C_CAN FD8
Controller Area Network
User’s Manual
Revision 2.1.0
22.01.2015

Robert Bosch GmbH
Automotive Electronics
LEGAL NOTICE

© Copyright 1998-2015 by Robert Bosch GmbH and its licensors. All rights reserved.

"Bosch" is a registered trademark of Robert Bosch GmbH.

The content of this document is subject to continuous developments and improvements. All particulars and its use contained in this document are given by BOSCH in good faith.

NO WARRANTIES: TO THE MAXIMUM EXTENT PERMITTED BY LAW, NEITHER THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, NOR ANY PERSON, EITHER EXPRESSLY OR IMPLICITLY, WARRANTS ANY ASPECT OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, INCLUDING ANY OUTPUT OR RESULTS OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO UNLESS AGREED TO IN WRITING. THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IS BEING PROVIDED "AS IS", WITHOUT ANY WARRANTY OF ANY TYPE OR NATURE, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND ANY WARRANTY THAT THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IS FREE FROM DEFECTS.

ASSUMPTION OF RISK: THE RISK OF ANY AND ALL LOSS, DAMAGE, OR UNSATISFACTORY PERFORMANCE OF THIS SPECIFICATION (RESPECTIVELY THE PRODUCTS MAKING USE OF IT IN PART OR AS A WHOLE), SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO RESTS WITH YOU AS THE USER. TO THE MAXIMUM EXTENT PERMITTED BY LAW, NEITHER THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, NOR ANY PERSON EITHER EXPRESSLY OR IMPLICITLY, MAKES ANY REPRESENTATION OR WARRANTY REGARDING THE APPROPRIATENESS OF THE USE, OUTPUT, OR RESULTS OF THE USE OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, BEING CURRENT OR OTHERWISE. NOR DO THEY HAVE ANY OBLIGATION TO CORRECT ERRORS, MAKE CHANGES, SUPPORT THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, DISTRIBUTE UPDATES, OR PROVIDE NOTIFICATION OF ANY ERROR OR DEFECT, KNOWN OR UNKNOWN. IF YOU RELY UPON THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, YOU DO SO AT YOUR OWN RISK, AND YOU ASSUME THE RESPONSIBILITY FOR THE RESULTS. SHOULD THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL LOSSES, INCLUDING, BUT NOT LIMITED TO, ANY NECESSARY SERVICING, REPAIR OR CORRECTION OF ANY PROPERTY INVOLVED TO THE MAXIMUM EXTEND PERMITTED BY LAW.

DISCLAIMER: IN NO EVENT, UNLESS REQUIRED BY LAW OR AGREED TO IN WRITING, SHALL THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS OR ANY PERSON BE LIABLE FOR ANY LOSS, EXPENSE OR DAMAGE, OF ANY TYPE OR NATURE ARISING OUT OF THE USE OF, OR INABILITY TO USE THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO,
INCLUDING, BUT NOT LIMITED TO, CLAIMS, SUITS OR CAUSES OF ACTION INVOLVING ALLEGED INFRINGEMENT OF COPYRIGHTS, PATENTS, TRADEMARKS, TRADE SECRETS, OR UNFAIR COMPETITION.

INDEMNIFICATION: TO THE MAXIMUM EXTEND PERMITTED BY LAW YOU AGREE TO INDEMNIFY AND HOLD HARMLESS THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, AND EMPLOYEES, AND ANY PERSON FROM AND AGAINST ALL CLAIMS, LIABILITIES, LOSSES, CAUSES OF ACTION, DAMAGES, JUDGMENTS, AND EXPENSES, INCLUDING THE REASONABLE COST OF ATTORNEYS’ FEES AND COURT COSTS, FOR INJURIES OR DAMAGES TO THE PERSON OR PROPERTY OF THIRD PARTIES, INCLUDING, WITHOUT LIMITATIONS, CONSEQUENTIAL, DIRECT AND INDIRECT DAMAGES AND ANY ECONOMIC LOSSES, THAT ARISE OUT OF OR IN CONNECTION WITH YOUR USE, MODIFICATION, OR DISTRIBUTION OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, ITS OUTPUT, OR ANY ACCOMPANYING DOCUMENTATION.


MANDATORY LAW SHALL BE UNAFFECTED BY THE FOREGOING PARAGRAPHS.

INTELLECTUAL PROPERTY OWNERS/COPYRIGHT OWNERS/CONTRIBUTORS: ROBERT BOSCH GMBH, ROBERT BOSCH PLATZ 1, 70839 GERLINGEN, GERMANY AND ITS LICENSORS.
1. About this Document ........................................... 7
  1.1. Change Control ........................................... 7
    1.1.1. Current Status ...................................... 7
    1.1.2. Change History ...................................... 7
  1.2. Conventions ............................................. 7
  1.3. Scope ................................................... 7
  1.4. References .............................................. 7
  1.5. Terms and Abbreviations .................................. 8
2. Functional Description ......................................... 9
  2.1. Functional Overview ...................................... 9
  2.2. Block Diagram .......................................... 10
  2.3. Operating Modes ......................................... 11
    2.3.1. Software Initialisation ............................. 11
    2.3.2. CAN Message Transfer ............................... 11
    2.3.3. CAN FD Operation Mode .............................. 12
    2.3.4. Restricted Operation Mode ......................... 12
    2.3.5. Disabled Automatic Retransmission ................. 12
    2.3.6. Test Mode .......................................... 12
      2.3.6.1. Silent Mode ................................... 13
      2.3.6.2. Loop Back Mode ................................ 13
      2.3.6.3. Loop Back combined with Silent Mode ....... 14
      2.3.6.4. Basic Mode .................................. 14
      2.3.6.5. Software control of Pin can_tx ............ 15
3. Programmer’s Model ........................................... 17
  3.1. Hardware Reset Description .............................. 18
    3.1.1. Coding of Register Bit access / reset ........... 18
  3.2. CAN Protocol Related Registers ............................ 19
    3.2.1. CAN Control Register (addresses 0x01 & 0x00) .... 19
    3.2.2. Status Register (addresses 0x03 & 0x02) ........ 20
      3.2.2.1. Status Interrupts .............................. 21
    3.2.3. Error Counter (addresses 0x05 & 0x04) ........... 21
    3.2.4. Bit Timing Register (addresses 0x07 & 0x06) .... 21
    3.2.5. Test Register (addresses 0x08 & 0x0A) ........... 22
    3.2.6. BRP Extension Register (addresses 0x0D & 0x0C) .. 23
    3.2.7. FD Control Register (addresses 0x27 & 0x26) ..... 23
    3.2.8. Nominal Bit Timing Register 1 (addresses 0x29 & 0x28) 24
    3.2.9. Nominal Bit Timing Register 2 (addresses 0x2B & 0x2A) 24
    3.2.10. Data Bit Timing Register 1 (addresses 0x2D & 0x2C) 25
    3.2.11. Data Bit Timing Register 2 (addresses 0x2F & 0x2E) 25
    3.2.12. Transmitter Delay Compensation Register (addresses 0x31 & 0x30) 25
    3.2.13. SSP Position Register (addresses 0x33 & 0x32) ... 26
    3.2.14. Error Logging Register (addresses 0x35 & 0x34) .. 26
    3.2.15. FD Status Register (addresses 0x37 & 0x36) .... 27
  3.3. Message Interface Register Sets .......................... 29
    3.3.1. IFx Command Request Registers .................. 29
3.3.2. IFx Command Mask Registers ........................................... 30
  3.3.2.1. Direction = Write ................................................. 30
  3.3.2.2. Direction = Read .................................................. 32
3.3.3. IFx Message Buffer Registers ....................................... 33
  3.3.3.1. IFx Mask Registers .............................................. 33
  3.3.3.2. IFx Arbitration Registers ....................................... 33
  3.3.3.3. IFx Message Control Registers .................................. 34
  3.3.3.4. IFx Data A and Data B Registers ............................. 34
3.3.4. Message Object in the Message Memory .............................. 35

3.4. Message Handler Registers ............................................. 38
  3.4.1. Interrupt Register (addresses 0x09 & 0x08) ...................... 38
  3.4.2. Transmission Request Registers .................................. 38
  3.4.3. New Data Registers ................................................ 39
  3.4.4. Interrupt Pending Registers ...................................... 39
  3.4.5. Message Valid Registers .......................................... 39

3.5. Core Release Registers .................................................. 40
  3.5.1. Core Release Low (addresses 0x39 & 0x38) ........................ 40
  3.5.2. Core Release High (addresses 0x3B & 0x3A) ...................... 40

4. CAN Application ............................................................ 41
  4.1. Management of Message Objects ...................................... 41
  4.2. Message Handler State Machine ...................................... 41
    4.2.1. Data Transfer from / to Message RAM .......................... 41
    4.2.2. Transmission of Messages ...................................... 42
    4.2.3. Acceptance Filtering of Received Messages .................... 43
    4.2.3.1. Reception of Data Frame ..................................... 43
    4.2.3.2. Reception of Remote Frame ................................. 43
    4.2.4. Receive / Transmit Priority .................................... 44
  4.3. Configuration of a Transmit Object ................................ 44
  4.4. Updating a Transmit Object .......................................... 44
  4.5. Configuration of a Receive Object .................................. 45
  4.6. Handling of Received Messages ...................................... 45
  4.7. Configuration of a FIFO Buffer ...................................... 46
  4.8. Reception of Messages with FIFO Buffers .......................... 46
    4.8.1. Reading from a FIFO Buffer .................................... 46
  4.9. Handling of Interrupts ................................................ 48
  4.10. CAN with Flexible Data-Rate ......................................... 48
    4.10.1. Frame Transmission in CAN FD Mode .......................... 49
    4.10.2. Frame Reception in CAN FD Mode .............................. 50
    4.10.3. Configuration of the Transmitter Delay Compensation .......... 50
      4.10.3.1. Transmitter Delay Compensation Measurement ............ 51
      4.10.3.2. Fixed Transmitter Delay .................................. 51

5. Appendix ................................................................. 53
  5.1. List of Figures ........................................................ 53
1. About this Document

1.1 Change Control

1.1.1 Current Status
Revision 2.1.0

1.1.2 Change History

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draft</td>
<td>17.12.1998</td>
<td>First Draft</td>
</tr>
<tr>
<td>Draft</td>
<td>07.04.1999</td>
<td>DAR Mode added</td>
</tr>
<tr>
<td>Draft</td>
<td>20.04.1999</td>
<td>Signal names modified</td>
</tr>
<tr>
<td>Revision 1.0</td>
<td>28.09.1999</td>
<td>Revised version</td>
</tr>
<tr>
<td>Revision 1.1</td>
<td>10.12.1999</td>
<td>BRP Extension Register added</td>
</tr>
<tr>
<td>Revision 1.2</td>
<td>06.06.2000</td>
<td>Document restructured</td>
</tr>
<tr>
<td>Revision 2.0</td>
<td>18.12.2013</td>
<td>Upgraded for CAN FD8 support</td>
</tr>
<tr>
<td>Revision 2.0.1</td>
<td>17.03.2014</td>
<td>Bits PXE, PXHD, and EFBI added to FD Control Register, minor amendments and textual enhancements</td>
</tr>
<tr>
<td>Revision 2.1.0</td>
<td>22.01.2015</td>
<td>Bit NISO added, description of bit TDCNM and bit fields TDCO, TDCF, and SSPP updated</td>
</tr>
</tbody>
</table>

1.2 Conventions

The following conventions are used within this User’s Manual.

**Helvetica bold** Names of bits and signals

**Helvetica italic** States of bits and signals

1.3 Scope

This document describes the C_CAN FD8 module and its features from the application programmer’s point of view. All information necessary to integrate the C_CAN FD8 module into an user-defined ASIC is located in the C_CAN FD8 Module Integration Guide.

1.4 References

This document refers to the following documents.

<table>
<thead>
<tr>
<th>Ref</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ISO</td>
<td>ISO 11898-1: CAN data link layer and physical signalling</td>
</tr>
<tr>
<td>2</td>
<td>AE/PJ-SCI</td>
<td>C_CAN FD8 Module Integration Guide</td>
</tr>
</tbody>
</table>
### 1.5 Terms and Abbreviations

This document uses the following terms and abbreviations.

<table>
<thead>
<tr>
<th>Term</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSP</td>
<td>Bit Stream Processor</td>
</tr>
<tr>
<td>BTL</td>
<td>Bit Timing Logic</td>
</tr>
<tr>
<td>CAN</td>
<td>Controller Area Network</td>
</tr>
<tr>
<td>CAN FD</td>
<td>Controller Area Network with Flexible Data-rate</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check Register</td>
</tr>
<tr>
<td>DLC</td>
<td>Data Length Code</td>
</tr>
<tr>
<td>EML</td>
<td>Error Management Logic</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>mtq</td>
<td>minimum time quantum = system clock period</td>
</tr>
<tr>
<td>SJW</td>
<td>Synchronization Jump Width</td>
</tr>
<tr>
<td>system clock</td>
<td>clock input of the C_CAN FD8 module</td>
</tr>
<tr>
<td>SSP</td>
<td>Secondary Sample Point</td>
</tr>
<tr>
<td>tq</td>
<td>time quantum</td>
</tr>
<tr>
<td>TDC</td>
<td>Transmitter Delay Compensation</td>
</tr>
<tr>
<td>TSEG1</td>
<td>Time Segment before Sample Point</td>
</tr>
<tr>
<td>TSEG2</td>
<td>Time Segment after Sample Point</td>
</tr>
<tr>
<td>TTCAN</td>
<td>Time-Triggered CAN</td>
</tr>
</tbody>
</table>
2. Functional Description

2.1 Functional Overview

The C_CAN FD8 is a CAN module that can be integrated as stand-alone device or as part of
an ASIC. It is described in VHDL on RTL level, prepared for synthesis. It consists of the
components (see figure 1) CAN Core, Message RAM, Message Handler, Control Registers,
and Module Interface.

The CAN_Core performs communication according to ISO11898-1, 2003. It also supports
CAN FD communication with up to 8 byte data fields. For the connection to the physical layer
additional transceiver hardware is required.

For communication on a CAN network, individual Message Objects are configured. The
Message Objects and Identifier Masks for acceptance filtering of received messages are
stored in the Message RAM.

All functions concerning the handling of messages are implemented in the Message Handler. Those functions are the acceptance filtering, the transfer of messages between the CAN Core and the Message RAM, and the handling of transmission requests as well as the generation of the module interrupt.

The register set of the C_CAN FD8 can be accessed directly by an external CPU via the
module interface. These registers are used to control/configure the CAN Core and the
Message Handler and to access the Message RAM.

The Module Interfaces delivered with the C_CAN FD8 module can easily be replaced by a
customized module interface adapted to the needs of the user.

The C_CAN FD8 implements the following features:

• Conform to ISO11898-1, 2003
• CAN FD with up to 8 byte data fields supported,
  FD format currently being integrated into ISO11898-1
• 32 Message Objects
• Each Message Object has its own identifier mask
• Programmable FIFO mode (concatenation of Message Objects)
• Maskable interrupt
• **Disabled** Automatic Retransmission mode for Time Triggered CAN applications
• Programmable loop-back mode for self-test operation
• Example of an 8-bit non-multiplex Motorola HC08 compatible module interface
• Example of a 16-bit module interfaces to the AMBA APB bus from ARM

**Note:** The C_CAN FD8 is software compatible to the C_CAN as long as the reserved
addresses of the C_CAN are not accessed.
2.2 Block Diagram

The design consists of the following functional blocks (see figure 1):

**CAN Core**
CAN Protocol Controller and Rx/Tx Shift Register for serial/parallel conversion of messages.

**Message RAM**
Stores Message Objects and Identifier Masks.

**Registers**
All registers used to control and to configure the C_CAN FD8 module.

**Message Handler**
State Machine that controls the data transfer between the Rx/Tx Shift Register of the CAN Core and the Message RAM as well as the generation of interrupts as programmed in the Control and Configuration Registers.

**Module Interface**
The C_CAN FD8 module interfaces are documented in the C_CAN FD8 Module Integration Guide.

![Block Diagram of the C_CAN FD8](image)

Figure 1: Block Diagram of the C_CAN FD8
2.3 Operating Modes

2.3.1 Software Initialisation

The software initialization is started by setting the bit `Init` in the CAN Control Register, either by software or by a hardware reset, or by going `Bus_Off`.

While `Init` is set, all message transfer from and to the CAN bus is stopped, the status of the CAN bus output `can_t"x` is *recessive* (HIGH). The counters of the EML are unchanged. Setting `Init` does not change any configuration register.

To initialize the CAN Controller, the CPU has to set up the Bit Timing Register and each Message Object. If a Message Object is not needed, it is sufficient to set its `MsgVal` bit to not valid. Otherwise, the whole Message Object has to be initialized.

Access to the Bit Timing Register and to the BRP Extension Register for the configuration of the bit timing is enabled when both bits `Init` and `CCE` in the CAN Control Register are set.

Resetting `Init` (by CPU only) finishes the software initialisation. Afterwards the Bit Stream Processor BSP synchronizes itself to the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive *recessive* bits (equivalent to `Bus Idle`) before it can take part in bus activities and starts the message transfer.

The initialization of the Message Objects is independent of `Init` and can be done on the fly, but the Message Objects should all be configured to particular identifiers or set to not valid before the BSP starts the message transfer. To change the configuration of a Message Object during normal operation, the CPU has to start by setting `MsgVal` to not valid. When the configuration is completed, `MsgVal` is set to valid again.

2.3.2 CAN Message Transfer

Once the C_CAN FD8 is initialized and `Init` is reset to zero, the C_CAN FD8’s CAN Core synchronizes itself to the CAN bus and starts the message transfer.

Received messages are stored into their appropriate Message Objects if they pass the Message Handler’s acceptance filtering. The whole message including all arbitration bits, DLC and eight data bytes is stored into the Message Object. If the Identifier Mask is used, the arbitration bits which are masked to “don’t care” may be overwritten in the Message Object.

The CPU may read or write each message any time via the Interface Registers, the Message Handler guarantees data consistency in case of concurrent accesses.

Messages to be transmitted are updated by the CPU. If a permanent Message Object (arbitration and control bits set up during configuration) exists for the message, only the data bytes are updated and then `TxRqst` bit with `NewDat` bit are set to start the transmission. If several transmit messages are assigned to the same Message Object (when the number of Message Objects is not sufficient), the whole Message Object has to be configured before the transmission of this message is requested.

The transmission of any number of Message Objects may be requested at the same time, they are transmitted subsequently according to their internal priority. Messages may be updated or set to not valid any time, even when their requested transmission is still pending. The old data will be discarded when a message is updated before its pending transmission has started. Depending on the configuration of the Message Object, the transmission of a message may be requested autonomously by the reception of a remote frame with a matching identifier.

*Note:* Remote frames are always transmitted in Classical CAN format.
2.3.3 CAN FD Operation Mode

The C_CAN FD8 also supports CAN FD operation for data fields up to 8 byte. For configuration/control/status of CAN FD operation the new registers from address CAN Base + 0x26 to CAN Base + 0x36 have been introduced. To enable CAN FD operation bit FDOE in the FD Control Register has to be set. This can only be done while bits CCE and Init in the CAN Control Register are set.

2.3.4 Restricted Operation Mode

In Restricted Operation Mode the node is able to receive data and remote frames and to give acknowledge to valid frames, but it does not send data frames, remote frames, active error frames, or overload frames. In case of an error condition or overload condition, it does not send dominant bits (error or overload flags), instead it waits for the occurrence of bus idle condition to resynchronize itself to the CAN communication. The error counters (REC, TEC) are frozen while Error Logging is active (CEL0, CELN). The Host can set the C_CAN FD8 into Restricted Operation mode by setting bit REOM in the FD Control Register. The bit can only be written by the Host when both CCE and Init are set to ‘1’.

The Restricted Operation Mode can be used in applications that adapt themselves to different CAN bit rates. In this case the application tests different bit rates and leaves the Restricted Operation Mode after it has received a valid frame.

Note: The Restricted Operation Mode must not be combined with the Loop Back Mode (internal or external).

2.3.5 Disabled Automatic Retransmission

According to the CAN Specification (see ISO11898-1, 6.3.3 Recovery Management), the C_CAN FD8 provides means for automatic retransmission of frames that have lost arbitration or that have been disturbed by errors during transmission. The frame transmission service will not be confirmed to the user before the transmission is successfully completed. By default, this means for automatic retransmission is enabled. It can be disabled to enable the C_CAN FD8 to work within a Time-Triggered CAN (TTCAN, see ISO11898-4) environment.

The Disabled Automatic Retransmission mode is enabled by programming bit DAR in the CAN Control Register to one. In this operation mode the programmer has to consider the different behaviour of bits TxRqst and NewDat in the Control Registers of the Message Buffers:

- When a transmission starts, bit TxRqst of the respective Message Buffer is reset, while bit NewDat remains set.
- When the transmission completed successfully bit NewDat is reset.

When a transmission failed (lost arbitration or error) bit NewDat remains set. To restart the transmission the CPU has to set TxRqst back to one.

2.3.6 Test Mode

The Test Mode is entered by setting bit Test in the CAN Control Register to one. In Test Mode the bits Tx1, Tx0, LBack, Silent and Basic in the Test Register are writable. Bit Rx monitors the state of pin can_rx and therefore is only readable. All Test Register functions are disabled when bit Test is reset to zero. The Test Mode functions as described in the following subsections are intended for device tests outside normal operation. These functions should be used carefully. Switching between Test Mode functions and normal operation while communication is running (Init = ‘0’) should be avoided.
2.3.6.1 Silent Mode

In ISO 11898-1, the Silent Mode is called the Bus Monitoring Mode. The CAN Core can be set in Silent Mode by programming the Test Register bit Silent to one.

In Silent Mode, the C_CAN FD8 is able to receive valid data frames and valid remote frames, but it sends only recessive bits on the CAN bus and it cannot start a transmission. If the CAN Core is required to send a dominant bit (ACK bit, overload flag, active error flag), the bit is rerouted internally so that the CAN Core monitors this dominant bit, although the CAN bus may remain in recessive state. The Silent Mode can be used to analyse the traffic on a CAN bus without affecting it by the transmission of dominant bits (Acknowledge Bits, Error Frames). Figure 2 shows the connection of signals can_tx and can_rx to the CAN Core in Silent Mode.

![Figure 2: CAN Core in Silent Mode](image)

2.3.6.2 Loop Back Mode

The CAN Core can be set in Loop Back Mode by programming the Test Register bit LBack to one. In Loop Back Mode, the CAN Core treats its own transmitted messages as received messages and stores them (if they pass acceptance filtering) into a Receive Buffer. Figure 3 shows the connection of signals can_tx and can_rx to the CAN Core in Loop Back Mode.

![Figure 3: CAN Core in Loop Back Mode](image)

This mode is provided for self-test functions. To be independent from external stimulation, the CAN Core ignores acknowledge errors (recessive bit sampled in the acknowledge slot of a data/remote frame) in Loop Back Mode. In this mode the CAN Core performs an internal feedback from its Tx output to its Rx input. The actual value of the can_rx input pin is disregarded by the CAN Core. The transmitted messages can be monitored at the can_tx pin.
2.3.6.3 Loop Back combined with Silent Mode

It is also possible to combine Loop Back Mode and Silent Mode by programming bits `LBack` and `Silent` to one at the same time. This mode can be used for a “Hot Selftest”, meaning the C_CAN FD8 can be tested without affecting a running CAN system connected to the pins `can_tx` and `can_rx`. In this mode the `can_rx` pin is disconnected from the CAN Core and the `can_tx` pin is held recessive. Figure 4 shows the connection of signals `can_tx` and `can_rx` to the CAN Core in case of the combination of Loop Back Mode with Silent Mode.

![Figure 4: CAN Core in Loop Back combined with Silent Mode](image)

2.3.6.4 Basic Mode

The CAN Core can be set in Basic Mode by programming the Test Register bit `Basic` to one. In this mode the C_CAN FD8 module runs without the Message RAM.

The IF1 Registers are used as Transmit Buffer. The transmission of the contents of the IF1 Registers is requested by writing the `Busy` bit of the IF1 Command Request Register to ‘1’. The IF1 Registers are locked while the `Busy` bit is set. The `Busy` bit indicates that the transmission is pending.

As soon the CAN bus is idle, the IF1 Registers are loaded into the shift register of the CAN Core and the transmission is started. When the transmission has completed, the `Busy` bit is reset and the locked IF1 Registers are released.

A pending transmission can be aborted at any time by resetting the `Busy` bit in the IF1 Command Request Register while the IF1 Registers are locked. If the CPU has reset the Busy bit, a possible retransmission in case of lost arbitration or in case of an error is disabled.

The IF2 Registers are used as Receive Buffer. After the reception of a message the contents of the shift register is stored into the IF2 Registers, without any acceptance filtering.

Additionally, the actual contents of the shift register can be monitored during the message transfer. Each time a read Message Object is initiated by writing the `Busy` bit of the IF2 Command Request Register to ‘1’, the contents of the shift register is stored into the IF2 Registers.

In Basic Mode the evaluation of all Message Object related control and status bits and of the control bits of the IFx Command Mask Registers is turned off. The message number of the Command request registers is not evaluated. The `NewDat` and `MsgLst` bits of the IF2 Message Control Register retain their function, `DLC3-0` will show the received DLC, the other control bits will be read as ‘0’.

In Basic Mode the ready output `can_walk_b` is not active.
2.3.6.5 Software control of Pin can_tx

Four output functions are available for the CAN transmit pin can_tx. Additionally to its default function – the serial data output – it can drive the CAN Sample Point signal to monitor CAN_Core’s bit timing and it can drive constant dominant or recessive values. The last two functions, combined with the readable CAN receive pin can_rx, can be used to check the CAN bus’ physical layer.

The output mode of pin can_tx is selected by programming the Test Register bits Tx1 and Tx0 as described in section 3.2.5 on page 22.

The three test functions for pin can_tx interfere with all CAN protocol functions. can_tx must be left in its default function when CAN message transfer or any of the test modes Loop Back Mode, Silent Mode, or Basic Mode are selected.
## 3. Programmer's Model

The C_CAN FD8 module allocates an address space of 256 bytes. The registers are organized as 16-bit registers, with the high byte at the odd address and the low byte at the even address.

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Reset Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN Base + 0x00</td>
<td>CAN Control Register</td>
<td>0x0001</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x02</td>
<td>Status Register</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x04</td>
<td>Error Counter</td>
<td>0x0000</td>
<td>read only</td>
</tr>
<tr>
<td>CAN Base + 0x06</td>
<td>Bit Timing Register</td>
<td>0x2301</td>
<td>write enabled by CCE</td>
</tr>
<tr>
<td>CAN Base + 0x08</td>
<td>Interrupt Register</td>
<td>0x0000</td>
<td>read only</td>
</tr>
<tr>
<td>CAN Base + 0x0A</td>
<td>Test Register</td>
<td>0x00 &amp; 0bu00000000 1)</td>
<td>write enabled by Test</td>
</tr>
<tr>
<td>CAN Base + 0x0C</td>
<td>BRP Extension Register</td>
<td>0x0000</td>
<td>write enabled by CCE</td>
</tr>
<tr>
<td>CAN Base + 0x0E</td>
<td>— reserved</td>
<td>—3)</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x10</td>
<td>IF1 Command Request</td>
<td>0x0001</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x12</td>
<td>IF1 Command Mask</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x14</td>
<td>IF1 Mask 1</td>
<td>0xFFFF</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x16</td>
<td>IF1 Mask 2</td>
<td>0xFFFF</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x18</td>
<td>IF1 Arbitration 1</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x1A</td>
<td>IF1 Arbitration2</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x1C</td>
<td>IF1 Message Control</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x1E</td>
<td>IF1 Data A 1</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x20</td>
<td>IF1 Data A 2</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x22</td>
<td>IF1 Data B 1</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x24</td>
<td>IF1 Data B 2</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x26</td>
<td>FD Control Register</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x28</td>
<td>Nominal Bit Timing Register 1</td>
<td>0x0000</td>
<td>write enabled by CCE</td>
</tr>
<tr>
<td>CAN Base + 0x2A</td>
<td>Nominal Bit Timing Register 2</td>
<td>0x0000</td>
<td>write enabled by CCE</td>
</tr>
<tr>
<td>CAN Base + 0x2C</td>
<td>Data Bit Timing Register 1</td>
<td>0x0000</td>
<td>write enabled by CCE</td>
</tr>
<tr>
<td>CAN Base + 0x2E</td>
<td>Data Bit Timing Register 2</td>
<td>0x0000</td>
<td>write enabled by CCE</td>
</tr>
<tr>
<td>CAN Base + 0x30</td>
<td>Transmitter Delay Compens.</td>
<td>0x0000</td>
<td>write enabled by CCE</td>
</tr>
<tr>
<td>CAN Base + 0x32</td>
<td>SSP Position Register</td>
<td>0x0000</td>
<td>read only</td>
</tr>
<tr>
<td>CAN Base + 0x34</td>
<td>Error Logging Register</td>
<td>0x0000</td>
<td>read only</td>
</tr>
<tr>
<td>CAN Base + 0x36</td>
<td>FD Status Register</td>
<td>0x0700</td>
<td>read only</td>
</tr>
<tr>
<td>CAN Base + 0x38</td>
<td>Core Release Low</td>
<td>release ID</td>
<td>read only</td>
</tr>
<tr>
<td>CAN Base + 0x3A</td>
<td>Core Release High</td>
<td>release ID</td>
<td>read only</td>
</tr>
<tr>
<td>CAN Base + 0x3C-0x3E</td>
<td>— reserved</td>
<td>—3)</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x40-0x54</td>
<td>IF2 Registers</td>
<td>see note 2)</td>
<td>same as IF1 Registers</td>
</tr>
<tr>
<td>CAN Base + 0x56-0x7E</td>
<td>— reserved</td>
<td>—3)</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x80</td>
<td>Transmission Request 1</td>
<td>0x0000</td>
<td>read only</td>
</tr>
<tr>
<td>CAN Base + 0x82</td>
<td>Transmission Request 2</td>
<td>0x0000</td>
<td>read only</td>
</tr>
<tr>
<td>CAN Base + 0x84-0x8E</td>
<td>— reserved</td>
<td>—3)</td>
<td></td>
</tr>
<tr>
<td>CAN Base + 0x90</td>
<td>New Data 1</td>
<td>0x0000</td>
<td>read only</td>
</tr>
<tr>
<td>CAN Base + 0x92</td>
<td>New Data 2</td>
<td>0x0000</td>
<td>read only</td>
</tr>
</tbody>
</table>

1) u signifies the actual value of the can_rx pin.
2) The two sets of Message Interface Registers - IF1 and IF2 - have identical functions.
3) Reserved bits are read as ‘0’ except for IFx Mask 2 Register where they are read as ‘1’
The two sets of interface registers (IF1 and IF2) control the CPU access to the Message RAM. They buffer the data to be transferred to and from the RAM, avoiding conflicts between CPU accesses and message reception/transmission.

The C_CAN FD8 operates in a single clock domain, its system clock period is the minimum time quantum mtq for CAN communication.

### 3.1 Hardware Reset Description

After hardware reset, the registers of the C_CAN FD8 hold the values described in figure 5. Additionally the busoff state is reset and the output can_tx is set to recessive (HIGH). The value 0x0001 (Init = ‘1’) in the CAN Control Register enables the software initialisation. The C_CAN FD8 does not influence the CAN bus until the CPU resets Init to ‘0’.

The data stored in the Message RAM is not affected by a hardware reset. After power-on, the contents of the Message RAM is undefined.

#### 3.1.1 Coding of Register Bit access / reset

The coding shown in figure 6 is used with the C_CAN FD8 register descriptions.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r-&lt;reset value&gt;</td>
<td>read-only</td>
</tr>
<tr>
<td>rw-&lt;reset value&gt;</td>
<td>read / write</td>
</tr>
<tr>
<td>rp-&lt;reset value&gt;</td>
<td>read / protected write</td>
</tr>
<tr>
<td>x-&lt;reset value&gt;</td>
<td>reset-on-read</td>
</tr>
<tr>
<td>s-&lt;reset value&gt;</td>
<td>set-on-read</td>
</tr>
</tbody>
</table>

1) u signifies the actual value of the can_rx pin.
2) The two sets of Message Interface Registers - IF1 and IF2 - have identical functions.
3) Reserved bits are read as ‘0’ except for IFx Mask 2 Register where they are read as ‘1’
3.2 CAN Protocol Related Registers

These registers are related to the CAN protocol controller in the CAN Core. They control the operating modes and the configuration of the CAN bit timing and provide status information.

3.2.1 CAN Control Register (addresses 0x01 & 0x00)

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>Test</td>
<td>CCE</td>
<td>DAR</td>
<td>res</td>
<td>EIE</td>
<td>SIE</td>
<td>IE</td>
<td>Init</td>
</tr>
</tbody>
</table>

- **Test**: Test Mode Enable
  - **one**: Test Mode.
  - **zero**: Normal Operation.

- **CCE**: Configuration Change Enable
  - **one**: While Init = ‘1’, the CPU has write access to protected register bits (rp).
  - **zero**: The CPU has no write access to protected register bits (rp).

- **DAR**: Disable Automatic Retransmission
  - **one**: Automatic Retransmission disabled.
  - **zero**: Automatic Retransmission of disturbed messages enabled.

- **EIE**: Error Interrupt Enable
  - **one**: Enabled - A change of bits **BOff** or **EWarn** in the Status Register will cause the Interrupt Register to be set to Status Interrupt (0x8000).
  - **zero**: Disabled - No Error Status Interrupt will be generated.

- **SIE**: Status Change Interrupt Enable
  - **one**: Enabled - The Interrupt Register will be set to Status Interrupt (0x8000) when the C_CAN FD8 sets **LEC** to a value ≠ 7.
  - **zero**: Disabled - No Status Change Interrupt will be generated.

- **IE**: Module Interrupt Enable
  - **one**: Enabled - When the Interrupt Register is ≠ zero, the interrupt line **can_int** is set to active. **can_int** remains active until all interrupts are processed (Interrupt Register returns to zero).
  - **zero**: Disabled - Module Interrupt **can_int** is always inactive.

- **Init**: Initialization
  - **one**: Initialization is started.
  - **zero**: Normal Operation.

**Note**: The busoff recovery sequence (see CAN Specification Rev. 2.0) cannot be shortened by setting or resetting Init. If the device goes busoff, it will set Init of its own accord, stopping all bus activities. Once Init has been cleared by the CPU, the device will then wait for 129 occurrences of Bus Idle (129 • 11 consecutive recessive bits) before resuming normal operations. At the end of the busoff recovery sequence, the Error Management Counters will be reset.

During the waiting time after the resetting of Init, each time a sequence of 11 recessive bits has been monitored, a Bit0Error code is written to the Status Register, enabling the CPU to readily check up whether the CAN bus is stuck at dominant or continuously disturbed and to monitor the proceeding of the busoff recovery sequence.
3.2.2 Status Register (addresses 0x03 & 0x02)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>res</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>res</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>res</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>res</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>res</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>res</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>res</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>res</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>res</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>res</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>BOff</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>EWarn</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>EPass</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>RxOk</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>TxOk</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>LEC</td>
<td>0</td>
</tr>
</tbody>
</table>

**BOff**  
- **one**: The CAN module is in busoff state.  
- **zero**: The CAN module is not busoff.

**EWarn**  
- **one**: At least one of the error counters in the EML has reached the error warning limit of 96.  
- **zero**: Both error counters are below the error warning limit of 96.

**EPass**  
- **one**: The CAN Core is **error passive** as defined in the CAN Specification.  
- **zero**: The CAN Core is **error active**.

**RxOk**  
- **one**: Since this bit was last reset (to zero) by the CPU, a message has been successfully received (independent of the result of acceptance filtering).  
- **zero**: Since this bit was last reset by the CPU, no message has been successfully received. This bit is never reset by the CAN Core.

**TxOk**  
- **one**: Since this bit was last reset by the CPU, a message has been successfully (error free and acknowledged by at least one other node) transmitted.  
- **zero**: Since this bit was reset by the CPU, no message has been successfully transmitted. This bit is never reset by the CAN Core.

**LEC**  
- **0**: **No Error**: Message successfully transmitted or received.  
- **1**: **Stuff Error**: More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.  
- **2**: **Form Error**: Fixed format part of a received frame has the wrong format.  
- **3**: **AckError**: The message this CAN Core transmitted was not acknowledged by another node.  
- **4**: **Bit1Error**: During the transmission of a message (with the exception of the arbitration field), the device wanted to send a **recessive** level (bit of logical value ‘1’), but the monitored bus value was **dominant**.  
- **5**: **Bit0Error**: During the transmission of a message (or acknowledge bit, or active error flag, or overload flag), the device wanted to send a **dominant** level (data or identifier bit logical value ‘0’), but the monitored Bus value was **recessive**. During busoff recovery this status is set each time a sequence of 11 **recessive** bits has been monitored. This enables the CPU to monitor the proceeding of the busoff recovery sequence (indicating the bus is not stuck at **dominant** or continuously disturbed).  
- **6**: **CRCError**: The CRC check sum was incorrect in the message received, the CRC received for an incoming message does not match with the calculated CRC for the received data.  
- **7**: **NoChange**: When the LEC shows the value ‘7’, no CAN bus event was detected since the CPU wrote this value to the LEC.
**Note**: The LEC field holds a code which indicates the type of the last error to occur on the CAN bus. This field will be cleared to ‘0’ when a message has been transferred (reception or transmission) without error. The code ‘7’ may be written by the CPU to check for updates.

### 3.2.2.1 Status Interrupts

A Status Interrupt is generated by bits BOff and EWarn (Error Interrupt) or by RxOk, TxOk, and LEC (Status Change Interrupt) assumed that the corresponding enable bits in the CAN Control Register are set. A change of bit EPass or a write to RxOk, TxOk, or LEC will never generate a Status Interrupt.

Reading the Status Register will clear the Status Interrupt value (8000h) in the Interrupt Register, if it is pending.

### 3.2.3 Error Counter (addresses 0x05 & 0x04)

<table>
<thead>
<tr>
<th>RP</th>
<th>REC</th>
<th>TEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>r-0</td>
<td>r-000 0000</td>
<td>r-0000 0000</td>
</tr>
</tbody>
</table>

RP  
Receive Error Passive
- **one**: The Receive Error Counter has reached the *error passive* level as defined in the CAN Specification.
- **zero**: The Receive Error Counter is below the *error passive* level.

REC  
Receive Error Counter
Actual state of the Receive Error Counter. Values between 0 and 127.

TEC  
Transmit Error Counter
Actual state of the Transmit Error Counter. Values between 0 and 255.

### 3.2.4 Bit Timing Register (addresses 0x07 & 0x06)

This register configures the bit timing for Classic CAN operation (bit FDOE in the FD Control Register is not set).

<table>
<thead>
<tr>
<th>res</th>
<th>TSeg2</th>
<th>TSeg1</th>
<th>SJW</th>
<th>BRP</th>
</tr>
</thead>
<tbody>
<tr>
<td>r-0</td>
<td>rp-010</td>
<td>rp-0011</td>
<td>rp-00</td>
<td>rp-00 0001</td>
</tr>
</tbody>
</table>

- **TSeg2**: The time segment after the sample point
  - **0x0-0x7**: valid values for **TSeg2** are [ 0 … 7 ]. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

- **TSeg1**: The time segment before the sample point
  - **0x01-0x0F**: valid values for **TSeg1** are [ 1 … 15 ]. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

- **SJW**: (Re)Synchronisation Jump Width
  - **0x0-0x3**: Valid programmed values are 0-3. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.
BRP | Baud Rate Prescaler (base value)
---|---
0x01-0x3F | The value by which the system clock frequency is divided for generating the bit time quanta. The bit time is built up from a multiple of this quanta. Valid values for BRP are [ 0 … 63 ]. The actual interpretation by the hardware of this value is such that one more than the programmed value is used. The time quantum tq is (BRPE • 0x40 + BRP + 1) • mtq.

*Note*: With a module clock can_clk of 8 MHz, the reset value of 0x2301 configures the C_CAN FD8 for a bit rate of 500 kBit/s. The registers are only writable if bits CCE and Init in the CAN Control Register are set.

### 3.2.5 Test Register (addresses 0x0B & 0x0A)

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>Rx</td>
<td>Tx1</td>
<td>Tx0</td>
<td>LBack</td>
<td>Silent</td>
<td>Basic</td>
<td>res</td>
<td>res</td>
</tr>
<tr>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-u</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
</tr>
</tbody>
</table>

**Rx** | Monitors the actual value of the can_rx Pin

- *one* | The CAN bus is recessive (can_rx = ‘1’).
- *zero* | The CAN bus is dominant (can_rx = ‘0’).

**Tx1-0** | Control of can_tx pin

- *00* | Reset value, can_tx is controlled by the CAN Core.
- *01* | Sample Point can be monitored at can_tx pin.
- *10* | can_tx pin drives a dominant (‘0’) value.
- *11* | can_tx pin drives a recessive (‘1’) value.

**LBack** | Loop Back Mode

- *one* | Loop Back Mode is enabled.
- *zero* | Loop Back Mode is disabled.

**Silent** | Silent Mode

- *one* | The module is in Silent Mode
- *zero* | Normal operation.

**Basic** | Basic Mode

- *one* | IF1 Registers used as Tx Buffer, IF2 Registers used as Rx Buffer.
- *zero* | Basic Mode disabled.

*Note*: Write access to the Test Register is enabled by setting bit Test in the CAN Control Register. The different test functions may be combined, but Tx1-0 ≠ “00” disturbs message transfer.
3.2.6 BRP Extension Register (addresses 0x0D & 0x0C)

This is register configures the BRP extension for Classic CAN operation when bit \textit{FDOE} in the FD Control Register is not set.

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>BRPE</td>
</tr>
<tr>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>rp-0000</td>
<td></td>
</tr>
</tbody>
</table>

\textbf{BRPE} Baud Rate Prescaler Extension

0x00-0x0F By programming \textbf{BRPE} the Baud Rate Prescaler can be extended to values up to 1023. The actual interpretation by the hardware is that one more than the value programmed by \textbf{BRPE} (MSBs) and \textbf{BRP} (LSBs) is used.

3.2.7 FD Control Register (addresses 0x27 & 0x26)

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>NISO</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>DSIE</td>
<td>BRSR</td>
<td>PXE</td>
<td>PXHD</td>
<td>EFB</td>
<td>REOM</td>
<td>TDCNM</td>
<td>TDCE</td>
<td>BRSE</td>
<td>FDOE</td>
</tr>
<tr>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>x-0</td>
<td>rp-0</td>
<td>rp-0</td>
<td>rp-0</td>
<td>rp-0</td>
<td>rp-0</td>
<td>rp-0</td>
<td>rp-0</td>
</tr>
</tbody>
</table>

\textbf{NISO} If this bit is set, the C\_CAN FD8 uses the CAN FD frame format as specified by the Bosch CAN FD Specification V1.0.

\textit{one} CAN FD frame format according to Bosch CAN FD Specification V1.0
\textit{zero} CAN FD frame format according to ISO11898-1

\textbf{Note} : When the generic parameter \textit{iso\_only\_g} is set to ‘1’ in hardware synthesis, this bit becomes reserved and is read as ‘0’. The C\_CAN FD8 then always operates with the CAN FD frame format according to ISO11898-1.

\textbf{DSIE} Data Phase Status Change Interrupt Enable

\textit{one} Enabled - The Interrupt Register will be set to Status Interrupt (0x8000) when the C\_CAN FD8 sets \textbf{DLEC} to a value $\neq 7$
\textit{zero} Disabled - No Data Phase Status Change Interrupt generated when \textbf{DLEC} in the FD Status Register is updated

\textbf{BRSR} Bit Rate Switch Request

\textit{one} Request CAN FD transmissions with bit rate switching
\textit{zero} Request CAN FD transmissions without bit rate switching

\textbf{PXE} Protocol Exception Event

\textit{one} Protocol exception event occurred
\textit{zero} No protocol exception event occurred since last read access

\textbf{PXHD} Protocol Exception Handling Disable

\textit{one} Protocol exception handling disabled
\textit{zero} Protocol exception handling enabled

\textbf{Note} : When protocol exception handling is disabled, the C\_CAN FD8 will transmit an error frame when it detects a protocol exception condition.

\textbf{EFBI} Edge Filtering during Bus Integration

\textit{one} Two consecutive dominant tq required to detect an edge for hard sync
\textit{zero} Edge filtering disabled
REOM  Restricted Operation Mode
  *one*  Restricted Operation Mode active (see section 2.3.4)
  *zero*  Normal CAN operation

TDCNM  Transmitter Delay Compensation No Measurement
  *one*  The transmitter delay is given by TDCF (without delay measurement)
  *zero*  TDCO defines SSP position relative to delay measurement.

TDCE  Transmitter Delay Compensation Enable
  *one*  Transmitter Delay Compensation is enabled.
  *zero*  Transmitter Delay Compensation is disabled.

BRSE  Bit Rate Switch Enable
  *one*  Bit rate switching for transmissions enabled
  *zero*  Bit rate switching for transmissions disabled

FDOE  FD Operation Enable
  *one*  FD Operation enabled
  *zero*  FD Operation disabled, C_CAN compatible operation

*Note*: When FDOE is not set, the C_CAN FD8 operates exactly as previous versions of the C_CAN, the CAN FD format functions are disabled.

*Note*: When a Classic CAN frame is transmitted while FDOE is set, the nominal bit timing as configured by the Nominal Bit Timing Registers is used.

### 3.2.8 Nominal Bit Timing Register 1 (addresses 0x29 & 0x28)

When bit FDOE in the CAN Control Register is set, the Nominal Bit Timing Registers are used to configure the bit timing in the Arbitration Phase of a CAN FD frame (4 to 385 tq).

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>NTSeg2</td>
<td>NTSeg1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r-0</td>
<td>rp-000 0000</td>
<td>rp-0000 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NTSeg2  The time segment after the sample point
  *0x00-0x7F* valid values are 0-127. The actual interpretation by the hardware of this value is such that one more than the programmed value is used.

NTSeg1  The time segment before the sample point
  *0x01-0xFF* valid values are 1-255. The actual interpretation by the hardware of this value is such that one more than the programmed value is used.

### 3.2.9 Nominal Bit Timing Register 2 (addresses 0x2B & 0x2A)

When bit FDOE in the CAN Control Register is set, the Nominal Bit Timing Registers are used to configure the bit timing in the Arbitration Phase of a CAN FD frame (4 to 385 tq).

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>NSJW</td>
<td>NBRP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r-0</td>
<td>rp-000 0000</td>
<td>rp-0000 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NSJW  (Re)Synchronisation Jump Width Arbitration Phase
  *0x00-0x7F* valid values are 0-127. The actual interpretation by the hardware of this value is such that one more than the programmed value is used.
NBRP  Baud Rate Prescaler Arbitration Phase
0x00-0xFF  The nominal bit time is a multiple of the nominal time quanta. One
nominal time quantum is \((\text{NBRP} + 1) \times \text{mtq}\). The range of NBRP is 0
to 255.

3.2.10 Data Bit Timing Register 1 (addresses 0x2D & 0x2C)
When bit FDOE in the CAN Control Register is set, the Data Bit Timing Registers are used to
configure the bit timing in the Data Phase of a CAN FD frame (4 to 49 tq).

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>DTSeg2</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>DTSeg1</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>DTSeg1</td>
<td>res</td>
<td>res</td>
<td>res</td>
</tr>
<tr>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>rp-0000</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>rp-0000</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>rp-0000</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
</tr>
</tbody>
</table>

DTSeg2  The time segment after the sample point
0x0-0xF  Valid values for DTSeg2 are [ 0 ... 15 ]. The actual interpretation by
the hardware of this value is such that one more than the value pro-
gammed here is used.

DTSeg1  The time segment before the sample point
0x00-0x1F  Valid values for DTSeg1 are [ 0 ... 31 ]. The actual interpretation by
the hardware of this value is such that one more than the value pro-
gammed here is used.

3.2.11 Data Bit Timing Register 2 (addresses 0x2F & 0x2E)
When bit FDOE in the CAN Control Register is set, the Data Bit Timing Registers are used to
configure the bit timing in the Data Phase of a CAN FD frame (4 to 49 tq).

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>DSJW</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>DBRP</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>DBRP</td>
<td>res</td>
<td>res</td>
<td>res</td>
</tr>
<tr>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>rp-0000</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>rp-0000</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>rp-0000</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
</tr>
</tbody>
</table>

DSJW  (Re)Synchronisation Jump Width
0x0-0xF  Valid programmed values are 0-15. The actual interpretation by the
hardware of this value is such that one more than the value pro-
gammed here is used.

DBRP  Baud Rate Prescaler Data Phase
0x00-0x1F  The data bit time is a multiple of the data time quanta. One data time
quantum is \((\text{DBRP} + 1) \times \text{mtq}\). The range of DBRP is 0 to 31.

3.2.12 Transmitter Delay Compensation Register (addresses 0x31 & 0x30)

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>TDCO</td>
<td>res</td>
<td>TDCF</td>
<td>res</td>
<td>TDCF</td>
<td>res</td>
<td>TDCF</td>
<td>res</td>
<td>TDCF</td>
<td>res</td>
<td>TDCF</td>
<td>res</td>
<td>TDCF</td>
<td>res</td>
<td>TDCF</td>
</tr>
<tr>
<td>r-0</td>
<td>rp-0000 0000</td>
<td>r-0</td>
<td>rp-0000 0000</td>
<td>r-0</td>
<td>rp-0000 0000</td>
<td>r-0</td>
<td>rp-0000 0000</td>
<td>r-0</td>
<td>rp-0000 0000</td>
<td>r-0</td>
<td>rp-0000 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TDCO  Transmitter Delay Compensation Offset
0x00-0x7F  Offset value defining the distance between the measured or config-
ured delay from \text{can\_tx} to \text{can\_rx} and the SSP. The value is given in
mtq.
**TDCF** Transmitter Delay Compensation Filter Window Length

0x00-0x7F Defines the minimum value for the SSP position, dominant edges on can_rx that would result in an earlier SSP position are ignored for Transmitter Delay measurement. The feature is enabled when TDCF is configured to a value greater than TDCO. When transmitter delay compensation with fixed SSP position is selected (TDCNM = ‘1’), the transmitter delay value has to be configured here. Then the sum of TDCF and TDCO may not be greater than 0x7F, it gives the SSP position SSPP. In case the sum of TDCF and TDCO exceeds the specified limit of 0x7F, the sum will be limited to 7 bit, eliminating the MSB.

The value is given in mtq.

### 3.2.13 SSP Position Register (addresses 0x33 & 0x32)

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>SSPP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SSPP** SSP Position

0x00-0x7F Position of the secondary sample point, defined by the sum of the measured (and optionally filtered) delay from can_tx to can_rx and TDCO. When transmitter delay compensation with fixed SSP position is selected, the fixed value is shown here. SSPP is, in the data phase, the number of mtq between the start of the transmitted bit and the secondary sample point.

### 3.2.14 Error Logging Register (addresses 0x35 & 0x34)

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CELD</td>
<td>CELN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CELD** CAN Error Logging (Data Phase)

0x00-0xFF The counter is incremented each time when a CAN protocol error in the Data Phase of a CAN FD frame causes the Transmit Error Counter TEC or the Receive Error Counter REC to be incremented. It is reset by read access to CELD. The counter stops at 0xFF.

**CELN** CAN Error Logging (Arbitration Phase)

0x00-0xFF The counter is incremented each time when a CAN protocol error in the Arbitration Phase of a CAN FD frame causes the Transmit Error Counter TEC or the Receive Error Counter REC to be incremented. It is reset by read access to CELN. The counter stops at 0xFF.

*Note*: When the C_CAN FD8 is in Restricted Operation Mode (FD Control Register REOM = ‘1’), the receive and transmit error counters are not incremented when a CAN protocol error is detected, but CELD and CELN are still incremented.
3.2.15 FD Status Register (addresses 0x37 & 0x36)

When the C_CAN FD8 is operated in CAN FD mode (bit FDOE in the FD Control Register set) this register is to be used instead of the Status Register (CAN Base + 0x02). Bits 7 downto 0 are mirrored from the Status Register. Any modifications done in one register is also seen in the other register.

In case CAN FD operation is disabled (FDOE = ‘0’), reading the register will not modify the register contents (modification-on-read disabled) to assure software compatibility to existing C_CAN implementations. The activity state Act can be accessed in all operation modes.

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Act</td>
<td>RxFDF</td>
<td>RxBRS</td>
<td>RxESI</td>
<td>DLEC</td>
<td>BOff</td>
<td>EWarn</td>
<td>EPass</td>
<td>RxOk</td>
<td>TxOk</td>
<td>LEC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r-00</td>
<td>x-0</td>
<td>x-0</td>
<td>x-0</td>
<td>s-111</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>x-0</td>
<td>x-0</td>
<td>s-000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Act  
Activity (operation state)  
0  **Integrating**: Waiting for 11 recessive bits after reset or in busoff recovery  
1  **Idle**: Ready for Start of Frame  
2  **Receiver**: Node is receiving a frame  
3  **Transmitter**: Node is transmitting a frame  

**Note**: ACT is set to “00” after power-on, by hardware reset, when the node entered busoff state, or by a Protocol Exception Event.

RxFDF  
Message in CAN FD format (FDF = ‘1’) received  
**one**  Since this bit was reset by reading the register, a message in CAN FD format has been received (independent of the result of acceptance filtering). This bit is set together with **RxOk**.  
**zero**  Since this bit was reset by reading the register, no message in CAN FD format has been received. This bit is never reset by the CAN Core.

RxBRS  
Message in CAN FD format with BRS set received  
**one**  Since this bit was reset by reading the register, a message in CAN FD format has been received (independent of the result of acceptance filtering) which had its BRS flag set. This bit is set together with **RxFDF** and **RxOk**.  
**zero**  Since this bit was reset by reading the register, no message in CAN FD format has been received (independent of the result of acceptance filtering) which had its BRS flag set. This bit is never reset by the CAN Core.

RxESI  
Message in CAN FD format with ESI set received  
**one**  Since this bit was reset by reading the register, a message in CAN FD format has been received (independent of the result of acceptance filtering) which had its ESI flag set. This bit is set together with **RxFDF** and **RxOk**.  
**zero**  Since this bit was reset by reading the register, no message in CAN FD format has been received (independent of the result of acceptance filtering) which had its ESI flag set. This bit is never reset by the CAN Core.

DLEC  
Last Error Code of CAN FD format frame with its BRS flag set  
Same coding as **LEC**. Reading the register will set DLEC to ‘7’ (0b111).

BOff  
Busoff Status  
**one**  The CAN module is in busoff state.  
**zero**  The CAN module is not busoff.
EWarn  Warning Status  
\begin{align*}  
\text{one} & \quad \text{At least one of the error counters in the EML has reached the error warning limit of 96.} \\
\text{zero} & \quad \text{Both error counters are below the error warning limit of 96.} 
\end{align*}

EPass  Error Passive  
\begin{align*}  
\text{one} & \quad \text{The CAN Core is error passive as defined in the CAN Specification.} \\
\text{zero} & \quad \text{The CAN Core is error active.} 
\end{align*}

RxOk  Received a Message Successfully  
\begin{align*}  
\text{one} & \quad \text{Since this bit was reset by reading the register, a message has been successfully received (independent of the result of acceptance filtering).} \\
\text{zero} & \quad \text{Since this bit was reset by reading the register, no message has been successfully received. This bit is never reset by the CAN Core.} 
\end{align*}

TxOk  Transmitted a Message Successfully  
\begin{align*}  
\text{one} & \quad \text{Since this bit was reset by reading the register, a message has been successfully (error free and acknowledged by at least one other node) transmitted.} \\
\text{zero} & \quad \text{Since this bit was reset by reading the register, no message has been successfully transmitted. This bit is never reset by the CAN Core.} 
\end{align*}

LEC  Last Error Code (Type of the last protocol event to occur on the CAN bus)  
\begin{align*}  
0 & \quad \textbf{No Error:} \quad \text{Message successfully transmitted or received.} \\
1 & \quad \textbf{Stuff Error:} \quad \text{More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.} \\
2 & \quad \textbf{Form Error:} \quad \text{Fixed format part of a received frame has the wrong format.} \\
3 & \quad \textbf{AckError:} \quad \text{The message this CAN Core transmitted was not acknowledged by another node.} \\
4 & \quad \textbf{Bit1Error:} \quad \text{During the transmission of a message (with the exception of the arbitration field), the device wanted to send a recessive level (bit of logical value ‘1’), but the monitored bus value was dominant.} \\
5 & \quad \textbf{Bit0Error:} \quad \text{During the transmission of a message (or acknowledge bit, or active error flag, or overload flag), the device wanted to send a dominant level (data or identifier bit logical value ‘0’), but the monitored Bus value was recessive. During busoff recovery this status is set each time a sequence of 11 recessive bits has been monitored. This enables the CPU to monitor the proceeding of the busoff recovery sequence (indicating the bus is not stuck at dominant or continuously disturbed).} \\
6 & \quad \textbf{CRCError:} \quad \text{The CRC check sum was incorrect in the message received, the CRC received for an incoming message does not match with the calculated CRC for the received data.} \\
7 & \quad \textbf{NoChange:} \quad \text{When the LEC shows the value ‘7’, no CAN bus event was detected since the CPU wrote this value to the LEC.} 
\end{align*}

Note: The LEC field holds a code which indicates the type of the last error to occur on the CAN bus. This field will be cleared to ‘0’ when a message has been transferred (reception or transmission) without error. Reading the register will set LEC to ‘7’ (0b111).

Note: When a frame in CAN FD format has reached the Data Phase with BRS flag set, the next CAN event (error or valid frame) will be shown in DLEC instead of LEC. An error in a fixed stuff-bit of a CAN FD CRC Sequence will be shown as a Form Error, not Stuff Error. When BRS flag is not set, there is no Data Phase and the Data Phase status will not be updated.
3.3 Message Interface Register Sets

There are two sets of Interface Registers which are used to control the CPU access to the Message RAM. The Interface Registers avoid conflicts between CPU access to the Message RAM and CAN message reception and transmission by buffering the data to be transferred. A complete Message Object (see chapter 3.3.4) or parts of the Message Object may be transferred between the Message RAM and the IFx Message Buffer registers (see chapter 3.3.3) in one single transfer.

The function of the two interface register sets is identical (except for test mode Basic). They can be used the way that one set of registers is used for data transfer to the Message RAM while the other set of registers is used for the data transfer from the Message RAM, allowing both processes to be interrupted by each other. Figure 7 gives an overview of the two Interface Register sets.

Each set of Interface Registers consists of Message Buffer Registers controlled by their own Command Registers. The Command Mask Register specifies the direction of the data transfer and which parts of a Message Object will be transferred. The Command Request Register is used to select a Message Object in the Message RAM as target or source for the transfer and to start the action specified in the Command Mask Register.

<table>
<thead>
<tr>
<th>Address</th>
<th>IF1 Register Set</th>
<th>Address</th>
<th>IF2 Register Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN Base + 0x10</td>
<td>IF1 Command Request</td>
<td>CAN Base + 0x40</td>
<td>IF2 Command Request</td>
</tr>
<tr>
<td>CAN Base + 0x12</td>
<td>IF1 Command Mask</td>
<td>CAN Base + 0x42</td>
<td>IF2 Command Mask</td>
</tr>
<tr>
<td>CAN Base + 0x14</td>
<td>IF1 Mask 1</td>
<td>CAN Base + 0x44</td>
<td>IF2 Mask 1</td>
</tr>
<tr>
<td>CAN Base + 0x16</td>
<td>IF1 Mask 2</td>
<td>CAN Base + 0x46</td>
<td>IF2 Mask 2</td>
</tr>
<tr>
<td>CAN Base + 0x18</td>
<td>IF1 Arbitration 1</td>
<td>CAN Base + 0x48</td>
<td>IF2 Arbitration 1</td>
</tr>
<tr>
<td>CAN Base + 0x1A</td>
<td>IF1 Arbitration 2</td>
<td>CAN Base + 0x4A</td>
<td>IF2 Arbitration 2</td>
</tr>
<tr>
<td>CAN Base + 0x1C</td>
<td>IF1 Message Control</td>
<td>CAN Base + 0x4C</td>
<td>IF2 Message Control</td>
</tr>
<tr>
<td>CAN Base + 0x1E</td>
<td>IF1 Data A 1</td>
<td>CAN Base + 0x4E</td>
<td>IF2 Data A 1</td>
</tr>
<tr>
<td>CAN Base + 0x20</td>
<td>IF1 Data A 2</td>
<td>CAN Base + 0x50</td>
<td>IF2 Data A 2</td>
</tr>
<tr>
<td>CAN Base + 0x22</td>
<td>IF1 Data B 1</td>
<td>CAN Base + 0x52</td>
<td>IF2 Data B 1</td>
</tr>
<tr>
<td>CAN Base + 0x24</td>
<td>IF1 Data B 2</td>
<td>CAN Base + 0x54</td>
<td>IF2 Data B 2</td>
</tr>
</tbody>
</table>

Figure 7: IF1 and IF2 Message Interface Register Sets

3.3.1 IFx Command Request Registers

A message transfer is started as soon as the CPU has written the message number to the Command Request Register. With this write operation the Busy bit is automatically set to ‘1’ and output can_wait_b is activated to notify the CPU that a transfer is in progress. After a wait time of 3 to 6 can_clk periods, the transfer between the Interface Register and the Message RAM has completed. The Busy bit is set back to zero and can_wait_b is deactivated (see Module Integration Guide).
Busy

Busy Flag

*one* set to one when writing to the IFx Command Request Register

*zero* reset to zero when read/write action has finished.

Message Number

<table>
<thead>
<tr>
<th>Message Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01-0x20</td>
<td>Valid Message Number, the Message Object in the Message RAM is selected for data transfer.</td>
</tr>
<tr>
<td>0x00</td>
<td>Not a valid Message Number, interpreted as 0x20.</td>
</tr>
<tr>
<td>0x21-0x3F</td>
<td>Not a valid Message Number, interpreted as 0x01-0x1F.</td>
</tr>
</tbody>
</table>

*Note*: When a Message Number that is not valid is written into the Command Request Register, the Message Number will be transformed into a valid value and that Message Object will be transferred.

3.3.2 IFx Command Mask Registers

The control bits of the IFx Command Mask Register specify the transfer direction and select which of the IFx Message Buffer Registers are source or target of the data transfer.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>WR/RD</td>
<td>Write / Read</td>
</tr>
<tr>
<td></td>
<td>one</td>
</tr>
<tr>
<td></td>
<td>Write: Transfer data from the selected Message Buffer Registers to the Message Object addressed by the Command Request Register.</td>
</tr>
<tr>
<td></td>
<td>zero</td>
</tr>
<tr>
<td></td>
<td>Read: Transfer data from the Message Object addressed by the Command Request Register into the selected Message Buffer Registers.</td>
</tr>
</tbody>
</table>

The other bits of IFx Command Mask Register have different functions depending on the transfer direction:

3.3.2.1 Direction = Write

**Mask**

Access Mask Bits

*one* transfer **Identifier Mask + MDir + MXtd** to Message Object.

*zero* Mask bits unchanged.

**Arb**

Access Arbitration Bits

*one* transfer **Identifier + Dir + Xtd + MsgVal** to Message Object.

*zero* Arbitration bits unchanged.

**Control**

Access Control Bits

*one* transfer Control Bits to Message Object.

*zero* Control Bits unchanged.

**ClrIntPnd**

Clear Interrupt Pending Bit

*Note*: When writing to a Message Object, this bit is ignored.

**TxRqst/NewDat** Access Transmission Request Bit

*one* set TxRqst bit

*zero* TxRqst bit unchanged
**Note**: If a transmission is requested by programming bit **TxRqst/NewDat** in the IFx Command Mask Register, bit **TxRqst** in the IFx Message Control Register will be ignored.

**Data A**  
Access Data Bytes 0-3  
*one* transfer Data Bytes 0-3 to Message Object.  
*zero* Data Bytes 0-3 unchanged.

**Data B**  
Access Data Bytes 4-7  
*one* transfer Data Bytes 4-7 to Message Object.  
*zero* Data Bytes 4-7 unchanged.
3.3.2.2 Direction = Read

Mask Access Mask Bits
one transfer Identifier Mask + MDir + MXtd to IFx Message Buffer Register.
zero Mask bits unchanged.

Arb Access Arbitration Bits
one transfer Identifier + Dir + Xtd + MsgVal to IFx Message Buffer Register.
zero Arbitration bits unchanged.

Control Access Control Bits
one transfer Control Bits to IFx Message Buffer Register.
zero Control Bits unchanged.

ClrIntPnd Clear Interrupt Pending Bit
one clear IntPnd bit in the Message Object.
zero IntPnd bit remains unchanged.

TxRqst/NewDat Access New Data Bit
one clear NewDat bit in the Message Object.
zero NewDat bit remains unchanged.

Note: A read access to a Message Object can be combined with the reset of the control bits IntPnd and NewDat. The values of these bits transferred to the IFx Message Control Register always reflect the status before resetting these bits.

Data A Access Data Bytes 0-3
one transfer Data Bytes 0-3 to IFx Message Buffer Register.
zero Data Bytes 0-3 unchanged.

Data B Access Data Bytes 4-7
one transfer Data Bytes 4-7 to IFx Message Buffer Register.
zero Data Bytes 4-7 unchanged.
### 3.3.3 IFx Message Buffer Registers

The bits of the Message Buffer registers mirror the Message Objects in the Message RAM. The function of the Message Objects bits is described in chapter 3.3.3.

#### 3.3.3.1 IFx Mask Registers

<table>
<thead>
<tr>
<th>Mask Register</th>
<th>Address</th>
<th>Bits</th>
<th>Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF1 Mask 1 Register</td>
<td>0x15 &amp; 0x14</td>
<td>15-0</td>
<td>rw-1111 1111</td>
</tr>
<tr>
<td>IF1 Mask 2 Register</td>
<td>0x17 &amp; 0x16</td>
<td>MXtd MDir res Msk28-16</td>
<td>rw-1 rw-1 r-1 rw-1 1111</td>
</tr>
<tr>
<td>IF2 Mask 1 Register</td>
<td>0x45 &amp; 0x44</td>
<td>15-0</td>
<td>rw-1111 1111</td>
</tr>
<tr>
<td>IF2 Mask 2 Register</td>
<td>0x47 &amp; 0x46</td>
<td>MXtd MDir res Msk28-16</td>
<td>rw-1 rw-1 r-1 rw-1 1111</td>
</tr>
</tbody>
</table>

#### 3.3.3.2 IFx Arbitration Registers

<table>
<thead>
<tr>
<th>Arbitration Register</th>
<th>Address</th>
<th>Bits</th>
<th>Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF1 Arbitration 1 Register</td>
<td>0x19 &amp; 0x18</td>
<td>15-0</td>
<td>ID15-0</td>
</tr>
<tr>
<td>IF1 Arbitration 2 Register</td>
<td>0x1B &amp; 0x1A</td>
<td>MsgVal Xtd Dir ID28-16</td>
<td>rw-0 rw-0 rw-0 0000</td>
</tr>
<tr>
<td>IF2 Arbitration 1 Register</td>
<td>0x49 &amp; 0x48</td>
<td>15-0</td>
<td>ID15-0</td>
</tr>
<tr>
<td>IF2 Arbitration 2 Register</td>
<td>0x4B &amp; 0x4A</td>
<td>MsgVal Xtd Dir ID28-16</td>
<td>rw-0 rw-0 rw-0 0000</td>
</tr>
</tbody>
</table>
3.3.3.3 IFX Message Control Registers

The behaviour of the IFX Message Control Registers depends on the state of bit FDOE in the FD Control Register.

**FDOE = '0' - C_CAN compatibility mode:**

Bits at position 6 downto 4 not writable, the bits are always read as ‘0’.

### IF1 Message Control Register (addresses 0x1D & 0x1C)

```
<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

### IF2 Message Control Register (addresses 0x4D & 0x4C)

```
<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

**FDOE = ‘1’ - CAN FD operation mode:**

Bits FDF, BRS, and ESI on position 6 downto 4 are read/write.

### IF1 Message Control Register (addresses 0x1D & 0x1C)

```
<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewDat</td>
<td>MsgLst</td>
<td>IntPnd</td>
<td>UMask</td>
<td>TxIE</td>
<td>RxIE</td>
<td>RmtEn</td>
<td>TxRqst</td>
<td>EoB</td>
<td>res</td>
<td>res</td>
<td>res</td>
<td>DLC3-0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>r-0</td>
<td>rw-0000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

### IF2 Message Control Register (addresses 0x4D & 0x4C)

```
<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewDat</td>
<td>MsgLst</td>
<td>IntPnd</td>
<td>UMask</td>
<td>TxIE</td>
<td>RxIE</td>
<td>RmtEn</td>
<td>TxRqst</td>
<td>EoB</td>
<td>FDF</td>
<td>BRS</td>
<td>ESI</td>
<td>DLC3-0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Resetting FDOE will also reset bits FDF, BRS, and ESI. The state of these bits in the Message RAM is not affected.

### 3.3.3.4 IFx Data A and Data B Registers

The data bytes of CAN messages are stored in the IFx Message Buffer Registers in the following order:

```
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| IF1 Message Data A1 (addresses 0x1F & 0x1E) | Data(1) | Data(0) |
| IF1 Message Data A2 (addresses 0x21 & 0x20) | Data(3) | Data(2) |
| IF1 Message Data B1 (addresses 0x23 & 0x22) | Data(5) | Data(4) |
| IF1 Message Data B2 (addresses 0x25 & 0x24) | Data(7) | Data(6) |
| IF2 Message Data A1 (addresses 0x4F & 0x4E) | Data(1) | Data(0) |
| IF2 Message Data A2 (addresses 0x51 & 0x50) | Data(3) | Data(2) |
| IF2 Message Data B1 (addresses 0x53 & 0x52) | Data(5) | Data(4) |
| IF2 Message Data B2 (addresses 0x55 & 0x54) | Data(7) | Data(6) |
```

In a CAN Data Frame, Data(0) is the first, Data(7) is the last byte to be transmitted or received. In CAN’s serial bit stream, the MSB of each byte will be transmitted first.
3.3.4 Message Object in the Message Memory

There are 32 Message Objects in the Message RAM. To avoid conflicts between CPU access to the Message RAM and CAN message reception and transmission, the CPU cannot directly access the Message Objects, these accesses are handled via the IFx Interface Registers.

Figure 8 gives an overview of the structure of a Message Object. Bits **TxRqst**, **NewDat**, **IntPnd**, and **MsgVal** are implemented as FFs (see section 3.4.2 to section 3.4.5) and are therefore not located in the Message RAM.

<table>
<thead>
<tr>
<th>Message Object</th>
<th>Msk28-0</th>
<th>MXtd</th>
<th>MDir</th>
<th>UMask</th>
<th>TxIE</th>
<th>RxIE</th>
<th>RmtEn</th>
<th>EoB</th>
<th>FDF</th>
<th>BRS</th>
<th>ESI</th>
<th>DLC3-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID28-0</td>
<td>Xtd</td>
<td>Dir</td>
<td>MsgLst</td>
<td>Data 0</td>
<td>Data 1</td>
<td>Data 2</td>
<td>Data 3</td>
<td>Data 4</td>
<td>Data 5</td>
<td>Data 6</td>
<td>Data 7</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8: Structure of a Message Object in the Message RAM

**Msk28-0**  
- **one** The corresponding identifier bit is used for acceptance filtering.
- **zero** The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering.

**ID28-0**  
- **Message Identifier**
  - **ID28** - **ID0** 29-bit Identifier (“Extended Frame”).
  - **ID28** - **ID18** 11-bit Identifier (“Standard Frame”).

**MXtd**  
- **Mask Extended Identifier**
  - **one** The extended identifier bit (IDE) is used for acceptance filtering.
  - **zero** The extended identifier bit (IDE) has no effect on the acceptance filtering.

**Note**: When 11-bit (“standard”) Identifiers are used for a Message Object, the identifiers of received Data Frames are written into bits **ID28** to **ID18**. For acceptance filtering, only these bits together with mask bits **Msk28** to **Msk18** are considered.

**Xtd**  
- **Extended Identifier**
  - **one** The 29-bit (“extended”) Identifier will be used for this Message Object.
  - **zero** The 11-bit (“standard”) Identifier will be used for this Message Object.

**MDir**  
- **Mask Message Direction**
  - **one** The message direction bit (**Dir**) is used for acceptance filtering.
  - **zero** The message direction bit (**Dir**) has no effect on the acceptance filtering.

**Note**: The Arbitration Registers **ID28-0**, **Xtd**, and **Dir** are used to define the identifier and type of outgoing messages and are used (together with the mask registers **Msk28-0**, **MXtd**, and **MDir**) for acceptance filtering of incoming messages. A received message is stored into the valid Message Object with matching identifier and Direction=**receive** (Data Frame) or Direction=**transmit** (Remote Frame). Extended frames can be stored only in Message Objects with **Xtd** = **one**, standard frames in Message Objects with **Xtd** = **zero**. If a received message (Data Frame or Remote Frame) matches with more than one valid Message Object, it is stored into that with the lowest message number. For details see chapter 4.2.3 Acceptance Filtering of Received Messages.
Dir \hspace{2em} Message Direction
\begin{itemize}
  \item \textit{one} \hspace{2em} Direction = \textit{transmit}: On \textit{TxRqst}, the respective Message Object is transmitted as a Data Frame. On reception of a Remote Frame with matching identifier, the \textit{TxRqst} bit of this Message Object is set (if \textit{RmtEn} = \textit{one}).
  \item \textit{zero} \hspace{2em} Direction = \textit{receive}: On \textit{TxRqst}, a Remote Frame with the identifier of this Message Object is transmitted. On reception of a Data Frame with matching identifier, that message is stored in this Message Object.
\end{itemize}

UMask \hspace{2em} Use Acceptance Mask
\begin{itemize}
  \item \textit{one} \hspace{2em} Use Mask (Msk28-0, MXtd, and MDir) for acceptance filtering
  \item \textit{zero} \hspace{2em} Mask ignored.
\end{itemize}

\textbf{Note}: If the \textit{UMask} bit is set to \textit{one}, the Message Object’s mask bits have to be programmed during initialization of the Message Object before \textit{MsgVal} is set to \textit{one}.

MsgLst \hspace{2em} Message Lost (only valid for Message Objects with direction = \textit{receive})
\begin{itemize}
  \item \textit{one} \hspace{2em} The Message Handler stored a new message into this object when \textit{NewDat} was still set, the CPU has lost a message.
  \item \textit{zero} \hspace{2em} No message lost since last time this bit was reset by the CPU.
\end{itemize}

TxIE \hspace{2em} Transmit Interrupt Enable
\begin{itemize}
  \item \textit{one} \hspace{2em} \textit{IntPnd} will be set after a successful transmission of a frame.
  \item \textit{zero} \hspace{2em} \textit{IntPnd} will be left unchanged after the successful transmission of a frame.
\end{itemize}

RxIE \hspace{2em} Receive Interrupt Enable
\begin{itemize}
  \item \textit{one} \hspace{2em} \textit{IntPnd} will be set after a successful reception of a frame.
  \item \textit{zero} \hspace{2em} \textit{IntPnd} will be left unchanged after a successful reception of a frame.
\end{itemize}

RmtEn \hspace{2em} Remote Enable
\begin{itemize}
  \item \textit{one} \hspace{2em} At the reception of a Remote Frame, \textit{TxRqst} is set.
  \item \textit{zero} \hspace{2em} At the reception of a Remote Frame, \textit{TxRqst} is left unchanged.
\end{itemize}

EoB \hspace{2em} End of Buffer
\begin{itemize}
  \item \textit{one} \hspace{2em} Single Message Object or last Message Object of a FIFO Buffer.
  \item \textit{zero} \hspace{2em} Message Object belongs to a FIFO Buffer and is not the last Message Object of that FIFO Buffer.
\end{itemize}

\textbf{Note}: This bit is used to concatenate two or more Message Objects (up to 32) to build a FIFO Buffer. \textbf{For single Message Objects (not belonging to a FIFO Buffer) this bit must always be set to one.} For details on the concatenation of Message Objects see chapter 4.7.

FDF \hspace{2em} FD Format
\begin{itemize}
  \item \textit{one} \hspace{2em} Message in CAN FD format.
  \item \textit{zero} \hspace{2em} Message in Classic CAN format.
\end{itemize}

BRS \hspace{2em} Bit Rate Switch
\begin{itemize}
  \item \textit{one} \hspace{2em} Received message with bit rate switching.
  \item \textit{zero} \hspace{2em} Received message without bit rate switching.
\end{itemize}

ESI \hspace{2em} Error State Indicator
\begin{itemize}
  \item \textit{one} \hspace{2em} Message with ESI bit (Error Passive) in CAN FD format.
  \item \textit{zero} \hspace{2em} Message without ESI bit or not in CAN FD format.
**Note:** The bits FDF, BRS, and ESI in the IFx Interface Registers are always updated by transfer from the Message Object (from received messages). When FD operation is enabled (FDOE = '1'), bit FDF decides whether the frame is transmitted in FD format. For message transmission bit BRS is ignored. The ESI bit of the transmit buffer is or'ed with the error passive flag to decide the value of the ESI bit in the transmitted FD frame. As required by the CAN FD protocol specification, an error active node may optionally transmit the ESI bit recessive, but an error passive node will always transmit the ESI bit recessive.

**DLC3-0**  
Data Length Code  
0-8  CAN + CAN FD: frame has 0-8 data bytes  
9-15  CAN: frame has 8 data bytes  
9-15  CAN FD: frame has 12/16/24/32/48/64 data bytes

**Note:** The Data Length Code of a Message Object must be defined the same as in all the corresponding objects with the same identifier at other nodes. When the Message Handler stores a data frame, it will write the DLC to the value given by the received message.

**Data 0**  
1st data byte of a CAN Data Frame

**Data 1**  
2nd data byte of a CAN Data Frame

**Data 2**  
3rd data byte of a CAN Data Frame

**Data 3**  
4th data byte of a CAN Data Frame

**Data 4**  
5th data byte of a CAN Data Frame

**Data 5**  
6th data byte of a CAN Data Frame

**Data 6**  
7th data byte of a CAN Data Frame

**Data 7**  
8th data byte of a CAN Data Frame

**Note:** Byte Data 0 is the first data byte shifted into the shift register of the CAN Core during a reception, byte Data 7 is the last. When the Message Handler stores a Data Frame, it will write all the eight data bytes into a Message Object. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by non specified values.
3.4 Message Handler Registers

All Message Handler registers are read-only. Their contents (**TxRqst**, **NewDat**, **IntPnd**, and **MsgVal** bits of each Message Object and the Interrupt Identifier) is status information provided by the Message Handler FSM.

3.4.1 Interrupt Register (addresses 0x09 & 0x08)

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**IntId15-8**

- **r-0000 0000**

**IntId15-0**

- **Interrupt Identifier (the number here indicates the source of the interrupt)**
  - **0x0000** No interrupt is pending.
  - **0x0001-0x0020** Number of Message Object which caused the interrupt.
  - **0x0021-0x7FFF** unused.
  - **0x8000** Status Interrupt.
  - **0x8001-0xFFFF** unused

If several interrupts are pending, the CAN Interrupt Register will point to the pending interrupt with the highest priority, disregarding their chronological order. An interrupt remains pending until the CPU has cleared it. If **IntId** is different from 0x0000 and **IE** is set, the interrupt line to the CPU, **can_int**, is active. The interrupt line remains active until **IntId** is back to value 0x0000 (the cause of the interrupt is reset) or until **IE** is reset.

The Status Interrupt has the highest priority. Among the message interrupts, the Message Object’s interrupt priority decreases with increasing message number.

A message interrupt is cleared by clearing the Message Object’s **IntPnd** bit. The Status Interrupt is cleared by reading the Status Register resp. the FD Status Register when **FDOE** = ‘1’.

3.4.2 Transmission Request Registers

<table>
<thead>
<tr>
<th>Transmission Request 1 Register</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(addresses 0x81 &amp; 0x80)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TxRqst16-9</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TxRqst8-1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transmission Request 2 Register</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(addresses 0x83 &amp; 0x82)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TxRqst32-25</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TxRqst24-17</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>r-0000 0000</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TxRqst32-1** Transmission Request Bits (of all Message Objects)

- **one** The transmission of this Message Object is requested and is not yet done.
- **zero** This Message Object is not waiting for transmission

These registers hold the **TxRqst** bits of the 32 Message Objects. By reading out the **TxRqst** bits, the CPU can check for which Message Object a Transmission Request is pending. The **TxRqst** bit of a specific Message Object can be set/reset by the CPU via the I/Fx Message Interface Registers or by the Message Handler after reception of a Remote Frame or after a successful transmission.
### 3.4.3 New Data Registers

<table>
<thead>
<tr>
<th>New Data 1 Register (addresses 0x91 &amp; 0x90)</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewDat16-9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NewDat8-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>New Data 2 Register (addresses 0x93 &amp; 0x92)</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewDat32-25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NewDat24-17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r-0000 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r-0000 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NewDat32-1** New Data Bits (of all Message Objects)

- **one** The Message Handler or the CPU has written new data into the data portion of this Message Object
- **zero** No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU

These registers hold the **NewDat** bits of the 32 Message Objects. By reading out the **NewDat** bits, the CPU can check for which Message Object the data portion was updated. The **NewDat** bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers or by the Message Handler after reception of a Data Frame or after a successful transmission.

### 3.4.4 Interrupt Pending Registers

<table>
<thead>
<tr>
<th>Interrupt Pending 1 Register (addresses 0xA1 &amp; 0xA0)</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntPnd16-9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IntPnd8-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interrupt Pending 2 Register (addresses 0xA3 &amp; 0xA2)</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntPnd32-25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IntPnd24-17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r-0000 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r-0000 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**IntPnd32-1** Interrupt Pending Bits (of all Message Objects)

- **one** This message object is the source of an interrupt. The Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority.
- **zero** This message object is not the source of an interrupt

These registers hold the **IntPnd** bits of the 32 Message Objects. By reading out the **IntPnd** bits, the CPU can check for which Message Object an interrupt is pending. The **IntPnd** bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers or by the Message Handler after reception or after a successful transmission of a frame. This will also affect the value of **IntId** in the Interrupt Register.

### 3.4.5 Message Valid Registers

<table>
<thead>
<tr>
<th>Message Valid 1 Register (addresses 0xB1 &amp; 0xB0)</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MsgVal16-9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MsgVal8-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Message Valid 2 Register (addresses 0xB3 &amp; 0xB2)</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MsgVal32-25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MsgVal24-17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r-0000 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r-0000 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MsgVal32-1** Message Valid Bits (of all Message Objects)

- **one** This Message Object is configured and should be considered by the Message Handler
- **zero** This Message Object is ignored by the Message Handler
These registers hold the \textit{MsgVal} bits of the 32 Message Objects. By reading out the \textit{MsgVal} bits, the CPU can check which Message Object is valid. The \textit{MsgVal} bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers.

\textbf{Note} : The CPU must reset the \textit{MsgVal} bit of all unused Messages Objects during the initialization before it resets bit \textit{Init} in the CAN Control Register. This bit must also be reset before the identifier \textit{Id28-0}, the control bits \textit{Xtd}, \textit{Dir}, or the Data Length Code \textit{DLC3-0} are modified, or if the Messages Object is no longer required.

### 3.5 Core Release Registers

The design step of a C\_CAN FD8 implementation can be identified by reading the Core Release Registers Low/High.

<table>
<thead>
<tr>
<th>Release</th>
<th>Step</th>
<th>SubStep</th>
<th>Year</th>
<th>Month</th>
<th>Day</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>12</td>
<td>20</td>
<td>Revision 1.9.5, Date 2013/12/20</td>
</tr>
</tbody>
</table>

Figure 9: Example for Coding of Revisions

#### 3.5.1 Core Release Low (addresses 0x39 & 0x38)

- **MON7-0** Time Stamp Month
  - Two digits, BCD-coded. Configured by constant on C\_CAN FD8 synthesis.
- **DAY7-0** Time Stamp Day
  - Two digits, BCD-coded. Configured by constant on C\_CAN FD8 synthesis.

#### 3.5.2 Core Release High (addresses 0x3B & 0x3A)

<table>
<thead>
<tr>
<th>REL3-0</th>
<th>STEP3-0</th>
<th>SUBSTEP3-0</th>
<th>YEAR3-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
</tbody>
</table>

- **REL3-0** Core Release
  - One digit, BCD-coded.
- **STEP3-0** Step of Core Release
  - One digit, BCD-coded.
- **SUBSTEP3-0** Sub-step of Core Release
  - One digit, BCD-coded.
- **YEAR3-0** Time Stamp Year (2010 + digit)
  - One digit, BCD-coded. Configured by constant on C\_CAN FD8 synthesis.
4. CAN Application

4.1 Management of Message Objects

The configuration of the Message Objects in the Message RAM will (with the exception of the bits \texttt{MsgVal}, \texttt{NewDat}, \texttt{IntPnd}, and \texttt{TxRqst}) not be affected by resetting the C\_CAN FD8. All the Message Objects must be initialized by the CPU or they must be set not valid (\texttt{MsgVal} = '0'). The bit timing must be configured before the CPU clears the \texttt{Init} bit in the CAN Control Register.

The configuration of a Message Object is done by programming Mask, Arbitration, Control and Data field of one of the two interface register sets to the desired values. By writing to the corresponding IFx Command Request Register, the IFx Message Buffer Registers are loaded into the addressed Message Object in the Message RAM.

When the \texttt{Init} bit in the CAN Control Register is cleared, the CAN Protocol Controller state machine of the CAN\_Core and the Message Handler State Machine control the C\_CAN FD8's internal data flow. Received messages that pass the acceptance filtering are stored into the Message RAM, messages with pending transmission request are loaded into the CAN\_Core's Shift Register and are transmitted via the CAN bus.

The CPU reads received messages and updates messages to be transmitted via the IFx Interface Registers. Depending on the configuration, the CPU is interrupted on certain CAN message and CAN error events.

4.2 Message Handler State Machine

The Message Handler controls the data transfer between the Rx/Tx Shift Register of the CAN Core, the Message RAM and the IFx Registers.

The Message Handler FSM controls the following functions:

- Data Transfer from IFx Registers to the Message RAM
- Data Transfer from Message RAM to the IFx Registers
- Data Transfer from Shift Register to the Message RAM
- Data Transfer from Message RAM to Shift Register
- Data Transfer from Shift Register to the Acceptance Filtering unit
- Scanning of Message RAM for a matching Message Object
- Handling of \texttt{TxRqst} flags.
- Handling of interrupts.

4.2.1 Data Transfer from / to Message RAM

When the CPU initiates a data transfer between the IFx Registers and Message RAM, the Message Handler sets the \texttt{Busy} bit in the respective IFx Command Request Register to ‘1’. After the transfer has completed, the \texttt{Busy} bit is set back to ‘0’ (see figure 10).

The respective IFx Command Mask Register specifies whether a complete Message Object or only parts of it will be transferred. Due to the structure of the Message RAM it is not possible to write single bits/bytes of one Message Object, it is always necessary to write a complete Message Object to the Message RAM. Therefore the data transfer from the IFx Message Buffer Registers to the Message RAM (\texttt{WR/RD} = ‘1’) requires a read-modify-write cycle. First
that parts of the Message Object that are not to be changed are read from the Message RAM to the selected IFx Message Buffer Registers and then the complete contents of the selected IFx Message Buffer Registers is written to the Message Object.

**Figure 10: Data Transfer between IFx Registers and Message RAM**

After a partial write of a Message Object (WR/RD = ‘1’), the IFx Message Buffer Registers that are not selected by the respective IFx Command Mask Register will be set to the actual contents of the selected Message Object.

After the partial read of a Message Object (WR/RD = ‘0’), the IFx Message Buffer Registers that are not selected by the respective IFx Command Mask Register will be left unchanged.

### 4.2.2 Transmission of Messages

If the shift register of the CAN Core cell is ready for loading and if there is no data transfer between the IFx Registers and Message RAM, the **MsgVal** bits in the Message Valid Register and the **TxRqst** bits in the Transmission Request Register are evaluated. The valid Message Object with the highest priority pending transmission request is loaded into the shift register by the Message Handler and the transmission is started. The Message Object’s **NewDat** bit is reset.

After a successful transmission and if no new data was written to the Message Object (**NewDat** = ‘0’) since the start of the transmission, the **TxRqst** bit will be reset. If **TxIE** is set, **IntPnd** will be set after a successful transmission. If the C_CAN FD8 has lost the arbitration or
if an error occurred during the transmission, the message will be retransmitted as soon as the CAN bus is free again. If meanwhile the transmission of a message with higher priority has been requested, the messages will be transmitted in the order of their priority.

4.2.3 Acceptance Filtering of Received Messages

When the arbitration and control field (Identifier up to DLC) of an incoming message is completely shifted into the Rx/Tx Shift Register of the CAN Core, the Message Handler FSM starts the scanning of the Message RAM for a matching valid Message Object.

To scan the Message RAM for a matching Message Object, the Acceptance Filtering unit is loaded with the arbitration bits from the CAN Core shift register. Then the arbitration and mask fields (including MsgVal, UMask, NewDat, and EoB) of Message Object 1 are loaded into the Acceptance Filtering unit and are compared with the arbitration field from the shift register. This is repeated with each following Message Object until a matching Message Object is found or until the end of the Message RAM is reached.

If a match occurs, the scanning is stopped and the Message Handler FSM proceeds depending on the type of frame (Data Frame or Remote Frame) received.

4.2.3.1 Reception of Data Frame

The Message Handler FSM stores the message from the CAN Core shift register into the respective Message Object in the Message RAM. Not only the data bytes, but all arbitration bits and the Data Length Code are stored into the corresponding Message Object. This is implemented to keep the data bytes connected with the identifier even if arbitration mask registers are used.

The NewDat bit is set to indicate that new data (not yet seen by the CPU) has been received. The CPU should reset NewDat when it reads the Message Object. If at the time of the reception the NewDat bit was already set, MsgLst is set to indicate that the previous data (supposedly not seen by the CPU) is lost. If the RxIE bit is set, the IntPnd bit is set, causing the Interrupt Register to point to this Message Object.

The TxRqst bit of this Message Object is reset to prevent the transmission of a Remote Frame, while the requested Data Frame has just been received.

4.2.3.2 Reception of Remote Frame

When a Remote Frame is received, three different configurations of the matching Message Object have to be considered:

1) Dir = ‘1’ (direction = transmit), RmtEn = ‘1’, UMask = ‘1’ or ‘0’
   At the reception of a matching Remote Frame, the TxRqst bit of this Message Object is set. The rest of the Message Object remains unchanged.

2) Dir = ‘1’ (direction = transmit), RmtEn = ‘0’, UMask = ‘0’
   At the reception of a matching Remote Frame, the TxRqst bit of this Message Object remains unchanged; the Remote Frame is ignored.

3) Dir = ‘1’ (direction = transmit), RmtEn = ‘0’, UMask = ‘1’
   At the reception of a matching Remote Frame, the TxRqst bit of this Message Object is reset. The arbitration and control field (Identifier + IDE + RTR + DLC) from the shift register is stored into the Message Object in the Message RAM and the NewDat bit of this Message Object is set. The data field of the Message Object remains unchanged while bits FDF, BRS, and ESI are reset; the Remote Frame is treated similar to a received Data Frame.

Note: Remote frames are always transmitted in Classical CAN format.
4.2.4 Receive / Transmit Priority

The receive/transmit priority for the Message Objects is attached to the message number. Message Object 1 has the highest priority, while Message Object 32 has the lowest priority. If more than one transmission request is pending, they are serviced according to the priority of the corresponding Message Object.

4.3 Configuration of a Transmit Object

Figure 11 shows how a Transmit Object should be initialised.

<table>
<thead>
<tr>
<th>MsgVal</th>
<th>Arb</th>
<th>CAN_FD</th>
<th>Data</th>
<th>Mask</th>
<th>EoB</th>
<th>Dir</th>
<th>NewDat</th>
<th>MsgLst</th>
<th>RxIE</th>
<th>TxIE</th>
<th>IntPnd</th>
<th>RmtEn</th>
<th>TxRqst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>appl.</td>
<td>appl.</td>
<td>appl.</td>
<td>appl.</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>appl.</td>
<td>0</td>
<td>appl.</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 11: Initialisation of a Transmit Object

The Arbitration Registers (ID28-0 and Xtd bit) are given by the application. They define the identifier and type of the outgoing message. If an 11-bit Identifier (“Standard Frame”) is used, it is programmed to ID28 - ID18, ID17 - ID0 can then be disregarded.

The CAN FD format bits FDF, BRS, and ESI are also given by the application.

If the TxIE bit is set, the IntPnd bit will be set after a successful transmission of the Message Object.

If the RmtEn bit is set, a matching received Remote Frame will cause the TxRqst bit to be set; the Remote Frame will autonomously be answered by a Data Frame.

The Data Registers (DLC3-0, Data0-7) are given by the application, TxRqst and RmtEn may not be set before the data is valid.

The Mask Registers (Msk28-0, UMask, MXtd, and MDir bits) may be used (UMask='1') to allow groups of Remote Frames with similar identifiers to set the TxRqst bit (for details see section 4.2.3.2). The Dir bit should not be masked.

4.4 Updating a Transmit Object

The CPU may update the data bytes of a Transmit Object any time via the IFx Interface registers, neither MsgVal nor TxRqst have to be reset before the update.

Even if only a part of the data bytes are to be updated, all four bytes of the corresponding IFx Data A Register or IFx Data B Register have to be valid before the content of that register is transferred to the Message Object. Either the CPU has to write all four bytes into the IFx Data Register or the Message Object is transferred to the IFx Data Register before the CPU writes the new data bytes.

When only the (eight) data bytes are updated, first 0x0087 is written to the IFx Command Mask Register and then the number of the Message Object is written to the IFx Command Request Register, concurrently updating the data bytes and setting TxRqst.

To prevent the reset of TxRqst at the end of a transmission that may already be in progress while the data is updated, NewDat has to be set together with TxRqst. For details see section 4.2.2.

When NewDat is set together with TxRqst, NewDat will be reset as soon as the new transmission has started.
4.5 Configuration of a Receive Object

Figure 11 shows how a Receive Object should be initialised.

<table>
<thead>
<tr>
<th>MsgVal</th>
<th>Arb</th>
<th>CAN_FD</th>
<th>Data</th>
<th>Mask</th>
<th>EoB</th>
<th>Dir</th>
<th>NewDat</th>
<th>MsgLst</th>
<th>RxIE</th>
<th>TxIE</th>
<th>IntPnd</th>
<th>RmtEn</th>
<th>TxRqst</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>appl.</td>
<td>appl.</td>
<td>appl.</td>
<td>appl.</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>appl.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 12: Initialisation of a Receive Object

The Arbitration Registers (ID28-0 and Xtd bit) are given by the application. They define the identifier and type of accepted received messages. If an 11-bit Identifier (“Standard Frame”) is used, it is programmed to ID28 - ID18, ID17 - ID0 can then be disregarded. When a Data Frame with an 11-bit Identifier is received, ID17 - ID0 will be set to ‘0’.

The CAN FD format bits FDF, BRS, and ESI are also given by the application. They are taken from the received frame.

If the RxIE bit is set, the IntPnd bit will be set when a received Data Frame is accepted and stored in the Message Object.

The Data Length Code (DLC3-0) is given by the application. When the Message Handler stores a Data Frame in the Message Object, it will store the received Data Length Code and eight data bytes. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by non specified values.

The Mask Registers (Msk28-0, UMask, MXtd, and MDir bits) may be used (UMask=’1’) to allow groups of Data Frames with similar identifiers to be accepted (for details see section 4.2.3.1). The Dir bit should not be masked in typical applications.

4.6 Handling of Received Messages

The CPU may read a received message any time via the IFx Interface registers, the data consistency is guaranteed by the Message Handler state machine.

Typically the CPU will write first 0x007F to the IFx Command Mask Register and then the number of the Message Object to the IFx Command Request Register. That combination will transfer the whole received message from the Message RAM into the IFx Message Buffer Register. Additionally, the bits NewDat and IntPnd are cleared in the Message RAM (not in the Message Buffer).

If the Message Object uses masks for acceptance filtering, the arbitration bits show which of the matching messages has been received.

The actual value of NewDat shows whether a new message has been received since last time this Message Object was read. The actual value of MsgLst shows whether more than one message has been received since last time this Message Object was read. MsgLst will not be automatically reset.

By means of a Remote Frame, the CPU may request another CAN node to provide new data for a receive object. Setting the TxRqst bit of a receive object will cause the transmission of a Remote Frame with the receive object’s identifier. This Remote Frame triggers the other CAN node to start the transmission of the matching Data Frame. If the matching Data Frame is received before the Remote Frame could be transmitted, the TxRqst bit is automatically reset.
4.7 Configuration of a FIFO Buffer

With the exception of the \textit{EoB} bit, the configuration of Receive Objects belonging to a FIFO Buffer is the same as the configuration of a (single) Receive Object, see section 4.5.

To concatenate two or more Message Objects into a FIFO Buffer, the identifiers and masks (if used) of these Message Objects have to be programmed to matching values. Due to the implicit priority of the Message Objects, the Message Object with the lowest number will be the first Message Object of the FIFO Buffer. The \textit{EoB} bit of all Message Objects of a FIFO Buffer except the last have to be programmed to \textit{zero}. The \textit{EoB} bits of the last Message Object of a FIFO Buffer is set to \textit{one}, configuring it as the \textit{E}nd of the \textit{B}lock.

4.8 Reception of Messages with FIFO Buffers

Received messages with identifiers matching to a FIFO Buffer are stored into a Message Object of this FIFO Buffer starting with the Message Object with the lowest message number.

When a message is stored into a Message Object of a FIFO Buffer the \textbf{NewDat} bit of this Message Object is set. By setting \textbf{NewDat} while \textbf{EoB} is \textit{zero} the Message Object is locked for further write accesses by the Message Handler until the CPU has written the \textbf{NewDat} bit back to \textit{zero}.

Messages are stored into a FIFO Buffer until the last Message Object of this FIFO Buffer is reached. If none of the preceding Message Objects is released by writing \textbf{NewDat} to \textit{zero}, all further messages for this FIFO Buffer will be written into the last Message Object of the FIFO Buffer and therefore overwrite previous messages.

4.8.1 Reading from a FIFO Buffer

When the CPU transfers the contents of Message Object to the IFx Message Buffer Registers by writing its number to the IFx Command Request Register, the corresponding IFx Command Mask Register should be programmed the way that bits \textbf{NewDat} and \textbf{IntPnd} are reset to \textit{zero} (\texttt{TxRqst/NewDat} = ‘1’ and \texttt{ClrIntPnd} = ‘1’). The values of these bits in the IFx Message Control Register always reflect the status before resetting the bits.

To assure the correct function of a FIFO Buffer, the CPU should read out the Message Objects starting at the FIFO Object with the lowest message number.

Figure 13 shows how a set of Message Objects which are concatenated to a FIFO Buffer can be handled by the CPU.
Figure 13: CPU Handling of a FIFO Buffer

START Message Interrupt

Read Interrupt Pointer

case Interrupt Pointer

0x8000h

else

0x0000h

Status Change Interrupt Handling

MessageNum = Interrupt Pointer

Write MessageNum to IFx Command Request

(Read Message to IFx Registers,
Reset NewDat = 0,
Reset IntPnd = 0)

Read IFx Message Control

NewDat = 1

Yes

No

Read Data from IFx Data A,B

EoB = 1

Yes

No

MessageNum = MessageNum + 1

Figure 13: CPU Handling of a FIFO Buffer
4.9 Handling of Interrupts

If several interrupts are pending, the CAN Interrupt Register will point to the pending interrupt with the highest priority (\texttt{IntId}), disregarding their chronological order. An interrupt remains pending until the CPU has cleared it.

The interrupt identifier \texttt{IntId} in the Interrupt Register indicates the cause of the interrupt. When no interrupt is pending, the register will hold the value \texttt{zero}. If the value of the Interrupt Register is different from \texttt{zero}, then there is an interrupt pending and, if \texttt{IE} is set, the interrupt line to the CPU is active. The interrupt line remains active until the Interrupt Register is back to value \texttt{zero} (the cause of the interrupt is reset) or until \texttt{IE} is reset.

The Status Interrupt has the highest priority. Among the message interrupts, the Message Object’s interrupt priority decreases with increasing message number.

A message interrupt is cleared by clearing the Message Object’s \texttt{IntPnd} bit. The Status Interrupt is cleared by reading the Status Register.

The value \texttt{0x8000} indicates that an interrupt is pending because the CAN Core has updated (not necessarily changed) the Status Register resp. the FD Status Register (Error Interrupt or Status Interrupt). This interrupt has the highest priority. The CPU can update (reset) the Status Register bits \texttt{RxOk, TxOk}, and \texttt{LEC} by writing to the Status Register. A write access by the CPU to the Status Registers can never generate or reset an interrupt. When the CPU reads the FD Control Register while CAN FD operation is enabled (bit \texttt{FDOE} in the FD Control Register set), the FD Status Register bits \texttt{RxFDF, RxBRS, RxESI, RxOK, and TxOK} are reset while \texttt{DLEC} and \texttt{LEC} are set to \texttt{0b111}.

All other values indicate that the source of the interrupt is one of the Message Objects, \texttt{IntId} points to the pending message interrupt with the highest interrupt priority.

The CPU controls whether a change of the Status Registers may cause the Interrupt Register to be set to \texttt{IntId} Status Interrupt (bits \texttt{EIE} and \texttt{SIE} in the CAN Control Register as well as bit \texttt{DSIE} in the FD Control Register) and whether the interrupt line becomes active when the Interrupt Register is different from \texttt{zero} (bit \texttt{IE} in the CAN Control Register). The Interrupt Register will be updated even when \texttt{IE} is not set.

The CPU has two possibilities to follow the source of a message interrupt. First it can follow the \texttt{IntId} in the Interrupt Register and second it can poll the Interrupt Pending Register (see section 3.4.4).

An interrupt service routine reading the message that is the source of the interrupt may read the message and reset the Message Object’s \texttt{IntPnd} at the same time (bit \texttt{ClrIntPnd} in the \texttt{IFx Command Mask Register}). When \texttt{IntPnd} is cleared, the Interrupt Register will point to the next Message Object with a pending interrupt.

4.10 CAN with Flexible Data-Rate

CAN FD operation is enabled by setting bit \texttt{FDOE} in the FD Control Register. In case CAN FD with bit rate switching shall be used, also bit \texttt{BRSE} in the FD Control Register has to be set. Both bits can only be set while bits \texttt{Init} and \texttt{CCE} in the CAN Control Register are set. After the initialization mode is left (\texttt{Init} set to ‘0’), both bits are locked.

The previously reserved bit in CAN frames with 11-bit identifiers and the first previously reserved bit in CAN frames with 29-bit identifiers will now be decoded as \texttt{FDF} bit. \texttt{FDF = recessive} signifies a CAN FD frame, \texttt{FDF = dominant} signifies a Classic CAN frame. In a CAN FD frame, the two bits following \texttt{FDF, res} and \texttt{BRS}, decide whether the bit rate inside of this CAN FD frame is switched. A CAN FD bit rate switch is signified by \texttt{res = dominant} and
BRS = recessive. The coding of res = recessive is reserved for future expansion of the protocol. In case the C_CAN FD8 receives a frame with FDF = recessive and res = recessive, it will signal a Protocol Exception Event by setting bit PXE in the FD Control Register. When Protocol Exception Handling is enabled (FD Control Register PXHD = ‘0’), this causes the operation state to change from Receiver (FD Status Register ACT = “10”) to Integrating (FD Status Register ACT = “00”) at the next sample point. In case Protocol Exception Handling is disabled (FD Control Register PXHD = ‘1’), the C_CAN FD8 will treat a recessive res bit as an form error and will respond with an error frame.

In CAN FD frames, the coding of the DLC differs from the Classic CAN format. The DLC codes 0 to 8 have the same coding as in Classic CAN, the codes 9 to 15, which in Classic CAN all code a data field of 8 bytes, are coded according to figure 14.

<table>
<thead>
<tr>
<th>DLC</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Data Bytes</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>32</td>
<td>48</td>
<td>64</td>
</tr>
</tbody>
</table>

Figure 14: Coding of DLC in CAN FD

In CAN FD frames, the bit timing will be switched inside the frame, after the BRS (Bit Rate Switch) bit, if this bit is recessive. Before the BRS bit, in the CAN FD Arbitration Phase, the nominal CAN bit timing is used as defined by the Nominal Bit Timing Registers 1, 2. In the following CAN FD Data Phase, the data bit timing is used as defined by the Data Bit Timing Register. The bit timing is switched back from the data bit timing at the CRC Delimiter or when an error is detected, whichever occurs first.

In CAN FD frames, the value of the bit ESI (Error Status Indicator) is determined by the transmitter’s error state at the start of the transmission. If the transmitter is error passive, ESI is transmitted recessive, else it is transmitted dominant. For special applications bit ESI of an error active transmitter may be transmitted recessive by setting the Message Object’s ESI bit to ‘1’.

4.10.1 Frame Transmission in CAN FD Mode

To transmit frames in CAN FD format, FD operation has to be enabled by programming FDOE = ‘1’ in the FD Control Register. In addition, BRSE has to be set to enable transmission with bit rate switching. If enabled, bit rate switching can be requested by writing BRSR = ‘1’. A change of BRSR becomes effective when the next transmission after setting/resetting BRSR is started. Ongoing transmissions are not affected.

The CAN FD format bit FDF has to be set to configure a Message Object for CAN FD transmission. The state of BRS is ignored; whether the message is transmitted with bit rate switching or not depends only on the state of BRSR. In case ESI in the Message Object is configured to ‘0’, the value of ESI in the transmitted frame reflects the error state of the transmitters protocol engine. In case ESI in the Message Object is configured to ‘1’, the message is transmitted with ESI set, signalling to the receiver that the transmitter is error passive independent of its actual error state.

In case the DLC of a Transmit Object is configured to a value higher than eight, the bytes not defined by the Transmit Object are transmitted as “0xCC” (padding bytes).
4.10.2 Frame Reception in CAN FD Mode

To receive frames in CAN FD format, FD operation has to be enabled by programming FDOE = ‘1’ in the FD Control Register.

As with Classic CAN frames, CAN FD frames have to pass acceptance filtering to be stored in a matching Message Object. The CAN FD format bits FDF, BRS, and ESI are updated from the received frame. For CAN FD frames bit FDF is always ‘1’, while BRS indicates whether the frame was received with bit rate switching (BRS = ‘1’) or not (BRS = ‘0’). Bit ESI reflects the error state of the received frame’s transmitter. ESI = ‘0’ indicates that the transmitter was in error active state, with ESI = ‘1’ the transmitter was error passive (receive or transmit error count ≥ 128).

In case a CAN FD frame with more than eight data bytes is received, only the first eight bytes are stored into the matching Receive Object.

4.10.3 Configuration of the Transmitter Delay Compensation

In CAN FD, transmitter delay compensation is used in the data phase of transmitting nodes to compensate the delay from the transmitters can_tx output through the attached transceiver to its can_rx input.

During frame transmission, the transmitter compares the bit transmitted at its can_tx output against the received bit at its can_rx input. In case the transmitter delay causes this bit to be received after the transmitter’s sample point, this will result in a bit error at the transmitter.

The transmitter delay compensation enables configurations where the data bit time is shorter than the transmitter delay, it is described in detail in the new ISO11898-1. It is enabled by setting bit TDCE in the FD Control Register.

The received bit is compared against the transmitted bit at the Secondary Sample Point SSP. The SSP is defined as the sum of the measured delay from the can_tx output to the can_rx input and the transmitter delay compensation offset as configured by TDCO in the Transmitter Delay Compensation Register. TDCO can be used to adjust the position of the SSP inside the received bit. The actual transmitter delay compensation value SSPP is available in the SSP Position Register. SSPP is cleared when CCE and Init are set and is updated at each transmission of an FD frame while TDCE is set and TDCNM is not set.

The following boundary conditions have to be considered for the transmitter delay compensation implemented in the C_CAN FD8:

- The sum of the measured delay from can_tx to can_rx and the configured transmitter delay compensation offset TDCO has to be less or equal 127 mtq. In case this sum exceeds 127 mtq, the maximum value of 127 mtq is used for transmitter delay compensation.
- When in case of fixed transmitter delay compensation the sum of TDCF and TDCO exceeds the specified limit of 0x7F, the sum will be limited to 7 bit, eliminating the MSB.
- The resulting SSP position SSPP has to be less than 6 bit times in the data phase.
- The data phase ends at the sample point of the CRC delimiter, that stops checking of receive bits at the SSPs.

There are two possibilities to compensate the transmitter delay; transmitter delay measurement or using a fixed transmitter delay compensation value.
4.10.3.1 Transmitter Delay Compensation Measurement

If transmitter delay compensation measurement is enabled by programming TDCNM = ‘0’, the measurement is started within each transmitted CAN FD frame at the falling edge of bit FDF. The measurement is stopped when this edge is seen at the can_rx input of the transmitter. The resolution of this measurement is one mtq.

![Diagram of transmitter delay compensation measurement](image)

To avoid that a dominant glitch inside the received FDF bit ends the delay compensation measurement before the falling edge of the received res bit, resulting in a to early SSP position, the use of a transmitter delay compensation filter window can be enabled by programming TDCF in the Transmitter Delay Compensation Register. This defines a minimum value for the SSP position, dominant edges on can_rx, that would result in an earlier SSP position are ignored for transmitter delay measurement. The measurement is stopped when the SSP position is at least TDCF AND can_rx is low.

4.10.3.2 Fixed Transmitter Delay

Instead of measuring the transmitter delay, a fixed transmitter delay compensation value can be used. This is enabled by programming FD Control Register bit TDCNM = ‘1’. In this case the fixed transmitter delay value has to be written to TDCF in the Transmitter Delay Compensation Register. It directly defines the delay. Reading SSPP from the SSP Position Register will show the sum of TDCF and TDCO after CCE or Init is cleared (see section 3.2.12 and section 3.2.13).
1. Appendix

1.1 List of Figures

Figure 1: Block Diagram of the C_CAN FD8 ........................................... 10
Figure 2: CAN Core in Silent Mode ......................................................... 13
Figure 3: CAN Core in Loop Back Mode .................................................. 13
Figure 4: CAN Core in Loop Back combined with Silent Mode .................... 14
Figure 5: C_CAN FD8 Register Summary ............................................... 18
Figure 6: Coding Register Bit access / reset ........................................... 18
Figure 7: IF1 and IF2 Message Interface Register Sets ............................. 29
Figure 8: Structure of a Message Object in the Message RAM ................. 35
Figure 9: Example for Coding of Revisions ........................................... 40
Figure 10: Data Transfer between IFx Registers and Message RAM ........... 42
Figure 11: Initialisation of a Transmit Object ......................................... 44
Figure 12: Initialisation of a Receive Object .......................................... 45
Figure 13: CPU Handling of a FIFO Buffer ............................................ 47
Figure 14: Coding of DLC in CAN FD .................................................... 49
Figure 15: Calculation of SSP Position .................................................. 51