AURIX 2G MCMCAN Overview

Keith Tang
IFCN ATV SMD GC SAE MC
2018/5/2

TC3xx MCMCAN

This presentation briefly explains:

- CAN Protocol and Infineon Implementation
- MCMCAN module overview
- MCMCAN Interface and Interconnection
- Module and node control
- CAN Frame transmission and reception handling
- CAN FD Support in MCMCAN module
- Info

Agenda

1. CAN Protocol and Infineon Implementation
2. CAN module in AURIX2G MCMCAN
3. MCMCAN Interface and Interconnection
4. Module and Node Control
5. CAN FD

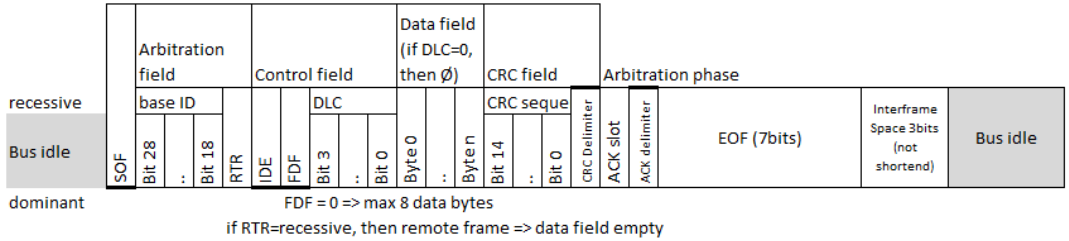
CAN Network and CAN protocol history

- **CAN characteristic**
  - multi-master bus system with broadcasting capability
  - CAN nodes do not have address, instead, CAN frame contains info about add, length of data, priority,..
  - Low cost bus system for real-time application, 'hot plug'….. but low bandwidth
  - Very reliable (single-bit error): at message level: CRC, form, ACK; at bit level: monitoring, bit stuffing

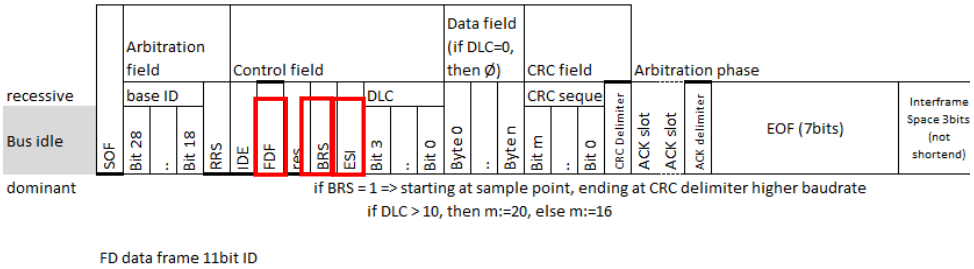
- **CAN protocol specification and history**
  - Bosch CAN specification V2.0: was submitted for int. standardization (1991)
    - 2.0A; specified for "Standard CAN" with 11 bit message ID's, total 2048 ID's available.
    - 2.0B; specified for "Extended CAN" with 29 bit message ID's, more than 536 million ID's available.
  - 2003: ISO11898-1: as successor of 2.0B
    - Revised Bosch CAN specification has become ISO standardization.
    - in this version the data link and high-speed physical layer is separated in ISO 11898-1 and -2
  - TTCAN (2000): time/event triggered bus protocol via the same physical bus
    - automotive industry has not adopted TTCAN
  - 2012: CAN FD was officially introduced
    - (ISO11898-1:2014): CAN FD is integrated into ISO11898-1
  - "no ISO CAN FD" and "ISO CAN FD"
    - CRC issue → ISO11898-1:2015

CAN frame: classical CAN - CAN FD

› Classical CAN: 11 bit ID data frame



› non-ISO CAN FD (ISO11898- 1 DIS 2014): 11-bit ID Data Frame

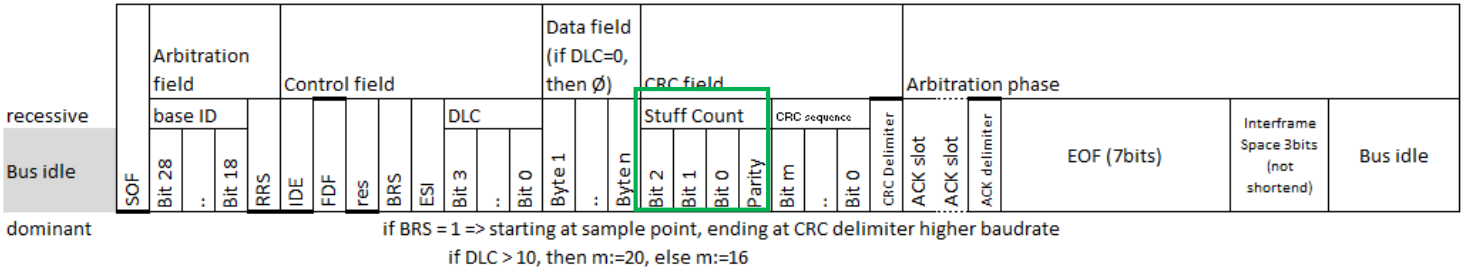


FD data frame 11bit ID

CAN FD:
› new bits: **FDF**, **BRS, ESI**
› no remote frame

ISO CAN FD:
› CRC Stuff Count

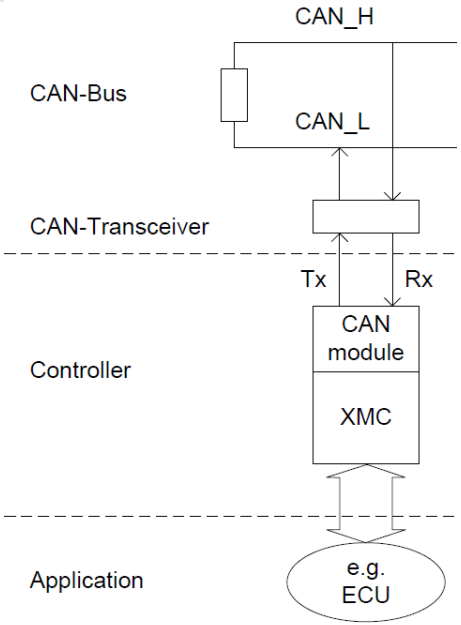
› ISO CAN FD (ISO 11898-1:2015): 11-bit ID Data Frame



Measurement on Aurix application Kit

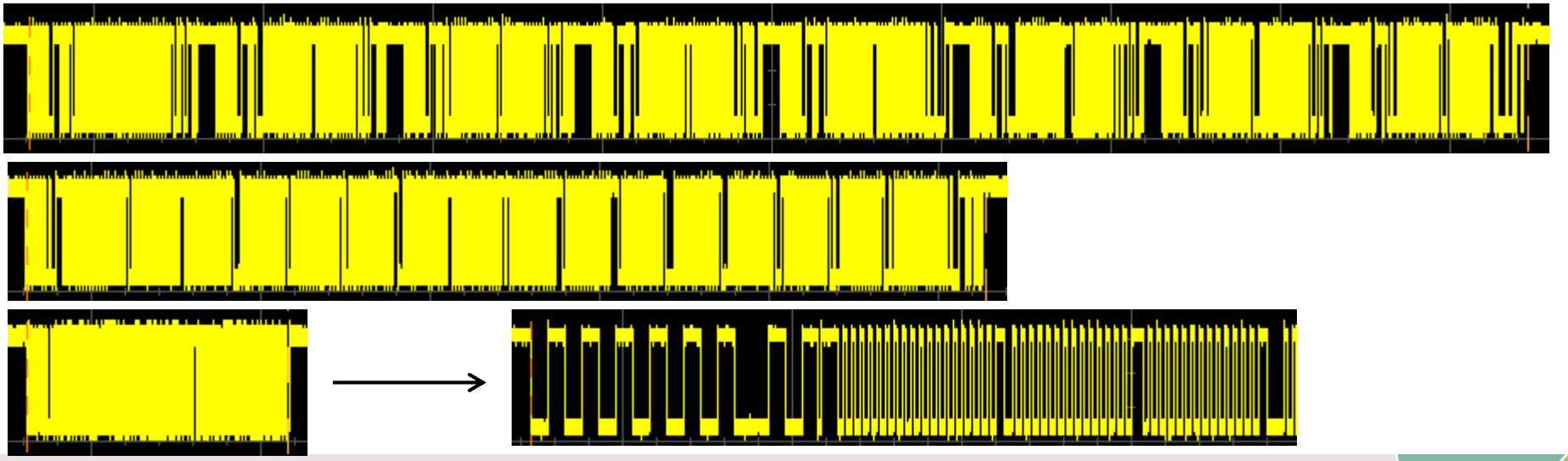
CAN FD support means:

- Baud rate switching (higher baud rate in data bytes)
- More data bytes (Max. 64 data bytes for CAN FD mode)



Example: 64 data bytes, nominal bit rate=500K, data bit rate=2000K

- Classical CAN: 1.77ms
- Long frame form, one bit rate: 1.13 ms
- Long frame form, dual bit rates: 308 us



Infineon CAN Implementations

Module name	Devices	Nodes	Message objects	FIFO/ Gateway	TTCAN	CAN FD
TwinCAN	XC16x	2	32	yes	-	-
	- 32 message objects can be individually assigned to one of the two CAN nodes. - FIFO participate in a 2, 4, 8, 16, 32 buffer…					
MultiCAN	XC800 XC2000/XE166 XMC4000 TriCore	Max. 6	Max. 256 (flexible assigned)	yes	on certain devices	-
MultiCAN+	Aurix 1G	Max. 6	Max. 256 (flexible assigned)	yes	on certain devices	yes
	- Message objects can be individually/dynamically assigned to one of the CAN nodes. - FIFO/Gateway buffer size is programmable …					
MCMCAN (Bosch M_CAN IP as CAN node)	Aurix 2G	Max. 4	freely configurable (one shared message RAM for all 4 nodes within module)	FIFO	yes	yes
	- See next slides					

Agenda

1. CAN Protocol and Infineon Implementation
2. CAN module in AURIX2G MCMCAN
3. MCMCAN Interface and Interconnection
4. Module and Node Control
5. CAN FD

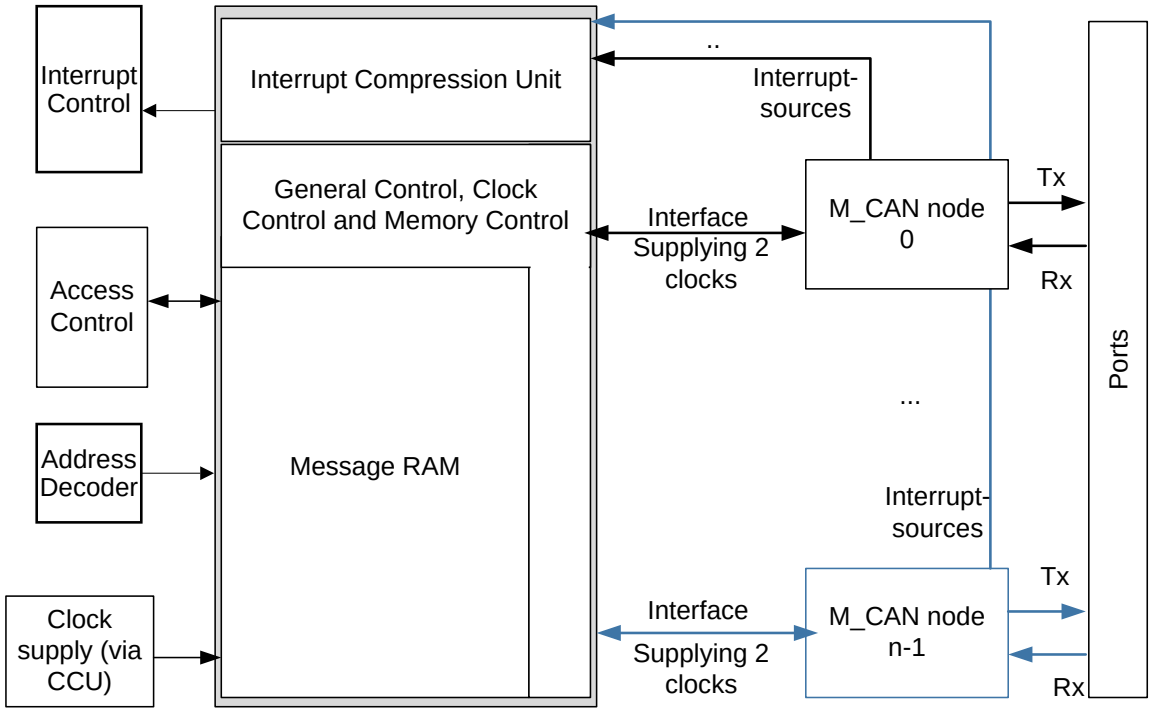
Short summary MCMCAN

- MCMCAN is the new CAN module for Aurix2G
- Each MCMCAN unit contains
  - 4 nodes (an integration of the Bosch **M_CAN** IP as CAN node)
  - Debug over CAN (CANm_BUFADR.TXBUF and RXBUF for start address)
  - Pretended Networking features
  - Freely assignable interrupts up to 16 interrupt nodes
  - Lists of filters for receive FIFOs
  - A range filter
  - An automatic transmission history list, including timestamp, ID and marker
  - Module internal loop back mode
  - MSG handling implemented by the Rx/Tx Handler
  - RX Handler: contains ID's Mask, store into MSG RAM, provides Status info
  - Tx Handler: from MSG RAM to CAN core, time schedule

MCMCAN Block Diagram

- CAN node: M_CAN
  - CAN protocol control
    - according to ISO 11898-1 (including CAN FD)/ISO 11898-4 (TTCAN)
    - CAN FD option can be used with event-/time-triggered
  - Receive and transmit time stamp generation
  - Transmit and receive handler

- MCMCAN user interface
  - One configurable message RAM
  - interrupt compression unit
  - Clock control block
  - Port pin



Message RAM

- One module has one message RAM with fixed size
- Message RAM freely configurable for each CAN node within module
- TC39x has 3 modules (CAN0/CAN1/CAN2).

module	RAM	Register	
CAN0	F020,0000-F020,7FFF	F020,8000-F020,8FFF	
CAN1	F021,0000-F021,3FFF	F021,8000-F021,8FFF	
CAN2	F022,0000-F022,3FFF	F022,8000-F022,8FFF	

Agenda

1. CAN Protocol and Infineon Implementation
2. CAN module in AURIX2G MCMCAN
3. MCMCAN Interface and Interconnection
4. Module and Node Control
5. CAN FD

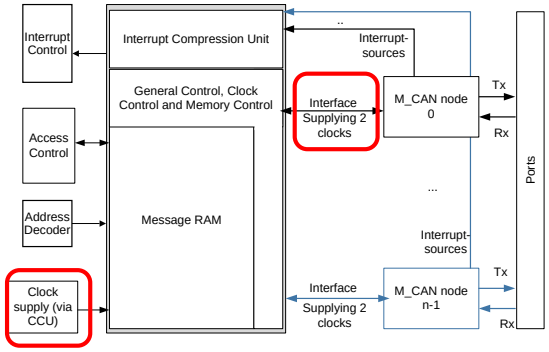
MCMCAN: Clock

- Module clock inputs

  module clock enable/disable via CANm_CLC (one bit for both clock inputs)

- Each node has an asynchronous and a synchronous clock input

  via register bitsfield CANm_MCR.CLKSELi



  - f_{SYN}: is used for inside MCMCAN for control logical and register
    - driven by f_{SPB} (max. 100MHz)
  - f_{ASYN}: is used for baud rate generation and it selectable from
    - f_{MCAN}: the peripheral clock (80, 40, 20 MHz)
    - f_{OSC0}: the oscillator input clock

Related register:

SCU_CCUCON0

SCU_CCUCON1

CANm_MCR

CANm_MCR.CLKSELi

MCMCAN: Port

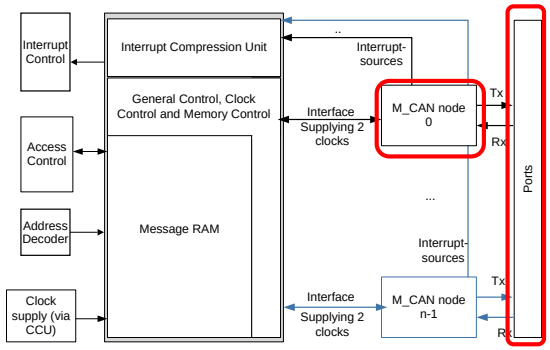
- Port pin and I/O lines control: via Input/output function (IOCR) and Pad driver (PDR)
- Node input/output
  - Input: via register CANm_NPCRi.RXSEL

    CAN0_NPCR0.RXSEL=010B; // P12.0 -> RXD_C

    P12_IOCR0.PC0= 00000B; // GPIO input

  - Output: via alternate output functions

    P12_IOCR0.PC1= 10101B; // P12.1 -> Alt5



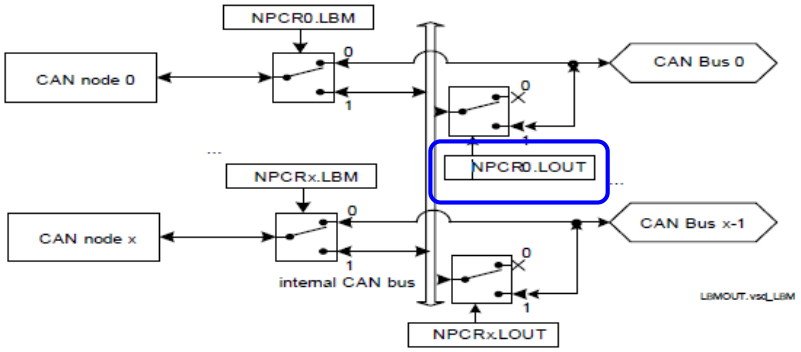
Note: up to 7 alternate functions (ALT1..ALT7) can be mapped to a single port pin.

Usually the MCMCAN transmit output pin uses Alt5.

- Special feature
  - module internal Loop back mode: bit LBM
  - Loop back mode out: bit LOUT

LOUT only in B-Step:

the internal loop back bus ←→ the external bus



Related register:

CANm_NPCRi

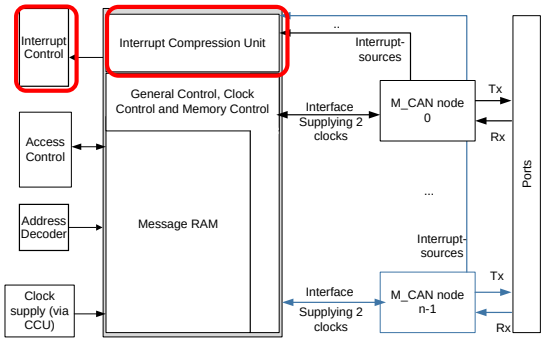
Pn_IOCRx

Pn_PDRx

MCMCAN: Interrupt

- Each **module** has 16 interrupt lines (INT_O0...INT_O15)
  - INT_Ox connects to SRC_CANmINTx (the service request control unit)

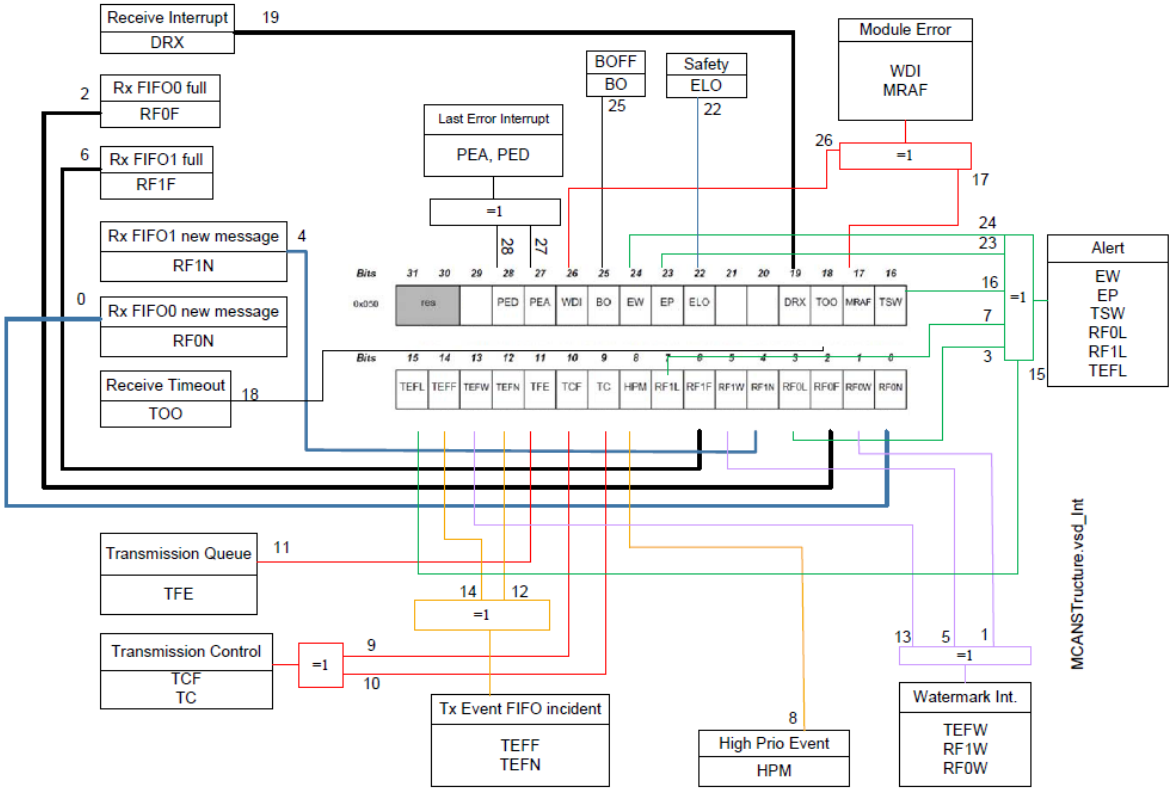
```
IfxSrc_init(&SRC_CAN1INT14, IfxSrc_Tos_cpu0, Priority_40); // Priority=40
IfxSrc_enable(&SRC_CAN1INT14); // set bit SRC_CANmINTx.SRE
```



- Each **module** contains an Interrupt Compressor Unit (see the next slides)
- Inside each CAN **node**
  - Interrupt event triggers interrupt flag in CANm_IRi(i=0..3) → total **28** flags each node
    - it is generated independent of signaling/status register CANm_ISREGi(i=0..3)
  - If its enable bit is set in CANm_IEi(i=0..3) ➔ can trigger INT_Ox (x=0..15)
    - The M_CAN always sets the interrupt flags in IR register when the respective condition occurs, But, only those interrupt flags that are enabled in IE register will be signaled via an interrupt line.
  - Each node has an interrupt signaling/status register: CANm_ISREGi(i=0..3) → total **16** interrupt group
    - ‚1' means at least one group member is showing an interrupt
    - Bit is cleared when all related bits in CANm_IRi(i=0..3) is cleared
  - Each interrupt group can be assigned to **16** interrupt lines via register CANm_GRINT1/2i(i=0..3)

MCMCAN: Interrupt Compressor Unit

- Interrupt Compressor Unit
  - total **28** Interrupt types each node → **16** status interrupt source each node



Alert watermark high prio event TxEventFIFO incident

Transmission control Trasmission Queue RxTimeout RxFIFO1 RxOK

RxFIFO0 RxOK RxFIFO1 full RxFIFO0 full Rx Interrupt

Last error BOFF/BO Saftey Module error

MCMCAN: Interrupt Assigning

› Example: TxBuffer0 (total 32 buffer) TxOK interrupt SW Interrupt

```
CAN0_N0_TX_BTIE.U = 0x00000001;  // TxBuffer0 to enable trigger flag CAN0_IR0.B.TC (bit9)
// TxBuffer (total 32) has own interrupt enable bit for TxOk/Cancellation
// bit IR.TCF (Transmission Cancellation Finished) will be set when CANm_TXBCIEi.TXBCIE=1
// bit IR.TC (Transmission Completed): flag will be set when CANm_TXBTIEi.TXBTIE=1

CAN0_N0_IE.B.TCE = 1;            // enable TCE (Tx completed interrupt)
CAN0_N0_GRINT2.B.TRACO = 14;     // enable INT_O14 line
```

› Example: RxFIFO0 level interrupt

```
CAN1_N0_IE.B. RF0WE = 1;         // RxFIFO0 level CAN1_IR0.RF0W (bit1)
CAN0_N0_GRINT1.B.WATI = 14;      // enable INT_O14 line
```

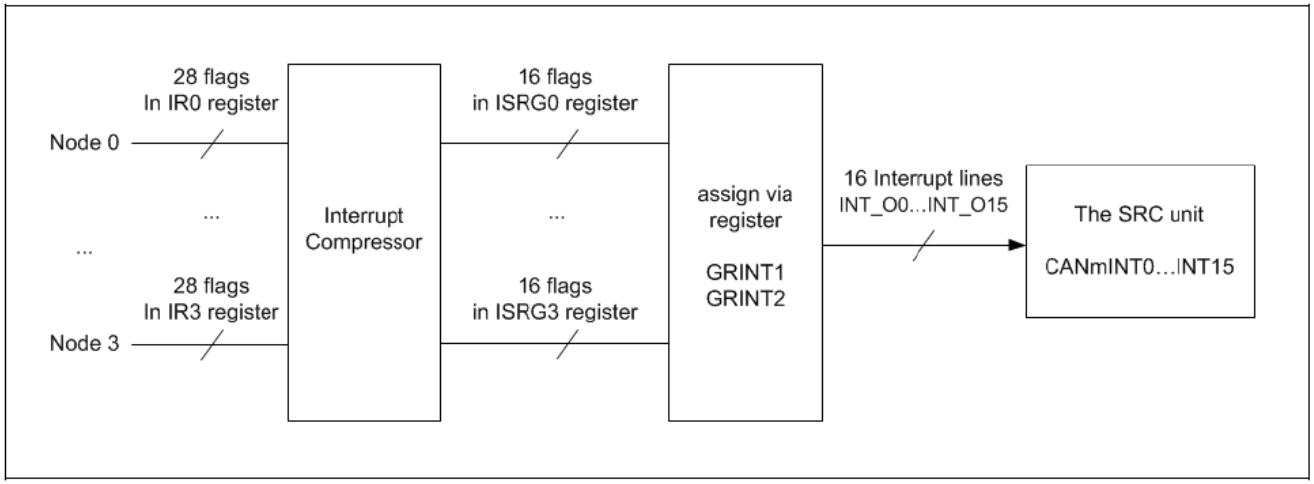
Related register:

CANm_IRi (m=module; i=node)

CANm_IEi (m=module; i=node)

CANm_ISREGi

CANm_GRINT1/CANm_GRINT2



Agenda

1. CAN Protocol and Infineon Implementation
2. CAN module in AURIX2G MCMCAN
3. MCMCAN Interface and Interconnection
4. Module and Node Control
5. CAN FD

MCMCAN: Global Control

- Module ID
- Module OCDS: OCS
- Access enable: CANm_ACCEN0/1
- clock control: CANm_CLC
  - switch on/off module clocks

CLC
CAN Clock Control Register (008000_H) Application Reset Value: $0000\ 0003_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0												EDIS	0	DISS	DISR
r												rw	r	rh	rw

Module Control Register (30_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCCE	CI	0		DXCM	NODE			0							
rw	rw	r		rw	rw			r							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								CLKSEL3		CLKSEL2		CLKSEL1		CLKSEL0	
r								rw		rw		rw		rw	

- Module control register: CANm_MCR
  - Special feature: select which CAN node is for timer measurement
  - Special feature: debug over CAN (via bit DXCM)
  - MCMCAN supports debugging using standard CAN tool access in parallel to regular CAN bus traffic.

- Module reset/release: CANm_KRST0/1
  - Reset module without powering down the controller completely

MCMCAN: Module Registers

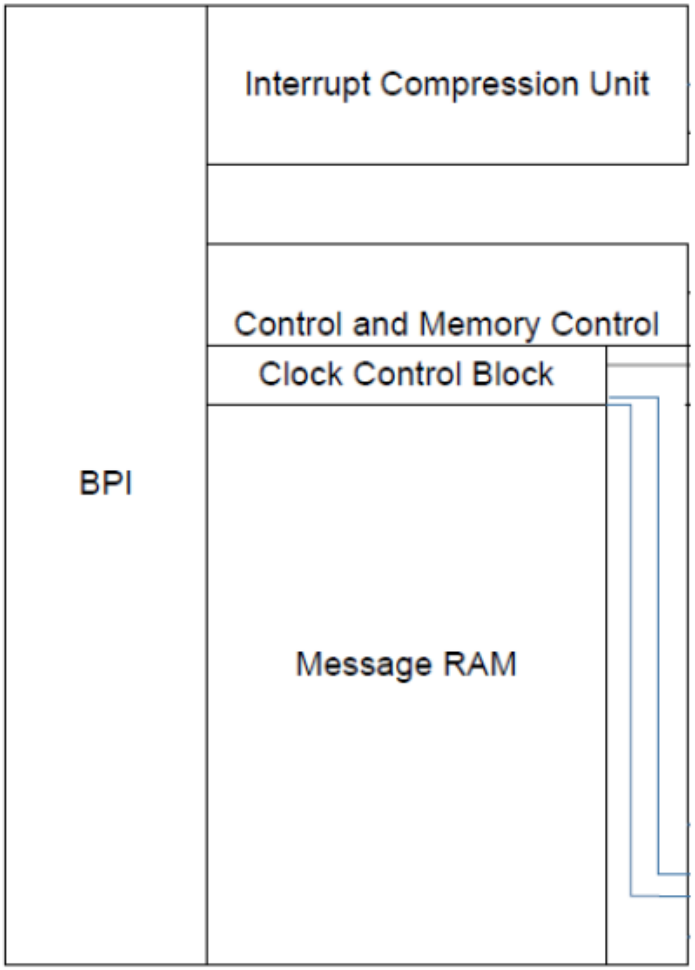
```
typedef volatile struct _Ifx_CAN
{
        Ifx_UReg_8Bit                   reserved_0[32768];      /**< \brief 0, \i
        Ifx_CAN_CLC                     CLC;                    /**< \brief 8000,
        Ifx_UReg_8Bit                   reserved_8004[4];       /**< \brief 8004,
        Ifx_CAN_ID                      ID;                     /**< \brief 8008,
        Ifx_UReg_8Bit                   reserved_800C[20];      /**< \brief 800C,
        Ifx_CAN_GRINT1                  GRINT1;                 /**< \brief 8020,
        Ifx_CAN_GRINT2                  GRINT2;                 /**< \brief 8024,
        Ifx_UReg_8Bit                   reserved_8028[8];       /**< \brief 8028,
        Ifx_CAN_MCR                     MCR;                    /**< \brief 8030,
        Ifx_CAN_BUFADR                  BUFADR;                 /**< \brief 8034,
        Ifx_UReg_8Bit                   reserved_8038[8];       /**< \brief 8038,
        Ifx_CAN_MECR                    MECR;                   /**< \brief 8040,
        Ifx_CAN_MESTAT                  MESTAT;                 /**< \brief 8044,
        Ifx_UReg_8Bit                   reserved_8048[144];     /**< \brief 8048,
        Ifx_CAN_ACCENCTR1               ACCENCTR1;              /**< \brief 80D8,
        Ifx_CAN_ACCENCTR0               ACCENCTR0;              /**< \brief 80DC,
        Ifx_UReg_8Bit                   reserved_80E0[8];       /**< \brief 80E0,
        Ifx_CAN_OCS                     OCS;                    /**< \brief 80E8,
        Ifx_CAN_KRSTCLR                 KRSTCLR;                /**< \brief 80EC,
        Ifx_CAN_KRST1                   KRST1;                  /**< \brief 80F0,
        Ifx_CAN_KRST0                   KRST0;                  /**< \brief 80F4,
        Ifx_CAN_ACCEN1                  ACCEN1;                 /**< \brief 80F8,
        Ifx_CAN_ACCEN0                  ACCEN0;                 /**< \brief 80FC,
        Ifx_CAN_N                       N[4];                   /**< \brief 8100,
        Ifx_UReg_8Bit                   reserved_8EFC[516];     /**< \brief 8EFC,
} Ifx_CAN;
```

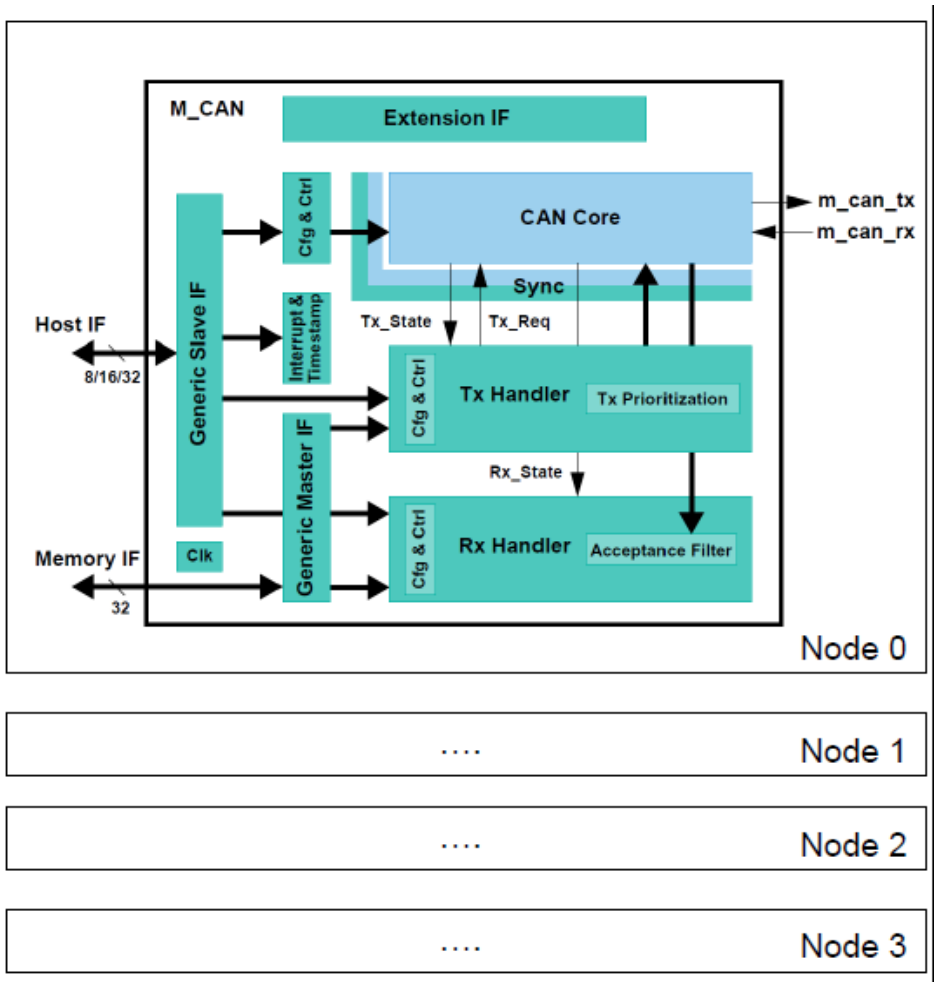
CAN node

CAN node M_CAN overview

- The M_CAN contains:
  - The CAN core (handling all 11898-1 protocol functions).
  - Clocks and synchronization control unit.
  - CAN protocol related configuration and control, such as bit timing, and an error counter for example.
  - Interrupt compression and enable unit.
  - Configuration and partition for message RAM.
  - Tx Handler:
    - Controls the message transfer from the message RAM to the CAN core.
    - Message prioritisation.
    - Transmission cancellation.
  - Rx Handler:
    - Controls the message transfer from the CAN core to the message RAM.
    - Restore message in different RAM areas.
    - Store additional information such as a timestamp for example.

M_CAN: Elements and Control units





M_CAN: Node Registers

```
typedef volatile struct _Ifx_CAN_N
{
    Ifx_CAN_N_ACCENNODE0_       ACCENNODE0_;    /**< \brief 0, A
    Ifx_CAN_N_ACCENNODE1_       ACCENNODE1_;    /**< \brief 4, A
    Ifx_CAN_N_STARTADR          STARTADR;       /**< \brief 8, S
    Ifx_CAN_N_ENDADR            ENDADR;         /**< \brief C, E
    Ifx_CAN_N_ISREG             ISREG;          /**< \brief 10,
    Ifx_UReg_8Bit               reserved_14[12]; /**< \brief 14,
    Ifx_CAN_N_NT                NT;             /**< \brief 20,
    Ifx_UReg_8Bit               reserved_34[12]; /**< \brief 34,
    Ifx_CAN_N_NPCR              NPCR;           /**< \brief 40,
    Ifx_UReg_8Bit               reserved_44[172]; /**< \brief 44,
    Ifx_CAN_N_TTCR              TTCR;           /**< \brief F0,
    Ifx_UReg_8Bit               reserved_F4[12]; /**< \brief F4,
    Ifx_CAN_N_CREL              CREL;           /**< \brief 100,
    Ifx_CAN_N_ENDN              ENDN;           /**< \brief 104,
    Ifx_UReg_8Bit               reserved_108[4]; /**< \brief 108,
    Ifx_CAN_N_DBTP              DBTP;           /**< \brief 10C,
    Ifx_CAN_N_TEST              TEST;           /**< \brief 110,
    Ifx_CAN_N_RWD               RWD;            /**< \brief 114,
    Ifx_CAN_N_CCCR              CCCR;           /**< \brief 118,
    Ifx_CAN_N_NBTP              NBTP;           /**< \brief 11C,
    Ifx_CAN_N_TSCC              TSCC;           /**< \brief 120,
    Ifx_CAN_N_TSCV              TSCV;           /**< \brief 124,
    Ifx_CAN_N_TOCC              TOCC;           /**< \brief 128,
    Ifx_CAN_N_TOCV              TOCV;           /**< \brief 12C,
    Ifx_UReg_8Bit               reserved_130[16]; /**< \brief 130,
    Ifx_CAN_N_ECR               ECR;            /**< \brief 140,
    Ifx_CAN_N_PSR               PSR;            /**< \brief 144,
    Ifx_CAN_N_TDCR              TDCR;           /**< \brief 148,
    Ifx_UReg_8Bit               reserved_14C[4]; /**< \brief 14C,
    Ifx_CAN_N_IR                IR;             /**< \brief 150,
    Ifx_CAN_N_IE                IE;             /**< \brief 154,
    Ifx_UReg_8Bit               reserved_158[8] ; /**< \brief 158,
    Ifx_UReg_8Bit               reserved_160[32]; /**< \brief 160,
    Ifx_CAN_N_GFC               GFC;            /**< \brief 180,
    Ifx_CAN_N_SIDFC             SIDFC;          /**< \brief 184,
    Ifx_CAN_N_XIDFC             XIDFC;          /**< \brief 188,
    Ifx_UReg_8Bit               reserved_18C[4]; /**< \brief 18C,
    Ifx_CAN_N_XIDAM             XIDAM;          /**< \brief 190,
    Ifx_CAN_N_HPMS              HPMS;           /**< \brief 194,
    Ifx_CAN_N_NDAT1             NDAT1;          /**< \brief 198,
    Ifx_CAN_N_NDAT2             NDAT2;          /**< \brief 19C,
    Ifx_CAN_N_RX                RX;             /**< \brief 1A0,
    Ifx_CAN_N_TX                TX;             /**< \brief 1C0,
    Ifx_UReg_8Bit               reserved_1FC[4]; /**< \brief 1FC,
    Ifx_CAN_N_TT                TT;             /**< \brief 200,
    Ifx_UReg_8Bit               reserved_244[444]; /**< \brief 244,
} Ifx_CAN_N;
```

```
typedef volatile struct _Ifx_CAN_N_TX
{
    Ifx_CAN_N_TX_BC             BC;             /**< \brief 0, T
    Ifx_CAN_N_TX_FQS            FQS;            /**< \brief 4, T
    Ifx_CAN_N_TX_ESC            ESC;            /**< \brief 8, T
    Ifx_CAN_N_TX_BRP            BRP;            /**< \brief C, T
    Ifx_CAN_N_TX_BAR            BAR;            /**< \brief 10,
    Ifx_CAN_N_TX_BCR            BCR;            /**< \brief 14,
    Ifx_CAN_N_TX_BTO            BTO;            /**< \brief 18,
    Ifx_CAN_N_TX_BCF            BCF;            /**< \brief 1C,
    Ifx_CAN_N_TX_BTIE           BTIE;           /**< \brief 20,
    Ifx_CAN_N_TX_BCIE           BCIE;           /**< \brief 24,
    Ifx_UReg_8Bit               reserved_28[8]; /**< \brief 28,
    Ifx_CAN_N_TX_EFC            EFC;            /**< \brief 30,
    Ifx_CAN_N_TX_EFS            EFS;            /**< \brief 34,
    Ifx_CAN_N_TX_EFA            EFA;            /**< \brief 38,
} Ifx_CAN_N_TX;
```

for CAN frame transmission

```
typedef volatile struct _Ifx_CAN_N_RX
{
    Ifx_CAN_N_RX_F0C            F0C;            /**< \brief 0,
    Ifx_CAN_N_RX_F0S            F0S;            /**< \brief 4,
    Ifx_CAN_N_RX_F0A            F0A;            /**< \brief 8,
    Ifx_CAN_N_RX_BC             BC;             /**< \brief C,
    Ifx_CAN_N_RX_F1C            F1C;            /**< \brief 10,
    Ifx_CAN_N_RX_F1S            F1S;            /**< \brief 14,
    Ifx_CAN_N_RX_F1A            F1A;            /**< \brief 18,
    Ifx_CAN_N_RX_ESC            ESC;            /**< \brief 1C,
} Ifx_CAN_N_RX;
```

for CAN frame reception

M_CAN: Node Control/Status

- Node control register
  - INIT bit: enable/disable CAN transfer on this node
  - CCE bit: enable/disable configuration mode
  - BRSE: bit rate switch on/off
  - FDOE: CAN FD enable/disable

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NISO	TXP	EFBI	PXHD	0	0	**BRSE**	**FDOE**	TEST	DAR	MON	CSR	CSA	ASM	**CCE**	**INIT**

- Node status register
  - INIT bit: enable/disable CAN transfer on this node
  - CCE bit: enable/disable configuration mode
  - BRSE: bit rate switch on/off
  - FDOE: CAN FD enable/disable

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
									TDCV						
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	PXE	RFDF	RBRS	RESI	**DLEC**			**BO**	EW	EP	ACT		**LEC**		

M_CAN: Node Bit Timing

- The data bit rate (if CAN FD is required)
  - According to ISO the time quanta for the data bit timing shall be programmable at least 5-25 Tq
  - Tq= (1 + DBRP) / f_{CAN}
  - Bit timing = 1 + (DTSEG1 + 1) + (DTSEG2 + 1) x Tq

Data bit timing Register DBTP:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								TDC	0		DBRP				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			DTSEG1					DTSEG2				DSJW			

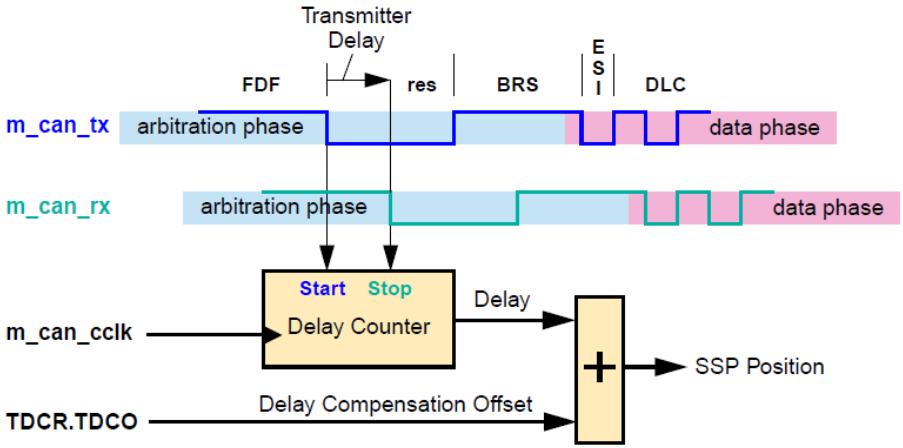
- The nominal bit rate
  - According to ISO the time quanta for the nominal bit timing 8-80 Tq.
  - Tq= (1 + NBRP) / f_{CAN}
  - Bit timing = 1 + (NTSEG1 + 1) + (NTSEG2 + 1) x Tq

Nominal bit timing Register NBTP:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NSJW							NBRP								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NTSEG1							NTSEG2								

M_CAN: Transmitter Delay Compensation

- Delay compensation for higher **data bit rate** independent of the delay in CAN transceiver and port delay
- described in detail in the new ISO11898-1



Related register:

CANm_DBTPi.TDC=1: enabled

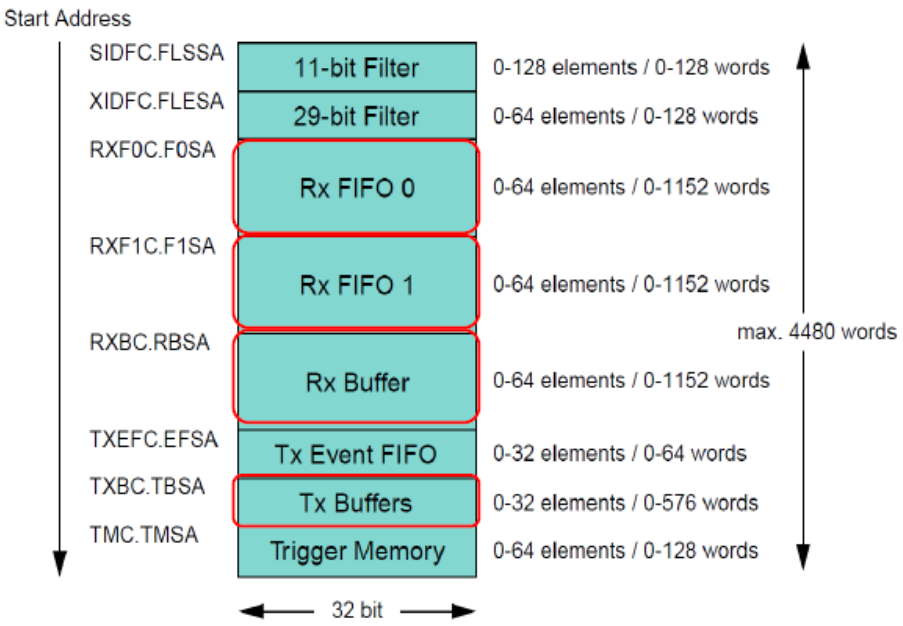
CANm_TDCRi.TDCO: offset (measured in Tq)

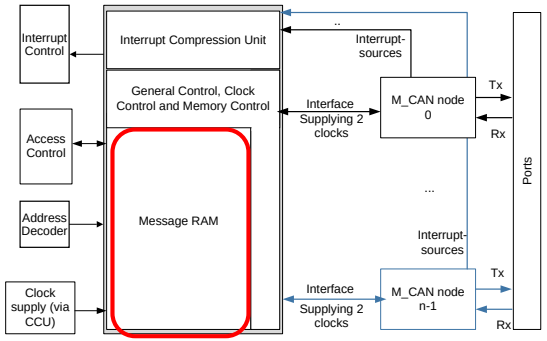
CANm_TDCRi.TDCF: min. value for the SSP position

CANm_PSRi.TDCV: actual value (Tq)

M_CAN: Message RAM for each Node

› Max. configurable elements for one CAN node





- The size of each element can be individually configured.
- The RAM structure is defined via the start address of different element blocks.
- Total memory defined for message RAM depends on the byte sizes of each element.

Note: It is not necessary to configure each of the sections, and there is no restriction with respect to the sequence of the element block.

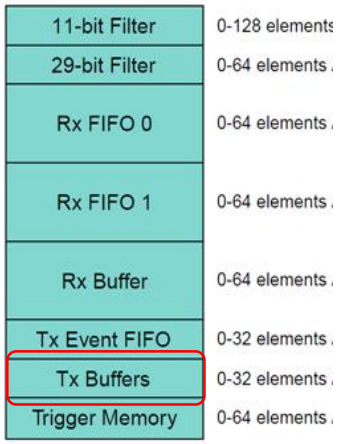
M_CAN: Frame Transmission

A. Message RAM to transmit CAN frame (Configuration)

B. Handling of Transmit(Tx Handler and Tx prioritization)

Transmission: Overview

- Each CAN **node** supports a maximum of 32 Tx Buffers and can be configured as
  - Tx FIFO
  - Tx Queue
  - Dedicated Tx Buffers
  - Combination of dedicated Tx Buffers with Tx FIFO
  - Combination of dedicated Tx Buffers with Tx Queue
- One DataFieldSize for each CAN **node**
- The Tx Handler controls prioritization/transmission/cancelation of Tx requests



Related register:

CANm_TXBCi: Tx buffer configuration (mode, size, start address)

CANm_TXESCi: byte size configuration (8/12/16/20/24/32/48/64 bytes)

CANm_TXBRPi: Tx Buffer RqPending (‘rh’)

CANm_TXBARi: Tx Buffer Add Request

CANm_TXBCRi: Tx Buffer Cancellation Request

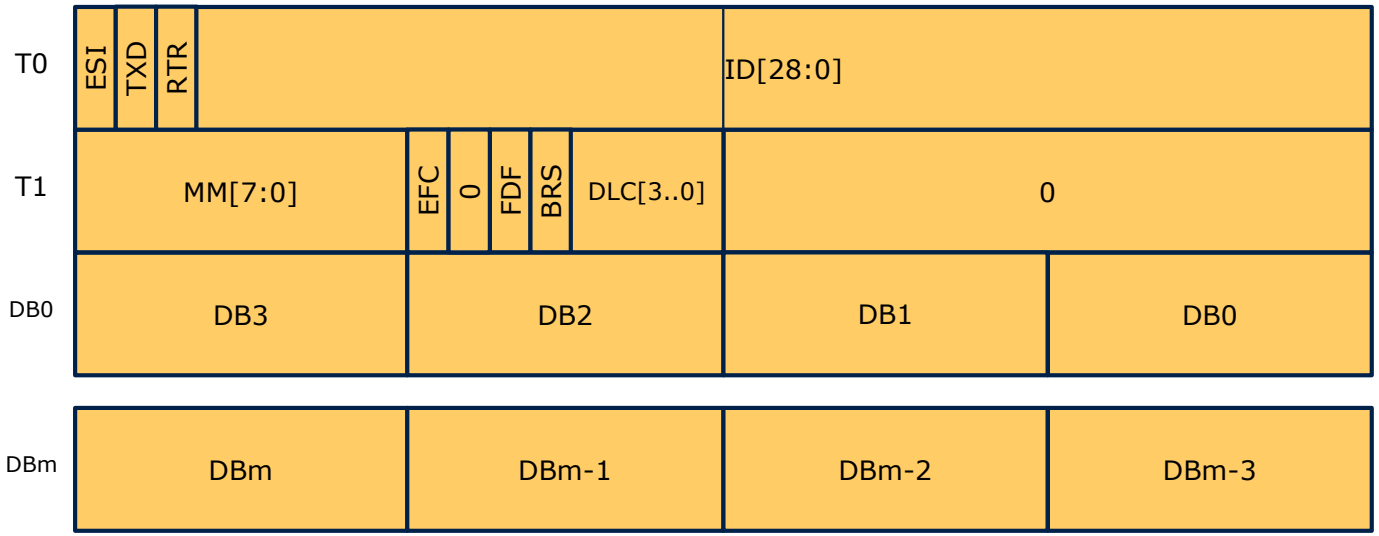
CANm_TXBTIEn: Tx Buffer transmission interrupt enable

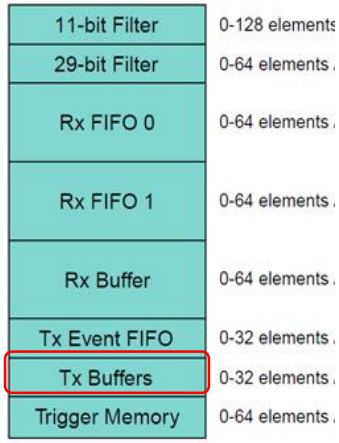
CANm_TXBCIEn: Tx Buffer transmission Cancellation finished interrupt enable

A. Element Structure: TxBuffer

- A single TxBuffer element has size of (8+TXESC.TBDS) words

  *for example, a 64 byte data field size: (2 * 4 + 64) = 72 bytes or 18 words*





XTD: extended ID RTR: remote trigger request ID: Identifier DLC: data length code

MM: message marker EFC: event FIFO control FDF: FD format BRS: bit rat switching

Register name:

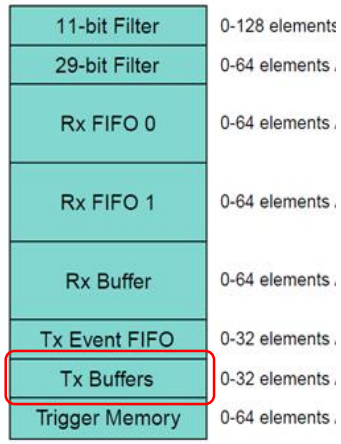
CANm_TxMsgk_T0(k=0..63)

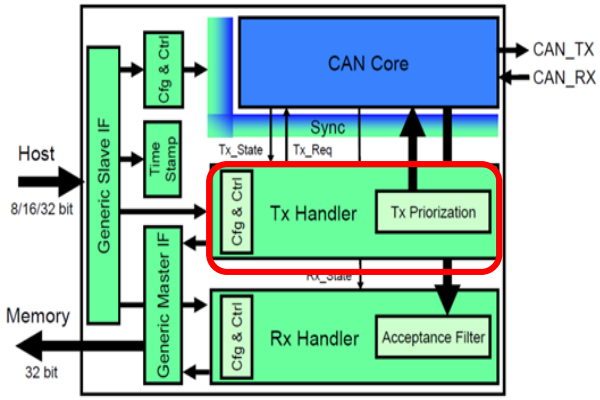
CANm_TxMsgk_T1(k=0..63)

CANm_TxMsgk_DBm(m=0..63)

B. Tx Handling Overview

- Functionality
  - Handles transmission for TxBuffer (TxBuffer/TxFIFO/TxQueue and mix modes)
  - Hardware Put/GetIndex handling for FIFO and Queue mode (see the next slide)
  - Control Tx Prioritization
  - Cancellation of a transmit request
- Special
  - Transmit pause
  - Transmit cancellation
  - Tx Event Handling: RAM TxEvent FIFO element
    - To store timestamp





B. Tx Handler Method

- Tx handling is depending on the selected mode
  - Dedicated mode:
    - Each element has a specific ID (several elements can have a same ID), lower ID → higher priority
    - If the data bytes has been updated, a transmission is requested by an 'add request' in TXBARi.Arx
  - Tx FIFO mode:
    - Tx sending starts the TXFQSi.**TFGI**(GetIndex) pointed element
    - SW writing accesses the TXFQSi.**TFQPI**(PutIndex) pointed element
    - Tx trigger requests to set related bit in TXBARi.ARx
  - Tx Queue mode:
    - Tx sending starts: TxBuffer with lowest ID → highest priority
      (same ID for multiple TxBuffer: lower number → higher priority)
    - SW writing accesses the TXFQSi.**TFQPI** pointed element
    - Tx trigger requests to set related bit in TXBARi.ARx

HW: calculates the GetIndex/PutIndex

B. Tx Prioritization

- Configuration/Prioritization : max. 32 Tx Buffers can be configured as
  - Dedicated Tx Buffers: the lowest ID wins
  - Tx FIFO: the first/oldest wins
  - Tx Queue: the lowest ID wins
  - dedicated Tx Buffers + Tx FIFO:
    - Get the lower ID in the dedicated Txbuffer and GetIndex pointer element
    - The lower ID in the both element → higher Prio.

      the lowest ID from (the first in FIFO & all elements in TxBuffer)
  - dedicated Tx Buffers +Tx Queue:
    - Get all elements with activated Arx
    - Low ID → higher Prio.

      the lowest ID from (all elements in Queue & TxBuffer)

Note: In case of multiple TxBuffers/Queues with same message ID → the lowest buffer number has the highest priority.

Note: it is better to use TxFIFO (instead of TxQueue) in case to transmit frames with same ID

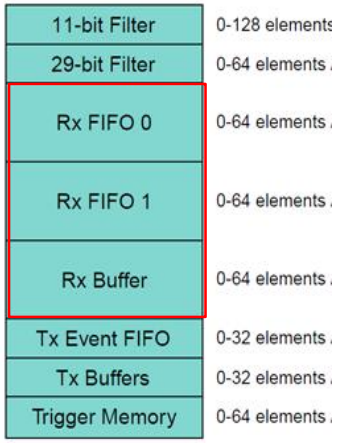
M_CAN: Frame Reception

A. Message RAM to store received CAN frame (Configuration)

B. Handling of reception (Rx Handler and Acceptance Filtering)

Reception: Rx Element

- Each CAN node provides RxFIFO0, RxFIFO1, dedicated RxBuffer;
  - Each block can configure byte size (8,12, 16, 20, 24, 32, 48, 64 bytes) individually
- Each CAN node supports 2 filter sets: one for 11_bit IDs/one for 29_bit IDs
  - Handles remote/data frame, non-matching frames…
- The Rx Handler controls
  - the acceptance filtering
  - the transfer of received frame to RxFIFO0/1 or dedicated RxBuffer
  - the Put/Get Indices



Related register:

CANm_RXBCi(i=0..3): Rx buffer configuration(start address)

CANm_RXESCi(i=0..3): Rx Buffer/FIFO element size configuration :

CANm_NDAT1/2: new data flag for 64 dedicated RxBuffer

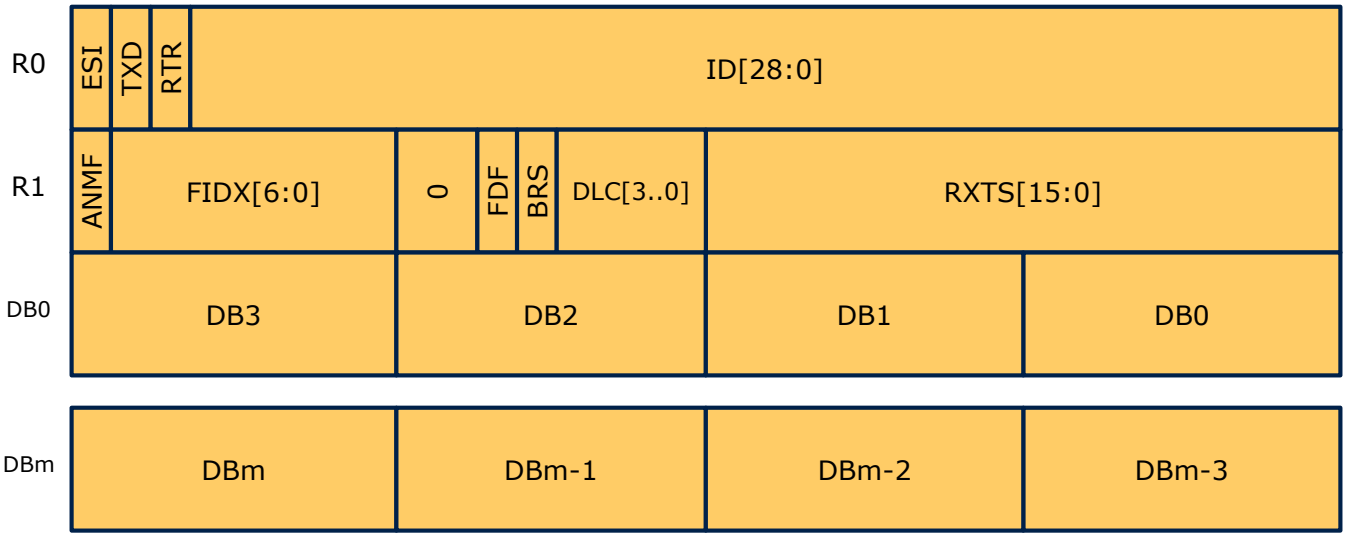
CANm_RXF0/1Ci:RxFIFO0/1 Configuration (mode, size, watermark level, start address)

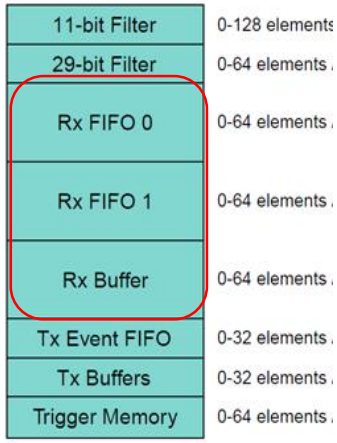
CANm_RXF0/1Si: RxFIFO0/1 Status (lost, full, level, put/get index)

CANm_RXF0/1Ai:RxFIFO0/1 Acknowledge index

A. Element structure: RxBuffer/2*RxFIFO

- RxBuffer and RxFIFO0/1 has the same structure
- RxBuffer/RxFIFO0/RxFIFO1 has own data bytes size





ESI: error indicator XTD: extended ID RTR: remote trigger request ID: Identifier

ANFM: accepted no-matching frame FIDX: filter index FDF: extended data length

BRS: bit rate switch DLC: data length code RXTS: Rx timestamp

Register name:

CANm_RxMsgk_R0(k=0..63)

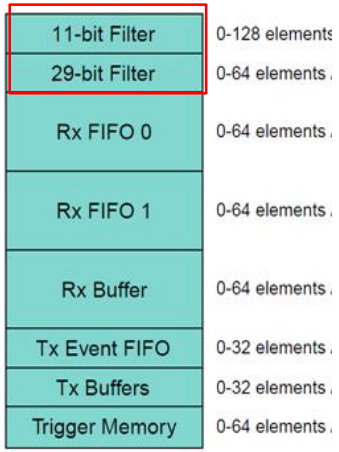
CANm_RxMsgk_R1(k=0..63)

CANm_RxMsgk_DBm(m=0..63)

Reception: Rx Filtering

- Each CAN node supports 2 filter sets: one for 11_bit IDs/one for 29_bit IDs
  - Handles remote/data frame, non-matching frames…

- Each CAN node has a Global filter register
  - no-matched 11-bit IDs and 29-bit IDs frames to be stored into RxFIFO0/1
  - Receive/reject 11-bit IDs and 29-bit IDs remote frames



Related register:

CANm_GFCi: Global filter Configuration

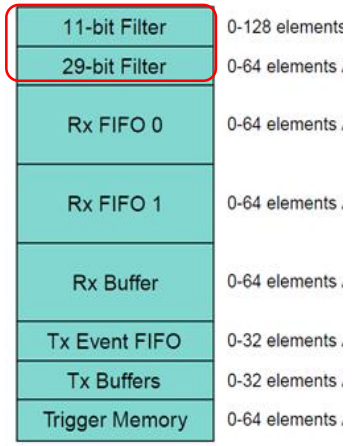
CANm_SIDFCi: 11-bit ID Filter configuration (**max. 128**) (size and start address)

CANm_XIDFCi: 29-ID Filter configuration (**max. 64**) (size and start address)

CANm_XIDAMi[28:0]: 29 bits ID Mask (used 29-bits IDs)

A. Filter Element Structure:

› For the 11_bits IDs frame



SFID2: standard filter ID2 SFID1: standard filter ID1

SFEC: standard filter element configuration SFT: standard filter types

Register name:

CANm_StdMsgk_S0(k=0..127)

› For the 29-bits IDs frame

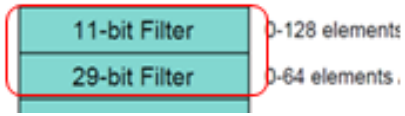
EFID2: extended filter ID2 EFT: extended filter type

Register name:

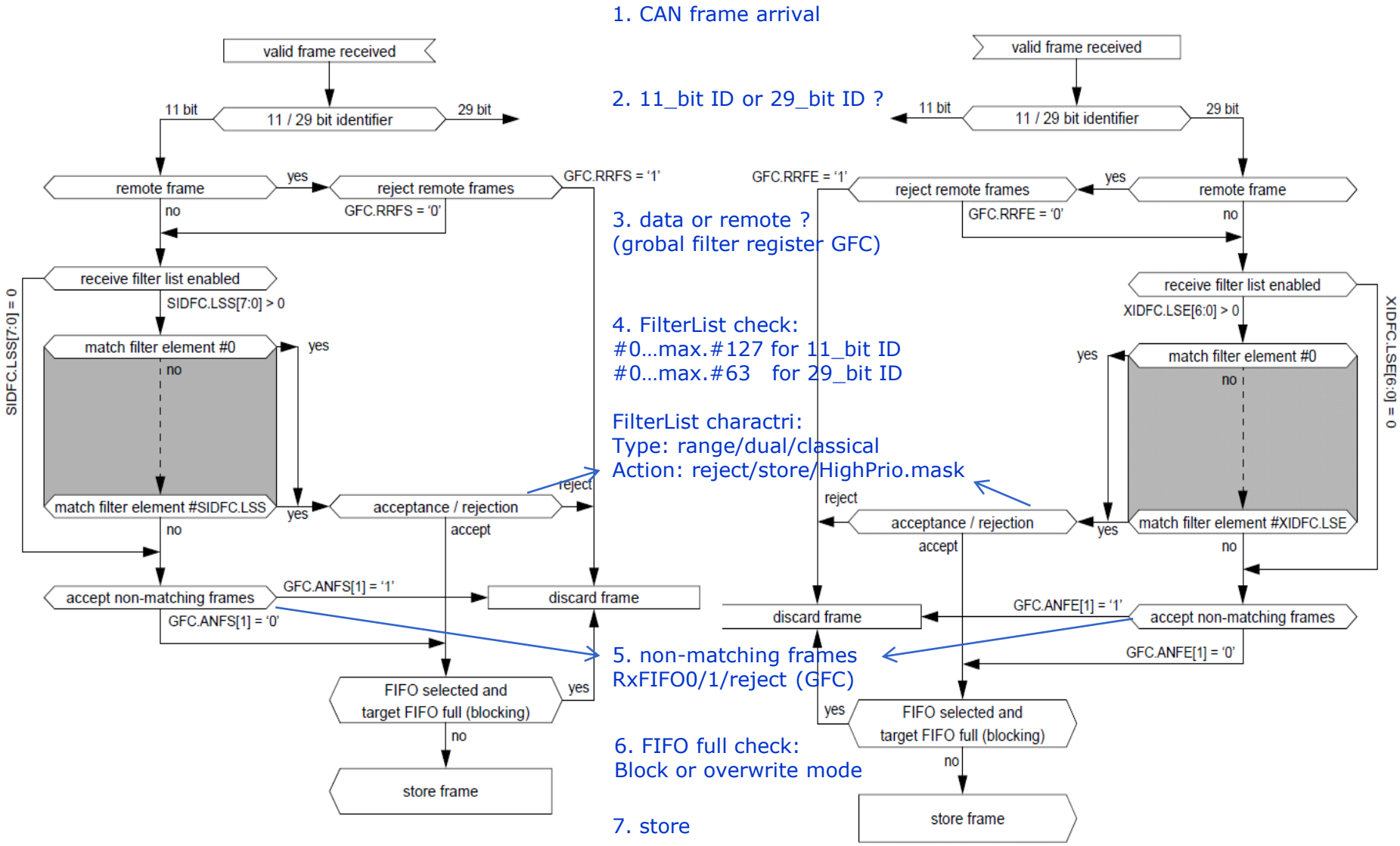
CANm_StdMsgk_S0(k=0..127)

B. 2 Filter Sets

- Acceptance filtering in hardware
  - **2 filter sets**: one for 11_bit IDs/one for 29_bit IDs
  - Each filter element **mode**:
    - range filter: for 11-bit/29-bit IDs
    - dedicated ID: one element for one/two IDs for 11-bit/29-bit IDs
    - classic bit masker: SFID1/EFID1 as ID; SFID2/EFID2 as mask
  - Each filter element: acceptance/rejection filtering
  - Each filter element can be enabled / disabled individually
  - Filters are checked sequentially starting at element #0; execution stops with the first matching element
  - **Store action** is depending on the element bits field S0.SFEC(11-bit IDs)/E0.EFEC(29-bit IDs)
    - Store received frame in RxFIFO0/1 or a dedicated RxBuffer
    - Reject received frame
    - Set High Priority Message interrupt flag **IR.HPM**
    - Set High Priority Message interrupt flag **IR.HPM** and store received frame in FIFO0/1
    - Store received frame in an Rx buffer or as a debug message



B. Filter Handling Flow



Agenda

1. CAN Protocol and Infineon Implementation
2. CAN module in AURIX2G MCMCAN
3. MCMCAN Interface and Interconnection
4. Module and node control
5. CAN FD

CAN FD support and Implementation

- CAN FD support consists:
  - Baud rate switching (higher baud rate in data bytes)
    - Max. nominal bit rate = 1Mbp; max. data bit rate = 5M bps
  - More data bytes
    - Max. 8 data bytes for classical CAN mode; 64 data bytes for CAN FD mode
  - CAN FD node is able to send classical CAN message
- CAN FD implementation in MCMCAN
  - Configuration of CAN FD is integrated into each M_CAN node
    - CAN FD: enable via bit CCCR.FDOE and BRSE (evaluated by FDOE='1')
    - Nominal & data bit timing configuration
    - each CAN node supports 'ISO CAN FD' and 'no ISO CAN FD' (via CCCR.NISO)
- Sending CAN FD or classical CAN frames is selectable in each element
  - More flexible: the element (TxBuffer/RxBuffer/RxFIFO) has its bit FDF and BRS

CCCR		Tx Buffer Element		Frame Transmission
BRSE	FDOE	FDF	BRS	
ignored	0	ignored	ignored	Classical CAN
0	1	0	ignored	Classical CAN
0	1	1	ignored	FD without bit rate switching
1	1	0	ignored	Classical CAN
1	1	1	0	FD without bit rate switching
1	1	1	1	FD with bit rate switching

CAN FD Support: Format Compatibility

Tx node / Rx node	**Node setting:** **FDOE=0** **Tx node behavior:** **a. Classical CAN**	**Node setting:** **FDOE=1 (CAN FD)** **BRSR=0 (without BRS)** **Tx node behavior:** **a. Classical CAN** **b. Long frame**	**Node setting:** **FDOE=1 (CAN FD)** **BRSR=1 (with BRS)** **Tx node behavior** **a. classical CAN** **b. long frame** **c. long + fast frames**
Node setting: FDOE=1 (CAN FD) Receive node behavior: - Classical CAN - Long frames - Long + fast frames	a. classical CAN ← Rx node: ACK	a. Classical ← Rx node: ACK b. Long frame ← Rx node: ACK	a. Classical ← Rx node: ACK b. Long frame ← Rx node: ACK c. Long + fast frame ← Rx node: ACK
Rx node: FDOE=0 (Classical CAN) Receive node behavior: - Classical CAN Frames	a. classical CAN ← Rx node: ACK	a. Classical ← Rx node: ACK b. Long frame ← Rx node: error frame	a. Classical ← Rx node: ACK b. Long frame ← Rx node: error frame c. Long + fast frame ← Rx node: error frame

Part of your life. Part of tomorrow.

Infineon