# CTU CAN FD IP CORE

## Datasheet

LOGIC DESIGN SERVICES l.t.d.

February 1, 2026

| Document Version | Date | Change description |
|---|---|---|
| 1.0 | 07-2015 | Initial version describing release 1.0 |
| 2.0 | 09-2016 | Added test framework description. Updated document to cover latest description of CAN Core. |
| | 07-2018 | Updated register map description, external references to generated maps. Updated block diagrams. Updated test framework description. Updated Synthesis table. |
| 2.1 | 10-2018 | Added Linux driver description |
| | 12-2018 | Added Register map block diagram after re-implementation of registers via Register map generator. |
| | 12-2018 | Added CRC Wrapper. Extended CRC description. |
| | 01-2019 | Added TIMESTAMP_LOW, TIMESTAMP_HIGH registers. |
| | 03-2019 | Re-worked Prescaler. Removed 0x3 in bits 23:20 of address. |
| 2.2 | 26-09-2019 | Split functional descripion and register map from original document. |
| | 21-10-2019 | Clarify TXT buffer behaviour when node goes bus-off. |
| | 31-10-2019 | Clarify Bus-off behaviour aftet Start-up. Clarify that frame must be inserted to TXT Buffer before sending. |
| | 18-11-2019 | Clarify behaviour of Transmitter delay measurement. Add notes on RX frame timestamping. Extend SSP position to 255. |
| | 13-12-2019 | Clarify that only TEC above 255 will cause node to go Bus off. |
| | 30-4-2020 | Add SETTING[PEX] and Protocol exception support. |
| | 28-10-2020 | Add frame filters examples, add TBFBO and FDRF bits in SETTINGS registers, minor refactoring. |
| | 05-11-2020 | Add general overview and TX frame type description. |
| | 4-2-2021 | Change license |
| 2.3 | 4-2-2021 | Added MODE[ROM] - Restricted operation mode. |
| | 23-2-2021 | Add TXTB_INFO and mention generic number of TXT buffers. |
| | 9-4-2021 | Add RETR_CTR register. |
| | 26-4-2021 | Add chapter about memory testability. |
| | 17-05-2021 | Add STATUfS[STCNT] and STATUS[STRGS] bits. |
| | 26-05-2021 | Reduce maximal number of bits on the fly during secondary sampling to 4. |
| | 29-05-2021 | Add detailed description of disabling node by SETTINGS[ENA]. |
| | 11-06-2021 | Add MODE[RXBAM] and COMMAND[RXRPMV] bits, describe RX buffer modes. |
| | 18-06-2021 | Add MODE[TTTM] bit to enable time-triggered transmission. |
| 2.4 | 28-08-2021 | Move to new release of CTU CAN FD. Bump document version accordingly. |
| | 1-4-2022 | Add MODE[TXBBM], MODE[SAM], STATUS[RXPE], STATUS[TXPE], COMMAND [CTPXE] and COMMAND[CRPXE]. Add FRAME_FORMAT_W bits which allow flipping of CRC or Stuff count. Add section on parity mechanism testing. |
| - | 27-6-2022 | SW commands on TXT Buffers in MODE[TXBBM] are automatically applied to "backup" TXT Buffers. Add ***reset_buffer_rams*** and ***active_timestamp_bits*** configuration parameters. |
| | 5-7-2022 | Add SETTINGS[PCHKE] bit to control enable / disable of parity checking. |
| 2.5 | 9-12-2023 | Move to new release of CTU CAN FD. Bump document version accordingly. |
| | 8-5-2024 | Add note about TXBHCI interrupt behavior. |

| | | |
|---|---|---|
| | 19-6-2024 | Add LBPF bit to RX frame. Add MODE[ERFM]. Add FRAME_FORMAT_W[ERF], FRAME_FORMAT_W[ERF_*]. Add Error frame reception to RX buffer. Add ERR_CAPT[ERR_ERP] bit. Add FRAME_FORMAT_W[LBTBI] and FRAME_FORMAT_W[IVLD]. |
| 2.6 | 19-6-2025 | Move to new relase of CTU CAN FD. |
| | 15-9-2025 | Fix code samples to readout RWCNT - 3 words. |
| | 17-12-2025 | Clarify Frame filters behavior in case of logged error frames. |
| | 21-12-2025 | Extend TRV_DELAY to 8 bits (max 255). Extend SSP position capacity to 511, remove SSP saturation since it is un-needed now. Extend max bits in flight to 7. |
| 2.7 | 04-1-2026 | Move to new relase of CTU CAN FD. |

# Contents

**Format** **1**

**1 Introduction** **2**

1.1 General overview . . . 2
1.2 Features . . . 2
1.3 License . . . 2
1.4 Source code access . . . 3
1.5 Block diagram . . . 3
1.6 Implementation parameters . . . 4
1.7 Configuration parameters . . . 4

**2 Functional description** **5**

2.1 Clock . . . 5
2.2 Reset . . . 5
2.3 Memory organization . . . 5
2.4 Time base . . . 5
2.5 Operating modes . . . 6
2.6 Initialization sequence . . . 7
2.7 De-initialization sequence . . . 7
2.8 CAN bus configuration . . . 7
2.8.1 Bit rate . . . 7
500 Kbit / 2 Mbit example . . . 8
2.8.2 Transmitter delay . . . 8
2.8.3 Secondary sampling point . . . 9
2.8.4 CAN FD support . . . 10
2.8.5 Protocol exception handling . . . 10
2.8.6 Implementation type . . . 11
2.8.7 Minimum bit time / Maximal bit rate . . . 11
2.9 CAN frame transmission . . . 12
2.9.1 TXT buffer selection . . . 12

2.9.2 Time triggered transmission mode . . . 13
2.9.3 Type of transmitted CAN frame . . . 14
2.9.4 Retransmitt limitation . . . 15
2.9.5 Abort . . . 15
2.9.6 TXT buffer - Bus-off behavior . . . 16
2.9.7 Sample code . . . 16
2.10 CAN frame reception . . . 17
2.10.1 Frame count . . . 17
2.10.2 Error frame reception . . . 18
2.10.3 RX buffer memory . . . 18
2.10.4 RX buffer status . . . 19
2.10.5 Overrun . . . 19
2.10.6 Flush . . . 19
2.10.7 Inconsistency protection . . . 19
2.10.8 Timestamping . . . 20
2.10.9 Frame filtering . . . 20
Bit filter . . . 21
Range filter . . . 21
2.10.10 Sample code 1 - Frame reception in automatic mode (32-bit access) . . . 21
2.10.11 Sample code 2 - Frame reception in manual mode (8-bit access) . . . 22
2.10.12 Sample code 3 - Bit filter configuration . . . 23
2.11 Fault confinement . . . 23
2.12 Interrupts . . . 24
2.12.1 Frame transmission and reception . . . 24
2.12.2 Fault confinement . . . 24
2.12.3 TXT buffers and RX buffer . . . 25
2.12.4 Error and Overload frame . . . 25
2.12.5 Other . . . 25
2.13 Fault Tolerance . . . 26
2.13.1 Parity protection on RX buffer RAM . . . 26
2.13.2 Parity protection on TXT Buffer RAMs . . . 27
2.13.3 TXT Buffer Backup mode . . . 28
2.13.4 Parity protection testing . . . 30
2.14 Special modes . . . 31
2.14.1 Loopback mode . . . 31
2.14.2 Self test mode . . . 31
2.14.3 Acknowledge forbidden mode . . . 32
2.14.4 Self acknowledge mode . . . 32
2.14.5 Bus monitoring mode . . . 32

2.14.6 Restricted operation mode . . . 32
2.14.7 Test mode . . . 32
2.15 Corrupting transmitted CAN frames . . . 33
2.15.1 Flip a bit of CRC field . . . 33
2.15.2 Flip a bit of Stuff count field . . . 33
2.15.3 Replace DLC with arbitrary value . . . 34
2.16 Other features . . . 34
2.16.1 Error code capture . . . 34
2.16.2 Arbitration lost capture . . . 34
2.16.3 Traffic counters . . . 34
2.16.4 Debug register . . . 34
2.16.5 Memory testability . . . 35

**3 CAN FD Core memory map 36**
3.1 Control registers . . . 37
3.1.1 DEVICE_ID . . . 38
3.1.2 VERSION . . . 38
3.1.3 MODE . . . 39
3.1.4 SETTINGS . . . 40
3.1.5 STATUS . . . 41
3.1.6 COMMAND . . . 42
3.1.7 INT_STAT . . . 43
3.1.8 INT_ENA_SET . . . 44
3.1.9 INT_ENA_CLR . . . 45
3.1.10 INT_MASK_SET . . . 45
3.1.11 INT_MASK_CLR . . . 46
3.1.12 BTR . . . 46
3.1.13 BTR_FD . . . 47
3.1.14 EWL . . . 48
3.1.15 ERP . . . 48
3.1.16 FAULT_STATE . . . 48
3.1.17 REC . . . 49
3.1.18 TEC . . . 49
3.1.19 ERR_NORM . . . 50
3.1.20 ERR_FD . . . 50
3.1.21 CTR_PRES . . . 50
3.1.22 FILTER_A_MASK . . . 51
3.1.23 FILTER_A_VAL . . . 52
3.1.24 FILTER_B_MASK . . . 52

3.1.25 FILTER_B_VAL . . . 53
3.1.26 FILTER_C_MASK . . . 54
3.1.27 FILTER_C_VAL . . . 54
3.1.28 FILTER_RAN_LOW . . . 55
3.1.29 FILTER_RAN_HIGH . . . 56
3.1.30 FILTER_CONTROL . . . 56
3.1.31 FILTER_STATUS . . . 57
3.1.32 RX_MEM_INFO . . . 58
3.1.33 RX_POINTERS . . . 58
3.1.34 RX_STATUS . . . 59
3.1.35 RX_SETTINGS . . . 59
3.1.36 RX_DATA . . . 60
3.1.37 TX_STATUS . . . 60
3.1.38 TX_COMMAND . . . 61
3.1.39 TXTB_INFO . . . 62
3.1.40 TX_PRIORITY . . . 63
3.1.41 ERR_CAPT . . . 63
3.1.42 RETR_CTR . . . 64
3.1.43 ALC . . . 65
3.1.44 TS_INFO . . . 65
3.1.45 TRV_DELAY . . . 66
3.1.46 SSP_CFG . . . 66
3.1.47 RX_FR_CTR . . . 67
3.1.48 TX_FR_CTR . . . 67
3.1.49 DEBUG_REGISTER . . . 68
3.1.50 YOLO_REG . . . 69
3.1.51 TIMESTAMP_LOW . . . 70
3.1.52 TIMESTAMP_HIGH . . . 70
3.2 TXT Buffer 1 . . . 72
3.3 TXT Buffer 2 . . . 73
3.4 TXT Buffer 3 . . . 74
3.5 TXT Buffer 4 . . . 75
3.6 TXT Buffer 5 . . . 76
3.7 TXT Buffer 6 . . . 77
3.8 TXT Buffer 7 . . . 78
3.9 TXT Buffer 8 . . . 79
3.10 Test registers . . . 80
3.10.1 TST_CONTROL . . . 80
3.10.2 TST_DEST . . . 81
3.10.3 TST_WDATA . . . 81
3.10.4 TST_RDATA . . . 82

**4 CAN FD frame format** **83**

4.1 CAN FD Frame format . . . 84
4.1.1 FRAME_FORMAT_W . . . 84
4.1.2 IDENTIFIER_W . . . 86
4.1.3 TIMESTAMP_L_W . . . 87
4.1.4 TIMESTAMP_U_W . . . 87
4.1.5 DATA_1_4_W . . . 88
4.1.6 DATA_5_8_W . . . 88
4.1.7 DATA_9_12_W . . . 89
4.1.8 DATA_13_16_W . . . 90
4.1.9 DATA_17_20_W . . . 90
4.1.10 DATA_21_24_W . . . 91
4.1.11 DATA_25_28_W . . . 92
4.1.12 DATA_29_32_W . . . 92
4.1.13 DATA_33_36_W . . . 93
4.1.14 DATA_37_40_W . . . 94
4.1.15 DATA_41_44_W . . . 94
4.1.16 DATA_45_48_W . . . 95
4.1.17 DATA_49_52_W . . . 96
4.1.18 DATA_53_56_W . . . 96
4.1.19 DATA_57_60_W . . . 97
4.1.20 DATA_61_64_W . . . 98
4.1.21 FRAME_TEST_W . . . 98

# Format

Throughout this datasheet following notation is kept:

- Common text is written with this font.
- Memory registers are described with capital letters:
  - e.g. REGISTER to describe a register
  - REGISTER [BIT_FIELD] to describe a bit field within a register.
- Explicit terms are written in apostrophe like so: “TX Failed”.

```
Source code examples are written by this font
```

# 1. Introduction

This document provides functional description of CTU CAN FD, programmers model, and parameters of CTU CAN FD. It is intended to be used as a reference for SW driver developers. Internal architecture of CTU CAN FD is described in [1].

## 1.1 General overview

CTU CAN FD is a soft IP-core written in VHDL with no vendor-specific libraries needed. It implements CAN FD protocol as specified by ISO11898-1.

## 1.2 Features

- Compliant with ISO11898-1 2015
- RX buffer FIFO with 32 - 4096 words (1-204 CAN FD frames with 64 byte of data)
- 2-8 TXT buffers (1 CAN FD frame in each TXT buffer)
- 32 bit slave memory interface (APB, AHB, RAM-like interface)
- Support of ISO and non-ISO CAN FD protocol
- Timestamping and Time triggered transmission
- Interrupts
- Loopback mode, Bus monitoring mode, ACK forbidden mode, Self-test mode, Restricted operation mode

## 1.3 License

RTL and testbench of CTU CAN FD IP core are published under following license:

THE COMPONENT IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHTHOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE COMPONENT OR THE USE OR OTHER DEALINGS IN THE COMPONENT.

Linux driver and low level driver are published under GPL v 2.0:

This program is free software; you can redistribute it and/or * modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

## 1.4 Source code access

CTU CAN FD source code is available in CTU FEE GitLab repository at:

https://gitlab.fel.cvut.cz/canbus/ctucanfd_ip_core

## 1.5 Block diagram

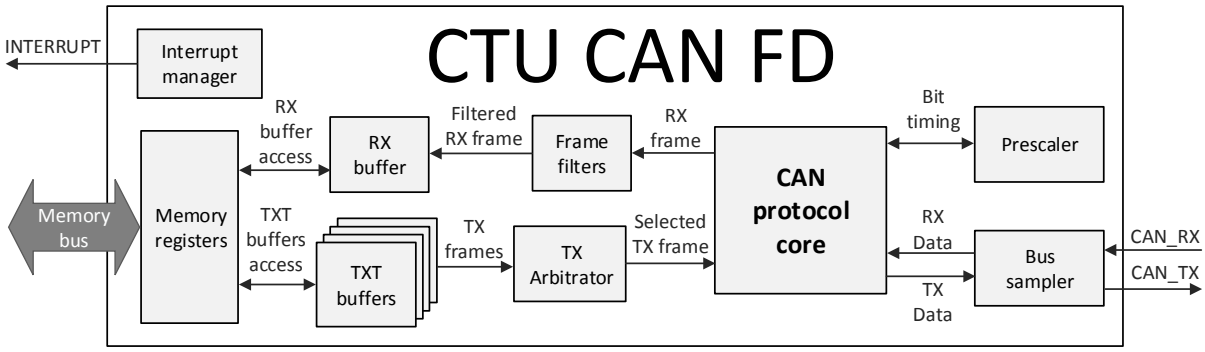


Figure 1.1: CTU CAN FD block diagram

## 1.6 Implementation parameters

| Parameter name | Value | Units |
|---|---|---|
| Minimum nominal time quanta | 1 | - |
| Minimum data time quanta | 1 | - |
| Information processing time | 2 | Minimum time quanta |
| Input delay ($t_{input}$) | 2 | System clock periods (see 2.1) |
| Nominal bit rate prescaler range (BTR[BRP] register) | 1 - 255 | |
| Data bit rate prescaler range (BTR_FD[BRP_FD] register) | 1 - 255 | |
| Minimal nominal bit time length | 8 | Time quanta |
| Minimal data bit time length | 5 | Time quanta |

Table 1.1: Implementation parameters

## 1.7 Configuration parameters

CTU CAN FD can be used with different options when implemented on ASIC or FPGA. These parameters are then readable by SW. Related parameters are described in 1.2.

| Parameter name | Value | Description |
|---|---|---|
| ***rx_buffer_size*** | 32 - 4096 | Size of RX buffer (number of 32bit words it can store). SW can read this value from RX_MEM_INFO[RX_BUFF_SIZE]. |
| ***sup_filt_A*** | true/false | Filter A is / is not present. If present, FILTER_STATUS[SFA] = 1. |
| ***sup_filt_B*** | true/false | Filter B is / is not present. If present, FILTER_STATUS[SFB] = 1. |
| ***sup_filt_C*** | true/false | Filter C is / is not present. If present, FILTER_STATUS[SFC] = 1. |
| ***sup_range*** | true/false | Range fitler is / is not present. If present, FILTER_STATUS[SFR] = 1. |
| ***sup_traffic_ctrs*** | true/false | Traffic counters are / are not present. If present, STATUS[STCNT] = 1. |
| ***txt_buffer_count*** | 2-8 | Number of TXT buffers available. Can be read from TXTB_INFO register. |
| ***sup_test_registers*** | true/false | Test registers for memory testability (Test Registers memory region) are / are not present . If present, STATUS[STRCNT] = 1. |
| ***sup_parity*** | true/false | Add parity bits to each word of TXT Buffer and RX buffer RAMs. If Parity protection is present, STATUS[SPRT] = 1. |
| ***reset_buffer_rams*** | true/false | When true, TXT Buffer and RX buffer RAMs are resettable by HW reset. |
| ***active_timestamp_bits*** | Integer | Number of active bits of CTU CAN FD timebase - 1. |

Table 1.2: Configuration parameters

# 2. Functional description

## 2.1 Clock

CTU CAN FD operates with a single clock called System clock. Every other timing parameter is derived from System clock. System clock frequency depends on the system that is integrating CTU CAN FD. System clock frequency corresponds to frequency of clock signal of CTU CAN FD.

## 2.2 Reset

After power-up CTU CAN FD shall be reset either by HW reset (see [1]), or by Soft reset. Soft reset is executed by writing MODE[RST] = 1. If HW reset was issued to CTU CAN FD , CTU CAN FD shall not be accessed for two clock periods of System clock. For example, if CTU CAN FD System clock is 100 MHz, SW shall wait 20 ns after HW reset was released. If Soft reset was issued, no waiting is required. Both, HW Reset and Soft reset have the same effect. By applying any reset, CTU CAN FD is put to following state:

- CTU CAN FD is disabled, it is not communicating on CAN bus (bus-off state).
- All memory registers of CTU CAN FD have reset value.
- Memories in CTU CAN FD (TXT buffer and RX buffer) are not reset.

## 2.3 Memory organization

CTU CAN FD memory map is organized as little-endian (e.g. EWL register is at address 0x2C, ERP register at address 0x2D, and FAULT_STATE register at address 0x2E). CTU CAN FD is a 32-bit peripheral, but all functionality of CTU CAN FD can be used by accessing the core by 8/16 bit accesses (with proper configuration, see settings MODE[RXBAM] - RX buffer Automatic Mode).

## 2.4 Time base

CTU CAN FD can have a time base available for Time triggered transmission or Timestamping of received CAN frames. Availability of such time base depends on integration of CTU CAN FD into a system. If such time base is available, its immediate value can be read from TIMESTAMP_H and TIMESTAMP_L registers. Time base is up-counting unsigned counter which measures flow of a time within the system where CTU CAN FD is integrated. Width of the time base may range from 1 to 64 bits, and it is defined by the system integrating CTU CAN FD. Number of active bits of time base is available in TS_INFO register.

## 2.5 Operating modes

After reset, CTU CAN FD is disabled, and it does not communicate on CAN bus (no transmission, reception, monitoring). Before CTU CAN FD is enabled, it shall be configured as is explained in 2.8. Once configured, CTU CAN FD can be enabled by writing SETTINGS[ENA] = 1. When SETTINGS[ENA] = 1 is set, CTU CAN FD starts bus integration, and joins CAN bus communication after receiving 11 consecutive recessive bits. When CTU CAN FD joins CAN bus communication, it becomes error-active. During bus integration CTU CAN FD is bus-off. Once CTU CAN FD becomes error active, it starts communicating on CAN bus. The moment when CTU CAN FD joined CAN bus communication can be determined by FCS interrupt, and subsequent probing of FAULT_STATE register (see 2.12). Basic operating modes of CTU CAN FD are shown in Figure 2.1.

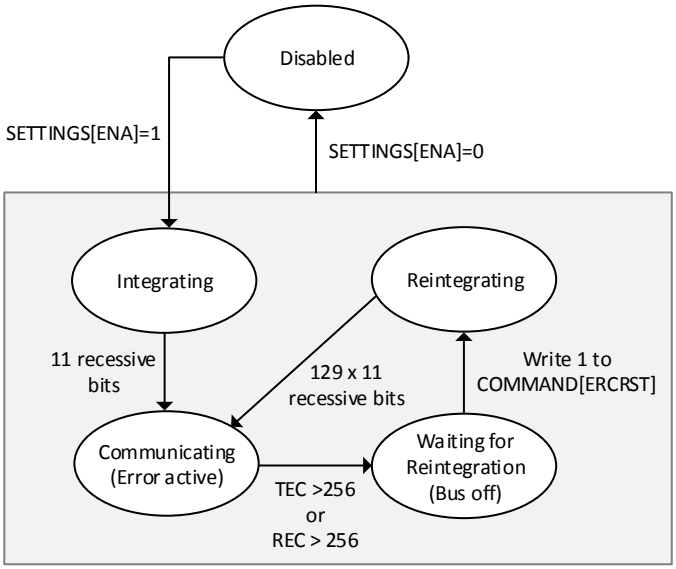


Figure 2.1: Operating modes

When CTU CAN FD is error-active, it takes part in CAN bus communication. If CTU CAN FD becomes error-passive, and later bus-off, it stops communicating on CAN bus. CTU CAN FD starts reintegrating to the bus when it receives Error counter reset command (writing COMMAND[ERCRST] = 1). Reintegration lasts until CTU CAN FD detects 129 sequences of 11 consecutive recessive bits. After 129 such sequences, CTU CAN FD becomes error-active again.

CTU CAN FD can be disabled at any time by writing logic 0 to SETTINGS[ENA] register. In such case:

- CTU CAN FD immediately stops communication on CAN bus, and transmits only recessive bits.
- TEC/REC counters are reset to 0, CTU CAN FD becomes bus-off.
- All TXT buffers move to "Empty" state (see 2.7), content of TXT buffer RAMs remains valid (memories are not reset).
- RX buffer is flushed (see 2.10.6).

It is recommended for CTU CAN FD not to be transmitting any frame when it is disabled by writing SETTINGS[ENA] = 0, as this would result in transmission of error frame by other nodes on CAN bus. SW driver operating on CTU CAN FD shall ensure that none of TXT buffers in CTU CAN FD is in "Ready", "TX in progress" or "Abort in progress" states (see 2.9).

**Note** COMMAND[ERCRST] is "sticky". If CTU CAN FD is not yet bus-off, and this command is issued, it will be remembered by CTU CAN FD, and it will automatically start reintegration upon nearest transition to bus-off. The reason is, that command can be issued in advance (during regular communication), and CTU CAN FD will re-integrate as quickly as possible after becoming bus-off (without SW additional delay caused by interaction with SW driver).

## 2.6 Initialization sequence

CTU CAN FD initialization sequence shall consist of following steps:

1. Reset (Either HW reset or Soft reset)
2. Configuration of CTU CAN FD:
   (a) Configure interrupts as in 2.12
   (b) Configure bit rate as in on this page
   (c) Configure other features (filters, special modes, etc...)
3. Enable CTU CAN FD by writing SETTINGS[ENA] = 1.
4. Poll on FAULT_STATE register, or wait on Fault confinement state changed interrupt (INT_STAT[FCSI]). Integration is finished when FAULT_STATE[ERA]=1 (CTU CAN FD becomes error-active).
5. Initialization is finished, SW driver can send and receive frames.

## 2.7 De-initialization sequence

CTU CAN FD de-initialization sequence shall consist of following steps:

1. Ensure that no TXT buffer is in any of "Ready", "TX in progress" or "Abort in progress" states. To do this SW issues **Set abort** command (see 2.9) to TXT buffers, and does not insert next frames for transmission into TXT Buffers.
2. Write SETTINGS[ENA]=0.

## 2.8 CAN bus configuration

### 2.8.1 Bit rate

Bit rate on CAN bus is derived from System clock (see 2.1). Basic unit of time on CAN bus is time quanta. Time quanta is derived from System clock by dividing its frequency by bit rate prescaler. CTU CAN FD has separate prescaler for nominal bit rate (BTR[BRP] register), and data bit rate (BTR_FD[BRP_FD] register). Bit rate on CTU CAN FD is configured by specifying Prop_Seg, Phase_Seg1 and Phase_Seg2 durations (as shown in Figure 2.2). These are specified in BTR (nominal bit rate) and BTR_FD (data bit rate) registers.

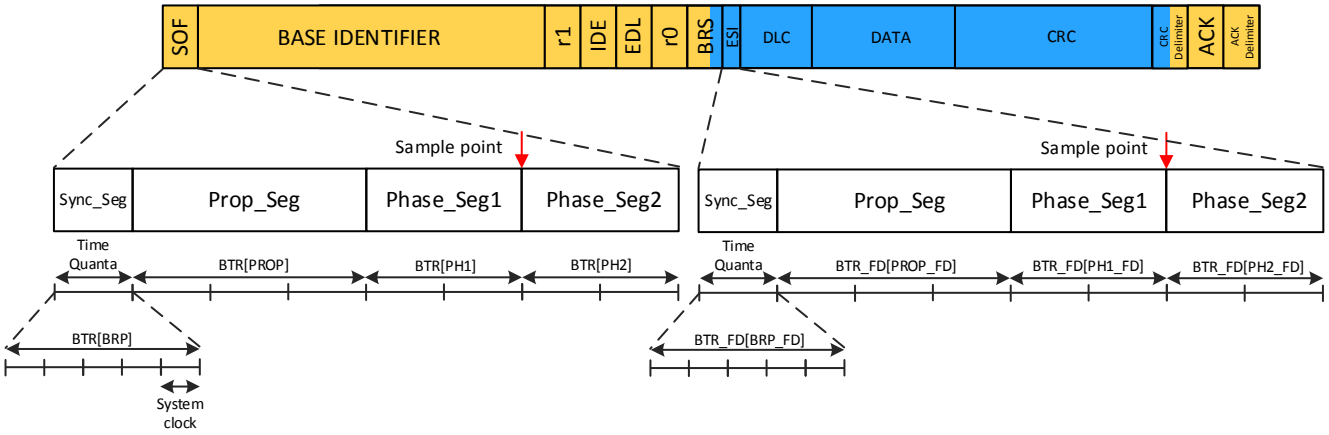


Figure 2.2: Bit time

#### 500 Kbit / 2 Mbit example

Common configuration of bit rate on CAN bus within automotive industry is 500 Kbit in nominal bit rate, and 2 Mbit in data bit rate. Following snippet shows example configuration assuming 100 MHz System clock frequency with sample point in 80% of bit:

```
#define CTU_CAN_FD_BASE 0x12000000
#define BTR_ADDR CTU_CAN_FD_BASE+0x24
#define BTR_FD_ADDR CTU_CAN_FD_BASE+0x28

uint32 btr;
btr = (4 << 19);     // Time Quanta: 4
btr |= 29;           // Prop: 29
btr |= (10 << 7);    // Phase 1: 10
btr |= (10 << 13);   // Phase 2: 10
btr |= (3 << 27);    // SJW: 3
can_write_32(BTR_ADDR, btr); // (29+10+10+1)*4=200*10ns=2us=500Kbit

uint32 btr_fd;
btr_fd = (1 << 19);     // Time Quanta: 1
btr_fd |= 29;           // Prop: 29
btr_fd |= (10 << 7);    // Phase 1: 10
btr_fd |= (10 << 13);   // Phase 2: 10
btr_fd |= (3 << 27);    // SJW: 3
can_write_32(BTR_FD_ADDR, btr_fd); // (29+10+10+1)*1=50*10ns=0.5us=2Mbit
```

### 2.8.2 Transmitter delay

Transmitter delay is a propagation delay of signal transmited by CTU CAN FD on CAN_TX output, back to CAN_RX input as is visualized in Figure 2.3. This delay involves propagation of signal to a physical layer transceiver, delay of transceiver itself, and delay from transceiver to CAN_RX input of CTU CAN FD. CTU CAN FD measures its own transmitter delay when it transmitts CAN FD frame (regardles of the fact if bit rate is switched in the frame) on recessive

to dominant edge between FDF (EDL), and r0 bits as is shown in Figure 2.4. Transmitter delay is readable after its measurement from TRV_DELAY register. Transmitter delay is measured in multiples of System clock period.

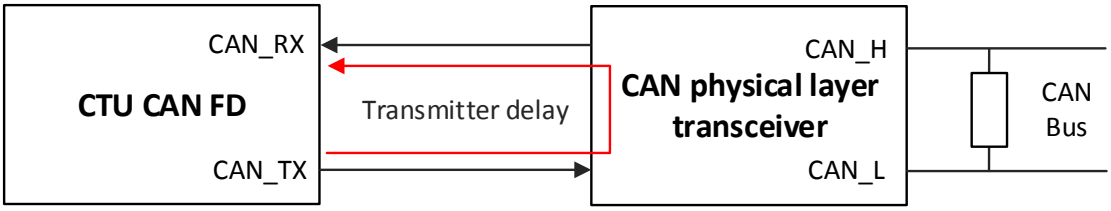


Figure 2.3: Transmitter delay

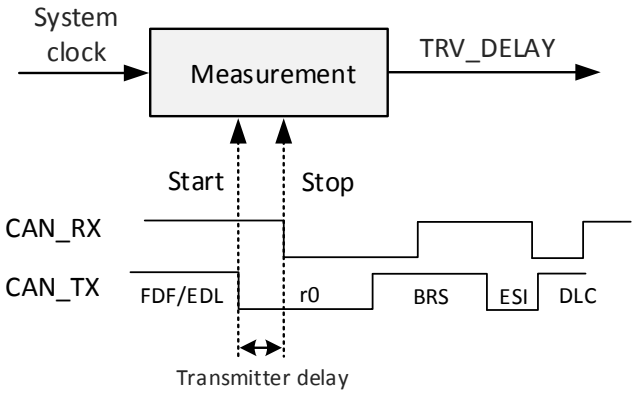


Figure 2.4: Transmitter delay measurement

**Note** Measured transmitter delay includes input delay of CTU CAN FD (which is 2 clock periods of System clock). Therefore, measured transmitter delay will be always higher by two than actual delay from CAN_TX to CAN_RX (e.g. if signal propagation from CAN_TX to CAN_RX takes 110 ns (11 System clock periods at 100 MHz), measured transmitter delay will be 13).

**Note** Transmitter delay measurement is saturated to 255 System clock periods. If delay between CAN_TX and CAN_RX is higher than 253 System clock periods, only 255 will be measured. With 100 MHz System clock frequency, the maximal measurable transciver delay is 2,53 us. This is more than most of CAN transceivers need.

### 2.8.3 Secondary sampling point

Secondary sampling point can be used by CTU CAN FD during data bit rate to detect bit errors. Its position is configured as a delay from start of bit (Sync_Seg) in multiples of System clock (not time quanta!). Secondary sampling point position can be fixed (SSP_CFG[SSP_OFFSET] only), derived from Transmitter delay (SSP_CFG[SSP_OFFSET] + TRV_DELAY), or it can be disabled (No SSP) as is shown in Figure 2.5. When Secondary sampling point is disabled, regular sampling point as configured by BRP_FD register is used by CTU CAN FD when transmitting in data bit rate.

**Note** Secondary sampling point offset (SSP_CFG[SSP_OFFSET]) is configurable between 0 - 255. Internal range of secondary sampling point position is 0 - 510 System clock periods.

**Note** Since CTU CAN FD input delay is 2 System clock periods (minimum time quanta), position of Secondary sampling point shall be configured to at least 2 to compensate its own input delay (if SSP_CFG[SSP_OFFSET] < 3 and

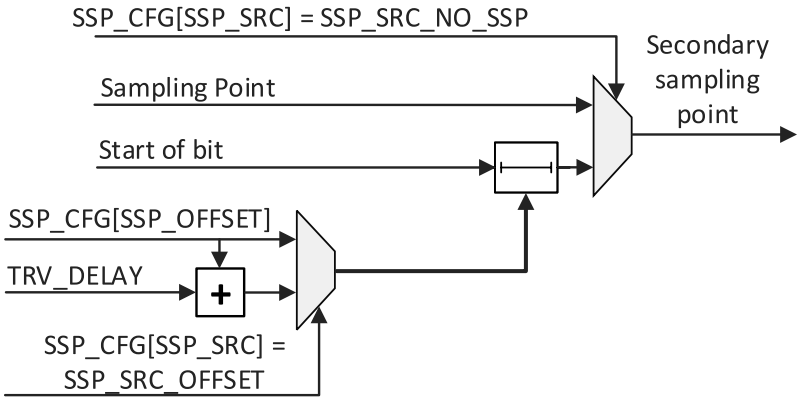


Figure 2.5: Secondary sampling point

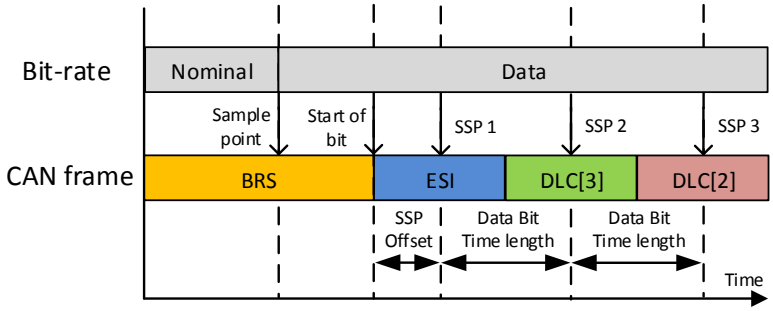


Figure 2.6: Secondary sampling point 2

SSP_CFG[SSP_SRC] = SSP_SRC_OFFSET], it is impossible to transmit CAN FD frames without detecting bit error in CTU CAN FDs own transmitted frame).

**Note** CTU CAN FD can handle at most 7 "bits in flight" between CAN_TX and CAN_RX pins when using secondary sampling point. E.g. if System clock is 100 MHz, and Data bit rate = 5 Mbit/s, then one data bit time is 20 System clock periods. Then, latest possible position of Secondary sampling point is 20 * 7 = 140 System clock periods. This limitation applies to final position of secondary sampling point (with SSP_CFG[SSP_OFFSET] / TRV_DELAY included). User shall not configure secondary sample point position later than 6 data bit times.

### 2.8.4 CAN FD support

CTU CAN FD supports both ISO, and non-ISO versions of CAN FD protocol. When ISO protocol version is chosen, CTU CAN FD is compliant to ISO11898-1 2015. When NON ISO version is chosen, CTU CAN FD is compliant to CAN FD specification 1.0. To choose between ISO and non-ISO variants, configure SETTINGS[NISOFD] bit. SETTINGS[NISOFD] shall be modified only when CTU CAN FD is disabled (SETTINGS[ENA] = 0).

### 2.8.5 Protocol exception handling

CTU CAN FD supports Protocol exception detection. Protocol exception is enabled by MODE[PEX] = 1. MODE[PEX] shall be changed only when CTU CAN FD is disabled (SETTINGS[ENA]=0). Protocol exception behavior differs for various CAN implementation types (see 2.1). If MODE[PEX] = 1 and CTU CAN FD detects Protocol exception, CTU

CAN FD enters bus integration state, and waits for 11 consecutive recessive bits to be monitored on CAN_RX signal. REC/TEC counters are not changed upon Protocol exception, nor is Fault confinement state of CTU CAN FD. When Protocol exception occurs, STATUS[PEXS] flag is set. To clear STATUS[PEXS], SW shall write COMMAND[CPEXS] = 1. If MODE[PEX] = 0, and conditions for Protocol exception are valid, CTU CAN FD transmits error frame instead.

### 2.8.6 Implementation type

ISO11898-1 2015 defines three implementation types of CAN protocol: Classical CAN, CAN FD tolerant and CAN FD enabled. CTU CAN FD supports all three implementation types, Compliance to each implementation type can be changed via MODE[FDE] and SETTINGS[PEX] bits. Both of these bits shall be modified only when CTU CAN FD is disabled (SETTINGS[ENA] = 0).

| Implementation type | MODE[FDE] | SETTING [PEX] | Behavior |
|---|---|---|---|
| Classical CAN | 0 | 0 | When CTU CAN FD detects recessive FDF bit (bit after IDE in Base frame, bit after RTR/r1 in Extended frame), it responds with error frame. |
| CAN FD tolerant | 0 | 1 | When CTU CAN FD detects recessive FDF bit, it detects Protocol exception and enters bus integration state. |
| CAN FD enabled | 1 | 0 | CTU CAN FD is able to receive / transmit CAN FD frames. When CTU CAN FD detects recessive value on position of "res" bit (one bit after FDF bit), it responds with error frame. |
| CAN FD enabled - with protocol exception | 1 | 1 | CTU CAN FD is able to receive / transmit CAN FD frames. When CTU CAN FD detects recessive value on position of "res" bit (one bit after FDF bit), it detects Protocol exception and enters bus integration state. This configuration tolerates future extensions of CAN FD protocol (e.g. CAN XL). |

Table 2.1: CAN implementation type

**Note** When CTU CAN FD is configured as Classical CAN / CAN FD tolerant node (MODE[FDE] = 0), and user attempts to send CAN FD frame (FRAME_FORMAT_W[FDF_BIT] = 1 in TXT buffer), CTU CAN FD will ignore frame type in TXT buffer , and send CAN 2.0 frame.

**Note** When CTU CAN FD is configured as Classical CAN / CAN FD tolerant node, SETTINGS[NISOFD] bit has no effect.

**Note** According to 10.9.10 of ISO11898-1 2015, CAN FD Enabled implementation shall not be set to a mode where it behaves as CAN FD tolerant implementation. It is therefore users responsibility to use this option only for evaluation / debugging purposes.

**Note** According to CAN 2.0 specification, R0 and R1 bits of any value shall be accepted by receivers, however ISO119898-1 2015 states (Table A.1) that Error frames shall be sent by Classical CAN implementation upon such event. CTU CAN FD resolves this inconsisency in CAN specifications in favor of ISO11898-1 2015.

### 2.8.7 Minimum bit time / Maximal bit rate

System clock period is equal to minimal time quanta, therefore it affects minimum bit rate achievable on CAN bus. CTU CAN FD has following limitations:

- Phase_Seg2 >= 2 minimal time quanta. This is valid for both nominal and data bit rate.
- Sync_Seg + Prop_Seg + Phase_Seg1 > 2 minimal time quanta. This is valid for both nominal and data bit rate.

With these conditions, it is possible to reach bit length of 5 time quanta. Note that for nominal bit rate this is possible, however, at least 8 time quantas per bit time are recommended (see 1.1). For data bit rate, 5 time quantas per bit time can be used.

As an example, when nominal bit rate is 250 Kbit/s, data bit rate is 1 Mbit/s, minimal possible System clock frequency is 5 MHz. Note that this is absolute maximum bit-rate, and gives very little margin in sample point position. Therefore it is recommended to use at least 10 MHz System clock in such case.

## 2.9 CAN frame transmission

CTU CAN FD transmits CAN frames from TXT buffers. CTU CAN FD contains 2-8 TXT buffers (number of TXT buffers is selected at synthesis time). To get actual number of TXT Buffers, SW can read TXTB_INFO register. If "N" TXT buffers are present, then its always buffers 1 - "N". Each TXT buffer can be in one of states as described in Figure 2.7. TX_STATUS register reflects state of TXT Buffers. To control TXT buffer state, SW issues commands to TX_COMMAND register. There are three types of commands:

**Set ready** requests TXT buffer to move to "Ready" state.

**Set abort** requests TXT buffer to move to "Aborted" or "Abort in progress" state.

**Set empty** requests TXT buffer to move to "Empty" state.

Each TXT buffer stores single CAN frame. A 64 byte CAN FD frame fits to single TXT buffer. TXT buffer is write only (CAN frame can't be read back). TXT buffer is accessible only when the buffer is in "Empty", "TX OK", "TX failed", "Aborted" or "Parity Error" states. To store CAN frame to a TXT buffer, SW writes to TXT Buffer 1 - TXT Buffer 8 memory regions described in Section 3.

First SW driver stores CAN frame to a TXT buffer, and then issues **Set ready** command to the TXT buffer to request transmission of CAN frame. TXT buffer moves to "Ready" state, and CTU CAN FD can transmit frame from this TXT buffer. When CTU CAN FD starts transmission, the TXT buffer moves to "TX in progress" state. CTU CAN FD starts transmission from TXT buffer in "Ready" state if it sampled dominant bit during third bit of intermission, or as soon as CAN bus is idle. Note that in Time triggered transmission mode, the behavior differs (see 2.9.2).

When CTU CAN FD is error-passive, and it was transmitter of previous frame, it suspend consecutive transmission for 8 bit times. When CTU CAN FD transmitted CAN frame succesfully (no arbitration lost, no error frame), TXT buffer moves to "TX OK" state. If an error frame occurs, or arbitration is lost, TXT buffer moves to "Ready" state and CTU CAN FD attempts to transmitt again in nearest intermission or bus idle.

**Note** When CTU CAN FD operates in Bus monitoring mode (MODE[BMM] = 1), or Restricted operation mode (MODE[ROM] = 1) it always ends up in "TX failed" state when **Set ready** command is issued, without any attempt to transmit the frame.

### 2.9.1 TXT buffer selection

When multiple TXT buffers are in "Ready" state, CTU CAN FD selects highest priority TXT buffer in "Ready" state, and transmitts CAN frame from this TXT buffer. SW configures priority of TXT buffers in TX_PRIORITY register. If two TXT buffers have equal priority, TXT buffer with lower index has precedence. The overall flow of transmission is shown in Figure 2.8.

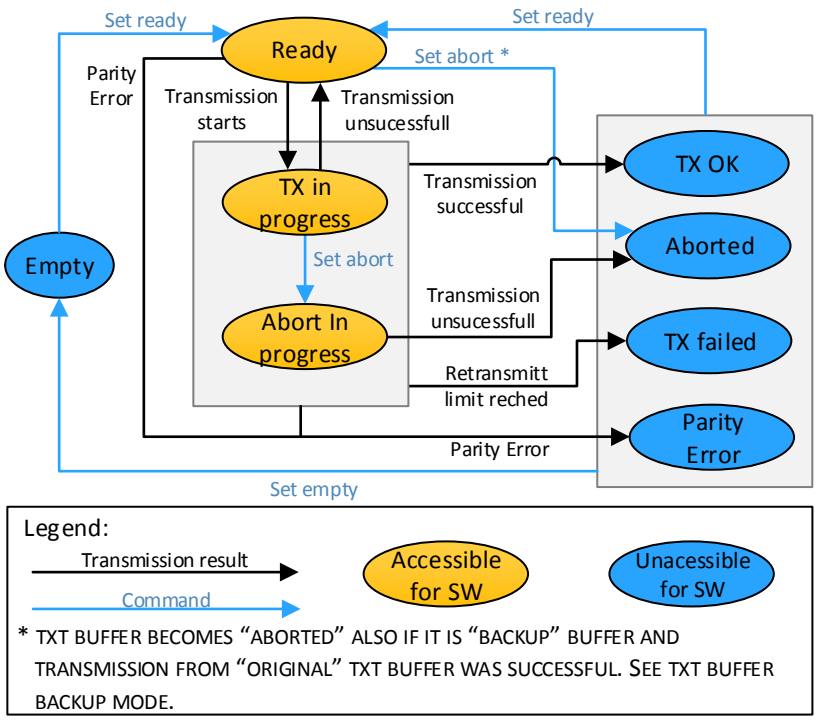


Figure 2.7: TXT buffer states

**Note** Higher value of TX_PRIORITY[TX*P] means TXT Buffer * has higher priority (e.g. if TX_PRIORITY[TX1P] = 2 and TX_PRIORITY[TX2P]=5, then TXT Buffer 2 has priority 5, and TXT Buffer 1 has priority 2. When both TXT Buffers are in ready state, CTU CAN FD will pick TXT Buffer 2 before TXT Buffer 1).

**Note** Priority of "backup" TXT Buffers when MODE[TXBBM] = 1 is not configurable by a TX_PRIORITY[TX*P] corresponding to them, but it is configured by a bit corresponding to "original" TXT Buffer. See 2.13.3.

### 2.9.2 Time triggered transmission mode

CTU CAN FD supports time-triggered transmission mode. To enable this mode, set MODE[TTTM] = 1. In time-triggered transmission mode, CTU CAN FD will attempt to transmitt frame from highest priority TXT buffer only when value of Time-Base (see 2.4) reaches Timestamp stored in TIMESTAMP_L_W and TIMESTAMP_U_W words of this TXT Buffer. CTU CAN FD assumes that Time base is an up-counting unsigned counter. When Time base reaches value stored in TIMESTAMP_L_W and TIMESTAMP_U_W, frame stored in TXT buffer is allowed for transmission (assuming that it is in highest priority TXT buffer in "Ready" state), as is visualized in Figure 2.9. CTU CAN FD will not transmit the frame immediately, it will transmit the frame when the CAN bus is free. If TXT buffer is in "Ready" state, and Time base counter did not reach moment of transmission yet, CTU CAN FD waits until this condition is satisfied. If during this time another node on CAN bus starts transmitting a frame, CTU CAN FD becomes receiver of such frame.

To transmitt CAN frame as soon as possible (no time triggered transmission), SW driver shall store 0x00000000 to TIMESTAMP_L_W, TIMESTAMP_U_W words. Note that time triggered transmission is always considered only from highest priority TXT buffer in "Ready" state. TXT buffer priority is always evaluated first before time triggered transmission. The behavior of the TXT buffer priority and time triggered transmission is following:

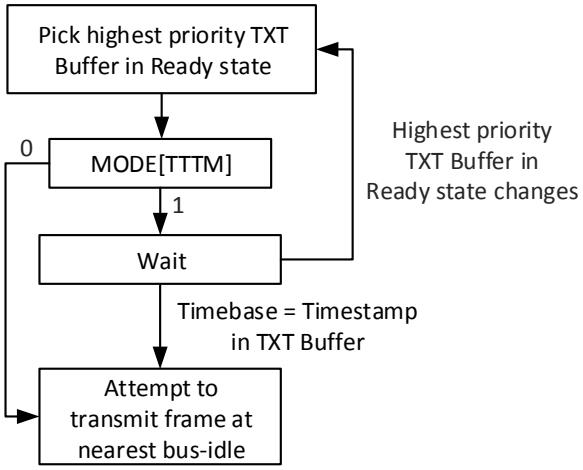


Figure 2.8: TXT Buffer selection

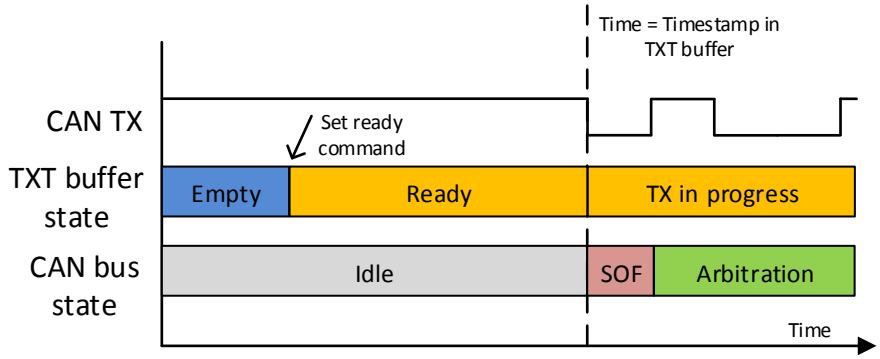


Figure 2.9: Time triggered transmission

- If TXT buffer A has higher priority than TXT buffer B, CTU CAN FD will pick frame from TXT buffer A even if its time of transmission is higher (transmission should start later) than the one from TXT Buffer B.
- If priority of TXT buffers changes (and highest priority TXT buffer in "Ready" state changes), then CTU CAN FD picks frame from new highest priority TXT buffer in "Ready" state. This is valid as long as frame from previously selected TXT buffer is waiting for Time base to reach its time of transmission. When frame transmission already starts, TXT buffer priority is not considered anymore (no frame swapping).

### 2.9.3 Type of transmitted CAN frame

SW chooses type of transmitted CAN frame by setting the value of FRAME_FORMAT_W in TXT buffer, and settings of CTU CAN FD as show in Figure 2.10.

**Note** When FRAME_FORMAT_W[FDF] = FD_CAN and MODE[FDE] = 0, CTU CAN FD transmits CAN 2.0 frame. If in such case TXT buffer contains CAN FD frame with more than 8 bytes of data payload, bytes above 8-th byte will not be sent.

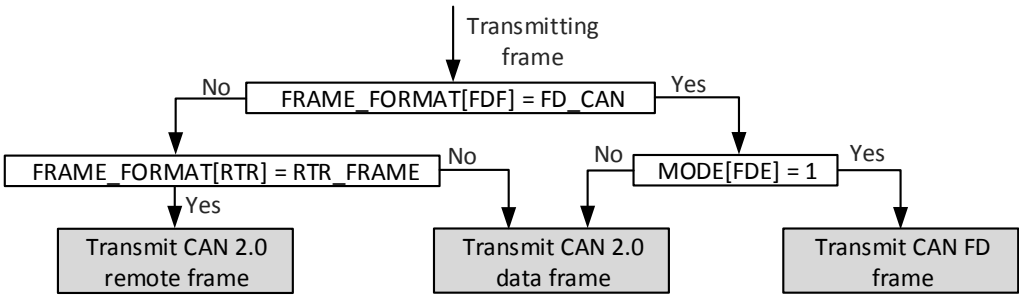


Figure 2.10: TX frame type

**Note** When FRAME_FORMAT_W[RTR] = RTR_FRAME and FRAME_FORMAT_W[FDF] = FD_CAN, CTU CAN FD ignores RTR flag and transmits CAN FD data frame (there are no remote frames in CAN FD protocol).

### 2.9.4 Retransmitt limitation

CTU CAN FD can limit number of retransmissions from single TXT buffer. Retransmitt limitation is enabled when SETTINGS[RTRLE] = 1. Number of retransmissions is configured in SETTINGS[RTRTH]. First attempt to transmitt CAN frame does not count as retransmission. Possible configuration options are shown in Table 2.2.

| SETTINGS [RTRTH] | SETTINGS [RTRLE] | Behaviour |
|---|---|---|
| - | 0 | Frame transmission is attempted without any limitation (until it is succesfull or unit turns bus-off). |
| 0 | 1 | Frame transmission is attempted only once, there are no retransmission attempts after first failed transmission (so called one shot mode). |
| 1 - 15 | 1 | Frame transmission is attempted SETTINGS[RTRTH] + 1 times (initial transmission + SETTINGS[RTRTH] retransmissions). |

Table 2.2: Retransmitt limitation configuration

If SETTINGS[RTRTH] consecutive retransmission are not succesfful (error frame occured or arbitration was lost) from single TXT buffer, this TXT buffer moves to "TX failed" state. If TXT buffer used for transmission changed between two transmissions (e.g it was picked due to higher priority), internal counter of retransmissions is erased, and new frame (from new TXT buffer) has again SETTINGS[RTRTH]+1 transmission attempts. If CTU CAN FD returns to transmission from original TXT buffer, it does not remember previous number of transmission attempts and again attempts to transmitt CAN frame SETTINGS[RTRTH]+1 times. Current number of transmission attempts of a single frame is held in an internal counter which is readable via RETR_CTR register.

### 2.9.5 Abort

If SW driver previously requested transmission of CAN frame by **Set ready** command, it can request abort of transmission by **Set abort** command. If TXT buffer is still in "Ready" state when it receives **Set abort** command (transmission did not start yet), it moves to "Aborted" state immediately. If TXT buffer is in "TX in progress" state (transmission has already started), it moves to "Abort in progress" state. Then the TXT Buffer will move to "Aborted" state upon nearest error frame or arbitration lost. Note that when TXT buffer is in "Abort in progress" state, it can move to TX OK state if current transmission succeeds, or to "TX failed state" if retransmitt limit was reached.

### 2.9.6 TXT buffer - Bus-off behavior

When CTU CAN FD becomes bus-off due to TEC > 255, TXT buffers can react to this event in two ways:

1. All TXT buffers which are in "Ready", "TX in Progress" or "Abort in Progress" immediately go to "TX failed" state. This option is enabled by setting SETTINGS[TBFBO] = 1, and it is default configuration of TXT buffers.
2. TXT buffer which was used for transmission at time when CTU CAN FD became bus-off, will behave as if any other error frame was transmitted. This option is enabled by setting SETTINGS[TBFBO] = 0. If no "Set abort" command was issued to this buffer, nor retransmitt limit was reached, the buffer will become "Ready". When CTU CAN FD finishes reintegration (see 2.5), transmission from this TXT buffer will begin as per regular TXT buffer selection by priority. This option allows going bus-off and re-integrating without the need of SW interaction with TXT buffers.

### 2.9.7 Sample code

```
#define CTU_CAN_FD_BASE 0x12000000
#define TX_COMMAND_ADDR (CTU_CAN_FD_BASE + 0x74)
#define TXT_BUFFER_1_BASE (CTU_CAN_FD_BASE + 0x100)

/* Insert CAN frame to TXT buffer 1 */
uint32_t frame_format_word = 0;
frame_format_word |= 4;                                 // DLC = 4
frame_format_word |= (1 << 7);                          // CAN FD Frame
frame_format_word |= (1 << 9);                          // Switch bit-rate
can_write_32(TXT_BUFFER_1_BASE, frame_format_word);     // Store frame format word

uint32_t id_word = (55 << 18);                          // Identifier: 55
can_write_32(TXT_BUFFER_1_BASE + 0x4, id_word);           // Store identifier word
can_write_32(TXT_BUFFER_1_BASE + 0x8, 1000);
can_write_32(TXT_BUFFER_1_BASE + 0xC, 0);                 // Transmitt at time 1000
can_write_32(TXT_BUFFER_1_BASE + 0x10, 0xAABBCCDD);       // Data: 0xAA 0xBB 0xCC 0xDD

/* Issue Set ready command */
uint32_t command = 0;
command |= 0x2;                                         // Set Ready command
command |= (1 << 8);                                    // Choose TXT Buffer 1
can_write_32(TX_COMMAND_ADDR, command);                 // Issue the command
```

**Note** When CTU CAN FD is enabled by writing SETTINGS[ENA] = 1, it is still bus-off during integration to the CAN bus. If during this time **Set ready** command is issued to TXT buffer, TXT buffer immediately moves to "Aborted" state when SETTINGS[TBFBO] = 1. SW shall wait until node is Error active (either polling FAULT_STATE or via FCS Interrupt) before issuing **Set ready** command to any TXT buffer.

**Note** TXT buffers are not initialized, nor reset. Therefore, before issuing **Set ready** command, SW shall fill according TXT buffer with valid CAN frame for transmission.

**Note** CTU CAN FD transmitts only reactive Overload frames. There are no internal conditions of CTU CAN FD which would cause transmission of Overload frame without detecting overload condition.

## 2.10 CAN frame reception

CTU CAN FD contains single FIFO-like RX buffer for received CAN frames. Size of the RX buffer is multiple of 32-bit words. SW can read the size of RX buffer from RX_MEM_INFO register. RX buffer stores two types of CAN frames:

- Regular frame - CTU CAN FD stores regular CAN frame to RX buffer when a CAN frame is received without error frames on CAN bus. Regular CAN frames contain all the data exchanged in the CAN communication. Regular frames include Remote Transmission Request frames.
- Error frame - CTU CAN FD stores error frame that occured on CAN bus to the RX buffer.

CAN frames (regular or error) are read by SW from RX buffer by consecutive reads from RX_DATA register. Single read from RX_DATA register reads one word from RX buffer. RX buffer can operate in one of two modes:

- Automatic mode - When SW reads RX_DATA register, CTU CAN FD automatically increments read pointer of RX buffer FIFO. Use this mode only when RX_DATA is read by 32-bit accesses. Writes to COMMAND[RXRPMV] = 1 have no effect in this mode.
- Manual mode - When SW reads RX_DATA register, CTU CAN FD does not increment read pointer of RX buffer FIFO. To increment read pointer, SW shall write COMMAND[RXRPMV] = 1. Use this mode when CTU CAN FD is mapped to a memory space where only 8/16 bit accesses are possible.

Mode of RX buffer is configured by MODE[RXBAM] bit. Section 3 describes CAN frame format in RX buffer and Figure 2.11 shows this format. CAN frame in RX buffer spans from 4 to 20 memory words. Its size is given as:

$$\text{Size of RX frame in words} = 4 + \text{ceil}(\text{Data field length} \ / \ 4)$$

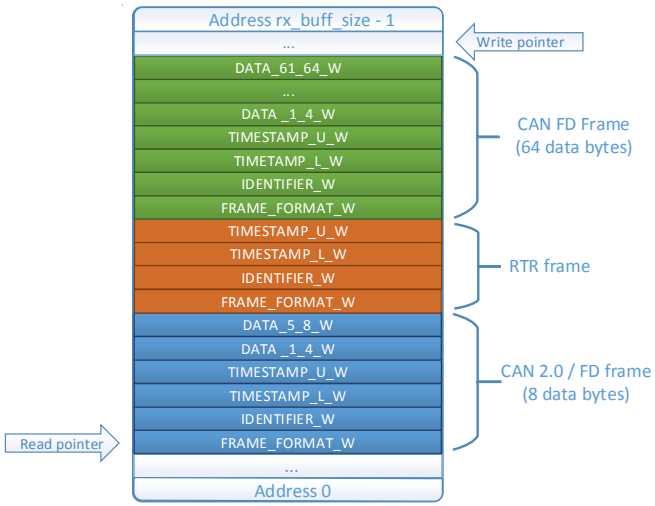


Figure 2.11: RX buffer

### 2.10.1 Frame count

CTU CAN FD contains a counter of CAN frames within the RX buffer. To get value of this counter, SW shall read RX_STATUS[RXFRCE] register. CTU CAN FD:

- Increments the counter when it stores a frame to RX buffer (regular or error frame)
- Decremented the counter when SW reads a last word of CAN frame from RX buffer.

### 2.10.2 Error frame reception

When SW sets MODE[ERFM]=1, CTU CAN FD stores Error frames that occur on CAN bus in the RX buffer. If MODE[ERFM]=0, only regular CAN frames are stored to RX buffer. Each error condition that caused transmission of a new Error frame is logged as separate Error frame in the RX buffer. Each Error frame in the RX buffer occupies exactly 4 memory words (FRAME_FORMAT_W, IDENTIFIER_W, TIMESTAMP_U_W and TIMESTAMP_L_W).

The error frames are distinguished from regular CAN frames by FRAME_FORMAT_W[ERF] bit. In Error frames, FRAME_FORMAT_W[ERF] = ERF_ERR_FRAME, while in regular CAN frames FRAME_FORMAT_W[ERF] = ERF_CAN_FRAME. The error frames have following differences from regular CAN frames:

- Only FRAME_FORMAT_W[ERF], FRAME_FORMAT_W[IVLD], FRAME_FORMAT_W[RWCNT], FRAME_FORMAT_W[ERF_POS] and FRAME_FORMAT_W[ERF_TYPE] are valid in FRAME_FORMAT_W. Other fields of FRAME_FORMAT_W are reserved, and shall be ignored by SW reading the Error frame from RX buffer.
- When FRAME_FORMAT_W[IVLD]=1, IDENTIFIER_W contains valid CAN identifier. Otherwise, IDENTIFIER_W value contains all zeroes.
- TIMESTAMP_U_W and TIMESTAMP_L_W contain CTU CAN FD timebase value at the moment when when error condition causing the Error frame transmission occured.

**Note** When CTU CAN FD operates in Restricted operation mode (MODE[ROM]=1), Error frames are not stored to RX buffer since error condition on the bus causes CTU CAN FD to move to Bus Integration state, and not transmit the Error frame.

**Note** Rules related to setting of ERR_CAPT (Error type priorities, etc...) apply also to Error frames stored in RX buffer.

**Note** In regular CAN frame, FRAME_FORMAT_W[IVLD] is always 1. If FRAME_FORMAT_W[IVLD]=1 in Error frame, then the IDENTIFIER_W contains a valid CAN identifier. When FRAME_FORMAT_W[IVLD] = 1, then IDENTIFIER_W can be used to track the CAN frame that lead to Error frame.

**Note** The error frames ocurring in CAN frames with Base identifier before IDE bit have FRAME_FORMAT_W[IVLD]=0. The error frames ocurring in CAN frames with Extended identifier before RTR/RRS bit have FRAME_FORMAT_W[IVLD]=0.

**Note** When CTU CAN FD stores error frame with FRAME_FORMAT_W[IVLD]=0 to RX Buffer, every such error frame is stored regardless of frame filters configuration. Therefore, invalid identifiers are not subject to frame filtration.

### 2.10.3 RX buffer memory

RX buffer memory provides following status information:

- Number of free memory words in RX_MEM_INFO[RX_MEM_FREE].
- Write pointer position in RX_POINTERS[RX_WPP].
- Read pointer position in RX_POINTERS[RX_RPP].

### 2.10.4 RX buffer status

RX buffer with no stored CAN frames is empty. When RX buffer is empty, then RX_STATUS[RXE]=1. RX buffer with all memory words occupied by CAN frames is full. When RX buffer is full, RX_STATUS[RXF]=1.

**Note** If RX buffer has e.g. 2 free memory words it is not full, however even smallest CAN frame does not fit into the RX buffer (smallest CAN frame takes 4 memory words).

### 2.10.5 Overrun

Overrun occurs when there is not enough free space in RX buffer during reception of CAN frame. Upon overrun, CTU CAN FD drops currently received frame (RX buffer FIFO overflows), and sets Overrun flag. Overrun flag is sticky (it remains set until SW clears it). SW reads Overrun flag from STATUS[DOR]. SW clears Overrun flag by writing COMMAND[CDO]=1.

### 2.10.6 Flush

To flush RX buffer, SW shall write COMMAND[RRB]=1. The flush of RX buffer has following effect:

- Content of RX buffer is kept (memory is not erased)
- Read and write pointers become 0
- Frame counter becomes 0.

After flush RX buffer is as-if there are no frames in it. If SW issues flush during CAN frame reception, currently received frame is also droped.

### 2.10.7 Inconsistency protection

Reading CAN frame from RX buffer involves multiple reads of RX_DATA register. Each read increments read pointer inside RX buffer (read operation with side effect). If an error occurs (e.g. bus error, ECC error) during read from RX_DATA register, then read data could be lost. SW driver now has two problems:

- CTU CAN FD incremented RX buffer read pointer. SW driver lost the word, therefore it can't read the frame correctly.
- SW driver may have lost track on what part of the frame was read from RX buffer. E.g. if an error ocurred during read of FRAME_FORMAT_W, the SW driver does not know how many words does the remaining CAN frame contains.

SW driver can read RX_STATUS[RXMOF] to recover from such state. When next read from RX_DATA register is about to return FRAME_FORMAT_W (beginning of new frame), RX_STATUS[RXMOF] = 0. Otherwise RX_STATUS[RXMOF] = 1 (RX buffer read pointer points to middle of frame). If SW driver gets into inconsistent state during readout of frame, it shall repetitively read from RX_DATA until RX_STATUS[RX_MOF] = 0. Upon such condition, RX_DATA points to FRAME_FORMAT_W word of new frame, or RX buffer is empty (if the error occured during readout of only frame in RX buffer).

### 2.10.8 Timestamping

When CTU CAN FD receives CAN frame, it stores its timestamp in TIMESTAMP_L_W, TIMESTAMP_U_W words within RX buffer. CTU CAN FD samples thevalue of external Time Base to obtain the timestamp of CAN frame. CTU CAN FD samples the Timestamp of received frame in:

- Sample point of Start of Frame bit. This mode is configured by RX_SETTINGS[RTSOP]=1
- 6th bit of End of Frame (moment when received CAN frame is considered valid according to ISO11898-1 2015). This mode is configured by RX_SETTINGS[RTSOP]=0.

### 2.10.9 Frame filtering

CTU CAN FD filters received CAN frames by HW filters. There are two types of filters in CTU CAN FD: Bit filter and Range filter. There are three instances of Bit filter (A,B,C) and one instance of Range filter. If received CAN frame passes at least one filter, CTU CAN FD stores the frame to RX buffer. CTU CAN FD filters the received frames only if Acceptance filter mode is enabled (MODE[AFM] = 1). SW shall modify MODE[AFM] only when CTU CAN FD is disabled (SETTINGS[ENA] = 0). When Acceptance filter mode is disabled, CTU CAN FD stores every received CAN frame to RX buffer.

SW can configure each filter to accept only certain types of:

- CAN frame types (CAN 2.0 frame / CAN FD frame)
- Identifier types (frame with Base identifier only, frame with Base + Extended identifier).

SW can configure this behavior in FILTER_CONTROL register. SW disables filter by setting all bits in FILTER_CONTROL register belonging to this filter to 0. Figure 2.12 describes Frame filters operation.

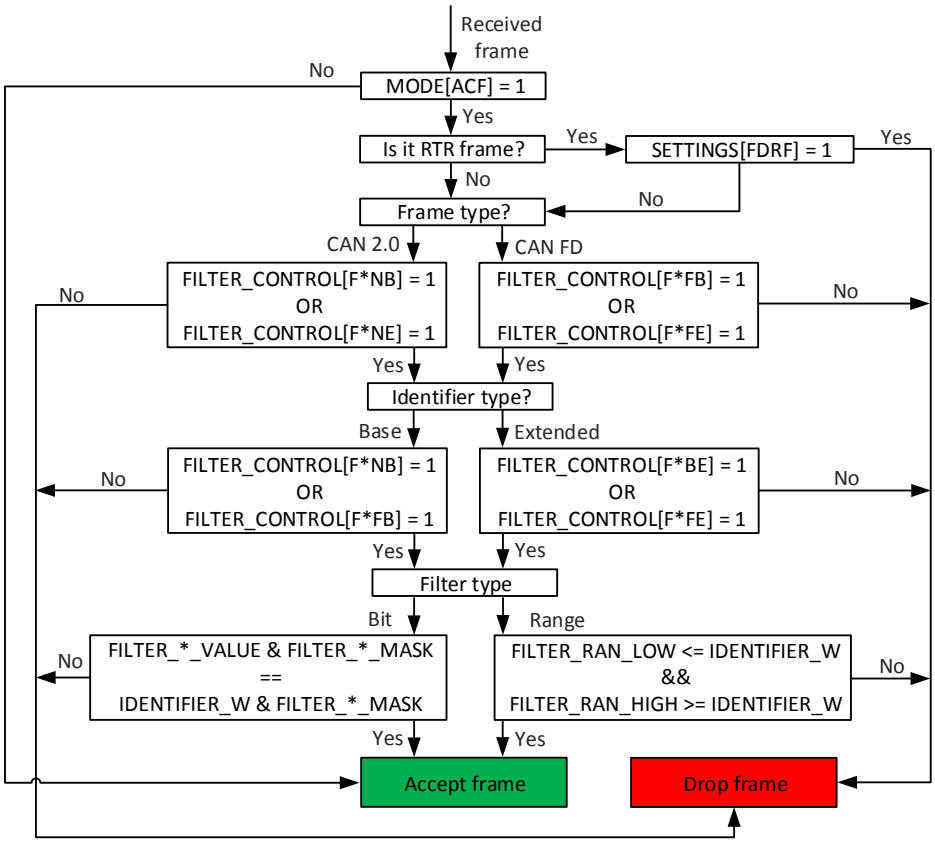


Figure 2.12: Frame filters operation (* stands for A/B/C/R based on filter type)

**Bit filter**

Bit filter checks if received CAN frame identifier is equal to predefined identifier in FILTER_X_VALUE register (X=A,B,C based on filter instance). Only bits given by filter mask in FILTER_X_MASK register are compared.

**Note** When using Bit filter to filter frames with Base identifiers only, set FILTER_X_MASK[17:0] = 0b00000000000000000.

**Range filter**

Range filter checks if received CAN frame identifier is within FILTER_RAN_LOW to FILTER_RAN_HIGH decimal range.

**Note** When using Range filter to filter frames with Base identifiers only, set FILTER_RAN_LOW[17:0] = 0b00000000000000000 and FILTER_RAN_HIGH[17:0] = 0b11111111111111111.

### 2.10.10 Sample code 1 - Frame reception in automatic mode (32-bit access)

```
#define CTU_CAN_FD_BASE 0x12000000
#define RX_DATA_ADDR (CTU_CAN_FD_BASE + 0x6C)
#define RX_STATUS_ADDR (CTU_CAN_FD_BASE + 0x68)
```

```
/* Poll on RX buffer until there is a frame in it */
uint32_t rx_status;
do {
    rx_status = can_read_32(RX_STATUS_ADDR);
} while ((rx_status & 0x1) == 0)

/* Read frame from RX buffer */
uint8_t data[64];
uint32_t tmp;
uint32_t ffw = can_read_32(RX_DATA_ADDR);
uint32_t id = can_read_32(RX_DATA_ADDR);
uint32_t ts_l = can_read_32(RX_DATA_ADDR);
uint32_t ts_h = can_read_32(RX_DATA_ADDR);

uint32_t rwcnt = (ffw >> 11) & 0x1F;
for(int i = 0; i < rwcnt - 3; i++){
    tmp = can_read_32(RX_DATA_ADDR);
    data[i*4] = tmp & 0xFF;
    data[i*4+1] = (tmp >> 8) & 0xFF;
    data[i*4+2] = (tmp >> 16) & 0xFF;
    data[i*4+3] = (tmp >> 24) & 0xFF;
}
```

### 2.10.11 Sample code 2 - Frame reception in manual mode (8-bit access)

```
#define CTU_CAN_FD_BASE 0x12000000
#define RX_DATA_ADDR (CTU_CAN_FD_BASE + 0x6C)
#define RX_STATUS_ADDR (CTU_CAN_FD_BASE + 0x68)
#define COMMAND_ADDR (CTU_CAN_FD_BASE + 0xC)
#define MOVE_RX_BUF_READ_PTR() can_write_8(COMMAND_ADDR, 1 << 2)

/* Poll on RX buffer until there is a frame in it */
uint8_t rx_status;
do {
    rx_status = can_read_8(RX_STATUS_ADDR);
} while ((rx_status & 0x1) == 0)

/* Read frame format word and move to RX pointer */
uint8_t data[64];
uint16_t ffw = (uint16_t)can_read_8(RX_DATA_ADDR);
ffw |= (((uint16_t)can_read_8(RX_DATA_ADDR + 0x1)) << 8);
MOVE_RX_BUF_READ_PTR();

/* Read CAN identifier and move RX pointer up to first data word */
uint32_t id = (uint32_t)can_read_8(RX_DATA_ADDR);
id |= ((uint32_t)can_read_8(RX_DATA_ADDR + 0x1)) << 8;
id |= ((uint32_t)can_read_8(RX_DATA_ADDR + 0x2)) << 16;
id |= ((uint32_t)can_read_8(RX_DATA_ADDR + 0x3)) << 24;
```

```
for (int i = 0; i < 3; i++)
    MOVE_RX_BUF_READ_PTR();

/* Read data bytes */
uint16_t rwcnt = (ffw >> 11) & 0x1F;
for(int i = 0; i < rwcnt - 3; i++){
    data[i*4] = can_read_8(RX_DATA_ADDR);
    data[i*4+1] = can_read_8(RX_DATA_ADDR + 0x1);
    data[i*4+2] = can_read_8(RX_DATA_ADDR + 0x2);
    data[i*4+3] = can_read_8(RX_DATA_ADDR + 0x3);
    MOVE_RX_BUF_READ_PTR();
}
```

### 2.10.12 Sample code 3 - Bit filter configuration

```
#define CTU_CAN_FD_BASE 0x12000000
#define FILTER_CONTROL_ADDR (CTU_CAN_FD_BASE + 0x5C)
#define FILTER_A_VAL_ADDR (CTU_CAN_FD_BASE + 0x40)
#define FILTER_A_MASK_ADDR (CTU_CAN_FD_BASE + 0x3C)

uint32_t filter_mask = 0xF << 18; // Compare 4 LSBs of Base ID
uint32_t filter_val = 0x2 << 18; // Must be equal to 0x2 (0010)

/* Configure filter A */
can_write_32(FILTER_A_VAL_ADDR, filter_val);
can_write_32(FILTER_A_MASK_ADDR, filter_mask);

/* Enable reception of CAN 2.0 and CAN FD frames with Base identifiers only */
uint32_t filter_control = 0x5; // FANB, FAFB
can_write_32(FILTER_CONTROL_ADDR, filter_control);
```

## 2.11 Fault confinement

SW can read Fault confinement state of CTU CAN FD from FAULT_STATE register. Figure 2.13 shows the Fault confinement state transition diagram. CTU CAN FD shows Fault confinement counters in REC and TEC registers. These counters correspond to transmitt error counter, and receive error counter as defined in ISO11898-1. CTU CAN FD additionally contains counters distigushing between errors detected in nominal bit rate, and errors detected in data bit rate. To read Nominal bit rate error counter, SW shall read ERR_NORM register. To read Data bit rate error counter, SW shall read ERR_FD register. CTU CAN FD increments each error counters by 1 when it detects error in the respective bit rate.

When CTU CAN FD is in test mode (MODE[TSTM] = 1), SW can change all four counters (REC, TEC, ERR_NORM, ERR_FD). SW can set these counters via CTR_PRES register. Thresholds for Error warning limit, and transition to error passive are in EWL and ERP registers. By default, EWL and ERP corresponds to ISO11898-1. In test mode (MODE[TSTM] = 1), SW can change EWL and ERP registers.

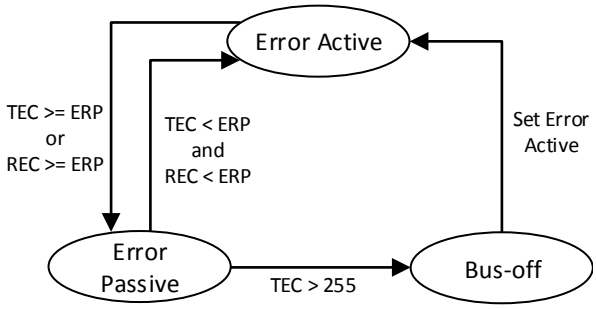


Figure 2.13: Fault confinement

## 2.12 Interrupts

CTU CAN FD generates interrupts upon various events. Each interrupt source has three parameters:

- Interrupt mask - Set by INT_MASK_SET, cleared by INT_MASK_CLR.
- Interrupt enable - Set by INT_ENA_SET, cleared by INT_ENA_CLR.
- Interrupt status - Set by HW upon event occurence, cleared by writing to INT_STAT.

Figure 2.14 shows the relationship between interrupt parameters. CTU CAN FD sets Interrupt status when a certain condition. To set the Interrupt status, its corresponding bit of Interrupt mask must be 0 (interrupt is unmasked). If Interrupt status is set, and corresponding interrupt is enabled, Interrupt is generated. Interrupt status can be read from CTU CAN FD via INT_STAT register. Note that when interrupt status is about to be set by HW at the same moment as it is being cleared by SW, interrupt remains set (set has priority over clear).

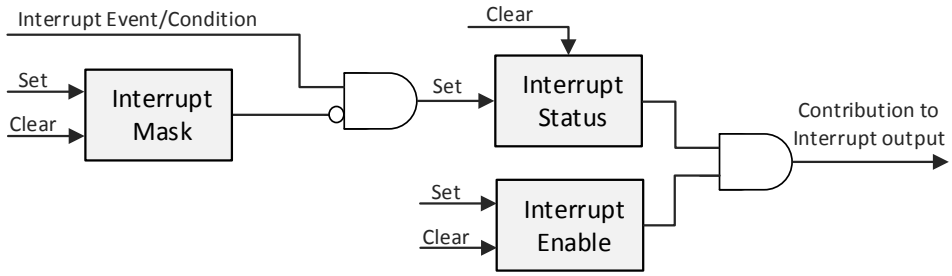


Figure 2.14: Interrupts

### 2.12.1 Frame transmission and reception

When CTU CAN FD transmitts CAN frame succesfully (no error frame until the end of EOF), it generates TX interrupt (INT_STAT[TXI]). When CTU CAN FD receives CAN frame successfully (no error frame until one bit before the end of EOF), it generates RX interrupt (INT_STAT[RXI]).

### 2.12.2 Fault confinement

When Transmitt error counter (TEC), or Receive error counter (REC) reach value in EWL register, CTU CAN FD generates Error warning limit interrupt (INT_STAT[EWLI]). When Fault confinement state changes, CTU CAN FD

generates Fault confinement state interrupt (INT_STAT[FCSI]). CTU CAN FD sets INT_STAT[FCSI] upon any Fault confinement state change (even bus-off to error-active).

### 2.12.3 TXT buffers and RX buffer

When Overrun occurs on RX buffer, CTU CAN FD generates data overrun interrupt (INT_STAT[DOI]). When RX buffer is full, CTU CAN FD generates RX buffer full interrupt (INT_STAT[RXFI]). If RX buffer is still full after INT_STAT[RXFI] was cleared, CTU CAN FD generates the interrupt again. When there is at least one CAN frame stored in RX buffer, CTU CAN FD generates RX buffer not empty interrupt (INT_STAT[RBNEI]). When any TXT buffer moves from "Ready", "TX in progress" or "Abort in progress" states to any of "TX OK", "Aborted", "TX failed" or "Parity Err" states, CTU CAN FD generates TXT buffer HW change interrupt (INT_STAT[TXBHCI]).

**Note** The INT_STAT[TXBHCI] has a corner-case. When a TXT Buffer is in "Ready" state, and moves to "Aborted" (due to SW writing TX_COMMAND[TXCA]=1), CTU CAN FD does not set INT_STAT[TXBHCI]. The meaning of INT_STAT[TXBHCI] is following: CTU CAN FD sets INT_STAT[TXBHCI] when TXT Buffer changed its state due to a CAN protocol core event (error frame occured on the bus, frame transmission finished, etc...). Since writing TX_COMMAND[TXCA]=1 is not a CAN protocol core event (it is an event caused by SW action), the INT_STAT[TXBHCI] is not set.

### 2.12.4 Error and Overload frame

When CTU CAN FD starts Error frame transmission, it generates Bus error interrupt (INT_STAT[BEI]). When CTU CAN FD transmits overload frame, it generates Overload frame interrupt (INT_STAT[OFI]).

### 2.12.5 Other

When CTU CAN FD switches bit rate on CAN bus, it generates Bit rate switch interrupt (INT_STAT[BSI]). When CTU CAN FD looses arbitration, it generates Arbitration lost interrupt (INT_STAT[ALI]).

## 2.13 Fault Tolerance

CTU CAN FD implements following fault tolerance mechanisms:

- Parity protection on RX buffer RAM
- Parity protection on TXT Buffer RAMs
- TXT Buffer Backup Mode (MODE[TXBBM]).

Following conditions must be met for these mechanisms to operate:

- CTU CAN FD must contain support for parity protection (STATUS[SPRT]=1). If STATUS[SPRT]=0, CTU CAN FD contains no parity protection (since it was not synthesized with it), and this section is not applicable.
- SW drivers sets SETTINGS[PCHKE] = 1. This bit enables parity error detection. SW shall modify SETTINGS[PCHKE] only when SETTINGS[ENA] = 0.

### 2.13.1 Parity protection on RX buffer RAM

When CTU CAN FD stores CAN frame to RX buffer RAM, it adds single parity bit to each word of RX buffer RAM. When SW driver reads the frame from RX buffer RAM, it can check if parity error occured in the frame by reading STATUS[RXPRE] bit. If parity bit in the word read from RX buffer RAM is not equal to calculated parity bit, CTU CAN FD sets STATUS[RXPRE]. CTU CAN FD sets STATUS[RXPRE] upon each read from RX_DATA register with parity error.

A single-event upset (SEU) in RX buffer RAM can potentially modify FRAME_FORMAT_W[DLC] word of a frame in RX buffer RAM. Therefore SEU may hamper the length of the RX frame as seen by SW driver, and thus get RX buffer into inconsistent state where SW driver has read only part of a received frame. In this situation, all further frames read from RX buffer would be corrupted. To avoid this situation, SW driver shall use following procedure when reading RX frames from RX buffer:

```
#define CTU_CAN_FD_BASE 0x12000000
#define STATUS_ADDR (CTU_CAN_FD_BASE + 0x8)
#define COMMAND_ADDR (CTU_CAN_FD_BASE + 0xC)
#define RX_STATUS_ADDR (CTU_CAN_FD_BASE + 0x68)
#define RX_DATA_ADDR (CTU_CAN_FD_BASE + 0x6C)

/* Read frame from RX buffer RAM, and check parity error.*/
uint8_t data[64];
uint32_t tmp;
uint32_t ffw = can_read_32(RX_DATA_ADDR);
uint32_t id = can_read_32(RX_DATA_ADDR);
uint32_t ts_l = can_read_32(RX_DATA_ADDR);
uint32_t ts_h = can_read_32(RX_DATA_ADDR);
/* If Parity error is in FRAME_FORMAT_W, RWCNT might be unreliable. */
if (can_read_32(STATUS_ADDR) >> 10) & 0x1)

    goto rx_buffer_flush;
```

```
uint32_t rwcnt = (ffw >> 11) & 0x1F;
for(int i = 0; i < rwcnt - 3; i++){
    tmp = can_read_32(RX_DATA_ADDR);
    data[i*4] = tmp & 0xFF;
    data[i*4+1] = (tmp >> 8) & 0xFF;
    data[i*4+2] = (tmp >> 16) & 0xFF;
    data[i*4+3] = (tmp >> 24) & 0xFF;

    if (can_read_32(STATUS_ADDR) >> 10) & 0x1)
        goto parity_err_handler;
}
return RX_FRAME_READ_OK;

/* Read out corrupted RX frame until start of new frame. */
parity_err_handler:

    int i=0;
    while (i < 16) {
        if (((can_read_32(RX_STATUS_ADDR) >> 2) & 0x1) == 0){
            can_write_32(COMMAND_ADDR, 0x200);
            return RX_FRAME_DROPPED;
        }
        i++;
        can_read_32(RX_DATA_ADDR);
    }

/* If we get here, there is a danger that RX buffer is not in consistent state. */
rx_buffer_flush:

    can_write_32(COMMAND_ADDR, 0x202);
    return RX_BUFFER_RESET;
```

**Note** The example above assumes that RX buffer is read in Automatic mode (MODE[RXBAM]=1). However, if single word from RX buffer is read via e.g. 4 x 8-bit accesses in MODE[RXBAM]=0, CTU CAN FD sets STATUS[RXPE] upon each read from a RX_DATA which contains parity error.

**Note** Writing COMMAND[CRXPE]=1 by SW clears STATUS[RXPE] bit.

**Note** When SETTINGS[PCHKE] = 0, CTU CAN FD ignores parity error detected in RX buffer (STATUS[RXPE] is not set, and COMMAND[CRXPE] has no effect).

**Note** When writing RX buffer RAM via Test Registers (see 2.16.5), parity bit of corresponding word of RX buffer RAM is not updated. See 2.13.4

### 2.13.2 Parity protection on TXT Buffer RAMs

When SW stores a CAN frame to TXT Buffer, CTU CAN FD appends a parity bit to each word in the TXT Buffer RAM. When CTU CAN FD attempts to transmit a frame from TXT Buffer where the frame contains a bit flip, CTU CAN FD behaves like so:

1. If CTU CAN FD detects parity error in FRAME_FORMAT_W, IDENTIFIER_W, TIMESTAMP_U_W or TIMESTAMP_L_W it does not attempt to transmit the CAN Frame.
2. If CTU CAN FD does not detect parity error in any of TXT Buffer words mentioned in previous point, it attempts to transmit the CAN Frame.
3. If CTU CAN FD detects parity error in any of DATA_1_4_W - DATA_61_64_W words during transmission of CAN frame, CTU CAN FD starts transmitting an error frame.

If CTU CAN FD detects a parity error in TXT Buffer RAM as described in Steps 1 or 3, the TXT Buffer moves to "Parity Error" state as shown in 2.7, and CTU CAN FD sets STATUS[TXPE] bit.

**Note** If CTU CAN FD detects a parity error in TXT Buffer, SW shall write the whole CAN frame to TXT Buffer again before it attempts to use it for further transmissions.

**Note** To clear STATUS[TXPE] bit, SW shall write COMMAND[CTXPE]=1.

**Note** When SETTINGS[PCHKE]=0, CTU CAN FD ignores parity errors detected in TXT buffers (STATUS[TXPE] is not set, COMMAND[CTXPE] has no effect, and TXT Buffers never move to Parity Error state).

**Note** When SW writes TXT Buffer RAM via Test Registers (see 2.16.5), parity bit of corresponding word in TXT Buffer RAM is not updated. See 2.13.4

**Note** CTU CAN FD does not detect parity errors in FRAME_TEST_W. Purpose of FRAME_TEST_W is to intentionally corrupt transmitted frame (e.g. for testing of error scenarios on CAN bus). Such feature is most likely not usefull in applications which require parity protection (high reliability application which aim for fault tolerance).

### 2.13.3 TXT Buffer Backup mode

When MODE[TXBBM]=1, CTU CAN FD operates in TXT Buffer Backup mode. In TXT Buffer Backup mode, TXT Buffers with adjacent indices form pairs (e.g. if TXTB_INFO[TXT_BUFFER_COUNT]=8 (CTU CAN FD contains 8 TXT Buffers) there are 4 TXT Buffer pairs: 1-2, 3-4, 5-6, 7-8) as is shown in Figure 2.15.

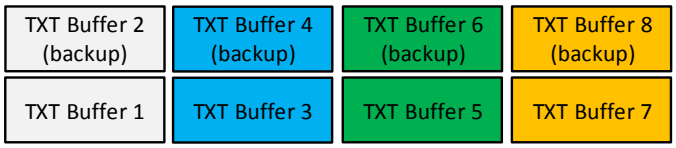


Figure 2.15: TXT Buffer pairs

Operation of CTU CAN FD in TXT Buffer Backup mode provides additional fault tolerance since TXT Buffer with higher index within TXT Buffer pair serves as "backup" in case of parity error in "original" TXT Buffer. The operation of CTU CAN FD in TXT Buffer Backup mode is shown in Figure 2.16 and explained in this section.

When MODE[TXBBM]=1, and CTU CAN FD detects a parity error in "original" TXT Buffer RAM, such TXT Buffer moves to "Parity Error" state, and CTU CAN FD attempts to transmit frame from its "backup" TXT Buffer (e.g. if CTU CAN FD detects parity error in TXT Buffer 3, it attempts to transmit a frame from TXT Buffer 4). If CTU CAN FD succesfully transmits a frame from "original" TXT Buffer, its "backup" Buffer moves to "Aborted" state (CTU CAN FD does not transmit frame in the "backup" TXT Buffer).

When CTU CAN FD is transmitting a frame from a "backup" TXT Buffer due to parity error in "original" TXT Buffer, and it detects parity error also in "backup" TXT Buffer RAM, CTU CAN FD sets STATUS[TXDPE] bit (Double parity error).

When CTU CAN FD operates in TXT Buffer Backup mode, SW control of TXT Buffers has following differences compared to MODE[TXBBM]=0 scenario:

- Priorities of both TXT Buffers within TXT Buffer pair are equal, and they are given by TX_PRIORITY[TX*P] of "original" TXT Buffer (e.g. priority of TXT Buffers 1 and 2 is given by TX_PRIORITY[TX1P], and TX_PRIORITY[TX2P] has no effect).
- CTU CAN FD automatically applies commands issued by SW to each "original" TXT Buffer also to its corresponding "backup" TXT buffer (e.g. if SW gives command to TXT Buffer 1 (TX_COMMAND[TXB1] = 1), CTU CAN FD automatically applies it also to TXT Buffer 2).

It is assumed that SW stores equal CAN frames to both TXT Buffers from TXT Buffer pair when attempting to send CAN frame. In such case, the effect of TXT Buffer Backup mode is following: If parity error occurs in "original" TXT Buffer RAM, the same frame is transmitted from "backup" TXT buffer.

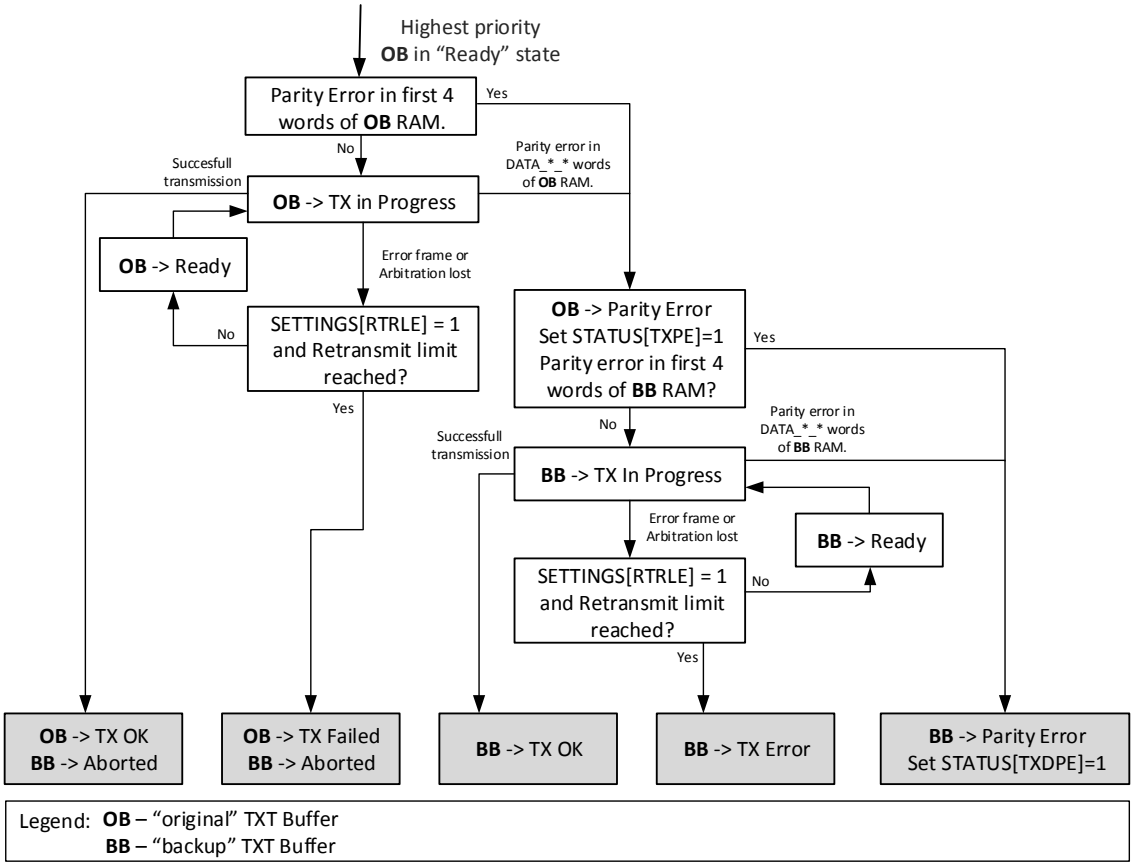


Figure 2.16: Operation in TXT Buffer Backup Mode

**Note** Storing equal frames to both TXT Buffers by separate memory accesses is intended by design. CTU CAN FD does not automatically store this frame to both TXT Buffers to avoid effect of potential SEU in the moment of storing

the frame to TXT Buffer. If such SEU occured, it could happend that frame is stored to both TXT Buffers with parity error already in it.

**Note** SW does not necessarily need to store equal frames to both TXT Buffers from a TXT Buffer pair. It may simply store any frame which shall be transmitted if parity error occurs in "original" TXT Buffer to "backup" TXT Buffer.

**Note** SW shall set MODE[TXBBM] = 1 together with SETTINGS[PCHKE] = 1. If MODE[TXBBM] = 1 together with SETTINGS[PCHKE] = 0, CTU CAN FD ignores parity errors in "original" TXT Buffers and never transmits frame from "backup" TXT Buffers.

**Note** If CTU CAN FD detects parity error in "original" TXT Buffer during CAN frame transmission, and another TXT Buffer with Higher priority than currently selected TXT buffer pair moved to Ready state (due to SW issuing Set Ready command), CTU CAN FD will attempt to transmit frame from higher priority TXT Buffer during next transmission (ignoring "backup" TXT Buffer).

**Note** TXT Buffer Backup mode is supported only when CTU CAN FD contains even number of TXT Buffers. If CTU CAN FD contains odd number of TXT Buffers, there exists one TXT Buffer which has no "backup" buffer. In such case SW shall not use this spare "original" TXT Buffer when MODE[TXBBM] = 1. If this TXT Buffer is available used when MODE[TXBBM], behavior of CTU CAN FD is undefined.

### 2.13.4 Parity protection testing

When Test registers memory region (see Section 3) is present in CTU CAN FD (STATUS[STRGS] = 1), write to TXT Buffer / RX buffer RAMs via this memory region does not update parity bit value stored in each memory word of TXT Buffer / RX buffer RAMs. This allows on-chip verification of parity detection capabilites on both TXT Buffer / RX buffer RAMs. Following sequence checks parity detection capabilities on RX buffer RAM:

1. CTU CAN FD receives CAN frame to RX buffer RAM.
2. SW reads RX buffer RAM memory via Test Registers memory region (reffer to [1] for details of such procedure).
3. SW modifies a bit in a memory word of CAN frame read in previous step, and stores such modified frame back to RX buffer RAM via Test Registers memory region.
4. SW reads a frame from RX buffer via RX_DATA register, and then reads STATUS[RXPE]. If STATUS[RXPE] = 1, then parity error detection mechanism on RX buffer RAM works correctly.

Following sequence checks parity detection capabilities of TXT Buffer RAM:

1. SW inserts CAN frame to TXT Buffer.
2. SW reads the frame via Test Registers memory region, modifies a bit in random word, and stores back such word via Test Registers memory region.
3. SW sends **Set ready** (via TX_COMMAND register) command to a TXT Buffer where CAN frame was stored in previous two steps.
4. CTU CAN FD attempts to transmit a frame from this TXT Buffer (assuming no other TXT Buffer is in "Ready" state). When reading a memory word which contains bit-flip, CTU CAN FD sends error frame, and sets STATUS[TXPE]=1.
5. SW reads STATUS[TXPE]. If yes STATUS[TXPE]=1, parity detection mechanism on TXT Buffer RAM works correctly.

**Note** When SW flips a random bit in TXT Buffer RAM, it must flip a bit in memory words which will be read by CTU CAN FD when it attempts to transmit the frame. E.g. if SW flips a bit in DATA_61_64_W, but inserted CAN frame only contains 8 data bytes (FRAME_FORMAT_W[DLC]=1000), CTU CAN FD will not attempt to read DATA_61_64_W word from TXT Buffer RAM (it will only read DATA_1_4_W and DATA_5_8_W), and therefore it will not set STATUS[TXPE] bit.

**Note** When accessing RX buffer / TXT Buffer RAMs via Test Registers Memory region, TSTCTRL[TMENA] (test access enable bit) must be set only when the access is executed, not during operation of the core. Typically, such access consists of:

1. Set TSTCTRL[TMENA]=1.
2. Read / Write RX buffer / TXT Buffer RAM via TST_DEST, TST_WDATA, TSTCTRL, TST_RDATA registers.
3. Set TSTCTRL[TMENA]=0.

## 2.14 Special modes

### 2.14.1 Loopback mode

In Loopback mode, CTU CAN FD stores every transmitted CAN frame to RX buffer. Such frame is called Loopback frame. Altough CTU CAN FD receives Loopback frame to RX buffer, CTU CAN FD still acts as a transmitter, therefore it does not acknowledge the Loopback frame on CAN bus. To sucesfully transmit Loopback frame, at least one of conditions shall be valid:

- The frame shall be acknowledged by other node on CAN bus.
- CTU CAN FD shall operate in Self-Acknowledge mode (MODE[SAM]=1).

A Loopback frame differs from CAN frame received on the CAN bus in:

- FRAME_FORMAT_W[LBPF]=1 - Indicates the frame is a Loopback frame
- FRAME_FORMAT_W[LBTBI] - Contains index of TXT Buffer used to transmit the Loopback frame. This field is reserved when FRAME_FORMAT_W[LBPF]=0.

**Note** Loopback frame is a "regular CAN frame" in context of RX buffer frame types (see. 2.10).

**Note** The frame filtering applies also on loopback frames.

Loopback mode is enabled when SETTINGS[ILBP]=1. SW shall modify SETTINGS[ILBP] only when CTU CAN FD is disabled (SETTINGS[ENA] = 0).

### 2.14.2 Self test mode

In Self test mode, CTU CAN FD considers transmitted frame valid even if does not receive dominant bit during ACK slot. SW can use this mode together with Loopback mode to verify operation of CTU CAN FD when it is a single node on a bus. SW enables Self test mode by setting MODE[STM]=1. SW shall modify MODE[STM] only when CTU CAN FD is disabled (SETTINGS[ENA] = 0).

### 2.14.3 Acknowledge forbidden mode

When Acknowledge forbidden mode is enabled, CTU CAN FD receiving CAN frame does not transmitt dominant bit during ACK slot even if received CRC matches calculated CRC. SW can enable Acknowledge forbidden mode by setting MODE[ACF] = 1. SW shall modify MODE[ACF] only when CTU CAN FD is disabled (SETTINGS[ENA] = 0).

### 2.14.4 Self acknowledge mode

When Self acknowledge mode is enabled, CTU CAN FD sends dominant ACK bit even when it transmitts CAN frame and it receives CRC matching to computed CRC. Self acknowledge mode is enabled when MODE[SAM] = 1. MODE[SAM] shall be modified only when SETTINGS[ENA] = 0.

### 2.14.5 Bus monitoring mode

In Bus monitoring mode, CTU CAN FD does not transmit any frames, it only receives CAN frames. If SW inserted CAN frame to a TXT buffer and issued **Set ready**, CTU CAN FD will not transmitt the frame, and TXT buffer will immediately move to "TX failed" state. In Bus monitoring mode, CTU CAN FD does not transmit any dominant bit to the bus. If dominant bit is about to be transmitted to the bus (e.g. ACK or error frame), it is re-routed internally so that CTU CAN FD receives this bit, but other nodes on CAN bus do not see this dominant bit. To enable Bus monitoring mode, SW shall write MODE[BMM] = 1. SW shall modify MODE[BMM] only when CTU CAN FD is disabled (SETTINGS[ENA] = 0).

### 2.14.6 Restricted operation mode

In Restricted operation mode, CTU CAN FD is able to receive frames on CAN bus, but it does not transmit any frames. If SW inserts CAN frame to a TXT buffer and issyes **Set ready** command, CTU CAN FD will not transmitt the frame, and TXT buffer will immediately move to "TX failed" state. In Restricted operation mode, CTU CAN FD gives ACK to valid frames, but it does not send Error frames nor Overload frames. If CTU CAN FD detects Error or Overload condition, it enters bus integration state, and waits until it monitors 11 consecutive recessive bits on the bus. CTU CAN FD does not modify REC and TEC counters in Restricted operation mode, therefore CTU CAN FD will always stay Error active. SW can enable Restricted operation mode by setting MODE[ROM] = 1. SW shall modify MODE[ROM] only when CTU CAN FD is disabled (SETTINGS[ENA] = 0).

### 2.14.7 Test mode

To enable a Test mode, SW shall write MODE[TSTM] = 1. In Test mode, CTU CAN FD has the following features:

- ERP register is writable, therefore threshold for transition from error-active to error-passive state is configurable.
- EWL register is writable, therefore threshold for generating Error warning limit interrupt (INT[EWLI]) is configurable.
- CTR_PRES register is writable, therefore all error counters can be modified by SW driver.
- CTU CAN FD corrupts transmitted CAN frame baesd on FRAME_TEST_W value from TXT buffer.

**Note** Test mode shall be used for debugging / development purpose only (e.g. testing of higher layers behavior during error-passive state). SW shall not use Test Mode during regular operation of CTU CAN FD.

## 2.15 Corrupting transmitted CAN frames

CTU CAN FD provides following means for corrupting/modifying transmitted CAN frame:

- Invert a bit of CRC field.
- Invert a bit of Stuff count field or Stuff Parity field.
- Replace DLC with arbitrary value.

All features for corrupting transmitted CAN frames are configured per each transmitted frame in FRAME_TEST_W memory word in TXT Buffer, details are explained in following subsections. These features are available only in Test mode (MODE[TSTM]=1). If MODE[TSTM]=0 , CTU CAN FD ignores this configuration, and transmitts uncorrupted frames. If CTU CAN FD is a receiver of a frame, it does not corrupt the frame. Therefore CTU CAN FD does not corrupt frames transmitted by other CAN nodes on the network.

**Note** Corrupting a bit, or replacing a bit field with alternative value applies before bit-stuffing, therefore effect of flipping the bit may alternate length of the frame due to additional/removed stuff bit.

**Note** To repeat transmission of a frame multiple times with corrupted bit, use standard "Retransmit limitation" mechanism, reffer to 2.2.

**Note** FRAME_TEST_W word of CAN frame is present only in TXT Buffers, it does not exist in RX buffer (longest CAN frame in RX buffer still has 20 words, not 21).

### 2.15.1 Flip a bit of CRC field

When FRAME_TEST_W[FCRC] = 1, CTU CAN FD transmitts inverted bit at CRC field bit position given by FRAME_TEST_W[TPRM]. E.g. :

- FRAME_TEST_W[TPRM] = 0x0 -> Bit at position 0 in CRC field (first bit of CRC field) is transmitted with opposite value.
- FRAME_TEST_W[TPRM] = 0xE -> Bit at position 14 in CRC field (15-th bit of CRC filed) is transmitted with opposite value.

**Note** If FRAME_TEST_W[FIND] is bigger than length of CRC field, no bit is flipped.

### 2.15.2 Flip a bit of Stuff count field

When FRAME_TEST_W[FSTC] = 1, CTU CAN FD transmitts inverted bit at Stuf count field bit position given by FRAME_TEST_W[TPRM]. E.g. :

- FRAME_FORMAT_W[TPRM] = 0x0 -> First bit of Stuff count field is transmitted with opposite value.
- FRAME_FORMAT_W[TPRM] = 0x2 -> Third bit of Stuff count field is transmitted with opposite value.
- FRAME_FORMAT_W[TPRM] = 0x3 -> Stuff Parity bit is transmitted with opposite value.

### 2.15.3 Replace DLC with arbitrary value

When FRAME_TEST_W[SDLC] = 1, CTU CAN FD transmitts FRAME_TEST_W[CPRM][3:0] bits instead of FRAME_TEST_W[DLC] in Data Length Code field of CAN frame. Number of data bytes transmitted is still derived from FRAME_TEST_W[DLC] field.

**Note** CRC transmitted is calculated from FRAME_TEST_W[TPRM] (swapped value).

## 2.16 Other features

### 2.16.1 Error code capture

An Error code capture register stores type, and position of last error on CAN bus which caused transmission of an error frame. CTU CAN FD updates Error code capture in sample point of a bit where it detected the error. SW can read Error code capture from ERR_CAPT. CAN FD standard does not define types of errors as mutually exclusive. For example, a bit error and stuff error may occur at the same time when transmitted stuff bit value is corrupted to opposite value. In such case, Error code capture stores only one type of error with highest priority. Priorities of error types are defined as (Form error having the highest priority):

| Priority | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Error type | Form error | Bit error | CRC error | ACK error | Stuff error |

**Note** CTU CAN FD reports Stuff error which occured during fixed bit stuffing method of CAN FD frame as Form error in Error code capture register.

**Note** There is an exception to above mentioned error priority order. If CTU CAN FD sends dominant stuff bit during arbitration field, and samples recessive value, then Error code capture register stores Stuff error, not Bit error.

### 2.16.2 Arbitration lost capture

Arbitration lost capture register (ALC) stores bit position within CAN arbitration field where CTU CAN FD last time lost arbitration.

### 2.16.3 Traffic counters

CTU CAN FD can measure number of CAN frames transmitted/received on CAN bus. Upon every succesfully transmitted CAN frame, CTU CAN FD increments TX_COUNTER register by 1. Upon every successfully received CAN frame, CTU CAN FD increments RX_COUNTER register by 1. To clear the TX_COUNTER register, SW shall write COMMAND[TXFCRST]=1. To clear the RX_COUNTER register, SW shall write COMMAND[RXFCRST]=1. When CTU CAN FD is in Loopback mode, and it stores own transmitted frame to RX buffer, CTU CAN FD also increments RX_COUNTER. Traffic counters are optional in CTU CAN FD. To check if traffic counters are available, SW shall read STATUS[STCNT] bit.

### 2.16.4 Debug register

CTU CAN FD contains a debug register (DEBUG_REGISTER) that directly reflects part/field of CAN frame which is currently being transmitted / received.

### 2.16.5 Memory testability

CTU CAN FD supports manufacturing testability of its internal memories (TXT buffer RAMs and RX buffer RAM) via Test Registers memory region. For details on memory testing reffer to [1].

# 3. CAN FD Core memory map

CTU CAN FD is 32 bit peripheria with support of 8, 16 or 32 bit access. Unaligned access is not supported. Byte or half word access is supported. The memory is organized as Big endian. Write to read only memory location will have no effect. Read from write only memory location can return undefined values. The memory map of CTU CAN FD consists of following memory regions:

| Memory region | Address offset |
|---|---|
| Control registers | 0x000 |
| TXT Buffer 1 | 0x100 |
| TXT Buffer 2 | 0x200 |
| TXT Buffer 3 | 0x300 |
| TXT Buffer 4 | 0x400 |
| TXT Buffer 5 | 0x500 |
| TXT Buffer 6 | 0x600 |
| TXT Buffer 7 | 0x700 |
| TXT Buffer 8 | 0x800 |
| Test registers | 0x900 |

## 3.1 Control registers

Control registers memory region.

| Bits [31:24] | Bits [23:16] | Bits [15:8] | Bits [7:0] | Address offset |
|---|---|---|---|---|
| VERSION | | DEVICE_ID | | 0x0 |
| SETTINGS | | MODE | | 0x4 |
| STATUS | | | | 0x8 |
| COMMAND | | | | 0xC |
| Reserved | | INT_STAT | | 0x10 |
| Reserved | | INT_ENA_SET | | 0x14 |
| Reserved | | INT_ENA_CLR | | 0x18 |
| Reserved | | INT_MASK_SET | | 0x1C |
| Reserved | | INT_MASK_CLR | | 0x20 |
| BTR | | | | 0x24 |
| BTR_FD | | | | 0x28 |
| FAULT_STATE | | ERP | EWL | 0x2C |
| TEC | | REC | | 0x30 |
| ERR_FD | | ERR_NORM | | 0x34 |
| CTR_PRES | | | | 0x38 |
| FILTER_A_MASK | | | | 0x3C |
| FILTER_A_VAL | | | | 0x40 |
| FILTER_B_MASK | | | | 0x44 |
| FILTER_B_VAL | | | | 0x48 |
| FILTER_C_MASK | | | | 0x4C |
| FILTER_C_VAL | | | | 0x50 |
| FILTER_RAN_LOW | | | | 0x54 |
| FILTER_RAN_HIGH | | | | 0x58 |
| FILTER_STATUS | | FILTER_CONTROL | | 0x5C |
| RX_MEM_INFO | | | | 0x60 |
| RX_POINTERS | | | | 0x64 |
| Reserved | RX_SETTINGS | RX_STATUS | | 0x68 |
| RX_DATA | | | | 0x6C |
| TX_STATUS | | | | 0x70 |
| TXTB_INFO | | TX_COMMAND | | 0x74 |
| TX_PRIORITY | | | | 0x78 |
| TS_INFO | ALC | RETR_CTR | ERR_CAPT | 0x7C |
| SSP_CFG | | TRV_DELAY | | 0x80 |
| RX_FR_CTR | | | | 0x84 |
| TX_FR_CTR | | | | 0x88 |
| DEBUG_REGISTER | | | | 0x8C |
| YOLO_REG | | | | 0x90 |
| TIMESTAMP_LOW | | | | 0x94 |
| TIMESTAMP_HIGH | | | | 0x98 |

| Reserved | ... |
|---|---|

### 3.1.1 DEVICE_ID

**Type:** read-only

**Offset:** 0x0

**Size:** 2 bytes

Identifer of CTU CAN FD. Can be used to check if CTU CAN FD is accessible correctly on its base address.

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | DEVICE_ID[15:8] | | | | | | | |
| Reset value | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | DEVICE_ID[7:0] | | | | | | | |
| Reset value | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |

**DEVICE_ID** Device ID

0b1100101011111101 - CTU_CAN_FD_ID - Identifier of CTU CAN FD.

### 3.1.2 VERSION

**Type:** read-only

**Offset:** 0x2

**Size:** 2 bytes

Version register. Returns version of CTU CAN FD.

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | VER_MAJOR | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | VER_MINOR | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**VER_MINOR** Minor part of CTU CAN FD version. E.g for version 2.1 this field has value 0x01.

**VER_MAJOR** Minor part of CTU CAN FD version. E.g for version 2.1 this field has value 0x02.

### 3.1.3 MODE

**Type:** read-write

**Offset:** 0x4

**Size:** 2 bytes

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | ERFM | SAM | TXBBM | RXBAM | TSTM |
| Reset value | - | - | - | X | 0 | 0 | 1 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | ACF | ROM | TTTM | FDE | AFM | STM | BMM | RST |
| Reset value | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

**RST** Soft reset. Writing logic 1 resets CTU CAN FD. After writing logic 1, logic 0 does not need to be written, this bit is automatically cleared.

**BMM** Bus monitoring mode. In this mode CTU CAN FD only receives frames and sends only recessive bits on CAN bus. When a dominant bit is sent, it is re-routed internally so that bus value is not changed. When this mode is enabled, CTU CAN FD will not transmit any frame from TXT Buffers,
0b0 - BMM_DISABLED - Bus monitoring mode disabled.
0b1 - BMM_ENABLED - Bus monitoring mode enabled.

**STM** Self Test Mode. In this mode transmitted frame is considered valid even if dominant acknowledge was not received.
0b0 - STM_DISABLED - Self test mode disabled.
0b1 - STM_ENABLED - Self test mode enabled.

**AFM** Acceptance Filters Mode. If enabled, only RX frames which pass Frame filters are stored in RX buffer. If disabled, every received frame is stored to RX buffer. This bit has meaning only if there is at least one filter available. Otherwise, this bit is reserved.
0b0 - AFM_DISABLED - Acceptance filter mode disabled
0b1 - AFM_ENABLED - Acceptance filter mode enabled

**FDE** Flexible data rate enable. When flexible data rate is enabled CTU CAN FD recognizes CAN FD frames (FDF bit = 1).
0b0 - FDE_DISABLE - Flexible data-rate support disabled.
0b1 - FDE_ENABLE - Flexible data-rate support enabled.

**TTTM** Time triggered transmission mode.
0b0 - TTTM_DISABLED - Time Triggerer transmission is disabled. A frame from highest priority TXT Buffer in Ready state is admitted for transmission regardless of its timestamp.
0b1 - TTTM_ENABLED - Time Triggerer transmission is enabled. A frame from highest priority TXT Buffer in Ready state is admitted for transmission when its timestamp lower than CTU CAN FD timebase.

**ROM** Restricted operation mode.
0b0 - ROM_DISABLED - Restricted operation mode is disabled.
0b1 - ROM_ENABLED - Restricted operation mode is enabled.

**ACF** Acknowledge Forbidden Mode. When enabled, acknowledge is not sent even if received CRC matches the calculated one.
0b0 - ACF_DISABLED - Acknowledge forbidden mode disabled.
0b1 - ACF_ENABLED - Acknowledge forbidden mode enabled.

**TSTM** Test Mode. In test mode several registers have special features. Reffer to description of Test mode for further details.

**RXBAM** RX Buffer Automatic mode.
0b0 - RXBAM_DISABLED - RX Buffer Automatic mode Disabled.
0b1 - RXBAM_ENABLED - RX Buffer Automatic mode Enabled.

**TXBBM** TXT Buffer Backup mode.
0b0 - TXBBM_DISABLED - TXT Buffer Backup mode disabled.
0b1 - TXBBM_ENABLED - TXT Buffer Backup mode enabled.

**SAM** Self-acknowledge mode.
0b0 - SAM_DISABLE - Do not send dominant ACK bit when CTU CAN FD sends Acknowledge bit.
0b1 - SAM_ENABLE - Send dominant ACK bit when CTU CAN FD transmits CAN frame.

**ERFM** Error Frame Receive mode. When set, CTU CAN FD receives Error frames on CAN bus into its RX buffer.
0b0 - ERFM_DISABLED - Error frames are not stored to RX Buffer.
0b1 - ERFM_ENABLED - Error frames are stored to RX buffer.

### 3.1.4 SETTINGS

**Type:** read-write

**Offset:** 0x6

**Size:** 2 bytes

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | PCHKE | FDRF | TBFBO | PEX |
| Reset value | - | - | - | - | X | 0 | 1 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | NISOFD | ENA | ILBP | RTRTH | | | | RTRLE |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**RTRLE** Retransmitt Limit Enable. If enabled, CTU CAN FD only attempts to retransmitt each frame up to RTR_TH times.
0b0 - RTRLE_DISABLED - Retransmitt limit is disabled.
0b1 - RTRLE_ENABLED - Retransmitt limit is enabled.

**RTRTH** Retransmitt Limit Threshold. Maximal amount of retransmission attempts when SETTINGS[RTRLE] is enabled.

**ILBP** Internal Loop Back mode. When enabled, CTU CAN FD receives any frame it transmitts.
0b0 - INT_LOOP_DISABLED - Internal loop-back is disabled.
0b1 - INT_LOOP_ENABLED - Internal loop-back is enabled.

**ENA** Main enable bit of CTU CAN FD. When enabled, CTU CAN FD communicates on CAN bus. When disabled, it is bus-off and does not take part of CAN bus communication.
0b0 - CTU_CAN_DISABLED - The CAN Core is disabled.
0b1 - CTU_CAN_ENABLED - The CAN Core is enabled.

**NISOFD** Non ISO FD. When this bit is set, CTU CAN FD is compliant to NON-ISO CAN FD specification (no stuff count field). This bit should be modified only when SETTINGS[ENA]=0.
0b0 - ISO_FD - The CAN Controller conforms to ISO CAN FD specification.
0b1 - NON_ISO_FD - The CAN Controller conforms to NON ISO CAN FD specification.

**PEX** Protocol exception handling. When this bit is set, CTU CAN FD will start integration upon detection of protocol exception. This should be modified only when SETTINGS[ENA] = '0'.
0b0 - PROTOCOL_EXCEPTION_DISABLED - Protocol exception handling is disabled.
0b1 - PROTOCOL_EXCEPTION_ENABLED - Protocol exception handling is enabled.

**TBFBO** All TXT buffers shall go to "TX failed" state when CTU CAN FD becomes bus-off.
0b0 - TXTBUF_FAILED_BUS_OFF_DISABLED - TXT Buffers dont go to "TX failed" state when CTU CAN FD becomes bus-off.
0b1 - TXTBUF_FAILED_BUS_OFF_ENABLED - TXT Buffers go to "TX failed" state when CTU CAN FD becomes bus-off.

**FDRF** Frame filters drop Remote frames.
0b0 - DROP_RF_DISABLED - Frame filters accept RTR frames.
0b1 - DROP_RF_ENABLED - Frame filters drop RTR frames.

**PCHKE** Enable Parity checks in TXT Buffers and RX Buffer.

### 3.1.5 STATUS

**Type:** read-only

**Offset:** 0x8

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | |
| Reset value | - | - | - | - | - | - | - | - |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | SPRT | STRGS | STCNT |
| Reset value | - | - | - | - | - | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | TXDPE | TXPE | RXPE | PEXS |
| Reset value | - | - | - | - | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | IDLE | EWL | TXS | RXS | EFT | TXNF | DOR | RXNE |
| Reset value | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

**RXNE** RX buffer not empty. This bit is 1 when least one frame is stored in RX buffer.

**DOR** Data Overrun flag. This bit is set when frame was dropped due to lack of space in RX buffer. This bit can be cleared by COMMAND[RRB] or COMMAND[CDO].

**TXNF** TXT buffers status. This bit is set if at least one TXT buffer is in "Empty" state.

**EFT** Error frame is being transmitted at the moment.

**RXS** CTU CAN FD is receiver of CAN Frame.

**TXS** CTU CAN FD is transmitter of CAN Frame.

**EWL** TX Error counter (TEC) or RX Error counter (REC) is equal to, or higher than Error warning limit (EWL).

**IDLE** Bus is idle (no frame is being transmitted/received) or CTU CAN FD is bus-off.

**PEXS** Protocol exception status (flag). Set when Protocol exception occurs. Cleared by writing COMMAND[CPEXS]=1.

**RXPE** Set when parity error is detected during read of CAN frame from RX Buffer via RX_DATA register.

**TXPE** TXT Buffers Parity Error flag. Set When Parity Error is detected in a TXT Buffer during transmission from this buffer.

**TXDPE** TXT Buffer double parity error. Set in TXT Buffer Backup mode when parity error is detected in "backup" TXT Buffer.

**STCNT** Support of Traffic counters. When this bit is 1, Traffic counters are present.

**STRGS** Support of Test Registers for memory testability. When this bit is 1, Test Registers are present.

**SPRT** Support of Parity protection on each word of TXT Buffer RAM and RX Buffer RAM.

### 3.1.6 COMMAND

**Type:** write-only

**Offset:** 0xC

**Size:** 4 bytes

Allows issuing commands to CTU CAN FD. Writing logic 1 to each bit gives a command to CTU CAN FD. After writing logic 1, logic 0 does not need to be written.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | |
| Reset value | - | - | - | - | - | - | - | - |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | |
| Reset value | - | - | - | - | - | - | - | - |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | CTXDPE | CTXPE | CRXPE |
| Reset value | - | - | - | - | - | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | CPEXS | TXFCRST | RXFCRST | ERCRST | CDO | RRB | RXRPMV | Reserved |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | X | - |

**RXRPMV** RX Buffer read pointer move.

**RRB** Release RX Buffer. This command flushes RX buffer and resets its memory pointers.

**CDO** Clear Data Overrun flag in RX buffer.

**ERCRST** Error Counters Reset. When unit is bus off, issuing this command will request erasing TEC, REC counters after 128 consecutive ocurrences of 11 recessive bits. Upon completion, TEC and REC are erased and fault confinement state is set to error-active. When unit is not bus-off, or when unit is bus-off due to being disabled (SETTINGS[ENA] = '0'), this command has no effect.

**RXFCRST** Clear RX bus traffic counter (RX_COUNTER register).

**TXFCRST** Clear TX bus traffic counter (TX_COUNTER register).

**CPEXS** Clear Protocol exception status (STATUS[PEXS]).

**CRXPE** Clear STATUS[RXPE] flag.

**CTXPE** Clear STATUS[TXPE] flag.

**CTXDPE** Clear STATUS[TXDPE] flag.

### 3.1.7 INT_STAT

**Type:** read-writeOnce

**Offset:** 0x10

**Size:** 2 bytes

Interrupt Status register. Reading this register returns logic 1 for each interrupt which ocurred. Writing logic 1 to any bit clears according interrupt status. Writing logic 0 has no effect.

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | TXBHCI | RBNEI | BSI | RXFI |
| Reset value | - | - | - | - | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | OFI | BEI | ALI | FCSI | DOI | EWLI | TXI | RXI |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**RXI** Frame received interrupt.

**TXI** Frame transmitted interrupt.

**EWLI** Error warning limit interrupt. When both TEC and REC are lower than EWL and one of the becomes equal to or higher than EWL, or when both TEC and REC become less than EWL, this interrupt is generated. When Interrupt is cleared and REC, or TEC is still equal to or higher than EWL, Interrupt is not generated again.

**DOI** Data overrun interrupt. Before this interrupt is cleared , STATUS[DOR] must be cleared to avoid setting of this interrupt again.

**FCSI** Fault confinement state changed interrupt. Interrupt is set when node turns error-passive (from error-active), bus-off (from error-passive) or error-active (from bus-off after reintegration or from error-passive).

**ALI** Arbitration lost interrupt.

**BEI** Bus error interrupt.

**OFI** Overload frame interrupt.

**RXFI** RX buffer full interrupt.

**BSI** Bit rate shifted interrupt.

**RBNEI** RX buffer not empty interrupt. Clearing this interrupt and not reading out content of RX Buffer via RX_DATA will re-activate the interrupt.

**TXBHCI** TXT buffer HW command interrupt. Anytime TXT buffer receives HW command from CAN Core which changes TXT buffer state to "TX OK", "Error" or "Aborted", this interrupt will be generated.

### 3.1.8 INT_ENA_SET

**Type:** read-writeOnce

**Offset:** 0x14

**Size:** 2 bytes

Interrupt Enable Set. Writing logic 1 to a bit enables according interrupt. Writing logic 0 has no effect. Reading this register returns logic 1 for each enabled interrupt. If interrupt is captured in INT_STAT, enabled interrupt will cause CTU CAN FD to raise interrupt. Interrupts are level-based, it remains active until Interrupt status is cleared or interrupt is disabled.

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | INT_ENA_SET[11:8] | | | |
| Reset value | - | - | - | - | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | INT_ENA_SET[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**INT_ENA_SET** Bit meaning is equivalent to register INT_STAT.

### 3.1.9 INT_ENA_CLR

**Type:** write-only

**Offset:** 0x18

**Size:** 2 bytes

Interrupt Enable Clear register. Writing logic 1 disables according interrupt. Writing logic 0 has no effect. Reading this register has no effect. Disabled interrupt wil not cause interrupt to be raised by CTU CAN FD even if it is set in Interrupt status register.

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | INT_ENA_CLR[11:8] | | | |
| Reset value | - | - | - | - | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | INT_ENA_CLR[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**INT_ENA_CLR** Bit meaning is equivalent to register INT_STAT.

### 3.1.10 INT_MASK_SET

**Type:** read-writeOnce

**Offset:** 0x1C

**Size:** 2 bytes

Interrupt Mask set. Writing logic 1 masks according interrupt. Writing logic 0 has no effect. Reading this register returns logic 1 for each masked interrupt. If particular interrupt is masked, it won't be captured in INT_STAT register when internal conditions for this interrupt are met (e.g RX buffer is not empty for RXNEI).

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | INT_MASK_SET[11:8] | | | |
| Reset value | - | - | - | - | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | INT_MASK_SET[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**INT_MASK_SET** Bit meaning is equivalent to register INT_STAT.

### 3.1.11 INT_MASK_CLR

**Type:** write-only

**Offset:** 0x20

**Size:** 2 bytes

Interrupt Mask clear register. Writing logic 1 un-masks according interrupt. Writing logic 0 has no effect. Reading this register has no effect. If particular interrupt is un-masked, it will be captured in INT_STAT register when internal conditions for this interrupt are met (e.g RX buffer is not empty for RXNEI).

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | INT_MASK_CLR[11:8] | | | |
| Reset value | - | - | - | - | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | INT_MASK_CLR[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**INT_MASK_CLR** Bit meaning is equivalent to register INT_STAT.

### 3.1.12 BTR

**Type:** read-write

**Offset:** 0x24

**Size:** 4 bytes

**Note:** Register can be only written when SETTINGS[ENA] = 0, otherwise write has no effect.

Bit timing register for nominal bit rate.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | SJW | | | | | BRP[7:5] | | |
| Reset value | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | BRP[4:0] | | | | | PH2[5:3] | | |
| Reset value | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | PH2[2:0] | | | PH1[5:1] | | | | |
| Reset value | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | PH1[0] | PROP | | | | | | |
| Reset value | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

**PROP** Propagation segment

**PH1** Phase 1 segment

**PH2** Phase 2 segment

**BRP** Bit rate prescaler

**SJW** Synchronisation jump width

### 3.1.13 BTR_FD

**Type:** read-write

**Offset:** 0x28

**Size:** 4 bytes

**Note:** Register can be only written when SETTINGS[ENA] = 0, otherwise write has no effect.

Bit timing register for data bit rate.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | SJW_FD | | | | | BRP_FD[7:5] | | |
| Reset value | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | BRP_FD[4:0] | | | | | Reserved | PH2_FD[4:3] | |
| Reset value | 0 | 0 | 1 | 0 | 0 | - | 0 | 0 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | PH2_FD[2:0] | | | Reserved | PH1_FD[4:1] | | | |
| Reset value | 0 | 1 | 1 | - | 0 | 0 | 0 | 1 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | PH1_FD[0] | Reserved | PROP_FD | | | | | |
| Reset value | 1 | - | 0 | 0 | 0 | 0 | 1 | 1 |

**PROP_FD** Propagation segment

**PH1_FD** Phase 1 segment

**PH2_FD** Phase 2 segment

**BRP_FD** Bit rate prescaler

**SJW_FD** Synchronisation jump width

### 3.1.14 EWL

**Type:** read-write

**Offset:** 0x2C

**Size:** 1 byte

**Note:** Register can be only written when MODE[TSTM] = 1, otherwise write has no effect.

Error warning limit register. This register shall be modified only when SETTINGS[ENA]=0.

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | EW_LIMIT | | | | | | | |
| Reset value | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

**EW_LIMIT** Error warning limit. If error warning limit is reached interrupt can be generated. Error warning limit indicates heavily disturbed bus.

### 3.1.15 ERP

**Type:** read-write

**Offset:** 0x2D

**Size:** 1 byte

**Note:** Register can be only written when MODE[TSTM] = 1, otherwise write has no effect.

Error passive limit register. This register shall be modified only when SETTINGS[ENA]=0.

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | ERP_LIMIT | | | | | | | |
| Reset value | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**ERP_LIMIT** Error Passive Limit. When one of error counters (REC/TEC) exceeds this value, Fault confinement state changes to error-passive.

### 3.1.16 FAULT_STATE

**Type:** read-only

**Offset:** 0x2E

**Size:** 2 bytes

Fault Confinement state of the CTU CAN FD.

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | |
| Reset value | - | - | - | - | - | - | - | - |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | BOF | ERP | ERA |
| Reset value | - | - | - | - | - | 1 | 0 | 0 |

**ERA** Error-active

**ERP** Error-passive

**BOF** Bus-off

### 3.1.17 REC

**Type:** read-only

**Offset:** 0x30

**Size:** 2 bytes

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | REC_VAL[8] |
| Reset value | - | - | - | - | - | - | - | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | REC_VAL[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**REC_VAL** RX error counter (REC).

### 3.1.18 TEC

**Type:** read-only

**Offset:** 0x32

**Size:** 2 bytes

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | TEC_VAL[8] |
| Reset value | - | - | - | - | - | - | - | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | TEC_VAL[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**TEC_VAL** TX error counter (TEC).

### 3.1.19 ERR_NORM

**Type:** read-only

**Offset:** 0x34

**Size:** 2 bytes

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | ERR_NORM_VAL[15:8] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | ERR_NORM_VAL[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**ERR_NORM_VAL** Number of errors which occured in nominal bit rate.

### 3.1.20 ERR_FD

**Type:** read-only

**Offset:** 0x36

**Size:** 2 bytes

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | ERR_FD_VAL[15:8] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | ERR_FD_VAL[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**ERR_FD_VAL** Number of errors which occured in data bit rate.

### 3.1.21 CTR_PRES

**Type:** write-only

**Offset:** 0x38

**Size:** 4 bytes

**Note:** Register can be only written when MODE[TSTM] = 1, otherwise write has no effect.

Counter preset register. Error counters can be modified via this register.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | |
| Reset value | - | - | - | - | - | - | - | - |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | |
| Reset value | - | - | - | - | - | - | - | - |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | EFD | ENORM | PRX | PTX | CTPV[8] |
| Reset value | - | - | - | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | CTPV[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**CTPV** Counter value to set.

**PTX** Preset value from CTPV to TX Error counter (TEC).

**PRX** Preset value from CTPV to RX Error counter (REC).

**ENORM** Erase Nominal bit rate error counter (ERR_NORM).

**EFD** Erase Data bit rate error counter (ERR_FD).

### 3.1.22 FILTER_A_MASK

**Type:** read-write

**Offset:** 0x3C

**Size:** 4 bytes

**Note:** Register is present only when sup_filt_A = true. Otherwise this address is reserved.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | BIT_MASK_A_VAL[28:24] | | | | |
| Reset value | - | - | - | 0 | 0 | 0 | 0 | 0 |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_MASK_A_VAL[23:16] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_MASK_A_VAL[15:8] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_MASK_A_VAL[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**BIT_MASK_A_VAL** Filter A mask. The identifier format is the same as in IDENTIFIER_W of TXT buffer or RX buffer. If filter A is not present, writes to this register have no effect and read will return all zeroes.

### 3.1.23 FILTER_A_VAL

**Type:** read-write

**Offset:** 0x40

**Size:** 4 bytes

**Note:** Register is present only when sup_filt_A = true. Otherwise this address is reserved.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | BIT_VAL_A_VAL[28:24] | | | | |
| Reset value | - | - | - | 0 | 0 | 0 | 0 | 0 |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_VAL_A_VAL[23:16] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_VAL_A_VAL[15:8] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_VAL_A_VAL[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**BIT_VAL_A_VAL** Filter A value. The identifier format is the same as in IDENTIFIER_W of TXT buffer or RX buffer. If filter A is not present, writes to this register have no effect and read will return all zeroes.

### 3.1.24 FILTER_B_MASK

**Type:** read-write

**Offset:** 0x44

**Size:** 4 bytes

**Note:** Register is present only when sup_filt_B = true. Otherwise this address is reserved.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | BIT_MASK_B_VAL[28:24] | | | | |
| Reset value | - | - | - | 0 | 0 | 0 | 0 | 0 |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_MASK_B_VAL[23:16] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_MASK_B_VAL[15:8] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_MASK_B_VAL[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**BIT_MASK_B_VAL** Filter B mask. The identifier format is the same as in IDENTIFIER_W of TXT buffer or RX buffer. If filter A is not present, writes to this register have no effect and read will return all zeroes.

### 3.1.25 FILTER_B_VAL

**Type:** read-write

**Offset:** 0x48

**Size:** 4 bytes

**Note:** Register is present only when sup_filt_B = true. Otherwise this address is reserved.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | BIT_VAL_B_VAL[28:24] | | | | |
| Reset value | - | - | - | 0 | 0 | 0 | 0 | 0 |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_VAL_B_VAL[23:16] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_VAL_B_VAL[15:8] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_VAL_B_VAL[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**BIT_VAL_B_VAL** Filter B value. The identifier format is the same as in IDENTIFIER_W of TXT buffer or RX buffer. If filter A is not present, writes to this register have no effect and read will return all zeroes.

### 3.1.26 FILTER_C_MASK

**Type:** read-write

**Offset:** 0x4C

**Size:** 4 bytes

**Note:** Register is present only when sup_filt_C = true. Otherwise this address is reserved.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | BIT_MASK_C_VAL[28:24] | | | | |
| Reset value | - | - | - | 0 | 0 | 0 | 0 | 0 |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_MASK_C_VAL[23:16] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_MASK_C_VAL[15:8] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_MASK_C_VAL[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**BIT_MASK_C_VAL** Filter C mask. The identifier format is the same as in IDENTIFIER_W of TXT buffer or RX buffer. If filter A is not present, writes to this register have no effect and read will return all zeroes.

### 3.1.27 FILTER_C_VAL

**Type:** read-write

**Offset:** 0x50

**Size:** 4 bytes

**Note:** Register is present only when sup_filt_C = true. Otherwise this address is reserved.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | BIT_VAL_C_VAL[28:24] | | | | |
| Reset value | - | - | - | 0 | 0 | 0 | 0 | 0 |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_VAL_C_VAL[23:16] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_VAL_C_VAL[15:8] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_VAL_C_VAL[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**BIT_VAL_C_VAL** Filter C value. The identifier format is the same as in IDENTIFIER_W of TXT buffer or RX buffer. If filter A is not present, writes to this register have no effect and read will return all zeroes.

### 3.1.28 FILTER_RAN_LOW

**Type:** read-write

**Offset:** 0x54

**Size:** 4 bytes

**Note:** Register is present only when sup_range = true. Otherwise this address is reserved.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | BIT_RAN_LOW_VAL[28:24] | | | | |
| Reset value | - | - | - | 0 | 0 | 0 | 0 | 0 |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_RAN_LOW_VAL[23:16] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_RAN_LOW_VAL[15:8] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_RAN_LOW_VAL[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**BIT_RAN_LOW_VAL** Filter Range Low threshold. The identifier format is the same as in IDENTIFIER_W of TXT buffer or RX buffer. If Range filter is not supported, writes to this register have no effect and read will return all zeroes.

### 3.1.29 FILTER_RAN_HIGH

**Type:** read-write

**Offset:** 0x58

**Size:** 4 bytes

**Note:** Register is present only when sup_range = true. Otherwise this address is reserved.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | BIT_RAN_HIGH_VAL[28:24] | | | | |
| Reset value | - | - | - | 0 | 0 | 0 | 0 | 0 |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_RAN_HIGH_VAL[23:16] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_RAN_HIGH_VAL[15:8] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | BIT_RAN_HIGH_VAL[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**BIT_RAN_HIGH_VAL** Range filter High threshold. The identifier format is the same as in IDENTIFIER_W of TXT buffer or RX buffer. If Range filter is not supported, writes to this register have no effect and read will return all zeroes.

### 3.1.30 FILTER_CONTROL

**Type:** read-write

**Offset:** 0x5C

**Size:** 2 bytes

Filter control register. Configures Frame filters to accept only selected frame types. Every bit is active in logic 1.

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | FRFE | FRFB | FRNE | FRNB | FCFE | FCFB | FCNE | FCNB |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | FBFE | FBFB | FBNE | FBNB | FAFE | FAFB | FANE | FANB |
| Reset value | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

**FANB** CAN Basic Frame is accepted by filter A.

**FANE** CAN Extended Frame is accepted by Filter A.

**FAFB** CAN FD Basic Frame is accepted by filter A.

**FAFE** CAN FD Extended Frame is accepted by filter A.

**FBNB** CAN Basic Frame is accepted by filter B.

**FBNE** CAN Extended Frame is accepted by Filter B.

**FBFB** CAN FD Basic Frame is accepted by filter B.

**FBFE** CAN FD Extended Frame is accepted by filter B.

**FCNB** CAN Basic Frame is accepted by filter C.

**FCNE** CAN Extended Frame is accepted by Filter C.

**FCFB** CAN FD Basic Frame is accepted by filter C.

**FCFE** CAN FD Extended Frame is accepted by filter C.

**FRNB** CAN Basic Frame is accepted by Range filter.

**FRNE** CAN Extended Frame is accepted by Range filter.

**FRFB** CAN FD Basic Frame is accepted by Range filter.

**FRFE** CAN FD Extended Frame is accepted by Range filter.

### 3.1.31 FILTER_STATUS

**Type:** read-only

**Offset:** 0x5E

**Size:** 2 bytes

Filter status indicates if frame filters are available in CTU CAN FD.

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | |
| Reset value | - | - | - | - | - | - | - | - |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | SFR | SFC | SFB | SFA |
| Reset value | - | - | - | - | X | X | X | X |

**SFA** Logic 1 when Filter A is available. Otherwise logic 0.

**SFB** Logic 1 when Filter B is available. Otherwise logic 0.

**SFC** Logic 1 when Filter C is available. Otherwise logic 0.

**SFR** Logic 1 when Range Filter is available. Otherwise logic 0.

### 3.1.32 RX_MEM_INFO

**Type:** read-only

**Offset:** 0x60

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | RX_MEM_FREE[12:8] | | | | |
| Reset value | - | - | - | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | RX_MEM_FREE[7:0] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | RX_BUFF_SIZE[12:8] | | | | |
| Reset value | - | - | - | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | RX_BUFF_SIZE[7:0] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**RX_BUFF_SIZE** Size of RX buffer in 32-bit words.

**RX_MEM_FREE** Number of free 32 bit words in RX buffer.

### 3.1.33 RX_POINTERS

**Type:** read-only

**Offset:** 0x64

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | RX_RPP[11:8] | | | |
| Reset value | - | - | - | - | 0 | 0 | 0 | 0 |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | RX_RPP[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | RX_WPP[11:8] | | | |
| Reset value | - | - | - | - | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | RX_WPP[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**RX_WPP** Write pointer position in RX buffer. Upon store of received frame write pointer is updated.

**RX_RPP** Read pointer position in RX buffer. Upon read of received frame read pointer is updated.

### 3.1.34 RX_STATUS

**Type:** read-only

**Offset:** 0x68

**Size:** 2 bytes

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | RXFRC[10:4] | | | | | | |
| Reset value | - | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | RXFRC[3:0] | | | | Reserved | RXMOF | RXF | RXE |
| Reset value | 0 | 0 | 0 | 0 | - | 0 | 0 | 1 |

**RXE** RX buffer is empty. There is no CAN Frame stored in it.

**RXF** RX buffer is full, all memory words of RX buffer are occupied.

**RXMOF** RX Buffer middle of frame. When RXMOF = 1, next read from RX_DATA register will return other than first word (FRAME_FORMAT_W) of CAN frame.

**RXFRC** RX buffer frame count. Number of CAN frames stored in RX buffer.

### 3.1.35 RX_SETTINGS

**Type:** read-write

**Offset:** 0x6A

**Size:** 1 byte

Settings of RX buffer FIFO.

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | RTSOP |
| Reset value | - | - | - | - | - | - | - | 0 |

**RTSOP** Receive buffer timestamp option. This register should be modified only when SETTINGS[ENA]=0.
0b0 - RTS_END - Timestamp of received frame in RX FIFO is captured in last bit of EOF field.
0b1 - RTS_BEG - Timestamp of received frame in RX FIFO is captured in SOF field.

### 3.1.36 RX_DATA

**Type:** read-only

**Offset:** 0x6C

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | RX_DATA[31:24] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | RX_DATA[23:16] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | RX_DATA[15:8] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | RX_DATA[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**RX_DATA** RX buffer data at read pointer position in FIFO. By reading from this register, read pointer is automatically incremented if MODES[RXBAM]=1 and RX Buffer is not empty. If MODE[RXBAM]=1, this register must be read by 32 bit access. Upon read from this register, STATUS[RXPE] is set if there is parity error detected in RX Buffer word which is being read.

### 3.1.37 TX_STATUS

**Type:** read-only

**Offset:** 0x70

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | TX8S | | | | TX7S | | | |
| Reset value | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | TX6S | | | | TX5S | | | |
| Reset value | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | TX4S | | | | TX3S | | | |
| Reset value | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | TX2S | | | | TX1S | | | |
| Reset value | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

**TX1S** Status of TXT buffer 1.

0b0000 - TXT_NOT_EXIST - TXT buffer does not exist in the core (applies only to TXT buffers 3-8, when CTU CAN FD was synthesized with less than 8 TXT buffers).
0b0001 - TXT_RDY - TXT buffer is in "Ready" state, it is waiting for CTU CAN FD to start transmission from it.
0b0010 - TXT_TRAN - TXT buffer is in "TX in progress" state. CTU CAN FD is transmitting frame.
0b0011 - TXT_ABTP - TXT buffer is in "Abort in progress" state.
0b0100 - TXT_TOK - TXT buffer is in "TX OK" state.
0b0110 - TXT_ERR - TXT buffer is in "Failed" state.
0b0111 - TXT_ABT - TXT buffer is in "Aborted" state.
0b1000 - TXT_ETY - TXT buffer is in "Empty" state.
0b1001 - TXT_PER - TXT Buffer is in "Parity Error" state. CTU CAN FD detected parity error on this buffer.

**TX2S** Status of TXT buffer 2. Bit field meaning is analogous to TX1S.

**TX3S** Status of TXT buffer 3. Bit field meaning is analogous to TX1S.

**TX4S** Status of TXT buffer 4. Bit field meaning is analogous to TX1S.

**TX5S** Status of TXT buffer 5. Bit field meaning is analogous to TX1S.

**TX6S** Status of TXT buffer 6. Bit field meaning is analogous to TX1S.

**TX7S** Status of TXT buffer 7. Bit field meaning is analogous to TX1S.

**TX8S** Status of TXT buffer 8. Bit field meaning is analogous to TX1S.

### 3.1.38 TX_COMMAND

**Type:** write-only

**Offset:** 0x74

**Size:** 2 bytes

Command register for TXT buffers. Command is activated by writing logic 1 to TXC(E|R|A) bit. TXT buffer that receives the command is selected by setting bit TXB[1-8] to logic 1. Command and index can be set by single access, or index can be set in advance. TXC(E|R|A) bits are automatically erased upon the command completion. Reffer to description of TXT buffer for meaning of commands. If TXCE and TXCR are applied simultaneously, only TXCE command is applied. If multiple commands are applied at once, only those which have effect in immediate state of TXT buffer are applied to the buffer.

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | TXB8 | TXB7 | TXB6 | TXB5 | TXB4 | TXB3 | TXB2 | TXB1 |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | TXCA | TXCR | TXCE |
| Reset value | - | - | - | - | - | 0 | 0 | 0 |

**TXCE** Issues "set empty" command.

**TXCR** Issues "set ready" command.

**TXCA** Issues "set abort" command.

**TXB1** Command is issued to TXT Buffer 1.

**TXB2** Command is issued to TXT Buffer 2.

**TXB3** Command is issued to TXT Buffer 3. If number of TXT Buffers is less than 3, this field is reserved and has no function.

**TXB4** Command is issued to TXT Buffer 4. If number of TXT Buffers is less than 4, this field is reserved and has no function.

**TXB5** Command is issued to TXT Buffer 5. If number of TXT Buffers is less than 5, this field is reserved and has no function.

**TXB6** Command is issued to TXT Buffer 6. If number of TXT Buffers is less than 6, this field is reserved and has no function.

**TXB7** Command is issued to TXT Buffer 7. If number of TXT Buffers is less than 7, this field is reserved and has no function.

**TXB8** Command is issued to TXT Buffer 8. If number of TXT Buffers is less than 8, this field is reserved and has no function.

### 3.1.39 TXTB_INFO

**Type:** read-only

**Offset:** 0x76

**Size:** 2 bytes

Register with information about supported features of TXT buffers.

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | |
| Reset value | - | - | - | - | - | - | - | - |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | TXT_BUFFER_COUNT | | | |
| Reset value | - | - | - | - | X | X | X | X |

**TXT_BUFFER_COUNT** Number of TXT buffers present in CTU CAN FD. Lowest buffer is always 1. Highest buffer is at index equal to number of present buffers.

### 3.1.40 TX_PRIORITY

**Type:** read-write

**Offset:** 0x78

**Size:** 4 bytes

Priority of TXT buffers. Highest priority TXT buffer in "Ready" state is selected for transmission.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | TXT8P | | | Reserved | TXT7P | | |
| Reset value | - | 0 | 0 | 0 | - | 0 | 0 | 0 |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | TXT6P | | | Reserved | TXT5P | | |
| Reset value | - | 0 | 0 | 0 | - | 0 | 0 | 0 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | TXT4P | | | Reserved | TXT3P | | |
| Reset value | - | 0 | 0 | 0 | - | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | TXT2P | | | Reserved | TXT1P | | |
| Reset value | - | 0 | 0 | 0 | - | 0 | 0 | 1 |

**TXT1P** Priority of TXT buffer 1.

**TXT2P** Priority of TXT buffer 2.

**TXT3P** Priority of TXT buffer 3. If number of TXT Buffers is less than 3, this field is reserved and has no function.

**TXT4P** Priority of TXT buffer 4. If number of TXT Buffers is less than 4, this field is reserved and has no function.

**TXT5P** Priority of TXT buffer 5. If number of TXT Buffers is less than 5, this field is reserved and has no function.

**TXT6P** Priority of TXT buffer 6. If number of TXT Buffers is less than 6, this field is reserved and has no function.

**TXT7P** Priority of TXT buffer 7. If number of TXT Buffers is less than 7, this field is reserved and has no function.

**TXT8P** Priority of TXT buffer 8. If number of TXT Buffers is less than 8, this field is reserved and has no function.

### 3.1.41 ERR_CAPT

**Type:** read-only

**Offset:** 0x7C

**Size:** 1 byte

Error code capture register. Determines position within CAN frame where last error was detected.

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | ERR_TYPE | | | ERR_ERP | ERR_POS | | | |
| Reset value | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

**ERR_POS** Position of last error.
- 0b0000 - ERC_POS_SOF - Error in Start of Frame
- 0b0001 - ERC_POS_ARB - Error in Arbitration Filed
- 0b0010 - ERC_POS_CTRL - Error in Control field
- 0b0011 - ERC_POS_DATA - Error in Data Field
- 0b0100 - ERC_POS_CRC - Error in CRC Field
- 0b0101 - ERC_POS_ACK - Error in CRC delimiter, ACK field or ACK delimiter
- 0b0110 - ERC_POS_EOF - Error in End of frame field
- 0b0111 - ERC_POS_ERR - Error during Error frame
- 0b1000 - ERC_POS_OVRL - Error in Overload frame
- 0b1111 - ERC_POS_OTHER - Other position of error

**ERR_ERP** CTU CAN FD was error passive the time when last error was detected.
- 0b0 - ERR_ERP_ACTIVE - CTU CAN FD was error passive when it detected the error condition.
- 0b1 - ERR_ERP_PASSIVE - CTU CAN FD was error active when it detected the error condition.

**ERR_TYPE** Type of last error.
- 0b000 - ERC_BIT_ERR - Bit Error
- 0b001 - ERC_CRC_ERR - CRC Error
- 0b010 - ERC_FRM_ERR - Form Error
- 0b011 - ERC_ACK_ERR - Acknowledge Error
- 0b100 - ERC_STUF_ERR - Stuff Error
- 0b101 - ERC_PRT_ERR - Parity Error in TXT Buffer RAM DATA_1_4_W ... DATA_61_64_W words.

### 3.1.42 RETR_CTR

**Type:** read-only

**Offset:** 0x7D

**Size:** 1 byte

Current value of Retransmit counter.

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | RETR_CTR_VAL | | | |
| Reset value | - | - | - | - | 0 | 0 | 0 | 0 |

**RETR_CTR_VAL** Current value of retransmitt counter.

### 3.1.43 ALC

**Type:** read-only

**Offset:** 0x7E

**Size:** 1 byte

Arbitration lost capture register. Determines position of last arbitration loss within CAN frame.

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | ALC_ID_FIELD | | | ALC_BIT | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**ALC_BIT** Arbitration lost capture bit position. If ALC_ID_FIELD = ALC_BASE_ID then bit index of BASE identifier in which arbitration was lost is given as: 11 - ALC_VAL. If ALC_ID_FIELD = ALC_EXTENSION then bit index of EXTENDED identifier in which arbitration was lost is given as: 18 - ALC_VAL. For other values of ALC_ID_FIELD, this value is undefined.

**ALC_ID_FIELD** Part of CAN Identifier in which arbitration was lost.
0b000 - ALC_RSVD - Unit did not loose arbitration since last reset.
0b001 - ALC_BASE_ID - Arbitration was lost during base identifier.
0b010 - ALC_SRR_RTR - Arbitration was lost during first bit after base identifier (SRR of Extended Frame, RTR bit of CAN 2.0 Base Frame)
0b011 - ALC_IDE - Arbitration was lost during IDE bit.
0b100 - ALC_EXTENSION - Arbitration was lost during Identifier extension.
0b101 - ALC_RTR - Arbitration was lost during RTR bit after Identifier extension!

### 3.1.44 TS_INFO

**Type:** read-only

**Offset:** 0x7F

**Size:** 1 byte

Timestamp integration information

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | TS_BITS | | | | | |
| Reset value | - | - | X | X | X | X | X | X |

**TS_BITS** Number of active bits of CTU CAN FD time-base minus 1 (0x3F = 64 bit time-base).

### 3.1.45 TRV_DELAY

**Type:** read-only

**Offset:** 0x80

**Size:** 2 bytes

Transmitter delay register. When transmitting CAN FD Frame, Transmitter delay is measured. After the measurement (after FDF bit), it can be read out from this register. The value in this register is valid since first transmission of CAN FD frame. After each next measurement the value is updated.

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | |
| Reset value | - | - | - | - | - | - | - | - |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | TRV_DELAY_VALUE | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**TRV_DELAY_VALUE** Measured Transmitter delay in multiple of minimal Time quanta.

### 3.1.46 SSP_CFG

**Type:** read-write

**Offset:** 0x82

**Size:** 2 bytes

**Note:** Register can be only written when SETTINGS[ENA] = 0, otherwise write has no effect.

Secondary sampling point configuration register. Used by transmitter in data bit rate for calculation of Secondary sampling point.

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | SSP_SRC | |
| Reset value | - | - | - | - | - | - | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | SSP_OFFSET | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

**SSP_OFFSET** Secondary sampling point offset. Value is given as multiple of minimal Time quanta.

**SSP_SRC** Source of Secondary sampling point.
0b00 - SSP_SRC_MEAS_N_OFFSET - SSP position = TRV_DELAY (Measured Transmitter delay) + SSP_OFFSET.
0b01 - SSP_SRC_NO_SSP - SSP is not used. Transmitter uses regular Sampling Point during data bit rate.
0b10 - SSP_SRC_OFFSET - SSP position = SSP_OFFSET. Measured Transmitter delay value is ignored.

### 3.1.47 RX_FR_CTR

**Type:** read-only

**Offset:** 0x84

**Size:** 4 bytes

**Note:** Register is present only when sup_traffic_ctrs = true. Otherwise this address is reserved.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | RX_FR_CTR_VAL[31:24] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | RX_FR_CTR_VAL[23:16] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | RX_FR_CTR_VAL[15:8] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | RX_FR_CTR_VAL[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**RX_FR_CTR_VAL** Number of received frames by CTU CAN FD.

### 3.1.48 TX_FR_CTR

**Type:** read-only

**Offset:** 0x88

**Size:** 4 bytes

**Note:** Register is present only when sup_traffic_ctrs = true. Otherwise this address is reserved.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | TX_FR_CTR_VAL[31:24] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | TX_FR_CTR_VAL[23:16] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | TX_FR_CTR_VAL[15:8] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | TX_FR_CTR_VAL[7:0] | | | | | | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**TX_FR_CTR_VAL** Number of transmitted frames by CTU CAN FD.

### 3.1.49 DEBUG_REGISTER

**Type:** read-only

**Offset:** 0x8C

**Size:** 4 bytes

Register for reading state of the controller. This register is only for debugging purposes!

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | |
| Reset value | - | - | - | - | - | - | - | - |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | PC_SOF | PC_OVR | PC_SUSP |
| Reset value | - | - | - | - | - | 0 | 0 | 0 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | PC_INT | PC_EOF | PC_ACKD | PC_ACK | PC_CRCD | PC_CRC | PC_STC | PC_DAT |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | PC_CON | PC_ARB | DESTUFF_COUNT | | | STUFF_COUNT | | |
| Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**STUFF_COUNT** Actual stuff count modulo 8 as definned in ISO FD protocol. Stuff count is erased in the beginning of CAN frame and increased by one with each stuff bit until Stuff count field in ISO FD frame. Then it stays fixed until the beginning of next frame. In non-ISO FD or normal CAN stuff bits are counted until the end of a frame. Note that this field is NOT gray encoded as defined in ISO FD standard. Stuff count is calculated only as long as controller is transceiving on the bus. During the reception this value remains fixed!

**DESTUFF_COUNT** Actual de-stuff count modulo 8 as defined in ISO FD protocol. De-Stuff count is erased in the beginning of the frame and increased by one with each de-stuffed bit until Stuff count field in ISO FD Frame. Then it stays fixed until beginning of next frame. In non-ISO FD or normal CAN de-stuff bits are counted until the end of the frame. Note that this field is NOT grey encoded as defined in ISO FD standard. De-stuff count is calculated in both. Transceiver as well as receiver.

**PC_ARB** Protocol control state machine is in Arbitration field.

**PC_CON** Protocol control state machine is in Control field.

**PC_DAT** Protocol control state machine is in Data field.

**PC_STC** Protocol control state machine is in Stuff Count field.

**PC_CRC** Protocol control state machine is in CRC field.

**PC_CRCD** Protocol control state machine is in CRC Delimiter field.

**PC_ACK** Protocol control state machine is in ACK field.

**PC_ACKD** Protocol control state machine is in ACK Delimiter field.

**PC_EOF** Protocol control state machine is in End of file field.

**PC_INT** Protocol control state machine is in Intermission field.

**PC_SUSP** Protocol control state machine is in Suspend transmission field.

**PC_OVR** Protocol control state machine is in Overload field.

**PC_SOF** Protocol control state machine is in Start of frame field.

### 3.1.50 YOLO_REG

**Type:** read-only

**Offset:** 0x90

**Size:** 4 bytes

Register for fun :)

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | YOLO_VAL[31:24] | | | | | | | |
| Reset value | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | YOLO_VAL[23:16] | | | | | | | |
| Reset value | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | YOLO_VAL[15:8] | | | | | | | |
| Reset value | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | YOLO_VAL[7:0] | | | | | | | |
| Reset value | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |

**YOLO_VAL** What else could be in this register??

### 3.1.51 TIMESTAMP_LOW

**Type:** read-only

**Offset:** 0x94

**Size:** 4 bytes

Register with current value of CTU CAN FD time base. No shadowing is implemented on TIMESTAMP_LOW/HIGH registers and user has to take care of proper read from both registers, since overflow of TIMESTAMP_LOW might occur between read of TIMESTAMP_LOW and TIMESTAMP_HIGH.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | TIMESTAMP_LOW[31:24] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | TIMESTAMP_LOW[23:16] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | TIMESTAMP_LOW[15:8] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | TIMESTAMP_LOW[7:0] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**TIMESTAMP_LOW** Bits 31:0 of time base.

### 3.1.52 TIMESTAMP_HIGH

**Type:** read-only

**Offset:** 0x98

**Size:** 4 bytes

Register with current value of CTU CAN FD time base. No shadowing is implemented on TIMESTAMP_LOW/HIGH registers and user has to take care of proper read from both registers, since overflow of TIMESTAMP_LOW might occur between read of TIMESTAMP_LOW and TIMESTAMP_HIGH.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | TIMESTAMP_HIGH[31:24] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | TIMESTAMP_HIGH[23:16] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | TIMESTAMP_HIGH[15:8] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | TIMESTAMP_HIGH[7:0] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**TIMESTAMP_HIGH** Bits 63:32 of time base.

## 3.2 TXT Buffer 1

Access to this memory region is mapped to TXT buffer 1. CAN FD frame for transmittion can be inserted to this buffer. The frame layout corresponds to the layout described in Chapter "CAN FD frame format". First adress in this region (TXTB1_DATA_1) corresponds to FRAME_FORMAT_W, second address (TXTB1_DATA_2) corresponds to IDENTIFIER_W etc. The last address (TXTB1_DATA_20) corresponds to DATA_61_64_W. The adresses in between correspond linearly. This memory region is write only and read access will return all zeroes. This region is write only.

| Bits [31:24] | Bits [23:16] | Bits [15:8] | Bits [7:0] | Address offset |
|---|---|---|---|---|
| TXTB1_DATA_1 | | | | 0x100 |
| TXTB1_DATA_2 | | | | 0x104 |
| TXTB1_DATA_3 | | | | 0x108 |
| TXTB1_DATA_4 | | | | 0x10C |
| TXTB1_DATA_5 | | | | 0x110 |
| TXTB1_DATA_6 | | | | 0x114 |
| TXTB1_DATA_7 | | | | 0x118 |
| TXTB1_DATA_8 | | | | 0x11C |
| TXTB1_DATA_9 | | | | 0x120 |
| TXTB1_DATA_10 | | | | 0x124 |
| TXTB1_DATA_11 | | | | 0x128 |
| TXTB1_DATA_12 | | | | 0x12C |
| TXTB1_DATA_13 | | | | 0x130 |
| TXTB1_DATA_14 | | | | 0x134 |
| TXTB1_DATA_15 | | | | 0x138 |
| TXTB1_DATA_16 | | | | 0x13C |
| TXTB1_DATA_17 | | | | 0x140 |
| TXTB1_DATA_18 | | | | 0x144 |
| TXTB1_DATA_19 | | | | 0x148 |
| TXTB1_DATA_20 | | | | 0x14C |
| TXTB1_DATA_21 | | | | 0x150 |
| Reserved | | | | ... |

## 3.3 TXT Buffer 2

Access to this memory region is mapped to TXT buffer 2. CAN FD frame for transmittion can be inserted to this buffer. The frame layout corresponds to the layout described in Chapter "CAN FD frame format". First adress in this region (TXTB2_DATA_1) corresponds to FRAME_FORMAT_W, second address (TXTB2_DATA_2) corresponds to IDENTIFIER_W etc. The last address (TXTB2_DATA_20) corresponds to DATA_61_64_W. The adresses in between correspond linearly. This memory region is write only and read access will return all zeroes. This region is write only.

| Bits [31:24] | Bits [23:16] | Bits [15:8] | Bits [7:0] | Address offset |
|---|---|---|---|---|
| TXTB2_DATA_1 | | | | 0x200 |
| TXTB2_DATA_2 | | | | 0x204 |
| TXTB2_DATA_3 | | | | 0x208 |
| TXTB2_DATA_4 | | | | 0x20C |
| TXTB2_DATA_5 | | | | 0x210 |
| TXTB2_DATA_6 | | | | 0x214 |
| TXTB2_DATA_7 | | | | 0x218 |
| TXTB2_DATA_8 | | | | 0x21C |
| TXTB2_DATA_9 | | | | 0x220 |
| TXTB2_DATA_10 | | | | 0x224 |
| TXTB2_DATA_11 | | | | 0x228 |
| TXTB2_DATA_12 | | | | 0x22C |
| TXTB2_DATA_13 | | | | 0x230 |
| TXTB2_DATA_14 | | | | 0x234 |
| TXTB2_DATA_15 | | | | 0x238 |
| TXTB2_DATA_16 | | | | 0x23C |
| TXTB2_DATA_17 | | | | 0x240 |
| TXTB2_DATA_18 | | | | 0x244 |
| TXTB2_DATA_19 | | | | 0x248 |
| TXTB2_DATA_20 | | | | 0x24C |
| TXTB2_DATA_21 | | | | 0x250 |
| Reserved | | | | ... |

## 3.4 TXT Buffer 3

Access to this memory region is mapped to TXT buffer 3. CAN FD frame for transmittion can be inserted to this buffer. The frame layout corresponds to the layout described in Chapter "CAN FD frame format". First adress in this region (TXTB3_DATA_1) corresponds to FRAME_FORMAT_W, second address (TXTB3_DATA_2) corresponds to IDENTIFIER_W etc. The last address (TXTB3_DATA_20) corresponds to DATA_61_64_W. The adresses in between correspond linearly. This memory region is write only and read access will return all zeroes. This region is write only.

| Bits [31:24] | Bits [23:16] | Bits [15:8] | Bits [7:0] | Address offset |
|---|---|---|---|---|
| TXTB3_DATA_1 | | | | 0x300 |
| TXTB3_DATA_2 | | | | 0x304 |
| TXTB3_DATA_3 | | | | 0x308 |
| TXTB3_DATA_4 | | | | 0x30C |
| TXTB3_DATA_5 | | | | 0x310 |
| TXTB3_DATA_6 | | | | 0x314 |
| TXTB3_DATA_7 | | | | 0x318 |
| TXTB3_DATA_8 | | | | 0x31C |
| TXTB3_DATA_9 | | | | 0x320 |
| TXTB3_DATA_10 | | | | 0x324 |
| TXTB3_DATA_11 | | | | 0x328 |
| TXTB3_DATA_12 | | | | 0x32C |
| TXTB3_DATA_13 | | | | 0x330 |
| TXTB3_DATA_14 | | | | 0x334 |
| TXTB3_DATA_15 | | | | 0x338 |
| TXTB3_DATA_16 | | | | 0x33C |
| TXTB3_DATA_17 | | | | 0x340 |
| TXTB3_DATA_18 | | | | 0x344 |
| TXTB3_DATA_19 | | | | 0x348 |
| TXTB3_DATA_20 | | | | 0x34C |
| TXTB3_DATA_21 | | | | 0x350 |
| Reserved | | | | ... |

## 3.5 TXT Buffer 4

Access to this memory region is mapped to TXT buffer 4. CAN FD frame for transmittion can be inserted to this buffer. The frame layout corresponds to the layout described in Chapter "CAN FD frame format". First adress in this region (TXTB4_DATA_1) corresponds to FRAME_FORMAT_W, second address (TXTB4_DATA_2) corresponds to IDENTIFIER_W etc. The last address (TXTB4_DATA_20) corresponds to DATA_61_64_W. The adresses in between correspond linearly. This memory region is write only and read access will return all zeroes. This region is write only.

| Bits [31:24] | Bits [23:16] | Bits [15:8] | Bits [7:0] | Address offset |
|---|---|---|---|---|
| TXTB4_DATA_1 | | | | 0x400 |
| TXTB4_DATA_2 | | | | 0x404 |
| TXTB4_DATA_3 | | | | 0x408 |
| TXTB4_DATA_4 | | | | 0x40C |
| TXTB4_DATA_5 | | | | 0x410 |
| TXTB4_DATA_6 | | | | 0x414 |
| TXTB4_DATA_7 | | | | 0x418 |
| TXTB4_DATA_8 | | | | 0x41C |
| TXTB4_DATA_9 | | | | 0x420 |
| TXTB4_DATA_10 | | | | 0x424 |
| TXTB4_DATA_11 | | | | 0x428 |
| TXTB4_DATA_12 | | | | 0x42C |
| TXTB4_DATA_13 | | | | 0x430 |
| TXTB4_DATA_14 | | | | 0x434 |
| TXTB4_DATA_15 | | | | 0x438 |
| TXTB4_DATA_16 | | | | 0x43C |
| TXTB4_DATA_17 | | | | 0x440 |
| TXTB4_DATA_18 | | | | 0x444 |
| TXTB4_DATA_19 | | | | 0x448 |
| TXTB4_DATA_20 | | | | 0x44C |
| TXTB4_DATA_21 | | | | 0x450 |
| Reserved | | | | ... |

## 3.6 TXT Buffer 5

Access to this memory region is mapped to TXT buffer 5. CAN FD frame for transmittion can be inserted to this buffer. The frame layout corresponds to the layout described in Chapter "CAN FD frame format". First adress in this region (TXTB5_DATA_1) corresponds to FRAME_FORMAT_W, second address (TXTB5_DATA_2) corresponds to IDENTIFIER_W etc. The last address (TXTB5_DATA_20) corresponds to DATA_61_64_W. The adresses in between correspond linearly. This memory region is write only and read access will return all zeroes. This region is write only.

| Bits [31:24] | Bits [23:16] | Bits [15:8] | Bits [7:0] | Address offset |
|---|---|---|---|---|
| TXTB5_DATA_1 | | | | 0x500 |
| TXTB5_DATA_2 | | | | 0x504 |
| TXTB5_DATA_3 | | | | 0x508 |
| TXTB5_DATA_4 | | | | 0x50C |
| TXTB5_DATA_5 | | | | 0x510 |
| TXTB5_DATA_6 | | | | 0x514 |
| TXTB5_DATA_7 | | | | 0x518 |
| TXTB5_DATA_8 | | | | 0x51C |
| TXTB5_DATA_9 | | | | 0x520 |
| TXTB5_DATA_10 | | | | 0x524 |
| TXTB5_DATA_11 | | | | 0x528 |
| TXTB5_DATA_12 | | | | 0x52C |
| TXTB5_DATA_13 | | | | 0x530 |
| TXTB5_DATA_14 | | | | 0x534 |
| TXTB5_DATA_15 | | | | 0x538 |
| TXTB5_DATA_16 | | | | 0x53C |
| TXTB5_DATA_17 | | | | 0x540 |
| TXTB5_DATA_18 | | | | 0x544 |
| TXTB5_DATA_19 | | | | 0x548 |
| TXTB5_DATA_20 | | | | 0x54C |
| TXTB5_DATA_21 | | | | 0x550 |
| Reserved | | | | ... |

## 3.7 TXT Buffer 6

Access to this memory region is mapped to TXT buffer 6. CAN FD frame for transmittion can be inserted to this buffer. The frame layout corresponds to the layout described in Chapter "CAN FD frame format". First adress in this region (TXTB6_DATA_1) corresponds to FRAME_FORMAT_W, second address (TXTB6_DATA_2) corresponds to IDENTIFIER_W etc. The last address (TXTB6_DATA_20) corresponds to DATA_61_64_W. The adresses in between correspond linearly. This memory region is write only and read access will return all zeroes. This region is write only.

| Bits [31:24] | Bits [23:16] | Bits [15:8] | Bits [7:0] | Address offset |
|---|---|---|---|---|
| TXTB6_DATA_1 | | | | 0x600 |
| TXTB6_DATA_2 | | | | 0x604 |
| TXTB6_DATA_3 | | | | 0x608 |
| TXTB6_DATA_4 | | | | 0x60C |
| TXTB6_DATA_5 | | | | 0x610 |
| TXTB6_DATA_6 | | | | 0x614 |
| TXTB6_DATA_7 | | | | 0x618 |
| TXTB6_DATA_8 | | | | 0x61C |
| TXTB6_DATA_9 | | | | 0x620 |
| TXTB6_DATA_10 | | | | 0x624 |
| TXTB6_DATA_11 | | | | 0x628 |
| TXTB6_DATA_12 | | | | 0x62C |
| TXTB6_DATA_13 | | | | 0x630 |
| TXTB6_DATA_14 | | | | 0x634 |
| TXTB6_DATA_15 | | | | 0x638 |
| TXTB6_DATA_16 | | | | 0x63C |
| TXTB6_DATA_17 | | | | 0x640 |
| TXTB6_DATA_18 | | | | 0x644 |
| TXTB6_DATA_19 | | | | 0x648 |
| TXTB6_DATA_20 | | | | 0x64C |
| TXTB6_DATA_21 | | | | 0x650 |
| Reserved | | | | ... |

## 3.8 TXT Buffer 7

Access to this memory region is mapped to TXT buffer 7. CAN FD frame for transmittion can be inserted to this buffer. The frame layout corresponds to the layout described in Chapter "CAN FD frame format". First adress in this region (TXTB7_DATA_1) corresponds to FRAME_FORMAT_W, second address (TXTB7_DATA_2) corresponds to IDENTIFIER_W etc. The last address (TXTB7_DATA_20) corresponds to DATA_61_64_W. The adresses in between correspond linearly. This memory region is write only and read access will return all zeroes. This region is write only.

| Bits [31:24] | Bits [23:16] | Bits [15:8] | Bits [7:0] | Address offset |
|---|---|---|---|---|
| TXTB7_DATA_1 | | | | 0x700 |
| TXTB7_DATA_2 | | | | 0x704 |
| TXTB7_DATA_3 | | | | 0x708 |
| TXTB7_DATA_4 | | | | 0x70C |
| TXTB7_DATA_5 | | | | 0x710 |
| TXTB7_DATA_6 | | | | 0x714 |
| TXTB7_DATA_7 | | | | 0x718 |
| TXTB7_DATA_8 | | | | 0x71C |
| TXTB7_DATA_9 | | | | 0x720 |
| TXTB7_DATA_10 | | | | 0x724 |
| TXTB7_DATA_11 | | | | 0x728 |
| TXTB7_DATA_12 | | | | 0x72C |
| TXTB7_DATA_13 | | | | 0x730 |
| TXTB7_DATA_14 | | | | 0x734 |
| TXTB7_DATA_15 | | | | 0x738 |
| TXTB7_DATA_16 | | | | 0x73C |
| TXTB7_DATA_17 | | | | 0x740 |
| TXTB7_DATA_18 | | | | 0x744 |
| TXTB7_DATA_19 | | | | 0x748 |
| TXTB7_DATA_20 | | | | 0x74C |
| TXTB7_DATA_21 | | | | 0x750 |
| Reserved | | | | ... |

## 3.9 TXT Buffer 8

Access to this memory region is mapped to TXT buffer 8. CAN FD frame for transmittion can be inserted to this buffer. The frame layout corresponds to the layout described in Chapter "CAN FD frame format". First adress in this region (TXTB8_DATA_1) corresponds to FRAME_FORMAT_W, second address (TXTB8_DATA_2) corresponds to IDENTIFIER_W etc. The last address (TXTB8_DATA_20) corresponds to DATA_61_64_W. The adresses in between correspond linearly. This memory region is write only and read access will return all zeroes. This region is write only.

| Bits [31:24] | Bits [23:16] | Bits [15:8] | Bits [7:0] | Address offset |
|---|---|---|---|---|
| TXTB8_DATA_1 | | | | 0x800 |
| TXTB8_DATA_2 | | | | 0x804 |
| TXTB8_DATA_3 | | | | 0x808 |
| TXTB8_DATA_4 | | | | 0x80C |
| TXTB8_DATA_5 | | | | 0x810 |
| TXTB8_DATA_6 | | | | 0x814 |
| TXTB8_DATA_7 | | | | 0x818 |
| TXTB8_DATA_8 | | | | 0x81C |
| TXTB8_DATA_9 | | | | 0x820 |
| TXTB8_DATA_10 | | | | 0x824 |
| TXTB8_DATA_11 | | | | 0x828 |
| TXTB8_DATA_12 | | | | 0x82C |
| TXTB8_DATA_13 | | | | 0x830 |
| TXTB8_DATA_14 | | | | 0x834 |
| TXTB8_DATA_15 | | | | 0x838 |
| TXTB8_DATA_16 | | | | 0x83C |
| TXTB8_DATA_17 | | | | 0x840 |
| TXTB8_DATA_18 | | | | 0x844 |
| TXTB8_DATA_19 | | | | 0x848 |
| TXTB8_DATA_20 | | | | 0x84C |
| TXTB8_DATA_21 | | | | 0x850 |
| Reserved | | | | ... |

## 3.10 Test registers

Test registers memory region. Contains registers with manufacturing testability features.

| Bits [31:24] | Bits [23:16] | Bits [15:8] | Bits [7:0] | Address offset |
|---|---|---|---|---|
| TST_CONTROL | | | | 0x900 |
| TST_DEST | | | | 0x904 |
| TST_WDATA | | | | 0x908 |
| TST_RDATA | | | | 0x90C |
| Reserved | | | | ... |

### 3.10.1 TST_CONTROL

**Type:** read-write

**Offset:** 0x900

**Size:** 4 bytes

**Note:** Register can be only written when MODE[TSTM] = 1, otherwise write has no effect.

Testability control register. Contains configuration of test functions.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | |
| Reset value | - | - | - | - | - | - | - | - |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | |
| Reset value | - | - | - | - | - | - | - | - |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | |
| Reset value | - | - | - | - | - | - | - | - |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | TWRSTB | TMAENA |
| Reset value | - | - | - | - | - | - | X | X |

**TMAENA** Enable test access to CTU CAN FD memories.

**TWRSTB** Writing 1 executes write acess to a memory/address given by TST_DEST register. 0 does not need to be written, this bit is cleared automatically.

### 3.10.2 TST_DEST

**Type:** read-write

**Offset:** 0x904

**Size:** 4 bytes

**Note:** Register can be only written when MODE[TSTM] = 1, otherwise write has no effect.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | |
| Reset value | - | - | - | - | - | - | - | - |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | TST_MTGT | | | |
| Reset value | - | - | - | - | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | TST_ADDR[15:8] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | TST_ADDR[7:0] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**TST_ADDR** Address for test memory access within tested memory.

**TST_MTGT** Target memory to be accessed.

- 0b0000 - TMTGT_NONE - No target memory is selected for test access.
- 0b0001 - TMTGT_RXBUF - RX buffer memory is selected for test access.
- 0b0010 - TMTGT_TXTBUF1 - TXT buffer 1 memory is selected for test access.
- 0b0011 - TMTGT_TXTBUF2 - TXT buffer 2 memory is selected for test access.
- 0b0100 - TMTGT_TXTBUF3 - TXT buffer 3 memory is selected for test access.
- 0b0101 - TMTGT_TXTBUF4 - TXT buffer 4 memory is selected for test access.
- 0b0110 - TMTGT_TXTBUF5 - TXT buffer 5 memory is selected for test access.
- 0b0111 - TMTGT_TXTBUF6 - TXT buffer 6 memory is selected for test access.
- 0b1000 - TMTGT_TXTBUF7 - TXT buffer 7 memory is selected for test access.
- 0b1001 - TMTGT_TXTBUF8 - TXT buffer 8 memory is selected for test access.

### 3.10.3 TST_WDATA

**Type:** read-write

**Offset:** 0x908

**Size:** 4 bytes

**Note:** Register can be only written when MODE[TSTM] = 1, otherwise write has no effect.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | TST_WDATA[31:24] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | TST_WDATA[23:16] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | TST_WDATA[15:8] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | TST_WDATA[7:0] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**TST_WDATA** Write data for test access.

### 3.10.4 TST_RDATA

**Type:** read-only

**Offset:** 0x90C

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | TST_RDATA[31:24] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | TST_RDATA[23:16] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | TST_RDATA[15:8] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | TST_RDATA[7:0] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**TST_RDATA** Read data for test access.

# 4. CAN FD frame format

CAN Frame format describtion as it is stored in TXT Buffers and RX Buffer.

## 4.1 CAN FD Frame format

| Bits [31:24] | Bits [23:16] | Bits [15:8] | Bits [7:0] | Address offset |
|---|---|---|---|---|
| FRAME_FORMAT_W | | | | 0x0 |
| IDENTIFIER_W | | | | 0x4 |
| TIMESTAMP_L_W | | | | 0x8 |
| TIMESTAMP_U_W | | | | 0xC |
| DATA_1_4_W | | | | 0x10 |
| DATA_5_8_W | | | | 0x14 |
| DATA_9_12_W | | | | 0x18 |
| DATA_13_16_W | | | | 0x1C |
| DATA_17_20_W | | | | 0x20 |
| DATA_21_24_W | | | | 0x24 |
| DATA_25_28_W | | | | 0x28 |
| DATA_29_32_W | | | | 0x2C |
| DATA_33_36_W | | | | 0x30 |
| DATA_37_40_W | | | | 0x34 |
| DATA_41_44_W | | | | 0x38 |
| DATA_45_48_W | | | | 0x3C |
| DATA_49_52_W | | | | 0x40 |
| DATA_53_56_W | | | | 0x44 |
| DATA_57_60_W | | | | 0x48 |
| DATA_61_64_W | | | | 0x4C |
| FRAME_TEST_W | | | | 0x50 |

### 4.1.1 FRAME_FORMAT_W

**Type:**

**Offset:** 0x0

**Size:** 4 bytes

Frame format word with CAN frame metadata.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | LBTBI | | | IVLD |
| Reset value | - | - | - | - | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | ERF_TYPE | | | ERF_ERP | ERF_POS | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | RWCNT | | | | | ESI_RSV | BRS | LBPF |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | FDF | IDE | RTR | ERF | DLC | | | |
| Reset value | X | X | X | X | X | X | X | X |

**DLC** Data Length Code.

**ERF** Error Frame Flag. When set, the current frame in RX Buffer is an Error frame. This bit has no meaning in TXT buffers.
0b0 - ERF_CAN_FRAME - Frame in RX Buffer is an Error frame.
0b1 - ERF_ERR_FRAME - Frame in RX Buffer is a regular CAN frame

**RTR** Logic 1 indicates Remote frame. Has meaning only for CAN frames. CAN FD does not have RTR frames.
0b0 - NO_RTR_FRAME - CAN frame is not RTR frame.
0b1 - RTR_FRAME - CAN frame is RTR frame.

**IDE** Extended Identifier Type. Logic 1 indicates CAN frame with both Base identifier and Identifier extension. Logic 0 indicates CAN frame with only Base identifier.
0b0 - BASE - Frame Identifier is Basic (11 bits)
0b1 - EXTENDED - Frame Identifier is Extended (11 + 18 bits)

**FDF** Flexible Data-rate Format. Distinguishes between CAN 2.0 and CAN FD Frames.
0b0 - NORMAL_CAN - Frame is CAN frame.
0b1 - FD_CAN - Frame is CAN FD frame.

**LBPF** Loop-Back Frame. When this bit is set, the current framein RX Buffer was a frame transmitted by unit itself due tooperation in Loopback Mode. This bit has no meaning inTXT Buffers.
0b0 - LBPF_FOREIGN - The frame was transmitted by other node on the bus.
0b1 - LBPF_LOOPBACK - The frame was transmitted by the unit itself.

**BRS** Bit Rate Shift. In case of CAN FD frames indicates whether bit rate is shifted CAN FD frame. This bit has no meaning for CAN frames.
0b0 - BR_NO_SHIFT - Bit rate should not be shifted if frame is CAN FD frame.
0b1 - BR_SHIFT - Bit rate should be shifted if frame is CAN FD frame.

**ESI_RSV** Error State Indicator bit for received CAN FD frames. Bit has no meaning for CAN frames nor for transmitted CAN FD frames (in TXT buffer).
0b0 - ESI_ERR_ACTIVE - Transmitted of received CAN FD frame is error active.
0b1 - ESI_ERR_PASIVE - Transmitted of received CAN FD frame is error passive.

**RWCNT** Size of CAN frame in RX buffer without FRAME_FORMAT WORD. (E.g RTR frame RWCNT=3, 64 Byte FD frame RWCNT=19). In TXT buffer this field has no meaning.

**ERF_POS** Error Frame position of Error Frame recorder in RX Buffer. This bit has no meaning in TXT Buffers. The encoding of this field is equal to ERR_CAPT[ERR_POS] register.

**ERF_ERP** CTU CAN FD was Error Passive state at time when the error was detected and stored to the RX Buffer. This bit has no meaning in TXT Buffers. The encoding of this field is equal to ERR_CAPT[ERR_STATE] register.

**ERF_TYPE** Error frame type of Error Frame recorder in RX Buffer. This bit has no meaning in TXT Buffers. The encoding of this field is equal to ERR_CAPT[ERR_TYPE] register.

**IVLD** IDENTIFIER_W word contains a valid identifier. This field is valid only in RX Buffer, it has no meaning in TXT Buffers.
0b0 - IVLD_INALID - IDENTIFIER_W does not contain a valid CAN FD identifier.
0b1 - IVLD_VALID - IDENTIFIER_W contains a valid CAN FD identifier.

**LBTBI** Loop-Back TXT Buffer index. This field is only valid in RX Buffer and when FRAME_FORMAT_W[LBTBI]=1. This field has no meaning in TXT Buffers.
0b000 - TXT_BUF_1 - TXT Buffer 1
0b001 - TXT_BUF_2 - TXT Buffer 2
0b010 - TXT_BUF_3 - TXT Buffer 3
0b011 - TXT_BUF_4 - TXT Buffer 4
0b100 - TXT_BUF_5 - TXT Buffer 5
0b101 - TXT_BUF_6 - TXT Buffer 6
0b110 - TXT_BUF_7 - TXT Buffer 7
0b111 - TXT_BUF_8 - TXT Buffer 8

### 4.1.2 IDENTIFIER_W

**Type:**

**Offset:** 0x4

**Size:** 4 bytes

CAN Identifier.

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | IDENTIFIER_BASE[10:6] | | | | |
| Reset value | - | - | - | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | IDENTIFIER_BASE[5:0] | | | | | | IDENTIFIER_EXT[17:16] | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | IDENTIFIER_EXT[15:8] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | IDENTIFIER_EXT[7:0] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**IDENTIFIER_EXT** Extended Identifier of CAN frame. Has meaning only if IDE of FRAME_FORMAT_W is EXTENDED.

**IDENTIFIER_BASE** Base Identifier of CAN frame.

### 4.1.3 TIMESTAMP_L_W

**Type:**

**Offset:** 0x8

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | TIME_STAMP_L_W[31:24] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | TIME_STAMP_L_W[23:16] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | TIME_STAMP_L_W[15:8] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | TIME_STAMP_L_W[7:0] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**TIME_STAMP_L_W** Lower 32 bits of timestamp when the frame should be transmitted or when it was received.

### 4.1.4 TIMESTAMP_U_W

**Type:**

**Offset:** 0xC

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | TIMESTAMP_U_W[31:24] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | TIMESTAMP_U_W[23:16] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | TIMESTAMP_U_W[15:8] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | TIMESTAMP_U_W[7:0] | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**TIMESTAMP_U_W** Upper 32 bits of timestamp when the frame should be transmitted or when it was received.

### 4.1.5 DATA_1_4_W

**Type:**

**Offset:** 0x10

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_4 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_3 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_2 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_1 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**DATA_1** Data byte 1 of CAN Frame.

**DATA_2** Data byte 2 of CAN Frame.

**DATA_3** Data byte 3 of CAN Frame.

**DATA_4** Data byte 4 of CAN Frame.

### 4.1.6 DATA_5_8_W

**Type:**

**Offset:** 0x14

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_8 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_7 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_6 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_5 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**DATA_5** Data byte 5 of CAN Frame.

**DATA_6** Data byte 6 of CAN Frame.

**DATA_7** Data byte 7 of CAN Frame.

**DATA_8** Data byte 8 of CAN Frame.

### 4.1.7 DATA_9_12_W

**Type:**

**Offset:** 0x18

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_12 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_11 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_10 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_9 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**DATA_9** Data byte 9 of CAN Frame.

**DATA_10** Data byte 10 of CAN Frame.

**DATA_11** Data byte 11 of CAN Frame.

**DATA_12** Data byte 12 of CAN Frame.

### 4.1.8 DATA_13_16_W

**Type:**

**Offset:** 0x1C

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_16 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_15 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_14 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_13 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**DATA_13** Data byte 13 of CAN Frame.

**DATA_14** Data byte 14 of CAN Frame.

**DATA_15** Data byte 15 of CAN Frame.

**DATA_16** Data byte 16 of CAN Frame.

### 4.1.9 DATA_17_20_W

**Type:**

**Offset:** 0x20

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_20 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_19 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_18 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_17 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**DATA_17** Data byte 17 of CAN Frame.

**DATA_18** Data byte 18 of CAN Frame.

**DATA_19** Data byte 19 of CAN Frame.

**DATA_20** Data byte 20 of CAN Frame.

### 4.1.10 DATA_21_24_W

**Type:**

**Offset:** 0x24

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_24 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_23 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_22 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_21 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**DATA_21** Data byte 21 of CAN Frame.

**DATA_22** Data byte 22 of CAN Frame.

**DATA_23** Data byte 23 of CAN Frame.

**DATA_24** Data byte 24 of CAN Frame.

### 4.1.11 DATA_25_28_W

**Type:**

**Offset:** 0x28

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_28 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_27 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_26 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_25 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**DATA_25** Data byte 25 of CAN Frame.

**DATA_26** Data byte 26 of CAN Frame.

**DATA_27** Data byte 27 of CAN Frame.

**DATA_28** Data byte 28 of CAN Frame.

### 4.1.12 DATA_29_32_W

**Type:**

**Offset:** 0x2C

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_32 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_31 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_30 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_29 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**DATA_29** Data byte 29 of CAN Frame.

**DATA_30** Data byte 30 of CAN Frame.

**DATA_31** Data byte 31 of CAN Frame.

**DATA_32** Data byte 32 of CAN Frame.

### 4.1.13 DATA_33_36_W

**Type:**

**Offset:** 0x30

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_36 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_35 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_34 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_33 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**DATA_33** Data byte 33 of CAN Frame.

**DATA_34** Data byte 34 of CAN Frame.

**DATA_35** Data byte 35 of CAN Frame.

**DATA_36** Data byte 36 of CAN Frame.

### 4.1.14 DATA_37_40_W

**Type:**

**Offset:** 0x34

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_40 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_39 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_38 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_37 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**DATA_37** Data byte 37 of CAN Frame.

**DATA_38** Data byte 38 of CAN Frame.

**DATA_39** Data byte 39 of CAN Frame.

**DATA_40** Data byte 40 of CAN Frame.

### 4.1.15 DATA_41_44_W

**Type:**

**Offset:** 0x38

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_44 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_43 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_42 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_41 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**DATA_41** Data byte 41 of CAN Frame.

**DATA_42** Data byte 42 of CAN Frame.

**DATA_43** Data byte 43 of CAN Frame.

**DATA_44** Data byte 44 of CAN Frame.

### 4.1.16 DATA_45_48_W

**Type:**

**Offset:** 0x3C

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_48 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_47 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_46 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_45 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**DATA_45** Data byte 45 of CAN Frame.

**DATA_46** Data byte 46 of CAN Frame.

**DATA_47** Data byte 47 of CAN Frame.

**DATA_48** Data byte 48 of CAN Frame.

### 4.1.17 DATA_49_52_W

**Type:**

**Offset:** 0x40

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_52 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_51 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_50 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_49 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**DATA_49** Data byte 49 of CAN Frame.

**DATA_50** Data byte 50 of CAN Frame.

**DATA_51** Data byte 51 of CAN Frame.

**DATA_52** Data byte 52 of CAN Frame.

### 4.1.18 DATA_53_56_W

**Type:**

**Offset:** 0x44

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_54 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_55 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_56 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_53 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**DATA_53** Data byte 53 of CAN Frame.

**DATA_56** Data byte 56 of CAN Frame.

**DATA_55** Data byte 55 of CAN Frame.

**DATA_54** Data byte 54 of CAN Frame.

### 4.1.19 DATA_57_60_W

**Type:**

**Offset:** 0x48

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_60 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_59 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_58 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_57 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**DATA_57** Data byte 57 of CAN Frame.

**DATA_58** Data byte 58 of CAN Frame.

**DATA_59** Data byte 59 of CAN Frame.

**DATA_60** Data byte 60 of CAN Frame.

### 4.1.20 DATA_61_64_W

**Type:**

**Offset:** 0x4C

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_64 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_63 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_62 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | DATA_61 | | | | | | | |
| Reset value | X | X | X | X | X | X | X | X |

**DATA_61** Data byte 61 of CAN Frame.

**DATA_62** Data byte 62 of CAN Frame.

**DATA_63** Data byte 63 of CAN Frame.

**DATA_64** Data byte 64 of CAN Frame.

### 4.1.21 FRAME_TEST_W

**Type:**

**Offset:** 0x50

**Size:** 4 bytes

| Bit index | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | |
| Reset value | - | - | - | - | - | - | - | - |

| Bit index | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | | | |
| Reset value | - | - | - | - | - | - | - | - |

| Bit index | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | TPRM | | | | |
| Reset value | - | - | - | X | X | X | X | X |

| Bit index | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field name | Reserved | | | | | SDLC | FCRC | FSTC |
| Reset value | - | - | - | - | - | X | X | X |

**FSTC** Flip Stuff count field bit when this frame is transmitted. This field has effect only in transmitted frames.

**FCRC** Flip CRC field bit when this frame is transmitted. This field has effect only in transmitted frames.

**SDLC** Swap DLC in transmitted frame.

**TPRM** Test Parameter

# Bibliography

[1] CTU CAN FD, System architecture.