FlexCard.

fcBase API Documentation

by Eberspächer Electronics.

FlexCard Cyclone II	(3-0009-0S01)
FlexCard Cyclone II SE	(3-0009-0T01)
FlexCard PMC/PCI	(3-0033-0P01)
FlexCard PMC II	(3-0055-0P01)

Eberspächer

CONTACT INFORMATION

Eberspächer Electronics GmbH & Co. KG
Robert-Bosch-Str. 6
D-73037 Göppingen, Germany
Phone + 49 7161 9559-0
Phone + 49 7161 9559-222 (Support)
Fax + 49 7161 9559-455

Sales: Ebel-sales@eberspaecher.com
Support: Ebel-support@eberspaecher.com
www.eberspaecher.com/electronics

COMPANY DATA

Eberspächer Electronics GmbH & Co. KG, registered offices: Göppingen, register court Ulm, HRA 721096
Partner liable to unlimited extent: Eberspächer Electronics Verwaltungs-GmbH, registered offices: Göppingen, register court Ulm, HRB 722565
Represented by the executive board: Martin Peters, Dr. Leonhard Vilser

"Eberspächer Electronics" represents Eberspächer Electronics GmbH & Co. KG.

COPYRIGHT NOTICE

© Copyright 2009 Eberspächer Electronics GmbH & Co. KG. All Rights Reserved.

No part of this document may be reproduced in any form (photocopy, microfilm or another procedure) without prior written consent from *Eberspächer Electronics*.

TRADEMARKS

FlexRay™ is a trademark of the FlexRay consortium.
Any other trademarks used in this document are the property of their respective owners.

DISCLAIMER

The information contained in this document does not affect or change General Terms and Conditions of *Eberspächer Electronics*. *Eberspächer Electronics* does not guarantee the completeness and accuracy of the content of this document and assumes no responsibility for any errors which may appear in this document or due to this document. The content of this document or the associated products are subject to change without notice at any time.

Based on currently state of arts and science it is impossible to develop software that is bug-free in all applications. Therefore, the product is only allowed to be used in the sense of the product use case described herein.

Eberspächer Electronics makes no warranty express or implied, as to this document or the information content, materials or products for any particular purpose, nor does *Eberspächer Electronics* assume any liability arising out of the application or use of this product, and disclaims all liabilities, including without limitation resulting damages, as permissible by applicable law.

All operating parameters which are provided in this document can vary in different applications or over time. The herein described product solely is allowed to be used as described in chapter “Intended use”.

Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written consent of Eberspächer Electronics.

Eberspächer Electronics may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly stated in a written license agreement from *Eberspächer Electronics*, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Any semiconductor devices have an inherent chance of failure. You have to protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions. The safety and handling instructions in this document have to be followed strictly.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

REVISION HISTORY

Version	Date	Description
D1V0-F	06-Mar-2006	Initial release
D1V1-F	02-Nov-2006	API functions added and updated. Multicard usage.
D1V2-F	02-Mai-2007	New API functions added and updated. PMC and XENOMAI usage.
D1V3-F	10-Mai-2007	Corrected description and changed Xenomai usage.
D1V4-F	21-Jun-2007	VxWorks API functions added.
D1V5-F	30-Aug-2007	PMC, VxWorks and Linux functions changed and added.
D1V6-F	02-Dec-2007	FlexCard Cyclone II (SE) support self startup/synchronization.
D1V7-F	28-Jan-2008	Support of CC Timer, API functions added and updated.
D1V8-F	25-Feb-2008	VxWorks chapter updated.
D1V9-F	11-Juli-2008	FlexCard Cyclone II (SE) support CAN. New API functions added.
D1V10-F	29-Oct-2008	FlexCard PMC/PCI support CAN. New FlexRay API functions added.
D1V11-F	27-Feb-2009	FlexCard PMC II support and new API functions added.
D1V12-F	16-Apr-2009	Corrected description. Linux driver supports FlexCard PMC II.

RELATED FIRMWARE AND HARDWARE VERSIONS (COMMON)

Component	Ref.No., Version	Remarks
FlexCard Cyclone II Firmware	3-0009-0C04, S5V1-F	Current version
FlexCard Cyclone II SE Firmware	3-0009-0C05, S5V1-F	Current version
FlexCard PMC Firmware	3-0033-0B01, S5V1-F	Current version
FlexCard PMC II Firmware	3-0055-0C01, S5V1-F	Current version
FlexCard Cyclone II Hardware	3-0009-0A04, H1V1-F	Initial version
FlexCard Cyclone II SE Hardware	3-0009-0A05, H1V1-F	Initial version
FlexCard PMC Hardware	3-0033-0A01, H1V0-F	Initial version
FlexCard PMC II Hardware	3-0055-0A01, H1V1-F	Initial version

RELATED SOFTWARE VERSIONS (WINDOWS)

Component	Ref.No., Version	Remarks
fcBase API	3-0009-0K03, S5V1-F	Current version supports: FlexCard Cyclone II (SE), FlexCard PMC (II)
Device Driver	3-0009-0E05, S5V1-F	Current version

RELATED SOFTWARE VERSIONS (LINUX)

Component	Ref.No., Version	Remarks
libfcBase API	3-0009-0U01, S5V1-F	Current version supports: FlexCard Cyclone II (SE), FlexCard PMC (II)
Kernel module	3-0009-0U01, S5V1-F	Current version

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

RELATED SOFTWARE VERSIONS (XENOMAI)

Component	Ref.No., Version	Remarks
libfcBase API	3-0009-0V01, S4V2-F	Initial version supports: FlexCard Cyclone II (SE), FlexCard PMC
Kernel module	3-0009-0V01, S4V2-F	Current version

RELATED SOFTWARE VERSIONS (VXWORKS)

Component	Ref.No., Version	Remarks
FlexCard PMC Driver	3-0033-0D01, S1V2-F	Current version supports: FlexCard PMC

RELATED DOCUMENTS

Document	Version	Ordering number
FlexCard Cyclone II SE Instructions for Use	D2V7-F	3-0009-0T01-D01
FlexCard PMC Instructions for Use	D1V5-F	3-0033-0P01-D01
FlexCard PMC II Instructions for Use	D1V1-F	3-0055-0P01-D05
FlexCard Cyclone II SE Getting started	D1V3-F	3-0009-0S01-D02
FlexCard PMC (II) Getting started	D1V0-F	3-0055-0P01-D07

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

CONTENTS

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

1 GENERAL

1.1 INTENDED USE

This document describes the application programming interface of the FlexCard to build own software applications.
The FlexCard is designed, intended and authorized exclusively for

a) EU: laboratory applications
b) US: industrial test equipment

Any other use needs the prior express written consent of *Eberspächer Electronics GmbH & Co. KG*.

1.2 USER GROUP

This documentation is written for software developers who are familiar with C/C++ programming language under the Windows™ operating systems and the FlexRay protocol specification.

Any person involved with installation, usage or maintenance of the FlexCard has to

- be a qualified technician
- strictly adhere to this guidance
- receive a briefing by an authorized person

1.3 PICTOGRAMS

	**NOTICE**
	Used to indicate a situation which may result in an operating failure. Damage of the product may occur, but there is no hazard of injury if not avoided.

	**Information**
	Used to indicate information provided only for purposes of clarification, illustration, and general information.

1.4 MEANING OF TEXT STYLES

In this document *filenames*, `source code`, FlexRay Protocol Variable, **`functions`** and **`structs`** are marked with a different text format.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

2 OVERVIEW

This document describes the application programming interface (API) *fcBase API* for the FlexCard. The API defines the basic functions and structures which are used to communicate with the FlexCard hardware, the FlexRay™ and CAN bus. With these functions the developer is able to integrate the FlexCard in a FlexRay cluster and CAN network.

The following figure illustrates a typical approach of accessing the FlexRay and CAN bus via the FlexCard:

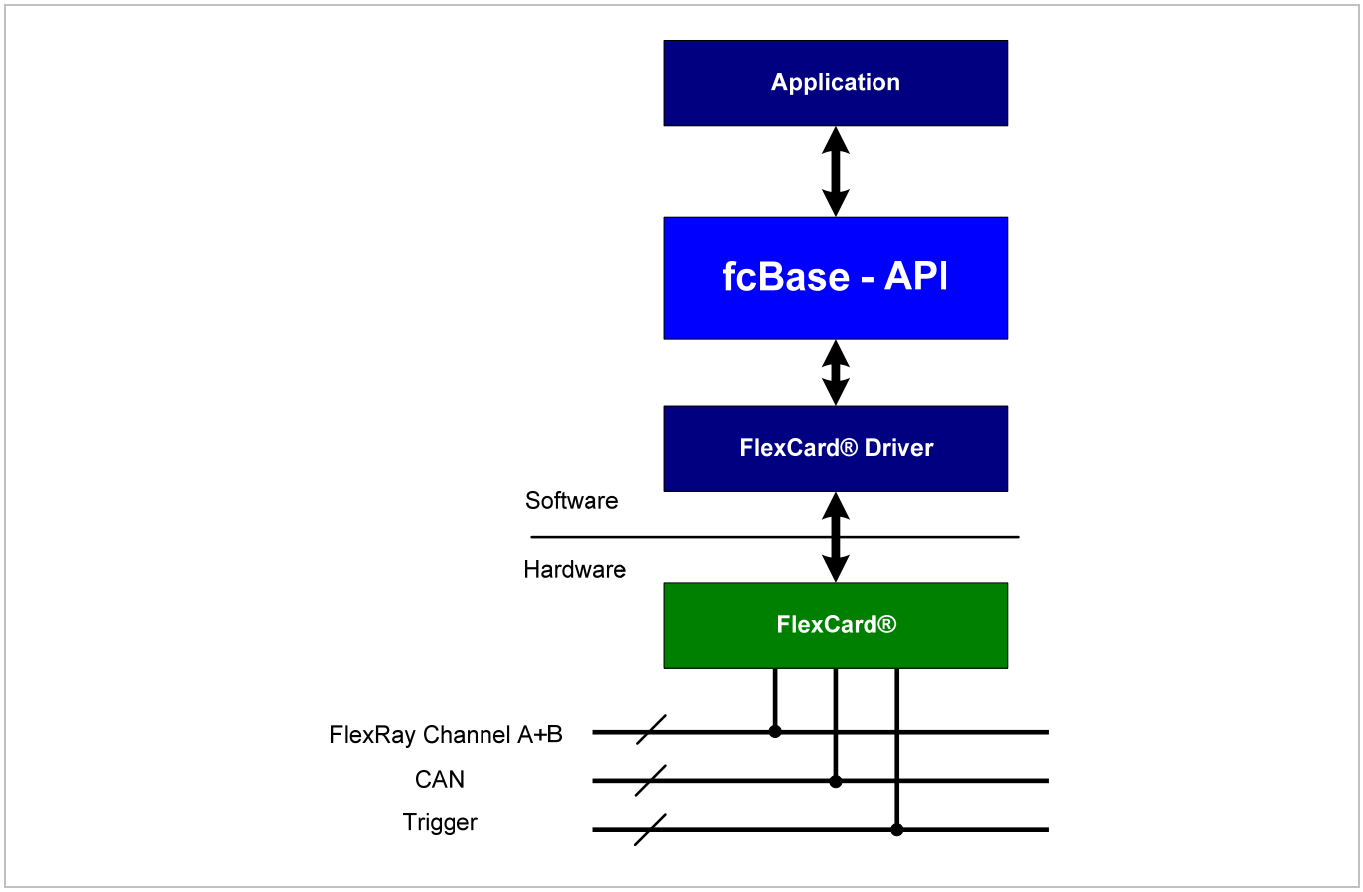


Figure 1: Overview of a typical FlexCard system with hardware and software

The *fcBase API* consists of the following groups of functions:

- **Error handling** → Functions to get detailed error information
- **Configuration** → Functions and structures to configure the available communication controller (e.g. message buffers) and the FlexCard hardware.
- **Initialization** → Functions to enumerate the FlexCards in the system, to establish a connection to a FlexCard and to start and stop the monitoring of the FlexRay and CAN bus.
- **Transmit / Receive** → Functions to receive FlexRay and CAN frames or informational frames (e.g. Trigger information), or to transmit a FlexRay and CAN frame on a specific slot or id.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

- **Event handling** → Functions to obtain event handles which are signalled if a specific time elapses, a wakeup pattern is detected or at the start of a new FlexRay cycle.

Additional there is a tracing module, which can only be accessed by the tracing control application. For further information refer to chapter 13 in this document.

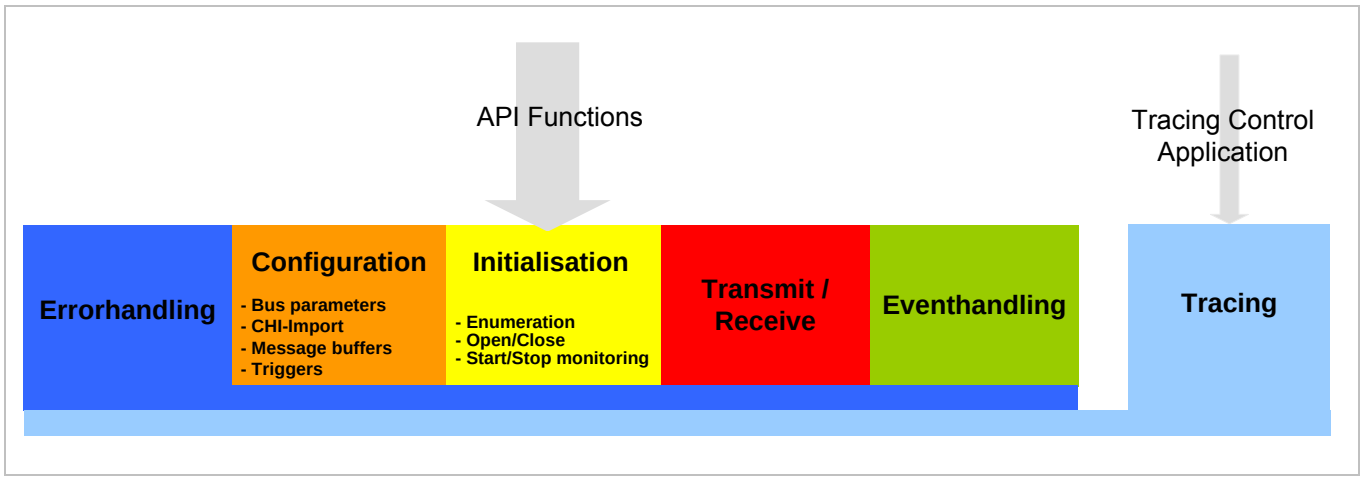


Figure 2: *fcBase API* groups

2.1 SUPPORT

There is support available by *Eberspächer Electronics* regarding software (device driver and API) and hardware. Before you submit a problem, ensure that you have the latest release of the software. The latest versions of the device driver and API are available from our support team or on our web site: http://www.eberspaecher.com/electronics

If you encounter a problem, please send an email to ebel-support@eberspaecher.com, including the following information:

- Description of your problem
- Detailed steps to reproduce the problem
- Version number of the device driver or loadable kernel module
- Version number of the DLL or shared object library
- Version number of the hardware
- Version number of the firmware
- Serial number of your FlexCard
- The application you are using
- Your computer system (manufacturer and type of PC, e.g. Dell Inspiron 7500)
- Your operating system (Windows 2000, XP, Vista, Linux, Xenomai, VxWorks)
- The cardbus or PCI adapter in your PC (e.g. Texas Instruments, …)
- If possible the CC configuration file or string (either CHI or CANdb) or a CC parameter list

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

3 GETTING STARTED

In this section the necessary steps for developing a FlexCard application with Windows operating systems are specified. First, the setup of the files and the integration in an Integrated Development Environment (IDE) is described. The next section provides a guideline with important steps to create an application. This includes the functions and structures which should normally be used. A more in depth view about the used functions can be found in chapter 4 et seq.

3.1 INSTALLATION

For details about the installation process please refer to the installation section in [1]. After a successful installation of the developer package the following directory structure should exist:

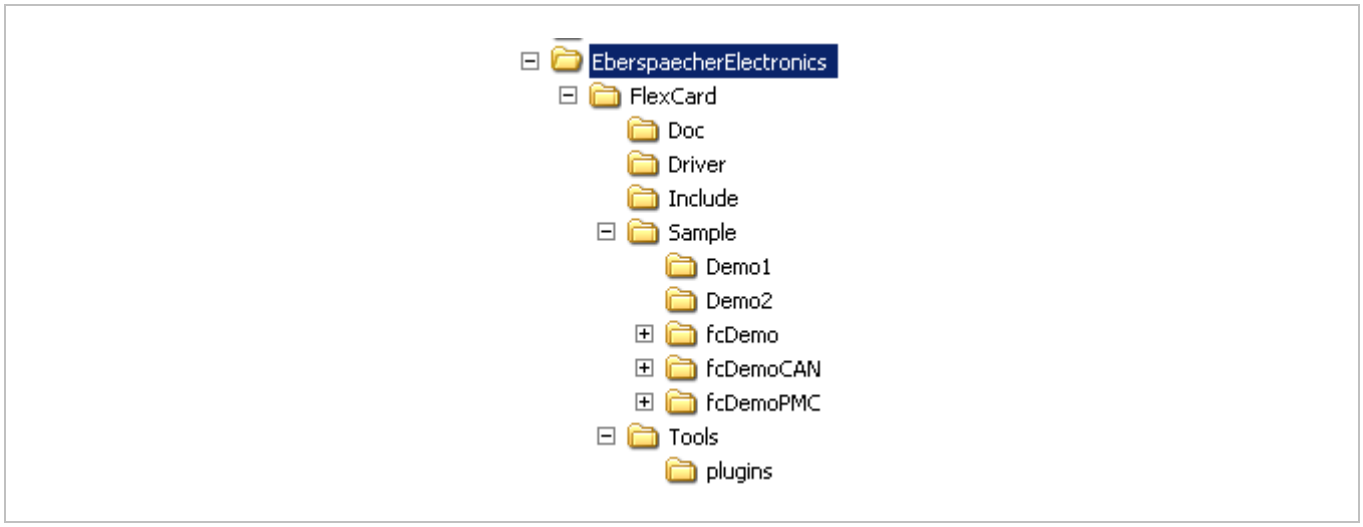


Figure 3: FlexCard directory structure

The directory *Doc* contains the documentation (API documentation, User manuals and Getting started guides) in PDF format.

The directory *Driver* contains the files for the manual installation of the device driver:

- *fce052k.sys* (Device driver for the Windows™ 2000 operating system)
- *fce05xp.sys* (Device driver for the Windows™ XP and Vista operating system)
- *fce05.inf* (Text file containing information which is needed for the installation of the device driver)
- *fce05.cat* (Catalog file which is needed for driver signing)
- *fcBase.dll* (fcBase Library)

The previous directory is not required for developing user defined application, whereas the two following directories are a must-have for a developer.

The directory *Include* contains the API definition, namely the *fcBase* header file:

- *fcBase.h*: The file includes the definition of the basic API functions.
- *fcBaseTypes.h*: The file contains the data types and enumerations (e.g. possible error codes) used by the basic functions.
- *fcBaseFlexRay.h*: This file contains definitions of functions specific for FlexRay.
- *fcBaseTypesFlexRay.h*: The data types and enumerations for the FlexRay functions are defined here.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

- *fcBasePMC.h*: The file includes additional definitions of API functions which are to use with FlexCard PMC only.
- *fcBaseTypesPMC.h*: The file (for the FlexCard PMC only) contains additional data types and enumerations used in this library.
- *fcBaseCAN.h*: The file includes the definition of the API functions which are to be used with a CAN license only.
- *fcBaseTypesCAN.h*: The file (for FlexCards with CAN license only) contains additional data types and enumerations used in this library.
- *fcBase.lib*: Exported functions

The directory *Sample* contains the following directories:

- *Demo1*: Configuration files for a cluster composed of two FlexCards
- *Demo2*: Configuration files for a cluster composed of one FlexCard and two FlexNodes.
- *fcDemo:* Contains the source files for the demo application.
- *fcDemoCAN*: Contains the source files for the CAN demo application for a FlexCard.
- *fcDemoPMC:* Contains the source files for the demo application for a FlexCard PMC.

The directory *Tools* contains the following applications:

- *CANBaudrateCalculator.exe*: Application to calculate CAN CC configuration for fcBase CAN API.
- *fcDemo.exe:* The demo application for one FlexRay CC.
- *fcDemoCAN.exe*: The demo application for two CAN CCs.
- *fcDemoPMC.exe:* The demo application for two FlexRay CCs.
- *FlexAlyzerV2.exe*: FlexRay and CAN monitoring application for FlexCard products.
- *FlexUpdate.exe:* Firmware and license update application for FlexCard products.

	Information
	The windows installer will copy the fcBase.dll into your <windows>\system32 directory. If you do not use the windows installer, please check if the desired version of the DLL is loaded. A description of the DLL search order which is used by the Windows operating system can be found in [2].

3.2 INTEGRATION

There are different ways to integrate the fcBase DLL into your application depending on the development platform and language. Under Microsoft Visual Studio the integration is done via the property pages/project settings.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

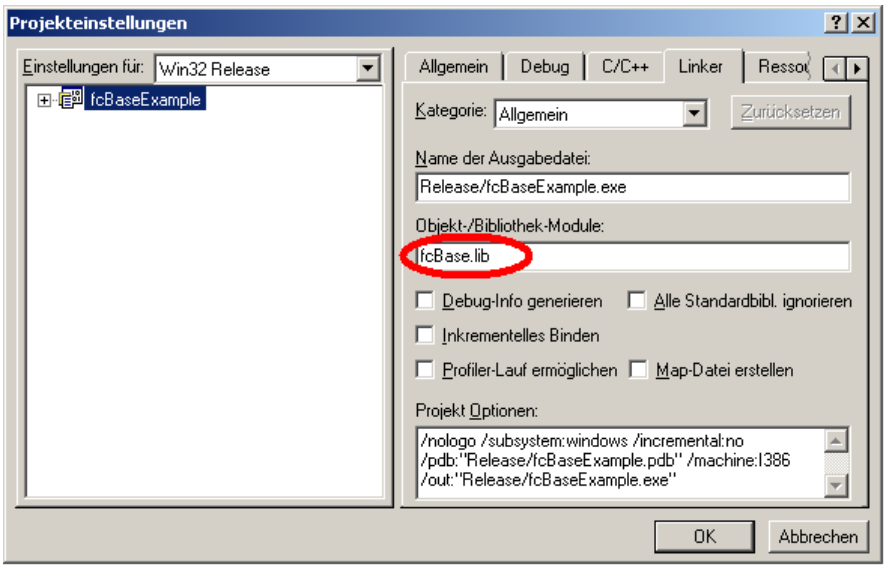


Figure 4: Integration under Visual Studio 6.0

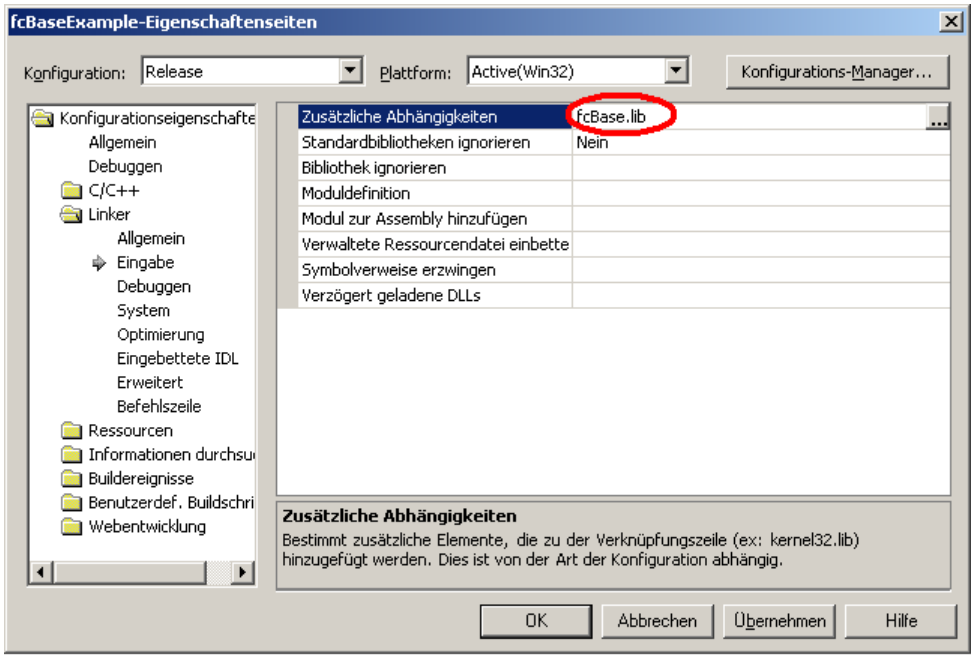


Figure 5: Integration under Visual Studio .NET 2003

Another alternative for Microsoft compiler users is to include the fcBase API via the Microsoft specific pre-processor command:

```
#pragma comment( lib , « fcBase.lib »)
```

To complete the integration of the fcBase API into your user defined application, you have to add the files *fcBaseTypes.h*, *fcBaseTypesFlexRay.h*, *fcBase.h* and *fcBaseFlexRay.h*. The include order is important because the file *fcBase.h* uses definitions which are declared in *fcBaseTypes.h* and the file *fcBaseFlexRay.h* uses definitions which are declared in *fcBaseTypes.h* and *fcBaseTypeFlexRay.h*. For FlexCard PMC usage please also include the files *fcBaseTypesPMC.h* and *fcBasePMC.h* in the right order.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

In case your FlexCard is licensed to use CAN, the files *fcBaseTypesCAN.h* and *fcBaseCAN.h* should be also included.

```
#include “fcBaseTypes.h”
#include “fcBaseTypesFlexRay.h”
#include “fcBase.h”
#include “fcBaseFlexRay.h”

//Additional for PMC usage
#include “fcBaseTypesPMC.h”
#include “fcBasePMC.h”

//Additional for CAN usage
#include “fcBaseTypesCAN.h”
#include “fcBaseCAN.h”
```

The setup program sets the environment variable FLEXCARD_INC which directly points to the fcBase include directory. This variable can be used as shown in the figures below.

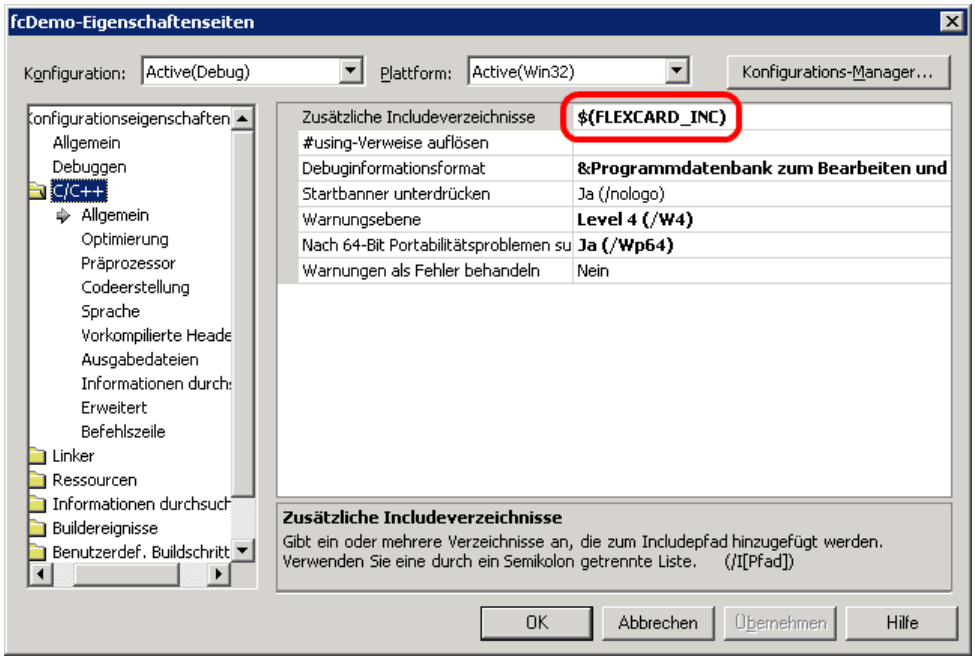


Figure 6: Using the variable FLEXCARD_INC under Visual Studio .NET 2003 (Compiler)

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

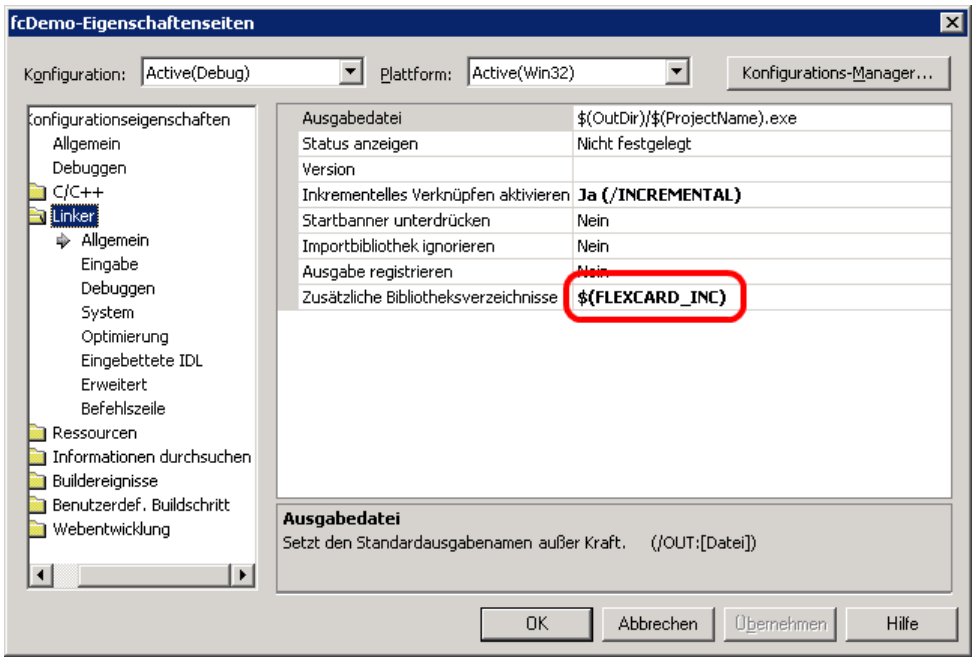


Figure 7: Using the variable FLEXCARD_INC under Visual Studio .NET 2003 (Linker)

	NOTICE
	Ensure you use the directory of the fcBase library and header files which corresponds to the loaded DLL. A description of the DLL search order which is used by the windows operating system can be found in [2].

3.2.1 CALLING CONVENTION

The dynamic link library was developed under Visual Studio C++.NET 2003. The Microsoft C/C++ compiler supports five calling conventions (__cdecl, __stdcall, __fastcall, this, naked). To provide access to the API functions for other languages (e.g. Visual Basic), the functions are declared with __stdcall calling convention (function arguments are pushed onto the stack from right to left, the callee cleans the stack).

3.2.2 MULTITHREADING

All functions, which are not declared as obsolete, of the fcBase library are thread-safe. If you are using the fcBase functions in the context of a multi-threaded program, the library ensures that only one thread is accessing the internal shared data at any given time.

3.3 BASIC WORKFLOW

This section will guide you through the necessary workflow for creating an application for the FlexCard. Figure 8 shows a typical workflow. The main functions and principles for building a user defined application are introduced in this chapter. The chapter is based on the provided C/C++ example.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

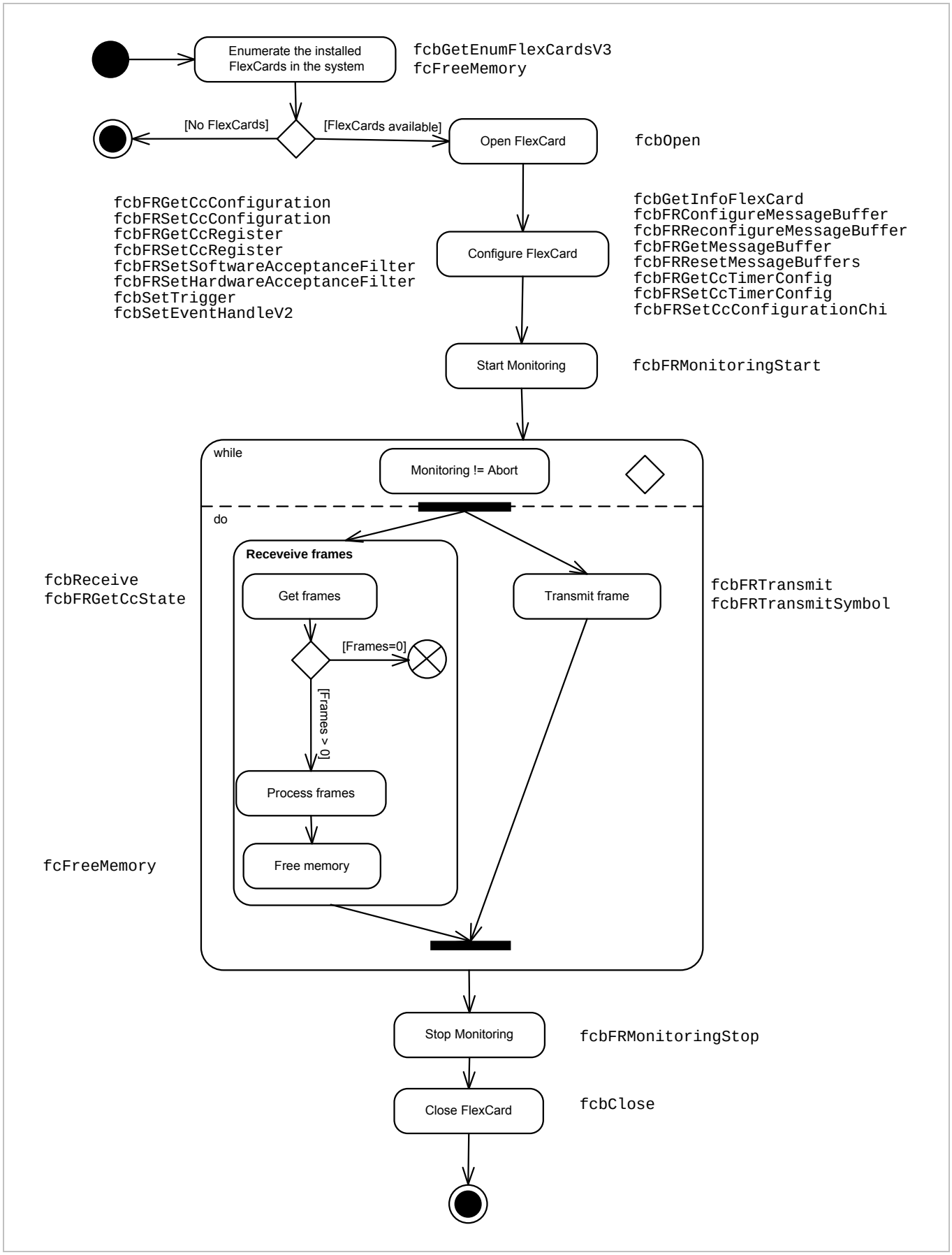


Figure 8: Typical FlexRay function workflow

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

3.3.1 SETTING UP THE PROJECT

For the development of the example project we will use Microsoft Visual Studio 2003 with the programming language C/C++ and the Microsoft Foundation Class (MFC). The Visual Studio project wizard will generate the framework for our MFC dialog based application (for more details, please refer to the documentation of Microsoft Visual Studio 2003).

As described in the chapter Integration, we have to add the library and header files of the fcBase API. This can be done easily at the end of the file *stdafx.h*. Ensure your compiler and linker use the correct path to the fcBase header and library files.

```
...

// fcBase API
#pragma comment(lib, »fcBase.lib »)
#include “fcBaseTypes.h”
#include “fcBaseTypesFlexRay.h”
#include “fcBase.h”
#include “fcBaseFlexRay.h”

// for FlexCard PMC only
#include “fcBaseTypesPMC.h”
#include “fcBasePMC.h”

// for FlexCards with CAN license only
#include “fcBaseTypesCAN.h”
#include “fcBaseCAN.h”
```

3.3.2 GET THE INSTALLED FLEXCARDS

Before we can open a connection to a FlexCard, we require a valid FlexCard identifier. This can be done with the function fcbGetEnumFlexCardsV3 which returns a list of FlexCards found in the system. In the method `CselectFlexCardDlg::OnInitDialog()` in our example we call fcbGetEnumFlexCardsV3 to fill the combo box with available FlexCards found in the system.

	Information
	The fcInfoHwSw structure contains valid FlexCard information only if the member `FlexCardId` is greater than 0. The `FlexCardId` is later used to open a connection to the FlexCard.

```
fcError e = fcbGetEnumFlexCardsV3(&m_pInfoHwSw, false);
if (0 == e)
{
      // Iterate through the list of flexcards
      fcInfoHwSw* pCurrent = m_pInfoHwSw;
      while (NULL != pCurrent)
      {
            // only if we got a valid flexcard identfier
            if ( 0 != pCurrent->FlexCardId )
            {
                  Cstring szItem;
                  szItem.Format(“FlexCard #%d”,pCurrent->InfoHardware.Serial);

                  // Add the string to the combo box
                  int nIndex = m_FlexCardComboBox.InsertString(0,szItem);
                  m_FlexCardComboBox.SetItemDataPtr(nIndex,pCurrent);
            }
            // get the next flexcard
            pCurrent = pCurrent->pNext;
      }
}
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

If the user selects one of the items in the combo box, we save the member FlexCardId from the structure fcInfoHwSw into the member variable `m_flexcardIdentifier` (see `CselectFlexCardDlg::UpdateVersionInformation`).

```
int nCurrentSelection = m_FlexCardComboBox.GetCurSel();
if (-1 != nCurrentSelection)
{
      fcInfo* pCurrent =
            (fcInfo*)m_FlexCardComboBox.GetItemDataPtr(nCurrentSelection);

      // Save the flexcard identifier
      m_flexcardIdentifier = pCurrent->FlexCardId;
      ...
}
```

Once finished with the selection of a FlexCard, we have to free the memory which was allocated by the function fcbGetEnumFlexCardsV3.

```
CselectFlexCardDlg::~CselectFlexCardDlg()
{
      if (NULL != m_pInfo)
      {
            fcFreeMemory(fcMemoryTypeInfoHwSw,m_pInfo);
            m_pInfo = NULL;
      }
}
```

3.3.3 OPEN A CONNECTION

After getting a valid FlexCard identifier, we use it to open a connection to the FlexCard. The function fcbOpen expects this identifier and returns a handle (to the previously selected FlexCard) which is later used in all other functions.

	Information
	The function fcbOpen resets all configuration settings. That means that all communication controller registers are set to their default value and the message buffers are configured as FIFO buffer

```
...
m_hFlexCard = NULL;
fcError e = fcbOpen(&m_hFlexCard,dlg.FlexCardIdentifier());
...
```

3.3.4 CONFIGURE THE FLEXCARD

In order to integrate the FlexCard into a FlexRay cluster it is essential to configure its communication controller registers. These registers describe global cluster parameters (e.g. gdStaticSlot), node parameters (e.g. pMicroPerCycle) and communication controller specific settings. The global cluster parameters are identical for all nodes of a cluster, whereas the node parameters are set for each node individually.
If one of these parameters is not correct, the integration of the FlexCard and/or the communication may fail. Therefore it is recommended to use a tool (e.g. FlexConfig) which generates the configuration file for each node.

In our example we use a CHI-compatible string to configure the FlexCard. As the function fcbFRSetCcConfigurationChi expects a string, we read and parse the configuration file into a string.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Std::string s;
std::ifstream file(szPath);
if (! File.is_open())
{
      // Print error message
      return;
}

char ch;
while (file.get(ch)) s += ch;
file.close();
```

This string is passed to the function fcbFRSetCcConfigurationChi which will configure the specified Communication Controller registers described in the chi file. Setting a configuration via this function will override any previous configuration.

```
fcCC eCC = fcCC1;
fcError e = fcbFRSetCcConfigurationChi(m_hFlexCard, eCC, s.c_str());
```

As we want to transmit messages on the FlexRay bus, we have to configure transmit buffers for the FlexCard. To configure such a buffer two options exist: using the function fcbFRConfigureMessageBuffer or via the CHI configuration string. There is a significant difference between these two methods: While fcbFRConfigureMessageBuffer returns the index of the configured message buffer, fcbFRSetCcConfigurationChi does not. And considering that to transmit the content of a message buffer, the function fcbFRTransmit requires its index; we need a way to retrieve it. The following code performs this task for all configured transmit message buffers.

```
// Get all transmit message buffers
unsigned int bufferIdx = 1; // The first valid buffer is 1
while (true)
{
      fcMsgBufCfg cfg;
      fcCC eCC = fcCC1;

      // as long no error occurs we try to get each buffer
      fcError e = fcbFRGetMessageBuffer(m_hFlexCard,eCC,bufferIdx,&cfg);
      if (0 != e) break;

      // is this a tx buffer, then add it to our list
      if (fcMsgBufTx == cfg.Type)    Buffers[bufferIdx] = cfg;

      // next buffer index
      bufferIdx++;
}
```

The function fcbFRConfigureMessageBuffer is used to add a message buffer dynamically. This function checks the given message buffer settings and will report an error in the case of a mismatch with a global cluster parameter or a node specific parameter. The returned error informs the user about the mismatch.

```
fcMsgBufCfg cfg;

memset(&cfg, 0, sizeof(fcMsgBufCfg));
cfg.Type = fcMsgBufTx;
cfg.ChannelFilter = fcChannelA;
cfg.CycleCounterFilter = 0;
cfg.Tx.FrameId = 5;
cfg.Tx.MessageBufferInterrupt = 0;
cfg.Tx.PayloadLength = 16;
cfg.Tx.PayloadLengthMax = 16;
cfg.Tx.PayloadPreambleIndicator = 0;
cfg.Tx.StartupFrameIndicator = 0;
cfg.Tx.SyncFrameIndicator = 0;
cfg.Tx.TransmissionMode = fcMsgBufTxSingleShot;
cfg.Tx.TxAcknowledgeShowNullFrames = 0;
cfg.Tx.TxAcknowledgeShowPayload = 0;

fcCC eCC=fcCC1;
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
unsigned int bufferIdx = 0;
fcError e = fcbFRConfigureMessageBuffer(m_hFlexCard,eCC,&bufferIdx,cfg);

if (0 != e)
{
      ShowError(e);
}
```

Via the function fcbFRReconfigureMessageBuffer and with some restrictions (see message buffer structure), the user can modify an existing message buffer.

3.3.5 START AND STOP A MEASUREMENT

After having successfully configured the FlexCard, the monitoring can be started through the function fcbFRMonitoringStart. To use the FlexCard as a wakeup node, the flag `enableWakeup` has to be set to true (the FlexCard must have been previously configured with the correct wakeup settings). To use the FlexCard as a startup node, the flag `enableColdstart` has to be set to true (one transmit buffer with both startup and sync flags set must have been previously configured). In the case of the integration of a FlexCard into a running cluster, none of these two parameters has to be set. To be notified at the start of each cycle, the flag `enableCycleStartEvents` has to be set to true and the user has to provide an event object (used to signal when a new cycle starts) to the function fcbSetEventHandleV2.

```
// create the event handle which is signaled when a new cycle starts
const bool cyclestart = true;
fcCC eCC=fcCC1;

HANDLE hEvent = ::CreateEvent(NULL,FALSE,FALSE,NULL);

// inform the api that the event should be used when a new cycle starts
fcError e = fcbSetEventHandleV2(m_hFlexCard, eCC,
      hEvent, fcNotificationTypeCycleStarted);

// no coldstart and wakeup attempt have to be done
const bool coldstart = false;
const bool wakeup = false;

fcError e = fcbFRMonitoringStart(m_hFlexCard,eCC,fcMonitoringNormal,true,
            cyclestart,coldstart,wakeup);
```

After starting the monitoring, it is highly recommended to verify that the integration has succeeded. It can be determined either by receiving (via fcbReceive) a status packet with the flag `fcStatusStartupCompletedSuccessfully` set or by calling the function fcbFRGetCcState and checking that the return value is `fcStateNormalActive`.
Calling the function fcbFRMonitoringStop will stop the monitoring and set back the communication controller in its configuration state, `fcStateConfig`.

3.3.6 RECEIVE FLEXRAY FRAMES

Once the monitoring started, the FlexCard begins to monitor the FlexRay bus. The received FlexRay frames and the FlexCard generated packets (Info frame, Error frame, etc.) can be fetched by the function fcbReceive. A call to this function will get all available packets from the FlexCard.
The code below uses the cycle start event to collect the received data of the previous cycles. If the event is signalled or if the timeout elapses, we get the available received packets (fcPacket) by calling the function fcbReceive. The timeout is used as a fallback if the FlexCard is not successfully integrated and no cycle start events could be generated.

```
DWORD CdemoDlg::Thread()
{
      // … Code removed for listing …
      fcCC eCC=fcCC1;
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
	// create the event handle which is signaled when a new cycle starts

	hEvents[1] = ::CreateEvent(NULL,FALSE,FALSE,NULL);

	// inform the api that the event should be used when a new cycle starts
	fcError e = fcbSetEventHandleV2(m_hFlexCard,eCC,hEvents[1],
		fcNotificationTypeCycleStarted);

	// … Code removed for listing

	while (endlessLoop)
	{
		// Wait until an event is signaled or until timeout has elapsed
		DWORD dwResult = ::WaitForMultipleObjects(2,hEvents,false,
			dwTimeOutMilliseconds);

		switch (dwResult)
		{
		case WAIT_OBJECT_0+1:  // Cycle start event
		case WAIT_TIMEOUT:     // or time is elasped
			{
				//Update our transmit buffers
				AutomaticTransmit();

				fcPacket* pPacket = NULL;
				e = fcbReceive(m_hFlexCard, &pPacket);
				if (0 == e)
				{
					ProcessPackets(pPacket);
					e =   fcFreeMemory(fcMemoryTypePacket, pPacket);
				}
				else
				{
				 // … Code removed for listing
				}
			}
			break;

		// … Code removed for listing
	}
	// … Code removed for listing
}
```

The fcbReceive function returns the received data as a linked list of packets. The code below goes through the whole list and processes each packet.

```
Void CdemoDlg::ProcessPackets(fcPacket* pPackets)
{
	fcPacket* p = pPackets;
	while (NULL != p)
	{
		switch (p->Type)
		{
		case fcPacketTypeInfo:
			// … Code removed for listing
			break;
		case fcPacketTypeFlexRayFrame:
			// … Code removed for listing
			break;
		// … Code removed for listing
		default:

		};

		// get the next packet
		p = p->pNextPacket;
	}
}
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

	Information
	If an error packet with the flag fcErrFlexcardOverflow is received, the monitoring can not continue in case the FlexCard is set to stop if a packet overflow occurred (fcbSetContinueOnPacketOverflow). This error occurs if the application is too slow to receive and process the packets. In such a case it is necessary to stop the monitoring and start it again.

After processing the packets, the memory allocated by the packet list has to be released.

```
ProcessPackets(pPacket);
e =   fcFreeMemory(fcMemoryTypePacket, pPacket);
```

3.3.7 TRANSMIT FLEXRAY FRAMES

To transmit a frame on the FlexRay bus you need to have previously configured a transmit buffer and to know its index. The transmission is done by calling the fcbFRTransmit function.

```
fcCC eCC=fcCC1;
fcWord payload[fcPayloadMaximum];
payload[0] = 0x0001; // Update your payload data

fcError e = fcbFRTransmit(m_hFlexCard,eCC,bufferIdx,
            payload,payloadLength);
```

The transmit function expects the index of the communication controller, the index of the transmit buffer, the payload of the frame (the data) and the length of the payload section (the data length). The configured payload length (set during configuration of the transmit buffer) and the payload length to transmit (set during call to fcbFRTransmit) must match. The transmission may fail, if the buffer is currently in use (fcGetErrorCode returns `MSG_BUF_BUSY`). In that case retry later.

3.3.8 CLOSE A CONNECTION

Once the measurement finished, closing the connection to the FlexCard is done by calling the function fcbClose.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

4 API DESCRIPTION

This chapter describes the application programming interface in detail. Each section represents a group of operations dedicated to a common purpose (configuring, initializing, receiving…). For each group, the data definition (structures and enumerations) is first described, followed by the API functions sometimes illustrated with code samples.

For additional API descrition, which depends on the used operating system and/or used FlexCard device, please refer to the following major chapters.

4.1 GENERAL

The FlexCard API uses a well-defined naming convention. Each function, structure or enumeration is prefixed with fc or fcb. The prefix fcb (fcBase) stands for a function, a structure or an enumeration which is only available in the fcBase API. Functions, structures or enumerations which are prefixed with fc are not limited to the fcBase API and could also be available in other FlexCard APIs.

Each function of this library (except some error handling functions) returns an error code. If the return value is equal to zero, no error occurred. A number greater than zero indicates an error. To get more information about it, use the error handling functions described in chapter 4.3.

Some functions will allocate memory for you. In such a case the fcFreeMemory function needs to be called to release this memory.

4.2 OVERVIEW CHANGES

4.2.1 FROM S1V0-F TO S2V0-F

Change	Reason	Page	Remark
Definition of type fcQuad corrected	Portability	35	Downwardly compatible. Works with applications which are designed for S1V0-F.
Enumeration fcTransceiverState added	New feature	37	
Function fcbSetTransceiverState added	New feature	91	
Function fcbGetTransceiverState added	New feature	92	
Structure fcMsgBufCfgTx modified. New configuration options TxAcknowledgeShowNullFrames and TxAcknowledgeShowPayload added. TxAcknowledge packets work in all transmission modes.	Feature extended	54	Downwardly compatible. Works with applications which are designed for S1V0-F, if the reserved member of this structure was set to zero.
New member fcNoticiationTypeWakeup for enumeration fcNotifyType added for getting notification if one of the transceivers has detected a wakeup event.	Feature extended	65	Downwardly compatible. Works with applications which are designed for S1V0-F.
Function fcbNotificationPacket added	New feature	67	
Structure fcInfoPacket modified. Rate and offset	Feature	68	Downwardly compatible.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Change	Reason	Page	Remark
correction information added.	extended		Works with applications which are designed for S1V0-F.
Structure fcFlexRayFrame modified. Timestamp information added.	Feature extended.	68	Downwardly compatible. Works with applications which are designed for S1V0-F.
Structure fcTxAcknowledgePacket modified. Additional information about the transmitted frame added.	Feature extended.	70	Downwardly compatible. Works with applications which are designed for S1V0-F.
Structure fcNotificationPacket added	New feature	76	
Structure fcPacket modified. fcNotificationPacket information added	Feature extended.	79	Downwardly compatible. Works with applications which are designed for S1V0-F.
Enumeration fcErrorPacketFlag extended.	Feature extended.	80	Downwardly compatible. Works with applications which are designed for S1V0-F.

4.2.2 FROM S2V0-F TO S2V2-F

Change	Reason	Page	Remark
PMC functions added: fcbSetCCIndex, fcbGetCCIndex, fcbSetTermination, fcbGetTermination PMC Enumerations added: fcBusChannel, fcBusType	New features	150	
Added new trigger functionality for FlexCard Cyclone II and FlexCard Cyclone SE. Triggers can be OR-ed now.	Feature extended	58	
Added Xenomai support function for event handling	New feature	159	

4.2.3 FROM S2V2-F TO S3V0-F

Change	Reason	Page	Remark
Added Self synchronization for FlexCard Cyclone II (SE)	New features	145	Firmware-Version S3V0-F is needed

4.2.4 FROM S3V0-F TO S4V0-F

Change	Reason	Page	Remark
Added CAN API for FlexCard Cyclone II (SE)	New features	130	Firmware-Version S4V0-F is needed
Added function fcbResetTimestamp.	New feature	59	
Added function fcbGetNumberCcs.	New feature	58	
Added function fcbSetContinueOnPacketOverflow.	New feature	58	
Added function fcbCalculateMacrotickOffset.	New feature	101	
Added function fcbGetCcTimerConfig.	New feature	101	
Added function fcbSetCcTimerConfig.	New feature	100	

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Change	Reason	Page	Remark
Added function fcbCheckVersion	New feature	43	
New packets CAN packet and CAN error packet.	New feature	77, 78	
Extended enumeration fcPacketType	Feature extended	80	Downwardly compatible.
Extended enumeration fcCC	Feature extended	36	Downwardly compatible.
Extended enumeration fcTriggerConditionEx	Feature extended	62	Downwardly compatible.
Structure fcTriggerExInfoPacket modified. Reserved1 added	Feature extended	76	Downwardly compatible.
Structure fcCcTimerCfg added.	New feature	56	
Enumeration fcCyclePos added.	New feature	51	
Enumeration fcNotificationType modified. fcNotificationTypeCcTimer added.	Feature extended	65	Downwardly compatible.
Enumeration fcMemoryType modified. fcMemoryTypeInfoV2 added.	Feature extended	33	Downwardly compatible.
Structure fcInfoV2 added.	New feature	86	
Added function fcbGetEnumFlexCardsV2.	New feature	86	
Added function fcbReinitializeCcMessageBuffer	New feature	57	
Added function fcbReinitializeCcMessageBufferSelfSynchronization	New feature	148	
Added function fcbGetCurrentTimeStamp	New feature	59	

4.2.5 FROM S4V0-F TO S4V2-F

Change	Reason	Page	Remark
CAN API is supported by FlexCard Cyclone II (SE) and FlexCard PMC/PCI.	New features	130	Firmware-Version S4V2-F is needed
Added additional Linux API.	New feature	156	
Xenomai: Added thread-safe function for event handling.	New feature	158	
Added new thread-safe FlexRay API for all supported devices.	New features	107	
Extended enumeration fcNotificationType.	Feature extended	65	Downwardly compatible.
Added thread-safe function for event handling.	New feature	65	
Structure fcFlexRayFrame modified. AsyncMode added.	Feature extended	68	Downwardly compatible.

4.2.6 FROM S4V2-F TO S5V1-F

Change	Reason	Page	Remark
Structure fcMemoryType modified. fcMemoryTypeInfoHwSW added.	Feature extended	33	Downwardly compatible.
Enumeration fcFlexCardDeviceId modified. fcFlexCardPMCII added.	Feature extended	39	Downwardly compatible.
Structure fcVersionCC modified. IncorrectPhysicalLayer added.	Feature extended	41	Downwardly compatible.
Structure fcInfoHw added.	New features	42	
Structure fcInfoSw added.	New features	43	
Structure fcInfoHwSw added.	New features	43	
Added function fcbGetEnumFlexCardsV3	New features	44	

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Change	Reason	Page	Remark
Added function fcbGetInfoFlexCard	New features	47	
Added function fcbSetUserDefinedCardId	New features	48	Firmware-Version S5V1-F is needed
Added function fcbGetUserDefinedCardId	New features	48	Firmware-Version S5V1-F is needed
Added function fcbFRSetHardwareAcceptanceFilter	New features	125	Firmware-Version S5V1-F is needed
Structure fcFlexRayFrame modified. FrameCRC added.	Feature extended	68	Downwardly compatible.
Structure fcTxAcknowledgePacket modified. ValidFrame, SyntaxError, ContentError added.	Feature extended	70	Downwardly compatible.
Structure fcCANMonitoringMode modified. fcCANMonitoringSilent, fcCANMonitoringActive, fcCANMonitoringPassive added.	Feature extended	132	Downwardly compatible.
Structure fcCANBufCfgTx modified. newData added.	Feature extended	137	Downwardly compatible.
Structure fcCANBufCfgRemoteTx modified. newData added.	Feature extended	139	Downwardly compatible.
Added FlexCard PMC II description	New features	150	Firmware-Version S5V1-F is needed
Enumeration fcBusChannel modified. fcBusChannel5 to fcBusChannel8 added.	Feature extended	150	Downwardly compatible.

4.3 ERROR HANDLING

Almost every function in this library returns with an error status number. The meaning of this status code can be retrieved with the following functions and enumerations. Additional to this status code it is possible to get hints about the error if you use the tool fcTracerControl.exe. For more information about the tracing tool please refer to Tracing.

Information
In a few situations you will not get a meaningful error text. This happens for example if the device driver reports an error to the API. In such a case only the error code ACTION_FAILED is returned. To get a more detailed error description it may be helpful to use the tracing module.

4.3.1 TYPE DEFINITIONS

4.3.1.1 FCERROR

This type provides information about an error. A zero value means no error occurred. To extract the detailed information about an error, use the functions fcGetErrorType, fcGetErrorText and fcGetErrorCode.

```
Typedef unsigned int fcError;
```

Information
fcError is not equivalent to fcErrorCode (see fcErrorCode)

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

4.3.2 ENUMERATIONS

4.3.2.1 FCERRORCODE

This enumeration contains all error codes which are reported by the fcBase API. To extract the error code from a fcError use the fcGetErrorCode function. To get information for the error code use the fcGetErrorText function. For detailed error description please refer to the Headerfile *fcBaseTypes.h*.

See Also

fcGetErrorCode, fcGetErrorText, fcGetErrorType

4.3.2.2 FCERRORTYPE

This enumeration defines the type of error information. To get the `fcErrorType` from a fcError, use the fcGetErrorType function.

```
Typedef enum fcErrorType
{
      fcErrorTypeSuccess       = 0,
      fcErrorTypeInformation   = 1,
      fcErrorTypeWarning       = 2,
      fcErrorTypeError         = 3,
}fcErrorType;
```

Members

fcErrorTypeSuccess
: No error.

fcErrorTypeInformation
: The error should be treated as an information message. The function has succeeded but wants to give additional information.

fcErrorTypeWarning
: The error should be treated as a warning message. The function has succeeded but the input parameters are modified or not completely accepted.

fcErrorTypeError
: The error should be treated as an error message. The function has failed.

See Also

fcGetErrorType, fcGetErrorText, fcGetErrorCode

4.3.3 FCGETERRORCODE

This function returns the error code for a given error. A zero value indicates no error occurred. The list of all error codes can be found in the fcErrorCode enumeration (see *fcBaseTypes.h*).

```
fcErrorCode fcGetErrorCode(
      fcError error
);
```

Parameters

error
: [IN] An error value of type fcError

Return values

Error code

See Also

fcErrorCode, fcGetErrorType, fcGetErrorText

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

4.3.4 FCGETERRORTYPE

This function returns the error type for a given error. Please, refer to fcErrorType for more details.

```
fcErrorType fcGetErrorType(
      fcError error
);
```

Parameters

error

[IN] An error value of type fcError

Return values

Error type

See Also

fcErrorType, fcGetErrorCode, fcGetErrorText

Example

```
fcError e = fcbFRSetCcConfigurationChi(hFlexCard,eCC,pszChi);
if (0 != e)
{
      // oops, something went wrong
      fcErrorType etype = fcGetErrorType(e);
      if (fcErrorTypeInformation == etype || fcErrorTypeWarning == etype)
      {
            // ok, the function succeeds, but the function wants to give us some
            // information
            PrintInfo(e);
      }
      else
      {
            PrintError(e);
      }
}
```

4.3.5 FCGETERRORTEXT

This function returns the corresponding error text for a given error. To free the memory which was allocated by this function, please use the function fcFreeMemory with the type *fcMemoryTypeString* (see fcMemoryType). Some text will be generated at runtime to provide a more detailed error description.

```
fcError fcGetErrorText(
      fcError error,
      char** szText
);
```

Parameters

error

[IN] An error value of type fcError for which an error text should be returned.

szText

[OUT] Address of a char pointer which holds the address for the generated error text.

Return values

If the function succeeds (return value is zero), the parameter `szText` contains the requested error text. If the function fails `szText` isn't valid. The fcErrorCode `NULL_PARAMETER` is returned if the *szText* parameter is a null pointer, `TEXT_NOT_DEFINIED` if no error text for the given error could be found, or `MEMORY_ALLOCATION_FAILED` to indicate that the memory allocation for the error text failed.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Example

```
fcError e = fcbOpen(&hFlexCard,flexcardId);
if (fcErrorTypeSuccess != fcGetErrorType(e))
{
      char* szErrorText = NULL;
      if (0 == fcGetErrorText(e, &szErrorText))
      {
            // Print the error text and then free up the memory
            PrintErrorText(szErrorText);
            fcFreeMemory(fcMemoryTypeString, reinterpret_cast<void*>(szErrorText));
      }
}
```

See Also

fcFreeMemory, fcGetErrorType, fcGetErrorCode

4.4 MEMORY HANDLING

As the API allocates memory for you, it has to free up this memory. For this task the API provides the function fcFreeMemory which frees only the memory allocated by a function from the API. The reason why the API provides this mechanism is that your application may be linked to a different C/C++ run-time library than this library. Allocating memory in one module and freeing it in another one (with different run-time libraries) may fail or cause a run-time error. Another reason for this implementation is that the API can use its own memory management in order to reuse the memory blocks.

4.4.1 ENUMERATIONS

4.4.1.1 FCMEMORYTYPE

This enumeration defines the memory types needed to release the memory allocated by the functions fcGetErrorText, fcbGetEnumFlexCards (Obsolete), fcbGetEnumFlexCardsV2 (Obsolete), fcbGetEnumFlexCardsV3, fcbGetInfoFlexCard and fcbReceive.

```
Typedef enum fcMemoryType
{
      fcMemoryTypeString,
      fcMemoryTypeInfo,
      fcMemoryTypePacket,
      fcMemoryTypeInfoV2,
      fcMemoryTypeInfoHwSw,
} fcMemoryType;
```

Members

fcMemoryTypeString
: Memory is of the type “char[]”

fcMemoryTypeInfo
: Memory is of the type “fcInfo”

fcMemoryTypePacket
: Memory is of the type “fcPacket”

fcMemoryTypeInfoV2
: Memory is of the type “fcInfoV2”

fcMemoryTypeInfoHwSw
: Memory is of the type “fcInfoHwSw”

See Also

fcFreeMemory, fcGetErrorText, fcbGetEnumFlexCards (Obsolete), fcbGetEnumFlexCardsV2 (Obsolete), fcbGetEnumFlexCardsV3, fcbGetInfoFlexCard, fcbReceive

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

4.4.2 FCFREEMEMORY

This function releases the memory allocated by one of the API functions fcGetErrorText, fcbGetEnumFlexCards (Obsolete), fcbGetEnumFlexCardsV2 (Obsolete), fcbGetEnumFlexCardsV3, fcbGetInfoFlexCard or fcbReceive. The allocated memory can be used as long as necessary. If the connection to the FlexCard is closed, all allocated memory blocks must be released with this function.

```
fcError fcFreeMemory(
      const fcMemoryType memoryType,
      void* p
);
```

Parameters

memoryType

Type of memory to be released.

P

Pointer to the memory to be released.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information. The fcErrorCode `INVALID_PARAMETER` is returned if the memoryType parameter wasn't correct, `NULL_PARAMETER` if the p parameter is a null pointer.

Example

```
fcError e = fcbOpen(&hFlexCard, flexcardId);
if (0 != e)
{
      char* szErrorText = NULL;
      if (0 == fcGetErrorText(e, &szErrorText))
      {
            // Print the error text and then free up the memory
            PrintErrorText(szErrorText);
            fcFreeMemory(fcMemoryTypeString, reinterpret_cast<void*>(szErrorText));
      }
}
```

See Also

fcMemoryType, fcGetErrorText, fcbGetEnumFlexCards (Obsolete), fcbGetEnumFlexCardsV2 (Obsolete), fcbGetEnumFlexCardsV3, fcbGetInfoFlexCard, fcbReceive

4.5 INITIALIZATION

The following section describes the data structures and features used for initialization.

4.5.1 TYPE DEFINITIONS

4.5.1.1 FCHANDLE

It defines a handle to a FlexCard object. A handle is returned by the function fcbOpen (assuming that a valid FlexCard identifier has been used).

```
Typedef void* fcHandle;
```

4.5.1.2 FCBYTE

Unsigned 8-Bit value.

```
Typedef unsigned char fcByte;
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

4.5.1.3 FCWORD

Unsigned 16-Bit value.

```
Typedef unsigned short fcWord;
```

4.5.1.4 FCDWORD

Unsigned 32-Bit value.

```
Typedef unsigned int fcDword;
```

4.5.1.5 FCQUAD

Unsigned 64-Bit value.

```
typedef unsigned long long fcQuad;
```

4.5.2 ENUMERATIONS

4.5.2.1 FCBUSTYPE

This enumeration defines the available FlexCard bus types.

```
Typedef enum fcBusType
{
      fcBusTypeFlexRay = 0,
      fcBusTypeCAN,
} fcBusType;
```

Members

fcBusTypeFlexRay
: The FlexRay bus is selected.

fcBusTypeCAN
: The CAN bus is selected.

See Also

fcVersionCC

4.5.2.2 FCCHANNEL

This enumeration defines the available channel combination of the FlexCard.

```
Typedef enum fcChannel
{
      fcChannelNone = 0x00,
      fcChannelA    = 0x01,
      fcChannelB    = 0x02,
      fcChannelBoth = fcChannelA | fcChannelB,
} fcChannel;
```

Members

fcChannelNone
: No FlexRay channel selected

fcChannelA
: Only FlexRay channel A is selected

fcChannelB
: Only FlexRay channel B is selected

fcChannelBoth
: FlexRay channel A and B are selected

See Also

fcMsgBufCfg

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

4.5.2.3 FCCC

This enumeration defines the available FlexCard communication controller index depending on the communication bus type.

```
Typedef enum fcCC
{
      fcCC1 = 0x00,
      fcCC2 = 0x01,
      fcCC3 = 0x02,
      fcCC4 = 0x03,
      fcCC5 = 0x04,
      fcCC6 = 0x05,
      fcCC7 = 0x06,
      fcCC8 = 0x07,
} fcCC;
```

Members

fcCC1
: The communication controller 1 is selected.

fcCC2
: The communication controller 2 is selected.

fcCC3
: The communication controller 3 is selected.

fcCC4
: The communication controller 4 is selected.

fcCC5
: The communication controller 5 is selected.

fcCC6
: The communication controller 6 is selected.

fcCC7
: The communication controller 7 is selected.

fcCC8
: The communication controller 8 is selected.

4.5.2.4 FCSTATE

This enumeration defines the possible communication controller POC states (FlexRay Protocol Specification: vPOC!State). For more details about communication controller POC states, please refer to [3].

```
Typedef enum fcState
{
      fcStateUnknown,
      fcStateConfig,
      fcStateNormalActive,
      fcStateNormalPassive,
      fcStateHalt,
      fcStateReady,
      fcStateStartup,
      fcStateWakeup,
      fcStateMonitorMode,
} fcState;
```

Members

fcStateUnknown
: Communication controller state is not known.

fcStateConfig
: Communication controller is in CONFIG state.

fcStateNormalActive
: Communication controller is in NORMAL_ACTIVE state.

fcStateNormalPassive
: Communication controller is in NORMAL_PASSIVE state.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

fcStateHalt
Communication controller is in HALT state.
fcStateReady
Communication controller is in READY state.
fcStateStartup
Communication controller is in STARTUP state.
fcStateWakeup
Communication controller is in WAKEUP state.
fcStateMonitorMode
Communication controller is in MONITORMODE state.

See Also
fcbFRGetCCState, fcbFRMonitoringStart

4.5.2.5 FCWAKEUPSTATUS

This enumeration defines the possible communication controller wakeup states (FlexRay Protocol Specification: vPOC!WakeupStatus). For more details about communication controller wakeup states, please refer to [3].

```
Typedef enum fcWakeupStatus
{
      fcWakeupStatusUndefined = 0,
      fcWakeupStatusReceiveHeader,
      fcWakeupStatusReceiveWUP,
      fcWakeupStatusCollisionHeader,
      fcWakeupStatusCollisionWUP,
      fcWakeupStatusCollisionUnknown
      fcWakeupStatusTransmitted,
} fcWakeupStatus;
```

Members
fcWakeupStatusUndefined
FlexRay Protocol Specification: UNDEFINED
fcWakeupStatusReceiveHeader
FlexRay Protocol Specification: RECEIVE_HEADER
fcWakeupStatusReceiveWUP
FlexRay Protocol Specification: RECEIVE_WUP
fcWakeupStatusCollisionHeader
FlexRay Protocol Specification: COLLISION_HEADER
fcWakeupStatusCollisionWUP
FlexRay Protocol Specification: COLLISION_WUP
fcWakeupStatusCollisionUnknown
FlexRay Protocol Specification: COLLISION_UNKNOWN
fcWakeupStatusTransmitted
FlexRay Protocol Specification: TRANSMITTED

See Also
fcStatusWakeupInfo, fcStatusPacket

4.5.2.6 FCTRANSCEIVERSTATE

This enumeration defines the different states of the FlexRay transceivers.

```
Typedef enum fcTransceiverState
{
      fcTransceiverNormal,
      fcTransceiverSleep,
} fcTransceiverState;
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Members

fcTransceiverNormal

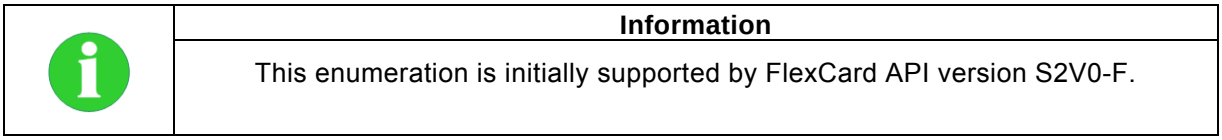
Transceiver is in normal mode and is able to transmit and receive data via the FlexRay bus.

fcTransceiverSleep

Transceiver is in low power mode and is not able to transmit and receive data, but is able to detect wake-up events on the bus. If a wakeup is detected the event fcNotificationTypeFRWakeup is fired.

See Also

fcbFRSetTransceiverState, fcbFRGetTransceiverState



4.5.2.7 FCSYMBOLTYPE

This enumeration defines the supported communication symbols when the communication controller is in POC state NORMAL_ACTIVE. For more details about these symbols, please refer to the FlexRay Protocol Specification. To send a wakeup symbol (WUS) or collision avoidance symbol (CAS), refer to the function fcbFRMonitoringStart.

```
Typedef enum fcSymbolType
{
      fcMediaAccessTestSymbolType = 1,
} fcSymbolType;
```

Members

fcMediaAccessTestSymbolType

Media Access Test Symbol (MTS)

See Also

fcbFRTransmitSymbol

4.5.2.8 FCCCTYPE

This enumeration defines the communication controller types supported by the API. The parameter *CCType* of the structure fcVersionCC, which is returned by the functions fcbGetEnumFlexCardsV3, indicates the communication controller used by the FlexCard.

```
Typedef enum fcCCType
{
      Undefined = 0,
      FreeScale_FPGA,
      Bosch_E_Ray,
      Bosch_DCAN,
}fcCCType;
```

Members

Undefined

Undefined communication controller.

FreeScale_FPGA

Communication controller is a FreeScale FPGA

Bosch_E_Ray

Communication controller is a Bosch E-Ray

Bosch_DCAN

Communication controller is a Bosch DCAN

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

See Also

fcVersionCC, fcbGetEnumFlexCardsV3

Remarks

Current FlexCard hardware (FlexCard PMC (II), FlexCard Cyclone II (SE)) supported by the latest driver versions integrate Bosch E-Ray and D-CAN communication controllers.

4.5.2.9 FCMONITORINGMODES

This enumeration defines the different modes available, used to monitor a FlexRay cluster.

```
Typedef enum fcMonitoringModes
{
      fcMonitoringNormal,
      fcMonitoringDebug,
      fcMonitoringDebugAsynchron,
      fcMonitoringDebugAsynchronBeforeStartup,
} fcMonitoringModes;
```

Members

fcMonitoringNormal

First, the FlexCard tries to synchronize itself with the cluster. Once the synchronization succeeds, the FlexCard enters in the NORMAL_ACTIVE state and is able to transmit and receive FlexRay frames, symbols and errors, as previously configured.

fcMonitoringDebug

The FlexCard does not try to synchronize itself with the cluster and is only able to receive FlexRay frames, symbols and errors from the FlexRay bus. This mode does not allow transmission; it is therefore not possible to perform a start-up or a wakeup. This mode is adapted for debugging purpose (e.g. start up of a FlexRay network fails).

Note: This mode is an E-Ray specific feature and is thereby only available for the E-Ray FlexCard version. To receive frames within this mode, you have to configure receive buffers. The FIFO receive buffers aren't working in this mode.

fcMonitoringDebugAsynchron

This debug operation mode of the FlexCard allows the reception of all frames without any message buffer and controller configuration. The only parameter to be set is the baudrate Register 0x0090: 10 Mbit/s: 0x00000000, 5 Mbit/s: 0x00004000, 2.5 Mbit/s: 0x00008000. This mode does not allow transmission. It is therefore not possible to perform a start-up or a wakeup. This mode is adapted for debugging purpose (e.g. start up of a FlexRay network fails or to monitor an unknown network). The timestamp accuracy in this mode is +/-2 µs. Incorrect data will be interpreted as received FlexRay frames. In this case the Valid Frame Bit is not set.

fcMonitoringDebugAsynchronBeforeStartup

This mode combines the mode fcMonitoringDebugAsynchron and fcMonitoringNormal. The mode fcMonitoringDebugAsynchron is used to receive all frames during startup. Unlike fcMonitoringDebug this mode allows to send synch frames. After the startup completed successfully, the FlexCard switchs directly to the mode fcMonitoringNormal.

See Also

fcbFRMonitoringStart

4.5.2.10 FCFLEXCARDDEVICEID

This enumeration defines the different FlexCard types. The driver supports all FlexCard types beside FlexCard PXI (aka FlexCard Cyclone II PXI).

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef enum fcFlexCardDeviceId
{
      fcNoDevice             = 0,
      fcFlexCardCycloneII    = 5,
      fcFlexCardCycloneIIPXI = 6,
      fcFlexCardPMC          = 7,
      fcFlexCardCycloneIISE  = 8,
      fcFlexCardPMCII        = 9,
} fcFlexCardDeviceId;
```

Members

fcNoDevice
: No FlexCard device was detected.

fcFlexCardCycloneII
: The device identifier of a FlexCard Cyclone II.

fcFlexCardCycloneIIPXI
: The device identifier of a FlexCard PXI.

fcFlexCardPMC
: The device identifier of a FlexCard PMC / PCI.

fcFlexCardCycloneIISE
: The device identifier of a FlexCard Cyclone II SE.

fcFlexCardPMCII
: The device identifier of a FlexCard PMC II.

See Also

fcInfoHw, fcbGetEnumFlexCardsV3, fcbGetInfoFlexCard

4.5.3 STRUCTURES

4.5.3.1 FCNUMBERCC

This structure provides information about the available number of communication controllers of the FlexCard hardware.

```
Typedef struct fcNumberCC
{
      fcByte FlexRay;
      fcByte CAN;
      fcByte LIN;
      fcByte MOST;
      fcByte Reserved[4];
} fcNumberCC;
```

Members

FlexRay
: Number of FlexRay communication controllers.

CAN
: Number of CAN communication controllers.

LIN
: Number of LIN communication controllers. This parameter is currently not supported.

MOST
: Number of MOST communication controllers. This parameter is currently not supported.

Reserved
: Reserved for future use.

See Also

fcInfoHw, fcbGetNumberCcs, fcbGetInfoFlexCard

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Information
This structure is initially supported by FlexCard API version S4V0-F.

4.5.3.2 FCVERSIONCC

This structure provides version information about the available FlexCard communication controllers.

```
Typedef struct fcVersionCC
{
       fcBusType BusType;
       fcCC CCIndex;
       fcCCType CCType;
       fcVersionNumber CCVersion;
       fcVersionNumber Protocol;
       fcVersionCC* pNext;
       fcDword IncorrectPhysicalLayer : 1;
       fcDword Reserved[2];
} fcVersionCC;
```

Members

BusType
: Communication controller bus type

CCIndex
: Communication controller identifier

CCType
: Communication controller type

CCVersion
: Communication controller version

Protocol
: Protocol version of the bus type

pNext
: Pointer to the next cc version. If the pointer is NULL, there are no more cc versions available.

IncorrectPhysicalLayer
: Physical layer module detection. A value <> 0 indicates a mismatch between communication controller type and physical layer module.

Reserved
: Reserved for future use

See Also

fcbGetNumberCcs

Information
This structure is initially supported by FlexCard API version S4V0-F.

4.5.3.3 FCVERSIONNUMBER

This structure describes the version of a FlexCard component (hardware or software). The function fcbGetEnumFlexCardsV3 returns the version numbers for the FlexCard components.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef struct fcVersionNumber
{
      fcDword Major;
      fcDword Minor;
      fcDword Update;
      fcDword Release;
} fcVersionNumber;
```

Members

Major
: An increment indicates a modification which is not downwardly compatible

Minor
: An increment indicates a light-weight modification

Update
: Indicates an update (bug fix) for a minor version

Release
: 0 indicates a release version, greater than 0 indicates a test version

See Also

fcInfoHw, fcInfoSw, fcbGetEnumFlexCardsV3, fcbGetInfoFlexCard

4.5.3.4 FCINFOHW

This structure provides information about the hardware components of a FlexCard.

```
typedef struct fcInfoHw
{
      fcQuad Serial;
      fcFlexCardDeviceId DeviceId;
      fcVersionNumber VersionFirmware;
      fcVersionNumber VersionHardware;
      fcNumberCC SupportedCCs;
      fcNumberCC LicensedCCs;
      fcNumberCC UseableCCs;
      fcVersionCC* pVersionCC;
      fcDword Reserved[8];
} fcInfoHw;
```

Members

Serial
: FlexCard serial number. A zero value indicates a non-valid FlexCard serial number.

DeviceId
: FlexCard Device ID

VersionFirmware
: Firmware (gateway software) version

VersionHardware
: FlexCard hardware version

SupportedCCs
: Possible FlexCard communication controller counts with the hardware.

LicensedCCs
: Licensed FlexCard communication controller counts with the hardware.

UseableCCs
: Useable FlexCard communication controller counts for the software.

pVersionCC
: Pointer to version information about the useable communication controllers.

Reserved
: Reserved for future use

See Also

fcInfoSw, fcInfoHwSw, fcbGetEnumFlexCardsV3, fcbGetInfoFlexCard

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Information
This structure is initially supported by FlexCard API version S5V1-F.

4.5.3.5 FCINFOSW

This structure provides information about the software components of a FlexCard.

```
typedef struct fcInfoSw
{
      fcVersionNumber VersionBaseDll;
      fcVersionNumber VersionDeviceDriver;
      fcDword LicensedForLinuxDriver : 1;
      fcDword LicensedForWindowsDriver : 1;
      fcDword LicensedForXenomaiDriver : 1;
      fcDword Reserved[4];
} fcInfoSw;
```

Members

VersionBaseDll
: DLL Base Version

VersionDeviceDriver
: Device driver version

LicensedForLinuxDriver
: Valid license for FlexCard Linux driver

LicensedForWindowsDriver
: Valid license for FlexCard Windows driver

LicensedForXenomaiDriver
: Valid license for FlexCard Xenomai driver

Reserved
: Reserved for future use

See Also

fcInfoHw, fcInfoHwSw, fcbGetEnumFlexCardsV3, fcbGetInfoFlexCard

Information
This structure is initially supported by FlexCard API version S5V1-F.

4.5.3.6 FCINFOHWSW

This structure provides information about the components, the identifiers and the current device state of a FlexCard. If more than one FlexCard is detected on the system, the fcbGetEnumFlexCardsV3 function returns a linked list of this structure; the function fcbGetInfoFlexCard function returns an item of this structure.

```
typedef struct fcInfoHwSw
{
      fcDword FlexCardId;
      fcDword UserDefinedCardId;
      fcInfoSw InfoSoftware;
      fcInfoHw InfoHardware;
      fcDword Busy : 1;
      fcInfoHwSw* pNext;
      fcDword Reserved[2];
} fcInfoHwSw;
```

Members

FlexCardId
: Unique number used to identify a FlexCard. This id is required to open a connection to the FlexCard.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

UserDefinedCardId
: User defined number used to identify a FlexCard. This id is not unique! A zero value indicates a non-valid or non-existing identifier.

InfoSoftware
: Information about software components of the FlexCard.

InfoHardware
: Information about hardware components of the FlexCard.

Busy
: The current device state. A value <> 0 indicates a connection to this FlexCard is already opened.

pNext
: Pointer to the next available FlexCard. If no other FlexCard exists, pNext is a null pointer.

Reserved
: Reserved for future use

See Also

fcInfoHw, fcInfoSw, fcbGetEnumFlexCardsV3, fcbGetInfoFlexCard

	Information
	This structure is initially supported by FlexCard API version S5V1-F.

4.5.4 FCBGETENUMFLEXCARDSV3

This function returns a linked list of the installed FlexCards found on the system. To free the memory, which was allocated by this function, please use the function fcFreeMemory with type *fcMemoryTypeInfoHwSw.*

```
fcError fcbGetEnumFlexCardsV3(
      fcInfoHwSw** pInfoHwSw,
      bool getBusyDevices
)
```

Parameters

pInfoHwSw
: [OUT] linked list of fcInfoHwSw objects

getBusyDevices
: [IN] Show busy devices in linked list. Set this parameter to false to get a linked list of the unused FlexCards found on the system.

Return values

If the function succeeds, the return value is 0. If the function fails the content of *pInfoHwSw* is not valid. The error code NULL_PARAMETER is returned if *pInfoHwSw* parameter is a null pointer. If the memory allocation fails, the error code MEMORY_ALLOCATION_FAILED is returned.

Remarks

If the function succeeds, there will always be one valid fcInfoHwSw structure regardless if there is a FlexCard in the system or not. This functionality is given to provide version information about the DLL / library. The version information concerning the hardware is only valid if the identifier **(*pInfoHwSw->FlexCardId*) is not 0**.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Information
This function allocates memory for use. To prevent memory leaks you have to free it up by calling the function fcFreeMemory with the type *fcMemoryTypeInfoHwSw*.

See Also

fcInfoHwSw

Example

```
//
// Get the installed FlexCards in the system and print the version numbers
//
fcInfoHwSw* pInfoHwSw = NULL;
fcError e = fcbGetEnumFlexCardsV3(&pInfoHwSw, true);
if (0 != e) return; // if it fails, return directly

fcInfoHwSw* pLoop = pInfoHwSw;
while (NULL != pLoop)
{
      // if FlexCard Id is equal to zero, we got NO FlexCard in the system
      bool bFlexCardAvailable = (0 != pLoop->FlexCardId);

      printf("\r\nFlexCard Id\t:  ");
      if (bFlexCardAvailable) printf("%d\r\n",pLoop->FlexCardId);
      else printf("not available\r\n");

      // if FlexCard isn't in use, we print out the version numbers
      if (bFlexCardAvailable && (0 == pLoop->Busy))
      {/*... print out the version numbers ...*/}
      else printf("FlexCard is in use\r\n");

      pLoop = pLoop->pNext; // get the next flexcard
}

// Don't forget to free the memory
fcFreeMemory(fcMemoryTypeInfoHwSw, pInfoHwSw);
```

Information
This function is initially supported by FlexCard API version S5V1-F.

4.5.5 FCBCHECKVERSION

This function checks the version combination of the installed driver ("fcBase.DLL and fce05xp.SYS / fce052k.SYS" or "flexcard.ko and libfcBase.so") and the FlexCard firmware. This function can only be called after fcbOpen.

```
fcError fcbCheckVersion(
      fcHandle hFlexCard
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

Return values

If the function succeeds, the return value is 0 and the opened FlexCard can be used with the SYS and DLL. If the return value is <> 0, use the functions described in the section Error Handling to get extended error information and the opened FlexCard cannot be used with SYS and DLL versions.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Example

```
fcError e = fcbCheckVersion(hFlexCard);
if (0 != e)
{
     // Error handling
}
```

	Information
	This function is initially supported by FlexCard API version S4V0-F.

4.5.6 FCBOPEN

This function opens a connection to a specified FlexCard and returns a handle to this FlexCard. The function modifies some communication controller registers (e.g. set the communication controller in its configuration state, *fcStateConfig*) and all message buffers are configured as receive FIFO buffers with maximum payload length.

```
fcError fcbOpen(
      fcHandle* phFlexCard,
      fcDword flexCardId
)
```

Parameters

phFlexCard

[OUT] Handle to a specific FlexCard.

flexCardId

[IN] Number which indicates the FlexCard you want to use. This identifier is stored in fcInfoHwSw objects returned by the function fcbGetEnumFlexCardsV3. Only FlexCardId greater than zero are valid FlexCard identifier.

Return values

If the function succeeds, `phFlexCard` holds a valid FlexCard handle and the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Remarks

Use the functions fcbGetEnumFlexCardsV3 to get a valid `FlexCardId`. The function fcbClose is used to close a connection previously opened with fcbOpen.

	Information
	If the FlexCard is closed and reopened, all previous (before closing) configuration settings are lost. After opening a connection it is necessary to configure the FlexCard.

See Also

fcbGetEnumFlexCardsV3, fcbClose

Example

```
...
fcInfoHwSw* pInfoHwSw = NULL;
fcHandle hFlexCard = NULL;

if (0 == fcbGetEnumFlexCardsV3(&pInfoHwSw, true))
{
      // Open the flexcard using the first flexcard identifier
      fcError e = fcbOpen(&hFlexCard, pInfoHwSw ->FlexCardId);
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
      // always free the memory which was allocated by fcbGetEnumFlexCardsV3
      fcFreeMemory(fcMemoryTypeInfoHwSw, pInfoHwSw);
      if (0 != e) // handle isn't valid
            printError(e);
}
...
```

4.5.7 FCBCLOSE

This function closes the connection to a FlexCard.

```
fcError fcbClose(
      fcHandle hFlexCard
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Remarks

If a monitoring is active, this function will first stop the monitoring and then close the connection.

See Also

fcbGetEnumFlexCardsV3, fcbOpen

Example

```
fcError e = fcbClose(hFlexCard);
if (0 == e)
{
      // This handle isn't valid anymore
      hFlexCard = NULL;
}
```

4.5.8 FCBGETINFOFLEXCARD

This function returns an item of the structure fcInfoHwSw, which provides information about the components, the identifiers and the current device state of the opened FlexCard device. To free the memory which was allocated by this function, please use the function fcFreeMemory with type *fcMemoryTypeInfoHwSw*.

```
fcError fcbGetInfoFlexCard(
      fcHandle hFlexCard,
      fcInfoHwSw** pInfoHwSw
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

pInfoHwSw

[OUT] Hard- and software information of a FlexCard.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Information
This function allocates memory for use. To prevent memory leaks you have to free it up by calling the function fcFreeMemory with the type *fcMemoryTypeInfoHwSw*.

See Also

fcbGetEnumFlexCardsV3, fcbOpen

Example

```
...
fcInfoHwSw* pInfoHwSw = NULL;
fcError e = fcbGetInfoFlexCard(hFlexCard, &pInfoHwSw);
if (0 == e)
{
      // Check open device
      …

      // always free the memory which was allocated by fcbGetInfoFlexCard
      fcFreeMemory(fcMemoryTypeInfoHwSw, pInfoHwSw);
      if (0 != e) // handle isn't valid
            printError(e);
}
...
```

Information
This function is initially supported by FlexCard API version S5V1-F.

4.5.9 FCBSETUSERDEFINEDCARDID

This function writes a persistent user ID to the FlexCard.

```
fcError fcbSetUserDefinedCardId (
      fcHandle hFlexCard,
      fcDword UserDefinedCardId
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

UserDefinedCardId

[IN] The ID that will be given to the FlexCard.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Information
This function is initially supported by FlexCard API version S5V1-F.

4.5.10 FCBGETUSERDEFINEDCARDID

This function reads the persistent ID from the FlexCard.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
fcError fcbGetUserDefinedCardId (
      fcHandle hFlexCard,
      fcDword* pUserDefinedCardId
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

UserDefinedCardId

[OUT] The user defined FlexCard ID.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

	Information
	This function is initially supported by FlexCard API version S5V1-F.

4.6 CONFIGURATION

This chapter describes the functions and data types used to configure both communication controller and hardware of a FlexCard. The configuration phase of a FlexCard is an essential part of its integration into a cluster and can not be skipped. Entering the bus parameters of an existent network is possible directly or by CHI/CANdb-Import. If one of the FlexCard configuration settings does not match the cluster ones, the FlexCard may not be able to monitor the bus. Therefore, it is highly recommended to use a configuration tool for designing a new FlexRay network. FlexConfig from *Eberspächer Electronics* is such a tool that outputs a CHI file. It automatically validates and generates for each FlexRay parameter the corresponding register values of each node in a cluster. Manual configuration of the FlexCard is also a possibility but will be a complex, time-consuming and error-prone method due to the large number of Eray registers used for configuration.

As the FlexCard uses the receive FIFO functionality from the communication controller to monitor the FlexRay frames, the fcBase API has to ensure that enough FIFO message buffers are configured, that means that not all message buffers are available for the user. Modifying the FIFO message buffers settings may affect the ability to correctly monitor the FlexRay bus.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

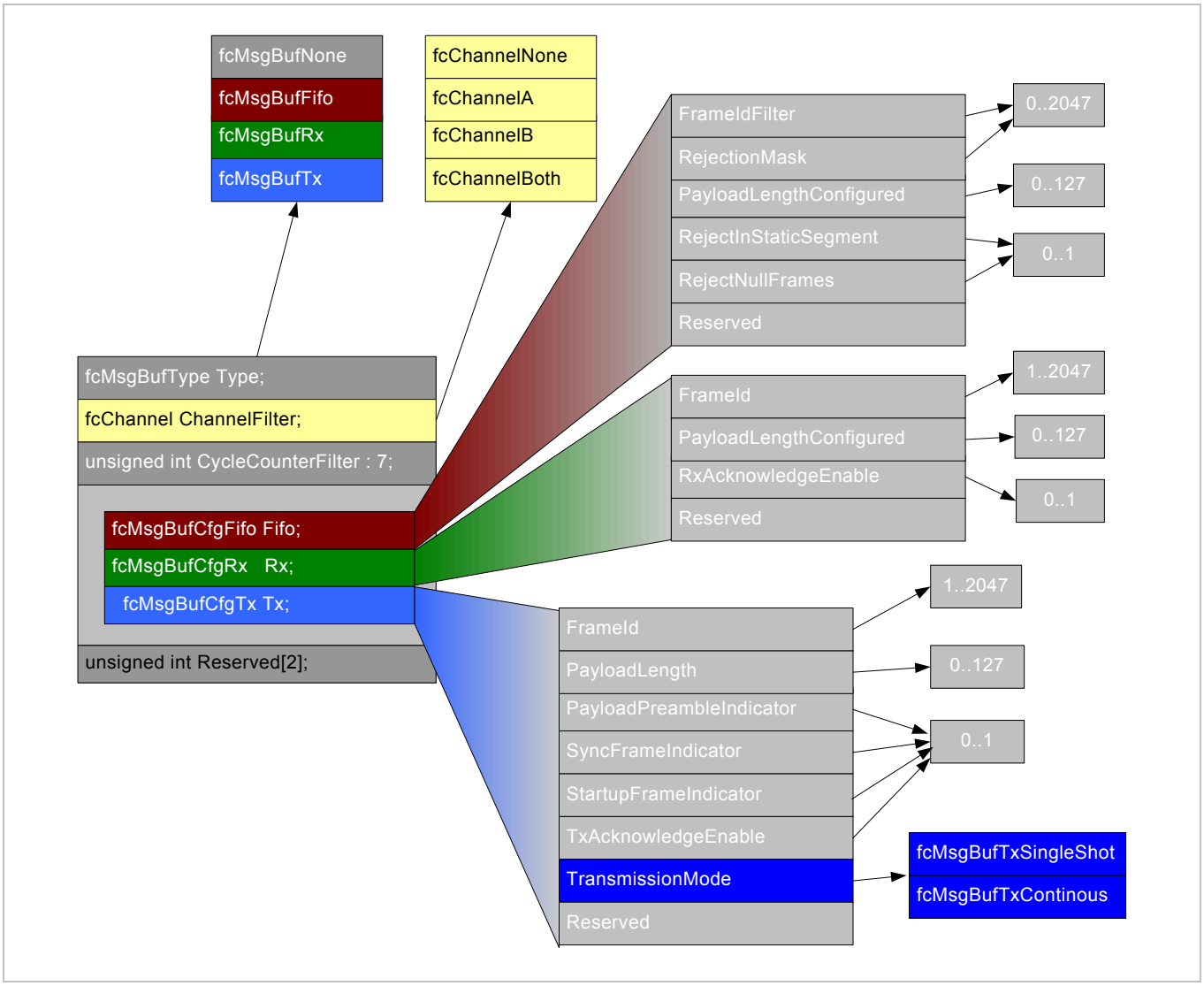


Figure 9: Overview fcbMsgBufCfg structure

4.6.1 CONSTANTS

4.6.1.1 FCPAYLOADMAXIMUM

Maximum number of 2-byte payload data words

```
const fcByte fcPayloadMaximum = 127
```

4.6.2 ENUMERATIONS

4.6.2.1 FCMSGBUFTYPE

For the transmission and reception of FlexRay frames the communication controller provides different types of message buffers. Each message buffer can be assigned with one of the following specific types.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef enum fcMsgBufType
{
      fcMsgBufNone,
      fcMsgBufRx,
      fcMsgBufTx,
      fcMsgBufFifo,
} fcMsgBufType;
```

Members

fcMsgBufNone

The message buffer is not used.

fcMsgBufRx

The message buffer is used as a receive buffer (e.g. to analyse a specific frame).

fcMsgBufTx

The message buffer is used as a transmit buffer (e.g. to transmit a message on a specific communication slot).

fcMsgBufFifo

The message buffer is used as a receive FIFO buffer. In that case, it will receive frames from different communication slots.

	Information
	In certain cases, it is not possible to receive all frames with only receive message buffers. To ensure that all frames will be received, we recommend to configure some FIFO message buffers.

See Also

fcMsgBufCfg

4.6.2.2 FCMSGBUFTXMODE

This enumeration defines the different modes of transmission.

```
Typedef enum fcMsgBufTxMode
{
      fcMsgBufTxSingleShot,
      fcMsgBufTxContinous,
}fcMsgBufTxMode;
```

Members

fcMsgBufTxSingleShot

Frame is transmitted once only if its corresponding message buffer content has been set and both frame id and cycle filter are matching. The function fcbFRTransmit sets the content of a given message buffer.

fcMsgBufTxContinous

Frame is transmitted each time when both the frame id and cycle filter are matching, regardless if its corresponding message buffer content has been set or not.

See Also

fcMsgBufCfgTx

4.6.2.3 FCCYCLEPOS

This enumeration defines various positions in a FlexRay cycle.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef enum fcCyclePos
{
      fcCyclePosNotDefined = 0,

      fcCyclePosStaticSlot,
      fcCyclePosDynamicMiniSlot,

      fcCyclePosEndStaticSegment,
      fcCyclePosStartDynamicSegment,
      fcCyclePosEndDynamicSegment,
      fcCyclePosStartSymbolWindow,
      fcCyclePosEndSymbolWindow,
      fcCyclePosStartNetworkIdleTime,
} fcCyclePos;
```

Members

fcCyclePosNotDefined
: No cycle position defined

fcCyclePosStaticSlot
: Defines the start of a static slot

fcCyclePosDynamicMiniSlot
: Defines the start of a dynamic mini slot

fcCyclePosEndStaticSegment
: Defines the end of the static segment

fcCyclePosStartDynamicSegment
: Defines the start of the dynamic segment

fcCyclePosEndDynamicSegment
: Defines the end of the dynamic segment

fcCyclePosStartSymbolWindow
: Defines the start of the symbol window

fcCyclePosEndSymbolWindow
: Defines the end of the symbol window

fcCyclePosStartNetworkIdleTime
: Defines the start of the network idle time

See Also

fcbFRCalculateMacrotickOffset

	Information
	This enumeration is initially supported by FlexCard API version S4V0-F.

4.6.3 STRUCTURES

4.6.3.1 FCMSGBUFCFGFIFO

This structure specifies the configuration of a FIFO buffer. The FIFO message buffers are used to receive FlexRay frames from different communication slots and allow therefore to receive more frames than message buffers exist.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef struct fcMsgBufCfgFifo
{
      fcDword FrameIdFilter : 11;
      fcDword RejectionMask : 11;
      fcDword PayloadLengthConfigured : 7;
      fcDword RejectInStaticSegment : 1;
      fcDword RejectNullFrames : 1;
      fcDword Reserved;
}fcMsgBufCfgFifo;
```

Members

FrameIdFilter
: Defines the acceptance filter used for frame id rejection. A zero value means that no frame is rejected.

RejectionMask
: Specifies the relevant bits used for rejection filtering.

PayloadLengthConfigured
: Defines the maximum number of 2-byte payload words received.

RejectInStaticSegment
: Set this flag to 1 to reject all received static frames of the FIFO. A zero value deactivates the FIFO static segment rejection. This feature is not available for FlexCard Cyclone I Card.

RejectNullFrames
: Set this flag to 1 to reject all received null frames of the FIFO. A zero value deactivates the FIFO null frame rejection. This feature is not available for FlexCard Cyclone I Card.

Reserved
: Reserved for future use.

	Information
	Modifying the FIFO configuration may affect the ability to receive all frames (e.g. setting the RejectInStaticSegment flag to 1 will disable the FlexCard to monitor frames in the static segment). Configuring (fcbFRConfigureMessageBuffer) the FIFO is only possible when the communication controller is in its configuration state, *fcStateConfig*. A reconfiguration (fcbFRReconfigureMessageBuffer) is not allowed for this buffer type.

See Also

fcMsgBufCfg

Example

```
// Configure fifo receive buffers
// -> Channels A+B, all frames (including null frames) on every cycles

fcMsgBufCfg cfg;
cfg.Type = fcMsgBufFifo;
cfg.ChannelFilter = fcChannelBoth;
cfg.CycleCounterFilter = 0;

cfg.Fifo.FrameIdFilter = 0;
cfg.Fifo.RejectionMask = 0;
cfg.Fifo.PayloadLengthConfigured = 127;
cfg.Fifo.RejectInStaticSegment = 0;
cfg.Fifo.RejectNullFrames = 0;

unsigned int bufferIdx = 0;
fcError e = fcbFRConfigureMessageBuffer(hFlexCard,fcCC1,&bufferIdx,cfg);
```

4.6.3.2 FCMSGBUFCFGRX

This structure specifies the configuration of a receive message buffer. This buffer type should be used to analyse a specific communication slot (=frame id).

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef struct fcMsgBufCfgRx
{
      fcDword FrameId : 11;
      fcDword PayloadLengthConfigured : 7;
      fcDword PayloadLengthMax : 7;
      fcDword RxAcknowledgeEnable: 1;
      fcDword Reserved;
} fcMsgBufCfgRx;
```

Members

FrameId

Defines the slot (=frame id) to be received in this message buffer. With the function fcbFRReconfigureMessageBuffer, this parameter can be changed while monitoring is active.

PayloadLengthConfigured

Defines the number of 2-byte payload words to be received. This parameter can be changed while monitoring is active. To do so, call the function fcbFRReconfigureMessageBuffer and set this parameter with a value between 0 and *PayloadLengthMax.* The reconfiguration of this parameter for message buffers assigned to the static segment is not allowed.

PayloadLengthMax

Defines the maximum payload reserved for this buffer in the message ram. This E-Ray specific parameter sets the range for the payload reconfiguration.

This parameter can not be changed while monitoring is active.

RxAcknowledgeEnable

Enables message buffer interrupt. This flag must be set to 1 to allow the function fcbReceive to get the received frame. This parameter can be changed while monitoring is active. To do so, call the function fcbFRReconfigureMessageBuffer.

Reserved

Reserved for future use

See Also

fcMsgBufCfg

4.6.3.3 FCMSGBUFCFGTX

This structure specifies the configuration of a transmit message buffer. This buffer type is used to transmit a frame on a specific communication slot.

```
Typedef struct fcMsgBufCfgTx
{
      fcDword FrameId : 11;
      fcDword PayloadLength : 7;
      fcDword PayloadLengthMax : 7;
      fcDword PayloadPreambleIndicator : 1;
      fcDword SyncFrameIndicator : 1;
      fcDword StartupFrameIndicator : 1;
      fcDword TxAcknowledgeEnable: 1;
      fcMsgBufTxMode TransmissionMode;
      fcDword TxAcknowledgeShowNullFrames : 1;
      fcDword TxAcknowledgeShowPayload : 1;
      fcDword Reserved : 29;
} fcMsgBufCfgTx
```

Members

FrameId

Defines the slot (=frame id) assigned to the transmit message buffer. With the function fcbFRReconfigureMessageBuffer, this parameter can be changed while monitoring is active.

PayloadLength

Defines the number of 2-byte payload words to be transmitted. This parameter can be changed while monitoring is active. To do so, call the function fcbFRReconfigureMessageBuffer and set this parameter with a value between 0 and *PayloadLengthMax.* The reconfiguration of this parameter for message buffers assigned to the static segment is not allowed.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

PayloadLengthMax

Defines the maximum payload reserved for this buffer in the message ram. This E-Ray specific parameter sets the range for the payload reconfiguration. This parameter can not be changed while monitoring is active.

PayloadPreambleIndicator

This parameter is protocol specific. For more information, refer to FlexRay Protocol Specification. With the function fcbFRReconfigureMessageBuffer, this parameter can be changed while monitoring is active.

SyncFrameIndicator

Set this flag to 1 to indicate that the frame is a sync frame. This parameter can not be changed while monitoring is active.

StartupFrameIndicator

Set this flag to 1 to indicate that the frame is a start-up frame. This parameter can not be changed while monitoring is active.

TxAcknowledgeEnable

Set this flag to 1 to get an acknowledge packet (fcTxAcknowledgePacket) once a frame is transmitted (includes null frames).

With the function fcbFRReconfigureMessageBuffer, this parameter can be changed while monitoring is active. This feature is only available on FlexCard based on E-Ray communication controller.

TransmissionMode

Type of transmission (refer to fcMsgBufTxMode).

With the function fcbFRReconfigureMessageBuffer, this parameter can be changed while monitoring is active.

TxAcknowledgeShowNullFrames

Set this flag to 1 to get TxAchnowledge packet for transmitted null frames. This flag is only evaluated if the TxAcknowledgeEnable flag is set.

TxAcknowledgeShowPayload

Set this flag to 1 to get the payload of the transmitted frame. The payload length of generated TxAchnowledge packet will otherwise be set to zero. This flag is only evaluated if the TxFrameEnable flag is set.

Reserved

Reserved for future use

See Also

fcMsgBufCfg

4.6.3.4 FCMSGBUFCFG

This structure describes the configuration of a message buffer.

```
Typedef struct fcMsgBufCfg
{
      fcMsgBufType Type;
      fcChannel ChannelFilter;
      fcDword CycleCounterFilter : 7;

      union
      {
            fcMsgBufCfgFifo Fifo;
            fcMsgBufCfgRx   Rx;
            fcMsgBufCfgTx   Tx;
      };

      fcDword Reserved[2];
} fcMsgBufCfg;
```

Members

Type

Defines the buffer type (FIFO, receive or transmit buffer)

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

ChannelFilter

Defines the channel(s) assigned to this buffer. With the function fcbFRReconfigureMessageBuffer, this parameter can only be changed while monitoring is active for receive and transmit buffer. For the configuration of a transmit or a receive message buffer assigned to a dynamic frame, only fcChannelA or fcChannelB is allowed.

CycleCounterFilter

Defines the filter used by the message buffer for cycle counter filtering. A zero value means that no cycle counter filtering is used. The cycle counter filter is composed of two parameters. The first one determines the cycle repetition and the second one the offset (the first cycle). The cycle repetition must be given in the form of 2^x where x is a number between 0 and 7. The offset must be less than the cycle repetition value. The two values are added. With the function fcbFRReconfigureMessageBuffer, this parameter can only be changed while monitoring is active for receive and transmit buffer.

Fifo

FIFO buffer configuration

Rx

Receive buffer configuration

Tx

Transmit buffer configuration

Reserved

Reserved for future use

See Also

fcbFRConfigureMessageBuffer, fcbFRReconfigureMessageBuffer, fcbFRGetMessageBuffer, fcMsgBufType, fcMsgBufCfgFifo, fcMsgBufCfgRx, fcMsgBufCfgTx

Example

```
// The following code configures a transmit buffer, which only transmits on cycles
6,14,22,30, …

fcMsgBufCfg cfg;
cfg.Type = fcMsgBufTx;
cfg.ChannelFilter = fcChannelA;


// Repetition: each 8 cycles
// Offset: 6 (First cycle will be cycle number 6)

cfg.CycleCounterFilter = 0x8 + 0x6;

cfg.Tx.FrameId = 61;
cfg.Tx.PayloadLength = 10;
cfg.Tx.PayloadLengthMax = 127;
cfg.Tx.PayloadPreambleIndicator = 0;
cfg.Tx.SyncFrameIndicator = 0;
cfg.Tx.StartupFrameIndicator = 0;
cfg.Tx.TxAcknowledgeEnable= 0;
cfg.Tx.TransmissionMode = fcMsgBufTxSingleShot;


unsigned int bufferIdx = 0;
fcError e = fcbFRConfigureMessageBuffer(hFlexCard,fcCC1,&bufferIdx,cfg);
```

4.6.3.5 FCCCTIMERCFG

This structure describes the configuration of a communication controller timer.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef struct fcCcTimerCfg
{
      fcDword ContinuousMode : 1;
      fcDword CycleCounterFilter : 7;
      fcDword MacrotickOffset : 14;
} fcCcTimerCfg;
```

Members

ContinuousMode

Defines the communication controller timer mode. Set to 1 for continuous mode or 0 for single-shot mode.

CycleCounterFilter

Defines the filter used by the cc timer for cycle counter filtering. A zero value means that no cycle counter filtering is used. The cycle counter filter is composed of two parameters. The first one determines the cycle repetition and the second one the offset (the first cycle). The cycle repetition must be given in the form of 2^x where x is a number between 0 and 7. The offset must be less than the cycle repetition value.

MacrotickOffset

Defines the macrotick offset from the beginning of the cycle when the cc timer interrupt has to occur. The cc timer interrupt occurs at this offset for each cycle of the cycle counter filter.

See Also

fcbFRSetCcTimerConfig, fcbFRGetCcTimerConfig, fcbFRCalculateMacrotickOffset

	Information
	This structure is initially supported by FlexCard API version S4V0-F.

4.6.4 FCBREINITIALIZECCMESSAGEBUFFER

This function re-initializes the message buffer configuration of the specified bus type and communication controller index. After calling this function the communication controller does not send old payload data. Re-initialization of message buffers is only allowed if the communication controller is in configuration state.

```
fcError fcbReinitializeCcMessageBuffer(
      fcHandle hFlexCard,
      fcBusType BusType,
      fcCC CC
)
```

Paramaters

hFlexCard

[IN] Handle to a FlexCard

BusType

[IN] The bus type.

CC

[IN] Index of the communication controller.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Information
This function is initially supported by FlexCard API version S4V0-F.

4.6.5 FCBGETNUMBERCCS

This function reads the number of the various communication controllers which are available on the FlexCard.

```
fcError fcbGetNumberCcs(
      fcHandle hFlexCard,
      fcNumberCC* pNumberCC
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

pNumberCC

[OUT] Pointer to the structure of the available communication controller numbers.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Example

```
fcNumberCC numberCC;
fcError e = fcbGetNumberCcs(hFlexCard, &numberCC);
if (0 == e)
{
      printf("Communication controllers: FlexRay: %d   CAN: %d   LIN: %d MOST: %d",
numberCC->FlexRay, numberCC->CAN, numberCC->LIN, numberCC->MOST);
}
```

Information
This function is initially supported by FlexCard API version S4V0-F.

4.6.6 FCBSETCONTINUEONPACKETOVERFLOW

This function configures the packet overflow handling of the FlexCard. The FlexCards default behaviour is to stop the monitoring if a buffer overflow was detected. This function can configure the FlexCard to continue with the monitoring when an amount of free RAM space is available again. An error packet fcErrFlexCardOverflow is generated in both cases.

```
fcError fcbSetContinueOnPacketOverflow(
      fcHandle hFlexCard,
      bool bContinue
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

bContinue

[IN] Set this flag to true to continue the monitoring in case of a packet buffer overflow being detected when RAM space is available again. Set to false to stop the monitoring.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Example

```
// Configure the FlexCard to continue on a message buffer overflow
fcError e = fcbSetContinueOnPacketOverflow (hFlexCard, true);
if (0 == e)
{
      printf("FlexCard will continue receiving after a message buffer overflow.");
}
```

Information
This function is initially supported by FlexCard API version S4V0-F.

4.6.7 FCBGETCURRENTTIMESTAMP

This function returns the current time stamp of the FlexCard device and the correlated performance counter value of the operating system.

```
fcError fcbGetCurrentTimeStamp(
      fcHandle hFlexCard,
      fcDword* pTimeStamp,
      fcQuad* pPerformanceCounter
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

pTimeStamp

[OUT] Current time stamp

pPerformanceCounter

[OUT] Correlated performance counter

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcQuad

Information
This function is initially supported by FlexCard API version S4V0-F.

4.6.8 FCBRESETTIMESTAMP

This function sets the FlexCard timestamp to 0.

```
fcError fcbResetTimestamp (
      fcHandle hFlexCard
)
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Parameters

hFlexCard

[IN] Handle to a FlexCard.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Example

```
fcError e = fcbResetTimestamp(hFlexCard);
if (0 == e)
{
      printf("Timestamp was reset.");
}
```

	Information
	This function is initially supported by FlexCard API version S4V0-F.

4.7 TRIGGER CONFIGURATION

The FlexCard is equipped with a trigger interface which provides two trigger lines. The FlexCard Cyclone II (SE) offers one IN and one OUT line. Via the IN trigger line it has the ability to receive trigger events and forward them to the user application. This feature allows e.g. a synchronization of different bus analyzers. The trigger lines of the FlexCard PMC (II) may be configured as IN or OUT. To configure and activate this feature, use the following structures and functions. The trigger event data is received as fcTriggerInfoPacket (Obsolete) or fcTriggerExInfoPacket with the fcbReceive function.

The following table lists the supported triggers during asynchronous and synchronous monitoring.

Trigger	Supported in Asynch-Mode	Supported in Synch-Mode
fcTriggerInOnSWPulse	OK	OK
fcTriggerInOnSWTimer	OK	OK
fcTriggerOutOnPulse	OK	OK
fcTriggerOutOnCycle	-	OK
fcTriggerOutOnSlotChX	-	OK
fcTriggerOutOnSlotInCycleChX	-	OK
fcTriggerOutOnCycleStart	-	OK
fcTriggerOutOnErrorDetected	-	OK
fcTriggerOutOnErrorX	-	OK
fcTriggerOutOnStartupCompleted	-	OK
fcTriggerOutOnStartDynamicSegment	-	OK
fcTriggerPMCIn	OK	OK
fcTriggerPMCOutOnPulse	OK	OK
fcTriggerPMCOutOnErrorDetected	-	OK
fcTriggerPMCOutOnStartupCompleted	-	OK
fcTriggerPMCOutOnCycleStart	-	OK

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

4.7.1 STRUCTURES

4.7.1.1 FCTRIGGERCONFIGURATIONEX

This structure configures the triggers of the FlexCard. Using the parameter *Condition* several triggers can be enabled. Therefore the enumeration fcTriggerConditionEx should be used. To set more than one trigger condition the conditions available in fcTriggerConditionEx must be binary OR-ed. Setting *Condition* to zero resets all triggers. In case you add additional trigger conditions, they have to be binary OR-ed with the former ones. Otherwise the previous settings will be reset. Some conditions need additional parameters:

The condition fcTriggerIn demands to set the parameter *onEdge*.
The condition fcTriggerInOnSWTimer demands to set the parameter *onTimePeriod*.
The condition fcTriggerOutOnCycle demands to set the parameter *onCycle*.
The condition fcTriggerOutOnSlotChA demands to set the parameter *onSlotChA*.
The condition fcTriggerOutOnSlotChB demands to set the parameter *onSlotChB*.
The condition fcTriggerOutOnSlotInCycleChA demands to set the parameters *onSlotChA* and *onCycle*.
The condition fcTriggerOutOnSlotInCycleChB demands to set the parameters *onSlotChB* and *onCycle*.
Please note that the configurations of FlexCard PMC (II) trigger lines are described in chapter 8 Additional PMC (II) card API.

```
Typedef struct fcTriggerConfigurationEx
{
       fcDword Condition;
       fcDword onEdge;
       fcDword onCycle;
       fcDword onSlotChA;
       fcDword onSlotChB;
       fcDword onTimePeriod;
       fcDword Reserved1[4];

       // for PMC (II) only:
       fcDword TriggerLineToConfigure;
       fcCC    TriggerGeneratingCC;
       fcDword Reserved2[4];
} fcTriggerConfigurationEx;
```

Members

Condition
This parameter can be set to zero to disable all trigger conditions. To configure specific conditions this parameter can be set to one or an OR-ed combination of given trigger conditions in the enumeration set fcTriggerConditionEx.

onEdge
This parameter has to be set when the condition fcTriggerInEnable is chosen.
Valid values are 0 = falling edge and 1 = rising edge.

onCycle
This parameter has to be set when at least on of the conditions fcTriggerOutOnCycle, fcTriggerOutOnSlotInCycleChA and fcTriggerOutOnSlotInCycleChB are chosen.
Valid values range from 0 to 63.

onSlotChA
This parameter has to be set when at least on of the conditions fcTriggerOutOnSlotChA or fcTriggerOutOnSlotInCycleChA are chosen.
Valid values range from 1 to 2047.

onSlotChB
This parameter has to be set when at least on of the conditions fcTriggerOutOnSlotChB or fcTriggerOutOnSlotInCycleChB are chosen.
Valid values range from 1 to 2047.

onTimePeriod
This parameter is only used in timer mode. Every TimePeriod milliseconds.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Reserved1[4]
Reserved Dwords for possible later use.
TriggerLineToConfigure
(PMC (II) only) This parameter sets the trigger line which should be configured.
Valid values range from 1 to 2.
TriggerGeneratingCC
(PMC (II) only) This parameter has to be set when a CC dependent trigger condition was set.
Valid values are fcCC1 to fcCC4.
Reserved2[4]
(PMC (II) only) Reserved Dwords for possible later use.

See Also
fcbSetTrigger, fcTriggerConditionEx, fcTriggerConditionPMC

4.7.2 ENUMERATIONS

4.7.2.1 FCTRIGGERCONDITIONEX

This enumeration defines the conditions available for a trigger configuration.

```
Typedef enum fcTriggerConditionEx
{
      fcTriggerIn                          = 0x00000002,
      fcTriggerOutOnPulse                  = 0x00000004,
      fcTriggerInOnSWPulse                 = 0x00000008,
      fcTriggerInOnSWTimer                 = 0x00000010,
      fcTriggerOutOnCycle                  = 0x00000040,
      fcTriggerOutOnSlotChA                = 0x00000080,
      fcTriggerOutOnSlotChB                = 0x00000100,
      fcTriggerOutOnSlotInCycleChA         = 0x00000200,
      fcTriggerOutOnSlotInCycleChB         = 0x00000400,
      fcTriggerOutOnCycleStart             = 0x00010000,
      fcTriggerOutOnErrorDetected          = 0x00020000,
      fcTriggerOutOnStartupCompleted       = 0x00040000,
      fcTriggerOutOnStartDynamicSegment    = 0x00080000,
      fcTriggerOutOnErrorSFBM              = 0x00100000,
      fcTriggerOutOnErrorSFO               = 0x00200000,
      fcTriggerOutOnErrorCCF               = 0x00400000,
      fcTriggerOutOnErrorSBVA              = 0x00800000,
      fcTriggerOutOnErrorPERR              = 0x01000000,
      fcTriggerOutOnErrorEDA               = 0x02000000,
      fcTriggerOutOnErrorLTVA              = 0x04000000,
      fcTriggerOutOnErrorTABA              = 0x08000000,
      fcTriggerOutOnErrorEDB               = 0x10000000,
      fcTriggerOutOnErrorLTVB              = 0x20000000,
      fcTriggerOutOnErrorTABB              = 0x40000000,
      fcTriggerOutOnErrorSBVB              = 0x80000000,
}fcTriggerConditionEx;
```

Members
fcTriggerIn
A trigger packet is generated as soon as the set edge (falling/rising) is detected on the input trigger line.
fcTriggerOutOnPulse
A signal is generated on the output trigger line as soon as the condition is set to the driver.
fcTriggerInOnSWPulse
A trigger packet is generated as soon as the condition is set to the driver.
fcTriggerInOnSWTimer
A trigger packet is generated by a set time interval.
fcTriggerOutOnCycle
A signal is generated on the output trigger line at each start of a set FlexRay cycle.
fcTriggerOutOnSlotChA
A signal is generated on the output trigger line at each start of a set slot on channel A.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

fcTriggerOutOnSlotChB
: A signal is generated on the output trigger line at each start of a set slot on channel B.

fcTriggerOutOnSlotInCycleChA
: A signal is generated on the output trigger line at each start of a set slot in a set cycle on channel A.

fcTriggerOutOnSlotInCycleChB
: A signal is generated on the output trigger line at each start of a set slot in a set cycle on channel B.

fcTriggerOutOnCycleStart
: A signal is generated on the output trigger line at a cycle start.

fcTriggerOutOnErrorDetected
: A signal is generated on the output trigger line at a detected error.

fcTriggerOutOnStartupCompleted
: A signal is generated on the output trigger line at a completed startup.

fcTriggerOutOnStartDynamicSegment
: A signal is generated on the output trigger line at the start of the dynamic segment.

fcTriggerOutOnErrorSFBM
: A signal is generated on the output trigger line at error SFBM (sync frame below minimum).

fcTriggerOutOnErrorSFO
: A signal is generated on the output trigger line at error SFO (sync frame overflow).

fcTriggerOutOnErrorCCF
: A signal is generated on the output trigger line at error CCF (clock correction failure).

fcTriggerOutOnErrorSBVA
: A signal is generated on the output trigger line at error SBVA (slot boundary violation channel A).

fcTriggerOutOnErrorPERR
: A signal is generated on the output trigger line at error PERR (parity error).

fcTriggerOutOnErrorEDA
: A signal is generated on the output trigger line at error EDA (error detected on channel A).

fcTriggerOutOnErrorLTVA
: A signal is generated on the output trigger line at error LTVA (latest transmit violation channel A).

fcTriggerOutOnErrorTABA
: A signal is generated on the output trigger line at error TABA (transmission across boundary channel A).

fcTriggerOutOnErrorEDB
: A signal is generated on the output trigger line at error EDB (error detected on channel B).

fcTriggerOutOnErrorLTVB
: A signal is generated on the output trigger line at error LTVB (latest transmit violation channel B).

fcTriggerOutOnErrorTABB
: A signal is generated on the output trigger line at error TABB (transmission across boundary channel B).

fcTriggerOutOnErrorSBVB
: A signal is generated on the output trigger line at error SBVB (slot boundary violation channel B).

See Also

fcbSetTrigger, fcTriggerConfigurationEx

Remarks

In the DebugAsynchron mode only the conditions *fcTriggerIn*, *fcTriggerOutOnPulse*, *fcTriggerInOnSWTimer* and *fcTriggerInOnSWPulse* can be used.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

4.7.3 FCBSETTRIGGER

This function configures and starts/stops triggers. For further information, refer to the structure fcTriggerConfigurationEx.

```
fcError fcbSetTrigger(
      fcHandle hFlexCard,
      fcTriggerConfigurationEx cfg
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

cfg

[IN] The trigger configuration

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcTriggerConfigurationEx, fcTriggerConditionEx

Example for cardbus FlexCards

```
// Generate a pulse at the beginning of any detected error and cycle 3
fcTriggerConfigurationEx triggerCfg;
memset(&triggerCfg, 0, sizeof(fcTriggerConfigurationEx));
triggerCfg.Condition = 0;
triggerCfg.Condition |= (fcDword)fcTriggerOutOnErrorDetected;
triggerCfg.Condition |= (fcDword)fcTriggerOutOnCycle;
triggerCfg.onCycle = 3;
// Generate a trigger packet all 1000 milliseconds
triggerCfg.Condition |= (fcDword)fcTriggerInOnSWTimer;
triggerCfg.onTimePeriod = 1000;

fcError e = fcbSetTrigger(hFlexCard,triggerCfg);
```

Example for PMC FlexCards

```
// Generate a pulse on trigger line 1 when the communication controller 2
// completed its startup
fcTriggerConfigurationEx triggerCfg;
memset(&triggerCfg, 0, sizeof(fcTriggerConfigurationEx));
triggerCfg.Condition = fcTriggerPMCOutOnStartupCompleted;
triggerCfg.TriggerLineToConfigure = 1;
triggerCfg.TriggerGeneratingCC = fcCC2;

fcError e = fcbSetTrigger(hFlexCard,triggerCfg);

// Generate a trigger packet when a pulse on trigger line 2 is detected
triggerCfg.Condition = fcTriggerPMCIn;
triggerCfg.TriggerLineToConfigure = 2;

fcError e = fcbSetTrigger(hFlexCard,triggerCfg);
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

4.8 EVENT

4.8.1 ENUMERATIONS

4.8.1.1 FCNOTIFICATIONTYPE

This enumeration defines different notification types. These types are used in the functions fcbSetEventHandleV2 and fcbSetEventHandleSemaphore to specify on which kind of event the application has to be notified.

```
Typedef enum fcNotificationType
{
      fcNotificationTypeCycleStarted   = 1,
      fcNotificationTypeFRCycleStarted = fcNotificationTypeCycleStarted,
      fcNotificationTypeTimer          = 2,
      fcNotificationTypeWakeup         = 3,
      fcNotificationTypeFRWakeup       = fcNotificationTypeWakeup,
      fcNotificationTypeCcTimer        = 12,
      fcNotificationTypeFRCcTimer      = fcNotificationTypeCcTimer,
} fcNotifyType;
```

Members

fcNotificationTypeCycleStarted
fcNotificationTypeFRCycleStarted
: Used to notify that a new cycle has started and that probably new data has been received.

fcNotificationTypeTimer
: Used to notify that the timer interval has elapsed. This notification requires the internal timer of the FlexCard to be enabled (See fcbSetTimer).

fcNotificationTypeWakeup
fcNotificationTypeFRWakeup
: Used to notify that one of the transceivers has received a wakeup event (only if one of the transceivers was in sleep mode).

fcNotificationTypeCcTimer
fcNotificationTypeFRCcTimer
: Used to notify that the configured cc timer macrotick offset has elapsed. This notification requires the E-Ray CC Timer0 to be enabled (See fcbFRSetCcTimerConfig).

See Also

fcbFRMonitoringStart, fcbSetEventHandleV2, fcbSetEventHandleSemaphore, fcbSetTimer, fcbFRSetCcTimerConfig

4.8.2 FCBSETEVENTHANDLEV2

This function registers an event handle for a specific notification type.

```
fcError fcbSetEventHandleV2(
      fcHandle hFlexCard,
      fcCC CC,
      fcHandle hEvent,
      fcNotificationType type
)
```

Parameters

hFlexCard
: [IN] Handle to a FlexCard

CC
: [IN] Communication controller index

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

hEvent

[IN] Event handle to be registered to signal when a new cycle starts or a timer interval has elapsed depending on the given *type*.

Type

[IN] The notification type for which the event has to be registered.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcNotificationType

Example

```
// Create the event objects
HANDLE hCycleStartEvent = ::CreateEvent(NULL,FALSE,FALSE,NULL);
fcCC eCC = fcCC1;

// Register our event handles
fcbSetEventHandleV2(hFlexCard, eCC, hCycleStartEvent,
   fcNotificationTypeFRCycleStarted);

// ...
// Use the event objects
// ...
```

	Information
	This function is initially supported by FlexCard API version S4V2-F.

	Information
	Please don't use this function with the FlexCard Linux driver, because it's not async-signal safe. To avoid deadlocks with the API use the function fcbSetEventHandleSemaphore instead.

4.8.3 FCBSETTIMER

This function enables or disables the internal FlexCard timer. To become notified when the timer interval has elapsed, an event of type *fcNotificationTypeTimer* has to be registered by the function fcbSetEventHandleV2 or fcbSetEventHandleSemaphore.

```
fcError fcbSetTimer(
      fcHandle hFlexCard,
      bool enable,
      fcDword timerInterval
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

enable

[IN] Set to true to enable the timer and to false to disable it.

timerInterval

[IN] Specifies the timer period in µs

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcNotificationType, fcbSetEventHandleV2, fcbSetEventHandleSemaphore

Example

```
// Create the event objects
HANDLE hCycleStartEvent = ::CreateEvent(NULL,FALSE,FALSE,NULL);
HANDLE hTimerEvent = ::CreateEvent(NULL,FALSE,FALSE,NULL);
fcCC eCC = fcCC1;

// Register our event handles
fcbSetEventHandleV2(hFlexCard, eCC, hCycleStartEvent,
   fcNotificationTypeCycleStarted);
fcbSetEventHandleV2(hFlexCard, eCC, hTimerEvent, fcNotificationTypeTimer);

// Enable the timer (1ms Interval)
fcbSetTimer(hFlexCard,true,1000);

// ...
// Use the event objects
// ...
```

4.8.4 FCBNOTIFICATIONPACKET

This function generates a notification packet each time the configured timer timeout has elapsed. This timer can be enabled / disabled by this function and the timeout can be set. The notification packets are inserted in the stream and received through the function fcbReceive.

```
fcError fcbNotificationPacket(
      fcHandle hFlexCard,
      bool enable,
      fcDword timerInterval
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

enable

[IN] Set to true to enable the timer and to false to disable it.

timerInterval

[IN] Specifies the time-out interval, in microseconds. A packet is generated as soon as the time-out has elapsed. The timer interval must be greater than 50us and smaller than 655350us. The value must be rounded to 10us units.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Information
This function is initially supported by FlexCard API version S2V0-F.

4.9 RECEIVE

4.9.1 TYPEDEFINITIONS

4.9.1.1 FCINFOPACKET

This structure describes an information packet. This packet type informs you about the start of a new cycle. All packets received between two consecutives info packets are part of the current cycle.

```
Typedef struct fcInfoPacket
{
    fcDword CurrentCycle;
    fcDword TimeStamp;
    fcDword RateCorrection : 12;
    fcDword OffsetCorrection : 19;
    fcDword ClockCorrectionFailedCounter : 4;
    fcDword PassiveToActiveCount : 5;
    fcCC    CC;
}fcInfoPacket;
```

Members

CurrentCycle
: The current cycle (FlexRay Protocol Specification: vRF!Header!CycleCount)

Timestamp
: The FlexCard time stamp (1 µs resolution). Indicates the time at which the packet was generated.

RateCorrection
: Rate correction value (two's complement). Indicates by how many microticks the node's cycle length should be changed.

OffsetCorrection
: Offset correction value (two' complement). Indicates the number of microticks that are added to the offset correction segment of the network idle time.

ClockCorrectionFailedCounter
: FlexRay Protocol Specification: vClockCorrectionFailed.

PassiveToActiveCount
: FlexRay Protocol Specification: vAllowPassiveToActive

CC
: The FlexCard CC which created this packet. This parameter will always be set to fcCC1 if only one FlexRay CC is available.

Remarks

A timestamp overflow occurs after approximately 4295 seconds.

See Also

fcPacket

4.9.1.2 FCFLEXRAYFRAME

This structure is equivalent to the FlexRay frame described in the FlexRay specification [3].

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef struct fcFlexRayFrame
{
    fcDword ID : 11;
    fcDword STARTUP : 1;
    fcDword SYNC : 1;
    fcDword NF : 1;
    fcDword PP : 1;
    fcDword R : 1;
    fcDword PayloadLength : 7;
    fcDword CycleCount : 6;
    fcDword HeaderCRC : 11;
    fcWord* pData;

    fcChannel Channel;
    fcDword ValidFrame : 1;
    fcDword SyntaxError : 1;
    fcDword ContentError : 1;
    fcDword SlotBoundaryViolation : 1;
    fcDword AsyncMode : 1;
    fcDword FrameCRC : 24;

    fcDword TimeStamp;
    fcCC    CC;
} fcFlexRayFrame;
```

Members

ID

The frame id defines the slot in which the frame was transmitted. (FlexRay Protocol Specification: vRF!Header!FrameID)

STARTUP

Indicates if the frame is a start up frame (=1) or not (=0) (FlexRay Protocol Specification: vRF!Header!SuFIndicator)

SYNC

Indicates if the frame is a sync frame (=1) or not (=0) (FlexRay Protocol Specification: vRF!Header!SyFIndicator)

NF

Set to 0, the null frame indicator indicates that *pData* contains no valid data. Set to 1, it indicates that *pData* contains valid data. (FlexRay Protocol Specification: vRF!Header!NFIndicator)

PP

The payload preamble indicator indicates whether or not an optional vector is contained within the payload segment of the frame transmitted. In the static segment, it indicates the presence of a network management vector at the beginning of the payload. In the dynamic segment it indicates the presence of a message id at the beginning of the payload, (FlexRay Protocol Specification: vRF!Header!PPIndicator).

R

Reserved bit (FlexRay Protocol Specification: vRF!Header!Reserved)

PayloadLength

Defines the number of 16 bit words contained in *pData* (FlexRay Protocol Specification: vRF!Header!Length)

CycleCount

The cycle in which the frame was received. (FlexRay Protocol Specification: vRF!Header!CycleCount)

HeaderCRC

The header CRC containing the cyclic redundancy check code is computed over the sync frame indicator, the start up frame indicator, the frame id and the payload length.(FlexRay Protocol Specification: vRF!Header!HeaderCRC)

pData

The pointer to the payload data. The payload is given in 16 bit words. (FlexRay Protocol Specification: vRF!Payload)

Channel

The channel (A or B) on which the frame was received.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

(FlexRay Protocol Specification: vRF!Channel)

ValidFrame

If a valid frame was received, this parameter is set to 1 (FlexRay Protocol Specification: vSS!ValidFrameA or vSS!ValidFrameB depends on Channel - Table 9-2: Slot status interpretation)

SyntaxError

If a syntax error was observed, this parameter is set to 1 (frame is syntactically incorrect). (FlexRay Protocol Specification: vSS!SyntaxErrorA or vSS!SyntaxErrorB depends on Channel)

ContentError

If a content error was observed, this parameter is set to 1 (frame is semantically incorrect). (FlexRay Protocol Specification: vSS!ContentErrorA or vSS!ContentErrorB depends on Channel)

SlotBoundaryViolation

If a slot boundary violation was observed, this parameter is set to 1 (FlexRay Protocol Specification: vSS!BviolationA or vSS!BviolationB depends on Channel)

AsyncMode

If the packet was generated by the asynchronous debug mode, this parameter is set to 1.

FrameCRC

If the packet was generated by the asynchronous debug mode, the FrameCRC contains the cyclic redundancy check code is computed over complete frame. In synchronous monitoring mode, this parameter is not set.

TimeStamp

The FlexCard time stamp (1 µs resolution). The timestamp marks the begin of the reception of the frame.

CC

The FlexCard CC which created this packet. This parameter will always be set to fcCC1 if only one FlexRay CC is available.

See Also

fcPacket

Information
The payload length is a multiple of 16 bit words. The payload data is also given in 16 bit words.

4.9.1.3 FCTXACKNOWLEDGEPACKET

This structure provides information about a transmit acknowledge packet. Transmit acknowledge packets are used to inform the user when a frame is transmitted.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef struct fcTxAcknowledgePacket
{
    fcDword BufferId;
    fcDword TimeStamp;
    fcDword CycleCount;

    fcDword ID : 11;
    fcDword STARTUP : 1;
    fcDword SYNC : 1;
    fcDword NF : 1;
    fcDword PP : 1;
    fcDword R : 1;
    fcDword PayloadLength : 7;
    fcDword ValidFrame : 1;
    fcDword SyntaxError : 1;
    fcDword ContentError : 1;
    fcDword HeaderCRC : 11;
    fcWord* pData;
    fcChannel Channel;
    fcCC CC;
} fcTxAcknowledgePacket;
```

Members

BufferId

The buffer id used to transmit the frame (equivalent to the buffer id returned by the function fcbFRConfigureMessageBuffer).

TimeStamp

The FlexCard time stamp (1 µs resolution). The timestamp marks the beginning of the transmission of the frame.

CycleCount

Indicates the cycle in which the frame was transmitted. (FlexRay Protocol Specification: vTF!Header!CycleCount)

ID

The frame id defines the slot in which the frame was transmitted.

STARTUP

Indicates if the frame was a start up frame (=1) or not (=0)

SYNC

Indicates if the frame was a sync frame (=1) or not (=0)

NF

Set to 0, the null frame indicator indicates that *pData* contains no valid data. Set to 1, it indicates that *pData* contains valid data.

PP

The payload preamble indicator indicates whether or not an optional vector is contained within the payload segment of the frame transmitted. In the static segment, it indicates the presence of a network management vector at the beginning of the payload. In the dynamic segment it indicates the presence of a message id at the beginning of the payload.

R

Reserved bit

PayloadLength

Defines the number of 16 bit words contain in *pData*

ValidFrame

If a valid frame was received, this parameter is set to 1 (FlexRay Protocol Specification: vSS!ValidFrameA or vSS!ValidFrameB depends on Channel - Table 9-2: Slot status interpretation)

SyntaxError

If a syntax error was observed, this parameter is set to 1 (frame is syntactically incorrect). (FlexRay Protocol Specification: vSS!SyntaxErrorA or vSS!SyntaxErrorB depends on Channel)

ContentError

If a content error was observed, this parameter is set to 1 (frame is semantically incorrect). (FlexRay Protocol Specification: vSS!ContentErrorA or vSS!ContentErrorB depends on Channel)

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

HeaderCRC

The header CRC contains the cyclic redundancy check code is computed over the sync frame indicator, the start up frame indicator, the frame id and the payload length.

pData

The pointer to the payload data. The payload is given in 16 bit words.

Channel

The channel (A or B) on which the frame was transmitted.
(FlexRay Protocol Specification: vRF!Channel)

CC

The FlexCard CC which created this packet. This parameter will always be set to fcCC1 if only one FlexRay CC is available. This parameter will always be set to fcCC2 for used SelfSync feature packets.

See Also

fcPacket

4.9.1.4 FCERRPOCERRORMODECHANGEDINFO

This structure provides additional information about the *fcErrPOCErrorModeChanged* error.

```
Typedef struct fcErrPOCErrorModeChangedInfo
{
    fcState State;
}fcErrPOCErrorModeChangedInfo;
```

Members

State

Contains the new POC error mode (HALT, NORMAL_ACTIVE or NORMAL_PASSIVE)

See Also

fcErrorPacket

4.9.1.5 FCERRSYNCFRAMESINFO

This structure provides additional information about the *fcErrSyncFramesBelowMinimum* and *fcErrSyncFrameOverflow* errors.

```
Typedef struct fcErrSyncFramesInfo
{
    fcDword SyncFramesEvenA : 4;
    fcDword SyncFramesEvenB : 4;
    fcDword SyncFramesOddA : 4;
    fcDword SyncFramesOddB : 4;
}fcErrPOCErrorModeChangedInfo;
```

Members

SyncFramesEvenA

Valid sync frame received and transmitted on channel A in even communication cycles

SyncFramesEvenB

Valid sync frame received and transmitted on channel B in even communication cycles

SyncFramesOddA

Valid sync frame received and transmitted on channel A in odd communication cycles

SyncFramesOddB

Valid sync frame received and transmitted on channel B in odd communication cycles

See Also

fcErrorPacket

4.9.1.6 FCERRCLOCKCORRECTIONFAILUREINFO

This structure provides additional information about the *fcErrClockCorrectionFailure* error.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef struct fcErrClockCorrectionFailureInfo
{
    fcDword MissingRateCorrection : 1;
    fcDword RateCorrectionLimitReached : 1;

    fcDword OffsetCorrectionLimitReached : 1;
    fcDword MissingOffsetCorrection : 1;

    fcDword SyncFramesEvenA : 4;
    fcDword SyncFramesEvenB : 4;
    fcDword SyncFramesOddA : 4;
    fcDword SyncFramesOddB : 4;
}fcErrClockCorrectionFailureInfo;
```

Members

MissingRateCorrection
: Is set to 1 if no rate correction can be performed because no pairs of even/odd sync frames were received.

RateCorrectionLimitReached
: Is set to 1 if the maximum rate correction limit is reached.

OffsetCorrectionLimitReached
: Is set to 1 if the maximum offset correction limit is reached.

MissingOffsetCorrection
: Is set to 1 if no offset correction can be performed because no sync frames were received.

SyncFramesEvenA
: Valid sync frame received and transmitted on channel A in even communication cycles

SyncFramesEvenB
: Valid sync frame received and transmitted on channel B in even communication cycles

SyncFramesOddA
: Valid sync frame received and transmitted on channel A in odd communication cycles

SyncFramesOddB
: Valid sync frame received and transmitted on channel B in odd communication cycles

See Also

fcErrorPacket

4.9.1.7 FCERRSLOTINFO

This structure provides additional information about the *fcErrSyntax*, *fcErrContent*, *fcErrSlotBoundaryViolation*, *fcErrTransmissionAcrossBoundary*, *fcErrLatestTransmitViolation* fcErrSyntaxSW, *fcErrSlotBoundaryViolationSW*, *fcErrTransmissionConflictSW*, *fcErrSyntaxNIT* and *fcErrSlotBoundaryViolationNIT* errors.

```
Typedef struct fcErrSlotInfo
{
    fcChannel Channel;
    fcDword SlotCount;
}fcErrSlotInfo;
```

Members

Channel
: The channel on which the error was observed.

SlotCount
: The approximate slot count when the error occurred.

See Also

fcErrorPacket

4.9.1.8 FCERRORPACKET

This structure provides information about an error packet.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef struct fcErrorPacket
{
    fcErrorPacketFlag Flag;
    fcDword TimeStamp;
    fcDword CycleCount;

    union AdditionalInfo
    {
        fcErrPOCErrorModeChangedInfo        ErrPOCErrorModeChangedInfo;
        fcErrSyncFramesInfo                 ErrSyncFramesInfo;
        fcErrSlotInfo                       ErrSlotInfo;
        fcErrClockCorrectionFailureInfo     ErrClockCorrectionFailureInfo;
    }AdditionalInfo;
    fcCC CC;

    fcDword Reserved;
}fcErrorPacket;
```

Members

Flag

Error type

TimeStamp

The FlexCard time stamp (1 µs resolution). Indicates the time at which the packet was generated.

CycleCount

The cycle in which the error occurred.

AdditionalInfo

- *ErrPOCErrorModeChangedInfo*
  Additional information about the fcErrPOCErrorModeChanged error.
- *ErrSyncFramesInfo*
  Additional information about the fcErrSyncFramesBelowMinimum , fcErrSyncFrameOverflow errors
- *ErrSlotInfo*
  Additional information about the fcErrSyntax, fcErrContent, fcErrSlotBoundaryViolation, fcErrTransmissionAcrossBoundary and fcErrLatestTransmitViolation errors
- *ErrClockCorrectionFailureInfo*
  Additional information about the fcErrClockCorrectionFailure error.

CC

The FlexCard CC which created this packet. This parameter will always be set to fcCC1 if only one FlexRay CC is available.

Reserved

Reserved for future use.

See Also

fcPacket

4.9.1.9 FCSTATUSWAKEUPINFO

This structure provides additional information about the *cStatusWakeupStatus* status.

```
Typedef struct fcStatusWakeupInfo
{
    fcWakeupStatus WakeupStatus;
} fcStatusWakeupInfo;
```

Members

WakeupStatus

Current wakeup state.

See Also

fcStatusPacket

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

4.9.1.10 FCSTATUSPACKET

This structure provides information about a status packet.

```
Typedef struct fcStatusPacket
{
    fcStatusPacketFlag Flag;
    fcDword TimeStamp;
    fcDword CycleCount;

    union AdditionalInfo
    {
         fcStatusWakeupInfo StatusWakeupInfo;
    }AdditionalInfo;
    fcCC    CC;
    fcDword Reserved[2];
}fcStatusPacket;
```

Members

Flag

Status type

TimeStamp

The FlexCard time stamp (1 µs resolution). Indicates the time at which the packet was generated.

CycleCount

The cycle in which the status has changed.

AdditionalInfo

StatusWakeupInfo

Additional information about fcStatusWakeupStatus status

CC

The FlexCard CC which created this packet. This parameter will always be set to fcCC1 if only one FlexRay CC is available. This parameter will always be set to fcCC2 for used SelfSync feature packets.

Reserved

Reserved for future use.

See Also

fcPacket, fcStatusPacketFlag, fcStatusWakeupInfo

4.9.1.11 FCNMVECTORPACKET

This structure provides information about a network management vector. (FlexRay Protocol Specification V2.0: Section 4.3.1 NMVector)

```
typedef struct fcNMVectorPacket
{
    fcDword TimeStamp;
    fcDword CycleCount;
    fcDword NMVectorLength;
    fcByte NMVector[12];
    fcCC CC;
    fcDword Reserved;
} fcNMVectorPacket;
```

Members

TimeStamp

The FlexCard time stamp (1 µs resolution). Indicates the time at which the packet was generated.

CycleCount

The cycle in which the network management vector was updated.

NMVectorLength

Length of network management vector in number of bytes. (FlexRay Protocol Specification: gNetworkManagementVectorLength)

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

NMVector

The data bytes of the network management vector.

CC

The FlexCard CC which created this packet. This parameter will always be set to fcCC1 if only one FlexRay CC is available.

Reserved

Reserved for future use.

See Also

fcPacket, fcCC

4.9.1.12 FCNOTIFICATIONPACKET

This structure provides information about a notification packet. A notification packet is generated each time the configured time out elapses. The generation of this packet can be controlled with the function fcbNotificationPacket.

```
Typedef struct fcNotificationPacket
{
    fcDword TimeStamp;
    fcDword SequenceCounter;
    fcDword Reserved;
} fcNotificationPacket;
```

Members

TimeStamp

The FlexCard time stamp (1 µs resolution). Indicates the time at which the packet was generated.

SequenceCounter

This parameter is incremented each time a notification packet is generated.

Reserved

See Also

fcPacket, fcbNotificationPacket

	Information
	This packet type is initially supported by FlexCard API version S2V0-F.

4.9.1.13 FCTRIGGEREXINFOPACKET

This structure provides information about a trigger packet.

```
Typedef struct fcTriggerExInfoPacket
{
      fcDword Condition;
      fcDword TimeStamp;
      fcDword SequenceCount;
      fcDword Reserved1;
      fcQuad PerformanceCounter;
      fcDword Edge;
      fcDword TriggerLine;
      fcDword reserved[4];
} fcTriggerInfoPacket;
```

Members

Condition

The fulfilled condition which has caused the trigger packet generation.

TimeStamp

The FlexCard time stamp (1 µs resolution). Indicates the time at which the packet was generated.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Reserved1
Reserved for future use.
SequenceCount
Sequence count for each signal.
PerformanceCounter
Variable that receives the current performance-counter value. This value is only valid for the trigger condition *fcTriggerInOnSWTimer.*
Edge
The edge on which the trigger was signalled.
TriggerLine
The trigger line which detected a trigger signal. This value is only valid for triggers of FlexCard PMC.
Reserved[4]
Reserved for future use.

See Also
fcPacket

	Information
	This packet type is initially supported by FlexCard API version S2V2-F.

4.9.1.14 FCCANPACKET

This structure provides information about a CAN packet.

```
Typedef struct fcCANPacket
{
      fcDword ID           : 29;
      fcDword ExtendedId   : 1;
      fcDword TimeStamp;
      fcDword BufferNumber : 8;
      fcDword DLC          : 4;
      fcDword Direction    : 1;
      fcDword RemoteFrame  : 1;
      fcDword MessageLost  : 1;
      fcDword Reserved;
      fcCC CC;
      fcByte Data[8];
}fcCANPacket;
```

Members
ID
The CAN message identifier which was received or transmitted.
ExtendedId
If this flag is 1 the CAN message is an extended frame. If set to 0 it is a standard frame.
TimeStamp
The FlexCard time stamp (1 µs resolution). Indicates the time at which the packet was generated.
BufferNumber
Indicates the corresponding buffer number for the CAN packet.
DLC
Indicates the data length (in bytes).
Direction
This flag depends on the parameter *RemoteFrame*. If *Direction* is 0 and *RemoteFrame* is 0, the CAN packet is a received data frame. If *Direction* is 1 and *RemoteFrame* is 0 the CAN packet is a transmit acknowledge frame generated by the FlexCard. If *RemoteFrame* is 1, see *RemoteFrame* for further description.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

RemoteFrame

This flag depends on the parameter *Direction*. If *RemoteFrame* is 1 and *Direction* is 0, the CAN packet is a remote rx frame. If *RemoteFrame* is 1 and *Direction* is 1, the CAN packet is a remote tx frame. If *Direction* is 0, see *Direction* for further description.

MessageLost

If this flag is 1 the CAN communication controller has lost a message. If 0 no message has been lost. This flag is only valid with *Direction* = 0.

Reserved

Reserved for future use.

CC

The CAN communication controller on which the frame was received or transmitted.

Data

The received or transmitted data. All of the 8 data bytes can be read. The corresponding parameter DLC indicates the length of the valid values.

See Also

fcPacket

	Information
	This packet type is initially supported by FlexCard API version S4V0-F.

4.9.1.15 FCCANERRORPACKET

This structure provides information about a CAN error packet.

```
Typedef struct fcCANErrorPacket
{
      fcCANErrorType Type;
      fcCANCcState State;
      fcDword TimeStamp;
      fcDword ReceiveErrorCounter;
      fcDword TransmitErrorCounter;
      fcCC CC;
      fcDword Reserved[2];
}fcCANErrorPacket;
```

Members

Type

Error type

State

Communication controller state

TimeStamp

The FlexCard time stamp (1 µs resolution). Indicates the time at which the packet was generated.

ReceiveErrorCounter

Actual state of the Receive Error Counter. Valid values range from 0 to 127.

TransmitErrorCounter

Actual state of the Transmit Error Counter. Values values range 0 to 255.

CC

The CC on which the packet was created.

Reserved[4]

Reserved for future use.

See Also

fcPacket, fcCANErrorType, fcCANCcState

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Information
This packet type is initially supported by FlexCard API version S4V0-F.

4.9.1.16 FCPACKET

This structure provides information about a packet.

```
Typedef struct fcPacket
{
    fcPacketType Type;
    union
    {
        fcFlexRayFrame*         FlexRayFrame;
        fcInfoPacket*           InfoPacket;
        fcErrorPacket*          ErrorPacket;
        fcStatusPacket*         StatusPacket;
        fcTriggerInfoPacket*    TriggerPacket;
        fcTxAcknowledgePacket*  TxAcknowledgePacket;
        fcNMVectorPacket*       NMVectorPacket;
        fcNotificationPacket*   NotificationPacket;
        fcTriggerExInfoPacket*  TriggerExPacket;
        fcCANPacket*            CANPacket;
        fcCANErrorPacket*       CANErrorPacket;
    };
    fcPacket*  pNextPacket;
}fcPacket;
```

Members

Type
: Type of packet.

FlexRayFrame
: Pointer to the packet data. The content depends on the type of packet.

InfoPacket
: Pointer to the packet data. The content depends on the type of packet.

ErrorPacket
: Pointer to the packet data. The content depends on the type of packet.

StatusPacket
: Pointer to the packet data. The content depends on the type of packet.

TriggerPacket
: Pointer to the packet data. The content depends on the type of packet.

TxAcknowledgePacket
: Pointer to the packet data. The content depends on the type of packet.

NMVectorPacket
: Pointer to the packet data. The content depends on the type of packet.

NotificationPacket
: Pointer to the packet data. The content depends on the type of packet.

TriggerExPacket
: Pointer to the packet data. The content depends on the type of packet.

CANPacket
: Pointer to the packet data. The content depends on the type of packet.

CANErrorPacket
: Pointer to the packet data. The content depends on the type of packet.

pNextPacket
: Pointer to the next packet. If the pointer is NULL, there are no more packets available.

See Also

fcInfoPacket, fcFlexRayFrame, fcTxAcknowledgePacket, fcErrorPacket, fcStatusPacket, fcTriggerInfoPacket, fcNMVectorPacket, fcNotificationPacket, fcTriggerExInfoPacket, fcCANPacket, fcCANErrorPacket

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

4.9.2 ENUMERATIONS

4.9.2.1 FCPACKETTYPE

This enumeration contains the different packet types.

```
Typedef enum fcPacketType
{
      fcPacketTypeInfo           = 1,
      fcPacketTypeFlexRayFrame   = 2,
      fcPacketTypeError          = 3,
      fcPacketTypeStatus         = 4,
      fcPacketTypeTrigger        = 5,
      fcPacketTypeTxAcknowledge  = 6,
      fcPacketTypeNMVector       = 7,
      fcPacketTypeNotification   = 8,
      fcPacketTypeTriggerEx      = 9,
      fcPacketTypeCAN            = 10,
      fcPacketTypeCANError       = 11,
}fcPacketType;
```

Members

fcPacketTypeInfo
: Frame is an info packet

fcPacketTypeFlexRayFrame
: Frame is a FlexRay frame

fcPacketTypeError
: Frame is an error packet

fcPacketTypeStatus
: Frame is a status packet

fcPacketTypeTrigger
: Frame is a trigger packet (obsolete)

fcPacketTypeTxAcknowledge
: Frame is a transmit acknowledge packet

fcPacketTypeNMVector
: Frame is a network management vector packet

fcPacketTypeNotification
: Frame is a notification packet

fcPacketTypeTriggerEx
: Frame is a trigger packet

fcPacketTypeCAN
: Frame is a CAN packet

fcPacketTypeCANError
: Frame is a CAN error packet

See Also

fcPacket, fcInfoPacket, fcFlexRayFrame, fcTxAcknowledgePacket, fcErrorPacket, fcStatusPacket, fcTriggerInfoPacket, fcNMVectorPacket, fcNotificationPacket, fcTriggerExInfoPacket, fcCANPacket, fcCANErrorPacket

4.9.2.2 FCERRORPACKETFLAG

This enumeration contains the different error types reported by an error packet.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef enum fcErrorPacketFlag
{
    fcErrNone = 0,
    fcErrFlexcardOverflow,
    fcErrPOCErrorModeChanged,
    fcErrSyncFramesBelowMinimum,
    fcErrSyncFrameOverflow,
    fcErrClockCorrectionFailure,
    fcErrParityError,
    fcErrReceiveFIFOOverrun,
    fcErrEmptyFIFOAccess,
    fcErrIllegalInputBufferAccess,
    fcErrIllegalOutputbufferAccess,
    fcErrSyntax,
    fcErrContent,
    fcErrSlotBoundaryViolation,
    fcErrTransmissionAcrossBoundary,
    fcErrLatestTransmitViolation,
    fcErrSyntaxSW,
    fcErrSlotBoundaryViolationSW,
    fcErrTransmissionConflictSW,
    fcErrSyntaxNIT,
    fcErrSlotBoundaryViolationNIT,
} fcErrorPacketFlag;
```

Members

fcErrNone
: No error occurred

fcErrFlexcardOverflow
: FlexCard buffer overflow. This error occurs if the application was too slow to receive and process the packets. If the FlexCard is configured to stop the monitoring it is necessary to stop and start the monitoring again. Else the FlexCard continue the monitoring when an amount of free RAM space is available again. In such a case the FlexCard loses packets.

fcErrPOCErrorModeChanged
: Protocol Operation Control error. Additional information are described in the structure fcErrPOCErrorModeChangedInfo

fcErrSyncFramesBelowMinimum
: Additional information are described in the structure fcErrSyncFramesInfo

fcErrSyncFrameOverflow
: Additional information described in the structure fcErrSyncFramesInfo

fcErrClockCorrectionFailure
: Additional information are described in the structure fcErrClockCorrectionFailureInfo

fcErrParityError
: Internal E-Ray error. No additional information is available

fcErrReceiveFIFOOverrun
: No additional information exists for the internal FlexCard error (fcErrorPacket.AdditionalInfo is not valid)

fcErrEmptyFIFOAccess
: No additional information exists for the internal FlexCard error

fcErrIllegalInputBufferAccess
: No additional information exists for the internal FlexCard error

fcErrIllegalOutputbufferAccess
: No additional information exists for the internal FlexCard error

fcErrSyntax
: A syntax error was observed (frame is syntactically incorrect). (FlexRay Protocol Specification: vSS!SyntaxErrorA or vSS!SyntaxErrorB depends on Channel) Additional information are described in the structure fcErrSlotInfo

fcErrContent
: A content error was observed (frame is semantically incorrect). (FlexRay Protocol Specification: vSS!ContentErrorA or vSS!ContentErrorB depends on Channel) Additional information described in the structure fcErrSlotInfo

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

fcErrSlotBoundaryViolation
A slot boundary violation was observed. (FlexRay Protocol Specification: vSS!BviolationA or vSS!BviolationB depends on Channel) Additional information described in the structure fcErrSlotInfo

fcErrTransmissionAcrossBoundary
Additional information are described in the structure fcErrSlotInfo

fcErrLatestTransmitViolation
Additional information are described in the structure fcErrSlotInfo

fcErrSyntaxSW
Syntax error in symbol window was observed. Additional information are described in the structure fcErrSlotInfo.

fcErrSlotBoundaryViolationSW
Slot boundary violation in symbol window was observed. Additional information are described in the structure fcErrSlotInfo.

fcErrTransmissionConflictSW
Transmission conflict in symbol window was observed. Additional information are described in the structure fcErrSlotInfo.

fcErrSyntaxNIT
Syntax error in network idle time was observed. Additional information are described in the structure fcErrSlotInfo.

fcErrSlotBoundaryViolationNIT
Slot boundary violation in network idle time was observed. Additional information are described in the structure fcErrSlotInfo.

See Also
fcErrorPacket, fcErrPOCErrorModeChangedInfo, fcErrSyncFramesInfo, fcErrClockCorrectionFailureInfo, fcErrSlotInfo

4.9.2.3 FCSTATUSPACKETFLAG

Possible hardware status flags are reported by a status packet.

```
Typedef enum fcStatusPacketFlag
{
    fcStatusNone = 0,
    fcStatusWakeupStatus,
    fcStatusCollisionAvoidanceSymbol,
    fcStatusStartupCompletedSuccessfully,
    fcStatusWakeupPatternChannelA,
    fcStatusWakeupPatternChannelB,
    fcStatusMTSReceivedonChannelA,
    fcStatusMTSReceivedonChannelB,
} fcStatusPacketFlags;
```

Members

fcStatusNone
No status change.

fcStatusWakeupStatus
Wakeup status has changed

fcStatusCollisionAvoidanceSymbol
Collision avoidance symbol was received

fcStatusStartupCompletedSuccessfully
Start up has been successfully completed

fcStatusWakeupPatternChannelA
Wakeup pattern received on Channel A

fcStatusWakeupPatternChannelB
Wakeup pattern received on Channel B

fcStatusMTSReceivedonChannelA
Media Access Test Symbol received on Channel A

fcStatusMTSReceivedonChannelB
Media Access Test Symbol received on Channel B

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

See Also

fcPacket, fcStatusPacket, fcStatusWakeupInfo

4.9.2.4 FCCANERRORTYPE

This enumeration contains the different error types reported by a CAN error packet.

```
Typedef enum fcCANErrorType
{
      fcCANErrorNone = 0,
      fcCANErrorStuff,
      fcCANErrorForm,
      fcCANErrorAcknowledge,
      fcCANErrorBit1,
      fcCANErrorBit0,
      fcCANErrorCRC,
      fcCANErrorParity,
}fcCANErrorType;
```

Members

fcCANErrorNone
: No error occurred.

fcCANErrorStuff
: More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.

fcCANErrorForm
: A fixed format part of a received frame has the wrong format.

fcCANErrorAcknowledge
: The message the CAN communication controller transmitted was not acknowledged by another node.

fcCANErrorBit1
: During the transmission of a message (with the exception of the arbitration field), the device wanted to send a recessive level (bit of logical value 1), but the monitored bus value was dominant (bit of logical value 0).

fcCANErrorBit0
: During the transmission of a message, the device wanted to send a dominant level (data or identifier bit logical value 0), but the monitored bus value was recessive (data or identifier bit logical value 1).

fcCANErrorCRC
: The CRC check sum was incorrect in the message received, the CRC received for an incoming message does not match with the calculated CRC for the received data.

fcCANErrorParity
: The parity check mechanism has detected an parity error in the message RAM of the communication controller.

See Also

fcCANErrorPacket

	Information
	This enumeration is initially supported by FlexCard API version S4V0-F.

4.9.3 FCBRECEIVE

This function reads all available packets from the FlexCard memory into a memory block allocated by the fcBase API. The frames are stored into a linked list. To free the memory allocated by this function, use the function fcFreeMemory with the type *fcMemoryTypePacket*.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
fcError fcbReceive(
      fcHandle hFlexCard,
      fcPacket** pPacket
);
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

pPacket

[OUT] Address of the fcPacket object pointer. The memory for this structure and its content is allocated by the fcBase API. Packets are available if the return code is 0 and *pPacket* is not a null pointer.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

	Information
	This function allocates memory. To prevent memory leaks the memory has to be released after having processed the packets.

Example

```
fcPacket* pPackets = NULL;
fcError e = fcbReceive(m_hFlexCard, &pPackets);
if (0 == e)
{
      fcPacket* pCurrentPacket = pPackets;
      while (NULL != pCurrentPacket)
      {
            switch (pCurrentPacket->Type)
            {
            case fcPacketTypeInfo:
                  printf("[fcPacketTypeInfo] TimeStamp: %f Cycle: %d\n",
                        (float)pCurrentPacket->InfoPacket->TimeStamp* 0.000001,
                        pCurrentPacket->InfoPacket->CurrentCycle);

            break;

            case fcPacketTypeFlexRayFrame:
            {
                  fcFlexRayFrame* pFrame = pCurrentPacket->FlexRayFrame;
                  printf("[fcPacketTypeFlexRayFrame] Cycle: %d Id: %d Channel:"
                        "%d PayloadLength: %d",  pFrame->CycleCount,
                        pFrame->ID,
                        pFrame->Channel,
                        pFrame->PayloadLength);

                  for (int i = 0; i < pFrame->PayloadLength; i++)
                  {
                        printf("%04X ", pFrame->pData[i]);
                  }

                  if (pFrame->PP) printf(" PP");
                  if (pFrame->NF) printf(" NF");
                  if (pFrame->SYNC) printf(" SYNC");
                  if (pFrame->STARTUP) printf(" STARTUP");
                  if (pFrame->SyntaxError) printf(" SyntaxError");
                  if (pFrame->ContentError) printf(" ContentError");
                  if (pFrame->ValidFrame) printf(" ValidFrame");
                  if (pFrame->SlotBoundaryViolation)
                        printf(" SlotBoundaryViolation");
                  if (pFrame->AsyncMode)
                        printf(" AsyncMode FrameCRC: 0x%06X", pFrame->FrameCRC);
                  printf("\n");
                  break;
            }
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
            case fcPacketTypeError:
                  printf("[fcPacketTypeError]\n");
                  break;

            case fcPacketTypeStatus:
                  printf("[fcPacketTypeStatus]\n");
                  break;

            case fcPacketTypeTrigger:
                  printf("[fcPacketTypeTrigger]\n");
                  break;

            case fcPacketTypeTxAcknowledge:
                  printf("[fcPacketTypeTxAcknowledge]\n");
                  break;

            case fcPacketTypeNMVector:
                  printf("[fcPacketTypeNMVector]\n");
                  break;
            }

            pCurrentPacket = pCurrentPacket->pNextPacket;
      }

      fcFreeMemory(fcMemoryTypePacket, pPacket);
}
```

4.10 OBSOLETE

4.10.1 FCINFO (OBSOLETE)

	Information
	This structure is obsolete. Please use fcInfoHwSw instead.

This structure provides information about the components and the identifier of a FlexCard. If more than one FlexCard is detected on the system, the fcbGetEnumFlexCards (Obsolete) function returns a linked list of this structure. If a connection to a FlexCard is already opened, this FlexCard does not appear in this list.

```
Typedef struct fcInfo
{
      fcDword FlexCardId;
      fcVersion Version;
      fcInfo* pNext;
} fcInfo;
```

Members

FlexCardId
: Unique number used to identify a FlexCard. This id is required to open a connection to the FlexCard.

Version
: Version information about hardware and software components of the FlexCard.

pNext
: Pointer to the next available FlexCard. If no other FlexCard exists, pNext is a null pointer.

See Also

fcVersion (OBSOLETE), fcbGetEnumFlexCards (Obsolete)

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

4.10.2 FCINFOV2 (OBSOLETE)

Information
This structure is obsolete. Please use fcInfoHwSw instead.

This structure provides information about the components, the identifier and the current device state of a FlexCard. If more than one FlexCard is detected on the system, the fcbGetEnumFlexCardsV2 (Obsolete) function returns a linked list of this structure.

```
Typedef struct fcInfoV2
{
      fcDword FlexCardId;
      fcVersion Version;
      fcDword Busy;
      fcDword Reserved;
      fcInfoV2* pNext;
} fcInfoV2;
```

Members

FlexCardId
: Unique number used to identify a FlexCard. This id is required to open a connection to the FlexCard.

Version
: Version information about hardware and software components of the FlexCard.

Busy
: The current device state. A value <> 0 indicates a connection to this FlexCard is already opened.

Reserved
: Reserved for future use.

pNext
: Pointer to the next available FlexCard. If no other FlexCard exists, pNext is a null pointer.

See Also

fcVersion (OBSOLETE), fcbGetEnumFlexCardsV2 (Obsolete)

4.10.3 FCVERSION (OBSOLETE)

Information
This structure is obsolete. Please use fcInfoHw and fcInfoSw instead.

This structure provides version information about the FlexCard hardware and software components.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef struct fcVersion
{
      fcVersionNumber BaseDll;
      fcVersionNumber DeviceDriver;
      fcVersionNumber Firmware;
      fcVersionNumber Hardware;
      fcCCType CCType;
      fcVersionNumber CC;
      fcVersionNumber BusGuardian;
      fcVersionNumber Protocol;
      fcDword Serial;
      fcFlexCardDeviceId DeviceId;
      fcVersionCC* pVersionCC;
      fcDword Reserved[2];
}fcVersion;
```

Members

BaseDll
: DLL Base Version

DeviceDriver
: Device driver version

Firmware
: Firmware (gateway software) version

Hardware
: FlexCard hardware version

CCType
: Communication controller type

CC
: Communication controller module version

BusGuardian
: Bus Guardian version

Protocol
: FlexRay Protocol version

Serial
: FlexCard serial number. A zero value indicates a non-valid FlexCard serial number.

DeviceId
: Device identifier to detect the FlexCard type (FlexCard Cyclone II, FlexCard Cyclone II SE or FlexCard PMC).

pVersionCC
: Pointer to version information about the available communication controllers.

Reserved
: Reserved for internal purpose

See Also

fcFlexCardDeviceId, fcInfo (Obsolete), fcInfoV2 (Obsolete), fcbGetEnumFlexCards (Obsolete), fcbGetEnumFlexCardsV2 (Obsolete)

4.10.4 FCBGETENUMFLEXCARDS (OBSOLETE)

	Information
	This function is obsolete. Please use fcbGetEnumFlexCardsV3 instead.

This function returns a linked list of the unused FlexCards found on the system. To free the memory, which was allocated by the function, please use the function fcFreeMemory with type *fcMemoryTypeInfo.*

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
fcError fcbGetEnumFlexCards(
      fcInfo** pInfo
)
```

Parameters

pInfo

[OUT] linked list of fcInfo (Obsolete) objects

Return values

If the function succeeds, the return value is 0. If the function fails the content of `pInfo` is not valid. The error code `NULL_PARAMETER` is returned if `pInfo` parameter is a null pointer. If the memory allocation fails, the error code `MEMORY_ALLOCATION_FAILED` is returned.

Remarks

If a connection to a FlexCard is already opened, this FlexCard does not appear in this list. If the function succeeds, there will always be one valid fcInfo (Obsolete) structure regardless if there is a FlexCard in the system or not. This functionality is given to provide version information about the DLL / library. The version information concerning the hardware is only valid if the identifier **(*pInfo->FlexCardId*) is not 0**.

	Information
	This function allocates memory for use. To prevent memory leaks you have to free it up by calling the function fcFreeMemory with the type *fcMemoryTypeInfo*. From FlexCard API version S2V0-F on it is possible to use four FlexCards in one PC. With FlexCard API versions up to S2V0-F it isn't possible to use two FlexCards in one PC at the same time. That means that only the first inserted FlexCard can be used. The second one doesn't appear in the list of available FlexCards.

See Also

fcInfo (Obsolete)

4.10.5 FCBGETENUMFLEXCARDSV2 (OBSOLETE)

	Information
	This function is obsolete. Please use fcbGetEnumFlexCardsV3 instead.

This function returns a linked list of the installed FlexCards found on the system. To free the memory, which was allocated by this function, please use the function fcFreeMemory with type *fcMemoryTypeInfoV2.*

```
fcError fcbGetEnumFlexCardsV2(
      fcInfoV2** pInfoV2
)
```

Parameters

pInfoV2

[OUT] linked list of fcInfoV2 (Obsolete) objects

Return values

If the function succeeds, the return value is 0. If the function fails the content of `pInfoV2` is not valid. The error code `NULL_PARAMETER` is returned if `pInfoV2` parameter is a null pointer. If the memory allocation fails, the error code `MEMORY_ALLOCATION_FAILED` is returned.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Remarks

If the function succeeds, there will always be one valid fcInfoV2 (Obsolete) structure regardless if there is a FlexCard in the system or not. This functionality is given to provide version information about the DLL / library. The version information concerning the hardware is only valid if the identifier **(*pInfoV2->FlexCardId*) is not 0**.

	Information
	This function allocates memory for use. To prevent memory leaks you have to free it up by calling the function fcFreeMemory with the type *fcMemoryTypeInfoV2*.

See Also

fcInfoV2 (Obsolete)

4.10.6 FCBMONITORINGSTART (OBSOLETE)

	Information
	This function is obsolete. Please use fcbFRMonitoringStart instead.

This function is used to start the monitoring of a FlexRay bus. Once called, the function changes the communication controller state from configuration state to normal active state (if the cluster integration succeeds). The current communication controller state can be read using the function fcbGetCcState (Obsolete). If the FlexCard is synchronized with the cluster the function fcbGetCcState (Obsolete) will return the value `fcStateNormalActive`.

```
fcError fcbMonitoringStart(
      fcHandle hFlexCard,
      fcMonitoringModes mode,
      bool restartTimestamps,
      bool enableCycleStartEvents
      bool enableColdstart,
      bool enableWakeup
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

Mode

[IN] The monitoring mode. Not every monitoring mode is supported by all communication controllers. See fcMonitoringModes for details.

restartTimestamps

[IN] Set this parameter to false to restart the measurement without resetting the FlexCard timestamp. Set it to true to start the measurement from the beginning. The timestamps have micro second resolution.

enableCycleStartEvents

[IN] Set this parameter to true to enable the cycle start events in order that at the beginning of every cycle the event `fcNotificationTypeCycleStarted` is signalled.

enableColdstart

[IN] Set this parameter to true to allow the FlexCard to initialize the cluster communication, otherwise the coldstart inhibit mode is active. This feature can not be used in the monitoring modes fcMonitoringDebug and fcMonitoringDebugAsynchron.

enableWakeup

[IN] Set this parameter to true to transmit a wakeup pattern on the configured wakeup channel (FlexRay Protocol Specification: pWakeupChannel). A cluster wakeup must precede the communication start up to ensure that all nodes in a cluster are awake. The minimum

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

requirement for a cluster wakeup is that all bus drivers are supplied with power. This feature can not be used in the monitoring modes fcMonitoringDebug and fcMonitoringDebugAsynchron.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Remarks

After the monitoring has started, the user should check if the integration in the cluster was successful: fcbGetCcState (Obsolete) should return the state `fcStateNormalActive`.

	Information
	After the monitoring has successfully started, the receive process has to be started as soon as possible to avoid an overflow (error packet fcErrFlexcardOverflow is received). Once an overflow occurred, no more packets can be received. The monitoring has to be stopped and started again.

See Also

fcbMonitoringStop (Obsolete), fcbGetCcState (Obsolete), fcMonitoringModes, fcbSetEventHandle (Obsolete)

4.10.7 FCBMONITORINGSTOP (OBSOLETE)

	Information
	This function is obsolete. Please use fcbFRMonitoringStop instead.

This function stops the FlexRay bus measurement. The communication controller is set back in its configuration state.

```
fcError fcbMonitoringStop(
      fcHandle hFlexCard
)
```

Parameters

hFlexCard

[IN] Handle to FlexCard

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcbMonitoringStart (Obsolete)

4.10.8 FCBGETCCSTATE (OBSOLETE)

	Information
	This function is obsolete. Please use fcbFRGetCCState instead.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

This function returns the current communication controller POC state. For a description of possible states, refer to the enumeration fcState. This function should be used to check if the integration into a FlexRay cluster has succeeded.

```
fcError fcbGetCcState(
      fcHandle hFlexCard,
      fcState* pState
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

pState

[OUT] Current communication controller state

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See

fcbMonitoringStart (Obsolete), fcbMonitoringStop (Obsolete)

4.10.9 FCBSETTRANSCEIVERSTATE (OBSOLETE)

	Information
	This function is obsolete. Please use fcbFRSetTransceiverState instead.

This function sets the transceiver mode individually for each channel.

```
fcError fcbSetTransceiverState (
      fcHandle hFlexCard,
      fcTransceiverState stateChannelA,
      fcTransceiverState stateChannelB
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

stateChannelA

[IN] The new transceiver state for channel A

stateChannelB

[IN] The new transceiver state for channel B

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Remarks

If one of the transceivers is in the sleep mode and the transceiver detects a wakeup event, the notification event *fcNotificationTypeWakeup* is fired once only.

See

fcTransceiverState, fcbMonitoringStart (Obsolete), fcbGetTransceiverState (Obsolete)

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

4.10.10 FCBGETTRANSCEIVERSTATE (OBSOLETE)

Information
This function is obsolete. Please use fcbFRGetTransceiverState instead.

This function gets the transceiver state individually for each channel.

```
fcError fcbGetTransceiverState (
      fcHandle hFlexCard,
      fcTransceiverState* stateChannelA,
      fcTransceiverState* stateChannelB
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

stateChannelA

[OUT] The current transceiver state for channel A

stateChannelB

[OUT] The current transceiver state for channel B

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Remarks

If one of the transceiver is in the sleep mode and the transceiver detects a wakeup event, the notification event *fcNotificationTypeWakeup* is fired once only.

See

fcTransceiverState, fcbMonitoringStart (Obsolete), fcbSetTransceiverState (Obsolete)

4.10.11 FCBSETEVENTHANDLE (OBSOLETE)

Information
This function is obsolete. Please use fcbSetEventHandleV2 or fcbSetEventHandleSemaphore instead.

This function registers an event handle for a specific notification type.

```
fcError fcbSetEventHandle(
      fcHandle hFlexCard,
      fcHandle hEvent,
      fcNotificationType type
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

hEvent

[IN] Event handle to be registered to signal when a new cycle starts or a timer interval has elapsed depending on the given *type*.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Type

[IN] The notification type for which the event has to be registered.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcNotificationType

4.10.12 FCBTRANSMIT (OBSOLETE)

	Information
	This function is obsolete. Please use fcbFRTransmit instead.

This function writes a data frame into a communication controller transmit buffer of the FlexCard. The frame should normally be transmitted in the next cycle. If the transmit acknowledgment is activated, an acknowledge packet is generated as soon as the frame has been transmitted. This function should only be called when the FlexCard is in normal active state or when all message buffer configurations have been done.

```
fcError fcbTransmit(
      fcHandle hFlexCard,
      fcDword bufferId,
      fcWord payload[],
      fcByte payloadLength
);
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

bufferId

[IN] The id of the message buffer used for the transmission

payload

The payload data to be transmitted

payloadLength

The size of the payload data (number of 2-byte words)

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

The transmission may fail, if the buffer is currently in use (fcGetErrorCode returns `MSG_BUFFER_BUSY`). In that case retry later.

Remarks

The payload data has to be organized as follows: if Data0 is the first byte to transmit and Data1 the second byte to transmit, then the high byte (Bit 8 – 15) of payload[0] contains Data1, the low byte (Bit0-7) of payload[0] contains Data0, etc.

Parameter `payload`	`payload[0]` (Word 0)		`payload[1]` (Word 1)		…
	High byte	Low byte	High byte	Low byte	…
FlexRay payload segment	Data 1	Data 0	Data 3	Data 2	…

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

4.10.13 FCBTRANSMITSYMBOL (OBSOLETE)

	Information
	This function is obsolete. Please use fcbFRTransmitSymbol instead.

This function transmits a symbol in the symbol window segment. It can only be called if the communication controller is in the POC state NORMAL_ACTIVE. For a list of available symbols to be transmitted, see the enumeration fcSymbolType.

```
fcError fcbTransmitSymbol(
      fcHandle hFlexCard,
      fcSymbolType symbol
);
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

symbol

[IN] Type of symbol to transmit

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

4.10.14 FCBSETCCREGISTER (OBSOLETE)

	Information
	This function is obsolete. Please use fcbFRSetCcRegister instead.

This function writes a value in a given register of the communication controller. Not every register can be written (e.g. the registers belonging to the message buffer configuration or some interrupt settings).

```
fcError fcbSetCcRegister(
      fcHandle hFlexCard,
      fcDword address,
      fcDword value
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

address

[IN] Address of the CC register to be written

value

[IN] The value to be written

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information. If the register can not be written the error code REGISTER_NOT_WRITEABLE is returned.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Remarks

For a register description, refer to the specification of the corresponding communication controller. Modifying one of the following registers will reset message buffers with their default settings (FIFO receive buffers). The user's message buffers configuration will not be valid anymore.
Bosch E-Ray: MHDC (0x0098) and GTUC7 (0x00B8)

Information
Not all register of a communication controller can be set. The base API will modify some parameters so that the operating of the FlexCard is guaranteed (e.g. interrupt settings). Access is denied to all registers which are used for message buffer configuration.

See Also

fcbGetCcRegister (Obsolete)

4.10.15 FCBGETCCREGISTER (OBSOLETE)

Information
This function is obsolete. Please use fcbFRGetCcRegister instead.

This function reads and returns the content of a given register of the communication controller.

```
fcError fcbGetCcRegister(
      fcHandle hFlexCard,
      fcDword  address,
      fcDword* pValue
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

address

[IN] Address of the CC register to be read.

pValue

[OUT] The content of the desired CC register.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information. If the register cannot be read the error code `REGISTER_NOT_READABLE` is returned.

Remarks

Not every register can be read. For a register description, refer to the specification of the corresponding communication controller.

See Also

fcbSetCcRegister (Obsolete)

3-0009-0S01-D03_API Documentation_D1V12-F.doc

4.10.16 FCBCHICCCONFIGURATION (OBSOLETE)

Information
This function is obsolete. Please use fcbFRSetCcConfigurationChi instead.

This function configures the communication controller of the FlexCard with a FlexConfig compatible configuration string (CHI File). The configuration string contains the global FlexRay parameter and/or the message buffer configuration. The payload data for transmit message buffers are not set by this function. Before the configuration of the communication controller starts, all message buffers are reset to their default settings (FIFO buffer).

```
fcError fcbChiCcConfiguration(
      fcHandle hFlexCard,
      const char* szChi
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

szChi

[IN] Pointer to null-terminated CHI content string (refer to the CHI string example section).
Please note: Do not use the CHI file name here, but the content of the CHI file as parameter value.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Information
Internally, the function uses the function fcbSetCcRegister (Obsolete); therefore the same restrictions as for writing registers exist.

See Also

fcbSetCcRegister (Obsolete)

4.10.17 FCBCANDBCCCONFIGURATION (OBSOLETE)

Information
This function is obsolete. Please use fcbFRSetCcConfigurationCANdb instead.

This function configures the communication controller of the FlexCard with a CANdb compatible string. The configuration string contains the global FlexRay parameter and/or the message buffer configuration. Before the configuration of the communication controller starts, all message buffers are reset to their default settings (FIFO buffer).

```
fcError fcbCanDbCcConfiguration(
      fcHandle hFlexCard,
      const char* szCanDb
)
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Parameters

hFlexCard

[IN] Handle to a FlexCard

szCanDb

[IN] Pointer to null-terminated CANdb string

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Remarks

This function is only available in the Windows FlexCard driver. The FlexCard Linux and Xenomai drivers don't support this function.

	Information
	Internally, the function uses the fcbSetCcRegister (Obsolete) function; therefore the same restrictions as for writing a register exist.

4.10.18 FCBCONFIGUREMESSAGEBUFFER (OBSOLETE)

	Information
	This function is obsolete. Please use fcbFRConfigureMessageBuffer instead.

This function configures transmit, receive and FIFO message buffers of the communication controller. Configuring message buffers is only allowed if the communication controller is in its configuration state, *fcStateConfig*.

```
fcError fcbConfigureMessageBuffer(
      fcHandle hFlexCard,
      fcDword* bufferId,
      fcMsgBufCfg cfg
);
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

bufferId

[OUT] Message buffer identifier. If the configuration was successful the message buffer identifier is greater than 0. This identifier will be required to transmit the content of the buffer (in the case of a transmit buffer).

Cfg

[IN] Message buffer configuration parameters

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Remarks

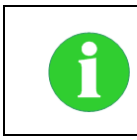
Before configuring the message buffers, it is necessary to set up the global communication parameters (cluster parameters). Internally the FlexCard uses the FIFO buffers as receive buffers, therefore we recommend using FIFO message buffers as much as possible.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

See Also

fcMsgBufCfg, fcMsgBufCfgTx, fcMsgBufCfgRx, fcMsgBufCfgFifo

4.10.19 FCBRECONFIGUREMESSAGEBUFFER (OBSOLETE)



Information
This function is obsolete. Please use fcbFRReconfigureMessageBuffer instead.

This function reconfigures transmit, receive and FIFO message buffers of the communication controller. A reconfiguration is only allowed for message buffers which are already configured. This function is available in all states of the CC. Not all configuration settings can be modified in monitoring state. Refer to the documentation of the message buffer structures for further details.

```
fcError fcbReconfigureMessageBuffer(
      fcHandle hFlexCard,
      fcDword bufferId,
      fcMsgBufCfg cfg
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

bufferId

[IN] The identifier of the message buffer which should be reconfigured.

Cfg

[IN] Message buffer configuration parameters.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcMsgBufCfg, fcMsgBufCfgTx, fcMsgBufCfgRx, fcMsgBufCfgFifo, fcbConfigureMessageBuffer (Obsolete), fcbGetCcMessageBuffer (Obsolete)

4.10.20 FCBGETCCMESSAGEBUFFER (OBSOLETE)

Information
This function is obsolete. Please use fcbFRGetMessageBuffer instead.

This function reads a specific message buffer configuration.

```
fcError fcbGetCcMessageBuffer(
      fcHandle hFlexCard,
      fcDword bufferId,
      fcMsgBufCfg* cfg
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

bufferId

[IN] The identifier of the message buffer to be read

cfg

[OUT] The configuration parameters of the specified message buffer.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Remarks

The buffer with id 1 is always a FIFO message buffer.

See Also

fcMsgBufCfg, fcMsgBufCfgTx, fcMsgBufCfgRx, fcMsgBufCfgFifo, fcbConfigureMessageBuffer (Obsolete)

4.10.21 FCBRESETCCMESSAGEBUFFER (OBSOLETE)

	Information
	This function is obsolete. Please use fcbFRResetMessageBuffers instead.

This function resets the communication controller message buffers. After calling this function, all message buffers are configured as receive FIFO – with maximal payload (depends on the communication controller).

```
fcError fcbResetCcMessageBuffer(
      fcHandle hFlexCard
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

4.10.22 FCBFILTER (OBSOLETE)

	Information
	This function is obsolete. Please use fcbFRSetSoftwareAcceptanceFilter or fcbFRSetHardwareAcceptanceFilter instead.

This function configures the frame ids accepted by the device driver. Only the ids which are in the filter list are forwarded to the user application, all other frames are rejected. To accept all frames set the parameters *pData* to NULL and nSize to zero or configure a single frame id of zero.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
fcError fcbFilter(
      fcHandle hFlexCard,
      fcChannel channel,
      fcDword* pData,
      fcDword size
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

channel

[IN] FlexCard channel(s) concerned by the filter

pData

[IN] Pointer to a fcDword array containing the ids accepted by the device driver. Each element (fcDword) contains one frame identifier.

fcDword	fcDword
ID x	ID y

size

[IN] Number of ids in the array

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

4.10.23 FCBSETCCTIMERCONFIG (OBSOLETE)

Information
This function is obsolete. Please use fcbFRSetCcTimerConfig instead.

This function configures the communication controller timer interrupt. To get a notification when the communication controller timer interval elapsed, an event of type *fcNotificationTypeCcTimer* has to be registered by the function fcbSetEventHandle (Obsolete). Additionally the communication controller timer can be enabled / disabled by this function.

```
fcError fcbSetCcTimerConfig(
      fcHandle hFlexCard,
      fcCcTimerCfg cfg,
      bool bEnable
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

Cfg

[IN] The communication controller timer configuration.

bEnable

[IN] Set to true to enable the cc timer, and to false to disable it.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcbSetEventHandle (Obsolete), fcCcTimerCfg, fcbGetCcTimerConfig (Obsolete)

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

4.10.24 FCBGETCCTIMERCONFIG (OBSOLETE)

Information
This function is obsolete. Please use fcbFRGetCcTimerConfig instead.

This function reads the communication controller timer configuration.

```
fcError fcbGetCcTimerConfig(
      fcHandle hFlexCard,
      fcCcTimerCfg* pCfg
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

pCfg

[OUT] The configuration parameters of the cc timer.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcCcTimerCfg, fcbSetCcTimerConfig (Obsolete)

4.10.25 FCBCALCULATEMACROTICKOFFSET (OBSOLETE)

Information
This function is obsolete. Please use fcbFRCalculateMacrotickOffset instead.

This function calculates the macrotick offset for a specific cycle position in a FlexRay cycle.

```
fcError fcbCalculateMacrotickOffset(
      fcHandle hFlexCard,
      fcCyclePos CyclePosition,
      fcDword SlotOrMiniSlotId,
      fcDword* pValue
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

CyclePosition

[IN] The cycle position of type fcCyclePos.

SlotOrMiniSlotId

[IN] This parameter is used for a cycle position of *fcCyclePosStaticSlot* and *fcCyclePosDynamicMiniSlot* to calculate the macrotick offset for a static slot or a dynamic mini slot id.

pValue

[OUT] The macrotick offset value.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcCyclePos, fcCcTimerCfg, fcbSetCcTimerConfig (Obsolete)

4.10.26 TRIGGER CONFIGURATION (OBSOLETE)

Information
This configuration is obsolete. Please see Trigger configuration instead.

If the FlexCard is equipped with a trigger interface, the FlexCard has the ability to receive trigger events and forward them to the user application. This feature allows e.g. a synchronization of different bus analyzer. To configure and activate this feature, use the following structures and functions. The trigger event data is received as fcTriggerInfoPacket (Obsolete) with the fcbReceive function.

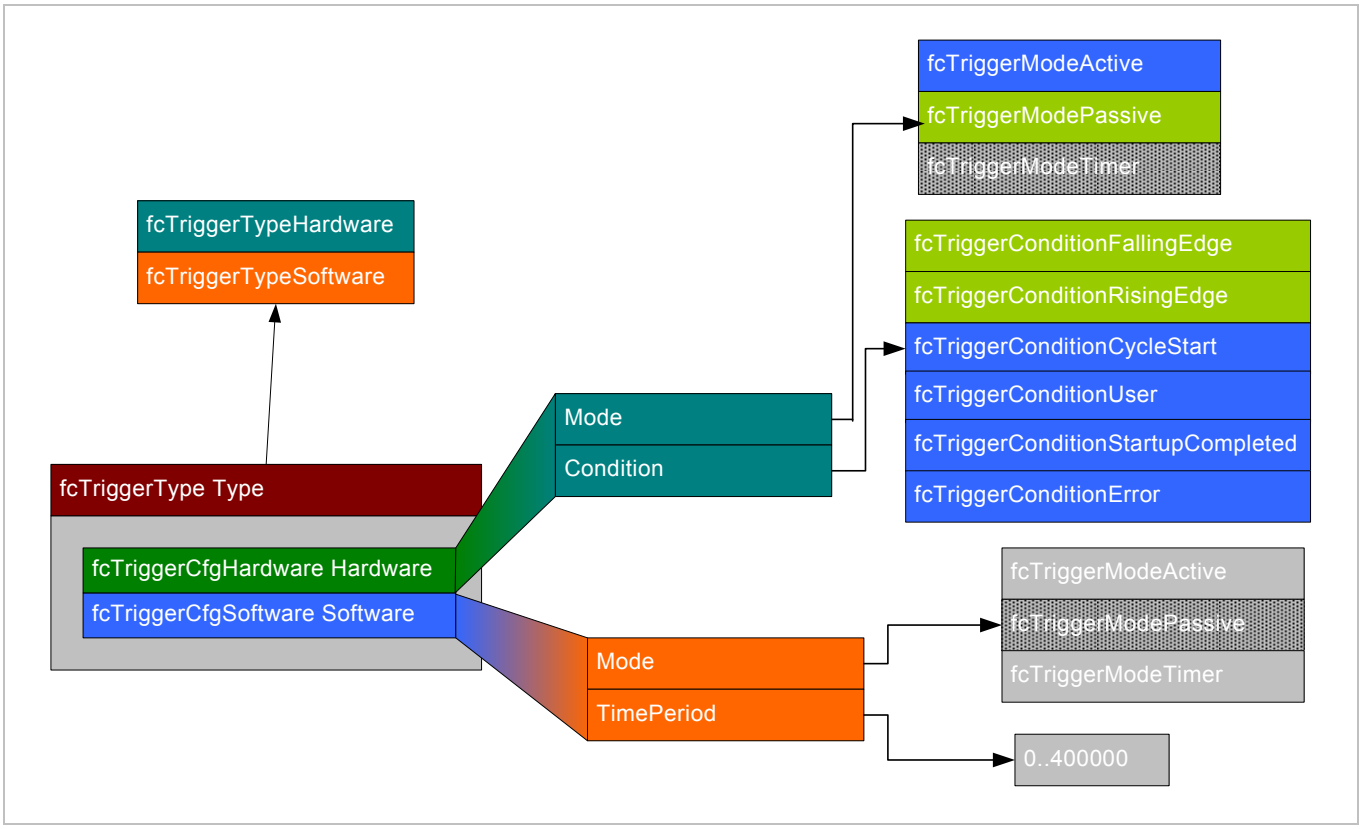


Figure 10: Overview fcbTriggerCfg structure

4.10.27 TYPEDEFINITIONS (OBSOLETE)

4.10.27.1 FCTRIGGERCFGHARDWARE (OBSOLETE)

This structure configures the hardware trigger. In the passive mode, the FlexCard waits for trigger events on its input line and generates a fcTriggerInfoPacket (Obsolete) object each time a trigger event is received. In this mode, the parameter *Condition* specifies on which condition the input signal will be recognized as a trigger event. In the active mode, the FlexCard generates a pulse on its output line when a

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

trigger event is signalled. In this mode, the parameter *Condition* specifies on which condition a pulse will be generated by the FlexCard. For information about the pin assignment of the input and output line, refer to the user manual of the FlexCard.

```
Typedef struct fcTriggerCfgHardware
{
      fcTriggerMode Mode;
      fcTriggerCondition Condition;
}fcTriggerCfgHardware;
```

Members

Mode

Set the trigger mode (active or passive mode). The hardware trigger does not support the timer mode.

fcTriggerCondition

Depending on the mode, the following conditions can be used:

- Passive mode:
  - Falling edge (Trigger packet is generated on falling edge of the input signal)
  - Rising edge (Trigger packet is generated on rising edge of the input signal)
- Active mode:
  - Cycle start (A pulse is generated on the output line when a new cycle starts)
  - User (A pulse is generated on the output line when the user is calling the function fcbTrigger)
  - Error (A pulse is generated on the output line when an error occurred)
  - Start up completed (A pulse is generated on the output line when the start up was completed)

See Also

fcTriggerCfg (Obsolete), fcTriggerCondition (Obsolete), fcTriggerMode (Obsolete)

4.10.27.2 FCTRIGGERCFGSOFTWARE (OBSOLETE)

This structure configures the software trigger. In active mode an fcTriggerInfoPacket (Obsolete) object is generated each time the function fcbTrigger (Obsolete) is called. In the timer mode an fcTriggerInfoPacket (Obsolete) object is generated every *TimePeriod* millisecond. A zero *TimePeriod* means that no fcTriggerInfoPacket (Obsolete) will be generated.

```
Typedef struct fcTriggerCfgSoftware
{
      fcTriggerMode Mode;
      fcDword  TimePeriod;
}fcTriggerCfgSoftware;
```

Members

Mode

Set the trigger mode (active or timer mode). The software trigger does not support the passive mode.

TimePeriod

This parameter is only used in timer mode. Every *TimePeriod* milliseconds (range: 0 – 400000) a trigger packet will be generated.

See Also

fcTriggerCfg (Obsolete), fcTriggerMode (Obsolete)

4.10.27.3 FCTRIGGERCFG (OBSOLETE)

This structure is used for the configuration of a trigger. Only one trigger at a time (hardware or software) can be used and the conditions cannot be combined.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef struct fcTriggerCfg
{
    fcTriggerType Type;
    union
    {
        fcTriggerCfgHardware Hardware;
        fcTriggerCfgSoftware Software;
    };
}fcTriggerCfg;
```

Members

Type
: Type of trigger (hardware or software)

Hardware
: Configuration of hardware trigger

Software
: Configuration of software trigger

See Also

fcTriggerType (Obsolete), fcTriggerCfgHardware (Obsolete), fcTriggerCfgSoftware (Obsolete), fcbTrigger (Obsolete)

4.10.27.4 FCTRIGGERINFOPACKET (OBSOLETE)

This structure provides information about a trigger packet.

```
Typedef struct fcTriggerInfoPacket
{
    fcTriggerType Type;
    fcTriggerCondition Condition;
    fcDword TimeStamp;
    fcDword SequenceCount;
    fcQuad PerformanceCounter;

}fcTriggerInfoPacket;
```

Members

Type
: Type of trigger info packet

Condition
: The fulfilled condition which has caused the trigger packet generation

TimeStamp
: The FlexCard time stamp (1 µs resolution). Indicates the time at which the packet was generated.

SequenceCount
: Sequence count for each signal

PerformanceCounter
: Variable that receives the current performance-counter value. This value is only valid for software triggers (*fcTriggerTypeSoftware*).

See Also

fcPacket

4.10.28 ENUMERATIONS (OBSOLETE)

4.10.28.1 FCTRIGGERCONDITION (OBSOLETE)

This enumeration defines the conditions available for a trigger configuration.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef enum fcTriggerCondition
{
      fcTriggerConditionFallingEdge         = 1,
      fcTriggerConditionRisingEdge          = 2,
      fcTriggerConditionCycleStart          = 3,
      fcTriggerConditionUser                = 4,
      fcTriggerConditionErrorDetected       = 5,
      fcTriggerConditionStartupCompleted    = 6,
      fcTriggerConditionTimer               = 7,
}fcTriggerEdge;
```

Members

fcTriggerConditionFallingEdge
: Passive mode condition: input trigger is detected on falling edge

fcTriggerConditionRisingEdge
: Passive mode condition: input trigger is detected on rising edge

fcTriggerConditionCycleStart
: *Active mode* condition*:* output trigger is set on start of a new FlexRay cycle

fcTriggerConditionUser
: Active mode condition: output trigger is set by the user

fcTriggerConditionErrorDetected
: Active mode condition: output trigger is set if an error was detected

fcTriggerConditionStartupCompleted
: Active mode condition: output trigger is set when the start-up was completed

fcTriggerConditionTimer
: Timer mode condition: Internal trigger is set by the software timer (neither input nor output trigger signal is used)

See Also

fcTriggerCfgHardware (Obsolete)

4.10.28.2 FCTRIGGERTYPE (OBSOLETE)

This enumeration defines the different trigger types.

```
Typedef enum fcTriggerType
{
    fcTriggerTypeHardware   = 1,
    fcTriggerTypeSoftware   = 2,
} fcTriggerType;
```

Members

fcTriggerTypeHardware
: Hardware trigger

fcTriggerTypeSoftware
: Software trigger

See Also

fcTriggerCfg (Obsolete)

4.10.28.3 FCTRIGGERMODE (OBSOLETE)

This enumeration defines the different trigger modes.

```
Typedef enum fcTriggerMode
{
    fcTriggerModeActive  = 1,
    fcTriggerModePassive = 2,
    fcTriggerModeTimer   = 3,
}fcTriggerMode;
```

Members

fcTriggerModeActive
: Active mode: triggered by FlexCard or by user

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

fcTriggerModePassive
Passive mode: triggered by external hardware
fcTriggerModeTimer
Timer mode: triggered by software timer.

See Also
fcTriggerCfgHardware (Obsolete), fcTriggerCfgSoftware (Obsolete)

4.10.29 FCBTRIGGER (OBSOLETE)

This function configures and starts/stops a trigger. For further information, refer to the structures fcTriggerCfgSoftware and fcTriggerCfgHardware.

```
fcError fcbTrigger(
      fcHandle hFlexCard,
      bool enable,
      fcTriggerCfg cfg
)
```

Parameters
hFlexCard
[IN] Handle to a FlexCard
enable
[IN] Set to true to enable the trigger, and to false to disable it.
Cfg
[IN] The trigger configuration

Return values
If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also
fcTriggerCfg (Obsolete)

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

5 ADDITIONAL FLEXRAY API

	Information
	All enumerations, structures and function in this chapter are initially supported by FlexCard API version S4V2-F.

5.1 INITIALIZATION

5.1.1 FCBFRMONITORINGSTART

This function is used to start the monitoring of a FlexRay bus. Once called, the function changes the communication controller state from configuration state to normal active state (if the cluster integration succeeds). The current communication controller state can be read using the function fcbFRGetCCState. If the FlexCard is synchronized with the cluster the function fcbFRGetCCState will return the value *fcStateNormalActive.*

```
fcError fcbFRMonitoringStart(
      fcHandle hFlexCard,
      fcCC CC,
      fcMonitoringModes mode,
      bool restartTimestamps,
      bool enableCycleStartEvents
      bool enableColdstart,
      bool enableWakeup
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

CC

[IN] Communication controller index

mode

[IN] The monitoring mode. Not every monitoring mode is supported by all communication controllers. See fcMonitoringModes for details.

restartTimestamps

[IN] Set this parameter to false to restart the measurement without resetting the FlexCard timestamp. Set it to true to start the measurement from the beginning. The timestamps have micro second resolution.

enableCycleStartEvents

[IN] Set this parameter to true to enable the cycle start events in order that at the beginning of every cycle the event `fcNotificationTypeFRCycleStarted` is signalled.

enableColdstart

[IN] Set this parameter to true to allow the FlexCard to initialize the cluster communication, otherwise the coldstart inhibit mode is active. This feature can not be used in the monitoring modes fcMonitoringDebug and fcMonitoringDebugAsynchron.

enableWakeup

[IN] Set this parameter to true to transmit a wakeup pattern on the configured wakeup channel (FlexRay Protocol Specification: pWakeupChannel). A cluster wakeup must precede the communication start up to ensure that all nodes in a cluster are awake. The minimum requirement for a cluster wakeup is that all bus drivers are supplied with power. This feature

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

can not be used in the monitoring modes fcMonitoringDebug and fcMonitoringDebugAsynchron.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Remarks

After the monitoring has started, the user should check if the integration in the cluster was successful: fcbFRGetCCState should return the state `fcStateNormalActive`.

	Information
	After the monitoring has successfully started, the receive process has to be started as soon as possible to avoid an overflow (error packet fcErrFlexcardOverflow is received). Once an overflow occurred, no more packets can be received. The monitoring has to be stopped and started again.

See Also

fcCC, fcbFRMonitoringStop, fcbFRGetCCState, fcMonitoringModes, fcbSetEventHandleV2, fcbSetEventHandleSemaphore

Example

```
// Precondition: valid flexcard handle exists and the flexcard is
// already configured.
fcCC eCC = fcCC1;
fcError e = fcbFRMonitoringStart(hFlexCard,eCC,fcMonitoringNormal,true,
                                 false,false,false);
if (0 == e)
{
      bool synchronized = false;
      bool timeout = false;
      DWORD maxTime = ::GetTickCount() + 2000;
      fcState currentState = fcStateUnknown;

      // Check if the FlexCard is synchronized
      do
      {
            fcbFRGetCcState(hFlexCard, eCC, &currentState);
            synchronized = (currentState == fcStateNormalActive);
            timeout           = ::GetTickCount() >= maxTime;

      } while ( ! synchronized && ! timeout);

      if (synchronized)
      {
            // Start your receive thread/routine
            //   ...
      }
      else
      {
            // if we timed out, we stop the monitoring
            fcbFRMonitoringStop(hFlexCard,eCC);
      }
}
else
{
      // error handling …
}
```

5.1.2 FCBFRMONITORINGSTOP

This function stops the FlexRay bus measurement. The communication controller is set back in its configuration state.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
fcError fcbFRMonitoringStop(
      fcHandle hFlexCard,
      fcCC CC
)
```

Parameters

hFlexCard

[IN] Handle to FlexCard

CC

[IN] Communication controller index

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcCC, fcbFRMonitoringStart

5.1.3 FCBFRGETCCSTATE

This function returns the current communication controller POC state. For a description of possible states, refer to the enumeration fcState. This function should be used to check if the integration into a FlexRay cluster has succeeded.

```
fcError fcbFRGetCcState(
      fcHandle hFlexCard,
      fcCC CC,
      fcState* pState
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

CC

[IN] Communication controller index

pState

[OUT] Current communication controller state

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See

fcCC, fcState, fcbFRMonitoringStart, fcbFRMonitoringStop

Example

See example fcbFRMonitoringStart

5.1.4 FCBFRSETTRANSCEIVERSTATE

This function sets the transceiver mode individually for each channel.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
fcError fcbFRSetTransceiverState (
      fcHandle hFlexCard,
      fcCC CC,
      fcTransceiverState stateChA,
      fcTransceiverState stateChB
)
```

Parameters

hFlexCard
[IN] Handle to a FlexCard

CC
[IN] Communication controller index

stateChA
[IN] The new transceiver state for channel A

stateChB
[IN] The new transceiver state for channel B

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Remarks

If one of the transceivers is in the sleep mode and the transceiver detects a wakeup event, the notification event *fcNotificationTypeFRWakeup* is fired once only.

See

fcCC, fcTransceiverState, fcbFRMonitoringStart, fcbFRGetTransceiverState

5.1.5 FCBFRGETTRANSCEIVERSTATE

This function gets the transceiver state of a selected communication controller individually for each channel.

```
fcError fcbFRGetTransceiverState (
      fcHandle hFlexCard,
      fcCC CC,
      fcTransceiverState* pStateChA,
      fcTransceiverState* pStateChB
)
```

Parameters

hFlexCard
[IN] Handle to a FlexCard

CC
[IN] Communication controller index

pStateChA
[OUT] The current transceiver state for channel A

pStateChB
[OUT] The current transceiver state for channel B

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Remarks

If one of the transceiver is in the sleep mode and the transceiver detects a wakeup event, the notification event fcNotificationTypeFRWakeup is fired once only.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

See

fcCC, fcTransceiverState, fcbFRMonitoringStart, fcbFRSetTransceiverState

5.2 CONFIGURATION

5.2.1 ENUMERATIONS

5.2.1.1 FCFRBAUDRATE

This enumeration defines the various baud rates on the FlexRay bus.

```
Typedef enum fcFRBaudRate
{
      fcFRBaudRateNone = 0,
      fcFRBaudRate2M5,
      fcFRBaudRate5M,
      fcFRBaudRate10M,
} fcFRBaudRate;
```

Members

fcFRBaudRateNone
: No baud rate defined

fcFRBaudRate2M5
: Defines the baud rate 2.5 Mbit/s

fcFRBaudRate5M
: Defines the baud rate 5 Mbit/s

fcFRBaudRate10M
: Defines the baud rate 10 Mbit/s

See Also

fcFRCcConfig

5.2.2 STRUCTURES

5.2.2.1 FCFRCCCONFIG

This structure describes the configuration of the FlexRay communication controller.

```
Typedef struct fcFRCcConfig
{
      fcFRBaudRate BaudRate;
      fcDword gdActionPointOffset;
      fcDword gdCASRxLowMax;
      fcDword gdDynamicSlotIdlePhase;
      fcDword gdMinislot;
      fcDword gdMinislotActionPointOffset;
      fcDword gdNIT;
      fcDword gdStaticSlot;
      fcDword gdTSSTransmitter;
      fcDword gdWakeupSymbolRxIdle;
      fcDword gdWakeupSymbolRxLow;
      fcDword gdWakeupSymbolRxWindow;
      fcDword gdWakeupSymbolTxIdle;
      fcDword gdWakeupSymbolTxLow;
      fcDword gColdStartAttempts;
      fcDword gListenNoise;
      fcDword gMacroPerCycle;
      fcDword gMaxWithoutClockCorrectionFatal;
      fcDword gMaxWithoutClockCorrectionPassive;
      fcDword gNetworkManagementVectorLength;
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
      fcDword gNumberOfMinislots;
      fcDword gNumberOfStaticSlots;
      fcDword gOffsetCorrectionStart;
      fcDword gPayloadLengthStatic;
      fcDword gSyncNodeMax;
      fcDword pdAcceptedStartupRange;
      fcDword pdListenTimeout;
      fcDword pdMaxDrift;
      fcDword pAllowHaltDueToClock;
      fcDword pAllowPassiveToActive;
      fcChannel pChannelsMTS;
      fcChannel pChannels;
      fcDword pClusterDriftDamping;
      fcDword pDecodingCorrection;
      fcDword pDelayCompensationA;
      fcDword pDelayCompensationB;
      fcDword pExternOffsetCorrection;
      fcDword pExternRateCorrection;
      fcDword pKeySlotUsedForStartup; //NOT USED.
      fcDword pKeySlotUsedForSync; //NOT USED.
      fcDword pLatestTx;
      fcDword pMacroInitialOffsetA;
      fcDword pMacroInitialOffsetB;
      fcDword pMicroInitialOffsetA;
      fcDword pMicroInitialOffsetB;
      fcDword pMicroPerCycle;
      fcDword pOffsetCorrectionOut;
      fcDword pRateCorrectionOut;
      fcDword pSingleSlotEnabled;
      fcChannel pWakeupChannel;
      fcDword pWakeupPattern;
      fcDword vExternOffsetControl;
      fcDword vExternRateControl;

      fcDword Reserved[16];
} fcFRCcConfig;
```

Members

BaudRate

Configures the baud rate on the FlexRay bus.

gdActionPointOffset

Configures the action point offset in macroticks within static slots and symbol window. Must be identical in all nodes of a cluster. Valid values are 1 to 63 MT.

gdCASRxLowMax

Configures the upper limit of the acceptance window for a collision avoidance symbol (CAS). Valid values are 67 to 99 bit times.

gdDynamicSlotIdlePhase

The duration of the dynamic slot idle phase has to be greater or equal than the idle detection time. Must be identical in all nodes of a cluster. Valid values are 0 to 2 Minislot.

gdMinislot

Configures the duration of a minislot in macroticks. The minislot length must be identical in all nodes of a cluster. Valid values are 2 to 63 MT.

gdMinislotActionPointOffset

Configures the action point offset in macroticks within the minislots of the dynamic segment. Must be identical in all nodes of a cluster. Valid values are 1 to 31 MT.

gdNIT

Configures the starting point of the Network Idle Time NIT at the end of the communication cycle expressed in terms of macroticks from the beginning of the cycle. The start of NIT is recognized if Macrotick = gMacroPerCycle – gdNIT -1 and the increment pulse of Macrotick is set. Must be identical in all nodes of a cluster. Valid values of “gMacroPerCycle – gdNIT -1” are 7 to 15997 MT. Therefore valid values for the parameter gdNIT are 2 to 805 MT.

gdStaticSlot

Configures the duration of a static slot in macroticks. The static slot length must be identical in all nodes of a cluster. Valid values are 4 to 659 MT.

gdTSSTransmitter

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Configures the duration of the Transmission Start Sequence (TSS) in terms of bit times (1 bit time = 4 µT = 100ns@10Mbps). Must be identical in all nodes of a cluster. Valid values are 3 to 15 bit times.

gdWakeupSymbolRxIdle

Configures the number of bit times used by the node to test the duration of the idle phase of the received wakeup symbol. Must be identical in all nodes of a cluster. Valid values are 14 to 59 bit times.

gdWakeupSymbolRxLow

Configures the number of bit times used by the node to test the duration of the low phase of the received wakeup symbol. Must be identical in all nodes of a cluster. Valid values are 10 to 55 bit times.

gdWakeupSymbolRxWindow

Configures the number of bit times used by the node to test the duration of the received wakeup pattern. Must be identical in all nodes of a cluster. Valid values are 76 to 301 bit times.

gdWakeupSymbolTxIdle

Configures the number of bit times used by the node to transmit the idle phase of the wakeup symbol. Must be identical in all nodes of a cluster. Valid values are 45 to 180 bit times.

gdWakeupSymbolTxLow

Configures the number of bit times used by the node to transmit the low phase of the wakeup symbol. Must be identical in all nodes of a cluster. Valid values are 15 to 60 bit times.

gColdStartAttempts

Configures the maximum number of attempts that a cold starting node is permitted to try to start up the network without receiving any valid response from another node. It can be modified in DEFAULT_CONFIG or CONFIG state only. Must be identical in all nodes of a cluster. Valid values are 2 to 31.

gListenNoise

Configures the upper limit for startup and wakeup listen timeout in the presence of noise expressed as a multiple of pdListenTimeout. The range for gListenNoise is 2 to 16.

gMacroPerCycle

Configures the duration of one communication cycle in macroticks. The cycle length must be identical in all nodes of a cluster. Valid values are 10 to 16000 MT.

gMaxWithoutClockCorrectionFatal

Defines the number of consecutive even/odd cycle pairs with missing clock correction terms that will cause a transition from NORMAL_ACTIVE or NORMAL_PASSIVE to HALT state. Must be identical in all nodes of a cluster. Valid values are 1 to 15 cycle pairs.

gMaxWithoutClockCorrectionPassive

Defines the number of consecutive even/odd cycle pairs with missing clock correction terms that will cause a transition from NORMAL_ACTIVE to NORMAL_PASSIVE state. Must be identical in all nodes of a cluster. Valid values are 1 to 15 cycle pairs.

gNetworkManagementVectorLength

Configures the length of the NM vector. The configured length must be identical in all nodes of a cluster. Valid values are 0 to 12 bytes.

gNumberOfMinislots

Configures the number of minislots within the dynamic segment of a cycle. The number of minislots must be identical in all nodes of a cluster. Valid values are 0 to 7986.

gNumberOfStaticSlots

Configures the number of static slots in a cycle. At least 2 coldstart nodes must be configured to startup a FlexRay network. The number of static slots must be identical in all nodes of a cluster. Valid values are 2 to 1023.

gOffsetCorrectionStart

Determines the start of the offset correction within the NIT phase, calculated from start of cycle. Must be identical in all nodes of a cluster. Valid values are 9 to 15999 MT.

gPayloadLengthStatic

Configures the cluster-wide payload length for all frames sent in the static segment in double bytes. The payload length must be identical in all nodes of a cluster. Valid values are 0 to 127.

gSyncNodeMax

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Maximum number of frames within a cluster with sync frame indicator bit SYN set to ’1’. Must be identical in all nodes of a cluster. Valid values are 2 to 15.

pdAcceptedStartupRange

Number of microticks constituting the expanded range of measured deviation for startup frames during integration. Valid values are 0 to 1875 µT.

pdListenTimeout

Configures wakeup/startup listen timeout in µT. The range for pdListenTimeout is 1284 to 1283846 µT.

pdMaxDrift

Maximum drift offset between two nodes that operate with unsynchronized clocks over one communication cycle in µT. Valid values are 2 to 1923 µT.

pAllowHaltDueToClock

Controls the transition to HALT state due to a clock synchronization error. Valid values are 0 to 1. If a clock sync error occurred the CC will enter HALT state or enter/remain in NORMAL_PASSIVE state.

pAllowPassiveToActive

Defines the number of consecutive even/odd cycle pairs that must have valid clock correction terms before the CC is allowed to transit from NORMAL_PASSIVE to NORMAL_ACTIVE state. If set to zero the CC is not allowed to transit from NORMAL_PASSIVE to NORMAL_ACTIVE state. It can be modified in DEFAULT_CONFIG or CONFIG state only. Valid values are 0 to 31 even/odd cycle pairs.

pChannelsMTS

Selects channels for MTS symbol transmission. The flag is reset by default and may be modified only in DEFAULT_CONFIG or CONFIG state.

pChannels

Configures which channel the node is connected to.

pClusterDriftDamping

Configures the cluster drift damping value used in clock synchronization to minimize accumulation of rounding errors. Valid values are 0 to 20 µT.

pDecodingCorrection

Configures the decoding correction value used to determine the primary time reference point. Valid values are 14 to 143 µT.

pDelayCompensationA

Used to compensate for reception delays on the indicated channel. This covers assumed propagation delay up to cPropagationDelayMax for microticks in the range of 0.0125 to 0.05µs. In practice, the minimum of the propagation delays of all sync nodes should be applied. Valid values are 0 to 200 µT.

pDelayCompensationB

Used to compensate for reception delays on the indicated channel. This covers assumed propagation delay up to cPropagationDelayMax for microticks in the range of 0.0125 to 0.05µs. In practice, the minimum of the propagation delays of all sync nodes should be applied. Valid values are 0 to 200 µT.

pExternOffsetCorrection

Holds the external offset correction value in microticks to be applied by the internal clock synchronization algorithm. The value is subtracted / added from / to the calculated offset correction value. The value is applied during NIT. May be modified in DEFAULT_CONFIG or CONFIG state only. Valid values are 0 to 7 µT.

pExternRateCorrection

Holds the external rate correction value in microticks to be applied by the internal clock synchronization algorithm. The value is subtracted / added from / to the calculated rate correction value. The value is applied during NIT. May be modified in DEFAULT_CONFIG or CONFIG state only. Valid values are 0 to 7 µT.

pKeySlotUsedForStartup

Defines whether the key slot is used to transmit startup frames. The bit can be modified in DEFAULT_CONFIG or CONFIG state only.

1 = Key slot used to transmit startup frame, node is leading or following coldstarter

0 = No startup frame transmission in key slot, node is non-coldstarter

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Not used during configuration. Is set when configuring a message buffer.

`pKeySlotUsedForSync`

Defines whether the key slot is used to transmit sync frames. The bit can be modified in DEFAULT_CONFIG or CONFIG state only.

1 = Key slot used to transmit sync frame, node is sync node

0 = No sync frame transmission in key slot, node is neither sync nor coldstart node

Not used during configuration. Is set when configuring a message buffer.

`pLatestTx`

Configures the maximum minislot value allowed before inhibiting frame transmission in the dynamic segment of the cycle. There is no transmission in dynamic segment if it is set to zero. Valid values are 0 to 7981 minislots.

`pMacroInitialOffsetA`

Configures the number of macroticks between the static slot boundary and the subsequent macrotick boundary of the secondary time reference point based on the nominal macrotick duration. Must be identical in all nodes of a cluster. Valid values are 2 to 72 MT.

`pMacroInitialOffsetB`

Configures the number of macroticks between the static slot boundary and the subsequent macrotick boundary of the secondary time reference point based on the nominal macrotick duration. Must be identical in all nodes of a cluster. Valid values are 2 to 72 MT.

`pMicroInitialOffsetA`

Configures the number of microticks between the actual time reference point on channel A and the subsequent macrotick boundary of the secondary time reference point. The parameter depends on pDelayCompensationA and therefore has to be set for each channel independently. Valid values are 0 to 240 µT.

`pMicroInitialOffsetB`

Configures the number of microticks between the actual time reference point on channel B and the subsequent macrotick boundary of the secondary time reference point. The parameter depends on pDelayCompensationB and therefore has to be set for each channel independently. Valid values are 0 to 240 µT.

`pMicroPerCycle`

Configures the duration of the communication cycle in microticks. Valid values are 640 to 640000 µT.

`pOffsetCorrectionOut`

Holds the maximum permitted offset correction value to be applied by the internal clock synchronization algorithm (absolute value). The CC checks only the internal offset correction value against the maximum offset correction value. Valid values are 5 to 15266 µT.

`pRateCorrectionOut`

Holds the maximum permitted rate correction value to be applied by the internal clock synchronization algorithm. The CC checks only the internal rate correction value against the maximum rate correction value (absolute value). Valid values are 2 to 1923 µT.

`pSingleSlotEnabled`

Selects the initial transmission slot mode. In SINGLE slot mode the CC may only transmit in the preconfigured key slot.

1 = SINGLE Slot Mode (default after hard reset)

0 = ALL Slot Mode.

`pWakeupChannel`

With this bit the Host selects the channel on which the CC sends the Wakeup pattern. The CC ignores any attempt to change the status of this bit when not in DEFAULT_CONFIG or CONFIG state.

1 = Send wakeup pattern on channel B

0 = Send wakeup pattern on channel A

`pWakeupPattern`

Configures the number of repetitions (sequences) of the Tx wakeup symbol. Valid values are 2 to 63.

`vExternOffsetControl`

By writing to vExternOffsetControl is enabled as specified below. Should be modified only outside NIT.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

00, 01 = No external offset correction
10 = External offset correction value subtracted from calculated offset correction value
11 = External offset correction value added to calculated offset correction value.

vExternRateControl

By writing to vExternRateControl is enabled as specified below. Should be modified only outside NIT.
00, 01 = No external rate correction
10 = External rate correction value subtracted from calculated rate correction value
11 = External rate correction value added to calculated rate correction value.

Reserved[16]

Reserved Dwords for possible later use.

See Also

fcbFRSetCcConfiguration, fcbFRGetCcConfiguration, fcFRBaudRate

5.2.3 FCBFRSETCCREGISTER

This function writes a value in a given register of the selected communication controller. Not every register can be written (e.g. the registers belonging to the message buffer configuration or some interrupt settings).

```
fcError fcbFRSetCcRegister(
      fcHandle hFlexCard,
      fcCC CC,
      fcDword address,
      fcDword value
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

CC

[IN] Communication controller index

address

[IN] Address of the CC register to be written. Must be a multiple of 4 bytes, otherwise an error will be returned

value

[IN] The value to be written

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information. If the register can not be written the error code `REGISTER_NOT_WRITEABLE` is returned.

Remarks

For a register description, refer to the specification of the corresponding communication controller. Modifying one of the following registers will reset message buffers with their default settings (FIFO receive buffers). The user's message buffers configuration will not be valid anymore.
Bosch E-Ray: MHDC (0x0098) and GTUC7 (0x00B8).

	Information
	Not all register of a communication controller can be set. The base API will modify some parameters so that the operating of the FlexCard is guaranteed (e.g. interrupt settings). Access is denied to all registers which are used for message buffer configuration.

See Also

fcCC, fcbFRGetCcRegister

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

5.2.4 FCBFRGETCCREGISTER

This function reads and returns the content of a given register of the selected communication controller.

```
fcError fcbFRGetCcRegister(
      fcHandle hFlexCard,
      fcCC CC,
      fcDword address,
      fcDword* pValue
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

CC

[IN] Communication controller index

address

[IN] Address of the CC register to be read. Must be a multiple of 4 bytes, otherwise an error will be returned

pValue

[OUT] The content of the desired CC register.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information. If the register cannot be read the error code REGISTER_NOT_READABLE is returned.

Remarks

Not every register can be read. For a register description, refer to the specification of the corresponding communication controller.

See Also

fcCC, fcbFRSetCcRegister

Example

```
fcDword value = 0xFFFFFFFF;
fcDword address = 0x0B8;
fcCC eCC = fcCC1;

if (0 != address % 4) return; //address not a multiple of 4 bytes!

fcError e = fcbFRGetCcRegister(hFlexCard,eCC,address,&value);
if (0 == e)
{
      printf("Register 0x%X=0x%X", address, value);
}
```

5.2.5 FCBFRSETCCCONFIGURATIONCHI

This function configures the selected communication controller of the FlexCard with a FlexConfig compatible configuration string (CHI File). The configuration string contains the global FlexRay parameter and/or the message buffer configuration. The payload data for transmit message buffers are not set by this function. Before the configuration of the communication controller starts, all message buffers are reset to their default settings (FIFO buffer).

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
fcError fcbFRSetCcConfigurationChi(
      fcHandle hFlexCard,
      fcCC CC,
      const char* szChi
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

CC

[IN] Communication controller index

szChi

[IN] Pointer to null-terminated CHI content string (refer to the CHI string example section).
Please note: Do not use the CHI file name here, but the content of the CHI file as parameter value.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Information
Internally, the function uses the fcbFRSetCcRegister function; therefore the same restrictions as for writing registers exist.

See Also

fcCC, fcbFRSetCcRegister

Example

See fcbFRSetCcConfigurationChi

5.2.6 FCBFRSETCCCONFIGURATIONCANDB

This function configures the communication controller of the FlexCard with a CANdb compatible string. The configuration string contains the global FlexRay parameter and/or the message buffer configuration. Before the configuration of the communication controller starts, all message buffers are reset to their default settings (FIFO buffer).

```
fcError fcbFRSetCcConfigurationCANdb(
      fcHandle hFlexCard,
      fcCC CC,
      const char* szCanDb
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

CC

[IN] Communication controller index

szCanDb

[IN] Pointer to null-terminated CANdb string

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Remarks

This function is only available in the Windows FlexCard driver. The FlexCard Linux and Xenomai drivers don't support this function.

	Information
	Internally, the function uses the fcbFRSetCcRegister function; therefore the same restrictions as for writing a register exist.

5.2.7 FCBFRSETCCCONFIGURATION

This function configures the FlexRay communication controller.

```
fcError fcbFRSetCcConfiguration(
      fcHandle hFlexCard,
      fcCC CC,
      fcFRCcConfig cfg
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

CC

[IN] The FlexCard communication controller which should be configured. For FlexCard Cyclone II (SE) this parameter will always be set to fcCC1.

Cfg

[IN] The FlexRay communication controller configuration.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcCC, fcFRCcConfig, fcbFRGetCcConfiguration

Example

```
fcCC eCC = fcCC1;
fcFRCcConfig frCcConfigSet;
memset(&frCcConfigSet, 0, sizeof(fcFRCcConfig));

// SUCC1
frCcConfigSet.pKeySlotUsedForStartup = 0;
frCcConfigSet.pKeySlotUsedForSync = 0;
frCcConfigSet.gColdStartAttempts = 31;
frCcConfigSet.pAllowPassiveToActive = 0;
frCcConfigSet.pWakeupChannel = 0;
frCcConfigSet.pSingleSlotEnabled = 0;
frCcConfigSet.pAllowHaltDueToClock = 1;
frCcConfigSet.pChannelsMTS = fcChannelNone;
frCcConfigSet.pChannels = fcChannelBoth;

// SUCC2
frCcConfigSet.pdListenTimeout = 80242;
frCcConfigSet.gListenNoise = 2;

// SUCC3
frCcConfigSet.gMaxWithoutClockCorrectionPassive = 2;
frCcConfigSet.gMaxWithoutClockCorrectionFatal = 2;

// NEMC
frCcConfigSet.gNetworkManagementVectorLength = 0;
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
// PRTC1
frCcConfigSet.gdTSSTransmitter = 7;
frCcConfigSet.gdCASRxLowMax = 99;
frCcConfigSet.BaudRate = fcFRBaudRate10M;
frCcConfigSet.gdWakeupSymbolRxWindow = 301;
frCcConfigSet.pWakeupPattern = 2;

// PRTC2
frCcConfigSet.gdWakeupSymbolRxIdle = 59;
frCcConfigSet.gdWakeupSymbolRxLow = 54;
frCcConfigSet.gdWakeupSymbolTxIdle = 180;
frCcConfigSet.gdWakeupSymbolTxLow = 60;

// MHDC
frCcConfigSet.gPayloadLengthStatic = 4;
frCcConfigSet.pLatestTx = 0;

// GTUC1
frCcConfigSet.pMicroPerCycle = 40000;

// GTUC2
frCcConfigSet.gMacroPerCycle = 1000;
frCcConfigSet.gSyncNodeMax = 2;

// GTUC3
frCcConfigSet.pMicroInitialOffsetA = 0;
frCcConfigSet.pMicroInitialOffsetB = 0;
frCcConfigSet.pMacroInitialOffsetA = 2;
frCcConfigSet.pMacroInitialOffsetB = 2;

// GTUC4
frCcConfigSet.gdNIT = 40;
frCcConfigSet.gOffsetCorrectionStart = 991;

// GTUC5
frCcConfigSet.pDelayCompensationA = 0;
frCcConfigSet.pDelayCompensationB = 0;
frCcConfigSet.pClusterDriftDamping = 1;
frCcConfigSet.pDecodingCorrection = 40;

// GTUC6
frCcConfigSet.pdAcceptedStartupRange = 258;
frCcConfigSet.pdMaxDrift = 121;

// GTUC7
frCcConfigSet.gdStaticSlot = 22;
frCcConfigSet.gNumberOfStaticSlots = 43;

// GTUC8
frCcConfigSet.gdMinislot = 11;
frCcConfigSet.gNumberOfMinislots = 0;

// GTUC9
frCcConfigSet.gdActionPointOffset = 1;
frCcConfigSet.gdMinislotActionPointOffset = 5;
frCcConfigSet.gdDynamicSlotIdlePhase = 2;

// GTUC10
frCcConfigSet.pOffsetCorrectionOut = 81;
frCcConfigSet.pRateCorrectionOut = 121;

// GTUC11
frCcConfigSet.vExternOffsetControl = 0;
frCcConfigSet.vExternRateControl = 0;
frCcConfigSet.pExternOffsetCorrection = 0;
frCcConfigSet.pExternRateCorrection = 0;

// Configure the FlexRay CC
e = fcbFRSetCcConfiguration(hFlexCard, eCC, frCcConfigSet);
if (0 != e) {/* Error handling */};
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

5.2.8 FCBFRGETCCCONFIGURATION

This function reads the FlexRay communication controller configuration.

```
fcError fcbFRGetCcConfiguration (
      fcHandle hFlexCard,
      fcCC CC,
      fcFRCcConfig* pCfg
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

CC

[IN] The FlexCard communication controller which should be configured. For FlexCard Cyclone II (SE) this parameter will always be set to fcCC1.

pCfg

[OUT] Pointer to the configuration parameters of the FlexRay communication controller.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcCC, fcFRCcConfig, fcbFRSetCcConfiguration

Example

```
fcCC eCC = fcCC1;
fcFRCcConfig frCcConfigGet;

e = fcbFRGetCcConfiguration(hFlexCard, eCC, &frCcConfigGet);
if (0 != e) {/* Error handling */};
```

5.2.9 FCBFRCONFIGUREMESSAGEBUFFER

This function configures transmit, receive and FIFO message buffers of the selected communication controller. Configuring message buffers is only allowed if the communication controller is in its configuration state, `fcStateConfig`.

```
fcError fcbFRConfigureMessageBuffer(
      fcHandle hFlexCard,
      fcCC CC,
      fcDword* pBufferId,
      fcMsgBufCfg cfg
);
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

CC

[IN] Communication controller index

pBufferId

[OUT] Message buffer identifier. If the configuration was successful the message buffer identifier is greater than 0. This identifier will be required to transmit the content of the buffer (in the case of a transmit buffer).

Cfg

[IN] Message buffer configuration parameters

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Remarks

Before configuring the message buffers, it is necessary to set up the global communication parameters (cluster parameters). Internally the FlexCard uses the FIFO buffers as receive buffers, therefore we recommend using FIFO message buffers as much as possible.

See Also

fcCC, fcMsgBufCfg, fcMsgBufCfgTx, fcMsgBufCfgRx, fcMsgBufCfgFifo

Example

```
// The following code configures a transmit buffer,
// which only transmits on cycles 6,14,22,30, …

fcMsgBufCfg cfg;
memset(&cfg, 0, sizeof(fcMsgBufCfg));
cfg.Type = fcMsgBufTx;
cfg.ChannelFilter = fcChannelA;


// Repetition: each 8 cycles
// Offset: 6 (First cycle will be cycle number 6)

cfg.CycleCounterFilter = 0x8 + 0x6;

cfg.Tx.FrameId = 61;
cfg.Tx.PayloadLength = 10;
cfg.Tx.PayloadLengthMax = 127;
cfg.Tx.PayloadPreambleIndicator = 0;
cfg.Tx.SyncFrameIndicator = 0;
cfg.Tx.StartupFrameIndicator = 0;
cfg.Tx.TxAcknowledgeEnable= 0;
cfg.Tx.TransmissionMode = fcMsgBufTxSingleShot;

fcCC eCC = fcCC1;
unsigned int bufferIdx = 0;
fcError e = fcbFRConfigureMessageBuffer(hFlexCard,eCC,&bufferIdx,cfg);
```

5.2.10 FCBFRRECONFIGUREMESSAGEBUFFER

This function reconfigures transmit, receive and FIFO message buffers of the selected communication controller. A reconfiguration is only allowed for message buffers which are already configured. This function is available in all states of the CC. Not all configuration settings can be modified in monitoring state. Refer to the documentation of the message buffer structures for further details.

```
fcError fcbFRReconfigureMessageBuffer(
      fcHandle hFlexCard,
      fcCC CC,
      fcDword bufferId,
      fcMsgBufCfg cfg
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

CC

[IN] Communication controller index

bufferId

[IN] The identifier of the message buffer which should be reconfigured.

Cfg

[IN] Message buffer configuration parameters.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcCC, fcMsgBufCfg, fcMsgBufCfgTx, fcMsgBufCfgRx, fcMsgBufCfgFifo, fcbFRConfigureMessageBuffer, fcbFRGetMessageBuffer

5.2.11 FCBFRGETMESSAGEBUFFER

This function reads a specific message buffer configuration.

```
fcError fcbFRGetMessageBuffer(
      fcHandle hFlexCard,
      fcCC CC,
      fcDword bufferId,
      fcMsgBufCfg* pCfg
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

CC

[IN] Communication controller index

bufferId

[IN] The identifier of the message buffer to be read

pCfg

[OUT] The configuration parameters of the specified message buffer.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Remarks

The buffer with id 1 is always a FIFO message buffer.

See Also

fcCC, fcMsgBufCfg, fcMsgBufCfgTx, fcMsgBufCfgRx, fcMsgBufCfgFifo, fcbFRConfigureMessageBuffer

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Example

```
// Get all configured transmit message buffers of communication controller 1
fcCC eCC = fcCC1;
std::map<unsigned int, fcMsgBufCfg> Buffers;
unsigned int bufferIdx = 1; // The first valid buffer is 1
while (true)
{
      fcMsgBufCfg cfg;

      // as long no error occurs we try to get each buffer
      fcError e = fcbFRGetMessageBuffer(m_hFlexCard,eCC,bufferIdx,&cfg);
      if (0 != e) break;

      // is this a tx buffer, then add it to our list
      if (fcMsgBufTx == cfg.Type)    Buffers[bufferIdx] = cfg;

      // next buffer index
      bufferIdx++;
}
```

5.2.12 FCBFRRESETMESSAGEBUFFERS

This function resets the communication controller message buffers. After calling this function, all message buffers are configured as receive FIFO – with maximal payload (depends on the communication controller).

```
fcError fcbFRResetMessageBuffers(
      fcHandle hFlexCard,
      fcCC CC
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

CC

[IN] Communication controller index

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

5.2.13 FCBFRSETSOFTWAREACCEPTANCEFILTER

This function configures the frame ids accepted by the device driver. Only the ids which are in the filter list are forwarded to the user application, all other frames are rejected. To accept all frames set the parameters *pData* to NULL and nSize to zero or configure a single frame id of zero.

```
fcError fcbFRSetSoftwareAcceptanceFilter(
      fcHandle hFlexCard,
      fcCC CC,
      fcChannel channel,
      fcDword* pData,
      fcDword size
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

CC

[IN] Communication controller index

channel

[IN] FlexCard channel(s) concerned by the filter

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

pData

[IN] Pointer to an fcDword array containing the ids accepted by the device driver. Each element (fcDword) contains one frame identifier.

fcDword	fcDword	fcDword	fcDword
ID x	ID y	ID z	…

size

[IN] Number of ids in the array.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Example

```
// Configure the filter to get only
// - the frames from frame id 15 and 60 on cc 1, channel A
// - and the frame ids 1,2,3,6 on cc 1, channel B

fcDword idsChA[2] = {15,60};
fcDword idsChB[4] = {1,2,3,6};
fcCC eCC = fcCC1;

fcError e = fcbFRSetSoftwareAcceptanceFilter(hFlexCard,eCC,fcChannelA,idsChA,2);
//…
e = fcbFRSetSoftwareAcceptanceFilter(hFlexCard,eCC,fcChannelB,idsChB,4);
```

5.2.14 FCBFRSETHARDWAREACCEPTANCEFILTER

This function configures the FlexRay frame ids accepted by the FlexCard firmware. Only the FlexRay ids which are in the filter list are forwarded to the device driver, all other FlexRay frames are rejected. To accept all frames set the parameters *pData* to NULL and *size* to zero or configure a single frame id of zero. When using this function, receiving frames is faster than using fcbFRSetSoftwareAcceptanceFilter.

```
fcError fcbFRSetHardwareAcceptanceFilter(
      fcHandle hFlexCard,
      fcCC CC,
      fcChannel channel,
      fcDword* pData,
      fcDword size,
      bool reset
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

CC

[IN] Communication controller index.

channel

[IN] FlexCard channel(s) concerned by the filter.

pData

[IN] Pointer to an fcDword array containing the ids accepted by the device driver. Each element (fcDword) contains one frame identifier.

fcDword	fcDword	fcDword	fcDword
ID x	ID y	ID z	…

size

[IN] Number of ids in the array.

reset

[IN] Set to true to reset the filter, before configure a new filter. Set to false to add the frame identifier to the existing filter.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcCC, fcChannel, fcbReceive

5.2.15 FCBFRSETCCTIMERCONFIG

This function configures the communication controller timer interrupt. To get a notification when the communication controller timer interval elapsed, an event of type *fcNotificationTypeFRCcTimer* has to be registered by the function fcbSetEventHandleV2. Additionally the communication controller timer can be enabled / disabled by this function.

```
fcError fcbFRSetCcTimerConfig(
      fcHandle hFlexCard,
      fcCC CC,
      fcCcTimerCfg cfg,
      bool bEnable
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

CC

[IN] Communication controller index

Cfg

[IN] The communication controller timer configuration.

bEnable

[IN] Set to true to enable the cc timer, and to false to disable it.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcCC, fcbSetEventHandleV2, fcbSetEventHandleSemaphore, fcCcTimerCfg, fcbFRGetCcTimerConfig

Example

See Example fcbFRCalculateMacrotickOffset

5.2.16 FCBFRGETCCTIMERCONFIG

This function reads the communication controller timer configuration.

```
fcError fcbFRGetCcTimerConfig(
      fcHandle hFlexCard,
      fcCC CC,
      fcCcTimerCfg* pCfg
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

CC

[IN] Communication controller index

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

pCfg

[OUT] The configuration parameters of the cc timer.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcCC, fcCcTimerCfg, fcbFRSetCcTimerConfig

Example

See Example fcbFRCalculateMacrotickOffset

5.2.17 FCBFRCALCULATEMACROTICKOFFSET

This function calculates the macrotick offset for a specific cycle position in a FlexRay cycle.

```
fcError fcbFRCalculateMacrotickOffset(
      fcHandle hFlexCard,
      fcCC CC,
      fcCyclePos CyclePosition,
      fcDword SlotOrMiniSlotId,
      fcDword* pValue
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

CC

[IN] Communication controller index

CyclePosition

[IN] The cycle position of type fcCyclePos.

SlotOrMiniSlotId

[IN] This parameter is used for a cycle position of *fcCyclePosStaticSlot* and *fcCyclePosDynamicMiniSlot* to calculate the macrotick offset for a static slot or a dynamic mini slot id.

pValue

[OUT] The macrotick offset value.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcCC, fcCyclePos, fcCcTimerCfg, fcbFRSetCcTimerConfig

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Example

```
//
// Configure the CC 1 timer to get notified of the static slot id 9 start,
// Check the configuration and start the CC 1 timer
//
fcCC eCC = fcCC1;
fcCcTimerCfg ccTimerConfigSet, ccTimerConfigGet;
memset(&ccTimerConfigSet, 0, sizeof(fcCcTimerCfg));
memset(&ccTimerConfigGet, 0, sizeof(fcCcTimerCfg));

ccTimerConfigSet.CycleCounterFilter = 1;
ccTimerConfigSet.ContinuousMode = 1;
ccTimerConfigSet.MacrotickOffset = 0;

// Calculate the macrotick offset for static slot id 9
fcDword dwMTOffset = 0;
fcDword dwSlotId = 9;
fcError e = fcbFRCalculateMacrotickOffset(hFlexCard, eCC,
      fcCyclePosStaticSlot, dwSlotId, &dwMTOffset);
if (0 != e) {/* Error handling */};
else ccTimerConfigSet.MacrotickOffset = dwMTOffset;

// Configure the CC 1 timer, but don't start
e = fcbFRSetCcTimerConfig(hFlexCard, eCC, ccTimerConfigSet, false);
if (0 != e) {/* Error handling */};

// Read the configuration
e = fcbFRGetCcTimerConfig(hFlexCard, eCC, &ccTimerConfigGet);
if (0 != e) {/* Error handling */};

// Check the configured timer
if (ccTimerConfigSet.CycleCounterFilter != ccTimerConfigGet.CycleCounterFilter
      || ccTimerConfigSet.ContinuousMode != ccTimerConfigGet.ContinuousMode
      || ccTimerConfigSet.MacrotickOffset != ccTimerConfigGet.MacrotickOffset)
      {return;}

// We passed the check, now start the cc timer with this config
e = fcbFRSetCcTimerConfig(hFlexCard, eCC, ccTimerConfigSet, true);
if (0 != e) {/* Error handling */};

// Wait for the CC 1 timer event ...
```

5.3 TRANSMIT

5.3.1 FCBFRTRANSMIT

This function writes a data frame into a communication controller transmit buffer of the FlexCard. The frame should normally be transmitted in the next cycle. If the transmit acknowledgment is activated, an acknowledge packet is generated as soon as the frame has been transmitted. This function should only be called when the FlexCard is in normal active state or when all message buffer configurations have been done.

```
fcError fcbFRTransmit(
      fcHandle hFlexCard,
      fcCC CC,
      fcDword bufferId,
      fcWord * pPayload,
      fcByte payloadLength
);
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

CC

[IN] Communication controller index

bufferId

[IN] The id of the message buffer used for the transmission

pPayload

The payload data to be transmitted

payloadLength

The size of the payload data (number of 2-byte words)

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

The transmission may fail, if the buffer is currently in use (fcGetErrorCode returns MSG_BUFFER_BUSY). In that case retry later.

Remarks

The payload data has to be organized as follows: if Data0 is the first byte to transmit and Data1 the second byte to transmit, then the high byte (Bit 8 – 15) of payload[0] contains Data1, the low byte (Bit0-7) of payload[0] contains Data0, etc.

Parameter payload	payload[0] (Word 0)		payload[1] (Word 1)		…
	High byte	Low byte	High byte	Low byte	…
FlexRay payload segment	Data 1	Data 0	Data 3	Data 2	…

Example

```
fcCC eCC = fcCC1;
fcWord payload[fcPayloadMaximum];
payload[0] = 0x0001; // Update your payload data

fcError e = fcbFRTransmit(m_hFlexCard,eCC,bufferIdx,payload,payloadLength);
```

5.3.2 FCBFRTRANSMITSYMBOL

This function transmits a symbol in the symbol window segment. It can only be called if the selected communication controller is in the POC state NORMAL_ACTIVE. For a list of available symbols to be transmitted, see the enumeration fcSymbolType.

```
fcError fcbFRTransmitSymbol(
      fcHandle hFlexCard,
      fcCC CC,
      fcSymbolType type
);
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

CC

[IN] Communication controller index

type

[IN] Type of symbol to transmit

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

6 OPTIONAL CAN API

The following section describes the data structures and features used for CAN functionality. To use these functions the FlexCard must be licensed for CAN.

	Information
	All enumerations, structures and function in this chapter are initially supported by FlexCard Windows API version S4V0-F and FlexCard Linux/Xenomai API version S4V2-F.

6.1 BASIC CAN WORKFLOW

Figure 11 shows a typical CAN workflow. Please note that the message buffers may be reconfiguered during monitoring, but the CAN configuration may only be changed when monitoring is not running.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

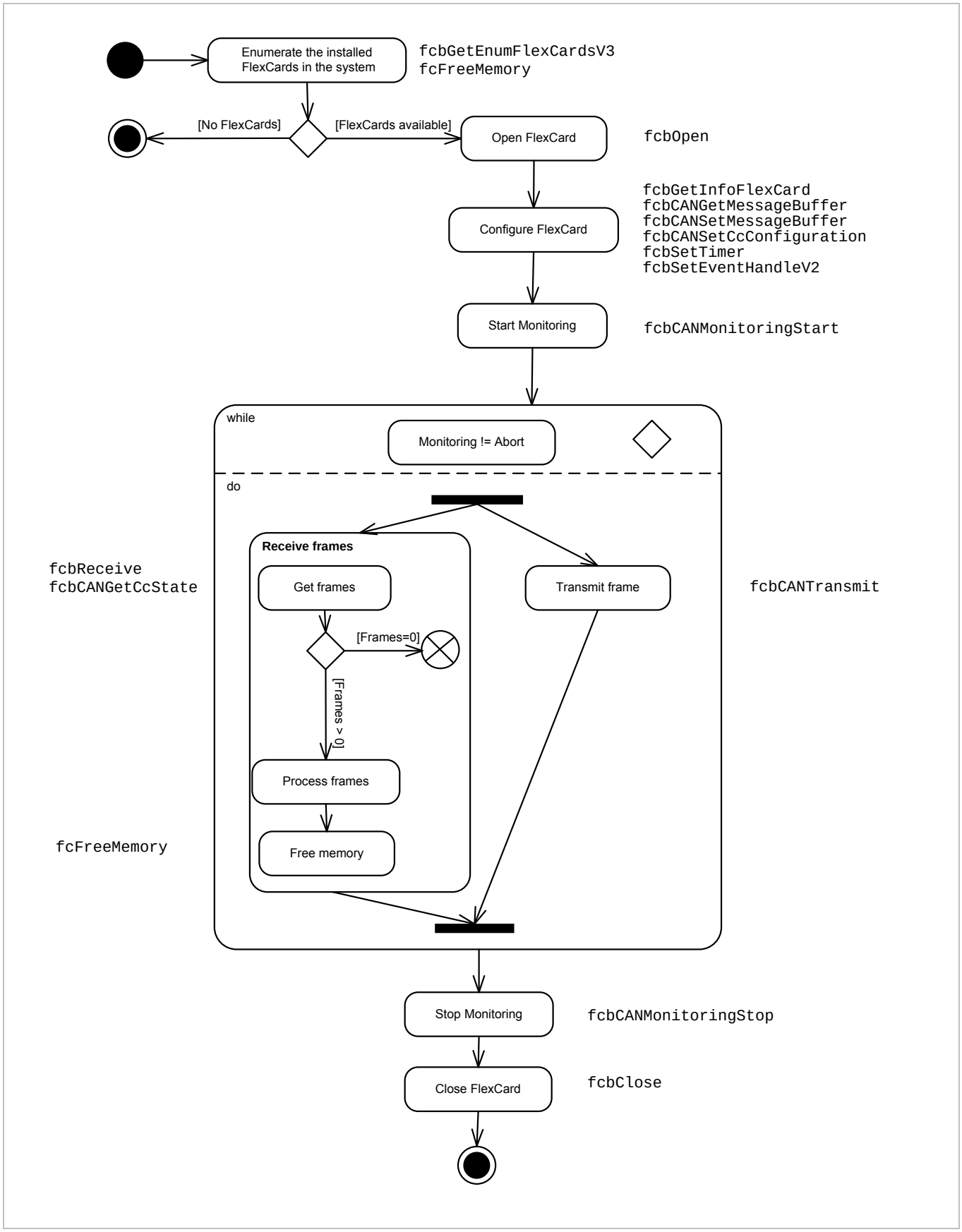


Figure 11: Typical CAN function workflow

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

6.2 INITIALIZATION

6.2.1 ENUMERATIONS

6.2.1.1 FCCANCCSTATE

This enumeration defines the CAN communication controller states. For more details about CAN communication controller states, please refer to the CAN Protocol Specification.

```
Typedef enum fcCANCcState
{
      fcCANCcStateUnknown = 0,
      fcCANCcStateConfig,
      fcCANCcStateNormalActive,
      fcCANCcStateWarning,
      fcCANCcStateErrorPassive,
      fcCANCcStateBusOff,
}fcCANCcState;
```

Members

fcCANCcStateUnknown
: Communication controller state is unknown.

fcCANCcStateConfig
: Communication controller is in configuration state.

fcCANCcStateNormalActive
: Communication controller is in normal active state.

fcCANCcStateWarning
: Communication controller is in error warning state. At least one of the error counters has reached the error warning limit of 96.

fcCANCcStateErrorPassive
: Communication controller is in error passive state. No CAN messages can be sent anymore except CAN passive errors.

fcCANCcStateBusOff
: Communication controller is in bus off state. No CAN messages can be sent anymore.

See Also

fcbCANGetCcState, fcbCANMonitoringStart

6.2.1.2 FCCANMONITORINGMODE

This enumeration defines the different modes available, used to monitor a CAN cluster.

```
Typedef enum fcCANMonitoringMode
{
      fcCANMonitoringNormal = 0,
      fcCANMonitoringActive = fcCANMonitoringNormal,
      fcCANMonitoringSilent = 1,
      fcCANMonitoringPassive = fcCANMonitoringSilent,
}fcCANMonitoringMode;
```

Members

fcCANMonitoringNormal

fcCANMonitoringActive
: The FlexCard will switch from configuration to normal active state as soon as possible. In normal active state CAN frames can be received and transmitted.

fcCANMonitoringSilent

fcCANMonitoringPassive
: The FlexCard will switch from configuration to normal passive state as soon as possible. In normal passive state CAN frames can be received only.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

See Also

fcbCANMonitoringStart

6.2.2 FCBCANMONITORINGSTART

This function is used to start the monitoring of a CAN bus. Once called, the function changes the communication controller state from configuration state to normal active state. The current communication controller state can be read using the function fcbCANGetCcState. If the FlexCard is started the function fcbCANGetCcState will return the value *fcCANCcStateNormalActive.*

```
fcError fcbCANMonitoringStart(
      fcHandle hFlexCard,
      fcCC CC,
      bool resetTimestamps,
      fcCANMonitoringMode mode
);
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

CC

[IN] Index of the CAN communication controller.

restartTimestamps

[IN] Set this parameter to false to restart the measurement without resetting the FlexCard timestamp. Set it to true to start the measurement from the beginning. The timestamps have micro second resolution.

Mode

[IN] The monitoring mode. See fcCANMonitoringMode for details which monitoring mode is supported.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Remarks

After the monitoring has started, the user should check if the integration in the cluster was successful, fcbCANGetCcState should return the state *fcCANCcStateNormalActive*.

	Information
	After the monitoring has successfully started, the receive process has to be started as soon as possible to avoid an overflow (error packet fcErrFlexcardOverflow is received). Once an overflow occurred, no more packets can be received. The monitoring has to be stopped and started again.

See Also

fcbCANMonitoringStop, fcbCANGetCcState, fcCANMonitoringMode

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Example

```
// Precondition: valid flexcard handle exists and the flexcard is
// already configured.

fcCC eCC = fcCC1;
fcError e = fcbCANMonitoringStart(hFlexCard,eCC,true,fcCANMonitoringNormal);
if (0 == e)
{
      bool active = false;
      bool timeout = false;
      DWORD maxTime = ::GetTickCount() + 2000;
      fcCANCcState state = fcCANCcStateUnknown;

      // Check if the FlexCard is in CAN normal active state
      do
      {
            fcbCANGetCcState(hFlexCard, eCC, &state);
            active= (state == fcCANCcStateNormalActive);
            timeout     = ::GetTickCount() >= maxTime;

      } while ( ! active && ! timeout);

      if (active)
      {
            // Start your receive thread/routine
            //   ...
      }
      else
      {
            // if we timed out, we stop the monitoring
            fcbCANMonitoringStop(hFlexCard);
      }
}
else
{
      // error handling …
}
```

6.2.3 FCBCANMONITORINGSTOP

This function stops the CAN bus monitoring of the selected communication controller. The communication controller is set back in its configuration state.

```
fcError fcbCANMonitoringStop(
      fcHandle hFlexCard,
      fcCC CC
)
```

Parameters

hFlexCard

[IN] Handle to FlexCard.

CC

[IN] Index of the CAN communication controller.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcbCANMonitoringStart

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

6.2.4 FCBCANGETCCSTATE

This function returns the current CAN communication controller state. For a description of possible states, refer to the enumeration fcCANCcState.

```
fcError fcbCANGetCcState(
      fcHandle hFlexCard,
      fcCC CC,
      fcCANCcState* pState
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

CC

[IN] communication controller index.

pState

[OUT] Current CAN communication controller state.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See

fcbCANMonitoringStart, fcbCANMonitoringStop

Example

For an example, see fcbCANMonitoringStart.

6.3 CONFIGURATION

6.3.1 ENUMERATIONS

6.3.1.1 FCCANBUFCFGTYPE

For the transmission and reception of CAN frames the communication controller provides different types of message buffers. For sending and receiving error frames or for receiving trigger packets, no message buffer is needed. Each message buffer can be assigned with one of the following specific types.

```
Typedef enum fcCANBufCfgType
{
      fcCANBufCfgTypeNone = 0,
      fcCANBufCfgTypeCommon,
      fcCANBufCfgTypeRxAll,
      fcCANBufCfgTypeRx,
      fcCANBufCfgTypeTx,
      fcCANBufCfgTypeRemoteRx,
      fcCANBufCfgTypeRemoteTx,
}fcCANBufCfgType;
```

Members

fcCANBufCfgTypeNone

The message buffer is not used. It can be used to reset a message buffer.

fcCANBufCfgTypeCommon

The message buffer is reserved for internal use only. (No support.)

fcCANBufCfgTypeRxAll

The message buffer is used for receiving all incoming CAN data and remote frames.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

fcCANBufCfgTypeRx
The message buffer is used as a receive buffer for either a specific message or a set of messages.
fcCANBufCfgTypeTx
The message buffer is used as a transmit buffer for a specific CAN message ID.
fcCANBufCfgTypeRemoteRx
The message buffer is used as a remote receive buffer. It is used for sending a remote request and receiving the according replying message.
fcCANBufCfgTypeRemoteTx
The message buffer is used as a remote transmission buffer. It can be transmitted automatically when a remote request is received.

See Also
fcCANBufCfg

6.3.1.2 FCCANBUFCFGRXALLCONDITION

This enumeration defines the acceptance conditions of an fcCANBufCfgRxAll buffer. To accept all incoming messages you need to binary OR all of them.

```
Typedef enum fcCANBufCfgRxAllCondition
{
      fcCANRxAllNone           = 0x0,
      fcCANRxAllIDStandard     = 0x1,
      fcCANRxAllIDExtended     = 0x2,
      fcCANRxAllFrameData      = 0x4,
      fcCANRxAllFrameRemote    = 0x8,
}fcCANBufCfgRxAllCondition;
```

Members
fcCANRxAllNone
Accept no frames.
fcCANRxAllIDStandard
Accept CAN frames with standard identifiers.
fcCANRxAllIDExtended
Accept CAN frames with extended identifiers.
fcCANRxAllFrameData
Accept all CAN data frames.
fcCANRxAllFrameRemote
Accept all CAN remote frames.

See Also
fcCANBufCfgRxAll

6.3.2 STRUCTURES

6.3.2.1 FCCANBUFCFGRXALL

This structure specifies a special CAN receive message buffer. This buffer type is used to receive all frames of the specified conditions.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef struct fcCANBufCfgRxAll
{
      fcDword Condition;
      fcDword Reserved[2];
}fcCANBufCfgRxAll;
```

Members

Condition

The acceptance condition for this buffer, which can be OR-ed.
At least one id condition and one frame condition must be used to receive frames.

Reserved

Reserved for future use.

See Also

fcCANBufCfg, fcCANBufCfgRxAllCondition

6.3.2.2 FCCANBUFCFGRX

This structure specifies the configuration of a CAN receive message buffer. This buffer type is used to receive a CAN message with a specific CAN ID only or a range of CAN IDs.

```
Typedef struct fcCANBufCfgRx
{
      fcDword ID;
      fcDword MaskID;
      fcDword enableIDExtended :1;
      fcDword enableMask       :1;
      fcDword Reserved[2];
}fcCANBufCfgRx;
```

Members

ID

Defines the CAN identifier to be received in this message buffer.
Valid values for a standard CAN Id range from 0x0 – 0x7FF.
Valid values for an extended CAN Id range from 0x0 – 0x1FFFFFFF.

MaskID

The bit mask. The corresponding identifier bits are used for acceptance filtering. If it is equal 0, all IDs will be accepted.

enableIDExtended

If set to 1 the message buffer is defined for extended CAN identifiers only. If set to 0 the message buffer is defined for standard CAN identifiers. It's not possible to receive both versions in one message buffer.

enableMask

Set this flag to 1 to enable the acceptance mask.

Reserved

Reserved for future use.

See Also

fcCANBufCfg

6.3.2.3 FCCANBUFCFGTX

This structure specifies the configuration of a CAN transmit message buffer. This buffer type is used to transmit a CAN message with a specific CAN ID only.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef struct fcCANBufCfgTx
{
      fcDword ID;
      fcByte  Data[8];
      fcDword DLC : 4;
      fcDword enableIDExtended : 1;
      fcDword enableTxAcknowledge : 1;
      fcDword enableTxRequest : 1;
      fcDword newData : 1;
      fcDword Reserved[2];
}fcCANBufCfgTx;
```

Members

ID

Defines the CAN identifier to be received in this message buffer.
Valid values for a standard CAN Id range from 0x0 – 0x7FF.
Valid values for an extended CAN Id range from 0x0 – 0x1FFFFFFF.

Data

Defines the data for transmission. All of the 8 data bytes can be set. The corresponding DLC parameter is used to define the number of bytes to transmit.

DLC

Defines the data length (in bytes) to be transmitted.

enableIDExtended

If set to 1 the message buffer is defined for extended CAN identifiers only. If set to 0 the message buffer is defined for standard CAN identifiers. It's not possible to receive both versions in one message buffer.

enableTxAcknowledge

Set this flag to 1 to enable the transmit acknowledge. The FlexCard generates a CAN packet with a direction flag = 1 (Tx), if the data is transmitted successfully.

enableTxRequest

Set this flag to 1 to indicate that the message is requested to be sent as soon as the communication controller is in state 'normal active'.

newData

Set this flag to 1 to update the data of the message buffer. Set to 0 if no new data shall be updated.

Reserved

Reserved for future use

See Also

fcCANBufCfg, fcCANPacket

6.3.2.4 FCCANBUFCFGREMOTERX

This structure specifies a CAN remote receive message buffer. This buffer type is used to send a CAN remote message to request a CAN message with the same CAN identifier. This will be received into the message buffer.

```
Typedef struct fcCANBufCfgRemoteRx
{
      fcDword ID;
      fcDword DLC                    :4;
      fcDword enableIDExtended       :1;
      fcDword enableTxAcknowledge    :1;
      fcDword enableTxRequest        :1;
      fcDword Reserved[2];
}fcCANBufCfgRemoteRx;
```

Members

ID

Defines the CAN identifier to be received in this message buffer.
Valid values for a standard CAN Id range from 0x0 – 0x7FF.
Valid values for an extended CAN Id range from 0x0 – 0x1FFFFFFF.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

DLC

Defines the data length (in bytes) to be transmitted.

enableIDExtended

If set to 1 the message buffer is defined for extended CAN identifiers only. If set to 0 the message buffer is defined for standard CAN identifiers. It's not possible to receive both versions in one message buffer.

enableTxAcknowledge

Set this flag to 1 to enable the transmit acknowledge. The FlexCard generates a CAN packet (RemoteTx) if the data are transmitted successfully.

enableTxRequest

Set this flag to 1 to indicate that the message is requested to be sent as soon as the communication controller is in state 'normal active'.

Reserved

Reserved for future use.

See Also

fcCANBufCfg, fcCANPacket

6.3.2.5 FCCANBUFCFGREMOTETX

This structure specifies a CAN remote transmit message buffer. This buffer type is used to transmit a CAN message when this Id is requested by a corresponding CAN remote frame.

```
Typedef struct fcCANBufCfgRemoteTx
{
      fcDword ID;
      fcByte  Data [8];
      fcDword DLC                 :4;
      fcDword enableIDExtended    :1;
      fcDword enableTxAcknowledge :1;
      fcDword enableTxRequest     :1;
      fcDword enableAutoResponse  :1;
      fcDword newData             :1;
      fcDword Reserved[2];
}fcCANBufCfgRemoteTx;
```

Members

ID

Defines the CAN identifier to be responded with the same id.
Valid values for a standard CAN Id range from 0x0 – 0x7FF.
Valid values for an extended CAN Id range from 0x0 – 0x1FFFFFFF.

Data

Defines the data for transmission. All of the 8 data bytes can be set. The corresponding DLC parameter is used to define the number of bytes to transmit.

DLC

Defines the data length (in bytes) to be transmitted.

enableIDExtended

If set to 1 the message buffer is defined for extended CAN identifiers only. If set to 0 the message buffer is defined for standard CAN identifiers. It's not possible to receive both versions in one message buffer.

enableTxAcknowledge

Set this flag to 1 to enable the transmit acknowledge. The FlexCard generates a CAN packet (RemoteTx) if the data are transmitted successfully and the parameter *enableAutoResponse* is set to 1 too.

enableTxRequest

Set this flag to 1 to indicate that the message is requested to be sent as soon as the communication controller is in state 'normal active'.

enableAutoResponse

Set this flag to 1 to enable the buffer to transmit a frame as soon as a corresponding CAN remote frame is received.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

newData
Set this flag to 1 to update the data of the message buffer. Set to 0 if no new data shall be updated.

Reserved
Reserved for future use.

See Also
fcCANBufCfg, fcCANPacket

6.3.2.6 FCCANBUFCFG

This structure describes the configuration of a CAN message buffer.

```
Typedef struct fcCANBufCfg
{
      fcCANBufCfgType Type;
      union
      {
            fcCANBufCfgCommon Common;
            fcCANBufCfgRxAll RxAll;
            fcCANBufCfgRx Rx;
            fcCANBufCfgTx Tx;
            fcCANBufCfgRemoteRx RemoteRx;
            fcCANBufCfgRemoteTx RemoteTx;
      };
}fcCANBufCfg;
```

Members

Type
Defines the CAN message buffer type. Using type fcCANBufCfgTypeNone disables/resets the message buffer.

Common
Used for internal purposes. (No support).

RxAll
Receive all buffer configuration.

Rx
Receive buffer configuration.

Tx
Transmit buffer configuration.

RemoteRx
Remote receive buffer configuration.

RemoteTx
Remote transmit buffer configuration.

See Also
fcCANBufCfgType, fcCANBufCfgRxAll, fcCANBufCfgRx, fcCANBufCfgTx, fcCANBufCfgRemoteRx, fcCANBufCfgRemoteTx, fcbCANSetMessageBuffer, fcbCANGetMessageBuffer

6.3.2.7 FCCANCCCONFIG

This structure describes the configuration of a CAN communication controller. Within this function all message buffers will be reset.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef struct fcCANCcConfig
{
      fcWord BaudRatePrescaler;
      fcWord SynchronizationJumpWidth;
      fcWord TimeSegment1;
      fcWord TimeSegment2;
      fcDword enableAutomaticRetransmission :1;
      fcDword Reserved[6];
}fcCANCcConfig;
```

Members

BaudRatePrescaler

Defines the baud rate prescaler (BRP). Valid values are from 0 to 1023.

SynchronizationJumpWidth

Defines the synchronization jump width (SJW). Valid values are from 0 to 3 and must not be larger than TSEG1 and TSEG2.

TimeSegment1

Defines the time segment 1 (TSEG1). Valid values are from 0 to 15.

TimeSegment2

Defines the time segment 2 (TSEG2). Valid values are from 0 to 7.

enableAutomaticRetransmission

Set this flag to 1 to enable automatic retransmission. If the CAN communication controller has lost the arbitration or if an error occurred during the transmission, the message will be retransmitted as soon as the CAN bus is free again.

Reserved

Reserved for future use.

See Also

fcbCANSetCcConfiguration

Remarks

The baud rate and the sample point calculation of the CAN communication controller depends on *BaudRatePrescaler*, *SynchronizationJumpWidth*, *TimeSegment1* and *TimeSegment2*.

Baud rate [baud] = 16 * 10^6 [Hz] / ((3 + TSEG1 + TSEG2) * (1 + BRP))

Sample point [%] = 100 * (2 + TSEG1) / (3 + TSEG1 + TSEG2)

	Information
	*Eberspächer Electronics* delivers a calculation tool “CANBaudRateCalculator”, which can be found in the tools directory in the FlexCard program menu.

6.3.3 FCBCANSETCCCONFIGURATION

This function configures the CAN communication controller. This function cannot be called during monitoring. Before the configuration of the CAN CC starts, all CAN message buffers are reset.

```
fcError fcbCANSetCcConfiguration(
      fcHandle hFlexCard,
      fcCC CC,
      fcCANCcConfig cfg
);
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

CC

[IN] CAN communication controller identifier.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Cfg

[IN] CAN communication controller configuration parameters.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcbCANGetMessageBuffer, fcCANCcConfig

6.3.4 FCBCANSETMESSAGEBUFFER

This function configures the message buffers of the CAN communication controller. Configuring message buffers is allowed in all communication controller states.

```
fcError fcbCANSetMessageBuffer(
      fcHandle hFlexCard,
      fcCC CC,
      fcDword bufferNumber,
      fcCANBufCfg cfg,
      bool ignoreTxRqstLock
);
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

CC

[IN] CAN communication controller identifier.

bufferNumber

[IN] Identifier of the message buffer to be configured.

Cfg

[IN] Message buffer configuration parameters.

ignoreTxRqstLock

Set this flag to 1 if you want to force a reconfiguration of this buffer although the previous message in this buffer was (probably) not transmitted yet.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcCANBufCfg

6.3.5 FCBCANGETMESSAGEBUFFER

This function reads a specific CAN message buffer configuration.

```
fcError fcbCANGetMessageBuffer(
      fcHandle hFlexCard,
      fcCC CC,
      fcDword bufferNumber,
      fcCANBufCfg* pCfg
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

CC

[IN] CAN communication controller identifier.

bufferNumber

[IN] Identifier of the message buffer to be read.

Cfg

[OUT] The configuration parameters of the specified message buffer.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcbCANSetMessageBuffer, fcCANBufCfg

Example

```
// Get all configured transmit message buffers
std::map<unsigned int, fcCANBufCfg > Buffers;
unsigned int bufferNr = 1; // The first valid buffer is 1
while (true)
{
      fcCANBufCfg cfg;

      // as long as no error occurs we try to get each buffer
      fcError e = fcbCANGetMessageBuffer (m_hFlexCard,fcCC1,bufferNr,&cfg);
      if (0 != e) break;

      // is this a tx buffer, then add it to our list
      if (fcCANBufCfgTypeTx == cfg.Type)
            Buffers[bufferNr] = cfg;

      // next buffer number
      bufferNr ++;
}
```

6.4 TRANSMIT

6.4.1 FCBCANTRANSMIT

This function writes the data bytes in a CAN communication controller transmit buffer of the FlexCard. The transmitted data bytes depend on the meassage buffer configuration. The CAN message should normally be transmitted as soon as possible. In case the transmission of any number of message buffers may be requested at the same time, they are transmitted subsequently according to their priority (The message object numbers are configurable from 1 up to 128, the lower is the message number, the higher is the priority). If the transmit acknowledgment is activated, a CAN packet with a direction flag = 1 (Tx) is generated as soon as the message has been transmitted. This function should only be called when the FlexCard is in normal active state or when all message buffer configurations have been done.

The transmission may fail, if the buffer is already locked for transmission (fcGetErrorCode returns MSG_BUF_LOCKED_FOR_TRANSMISSION). In that case retry later or set the parameter *ignoreTxRqstLock* to true.

```
fcError fcbCANTransmit(
      fcHandle hFlexCard,
      fcCC CC,
      fcDword bufferNumber,
      fcByte data[8],
      bool transmitNewData,
      bool ignoreTxRqstLock
);
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Parameters

hFlexCard

[IN] Handle to a FlexCard

CC

[IN] CAN communication controller identifier.

bufferNumber

[IN] The number of the message buffer used for the transmission.

Data

[IN] The data to be transmitted. The configured DLC in the message buffer determinates the size of bytes which will be copied in the transmit buffer.

transmitNewData

[IN] Set to true to update the data of the message buffer. Set to false if the previous data shall be sent again.

ignoreTxRqstLock

Set this flag to 1 if you want to force a reconfiguration of this buffer although the previous message was not transmitted yet.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Example

```
fcByte data[8];
for (int i=0; i<8;i++)
      data[i]=0xA;
// Transmit new data
fcError e = fcbCANTransmit(m_hFlexCard,fcCC1,bufferNumber,data,true,true);
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

7 ADDITIONAL CYCLONE II (SE) AND PMC (II) API

	Information
	All enumerations, structures and functions in this chapter are initially supported for FlexCard Cyclone II (SE) devices by: ➢ FlexCard Windows API version S3V0-F. ➢ FlexCard Linux API version S2V0-F. ➢ FlexCard Xenomai API version S4V2-F.

	Information
	This additional API is also initially supported for: ➢ FlexCard PMC devices with only one FlexRay communication controller and the FlexCard API version S4V2-F. ➢ FlexCard PMC II devices with only one FlexRay communication controller and the FlexCard API version S5V1-F.

7.1 SELF SYNCHRONIZATION

To also be able to test FlexRay nodes that don't take part actively in the synchronization process of a FlexRay network, the FlexCard provides the possibility to generate a second startup/synchronization frame. Thus, the FlexCard synchronizes the FlexRay network independently. Self synchronization runs on the first communication controller.

7.1.1 CONFIGURATION

7.1.1.1 FCBCONFIGUREMESSAGEBUFFERSELFSYNCHRONIZATION

This function configures up to 2 additional startup/synchronization message buffers. Configuring message buffers is only allowed if the communication controller is in its configuration state, `fcStateConfig`. The message buffer needs to be defined as fcMsgBufCfgTx. The SyncFrameIndicator and StartupFrameIndicator need to be set, while CycleCounterFilter must be set to 0.

```
fcError fcbConfigureMessageBufferSelfSynchronization(
      fcHandle hFlexCard,
      fcDword* bufferId,
      fcMsgBufCfg cfg
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

bufferId

[OUT] Message buffer identifier. If the configuration was successful the message buffer identifier is greater than 0. This identifier will be required to transmit the content of the buffer.

Cfg

[IN] Message buffer configuration parameters

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Remarks

Only one additional start-up/synchronization frame can be defined. Therefore only 2 additional message buffers can be configured. Before configuring the message buffers, it is necessary to set up the global communication parameters (cluster parameters). Loading a new CHI file will reset the additional start-up/synchronization frames.

See Also

fcMsgBufCfg, fcMsgBufCfgTx, fcbReconfigureMessageBufferSelfSynchronization, fcbGetCcMessageBufferSelfSynchronization, fcbResetCcMessageBuffersSelfSynchronization

Example

```
// The following code configures a self startup/synch transmit buffer
fcMsgBufCfg cfg;
memset(&cfg, 0, sizeof(fcMsgBufCfg));

cfg.Type = fcMsgBufTx;
cfg.ChannelFilter = fcChannelA;
cfg.CycleCounterFilter = 0x0;       // sync frames must appear in every cycle

cfg.Tx.FrameId = 3;                 // unused slotId of static segment
cfg.Tx.PayloadLength = 2;
cfg.Tx.PayloadLengthMax = 127;
cfg.Tx.PayloadPreambleIndicator = 0;
cfg.Tx.SyncFrameIndicator = 1;      // mandatory to be set to 1
cfg.Tx.StartupFrameIndicator = 1;   // mandatory to be set to 1
cfg.Tx.TxAcknowledgeEnable= 1;
cfg.Tx.TransmissionMode = fcMsgBufTxSingleShot;

unsigned int bufIdx = 0;
fcError e=fcbConfigureMessageBufferSelfSynchronization(hFlexCard,&bufIdx,cfg);
```

7.1.1.2 FCBRECONFIGUREMESSAGEBUFFERSELFSYNCHRONIZATION

This function reconfigures the additional transmit message buffers for self start-up/synchronization. A reconfiguration is only allowed for message buffers which are already configured and if the communication controller is in its configuration state, `fcStateConfig`. The message buffer needs to be defined for a start-up/synchronization transmit frame. Therefore it is mandatory to set the SyncFrameIndicator and StartupFrameIndicator to 1 and the CycleCounterFilter to 0.

```
fcError fcbReconfigureMessageBufferSelfSynchronization(
      fcHandle hFlexCard,
      fcDword bufferId,
      fcMsgBufCfg cfg
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

bufferId

[IN] The identifier of the message buffer which should be reconfigured.

Cfg

[IN] Message buffer configuration parameters.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcMsgBufCfg, fcMsgBufCfgTx, fcbConfigureMessageBufferSelfSynchronization, fcbGetCcMessageBufferSelfSynchronization, fcbResetCcMessageBuffersSelfSynchronization

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

7.1.1.3 FCBREINITIALIZECCMESSAGEBUFFERSELFSYNCHRONIZATION

This function re-initializes the message buffer configuration of the self-startup synchronization communication controller. After calling this function the communication controller does not send old payload data. Re-initialization of message buffers is only allowed if the communication controller is in configuration state.

```
fcError fcbReinitializeCcMessageBufferSelfSynchronization(
      fcHandle hFlexCard
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information

7.1.1.4 FCBGETCCMESSAGEBUFFERSELFSYNCHRONIZATION

This function reads a specific message buffer configuration of the additional message buffers for self startup/synchronization.

```
fcError fcbGetCcMessageBufferSelfSynchronization(
      fcHandle hFlexCard,
      fcDword bufferId,
      fcMsgBufCfg* cfg
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

bufferId

[IN] The identifier of the additional start-up/sync message buffer to be read

cfg

[OUT] The configuration parameters of the specified message buffer.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcMsgBufCfg, fcMsgBufCfgTx, fcbConfigureMessageBufferSelfSynchronization, fcbReconfigureMessageBufferSelfSynchronization, fcbResetCcMessageBuffersSelfSynchronization

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Example

```
// Get all configured additional startup/synchronization transmit
// message buffers
std::map<unsigned int, fcMsgBufCfg> Buffers;

// valid buffer indexes are only 1 and 2
for(unsigned int bufIdx = 1;bufIdx <=2; bufIdx++)
{
  fcMsgBufCfg cfg;

  // as long no error occurs we try to get each buffer
  fcError e=fcbGetCcMessageBufferSelfSynchronization(m_hFlexCard,bufIdx,&cfg);
  if (0 != e)
      continue;

  //and add it to our list
  Buffers[bufIdx] = cfg;
}
```

7.1.1.5 FCBRESETCCMESSAGEBUFFERSSELFSYNCHRONIZATION

This function resets the additional startup/synchronization message buffers.

```
fcError fcbResetCcMessageBufferSelfSynchronization(
      fcHandle hFlexCard
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

7.1.2 TRANSMIT

7.1.2.1 FCBTRANSMITSELFSYNCHRONIZATION

This function writes a data frame into a self start-up/synchronization transmit buffer of the FlexCard. The frame should normally be transmitted in the next cycle. If the transmit acknowledgment is activated, an acknowledge packet is generated as soon as the frame has been transmitted. This function should only be called when the FlexCard is in normal active state or when all message buffer configurations have been done.

```
fcError fcbTransmitSelfSynchronization(
      fcHandle hFlexCard,
      fcDword bufferId,
      fcWord payload[],
      fcByte payloadLength
);
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

bufferId

[IN] The id of the additional startup/synchronization message buffer used for the transmission

payload

The payload data to be transmitted

payloadLength

The size of the payload data (number of 2-byte words)

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

The transmission may fail, if the buffer is currently in use (fcGetErrorCode returns `MSG_BUFFER_BUSY`). In that case retry later.

Remarks

The payload data has to be organized as follows: if Data0 is the first byte to transmit and Data1 the second byte to transmit, then the high byte (Bit 8 – 15) of payload[0] contains Data1, the low byte (Bit0-7) of payload[0] contains Data0, etc.

Parameter `payload`	`payload[0]` (Word 0)		`payload[1]` (Word 1)		.
	High byte	Low byte	High byte	Low byte	.
FlexRay payload segment	Data 1	Data 0	Data 3	Data 2	.

Example

```
fcWord payload[fcPayloadMaximum];
payload[0] = 0x0001; // Update your payload data

fcError e = fcbTransmitSelfSynchronization(m_hFlexCard,bufferIdx,
            payload,payloadLength);
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

8 ADDITIONAL PMC (II) CARD API

There are some functional hardware and software differences between FlexCard Cyclone II (SE) and FlexCard PMC (II) which demand additional functions or enumerations. The differences are listed in the table below:

FlexCard Cyclone II (SE)	FlexCard PMC	FlexCard PMC II
1 CC for FlexRay and 2 CCs for CAN	2 CCs for FlexRay or 1 CC for FlexRay and 2 CCs for CAN	Variable interface configurations for FlexRay and CAN max. 4 FlexRay CCs or max. 8 CAN CCs with FlexTiny II possible.
1 IN trigger line, 1 OUT trigger line	2 trigger lines which can be defined as IN or OUT	
There are no bus terminations available on the cards. External bus termination needed.	FlexRay termination (90 Ohm) and CAN termination (120 Ohm) can be switched on by a software access. Therefore the dip switches for bus channel 3 and 4 must be configured correct.	FlexRay termination (90 Ohm) and CAN termination (120 Ohm) can be switched on by a software access for all bus channels.

8.1 ENUMERATIONS

8.1.1 FCBUSCHANNEL

This enumeration defines the bus channels available on the card.

```
Typedef enum fcBusChannel
{
      fcBusChannel1 = 1,
      fcBusChannel2 = 2,
      fcBusChannel3 = 3,
      fcBusChannel4 = 4,
      fcBusChannel5 = 5,
      fcBusChannel6 = 6,
      fcBusChannel7 = 7,
      fcBusChannel8 = 8,
} fcBusChannel;
```

Members

fcBusChannel1
Identifies bus channel 1.

fcBusChannel2
Identifies bus channel 2.

fcBusChannel3
Identifies bus channel 3.

fcBusChannel4
Identifies bus channel 4.

fcBusChannel5
Identifies bus channel 5.

fcBusChannel6
Identifies bus channel 6.

fcBusChannel7
Identifies bus channel 7.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

fcBusChannel8
Identifies bus channel 8.

See Also
fcbGetBusTermination, fcbSetBusTermination

8.2 FCBSETBUSTERMINATION

This function sets the bus termination for a bus channel.

```
fcError fcbSetBusTermination(
      fcHandle hFlexCard,
      fcBusChannel BusChannel,
      fcBusType BusType,
      bool bTermination
)
```

Parameters
hFlexCard
[IN] Handle to a FlexCard
BusChannel
[IN] The bus channel describes the channel at which the termination should be switched on or off.
BusType
[IN] The bus type describes which bus protocol/transceiver is used for the channel. Different bus protocols/transceivers demand different bus terminations.
bTermination
[IN] This parameter enables or disables the bus termination

Return values
If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also
fcbGetBusTermination

Example
```
fcBusChannel busChannel = fcBusChannel3;
fcBusType busType = fcBusTypeFlexRay;
bool bTerm = true; // enable termination

// set FlexRay termination on bus channel 3
fcError e = fcbSetBusTermination(m_hFlexCard,busChannel,busType,bTerm);
```

Remarks
The termination will not be switched off by the driver automatically if the application closes the device or the driver will be unloaded. So the bus will not be disturbed by termination loss in case the user application fails.

The bus channels for a FlexCard PMC (II) are named channel1 to channel 8 as shown in the figures below. Please note that the bus type (FlexRay or CAN) of channel 3 and 4 for a FlexCard PMC/PCI need to be set by dip switches as described in the FlexCard PMC (II) instructions for use.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

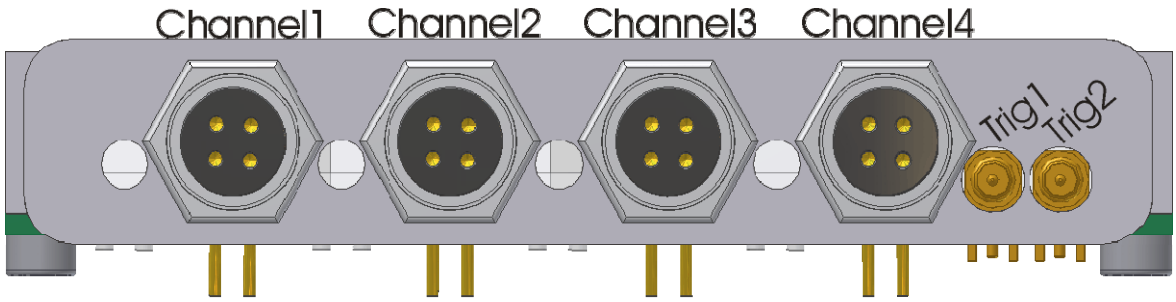


Figure 12: FlexCard PMC front panel

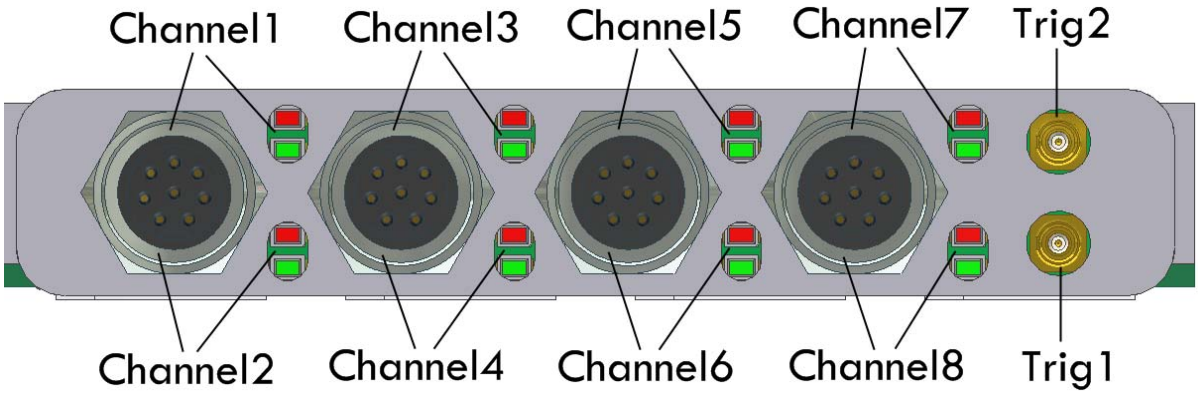


Figure 13: FlexCard PMC II front panel

8.3 FCBGETBUSTERMINATION

This function reads the bus termination configuration for a bus channel.

```
fcError fcbGetBusTermination(
      fcHandle hFlexCard,
      fcBusChannel BusChannel,
      fcBusType BusType,
      bool * pbTermination
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

BusChannel

[IN] The bus channel of the termination.

BusType

[IN] The bus type describes which bus termination type has be to be checked. Currently only FlexRay bus terminations are available.

pbTermination

[OUT] This parameter value describes whether the bus termination is enabled or disabled.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcbSetBusTermination

3-0009-0S01-D03_API Documentation_D1V12-F.doc

8.4 FCTRIGGERCONFIGURATIONEX

Please note that FlexCard PMC (II) trigger lines are not hard defined as IN or OUT trigger lines. Therefore you always need to set a valid value for the parameter *TriggerLineToConfigure* Valid values are range from 1 to 4 depending of the FlexCard device type. The trigger conditions of the FlexCard PMC (II) are defined in the enumeration *fcTriggerConditionPMC*. The **conditions cannot be OR-ed**. If you do it nevertheless, none of the conditions are set and an error message is returned.

The conditions *fcTriggerPMCOutOnErrorDetected*, *fcTriggerPMCOutOnCycleStart* and *fcTriggerPMCOutOnStartupCompleted* demand to set the parameter *TriggerGeneratingCC.*

8.4.1.1 FCTRIGGERCONDITIONPMC

This enumeration defines the conditions available for a trigger configuration of a FlexCard PMC (II). Please note that these conditions can not be OR-ed.

```
Typedef enum fcTriggerConditionPMC
{
       fcTriggerPMCNone = 0x00000000,
       fcTriggerPMCIn = 0x00000100,
       fcTriggerPMCOutOnPulse = 0x00001000,
       fcTriggerPMCOutOnErrorDetected = 0x00010000,
       fcTriggerPMCOutOnStartupCompleted = 0x00020000,
       fcTriggerPMCOutOnCycleStart = 0x00100000,
} fcTriggerConditionPMC;
```

Members

fcTriggerPMCNone
: This value can be used instead of zero.

fcTriggerPMCIn
: A trigger packet is generated as soon as the set edge (falling/rising) is detected on the input trigger line.

fcTriggerPMCOutOnPulse
: A signal is generated on the output trigger line as soon as the condition is set to the driver.

fcTriggerPMCOutOnErrorDetected
: A signal is generated on the output trigger line at a detected error.

fcTriggerPMCOutOnStartupCompleted
: A signal is generated on the output trigger line at a completed start up.

fcTriggerPMCOutOnCycleStart
: A signal is generated on the output trigger line at a cycle start.

See Also

fcbSetTrigger, fcTriggerConditionEx

Remarks

In the DebugAsynchron mode only the conditions *fcTriggerPMCNone*, *fcTriggerPMCIn* and *fcTriggerPMCOutOnPulse* can be used.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

8.5 OBSOLETE

8.5.1 FCBSETCCINDEX (OBSOLETE)

	Information
	This function is obsolete. Please use the functions in chapter 5 and 6 instead and specify the communication controller as parameter.

This function sets the FlexRay communication controller index. Following functions refer to the communication controller that was set.

```
fcError fcbSetCcIndex (
      fcHandle hFlexCard,
      fcCC     CCIndex
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

CCIndex

[IN] The FlexRay communication controller to be set.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcCC

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Remarks

The table below gives an overview of the functions which are CC specific.

CC specific functions	CC global functions
fcbMonitoringStart (Obsolete)	fcGetErrorCode
fcbMonitoringStop (Obsolete)	fcGetErrorType
fcbGetCcState (Obsolete)	fcGetErrorText
fcbSetTransceiverState (Obsolete)	fcFreeMemory
fcbGetTransceiverState (Obsolete)	fcbGetEnumFlexCards (Obsolete)
fcbSetCcRegister (Obsolete)	fcbOpen
fcbGetCcRegister (Obsolete)	fcbClose
fcbChiCcConfiguration (Obsolete)	fcbSetTrigger
fcbCanDbCcConfiguration (Obsolete)	fcbSetTimer
fcbConfigureMessageBuffer (Obsolete)	fcbNotificationPacket
fcbReconfigureMessageBuffer (Obsolete)	fcbReceive
fcbGetCcMessageBuffer (Obsolete)	fcbSetBusTermination
fcbResetCcMessageBuffer (Obsolete)	fcbGetBusTermination
fcbFilter (Obsolete)	fcbGetEnumFlexCardsV2 (Obsolete)
fcbSetEventHandle (Obsolete)	
fcbTransmit (Obsolete)	
fcbTransmitSymbol (Obsolete)	
fcbSetCcTimerConfig (Obsolete)	
fcbGetCcTimerConfig (Obsolete)	
fcbCalculateMacrotickOffset (Obsolete)	

8.5.2 FCBGETCCINDEX (OBSOLETE)

Information
This function is obsolete. Please use the functions in chapter 5 and 6 instead and specify the communication controller as parameter.

This function reads the index of the set FlexRay communication controller. Communication controller dependent functions refer to this communication controller only.

```
fcError fcbGetCcIndex (
      fcHandle hFlexCard,
      fcCC *   pCCIndex
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

pCCIndex

[OUT] The FlexRay communication controller which is currently set.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcbSetCcIndex (Obsolete), fcCC

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

9 ADDITIONAL LINUX API

There is an additional function available for the event handling in the FlexCard Linux driver.

9.1 INTEGRATION

For a detailed description of the installation process, please refer to the text file *Read_Me.txt* which is included in the zip archive.

After a successful installation please check the correct device initialization with 'cat /proc/flexcard'. All installed devices must be shown with versions and irq info. Please compare the irq info with used irqs (cat /proc/interrupts).

To use the additional Linux API, please include the header file *fcBaseLinux.h* in your application.

9.2 EVENT

The functions fcbSetEventHandle (Obsolete) and fcbSetEventHandleV2 register pthread conditions for notifying the user application. The pthread conditions are not async-signal safe and can result in deadlocks. Please use the function fcbSetEventHandleSemaphore to avoid deadlocks with the fcBase API.

9.2.1 FCBSETEVENTHANDLESEMAPHORE

This function registers an event handle (as semaphore) for a specific notification type. *hEvent* must be an unnamed POSIX semaphore from type (sem_t).

```
fcError fcbSetEventHandleSemaphore(
      fcHandle hFlexCard,
      fcCC CC,
      fcHandle hEvent,
      fcNotificationType type,
)
```

Parameters

hFlexCard
: [IN] Handle to a FlexCard

CC
: [IN] Communication controller index

hEvent
: [IN] Event handle to be registered to signal when a new cycle starts or a timer interval has elapsed depending on the given type.

Type
: [IN] The notification type for which the event has to be registered.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

See Also

fcCC, fcNotificationType

Remarks

The table below gives an overview of the fcNotificationType which are CC specific and which is not.

CC specific fcNotificationType	CC global fcNotificationType
fcNotificationTypeFRCycleStarted fcNotificationTypeFRWakeup fcNotificationTypeFRCcTimer	fcNotificationTypeTimer

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

10 ADDITIONAL XENOMAI API

There is a difference in the event handling between the FlexCard Xenomai driver and the other drivers for FlexCard Cyclone II (SE) and FlexCard PMC. Instead of fcbSetEventHandleV2 function, the fcbWaitForEventV2 function should be used.

10.1 INTEGRATION

For a detailed description of the installation process, please refer to the text file *Read_Me.txt* which is included in the zip archive.

After a successful installation please check the correct device initialisation with 'cat /proc/xeno_flexcard'. All installed devices must be shown with versions and irq info. Please compare the irq info with used irqs (cat /proc/interrupts). Make sure no non real time device shares an irq with a FlexCard.

To use the additional Linux API, please include the header file *fcBaseXENOMAI.h* in your application.

10.2 EVENT

10.2.1 FCBWAITFOREVENTV2

This function makes a safe real time I/O-Control that blocks the user process in kernel-space, until an event of the given type occurs or the event does not appear within the specified amount of time. The driver's kernel interrupt service routine then unblocks and the program routine continues. You don't need to set a handle with fcbSetEventHandle or fcbSetEventHandleV2.

```
fcError fcbWaitForEventV2(
      fcHandle hFlexCard,
      fcCC CC,
      fcNotificationType type,
      fcDword nTimeout
)
```

Parameters

hFlexCard
: [IN] Handle to a FlexCard

CC
: [IN] Communication controller index

type
: [IN] The notification type for which event has to be waited for.

nTimeout
: [IN] The maximum amount of time to wait for the event.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcCC, fcNotificationType

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Remarks

The table below gives an overview of the fcNotificationType which are CC specific and which is not.

CC specific fcNotificationType	CC global fcNotificationType
fcNotificationTypeFRCycleStarted fcNotificationTypeFRWakeup fcNotificationTypeFRCcTimer	fcNotificationTypeTimer

10.3 OBSOLETE

10.3.1 FCBWAITFOREVENT (OBSOLETE)

Information
This function is obsolete. Please use fcbWaitForEventV2 instead.

This function makes a safe real time I/O-Control that blocks the user process in kernel-space, until an event of the given type occurs or the event does not appear within the specified amount of time. The driver's kernel interrupt service routine then unblocks and the program routine continues. You don't need to set a handle with fcbSetEventHandle.

```
fcError fcbWaitForEvent(
      fcHandle hFlexCard,
      fcNotificationType hEvent,
      fcDword nTimeout
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

hEvent

[IN] The notification type for which event has to be waited for.

nTimeout

[IN] The maximum amount of time to wait for the event.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcNotificationType

Remarks

The table below gives an overview of the fcNotificationType which are CC specific and which is not. To use the CC specific fcNotificationType, the CC index has to be set.

CC specific fcNotificationType	CC global fcNotificationType
fcNotificationTypeCycleStarted fcNotificationTypeWakeup	fcNotificationTypeTimer

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

11 ADDITIONAL VXWORKS API

The VxWorks driver provides additional functionality for the FlexCard PMC. Please note that there are also some fcBase API functions and type definitions which were changed or are not supported by the VxWorks driver.

To use the driver in a user application, the header files *fcPmcDrv.h*, *fcBaseTypesVxWorks.h* and *fcbBaseVxWorks.h* have to be included in that order.

11.1 INTEGRATION

For a detailed description of the installation process, please refer to the text file *Read_Me.txt* which is included in the zip archive.

11.1.1 FCDRVINIT

This function initializes the FlexCard PMC VxWorks driver.

```
STATUS fcDrvInit()
```

Return values

If the function succeeds, the return value is (OK). If the value is (ERROR) the driver couldn't be initialized.

See Also

fcDrvExit

11.1.2 FCDRVEXIT

This function finalizes the FlexCard PMC VxWorks driver.

```
STATUS fcDrvExit()
```

Return values

If the function succeeds, the return value is (OK). If the value is (ERROR) the driver couldn't be finalized.

See Also

fcDrvInit

11.2 RESTRICTIONS / CHANGES

11.2.1 NOT SUPPORTED TYPE DEFINITIONS

The VxWorks driver doesn't support the following type definitions:

- fcFreeMemory

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

- fcTriggerCondition (Obsolete)
- fcTriggerType (Obsolete)
- fcTriggerMode (Obsolete)
- fcTriggerCfgHardware (Obsolete)
- fcTriggerCfgSoftware (Obsolete)
- fcTriggerCfg (Obsolete)
- fcTriggerInfoPacket (Obsolete)
- fcTriggerConditionEx

11.2.2 CHANGED TYPE DEFINITIONS

11.2.2.1 FCVERSION

This structure provides version information about the FlexCard hardware and software components.

```
Typedef struct fcVersion
{
      fcVersionNumber    DeviceDriver;
      fcVersionNumber    Firmware;
      fcVersionNumber    Hardware;
      fcCCType           CCType;
      fcVersionNumber    CC;
      fcVersionNumber    BusGuardian;
      fcVersionNumber    Protocol;
      fcDword            Serial;
      fcFlexCardDeviceId DeviceId;
      fcDword            Reserved[3];
} fcVersion;
```

Members

DeviceDriver
: Device driver version

Firmware
: Firmware (gateway software) version

Hardware
: FlexCard hardware version

CCType
: Communication controller type

CC
: Communication controller module version

BusGuardian
: Bus Guardian version

Protocol
: FlexRay Protocol version

Serial
: FlexCard serial number. A zero value indicates a non-valid FlexCard serial number.

DeviceId
: Device identifier to detect the FlexCard type (FlexCard Cyclone II, FlexCard Cyclone II SE or FlexCard PMC)

Reserved[3]
: Reserved for internal purpose

See Also

fcInfo, fcbGetEnumFlexCards (Obsolete), fcFlexCardDeviceId

11.2.2.2 FCTRIGGERCONFIGURATIONEX

This structure is used for the configuration of a trigger. By using the parameter *Condition* several triggers can be enabled. The trigger conditions of the FlexCard PMC are defined in the enumeration

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

fcTriggerConditionPMC. The **conditions cannot be combined (OR-ed)**. If it is done, none of the conditions will be set and an error message will be returned.

The conditions fcTriggerPMCOutOnErrorDetected, fcTriggerPMCOutOnCycleStart and fcTriggerPMCOutOnStartupCompleted demand to set the parameter *TriggerGeneratingCC.* Please note the FlexCard trigger lines are not hard defined as IN or OUT trigger lines. Therefore a valid value has always to be set for the parameter *TriggerLineToConfigure*.

```
Typedef struct fcTriggerConfigurationEx
{
      fcDword Condition;
      fcDword onEdge;
      fcDword TriggerLineToConfigure;
      fcCC    TriggerGeneratingCC;
      fcDword Reserved[4];
} fcTriggerConfigurationEx;
```

Members

Condition
: This parameter can either be set to 0 (fcTriggerPMCNone) to reset the trigger or to any condition available in fcTriggerConditionPMC.

onEdge
: This parameter has to be set when the condition fcTriggerPMCIn is chosen. Valid values are 0 = falling edge and 1 = rising edge.

TriggerLineToConfigure
: This parameter sets the trigger line which should be configured. Valid values range from 1 to 2.

TriggerGeneratingCC
: This parameter has to be set when a CC dependent trigger condition was set. Valid values range from fcCC1 to fcCC2.

Reserved[4]
: Reserved Dwords for possible later use.

See Also

fcbSetTrigger, fcTriggerConditionPMC

11.2.2.3 FCNOTIFICATIONTYPE

This enumeration defines different notification types. These types are used in the functions fcbSetEventHandle and fcbSetNotificationTypeCount to specify on which kind of event the application has to be notified.

```
Typedef enum fcNotificationType
{
      fcNotificationTypeCycleStarted      = 1,
      fcNotificationTypeTimer             = 2,
      fcNotificationTypeWakeup            = 3,
      fcNotificationTypeRxCount           = 4,
      fcNotificationTypeTxCount           = 5,
      fcNotificationTypeInfoCount         = 6,
      fcNotificationTypeErrorCount        = 7,
      fcNotificationTypeStatusCount       = 8,
      fcNotificationTypeTriggerCount      = 9,
      fcNotificationTypeNMVCount          = 10,
      fcNotificationTypeNotificationCount = 11,
      fcNotificationTypeCcTimer           = 12,
} fcNotificationType;
```

Members

fcNotificationTypeCycleStarted
: Used to notify that a new cycle has started and that probably new data has been received.

fcNotificationTypeTimer
: Used to notify that the timer interval has elapsed. This notification requires the internal timer of the FlexCard to be enabled (See fcbSetTimer).

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

fcNotificationTypeWakeup
Used to notify that one of the transceivers has received a wakeup event (only if one of the transceivers was in sleep mode).

fcNotificationTypeRxCount
Used to notify that the configured amount of FlexRay frames has been received. This notification can be configured (See fcbSetNotificationTypeCount).

fcNotificationTypeTxCount
Used to notify that the configured amount of TxAcknowledge frames has been received. This notification can be configured (See fcbSetNotificationTypeCount).

fcNotificationTypeInfoCount
Used to notify that the configured amount of info frames has been received. This notification can be configured (See fcbSetNotificationTypeCount).

fcNotificationTypeErrorCount
Used to notify that the configured amount of error frames has been received. This notification can be configured (See fcbSetNotificationTypeCount).

fcNotificationTypeStatusCount
Used to notify that the configured amount of status frames has been received. This notification can be configured (See fcbSetNotificationTypeCount).

fcNotificationTypeTriggerCount
Used to notify that the configured amount of trigger frames has been received. This notification can be configured (See fcbSetNotificationTypeCount).

fcNotificationTypeNMVCount
Used to notify that the configured amount of network management vector frames has been received. This notification can be configured (See fcbSetNotificationTypeCount).

fcNotificationTypeNotificationCount
Used to notify that the configured amount of notification frames has been received. This notification can be configured (See fcbSetNotificationTypeCount).

fcNotificationTypeCcTimer
Used to notify that the configured cc timer macrotick offset has elapsed.

See Also

fcbMonitoringStart, fcbSetEventHandle, fcbSetNotificationTypeCount, fcbSetTimer, fcbSetCcTimerConfig (Obsolete)

11.2.2.4 FCTRIGGEREXINFOPACKET

This structure provides information about a trigger packet.

```
Typedef struct fcTriggerExInfoPacket
{
      fcDword Condition;
      fcDword TimeStamp;
      fcDword SequenceCount;
      fcDword Edge;
      fcDword TriggerLine;
      fcDword Reserved[4];
} fcTriggerExInfoPacket;
```

Members

Condition
The fulfilled condition which has caused the trigger packet generation.

TimeStamp
The FlexCard time stamp (1 µs resolution). Indicates the time at which the packet was generated.

SequenceCount
Sequence count for each signal.

Edge
The edge on which the trigger was signalled.

TriggerLine
The trigger line which detected a trigger signal.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Reserved[4]
Reserved for future use.

See Also
fcPacket

11.2.2.5 FCPACKETTYPE

This enumeration contains the different packet types.

```
Typedef enum fcPacketType
{
      fcPacketTypeInfo          = 1,
      fcPacketTypeFlexRayFrame  = 2,
      fcPacketTypeError         = 3,
      fcPacketTypeStatus        = 4,
      fcPacketTypeTxAcknowledge = 6,
      fcPacketTypeNMVector      = 7,
      fcPacketTypeNotification  = 8,
      fcPacketTypeTriggerEx     = 9,
} fcPacketType;
```

Members
fcPacketTypeInfo
Frame is an info packet.
fcPacketTypeFlexRayFrame
Frame is a FlexRay frame.
fcPacketTypeError
Frame is an error packet.
fcPacketTypeStatus
Frame is a status packet.
fcPacketTypeTxAcknowledge
Frame is a transmit acknowledge packet.
fcPacketTypeNMVector
Frame is a network management vector packet.
fcPacketTypeNotification
Frame is a notification packet.
fcPacketTypeTriggerEx
Frame is a trigger packet.

See Also
fcPacket, fcInfoPacket, fcFlexRayFrame, fcTxAcknowledgePacket, fcErrorPacket, fcStatusPacket, fcNMVectorPacket, fcNotificationPacket, fcTriggerExInfoPacket

11.2.2.6 FCPACKET

This structure provides information about a packet.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef struct fcPacket
{
      fcPacketType Type;
      union
      {
            fcFlexRayFrame*        FlexRayFrame;
            fcInfoPacket*          InfoPacket;
            fcErrorPacket*         ErrorPacket;
            fcStatusPacket*        StatusPacket;
            fcTriggerExInfoPacket* TriggerExPacket;
            fcTxAcknowledgePacket* TxAcknowledgePacket;
            fcNMVectorPacket*      NMVectorPacket;
            fcNotificationPacket*  NotificationPacket;
      };
      fcPacket* pNextPacket;
} fcPacket;
```

Members

Type
: Type of packet.

FlexRayFrame
: Pointer to the packet data. The content depends on the type of packet.

InfoPacket
: Pointer to the packet data. The content depends on the type of packet.

ErrorPacket
: Pointer to the packet data. The content depends on the type of packet.

StatusPacket
: Pointer to the packet data. The content depends on the type of packet.

TriggerExPacket
: Pointer to the packet data. The content depends on the type of packet.

TxAcknowledgePacket
: Pointer to the packet data. The content depends on the type of packet.

NMVectorPacket
: Pointer to the packet data. The content depends on the type of packet.

NotificationPacket
: Pointer to the packet data. The content depends on the type of packet.

pNextPacket
: Pointer to the next packet. If the pointer is NULL, there are no more packets available.

See Also

fcPacketType, fcInfoPacket, fcFlexRayFrame, fcTxAcknowledgePacket, fcErrorPacket, fcStatusPacket, fcNMVectorPacket, fcNotificationPacket, fcTriggerExInfoPacket

11.2.2.7 FCSTATE

This enumeration defines the possible communication controller POC states (FlexRay Protocol Specification: vPOC!State). For more details about communication controller POC states, please refer to [3].

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
Typedef enum fcState
{
      fcStateUnknown = 0,
      fcStateDefaultConfig,
      fcStateReady,
      fcStateNormalActive,
      fcStateNormalPassive,
      fcStateHalt,
      fcStateMonitorMode,
      fcStateConfig,

      fcStateWakeupStandby,
      fcStateWakeupListen,
      fcStateWakeupSend,
      fcStateWakeupDetect,

      fcStateStartupPrepare,
      fcStateColdstartListen,
      fcStateColdstartCollisionResolution,
      fcStateColdstartConsistencyCheck,
      fcStateColdstartGap,
      fcStateColdstartJoin,
      fcStateIntegrationColdstartCheck,
      fcStateIntegrationListen,
      fcStateIntegrationConsistencyCheck,
      fcStateInitializeSchedule,
      fcStateAbortStartup,
      fcStateStartupSuccess,
}fcState;
```

Members

fcStateUnknown
: Communication controller state is not known.

fcStateDefaultConfig
: Communication controller is in DEFAULT_CONFIG state.

fcStateReady
: Communication controller is in READY state.

fcStateNormalActive
: Communication controller is in NORMAL_ACTIVE state.

fcStateNormalPassive
: Communication controller is in NORMAL_PASSIVE state.

fcStateHalt
: Communication controller is in HALT state.

fcStateMonitorMode
: Communication controller is in MONITORMODE state

fcStateConfig
: Communication controller is in CONFIG state.

fcStateWakeupStandby
: Communication controller is in WAKEUP_STANDBY state.

fcStateWakeupListen
: Communication controller is in WAKEUP_LISTEN state.

fcStateWakeupSend
: Communication controller is in WAKEUP_SEND state.

fcStateWakeupDetect
: Communication controller is in WAKEUP_DETECT state.

fcStateStartupPrepare
: Communication controller is in STARTUP_PREPARE state.

fcStateColdstartListen
: Communication controller is in COLDSTART_LISTEN state.

fcStateColdstartCollisionResolution
: Communication controller is in COLDSTART_COLLISION_RESOLUTION state.

fcStateColdstartConsistencyCheck
: Communication controller is in COLDSTART_CONSISTENCY_CHECK state.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

fcStateColdstartGap
Communication controller is in COLDSTART_GAP state.
fcStateColdstartJoin
Communication controller is in COLDSTART_JOIN state.
fcStateIntegrationColdstartCheck
Communication controller is in INTEGRATION_COLDSTART_CHECK state.
fcStateIntegrationListen
Communication controller is in INTEGRATION_LISTEN state.
fcStateIntegrationConsistencyCheck
Communication controller is in INTEGRATION_CONSISTENCY_CHECK state.
fcStateInitializeSchedule
Communication controller is in INITIALIZE_SCHEDULE state.
fcStateAbortStartup
Communication controller is in ABORT_STARTUP state.
fcStateStartupSuccess
Communication controller is in STARTUP_SUCCESS state.

See Also
fcbGetCcState, fcbMonitoringStart

11.2.3 NOT SUPPORTED FUNCTIONS

The VxWorks driver doesn't support the following functions:

- fcGetErrorText
- fcFreeMemory
- fcbCanDbCcConfiguration (Obsolete)
- fcbTrigger (Obsolete)
- fcbGetEnumFlexCardsV2 (Obsolete)

11.2.4 CHANGED FUNCTIONS

11.2.4.1 FCBMONITORINGSTART

This function is used to start the monitoring of a FlexRay bus. Once called, the function changes the communication controller state from configuration state to normal active state (if the cluster integration succeeds). The current communication controller state can be read using the function fcbGetCcState (Obsolete). If the FlexCard is synchronized with the cluster the function fcbGetCcState (Obsolete) will return the value *fcStateNormalActive.* Please note, that if an event for the event counter (for the several packet type) is registered with fcbSetEventHandle, this function activates the corresponding hardware interrupts and the application is notified if this event occurred.

```
fcError fcbMonitoringStart(
      fcHandle hFlexCard,
      fcMonitoringModes mode,
      bool restartTimestamps,
      bool enableCycleStartEvents
      bool enableColdstart,
      bool enableWakeup
)
```

Parameters
hFlexCard
[IN] Handle to a FlexCard.
Mode
[IN] The monitoring mode. See fcMonitoringModes for details.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

restartTimestamps

[IN] Set this parameter to false to restart the measurement without resetting the FlexCard timestamp. Set it to true to start the measurement from the beginning. The timestamps have micro second resolution.

enableCycleStartEvents

[IN] Set this parameter to true to enable the cycle start events in order that at the beginning of every cycle the event `fcNotificationTypeCycleStarted` is signalled.

enableColdstart

[IN] Set this parameter to true to allow the FlexCard to initialize the cluster communication, otherwise the coldstart inhibit mode is active. This feature can not be used in the monitoring modes fcMonitoringDebug and fcMonitoringDebugAsynchron.

enableWakeup

[IN] Set this parameter to true to transmit a wakeup pattern to the configured wakeup channel (FlexRay Protocol Specification: pWakeupChannel). A cluster wakeup must precede the communication start up to ensure that all nodes in a cluster are awake. The minimum requirement for a cluster wakeup is that all bus drivers are supplied with power. This feature can not be used in the monitoring modes fcMonitoringDebug and fcMonitoringDebugAsynchron.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Remarks

After the monitoring has started, the user should check if the integration in the cluster was successful: fcbGetCcState (Obsolete) should return the state `fcStateNormalActive`.

	Information
	After the monitoring has successfully started, the receive process has to be started as soon as possible to avoid an overflow (error packet fcErrFlexcardOverflow is received). Once an overflow occurred, no more packets can be received. The monitoring has to be stopped and started again.

See Also

fcbMonitoringStop, fcbGetCcState (Obsolete), fcMonitoringModes, fcbSetEventHandle

11.2.4.2 FCBMONITORINGSTOP

This function stops the FlexRay bus measurement. The communication controller is set back in its configuration state.

Please note, that if an event for the event counter (for the several packet types) is registered with fcbSetEventHandle, this function deactivates the corresponding hardware interrupts and the application is not notified if this event occurred.

```
fcError fcbMonitoringStop(
      fcHandle hFlexCard
)
```

Parameters

hFlexCard

[IN] Handle to FlexCard

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

See Also

fcbMonitoringStart

11.2.4.3 FCBSETEVENTHANDLE

This function registers an event handle for a specific notification type. The event handling is based on binary semaphores.

```
fcError fcbSetEventHandle(
      fcHandle hFlexCard,
      fcHandle hEvent,
      fcNotificationType type
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

hEvent

[IN] Event handle to be registered. This value depends on the given *type*. Set this parameter to NULL to deregister the event handle for the given type.

Type

[IN] The notification type for which the event has to be registered.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcNotificationType

Example

See Example in fcbSetNotificationTypeCount

11.2.4.4 FCBRECEIVE

This function reads all available packets from the FlexCard memory into a memory block allocated by the fcBase API during the initialization phase in fcbOpen. The frames are stored into a linked list. The memory allocated by this function is released by the fcbClose function. Please note, that every function call from fcbReceive overwrites the old frames in the memory block. The size of the memory block can be configured with fcbSetReceiveMemorySize.

```
fcError fcbReceive(
      fcHandle hFlexCard,
      fcPacket** pPacket
);
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

pPacket

[OUT] Address of the fcPacket object pointer. The memory for this structure and its content is allocated by the fcBase API. Packets are available if the return code is 0 and *pPacket* is not a null pointer.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Example

```
fcPacket* pPackets = NULL;
fcError e = fcbReceive(m_hFlexCard, &pPackets);
if (0 == e)
{
      fcPacket* pCurrentPacket = pPacket;
      while (NULL != pCurrentPacket)
      {
            switch (pCurrentPacket->Type)
            {
            case fcPacketTypeInfo:
            {
                  fcInfoPacket* pFrame = pCurrentPacket->InfoPacket;
                  printf("[fcPacketTypeInfo] CC: %d TimeStamp: %f Cycle: %d",
                        pFrame->CC + 1,
                        (float) pFrame->TimeStamp * 0.000001,
                        pFrame->CurrentCycle);
                  printf(" Rate Correction: %d", pFrame->RateCorrection);
                  printf(" Offset Correction: %d", pFrame->OffsetCorrection);
                  printf(" Clock Correction Failed Counter: %d",
                        pFrame->ClockCorrectionFailedCounter);
                  printf(" Passive to Active Count: %d",
                        pFrame->PassiveToActiveCount);
                  printf("\n");
                  break;
            }

            case fcPacketTypeFlexRayFrame:
            {
                  fcFlexRayFrame* pFrame = pCurrentPacket->FlexRayFrame;
                  printf("[fcPacketTypeFlexRayFrame] CC: %d TimeStamp: %f "
                        "Cycle: %d Id: %d Channel: %d PayloadLength: %d",
                        pFrame->CC + 1,
                        (float) pFrame->TimeStamp * 0.000001,
                        pFrame->CycleCount,
                        pFrame->ID,
                        pFrame->Channel,
                        pFrame->PayloadLength);

                  for (int i = 0; i < pFrame->PayloadLength; i++)
                  {
                        printf(" %04X", pFrame->pData[i]);
                  }

                  if (pFrame->PP) printf(" PP");
                  if (pFrame->NF) printf(" NF");
                  if (pFrame->SYNC) printf(" SYNC");
                  if (pFrame->STARTUP) printf(" STARTUP");
                  if (pFrame->SyntaxError) printf(" SyntaxError");
                  if (pFrame->ContentError) printf(" ContentError");
                  if (pFrame->ValidFrame) printf(" ValidFrame");
                  if (pFrame->SlotBoundaryViolation)
                        printf(" SlotBoundaryViolation");
                  printf("\n");
                  break;
            }

            case fcPacketTypeError:
                  printf("[fcPacketTypeError]\n");
                  break;

            case fcPacketTypeStatus:
                  printf("[fcPacketTypeStatus]\n");
                  break;

            case fcPacketTypeTriggerEx:
                  printf("[fcPacketTypeTriggerEx]\n");
                  break;

            case fcPacketTypeTxAcknowledge:
                  printf("[fcPacketTypeTxAcknowledge]\n");
                  break;

            case fcPacketTypeNMVector:
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

```
                    printf("[fcPacketTypeNMVector]\n");
                    break;
                }

                pCurrentPacket = pCurrentPacket->pNextPacket;
            }
}
```

11.3 CONFIGURATION

11.3.1 FCBSETPACKETGENERATION

This function allows to dis- or enable the generation of a packet type. It is designed to reduce the amount of packets, which will be generated by the FlexCard.

```
fcError fcbSetPacketGeneration(
      fcHandle hFlexCard,
      fcPacketType type,
      bool bEnable
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

type

[IN] The packet type.

bEnable

[IN] Set to true to enable the generation and to false to disable it.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcbReceive, fcPacketType

11.3.2 FCBSETRECEIVEMEMORYSIZE

This function allows configuring the size of memory, where fcbReceive will store all received frames. This function has to be called before you open a connection to the FlexCard. During the initialisation phase (in fcbOpen) the amount of memory is dynamically allocated. Closing the connection (by fcbClose) releases the memory automatically.

```
fcError fcbSetReceiveMemorySize(
      fcDword size;
)
```

Parameters

size

[IN] The size of memory. The default value is 128 kB and it is recommended to set *size* in a range from 20 kB to 70 MB. Other values than the recommended values are ignored and *size* will be set to default.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

See Also

fcbReceive, fcbOpen, fcbClose

11.4 EVENT

11.4.1 FCBSETNOTIFICATIONTYPECOUNT

This function allows configuring the event counter for the several packet types. *Count* represents the amount of packets (of a dedicated packet type) which need to be received to initiate an event of the chosen notification packet type.

```
fcError fcbSetNotificationTypeCount(
      fcHandle hFlexCard,
      fcNotificationType type,
      fcByte count
)
```

Parameters

hFlexCard

[IN] Handle to a FlexCard

type

[IN] The notification type for which the configuration has to be used. The notification types *fcNotificationTypeCycleStarted*, *fcNotificationTypeWakeup*, *fcNotificationTypeTimer* and *fcNotificationTypeCcTimer* are not supported.

Count

[IN] The value represents the amount of packets (of a dedicated packet type) which need to be received to initiate an event of the chosen notification packet type. Valid values range from 1 to 255.

Return values

If the function succeeds, the return value is 0. If the value is <> 0, use the functions described in the section Error Handling to get extended error information.

See Also

fcNotificationType, fcbSetEventHandle, fcbMonitoringStart, fcbMonitoringStop

Example

```
fcPacket* pPackets = NULL;
SEM_ID semInfoCount = NULL;
semInfoCount = semBCreate(SEM_Q_FIFO, SEM_EMPTY);
assert (NULL != semInfoCount);

fcError e = fcbSetEventHandle(m_hFlexCard, (void *) semInfoCount, \
      fcNotificationTypeInfoCount);
if (0 == e)
{
      // Configure the Info packet event counter
      fcbSetNotificationTypeCount(m_hFlexCard,fcNotificationTypeInfoCount, 2);

      // Start monitoring and wait for the event forever
      fcbMonitoringStart(m_hFlexCard, fcMonitoringNormal, 1, 0, 0, 0);
      semTake(semInfoCount, WAIT_FOREVER);

      // Min. 2 Info packets can be received now
      e = fcbReceive(m_hFlexCard, &pPackets);
      if (0 == e)
      { /* Process packets */ }
}
```


Copyright 2009 Eberspächer Electronics GmbH & Co. KG

12 POWER MANAGEMENT

Figure 14: System Standby Failed message box

On Windows 2000 and Windows XP stand-by and hibernation is prohibited by the FlexCard in order to provide a continuous monitoring of a bus and not to disturb a running network. On Windows Vista however, drivers are not allowed to prohibit power saving functions so that energy costs can be reduced and battery duration is improved.

When the PC enters standby or hibernation, the current measurement is stopped automatically. After standby the handle to the driver is not valid anymore. On resume, application developers should call fcbClose, fcbGetEnumFlexCardsV3, fcbOpen to initialize the FlexCard again.

Applications have the possibility to react to standby or resume with the Windows message `WM_POWERBROADCAST` containing the events `PBT_APMSUSPEND` and `PBT_APMRESUMESUSPEND`.

Application developers may inform the user under Windows Vista that standby and hibernation stops the current monitoring. This is possible in the user manual or with a message window at the first start of the application. Users have the possibility to deactivate the automatic standby in the control panel.

Application developers may consider deactivating idle recognition with the command SetThreadExecutionState() to prohibit automatic stand-by. However, manual switching to stand-by can not be prevented under Windows Vista.

	Information
	Please note: Power Management is only supported under Microsoft Windows operating systems. Under Linux, please deactivate kernel power management options to avoid undefined behaviour with the FlexCard Linux and Xenomai driver.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

13 TRACING

13.1 OVERVIEW

The tracing module allows the user to get more information about the fcBase DLL (Windows only) activity (e.g. in the case of an error).

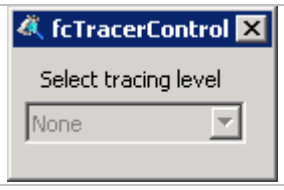
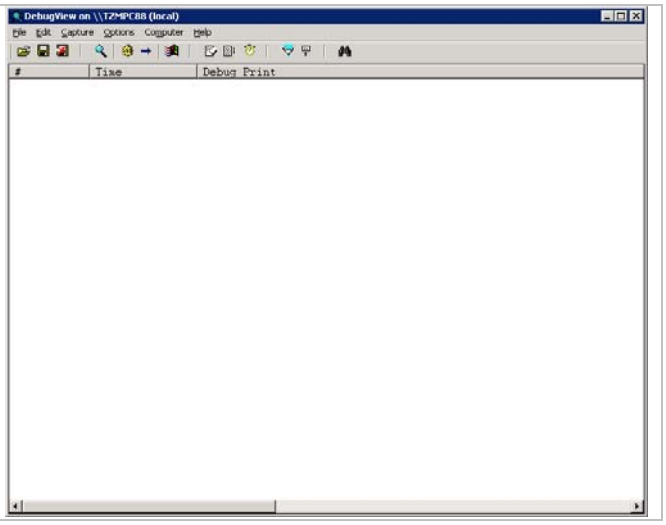
The tracing consists of three parts:

- The tracing module inside the fcBase dynamic link library. This module will send the trace messages to a debugger for displaying (using the windows function OutputDebugString).
- The tracing control application to choose the tracing level.
- A debug output viewer (e.g. DebugView from SysInternals) to view the trace messages. If you are debugging your own application, the messages appear normally in the debug output window of your IDE.

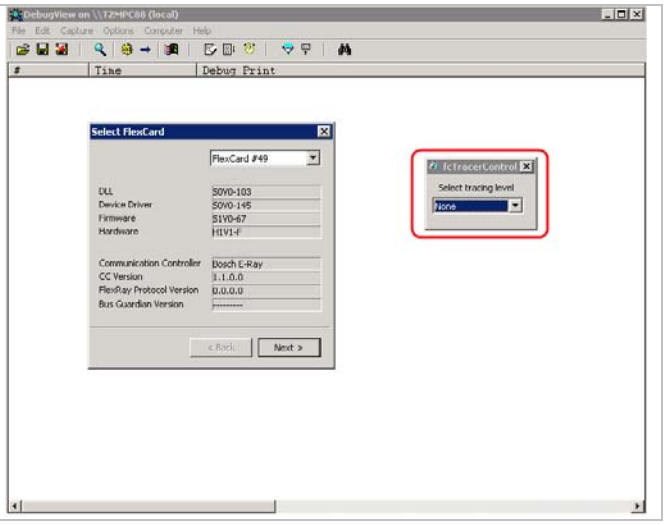
The followings tracing levels are available:

- Debug: all trace messages will be shown.
- Info: info and warning messages will be shown.
- Warn: only warning messages will be shown.
- Error: only error messages will be shown.
- Fatal: only fatal error message will be shown.
- None: tracing messages will not be generated.

To use the tracing the following steps are required:

Step 1 Start the tracing control application (fcTracerControl.exe)	
Step 2 Start the debug output viewer (DebugView.exe)	

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Step 3 Start your application. In our case we use the demo application (fcDemo.exe). Now, the tracing level should be selectable.	
Step 4 Activate the tracing by choosing a tracing level different of None (e.g. Debug). Use your application and view the trace messages.	

13.2 LIMITATION

The tracing module inside the fcBase DLL will update the new tracing level only by calling the following functions:

- fcbGetEnumFlexCards (Obsolete)
- fcbGetEnumFlexCardsV2 (Obsolete)
- fcbGetEnumFlexCardsV3
- fcbOpen
- fcbClose

That means a level modification by the tracing control application will only be passed to the tracing module inside the fcBase DLL if one of the above functions is called.

This limitation ensures that performance critical functions such as fcbReceive or fcbTransmit are not delayed.

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

14 APPENDIX

14.1 BIBLIOGRAPHY

[1] FlexCard Cyclone II (SE) Instruction for Use (3-0009-0T01-D01)
[2] MSDN: Dynamic-Link Library Search Order
[3] FlexRay Protocol Specification V2.1 Rev. A
[4] FlexRay Electrical Physical Layer Specification V2.1 Rev. A

14.2 ABBREVIATIONS

Abbreviations	Definition
API	Programming Interface
DLL	Dynamic Link Library
IDE	Integrated Development Environment
PDF	Portable Document Format
SYS	System device driver
MFC	Microsoft Foundation Class
CC	Communication controller
PMC	PCI Mezzazine Card
LKM	Loadable kernel module (for Linux OS)
LIB	Library (shared object file)

14.3 GLOSSARY

Term	Description
INF File	A text-based file containing information required by the system to install a device's software components
MFC	C++ Application framework for programming in Microsoft Windows
Qt	C++ Application framework for programming platform independent applications
Cluster	Network topology
CHI	File that configures a communication controller
CANdb	File that configures a communication controller

14.4 LIST OF FIGURES

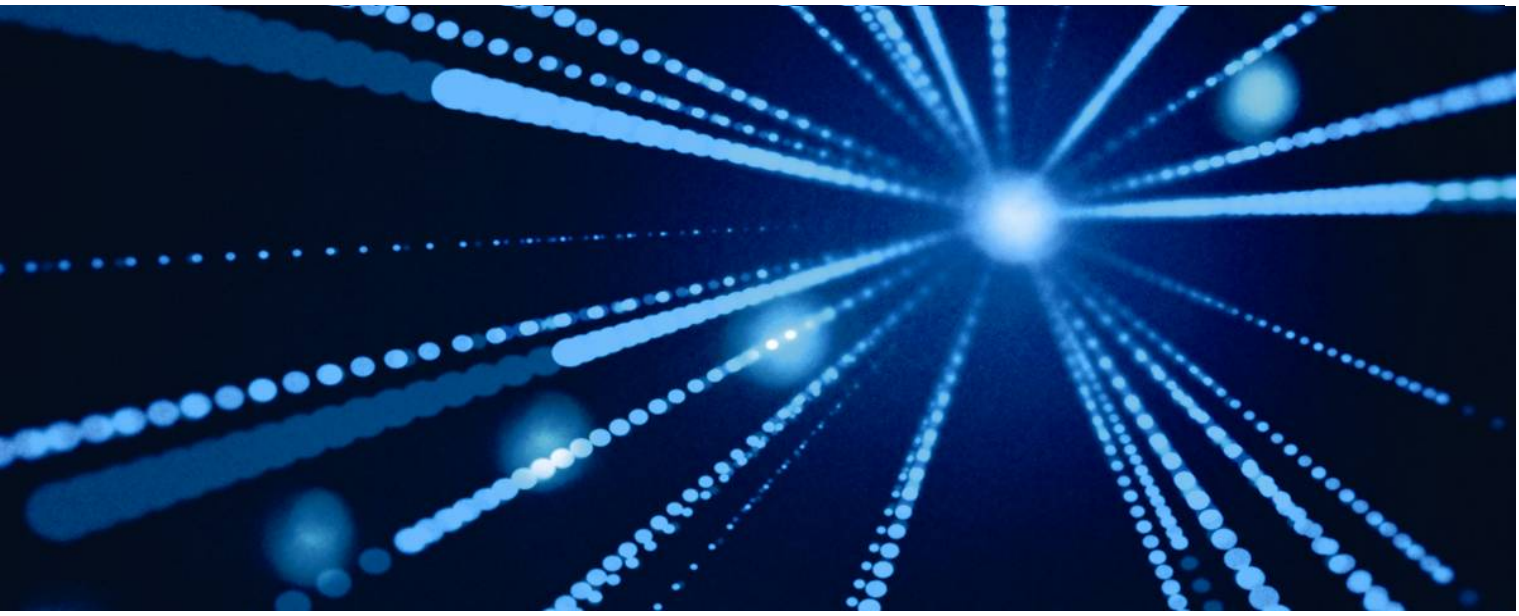
Copyright 2009 Eberspächer Electronics GmbH & Co. KG

14.5 INDEX

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Copyright 2009 Eberspächer Electronics GmbH & Co. KG

Copyright 2009 Eberspächer Electronics GmbH & Co. KG



www.eberspaecher.com/electronics

Eberspächer Electronics
GmbH & Co. KG
Robert-Bosch-Str. 6
73037 Göppingen
Phone +49 7161 9559-0
Fax +49 7161 9559-455
ebel-info@eberspaecher.com
www.eberspaecher.com/electronics

Eberspächer