FlexRay UNIFIED Driver

Application Examples User Guide

Based on
MFR4300/MC9S12XDP512, MC9S12XFR128,
MFR4310/MC9S12XDP512,
MFR4300/MPC5554, MPC5567, MPC5561, MPC5510,
MFR4310/MPC5554, MPC560xP
MC9S12XF512/384/256/128

UG
Rev. 1.3.1
01/2010

freescale.com

freescale™
semiconductor

Chapter 1
Introduction 11

Chapter 2
Installation Steps 13

2.1 CW Development Studio v1.5b2 for the MPC556x MCU 13
2.1.1 Modifying the Application 14
2.1.2 Used Compiler Platform 14
2.2 CW Development Studio v2.3 for the MPC5567 MCU 15
2.2.1 Modifying the Application 15
2.2.2 Used Compiler Platform 16
2.3 GreenHills MULTI Project Builder for the MPC556x MCU 16
2.3.1 Modifying the Application 17
2.3.2 Used Compiler Platform 17
2.4 WindRiver DIAB C Compiler for the MPC556x MCU 17
2.5 GreenHills MULTI Project Builder for the MPC556x MCU 18
2.5.1 Modifying the Application 19
2.5.2 Used Compiler Platform 19
2.6 WindRiver DIAB C Compiler for the MPC556x MCU 20
2.6.1 Modifying the Application 20
2.6.2 Used Compiler Platform 21
2.7 CW Development Studio for the MC9S12XFR128 MCU 21
2.7.1 Modifying the Application 22
2.7.2 Used Compiler Platform 22
2.8 CW Development Studio for the MC9S12XDP512 MCU with MFR43x0 CC 22
2.8.1 Modifying the Application 23
2.8.2 Used Compiler Platform 23
2.9 CW Development Studio for the MPC5554 MCU with MFR43x0 CC 23
2.9.1 Modifying the Application 24
2.9.2 Used Compiler Platform 24
2.10WindRiver DIAB C Compiler for the MPC5554 MCU with MFR43x0 CC 24
2.10.1 Modifying the Application 25
2.10.2 Used Compiler Platform 25
2.11GreenHills MULTI Project Builder for the MPC5554 MCU with MFR43x0 CC 25
2.11.1 Modifying the Application 26
2.11.2 Used Compiler Platform 26
2.12WindRiver DIAB C Compiler for the MPC5516 MCU 26
2.12.1 Modifying the Application 27
2.12.2 Used Compiler Platform 27
2.13GreenHills MULTI Project Builder for the MPC5516 MCU 27
2.13.1 Modifying the Application 28
2.13.2 Used Compiler Platform 28
2.14CW Development Studio for the MC9S12XF512/384/256/128 MCU 28

2.14.1 Modifying the Application 29
2.14.2 Used Compiler Platform 29
2.15CW Development Studio v2.3 for the MPC5517 MCU 29
2.15.1 Modifying the Application 30
2.15.2 Used Compiler Platform 30
2.16GreenHills MULTI Project Builder for the MPC560xP MCU 30
2.16.1 Modifying the Application 31
2.16.2 Used Compiler Platform 31
2.17CW Development Studio v2.5 for the MPC560xP MCU 32
2.17.1 Modifying the Application 32
2.17.2 Used Compiler Platform 33
2.18Installation of the FreeMASTER tool 33

Chapter 3
Description of the Application Examples 35

3.1 Application Examples Memory Requirements and Performance 35
3.1.1 UNIFIED Driver Performance Data 37
3.2 Transmit Receive Application Example 38
3.2.1 FlexRay Schedule and Timing 39
3.2.2 Configuration for Node 1 40
3.2.3 Configuration for Node 2 42
3.2.4 Detailed Description of Application Software for Node 1 43
3.2.5 Detailed Description of Application Software for Node 2 47
3.2.6 Demonstration of the Transmit Receive Application Example - Running the FreeMASTER tool 50
3.3 Status Monitoring Application Example 53
3.3.1 FlexRay Schedule and Timing 53
3.3.2 Configuration for Node 1 54
3.3.3 Configuration for Node 2 57
3.3.4 Detailed Description of Application Software for Node 1 60
3.3.5 Detailed Description of Application Software for Node 2 64
3.3.6 Demonstration of the Slot Status Application Example - Running the FreeMASTER tool 68

Chapter 4
Application Example Project Structures. 71

4.1 CW Development Studio v1.5b2 for the MPC556x MCU 71
4.2 CW Development Studio v2.3 for the MPC5567 MCU 74
4.3 GreenHills MULTI Project Builder for the MPC556x MCU 77
4.4 WindRiver DIAB C Compiler for the MPC556x MCU 80
4.5 CW Development Studio for the MC9S12XFR128 MCU 82
4.6 CW Development Studio for the MC9S12XDP512 MCU with MFR43x0 CC 85
4.7 CW Development Studio for the MPC5554 MCU with MFR43x0 CC 88
4.8 WindRiver DIAB C Compiler for the MPC5554 MCU with MFR43x0 CC 91

4.9 GreenHills MULTI Project Builder for the MPC5554 MCU with MFR43x0 CC 93
4.10WindRiver DIAB C Compiler for the MPC5516 MCU 96
4.11GreenHills MULTI Project Builder for the MPC5516 MCU 98
4.12CW Development Studio for the MC9S12XF512/384/256/128 MCU 101
4.13CW Development Studio v2.3 for the MPC5517 MCU 104
4.14GreenHills MULTI Project Builder for the MPC560xP MCU 107
4.15CW Development Studio v2.5 for the MPC560xP MCU 110

Chapter 5
Possible Errors and Workarounds. 113

FlexRay Freescale UNIFIED Driver Application Examples

User Guide

by:

David Svrcek, david.svrcek@freescale.com

Roman Chovanec, roman.chovanec@freescale.com

Jan Perny, jan.perny@freescale.com

Roznov Czech System Center
Roznov pod Radhostem, Czech Republic

To provide the most up-to-date information, the revision of our documents on the World Wide Web will be the most current. Your printed copy may be an earlier revision. To verify you have the latest information available, refer to:

http://www.freescale.com

The following revision history table summarizes changes contained in this document. For your convenience, the page number designators have been linked to the appropriate location.

Revision History

Date	Revision Level	Description	Page Number(s)
August 2006	0.1	Initial Release for 'Beta release' of the FlexRay UNIFIED Driver package	50
September 2006	1.0	Release of the FlexRay UNIFIED Driver package	56
October 2006	1.1	Description of the installation steps in the Chapter 2 and the application example project structure in the Chapter 4 for the MFR4300 FlexRay standalone CC and the MC9S12XFR128 Microcontroller added	64
November 2006	1.2	Description of Low Level Access to memory functions added into the Chapter 3 Support for MPC5561 added - changes in Chapter 2 and Chapter 4 Code, data segment sizes and performance data were added into the Chapter 3	68
December 2006	1.2.1	Corrections necessary for MPC5561 MCU situated on the MPC5567DEMO board are described in Chapter 2	70
December 2006	1.2.2	Corrected description of "Expected values" part in Chapter 3.3.5.2 Corrected description in Chapter 3.2.5.3 Support for WindRiver DIAB C Compiler for MPC5567 and MPC5561 added. New Chapters 2.3 and 4.3 added	72
February 2007	1.2.3	Support for both MFR4300, MFR4310 standalone Communication Controllers for MPC5554 platform for both WindRiver DIAB C Compiler and CodeWarrior Compiler added Support for MFR4310 and S12XDP512 Microcontroller for CodeWarrior Compiler added. New Chapters 2.6, 2.7, 4.6 and 4.7 added Changes in Chapters 2.5 and 4.5	82
March 2007	1.2.4	Support for both MFR4300, MFR4310 standalone Communication Controllers for MPC5554 platform and Green Hills MULTI Project Builder, for MPC5561 platform and Green Hills MULTI Project Builder. New Chapters 2.8, 4.8 added	84
April 2007	1.2.5	Support for WindRiver DIAB C Compiler and Green Hills MULTI Project Builder for MPC5516 added New Chapters 2.9, 2.10 and 4.9, 4.10 Application examples memory requirements and performance were updated - changes in chapter 3.1	92
May 2007	1.2.6	Support for the MC9S12XF512/384/256/128 Microcontrollers for CodeWarrior Compiler added New Chapters 2.11 and 4.11 Application examples memory requirements and performance were updated - changes in chapter 3.1 Updated headings in Chapters 2 and 4	96
October 2008	1.2.7	Support for MPC5567 for CodeWarrior v2.3 added New chapters 2.2 and 4.2 Application examples memory requirements and performance were updated - changes in chapter 3.1	102

Revision History

Date	Revision Level	Description	Page Number(s)
October 2008	1.2.8	Suggested Reading chapter on page 8 changed Target Device chapter on page 9 changed Support for MPC5517 for CodeWarrior v2.3 added New chapters 2.15 and 4.13 Application examples memory requirements and performance were updated - changes in chapter 3.1	110
December 2008	1.2.9	Suggested Reading chapter on page 8 changed Target Device chapter on page 9 changed Support for MPC560xP for the Green Hills MULTI Project Builder added New chapters 4.14 and 2.16 Added MPC560xP to FreeMASTER setting in chapter 2.17 Minor mistakes and typos correction mainly in chapters 4 and 5 Sequence of tasks in chapter 3.2.5.3 changed	114
March 2009	1.3.0	Added support for Green Hills MULTI Project Builder v5.1.6 for MPC560xP	114
January 2010	1.3.1	Added support for CodeWarrior v2.5	118

About this Document

This document describes utilisation of the FlexRay UNIFIED Driver. The driver package is distributed together with the driver source code, two application examples and documentation. This document contains a description of the application examples and describes all the necessary installation steps.

Audience

This document targets software developers of X-by-Wire projects based on the FlexRay communication protocol.

Suggested Reading

[1] FlexRay Protocol Specification V2.1

[2] One of following documents according used FlexRay module:
MFR4300 Block Guide, Freescale Semiconductor
MPC5567 Reference Manual, Freescale Semiconductor
MC9S12XFR128 Data Sheet, Freescale Semiconductor
MPC5561 Reference Manual, Freescale Semiconductor
MFR4310 Block Guide, Freescale Semiconductor
MCU9S12XF-Family Reference Manual, Freescale Semiconductor
MPC5516 Reference Manual, Freescale Semiconductor
MPC5514 Reference Manual, Freescale Semiconductor
MPC5517 Reference Manual, Freescale Semiconductor
MPC5604P Reference Manual, Freescale Semiconductor

[3] FlexRay UNIFIED Driver User Guide, Freescale Semiconductor

Definitions, Acronyms and Abbreviations

The following list defines the acronyms and abbreviations used in this document.

MT ... Macrotick

CC ... Communication Controller

POC ... Protocol Operation Control

MTS ... Media Test Symbol

MB ... Message Buffer

CHI ... Controller Host Interface

CW ... CodeWarrior™

GHS...GreenHills™

Target Device

The FlexRay application examples were developed and tested on the following platforms:

- Freescale MPC5567DEMO board with embedded FlexRay module, using
  - CW Development Studio for Freescale MPC5500 version 1.5
  - CW Development Studio for Freescale MPC5500 version 2.3
  - GreenHills MULTI Project Builder for Freescale MPC5500 version 4.2.1
  - WindRiver DIAB C Compiler version 5.2.1.0
- Freescale MPC5561DEMO board with embedded FlexRay module, using
  - CW Development Studio for Freescale MPC5500 version 1.5
  - GreenHills MULTI Project Builder for Freescale MPC5500 version 4.2.1
  - WindRiver DIAB C Compiler version 5.2.1.0
- Freescale MPC5554DEMO board with the FlexRay MPC55xx adapter board and the MFR4300, MFR4310 Freescale standalone FlexRay Communication Controllers, using
  - CW Development Studio for Freescale MPC5500 version 1.5
  - GreenHills MULTI Project Builder for Freescale MPC5500 version 4.2.1
  - WindRiver DIAB C Compiler version 5.2.1.0
- SofTec HCS12X Starter Kit with the MC9S12XDP512 microcontroller and the MFR4300, MFR4310 Freescale standalone FlexRay Communication Controllers, using
  - CW Development Studio for Freescale MC9S12 version 4.5
- MC9S12XFR128 EVB with embedded FlexRay module, using
  - CW Development Studio for Freescale MC9S12 version 4.5
- Freescale MPC5567DEMO board with connected MPC5561 microcontroller with embedded FlexRay module, using
  - CW Development Studio for Freescale MPC5500 version 1.5
  - GreenHills MULTI Project Builder for Freescale MPC5500 version 4.2.1
  - WindRiver DIAB C Compiler version 5.2.1.0.
- Freescale MPC5516EVB + MPC5516-144QFP Daughter Card with embedded FlexRay module, using
  - GreenHills MULTI Project Builder for Freescale MPC5500 version 5.0.0
  - WindRiver DIAB C Compiler version 5.2.1.0
- EVBS12XF512E with embedded FlexRay module, using
  - CW Development Studio for Freescale MC9S12 version 4.6
- Freescale MPC5516EVB + MPC5516-144QFP Daughter Card with connected MPC5517 microcontroller with embedded FlexRay module, using
  - CW Development Studio for Freescale MPC5500 version 2.3
- Freescale XPC56XX EVB Motherboard with XPC560P 144LQFP Mini-module with connected MPC5604P microcontroller with embedded FlexRay module, using
  - GreenHills MULTI Project Builder for Freescale MPC5500 version 5.0.5
  - GreenHills MULTI Project Builder for PowerPC version 5.1.6

For FlexRay network monitoring, the FreeMASTER tool or various third party tools such as the Decomsys BUSDOCTOR can be used.

Chapter 1 Introduction

The FlexRay UNIFIED Driver isolates the hardware specific functionality into a set of driver functions with a defined Application Program Interface (API) and provides hardware independency for user applications. The FlexRay UNIFIED Driver covers most of the FlexRay module features by available functions, and provides a possible method (called Low Level Access Support) of covering the FlexRay module functionality completely.

The FlexRay UNIFIED Driver package is distributed with two application examples to show possible usage of the FlexRay driver. A short overview of each individual example follows:

1. Transmit Receive application example

   This example shows continuous communication between two FlexRay nodes, where one node operates in the interrupt driven mode, the second node in the poll driven mode. The example covers the use of:

   a. transmit message buffers
   b. receive message buffers
   c. FIFO storage
   d. double transmit message buffers
   e. wakeup mechanism
   f. FlexRay timers
   g. FlexRay interrupts (e.g. cycle start interrupt)
   h. CHI related error, frame transmission across boundary, frame transmission in dynamic segment violation, and internal protocol error checking.

2. Status Monitoring application example

   This example shows usage of the FlexRay UNIFIED Driver to monitor various FlexRay CC states. The example covers the use of:

   a. transmit message buffers
   b. receive message buffers
   c. slot status monitoring
   d. slot status counter monitoring
   e. channel status error monitoring
   f. network management vector
   g. HALT and FREEZE protocol commands
   h. MTS functionality
   i. wakeup mechanism

j. synchronisation state detection

k. FlexRay module re-configuration and reintegration into the FlexRay network.

The FlexRay cluster connectivity based on the FlexRay UNIFIED Driver can be established by using various Freescale FlexRay Communication Controllers, such as the MFR4300 or MFR4310 standalone FlexRay communication controllers, the MC9S12XFR128, MC9S12XF512/384/256/128, MPC5567, MPC5561, MPC5516, MPC5517, MPC560xP microcontrollers with integrated FlexRay modules.

These examples demonstrate transmission and reception in both static and dynamic parts of the communication cycle, the transmit and receive buffer configurations, timer configurations, slot status configuration and slot status counter configuration.

The cycle configuration for both application examples and the observed variables (buffer structures) are shown with their expected values in Chapter 3 Description of the Application Examples.

The file structures of presented application example projects are described in Chapter 4 Application Example Project Structures.

Chapter 2 Installation Steps

The installation steps are slightly different for each compiler used and therefore the description is separated into the individual chapters.

It is possible to define/undefine the *LEDS_USED*, *FREEMASTER_USED* and *SCI0_USED* parameters. The first definition enables the use of the LED's by the application to enable the user to observe the progress of that application. Other two definitions enable the use of the FreeMASTER tool by the application to allow the user to control and visualise the application variables in the FreeMASTER control window.

In the case of the Lauterbach TRACE32 debugger, it is possible to use the predefined *.cmm* files stored in the *trace32_mpc5554, trace32_mpc5567, trace32_mpc5561, trace32_mpc5516 or trace32_mpc5517* folder under each application example.

2.1 CW Development Studio v1.5b2 for the MPC556x MCU

The following description is valid for both MPC5567 and MPC5561 platforms, which are uniformly called MPC556x.

1. Please store the package in a location on your computer and install the FlexRay UNIFIED Driver and application examples
2. Open the CW project file for Node 1 in your CW Development Studio tool:
   a. the *transmit_receive_node1.mcp* file, located in the *fr_freescale_mpc556x/transmit_receive/node1* folder for the Transmit Receive application example
   b. the *status_monitoring_node1.mcp* file, located in the *fr_freescale_mpc556x/status_monitoring/node1* folder for the Status Monitoring application example
3. Compile the application using the *ExtRam* or the *IntFlash* target for the MPC5567 or *IntFlash* target for the MPC5561
4. Download it to your MPC556x board (by use of, e.g. Lauterbach TRACE32 debugger, Nexus PPC) and run the application
5. Open the CW project file for Node 2 in your CW Development Studio tool:
   c. the *transmit_receive_node2.mcp* file, located in the *fr_freescale_mpc556x/transmit_receive/node2* folder for the Transmit Receive application example
   d. the *status_monitoring_node2.mcp* file, located in the *fr_freescale_mpc556x/status_monitoring/node2* folder for the Status Monitoring application example
6. Compile the application using the *ExtRam* or *IntFlash* target for the MPC5567 or the *IntFlash* target for the MPC5561
7. Download it to your MPC556x board (by using a tool, e.g. Lauterbach TRACE32 debugger, Nexus

PPC) and run the application

8. Now, the application is running and FlexRay communication between the nodes should start-up.

Be sure that your nodes are correctly connected together by FlexRay network cables (both channels) and also that the 'grounds' of both nodes are connected together.

If communication does not start, you can try disconnecting the power supply for both nodes and connecting it back again.

Important: To activate the LEDs on the MPC5567DEMO or MPC5561DEMO board, it is necessary to connect GPIO[122], GPIO[126], GPIO[130] pins with the user LEDs by additional wires.

2.1.1 Modifying the Application

Users may change the configuration of the FlexRay module according to their requirements by altering the configuration data in the *Fr_UNIFIED_cfg.c*, *Fr_UNIFIED_cfg.h* and *Fr_UNIFIED_types.h* files, which are located under each CW project in the *cfg* folder.

The *transmit_receive_node1.c, transmit_receive_node2.c, status_monitoring_node1.c* and *status_monitoring_node2.c* files contain the application codes, so the user can arbitrarily modify these files.

The *appconfig.h* file can be also modified to change the MCU settings.

In case that the MPC5561 microcontroller is used with the MPC5567DEMO board, it is necessary to modify the *appconfig.h* file. Please see the instructions below:

1. Extract the *FR_UNIFIED_Driver_Code_and_Example_for_MPC5561_CW.exe* file
2. Open the *appconfig.h* file for the desired example and node in your text editor program. The file is situated e.g. in the *...\status_monitoring\node1\ApplicationConfig* folder
3. Go to the “SIU Configuration section” in the *appconfig.h* file
4. Un-comment the following macros:

   SIU_PCR85_INIT,

   SIU_PCR86_INIT,

   SIU_PCR107_INIT
5. Comment the following macros:

   SIU_PCR134_INIT,

   SIU_PCR135_INIT,

   SIU_PCR137_INIT
6. Save changes.

2.1.2 Used Compiler Platform

CodeWarrior Development Studio MPC55xx v1.5b2

Both application examples were tested with the *ExtRam* and *IntFlash* project targets for the MPC5567 and with the *IntFlash* project target for the MPC5561.

2.2 CW Development Studio v2.3 for the MPC5567 MCU

The following description is valid only for MPC5567.

1. Please store the package in a location on your computer and install the FlexRay UNIFIED Driver and application examples
2. Open the CW project file for Node 1 in your CW Development Studio tool:
   a. the *transmit_receive_node1.mcp* file, located in the *fr_freescale_mpc5567_reva/transmit_receive/node1* folder for the Transmit Receive application example
   b. the *status_monitoring_node1.mcp* file, located in the *fr_freescale_mpc5567_reva/status_monitoring/node1* folder for the Status Monitoring application example
3. Compile the application using the *IntFlash* target
4. Download it to your MPC5567 board (by use of, e.g. Lauterbach TRACE32 debugger, Nexus PPC) and run the application
5. Open the CW project file for Node 2 in your CW Development Studio tool:
   c. the *transmit_receive_node2.mcp* file, located in the *fr_freescale_mpc5567_reva/transmit_receive/node2* folder for the Transmit Receive application example
   d. the *status_monitoring_node2.mcp* file, located in the *fr_freescale_mpc5567_reva/status_monitoring/node2* folder for the Status Monitoring application example
6. Compile the application using the *IntFlash* target
7. Download it to your MPC5567 board (by using a tool, e.g. Lauterbach TRACE32 debugger, Nexus PPC) and run the application
8. Now, the application is running and FlexRay communication between the nodes should start-up.

Be sure that your nodes are correctly connected together by FlexRay network cables (both channels) and also that the 'grounds' of both nodes are connected together.

If communication does not start, you can try disconnecting the power supply for both nodes and connecting it back again.

Important: To activate the LEDs on the MPC5567DEMO board, it is necessary to connect GPIO[122], GPIO[126], GPIO[130] pins with the user LEDs by additional wires.

2.2.1 Modifying the Application

Users may change the configuration of the FlexRay module according to their requirements by altering the configuration data in the *Fr_UNIFIED_cfg.c*, *Fr_UNIFIED_cfg.h* and *Fr_UNIFIED_types.h* files, which are located under each CW project in the *cfg* folder.

The *transmit_receive_node1.c, transmit_receive_node2.c, status_monitoring_node1.c* and *status_monitoring_node2.c* files contain the application codes, so the user can arbitrarily modify these files.

The MCU settings can be changed by modification the ***InitMCU()*** function.

2.2.2 Used Compiler Platform

CodeWarrior Development Studio MPC55xx v2.3

Both application examples were tested with the *IntFlash* project target.

2.3 GreenHills MULTI Project Builder for the MPC556x MCU

The following description is valid for both MPC5567 and MPC5561 platforms, which are uniformly called MPC556x.

1. Please store the package in a location on your computer and install the FlexRay UNIFIED Driver and application examples
2. Open the GreenHills MULTI project file for Node 1 in your GHS MULTI Project Builder tool:
   a. the *transmit_receive_node1.gpj* file, located in the *fr_freescale_mpc556x_GHS/transmit_receive/node1* folder for the Transmit Receive application example
   b. the *status_monitoring_node1.gpj* file, located in the *fr_freescale_mpc556x_GHS/status_monitoring/node1* folder for the Status Monitoring application example
3. Compile the application - the *SystemConfig\startupcode.gpj* file and then the *extram.gpj* or the *intflash.gpj* file for the MPC5567 or *intflash.gpj* target for the MPC5561
4. Download it to your MPC556x board (by use of, e.g. Lauterbach TRACE32 debugger, Nexus PPC) and run the application
5. Open the GreenHills MULTI project file for Node 2 in your GreenHills MULTI Project Builder tool:
   c. the *transmit_receive_node2.gpj* file, located in the *fr_freescale_mpc556x_GHS/transmit_receive/node2* folder for the Transmit Receive application example
   d. the *status_monitoring_node2.gpj* file, located in the *fr_freescale_mpc556x_GHS/status_monitoring/node2* folder for the Status Monitoring application example
6. Compile the application - the *SystemConfig\startupcode.gpj* file and then the *extram.gpj* or the *intflash.gpj* file for the MPC5567 or *intflash.gpj* target for the MPC5561
7. Download it to your MPC556x board (by use of, e.g. Lauterbach TRACE32 debugger, Nexus PPC) and run the application
8. Now, the application is running and FlexRay communication between the nodes should start-up.

Be sure that your nodes are correctly connected together by FlexRay network cables (both channels) and also that the 'grounds' of both nodes are connected together.

If communication does not start, you can try disconnecting the power supply for both nodes and connecting it back again.

Important: To activate the LEDs on the MPC5567DEMO or MPC5561DEMO board, it is necessary to connect GPIO[122], GPIO[126], GPIO[130] pins with the user LEDs by additional wires.

2.3.1 Modifying the Application

Users may change the configuration of the FlexRay module according to their requirements by changing the configuration data in the *Fr_UNIFIED_cfg.c*, *Fr_UNIFIED_cfg.h* and *Fr_UNIFIED_types.h* files, which are located under each GreenHills project in the *cfg* folder.

The *transmit_receive_node1.c, transmit_receive_node2.c, status_monitoring_node1.c* and *status_monitoring_node2.c* files contain the application codes, so the user can arbitrarily modify these files.

The *appconfig.h* file can be also modified to change the MCU settings.

In case that the MPC5561 microcontroller is used with the MPC5567DEMO board, it is necessary to modify the *appconfig.h* file. Please see the instructions below:

1. Extract the *FR_UNIFIED_Driver_Code_and_Example_for_MPC5561_GHS.exe* file
2. Open the *appconfig.h* file for the desired example and node in your text editor program. The file is situated e.g. in the *...\status_monitoring\node1\ApplicationConfig* folder
3. Go to the “SIU Configuration section” in the *appconfig.h* file
4. Un-comment the following macros:

   SIU_PCR85_INIT,

   SIU_PCR86_INIT,

   SIU_PCR107_INIT
5. Comment the following macros:

   SIU_PCR134_INIT,

   SIU_PCR135_INIT,

   SIU_PCR137_INIT
6. Save changes.

2.3.2 Used Compiler Platform

GreenHills MULTI Project Builder v4.2.1

Both application examples were tested with the *extram* and the *intflash* project targets for the MPC5567 and with the *intflash* project target for the MPC5561.

2.4 WindRiver DIAB C Compiler for the MPC556x MCU

The following description is valid for both MPC5567 and MPC5561 platforms, which are uniformly called MPC556x.

1. Please store the package in a location on your computer and install the FlexRay UNIFIED Driver and application examples
2. Change the “DIAB_DATA_SOURCE*”* variable in the *makefile* file in case that the variable does not point at your DIAB compiler installation folder. The *makefile* file is situated in the root directory for each node (e.g. *...\Fr_freescale_mpc5567_DIAB\transmit_receive\node1* directory)
3. Compile the applications for both Node1 and Node 2 - run *runmake.bat* files located in the root directory for each node
4. Download the compiled code (*intflash.mot* file situated in the *bin* folder) to your MPC556x boards e.g. by use of the eSys Flasher program

5. Press reset buttons to start the application
6. Now, the application is running and FlexRay communication between the nodes should start-up.

Be sure that your nodes are correctly connected together by FlexRay network cables (both channels) and also that the 'grounds' of both nodes are connected together.

If communication does not start, you can try disconnecting the power supply for both nodes and connecting it back again.

Important: To activate the LEDs on the MPC5567DEMO or MPC5561DEMO board, it is necessary to connect GPIO[122], GPIO[126], GPIO[130] pins with the user LEDs by additional wires.

2.5 GreenHills MULTI Project Builder for the MPC556x MCU

The following description is valid for both MPC5567 and MPC5561 platforms, which are uniformly called MPC556x.

1. Please store the package in a location on your computer and install the FlexRay UNIFIED Driver and application examples
2. Open the GreenHills MULTI project file for Node 1 in your GHS MULTI Project Builder tool:
   a. the *transmit_receive_node1.gpj* file, located in the *fr_freescale_mpc556x_GHS/transmit_receive/node1* folder for the Transmit Receive application example
   b. the *status_monitoring_node1.gpj* file, located in the *fr_freescale_mpc556x_GHS/status_monitoring/node1* folder for the Status Monitoring application example
3. Compile the application - the *SystemConfig\startupcode.gpj* file and then the *extram.gpj* or the *intflash.gpj* file for the MPC5567 or *intflash.gpj* target for the MPC5561
4. Download it to your MPC556x board (by use of, e.g. Lauterbach TRACE32 debugger, Nexus PPC) and run the application
5. Open the GreenHills MULTI project file for Node 2 in your GreenHills MULTI Project Builder tool:
   c. the *transmit_receive_node2.gpj* file, located in the *fr_freescale_mpc556x_GHS/transmit_receive/node2* folder for the Transmit Receive application example
   d. the *status_monitoring_node2.gpj* file, located in the *fr_freescale_mpc556x_GHS/status_monitoring/node2* folder for the Status Monitoring application example
6. Compile the application - the *SystemConfig\startupcode.gpj* file and then the *extram.gpj* or the *intflash.gpj* file for the MPC5567 or *intflash.gpj* target for the MPC5561
7. Download it to your MPC556x board (by use of, e.g. Lauterbach TRACE32 debugger, Nexus PPC) and run the application
8. Now, the application is running and FlexRay communication between the nodes should start-up.

Be sure that your nodes are correctly connected together by FlexRay network cables (both channels) and also that the 'grounds' of both nodes are connected together.

If communication does not start, you can try disconnecting the power supply for both nodes and connecting it back again.

Important: To activate the LEDs on the MPC5567DEMO or MPC5561DEMO board, it is necessary to connect GPIO[122], GPIO[126], GPIO[130] pins with the user LEDs by additional wires.

2.5.1 Modifying the Application

Users may change the configuration of the FlexRay module according to their requirements by changing the configuration data in the *Fr_UNIFIED_cfg.c*, *Fr_UNIFIED_cfg.h* and *Fr_UNIFIED_types.h* files, which are located under each GreenHills project in the *cfg* folder.

The *transmit_receive_node1.c, transmit_receive_node2.c, status_monitoring_node1.c* and *status_monitoring_node2.c* files contain the application codes, so the user can arbitrarily modify these files.

The *appconfig.h* file can be also modified to change the MCU settings.

In case that the MPC5561 microcontroller is used with the MPC5567DEMO board, it is necessary to modify the *appconfig.h* file. Please see the instructions below:

1. Extract the *FR_UNIFIED_Driver_Code_and_Example_for_MPC5561_GHS.exe* file
2. Open the *appconfig.h* file for the desired example and node in your text editor program. The file is situated e.g. in the *...\status_monitoring\node1\ApplicationConfig* folder
3. Go to the “SIU Configuration section” in the *appconfig.h* file
4. Un-comment the following macros:

   SIU_PCR85_INIT,

   SIU_PCR86_INIT,

   SIU_PCR107_INIT
5. Comment the following macros:

   SIU_PCR134_INIT,

   SIU_PCR135_INIT,

   SIU_PCR137_INIT
6. Save changes.

2.5.2 Used Compiler Platform

GreenHills MULTI Project Builder v4.2.1

Both application examples were tested with the *extram* and the *intflash* project targets for the MPC5567 and with the *intflash* project target for the MPC5561.

2.6 WindRiver DIAB C Compiler for the MPC556x MCU

The following description is valid for both MPC5567 and MPC5561 platforms, which are uniformly called MPC556x.

1. Please store the package in a location on your computer and install the FlexRay UNIFIED Driver and application examples
2. Change the “DIAB_DATA_SOURCE” variable in the *makefile* file in case that the variable does not point at your DIAB compiler installation folder. The *makefile* file is situated in the root directory for each node (e.g. ...*Fr_freescale_mpc5567_DIAB**transmit_receive**node1* directory)
3. Compile the applications for both Node1 and Node 2 - run *runmake.bat* files located in the root directory for each node
4. Download the compiled code (*intflash.mot* file situated in the *bin* folder) to your MPC556x boards e.g. by use of the eSys Flasher program
5. Press reset buttons to start the application
6. Now, the application is running and FlexRay communication between the nodes should start-up.

Be sure that your nodes are correctly connected together by FlexRay network cables (both channels) and also that the 'grounds' of both nodes are connected together.

If communication does not start, you can try disconnecting the power supply for both nodes and connecting it back again.

Important: To activate the LEDs on the MPC5567DEMO or MPC5561DEMO board, it is necessary to connect GPIO[122], GPIO[126], GPIO[130] pins with the user LEDs by additional wires.

2.6.1 Modifying the Application

Users may change the configuration of the FlexRay module according to their requirements by changing the configuration data in the *Fr_UNIFIED_cfg.c*, *Fr_UNIFIED_cfg.h* and *Fr_UNIFIED_types.h* files, which are located under each DIAB project in the *cfg* folder.

The *transmit_receive_node1.c, transmit_receive_node2.c, status_monitoring_node1.c* and *status_monitoring_node2.c* files contain the application codes, so the user can arbitrarily modify these files.

The *appconfig.h* file can be also modified to change the MCU settings.

In case that the MPC5561 microcontroller is used with the MPC5567DEMO board, it is necessary to modify the *appconfig.h* file. Please see the instructions below:

1. Extract the *FR_UNIFIED_Driver_Code_and_Example_for_MPC5561_DIAB.exe* file
2. Open the *appconfig.h* file for the desired example and node in your text editor program. The file is situated e.g. in the ...*status_monitoring**node1**ApplicationConfig* folder
3. Go to the “SIU Configuration section” in the *appconfig.h* file
4. Un-comment the following macros:

   SIU_PCR85_INIT,

   SIU_PCR86_INIT,

   SIU_PCR107_INIT
5. Comment the following macros:

SIU_PCR134_INIT,
SIU_PCR135_INIT,
SIU_PCR137_INIT

Save changes.

2.6.2 Used Compiler Platform

WindRiver DIAB C compiler version 5.2.1.0

Both application examples were tested with the *intflash* project target for both MPC5567 and MPC5561 microcontrollers.

2.7 CW Development Studio for the MC9S12XFR128 MCU

1. Please store the package in a location on your computer and install the FlexRay UNIFIED Driver and application examples
2. Open the CW project file for Node 1 in your CW Development Studio tool:
   a. the *transmit_receive_node1.mcp* file, located in the *fr_freescale_s12xfr/transmit_receive/node1* folder for the Transmit Receive application example
   b. the *status_monitoring_node1.mcp* file, located in the *fr_freescale_s12xfr/status_monitoring/node1* folder for the Status Monitoring application example
3. Compile the application
4. Download it to your MC9S12XFR128 board (by use of, e.g. P&E USB Multilink) and run the application
5. Open the CW project file for Node 2 in your CW Development Studio tool:
   c. the *transmit_receive_node2.mcp* file, located in the *fr_freescale_s12xfr/transmit_receive/node2* folder for the Transmit Receive application example
   d. the *status_monitoring_node2.mcp* file, located in the *fr_freescale_s12xfr/status_monitoring/node2* folder for the Status Monitoring application example
6. Compile the application
7. Download it to your MC9S12XFR128 board (by use of, e.g. P&E USB Multilink) and run the application
8. Now, the application is running and FlexRay communication between the nodes should start-up.

Be sure that your nodes are correctly connected together by FlexRay network cables (both channels) and also that the 'grounds' of both nodes are connected together.

If communication does not start, you can try disconnecting the power supply for both nodes and connecting it back again.

2.7.1 Modifying the Application

Users may change the configuration of the FlexRay module according to their requirements by altering the configuration data in the *Fr_UNIFIED_cfg.c*, *Fr_UNIFIED_cfg.h* and *Fr_UNIFIED_types.h* files, which are located under each CW project in the *cfg* folder.

The *transmit_receive_node1.c, transmit_receive_node2.c, status_monitoring_node1.c* and *status_monitoring_node2.c* files contain the application codes, so the user can arbitrarily modify these files.

2.7.2 Used Compiler Platform

CodeWarrior Development Studio for Freescale MC9S12 version 4.5

Both application examples were tested with *P&E ICD* project target.

2.8 CW Development Studio for the MC9S12XDP512 MCU with MFR43x0 CC

The following description for the MC9S12XDP512 MCU with a standalone communication controller is valid for both MFR4300 and MFR4310 communication controllers, which are uniformly called MFR43x0.

1. Please store the package in a location on your computer and install the FlexRay UNIFIED Driver and application examples
2. Open the CW project file for Node 1 in your CW Development Studio tool:
   a. the *transmit_receive_node1.mcp* file, located in the *fr_freescale_s12x_mfr43x0/transmit_receive/node1* folder for the Transmit Receive application example
   b. the *status_monitoring_node1.mcp* file, located in the *fr_freescale_s12x_mfr43x0/status_monitoring/node1* folder for the Status Monitoring application example
3. Compile the application using the *P&E Multilink CyclonePro* or *SofTec HCS12* target
4. Download it to your MC9S12XDP512 board (by use of, e.g. P&E USB Multilink) and run the application
5. Open the CW project file for Node 2 in your CW Development Studio tool:
   c. the *transmit_receive_node2.mcp* file, located in the *fr_freescale_s12x_mfr43x0/transmit_receive/node2* folder for the Transmit Receive application example
   d. the status_monitoring_node2.mcp file, located in the *fr_freescale_s12x_43x0/status_monitoring/node2* folder for the Status Monitoring application example
6. Compile the application using the *P&E Multilink CyclonePro* or *SofTec HCS12* target
7. Download it to your MC9S12XDP512 board (by use of, e.g. P&E USB Multilink) and run the application
8. Now, the application is running and FlexRay communication between the nodes should start-up.

Be sure that your nodes are correctly connected together by FlexRay network cables (both channels) and also that the 'grounds' of both nodes are connected together.

If communication does not start, you can try disconnecting the power supply for both nodes and connecting it back again.

2.8.1 Modifying the Application

Users may change the configuration of the FlexRay module according to their requirements by altering the configuration data in the *Fr_UNIFIED_cfg.c*, *Fr_UNIFIED_cfg.h* and *Fr_UNIFIED_types.h* files, which are located under each CW project in the *cfg* folder.

The *transmit_receive_node1.c, transmit_receive_node2.c, status_monitoring_node1.c* and *status_monitoring_node2.c* files contain the application codes, so the user can arbitrarily modify these files.

2.8.2 Used Compiler Platform

CodeWarrior Development Studio for Freescale MC9S12 version 4.5

Both application examples were tested with *P&E Multilink CyclonePro* project target.

2.9 CW Development Studio for the MPC5554 MCU with MFR43x0 CC

The following description for the MPC5554 MCU with a standalone communication controller is valid for both MFR4300 and MFR4310 communication controllers, which are uniformly called MFR43x0.

1. Please store the package in a location on your computer and install the FlexRay UNIFIED Driver and application examples
2. Open the CW project file for Node 1 in your CW Development Studio tool:
   a. the *transmit_receive_node1.mcp* file, located in the *fr_freescale_mpc5554_mfr43x0/transmit_receive/node1* folder for the Transmit Receive application example
   b. the *status_monitoring_node1.mcp* file, located in the *fr_freescale_mpc5554_mfr43x0/status_monitoring/node1* folder for the Status Monitoring application example
3. Compile the application using the *IntRam* or *IntFlash* target
4. Download it to your MPC5554 board (by use of, e.g. Lauterbach TRACE32 debugger, Nexus PPC) and run the application
5. Open the CW project file for Node 2 in your CW Development Studio tool:
   c. the *transmit_receive_node2.mcp* file, located in the *fr_freescale_mpc5554_mfr43x0/transmit_receive/node2* folder for the Transmit Receive application example
   d. the status_monitoring_node2.mcp file, located in the *fr_freescale_mpc5554_mfr43x0/status_monitoring/node2* folder for the Status Monitoring application example
6. Compile the application using the *IntRam* or *IntFlash* target
7. Download it to your MPC5554 board (by use of, e.g. Lauterbach TRACE32 debugger, Nexus PPC) and run the application
8. Now, the application is running and FlexRay communication between the nodes should start-up.

Be sure that your nodes are correctly connected together by FlexRay network cables (both channels) and also that the 'grounds' of both nodes are connected together.

If communication does not start, you can try disconnecting the power supply for both nodes and connecting it back again.

2.9.1 Modifying the Application

Users may change the configuration of the FlexRay module according to their requirements by altering the configuration data in the *Fr_UNIFIED_cfg.c*, *Fr_UNIFIED_cfg.h* and *Fr_UNIFIED_types.h* files, which are located under each CW project in the *cfg* folder.

The *transmit_receive_node1.c, transmit_receive_node2.c, status_monitoring_node1.c* and *status_monitoring_node2.c* files contain the application codes, so the user can arbitrarily modify these files.

NOTE

The small change of the code in the FLEXRAY_ISR routine is necessary for the application examples intended for the MPC5554 platform with a MFR43x0 Communication Controller. The MPC55xx family uses external IRQ pins as edge detect pins. To ensure that all incoming FlexRay interrupts are serviced, the Fr_interrupt_handler() routine has to be repeatedly called while the external IRQ line is in active state. See the application examples for the MPC5554 platform with a MFR43x0 Communication Controller.

The *appconfig.h* file can be also modified to change the MCU settings.

2.9.2 Used Compiler Platform

CodeWarrior Development Studio MPC55xx v1.5b2

Both application examples were tested with the *IntRam* and *IntFlash* project targets.

2.10 WindRiver DIAB C Compiler for the MPC5554 MCU with MFR43x0 CC

The following description for the MPC5554 MCU with a standalone communication controller is valid for both MFR4300 and MFR4310 communication controllers, which are uniformly called MFR43x0.

1. Please store the package in a location on your computer and install the FlexRay UNIFIED Driver and application examples
2. Change the “DIAB_DATA_SOURCE*”* variable in the *makefile* file in case that the variable does not point at your DIAB compiler installation folder. The *makefile* file is situated in the root directory for each node (e.g. *...\Fr_freescale_mpc5554_mfr43x0_DIAB\transmit_receive\node1* directory)
3. Compile the applications for both Node1 and Node 2 - run *runmake.bat* files located in the root directory for each node
4. Download the compiled code (*intflash.mot* file situated in the *bin* folder) to your MPC5554 boards e.g. by use of the eSys Flasher program
5. Press reset buttons to start the application
6. Now, the application is running and FlexRay communication between the nodes should start-up.

Be sure that your nodes are correctly connected together by FlexRay network cables (both channels) and also that the 'grounds' of both nodes are connected together.

If communication does not start, you can try disconnecting the power supply for both nodes and connecting it back again.

2.10.1 Modifying the Application

Users may change the configuration of the FlexRay module according to their requirements by changing the configuration data in the *Fr_UNIFIED_cfg.c*, *Fr_UNIFIED_cfg.h* and *Fr_UNIFIED_types.h* files, which are located under each DIAB project in the *cfg* folder.

The *transmit_receive_node1.c, transmit_receive_node2.c, status_monitoring_node1.c* and *status_monitoring_node2.c* files contain the application codes, so the user can arbitrarily modify these files.

NOTE

The small change of the code in FLEXRAY_ISR routine is necessary for the application examples intended for the MPC5554 platform with a MFR43x0 Communication Controller. The MPC55xx family uses external IRQ pins as edge detect pins. To ensure that all incoming FlexRay interrupts are serviced, the Fr_interrupt_handler() routine has to be repeatedly called while the external IRQ line is in active state. See the application examples for the MPC5554 platform with a MFR43x0 Communication Controller.

The *appconfig.h* file can be also modified to change the MCU settings.

2.10.2 Used Compiler Platform

WindRiver DIAB C compiler version 5.2.1.0

Both application examples were tested with the *intflash* project target.

2.11 GreenHills MULTI Project Builder for the MPC5554 MCU with MFR43x0 CC

The following description for the MPC5554 MCU with a standalone communication controller is valid for both MFR4300 and MFR4310 communication controllers, which are uniformly called MFR43x0.

1. Please store the package in a location on your computer and install the FlexRay UNIFIED Driver and application examples
2. Open the GreenHills MULTI project file for Node 1 in your GHS MULTI Project Builder tool:
   a. the *transmit_receive_node1.gpj* file, located in the *fr_freescale_mpc5554_mfr43x0_GHS/transmit_receive/node1* folder for the Transmit Receive application example
   b. the *status_monitoring_node1.gpj* file, located in the *fr_freescale_mpc5554_mfr43x0_GHS/status_monitoring/node1* folder for the Status Monitoring application example
3. Compile the application - the *SystemConfig\startupcode.gpj* file and then the *intflash.gpj* file
4. Download it to your MPC5554 board (by use of, e.g. Lauterbach TRACE32 debugger, Nexus PPC)

and run the application

5. Open the GreenHills MULTI project file for Node 2 in your GreenHills MULTI Project Builder tool:
   c. the *transmit_receive_node2.gpj* file, located in the *fr_freescale_mpc5554_mfr43x0_GHS/transmit_receive/node2* folder for the Transmit Receive application example
   d. the *status_monitoring_node2.gpj* file, located in the *fr_freescale_mpc5554_mfr43x0_GHS/status_monitoring/node2* folder for the Status Monitoring application example
6. Compile the application - the *SystemConfig\startupcode.gpj* file and then the *intflash.gpj* file
7. Download it to your MPC5554 board (by use of, e.g. Lauterbach TRACE32 debugger, Nexus PPC) and run the application
8. Now, the application is running and FlexRay communication between the nodes should start-up.

Be sure that your nodes are correctly connected together by FlexRay network cables (both channels) and also that the 'grounds' of both nodes are connected together.

If communication does not start, you can try disconnecting the power supply for both nodes and connecting it back again.

2.11.1 Modifying the Application

Users may change the configuration of the FlexRay module according to their requirements by changing the configuration data in the *Fr_UNIFIED_cfg.c*, *Fr_UNIFIED_cfg.h* and *Fr_UNIFIED_types.h* files, which are located under each GreenHills project in the *cfg* folder.

The *transmit_receive_node1.c, transmit_receive_node2.c, status_monitoring_node1.c* and *status_monitoring_node2.c* files contain the application codes, so the user can arbitrarily modify these files.

NOTE

The small change of the code in FLEXRAY_ISR routine is necessary for the application examples intended for the MPC5554 platform with a MFR43x0 Communication Controller. The MPC55xx family uses external IRQ pins as edge detect pins. To ensure that all incoming FlexRay interrupts are serviced, the Fr_interrupt_handler() routine has to be repeatedly called while the external IRQ line is in active state. See the application examples for the MPC5554 platform with a MFR43x0 Communication Controller.

The *appconfig.h* file can be also modified to change the MCU settings.

2.11.2 Used Compiler Platform

GreenHills MULTI Project Builder v4.2.1

Both application examples were tested with the *intflash* project target.

2.12 WindRiver DIAB C Compiler for the MPC5516 MCU

1. Please store the package in a location on your computer and install the FlexRay UNIFIED Driver and application examples

2. Change the “DIAB_DATA_SOURCE” variable in the *makefile* file in case that the variable does not point at your DIAB compiler installation folder. The *makefile* file is situated in the root directory for each node (e.g. ...*Fr_freescale_mpc5516_DIAB**transmit_receive**node1* directory)
3. Compile the applications for both Node1 and Node 2 - run *runmake.bat* files located in the root directory for each node
4. Download the compiled code (*intflash.mot* file situated in the *bin* folder) to your MPC5516 board e.g. by the TRACE32 program go to the File/Run Batchfile/programflash.cmm
5. Press reset buttons to start the application
6. Now, the application is running and FlexRay communication between the nodes should start-up.

Be sure that your nodes are correctly connected together by FlexRay network cables (both channels) and also that the 'grounds' of both nodes are connected together.

If communication does not start, you can try disconnecting the power supply for both nodes and connecting it back again.

Important: To activate the LEDs on the MPC5516 board, it is necessary to connect GPIO[121], GPIO[122], GPIO[123] pins with the user LEDs by additional wires.

2.12.1 Modifying the Application

Users may change the configuration of the FlexRay module according to their requirements by changing the configuration data in the *Fr_UNIFIED_cfg.c*, *Fr_UNIFIED_cfg.h* and *Fr_UNIFIED_types.h* files, which are located under each DIAB project in the *cfg* folder.

The *transmit_receive_node1.c, transmit_receive_node2.c, status_monitoring_node1.c* and *status_monitoring_node2.c* files contain the application codes, so the user can arbitrarily modify these files.

2.12.2 Used Compiler Platform

WindRiver DIAB C compiler version 5.2.1.0

Both application examples were tested with the *intflash* and the *intram* project targets.

2.13 GreenHills MULTI Project Builder for the MPC5516 MCU

1. Please store the package in a location on your computer and install the FlexRay UNIFIED Driver and application examples
2. Open the GreenHills MULTI project file for Node 1 in your GHS MULTI Project Builder tool:
   a. the *transmit_receive_node1.gpj* file, located in the *fr_freescale_mpc5516_GHS/transmit_receive/node1* folder for the Transmit Receive application example
   b. the *status_monitoring_node1.gpj* file, located in the *fr_freescale_mpc5516_GHS/status_monitoring/node1* folder for the Status Monitoring application example
3. Compile the application - the *SystemConfig**startupcode.gpj* file and then the *inflash.gpj* or the *intram.gpj* file
4. Download it to your MPC5516 board (by use of, e.g. Lauterbach TRACE32 debugger) and run the

application

5. Open the GreenHills MULTI project file for Node 2 in your GreenHills MULTI Project Builder tool:
   c. the *transmit_receive_node2.gpj* file, located in the *fr_freescale_mpc5516_GHS/transmit_receive/node2* folder for the Transmit Receive application example
   d. the *status_monitoring_node2.gpj* file, located in the *fr_freescale_mpc5516_GHS/status_monitoring/node2* folder for the Status Monitoring application example
6. Compile the application - the *SystemConfig\startupcode.gpj* file and then the *intflash.gpj* or the *intram.gpj* file
7. Download it to your MPC5516 board (by use of, e.g. Lauterbach TRACE32 debugger) and run the application
8. Now, the application is running and FlexRay communication between the nodes should start-up.

Be sure that your nodes are correctly connected together by FlexRay network cables (both channels) and also that the 'grounds' of both nodes are connected together.

If communication does not start, you can try disconnecting the power supply for both nodes and connecting it back again.

Important: To activate the LEDs on the MPC5516 board, it is necessary to connect GPIO[121], GPIO[122], GPIO[123] pins with the user LEDs by additional wires.

2.13.1 Modifying the Application

Users may change the configuration of the FlexRay module according to their requirements by changing the configuration data in the *Fr_UNIFIED_cfg.c*, *Fr_UNIFIED_cfg.h* and *Fr_UNIFIED_types.h* files, which are located under each GreenHills project in the *cfg* folder.

The *transmit_receive_node1.c, transmit_receive_node2.c, status_monitoring_node1.c* and *status_monitoring_node2.c* files contain the application codes, so the user can arbitrarily modify these files.

The *appconfig.h* file can be also modified to change the MCU settings.

2.13.2 Used Compiler Platform

GreenHills MULTI Project Builder v5.0.0

Both application examples were tested with the *intflash* and the *intram* project targets.

2.14 CW Development Studio for the MC9S12XF512/384/256/128 MCU

1. Please store the package in a location on your computer and install the FlexRay UNIFIED Driver and application examples
2. Open the CW project file for Node 1 in your CW Development Studio tool:
   a. the *transmit_receive_node1.mcp* file, located in the *fr_freescale_s12xf/transmit_receive/node1* folder for the Transmit Receive application example

   b. the *status_monitoring_node1.mcp* file, located in the *fr_freescale_s12xf/status_monitoring/node1* folder for the Status Monitoring application example
3. Compile the application
4. Download it to your MC9S12XF board (by use of, e.g. P&E USB Multilink) and run the application
5. Open the CW project file for Node 2 in your CW Development Studio tool:
   c. the *transmit_receive_node2.mcp* file, located in the *fr_freescale_s12xf/transmit_receive/node2* folder for the Transmit Receive application example
   d. the *status_monitoring_node2.mcp* file, located in the *fr_freescale_s12xf/status_monitoring/node2* folder for the Status Monitoring application example
6. Compile the application
7. Download it to your MC9S12XF board (by use of, e.g. P&E USB Multilink) and run the application
8. Now, the application is running and FlexRay communication between the nodes should start-up.

Be sure that your nodes are correctly connected together by FlexRay network cables (both channels) and also that the 'grounds' of both nodes are connected together.

If communication does not start, you can try disconnecting the power supply for both nodes and connecting it back again.

2.14.1 Modifying the Application

Users may change the configuration of the FlexRay module according to their requirements by altering the configuration data in the *Fr_UNIFIED_cfg.c*, *Fr_UNIFIED_cfg.h* and *Fr_UNIFIED_types.h* files, which are located under each CW project in the *cfg* folder.

The *transmit_receive_node1.c, transmit_receive_node2.c, status_monitoring_node1.c* and *status_monitoring_node2.c* files contain the application codes, so the user can arbitrarily modify these files.

2.14.2 Used Compiler Platform

CodeWarrior Development Studio for Freescale MC9S12 version 4.6

Both application examples were tested with *P&E Multilink CyclonePro* project target.

2.15 CW Development Studio v2.3 for the MPC5517 MCU

The following description is valid only for MPC5517.

1. Please store the package in a location on your computer and install the FlexRay UNIFIED Driver and application examples
2. Open the CW project file for Node 1 in your CW Development Studio tool:
   a. the *transmit_receive_node1.mcp* file, located in the *fr_freescale_mpc5517/transmit_receive/node1* folder for the Transmit Receive application example

b. the *status_monitoring_node1.mcp* file, located in the *fr_freescale_mpc5517/status_monitoring/node1* folder for the Status Monitoring application example

3. Compile the application using the *IntFlash* target
4. Download it to your MPC5510 board (by use of e.g. Lauterbach TRACE32 debugger, Nexus PPC) and run the application
5. Open the CW project file for Node 2 in your CW Development Studio tool:
   c. the *transmit_receive_node2.mcp* file, located in the *fr_freescale_mpc5517/transmit_receive/node2* folder for the Transmit Receive application example
   d. the *status_monitoring_node2.mcp* file, located in the *fr_freescale_mpc5517/status_monitoring/node2* folder for the Status Monitoring application example
6. Compile the application using the *IntFlash* target
7. Download it to your MPC5510 board (by using of e.g. Lauterbach TRACE32 debugger, Nexus PPC) and run the application
8. Now, the application is running and FlexRay communication between the nodes should start-up.

Be sure that your nodes are correctly connected together by FlexRay network cables (both channels) and also that the 'grounds' of both nodes are connected together.

If communication does not start, you can try disconnecting the power supply for both nodes and connecting it back again.

Important: To activate the LEDs on the MPC5510 board, it is necessary to connect PB[0], PB[1], PB[2] pins with the user LEDs by additional wires.

2.15.1 Modifying the Application

Users may change the configuration of the FlexRay module according to their requirements by altering the configuration data in the *Fr_UNIFIED_cfg.c*, *Fr_UNIFIED_cfg.h* and *Fr_UNIFIED_types.h* files, which are located under each CW project in the *cfg* folder.

The *transmit_receive_node1.c, transmit_receive_node2.c, status_monitoring_node1.c* and *status_monitoring_node2.c* files contain the application codes, so the user can arbitrarily modify these files.

The MCU settings can be changed by modification the ***InitMCU()*** function.

2.15.2 Used Compiler Platform

CodeWarrior Development Studio MPC55xx v2.3

Both application examples were tested with the *IntFlash* project target.

2.16 GreenHills MULTI Project Builder for the MPC560xP MCU

The following description is valid for MPC560xP platform only. Please store the package in a location on your computer and install the FlexRay UNIFIED Driver and application examples

1. Open the GreenHills MULTI project file for Node 1 in your GHS MULTI Project Builder tool:
   a. the *transmit_receive_node1.gpj* file, located in the *Fr_freescale_MPC560xP_GHS/transmit_receive/node1* folder for the Transmit Receive application example
   b. the *status_monitoring_node1.gpj* file, located in the *Fr_freescale_MPC560xP_GHS/status_monitoring/node1* folder for the Status Monitoring application example
2. Compile the application - the the *intflash.gpj* file for the MPC560xP.
3. Download it to your MPC560xP board (by use of, e.g. Lauterbach TRACE32 debugger, Nexus PPC) and run the application
4. Open the GreenHills MULTI project file for Node 2 in your GreenHills MULTI Project Builder tool:
   c. the *transmit_receive_node2.gpj* file, located in the *fr_freescale_MPC560xP_GHS/transmit_receive/node2* folder for the Transmit Receive application example
   d. the *status_monitoring_node2.gpj* file, located in the *fr_freescale_MPC560x_GHS/status_monitoring/node2* folder for the Status Monitoring application example
5. Compile the application - the the *intflash.gpj* file for the MPC560xP.
6. Download it to your MPC560xP board (by use of, e.g. Lauterbach TRACE32 debugger, Nexus PPC) and run the application
7. Now, the application is running and FlexRay communication between the nodes should start-up.

Be sure that your nodes are correctly connected together by FlexRay network cables (both channels) and also that the 'grounds' of both nodes are connected together.

If communication does not start, you can try disconnecting the power supply for both nodes and connecting it back again.

Important: The switches jumpers (header J9) on the MPC560xP board must be removed to avoid disturbance of the FlexRay communication by their additional capacity. To activate the LEDs on the MPC560xP EVB Motheboard, it is necessary to plug in jumpers 2, 3 and 4 on J7 header.

2.16.1 Modifying the Application

Users may change the configuration of the FlexRay module according to their requirements by changing the configuration data in the *Fr_UNIFIED_cfg.c*, *Fr_UNIFIED_cfg.h* and *Fr_UNIFIED_types.h* files, which are located under each GreenHills project in the *cfg* folder.

The *transmit_receive_node1.c, transmit_receive_node2.c, status_monitoring_node1.c* and *status_monitoring_node2.c* files contain the application codes, so the user can arbitrarily modify these files.

2.16.2 Used Compiler Platform

GreenHills MULTI Project Builder v5.0.5 or GreenHills MULTI Project Builder v5.1.6

Both application examples were tested with the *intflash* project target for the MPC5604P.

2.17 CW Development Studio v2.5 for the MPC560xP MCU

The following description is valid only for MPC560xP.

1. Please store the package in a location on your computer and install the FlexRay UNIFIED Driver and application examples
2. Open the CW project file for Node 1 in your CW Development Studio tool:
   a. the *transmit_receive_node1.mcp* file, located in the *Fr_freescale_MPC560xP_CW/transmit_receive/node1* folder for the Transmit Receive application example
   b. the *status_monitoring_node1.mcp* file, located in the *Fr_freescale_MPC560xP_CW/status_monitoring/node1* folder for the Status Monitoring application example
3. Compile the application using the *internal_FLASH* target
4. Download it to your MPC560xP board (by use of e.g. Lauterbach TRACE32 debugger, Nexus PPC) and run the application
5. Open the CW project file for Node 2 in your CW Development Studio tool:
   c. the *transmit_receive_node2.mcp* file, located in the *Fr_freescale_MPC560xP_CW/transmit_receive/node2* folder for the Transmit Receive application example
   d. the *status_monitoring_node2.mcp* file, located in the *Fr_freescale_MPC560xP_CW/status_monitoring/node2* folder for the Status Monitoring application example
6. Compile the application using the *internal_FLASH* target
7. Download it to your MPC560xP board (by using of e.g. Lauterbach TRACE32 debugger, Nexus PPC) and run the application
8. Now, the application is running and FlexRay communication between the nodes should start-up.

Be sure that your nodes are correctly connected together by FlexRay network cables (both channels) and also that the 'grounds' of both nodes are connected together.

If communication does not start, you can try disconnecting the power supply for both nodes and connecting it back again.

Important: The switches jumpers (header J9) on the MPC560xP board must be removed to avoid disturbance of the FlexRay communication by their additional capacity. To activate the LEDs on the MPC560xP EVB Motheboard, it is necessary to plug in jumpers 2, 3 and 4 on J7 header.

2.17.1 Modifying the Application

Users may change the configuration of the FlexRay module according to their requirements by altering the configuration data in the *Fr_UNIFIED_cfg.c*, *Fr_UNIFIED_cfg.h* and *Fr_UNIFIED_types.h* files, which are located under each CW project in the *cfg* folder.

The *transmit_receive_node1.c, transmit_receive_node2.c, status_monitoring_node1.c* and *status_monitoring_node2.c* files contain the application codes, so the user can arbitrarily modify these files.

The MCU settings can be changed by modification the ***InitMCU()*** function.

2.17.2 Used Compiler Platform

CodeWarrior Development Studio MPC55xx v2.5

Both application examples were tested with the *Internal_FLASH* project target.

2.18 Installation of the FreeMASTER tool

To use the FreeMASTER tool, it is necessary to have the FreeMASTER tool installed on your PC (version 1.2.39 or later is required). The installation file can be found at http://www.freescale.com. The FreeMASTER application can run on any computer with a Microsoft Windows 98 or later operating system. Before installing, the Internet Explorer 4.0 or higher (5.5 is recommended) should be installed.

The user should perform the following installation steps:

a. Run the executable file by clicking on the *fmaster12-39.exe* file

b. Follow the instructions on the screen

For demonstration of an application example, two FreeMASTER screens (one for each node) should be opened. The FreeMASTER project files are located for different platforms and compilers in the *fr_freescale_'platform'_'compiler' / FreeMASTER* folders (e.g. in the *fr_freescale_mpc5567_GHS / FreeMASTER* folder). Please be sure that the *FREEMASTER_USED* parameter is defined in the application example codes.

The user should perform the following steps for running and configurating the FreeMASTER application (be sure that the RS232 cable is properly connected to the PC).

The FreeMASTER connection between host the PC and the node is described below. The user should execute the following steps for both nodes separately

1. Open appropriate *fr_freescale_'platform'_'compiler' / FreeMASTER* folder (e.g. the *fr_freescale_mpc5567_GHS / FreeMASTER* folder)
2. Click on the *transmit_receive_node_1.pmp,* the *transmit_receive_node_2.pmp*, the *status_monitoring_node_1.pmp* or the *status_monitoring_node_1.pmp* file to open the FreeMASTER project file for the required node observation
3. Users might encounter the following warning messages 'Could not open the communication port', 'can not detect the board information' or 'Missing symbol definition'. Click on the *Ok*, *Close port* and/or *Continue* buttons
4. The FreeMASTER project settings must be updated to reflect the local user COM port settings
   a. Select the proper COM port in the *Project / Options* menu, on the *Comm* tab
   b. Set the communication speed to:
      i. 19200Bd for an application example running on the MC9S12XFR128 or MFR43x0/MC9S12XDP512 platforms
      ii. 115200Bd for an application example running on the MPC55xx or MPC560xP platform
   c. Clear the 'Save settings to registry' check box
5. Click on the *OK* button
6. Save the FreeMASTER project settings by clicking on the *File / Save project*
7. Start communication by clicking on the *File / Start communication* (or click the *STOP* icon at the top left)

8. Reload the symbol file by clicking on the *Project/Reload Symbol File* (or press Ctrl+M)
9. Now the FreeMASTER tool should be connected to the required node.

Chapter 3 Description of the Application Examples

Both the Transmit Receive and Status Monitoring application examples use the same configurations for general cluster settings. A short overview is extracted in section Table 3-1 FlexRay Configuration Data - General Cluster Parameters.

Table 3-1 FlexRay Configuration Data - General Cluster Parameters

General parameters	
Bit rate	10 Mbit/s
Maximum number of sync nodes	5
Macrotick length	1us
Cycle length	5000 MT
Static segment	3000 MT
Dynamic segment	880 MT
Symbol window	13 MT
Offset correction start	4920 MT
TSS transmitter	11
Static segment	
Number of static slots	60
Static slot length	50 MT
Action point offset	3 MT
Payload length static	16 x 16 bit
Dynamic segment	
Maximum number of minislots	22
Minislot length	40 MT
Minislot action point offset	3 MT
Payload length dynamic max	8 x 16 bit

3.1 Application Examples Memory Requirements and Performance

The UNIFIED Driver code has different memory requirements for various platforms and compilers. These memory requirements are written in the Table 3-2.

Table 3-2 UNIFIED Driver Memory Requirements

Compiler	Platform	Code Size [Bytes]
GreenHills	MPC556x	26288*/14900**
	MPC5554/MFR43x0	26288*/14900**
	MPC5516	24372*/14600**
	MPC560xP	17996*/11284**

Table 3-2 UNIFIED Driver Memory Requirements

WindRiver DIAB C	MPC556x	24064#/17548##
	MPC5554/MFR43x0	24064#/17548##
	MPC5516	24064#/17548##
CodeWarrior v1.5b2	MPC556x	24812x
	MPC5554/MFR43x0	24812x
	MC9S12XFR128	9945xx
	MC9S12XDP512/MFR43x0	10778xx
	MC9S12XF512	9945xx
CodeWarrior v2.3	MPC5567	16664$
	MPC5517	16994$

* Note, “No Optimizations” switch option set in GreenHills v4.2.1 or v5.0.0 for MPC5516 or v5.1.6 for MPC560xP
** Note, “Optimize for General Use” switch option set in GreenHills v4.2.1 or v5.0.0 for MPC5516 or v5.1.6 for MPC560xP
# Note, not optimized in WindRiver DIAB C Compiler
Note, optimized with the following options “-XO” and “-Xsize-opt” in WindRiver DIAB C Compiler
x Note, optimizations are set to “Off” in CodeWarrior Development Studio MPC55xx v 1.5b2
xx Note, all optimizations are enabled in CodeWarrior Development Studio for Freescale MC9S12 v4.5 or v4.6 for MC9S12XF
$ Note, optimizations are set to “Off” in CodeWarrior Development Studio MPC5567 v 2.3

The Table 3-3 summarizes application examples memory requirements for different platforms and compilers.

NOTE

The application examples were compiled without FreeMASTER support

NOTE

Each of four application examples uses different number of the FlexRay UNIFIED Driver functions, thus the downloaded code will have various sizes for different application examples

Table 3-3 Application Examples Memory Requirements

Compiler	Platform		Transmit Receive		Slot Status	
			Node 1 [Bytes]	Node 2 [Bytes]	Node 1 [Bytes]	Node 2 [Bytes]
GreenHills*	MPC556x	UNIFIED Driver - Code/Data	10036/248	9564/388	11924/280	12488/236
		Sample program - Code/Data	1444/92	1328/144	1772/156	1864/120
	MPC5554/ MFR43x0	UNIFIED Driver - Code/Data	10036/248	9564/388	11924/280	12488/236
		Sample program - Code/Data	1500/92	1384/144	1828/156	1920/120
	MPC5516	UNIFIED Driver - Code/Data	10008/248	9504/388	11772/280	12260/236
		Sample program - Code/Data	1580/88	1420/140	1848/152	1960/116
	MPC560xP	UNIFIED Driver - Code/Data	7968/256	7652/396	9130/292	9496/248
		Sample program - Code/Data	1984/88	1892/140	2226/152	2292/116

Table 3-3 Application Examples Memory Requirements

WindRiver DIAB C[#]	MPC556x	Sample program - Code/Data	12332/256	11792/396	14308/288	14908/244
		Sample program - Code/Data	1264/118	1172/173	1600/187	1668/170
	MPC5554/ MFR43x0	UNIFIED Driver - Code/Data	12332/256	11792/396	14308/288	14908/244
		Sample program - Code/Data	1352/118	1260/173	1688/187	1756/170
	MPC5516	UNIFIED Driver - Code/Data	12332/256	11792/396	14308/288	14908/244
		Sample program - Code/Data	1440/118	1340/173	1776/187	1844/170
CodeWarrior v1.5b2[x]	MPC556x	UNIFIED Driver - Code/Data	17492/248	16756/320	20564/280	21384/240
		Sample program - Code/Data	1528/100	1612/149	2096/195	2192/141
	MPC5554/ MFR43x0	UNIFIED Driver - Code/Data	17492/248	16756/320	20564/280	21384/240
		Sample program - Code/Data	1544/100	1628/149	2112/195	2208/141
CodeWarrior[xx]	MC9S12XFR128	UNIFIED Driver - Code/Data	7310/167	5734/246	8246/185	8519/160
		Sample program - Code/Data	631/102	615/154	851/166	900/134
	MC9S12XDP512/ MFR43x0	UNIFIED Driver - Code/Data	7786/168	6143/247	8847/186	9119/161
		Sample program - Code/Data	621/102	618/154	861/166	910/134
CodeWarrior[xx]	MC9S12XF512	UNIFIED Driver - Code/Data	7310/167	5734/246	8246/185	8519/160
		Sample program - Code/Data	682/102	669/154	902/166	951/134
CodeWarrior v2.3[$]	MPC5567	UNIFIED Driver - Code/Data	11946/248	9368/306	13918/260	14384/258
		Sample program - Code/Data	1302/97	1094/148	1696/160	1766/128
	MPC5517	UNIFIED Driver - Code/Data	12142/248	9584/306	14102/260	14568/258
		Sample program - Code/Data	1368/97	1162/148	1762/160	1832/128

[*] Note, “Optimize for General Use” switch option set in GreenHills v4.2.1 or v5.0.0 for MPC5516 or v5.1.6 for MPC560xP
[#] Note, optimized with the following options “-XO” and “-Xsize-opt” in WindRiver DIAB C Compiler
[x] Note, optimizations are set to “Off” in CodeWarrior Development Studio MPC55xx v 1.5b2
[xx] Note, all optimizations are enabled in CodeWarrior Development Studio for Freescale MC9S12 v4.5 or v4.6 for MC9S12XF
[$] Note, optimizations are set to “Off” in CodeWarrior Development Studio MPC5567 v 2.3

3.1.1 UNIFIED Driver Performance Data

The comparison of the UNIFIED Driver CPU usage for different platforms was done. For this purpose the Transmit Receive Application Example Node1 was used. This node runs in the interrupt driven mode, see section 3.2.4 Detailed Description of Application Software for Node 1. The Table 3-4 shows total CPU time needed to handle the interrupts incoming from the FlexRay module in one communication cycle. The CPU usage represents the time spent in the FlexRay related interrupts according to the following equation:

$$CPU_{Load} = T_{Interrupt} / T_{Cycle} * 100[\%] \quad (3\text{-}1)$$

where:

CPU_{Load} is the CPU load in percent;
$T_{Interrupt}$ is the amount of time spent in all FlexRay interrupts;
T_{Cycle} is the time of FlexRay communication cycle.

Table 3-4 UNIFIED Driver CPU usage

Platform	Conditions	CPU_{Load} time [%]
MPC5561 and MPC5567 (embedded FlexRay module)	f_{system} = 120 [MHz]	0.45[x]
MPC5516 (embedded FlexRay module)	f_{system} = 80 [MHz]	2.99[x]
MPC5554 with standalone MFR4310 CC	f_{system} = 120 [MHz]*	0.67[x]
MC9S12XFR128 (embedded FlexRay module)	f_{bus}= 40 [MHz]	2.30[xx]
MC9S12XDP512 with standalone MFR4300 CC	f_{bus}= 25 [MHz]**	3.96[xx]
MC9S12XF512 (embedded FlexRay module)	f_{bus}= 40 [MHz]	2.30[xx]
MPC560xP	f_{system} = 80 [MHz]	1.215[$]

* Note, three stretch cycles are used
** Note, five wait states in single cycle were used (see appconfig.h file)
x Note, optimized with the following options “-XO” and “-Xsize-opt” in WindRiver DIAB C Compiler
xx Note, all optimizations are enabled in CodeWarrior Development Studio for Freescale MC9S12 v4.5 or v4.6 for MC9S12XF
$ Note, “Optimize for General Use” switch option set in GreenHills v5.1.6 for MPC560xP

3.2 Transmit Receive Application Example

The first example shows continuous communication between two FlexRay nodes (called Node 1 and Node 2), where one node operates in the interrupt driven mode, and the second node in the poll driven mode.

The application software:

- configures the FlexRay module with an appropriate FlexRay schedule and timing, see section section 3.2.1 FlexRay Schedule and Timing
- configures receive and transmit double message buffers, allowing transmission and reception of FlexRay messages, and enables message buffer interrupts
- configures FIFO storage and enables FIFO interrupt
- configures one absolute timer and one relative timer and enables the timer interrupts
- transmits wake-up symbols
- configures FlexRay *cycle start* interrupt
- establishes FlexRay communication between the two nodes - they are configured as *coldstart nodes (*see *[FR_PROTOCOL]* for details*)*
- updates transmit message buffers with new data when a *message buffer* interrupt occurs
- gets a global cluster time when a timer 1, timer 2 or *cycle start* interrupt arises
- stores the slot status of the received frame when a message buffer interrupt arises
- updates a data array, which will be transmitted in the next communication cycle, with the received header indexes when a FIFO interrupt occurs.

3.2.1 FlexRay Schedule and Timing

Nodes are configured with the cluster configuration data specified in section Table 3-1 FlexRay Configuration Data - General Cluster Parameters. Message buffers are configured for transmitting and receiving as shown in section Table 3-5 Slot Assignment.

Table 3-5 Slot Assignment

Slot	Transmitting	Receiving	FIFO receiving
Slot assignment - Static segment			
Slot 1	Node 1, channel A and B, startup and sync frame	Node 2, channel A	
Slot 4	Node 2, channel A and B, startup and sync frame	Node 1, channel A	
Slot 5	Node 2, channel A and B		
Slot assignment - Dynamic segment			
Slot 62	Node 2, channel A		Node 1, channel A
Slot 63	Node 2, channel A		Node 1, channel A

The application example shows transmission in the static and also in the dynamic part of the communication cycle. One of the static slots of each node is also used simultaneously as a synchronisation/start-up and a data slot. The FlexRay communication cycle period is 5 ms.

Node 1 sends data in the 1st slot, Node 2 sends data in the 4th, 5th, 62nd and 63rd slots of the communication cycle. This configuration is shown in section Figure 3-1. Configuration of the FlexRay Communication Cycle for Transmit Receive Application Example.

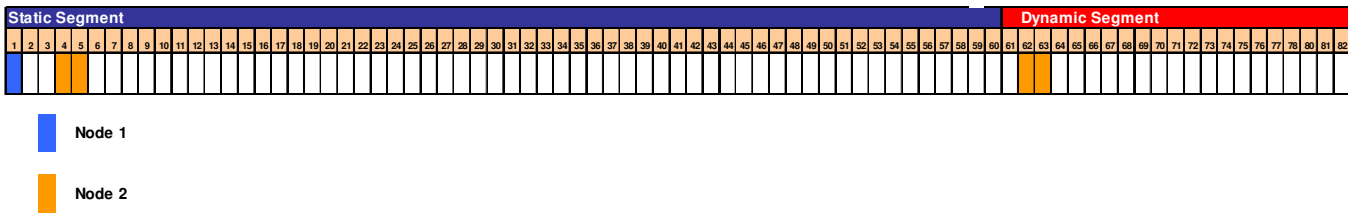

Figure 3-1. Configuration of the FlexRay Communication Cycle for Transmit Receive Application Example

The transmit and receive message buffers are configured as described in section Table 3-7 Message Buffers Configuration for Node 1 and section Table 3-11 Message Buffers Configuration for Node 2.

The host application uses two individual message buffer segments. All message buffers configured for transmission or reception in the static part of the communication cycle belong to the first segment (segment 1), transmit message buffers configured for transmission in the dynamic part of the communication cycle belong to the second segment (segment 2). Assignment of the message buffer number to message buffer segments is shown in section Table 3-6 Connection Among the Message Buffers and Segments.

Table 3-6 Connection Among the Message Buffers and Segments

Type of Segment		MB Number		Data size [Words]
		From	To	
Individual MB	Segment 1	0	10	16
	Segment 2	11	18	8
FIFO A (configured only for Node 1)		19	28	8
FIFO B		Not configured		

This application example also demonstrates the FlexRay module on-chip timers. Both absolute *timer 1* and relative *timer 2* operate in an interrupt-driven mechanism. A service routine is called when the timers reach their limits.

3.2.2 Configuration for Node 1

Static segment:

Transmission: Slot 1

Reception: Slot 4

Dynamic segment:

FIFO reception: Slot 62, Slot 63

Table 3-7 Message Buffers Configuration for Node 1

Segment	Static		Dynamic	
Slot number	1	4	62	63
Channels	A,B	A	A	A
Assigned Tx message buffer	0, 1			
Assigned Rx message buffer		3	FIFO A (from 19 to 28)	FIFO A (from 19 to 28)
Message buffer type (only Tx)	Double			
Payload length (only Tx) [words]	16		8	8
Interrupt enabled	At transmit side of double MB	Yes	Yes	Yes
Name of interrupt service routine	CC_interrupt_slot_1	CC_interrupt_slot_4	CC_interrupt_FIFO_A	CC_interrupt_FIFO_A
Data update (only Tx)	Each cycle			

Table 3-7 Message Buffers Configuration for Node 1

Name of data array used for data updating or storing	tx_data_1	rx_data_4	fifo_data	fifo_data

For correct frame reception, it is necessary to configure the Shadow message buffers as shown in section Table 3-8 Shadow Message Buffers Configuration for Node 1. For more information on the Shadow message buffer concept, please see [FLEXRAY_MODULE], chapter Message Buffer Types.

Table 3-8 Shadow Message Buffers Configuration for Node 1

Segment	Segment 1		Segment 2	
Channel	A	B	A	B
Assigned buffer index	8	9	17	18

The message buffer segments are configured according to the section Table 3-6 Connection Among the Message Buffers and Segments as shown in section Table 3-9 Message Buffer Segments Configuration for Node 1.

Table 3-9 Message Buffer Segments Configuration for Node 1

Parameter	Configured value
Data size for segment 1 [W]	16
Data size for segment 2 [W]	8
The number of MB's in segment 1	11
Last MB in segment 1	10
The number of MB's in segment 2	8
Last individual MB	18
FIFO depth	10
Total number of used MB's (Last individual MB + 1 + FIFO depth)	29

FlexRay timers are configured as shown in section Table 3-10 FlexRay Timers Configuration for Node 1.

Table 3-10 FlexRay Timers Configuration for Node 1

Timer	T1	T2
Timer timebase	Absolute	Relative
Repetitive mode	Repetitive	Repetitive
Macrotick offset	2050	1000000
Filter mask	0	-
Filter value	0	-

All configuration information is stored in the *Fr_UNIFIED_cfg.c* file as section Chapter 4 Application Example Project Structures describes.

General cluster configuration data is stored in the *Fr_low_level_cfg_set_00* structure (defined as type *Fr_low_level_config_type*).

Message buffer configuration data is stored in *the Fr_buffer_cfg_00[]* array of structures (defined as type *Fr_buffer_info_type*). Message buffers to be configured are selected by means of the *Fr_buffer_cfg_set_00[]* array of indexes (defined as type *Fr_index_selector_type*). This array must have the FR_LAST_MB (0xFF) value stored in the last item for correct the FlexRay UNIFIED Driver operations.

Shadow buffers configuration data is stored in the *Fr_rx_shadow_cfg* structure (defined as type *Fr_receive_shadow_buffers_config_type*).

The references on configuration data for timers are stored in the *Fr_timers_cfg_00_ptr[]* array (defined as type *Fr_timer_config_type*); this array has to be ended by a NULL pointer for correct driver function. Timer 1 configuration data is stored in the *Fr_timer_1_cfg* structure and Timer 2 in the *Fr_timer_2_cfg* structure.

3.2.3 Configuration for Node 2

Static segment:

Transmission: Slot 4, Slot 5

Reception: Slot 1

Dynamic segment:

Transmission: Slot 62, Slot 63

Table 3-11 Message Buffers Configuration for Node 2

Segment	Static			Dynamic	
Slot number	1	4	5	62	63
Channels	A	A, B	A, B	A	A
Assigned Tx message buffer		1	4, 5	11	12
Assigned Rx message buffer	2				
Message buffer type (only Tx)		Single	Double	Single	Single
Payload length (only Tx) [words]		16	16	8	8
Interrupt enabled	No	No	No	No	No
Name of interrupt service routine					
Data update (only Tx)		Each cycle	Each cycle	Each cycle	Each cycle
Name of data array used for data updating or storing	rx_data_1	tx_data_4	tx_data_5	tx_data_62	tx_data_63

For correct frame reception, it is necessary to configure the Shadow message buffers as shown in section Table 3-12 Shadow Message Buffers Configuration for Node 2. For more information on the Shadow message buffer concept, please see [FLEXRAY_MODULE], chapter Message Buffer Types.

Table 3-12 Shadow Message Buffers Configuration for Node 2

Segment	Segment 1		Segment 2	
Channel	A	B	A	B
Assigned buffer index	8	9	17	18

The message buffer segments are configured according to section Table 3-6 Connection Among the Message Buffers and Segments as shown in section Table 3-13 Message Buffer Segments Configuration for Node 2.

Table 3-13 Message Buffer Segments Configuration for Node 2

Parameter	Configured value
Data size for segment 1 [W]	16
Data size for segment 2 [W]	8
Last MB in segment 1	10
Last individual MB	18
FIFO depth	Not configured
Total number of used MB's (Last individual MB + 1 + FIFO depth)	19

All configuration information is stored in the *Fr_UNIFIED_cfg.c* file as section Chapter 4 Application Example Project Structures describes.

General cluster configuration data is stored in the *Fr_low_level_cfg_set_00* structure (defined as type *Fr_low_level_config_type*).

Message buffer configuration data is stored in the *Fr_buffer_cfg_00[]* array of structures (defined as type *Fr_buffer_info_type*). Message buffers to be configured are selected by means of the *Fr_buffer_cfg_set_00[]* array of indexes (defined as type *Fr_index_selector_type*). This array must have the FR_LAST_MB (0xFF) value stored in the last item for correct the FlexRay UNIFIED Driver operation.

Shadow buffers configuration data is stored in the *Fr_rx_shadow_cfg* structure (defined as type *Fr_receive_shadow_buffers_config_type*).

3.2.4 Detailed Description of Application Software for Node 1

Node 1 is configured to transmit and receive data in the interrupt driven mode.

3.2.4.1 Initialization procedure

After the microcontroller initialization by the *InitMCU*() function, the FlexRay module is initialized by means of the following functions:

- the ***Fr_init(****&Fr_HW_cfg_00, &Fr_low_level_cfg_set_00)* function. The first parameter references the hardware configuration structure and the second parameter points to the low level parameter configuration structure. The ***Fr_init()*** function should be called only once during runtime, even if a FlexRay module is reconfigured. This function stores the base address of the FlexRay module, configures the channels, resets the FlexRay module and forces the module into the *POC:config* state
- the ***Fr_set_configuration(****&Fr_HW_cfg_00, &Fr_low_level_cfg_set_00)* function is called with the same input parameters as the previous function. This function initializes the data structures used by the FlexRay UNIFIED Driver and configures low level parameters (e.g. cluster configuration)
- the ***Fr_buffers_init(****&Fr_buffer_cfg_00[0], &Fr_buffer_cfg_set_00[0])* function; the first parameter points to the array of message buffer structures and the second parameters references the array with indexes to determine which message buffers defined in the *Fr_buffer_cfg_00[]* array will be used for FlexRay module configuration. This approach allows the use of more configurations by several configuration sets (several *Fr_buffer_cfg_set_xx[]* arrays) and only one array of structures (one *Fr_buffer_cfg_00[]* array)
- the ***Fr_timers_init(****&Fr_timers_cfg_00_ptr)* function for initializing of the absolute and relative timers. The input parameter points to the array with timer configuration structures.

As the next step, the interrupt callback functions are set, for:

- a message buffer interrupt by calling the ***Fr_set_MB_callback()*** functions. The first parameter is the address of the function which is called when an interrupt arises, the second parameter determines for which message buffer the callback function will be set. The driver stores the address of the callback function into its internal structures and when an interrupt occurs, this function is called by the ***Fr_interrupt_handler()*** function
- an interrupt caused by Protocol Interrupts, precisely by the *Protocol Interrupt Flag Register 0 (PIFR0).* For more information on Protocol Interrupts, please see [FLEXRAY_MODULE]. The ***Fr_set_protocol_0_IRQ_callback()*** functions are called with two parameters:
  - the address of the callback function
  - the type of protocol interrupt for which the interrupt callback function will be configured
- a FIFO interrupt by calling of the ***Fr_set_fifo_IRQ_callback(****&CC_interrupt_FIFO_A, FR_FIFO_A_IRQ)* function. The input parameters have the same meaning as in the previous cases.

After FlexRay module initialization, it is convenient to call the ***Fr_leave_configuration_mode()*** function to exit the *POC:config* state.

As soon as the FlexRay module reaches the *POC:ready* state, the ***Fr_send_wakeup()*** function is called to send the wakeup pattern over channel A in case if no wakeup pattern has been received yet (the ***Fr_get_wakeup_state()*** is called). Subsequential calls of the ***Fr_get_POC_state()*** function are necessary to wait on the *POC:ready* state (after the *POC:wakeup* state).

Once the FlexRay module is at *POC:ready*, the ***Fr_start_communication()*** function can be used to initialize of the transition from *POC:ready* to *POC:startup* state. If the start up of the module is successful, the state *POC:normal active* is automatically reached.

The transmit message buffer is initialized (at the first update of the MB, the buffer is committed to a transmission) by the ***Fr_transmit_data(****TX_SLOT_1, &tx_data_1[0], 16)* function. The first parameter determines the message buffer used, the second parameter points to the array with the data to be transmitted, and the third parameter gives the length of the data.

The ***Fr_get_wakeup_state()*** function returns the outcome of the executing of the wakeup mechanism and the state is stored in the *wakeup_status* variable.

Since the application example for Node 1 runs in an interrupt driven mode, the ***Fr_enable_interrupts(****(FR_PROTOCOL_IRQ | FR_FIFO_A_IRQ | FR_RECEIVE_IRQ | FR_TRANSMIT_IRQ), (FR_TIMER_1_EXPIRED_IRQ | FR_TIMER_2_EXPIRED_IRQ | FR_CYCLE_START_IRQ)*, 0**)** function enables the FlexRay interrupts according to the given values in the input parameters.
Interrupt services provided by the FlexRay UNIFIED Driver are strictly connected to the *Global Interrupt Flag and Enable Register (GIFER)*, the *Protocol Interrupt Enable Register 0 (PIER0)* and the *Protocol Interrupt Enable Register 1(PIER1)*. This means that the first parameter controls the interrupt request lines of the *GIFER* register, the second parameter controls the *PIER0* register, and the third parameter the *PIER1* register.

The application example operates with two timers, the absolute timer T1 and the relative timer T2. The timers are started by means of the ***Fr_start_timer(****FR_TIMER_T1**)* and ***Fr_start_timer(****FR_TIMER_T2**)* functions. When a *timer* interrupt occurs, the function *CC_interrupt_timer_1()* or *CC_interrupt_timer_2()* is called by the ***Fr_interrupt_handler()***.

3.2.4.2 Interrupt services

When an interrupt arises the ***Fr_interrupt_handler()*** routine is called, and if a callback function has been configured for the related interrupt, this callback function is called. A short description of each function is below:

- the *CC_interrupt_slot_1()* routine should be called in each cycle after message buffer 0 (configured for slot 1) has been transmitted. The ***Fr_transmit_data(****TX_SLOT_1, &tx_data_1[0], 16)* function updates message buffer 0 with new data provided by reference to the *tx_data_1[]* array. If the MB has been successfully updated, the first element of the data array (*tx_data_1[0]*) is incremented.
  Expected values:
  – *tx_data_1[0]* increased by 1 in each cycle from 0 to 0xFFFF, then wraps to 0

- the *CC_interrupt_slot_4()* routine should be called in each cycle after a valid non-null frame is received by receive message buffer 3 (configured for slot 4).
  The ***Fr_receive_data(****buffer_idx, &rx_data_4[0], &rx_data_length, &rx_status_slot**)* function stores received data from the MB given by the *buffer_idx* parameter into the *rx_data_4[]* array, stores the length of received payload data into the *rx_data_length* variable, and stores the slot status of the received frame into the *rx_status_slot* variable. The *buffer_idx* value is given by the ***Fr_interrupt_handler()*** function and determines which MB generated the interrupt.
  In the following lines of the application code, the value of the slot status is copied into the second array element (*tx_data_1[1]*) (the transmit MB will be updated with this data payload during the next cycle), and the third array element (*tx_data_1[2])* is incremented. This approach allows monitoring the slot status and the frequency of *receive* MB interrupts by means of the FlexRay bus monitoring tool.
  Expected values:

  - *buffer_idx* equal to 3
  - *tx_data_1[1]* equal to 0x00F0 (receive MB is configured for reception on channel A)
  - *tx_data_1[2]* increased by 1 in each cycle from 0 to 0xFFFF, then wraps to 0

- the *CC_interrupt_timer_1()* routine is called when a timer interrupt from absolute timer T1 occurs, i.e. in 2050 MT of the communication cycle.
  The ***Fr_get_global_time(****&current_cycle, &current_macrotick****)*** function gets the current values of the macrotick and cycle. These values are then stored into the elements *tx_data_1[14]* and *tx_data_1[15]* of the data payload array, to be transmitted later on.
  Expected values:
  - *tx_data_1[14]* from 2050 to 2060
  - *tx_data_1[15]* increased by 1 in each cycle from 0 to 63

- the *CC_interrupt_timer_2()* routine is called when a timer interrupt from relative timer T2 occurs, i.e. approximately each second - after 1 million of macroticks
  The ***Fr_get_global_time(****&current_cycle, &current_macrotick****)*** function gets the current values of the macrotick and cycle. These values are then stored into the elements *tx_data_1[12]* and *tx_data_1[13]* of the data payload array, to be transmitted later on.
  Expected values:
  - *tx_data_1[12]* arbitrary from 0 to 5000
  - *tx_data_1[13]* arbitrary from 0 to 63

- the *CC_interrupt_cycle_start()* routine is called when a cycle start protocol interrupt occurred, i.e. each cycle at the beginning.
  The ***Fr_get_global_time(****&current_cycle, &current_macrotick****)*** function gets the current values of the macrotick and cycle. The cycle value is then stored into the element *tx_data_1[11]* of the data payload array to be transmitted later on.
  Expected value:
  - *tx_data_1[11]* increased by 1 in each cycle from 0 to 63

- the CC_interrupt_FIFO_A() routine should be called twice in each cycle when FIFO A is not empty. The FIFO A storage is configured to receive only those frames with slot number between 60 and 64, therefore, one interrupt should occur after receiving a frame in 62nd slot and the second interrupt after receiving a frame in the 63rd slot.
  The ***Fr_receive_fifo_data(****header_idx, &fifo_data[0], &fifo_data_length, &fifo_slot_idx, &fifo_status_slot****)*** function stores
  - received data from FIFO into the *fifo_data[]* array
  - the length of the received payload data into the *fifo_data_length* variable
  - the received frame ID into the *fifo_slot_idx* variable
  - the slot status of the received frame into the *fifo_status_slot* variable.

  The *header_idx* value is given by the ***Fr_interrupt_handler()*** function and determines which next available receive FIFO MB can be read.
  Where data has been received into FIFO storage, the following operations are executed:
  - the *header_idx* value is copied as the fourth data array element (*tx_data_1[3])* to be transmitted later on
  - the first item value of the received payload data (*fifo_data[0]*) is copied as the fifth array element (*tx_data_1[4])*

– according to the slot number of the received frame, the application:
  i. stores a *0xF0F3* error value into the array element *tx_data_1[5]* if the slot number equals 4 - this means that the FIFO received a frame in the 4th slot and this situation should never occur because the FIFO is not configured to receive this frame
  ii. stores a *0xF0F2* error value into the array element *tx_data_1[6]* if the slot number equals 5 - this means that the FIFO received a frame in the 5th slot and this situation should never occur
  iii. increments the array element *tx_data_1[7]* by one when the slot number equals 62
  iv. increments the array element *tx_data_1[8]* by one when the slot number equals 63
  v. stores a *0xF0F0 error* value into the array element *tx_data_1[9]* if the slot number does not equal any previously described case - this situation should never occur.

Where data has not been received by the FIFO storage, the application stores a *0xFFFF* error value into the array element *tx_data_1[10]*. This situation should never arise.

Expected values:
– *tx_data_1[3]* from 19 to 28
– *tx_data_1[4]* arbitrary from 0 to 0xFFFF, the value is updated after receiving the frame in the 62nd slot, however, this is almost immediately replaced by payload data from the frame in the 63rd slot (and this value is sent over the FlexRay bus later on)
– *tx_data_1[5]* equal to 0
– *tx_data_1[6]* equal to 0
– *tx_data_1[7]* incremented by 1 in each cycle from 0 to 0xFFFF, then wraps back to 0
– *tx_data_1[8]* incremented by 1 in each cycle from 0 to 0xFFFF, then wraps back to 0
– *tx_data_1[9]* equal to 0
– *tx_data_1[10]* equal to 0

3.2.5 Detailed Description of Application Software for Node 2

Node 2 is defined to transmit and receive data in the poll driven mode.

3.2.5.1 Initialization procedure

The initialization sequence is the same as for Node 1 (see section 3.2.4.1 Initialization procedure) except for enabling the FlexRay interrupts by the ***Fr_enable_interrupts()*** function and setting the callback functions by means of the ***Fr_set_MB_callback()***, ***Fr_set_protocol_0_IRQ_callback()*** and ***Fr_set_fifo_IRQ_callback()*** routines.

3.2.5.2 Interrupt services

No FlexRay interrupts are enabled.

3.2.5.3 Application code

In never-ending *for()* loop, the ***Fr_check_cycle_start(****&current_cycle****)*** function is called to determine whether the communication cycle has been started or not. The address of the *current_cycle* variable is given as an output parameter to store the current cycle number. The following tasks are performed where the cycle has been started:

- The received data is copied from receive message buffer 2 (configured for slot 1).
  The ***Fr_check_rx_status(****RX_SLOT_1)* function is called to check the status of the receive message buffer determined by the input parameter. In case that a valid non-null frame has been received in the last matching slot into receive message buffer 2 (*FR_RECEIVED* returned value):
  – the ***Fr_receive_data(****RX_SLOT_1, &rx_data_1[0], &rx_data_length, &rx_status_slot)* function stores received data from the MB given by the first parameter into the *rx_data_1[]* array, stores the length of the received payload data into the *rx_data_length* variable, and stores the slot status of the received frame into the *rx_status_slot* variable
  – the second element of the transmit data array for slot 5 (*tx_data_5[1]*) is incremented (the transmit MB will be updated with this data payload in the next cycle)
  – the slot status information is stored into the third array element (*tx_data_5[2])*. This approach allows monitor the slot status and the frequency of frame reception by means of the FlexRay bus monitoring tool.
  – received data is from the previous cycle

  Expected values:
  – tx_data_5[1] incremented by 1 in each cycle from 0 to 0xFFFF, then wraps back to 0
  – *tx_data_5[2]* equal to 0x00F0 (the receive MB is configured for reception on channel A)

- The transmit single message buffer 1 is updated (configured for slot 4).
  The ***Fr_check_tx_status(****TX_SLOT_4)* function is called to check the status of the transmit single buffer determined by the input parameter. Where the data has been transmitted (*FR_TRANSMITTED* returned value):
  – the first element of the data array (*tx_data_4[0]*) is decremented
  – the ***Fr_transmit_data(****TX_SLOT_4, &tx_data_4[0], 16)* function updates the message buffer 1 with new data provided by reference to the *tx_data_4[]* array.

  Expected values:
  – *tx_data_4[0]* decremented by 100 in each cycle from 0xFFFF to 0, then wraps back to 0xFFFF

- The transmit double message buffer 4 is updated (configured for slot 5).
  The ***Fr_check_tx_status(****TX_SLOT_5)* function is called to check the status of the transmit double buffer determined by the input parameter. Where an Internal Message Transfer has been performed (*FR_INTERNAL_MESSAGE_TRANSFER_DONE* returned value), or data has been transmitted from the transmit side of the double MB (*FR_TRANSMITTED* returned value):
  – the first element of the data array (*tx_data_5[0]*) is incremented
  – the ***Fr_transmit_data(****TX_SLOT_5, &tx_data_5[0], 16)* function updates message buffer 4 with new data provided by reference to the *tx_data_5[]* array.

  Expected values:
  – *tx_data_5[0]* incremented by 100 in each cycle from 0 to 0xFFFF, then wraps back to 0

- The transmit message buffer 11 is updated (configured for slot 62 - dynamic segment).
  The ***Fr_check_tx_status(****TX_SLOT_62)* function is called to check the status of the transmit single buffer determined by the input parameter. In case that data has been transmitted (*FR_TRANSMITTED* returned value):

  - the first element of the data array (*tx_data_62[0]*) is incremented
  - the ***Fr_transmit_data(****TX_SLOT_62, &tx_data_62[0], 0)* function updates message buffer 11 with new data provided by reference to the *tx_data_62[]* array. The FlexRay UNIFIED Driver determines the data payload length from message buffer configuration registers when the *data length* input parameter equals zero. This approach makes sense for a transmit MB configured for the dynamic segment of the communication cycle.

  Expected values:
  - *tx_data_62[0]* increased by 50 in each cycle from 0 to 0xFFFF, then wraps back to 0

- The fourth element of the transmit data array for slot 5 (*tx_data_5[3]*) is incremented (the transmit MB will be updated with this data payload during the next cycle)
  Expected values:
  - *tx_data_5[3]* incremented by 1 in each cycle from 0 to 0xFFFF, then wraps back to 0

- The current cycle number is stored into the fifth array element (*tx_data_5[4])*. This approach allows monitoring of the slot status and the frequency of frame reception by means of the FlexRay bus monitoring tool
  Expected values:
  - *tx_data_5[4]* incremented by 1 in each cycle from 0 to 63

- The status of the CHI related error flags are obtained by means of the ***Fr_check_CHI_error()*** function. In case that any error is detected on channel A or B, the *chi_error* variable is incremented and then it is stored to the seventh element of the transmit data array for slot 5 (*tx_data_5[6]*).
  Expected values:
  - *tx_data_5[6]* equal to 0

- The ***Fr_check_transmission_across_boundary(****FR_CHANNEL_AB)* function is called to check whether or not a frame transmission across boundary has occurred on channel A or B. In such a case, the *transmission_across_boundary* variable is incremented and then stored to the eighth element of the transmit data array for slot 5 (*tx_data_5[7]*).
  Expected values:
  - *tx_data_5[7]* equal to 0

- The ***Fr_check_violation(****FR_CHANNEL_AB)* function is called to check whether or not a frame transmission in the dynamic segment exceeded the dynamic segment boundary on channel A or B. In such a case, the *violation* variable is incremented and then stored to the ninth element of the transmit data array for slot 5 (*tx_data_5[8]*).
  Expected values:
  - *tx_data_5[8]* equal to 0

The next task performed in the never-ending *for()* loop checks on whether or not the protocol engine has detected an internal protocol error. The ***Fr_check_internal_protocol_error()*** function is called and in case that an internal protocol error has occurred, the *protocol_error* variable is set (and the protocol engine goes into the *POC:halt* state).

Expected values:

– *protocol_error* equal to FALSE

The following routine updates transmit message buffer 12, which is configured for the dynamic segment of the communication cycle. The ***Fr_check_tx_status(****TX_SLOT_63****)*** function is called to check the status of the transmit single buffer determined by the input parameter. In case that data has been transmitted (*FR_TRANSMITTED* returned value):

– the first element of the data array (*tx_data_63[0]*) is decremented
– the ***Fr_transmit_data(****TX_SLOT_63, &tx_data_63[0], 0****)*** function updates message buffer 12 with new data provided by reference to the *tx_data_63[]* array. The FlexRay UNIFIED Driver determines the data payload length from message buffer configuration registers where the *data length* input parameter is equal to zero. This approach makes sense to a transmit MB configured for the dynamic segment of the communication cycle.

Expected values:

– *tx_data_63[0]* decremented by 50 in each cycle from 0xFFFF to 0, then wraps back to 0xFFFF

The ***Fr_check_tx_status()*** function is called several times during one communication cycle, however, the *FR_TRANSMITTED* value should be returned only once.

NOTE

The updating of the transmit message buffer need not be synchronised to any time of the communication cycle (e.g. cycle start, or any timer interrupt) if the ***Fr_check_tx_status()*** *function is called before updating.*

For checking on whether or not any *access error* occurred during access to the message buffers, the number of errors is stored into the sixth element of the data array for slot 5 (*tx_data_5[5]*), to monitor an error occurrence by means of the FlexRay bus monitoring tool.

Expected values:

– *tx_data_5[5]* equal to 0

3.2.6 Demonstration of the Transmit Receive Application Example - Running the FreeMASTER tool

The observed variables, transmit and receive data can be displayed in the FreeMASTER tool as shown in section Figure 3-2. FreeMASTER Control Page for Node 1 of the Transmit Receive Application Example and section Figure 3-3. FreeMASTER Control Page for Node 2 of the Transmit Receive Application Example. More details on the observed variables can be found in chapters section 3.2.4 Detailed Description of Application Software for Node 1and section 3.2.5 Detailed Description of Application Software for Node 2.

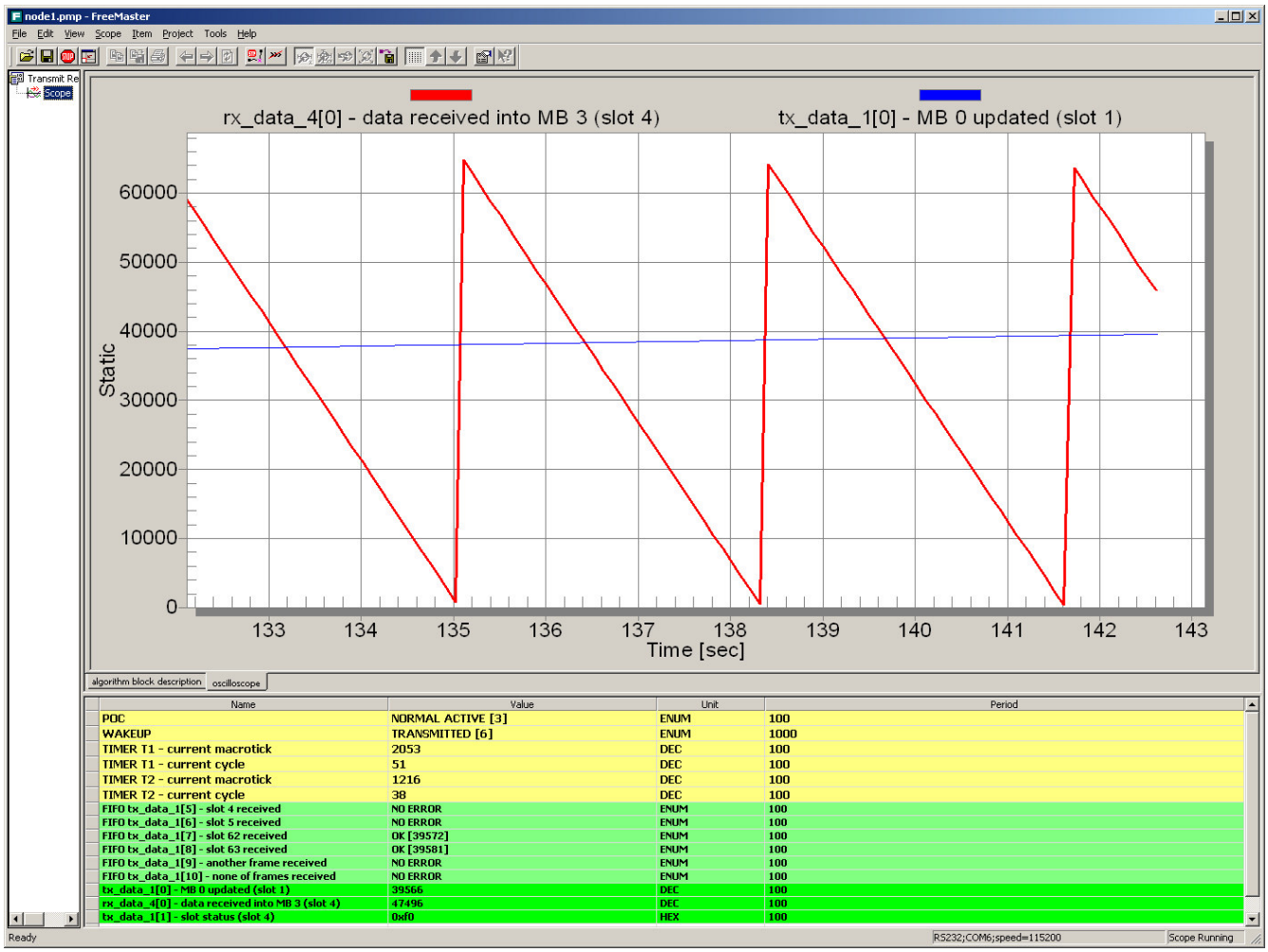

Figure 3-2. FreeMASTER Control Page for Node 1 of the Transmit Receive Application Example

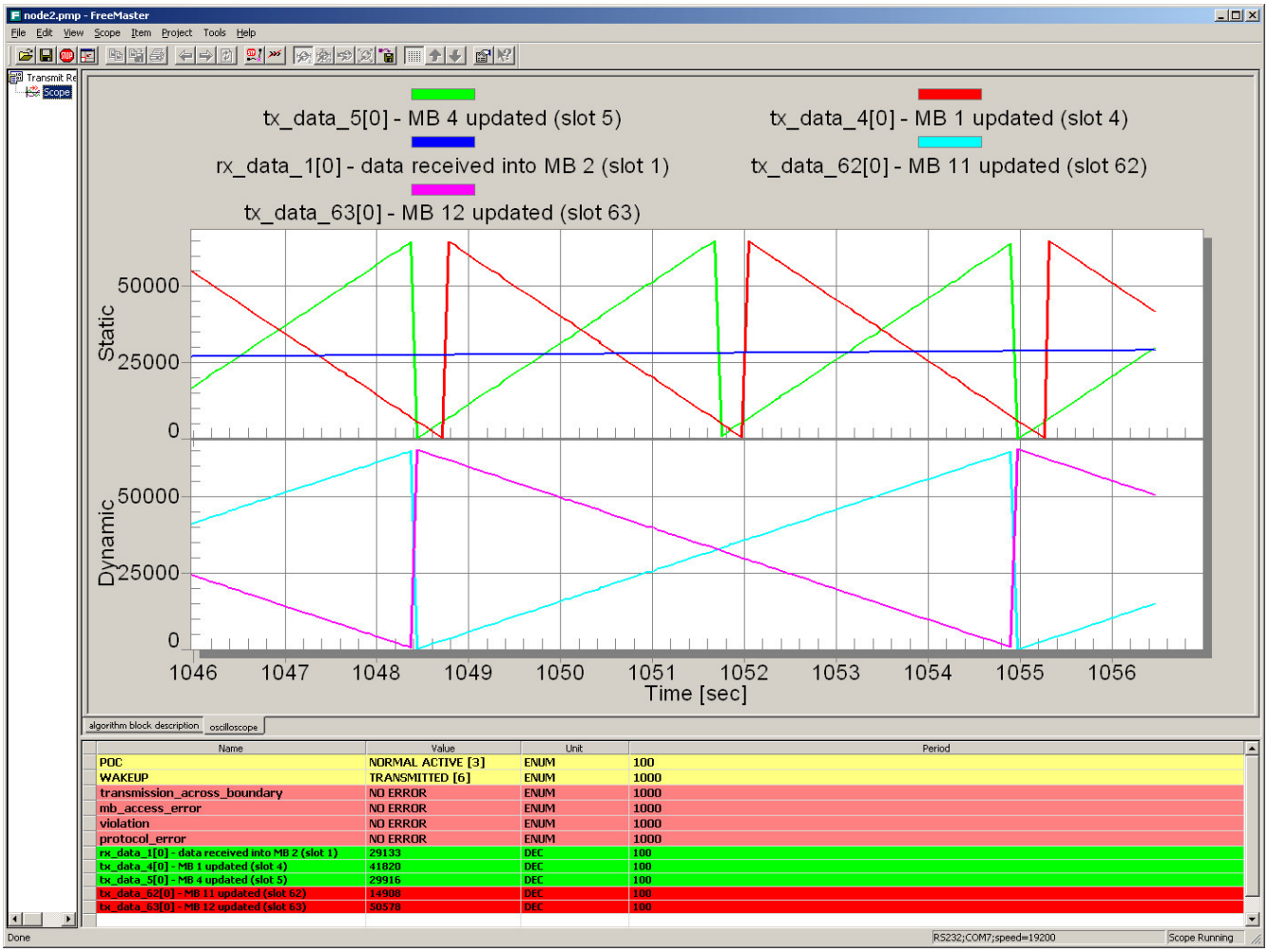

Figure 3-3. FreeMASTER Control Page for Node 2 of the Transmit Receive Application Example

3.3 Status Monitoring Application Example

The second example shows continuous communication between two FlexRay nodes (called Node 1 and Node 2). Both nodes operate in the interrupt driven mode.

The host application software:

- configures the FlexRay module with an appropriate FlexRay schedule and timing, see section section 3.3.1 FlexRay Schedule and Timing
- configures the receive and transmit double message buffers, allowing transmission and reception of the FlexRay messages, and enables message buffer interrupts
- configures one absolute timer and enables the timer interrupt
- transmits wake-up symbols
- establishes FlexRay communication between the two nodes - they are configured as coldstart nodes *(see [FR_PROTOCOL] for details)*
- updates transmit message buffers with new data when a message buffer interrupt occurs
- shows usage of:
  - slot status monitoring
  - slot status counter monitoring
  - channel status error monitoring
  - Network Management Vector
  - synchronisation state detection
  - Low Level Access Support functions
- stops communication by calling the HALT and FREEZE protocol commands
- transmits the Media Test Access Symbol on channel B
- is able to re-configure the FlexRay module and reintegrate it again into the FlexRay cluster.

3.3.1 FlexRay Schedule and Timing

Nodes are configured with the cluster configuration data specified in section Table 3-1 FlexRay Configuration Data - General Cluster Parameters. Message buffers are configured for transmitting and receiving as shown in section Table 3-1 Slot Assignment.

Table 3-1 Slot Assignment

Slot	Transmitting	Receiving	FIFO receiving
Slot assignment - Static segment			
Slot 1	Node 1, channel A and B, startup and sync frame	Node 2, channel A	
Slot 4	Node 2, channel A and B, startup and sync frame	Node 1, channel A	
Slot 12	Node 1, channel A and B		
Slot 20	Node 2, channel A and B		
Slot assignment - Dynamic segment			
Not configured			

The application example shows transmission in the static part of the communication cycle. One of the static slots of each node is also used simultaneously as a synchronisation/start-up and as a data slot. The FlexRay communication cycle period is 5 ms.

Node 1 sends data in the 1st and in the 12th slots, Node 2 sends data in the 4th and in the 20th slots of the communication cycle. This configuration is shown in section Figure 3-4. Configuration of the FlexRay Communication Cycle for Transmit Receive Application Example.

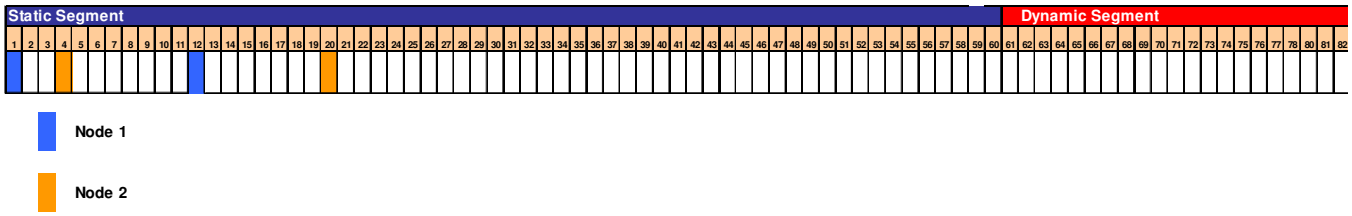

Figure 3-4. Configuration of the FlexRay Communication Cycle for Transmit Receive Application Example

The transmit and receive message buffers are configured as described in section Table 3-2 Message Buffers Configuration for Node 1 and section Table 3-8 Message Buffers Configuration for Node 2.

The application configures both individual message buffer segments. However, it uses only the first segment (segment 1) for the message buffers configured for transmission or reception in the static part of the communication cycle. Assignment of the message buffer number to message buffer segments is shown in section Table 3-3 Connection Among the Message Buffers and Segments for Node 1 and section Table 3-9 Connection Among the Message Buffers and Segments for Node 2.

3.3.2 Configuration for Node 1

Static segment:

Transmission: Slot 1, Slot 12

Reception: Slot 4

Table 3-2 Message Buffers Configuration for Node 1

Segment	Static		
Slot number	1	4	12
Channels	A,B	A	A,B
Assigned Tx message buffer	4		5
Assigned Rx message buffer		1	
Message buffer type (only Tx)	Single		Single
Payload length (only Tx) [words]	16		16
Interrupt enabled	Yes	Yes	Yes

Table 3-2 Message Buffers Configuration for Node 1

Name of interrupt service routine	CC_interrupt_slot_1	CC_interrupt_slot_4	CC_interrupt_slot_12
Data update (only Tx)	Each cycle		Each cycle
Name of data array used for data updating or storing	tx_data_1	rx_data_4	tx_data_12

The message buffer segments are configured according to the section Table 3-3 Connection Among the Message Buffers and Segments for Node 1 as shown in section Table 3-4 Message Buffer Segments Configuration for Node 1.

Table 3-3 Connection Among the Message Buffers and Segments for Node 1

Type of Segment		Number of MB		Data size [Words]
		From	To	
Individual MB	Segment 1	0	11	16
	Segment 2	12	19	8
FIFO A		Not configured		
FIFO B		Not configured		

Table 3-4 Message Buffer Segments Configuration for Node 1

Parameter	Configured value
Data size for segment 1 [W]	16
Data size for segment 2 [W]	8
The number of MB's in segment 1	12
Last MB in segment 1	11
The number of MB's in segment 2	8
Last individual MB	19
FIFO depth	0
Total number of used MB's (Last individual MB + 1 + FIFO depth)	20

For correct frame reception, it is necessary to configure the Shadow message buffers as shown in section Table 3-5 Shadow Message Buffers Configuration for Node 1. The Shadow message buffers are configured only for the individual segment 1 because the individual segment 2 is not used for reception. For more information on the Shadow message buffer concept, please see [FLEXRAY_MODULE], chapter Message Buffer Types.

Table 3-5 Shadow Message Buffers Configuration for Node 1

Segment	Segment 1		Segment 2	
Channel	A	B	A	B
Assigned buffer index	10	11	Not configured	

One absolute timer is configured in the application example as shown in section Table 3-6 FlexRay Timer Configuration for Node 1.

Table 3-6 FlexRay Timer Configuration for Node 1

Timer	T1	T2
Timer timebase	Absolute	Not configured
Repetitive mode	Repetitive	
Macrotick offset	1000	
Filter mask	0	
Filter value	0	

The slot status functionality is used to detect which channel is connected to the cluster, as shown in section Table 3-7 Slot Status Configuration.

Table 3-7 Slot Status Configuration

Configured slot status register (SSSRn)	Observed slot
0	4
1	-
2	-
3	-

All configuration information is stored in the *Fr_UNIFIED_cfg.c* file as it is described in section Chapter 4 Application Example Project Structures.

General cluster configuration data is stored in the *Fr_low_level_cfg_set_00* structure (defined as type *Fr_low_level_config_type*).

Message buffer configuration data is stored in *the Fr_buffer_cfg_00[]* array of structures (defined as type *Fr_buffer_info_type*). Message buffers to be configured are selected by means of the *Fr_buffer_cfg_set_00[]* array of indexes (defined as type *Fr_index_selector_type*). This array must have the FR_LAST_MB (0xFF) value stored in the last item for correct the FlexRay UNIFIED Driver operation.

Shadow buffers configuration data is stored in the *Fr_rx_shadow_cfg* structure (defined as type *Fr_receive_shadow_buffers_config_type*).

The reference on configuration data for the timers is stored in the *Fr_timers_cfg_00_ptr[]* array (defined as type *Fr_timer_config_type*); this array has to be ended by a NULL pointer for correct driver function

operation.
Timer 1 configuration data is stored in the *Fr_timer_1_cfg* structure.

Slot status configuration data is stored in the *Fr_slot_status_cfg_set_00* structure (defined as type *Fr_slot_status_config_type*).

3.3.3 Configuration for Node 2

Static segment:

Transmission: Slot 4, Slot 20

Reception: Slot 1

Table 3-8 Message Buffers Configuration for Node 2

Segment	Static		
Slot number	1	4	20
Channels	A	A, B	A, B
Assigned Tx message buffer		0	2
Assigned Rx message buffer	1		
Message buffer type (only Tx)		Single	Single
Payload length (only Tx) [words]		16	16
Interrupt enabled	Yes	Yes	Yes
Name of interrupt service routine	CC_interrupt_slot_1	CC_interrupt_slot_4	CC_interrupt_slot_20
Data update (only Tx)		Each cycle	Each cycle
Name of data array used for data updating or storing	rx_data_1	tx_data_4	tx_data_20

The message buffer segments are configured according to the section Table 3-9 Connection Among the Message Buffers and Segments for Node 2 as shown in section Table 3-10 Message Buffer Segments Configuration for Node 2.

Table 3-9 Connection Among the Message Buffers and Segments for Node 2

Type of Segment		Number of MB's		Data size [Words]
		From	To	
Individual MB	Segment 1	0	12	16
	Segment 2	13	20	8
FIFO A		Not configured		
FIFO B		Not configured		

Table 3-10 Message Buffer Segments Configuration for Node 2

Parameter	Configured value
Data size for segment 1 [W]	16
Data size for segment 2 [W]	8
The number of MB in segment 1	13
Last MB in segment 1	12
The number of MB in segment 2	8
Last individual MB	20
FIFO depth	0
Total number of used MB (Last individual MB + 1 + FIFO depth)	21

For correct frame reception, it is necessary to configure the Shadow message buffers as shown in section Table 3-11 Shadow Message Buffers Configuration for Node 2. The Shadow message buffers are configured only for the individual segment 1 because the individual segment 2 is not used for reception. For more information on the Shadow message buffer concept, please see [FLEXRAY_MODULE], chapter Message Buffer Types.

Table 3-11 Shadow Message Buffers Configuration for Node 2

Segment	Segment 1		Segment 2	
Channel	A	B	A	B
Assigned buffer index	11	12	Not configured	

One absolute timer is configured in application example as shown in section Table 3-12 FlexRay Timer Configuration for Node 2.

Table 3-12 FlexRay Timer Configuration for Node 2

Timer	T1	T2
Timer timebase	Absolute	Not configured
Repetitive mode	Repetitive	
Macrotick offset	1000	
Filter mask	0	
Filter value	0	

The application example shows the usage and configuration of the slot status functionality, and detection of which channel is connected to the cluster. Configuration of the slot status registers is shown in section Table 3-13 Slot Status Configuration for Node 2.

Table 3-13 Slot Status Configuration for Node 2

Configured slot status register (SSSRn)	Observed slot
0	1
1	12
2	-
3	-

The Media Access Test Symbol (MTS) configuration data is displayed in section Table 3-14 FlexRay MTS Configuration for Node 2. The transmission is set on channel B during the *Symbol Window* of each communication cycle.

Table 3-14 FlexRay MTS Configuration for Node 2

Channel	A	B
Filter mask	Not configured	0
Filter value		0

The slot status counters are configured as shown in section Table 3-15 FlexRay Slot Status Counters Configuration for Node 2. For more information on the Slot Status Counter concept, please see [FLEXRAY_MODULE], chapter Status Monitoring.

Table 3-15 FlexRay Slot Status Counters Configuration for Node 2

Counter number	0	1	2	3
Counter configuration	Incremented by 1 on channel A or B, incremented by 2 on channel A and B	Incremented by 1 on channel A or B, incremented by 2 on channel A and B	Incremented by 1 on channel A or B	Not configured
Multi cycle selection	Over multiple cycles	Over multiple cycles	Previous cycle only	
Frame restriction	Valid, sync	No	Valid, startup	
Error counting	No	Content error, boundary violation, transmission conflict	No	

All configuration information is stored in the *Fr_UNIFIED_cfg.c* file as it is described in section Chapter 4 Application Example Project Structures.

General cluster configuration data is stored in the *Fr_low_level_cfg_set_00* structure (defined as type *Fr_low_level_config_type*).

Message buffer configuration data is stored in *the Fr_buffer_cfg_00[]* array of structures (defined as type *Fr_buffer_info_type*). Message buffers to be configured are selected by means of the *Fr_buffer_cfg_set_00[]* array of indexes (defined as type *Fr_index_selector_type*). This array must have the FR_LAST_MB (0xFF) value stored in the last item for correct the FlexRay UNIFIED Driver operation.

Shadow buffers configuration data is stored in the *Fr_rx_shadow_cfg* structure (defined as type *Fr_receive_shadow_buffers_config_type*).

The reference on configuration data for timers is stored in the *Fr_timers_cfg_00_ptr[]* array (defined as type *Fr_timer_config_type*); this array has to be ended by a NULL pointer for correct driver function operation.
Timer 1 configuration data is stored in the *Fr_timer_1_cfg* structure.

Slot status configuration data is stored in the *Fr_slot_status_cfg_set_00* structure (defined as type *Fr_slot_status_config_type*).

The MTS configuration data is stored in the *Fr_MTS_B_cfg* structure (defined as *type Fr_MTS_config_type*).

The references on configuration data for slot status counters are stored in the *Fr_slot_status_counter_cfg_00_ptr[]* array (defined as type *Fr_slot_status_counter_config_type*); this array has to be ended by NULL pointer for correct driver function operation. Slot status counter 0 configuration data is stored in the *Fr_slot_status_counter_0_cfg* structure, slot status counter 1configuration data in the *Fr_slot_status_counter_1_cfg* structure, and slot status 2 configuration in the *Fr_slot_status_counter_2_cfg* structure.

3.3.4 Detailed Description of Application Software for Node 1

Node 1 is configured mainly to demonstrate the reception of the MTS, usage of the Network Management Vector, channel status error and low level access support.

3.3.4.1 Initialization procedure

The initialization sequence is similar to that in the Transmit Receive application example for the Node 1 (see section 3.2.4.1 Initialization procedure).

The ***Fr_init(****&Fr_HW_cfg_00, &Fr_low_level_cfg_set_00)* function is called only once during runtime, even if the FlexRay module is reconfigured. All other configuration and startup functions are concentrated into the *CC_init_and_startup()* routine which enables to re-configure the FlexRay module and reintegrate it again easily into the FlexRay cluster.
The ***Fr_slot_status_init(****&Fr_slot_status_cfg_set_00)* function is called in the *CC_init_and_startup()* routine to initialize the slot status functionality. The input parameter points to the configuration structure of the slot status registers.
All used message buffers run in the interrupt driven mode and the relevant callback functions are set by the consecutive calling of the ***Fr_set_MB_callback()*** functions.
The application also configures one absolute timer through the ***Fr_timers_init(****&Fr_timers_cfg_00_ptr)* function.

In case that the FlexRay module is not able to integrate into the FlexRay cluster a defined time (it remains in the *POC:startup* state), the module is re-configured again later on.

3.3.4.2 Application code

After initialising and starting of the Flexray module, the two available Low Level Access functions are called to show one possible way of controlling the FlexRay module by means of this Low Level Access method:

- the ***Fr_low_level_access_read_reg(****FrMVR)* function reads the content of the MVR register and stores it into the *mvr_register_value* variable

  Expected values:
  – *mvr_register_value* equal to 0x3535, 0x3434 or similar, according to the available version of the FlexRay module

- the ***Fr_low_level_access_write_reg(****FrPIFR1, 0x0000)* function writes the value 0x0000 into the PIFR1 register. Since writing 0 does not change the state of the flags in the PIFR1 register, it does not influence any FlexRay module function.

In the never-ending *for()* loop, the following tasks are performed:

- the ***Fr_low_level_access_read_memory(****16)* function is called to show one possible way of controlling the FlexRay memory by means of the Low Level Access method. The function reads the content of the Data Field Offset of the first message buffer Header Field from the FlexRay memory. The value is then stored into the element *tx_data_12[2]* of data payload array for slot 12, to be transmitted later on.

  Expected values:
  tx_data_12[2] equal to 0x00E8

- the ***Fr_low_level_access_read_memory(****data_field_off_value****)*** function is called to read the first 2-bytes of the data from the message buffer Data Field of the first message buffer from the FlexRay memory. The value is then stored into the element *tx_data_12[3]* of data payload array for slot 12, to be transmitted later on.

  Expected values:
  tx_data_12[3] equal to the first 2-bytes of received payload data for slot 4 - it should be equal to the *rx_data_4[0]* after copying of received data into the *rx_data_4[0]* data array

- the ***Fr_get_MTS_state(****FR_CHANNEL_B****)*** function is called to query the status of MTS reception on channel B. The status value is then stored into the element *tx_data_1[2]* of data payload array for slot 1, to be transmitted later on. This approach allows the monitoring of the MTS state by means of the FlexRay bus monitoring tool.

  Expected values:
  – *tx_data_1[2]* equal to 0x0000 (FR_MTS_RCV)

- the ***Fr_get_sync_state()*** function is called to find out whether the FlexRay module is synchronous to the cluster. In case that the node is not synchronised for any reason (or a previous integration process has not been successful), it is convenient to re-integrate the FlexRay module into the cluster. To do so, it is necessary to go correctly through the POC states of the FlexRay module. For more information on the POC concept, please see [FR_PROTOCOL], chapter The Protocol Operation Control process. The ***Fr_get_POC_state()*** function is called to query the current POC state. If the module is not in the *POC:halt* state, the ***Fr_stop_communication(****FR_ABORT_COMMUNICATION****)*** function is called to move the FlexRay module to the *POC:halt* state. Once the FlexRay module is in the *POC:halt* state, the *CC_reconfiguration()* host application routine is called to re-configure the FlexRay module. In this routine, the ***Fr_enter_configuration_mode()*** function is called to enter into the *POC:config* state (through the *POC:default config* state). After that, all initialization processes and integration are performed in the *CC_init_and_startup()* function. In case that the FlexRay module is not able to integrate into the FlexRay cluster a defined time (it remains in the *POC:startup* state), the module is re-configured again.

  NOTE

  In case that the FREEMASTER_USED parameter has been defined in the host application, in addition to the previously described reintegration process, the abortion of the FlexRay communication and subsequent reintegration into the cluster can be controlled by the FreeMASTER control page. See section 3.3.6.1 Detailed Description of the FreeMASTER Control Page for Node 1 for more information

3.3.4.3 Interrupt services

When an interrupt arises, the ***Fr_interrupt_handler()*** routine is called, and if a callback function has been configured for the related interrupt, this callback function is called. A short description of each function is below:

- the *CC_interrupt_slot_1()* routine should be called in each cycle after message buffer 4 (configured for slot 1) has been transmitted. The ***Fr_transmit_data(****TX_SLOT_1, &tx_data_1[0], 16****)*** function updates message buffer 4 with new

data provided by reference on the *tx_data_1[]* array. If the MB has been successfully updated, the first element of the data array (*tx_data_1[0]*) is incremented.

Expected values:
– *tx_data_1[0]* incremented by 1 in each cycle from 0 to 0xFFFF, then wraps to 0

- the *CC_interrupt_slot_4()* routine should be called in each cycle after a valid non-null frame is received by receive message buffer 1 (configured for slot 4). The ***Fr_receive_data(****buffer_idx, &rx_data_4[0], &rx_data_length, &rx_status_slot****)*** function stores received data from the MB given by the *buffer_idx* parameter into the *rx_data_4[]* array, stores the length of the received payload data into *the rx_data_length* variable, and stores the slot status of the received frame into the *rx_status_slot* variable. The *buffer_idx* value is given by the ***Fr_interrupt_handler()*** function and determines which MB has generated the interrupt.
  In the following line of the application code, the second array element (*tx_data_1[1])* is incremented (the transmit MB will be updated with this data payload during the next cycle). This approach allows monitoring the frequency of *receive* MB interrupts by means of the FlexRay bus monitoring tool.

  Expected values:
  – *buffer_idx* equal to 1
  – *tx_data_1[1]* incremented by 1 in each cycle from 0 to 0xFFFF, then wraps to 0

- the *CC_interrupt_slot_12()* routine should be called in each cycle after message buffer 5 (configured for slot 12) has been transmitted. The ***Fr_transmit_data(****buffer_idx, &tx_data_12[0], 16****)*** function updates message buffer 5 with new data provided by reference to the *tx_data_12[]* array. If the MB has been successfully updated, the second element of the data array (*tx_data_12[1]*) is incremented. Note, that the Network Management Vector is transmitted in the first word of the data payload (*tx_data_12[0]*).

  Expected values:
  – *tx_data_12[0]* equal to 0x0001
  – *tx_data_12[1]* incremented by 1 in each cycle from 0 to 0xFFFF, then wraps to 0

- the *CC_interrupt_timer_1()* routine is called when a timer interrupt from absolute timer T1 occurs, i.e. in 1000 MT of the communication cycle.
  The ***Fr_get_channel_status_error_counter_value(****FR_CHANNEL_A, &error_counter_A****)*** and ***Fr_get_channel_status_error_counter_value(****FR_CHANNEL_B, &error_counter_B****)*** functions get the current values of the channel status error counters. These values are then stored into the elements *tx_data_1[3]* and *tx_data_1[4]* of the data payload array to be transmitted later on.
  Expected values:
  – *tx_data_1[3]* equal to 0
  – *tx_data_1[4]* equal to 0

  To show the Network Management Vector usage, both nodes are configured to send one Network Management Vector during the communication cycle. Node 1 sends its vector in slot 12, in the first element of the data payload (*tx_data_12[0]*), and the value is set to 0x0001 (the first bit is set). Node 2 sends its vector in slot 20, in the first element of the data payload (*tx_data_20[0]*), and the value is set to 0x0002 (the second bit is set).

The FlexRay module accumulates all the received Network Management Vectors during the last communication cycle. This value is stored through the ***Fr_get_network_management_vector(****&network_management_vector[0])* function into the *network_management_vector[]* array and then to the sixth array element (*tx_data_1[5])* to be transmitted later on.
To determine which node is connected to the cluster (in fact, from which node the Network Management Vector was received), it is possible to check which bit is set in the received Network Management Vector.

It is convenient to use the slot status functionality to determine which channel is connected into the cluster. It is necessary to configure at least one slot status register to store the slot status information from some received frame (no MB needs to be assigned to this frame). The ***Fr_get_slot_status_reg_value(****4, FR_CHANNEL_A, FR_SLOT_STATUS_CURRENT, &slot_status_4)* function reads the latest updated slot status information from slot 4. The application then checks whether a valid frame has been received in the current communication cycle on each channel (or the previous one, if the ***Fr_get_slot_status_reg_value()*** is called before slot 4 is received). Where a valid frame has been received on a given channel (that channel is connected to the cluster), the 13th array element (*tx_data_1[12])* is modified appropriately.

Expected values:
– *tx_data_1[12]* equal to:
  i. value 0x0000, if no channel is connected to the cluster
  ii. value 0x0100, if only channel B is connected to the cluster
  iii. value 0x0001, if only channel A is connected to the cluster
  iv. value 0x0101, if both channels are connected to the cluster

3.3.5 Detailed Description of Application Software for Node 2

Node 2 is configured mainly to demonstrate the transmission of the MTS, usage of the slot status and slot status counter.

3.3.5.1 Initialization procedure

The initialization sequence is the same as for the Node 1 (see section 3.3.4.1 Initialization procedure). Moreover, the slot status counters are configured by means of the ***Fr_slot_status_counter_init(****&Fr_slot_status_counter_cfg_00_ptr)* function. The input parameter points to the slot status counter configuration data structure.

3.3.5.2 Application code

After initialising and starting the Flexray module,

- the ***Fr_send_MTS(****FR_CHANNEL_B, &Fr_MTS_B_cfg)* function initializes the transmission of the MTS over channel B, if the FreeMASTER tool is not used (the FREEMASTER_USED parameter is not defined). The second input parameter points to the MTS configuration data structure
- the ***Fr_low_level_access_write_memory(****46, TEST_NUMBER)* function is called to show one possible way of controlling the FlexRay memory by means of the Low Level Access method. The function writes a test value (*TEST_NUMBER*) into the Data Field Offset of the fourth message

buffer Header Field in the FlexRay memory.
Since the fourth message buffer is not configured, the application can access to the Data Field Offset of this message buffer (the FlexRay module do not use this memory field), otherwise the host has to follow the access restriction given in [FLEXRAY_MODULE], chapters Message Buffer Data Field Read Access and Message Buffer Data Field Write Access.

- the ***Fr_low_level_access_read_memory(****46)* function is called to check whether the previous write to FlexRay memory has been successful. Read value is then stored into the element *tx_data_20[2]* of data payload array for slot 20, to be transmitted later on.

  Expected values:
  tx_data_20[2] equal to 0xABCD (*TEST_NUMBER*)

In the never-ending *for()* loop, the following tasks are performed:

- the ***Fr_send_MTS(****FR_CHANNEL_B, &Fr_MTS_B_cfg)* function initializes the transmission of the MTS over channel B, if the FreeMASTER tool is used (the FREEMASTER_USED parameter is defined) and a user enables it by clicking on the 'Send MTS' button. For more information, see section 3.3.6.2 Detailed Description of the FreeMASTER Control Page for Node 2.
- the ***Fr_get_MTS_state(****FR_CHANNEL_A)* function is called to query the status of MTS reception on channel A. The status value is then stored into the *tx_data_4[2]* element of the data payload array for slot 4, to be transmitted later on. Since Node 1 does not transmit the MTS, no MTS symbol should be received. This approach allows monitoring of the MTS state by means of the FlexRay bus monitoring tool.

  Expected values:
  – *tx_data_4[2]* equal to 0x0004 (FR_MTS_NOT_RCV)

- the Flexray module is re-configured and reintegrated into the FlexRay cluster again if necessary. The code sequence is the same as for Node 1, the detail description can be found in section 3.3.4.2 Application code.

NOTE

Where the FREEMASTER_USED parameter is defined in the host application, in addition to the previously described reintegration process, the abortion of the FlexRay communication and subsequent reintegration into the cluster can be controlled by the FreeMASTER control page. See section 3.3.6.2 Detailed Description of the FreeMASTER Control Page for Node 2 for more information

3.3.5.3 Interrupt services

When an interrupt arises, the ***Fr_interrupt_handler()*** routine is called, and if a callback function has been configured for the related interrupt, this callback function is called. A short description of each function is below:

- the *CC_interrupt_slot_4()* routine should be called in each cycle after message buffer 0 (configured for slot 4) has been transmitted. The ***Fr_transmit_data(****buffer_idx, &tx_data_4[0], 16)* function updates message buffer 0 with new data provided by reference to the *tx_data_4[]* array. If the MB

has been successfully updated, the first element of the data array (*tx_data_4[0]*) is incremented. The *buffer_idx* value is given by the ***Fr_interrupt_handler()*** function and determines which MB generated the interrupt.

Expected values:
– *tx_data_4[0]* incremented by 1 in each cycle from 0 to 0xFFFF and then wraps to 0

- the *CC_interrupt_slot_1()* routine should be called in each cycle after a valid non-null frame is received by receive message buffer 1 (configured for slot 1). The ***Fr_receive_data(****buffer_idx, &rx_data_1[0], &rx_data_length, &rx_status_slot****)*** function stores received data from a MB given by the *buffer_idx* parameter into the *rx_data_1[]* array, stores the length of the received data payload into *the rx_data_length* variable, and stores the slot status of the received frame into the *rx_status_slot* variable. The *buffer_idx* value is given by the ***Fr_interrupt_handler()*** function and determines which MB generated the interrupt.
  In the following line of the application code, the second array element (*tx_data_4[1])* is incremented (the transmit MB will be updated with this data payload during the next cycle). This approach allows monitoring of the frequency of receive MB interrupts by means of the FlexRay bus monitoring tool.

  Expected values:
  – *buffer_idx* equal to 1
  – *tx_data_4[1]* incremented by 1 in each cycle from 0 to 0xFFFF and then wraps to 0

- the *CC_interrupt_slot_20()* routine should be called in each cycle after message buffer 2 (configured for slot 20) has been transmitted. The ***Fr_transmit_data(****buffer_idx, &tx_data_20[0], 16****)*** function updates message buffer 2 with new data provided by reference on the *tx_data_20[]* array. If the MB has been successfully updated, the second element of the data array (*tx_data_20[1]*) is incremented. Note, that the Network Management Vector is transmitted in the first word of the data payload (*tx_data_20[0]*).

  Expected values:
  – *tx_data_20[0]* equal to 0x0002
  – *tx_data_20[1]* incremented by 1 in each cycle from 0 to 0xFFFF, then wraps to 0

- the *CC_interrupt_timer_1()* routine is called when a timer interrupt from absolute timer T1 has occurred, i.e. in 1000 MT of the communication cycle.
  To show the Network Management Vector usage, both nodes are configured to send one Network Management Vector during the communication cycle. Node 1 sends its vector in slot 12, in the first element of the data payload (*tx_data_12[0]*), and its value is set to 0x0001 (the first bit is set). Node 2 sends its vector in slot 20, in the first element of the data payload (*tx_data_20[0]*), and its value is set to 0x0002 (the second bit is set).
  The FlexRay module accumulates all the received Network Management Vectors in the last communication cycle. This value is stored by means of the ***Fr_get_network_management_vector(****&network_management_vector[0]****)*** function into the *network_management_vector[]* array and then to the sixth array element (*tx_data_4[5])*, to be transmitted later on.
  To determine which node is connected to the cluster (in fact, from which node the Network Management Vector was received), it is possible to check which bit is set in the received Network Management Vector.

It is convenient to use the slot status functionality to determine which channel is connected to the cluster. It is necessary to configure at least one slot status register to store the slot status information from some received frame (no MB need be assigned to this frame). The ***Fr_get_slot_status_reg_value(****1, FR_CHANNEL_B, FR_SLOT_STATUS_CURRENT, &slot_status_1****)*** function reads the latest updated slot status information from slot 1. The application then checks whether a valid frame has been received in the current ommunication cycle on each channel (or in previous one, if ***Fr_get_slot_status_reg_value()*** is called before slot 1is received). Where a valid frame has been received on a given channel (that channel is connected to the cluster), the 13th element of the *tx_data_4[12]* array is modified appropriately.

Expected values:
– *tx_data_4[12]* equals to:
  i. value 0x0000, if no channel is connected to the cluster
  ii. value 0x0100, if only channel B is connected to the cluster
  iii. value 0x0001, if only channel A is connected to the cluster
  iv. value 0x0101, if both channels are connected to the cluster

The value of the *slot_status_1* variable is stored to the seventh element of the *tx_data_4[6]* array, to be transmitted later on.
The ***Fr_get_slot_status_reg_value(****12, FR_CHANNEL_A, FR_SLOT_STATUS_PREVIOUS, &slot_status_12****)*** function gets the slot status information for slot 12 from the last communication cycle or the cycle before that. The value of the *slot_status_12* variable is stored in the eighth element of the *tx_data_4[7]* array, to be transmitted later on.

Expected values:
– *tx_data_4[6]* equal to 0xF0F0 (a valid, sync and startup frame has been received in slot 1)
– *tx_data_4[7]* equal to 0xA0A0 (a valid frame has been received in slot 12)

To show the slot status counter functionality the
Fr_get_slot_status_counter_value(*FR_SLOT_STATUS_COUNTER_0, &slot_status_counter_0****)***,
Fr_get_slot_status_counter_value(*FR_SLOT_STATUS_COUNTER_1, &slot_status_counter_1****)*** and
Fr_get_slot_status_counter_value(*FR_SLOT_STATUS_COUNTER_2, &slot_status_counter_2****)*** functions are called. The first input parameter determines the slot status counter used (one of four available), the second parameter points to the variable to which the value of the corresponding counter will be stored.
The slot status counter 0 is configured to increment its value when a sync frame is received during the communication cycle. The counter value is stored in the tenth element of the *tx_data_4[9]* array, to be transmitted later on.
The slot status counter 1 is configured to increment its value when a content error, boundary violation or transmission conflict indicators is set during the communication cycle. The counter value is stored in the eleventh element of the *tx_data_4[10]* array, to be transmitted later on.
The slot status counter 2 is configured to increment its value when a startup frame is received during the communication cycle. This counter provides information on the previous communication cycle only. The counter value is stored in the twelfth element of the *tx_data_4[11]* array, to be

transmitted later on.

Expected values:
– *tx_data_4[9]* incremented:
  i. by 2 in each cycle from 0 to 0xFFFF, then wraps to 0, in the case of both channels being connected (one sync frame has been received on each channel)
  ii. by 1 in each cycle from 0 to 0xFFFF, then wraps to 0, in the case of only one channel being connected (one sync frame has been received on one channel)
– *tx_data_4[10]* equal to 0x0000 (no error indicator set)
– *tx_data_4[11]* equal to 0x0001 (one startup frame has been received during the previous communication cycle)

3.3.6 Demonstration of the Slot Status Application Example - Running the FreeMASTER tool

The observed variables, transmit and receive data can be displayed in the FreeMASTER tool as shown in section Figure 3-5. FreeMASTER Control Page for Node 1 of the Slot Status Application Example and section Figure 3-6. FreeMASTER Control Page for Node 2 of the Slot Status Application Example. More details on observed variables can be found in chapters section 3.3.4 Detailed Description of Application Software for Node 1and section 3.3.5 Detailed Description of Application Software for Node 2.

3.3.6.1 Detailed Description of the FreeMASTER Control Page for Node 1

The FreeMASTER Node 1 control page visualises and controls the main features configured for Node 1 (see section 3.3.2 Configuration for Node 1), such as stop communication by calling the FREEZE protocol command, the reception of the media test access symbol (MTS), usage of the network management vector, slot status, channel status error, etc.

The FreeMASTER control page for Node 1 of the Status Monitoring application example consists of one switch, ‘Abort Communication’, controlling the FREEZE protocol command, and five indicators (represented by illustrated LEDs) for monitoring the slot status and the network management vector.

By clicking on the ‘Abort Communication’ switch, the ***Fr_stop_communication(****FR_ABORT_COMMUNICATION)* function is called in the host application, and the FlexRay module immediatelly halts the communication and moves onto the *POC:halt* state. The *POC* variable in the FreeMASTER control page, which reflects the state of the *protocol_state* variable in the host application, will show the HALT[2] state. By next clicking ‘Restart’, the ***Fr_enter_configuration_mode()*** function and the *CC_init_and_startup()* routine are called to reconfigure the FlexRay module.

The channel A and Channel B indicators (represented by the illustrated LEDs) monitor the slot status, and show that channel A and channel B are connected, by a green color or disconnected, by a red.

Node 1, Node2 and Node 3 indicators monitor the network management vector status. These indicators show whether or not each individual node is connected. In this application example, the FlexRay cluster consists of two nodes, therefore only the Node 1 and Node 2 indicators can be activated.

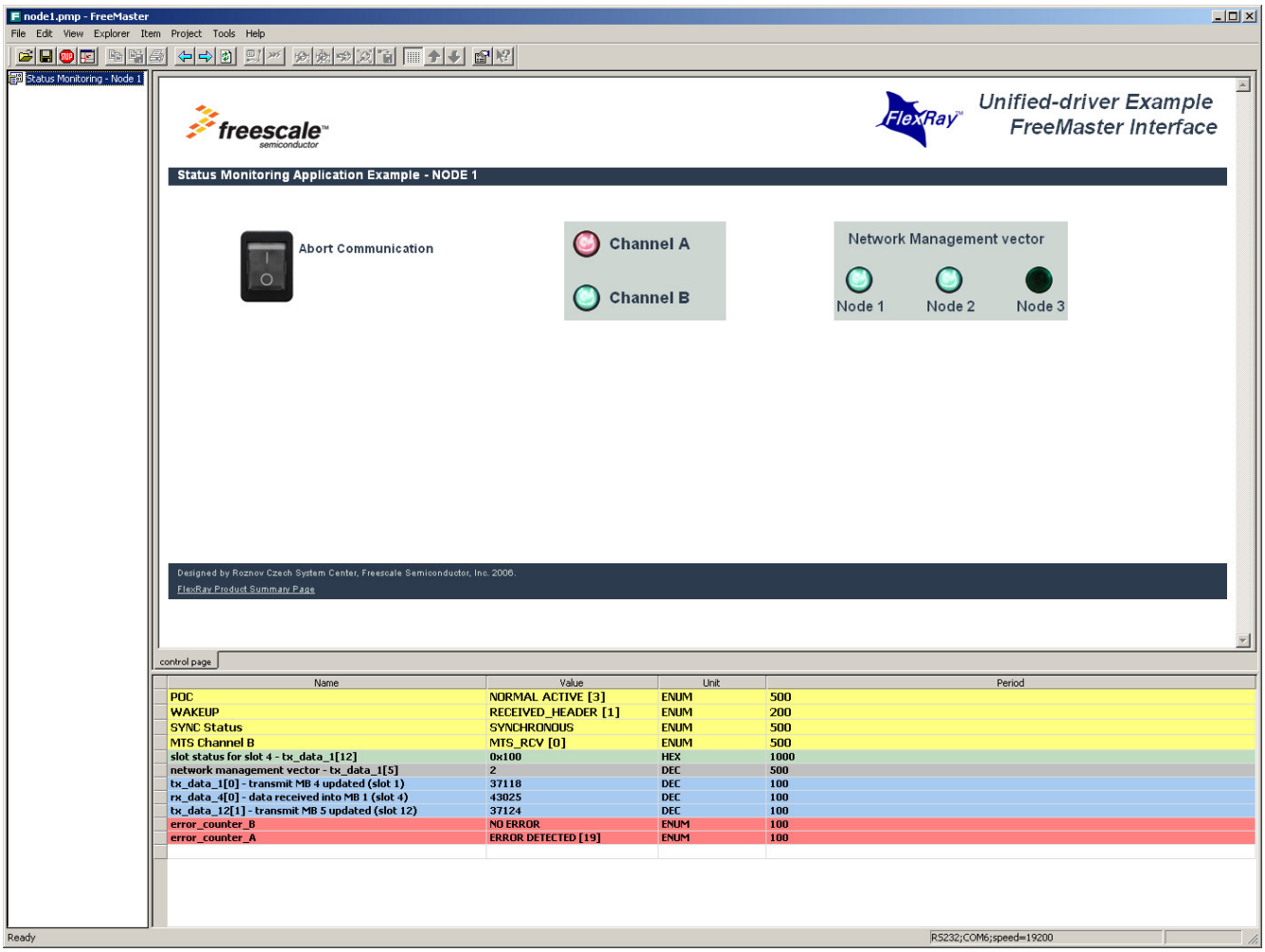

Figure 3-5. FreeMASTER Control Page for Node 1 of the Slot Status Application Example

3.3.6.2 Detailed Description of the FreeMASTER Control Page for Node 2

The FreeMASTER Node 2 control page visualises and controls the main features configured for Node 2 (see section 3.3.3 Configuration for Node 2), such as stop communication by calling the HALT protocol command, the transmission of the media test access symbol (MTS), usage of the network management vector, slot status, etc.

The FreeMASTER control page for Node 2 of the Status Monitoring application example consists of one switch, ‘Halt Communication’, controlling the HALT protocol command, one button, ‘Send MTS’, controlling the MTS transmission, and five indicators (represented by the illustrated LEDs) for monitoring the slot status and the network management vector.

By clicking on the ‘Halt Communication’ switch, the ***Fr_stop_communication****(FR_HALT_COMMUNICATION)* function is called in the host application, and the FlexRay module halts the FlexRay communication, and moves onto the *POC:halt* state at the end of

the current communication cycle. The *POC* variable in the FreeMASTER control page, which reflects the state of the *protocol_state* variable in the host application, will show the HALT[2] state. By next clicking ‘Restart’, the ***Fr_enter_configuration_mode()*** function and the *CC_init_and_startup()* routine are called to reconfigure the FlexRay module.

By clicking the ‘Send MTS’ button, the media test access symbol will be sent on channel B. The FreeMASTER control page for Node 1 should show the reception of the media test access symbol through the *MTS Channel B* variable, which reflects the state of the *mts_return_value* in the host application.

The channel A and Channel B indicators (represented by the illustrated LEDs) monitor the slot status, and show that channel A and channel B are connected, by a green color or disconnected, by a red.

Node 1, Node2, and Node 3 indicators monitor the network management vector status. These indicators show whether or not each individual node is connected. In this application example, the FlexRay cluster consists of two nodes, therefore only the Node 1 and Node 2 indicators can be activated.

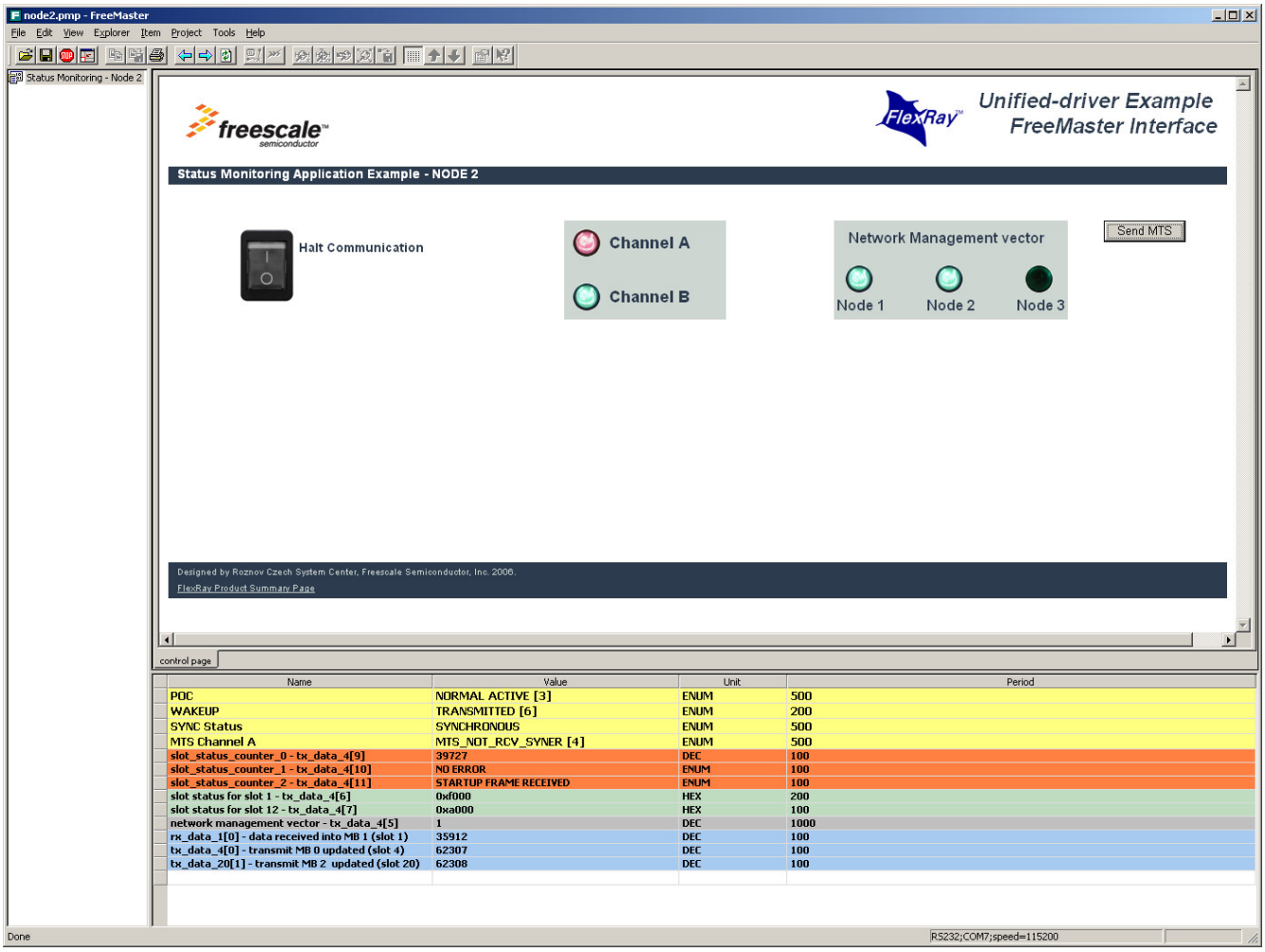

Figure 3-6. FreeMASTER Control Page for Node 2 of the Slot Status Application Example

Chapter 4 Application Example Project Structures

The application example project structures are slightly different for each compiler used, and therefore the description is separated into the following chapters.

4.1 CW Development Studio v1.5b2 for the MPC556x MCU

The following description is valid for both MPC5567 and MPC5561 platforms, which are uniformly called MPC556x.

By installing the FlexRay UNIFIED Driver package, a folder tree as shown in Figure 4-1. Application Example Folder Tree shall be created.

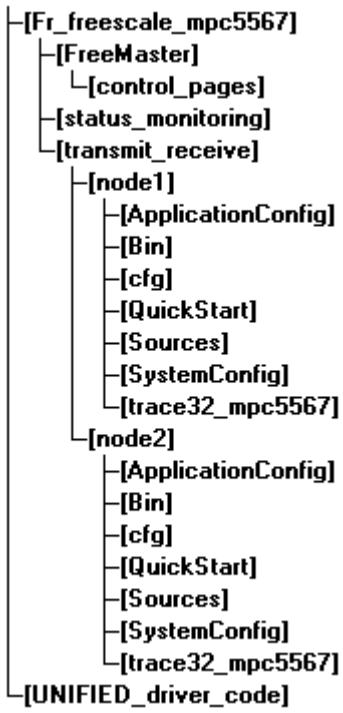

Figure 4-1. Application Example Folder Tree

The driver package contains the following files and folders:

- *UNIFIED_driver_code* folder - contains the general source code and header files. The FlexRay UNIFIED Driver also uses other three files stored under the *cfg* folder in each project
  - *Fr_UNIFIED.c* - the general source code
  - *Fr_UNIFIED.h* - the general header file with definition of the FlexRay registers, structures and C definition

- *fr_freescale_mpc556x* folder - contains application examples intended for the MPC556x Microcontroller (e.g. *fr_freescale_mpc5567* folder contains example intended for the MPC5567)
  – *transmit_receive* subfolder - contains the Transmit Receive application example

    a. *node1* subfolder - contains the application example for Node 1
       i. *ApplicationConfig* folder - contains MCU configuration data in the *appconfig.h* file
       ii. *cfg* folder - contains FlexRay module configuration data in the *Fr_UNIFIED_cfg.c* file, the declaration of the external variables/structures in the *Fr_UNIFIED_cfg.h* file and definition of the compiler types in the *Fr_UNIFIED_types.h* file.
       All these files may be modified by a user
       iii. *Sources* folder - contains application code in the *transmit_receive_node1.c* file (this can be modified by a user)
       iv. *QuickStart* folder - the project is based on standard Quick Start stationary, the FreeMASTER source files are included
       v. *trace32_mpc556x* folder - if the Lauterbach TRACE32 debugger is used, it is possible to use predefined **.cmm* files
       vi. *transmit_receive_node1.mcp* - the CW project file

    b. *node2* subfolder - contains the application example for Node 2
       i. this folder contains the same folders/files structure as in the *node1* folder. Application code is stored in the *transmit_receive_node2.c* file and the CW project file is named *transmit_receive_node2.mcp*

  – *status_monitoring* subfolder - contains the Status Monitoring application example.
    This folder contains the same folders/files structure as in the *transmit_receive* folder. Application codes are stored in the *status_monitoring_node1.c* and *status_monitoring_node2.c* files, and the CW project files are named *status_monitoring_node1.mcp* and *status_monitoring_node2.mcp*

  – *FreeMASTER* subfolder - contains the necessary files for the FreeMASTER tool and control pages

    a. *transmit_receive_node_1.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 1
    b. *transmit_receive_node_2.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 2
    c. *status_monitoring_node_1.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 1
    d. *status_monitoring_node_2.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 2
    e. *control_pages* folder - contains support files which are necessary for the FreeMASTER project files

i. *algorithm_block1.htm* - algorithm block description page for the *transmit_receive_node_1.pmp* FreeMASTER project file

ii. *algorithm_block2.htm* - algorithm block description page for the *transmit_receive_node_2.pmp* FreeMASTER project file

iii. *control_page1.htm* - control page for the *status_monitoring_node_1.pmp* FreeMASTER project file

iv. *control_page2.htm* - control page for the *status_monitoring_node_2.pmp* FreeMASTER project file.

The presented application example is structured in the CodeWarrior Development Studio as shown in Figure 4-2. CodeWarrior Project Tree.

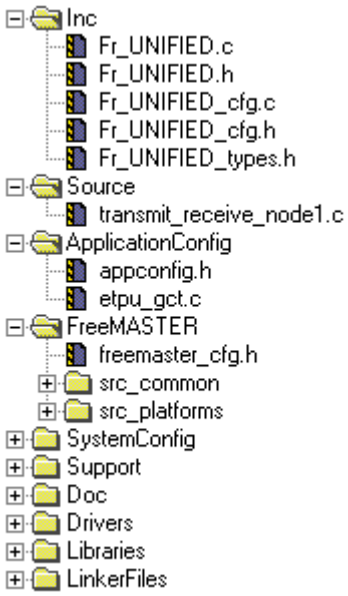

Figure 4-2. CodeWarrior Project Tree

The application examples were tested with the *ExtRam* and the *IntFlash* project targets for the MPC5567 and with the *IntFlash* project target for the MPC5561.

4.2 CW Development Studio v2.3 for the MPC5567 MCU

The following description is valid only for MPC5567.

By installing the FlexRay UNIFIED Driver package, a folder tree as shown in Figure 4-3. Application Example Folder Tree shall be created.

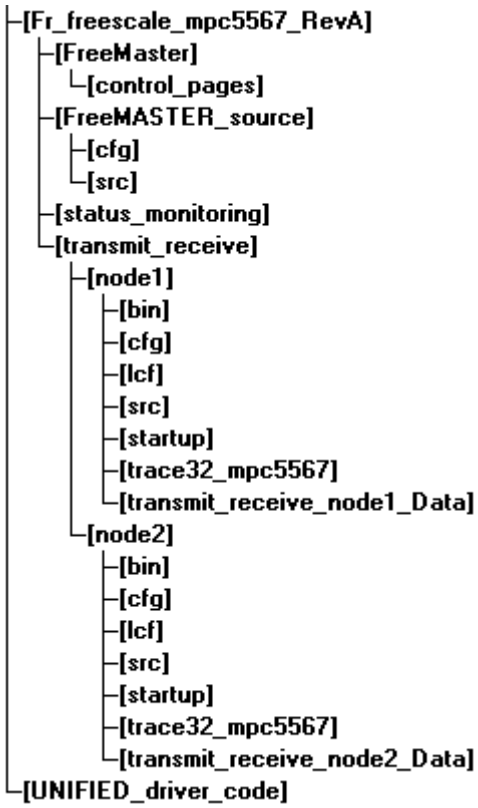

Figure 4-3. Application Example Folder Tree

The driver package contains the following files and folders:

- *UNIFIED_driver_code* folder - contains the general source code and header files. The FlexRay UNIFIED Driver also uses other three files stored under the *cfg* folder in each project
  - *Fr_UNIFIED.c* - the general source code
  - *Fr_UNIFIED.h* - the general header file with definition of the FlexRay registers, structures and C definition
- *fr_freescale_mpc5567_reva* folder - contains application examples for the MPC5567 Microcontroller
  - *transmit_receive* subfolder - contains the Transmit Receive application example

    a. *node1* subfolder - contains the application example for Node 1

       i. *startup* folder - contains MCU startup and basic configuration code based on CW stationary created using the MPC55xx New Project Wizard

ii. *cfg* folder - contains FlexRay module configuration data in the *Fr_UNIFIED_cfg.c* file, the declaration of the external variables/structures in the *Fr_UNIFIED_cfg.h* file, definition of the compiler types in the *Fr_UNIFIED_types.h* file and the FreeMASTER configuration in the *freemaster_cfg.h* file.
All these files may be modified by a user

iii. *Sources* folder - contains application code in the *transmit_receive_node1.c* file (this can be modified by a user)

iv. *trace32_mpc5567* folder - if the Lauterbach TRACE32 debugger is used, it is possible to use predefined **.cmm* files

v. *transmit_receive_node1.mcp* - the CW project file

b. *node2* subfolder - contains the application example for Node 2

i. this folder contains the same folders/files structure as in the *node1* folder. Application code is stored in the *transmit_receive_node2.c* file and the CW project file is named *transmit_receive_node2.mcp*

– *status_monitoring* subfolder - contains the Status Monitoring application example.
This folder contains the same folders/files structure as in the *transmit_receive* folder. Application codes are stored in the *status_monitoring_node1.c* and *status_monitoring_node2.c* files, and the CW project files are named *status_monitoring_node1.mcp* and *status_monitoring_node2.mcp*

– *FreeMASTER* subfolder - contains the necessary files for the FreeMASTER tool and control pages

a. *transmit_receive_node_1.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 1

b. *transmit_receive_node_2.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 2

c. *status_monitoring_node_1.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 1

d. *status_monitoring_node_2.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 2

e. *control_pages* folder - contains support files which are necessary for the FreeMASTER project files

i. *algorithm_block1.htm* - algorithm block description page for the *transmit_receive_node_1.pmp* FreeMASTER project file

ii. *algorithm_block2.htm* - algorithm block description page for the *transmit_receive_node_2.pmp* FreeMASTER project file

iii. *control_page1.htm* - control page for the *status_monitoring_node_1.pmp* FreeMASTER project file

iv. *control_page2.htm* - control page for the *status_monitoring_node_2.pmp* FreeMASTER project file.

– *FreeMASTER_source* subfolder - contains the necessary source files for the FreeMASTER tool

The presented application example is structured in the CodeWarrior Development Studio as shown in Figure 4-4. CodeWarrior Project Tree.

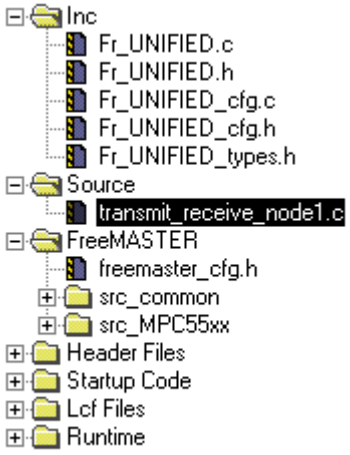

Figure 4-4. CodeWarrior Project Tree

The application examples were tested with the *IntFlash* project target.

4.3 GreenHills MULTI Project Builder for the MPC556x MCU

The following description is valid for both MPC5567 and MPC5561 platforms, which are uniformly called MPC556x.

By installing the FlexRay UNIFIED Driver package, a folder tree as shown in Figure 4-5. Application Example Folder Tree shall be created.

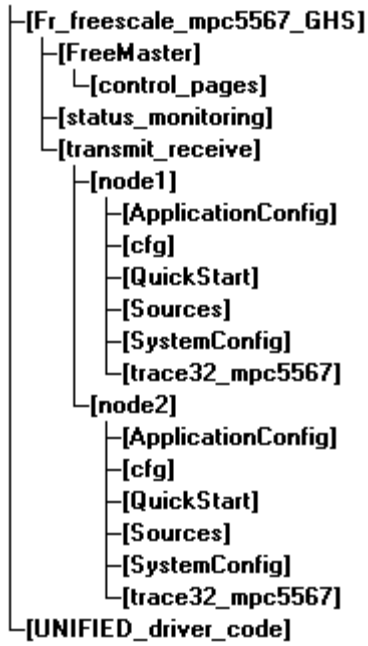

Figure 4-5. Application Example Folder Tree

The driver package contains the following files and folders:

- *UNIFIED_driver_code* folder - contains the general source code and header files. The FlexRay UNIFIED Driver also uses other three files stored under the *cfg* folder in each project
  - *Fr_UNIFIED.c* - the general source code
  - *Fr_UNIFIED.h* - the general header file with definition of the FlexRay registers, structures and C definition
- *fr_freescale_mpc556x_GHS* folder - contains application examples intended for the MPC556x Microcontroller (e.g. *fr_freescale_mpc5561_GHS* folder contains example intended for the MPC5561)

  – *transmit_receive* subfolder - contains the Transmit Receive application example

    a. *node1* subfolder - contains the application example for Node 1

      i. *ApplicationConfig* folder - contains MCU configuration data in the *appconfig.h* file

ii. *cfg* folder - contains FlexRay module configuration data in the *Fr_UNIFIED_cfg.c* file, the declaration of the external variables/structures in the *Fr_UNIFIED_cfg.h* file and definition of the compiler types in the *Fr_UNIFIED_types.h* file.
All these files may be modified by a user

iii. *Sources* folder - contains application code in the *transmit_receive_node1.c* file (this can be modified by a user)

iv. *QuickStart* folder - the project is based on standard Quick Start stationary, the FreeMASTER source files are included

v. the *trace32_mpc556x* folder - if the Lauterbach TRACE32 debugger is used, it is possible to use predefined **.cmm* files

vi. the *transmit_receive_node1.gpj* - the GreenHills MULTI Project Builder project file

vii. the *flexray.gpj* file - contains the definition of access paths to the included FlexRay UNIFIED Driver files and also to the application code file

b. *node2* subfolder - contains the application example for Node 2

i. this folder contains the same folders/files structure as in the *node1* folder. Application code is stored in the *transmit_receive_node2.c* file and the GreenHills MULTI project file is named *transmit_receive_node2.gpj*

– *status_monitoring* subfolder - contains the Status Monitoring application example.
This folder contains the same folders/files structure as in the *transmit_receive* folder. Application codes are stored in the *status_monitoring_node1.c* and *status_monitoring_node2.c* files and the GreenHills MULTI project files are named *status_monitoring_node1.gpj* and *status_monitoring_node2.gpj*

a. *transmit_receive_node_1.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 1

b. *transmit_receive_node_2.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 2

c. *status_monitoring_node_1.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 1

d. *status_monitoring_node_2.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 2

e. *control_pages* folder - contains support files which are necessary for the FreeMASTER project files

i. *algorithm_block1.htm* - algorithm block description page for the *transmit_receive_node_1.pmp* FreeMASTER project file

ii. *algorithm_block2.htm* - algorithm block description page for the *transmit_receive_node_2.pmp* FreeMASTER project file

iii. *control_page1.htm* - control page for the *status_monitoring_node_1.pmp* FreeMASTER project file

iv. *control_page2.htm* - control page for the *status_monitoring_node_2.pmp* FreeMASTER project file

– *FreeMASTER_source* subfolder - contains the FreeMASTER source files (header files, etc.).

The presented application example is structured in the GreenHills MULTI Project Builder as shown in Figure 4-6. GreenHills MULTI Project Builder Project Tree.

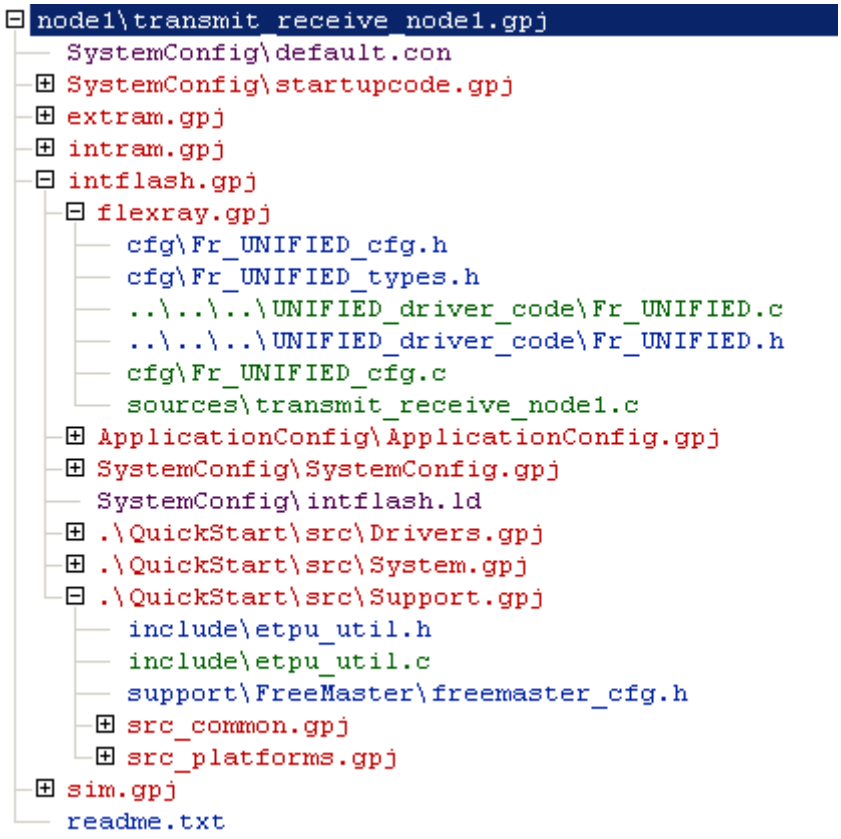

Figure 4-6. GreenHills MULTI Project Builder Project Tree

4.4 WindRiver DIAB C Compiler for the MPC556x MCU

The following description is valid for both MPC5567 and MPC5561 platforms, which are uniformly called MPC556x.

By installing the FlexRay UNIFIED Driver package, a folder tree as shown in Figure 4-7. Application Example Folder Tree shall be created.

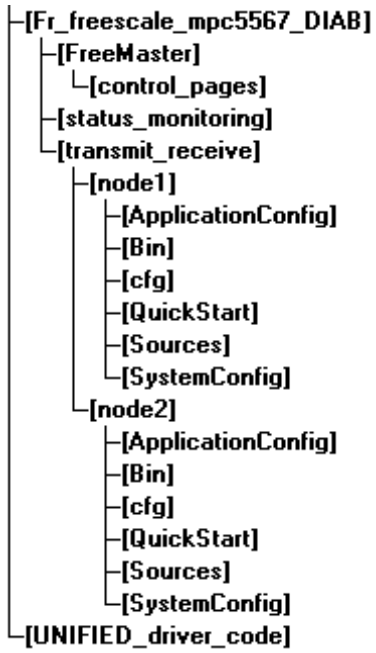

Figure 4-7. Application Example Folder Tree

The driver package contains the following files and folders:

- *UNIFIED_driver_code* folder - contains the general source code and header files. The FlexRay UNIFIED Driver also uses other three files stored under the *cfg* folder in each project
  - *Fr_UNIFIED.c* - the general source code
  - *Fr_UNIFIED.h* - the general header file with definition of the FlexRay registers, structures and C definition
- *fr_freescale_mpc556x_DIAB* folder - contains application examples intended for the MPC556x Microcontroller (e.g. *fr_freescale_mpc5561_DIAB* folder contains example intended for the MPC5561)

  – *transmit_receive* subfolder - contains the Transmit Receive application example

    a. *node1* subfolder - contains the application example for Node 1

      i. *ApplicationConfig* folder - contains MCU configuration data in the *appconfig.h* file

ii. *cfg* folder - contains FlexRay module configuration data in the *Fr_UNIFIED_cfg.c* file, the declaration of the external variables/structures in the *Fr_UNIFIED_cfg.h* file and definition of the compiler types in the *Fr_UNIFIED_types.h* file.
All these files may be modified by a user

iii. *Sources* folder - contains application code in the *transmit_receive_node1.c* file (this can be modified by a user)

iv. *QuickStart* folder - the project is based on standard Quick Start stationary, the FreeMASTER source files are included

v. *makefile* - project default makefile

vi. *runmake.bat* file - the file which runs the DIAB compiler

vii. *make.exe* file - *makefile* implementer

b. *node2* subfolder - contains the application example for Node 2

i. this folder contains the same folders/files structure as in the *node1* folder. Application code is stored in the *transmit_receive_node2.c* file

– *status_monitoring* subfolder - contains the Status Monitoring application example.
This folder contains the same folders/files structure as in the *transmit_receive* folder. Application codes are stored in the *status_monitoring_node1.c* and *status_monitoring_node2.c* files

a. *transmit_receive_node_1.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 1

b. *transmit_receive_node_2.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 2

c. *status_monitoring_node_1.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 1

d. *status_monitoring_node_2.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 2

e. *control_pages* folder - contains support files which are necessary for the FreeMASTER project files

i. *algorithm_block1.htm* - algorithm block description page for the *transmit_receive_node_1.pmp* FreeMASTER project file

ii. *algorithm_block2.htm* - algorithm block description page for the *transmit_receive_node_2.pmp* FreeMASTER project file

iii. *control_page1.htm* - control page for the *status_monitoring_node_1.pmp* FreeMASTER project file

iv. *control_page2.htm* - control page for the *status_monitoring_node_2.pmp* FreeMASTER project file

– *FreeMASTER_source* subfolder - contains the FreeMASTER source files (header files, etc.).

4.5 CW Development Studio for the MC9S12XFR128 MCU

By installing the FlexRay UNIFIED Driver package, a folder tree as shown in Figure 4-8. Application Example Folder Tree shall be created.

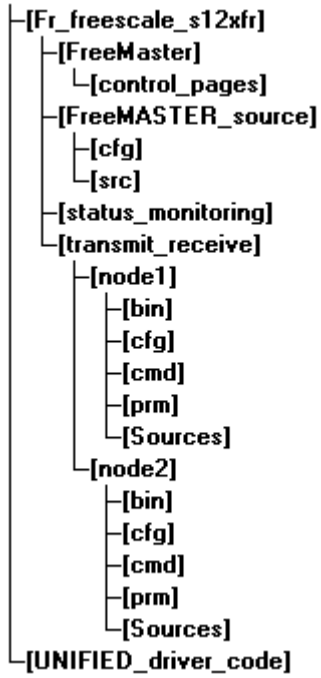

Figure 4-8. Application Example Folder Tree

The driver package contains the following files and folders:

- *UNIFIED_driver_code* folder - contains the general source code and header files. The FlexRay UNIFIED Driver also uses other three files stored under the *cfg* folder in each project
  - *Fr_UNIFIED.c* - the general source code
  - *Fr_UNIFIED.h* - the general header file with definition of the FlexRay registers, structures and C definition
- *fr_freescale_s12xfr* folder - contains application examples intended for the MC9S12XFR128 Microcontroller
  - *transmit_receive* subfolder - contains the Transmit Receive application example

    a. *node1* subfolder - contains the application example for Node 1

       i. *cfg* folder - contains FlexRay module configuration data in the *Fr_UNIFIED_cfg.c* file, the declaration of the external variables/structures in the *Fr_UNIFIED_cfg.h* file and definition of the compiler types in the *Fr_UNIFIED_types.h* file.
       All these files may be modified by a user

       ii. *prm* folder - contains linker parameter file for the MC9S12XFR128 Microcontroller

       iii. *Sources* folder - contains application code in the *transmit_receive_node1.c* file (this can be modified by a user)

iv. *transmit_receive_node1.mcp* - the CW project file

b. *node2* subfolder - contains the application example for Node 2

i. this folder contains the same folders/files structure as in the *node1* folder. Application code is stored in the *transmit_receive_node2.c* file and the CW project file is named *transmit_receive_node2.mcp*

– *status_monitoring* subfolder - contains the Status Monitoring application example. This folder contains the same folders/files structure as in the *transmit_receive* folder. Application codes are stored in the *status_monitoring_node1.c* and *status_monitoring_node2.c* files, and the CW project files are named *status_monitoring_node1.mcp* and *status_monitoring_node2.mcp*

– *FreeMASTER* subfolder - contains the necessary files for the FreeMASTER tool and control pages

a. *transmit_receive_node_1.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 1

b. *transmit_receive_node_2.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 2

c. *status_monitoring_node_1.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 1

d. *status_monitoring_node_2.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 2

e. *control_pages* folder - contains support files which are necessary for the FreeMASTER project files

i. *algorithm_block1.htm* - algorithm block description page for the *transmit_receive_node_1.pmp* FreeMASTER project file

ii. *algorithm_block2.htm* - algorithm block description page for the *transmit_receive_node_2.pmp* FreeMASTER project file

iii. *control_page1.htm* - control page for the *status_monitoring_node_1.pmp* FreeMASTER project file

iv. *control_page2.htm* - control page for the *status_monitoring_node_2.pmp* FreeMASTER project file

– *FreeMASTER_source* subfolder - contains the FreeMASTER source files (header files, etc.).

The presented application example is structured in the CodeWarrior Development Studio as shown in figure Figure 4-9. CodeWarrior Project Tree.

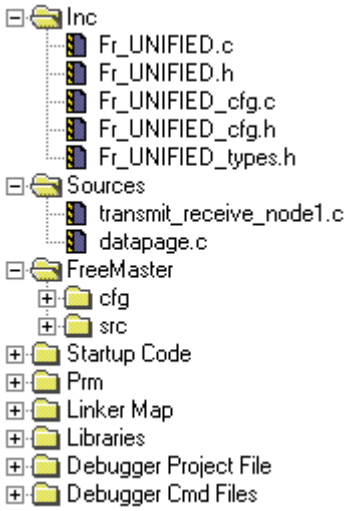

Figure 4-9. CodeWarrior Project Tree

4.6 CW Development Studio for the MC9S12XDP512 MCU with MFR43x0 CC

The following description for the MC9S12XDP512 MCU with a standalone communication controller is valid for both MFR4300 and MFR4310 communication controllers, which are uniformly called MFR43x0.

By installing the FlexRay UNIFIED Driver package, a folder tree as shown in Figure 4-10. Application Example Folder Tree shall be created.

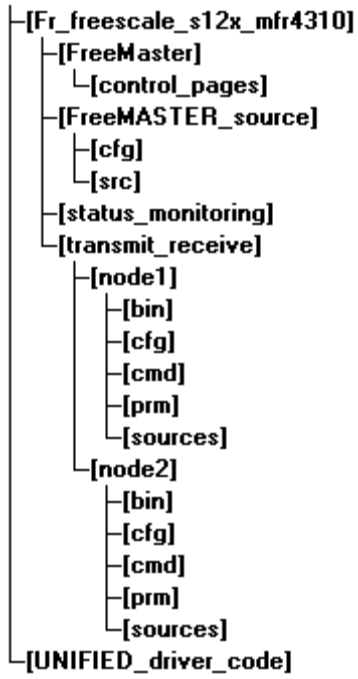

Figure 4-10. Application Example Folder Tree

The driver package contains the following files and folders:

- *UNIFIED_driver_code* folder - contains the general source code and header files. The FlexRay UNIFIED Driver also uses other three files stored under the *cfg* folder in each project
  - *Fr_UNIFIED.c* - the general source code
  - *Fr_UNIFIED.h* - the general header file with definition of the FlexRay registers, structures and C definition
- *fr_freescale_s12x_mfr43x0* folder - contains application examples intended for the MC9S12XDP512 Microcontroller with standalone MFR43x0 Communication Controller (e.g. *fr_freescale_s12x_mfr4310* folder contains example intended for the MC9S12XDP512 Microcontroller with standalone MFR4310 Communication Controller)
  - *transmit_receive* subfolder - contains the Transmit Receive application example

    a. *node1* subfolder - contains the application example for Node 1

       i. *cfg* folder - contains FlexRay module configuration data in the *Fr_UNIFIED_cfg.c* file, the declaration of the external variables/structures in the *Fr_UNIFIED_cfg.h* file and definition of the compiler types in the *Fr_UNIFIED_types.h* file.
       All these files may be modified by a user

ii. *prm* folder - contains linker parameter file for the MC9S12XDP512 Microcontroller

iii. *Sources* folder - contains application code in the *transmit_receive_node1.c* file (this can be modified by a user)

iv. *transmit_receive_node1.mcp* - the CW project file

b. *node2* subfolder - contains the application example for Node 2

i. this folder contains the same folders/files structure as in the *node1* folder. Application code is stored in the *transmit_receive_node2.c* file and the CW project file is named *transmit_receive_node2.mcp*

– *status_monitoring* subfolder - contains the Status Monitoring application example. This folder contains the same folders/files structure as in the *transmit_receive* folder. Application codes are stored in the *status_monitoring_node1.c* and *status_monitoring_node2.c* files, and the CW project files are named *status_monitoring_node1.mcp* and *status_monitoring_node2.mcp*

– *FreeMASTER* subfolder - contains the necessary files for the FreeMASTER tool and control pages

a. *transmit_receive_node_1.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 1

b. *transmit_receive_node_2.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 2

c. *status_monitoring_node_1.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 1

d. *status_monitoring_node_2.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 2

e. *control_pages* folder - contains support files which are necessary for the FreeMASTER project files

i. *algorithm_block1.htm* - algorithm block description page for the *transmit_receive_node_1.pmp* FreeMASTER project file

ii. *algorithm_block2.htm* - algorithm block description page for the *transmit_receive_node_2.pmp* FreeMASTER project file

iii. *control_page1.htm* - control page for the *status_monitoring_node_1.pmp* FreeMASTER project file

iv. *control_page2.htm* - control page for the *status_monitoring_node_2.pmp* FreeMASTER project file

– *FreeMASTER_source* subfolder - contains the FreeMASTER source files (header files, etc.).

The presented application example is structured in the CodeWarrior Development Studio as shown in figure Figure 4-11. CodeWarrior Project Tree.

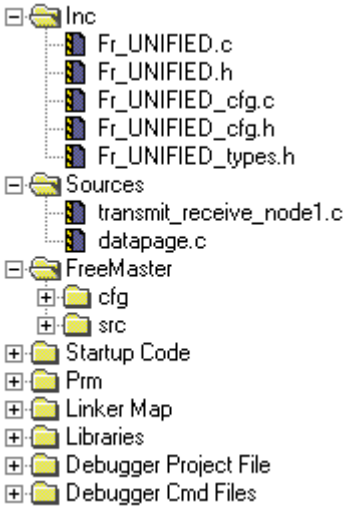

Figure 4-11. CodeWarrior Project Tree

4.7 CW Development Studio for the MPC5554 MCU with MFR43x0 CC

The following description for the MPC5554 MCU with a standalone communication controller is valid for both MFR4300 and MFR4310 communication controllers, which are uniformly called MFR43x0.

By installing the FlexRay UNIFIED Driver package, a folder tree as shown in Figure 4-12. Application Example Folder Tree shall be created.

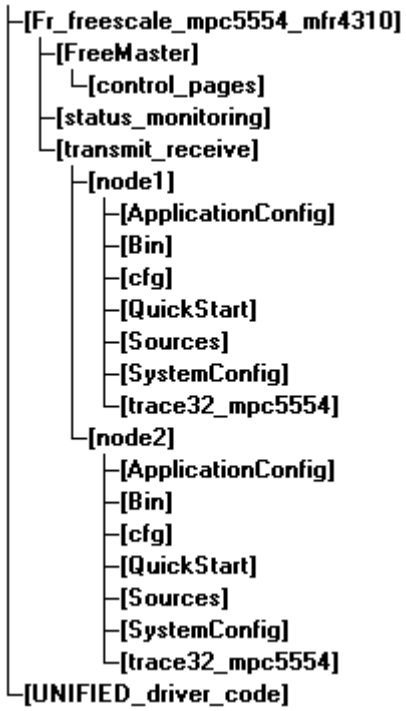

Figure 4-12. Application Example Folder Tree

The driver package contains the following files and folders:

- *UNIFIED_driver_code* folder - contains the general source code and header files. The FlexRay UNIFIED Driver also uses other three files stored under the *cfg* folder in each project
  - *Fr_UNIFIED.c* - the general source code
  - *Fr_UNIFIED.h* - the general header file with definition of the FlexRay registers, structures and C definition
- *fr_freescale_mpc5554_mfr43x0* folder - contains application examples intended for the MPC5554 Microcontroller with standalone MFR43x0 Communication Controller (e.g. *fr_freescale_mpc5554_mfr4310* folder contains example intended for the MPC5554 Microcontroller with standalone MFR4310 Communication Controller)
  - *transmit_receive* subfolder - contains the Transmit Receive application example

    a. *node1* subfolder - contains the application example for Node 1

       i. *ApplicationConfig* folder - contains MCU configuration data in the *appconfig.h* file

ii. *cfg* folder - contains FlexRay module configuration data in the *Fr_UNIFIED_cfg.c* file, the declaration of the external variables/structures in the *Fr_UNIFIED_cfg.h* file and definition of the compiler types in the *Fr_UNIFIED_types.h* file.
All these files may be modified by a user

iii. *Sources* folder - contains application code in the *transmit_receive_node1.c* file (this can be modified by a user)

iv. *QuickStart* folder - the project is based on standard Quick Start stationary, the FreeMASTER source files are included

v. *trace32_mpc5554* folder - if the Lauterbach TRACE32 debugger is used, it is possible to use predefined **.cmm* files

vi. *transmit_receive_node1.mcp* - the CW project file

b. *node2* subfolder - contains the application example for Node 2

i. this folder contains the same folders/files structure as in the *node1* folder. Application code is stored in the *transmit_receive_node2.c* file and the CW project file is named *transmit_receive_node2.mcp*

– *status_monitoring* subfolder - contains the Status Monitoring application example.
This folder contains the same folders/files structure as in the *transmit_receive* folder. Application codes are stored in the *status_monitoring_node1.c* and *status_monitoring_node2.c* files, and the CW project files are named *status_monitoring_node1.mcp* and *status_monitoring_node2.mcp*

– *FreeMASTER* subfolder - contains the necessary files for the FreeMASTER tool and control pages

a. *transmit_receive_node_1.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 1

b. *transmit_receive_node_2.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 2

c. *status_monitoring_node_1.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 1

d. *status_monitoring_node_2.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 2

e. *control_pages* folder - contains support files which are necessary for the FreeMASTER project files

i. *algorithm_block1.htm* - algorithm block description page for the *transmit_receive_node_1.pmp* FreeMASTER project file

ii. *algorithm_block2.htm* - algorithm block description page for the *transmit_receive_node_2.pmp* FreeMASTER project file

iii. *control_page1.htm* - control page for the *status_monitoring_node_1.pmp* FreeMASTER project file

iv. *control_page2.htm* - control page for the *status_monitoring_node_2.pmp* FreeMASTER project file

– *FreeMASTER_source* subfolder - contains the FreeMASTER source files (header files, etc.).

The presented application example is structured in the CodeWarrior Development Studio as shown in figure Figure 4-13. CodeWarrior Project Tree.

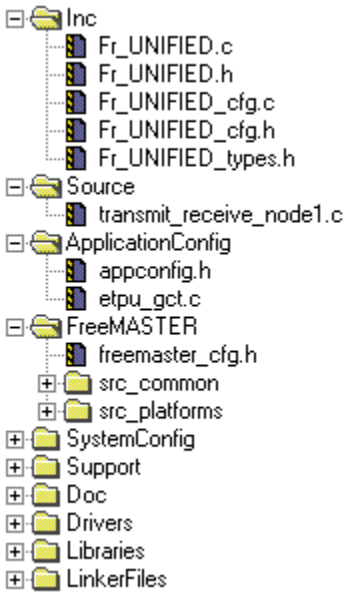

Figure 4-13. CodeWarrior Project Tree

4.8 WindRiver DIAB C Compiler for the MPC5554 MCU with MFR43x0 CC

The following description for the MPC5554 MCU with a standalone communication controller is valid for both MFR4300 and MFR4310 communication controllers, which are uniformly called MFR43x0

By installing the FlexRay UNIFIED Driver package, a folder tree as shown in Figure 4-14. Application Example Folder Tree shall be created.

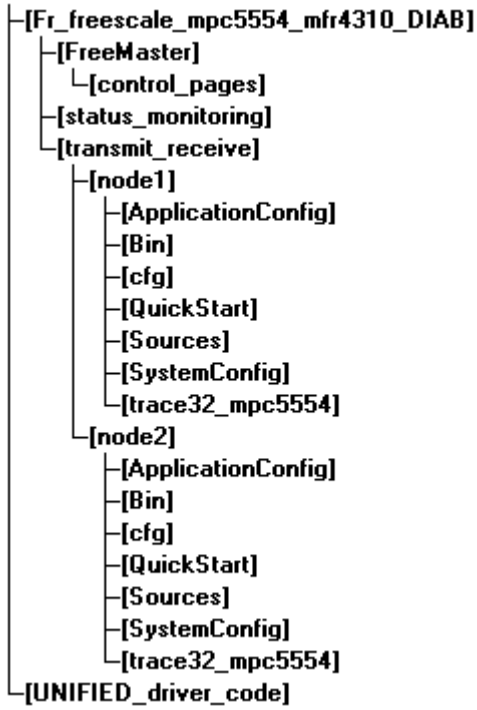

Figure 4-14. Application Example Folder Tree

The driver package contains the following files and folders:

- *UNIFIED_driver_code* folder - contains the general source code and header files. The FlexRay UNIFIED Driver also uses other three files stored under the *cfg* folder in each project
  - *Fr_UNIFIED.c* - the general source code
  - *Fr_UNIFIED.h* - the general header file with definition of the FlexRay registers, structures and C definition
- *fr_freescale_mpc5554_mfr43x0_DIAB* folder - contains application examples intended for the MPC5554 Microcontroller with standalone MFR43x0 Communication Controller (e.g. *fr_freescale_mpc5554_mfr4310_DIAB* folder contains example intended for the MPC5554 Microcontroller with standalone MFR4310 Communication Controller)

  – *transmit_receive* subfolder - contains the Transmit Receive application example

    a. *node1* subfolder - contains the application example for Node 1

       i. *ApplicationConfig* folder - contains MCU configuration data in the *appconfig.h* file

ii. *cfg* folder - contains FlexRay module configuration data in the *Fr_UNIFIED_cfg.c* file, the declaration of the external variables/structures in the *Fr_UNIFIED_cfg.h* file and definition of the compiler types in the *Fr_UNIFIED_types.h* file.
All these files may be modified by a user

iii. *Sources* folder - contains application code in the *transmit_receive_node1.c* file (this can be modified by a user)

iv. *QuickStart* folder - the project is based on standard Quick Start stationary, the FreeMASTER source files are included

v. *makefile* - project default makefile

vi. *runmake.bat* file - the file which runs the DIAB compiler

vii. *make.exe* file - *makefile* implementer

b. *node2* subfolder - contains the application example for Node 2

i. this folder contains the same folders/files structure as in the *node1* folder. Application code is stored in the *transmit_receive_node2.c* file

– *status_monitoring* subfolder - contains the Status Monitoring application example.
This folder contains the same folders/files structure as in the *transmit_receive* folder. Application codes are stored in the *status_monitoring_node1.c* and *status_monitoring_node2.c* files

a. *transmit_receive_node_1.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 1

b. *transmit_receive_node_2.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 2

c. *status_monitoring_node_1.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 1

d. *status_monitoring_node_2.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 2

e. *control_pages* folder - contains support files which are necessary for the FreeMASTER project files

i. *algorithm_block1.htm* - algorithm block description page for the *transmit_receive_node_1.pmp* FreeMASTER project file

ii. *algorithm_block2.htm* - algorithm block description page for the *transmit_receive_node_2.pmp* FreeMASTER project file

iii. *control_page1.htm* - control page for the *status_monitoring_node_1.pmp* FreeMASTER project file

iv. *control_page2.htm* - control page for the *status_monitoring_node_2.pmp* FreeMASTER project file

– *FreeMASTER_source* subfolder - contains the FreeMASTER source files (header files, etc.).

4.9 GreenHills MULTI Project Builder for the MPC5554 MCU with MFR43x0 CC

The following description for the MPC5554 MCU with a standalone communication controller is valid for both MFR4300 and MFR4310 communication controllers, which are uniformly called MFR43x0.

By installing the FlexRay UNIFIED Driver package, a folder tree as shown in Figure 4-15. Application Example Folder Tree shall be created.

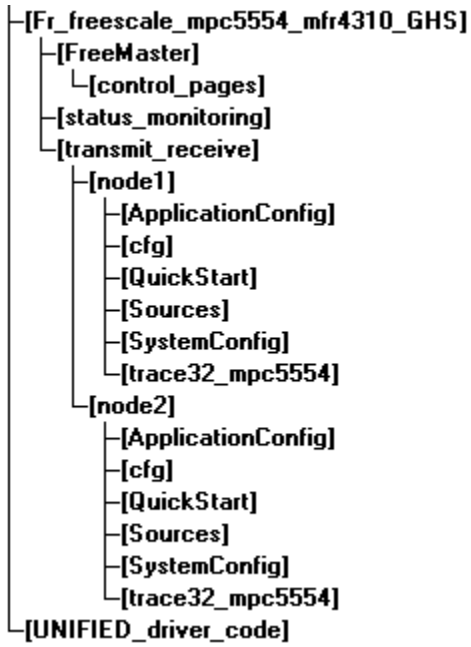

Figure 4-15. Application Example Folder Tree

The driver package contains the following files and folders:

- *UNIFIED_driver_code* folder - contains the general source code and header files. The FlexRay UNIFIED Driver also uses other three files stored under the *cfg* folder in each project
  - *Fr_UNIFIED.c* - the general source code
  - *Fr_UNIFIED.h* - the general header file with definition of the FlexRay registers, structures and C definition
- *fr_freescale_mpc5554_mfr43x0_GHS* folder - contains application examples intended for the MPC5554 Microcontroller with standalone MFR43x0 Communication Controller(e.g. *fr_freescale_mpc5554_mfr4310_GHS* folder contains example intended for the MPC5554 Microcontroller with standalone MFR4310 Communication Controller)

  – *transmit_receive* subfolder - contains the Transmit Receive application example

    a. *node1* subfolder - contains the application example for Node 1

       i. *ApplicationConfig* folder - contains MCU configuration data in the *appconfig.h* file

ii. *cfg* folder - contains FlexRay module configuration data in the *Fr_UNIFIED_cfg.c* file, the declaration of the external variables/structures in the *Fr_UNIFIED_cfg.h* file and definition of the compiler types in the *Fr_UNIFIED_types.h* file.
All these files may be modified by a user

iii. *Sources* folder - contains application code in the *transmit_receive_node1.c* file (this can be modified by a user)

iv. *QuickStart* folder - the project is based on standard Quick Start stationary, the FreeMASTER source files are included

v. the *trace32_mpc5554* folder - if the Lauterbach TRACE32 debugger is used, it is possible to use predefined **.cmm* files

vi. the *transmit_receive_node1.gpj* - the GreenHills MULTI Project Builder project file

vii. the *flexray.gpj* file - contains the definition of access paths to the included FlexRay UNIFIED Driver files and also to the application code file

b. *node2* subfolder - contains the application example for Node 2

i. this folder contains the same folders/files structure as in the *node1* folder. Application code is stored in the *transmit_receive_node2.c* file and the GreenHills MULTI project file is named *transmit_receive_node2.gpj*

– *status_monitoring* subfolder - contains the Status Monitoring application example.
This folder contains the same folders/files structure as in the *transmit_receive* folder. Application codes are stored in the *status_monitoring_node1.c* and *status_monitoring_node2.c* files and the GreenHills MULTI project files are named *status_monitoring_node1.gpj* and *status_monitoring_node2.gpj*

a. *transmit_receive_node_1.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 1

b. *transmit_receive_node_2.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 2

c. *status_monitoring_node_1.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 1

d. *status_monitoring_node_2.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 2

e. *control_pages* folder - contains support files which are necessary for the FreeMASTER project files

i. *algorithm_block1.htm* - algorithm block description page for the *transmit_receive_node_1.pmp* FreeMASTER project file

ii. *algorithm_block2.htm* - algorithm block description page for the *transmit_receive_node_2.pmp* FreeMASTER project file

iii. *control_page1.htm* - control page for the *status_monitoring_node_1.pmp* FreeMASTER project file

iv. *control_page2.htm* - control page for the *status_monitoring_node_2.pmp* FreeMASTER project file

– *FreeMASTER_source* subfolder - contains the FreeMASTER source files (header files, etc.).

The presented application example is structured in the GreenHills MULTI Project Builder as shown in Figure 4-16. GreenHills MULTI Project Builder Project Tree.

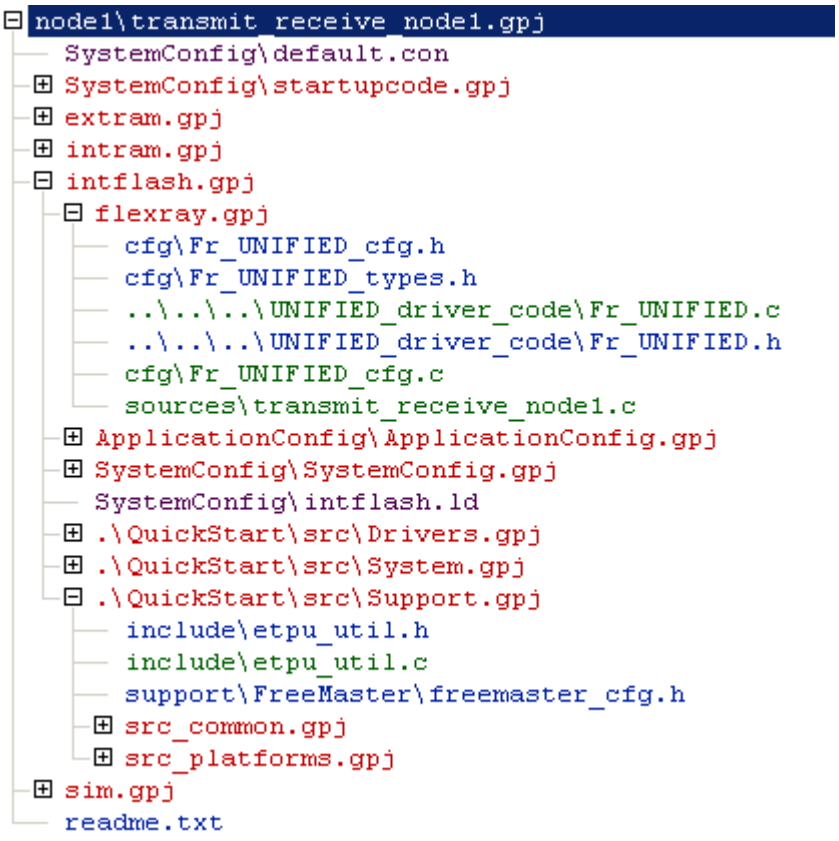

Figure 4-16. GreenHills MULTI Project Builder Project Tree

4.10 WindRiver DIAB C Compiler for the MPC5516 MCU

By installing the FlexRay UNIFIED Driver package, a folder tree as shown in Figure 4-17. Application Example Folder Tree shall be created.

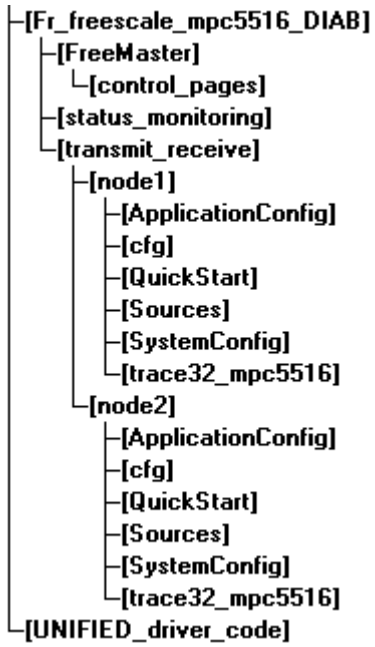

Figure 4-17. Application Example Folder Tree

The driver package contains the following files and folders:

- *UNIFIED_driver_code* folder - contains the general source code and header files. The FlexRay UNIFIED Driver also uses other three files stored under the *cfg* folder in each project
  - *Fr_UNIFIED.c* - the general source code
  - *Fr_UNIFIED.h* - the general header file with definition of the FlexRay registers, structures and C definition
- *fr_freescale_mpc5516_DIAB* folder - contains application examples intended for the MPC5516 Microcontroller
  - *transmit_receive* subfolder - contains the Transmit Receive application example
    a. *node1* subfolder - contains the application example for Node 1
       i. *ApplicationConfig* folder - contains MCU configuration data in the *appconfig.h* file
       ii. *cfg* folder - contains FlexRay module configuration data in the *Fr_UNIFIED_cfg.c* file, the declaration of the external variables/structures in the *Fr_UNIFIED_cfg.h* file and definition of the compiler types in the *Fr_UNIFIED_types.h* file.
       All these files may be modified by a user
       iii. *Sources* folder - contains application code in the *transmit_receive_node1.c* file (this can be modified by a user)

iv. *QuickStart* folder - the project is based on standard Quick Start stationary, the FreeMASTER source files are included

v. *makefile* - project default makefile

vi. *runmake.bat* file - the file which runs the DIAB compiler

vii. *make.exe* file - *makefile* implementer

b. *node2* subfolder - contains the application example for Node 2

i. this folder contains the same folders/files structure as in the *node1* folder. Application code is stored in the *transmit_receive_node2.c* file

– *status_monitoring* subfolder - contains the Status Monitoring application example. This folder contains the same folders/files structure as in the *transmit_receive* folder. Application codes are stored in the *status_monitoring_node1.c* and *status_monitoring_node2.c* files

a. *transmit_receive_node_1.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 1

b. *transmit_receive_node_2.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 2

c. *status_monitoring_node_1.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 1

d. *status_monitoring_node_2.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 2

e. *control_pages* folder - contains support files which are necessary for the FreeMASTER project files

i. *algorithm_block1.htm* - algorithm block description page for the *transmit_receive_node_1.pmp* FreeMASTER project file

ii. *algorithm_block2.htm* - algorithm block description page for the *transmit_receive_node_2.pmp* FreeMASTER project file

iii. *control_page1.htm* - control page for the *status_monitoring_node_1.pmp* FreeMASTER project file

iv. *control_page2.htm* - control page for the *status_monitoring_node_2.pmp* FreeMASTER project file

FreeMASTER_source subfolder - contains the FreeMASTER source files (header files, etc.).

4.11 GreenHills MULTI Project Builder for the MPC5516 MCU

By installing the FlexRay UNIFIED Driver package, a folder tree as shown in Figure 4-18. Application Example Folder Tree shall be created

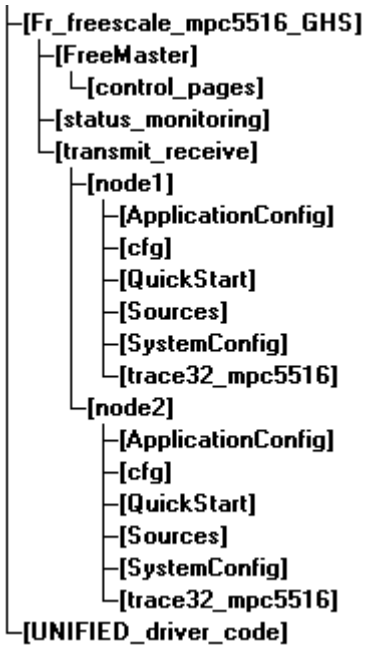

Figure 4-18. Application Example Folder Tree

The driver package contains the following files and folders:

- *UNIFIED_driver_code* folder - contains the general source code and header files. The FlexRay UNIFIED Driver also uses other three files stored under the *cfg* folder in each project
  - *Fr_UNIFIED.c* - the general source code
  - *Fr_UNIFIED.h* - the general header file with definition of the FlexRay registers, structures and C definition
- *fr_freescale_mpc5516_GHS* folder - contains application examples intended for the MPC5516 Microcontroller
  - *transmit_receive* subfolder - contains the Transmit Receive application example
    a. *node1* subfolder - contains the application example for Node 1
       i. *ApplicationConfig* folder - contains MCU configuration data in the *appconfig.h* file
       ii. *cfg* folder - contains FlexRay module configuration data in the *Fr_UNIFIED_cfg.c* file, the declaration of the external variables/structures in the *Fr_UNIFIED_cfg.h* file and definition of the compiler types in the *Fr_UNIFIED_types.h* file.
       All these files may be modified by a user
       iii. *Sources* folder - contains application code in the *transmit_receive_node1.c* file (this can be modified by a user)

iv. *QuickStart* folder - the project is based on standard Quick Start stationary, the FreeMASTER source files are included

v. the *trace32_mpc5516* folder - if the Lauterbach TRACE32 debugger is used, it is possible to use predefined **.cmm* files

vi. the *transmit_receive_node1.gpj* - the GreenHills MULTI Project Builder project file

vii. the *flexray.gpj* file - contains the definition of access paths to the included FlexRay UNIFIED Driver files and also to the application code file

b. *node2* subfolder - contains the application example for Node 2

i. this folder contains the same folders/files structure as in the *node1* folder. Application code is stored in the *transmit_receive_node2.c* file and the GreenHills MULTI project file is named *transmit_receive_node2.gpj*

– *status_monitoring* subfolder - contains the Status Monitoring application example. This folder contains the same folders/files structure as in the *transmit_receive* folder. Application codes are stored in the *status_monitoring_node1.c* and *status_monitoring_node2.c* files and the GreenHills MULTI project files are named *status_monitoring_node1.gpj* and *status_monitoring_node2.gpj*

a. *transmit_receive_node_1.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 1

b. *transmit_receive_node_2.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 2

c. *status_monitoring_node_1.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 1

d. *status_monitoring_node_2.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 2

e. *control_pages* folder - contains support files which are necessary for the FreeMASTER project files

i. *algorithm_block1.htm* - algorithm block description page for the *transmit_receive_node_1.pmp* FreeMASTER project file

ii. *algorithm_block2.htm* - algorithm block description page for the *transmit_receive_node_2.pmp* FreeMASTER project file

iii. *control_page1.htm* - control page for the *status_monitoring_node_1.pmp* FreeMASTER project file

iv. *control_page2.htm* - control page for the *status_monitoring_node_2.pmp* FreeMASTER project file

– *FreeMASTER_source* subfolder - contains the FreeMASTER source files (header files, etc.).

The presented application example is structured in the GreenHills MULTI Project Builder as shown in Figure 4-19. GreenHills MULTI Project Builder Project Tree

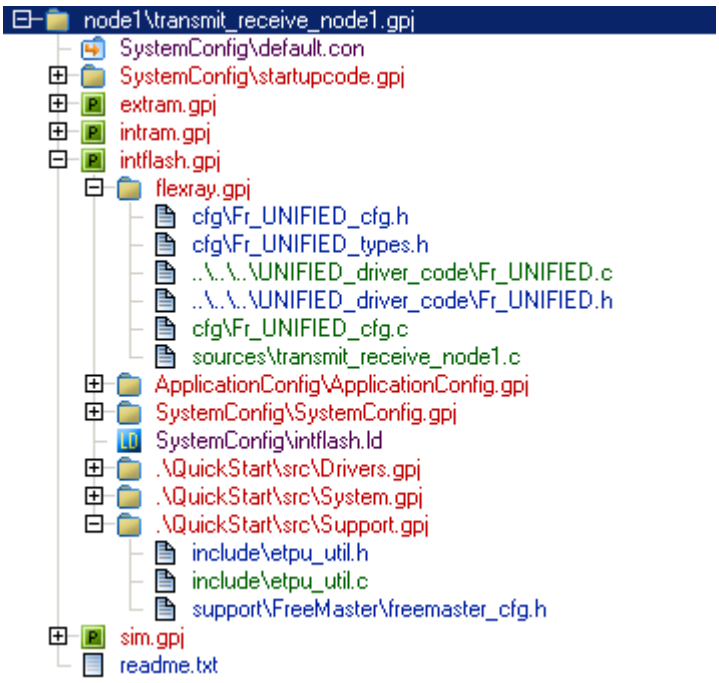

Figure 4-19. GreenHills MULTI Project Builder Project Tree

4.12 CW Development Studio for the MC9S12XF512/384/256/128 MCU

By installing the FlexRay UNIFIED Driver package, a folder tree as shown in Figure 4-20. Application Example Folder Tree shall be created.

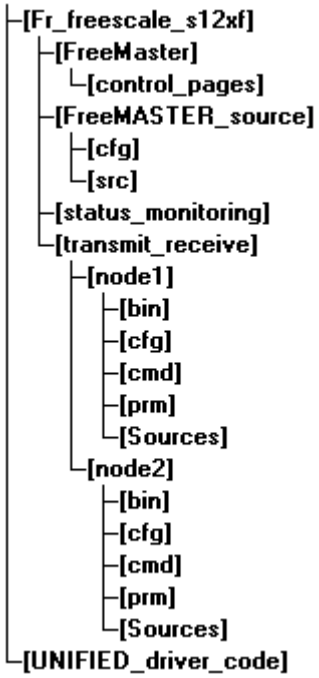

Figure 4-20. Application Example Folder Tree

The driver package contains the following files and folders:

- *UNIFIED_driver_code* folder - contains the general source code and header files. The FlexRay UNIFIED Driver also uses other three files stored under the *cfg* folder in each project
  - *Fr_UNIFIED.c* - the general source code
  - *Fr_UNIFIED.h* - the general header file with definition of the FlexRay registers, structures and C definition
- *fr_freescale_s12xf* folder - contains application examples intended for the MC9S12XF512/384/256/128 Microcontrollers
  - *transmit_receive* subfolder - contains the Transmit Receive application example

    a. *node1* subfolder - contains the application example for Node 1

       i. *cfg* folder - contains FlexRay module configuration data in the *Fr_UNIFIED_cfg.c* file, the declaration of the external variables/structures in the *Fr_UNIFIED_cfg.h* file and definition of the compiler types in the *Fr_UNIFIED_types.h* file.
       All these files may be modified by a user

       ii. *prm* folder - contains linker parameter file for the MC9S12XF512/384/256/128 Microcontrollers

       iii. *Sources* folder - contains application code in the *transmit_receive_node1.c* file (this can be modified by a user)

iv. *transmit_receive_node1.mcp* - the CW project file

b. *node2* subfolder - contains the application example for Node 2

i. this folder contains the same folders/files structure as in the *node1* folder. Application code is stored in the *transmit_receive_node2.c* file and the CW project file is named *transmit_receive_node2.mcp*

– *status_monitoring* subfolder - contains the Status Monitoring application example. This folder contains the same folders/files structure as in the *transmit_receive* folder. Application codes are stored in the *status_monitoring_node1.c* and *status_monitoring_node2.c* files, and the CW project files are named *status_monitoring_node1.mcp* and *status_monitoring_node2.mcp*

– *FreeMASTER* subfolder - contains the necessary files for the FreeMASTER tool and control pages

a. *transmit_receive_node_1.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 1

b. *transmit_receive_node_2.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 2

c. *status_monitoring_node_1.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 1

d. *status_monitoring_node_2.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 2

e. *control_pages* folder - contains support files which are necessary for the FreeMASTER project files

i. *algorithm_block1.htm* - algorithm block description page for the *transmit_receive_node_1.pmp* FreeMASTER project file

ii. *algorithm_block2.htm* - algorithm block description page for the *transmit_receive_node_2.pmp* FreeMASTER project file

iii. *control_page1.htm* - control page for the *status_monitoring_node_1.pmp* FreeMASTER project file

iv. *control_page2.htm* - control page for the *status_monitoring_node_2.pmp* FreeMASTER project file

– *FreeMASTER_source* subfolder - contains the FreeMASTER source files (header files, etc.).

The presented application example is structured in the CodeWarrior Development Studio as shown in figure Figure 4-21. CodeWarrior Project Tree.

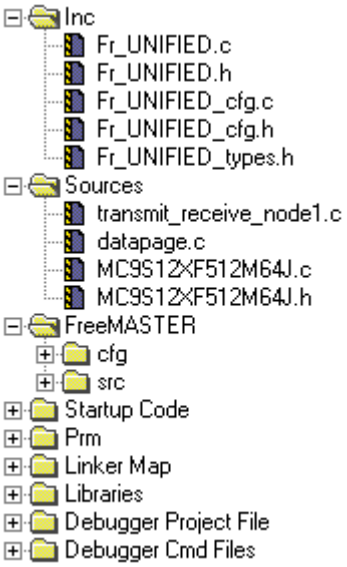

Figure 4-21. CodeWarrior Project Tree

4.13 CW Development Studio v2.3 for the MPC5517 MCU

The following description is valid only for MPC5517.

By installing the FlexRay UNIFIED Driver package, a folder tree as shown in Figure 4-22. Application Example Folder Tree shall be created.

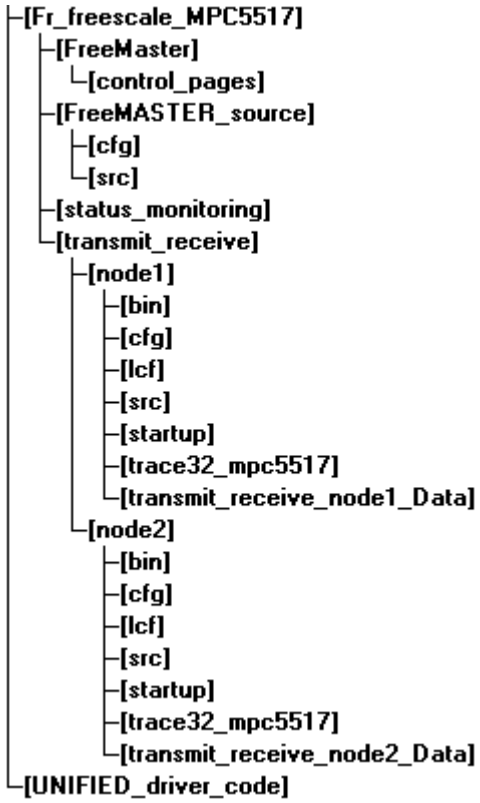

Figure 4-22. Application Example Folder Tree

The driver package contains the following files and folders:

- *UNIFIED_driver_code* folder - contains the general source code and header files. The FlexRay UNIFIED Driver also uses other three files stored under the *cfg* folder in each project
  - *Fr_UNIFIED.c* - the general source code
  - *Fr_UNIFIED.h* - the general header file with definition of the FlexRay registers, structures and C definition
- *fr_freescale_mpc5517* folder - contains application examples for the MPC5517 Microcontroller
  - *transmit_receive* subfolder - contains the Transmit Receive application example

    a. *node1* subfolder - contains the application example for Node 1

       i. *startup* folder - contains MCU startup and basic configuration code based on CW stationary created using the MPC55xx New Project Wizard

ii. *cfg* folder - contains FlexRay module configuration data in the *Fr_UNIFIED_cfg.c* file, the declaration of the external variables/structures in the *Fr_UNIFIED_cfg.h* file, definition of the compiler types in the *Fr_UNIFIED_types.h* file and the FreeMASTER configuration in the *freemaster_cfg.h* file.
All these files may be modified by a user

iii. *Sources* folder - contains application code in the *transmit_receive_node1.c* file (this can be modified by a user)

iv. *trace32_mpc5517* folder - if the Lauterbach TRACE32 debugger is used, it is possible to use predefined **.cmm* files

v. *transmit_receive_node1.mcp* - the CW project file

b. *node2* subfolder - contains the application example for Node 2

i. this folder contains the same folders/files structure as in the *node1* folder. Application code is stored in the *transmit_receive_node2.c* file and the CW project file is named *transmit_receive_node2.mcp*

– *status_monitoring* subfolder - contains the Status Monitoring application example.
This folder contains the same folders/files structure as in the *transmit_receive* folder. Application codes are stored in the *status_monitoring_node1.c* and *status_monitoring_node2.c* files, and the CW project files are named *status_monitoring_node1.mcp* and *status_monitoring_node2.mcp*

– *FreeMASTER* subfolder - contains the necessary files for the FreeMASTER tool and control pages

a. *transmit_receive_node_1.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 1

b. *transmit_receive_node_2.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 2

c. *status_monitoring_node_1.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 1

d. *status_monitoring_node_2.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 2

e. *control_pages* folder - contains support files which are necessary for the FreeMASTER project files

i. *algorithm_block1.htm* - algorithm block description page for the *transmit_receive_node_1.pmp* FreeMASTER project file

ii. *algorithm_block2.htm* - algorithm block description page for the *transmit_receive_node_2.pmp* FreeMASTER project file

iii. *control_page1.htm* - control page for the *status_monitoring_node_1.pmp* FreeMASTER project file

iv. *control_page2.htm* - control page for the *status_monitoring_node_2.pmp* FreeMASTER project file.

– *FreeMASTER_source* subfolder - contains the necessary source files for the FreeMASTER tool

The presented application example is structured in the CodeWarrior Development Studio as shown in Figure 4-23. CodeWarrior Project Tree.

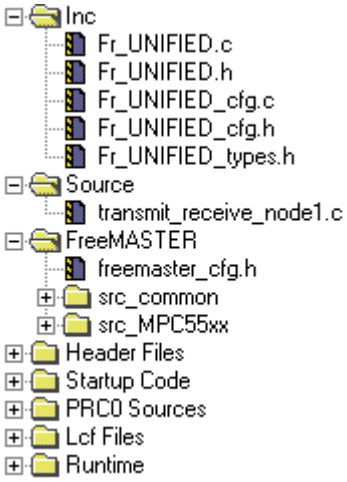

Figure 4-23. CodeWarrior Project Tree

The application examples were tested with the *IntFlash* project target.

4.14 GreenHills MULTI Project Builder for the MPC560xP MCU

By installing the FlexRay UNIFIED Driver package, a folder tree as shown in Figure 4-24. Application Example Folder Tree shall be created

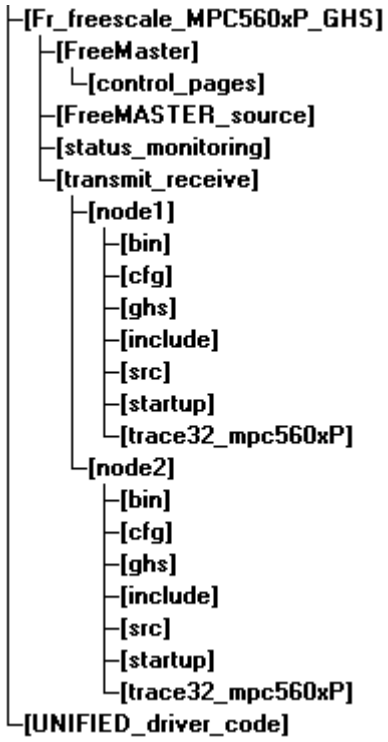

Figure 4-24. Application Example Folder Tree

The driver package contains the following files and folders:

- *UNIFIED_driver_code* folder - contains the general source code and header files. The FlexRay UNIFIED Driver also uses other three files stored under the *cfg* folder in each project
  - *Fr_UNIFIED.c* - the general source code
  - *Fr_UNIFIED.h* - the general header file with definition of the FlexRay registers, structures and C definition
- *Fr_freescale_mpc560xP_GHS* folder - contains application examples intended for the MPC560xP Microcontroller

  - *transmit_receive* subfolder - contains the Transmit Receive application example

    a. *node1* subfolder - contains the application example for Node 1

       i. *bin* folder - contains created binary files

ii. *cfg* folder - contains FlexRay module configuration data in the *Fr_UNIFIED_cfg.c* file, the declaration of the external variables/structures in the *Fr_UNIFIED_cfg.h* file, definition of the compiler types in the *Fr_UNIFIED_types.h* file, the *freemaster_cfg.h* file with the FreeMASTER configuration and the *intflash.ld* file which contains a linker configuration. All these files may be modified by a user

iii. *src* folder - contains application code in the *transmit_receive_node1.c* file (this can be modified by a user)

iv. *include* folder - this folder contains included header files

v. *startup* folder - a file *startup_vle.s* with the startup code and a file *vector_vle.s* with the interrupt vector table are present in this folder

vi. *ghs* folder - contains the GreenHills MULTI project files used to manage a project tree

vii. *trace32_mpc560xP* folder - if the Lauterbach TRACE32 debugger is used, it is possible to use predefined **.cmm* files

viii. *transmit_receive_node1.gpj* - the GreenHills MULTI Project Builder project file

b. *node2* subfolder - contains the application example for Node 2

i. this folder contains the same folders/files structure as in the *node1* folder. Application code is stored in the *transmit_receive_node2.c* file and the GreenHills MULTI project file is named *transmit_receive_node2.gpj*

– *status_monitoring* subfolder - contains the Status Monitoring application example.
This folder contains the same folders/files structure as in the *transmit_receive* folder. Application codes are stored in the *status_monitoring_node1.c* and *status_monitoring_node2.c* files and the GreenHills MULTI project files are named *status_monitoring_node1.gpj* and *status_monitoring_node2.gpj*

– *FreeMASTER* subfolder - contains the necessary files for the FreeMASTER tool and control pages

a. *transmit_receive_node_1.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 1

b. *transmit_receive_node_2.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 2

c. *status_monitoring_node_1.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 1

d. *status_monitoring_node_2.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 2

e. *control_pages* folder - contains support files which are necessary for the FreeMASTER project files

i. *algorithm_block1.htm* - algorithm block description page for the *transmit_receive_node_1.pmp* FreeMASTER project file

ii. *algorithm_block2.htm* - algorithm block description page for the *transmit_receive_node_2.pmp* FreeMASTER project file

iii. *control_page1.htm* - control page for the *status_monitoring_node_1.pmp* FreeMASTER project file

iv. *control_page2.htm* - control page for the *status_monitoring_node_2.pmp* FreeMASTER project file

– *FreeMASTER_source* subfolder - contains the FreeMASTER source files (header files, etc.).

The presented application example is structured in the GreenHills MULTI Project Builder as shown in Figure 4-25. GreenHills MULTI Project Builder Project Tree

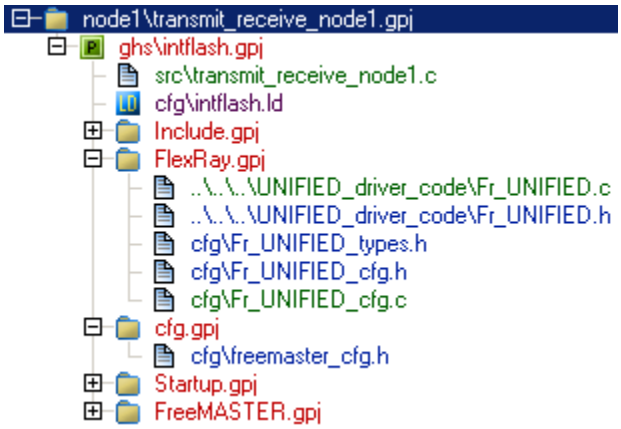

Figure 4-25. GreenHills MULTI Project Builder Project Tree

4.15 CW Development Studio v2.5 for the MPC560xP MCU

The following description is valid only for MPC560xP.

By installing the FlexRay UNIFIED Driver package, a folder tree as shown in Figure 4-26. Application Example Folder Tree shall be created.

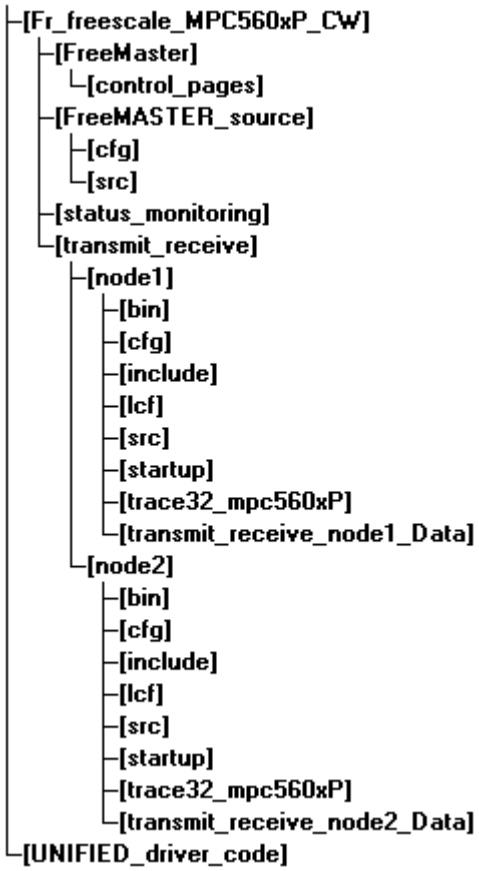

Figure 4-26. Application Example Folder Tree

The driver package contains the following files and folders:

- *UNIFIED_driver_code* folder - contains the general source code and header files. The FlexRay UNIFIED Driver also uses other three files stored under the *cfg* folder in each project
  - *Fr_UNIFIED.c* - the general source code
  - *Fr_UNIFIED.h* - the general header file with definition of the FlexRay registers, structures and C definition
- *Fr_freescale_MPC560xP_CW* folder - contains application examples for the MPC560xP Microcontroller
  - *transmit_receive* subfolder - contains the Transmit Receive application example

    a. *node1* subfolder - contains the application example for Node 1

       i. *startup* folder - contains MCU startup and basic configuration code based on CW stationary created using the MPC55xx New Project Wizard

ii. *cfg* folder - contains FlexRay module configuration data in the *Fr_UNIFIED_cfg.c* file, the declaration of the external variables/structures in the *Fr_UNIFIED_cfg.h* file, definition of the compiler types in the *Fr_UNIFIED_types.h* file and the FreeMASTER configuration in the *freemaster_cfg.h* file.
All these files may be modified by a user

iii. *src* folder - contains application code in the *transmit_receive_node1.c* file (this can be modified by a user)

iv. *trace32_mpc560xP* folder - if the Lauterbach TRACE32 debugger is used, it is possible to use predefined **.cmm* files

v. *transmit_receive_node1.mcp* - the CW project file

b. *node2* subfolder - contains the application example for Node 2

i. this folder contains the same folders/files structure as in the *node1* folder. Application code is stored in the *transmit_receive_node2.c* file and the CW project file is named *transmit_receive_node2.mcp*

– *status_monitoring* subfolder - contains the Status Monitoring application example.
This folder contains the same folders/files structure as in the *transmit_receive* folder. Application codes are stored in the *status_monitoring_node1.c* and *status_monitoring_node2.c* files, and the CW project files are named *status_monitoring_node1.mcp* and *status_monitoring_node2.mcp*

– *FreeMASTER* subfolder - contains the necessary files for the FreeMASTER tool and control pages

a. *transmit_receive_node_1.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 1

b. *transmit_receive_node_2.pmp* - the FreeMASTER project file for the Transmit Receive application example - Node 2

c. *status_monitoring_node_1.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 1

d. *status_monitoring_node_2.pmp* - the FreeMASTER project file for the Status Monitoring application example - Node 2

e. *control_pages* folder - contains support files which are necessary for the FreeMASTER project files

i. *algorithm_block1.htm* - algorithm block description page for the *transmit_receive_node_1.pmp* FreeMASTER project file

ii. *algorithm_block2.htm* - algorithm block description page for the *transmit_receive_node_2.pmp* FreeMASTER project file

iii. *control_page1.htm* - control page for the *status_monitoring_node_1.pmp* FreeMASTER project file

iv. *control_page2.htm* - control page for the *status_monitoring_node_2.pmp* FreeMASTER project file.

– *FreeMASTER_source* subfolder - contains the necessary source files for the FreeMASTER tool

The presented application example is structured in the CodeWarrior Development Studio as shown in Figure 4-27. CodeWarrior Project Tree.

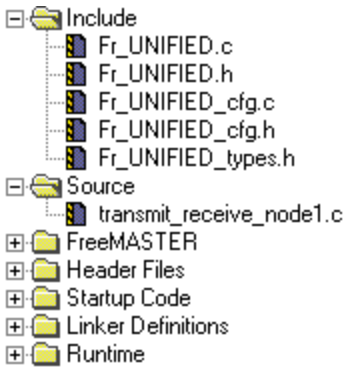

Figure 4-27. CodeWarrior Project Tree

The application examples were tested with the *Internal_FLASH* project target.

Chapter 5 Possible Errors and Workarounds

As a first help in case of any trouble, please Reset the boards by pushing the ‘Reset’ buttons on the boards.

Warning!

In case the FlexRay™ UNIFIED Driver installation package file fails (e.g. the following window is shown), please press “**Cancel**” button and run the installation file from a shorter installation path, such as “c:\” or “d:\”. The problem can be caused by a long file name and/or a long directory path.

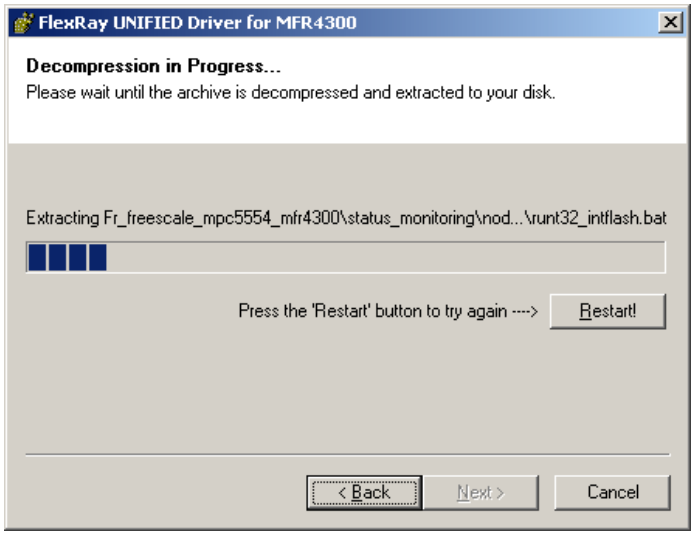

Figure 5-1. FlexRay UNIFIED Driver Installation Error Window

How to Reach Us:

USA/Europe/Locations not listed:
Freescale Semiconductor Literature Distribution
P.O. Box 5405, Denver, Colorado 80217
1-800-521-6274 or 480-768-2130

Japan:
Freescale Semiconductor Japan Ltd.
SPS, Technical Information Center
3-20-1, Minami-Azabu
Minato-ku
Tokyo 106-8573, Japan
81-3-3440-3569

Asia/Pacific:
Freescale Semiconductor H.K. Ltd.
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
852-26668334

Learn More:
For more information about Freescale Semiconductor products, please visit
http://www.freescale.com

UG
Rev. 1.3.1, 01/2010

freescale™
semiconductor