XMC4000

32-bit Microcontroller Series for Industrial Applications

Controller Area Network Controller (MultiCAN)

AP32300

Application Note

About this document

Scope and purpose

The Infineon MultiCAN module contains independently operating CAN nodes with full-CAN functionality to meet the ISO11898_1 standard.

This document combines a brief overview of the MultiCAN module in the XMC4000 family with a more detailed operation description of different features, such as:

- FIFO
- Gateway
- Baudrate detection

Note: Depending on the configuration of each microcontroller derivative which includes the MultiCAN module, the number of nodes and message objects might be different.

Applicable Products

- XMC4000 Microcontroller Family

References

The example code “AP32300_XMC4000_MultiCAN_SW” can be downloaded from **www.infineon.com/XMC**.

For Application Kits and DAVE™, please refer to **www.infineon.com/xmc-dev**.

Table of Contents

About this document **1**

Table of Contents **2**

1 CAN protocol and the Infineon CAN module **3**
1.1 Typical CAN node structure 5
1.2 CAN frame format 5
1.3 CAN node station 6

2 MultiCAN in the XMC4000 family **7**
2.1 MultiCAN structure 7
2.1.1 Node Control Unit 8
2.1.2 The Linked List Controller 10
2.1.3 The Interrupt Control Unit 11
2.1.4 Node Receive input selection 12
2.1.5 Message controller 12
2.1.6 Interfaces and interconnects 12
2.1.7 Bit timing 14
2.2 Hardwaer handling of CAN frame reception and transmission 16
2.3 Programming the MultiCAN module 17
2.3.1 Software initialization of MultiCAN 17
2.3.2 Software handling on CAN node 18
2.3.3 Software control of a message transfer 19
2.4 MultiCAN special features 22
2.4.1 Loop-back mode 22
2.4.2 CAN analyzer mode 22
2.4.3 MultiCAN FIFO 22
2.4.4 MultiCAN Gateway 24

3 Implementing the example **27**
3.1 First steps 27
3.2 Example_1: Standard Message Object Transmission and Receive 27
3.3 Example_2: Using Receive FIFO 27
3.4 Example_3: Using Transmit FIFO 27
3.5 Example_4: Using Gateway without FIFO 27
3.6 Example_5: Using Gateway with FIFO 28
3.7 Example_6: Baudrate detection 28

4 Running example code **30**

5 DAVE™ and XMCLib **32**
5.1 Implementation with XMCLib 32

6 Appendix **34**
6.1 Kit information 34

7 Revision History **35**

1 CAN protocol and the Infineon CAN module

CAN is a multi-master bus system with broadcasting capability. In the CAN protocol, the bus nodes do not have a specific address. Instead, the address information is contained in the identifiers of the transmitted messages, indicating the message content and the priority of the message for arbitration.

CAN is a low cost protocol for real-time applications with a high reliability. Nodes can be easily connected or disconnected without disturbing the communication of the other nodes.

Uses

The CAN bus is most widely used in the automotive and industrial market segments:

- CAN is used in the automotive industry to enable data exchange between ECUs over the complete car.
- CAN is used in industrial automation to connect control units, sensors, and actuators for example. There are several CAN-based high-layer protocols, including DeviceNet, CANopen, and J1939, which are internationally standardized and their networks are used in width application fields.
- CAN is used for initialization, program and parameter up-/download, exchange of rated values / actual values, and diagnosis for example, in end-of-line or on-the-fly of software updates.

CAN protocol specification and history

- 2.0A; specify "Standard CAN" - 11 bit message ID’s, total 2048 ID’s available.
- 2.0B; specify “Extended CAN” – 29 bit message ID’s, more than 536 million ID’s available.
- ISO11898-1 as successor of 2.0B.
- TTCAN: Time-triggered communication on CAN.
- CAN FD integration into ISO11898-1.

Table 1 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. - Analyzer Mode.					
MultiCAN	XC800 XC2000/XE166 XMC4000 TriCore	Max. 8	Max. 256 (flexible assigned)	yes	on certain devices	-
	- Message objects can be individually/dynamically assigned to one of the CAN nodes. - FIFO/Gateway buffer size is programmable. - Analyzer mode and bit timing mode. *Note: The number of nodes, message objects and TTCAN features of MultiCAN module in each device are varied. For details please refer to the appropriate Data Sheet.*					
MultiCAN+	AURIX	Max. 8	Max. 256 (flexible assigned)	yes	on certain devices	yes
	- ISO CAN FD					

1.1 Typical CAN node structure

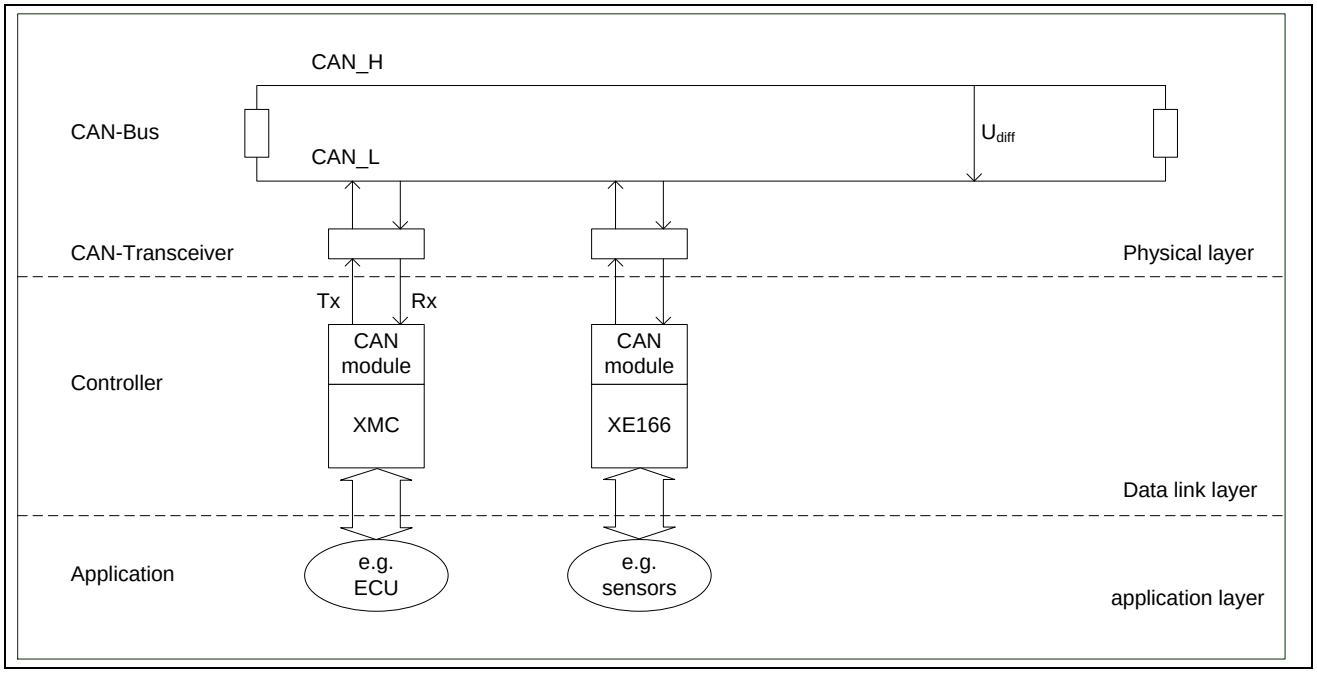

Figure 1 Typical CAN node structure

CAN is insensitive to electromagnetic interference. The maximum CAN bus speed is 1 MBaud, which can be achieved with a bus length of up to 40 meters when using a twisted pair wire.

1.2 CAN frame format

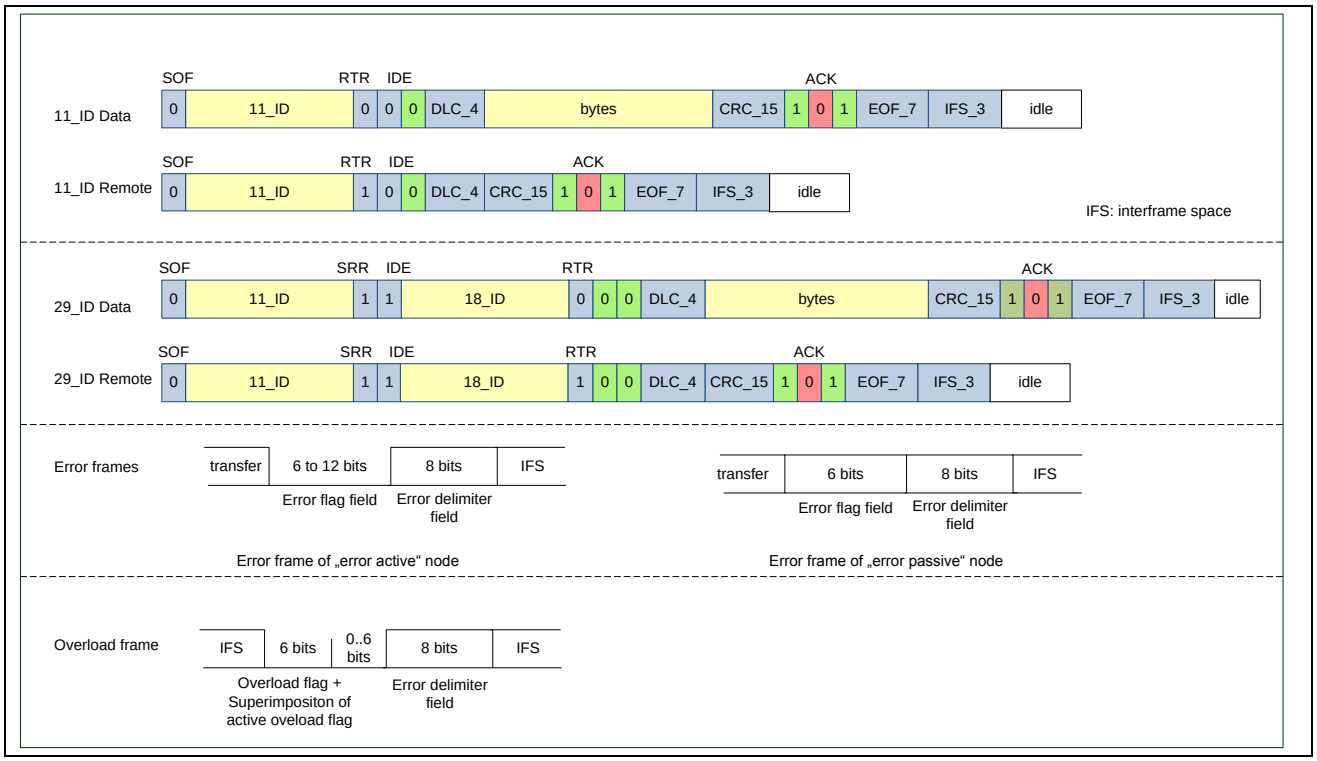

Figure 2 CAN frame formats (Data, Remote, Error, Overload frame)

1.3 CAN node station

- Each CAN node can enter one of three error states , according to the value of their internal error counters:
  - error active
  - error passive
  - bus off
- Each CAN node implements one receive and one transmit error counter. Counting increases or decreases according to ISO 11898-1.
- The error-active state is the usual state after reset. The bus node can then receive and transmit messages and transmit active Error Frames (made of dominant bits) without any restrictions. An ‘error-active’ node may access the bus as soon as the bus is free.
- In the error-passive state, messages can still be received and transmitted, although, after transmission of a message the node must suspend transmission. It must wait 8 bit times longer than error-active nodes before it may transmit another message. In terms of error signaling, only passive Error Frames (made of recessive bits) may be transmitted by an error-passive node.
- In the bus-off state it is temporarily impossible for the node to participate in the bus communication. During this state, messages can be neither received nor transmitted.

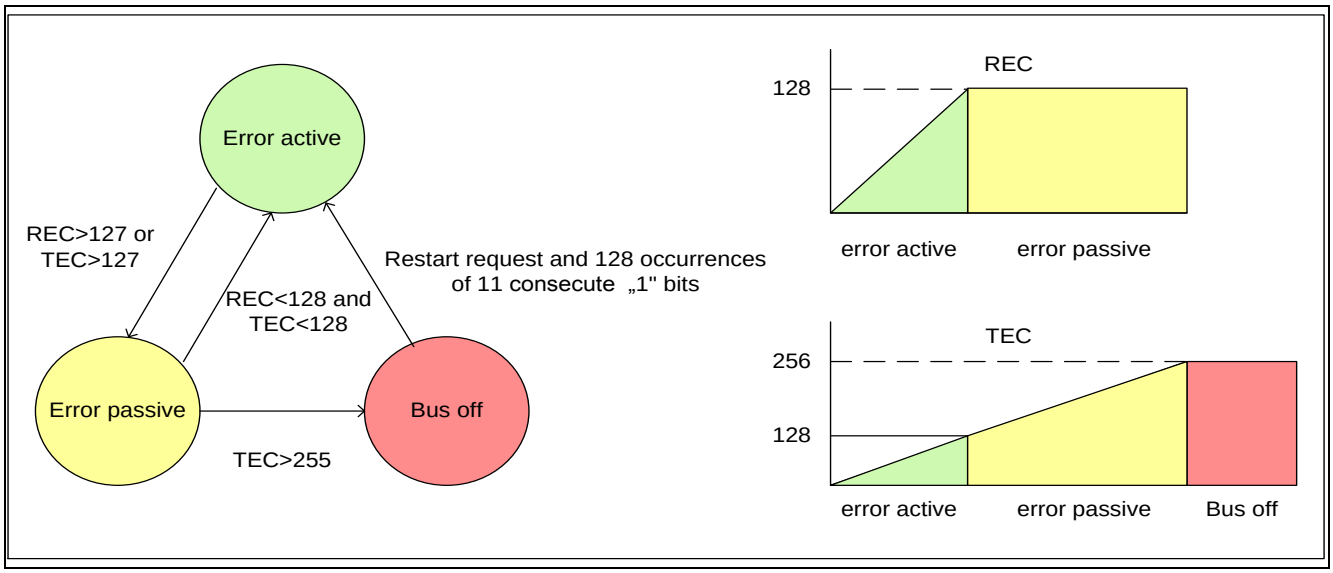

Figure 3 CAN node stations (Error active, Error passive, Bus off)

2 MultiCAN in the XMC4000 family

The MultiCAN module in the XMC4000 family supports the following functionality:

- Compliant with classical CAN as defined in ISO 11898-1.
- Baudrate programmable for each node.
- FIFO/Gateway functionality.
- Acceptance mask filtering for each MO.
- Frame counter of each CAN node is selectable for frame count, the time stamp, or the bit timing mode.
- Freely assignable interrupt sources to interrupt nodes. 8 interrupt output lines are available.
- Prioritization of message objects on ID or list number.

Table 2 XMC4000 derivatives with MultiCAN implementation (Refer also to Data Sheet)

Derivative	Package	Nodes (max.)	Message Objects (max.)	Max. f_{CAN} (f_{CLC})= f_{PERIPH}	Module Address Space
XMC4500	PG-LFBGA-144 PG-LQFP-144 PG-LQFP-100	3	64	120MHz	$4801\ 4000_H$ - $4801\ 7FFF_H$
XMC440x	PG-LQFP-100 PG-LQFP-64	2	64	120MHz	
XMC4200 XMC4100	PG-LQFP-64 PG-VQFN-48	2	64	80MHz	
XMC4108	PG-LQFP-64 PG-VQFN-48	1	32	80MHz	

2.1 MultiCAN structure

The MultiCAN module consists of:

- Node Control Unit.
- The Linked ListController.
- The Interrupt Control Unit.
- Node Receive Input Selection.
- Message Controller.
- Interfaces and Interconnects.

2.1.1 Node Control Unit

Each CAN node consists of several sub-units:

- Bitstream processor.
  - Performs data, remote, error, and overload frame processing according to the ISO 11898 standard.
  - It checks bus idle, adds SOF and EOF, controls the CRC generation and arbitration procedure, and monitors ACK slot. If an error mismatch is detected an error event is generated in register NSRx.
- Error handling unit.
  - Performs Tx/Rx error counter and sets node into an error-active/-passive and bus-off state according to the ISO 11898 standard.
- Bit timing unit.
  - For baudrate detection and resynchronization.
- Node control bits.
  - To enable/disable CAN transfer on this node.
- Interrupt control unit.
  - For node-specific events.

Node control register NCRx (x=0, 1, 2)

- INIT (‘wrh’)
  - Set by hardware when the CAN node enters the bus-off state.
  - It is cleared by software to enable the participation of the node in the CAN traffic.
- CCE (Configuration Change Enable).
  - Register NBCTRx, NECNTx, and NPCRx can be written to only when CCE=1.

Node status register NSRx (x=0, 1, 2)

- ALERT (alert warning). An ALERT is set when:
  - A change of bit NSRx.BOFF.
  - A change of bit NSRx.EWRN.
  - A List Length Error, which also sets bit NSRx.LLE.
  - A List Object Error, which also sets bit NSRx.LOE.

Note: Flag ALERT must be reset by software (write 0).

- EWRN and BOFF (Bus-off).
  - Both are ‘rh’ bits.
  - They are updated by hardware.

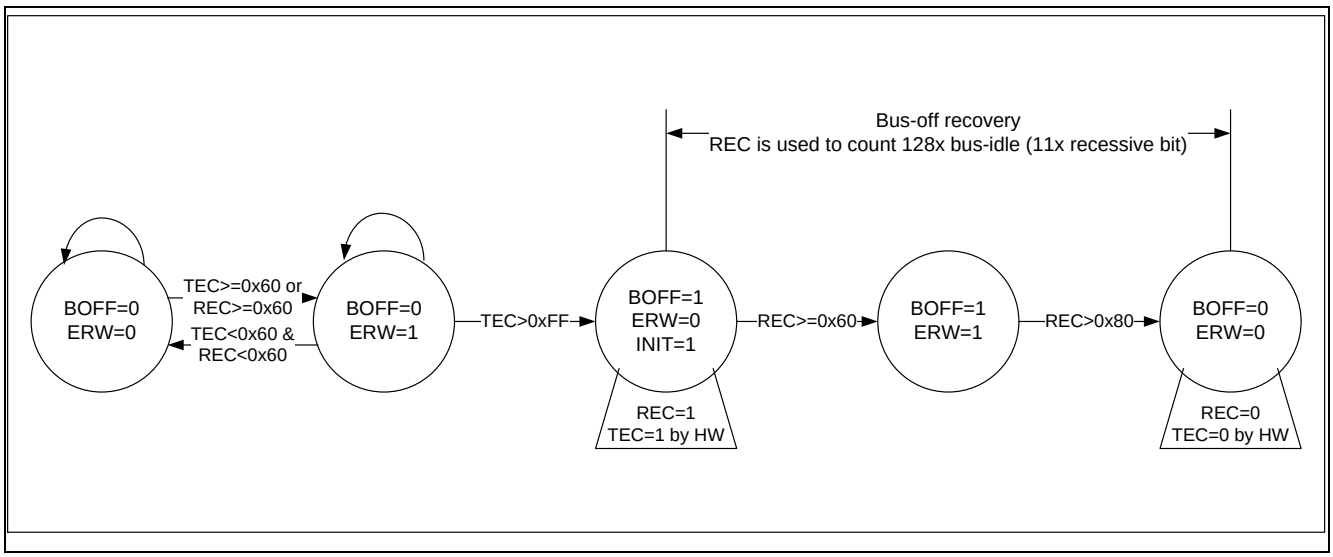

Figure 4 Status flags changing of MultiCAN register NSR

Bus-off state and INIT state

According to the CAN specification, in MultiCAN the bus-off state is activated if the Transmit Error Counter equals or exceeds the busoff limit of 256. This state is reported by flag BOFF.

“Bus-off recovery sequence” is started automatically when the CAN mode is in the bus-off state. **Figure 4** shows CAN node states changing in MultiCAN.

- During the bus-off-state, all respective control and message object registers hold their current values and the error counters are reset.
- 128 bus-idle events (11 consecutive ‘recessive’ bits) have to be detected, before the synchronization sequence can be initiated. The monitoring of the bus idle events is immediately started by hardware after entering the bus-off state. The number of already detected bus-idle events is counted and indicated by the receive error counter. After completion of the bus-off recovery sequence, the MultiCAN clears the bit BOFF, while INIT and ALERT will remain set.
- After 128 bus-idle events bit INIT will be tested by hardware.
  - If INIT is still set, the affected CAN node controller waits until INIT is cleared and at least one bus-idle event is detected on the CAN bus, before the node takes part in CAN traffic again; for example MultiCAN enters 'power-on' state.
  - If INIT has been already cleared (software has reset INIT during or after the recovery phase), the message transfer between the affected CAN node controller and its associated CAN bus is immediately enabled; for example MultiCAN enters 'idle' state.

2.1.2 The Linked List Controller

The message objects are organized in double-chained lists. The list organized message objects benefit from a flexible allocation of message objects for their CAN node.

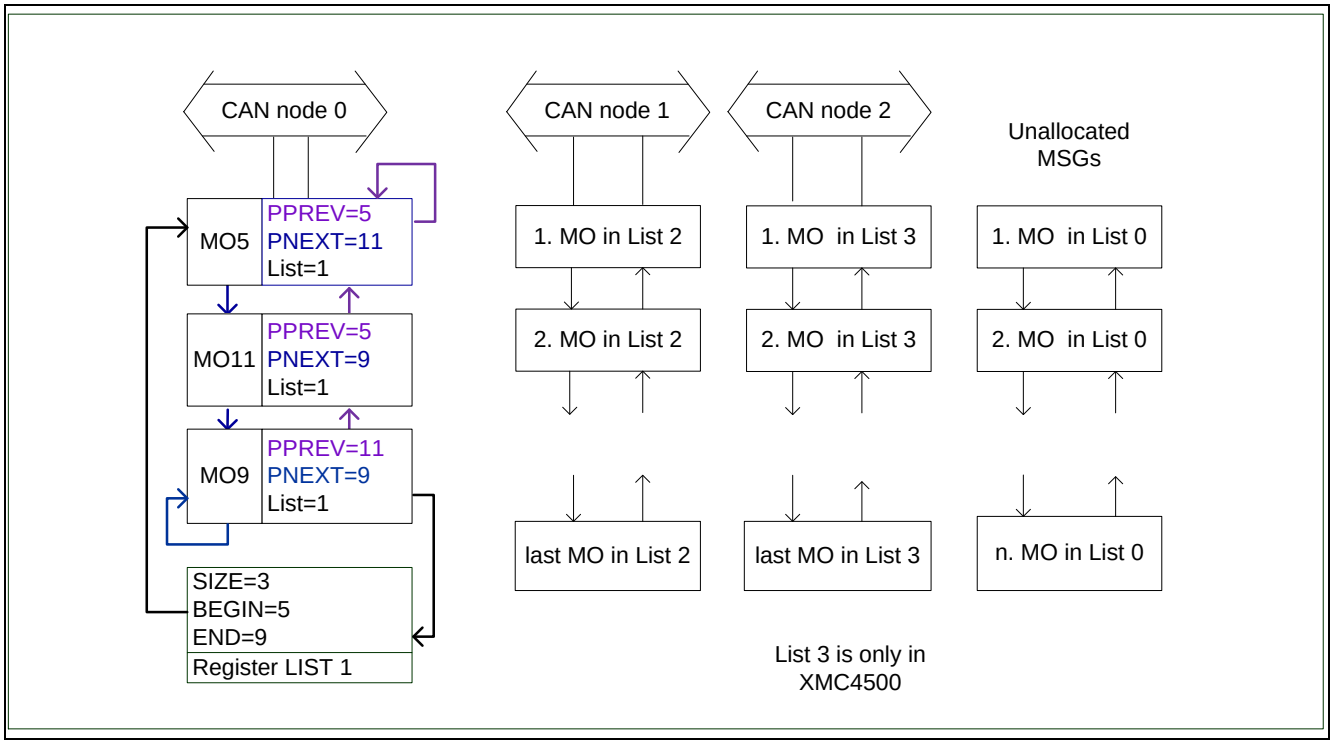

Figure 5 The Linked List

- FIFO and gateway message objects are based on a list structure.
- After reset, all message objects are on the list (list 0) of unallocated elements.
- After each write operation the bit PANCTR.BUSY must be checked in order to ensure correct list pointers in each message object.

2.1.3 The Interrupt Control Unit

The ICU controls the interrupt generation for different conditions. There are 140 hardware interrupt events.

- CAN node interrupts
  - Each CAN node has 4 interrupt sources (alert, last error, Tx/Rx ok, CFC overflow).
- Message object interrupts
  - Each message object has 2 interrupt sources (TxOk and RxOk).

MultiCAN contains 8 interrupt output lines (INT_O0…7), which are assigned to the NVIC (CAN.SR0…SR7).

Table 3 MultiCAN interrupts

Flag	Enabled by	SRx selected by	Flag to be cleared by software	Indication
CAN node				
NSRx.TXOK	NCRx.TRIE	NIPRx.TRINP	NSRx.TXOK=0	Frame has been transmitted on the CAN node.
NSRx.RXOK	NCRx.TRIE	NIPRx.TRINP	NSRx.RXOK=0	Frame has been successflly understood by the CAN node.
NSRx.ALERT	NCRx.ALIE	NIPRx.ALINP	NSRx.ALERT=0	Alert warning.
NSRx.BOFF	-	-	only by HW	
NSRx.EWRN	-	-	only by HW	Rx/Tx level limit error.
NSRx.LLE	-	-	NSRx. LLE=0	List length error.
NSRx.LOE	-	-	NSRx.LOE=0	List object error.
SRx.LEC	NCRx.LECIE	NIPRx.LECINP	Only by HW. Any write to LEC will result in a 0x7	Last error code change has been updated.
NFCRx.CFCOV	NFCRx.CFCIE	NIPRx.CFCINP	NFCRx.CFCOV=0	Frame counter overflow interrupt. CAN frame counter mode: - Frame count mode. - Time stamp mode. - Bit timing mode.
Message Object				
MOSTATn.RXPND	MOFCRn.RXIE	MOIPRn.RXINP	MOCTRn=0x1	Frame has been received in the MO.
MOSTATn.TXPND	MOFCRn.TXIE	MOIPRn.TXINP	MOCTRn=0x2	Frame has been transmitted from the MO.

2.1.4 Node Receive input selection

The MultiCAN module contains a switch in order to select different node's receive input lines via the Node Port Control Register NPCRx (x=0, 1, 2). The selected input signal for each CAN node is made available by its internal signal CANxINS, which is connected to other peripherals.

- Select the input line via bit field NPCRx.RXSEL.
- LBM=1: loop-back mode.
- Internal signal CANxINS is connected to other peripherals (USIC), but it can not trigger ERU directly.

Note: In the XMC4000 family the intenal signal CANxINS can not trigger ERU directly. Instead, the defined CAN input pin triggers ERU directly.

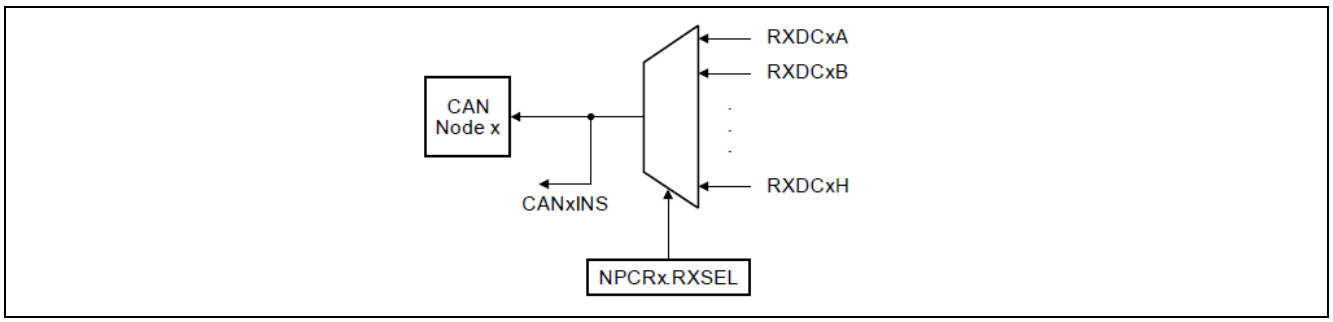

Figure 6 Node input control registers

2.1.5 Message controller

The Message controller handles CAN frames between the node and the message RAM. It performs the following functions:

- Receive acceptance filtering for storing the CAN frame received on the node into the message object.
- Transmit acceptance filtering for detection and prioritization of the message object to be transmitted.
- FIFO and gateway functionality.

2.1.6 Interfaces and interconnects

The MultiCAN module interfaces with the clock, port, and interrupt/DMA connections are described here.

Clock input

Clock f_{PERIPH} from the XMC4000 Clock Control Unit (CCU) is used as the MultiCAN module clock input. In the XMC4000 family all peripherals can be individually controlled via the registers PRSETx/PRCLRx.

After power-on-reset, MultiCAN module clocked with f_{PERIPH} will remain in a reset state. To release MultiCAN, PRCLR1.MCAN0RS must be set to 1.

f_{CLC} is used for internal logic and register operation.

f_{CAN} is used baud rate generation.

In order to get a maximum of accesses to the MultiCAN module the $f_{CLC} = f_{PERIPH}$ should use the maximum f_{PLL}.

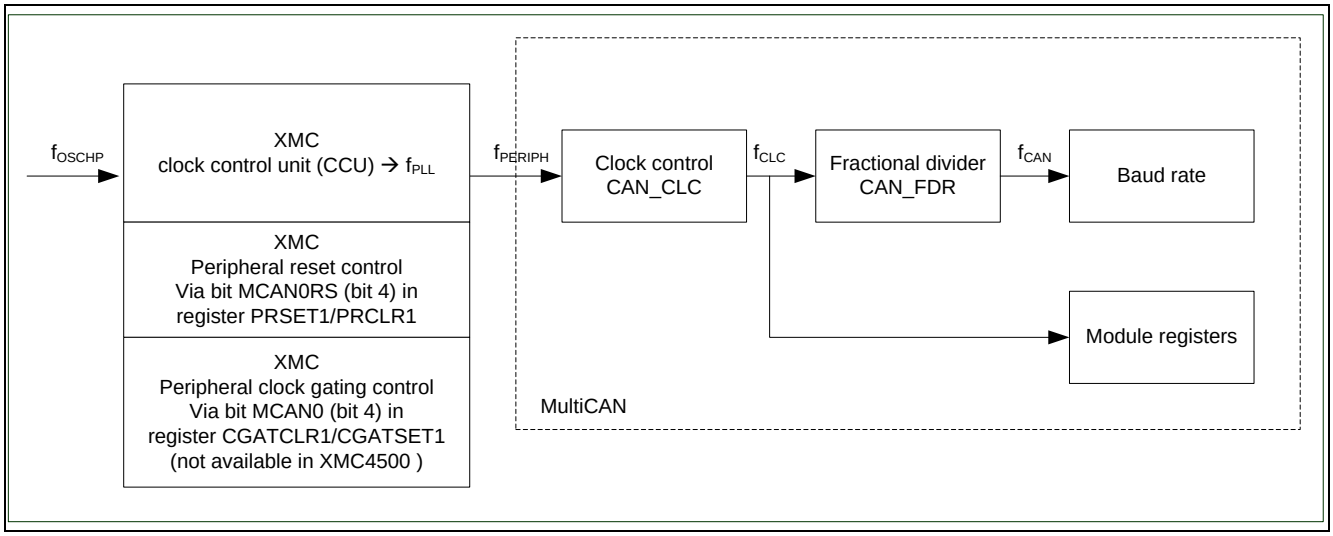

Figure 7 MultiCAN clock generation

Interrupt trigger (to DMA, to other peripherals, to Interrupt Control Unit)

- CAN.SR0 through to SR7 (IRQ number 76...83).
  - CAN.SR0 through to SR3 can be used for DMA service.
- The interrupt priority level and enable/disable are controlled by the NVIC unit in the XMC family.

Port pin and I/O lines control

The interconnections between the MultiCAN and the port I/O port lines are controlled in the port logic.

For each CAN node its input receive pin, selected via NPCRx.RXSEL should be initialized as “direct input” with Pn_IOCRy.PCx=00000_B to receive the CAN frame, while the transmit output pin must be configured as alternate output pin by Pn_IOCRy.PCx=$100xx_B$ with push-pull driven mode.

In XMC4000 up to 4 alternate output functions (ALT1/2/3/4) can be mapped to a single port pin. Usually the MultiCAN transmit output pin uses ALT1/2. (Please refer to the appropriate Reference Manual or Data Sheet).

2.1.7 Bit timing

In the ISO 11898-1 classical CAN part, one CAN bit time is sub-divided into 4 segments and contains 8-25 Time Quanta t_q.

- Synchronization Segment ($T_{Sync}= 1 \times t_q$)
  - Used to synchronize the various CAN nodes on the bus and an edge is expected within this segment.
- Propagation Time Segment ($T_{Prop}= 1 \ldots 8 \times t_q$)
  - Used to compensate for signal delays of the actual network.
- Phase buffer segment ($T_{b1} = 1 \ldots 8 \times t_q$)
  - Used to compensate for a mismatch between transmitter and receiver clock phases detected in T_{Sync}.
  - It may be lengthened by re-synchronization.
- Phase buffer segment ($T_{b2} = 1 \ldots T_{b1}$):
  - Same as T_{b1}.
  - It may be shortened by re-synchronization.

The amount of lengthening and shortening of T_{b1} and T_{b2} is determined by the maximum value given by the SJW.

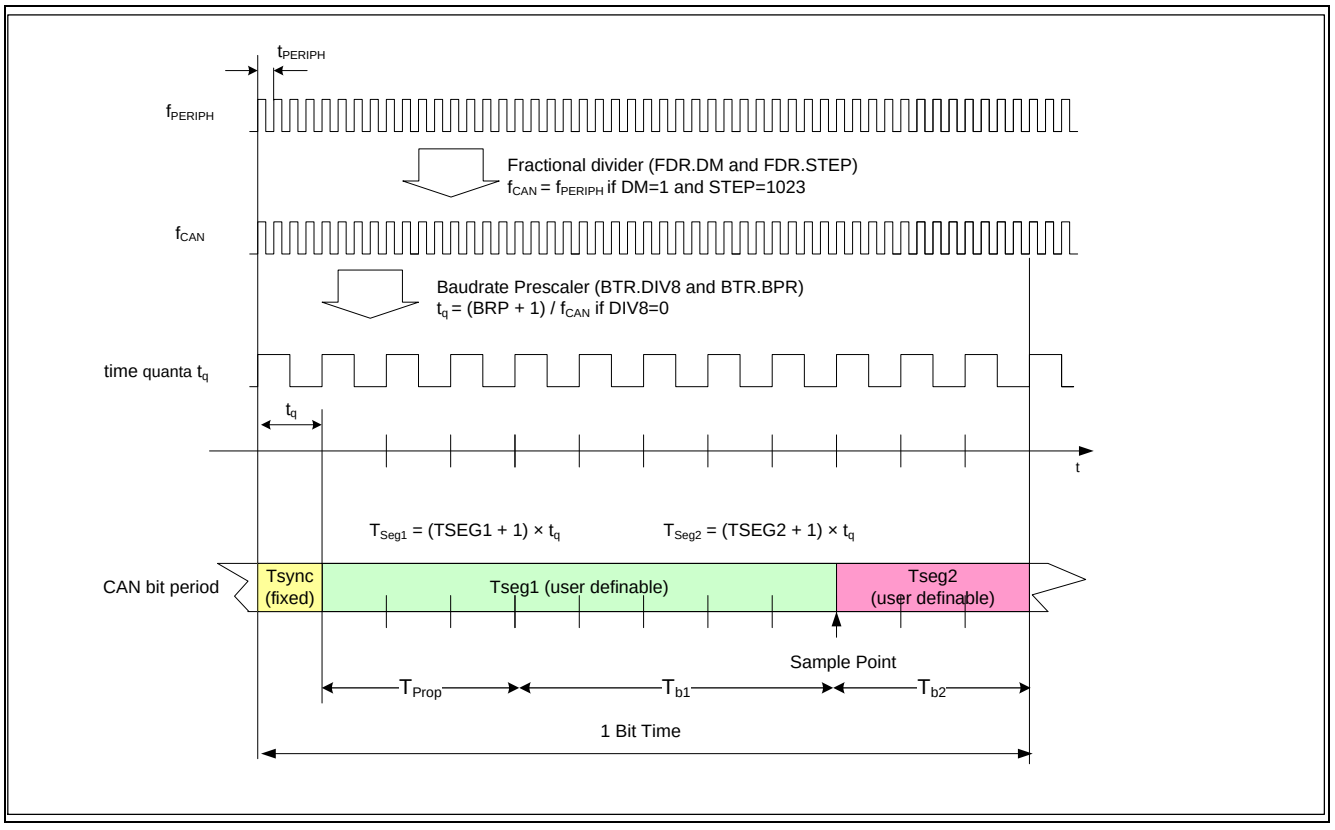

Figure 8 MultiCAN bit timing

The bit rate, the sample point, and SJW are user programmable in MultiCAN.

- CAN bit time
  - This is subd-ivided into the three, non-overlapping segments T_{Sync}, T_{Seg1} and T_{Seg2}
  - T_{Prop} and T_{b1} are sufficient to T_{Seg1}.
  - $t_q = (BRP + 1) / f_{CAN}$ (if DIV8=0) or 8 x (BRP + 1) /f_{CAN} (if DIV8=1)
  - $T_{Sync} = 1 \times t_q$
  - $T_{Seg1} = (TSEG1 + 1) \times t_q$
  - $T_{Seg2} = (TSEG2 + 1) \times t_q$

    Bit time = $T_{Sync} + T_{Seg1} + T_{Seg2}$

    Sample point = $(T_{Sync} + T_{Seg1})$ / Bit time
- The sample point is an important parameter.
  - Choosing a later sample point in the bit period results in more tolerance with respect to propagation delay and therefore greater bus length. Conversely, choosing a sample point closer to the mid-point of the bit period will allow a greater oscillator tolerance for each node in the system.
  - Obviously a large allowable oscillator tolerance and a long bus length are conflicting goals, which can only be accomplished through optimization of the bit timing parameters. A good general rule is to set the sample point to about 80% of the bit timing.
- The control register BTR is used to set up the bit timing parameters.
  - Number of time quanta for SJW (N_{Tsjw}): $1 \le N_{Tsjw} \le 4$ and $N_{Tsjw} \le N_{Tseg2}$

On the internet you can find possible MultiCAN register values for CAN bit rates (see **CAN Bit Time Calculation**). The next table gives some typical bit timing parameters.

Table 4 MultiCAN bit time settings (SJW=1, SP=80%)

Baud Rate	fsys	BRP	N_{tq} (8..25)	N_{Tseg1}= 1+ (1+TSEG1) (3..16)	N_{Tseg2}= (1+TSEG2) (2..8)	NBTR
1000	120	6	20	1+15	4	0x3E06
		8	15	1+11	3	0x2A07
		10	10	1+7	2	0x160B
500	120	12	20	1+15	4	0x3E0B
		16	15	1+11	3	0x2A0F
		24	10	1+7	2	0x1617

2.2 Hardwaer handling of CAN frame reception and transmission

The following figure illustrates the MultiCAN hardware handling of a CAN frame reception and a CAN frame transmission.

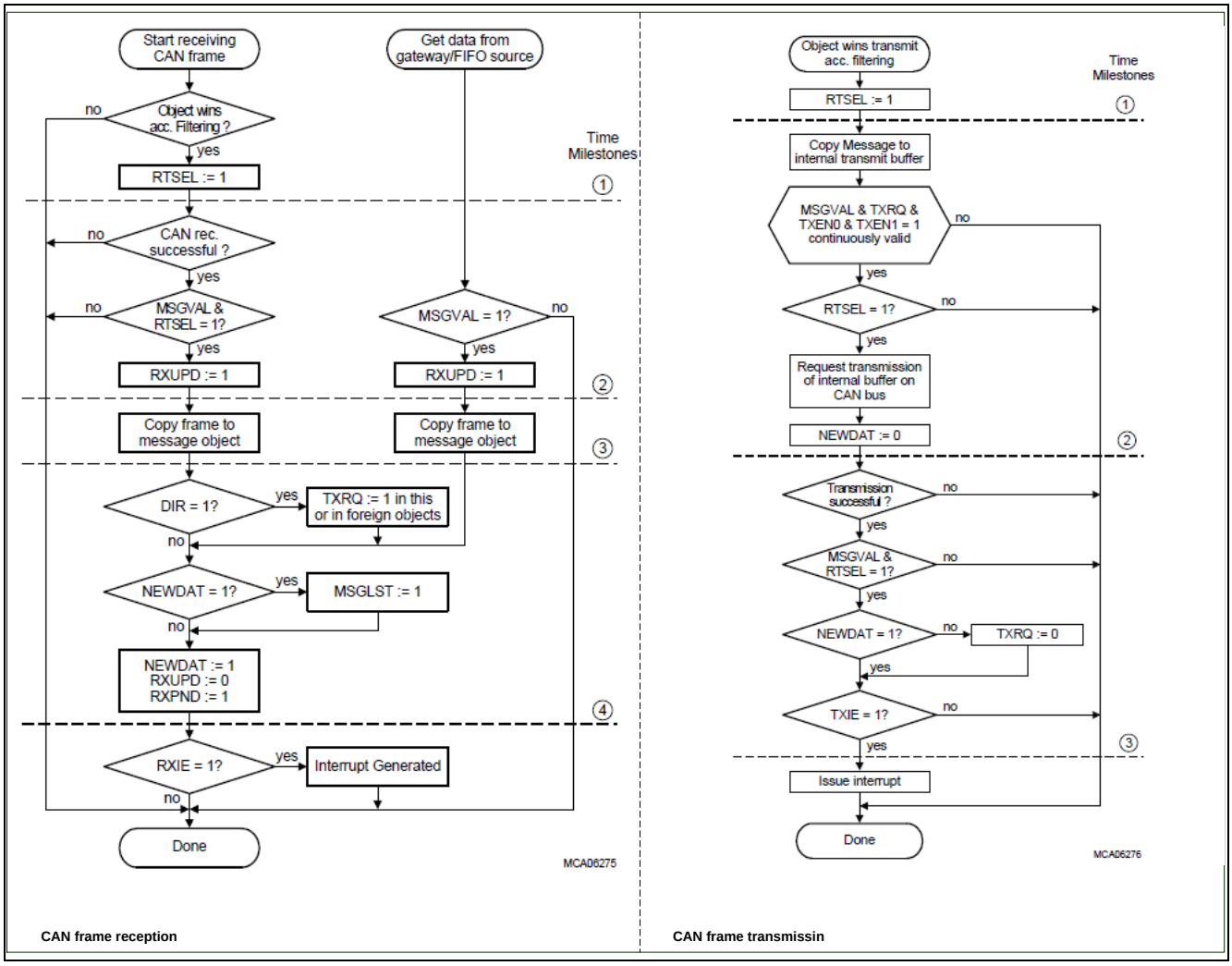

Figure 9 CAN frame reception and transmission by the MultiCAN node

Transmission process

The transmission process of a message object (DIR=’1’ or DIR=’0’) starts after winning the transmit acceptance filtering.

A message object with the following settings wins an ‘effective transmit request’:

- MSGVAL=’1’
- TXEN0=’1’
- TXEN1=’1’
- TXRQ=’1’
- PRI != ‘0’

A trigger transmission request in a transmit message object (DIR=’1’) generates a data frame.

A trigger transmission request in a receive message object (DIR=’0’) generates a remote frame.

Reception process

The reception process of a message object (DIR='0' or DIR='1') starts after winning the receive acceptance filtering.

A message object is qualified for reception of a frame if the following conditions are fulfilled:

- MSGVAL ='1'
- RXEN ='1'
- DIR='1' accepts only remote frame; DIR='0 accepts only data frame
- MIDE='0' accepts both 11_IDs and 29_IDs; MIDE ='1' accepts only its specified IDs via MOARn.IDE
- The ID of the message object matches the received ID through the acceptance mask (MOAMRn.AM)
- PRI != '0'

Incoming remote frames are stored in a corresponding transmit message object (DIR='1').

Arriving data frames are saved in a matching receive message object (DIR='0').

Resolution of multiple message objects

If several message objects assigned on the CAN node meet the conditions above, the message object with the highest PRI wins the transmit/receive acceptance filtering.

2.3 Programming the MultiCAN module

This section explains how to program the MultiCAN module for some typical use cases.

The following software tasks are processed in the CAN application:

- Configuration of CAN node.
- Allocate message objects via list commands.
- Initialization of associated message objects.
- Controlling a message transfer.
- CAN error monitoring and restarting the CAN module.

2.3.1 Software initialization of MultiCAN

The initialization routine should process the following tasks:

- Enable clock for MultiCAN module (see **Figure 8**)
  - Release the MultiCAN clock gating via clear register bit CGATCLR1. MCAN0 (not in XMC4500).
  - Release the MultiCAN peripheral via clear register bit PRCLR1.MCAN0RS.
  - Switch clock f_{CLC} on via clear register bit CAN_CLC.DISR and wait until flag CAN_CLC.DISS =0, which indicates the MultiCAN has been enabled.
  - Configure the module clock f_{CAN} via register FDR for the bit timing configuration.

Note:

1. *Two modes, normal divider mode and fractional divider mode, can be used via CAN_FDRx.DM. In general the fractional divider mode provides the average output clock frequency with a simulated higher accuracy than the normal divider mode, but f_{FD} can have a maximum period jitter of one f_{PERIPH} period. It is NOTadvised to use the fractional divider mode. If the fractional divider is used, one f_{PERIPH} cycle has to be added to the jitter calculation.*

2. *After the clock has been switched on, the CAN RAM is automatically initialized. The end of this CAN RAM initialization is indicated by bit PANCTR.BUSY becoming in-active. Due to synchronisation effects, it is advised to read back the previous register write (FDR), so that the BUSY bit is polled, when it is already set the first time.*

- Configuration of CAN nodes.
  - Set bit CAN_NCRx.CCE and INIT to active configuration mode of the CAN node without participation in the CAN traffic. Configuration Mode is activated when bit NCRx.CCE is set to 1 (see **page 8**).
  - Write CAN_NBTR for CAN baud rate configuration.
  - Select CAN input pin or loopback mode via CAN_NPCRx.
  - Interrupts and special node frame nodes can be optionally initialized via CAN_NIPRx and CAN_NFCRx.
- Initialization of message objects.
  - Allocate message objects to the corresponding CAN node via register CAN_PANCTR.
  - Define and configure message objects for different tasks via register MOCTRn (DIR='1' or '0').
  - Program message objects identifier (MOARn) and acceptance mask for filtering (MOAMRn).
  - Initialize data length code (MOFCRn.DLC) and data value TX message object.
  - Enable interrupt for message object transmission and reception via register MOFCRn.
- Initialize interfaces of MultiCAN, such as port pins for alternate function and interrupts, using CMSIS functions.
- Set CCE to 1 to protect against un-intended modification. Reset bit INIT to enable the participation of the CAN node in the CAN traffic.

2.3.2 Software handling on CAN node

The software can use the bit TXOK/RXOK and all errors flags for evaluation of the node status.

In MultiCAN each node is equipped with a frame counter with the following selectable modes:

- Frame count mode
  - The default setting is that frame counter NFCRx.CFC is incremented upon reception/transmission of defined frames (depending of NFCRx.CFSEL).
- Bit timing mode
  - CFC is used for analysis of the bit timing. Together with the **CAN analyzer** this feature is used for baudrate detection. Example code is provided (see **Example_6: Baudrate**).

Table 5 Frame counter modes

Mode (NFCRx.CFMOD)	Flag NFCRx.CFCOV generated by	Function selection NFCR.CFSEL	Interrupt SRx	Frame count value NFCRx.CFC
00_B: frame count mode	Transition from 0xFFFF to 0x0000	Selectable: $xx1_B/x1x_B/1xx_B$	Selectable: SR0…SR7	Frame count value
01_B: Time stamp mode	Transition from 0xFFFF to 0x0000	Only 000_B	Selectable: SR0…SR7	Captured bit time count value
10_B: Bit time mode	update event of CFC	Selectable xxx_B, e.g. 000_B: baudrate detection	Fixed on SRx, where x is the CAN node number	f_{CLC} clock cycles

- Time-stamp mode
  - CFC is used to count bit times. The frame counter is continuously incremented (internally) with the beginning of a new bit time. Its value is permanently sampled in the NFCRx.CFC field while the bus is idle. The value sampled just before the SOF bit of a new frame is detected is written to the corresponding message object. When the treatment of a message object is finished, the sampling continues.

2.3.3 Software control of a message transfer

Table 2 lists the maximum number of m essage objects in XMC4000 derivatives and the message objects which can be set up for transmit or receive operation according to the selected value for control bit DIR.

- TX message object (DIR=1); Set for data frames' transmission and for remote frames' reception.
- RX message object (DIR=0); Set for data frames' reception and for remote frames' transmission.

Note: To enable CAN (data or remote) frame reception, bit RXEN must be set. For example, if RXEN='0' in a TX message object (DIR='1') then a remote frame from CAN bus can not be restored in this object.

Software handling of a transmit message objct

Figure 10 demonstrates the software handling of a transmit (TX) message object (DIR='1').

If automatic handling is requested, bit TXEN0 must be initialized with '1'.

Together with TXEN1=1, the data transmission is started when flag TXRQ has been set by the hardware because a received remote frame has a matching identifier.

Software handling of a receive message object

Figure 11 demonstrates the software handling of a receive (RX) message object (DIR='0').

RXEN must be set in a RX message object to enable receive data frame.

The reception of a data frame by hardware is indicated by NEWDAT='1' and RXPND='1'.

Software processing of a received data frame should start by clearing NEWDAT and RXPND, after scanning MSGLST. In an overwrite situation, the received information should be copied to an application data buffer in order to release the message object for a new data frame.

Finally, NEWDAT and RXUPD should be checked again to ensure that the processing was based on a consistent set of data and not on a part of the new message.

The software initialization or re-configuration of the message object properties always starts with disabling via MSGVAL='reset'. After re-configuration to activate the message object, bit MSGVAL must be reset using RTSEL.

Bits TXEN0 and TXEN1 are software control bits used for different tasks:

- Bit TEXN0 is defined for software control of the CAN frame trigger. For example:
  - If a remote frame has been received in a TX message object, the send request TXRQ is set by hardware automatically. When TXEN0=0 the transmission of a data frame from this TX message object is suspended until it is re-enabled by software by setting TXEN0.
  - In gateway structure, TXRQ is set in the destination object by hardware automatically when bit GDFS = '1' in the source object is initialized.
- Bit TXEN1 is defined to select the active TX message object in TxFIFO structure. TXEN1 with value '1' moves along the TxFIFO structure as a token by hardware automatically.

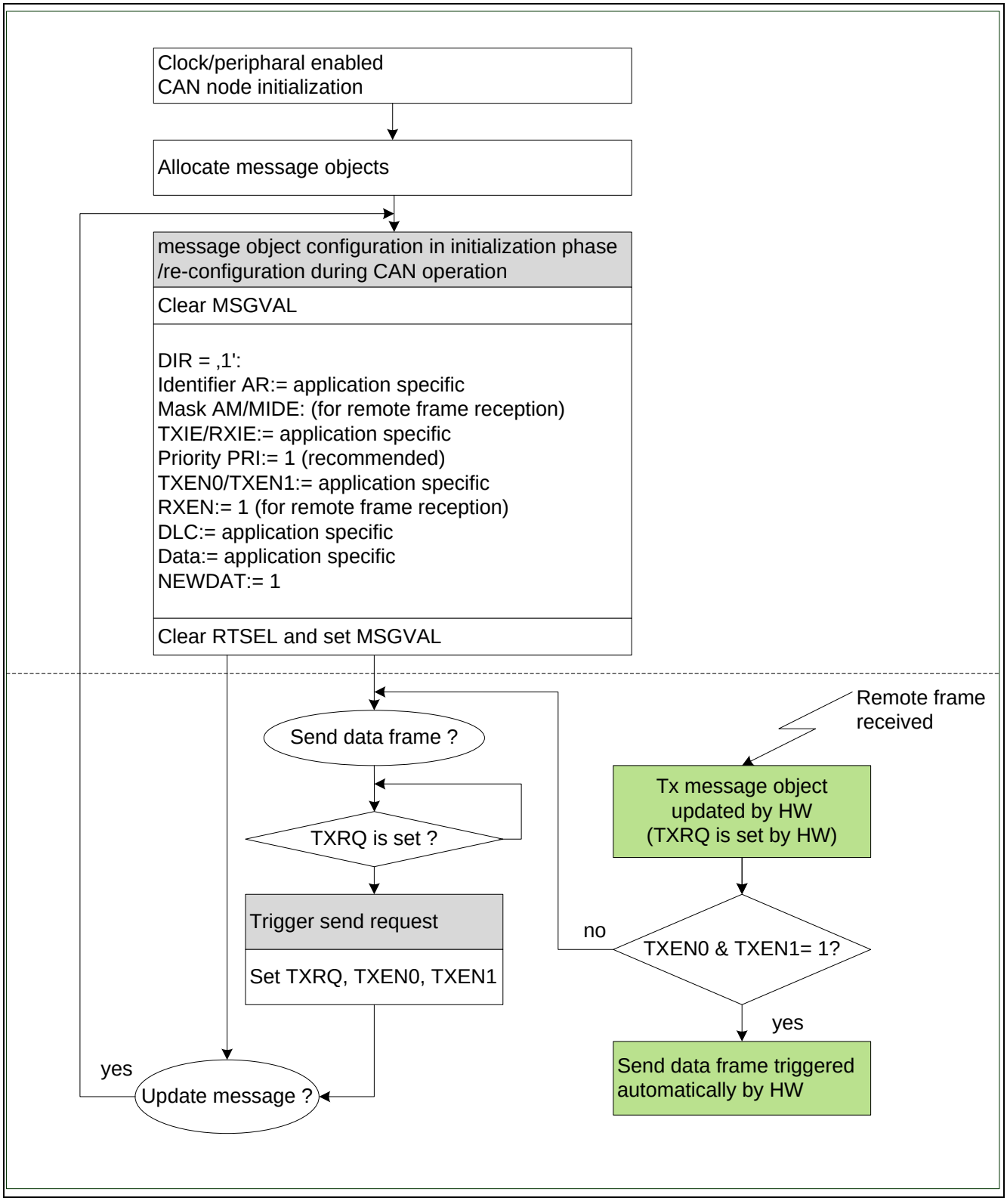

Figure 10 Software handling of a TX message object (DIR='1')

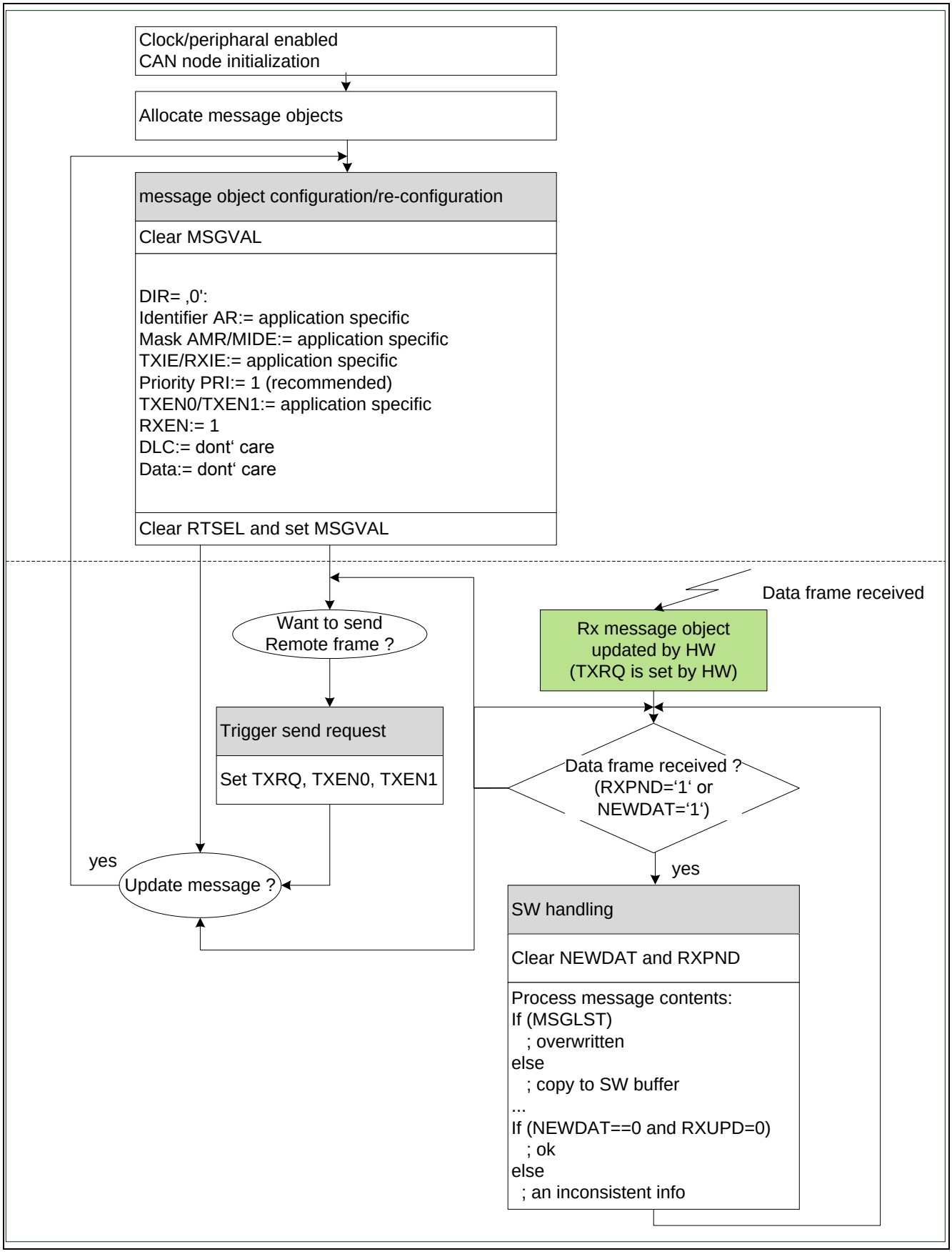

Figure 11 Software handling of a RX message object (DIR=’0’)

2.4 MultiCAN special features

2.4.1 Loop-back mode

Loop-back mode is used for internal testing.

In this mode the software driver can be developed and tested without being connected to a CAN bus system, or safety tests can be run without being visible to the outside.

2.4.2 CAN analyzer mode

This mode is used for baudrate detection (see **Example_6: Baudrate**). The hardware setting and software initialization are the same as in a normal CAN system.

Bit CAN_NCRx.CALM must be set to active the analyzer mode.

In analyzer mode, data and remote frames are monitored without an active participation in any CAN transfer (the CAN transmit pin is held on recessive level).

In this mode the data and remote frame can still be received and stored in the corresponding message object, and interrupts are also generated, when this CAN frame is acknowledged by at least one other CAN node.

2.4.3 MultiCAN FIFO

MultiCAN FIFO is based on the list structure; i.e. FIFO size is up to the maximum available message objects (64 message objects in XMC4500 device).

As with the standard TX/RX message object, all FIFO elements must be assigned to the CAN node via panel commands first. After assignment all FIFO objects are chained together in a list structure; each element has its previous (PPREV) and next (PNEXT) message object.

A FIFO structure can have only one base object and several slave objects. The base object defines the FIFO size with the point TOP and BOT, and additionally the CUR points to the active object for the next process.

Software programming for TxFIFO structure

In the TxFIFO structure the base and slave objects can be initialized with different IDs (including mask register) and data contents. Each element is active for qualification in transmit acceptance filtering.

The right-hand side of **Figure 12** shows the FIFO structure in one common list and the initialization of TxFIFO message objects:

- Allocate FIFO elements in a common list.
- Configure TxFIFO base object and TxFIFO slave objects.
- Set in all message objects excepted the CUR pointed object:
  - TXEN1=0
- Set in all message objects:
  - TXRQ=0 if an automatically transmit process through TxFIFO via only one trigger request in the base object is requested.

After the initialization routine, test software sets TXRQ in CUR pointed object.

The data frame information stored in this object will be sent on the CAN bus via token TXEN1, through all TxFIFO elements.

In this example CUR moves MO8 →MO9 → MO10 → MO11 → MO8…

Additionally, by initialization with the point SEL= MO8 in the base object, the FIFO overflow interrupt is generated if CUR becomes equals to SEL; i.e. MO8…MO11 have finished their data frame transmission.

Note: For the TxFIFO structure the FIFO overflow interrupt is shared with the receive interrupt. In the example above it triggers the MO8 receive interrupt.

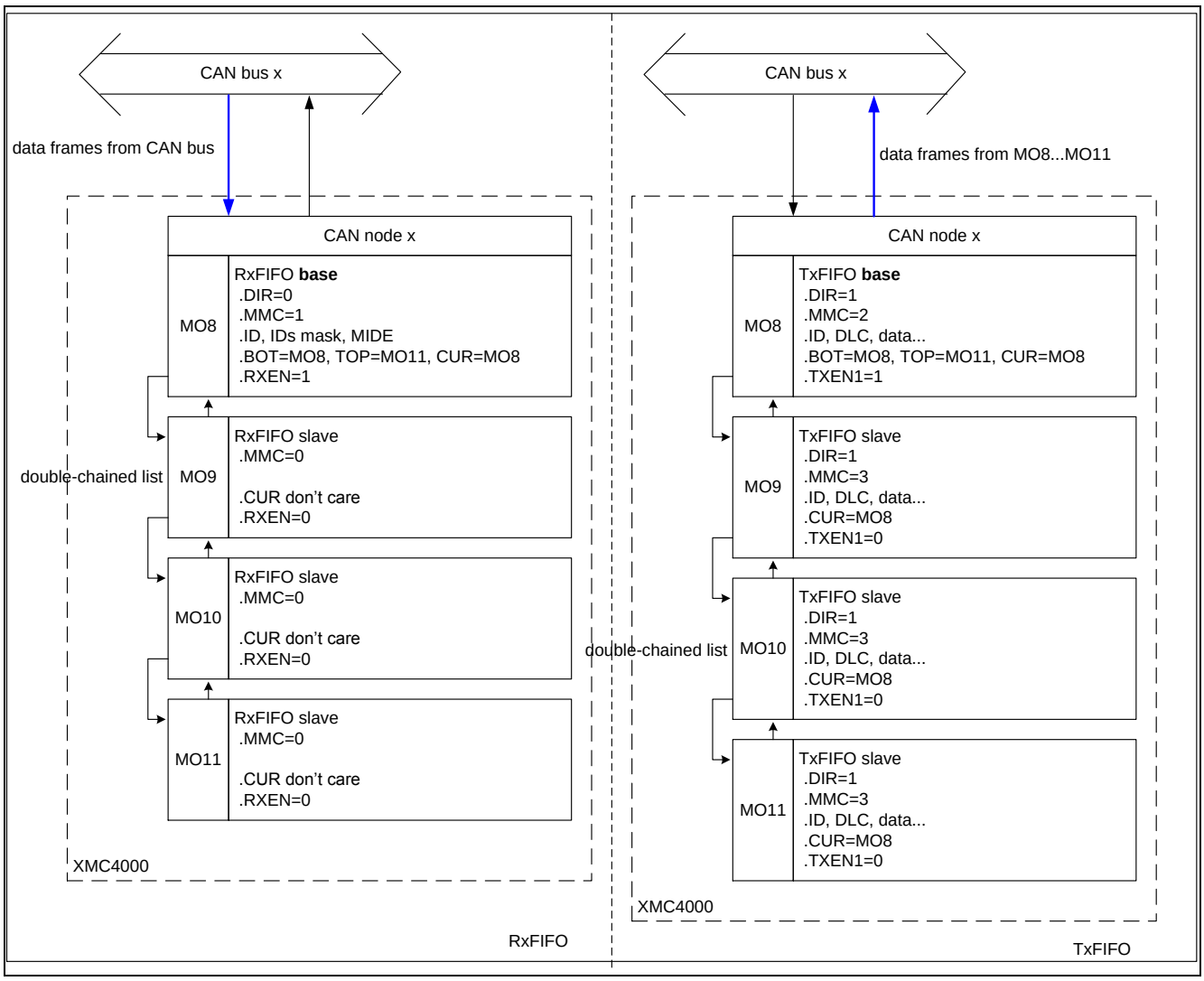

Figure 12 MultiCAN FIFO in one common list

Software programming for RxFIFO structure

In the RxFIFO structure:

- the base object is specified with MMC=0001_B
- the message object with MMC = 0000_B is implicitly assumed for the slave object; i.e. slave objects perform a standard RX message object delivery. This property cretes an RxFIFO structure to store CAN frames with different IDs.

Note: In order to avoid direct reception of a message by a slave message object, as if it was an independent message object and not a part of a FIFO, the bit RXEN of each slave object must be cleared. In this case only the base object is active to qualify in receive acceptance filtering.

The left-hand side of **Figure 12** shows the FIFO structure in one common list and initialization of RxFIFO message objects:

- Allocate FIFO elements in a common list.
- Configure the RxFIFO base object and the RxFIFO slave object.

Note: TXEN0 and TXEN1 have to be considered only when RxFIFO is used for transmitting remote frames.

In this example all received data frames with a matching identifier specified in MO8 (via MOARn and MOAMR) are stored in the RxFIFO buffer.

The CUR moves MO8 →MO9 → MO10 → MO11 → MO8…

Additionally, by initialization with the point SEL= MO8 in the base object, the RxFIFO interrupt is generated if CUR becomes equal to SEL; i.e. MO8…MO11 has been filled.

Note: For the RxFIFO structure the FIFO overflow interrupt is shared with the transmit interrupt. In the example above, it triggers the MO8 transmit interrupt.

2.4.4 MultiCAN Gateway

The Gateway feature allows an automatic CAN frame re-routing between two independent CAN busses without CPU interaction.

The Gateway in MultiCAN operates on the message object level; i.e. CAN frame information can be modified including its identifier, baud rate, and data information for example, during transfer between two CAN bus systems.

As with the FIFO, the Gateway structure in MultiCAN is realized by the list structure. A message object in the Gateway structure is named ‘Gateway Source Object’ and ‘Gateway Destination Object’.

The ‘Gateway Source Object’ behaves as a standard message object with the following additional, selectable actions:

Table 6 MultiCAN gateway feature (configured by the Gateway Source Object)

Source object	Destination object
$MOFCR_{source}$.IDC = ‘1’	$MOAR_{destination}$ updated with $MOAR_{source}$
$MOFCR_{source}$.DLCC = ‘1’	$MOFCR_{destination}$.DLC updated with $MOFCR_{source}$.DLC
$MOFCR_{source}$.DATC = ‘1’	$MODATAH/L_{destination}$ copied from $MODATAH/L_{source}$
$MOFCR_{source}$.GDFS = ‘1’	TXRQ is set in the ‘destination object’
	Note: the actual destination object is activated by CUR point of the Source Object.
Use case 1: Gateway Source Object: DIR=‘0’ and Gateway DestinationObject: DIR=‘1’ **Data** frame reception on Source Object side and transfer through Gateway to Destination side **Figure 13** shows Gateway use case for **data** frame reception.	
Use case 2: Gateway Source Object: DIR=‘1’ and Gateway Destination Object: DIR=‘0’ **Remote** frame reception on Source Object side and transfer through Gateway to Destination side	

Note: In the Gateway structure the 'Gateway Source Object' and 'Gateway Destination Object' must be assigned to two different CAN nodes.

MMC=0100_B specifies the 'Gateway Source Object'.

As with the base object in FIFO structure, the 'Gateway Source Object' configures the Gateway Destination structure via TOP, BOT, and CUR. The Gateway Destination structure may contain a single message object or a FIFO structure.

Initialization of Gateway message objects

- Allocate the Gateway Source Object and the Gateway Destination Objects in separate lists.
- Configure Gateway Source Object (DIR='0') and parameters (IDC, DLCC, DATC, GDFS).
- Configure Gateway Destionation Objects (DIR='1').
  - Single Gateway Destination Object with MOFCRn.MMC = '0000_B'.
  - Gateway Destination Objects in TxFIFO (TxFIFO base object + TxFIFO slave objects).

Note:

1. *Clear RXEN bit in the Gateway Destionation Objects in order to avoid direct remote frame receive.*
2. *Clear NEWDAT and TXRQ in the Gateway Destination Objects in order to avoid conflict with trigger signal from the Gateway Source Object.*

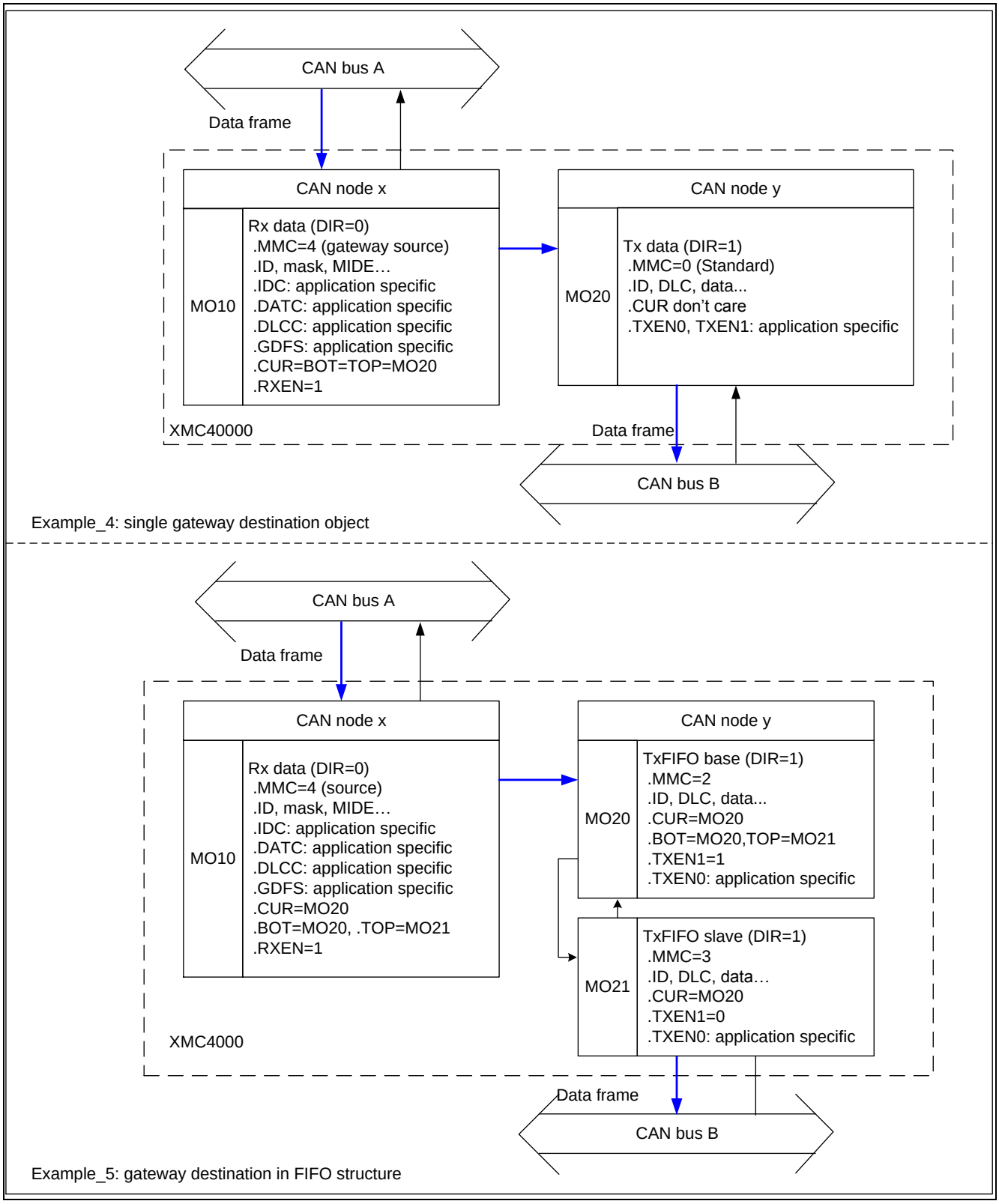

Figure 13 MultiCAN Gateway

3 Implementing the example

All example code supplied with this document can be integrated directly in compiler tools and run on all XMC4000 Kits listed in **Table 8**.

Infineon provides freeware tool **DAVE™**, which integrates code generation, flash programming and debugging.

3.1 First steps

- Create a new “empty project” in the DAVE™ tool
- Copy the example *.h and *.c files into the project directory.
- Select macro definition in **main.h** for one of XMC4200, XMC4400, XMC4500 kit.

3.2 Example_1: Standard Message Object Transmission and Receive

- Initialization:
  - CAN_node0: MO8: TX message object (RXEN=1: receive remote frame).
  - CAN_node1: MO16: RX message object.
- Test_1: set trigger TXRQ (with TXEN0=1 and TXEN1=1) in MO8 to send data frame.
- Test_2: set trigger TXRQ (with TXEN0=1 and TXEN1=1) in MO16 to send remote frame.

3.3 Example_2: Using Receive FIFO

- Initialization:
  - CAN_node0: MO16…MO19 RxFIFO structure within MO16 = RxFIFO base object for store data frame.
  - CAN_node1: MO8: TX message object.
- Test: load transmission data in MO8 and trigger send request afterwards for several times, and check the received data in RxFIFO message objects.

3.4 Example_3: Using Transmit FIFO

- Initialization:
  - CAN_node0: MO8…MO11 TxFIFO within MO8 = TxFIFO base object. Each object has different 11bit_IDs, DLC and data information, and prepared as before.
  - CAN_node1: MO16…MO19 RxFIFO structure within MO16=RxFIFO base object.
- Test: in this test TXRQ is set in MO9…MO11 during initialization routine so that all four data frames from TxFIFO are transmitted by single trigger request in MO8 automatically.

3.5 Example_4: Using Gateway without FIFO

Data frame with 11bit_ID=0x444 are received on CAN_node2 and transmitted on CAN_node1 with modification of ID=0x777 via the Gateway feature.

- Initialization:
  - CAN_node0: MO10: Gateway Source Object with ID=0x444 and DATC=1, DLCC=1, IDC=0, GDFS=1(copy data bytes including DLC and set TXRQ).

  - CAN_node1: MO20: Gateway Destination Object with 11bit_ID=0x777 (TXEN0=1, TXEN1=1 for automatical trigger if data frame forwarded from source side).
  - CAN_node2: test object MO50 (DIR=1, ID=0x444) and MO51 (DIR=0, ID=0x777).
- Test: data frames with 11bit_ID=0x444 are created for MO10 on the bus (connected on CAN_node2). It will be forwarded to MO20 (Gateway Destination side) and transmitted on CAN_node1 automatically without any software control.

Note:

1. *CAN_node2 with MO50, MO51 is defined here for a simple test of the Gateway function in loop-back mode, because on the XMC4000 kit only the CAN_TX/_RX pin of CAN_node2 are available.*
2. *Because this example code uses loop-back mode, all RX message objects have dedicated IDs (with acceptance mask on) in order to avoid endless transmit due to unintended message object storage.*

3.6 Example_5: Using Gateway with FIFO

Data frames with 11bit_ID=0x444 are received on CAN_node2 and transmitted on CAN_node1 with modification of ID=0x777 via the Gateway feature.

- Initialization:
  - CAN_node0: MO10 Gateway Source Object with ID=0x444 and DATC=1, DLCC=1, IDC=0, GDFS=0 (copy data bytes including DLC, but do not set TXRQ).
  - CAN_node1: MO20/MO21 Gateway Destination Objects in FIFO structure with ID=0x777 (TXEN0=0 in MO2/MO21: Sending on Destination side is controlled by software).
  - CAN_node2: test objects MO50 (DIR=1, ID=0x444) and MO51 (DIR=0, ID=0x777).
- Test: data frames with 11bit_ID=0x444 are created for MO10 on the bus (connected on CAN_node2). Software checks information in MO20/MO21 (Gateway Destination side). At the end the data frame in MO20/MO21 is sent out on bus (connected on CAN_node1) and via set TXEN0 in MO20/MO21.

3.7 Example_6: Baudrate detection

In some applications it is necessary to detect the baudrate without any influence on the bus. The MultiCAN analyzer can be used to monitor bus transfer without any activity on the bus itself.

The main point for baudrate detection is to detect the minimal time (one bit ‘0’ and one bit ‘1’) of a whole CAN frame, therefore here it requires a specific ID or data field (byte 0x55) from the active CAN node.

The bit timing frame mode is implemented in MultiCAN. In setting NCRn.CFMOD=10_B with NFCRn.CFCSEL=000_B, the clock cycles of f_{CAN} counts up at the dominant edge monitored on the CAN receive input line, and is then stored in the NFCRn.CFC.

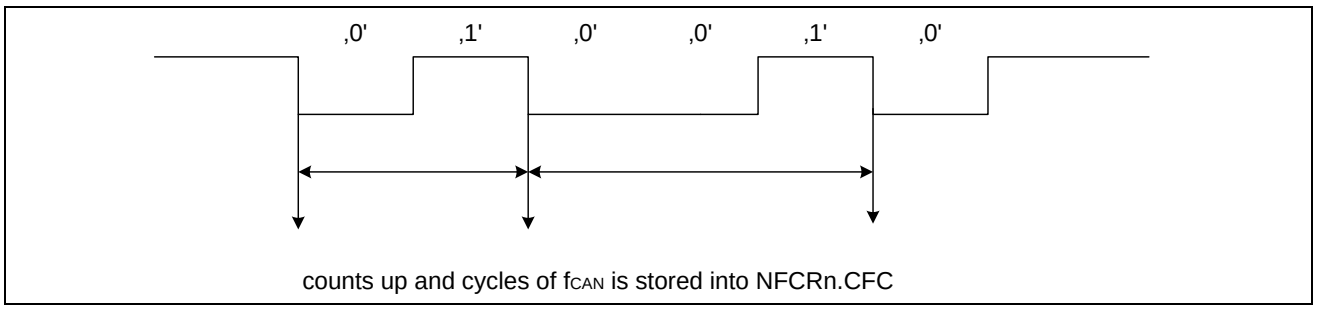

Figure 14 Baudrate detection

Implementing the example

In this example the minimum duration is calculated after 200 samples. The recorded minimum value is used for the baudrate calculation.

One CAN bit must be from 8 to maximum 25 time quanta. In test software typical values for the bit timing setting (register BTR.TSEG1, TSEG2 for 8…25 t_q) are pre-defined and used for searching a suitable BTR parameter.

Due to PLL jitter in this test, the MultiCAN clock links directly to the external oscillator clock (f_{osc}=12MHz).

The following table shows register BTR settings inside MultiCAN module when baudrate has been detected in test.

Table 7 The BTR value in XMC by testing

CAN frame with baudrate from host CAN (e.g. CANalyzer)	BTR value(with $f_{CAN}=f_{OSC}$ = 12MHz)
1000K	0x27C0
500K	0x6FC0
250K	0x6FC1
125K	0x6FC3
100K	0x6FC4

4 Running example code

To run the **Example_6: Baudrate** , an XMC4x00 CPU kit and COM_ETH are required. See **Table 8** for the different CAN pins on XMC4x00 CPU kits.

Other examples use loop-back mode, and only one XMC4x00 CPU kit is required.

Running in loop-back mode:

- XMC4x00 CPU kit in the normal boot mode (switch: BSL=OFF, CAN/UART=does not matter).
- Connect the on-board USB connector (for power supply and debug tool) to the PC USB port.
- Start the DAVE™ tool, start the compiler, download the code and run it.

Running Example_6: Baudrate

- All hardware should be connected as in **Figure 15**.
- XMC4x00 CPU kit in the normal boot mode (switch: BSL=OFF, CAN/UART=does not matter).
- Connect the on-board USB connector (for power supply and debug tool) to the PC USB port.
- Connect CAN cable CANH/CANL between host device (the CANalyzer tool for example) and XMC4x00 board:

In CANalyzer tool (host device):

- Set the baud rate and active setting ACK.
- Insert a generator block and create one data frame (recommended: ID=0x555, bytes=0x55).

After all of the steps above have been successfully completed, the tool can be started:

- Start the DAVE tool, start the compiler, download the code and run it.
- Start the CANalyzer and trigger a data frame.

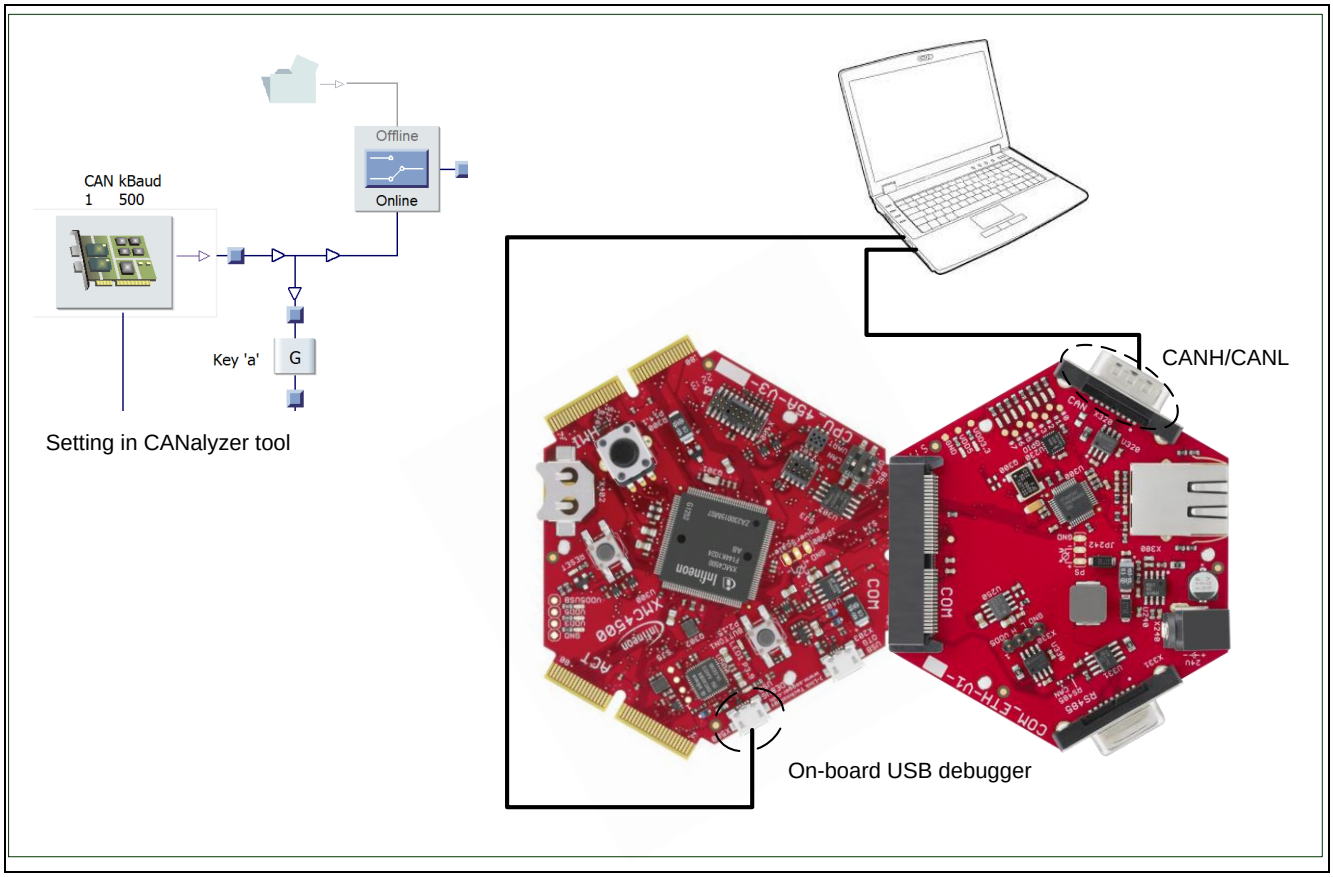

Figure 15 Running the baudrate detection example

5 DAVE™ and XMCLib

All example code supplied with this document is also created in DAVE™4 by using XMCLib. In the source code you can find information about using the MultiCAN XMCLib.

5.1 Implementation with XMCLib

This section will provide a guide to set up a basic project for CAN communication using the Infineon XMCLib.

Definition and configuration:

- Global CAN frequency definition:

```
#define CAN_FREQUENCY_120 120000000
```

- CAN bit timing configuration:

```
XMC_CAN_NODE_NOMINAL_BIT_TIME_CONFIG_t CanBaud_cfg=
{
  .can_frequency = CAN_FREQUENCY_120, // fCAN=120MHz
  .baudrate = (1000 * 1000),          // baudrate=1000K
  .sample_point = (80 * 100),         // Sample point=80%
  .sjw = 2                            // SJW=1+1
};
```

- User CAN message object definition:

```
XMC_CAN_MO_t  userSW1_MO8_Tx = {
  .can_mo_type      = XMC_CAN_MO_TYPE_TRANSMSGOBJ,
  .can_id_mode      = XMC_CAN_FRAME_TYPE_STANDARD_11BITS,
  .can_priority     = XMC_CAN_ARBITRATION_MODE_ORDER_BASED_PRIO_1,
  .can_identifier   = (uint32_t)0x111,
  .can_id_mask      = (uint32_t)0x7ff,
  .can_ide_mask     = 1U,
  .can_mo_ptr       = (CAN_MO_TypeDef*)CAN_MO8,
  .can_data_length  = (uint8_t)8,
  .can_data[1]      = 0x88888888,
  .can_data[0]      = 0x88888888
};
```

Initialization:

- Global initialization:

```
// release MultiCAN module via PRSTAT1,
// Configuration of CAN clock:
// registers: CAN->CLC and CAN->FDR, fcan=120Mhz
XMC_CAN_Init((CAN_GLOBAL_TypeDef*)CAN, CAN_FREQUENCY_120);
```

- CAN node initialization:

```
// CAN node configuration and message object configuration
```

```
XMC_CAN_NODE_NominalBitTimeConfigure(CAN_NODE2,  &CanBaud_cfg);
XMC_CAN_NODE_EnableConfigurationChange(CAN_NODE2);
XMC_CAN_NODE_EnableLoopBack(CAN_NODE2);
XMC_CAN_NODE_DisableConfigurationChange(CAN_NODE2);
```

- Initialization Message object initialization

```
// Configuration of the CAN Message Object List Structure:
XMC_CAN_AllocateMOtoNodeList((CAN_GLOBAL_TypeDef*)CAN, 2, 8);

// Configuration of the CAN Message Objects:
XMC_CAN_MO_Config(&userSW1_MO8_Tx);
```

- Disable configuration mode and enable CAN node

```
//   Start the CAN Nodes:
XMC_CAN_NODE_DisableConfigurationChange (CAN_NODE2);
XMC_CAN_NODE_ResetInitBit(CAN_NODE2);
```

Definition and configuration Function implementation:

- Data frame transmission: following code is usually called for data frame transmission

```
// test value
uint8_t TestSW1_TxData[8]={0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77}

// load DLC and TxData bytes
userSW1_MO8.can_data_length = 8;
for(i=0; i<8; i++)
userSW1_ MO8.can_data_byte[i] =TestSW1_TxData[i];
XMC_CAN_MO_UpdateData(&userSW1_MO8);

// set trigger
XMC_CAN_MO_Transmit(&userSW1_MO8);
```

- Data frame reception

```
// update all information defined in user CAN message object SW_MO
XMC_CAN_MO_Receive(&userSW1_MO16_Rx);
// update only data bytes in user CAN message object SW_MO
XMC_CAN_MO_ReceiveData(&userSW1_MO16_Rx);
```

6 Appendix

6.1 Kit information

Infineon provides several **XMC4000 Application kits.**The CAN interfaces use different pins (hard-wired) on different board versions.

Table 8 XMC4000 Application Kit signal connections

Host board kits		
	CAN node (in the normal boot mode)	**LED**
CPU_45A_V3 (XMC4500_AC)	CAN connector is not available on CPU_45A_V3. CAN pins must be connected via COM pin connector. CAN2_TxD=P1.9 => Pin28. CAN2_RxD=P1.8 => Pin30.	P3.9
CPU_44A_V2 (XMC4400_AB)	CAN connector is not available on CPU_44A_V2. CAN pins must be connected via COM pin connector. CAN1_TxD=P1.12 => Pin28. CAN1_RxD=P1.4 =>Pin30.	P1.8
COM pin connector	Pin connector between CPU kit and COM_ETH kit.	
COM_ETH_V1	CAN transceiver with CAN connector (DE-9). CAN signal: Pin28=CAN_TXD. Pin30=CAN_RXD.	
CPU_42A_V1 (XMC4200_AA)	CAN connector is available on CPU_42A_V1: CAN transceiver with CAN connector (DE-9). CAN1_TxD=P1.5. CAN1_RxD=P1.4.	P2.1

Note: On all XMC CPU kits an external 12 MHz crystal provides the clock signal to the XMC microcontroller.

7 Revision History

Major changes since the last revision

Page or Reference	Description of change
V1.0	Initial Version

Trademarks of Infineon Technologies AG

AURIX™, C166™, CanPAK™, CIPOS™, CIPURSE™, CoolGaN™, CoolMOS™, CoolSET™, CoolSiC™, CORECONTROL™, CROSSAVE™, DAVE™, DI-POL™, DrBLADE™, EasyPIM™, EconoBRIDGE™, EconoDUAL™, EconoPACK™, EconoPIM™, EiceDRIVER™, eupec™, FCOS™, HITFET™, HybridPACK™, ISOFACE™, IsoPACK™, i-Wafer™, MIPAQ™, ModSTACK™, my-d™, NovalithIC™, OmniTune™, OPTIGA™, OptiMOS™, ORIGA™, POWERCODE™, PRIMARION™, PrimePACK™, PrimeSTACK™, PROFET™, PRO-SIL™, RASIC™, REAL3™, ReverSave™, SatRIC™, SIEGET™, SIPMOS™, SmartLEWIS™, SOLID FLASH™, SPOC™, TEMPFET™, thinQ!™, TRENCHSTOP™, TriCore™.

Other Trademarks

Advance Design System™ (ADS) of Agilent Technologies, AMBA™, ARM™, MULTI-ICE™, KEIL™, PRIMECELL™, REALVIEW™, THUMB™, µVision™ of ARM Limited, UK. ANSI™ of American National Standards Institute. AUTOSAR™ of AUTOSAR development partnership. Bluetooth™ of Bluetooth SIG Inc. CAT-iq™ of DECT Forum. COLOSSUS™, FirstGPS™ of Trimble Navigation Ltd. EMV™ of EMVCo, LLC (Visa Holdings Inc.). EPCOS™ of Epcos AG. FLEXGO™ of Microsoft Corporation. HYPERTERMINAL™ of Hilgraeve Incorporated. MCS™ of Intel Corp. IEC™ of Commission Electrotechnique Internationale. IrDA™ of Infrared Data Association Corporation. ISO™ of INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. MATLAB™ of MathWorks, Inc. MAXIM™ of Maxim Integrated Products, Inc. MICROTEC™, NUCLEUS™ of Mentor Graphics Corporation. MIPI™ of MIPI Alliance, Inc. MIPS™ of MIPS Technologies, Inc., USA. muRata™ of MURATA MANUFACTURING CO., MICROWAVE OFFICE™ (MWO) of Applied Wave Research Inc., OmniVision™ of OmniVision Technologies, Inc. Openwave™ of Openwave Systems Inc. RED HAT™ of Red Hat, Inc. RFMD™ of RF Micro Devices, Inc. SIRIUS™ of Sirius Satellite Radio Inc. SOLARIS™ of Sun Microsystems, Inc. SPANSION™ of Spansion LLC Ltd. Symbian™ of Symbian Software Limited. TAIYO YUDEN™ of Taiyo Yuden Co. TEAKLITE™ of CEVA, Inc. TEKTRONIX™ of Tektronix Inc. TOKO™ of TOKO KABUSHIKI KAISHA TA. UNIX™ of X/Open Company Limited. VERILOG™, PALLADIUM™ of Cadence Design Systems, Inc. VLYNQ™ of Texas Instruments Incorporated. VXWORKS™, WIND RIVER™ of WIND RIVER SYSTEMS, INC. ZETEX™ of Diodes Zetex Limited.

Last Trademarks Update 2014-07-17

www.infineon.com

Edition 2015-07

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2015 Infineon Technologies AG. All Rights Reserved.

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference

AP32300

Legal Disclaimer

THE INFORMATION GIVEN IN THIS APPLICATION NOTE (INCLUDING BUT NOT LIMITED TO CONTENTS OF REFERENCED WEBSITES) IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office. Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.