></div>

Name	Type	Description
election_ustime	uint32_t	<p>Bus arbiter election time in microseconds.</p> <div>  <p>Several bus arbiters can be started on the FIP network, but only one is <i>active</i> at a time (the others are in <i>idle</i> mode). This time is set differently for each potential bus arbiter. When the active bus arbiter fails (no traffic on the line), the other arbiters start their <i>election</i> countdown. The first one to arrive at the end of the countdown becomes <i>active</i> (master) and the other nodes are put to <i>idle</i> mode.</p> </div>
msg_req_fifo_size	uint32_t	Bus arbiter queue size for aperiodic message requests.
urgent_var_req_fifo_size	uint32_t	Bus arbiter queue size for aperiodic variable requests (urgent priority).
normal_var_req_fifo_size	uint32_t	Bus arbiter queue size for aperiodic variable requests (normal priority).

5.3.10. node_cfg

Description

Node configuration structure.

Definition

```
struct pwrfig_node_cfg {  
    /**  
     * General FIP settings  
     */  
    uint8_t fip_phy_addr;  
    uint8_t fip_seg_num;  
    uint8_t fip_frm_type;  
    uint8_t fip_bitrate;  
    uint32_t turn_around_ustime;  
    uint32_t silence_ustime;  
    /**  
     * Extra FIP settings  
     */  
    int enable_bimedium;  
    struct pwrfig_node_msg_cfg msg;  
    struct pwrfig_node_ba_cfg ba;  
    /**  
     * Hardware access  
     */  
    struct pwrfig_dev *dev;  
    unsigned long dpm_base_addr;  
    /**  
     * General Handlers  
     */  
    void *user_ctx;  
    void (* pwrfig_var_handler)(  
        struct pwrfig_node *node,  
        struct pwrfig_var *var,  
        struct pwrfig_event *evt);  
    void (* pwrfig_msg_rcv_handler)(  
        struct pwrfig_node *node,  
        struct pwrfig_msg *msg,  
        struct pwrfig_event *evt);  
    void (* pwrfig_msg_send_handler)(  
        struct pwrfig_node *node,  
        struct pwrfig_msg *msg,  
        struct pwrfig_event *evt);  
    void (* pwrfig_urg_req_handler)(  
        struct pwrfig_node *node,  
        struct pwrfig_event *evt);  
    void (* pwrfig_nor_req_handler)(  
        struct pwrfig_node *node,
```


```

    struct pwrfig_event *evt);
void (* pwrfig_ba_state_handler)(
    struct pwrfig_node *node,
    struct pwrfig_event *evt);
void (* pwrfig_error_handler)(
    struct pwrfig_node *node,
    enum pwrfig_error_code error);
void (* pwrfig_reset_handler)(
    struct pwrfig_node *node);
};

```

Members

Name	Type	Description
fip_phy_addr	uint8_t	Physical address of the agent
fip_seg_num	uint8_t	FIP segment number to which the agent belongs
fip_frm_type	uint8_t	FIP frame type See enum pwrfig_frame_type
fip_bitrate	uint8_t	FIP bitrate See enum pwrfig_bitrate

Name	Type	Description
turn_around_ustime	uint32_t	<p>Turn-Around time in microseconds (0=default time)</p> <div>  <div> <p>Time range according to bitrate</p> <ul style="list-style-type: none"> • 31.25Kbps <ul style="list-style-type: none"> ◦ min=424us ◦ max=2240us ◦ default=424us • 1Mbps <ul style="list-style-type: none"> ◦ min=10us ◦ max=70us ◦ default=30us • 2.5Mbps <ul style="list-style-type: none"> ◦ min=14us ◦ max=40us ◦ default=14us • 5Mbps <ul style="list-style-type: none"> ◦ min=32us ◦ max=48us ◦ default=32us </div> </div>

Name	Type	Description
silence_ustime	<code>uint32_t</code>	<p>Silence time in microseconds (0=default time)</p> <div>  <p>Time range according to bitrate</p> <ul style="list-style-type: none"> • 31.25Kbps <ul style="list-style-type: none"> ◦ min=4096us ◦ max=64512us ◦ default=4096us • 1Mbps <ul style="list-style-type: none"> ◦ min=70us ◦ max=2056us ◦ default=150us • 2.5Mbps <ul style="list-style-type: none"> ◦ min=96us ◦ max=838us ◦ default=96us • 5Mbps <ul style="list-style-type: none"> ◦ min=92us ◦ max=252us ◦ default=92us </div>
enable_bimedium	<code>int</code>	Enable/Disable Medium redundancy (eq. <i>fieldual</i>)
msg	<code>struct pwrfig_node_msg_cfg</code>	Messaging capability. <i>[Optional]</i> See <code>struct pwrfig_node_msg_cfg</code>
ba	<code>struct pwrfig_node_ba_cfg</code>	Master (Bus Arbiter) capability. <i>[Optional]</i> See <code>struct pwrfig_node_ba_cfg</code>
dev	<code>struct pwrfig_dev *</code>	PCI/PCIe device to bind with node. Opaque structure created by the <code>pwrfig_device_open()</code> function.
dpm_base_addr	<code>unsigned long</code>	Dual-port memory area start address
user_ctx	<code>void *</code>	User context pointer <i>[Optional]</i>

Name	Type	Description
pwrfip_var_handler	<pre>void (* handler) (struct pwrfip_node *node, struct pwrfip_var *var, struct pwrfip_event *evt)</pre>	General variable handler (Node Level) <i>[Optional]</i> See: struct pwrfip_node struct pwrfip_var struct pwrfip_event
pwrfip_msg_rcv_handler	<pre>void (* handler) (struct pwrfip_node *node, struct pwrfip_msg *msg, struct pwrfip_event *evt)</pre>	Message reception handler <i>[Optional]</i> See: struct pwrfip_node struct pwrfip_msg struct pwrfip_event
pwrfip_msg_send_handler	<pre>void (* handler) (struct pwrfip_node *node, struct pwrfip_msg *msg, struct pwrfip_event *evt)</pre>	Message emission handler <i>[Optional]</i> See: struct pwrfip_node struct pwrfip_msg struct pwrfip_event
pwrfip_urg_req_handler	<pre>void (* handler) (struct pwrfip_node *node, struct pwrfip_event *evt)</pre>	Urgent aperiodic variable transmission request handler <i>[Optional]</i> See: struct pwrfip_node struct pwrfip_event
pwrfip_nor_req_handler	<pre>void (* handler) (struct pwrfip_node *node, struct pwrfip_event *evt)</pre>	Normal aperiodic variable transmission request handler <i>[Optional]</i> See: struct pwrfip_node struct pwrfip_event
pwrfip_ba_state_handler	<pre>void (* handler) (struct pwrfip_node *node, struct pwrfip_event *evt)</pre>	Bus arbiter state event handler <i>[Optional]</i> See: struct pwrfip_node struct pwrfip_event
pwrfip_error_handler	<pre>void (* handler) (struct pwrfip_node *node, enum pwrfip_error_code error)</pre>	Internal library error handler [Mandatory] See: struct pwrfip_node enum pwrfip_error_code

Name	Type	Description
pwrfig_reset_handler	<pre>void (* handler) (struct pwrfig_node *node)</pre>	Reset chip procedure handler [Mandatory] See <code>struct pwrfig_node</code>

Remarks



Hardware access

Two methods are provided to bind the FIP component with the library:

1. Use the `.dpm_base_addr` field to manually configure the PowerFIP chip DPM (Dual-Port Memory) address access.
2. Use the `.dev` field to attach a PCI/PCIe device.

Note: In this case no need to configure `.dpm_base_addr` field.

5.3.11. node_msg_cfg

Description

Messaging configuration structure for a FIP node.

Definition

```
#define PWRFIP_MAX_TX_MSG_FIFO_COUNT 9

struct pwrfip_node_msg_cfg {
    int enable;
    /**
     * => Reception settings
     */
    uint32_t rx_fifo_size;
    uint8_t rx_segment_tab[256];
    /**
     * => Emission settings
     */
    uint32_t tx_fifo_size[PWRFIP_MAX_TX_MSG_FIFO_COUNT];
    uint8_t tx_max_repeat;
};
```

Members

Name	Type	Description
enable	int	Enable/Disable FIP messaging. If disabled, all other structure members are non significant.
rx_fifo_size	uint32_t	Consumption queues sizes for FIP messages min=1, max=64, default=30 (0: default value)
rx_segment_tab[256]	uint8_t	Sensitivity of the node for the message reception according to the segment destination. See enum pwrfip_msg_rx_seg_cap
tx_fifo_size[9]	uint32_t	Transmission queues sizes for FIP messages * index 0: aper. msg queue * index 1-8: per. msg queues min=1, max=64; default=24 (0: default value)
tx_max_repeat	uint8_t	Maximum number of repeats for acknowledged send messaging.

5.3.12. var_cons_cfg

Description

Specific set-up structure for a consumption FIP variable.

Definition

```
struct pwr_fip_var_cons_cfg {  
    uint16_t payload_bsz;  
    uint16_t flags;  
    uint32_t promptness_ustime;  
    uint32_t evt_type;  
};
```

Members

Name	Type	Description
payload_bsz	uint16_t	User data payload in bytes
flags	uint16_t	Set-up flags for consumption. See enum <code>pwr_fip_var_flags</code>
promptness_ustime	uint32_t	Promptness period in microseconds
evt_type	uint32_t	Set-up flags for event (Enable/Disable + Lifetime). See enum <code>pwr_fip_evt_type</code>

5.3.13. var_prod_cfg

Description

Specific set-up structure for a production FIP variable.

Definition

```
struct pwr_fip_var_prod_cfg {  
    uint16_t payload_bsz;  
    uint16_t flags;  
    uint32_t refreshment_ustime;  
    uint32_t evt_type;  
};
```

Members

Name	Type	Description
payload_bsz	uint16_t	User data payload in bytes
flags	uint16_t	Set-up flags for production. See enum <code>pwr_fip_var_flags</code>
refreshment_ustime	uint32_t	Refreshment period in microseconds
evt_type	uint32_t	Set-up flags for event (Enable/Disable + Lifetime). See enum <code>pwr_fip_evt_type</code>

5.3.14. var_sync_cfg

Description

Specific set-up structure for a synchronization FIP variable.

Definition

```
struct pwrfig_var_sync_cfg {  
    uint32_t reserved[3];  
};
```

Members

Name	Type	Description
reserved	uint32_t[3]	Unused parameters

Remarks

A **synchronization** data has no user payload attached to it (no *RP_DAT* frame on network).

Only a **FIP identifier** and a **permanent event** are set-up.

When the *ID_DAT* frame associated with this FIP identifier is received - or produced (master) - by the node, an **event** is triggered and the *user handler* is executed.

5.3.15. var_cfg

Description



Generic set-up structure of a FIP variable for a FIP node.

Definition

```
struct pwrfig_var_cfg {
    enum pwrfig_var_type type;
    uint16_t id;
    union {
        struct pwrfig_var_cons_cfg cons;
        struct pwrfig_var_prod_cfg prod;
        struct pwrfig_var_sync_cfg sync;
    };
    void (* pwrfig_var_handler) (
        struct pwrfig_node *node,
        struct pwrfig_var *var,
        struct pwrfig_event *evt
    );
};
```

Members

Name	Type	Description
type	<code>enum pwrfig_var_type</code>	Variable type (Production, Consumption or Synchronization) See <code>enum pwrfig_var_type</code> .
id	<code>uint16_t</code>	FIP identifier
cons/prod/sync	<pre>union { struct pwrfig_var_cons_cfg cons; struct pwrfig_var_prod_cfg prod; struct pwrfig_var_sync_cfg sync; }</pre>	Specialized configuration structure according to the <code>.type</code> field of the variable. See: <pre>struct pwrfig_var_cons_cfg struct pwrfig_var_prod_cfg struct pwrfig_var_sync_cfg</pre>

Name	Type	Description
pwrfig_var_handler	<pre>void (* handler) (struct pwrfig_node *node, struct pwrfig_var *var, struct pwrfig_event *evt)</pre>	<p>Local handler definition for a <i>synchronization</i> variable.</p> <p>Or, if <code>.evt_type</code> field is set, for a <i>production/consumption</i> variable.</p> <div>  <p>If <code>pwrfig_var_handler=NULL</code>, the general handler, with the same name and located in the <code>node_cfg</code> structure, will take over in case of an event.</p> </div> <div>  <p>If neither - local or general - handler is set, the user will not be notified of the variable events.</p> </div> <p>See:</p> <pre>struct pwrfig_node struct pwrfig_var struct pwrfig_event</pre>

5.4. Objects

5.4.1. node

Description

FIP Node object.

This is the handle structure to interact with a FIP node.

Definition

```
struct pwrfip_node {  
    struct pwrfip_node_infos *infos;  
    void *priv;  
};
```

Members

Name	Type	Description
infos	<code>struct pwrfip_node_infos *</code>	Pointer to the FIP node information See <code>struct pwrfip_node_infos</code>
priv	<code>void *</code>	Pointer to a reserved opaque structure

5.4.2. var

Description

FIP Variable object.

This is the handle structure to interact with a FIP variable.

Definition

```
struct pwrfig_var {  
    uint16_t id;  
    uint16_t bsz;  
    uint8_t *buffer;  
    uint8_t error;  
    void *priv;  
};
```

Members

Name	Type	Description
id	uint16_t	FIP identifier of the variable
bsz	uint16_t	Useful variable byte size
buffer	uint8_t *	Pointer to the useful variable data.
error	uint8_t	Error code for the read/write operation on the variable (eq. var_state). This report is generated directly by the coprocessor according to the state of the variable in its local database (freshness, promptness, etc). See enum pwrfig_var_err_code
priv	void *	Pointer to a reserved opaque structure

5.5. Infos/Status/Report

5.5.1. ba_status

Description

Bus Arbiter status of FIP coprocessor.

Definition

```
struct pwrfip_ba_status {  
    uint16_t state;  
    uint16_t window;  
    uint32_t index;  
};
```

Members

Name	Type	Description
state	uint16_t	Bus arbiter state (FSM: Finite State Machine) See enum <code>pwrfip_ba_state</code>
window	uint16_t	Macrocycle window currently active See enum <code>pwrfip_ba_wind_type</code>
index	uint32_t	Macrocycle index currently active

5.5.2. medium_status

Description

Medium (Channels) status of FIP coprocessor.

Definition

```
struct pwr_fip_medium_status {  
    uint16_t state;  
    uint16_t reserved;  
};
```

Members

Name	Type	Description
state	uint16_t	Medium state. See enum <code>pwr_fip_medium_state</code> .

5.5.3. node_infos

Description

Structure containing information about the FIP node (node configuration, software versions/dates).

Definition

```
struct pwrfig_node_infos {  
    struct pwrfig_node_cfg cfg;  
    uint64_t cpu_id;  
    struct pwrfig_version gw_version;  
    struct pwrfig_version fw_version;  
    struct pwrfig_version drv_version;  
    struct pwrfig_version lib_version;  
};
```

Members

Name	Type	Description
cfg	<code>struct pwrfig_node_cfg</code>	User configuration attached to the node See <code>struct pwrfig_node_cfg</code>
cpu_id	<code>uint64_t</code>	Coprocessor unique identifier
gw_version	<code>struct pwrfig_version</code>	Coprocessor gateware version (FPGA) See <code>struct pwrfig_version</code>
fw_version	<code>struct pwrfig_version</code>	Coprocessor firmware version (C binary) <i>powerfig-firmware.bin</i> See <code>struct pwrfig_version</code>
drv_version	<code>struct pwrfig_version</code>	Driver version See <code>struct pwrfig_version</code>
lib_version	<code>struct pwrfig_version</code>	Library version See <code>struct pwrfig_version</code>

5.5.4. node_report

Description

Global report of the FIP coprocessor.

Definition

```
struct pwrfig_node_report {  
    struct pwrfig_node_status node_status;  
    struct pwrfig_ba_status ba_status;  
    struct pwrfig_medium_status medium_status;  
    /* stats */  
    struct pwrfig_tx_err tx_err;  
    struct pwrfig_rx_err rx_err;  
};
```

Members

Name	Type	Description
node_status	<code>struct pwrfig_node_status</code>	Node FSM and operation status See <code>struct pwrfig_node_status</code>
ba_status	<code>struct pwrfig_ba_status</code>	Bus arbiter FSM and window status See <code>struct pwrfig_ba_status</code>
medium_status	<code>struct pwrfig_medium_status</code>	Medium (Channels) Status See <code>struct pwrfig_medium_status</code>
tx_err	<code>struct pwrfig_tx_err</code>	Transmission errors statistics See <code>struct pwrfig_tx_err</code>
rx_err	<code>struct pwrfig_rx_err</code>	Reception errors statistics See <code>struct pwrfig_rx_err</code>

5.5.5. node_status

Description

General status of FIP node.

Definition

```
struct pwrfig_node_status {  
    uint16_t state;  
    uint16_t op;  
};
```

Members

Name	Type	Description
state	uint16_t	Node state (FSM: Finite State Machine) See enum <code>pwrfig_node_state</code>
op	uint16_t	Operation currently in progress See enum <code>pwrfig_node_operation</code>

5.5.6. rx_err

Description

Report of RX error by the coprocessor.

Definition

```
struct pwrfip_rx_err {  
    /**  
     * MAU frame errors  
     */  
    /* no mau rx error occurs */  
    uint64_t ok;  
    /* -> Physical Layer errors */  
    uint64_t pre_mis;  
    uint64_t fsd_mis;  
    uint64_t fsd_unk;  
    uint64_t fed_mis;  
    /* -> Data-Link Layer errors */  
    uint64_t crc_bad;  
    uint64_t reserved[10];  
    /**  
     * Application layer errors  
     */  
    uint64_t pdu_bad;  
    uint64_t len_bad;  
    uint64_t not_fresh;  
    uint64_t not_prompt;  
};
```

Members

Name	Type	Description
ok	uint64_t	Frame Ok. No MAU TX error occurs
pre_mis	uint64_t	Missing preamble (glitches on line)
fsd_mis	uint64_t	Missing Frame Start Delimiter (FSD)
fsd_unk	uint64_t	Unknown Frame Start Delimiter (FSD)
fed_mis	uint64_t	Missing Frame End Delimiter (FED)
crc_bad	uint64_t	Wrong Cyclic Redundancy Check (CRC)
reserved	uint64_t[10]	Internal errors (not documented)
pdu_bad	uint64_t	Unknown FIP Protocol Data Unit (PDU)
len_bad	uint64_t	Bad FIP frame length (LEN)

Name	Type	Description
not_fresh	uint64_t	Consumed variable has not been correctly refreshed by a remote fip node (agent)
not_prompt	uint64_t	Consumed variable is not prompt. Local database hasn't been read (by the user) during promptness period

5.5.7. tx_err

Description

Report of TX error by the coprocessor.

Definition

```
struct pwrfig_tx_err {  
    /**  
     * Medium Attachment Unit (MAU) errors  
     */  
    uint64_t ok;  
    uint64_t collision;  
    uint64_t consistency;  
    uint64_t reserved[5];  
    /**  
     * Application layer errors  
     */  
    uint64_t not_fresh;  
};
```

Members

Name	Type	Description
ok	uint64_t	Frame Ok. No MAU TX error occurs
collision	uint64_t	Other frame present on network during transmission
consistency	uint64_t	The coprocessor listens to the FIP line at the time of its own transmission. This counter is incremented if a consistency problem is detected between the data sent and the data read back
reserved	uint64_t[5]	Internal errors (not documented)
not_fresh	uint64_t	Produced variable is not correctly refreshed inside local database (refreshment period has expired)

5.6. Event

5.6.1. event

Description

FIP event.

Definition

```
struct pwrfig_event {  
    uint64_t epoch;  
    uint16_t code;  
    uint16_t param;  
};
```

Members

Name	Type	Description
epoch	uint64_t	Event epoch (10ns)
code	uint16_t	Event Code See enum pwrfig_event_code
param	uint16_t	Event Parameter. This parameter is significant only for some event codes. For more information, see enum pwrfig_event_code .

Chapter 6. Enumerations

6.1. aper_var_channel_type

Description

Priority level for the channel dedicated to aperiodic variable requests.

Definition

```
enum pwrfig_aper_var_channel_type {  
    PWRFIG_APER_VAR_NORMAL_CH = 0,  
    PWRFIG_APER_VAR_URGENT_CH,  
};
```

Values

Constant	Value	Description
PWRFIG_BA_ID_DAT	0	Normal priority.
PWRFIG_BA_ID_MSG	1	Urgent priority.

6.2. ba_id_type

Description

ID request type allowed inside a macrocycle periodic window (for a master node only).

Definition

```
enum pwrfig_ba_id_type {  
    PWRFIG_BA_ID_DAT = 0x03,  
    PWRFIG_BA_ID_MSG = 0x05,  
};
```

Values

Constant	Value	Description
PWRFIG_BA_ID_DAT	0x03	Bus arbiter request for a FIP variable.
PWRFIG_BA_ID_MSG	0x05	Bus arbiter request for a FIP message.

6.3. ba_startup_mode

Description

Start-up mode of the bus arbiter.

In a context where several bus arbiters are competing on the network to become master, it is necessary to give election priorities for each of them.

This enumeration defines these calculation methods (or mode) allowing to set-up the start-up/election times of the bus arbiter according to the characteristics of the FIP node.



Bus Arbiter 's Start-up and Election times

- BA_StartUp_Time (microseconds) = $T0 \times BA_StartUp_Par$
- BA_Election_Time (microseconds) = $T0 \times BA_Election_Par$


Where T0 is the Silence Time in microseconds.

Definition

```
enum pwrfig_ba_startup_mode {
    PWRFIG_BA_STUP_STANDARD,
    PWRFIG_BA_STUP_OPTIMIZE_1,
    PWRFIG_BA_STUP_OPTIMIZE_2,
    PWRFIG_BA_STUP_OPTIMIZE_3,
};
```

Values

Constant	Value	Description
PWRFIG_BA_STUP_STANDARD	0	<ul style="list-style-type: none"> • BA_StartUp_Par = 8712 • BA_Election_Par = $2 \times [256 \times (BA_Priority + 1) + Node_Phy_Addr + 3]$
PWRFIG_BA_STUP_OPTIMIZE_1	1	<ul style="list-style-type: none"> • BA_StartUp_Par = $2 \times [(BA_Max_Phy_Addr + 1) \times 16 + 257 + 3]$ • BA_Election_Par = $2 \times [(BA_Max_Phy_Addr + 1) \times (BA_Priority + 1) + Node_Phy_Addr + 3]$
PWRFIG_BA_STUP_OPTIMIZE_2	2	<ul style="list-style-type: none"> • BA_StartUp_Par = $2 \times [(BA_Max_Phy_Addr + 1) \times (BA_Max_Priority + 1) + 257 + 3]$ • BA_Election_Par = $2 \times [(BA_Max_Phy_Addr + 1) \times (BA_Priority + 1) + Node_Phy_Addr + 3]$

Constant	Value	Description
PWRFIP_BA_STUP_OPTIMIZE_3	3	<ul style="list-style-type: none"> • $BA_StartUp_Par = 2 \times [(BA_Max_Phy_Addr + 1) \times (BA_Max_Priority + 1) + 3]$ • $BA_Election_Par = 2 \times [(BA_Max_Phy_Addr + 1) \times (BA_Priority + 1) + Node_Phy_Addr + 3]$ <div>  Only for mono-medium topology. </div>

Remarks

These times can be automatically calculated with `pwrfig_ba_startup_calculate()` function.

6.4. ba_state

Description

State of the bus arbiter.

Definition

```
enum pwrfig_ba_state {  
    PWRFIG_BA_STATE_INITIAL,  
    PWRFIG_BA_STATE_READY,  
    PWRFIG_BA_STATE_STARTING,  
    PWRFIG_BA_STATE_IDLE,  
    PWRFIG_BA_STATE_RUNNING,  
    _PWRFIG_BA_STATE_MAX,  
};
```

Values

Constant	Value	Description
PWRFIG_BA_STATE_INITIAL	0	Initial state. No bus arbiter config loaded
PWRFIG_BA_STATE_READY	1	User macrocycles context loaded. Here, FIP node is in stopped state, and is ready to start.
PWRFIG_BA_STATE_STARTING	2	Starting-up state (not yet active). The bus arbiter is inside start-up procedure.
PWRFIG_BA_STATE_IDLE	3	Idle state (maybe another bus arbiter is active on network)
PWRFIG_BA_STATE_RUNNING	4	Running state. Current node is master. Bus arbiter is active on network
_PWRFIG_BA_STATE_MAX	5	Max BA state number

6.5. ba_wind_type

Description

Type of macrocycle window.

Definition

```
enum pwrfig_ba_wind_type {  
    _PWRFIG_BA_WIND_TYPE_NONE,  
    _PWRFIG_BA_WIND_TYPE_MIN,  
    PWRFIG_BA_WIND_PER = _PWRFIG_BA_WIND_TYPE_MIN,  
    PWRFIG_BA_WIND_APER_VAR,  
    PWRFIG_BA_WIND_APER_MSG,  
    PWRFIG_BA_WIND_WAIT,  
    _PWRFIG_BA_WIND_TYPE_MAX,  
};
```

Values

Constant	Value	Description
<code>_PWRFIG_BA_WIND_TYPE_NONE</code>	0	Invalid BA window type
<code>_PWRFIG_BA_WIND_TYPE_MIN</code>	1	Minimal valid type for a bus arbiter window
<code>PWRFIG_BA_WIND_PER</code>	1	Periodic window
<code>PWRFIG_BA_WIND_APER_VAR</code>	2	Aperiodic variable window
<code>PWRFIG_BA_WIND_APER_MSG</code>	3	Aperiodic message window
<code>PWRFIG_BA_WIND_WAIT</code>	4	External/Internal resync waiting window
<code>_PWRFIG_BA_WIND_TYPE_MAX</code>	5	Maximal valid type for a bus arbiter window

6.6. bitrate

Description

FIP bitrate.

Definition

```
enum pwrfig_bitrate {  
    _PWRFIG_BITRATE_MIN = 1,  
    PWRFIG_BITRATE_31K25 = _PWRFIG_BITRATE_MIN,  
    PWRFIG_BITRATE_1M,  
    PWRFIG_BITRATE_2M5,  
    PWRFIG_BITRATE_5M,  
    _PWRFIG_BITRATE_MAX,  
    _PWRFIG_BITRATE_UNKNOWN = 0,  
};
```

Values

Constant	Value	Description
_PWRFIG_BITRATE_MIN	1	Minimum valid enum code for bitrate.
PWRFIG_BITRATE_31K25	1	FIP/WorldFIP bitrate at 31.25Kbps.
PWRFIG_BITRATE_1M	2	FIP/WorldFIP bitrate at 1Mbps.
PWRFIG_BITRATE_2M5	3	FIP/WorldFIP bitrate at 2.5Mbps.
PWRFIG_BITRATE_5M	4	FIP/WorldFIP bitrate at 5Mbps.
_PWRFIG_BITRATE_MAX	5	Maximum enum code for bitrate.
_PWRFIG_BITRATE_UNKNO WN	0	Not supported or unknown FIP bitrate.

6.7. error_code

Description

Library and coprocessor communication error codes.

Definition

```
enum pwrfig_error_code {
    _PWFIP_ERR_CODE_MIN = 256,
    /* library error codes */
    PWFIP_ERR_DEV_ALREADY_BIND = _PWFIP_ERR_CODE_MIN,
    PWFIP_ERR_DEV_IRQ_HANDLER_STARTED,
    PWFIP_ERR_DEV_IRQ_HANDLER_STOPPED,
    PWFIP_ERR_AELE_NOT_STOP,
    PWFIP_ERR_AELE_NOT_RUN,
    PWFIP_ERR_BA_NOT_STOP,
    PWFIP_ERR_BA_NOT_RUN,
    PWFIP_ERR_INVALID_CTX,
    PWFIP_ERR_CFG_VAR_DIR,
    PWFIP_ERR_CFG_MSG_PROD,
    PWFIP_ERR_CFG_MSG_DIR,
    PWFIP_ERR_CFG_VAR_EXIST,
    PWFIP_ERR_MCYCLE_WIND_COUNT,
    PWFIP_ERR_MCYCLE_WIND_UNKNOWN,
    PWFIP_ERR_MCYCLE_WIND_END,
    PWFIP_ERR_MCYCLE_PER_WIND_REQ_COUNT,
    PWFIP_ERR_MCYCLE_PER_WIND_REQ_UNKNOWN,
    PWFIP_ERR_MCYCLE_WIND_TIME_INC,
    PWFIP_ERR_NODE_HANDLER_MISSING,
    PWFIP_ERR_NODE_FRM_TYPE_INVALID,
    PWFIP_ERR_NODE_BITRATE_INVALID,
    PWFIP_ERR_NODE_RX_MSG_FIFO_SZ,
    PWFIP_ERR_NODE_RX_MSG_SEG_CAP,
    PWFIP_ERR_NODE_TX_MSG_FIFO_SZ,
    PWFIP_ERR_NODE_TX_MSG_REPEAT,
    PWFIP_ERR_NODE_BA_STUP_TIMES,
    PWFIP_ERR_NODE_BA_REQ_FIFO_SZ,
    PWFIP_ERR_NODE_TR_INVALID,
    PWFIP_ERR_NODE_TS_INVALID,
    PWFIP_ERR_BA_STUP_TS_INVALID,
    PWFIP_ERR_BA_STUP_PHY_ADDR_INVALID,
    PWFIP_ERR_BA_STUP_PRIO_INVALID,
    /* communication errors codes (mailboxes) */
    _PWFIP_ERR_COM_CODE_OFFSET,
    PWFIP_ERR_COM_DIR_UNKNOWN = _PWFIP_ERR_COM_CODE_OFFSET,
    PWFIP_ERR_COM_NOT_OUTBOX,
    PWFIP_ERR_COM_NOT_INBOX,
    PWFIP_ERR_COM_INVAL,
    PWFIP_ERR_COM_TMO,
```

```

PWRFIP_ERR_COM_BUSY,
PWRFIP_ERR_COM_NO_PKT,
PWRFIP_ERR_COM_PKT_BAD_SZ,
PWRFIP_ERR_COM_PKT_BAD_CMD,
PWRFIP_ERR_COM_PKT_RES_FAILED,
_PWRFIP_ERR_CODE_MAX,
};

```

Values

Constant	Description
PWRFIP_ERR_DEV_ALREADY_BIND	The device is already bound to another FIP node session.
PWRFIP_ERR_DEV_IRQ_HANDLER_STARTED	IRQ handler is already started.
PWRFIP_ERR_DEV_IRQ_HANDLER_STOPPED	IRQ handler is already stopped.
PWRFIP_ERR_AELE_NOT_STOP	FIP node is currently running.
PWRFIP_ERR_AELE_NOT_RUN	FIP node is currently stopped.
PWRFIP_ERR_BA_NOT_STOP	Macrocycle (BA) is not stopped.
PWRFIP_ERR_BA_NOT_RUN	No BA is currently running.
PWRFIP_ERR_INVALID_CTX	Objects (aele, mcycle...) do not belong to the same node context.
PWRFIP_ERR_CFG_VAR_DIR	Impossible to change the direction of the variable (prod/cons) for this ID.
PWRFIP_ERR_CFG_MSG_PROD	Impossible to link a produced message on this ID. A consumed variable is already attached to it.
PWRFIP_ERR_CFG_MSG_DIR	Impossible to change direction. A produced message is already attached to it.
PWRFIP_ERR_CFG_VAR_EXIST	A variable already exists for this ID.
PWRFIP_ERR_MCYCLE_WIND_COUNT	Invalid macrocycle windows count (should be at least 1).
PWRFIP_ERR_MCYCLE_WIND_UNKNOWN	Unknown macrocycle windows type.
PWRFIP_ERR_MCYCLE_WIND_END	Invalid macrocycle (should end with a WAIT window).
PWRFIP_ERR_MCYCLE_PER_WIND_REQUEST_COUNT	At least one request is required for BA periodic window.
PWRFIP_ERR_MCYCLE_PER_WIND_REQUEST_UNKNOWN	Unknown macrocycle periodic request (ID_DAT or ID_MSG).
PWRFIP_ERR_MCYCLE_WIND_TIME_INC	Overlap on macrocycle windows end times.

Constant	Description
PWRFIP_ERR_NODE_HANDLER_MIS SING	Reset/Error/Warning handlers are mandatory.
PWRFIP_ERR_NODE_FRM_TYPE_INV ALID	Invalid frame type (FIP or WorldFIP).
PWRFIP_ERR_NODE_BITRATE_INVA LID	Invalid node bitrate configuration.
PWRFIP_ERR_NODE_RX_MSG_FIFO_ SZ	Queue size for message consumption should be in [1..64] range.
PWRFIP_ERR_NODE_RX_MSG_SEG_C AP	Segment capability for consumption message should be [0..2].
PWRFIP_ERR_NODE_TX_MSG_FIFO_ SZ	Queue size for message transmission should be in [1..64] range.
PWRFIP_ERR_NODE_TX_MSG_REPEA T	Maximum repeats for acknowledged message transmission should be in [0..3] range.
PWRFIP_ERR_NODE_BA_STUP_TIME S	The election time must be shorter than the start-up time.
PWRFIP_ERR_NODE_BA_REQ_FIFO_S Z	Queue size for BA requests should be in [1..64] range.
PWRFIP_ERR_NODE_TR_INVALID	Invalid FIP turn around time configuration.
PWRFIP_ERR_NODE_TS_INVALID	Invalid FIP silence time configuration.
PWRFIP_ERR_BA_STUP_TS_INVALID	Silence time should not be 0.
PWRFIP_ERR_BA_STUP_PHY_ADDR_I NVALID	Local physical address should not exceed maximum network address.
PWRFIP_ERR_BA_STUP_PRIO_INVAL ID	Local BA priority should not exceed max priority (0: highest prio).
PWRFIP_ERR_COM_DIR_UNKNOW	Unknown direction for the mailbox.
PWRFIP_ERR_COM_NOT_OUTBOX	The mailbox is not configured as output.
PWRFIP_ERR_COM_NOT_INBOX	The mailbox is not configured as input.
PWRFIP_ERR_COM_INVAL	Mailbox invalid argument.
PWRFIP_ERR_COM_TMO	Mailbox timeout.
PWRFIP_ERR_COM_BUSY	Mailbox is busy.
PWRFIP_ERR_COM_NO_PKT	Mailbox has no packet to treat.
PWRFIP_ERR_COM_PKT_BAD_SZ	Incorrect packet size for the mailbox.
PWRFIP_ERR_COM_PKT_BAD_CMD	Unknown packet command id for the mailbox.

Constant	Description
PWRFIP_ERR_COM_PKT_RES_FAILED	Error during response procedure (inbox).

6.8. event_code



Description




FIP event codes of the library.


Definition

```
enum pwrfig_event_code {  
    PWRFIG_EVT_SEND_VAR_P = 0x8100,  
    PWRFIG_EVT_SEND_VAR_T = 0x0100,  
    PWRFIG_EVT_RECEIVE_VAR_P = 0x8200,  
    PWRFIG_EVT_RECEIVE_VAR_T = 0x0200,  
    PWRFIG_EVT_RECEIVE_MSG = 0x0240,  
    PWRFIG_EVT_SEND_MSG = 0x0140,  
    PWRFIG_EVT_SEND_APU = 0x0130,  
    PWRFIG_EVT_SEND_APN = 0x0131,  
    PWRFIG_EVT_BA_ACTIVITY = 0x0400,  
    PWRFIG_EVT_BA_STOP_TMO = 0x0401,  
    PWRFIG_EVT_BA_STOP_ERR = 0x0402,  
    PWRFIG_EVT_BA_STOP_USR = 0x0404,  
    PWRFIG_EVT_BA_IDLE = 0x0408,  
};
```

Values

Constant	Value	Description
PWRFIG_EVT_SEND_VAR_P	0x8100	<p>A variable set-up with <i>permanent</i> event has been sent on the FIP network.</p> <div><p>The variable ID attached to this event code is saved in <code>.param</code> field of <code>struct pwrfig_event</code></p></div>
PWRFIG_EVT_SEND_VAR_T	0x0100	<p>A variable set-up with <i>temporary</i> event (once) has been sent on the FIP network.</p> <div><p>The variable ID attached to this event code is saved in <code>.param</code> field of <code>struct pwrfig_event</code></p></div>

Constant	Value	Description
PWRFIP_EVT_RECEIVE_VA R_P	0x8200	<p>A variable set-up with <i>permanent</i> event has been received from the FIP network.</p> <div>  <p>The variable ID attached to this event code is saved in <code>.param</code> field of <code>struct pwr_fip_event</code></p> </div>
PWRFIP_EVT_RECEIVE_VA R_T	0x0200	<p>A variable set-up with <i>temporary</i> event (once) has been received from the FIP network.</p> <div>  <p>The variable ID attached to this event code is saved in <code>.param</code> field of <code>struct pwr_fip_event</code></p> </div>
PWRFIP_EVT_RECEIVE_MS G	0x0240	A FIP message has been received from the FIP network.
PWRFIP_EVT_SEND_MSG	0x0140	A FIP message has been sent to the FIP network.
PWRFIP_EVT_SEND_APU	0x0130	An <i>urgent</i> aperiodic variable list has been sent to the FIP network.
PWRFIP_EVT_SEND_APN	0x0131	An <i>normal</i> aperiodic variable list has been sent to the FIP network.
PWRFIP_EVT_BA_ACTIVITY	0x0400	The bus arbiter is running. The local node is master.
PWRFIP_EVT_BA_STOP_TM O	0x0401	<p>The bus arbiter has stopped on timeout.</p> <div>  <p>This event occurs when the macrocycle executes a waiting window (with external trigger waiting). If the external signal does not occur within the configured time, this event is emitted and the bus arbiter stops.</p> </div>

Constant	Value	Description
PWRFIP_EVT_BA_STOP_ERR	0x0402	<p>The bus arbiter has stopped on network fault.</p> <div>  <p>This event occurs if the macrocycle executes an unknown or incorrect program instruction code. (see <code>struct pwrfig_ba_wind_cfg</code> - <code>.type</code> field)</p> <p>It can also occur during the start-up phase of the macrocycle, if tx errors are reported during sending of 3 padding frames.</p> </div>
PWRFIP_EVT_BA_STOP_USR	0x0404	The bus arbiter has been stopped by an user command.
PWRFIP_EVT_BA_IDLE	0x0408	The bus arbiter has switched to IDLE mode. Another BA is already active on the network.

6.9. evt_type

Description

FIP event type.

Definition

```
enum pwrfig_evt_type {  
    PWRFIG_EVT_TYPE_NONE,  
    PWRFIG_EVT_TYPE_PERMANENT,  
    PWRFIG_EVT_TYPE_TEMPORARY,  
};
```

Values

Constant	Value	Description
PWRFIG_EVT_TYPE_NONE	0	No event.
PWRFIG_EVT_TYPE_PERMANENT	1	Permanent event. Raises a synchronous event each time it happens.
PWRFIG_EVT_TYPE_TEMPORARY	2	Temporary event. Raises a synchronous event only once at the time it occurs.

6.10. frame_type



Description

Frame type.

Definition

```
enum pwrfig_frame_type {
    _PWRFIG_FRM_TYPE_MIN = 1,
    PWRFIG_FRM_FIP = _PWRFIG_FRM_TYPE_MIN,
    PWRFIG_FRM_WORLDFIP,
    _PWRFIG_FRM_TYPE_MAX,
    _PWRFIG_FRM_TYPE_UNKNOWN = 0,
};
```

Values

Constant	Value	Description
<code>_PWRFIG_FRM_TYPE_MIN</code>	1	Minimum valid enum code for frame type.
<code>PWRFIG_FRM_FIP</code>	1	FIP frame type. <div>  <div> <i>Type of frame delimiters and CRC</i> UTE (Union Technique de l'Electricité) </div> </div>
<code>PWRFIG_FRM_WORLDFIP</code>	2	WorldFIP frame type. <div>  <div> <i>Type of frame delimiters and CRC</i> IEC (International Electrotechnical Commission) </div> </div>
<code>_PWRFIG_FRM_TYPE_MAX</code>	3	Maximum enum code for frame type.
<code>_PWRFIG_FRM_TYPE_UNKNOWN</code>	0	Unknown frame type.

6.11. medium_cmd_flag

Description

Medium management command.

The PowerFIP coprocessor provides a medium redundancy management solution for a FIP/WorldFIP bi-medium connection node. These commands allow to manage these two FIP channels.

Definition

```
enum pwrfig_medium_cmd_flag {
    PWRFIG_MEDIUM_CMD_ENABLE_CH_1 = (1 << 0),
    PWRFIG_MEDIUM_CMD_DISABLE_CH_1 = (1 << 1),
    PWRFIG_MEDIUM_CMD_ENABLE_CH_2 = (1 << 2),
    PWRFIG_MEDIUM_CMD_DISABLE_CH_2 = (1 << 3),
    PWRFIG_MEDIUM_CMD_ENABLE_CH_1_2 = ((PWRFIG_MEDIUM_CMD_ENABLE_CH_1) |
                                         (PWRFIG_MEDIUM_CMD_ENABLE_CH_2)),
    PWRFIG_MEDIUM_CMD_DISABLE_CH_1_2 = ((PWRFIG_MEDIUM_CMD_DISABLE_CH_1) |
                                         (PWRFIG_MEDIUM_CMD_DISABLE_CH_2)),
    PWRFIG_MEDIUM_CMD_CLEAR_TX_ERR = (1 << 4),
    PWRFIG_MEDIUM_CMD_RESET_CH_1 = (1 << 5),
    PWRFIG_MEDIUM_CMD_RESET_CH_2 = (1 << 6),
    PWRFIG_MEDIUM_CMD_RESET_CH_1_2 = ((PWRFIG_MEDIUM_CMD_RESET_CH_1) |
                                       (PWRFIG_MEDIUM_CMD_RESET_CH_2)),
};
```

Values

Constant	Value	Description
PWRFIG_MEDIUM_CMD_ENABLE_CH_1	0x0001	Enable FIP channel 1.
PWRFIG_MEDIUM_CMD_DISABLE_CH_1	0x0002	Disable FIP channel 1.
PWRFIG_MEDIUM_CMD_ENABLE_CH_2	0x0004	Enable FIP channel 2.
PWRFIG_MEDIUM_CMD_DISABLE_CH_2	0x0008	Disable FIP channel 2.
PWRFIG_MEDIUM_CMD_ENABLE_CH_1_2	0x0005	Enable both FIP channels.
PWRFIG_MEDIUM_CMD_DISABLE_CH_1_2	0x000a	Disable both FIP channels.
PWRFIG_MEDIUM_CMD_CLEAR_TX_ERR	0x0010	Clear TX error flag for both channels.

Constant	Value	Description
PWRFIP_MEDIUM_CMD_RESET_CH_1	0x0020	Reset FIP channel 1.
PWRFIP_MEDIUM_CMD_RESET_CH_2	0x0040	Reset FIP channel 2.
PWRFIP_MEDIUM_CMD_RESET_CH_1_2	0x0060	Reset both channels.

6.12. medium_state

Description

Flags describing FIP medium (channels) state.

Definition

```
enum pwrfig_medium_state {  
    PWRFIG_MEDIUM_STATE_CH1_VALID = (1 << 0),  
    PWRFIG_MEDIUM_STATE_CH2_VALID = (1 << 1),  
    PWRFIG_MEDIUM_STATE_CH1_TX_ERROR = (1 << 2),  
    PWRFIG_MEDIUM_STATE_CH2_TX_ERROR = (1 << 3),  
    PWRFIG_MEDIUM_STATE_CH1_WATCHDOG = (1 << 4),  
    PWRFIG_MEDIUM_STATE_CH2_WATCHDOG = (1 << 5),  
};
```

Values

Constant	Value	Description
PWRFIG_MEDIUM_STATE_CH1_VALID	0x0001	Channel 1 is active
PWRFIG_MEDIUM_STATE_CH2_VALID	0x0002	Channel 2 is active
PWRFIG_MEDIUM_STATE_CH1_TX_ERROR	0x0004	TX Error detected on channel 1
PWRFIG_MEDIUM_STATE_CH2_TX_ERROR	0x0008	TX Error detected on channel 2
PWRFIG_MEDIUM_STATE_CH1_WATCHDOG	0x0010	Watchdog on channel 1. MAU has to be reset.
PWRFIG_MEDIUM_STATE_CH2_WATCHDOG	0x0020	Watchdog on channel 2. MAU has to be reset.

6.13. msg_rx_seg_cap

Description

Message reception capability for the node depending on the destination segment of the FIP message.




A FIP node can be configured to be more or less sensitive to receiving FIP messages from the network.

Depending on the header of the received message (destination ID, destination Segment), it is possible to filter all messages destined to a particular FIP segment or to accept only particular header values.

Definition

```
enum pwrfig_msg_rx_seg_cap {  
    _PWRFIG_MSG_SEG_CAP_MIN = 0,  
    PWRFIG_MSG_SEG_IGNORE = _PWRFIG_MSG_SEG_CAP_MIN,  
    PWRFIG_MSG_SEG_ACCEPT_ALL,  
    PWRFIG_MSG_SEG_ACCEPT_LTD,  
    _PWRFIG_MSG_SEG_CAP_MAX,  
};
```

Values

Constant	Value	Description
<code>_PWRFIG_MSG_SEG_CAP_MIN</code>	0	Minimum capacity value.
<code>PWRFIG_MSG_SEG_IGNORE</code>	0	Ignore all messages sent to the segment.
<code>PWRFIG_MSG_SEG_ACCEPT_ALL</code>	1	Accept all messages sent to the segment (regardless of the destination id [msg_hdr]).
<code>PWRFIG_MSG_SEG_ACCEPT_LTD</code>	2	Limited acceptance. <div> Only if the recipient identifier field (msg_hdr: dst_id) is configured for the node (and accept message reception).</div>
<code>_PWRFIG_MSG_SEG_CAP_MAX</code>	3	Maximum capacity value.

6.14. node_operation

Description

Operation type inside a FIP node.

Definition

```
enum pwrfig_node_operation {  
    _PWRFIG_NODE_OP_UNKNOWN,  
    PWRFIG_NODE_OP_WAIT_RX_RP_FRM,  
    PWRFIG_NODE_OP_WAIT_TX_RP_FRM,  
    PWRFIG_NODE_OP_WAIT_RX_ID_FRM,  
    PWRFIG_NODE_OP_WAIT_TX_ID_FRM,  
    _PWRFIG_NODE_OP_MAX,  
};
```

Values

Constant	Value	Description
_PWRFIG_NODE_OP_UNKN OWN	0	Unknown operation
PWRFIG_NODE_OP_WAIT_ RX_RP_FRM	1	Wait for reception of <i>RP_XX</i> frame
PWRFIG_NODE_OP_WAIT_ TX_RP_FRM	2	<i>RP_XX</i> frame transmission in progress
PWRFIG_NODE_OP_WAIT_ RX_ID_FRM	3	Wait for reception of <i>ID_XX</i> frame
PWRFIG_NODE_OP_WAIT_ TX_ID_FRM	4	<i>ID_XX</i> frame transmission in progress
_PWRFIG_NODE_OP_MAX	5	Max node operation

6.15. node_state

Description

FSM (Finite State Machine) for a FIP node.

Definition

```
enum pwrfig_node_state {  
    PWRFIG_NODE_STATE_INITIAL,  
    PWRFIG_NODE_STATE_LOADED,  
    PWRFIG_NODE_STATE_READY,  
    PWRFIG_NODE_STATE_RUNNING,  
    _PWRFIG_NODE_STATE_MAX,  
};
```

Values

Constant	Value	Description
PWRFIG_NODE_STATE_INITIAL	0	Initial state. No config loaded
PWRFIG_NODE_STATE_LOADED	1	General node config loaded.
PWRFIG_NODE_STATE_READY	2	User context loaded (AE/LE). Here, FIP node is in stopped state, and is ready to start.
PWRFIG_NODE_STATE_RUNNING	3	Running state. Node is active on network
_PWRFIG_NODE_STATE_MAX	4	Max node state number

6.16. var_err_code

Description


FIP variable error codes after read/write operation.

Definition

```
enum pwrfig_var_err_code {
    _PWRFIG_VAR_OK,
    _PWRFIG_VAR_ERR_MIN,
    /* configuration errors */
    PWRFIG_VAR_ID_UNKNOWN = _PWRFIG_VAR_ERR_MIN,
    PWRFIG_VAR_NOT_PRODUCING,
    PWRFIG_VAR_NOT_CONSUMING,
    PWRFIG_VAR_TX_APER_CH_UNKNOWN,
    /* context error */
    PWRFIG_VAR_PDU_INCONSISTENT,
    PWRFIG_VAR_LEN_TOO_LONG,
    PWRFIG_VAR_LEN_TOO_SHORT,
    PWRFIG_VAR_NEVER_RECEIVED,
    PWRFIG_VAR_TX_APER_FIFO_EMPTY,
    PWRFIG_VAR_TX_APER_FIFO_FULL,
    /* payload error */
    /* -> cons var */
    PWRFIG_VAR_NOT_MEANING,
    PWRFIG_VAR_NOT_REFRESH,
    PWRFIG_VAR_NOT_PROMPT,
    PWRFIG_VAR_BAD_PROMPT_PER,
    /* -> prod var */
    PWRFIG_VAR_BAD_REFRESH_PER,
    _PWRFIG_VAR_ERR_MAX,
};
```

Values

Constant	Value	Description
_PWRFIG_VAR_OK	0	No error. Operation correctly performed.
_PWRFIG_VAR_ERR_MIN	1	Minimal error code.
PWRFIG_VAR_ID_UNKNOWN	1	Unknown variable ID. The FIP identifier is not set to support a variable. No RP_DAT frame attached to local database.
PWRFIG_VAR_NOT_PRODUCING	2	The FIP variable is not set to production.
PWRFIG_VAR_NOT_CONSUMING	3	The FIP variable is not set to consumption.

Constant	Value	Description
PWRFIP_VAR_TX_APER_CH_UNKNOWN	4	Unknown TX channel number for aperiodic variable. <div>  <ul style="list-style-type: none"> • Normal channel number = 0 • Urgent channel number = 1 </div>
PWRFIP_VAR_PDU_INCONSISTENT	5	Inconsistent variable PDU (Protocol Data Unit). The PDU read on the FIP network does not match with the PDU configured for the variable.
PWRFIP_VAR_LEN_TOO_LONG	6	The size of the variable read on the FIP network is longer than the one configured in the local database.
PWRFIP_VAR_LEN_TOO_SHORT	7	The size of the variable read on the FIP network is shorter than the one configured in the local database.
PWRFIP_VAR_NEVER_RECEIVED	8	The variable has never been received on the FIP node.
PWRFIP_VAR_TX_APER_FIFO_EMPTY	9	TX aperiodic variable queue is empty.
PWRFIP_VAR_TX_APER_FIFO_FULL	10	TX aperiodic variable queue is full.
PWRFIP_VAR_NOT_MEANING	11	The FIP variable read is not significant.
PWRFIP_VAR_NOT_REFRESH	12	The FIP variable is not fresh. (see <i>production status</i> byte)
PWRFIP_VAR_NOT_PROMPT	13	The FIP variable is not prompt.
PWRFIP_VAR_BAD_PROMPT_PER	14	Bad reading frequency for application layer (promptness). User app is not reading the FIP variable with a correct period. This may be due to excessive OS latencies.
PWRFIP_VAR_BAD_REFRESH_PER	15	Bad writing frequency for application layer (refreshment) User app is not writing the FIP variable with a correct period. This may be due to excessive OS latencies.
_PWRFIP_VAR_ERR_MAX	16	Maximum error code

6.17. var_flags



Description





Set-up flags for a FIP variable

Definition

```
enum pwrfig_var_flags {  
    /**  
     * PROD/CONS  
     */  
    PWRFIG_VAR_FLAGS_REFRESH = (1 << 0),  
    PWRFIG_VAR_FLAGS_DYN_REFRESH = (1 << 1),  
    PWRFIG_VAR_FLAGS_MSG_REC = (1 << 2),  
    /**  
     * PROD only  
     */  
    PWRFIG_VAR_FLAGS_APER_VAR_REQ = (1 << 8),  
    PWRFIG_VAR_FLAGS_APER_MSG_REQ = (1 << 9),  
    /**  
     * CONS only  
     */  
    PWRFIG_VAR_FLAGS_PROMPT = (1 << 12),  
    PWRFIG_VAR_FLAGS_CHK_PDU_LEN = (1 << 13),  
  
    PWRFIG_VAR_FLAGS_MAX_VAL = (1 << 15),  
};
```

Values

Constant	Value	Description
PWRFIP_VAR_FLAGS_REFRESH	0x0001	<p>Enable/Disable production status (refreshment)</p> <div>  <ul style="list-style-type: none"> • Production variable case: If this option is enabled, an extra byte - called <i>production status</i> - is automatically added to the end of the payload by the coprocessor. This byte is updated by the producer to inform other nodes if the payload is correctly refreshed inside its local database. • Consumption variable case: If this option is enabled, the last useful byte of the frame is considered to be a <i>production state</i>. It is then interpreted by the coprocessor to know the freshness state of the variable consumed on the network. </div>
PWRFIP_VAR_FLAGS_DYN_REFRESH	0x0002	<p>Enable/Disable dynamic production status (eq. var_time)</p> <div>  <p>In addition to the <i>production status</i> byte added at the end of the user payload, 4 extra bytes are inserted by the producer to inform the <i>production time</i> of the variable to other nodes. This time - in microseconds - expresses the delay between the user's write command and the actual production by the MAU of producer on the network.</p> </div>
PWRFIP_VAR_FLAGS_MSG_REC	0x0004	<p>Enable/Disable message reception for the attached FIP ID</p>

Constant	Value	Description
PWRFIP_VAR_FLAGS_APER_VAR_REQ	0x0100	Enable/Disable aperiodic variable request capability  This flag is reserved only for a variable set in production .
PWRFIP_VAR_FLAGS_APER_MSG_REQ	0x0200	Enable/Disable aperiodic message request capability  This flag is reserved only for a variable set in production .
PWRFIP_VAR_FLAGS_PROMPT	0x1000	Enable/Disable promptness status  This flag is reserved only for a variable set in consumption .
PWRFIP_VAR_FLAGS_CHK_PDU_LEN	0x2000	Enable/Disable PDU+LEN bytes frame check  This flag is reserved only for a variable set in consumption .
PWRFIP_VAR_FLAGS_MAX_VAL	0x8000	Maximal setting flag for variable

6.18. var_type


Description

FIP variable type.

Definition

```
enum pwrfig_var_type {  
    PWRFIG_VAR_TYPE_CONS,  
    PWRFIG_VAR_TYPE_PROD,  
    PWRFIG_VAR_TYPE_SYNC,  
};
```

Values

Constant	Value	Description
PWRFIG_VAR_TYPE_CONS	0	Consumption variable.
PWRFIG_VAR_TYPE_PROD	1	Production variable.
PWRFIG_VAR_TYPE_SYNC	2	Synchronization variable. <div> No payload is attached to this kind of variable; but the node is sensitive to (RX/TX) ID_DAT frame and signal the user-space with a synchronous event when it occurs.</div>

Appendix A: Revision History

Revision	Changes	Authors	Date
0.7.2	<ul style="list-style-type: none">Mailboxes processing inside the driver (Windows)High thread priority for internal library threadAdd <i>index</i> field to <code>struct pwrfig_dev_infos</code>	MC	2021-09-21
0.7.1	Windows 7/8/10 OS support	MC	2021-08-04
0.7.0	First version (Linux OS)	MC	2021-07-05