### Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/07/2007</td>
<td>XD, initial version</td>
</tr>
<tr>
<td></td>
<td>V1.0</td>
</tr>
</tbody>
</table>

This document contains 50 pages.
Warranty and Disclaimer

To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH restricts its warranties and its liability for all products delivered free of charge (e.g. software include or header files, application examples, target boards, evaluation boards, engineering samples of IC’s etc.), its performance and any consequential damages, on the use of the Product in accordance with (i) the terms of the License Agreement and the Sale and Purchase Agreement under which agreements the Product has been delivered, (ii) the technical descriptions and (iii) all accompanying written materials. In addition, to the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH disclaims all warranties and liabilities for the performance of the Product and any consequential damages in cases of unauthorised decompiling and/or reverse engineering and/or disassembling. Note, all these products are intended and must only be used in an evaluation laboratory environment.

1. Fujitsu Microelectronics Europe GmbH warrants that the Product will perform substantially in accordance with the accompanying written materials for a period of 90 days form the date of receipt by the customer. Concerning the hardware components of the Product, Fujitsu Microelectronics Europe GmbH warrants that the Product will be free from defects in material and workmanship under use and service as specified in the accompanying written materials for a duration of 1 year from the date of receipt by the customer.

2. Should a Product turn out to be defect, Fujitsu Microelectronics Europe GmbH’s entire liability and the customer’s exclusive remedy shall be, at Fujitsu Microelectronics Europe GmbH’s sole discretion, either return of the purchase price and the license fee, or replacement of the Product or parts thereof, if the Product is returned to Fujitsu Microelectronics Europe GmbH in original packing and without further defects resulting from the customer’s use or the transport. However, this warranty is excluded if the defect has resulted from an accident not attributable to Fujitsu Microelectronics Europe GmbH, or abuse or misapplication attributable to the customer or any other third party not relating to Fujitsu Microelectronics Europe GmbH.

3. To the maximum extent permitted by applicable law Fujitsu Microelectronics Europe GmbH disclaims all other warranties, whether expressed or implied, in particular, but not limited to, warranties of merchantability and fitness for a particular purpose for which the Product is not designated.

4. To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH’s and its suppliers’ liability is restricted to intention and gross negligence.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES

To the maximum extent permitted by applicable law, in no event shall Fujitsu Microelectronics Europe GmbH and its suppliers be liable for any damages whatsoever (including but without limitation, consequential and/or indirect damages for personal injury, assets of substantial value, loss of profits, interruption of business operation, loss of information, or any other monetary or pecuniary loss) arising from the use of the Product.

Should one of the above stipulations be or become invalid and/or unenforceable, the remaining stipulations shall stay in full effect.
Contents

REVISION HISTORY ............................................................................................................ 2
WARRANTY AND DISCLAIMER ......................................................................................... 3
CONTENTS .......................................................................................................................... 4
0 INTRODUCTION .............................................................................................................. 6
1 SOFTWARE EXAMPLES ................................................................................................ 7
  1.1 Template: 96340_template_96348rs ....................................................................... 7
  1.2 Example1: ADA_96340_88121_FLEXRAY_static1 ................................................. 8
      1.2.1 General description......................................................................................... 8
      1.2.1.1 Application flow.................................................................................. 9
      1.2.1.2 Signal flow ....................................................................................... 10
      1.2.1.3 FlexRay Bus Settings................................................................. 10
      1.2.2 Detailed Description.................................................................................... 11
      1.2.2.1 Project folders.............................................................................. 11
      1.2.2.2 The application using Fujitsu FlexRay Driver ................................ 11
      1.2.2.3 The application using DECOMSYS::COMMSTACK .................. 16
      1.2.2.4 Frame payload............................................................................. 18
  1.3 Example2: ADA_96340_88121_FLEXRAY_dynamic1 .......................................... 19
      1.3.1 General description.................................................................................... 19
      1.3.1.1 Application flow............................................................................... 20
      1.3.1.2 Signal flow ...................................................................................... 21
      1.3.1.3 FlexRay Bus Settings..................................................................... 21
      1.3.2 Detailed Description.................................................................................... 22
      1.3.2.1 Project folders.............................................................................. 22
      1.3.2.2 The application using Fujitsu FlexRay Driver ................................ 22
      1.3.2.3 The application using DECOMSYS::COMMSTACK .................. 27
  1.4 Example3: ADA_96340_88121_FLEXRAY_dynamic1_int..................................... 29
      1.4.1 General description.................................................................................... 29
      1.4.1.1 Application flow............................................................................... 30
      1.4.2 ERAY interrupt......................................................................................... 31
      1.4.3 Detailed Description.................................................................................... 32
      1.4.3.1 Project folders............................................................................... 32
      1.4.3.2 Source files of the application ...................................................... 32
  1.5 Example4: ADA_96340_88121_FLEXRAY_static1_DMA ..................................... 34
1.5.1 General description.................................................................................. 34
  1.5.1.1 Application flow........................................................................ 35
1.5.2 DMA ........................................................................................................ 36
1.5.3 Detailed Description................................................................................. 37
  1.5.3.1 Project folders.......................................................................... 37
  1.5.3.2 Source files of the application .................................................. 38
2 FLEXRAY SOFTWARE DRIVER................................................................................... 41
  2.1 Fujitsu FlexRay Driver (FFRD) ............................................................... 41
    2.1.1 Installation ............................................................................................... 41
    2.1.2 Structure.................................................................................................. 41
    2.1.3 Fujitsu FlexRay driver setup .................................................................... 42
    2.1.4 License Agreement.................................................................................. 42
  2.2 DECOMSYS::COMMSTACK V1.8.2 library for 16FX series ......................... 43
    2.2.1 Installation ............................................................................................... 43
    2.2.2 Structure.................................................................................................. 44
    2.2.3 License Agreement.................................................................................. 44
3 CONFIGURATION TOOLS............................................................................................ 45
  3.1 FlexConfig® .......................................................................................................... 45
    3.1.1 Installation ............................................................................................... 45
    3.1.2 Project examples..................................................................................... 46
    3.1.3 License Agreement.................................................................................. 46
  3.2 DECOMSYS::DESIGNERPRO.............................................................................. 47
    3.2.1 Installation ............................................................................................... 47
    3.2.2 Project examples..................................................................................... 47
4 APPENDIX..................................................................................................................... 48
  4.1 Information on the WWW ...................................................................................... 48
  4.2 Figures .................................................................................................................. 49
  4.3 Tables ................................................................................................................... 49
  4.4 Abbreviations ...................................................................................................... 50
0 Introduction

This software guide briefly describes the special FlexRay software, delivered with the adapter board ADA-96340-88121-FLEXRAY.

For information about features of 16FX MB96340 series please see the hardware manual or application notes, which are available in the adapter board CD or Internet.
1 Software examples

This chapter describes the FlexRay software examples included in the CD of adapter board ADA-96340-88121-FLEXRAY.

1.1 Template: 96340_template_96348rs

This template example should be used to start a new project. All required files are included. They are:

1. IO-map header file: mb96348rs.asm
2. C-definitions for IO-symbols: mb96348rs.h
3. Start-up file for memory and basic controller initialisation: start.asm
4. Interrupt level setting and interrupt vector definition: vectors.c

Tool settings of SOFTUNE workbench (e.g. Assembler, C/C++-Compiler, and Linker) are prepared for the DEBUG mode, which is used for emulator debugging and final application in Flash chip.

For details about the template example and the emulator debugger please refer to the application notes:

1. Emulator start guide: mcu-an-300217-e-vXX-16fx_mb2198_01_getting_started.pdf
2. Emulator install guide: mcu-an-300209-e-vXX-16fx_mb2198_01_install.pdf
3. SOFTUNE workbench: mcu-an-300233-e-vXX-16fx_swb_getting_started.doc
4. FLASH programming: mcu-an-300224-e-vXX-16fx_mcu_flash_prog_boot_rom.pdf
1.2 Example1: ADA_96340_88121_FLEXRAY_static1

This example shows how to set up a FlexRay communication between two bus-nodes using static slot only.

1.2.1 General description

The application for both projects (Node1 and Node2) is identical.

Pressing IN0 (input capture unit 0) or INT1 (external interrupt 1) button of the evaluation board FLASH-CAN-100P-340 creates interrupt to MCU. In the IN0 interrupt service routine a variable (isr_cnt) is increased by one whereas in the INT1 interrupt service routine the same variable is increased by three. Another variable (nCounter) is counted every time when the FlexRay task (ttTask) is executed. Both variables are transmitted via the FlexRay channel A and B (redundant transfer).

The Reload Timer 2 is used to represent FlexRay cycle length. Every 3 ms it causes an interrupt. Within the ISR the FlexRay task (ttTASK) is called to write data to input buffer for data transmission and check output buffer for data reception.

Since N1 resistor network is removed from the evaluation board to make the MCU pins available for external bus interface, LEDs of the evaluation board are not applicable to present the transmitted and received data. The terminal program SKwizard is therefore used to display data and CC status in monitor. Accordingly the evaluation board is connected to PC via UARTA(X3) and UART 0 is initialized in software.

The access to the FlexRay CC MB88121 is done via the FlexRay driver (FFRD V1.3) or DECOMSYS::COMMSTACK V1.8.2 libray.
1.2.1.1 Application flow

The application begins with initialisation of MCU and CC. Then reload timer 2 is started to generate the 3 ms FlexRay communication cycle via periodic interrupt. Within the ISR of reload timer 2 the FlexRay task is carried out to transmit and receive data through FlexRay driver. Meanwhile CC status is output to monitor.

Value accumulated by pressing IN0 button and INT1 button is sent only in static slot.

Figure 1-1: Application flow of the example static1
1.2.1.2 Signal flow

![Figure 1-2: Signal flow of the example static1](image)

1.2.1.3 FlexRay Bus Settings

<table>
<thead>
<tr>
<th></th>
<th>Node1</th>
<th>Node2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle length</td>
<td>3 ms</td>
<td>3 ms</td>
</tr>
<tr>
<td>Channels</td>
<td>CH. A &amp; B (redundant)</td>
<td>CH. A &amp; B (redundant)</td>
</tr>
<tr>
<td>Frame ID 3</td>
<td>TX</td>
<td>RX</td>
</tr>
<tr>
<td>Frame ID 6</td>
<td>RX</td>
<td>TX</td>
</tr>
</tbody>
</table>

Table 1-1: FlexRay settings of the example static1
1.2.2 Detailed Description

1.2.2.1 Project folders

The folder structure of this example is as following:

In the main folder (ADA_96340_88121_FLEXRAY_static1_v2) locates the workspace file 'ADA_96340_88121_FLEXRAY_static1_v2.wsp', the introduction file 'readme.txt' and the following sub-folder:

- Node1_ffrd
  - includes folder structure of Node1 project using Fujitsu FlexRay Driver
- Node1_dcs
  - includes folder structure of Node1 project using DECOMSYS::COMMSTACK
- Node2_ffrd
  - includes folder structure of Node2 project using Fujitsu FlexRay Driver
- Node2_dcs
  - includes folder structure of Node2 project using DECOMSYS::COMMSTACK
- Generated_files\src_FlexConfig
  - includes '.chi' files generated by tools for CC initialization.
- Generated_files\dcs
  - includes application-specific source files for CC initialization.
- src_shared
  - includes header file 'global.h' shared by both projects
- ffrd
  - includes Fujitsu FlexRay driver, needs to be installed separately.
- dcsCstFr
  - includes DECOMSYS::COMMSTACK FlexRay driver library, needs to be installed separately.

1.2.2.2 The application using Fujitsu FlexRay Driver

The C source files in folder Node1_ffrd\Src (Node2_ffrd\Src) are as following:

1.2.2.2.1 Start.asm

The file initialises the MCU and calls the main function. Following settings are made in the Start.asm file:
1. section 4.8 Clock Selection

```
#set      CRYSTAL         FREQ_4MHZ
#set      CLOCK_SPEED     CPU_48MHZ_CLKP2_16MHZ
```

External crystal frequency: 4MHz
Bus clock (CLKB): 48MHz
Peripheral clock 1 (CPUP1): 48 MHz
Peripheral clock 2 (CPUP2): 16 MHz

2. section 4.10 External Bus Interface

```
#set      BUSMODE INTROM_EXTBUS
#set      ADDRESSMODE MULTIPLEXED
#set      CHIP_SELECT2 ON       ; enable chip select area 2
#set      EXT_READY ON          ; select external Ready function
#set      EXT_CLOCK_ENABLE ON    ; select external bus clock output
#set      EXT_CLOCK_INVERT ON    ; select clock inversion
#set      EXT_CLOCK_DIVISION EXT_CLOCK_DIV2
#set      ADDR_PINS_15_8 B'000000111; select used address lines A10…A0
#set      ADDR_PINS_7_0  B'11111111
#set      LOW_WRITE_STROBE ON    ; select write strobe signal WRLX/WRX
#set      READ_STROBE ON         ; select read strobe signal RDX
#set      ADDRESS_STROBE ON      ; select address strobe signal ALE/ASX
#set      CS2_CONFIG B'0000100000111010  ; little endian, 16bit bus width
#set      CS2_START 0x10  ; memory bank 10 as chip select area 2
```

At the end of the start-up file main() function is called.

```
;====================================================================
; 6.13 Call C-language main function
;====================================================================
#if MEMMODEL == SMALL || MEMMODEL == COMPACT
CALL _main     ; Start main function
#else
CALLP _main    ; MEDIUM, LARGE, AUTOMODEL
 ; ignore remaining word on stack,
 ; if main was completed by RET
#endif
```
1.2.2.2 Main.c

Main()

The main() function is called from the file start.asm. First the I/O ports are enabled and UART0, ICU0, INT1 and reload timer 2 are initialized by function init(). Then the function InitIrqLevels() sets interrupt control register, which defines the interrupt level of each peripheral. After that the FlexRay communication is initialised via the function ttStartupHook(). If the initialisation of the FlexRay bus is successful, the reload timer2 is started to generate the FlexRay cycle tick.

void init(void)

Init_ICU01() and Init_extint_1() (defined in file extint.c) initialise input capture unit 0 and external interrupt 1 for user button IN0 and INT1. Both create interrupt by detecting signal falling edge. Reload Timer 2 realizes 3 ms FlexRay cycle tick. The reload value is 2250. Cycle length is thus 2250 x (64/48MHz) = 3 ms. Function Init_rldtmr_2() is defined in file reloadtimer.c. UART 0 interface for terminal program works at 19200 Baud. Function InitUart0() is defined in file uart.c.

__interrupt void IsrReloadTimer2(void)

The interrupt service routine (ISR) of reload timer 2 is used as FlexRay cluster tick. It causes an interrupt every 3 ms. Within the ISR an offset correction between host MCU and FlexRay CC is done. The global time of the FlexRay bus (MacroTick) is read out from register MTCCV and compared against the host time. If necessary the host time (reload value) is corrected. After the time correction the FlexRay task is called. (task_Nodex() defined in TTask.c)

1.2.2.3 TTask.c

ttTASK(NodeX)

The ttask() function is responsible for the access of the ERAY input and output buffer. Every 100 calls (variable nTaskInvocations) the function checks if the FlexRay CC is still synchronous via the FlexRay driver function ffrd_api_get_poc_status() (state normal_active). In case synchronisation is lost, the FlexRay CC restarts.

if(!ffrd_api_pocs_is_halt())
    ffrd_api_poc_command(FFRD_POCC_FREEZE); /* if not sync, enter HALT state */
    ffrd_api_poc_command(FFRD_POCC_CONFIG); /* enter DEFAULT.getConfig state */
    ffrd_api_poc_command(FFRD_POCC_CONFIG); /* enter CONFIG state */
    ffrd_api_poc_command(FFRD_POCC_READY); /* enter READY state */
    ffrd_api_poc_command(FFRD_POCC_RUN); /* enter RUN state */
    ffrd_api_poc_command(FFRD_POCC_RESET_STATUS_INDICATORS);
    ffrd_api_poc_command(FFRD_POCC_ALLOW_COLDSTART); /* do a coldstart or integration start */
In case of valid static transmission request (variable tx_flag, set in ISR of ICU0 and INT1),
the transmission data is copied into FlexRay frame payload (structure buffer). Via the driver
function ffrd_api_tx_handler_buffer() the static data is copied into the corresponding ERAY
message buffer 0 in message RAM.

Via the function ffrd_api_rx_handler_buffer() output buffer is checked. If new valid data are
received, they are stored in the dedicated buffer (structure sRx1, sRx2) and output to
monitor via terminal program SKwizard.

```c
void ttStartupHook(void)
```

The function is called from main() function during initialisation phase. It initialises the
FlexRay driver and the communication controller by calling the function ffrd_api_init_chi(),
which configures the FlexRay CC via .chi file (function ffrd_api_include_chi) and sets
the CC to CONFIGURE state.

```c
void ffrd_api_include_chi ( void ){
    #include "static_demol_CHI_Nodex.chi"  /*.chi file for this application*/
}
```

In case that the initialisation is not O.K. the initialisation phase is broken by calling function
ttShutdownHook(), which prints the shutdown message to monitor. Otherwise function
ffrd_api_go_active() (defined in file ffrd_api_control_service.c) brings the CC to normal
active. After this function it is possible to use the FlexRay system for data transmission and
reception.

### 1.2.2.4 Print_status.c

The function printFlexRayStatus() is called in ttTask(Nodex). The purpose is to output the
five last status of the CC via UART0 serial interface. The function also checks if the FlexRay
CC is still synchronous and outputs the corresponding messages.

### 1.2.2.5 Vectors.c

Via the #pragma intvect instruction the interrupt vector table is built up. The entries for
external interrupt 1, reload timer 2 and ICU0 are set.

```c
void InitIrqLevels(void)
```

The function sets the value of the interrupt control register ICR, which defines the interrupt
level (priority) of the peripheral resource. 0 is strongest. 7 is no interrupt. Levels for INT1,
reload timer 2 and ICU0 are set.
1.2.2.6 ffrd_api_init_chi_def.h

The file is used to configure the clock settings in MB88121 series via register CCNT. Depending on the used external crystal the PLL multiplication ratio needs to be set accordingly. See also datasheet of MB88121 series. In this example the external crystal frequency is 4 MHz.

```
#if FFRD_FRCC <= FFRD_LAST_STANDALONE
    #define FFRD_DEF_CCNT_PON     1L /**<== PLL oscillator enable */
    #define FFRD_DEF_CCNT_SSEL    1L /**<== System Clock Selection */
    #define FFRD_DEF_CCNT_PMUL    3L /**<== PLL Multiplier Selection */
    #define FFRD_DEF_CCNT_STOP    0L /**<== Clock Stop */
#else
    #define FFRD_DEF_CCNT_RCLK    0L /**<== RAM Clock Selection */
    #define FFRD_DEF_CCNT_SDIV    0L /**<== Division for system clock */
#endif

These parameters are used in function ffrd_ccal_init_chi().
```

1.2.2.7 ffrd_api_global_def.h

The file belongs to the FlexRay driver and should be set according to the application. Following settings have to be checked and set as listed for this example:

```
#define FFRD_MCU MB96348RS
selects host MCU type
#define FFRD_FRCC MB88121B
selects communication controller type
#define FFRD_FRCC_OFFSET 0x00100000
address offset of CC registers
#define FFRD_MCU_FRCC_CONNECT PARALLEL_BUS
PARALLEL_BUS between CC and MCU
#define FFRD_INIT_MODE CHI
selects CHI file as CC initialisation mode
#define FFRD_TIME_SERVICE YES
Selects which ffrd services shall be available to reduce code size. Time service is used in function ffrd_ccal_get_mtick()
#define FFRD_STARTUP_RELOAD_TIMER 3
selects reload timer 3 for PLL stabilisation during CC start-up, used in function ffrd_ccal_init_chi()"
The frame payload consists of 8bit long entry only for 16FX MCU using little-endian. This is not required by 32bit FR MCU, which uses big-endian for byte ordering. A mixed frame payload with different entry length is not supported currently.

1.2.2.3 The application using DECOMSYS::COMMSTACK

The behaviour of the application by both software drivers is identical. Only the important changes are listed here.

1.2.2.3.1 TTask.c (Nodex_dcs\Src)

ttTASK(NodeX)

The ttask() function controls the access to the ERAY input and output buffer.

Every 100 calls the function checks if the FlexRay CC is still synchronous via the driver function TDDLL_GetCtrlState(0). In case that synchronisation is lost, the CC is switched off from the FlexRay bus (TDDLL_DoCtrlTransition(0, TDDLL_T_ABORT)) and performs a restart. (TDDLL_DoCtrlTransition(0, TDDLL_T_STARTUP)).

Every 300 calls the function checks if the FlexRay CC is still online. If it is not, a restart / re-synchronisation (cold start or integration start) is processed.

In case of valid static transmission request (indicated by variable tx_flag), data is copied into the frame payload structure ‘buffer’. Via the driver function TDDLL_TxFrameByID() the static data is copied into the corresponding ERAY message buffer.

Via the function TDDLL_RxFrameByID() it is checked if new valid data is received and stored in the dedicated input buffer. In case of new data (return TDDLL_E_OK by TDDLL_TxFrameByID()) its value is compared with previous one and output to monitor if they are different.

void ttStartupHook(void)

The function ttStartupHook() is called from main() function during initialisation phase. It initialises the FlexRay driver and the FlexRay communication controller.

Before initialising the FlexRay driver, the PLL of CC is activated. Reload timer 3 is used to ensure the PLL oscillation stabilisation time.

The function TDDLL_Init() initialises the FlexRay driver itself, so that the driver is able to access the FlexRay CC MB88121. To ensure a correct behaviour the CC is rest via function TDDLL_DoCtrlTransition(0, TDDLL_T_RESET). Afterwards it is set to CONFIG state (TDDLL_S_CONFIG) by function TDDLL_DoCtrlTransition(0, TDDLL_T_ENTER_CONFIG).

The initialisation of the FlexRay CC is done by calling the function TDDLL_CtrlInit(0), which initialises the E-Ray registers and buffers. In case if the initialisation fails (not TDDLL_E_OK), the initialisation phase is aborted and the FlexRay part is shutdown via the function ttShutdownHook().

Via the function TDDLL_DoCtrlTransition(0, TDDLL_T_LEAVE_CONFIG) the CONFIG state is left to READY state. Via function TDDLL_DoCtrlTransition(0, TDDLL_T_STARTUP) the FlexRay CC initiates a cold start and tries to establish a FlexRay communication.

After these functions the FlexRay system is running. Transmission and reception are possible.
1.2.2.3.2  dcsCstFr_Nodex_Cfg.c (Generated_files\dcs)

This file is an output file of DECOMSYS::DESIGNERPro. It contains the FlexRay schedule settings and E-Ray registers initialisation.

1.2.2.3.3  dcsCstFr_Nodex_Memory_Cfg.c (Generated_files\dcs)

This file is an output file of DECOMSYS::DESIGNERPro. The memory usage functions / buffers are set here.

1.2.2.3.4  dcsCstFr_CtrlHW_Cfg.c (\dcsCstFr\cfg\arch-fujitsu-fme)

This file is used to choose the CC type (ERAY10\ MFR4200\ MFR4300\ PHIP1) and set the register offset correspondingly for the DECOMSYS::COMMSTACK.

/* choose FlexRay communication controller */
TDDLL_CtrlMapping[TDDLL_CTRL_NUMBER_MAX]=
{
   { TDDLL_CTRL_TYPE_ERAY10,
       0
   }
};

/* E-Ray Register Offset for ADA-96340-88121-FLEXRAY */
TDDLL_ERAY10_Ctrl_List[TDDLL_MAX_CTRL_ERAY10]=
{
   { (FCAL_ERAY10_CtrlHandleType) 0x00100000,
       NULL
   }
};
1.2.2.4 Frame payload

The structure of frame payload (buffer, sRx1, sRx2) for 16FX MCU should be uniform. The current Fujitsu driver supports three types: 8bit, 16bit and 32bit. By DECOMSYS driver only 8bit is possible. All examples of this software guide use the same payload structure definition ‘data_content’ with 20 bytes altogether. Array ‘empty[x]’ in the payload contains constant data for testing byte ordering by receiver side.

<table>
<thead>
<tr>
<th>Structure of frame payload</th>
<th>8 bit entry</th>
<th>16 bit entry</th>
<th>32 bit entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_content.CNT</td>
<td>nCounter</td>
<td>nCounter</td>
<td>nCounter</td>
</tr>
<tr>
<td>data_content.Tx</td>
<td>0xAA/ 0x55</td>
<td>0xAA/ 0x55</td>
<td>0xAA/ 0x55</td>
</tr>
<tr>
<td>data_content.Port</td>
<td>tx_data</td>
<td>tx_data</td>
<td>tx_data</td>
</tr>
<tr>
<td>data_content.empty[0]</td>
<td>0x78</td>
<td>0xDEF0</td>
<td>0x9BAC1425</td>
</tr>
<tr>
<td>data_content.empty[1]</td>
<td>0x9A</td>
<td>0x1357</td>
<td>0x36475869</td>
</tr>
<tr>
<td>data_content.empty[2]</td>
<td>0xBC</td>
<td>0x2468</td>
<td>\</td>
</tr>
<tr>
<td>data_content.empty[3]</td>
<td>0xDE</td>
<td>0x9BAC</td>
<td>\</td>
</tr>
<tr>
<td>data_content.empty[4]</td>
<td>0xF0</td>
<td>0x1425</td>
<td>\</td>
</tr>
<tr>
<td>data_content.empty[5]</td>
<td>0x13</td>
<td>0x3647</td>
<td>\</td>
</tr>
<tr>
<td>data_content.empty[6]</td>
<td>0x57</td>
<td>0x5869</td>
<td>\</td>
</tr>
<tr>
<td>data_content.empty[7]</td>
<td>0x24</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>data_content.empty[8]</td>
<td>0x68</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>data_content.empty[9]</td>
<td>0x9B</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>data_content.empty[10]</td>
<td>0xAC</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>data_content.empty[11]</td>
<td>0x14</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>data_content.empty[12]</td>
<td>0x25</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>data_content.empty[13]</td>
<td>0x36</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>data_content.empty[14]</td>
<td>0x47</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>data_content.empty[15]</td>
<td>0x58</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>data_content.empty[16]</td>
<td>0x69</td>
<td>\</td>
<td>\</td>
</tr>
</tbody>
</table>

Table 1-2 Frame payload
1.3 Example2: ADA_96340_88121_FLEXRAY_dynamic1

This example shows how to set up a FlexRay communication between two bus-nodes using static and dynamic slot.

1.3.1 General description

In principle this example is an extension of the last example static1. In addition to the static slots, dynamic slots are also used.

Pressing IN0 (input capture unit 0) or INT1 (external interrupt 1) button of the evaluation board FLASH-CAN-100P-340 creates an interrupt to MCU. In the IN0 ISR a variable (isr_cnt) is increased by one. In the INT1 ISR an AD conversion is started. When it is finished, it causes also an interrupt to save the ADC result into frame payload (structure buffer2). Variable nCounter is counted every time when the FlexRay task (ttTask) is executed.

Variable nCounter and isr_cnt are transmitted in static frame payload (structure buffer). In dynamic frame payload (structure buffer2) only ADC result is transmitted.

The Reload Timer 2 is used to represent FlexRay cycle length. Every 3 ms it causes an interrupt. Within the ISR the FlexRay task (ttTASK) is called to write data to input buffer for data transmission and check output buffer for data reception.

Since N1 resistor network is removed from the evaluation board to make the MCU pins available for external bus interface, LEDs of the evaluation board are not applicable to present the transmitted and received data. The terminal program SKwizard is therefore used to display data and CC status in monitor. Accordingly the evaluation board is connected to PC via UARTA(X3) and UART 0 is initialized in software.

The access to the FlexRay CC MB88121 is done via the FlexRay driver (FFRD V1.3) or DECOMSYS::COMMSTACK V1.8.
1.3.1.1 Application flow

The application begins with initialisation of MCU and CC. Then reload timer 2 is started to generate the 3 ms FlexRay communication cycle via periodic interrupt. Within the ISR of reload timer 2 the FlexRay task is carried out to transmit and receive data through FlexRay driver. Meanwhile CC status is output to monitor.

Value accumulated by pressing input capture unit 0 (IN0 button) is sent in static slot, values resulted from ADC (triggered by INT1 button) is sent in dynamic slot.

![Diagram showing the application flow of the example dynamic 1](image)

**Figure 1-3: Application flow of the example dynamic 1**
1.3.1.2 Signal flow

Figure 1-4: Signal flow of the example dynamic1

1.3.1.3 FlexRay Bus Settings

<table>
<thead>
<tr>
<th></th>
<th>Node1</th>
<th>Node2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cycle length</strong></td>
<td>3 ms</td>
<td>3 ms</td>
</tr>
<tr>
<td><strong>Static Channels</strong></td>
<td>CH. A &amp; B (redundant)</td>
<td>CH. A &amp; B (redundant)</td>
</tr>
<tr>
<td><strong>Static slot: ID 3</strong></td>
<td>TX</td>
<td>RX</td>
</tr>
<tr>
<td><strong>Static slot: ID 6</strong></td>
<td>RX</td>
<td>TX</td>
</tr>
<tr>
<td><strong>Dynamic slot. ID 41</strong></td>
<td>TX</td>
<td>RX</td>
</tr>
<tr>
<td><strong>Channel A</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dynamic Slot ID 45</strong></td>
<td>RX</td>
<td>TX</td>
</tr>
<tr>
<td><strong>Channel B</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1-3: Dynamic1 example FlexRay settings
1.3.2 Detailed Description

1.3.2.1 Project folders

The folder structure of the example is as following:

In the main folder (ADA_96340_88121_FLEXRAY_dynamic1_v2) locates the workspace file ‘ADA_96340_88121_FLEXRAY_dynamic1_v2.wsp’, the introduction file ‘readme.txt’ and the following sub-folder:

- **Node1_ffrd**
  - includes folder structure of Node1 project using Fujitsu FlexRay Driver
- **Node1_dcs**
  - includes folder structure of Node1 project using DECOMSYS::COMMSTACK
- **Node2_ffrd**
  - includes folder structure of Node2 project using Fujitsu FlexRay Driver
- **Node2_dcs**
  - includes folder structure of Node2 project using DECOMSYS::COMMSTACK
- **Generated_files\src_FlexConfig**
  - includes ‘.chi’ files generated by tools for CC initialization.
- **Generated_files\dcs**
  - includes application-specific source files for CC initialization.
- **src_shared**
  - includes header file ‘global.h’ shared by both projects
- **ffrd**
  - includes Fujitsu FlexRay driver, needs to be installed separately.
- **dcsCstFr**
  - includes DECOMSYS::COMMSTACK FlexRay driver library, needs to be installed separately.

1.3.2.2 The application using Fujitsu FlexRay Driver

The C source files in folder Node1_ffrd\Src (Node2_ffrd\Src) are as following:

1.3.2.2.1 Start.asm

The file initialises the MCU and calls the main function. Following settings are made in the Start.asm file:
1. Section 4.8 Clock Selection

```
#set      CRYSTAL         FREQ_4MHZ
#set      CLOCK_SPEED     CPU_48MHZ_CLKP2_16MHZ
```

External crystal frequency: 4MHz
Bus clock (CLKB): 48MHz
Peripheral clock 1 (CPUP1): 48 MHz
Peripheral clock 2 (CPUP2): 16 MHz

2. Section 4.10 External Bus Interface

```
#set      BUSMODE INTROM_EXTBUS
#set      ADDRESSMODE MULTIPLEXED
#set      CHIP_SELECT2       ON    ; enable chip select area 2
#set      EXT_READY          ON    ; select external Ready function
#set      EXT_CLOCK_ENABLE   ON    ; select external bus clock output
#set      EXT_CLOCK_INVERT   ON    ; select clock inversion
#set      EXT_CLOCK_DIVISION EXT_CLOCK_DIV2
#set      ADDR_PINS_15_8 B'000000111 ; select used address lines A10…A0
#set      ADDR_PINS_7_0  B'11111111
#set      LOW_WRITE_STROBE   ON    ; select write strobe signal WRLX/WRX
#set      READ_STROBE        ON    ; select read strobe signal RDX
#set      ADDRESS_STROBE     ON    ; select address strobe signal ALE/ASX
#set      CS2_CONFIG B'0000100000111010  ; little endian, 16bit bus width
#set      CS2_START 0x10  ; memory bank 10 as chip select area 2
```

At the end of the start-up file main() function is called.

1.3.2.2.2 Main.c

Main()

The main() function is called from the file start.asm. First MCU resources used in this example are initialized by function init(). Then the function InitIrqLevels() defines the interrupt level of each peripheral. After that the FlexRay communication is initialised via the function ttStartupHook(). If the initialisation is successful, the reload timer2 is started to generate the FlexRay cycle tick.

void init(void)

Init_ICU01() and Init_extint_1() (defined in file icu.c and extint.c) initialise input capture unit 0 and external interrupt 1 for user button IN0 and INT1. Both create interrupt by detecting signal falling edge. Reload Timer 2 realizes 3 ms FlexRay cycle tick. The reload value is 2250. Cycle length is thus 2250 x (64/48MHz) = 3 ms. Function Init_rldtmr_2() is defined in file reloadtimer.c. UART 0 interface for terminal program works at 19200 Baud. Function
InitUart0() is defined in file uart.c. AD conversion begins with channel 0 and ends at channel 2. The result is 8 bit long. Function init_ADC() is defined in file adc.c.

__interrupt void IsrReloadTimer2(void)
The reload timer 2 is used to present the FlexRay cluster tick. It causes an interrupt every 3 ms. Within the ISR host MCU and CC are first synchronized. The global time of the FlexRay bus (Macrotick) is read out from register MTCCV and compared against the host time. If necessary the host time (reload timer value) is corrected. After that the FlexRay task is called. (task_Nodex() defined in TTTask.c)

1.3.2.2.3 TTTask.c

ttTASK(NodeX)

The ttask() function is responsible for the access of the ERAY input and output buffer.

Every 100 calls (variable nTaskInvocations) the function checks if the FlexRay CC is still synchronous via the FlexRay driver function ffrd_api_get_poc_status() (state normal_active). In case synchronisation is lost, the FlexRay CC restarts.

```c
if(!ffrd_api_pocs_is_halt())
    ffrd_api_poc_command(FFRD_POCC_FREEZE); /* if not sync, enter HALT state */
    ffrd_api_poc_command(FFRD_POCC_DEFAULT_CONFIG); /* enter DEFAULT_CONFIG state */
    ffrd_api_poc_command(FFRD_POCC_CONFIG); /* enter CONFIG state */
    ffrd_api_poc_command(FFRD_POCC_READY); /* enter READY state */
    ffrd_api_poc_command(FFRD_POCC_RUN); /* enter RUN state */
    ffrd_api_poc_command(FFRD_POCC_RESET_STATUS_INDICATORS);
    ffrd_api_poc_command(FFRD_POCC_ALLOW_COLDSTART); /* do a coldstart or integration start */
```

In case of valid static transmission request (variable tx_flag, set in ISR of ICU0), the transmission data is copied into FlexRay frame payload (structure buffer). Via the driver function ffrd_api_tx_handler_buffer() the frame payload is copied into the ERAY message buffer 0 in message RAM.

In case of valid dynamic transmission request (variable dynamic_tx, set in ISR of INT1), the frame payload (structure buffer2) is copied into the ERAY message buffer 4 in message RAM.

Via the function ffrd_api_rx_handler_buffer() output buffer is checked. If new valid data are received, they are stored in the dedicated buffer (structure sRx1, sRx2 for static data, sRx3 for dynamic data) and output to monitor via terminal program SKwizard.
void ttStartupHook(void)

The function is called from main() function during initialisation phase. It initialises the FlexRay driver and the communication controller. Function ffrd_api_init_chi() configures the FlexRay CC via .chi file and sets the CC to CONFIGURE state.

`ffrd_api_init_chi()=>ffrd_ccal_init_chi()=>ffrd_api_include_chi()

void ffrd_api_include_chi( void ){
    #include "dynamic_demol_CHIP_Nodex.chi"
}

In case that the initialisation is not O.K, the initialisation phase is broken by calling function ttShutdownHook(), which prints the shutdown message to monitor. Otherwise function ffrd_api_go_active() (defined in file ffrd_api_control_service.c) brings the CC to normal active. After that it is possible to use the FlexRay system for data transmission and reception.

void send_dynamic_tx (uint8_t tx_val)

This function copies the converted analogue value tx_val into dynamic frame payload (structure buffer2). Afterwards the transmission flag (dynamic_tx) is set. This function is called in ISR of ADC.

1.3.2.2.4 Print_status.c

The function printFlexRayStatus() is called in ttTask(Nodex). The purpose is to output the five last status of the CC via UART0 serial interface. The function also checks if the FlexRay CC is still synchronous and outputs the corresponding messages.

1.3.2.2.5 Vectors.c

Via the #pragma intvect instruction the interrupt vector table is built up. The entries of ADC, external Interrupt 1, reload Timer 2 and input capture unit 0 are set.

void InitIrqLevels(void)

The function defines the interrupt level (priority) of the peripheral resource via configuration of the interrupt control register ICR. 0 is strongest. 7 is no interrupt. Levels for ADC, INT1, reload timer 2 and ICU0 are set.

1.3.2.2.6 ffrd_api_global_def.h

The file belongs to the FlexRay driver and should be set according to the application. Following settings have to be checked and set as listed for this example:

`#define FFRD_MCU MB96348RS`

selects host MCU type

`#define FFRD_FRCC MB88121B`

selects communication controller type
#define FFRD_FRCC_OFFSET 0x00100000

address offset of CC registers

#define FFRD_MCU_FRCC_CONNECT PARALLEL_BUS

PARALLEL_BUS between CC and MCU

#define FFRD_INIT_MODE CHI

selects CHI file as CC initialisation mode

#define FFRD_TIME_SERVICE YES

Selects which ffrd services shall be available to reduce code size. Time service is used in function ffrd_ccal_get_mtick()

#define FFRD_STARTUP_RELOAD_TIMER 3

selects reload timer 3 for the PLL stabilisation during CC start-up, used in function ffrd_ccal_init_chi()

#define FFRD_FRAME_PAYLOAD FFRD_8_bit_PAYLOAD_ENTRY

The frame payload consists of 8bit long entry only for 16FX MCU using little-endian. This is not required by 32bit FR MCU, which uses big-endian for byte ordering. A mixed frame payload with different entry length is not supported currently.

1.3.2.2.7 ffrd_api_init_chi_def.h

The file is used to configure the clock settings in MB88121 series via register CCNT. Depending on the used external crystal the PLL multiplication ratio needs to be set accordingly. See also datasheet of MB88121 series. In this example the external crystal frequency is 4 MHz.

#if FFRD_FRCC <= FFRD_LAST_STANDALONE

#define FFRD_DEF_CCNT_PON 1L /**<== PLL oscillator enable */
#define FFRD_DEF_CCNT_SSEL 1L /**<== System Clock Selection */
#define FFRD_DEF_CCNT_PMUL 3L /**<== PLL Multiplier Selection */
#define FFRD_DEF_CCNT_STOP 0L /**<== Clock Stop */
#endif

#if FFRD_FRCC_VERSION >= FFRD_ERAY_VERSION_PRE_BETA2_UPDATE

#define FFRD_DEF_CCNT_RCLK 0L /**<== RAM Clock Selection */
#define FFRD_DEF_CCNT_SDIV 0L /**<== Division for system clock */
#endif

These parameters are used in function ffrd_ccal_init_chichi().
1.3.2.3 The application using DECOMSYS::COMMSTACK

The behaviour of the application by both software drivers is identical. Only the important changes are listed here.

1.3.2.3.1 TTASK(NodeX)

The ttask() function controls the access to the ERAY input and output buffer.

Every 100 calls the function checks if the FlexRay CC is still synchronous via the driver function TDDLL_GetCtrlState(0). In case that synchronisation is lost, the CC is switched off from the FlexRay bus (TDDLL_DoCtrlTransition(0, TDDLL_T_ABORT)) and performs a restart. (TDDLL_DoCtrlTransition(0, TDDLL_T_STARTUP)).

Every 300 function calls the function checks if the FlexRay CC is still online. If it is not, a restart / re-synchronisation (cold start or integration start) is processed.

In case of valid static transmission request (indicated by variable tx_flag), data is copied into the frame payload structure ‘buffer’. Via the driver function TDDLL_TxFrameById() the static data is copied into the corresponding ERAY message buffer.

In case of valid dynamic transmission request (indicated by variable dynamic_tx), dynamic data (structure ‘buffer2’) is copied into the corresponding ERAY message buffer.

Via the function TDDLL_RxFrameById() it is checked if new valid data is received and stored in the dedicated input buffer. In case of new data (return TDDLL_E_OK by TDDLL_TxFrameById()) its value is compared with previous one and output to monitor if they are different.

void ttStartupHook(void)

The function ttStartupHook() is called from main() function during initialisation phase. It initialises the FlexRay driver and the FlexRay communication controller.

Before initialising the FlexRay driver, the PLL of CC is activated. Reload timer 3 is used to ensure the PLL oscillation stabilisation time.

The function TDDLL_Init() initialises the FlexRay driver itself, so that the driver is able to access the FlexRay CC MB88121. To ensure a correct behaviour the CC is rest via function TDDLL_DoCtrlTransition(0, TDDLL_T_RESET). Afterwards it is set to CONFIG state (TDDLL_S_CONFIG) by function TDDLL_DoCtrlTransition(0, TDDLL_T_ENTER_CONFIG).

The initialisation of the FlexRay CC is done by calling the function TDDLL_CtrlInit(0), which initialises the E-Ray registers and buffers. In case if the initialisation fails (not TDDLL_E_OK), the initialisation phase is aborted and the FlexRay part is shutdown via the function ttShutdownHook().

Via the function TDDLL_DoCtrlTransition(0, TDDLL_T_LEAVE_CONFIG) the CONFIG state is left to READY state. Via function TDDLL_DoCtrlTransition(0, TDDLL_T_STARTUP) the FlexRay CC initiates a cold start and tries to establish a FlexRay communication.

After these functions the FlexRay system is running. Transmission and reception are possible.
1.3.2.3.2 dcsCstFr_Nodex_Cfg.c (Generated_files\dcs)

This file is an output file of DECOMSYS::DESIGNERPro. It contains the FlexRay schedule settings and E-Ray registers initialisation.

1.3.2.3.3 dcsCstFr_Nodex_Memory_Cfg.c (Generated_files\dcs)

This file is an output file of DECOMSYS::DESIGNERPro. The memory usage functions / buffers are set here.

1.3.2.3.4 dcsCstFr_CtrlHW_Cfg.c (\dcsCstFr\cfg\arch-fujitsu-fme)

This file is used to choose the CC type (ERAY10\ MFR4200\ MFR4300\ PHIP1) and set the register offset correspondingly for the DECOMSYS::COMMSTACK.

```c
/* choose FlexRay communication controller */
TDDLL_CtrlMapping[TDDLL_CTRL_NUMBER_MAX]=
{
  { TDDLL_CTRL_TYPE_ERAY10, 0 },
};

/* E-Ray Register Offset for ADA-96340-88121-FLEXRAY */
TDDLL_ERAY10_Ctrl_List[TDDLL_MAX_CTRL_ERAY10]=
{
  { (FCAL_ERAY10_CtrlHandleType) 0x00100000, NULL },
};
```
1.4 Example3: ADA_96340_88121_FLEXRAY_dynamic1_int

This example shows how to use the ERAY interrupt

1.4.1 General description

This example is an extension of the last example dynamic1. In addition ERAY interrupt is involved to receive dynamic frame.

Pressing IN0 (input capture unit 0) or INT1 (external interrupt 1) button of the evaluation board FLASH-CAN-100P-340 creates an interrupt to MCU. In the IN0 ISR a variable (isr_cnt) is increased by one. In the INT1 ISR an AD conversion is started. When it is finished, it causes also an interrupt to save the ADC result into frame payload (structure buffer2). Variable nCounter is counted every time when the FlexRay task (ttTask) is executed.

Variable nCounter and isr_cnt are transmitted in static frame payload (structure buffer). In dynamic frame payload (structure buffer2) only ADC result is transmitted.

The Reload Timer 2 is used to represent FlexRay cycle length. Every 3 ms it causes an interrupt. Within the ISR the FlexRay task (ttTASK) is called to write data to input buffer for data transmission and check output buffer for static data reception.

By receiving dynamic data CC requests MCU (via external interrupt 3) to read out data from message buffer.

Since N1 resistor network is removed from the evaluation board to make the MCU pins available for external bus interface, LEDs of the evaluation board are not applicable to present the transmitted and received data. The terminal program SKwizard is therefore used to display data and CC status in monitor. Accordingly the evaluation board is connected to PC via UARTA(X3) and UART 0 is initialized in software.

The access to the FlexRay CC MB88121 is done only via the FlexRay driver (FFRD V1.3).
1.4.1.1 Application flow

The application begins with initialisation of MCU and CC. Then reload timer 2 is started to generate the 3 ms FlexRay communication cycle via periodic interrupt. Within the ISR of reload timer 2 the FlexRay task is carried out through FlexRay driver. Meanwhile CC status is output to monitor.

Value accumulated by pressing input capture unit 0 (IN0 button) is sent in static slot, value resulted from ADC (triggered by INT1 button) is sent in dynamic slot.

The reception of dynamic data is performed in ERAY ISR and thus independent of the reload timer ISR.

---

**Figure 1-5 Application flow of example dynamic_int1**
1.4.2 ERAY interrupt

ERAY provides two types of interrupt: error interrupt and status interrupt. The reception interrupt used in this example belongs to the status interrupt (register SIR bit RXI). It is assigned to interrupt line 1 (CC INT1 pin). Via the interrupt registers (SILS, SIES, ILE) reception interrupt is enabled and interrupt line 1 is activated. Additionally the MBI bit of the register WRHS1 is set to enable the message buffer 3 interrupt. The corresponding register settings (in .chi file) are:

<table>
<thead>
<tr>
<th>Register</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SILS.RXIL=1</td>
<td>Reception interrupt is assigned to interrupt line eray_int1</td>
</tr>
<tr>
<td>SIES.RXIE=1</td>
<td>Enable the reception interrupt</td>
</tr>
<tr>
<td>ILE.EINT1=1</td>
<td>Interrupt line eray_int1 is enabled</td>
</tr>
<tr>
<td>WRHS1.MBI=1</td>
<td>Enable message buffer 3 interrupt (RXI, TXI, MBSI)</td>
</tr>
</tbody>
</table>

In this case reception interrupt RXI

Table 1-4 Register settings for ERAY interrupt

CC is connected to MCU in 16-bit multiplexed mode. Five CC interrupt pins are connected to MCU external interrupt inputs:

<table>
<thead>
<tr>
<th>Interrupt type</th>
<th>CC pin</th>
<th>MCU pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-Ray interrupt line0</td>
<td>INT0</td>
<td>MCU external interrupt 2(P07_2)</td>
</tr>
<tr>
<td>E-Ray interrupt line1</td>
<td>INT1</td>
<td>MCU external interrupt 3(P07_3)</td>
</tr>
<tr>
<td>E-Ray timer0 interrupt</td>
<td>INT2</td>
<td>MCU external interrupt 4(P07_4)</td>
</tr>
<tr>
<td>E-Ray timer1 interrupt</td>
<td>INT3</td>
<td>MCU external interrupt 5(P07_5)</td>
</tr>
<tr>
<td>Low voltage detection</td>
<td>INT4</td>
<td>MCU external interrupt 6(P07_6)</td>
</tr>
</tbody>
</table>

Table 1-5 Interrupt signal map

The CC interrupt request is high active. Therefore the MCU external interrupt 3 is initialised to detect signal rising-edge (register ELVR0). Its ISR is predefined in the FlexRay driver: function ffrd_api_interrupt_line1() (defined in file ffrd_api_interrupt_service.c). By the entry ‘case FFRD_SIR_RECEIVE_INTERRUPT’ function get_dynamic_message() (defined in TTask.c) performs the dynamic data reception.

Since the whole process causes two interrupt request flags (RXI flag and the external interrupt flag), both of them must be cleared in the ISR. This is done automatically by the driver.

<table>
<thead>
<tr>
<th>Interrupt request flag</th>
<th>Responsible function in ISR</th>
</tr>
</thead>
<tbody>
<tr>
<td>External interrupt 3</td>
<td>Macro ffrd_hal_clear_int_flag_line_1</td>
</tr>
<tr>
<td>(EIRR0_ER3)</td>
<td>defined in file ffrd_hal_mcu.h</td>
</tr>
<tr>
<td>ERAY reception interrupt</td>
<td>Function ffrd_ccal_get_status_interrupt_flags ( FRD_INT_LINE_1 )</td>
</tr>
<tr>
<td>(SIR.RXI)</td>
<td>defined in file ffrd_ccal_interrupt_service.c</td>
</tr>
</tbody>
</table>

Table 1-6 Interrupt flag reset
1.4.3 Detailed Description

1.4.3.1 Project folders

The folder structure of the example is as following:
In the main folder (ADA_96340_88121_FLEXRAY_dynamic1_int_v1) locates the workspace file 'ADA_96340_88121_FLEXRAY_dynamic1_int_v1.wsp', the introduction file 'readme.txt' and the following sub-folder:

- Node1_ffrd
  - includes Node1 project folder structure

- Node2_ffrd
  - includes Node2 project folder structure

- Generated_files\src_FlexConfig
  - includes '.chi' files generated by tools for CC initialization.

- ffrd
  - includes Fujitsu FlexRay driver, needs to be installed

1.4.3.2 Source files of the application

The C source files used for this application are mostly identical with those in the example dynamic1. Therefore only the differences are listed here.

1.4.3.2.1 Main.c

void init(void)

Function Init_extint_3() (defined in file extint.c) initialises external interrupt 3 for ERAY reception interrupt. Since the interrupt signal is high active, rising edge detection is used.

1.4.3.2.2 vectors.c

The entry for the external Interrupt 3 in the interrupt vector table is assigned with ISR ffrd_api_interrupt_line1 (defined in file ffrd_api_interrupt_service.c). Function InitIrqLevels() defines the interrupt level.

1.4.3.2.3 TTTask.c

Dynamic data reception via function ffrd_api_rx_handler_buffer() is not carried out in ttTASK(Nodex). CC is initialized by file 'dynamic_int_demo1_CHI_Nodex.chi' in function ffrd_api_include_chi().

void get_dynamic_message(void)

This function is a component of the ISR. It is called by function ffrd_api_interrupt_line1() after interrupt flags are cleared and performs dynamic data reception instead of ttTASK(Nodex).

1.4.3.2.4 ffrd_api_interrupt_service.c

After installation of the FlexRay driver all entries in the function ffrd_api_interrupt_line1() are assigned with default ISR ffrd_api_default_irq_handler(). For used interrupt type corresponding entry should be set.
1.4.3.2.5  ffrd_api_global_def.h

#define FFRD_INTERRUPT_SERVICE  YES

This parameter works as a compilation switch in file ffrd_api_interrupt_service.c. To use the function ffrd_api_interrupt_line1() for ERAY ISR this parameter should be set YES.
1.5 Example4: ADA_96340_88121_FLEXRAY_static1_DMA

This example shows how to use DMA to transfer data between CC and MCU.

1.5.1 General description

This example is an extension of the example static1. In addition to the usage of static slot for data transmission between two bus-nodes, DMA is enabled to transfer data between MCU and CC, respectively, between RAM and input/output buffer.

Pressing IN0 (input capture unit 0) button of the evaluation board FLASH-CAN-100P-340 creates interrupt to MCU. In the IN0 interrupt service routine a variable (isr_cnt) is counted up. Its value is transmitted to another bus-node via the FlexRay channel A and B (redundant transfer).

The reload timer 2 is used to represent FlexRay cycle length. Every 3 ms it causes an interrupt. Within the ISR the FlexRay task (ttTASK) is called to write data to input buffer for data transmission and read data from output buffer for data reception.

Since N1 resistor network is removed from the evaluation board to make the MCU pins available for external bus interface, LEDs of the evaluation board are not applicable to present the transmitted and received data. The terminal program SKwizard is therefore used to display data and CC status in monitor. Accordingly the evaluation board is connected to PC via UARTA(X3) and UART 0 is initialized in software.

The access to the FlexRay CC MB88121 is done via the FlexRay driver (FFRD V1.3).
1.5.1.1 Application flow

The application begins with initialisation of MCU and CC. Then reload timer 2 is started to generate the 3 ms FlexRay communication cycle via periodic interrupt. Within the ISR of reload timer 2 the FlexRay task is carried out through FlexRay driver. Meanwhile CC status is output to monitor.

Values accumulated by pressing IN0 button is sent in the static slot.

In FlexRay task DMA functionality is enabled to free the MCU from data transmission between peripheral resource (input/output buffer) and memory. DMA request is sent to MCU external interrupt, respectively, DMA is triggered by external interrupt 7.

---

**Figure 1-6 Application flow of the example static1_DMA**

**Reset / Power-on**
- Initialization of MCU and FlexRay CC
- Start FlexRay communication
- Carry out FlexRay task in every 3 ms
- In case of CC failure
  - Shutdown FlexRay

**FlexRay task every 3 ms**
- Reload timer 2 ISR: call ttTask()
- ttTask(): Transmission, reception of static data

**User Interaction via IN0 button**
- IN0 ISR: 
  - isr_cnt+1
  - set tx_flag

**Ext INT 7**
- Ext INT7 ISR: 
  - clear INT flag, disable the DMA request, reinitialisation of DMA registers
1.5.2 DMA

DMA enables automatic data transfer between memory and memory/peripheral resource registers. Thus MCU processing load is reduced and the transfer operation is accelerated. In this example MCU is involved in two types of data transmission:

1. peripheral resource (output buffer RDDS) to memory (variable sRx1,sRx2)
2. memory (variable buffer) to peripheral resource (input buffer WRDS)

The first one is for data reception. Received frame payload (20 byte) is copied from register RDDS (in chip select area 2, memory bank 10) to variable ‘sRx1’ and ‘sRx2’ (in RAM area, memory bank 00). The second one is for data transmission. Transfer data is copied from variable ‘buffer’ (in RAM area, memory bank 00) to register WRDS (in chip select area 2, memory bank 10).

The DMA process is:

1. The peripheral resource (CC) requests DMA transfer (CC pin DMA_REQ) by asserting an interrupt (MCU external interrupt 7).
2. This interrupt is handled by the DMA controller. DMA data transfer is started.
3. After each word (16 bit) is transferred, register DCT, IOA, BAP are updated.
4. If register DCT reaches 0, DMA transfer completes. External interrupt 7 is handled by MCU interrupt controller. Its ISR is carried out.

The corresponding register settings are listed below. DMA channel 0 is in use.

<table>
<thead>
<tr>
<th>Register</th>
<th>Data reception</th>
<th>Data transmission</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISEL0</td>
<td>Interrupt request selection</td>
<td>DISEL0=24 IRQ number of the ext. interrupt 7</td>
</tr>
<tr>
<td>DSR</td>
<td>DMA status register</td>
<td>DSR_DTE0=1 indicates the end of DMA transfer</td>
</tr>
<tr>
<td>DER</td>
<td>DMA enable register</td>
<td>DER_EN0=1 enable DMA channel 0</td>
</tr>
<tr>
<td>DCT0</td>
<td>Data count register</td>
<td>DCT0=20 byte frame payload length</td>
</tr>
<tr>
<td>IOA0</td>
<td>16bit address pointer</td>
<td>‘sRx1’ or ‘sRx2’ address</td>
</tr>
<tr>
<td>BAP0</td>
<td>24bit address pointer</td>
<td>RDDS address</td>
</tr>
<tr>
<td>DMACS0</td>
<td>DMA control register</td>
<td>DMACS0=0x0A IOA and BAP are updated. Word transfer from @BAP to @IOA</td>
</tr>
</tbody>
</table>

Table 1-7 DMA register settings

Notes:
1. CC pin DMA_REQ is physical connected to MCU pin4 (input capture unit 3) via jumper JP27. However ICU3 interrupt is proved to be unsuitable for this example. If
ICU3 interrupt is chosen as DMA trigger, then it requires 10 consecutive interrupt signal-edge to finish 20 byte data transfer, respectively, one interrupt one word. The reason is that ICU supports only signal-edge detection. After one word is transferred by DMA, the ICU3 interrupt is cleared by DMA controller so that 20 byte cannot be transferred in one time. Since signal level of DMA_REQ is controlled by MCU, level detection of the request signal should be selected for DMA, which makes the external interrupt preferred.

2. External INT0, INT1 are used by button INT0, INT1. External INT2 to INT6 are connected to CC pin INT0 to INT4. Therefore external interrupt 7 is initialised as DMA trigger.

3. To bypass the MCU IN3 pin, jumper JP27 of the adapter board should be set open. An additional cable is needed to connect JP27 and MCU pin56 (X2).

4. DMA should be reinitialised before using it again, since register DCT0, IOA0, BAP0 are changed during DMA transfer.

5. EVA chip MB96V300 supports 16 DMA channels, whereas MB9634x only 6. By using emulator unused DISEL registers should be initialised to 0xFF.

6. Since the connection between MCU and CC is 16-bit multiplexed mode. DMA must be initialised as 16-bit word transfer (DMACS0_BW=1). Otherwise the transfer result is wrong.

7. Register DCT0, IOA0, BAP0, DMACS0 must be initialized before setting DER_EN0 to ‘1’ because their initial values are undefined.

8. The linkage order of the project (right click the project name → ’set linkage order’) is important. The correct order is: api functions → ccal functions → application-specific functions (main.c, vectors.c, ttask.c…).

1.5.3 Detailed Description

1.5.3.1 Project folders

The folder structure of the example is as following:
In the main folder (ADA_96340_88121_FLEXRAY_static1_DMA_v2) locates the workspace file ‘DMA_static_v2.wsp’, the introduction file ‘readme.txt’ and the following sub-folder:

- Node1_ffrd
  - includes folder structure of project Node1

- Node2_ffrd
  - includes folder structure of project Node2

- Generated_files\src_FlexConfig
  - includes ‘.chi’ files generated by tools for CC initialization.

- ffrd
  - includes Fujitsu FlexRay driver, needs to be installed first
1.5.3.2 Source files of the application

The C source files used for this application are mostly identical with those in the example static1. Therefore only the differences are listed here.

1.5.3.2.1 Main.c

```c
void init_dma(void)
```

DMA initialization includes three parts. In this function register DSR and DER are first reset to disable the DMA functionality. Then the external interrupt 7 is arranged to trigger the DMA transfer on channel 0 (macro `ffrd_hal_dma_IRQ_select()`).

In function `ttTASK(Nodex)` register DMACS0 is initialised twice for data transmission and reception (macro `ffrd_hal_dma_configuration()`). The reason is that the direction of DMA transfer (bit DMACS0_BW) is different. It is from memory bank 00 to bank 10 by data transmission and vice versa by data reception.

In function `ffrd_ccal_init_dma_host()` register IOA0, DCT0, BAP0 are initialised. Their value are provided by function `ffrd_api_tx_handler_buffer()` and `ffrd_api_rx_handler_buffer()`. Since IOA is 16-bit address pointer and BAP is 24-bit, IOA is assigned to point memory bank 00 (RAM area) and BAP to bank 10 (chip select area 2). DMA channel 0 is enabled finally after all corresponding registers are initialised (macro `ffrd_hal_dma_enable()`).

Header-file ‘`ffrd_hal_mcu.h’` includes the definition of the used macros:

- `ffrd_hal_dma_enable()`
- `ffrd_hal_dma_IRQ_select()`
- `ffrd_hal_dma_configuration(val)`

1.5.3.2.2 vectors.c

The entry of the external Interrupt 7 in the interrupt vector table is assigned with ISR `IsrExtInt7` (defined in file `extint.c`). Function `InitlrqLevels()` defines the interrupt level.

1.5.3.2.3 TTTask.c

```c
buffer_payload_switch()
```

Since MCU is bypassed by DMA, user should manually correct the byte order of the frame payload so that other bus-node can interpret the received data properly. In this example this is done by function `buffer_payload_switch()`. An example by data transmission is listed below.
### 16 bit payload entry

<table>
<thead>
<tr>
<th>Payload entry value</th>
<th>Variable ‘buffer’ in memory bank 00</th>
<th>After byte order correction</th>
<th>WRDS register in memory bank 10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Memory address</td>
<td>Memory content</td>
<td>Memory content</td>
</tr>
<tr>
<td>0x1234</td>
<td>&amp;buffer+0</td>
<td>0x34</td>
<td>0x12</td>
</tr>
<tr>
<td></td>
<td>&amp;buffer +1</td>
<td>0x12</td>
<td>0x34</td>
</tr>
<tr>
<td>0x5678</td>
<td>&amp;buffer +2</td>
<td>0x78</td>
<td>0x56</td>
</tr>
<tr>
<td></td>
<td>&amp;buffer +3</td>
<td>0x56</td>
<td>0x78</td>
</tr>
</tbody>
</table>

### 32 bit payload entry

<table>
<thead>
<tr>
<th>Payload entry value</th>
<th>Variable ‘buffer’ in memory bank 00</th>
<th>After byte order correction</th>
<th>WRDS register in memory bank 10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Memory address</td>
<td>Memory content</td>
<td>Memory content</td>
</tr>
<tr>
<td>0x12345678</td>
<td>&amp;buffer+0</td>
<td>0x78</td>
<td>0x12</td>
</tr>
<tr>
<td></td>
<td>&amp;buffer +1</td>
<td>0x56</td>
<td>0x34</td>
</tr>
<tr>
<td></td>
<td>&amp;buffer +2</td>
<td>0x34</td>
<td>0x56</td>
</tr>
<tr>
<td></td>
<td>&amp;buffer +3</td>
<td>0x12</td>
<td>0x78</td>
</tr>
</tbody>
</table>

**Table 1-8 Byte ordering correction**

#### 1.5.3.2.4 extint.c

External interrupt 7 is initialised to detect signal ‘HIGH’ level. Its ISR is performed after DMA transfer is finished. DMA request signal from CC is pulled down (deactivated) through function `ffrd_hal_write32(0x00000001,FFRD_REG_DMAS)`. DMA is reinitialised.

#### 1.5.3.2.5 ffrd_api_global_def.h

```c
#define FFRD_DMA_SERVICE YES
```

This parameter works as a compilation switch in file

- `ffrd_ccal_rx_handler.c`
- `ffrd_ccal_tx_handler.c`
- `ffrd_ccal_dma_service.c`
- `ffrd_hal_mcu.h`
- application specific functions: main.c, vectors.c, extint.c, TTTask.c

To enable the DMA functionality this parameter should be set to YES. If it is switched off, this example is identical with example ‘static1’. User can accordingly compare the differences between two mechanisms.

```c
#define FFRD_DMA_CHANNEL 0
```
The available DMA channel for MB96F348RS is 0 to 5. This parameter is relevant for DMA initialization in file

- `ffrd_ccal_dma_service.c`
- `ffrd_hal_mcu.h`
2 FlexRay software driver

This chapter describes the FlexRay software driver included in the CD.

2.1 Fujitsu FlexRay Driver (FFRD)

Within the delivered CD there is also a FlexRay driver, called Fujitsu FlexRay Driver (FFRD). This driver includes C source code and can be used only for evaluation purposes (see license agreement). With this FlexRay driver it is possible to access the FlexRay communication controller without knowing details of all the CC registers.

2.1.1 Installation

Browse the folder “Software\FlexRay_driver\ffrd” and click the ‘ffrd_v1-3.exe’ file. To use the Fujitsu FlexRay driver the license agreement must be first accepted. Choose the destination folder and the driver library will be installed.

![Figure 2-1: FlexRay Driver installation](image)

2.1.2 Structure

The Installation generates the “FujitsuFlexRayDriver” folder which includes following subfolder:

- Additional Information
- FFRD
In the “Additional Information” folder locates the documentation of the FlexRay driver. The API user manual is stored in subfolder “PDF”. The HTML based documentation is stored in subfolder “HTML”. The documentation ‘Fujitsu-FlexRay-Driver-Manual-V13.pdf’ explains all ‘api’ functions of the FlexRay driver.

The folder “FFRD” contains the C source files of the FlexRay driver.

### 2.1.3 Fujitsu FlexRay driver setup

In subfolder “api” following files include application-specific settings:

- `ffrd_api_global_def.h` in ‘api/include’
- `ffrd_api_init_chi_def.h` in ‘api/include’
- `ffrd_api_init_chi.c` in ‘api/src’

In case of several projects with different settings in one workspace, these files should be copied into each project folder (\Nodex_ffrd\Src).

### 2.1.4 License Agreement

Software Disclaimer

Disclaimer of Warranty:
This software is provided by Fujitsu Microelectronics Europe GmbH “as is” and any express or implied warranty is disclaimed, including, but not limited to, the implied warranty of merchantability, title, non-infringement and fitness for a particular purpose.

Limitation of Liability:

In no event and on no legal basis, whether in tort (including negligence), contract, or otherwise, unless required by pertaining law or agreed to in writing, shall Fujitsu Microelectronics Europe GmbH be liable for damages, including any direct, indirect, special, incidental, exemplary, or consequential damages of any character arising as a result of the use of or the inability to use this software (including, but not limited to, damages for loss of goodwill, procurement of substitute goods or services, work stoppage, computer failure or malfunction, or loss of use, data or profits and all other commercial damages or losses), even if advised of the possibility of such damages.

Should one of the above stipulations be, or become, invalid and/or unenforceable, the remaining stipulations shall remain fully effective.

Note:
The enclosed software is intended to be used for evaluation purposes only. Any use beyond evaluation purposes which may include qualifications or a certain level of maturity (e.g. in series/mass production or test vehicles) is at disposal, risk, and liability of the user of this software.
2.2  DECOMSYS::COMMSTACK V1.8.2 library for 16FX series

The DECOMSYS::COMMSTACK V1.8.2 for Fujitsu 16FX series is a third party driver from company DECOMSYS. The initialisation code for this driver is generated from DECOMSYS::DesignerPro Generator.

2.2.1  Installation

Browse the folder “Software\FlexRay_driver\dcsCstFr” on the CD. Click the ‘dcsCstFr_V182_16FX_v1-0.exe’ file. To use the DECOMSYS::COMMSTACK library the license agreement must be accepted. Select the destination folder. The driver library will be installed into the selected folder.

![Figure 2-2: FlexRay Driver installation](image-url)
2.2.2 Structure
The Installation generates the “dcsCstFr” folder which includes following subfolder:

- Documentation
- include
- lib
- cfg

In the “Documentation” folder COMMSTACK_UserManual.pdf explains all functions of the FlexRay driver library. In the “lib” folder locates the FlexRay driver library. The header files are located in the folder “include”. The file in the “cfg” folder needs to be included into the application to define the E-Ray register offset.

2.2.3 License Agreement

DECOMSYS::COMMSTACK Library License Agreement

The DECOMSYS::COMMSTACK library is provided by DECOMSYS. It is the property of that company. Fujitsu Microelectronics Europe GmbH expressly disclaims all warranty, expressed, implied or statutory, including but not limited to any implied warranty of merchantability, fitness for a particular purpose or non-infringement.

The DECOMSYS::COMMSTACK library must be used with Fujitsu evaluation boards, only.

For any other purposes, including but not limited to licensing issues, technical problems contact DECOMSYS.

Contact Details:
DECOMSYS - Dependable Computer Systems,
Hardware und Software Entwicklung GmbH
Stumpergasse 48/28
A-1060 Vienna, Austria
Web: http://www.decomsys.com
3 Configuration Tools

This chapter describes the configuration tool included in the CD.

3.1 FlexConfig®

The FlexConfig® Tool is a configuration tool for FlexRay communication controllers. With FlexConfig® all the necessary parameters can be set via a convenient windows application. Every parameter is checked against general limits or specific constraints so that no faulty configuration is possible.

The FlexConfig® Tool is provided by the company TransferZentrum Mikroelektronik (TZM). This demo version for the adapter board is limited to 4 ID’s for each cluster.

Figure 3-1: FlexConfig® Start Window

3.1.1 Installation

To install the FlexConfig® Tool browse on the adapter board CD to Software\FlexConfig® folder and click on the installation file. Then follow the instruction of the installation Wizard.
3.1.2 Project examples

For the FlexRay example static1 and dynamic1 the FlexConfig project file (*.pro) is also available. Find the project file at the example folder in “Generated_files\src_FlexConfig”

3.1.3 License Agreement

DISCLAIMER OF FUJITSU MICROELECTRONICS EUROPE GMBH
FLEXCONFIG SOFTWARE LICENSE AGREEMENT

THE FLEXCONFIG SOFTWARE IS THE INTELLECTUAL PROPERTY OF TRANSFERZENTRUM MIKROELEKTRONIK AND WILL BE EXCLUSIVELY PROVIDED BY TRANSFERZENTRUM MIKROELEKTRONIK. FUJITSU MICROELECTRONICS EUROPE GMBH EXPRESSLY DISCLAIMS ALL WARRANTY, EXPRESSED, IMPLIED OR STATUTORY, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY’S INTELLECTUAL PROPERTY.

FOR ANY OTHER PURPOSES, INCLUDING BUT NOT LIMITED TO LICENSING ISSUES AND/OR TECHNICAL PROBLEMS PLEASE CONTACT TRANSFERZENTRUM MIKROELEKTRONIK.

CONTACT DETAILS:
TRANSFERZENTRUM MIKROELEKTRONIK
ROBERT-BOSCH-STRASSE 6
D-73037 GÖPPINGEN, GERMANY
WEB: HTTP://WWW.TZM.DE
3.2 DECOMSYS::DESIGNERPRO

Within the delivery package of the ADA-96340-88121-FLEXRAY there is an additional CD including the DECOMSYS::TOOLCHAIN. This version has full functionality, but restricted runtime of 30 days, which can be obtained by registering for a demo license (Follow instructions on CD). With the DECOMSYS::TOOLCHAIN it is possible to generate the FlexRay CC initialisation code by a graphical interface.

3.2.1 Installation

Insert the DECOMSYS::TOOLCHAIN CD, the CD automatically opens a dialog window. If auto start function is disabled at your PC, start manually via double-click on CD_Start.exe in the root folder of the CD. Via “Install and Explore Design Tools” the installation window is selected.

Install and register the Tool chain as shown in this window.

In the Main menu find an overview about the tools via the “Deploy FlexRay with FUJITSU Eval-Kit” selection.

![DECOMSYS Demo CD Main Menu](image)

3.2.2 Project examples

For the FlexRay examples 91460_static1_91467d and 91460_dynamic1_91467d there is also the DECOMSYS::DESIGNERPro project file (*.bor) available. Find the project files at the example folder in “Generated_files\src_dcsCstFr”
4 Appendix

4.1 Information on the WWW

Information about FUJITSU MICROELECTRONICS Products can be found on the following Internet pages:

Microcontrollers (8-, 16- and 32bit),
Datasheets and Hardware Manuals, Support Tools (Hard- and Software)


FlexRay products:

http://www.fujitsu.com/emea/services/microelectronics/micros/flexray/

Automotive products: MCU, Graphic display controller, MPEG en/decoder, gyro sensors

http://www.fujitsu.com/emea/services/industries/automotive/

Linear Products: Power Management, A/D and D/A Converters

http://www.fujitsu.com/emea/services/microelectronics/linears/

Media Products: SAW filters, acoustic resonators and VCOs

http://www.fujitsu.com/emea/services/microelectronics/saw/

For more information about FUJITSU MICROELECTRONICS

http://emea.fujitsu.com/microelectronics
4.2 Figures

Figure 1-1: Application flow of the example static1 ................................................................. 9
Figure 1-2: Signal flow of the example static1 ...................................................................... 10
Figure 1-3: Application flow of the example dynamic1 .......................................................... 20
Figure 1-4: Signal flow of the example dynamic1 ................................................................. 21
Figure 1-5 Application flow of example dynamic_int1 ......................................................... 30
Figure 1-6 Application flow of the example static1_DMA .................................................... 35
Figure 2-1: FlexRay Driver installation ................................................................................. 41
Figure 2-2: FlexRay Driver installation ................................................................................. 43
Figure 3-1: FlexConfig® Start Window ................................................................................ 45
Figure 3-1: DECOMSYS Demo CD Main Menu ................................................................. 47

4.3 Tables

Table 1-1: FlexRay settings of the example static1 .............................................................. 10
Table 1-2 Frame payload ..................................................................................................... 18
Table 1-3: Dynamic1 example FlexRay settings ................................................................ 21
Table 1-4 Register settings for ERAY interrupt .................................................................... 31
Table 1-5 Interrupt signal map ......................................................................................... 31
Table 1-6 Interrupt flag reset ............................................................................................. 31
Table 1-7 DMA register settings ......................................................................................... 36
Table 1-8 Byte ordering correction .................................................................................... 39
4.4 Abbreviations

ADC  Analogue Digital Converter
ASSP Application Specific Standard Product
CAN Controller Area Network
CC Communication Controller (MB88121B)
CS Chip Select
FME Fujitsu Microelectronics Europe GmbH
FR Fujitsu Risc
HW Hardware
ISR Interrupt service routine
INT Interrupt
LB Lower Byte
LIN Local Interconnect Network
MCU Microcontroller Unit (16FX MB96340)
PCB Printed Circuit Board
SRAM Static Random Access Memory
TZM Transferzentrum Mikroelektronik
UART Universal Asynchronous Receiver Transmitter
UB Upper Byte
UG User Guide
USART Universal Synchronous Asynchronous Receiver Transmitter

-- END --