XS_CAN r01d1

User Manual

Document version	v2.4
Document status	Released
Date of issue	11 August, 2026
Confidentiality Security Class	C-SC2 Confidential

Mobility Electronics
Robert Bosch GmbH

1 XS_CAN

The XS_CAN is a CAN communication controller IP supporting CAN Classic (CAN CC), CAN Flexible Data-Rate (CAN FD) and CAN extended data field length (CAN XL). It can be integrated as part of an SoC. It is described in VHDL on RTL level, prepared for synthesis. The XS_CAN performs communication according to ISO11898-1:2024, see [1] for more details. Additional transceiver hardware is required for connection to the CAN bus (physical layer). CAN messages to be transmitted and received are stored in a local memory (LMEM) outside the IP. Multiple XS_CAN controllers can share the same LMEM.

The XS_CAN can be connected to a HOST CPU via the 32-bit AHB slave interface. The clock domain concept allows the separation between the high precision CAN clock and the HOST clock, which may be generated by an FM-PLL.

1.1 Key Features

CAN CC, CAN FD and CAN XL as specified in ISO 11898-1:2024 [1]
Supports CAN CC with payload up to 8 bytes and 1Mbps
Supports CAN FD with payload up to 64 bytes and 8Mbps
Supports high speed CAN FD light Commander up to 8Mbps
Supports CAN XL with payload up to 2048 bytes and up to 20Mbps
1 TX FIFO Queue
2 RX FIFO Queues
1 TX Priority Queue with a programmable number of slots, up to 32
1 TX Event FIFO Queue programmable up to 64 elements
2 Cut Through Buffers (CTB)
Two modes of operation- Cut Through Mode (CTM) only with CTDMA Add-On Module and Full Message Mode (FMM)
Virtual buffers for external Direct Memory Access (DMA) friendly interface.
RX message filtering with up to 127 filter elements and 254 Reference Value-Mask pairs
AHB Slave Interface to communicate with the host (compliant to AMBA 2 ARM Ltd protocol, see [5])
Local Memory Interface to communicate with local memory external to the IP
Multiple XS_CAN can share the same LMEM
Maskable module interrupts in four categories: TX Functional, RX Functional, Error and Safety
Three clock domains (HOST, CAN, TIMEBASE)

1.2 Block Diagram

The following block diagram shows the internal structure of the XS_CAN IP and the interconnection to the SoC. The chapter “Functional Description” provides detailed information to this figure.
Clocks and resets are detailed in the chapter ‘Clock Domains’.

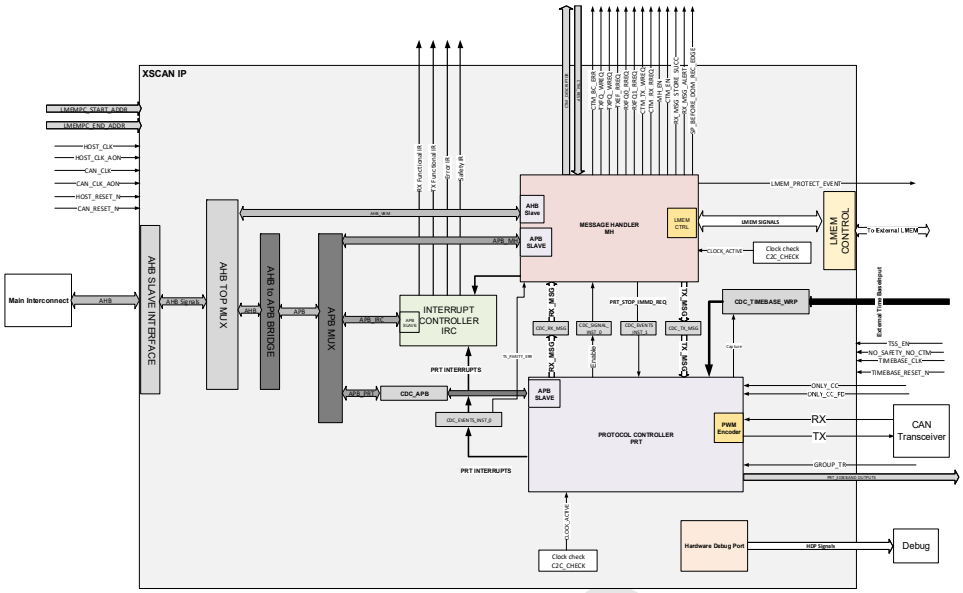


Figure 1. XS_CAN Block Diagram

1.3 Hardware Interface

The following table provides the list of signals which require to be connected at SoC pin level. The full list of signals of IP interface is provided in the chapter 'Detailed Design Information', section 'Port Description'.

Table 1. Hardware Interface

Signal	Bit Width	I/O	Description	Active Level	Clock Domain
CAN Physical Layer					
CAN_RX	1	I	Serial digital receive from CAN Transceiver	na	async
TXD	1	O	Serial digital transmit to CAN Transceiver	na	CAN
Hardware Debugging					
HDP	16	O	Hardware Debug Port	na	HOST

The connection to the CAN Transceiver via **CAN_RX** and **TXD** signals is mandatory for CAN communication.
The vector **HDP** is intended to be multiplexed to SoC output pins in a special XS_CAN hardware debug mode. It is highly recommended to support hardware debugging of XS_CAN on SoC level.

1.4 TOP - Top Level

The top level of the XS_CAN embeds all digital blocks required for communication on one CAN bus. To start up the XS_CAN, the Message Handler and the Protocol Controller must be configured beforehand. The XS_CAN can be started by writing the start command to PRT. This will enable the MH. MH can be separately enabled and disabled by the host only if PRT is not started. Details of MH and PRT are described later.

The blocks of the top level are described in the following chapters.

1.4.1 Software Interface

1.4.1.1 XS_CAN REGISTER ADDRESS Map

The register banks of the modules are address mapped by the AHB Multiplexer as depicted in the following figure. Register modules have APB interface and the top level AHB transactions are internally converted to APB transactions via a bridge. If the registers are accessed when reset is active, XS_CAN does not give slave error response.

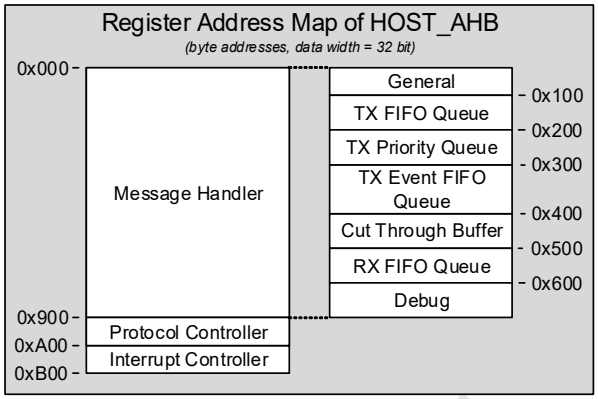


Figure 2. XS_CAN_TOP Register Address Map

Detailed register description of Message Handler (MH), Protocol Controller (PRT) and Interrupt Controller (IRC) can be found in the 'Software Interface' section of the respective chapters.

1.4.1.2 XS_CAN MEMORY Map

Table 2. XS_CAN IP Memory Map

Start Location Address	End Location Address	Symbol	Name
0x00000	0x008FC	MH reg	Message Handler registers
0x00900	0x009FC	PRT reg	Protocol Controller registers
0x00A00	0x00AFC	IRC reg	Interrupt Controller registers
0x00B00	0x00FFC	reserved	reserved
0x01000	0x01808	TXFQ	TX FIFO Queue (TXFQ)
0x0180C	0x01FFC	reserved	reserved
0x02000	0x02808	TXPQ	TX Priority Queue (TXPQ)
0x0280C	0x02FFC	reserved	reserved
0x03000	0x03810	RXFQ0	RX FIFO 0 Queue (RXFQ0)
0x03814	0x03FFC	reserved	reserved
0x04000	0x04810	RXFQ1	RX FIFO 1 Queue (RXFQ1)
0x04814	0x04FFC	reserved	reserved
0x05000	0x05010	TEFQ	TX Event FIFO Queue (TEFQ)
0x05014	0x05FFC	reserved	reserved
0x06000	0x0603C	CTB	Cut Through Buffer (CTB)
0x06040	0x3FFFC	reserved	reserved
0x40000	0x7FFFC	LMEM TA	LMEM Transparent Access

1.4.2 Functional Description

1.4.2.1 AHB (Advanced High performance Bus) to APB (Advanced Peripheral Bus) Bridge

AHB to APB bridge is used for converting AHB transactions to APB transactions for writing and reading the registers of PRT, MH and IRC through HOST_AHB interface. This bridge is required because the top level slave interface of the IP is AHB and the registers have APB interface.It is situated between AHB top mux and APB top mux. The bridge works on **HOST_CLK_AON**. The interrupt generation logic for illegal accesses to registers of MH, PRT and IRC is present in the bridge. The generated IR is HOST_ARA.

1.4.2.1.1 Description

Since APB does not support burst operation, host must ensure that accesses to the configuration registers must be single accesses. If the host issues burst accesses to registers, those are ignored with OK response and no flag/IR is raised.

Reading or writing of 32-bit (1 word) data is supported. Unaligned write accesses (e.g., 8-bit or 16-bit) are ignored with an OK response and unaligned read accesses are given read data='hCAFECAFE with OK response. No flag/IR is raised in this case.
Strobe is not supported on APB bus.
If HOST reads/writes a register which is in the reserved area, the IP gives an ERROR response back to the host. In case of a write, the transaction is ignored and in case of a read, the default data 0xCAFECAFE is given. The interrupt HOST_ARA in IRC is raised if enabled, for both write and read one clock cycle after HOST_AHB_HREADYOUT is given for that access. This is applicable for registers in PRT, MH and IRC. ERROR response is given to reserved address access within the range 0x00000-0x00FFC.

Note that IP gives ERROR response to read from write only register addresses and write to read only register addresses in MH, PRT and IRC. In case of a write, the transaction is ignored and in case of a read, the default data 0xCAFECAFE is given. The interrupt HOST_ARA in IRC is raised for both write and read to such registers in MH, PRT and IRC, if enabled. Interrupt is generated one clock after HOST_AHB_HREADYOUT is given for that access. The only exception to this is write access to **PRT.FIMC** and **PRT.TEST** registers when they are in read-only mode. In this case, OK response is given. Refer section 1.6.6.6 Host Access to Protocol Controller in [11].

1.4.2.2 AHB Top Multiplexer

AHB Top Multiplexer decodes the host AHB accesses and redirects it to the message handler or to AHB to APB bridge. All LMEM accesses are redirected to the AHB interface of the MH and all the register accesses are redirected to the AHB to APB bridge.

1.4.2.3 APB Multiplexer

APB Multiplexer module decodes the APB transactions coming from AHB to APB bridge and redirects it to the correct module. All register accesses to the message handler, protocol controller and the interrupt controller are decoded by this multiplexer.

1.4.2.4 Message Handler

All functions concerning the storage and scheduling of CAN messages are implemented in the Message Handler (MH). The TX path supports one TX FIFO Queue, one TX Priority Queue and one TX Event FIFO Queue. The RX path provides 2 RX FIFO Queues. RX Filters provide methods to accept or deny CAN Messages and to determine the target RX FIFO for data storage. Message Handler supports 2 modes of operation: FMM (Full Message Mode) and CTM (Cut-Through Mode).

The MH is configured and controlled by HOST CPU. CAN messages are fetched from or enqueued to LMEM by accessing the virtual buffers (VB) provided by the MH via HOST_AHB interface. The MH communicates with the LMEM via LMEM interface. Depending on the chosen SoC integration, multiple XS_CAN can share the same LMEM.

1.4.2.5 Protocol Controller

The Protocol Controller (PRT) performs CAN communication as standardized in [1] (CAN CC and CAN FD,and CAN XL). The PRT also supports CAN FD Light Commander communication as specified in CiA 604-4 The bitrate can be configured to values up to 20MBit/s. For the connection to the physical layer, additional transceiver hardware is required.

The PRT does not provide internal buffering of frames. Data is transferred by XS_CAN internal message buses in 32 bit slices in real-time (from) to PRT, while (de)-serializing them on the CAN Bus. Thus, single data transfers at the internal message buses are closely time-synchronized to the schedule at the CAN bus.

The module PWME inside PRT implements the PWM encoding as specified in [1].

1.4.2.6 Clock Check

The XS_CAN gets two types of clock inputs. One type of clock that goes to the MH and PRT submodule's APB configuration interface (HOST_CLK_AON and CAN_CLK_AON) should always be on and never turned off. The second type of clock (HOST_CLK and CAN_CLK) is for internal logic and could be switched off externally by the system for power down mode. There is no internal mechanism in the IP to turn off the clocks. The clock to clock checker module checks the HOST_CLK and CAN_CLK and provides information to MH and PRT whether it is on or off.

1.4.2.7 CDC

The IP supports three clocks- Host clock (HOST_CLK), CAN clock (CAN_CLK) and Timebase clock (TIMEBASE_CLK). PRT works in CAN clock domain and other modules work in host clock domain. The timebase clock is used to capture the timebase input and is then synchronized to CAN clock. CDC blocks are used to synchronize signals between these 3 clock domains. Note that the CDC blocks cannot be enabled/disabled by the software and are always enabled.
The clock supply of the Protocol Controller must fulfill the requirements for the CAN communication, described in PRT chapter. Detailed description is provided by chapter 'Clock Domains'.

1.4.2.8 LMEM Interface

The XS_CAN needs an external memory locally where all the messages and RX Filter elements are stored. This is called Local Memory (LMEM). The IP communicates with the LMEM via this interface. Up to 256KB of LMEM is accessible by a single XS_CAN IP instance. The address width is 18bits and the data width is 32 bits. All addresses are word aligned (32 bits) and hence the lower 2bits of address are tied to 0 at LMEM interface.Data is always stored into LMEM as words(32 bits).The LMEM interface signals and timing diagrams are shown in the following sub chapters. The LMEM must provide a ready signal as input to the IP to indicate that the LMEM finished the request. This ready input is taken as acknowledgment by the IP that the transfer is successful.

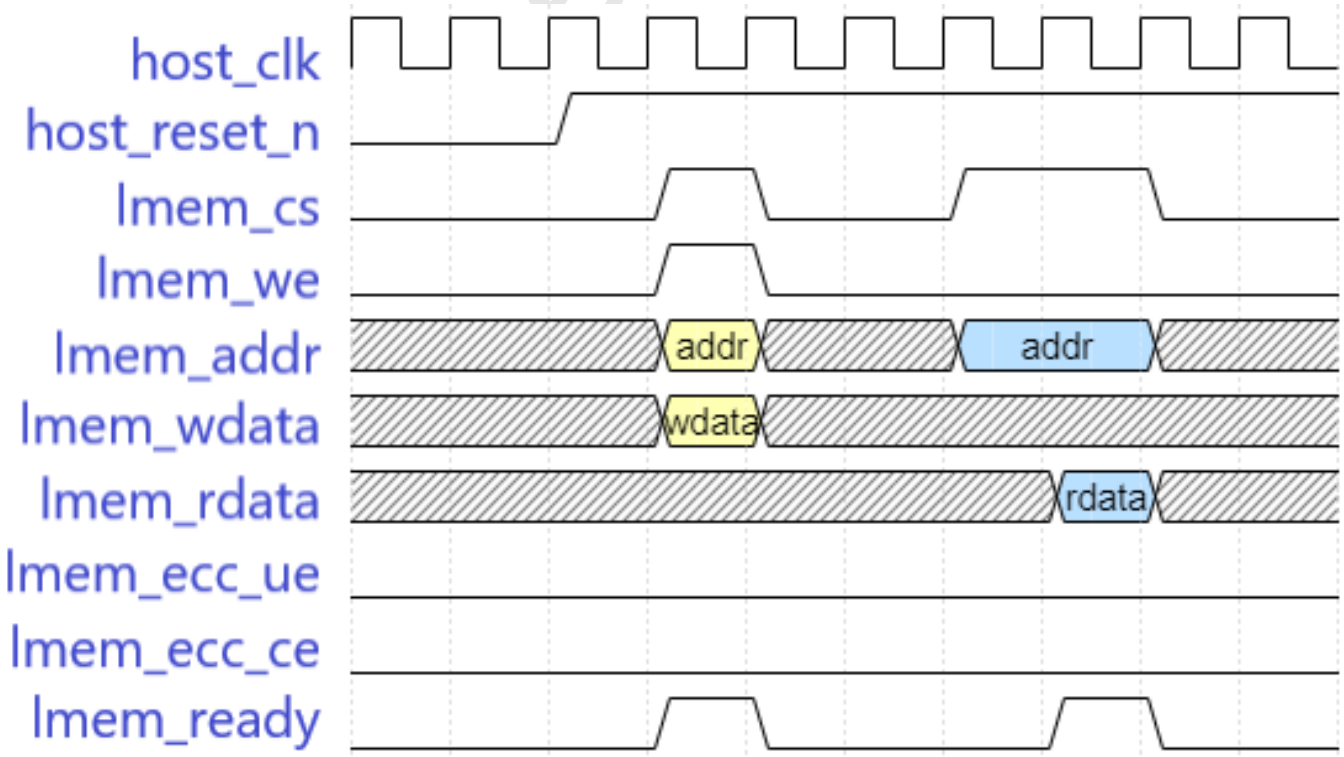
LMEM writes can be completed in one host clock cycle if LMEM_READY is already high. So the minimum length of a single write access is one host clock cycle i.e. LMEM latency =1 host clock for write. When LMEM_READY is zero when the request is placed, the access is extended to more than one host clock cycle i.e. LMEM latency is greater than 1.

LMEM read accesses has a minimum length of two host clock cycles i.e. LMEM latency for read=2 host clocks. Once the request is placed to LMEM, the read data is sampled by the IP in the next clock cycle if LMEM_READY is high. If LMEM_READY is low, the access gets extended to more than two host clock cycles, i.e. LMEM latency is greater than 2.

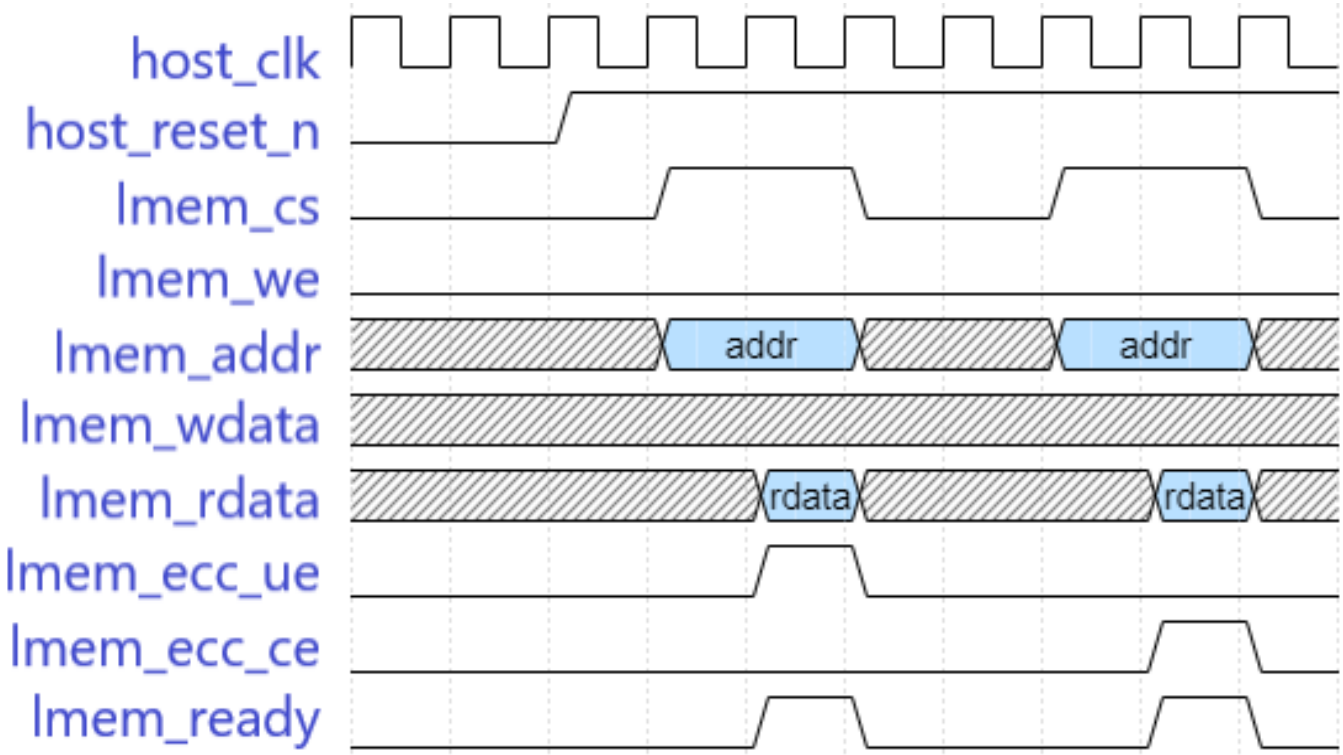
The LMEM interface itself does not have an upper limit for the delay of one ready signal.

1.4.2.8.1 Timing Diagram

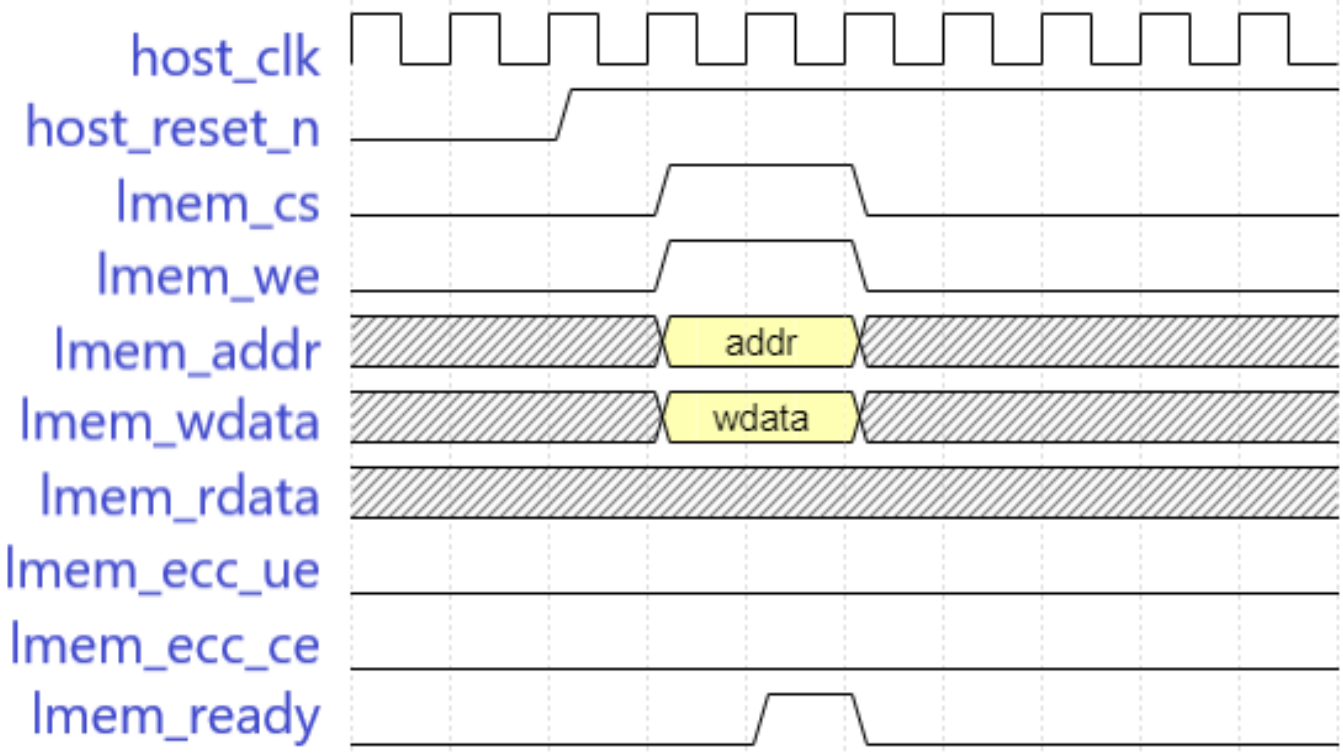
1.4.2.8.1.1 Normal read and write transaction



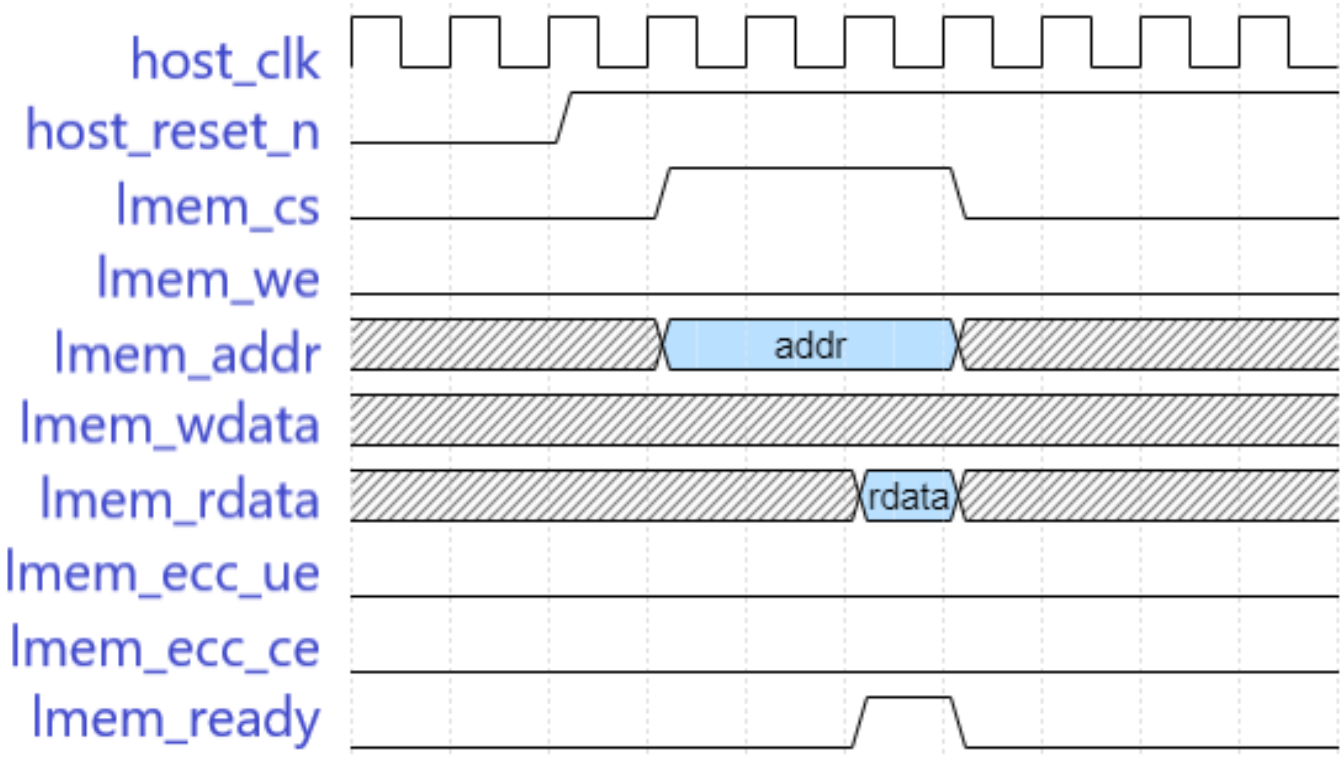
1.4.2.8.1.2 Read transaction with ECC error pulse high



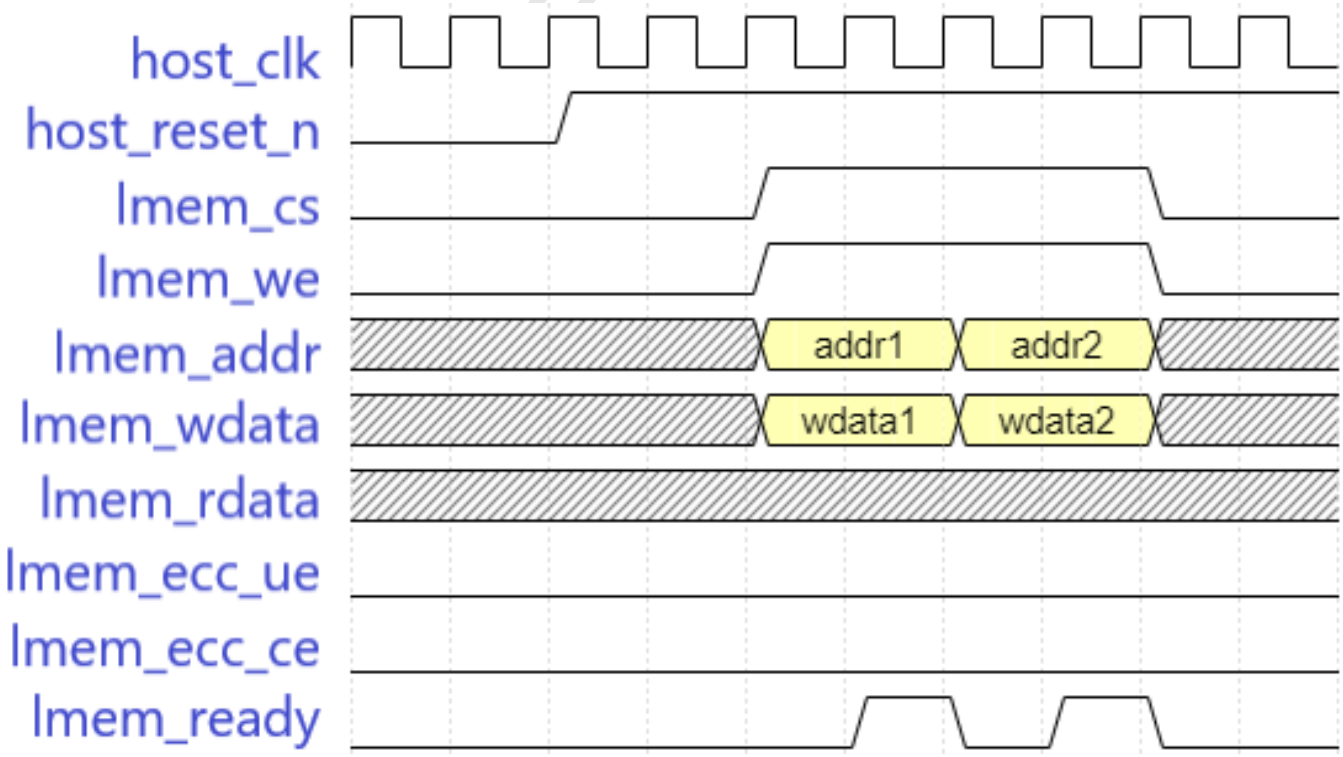
1.4.2.8.1.3 Write transaction with wait (ready delayed)



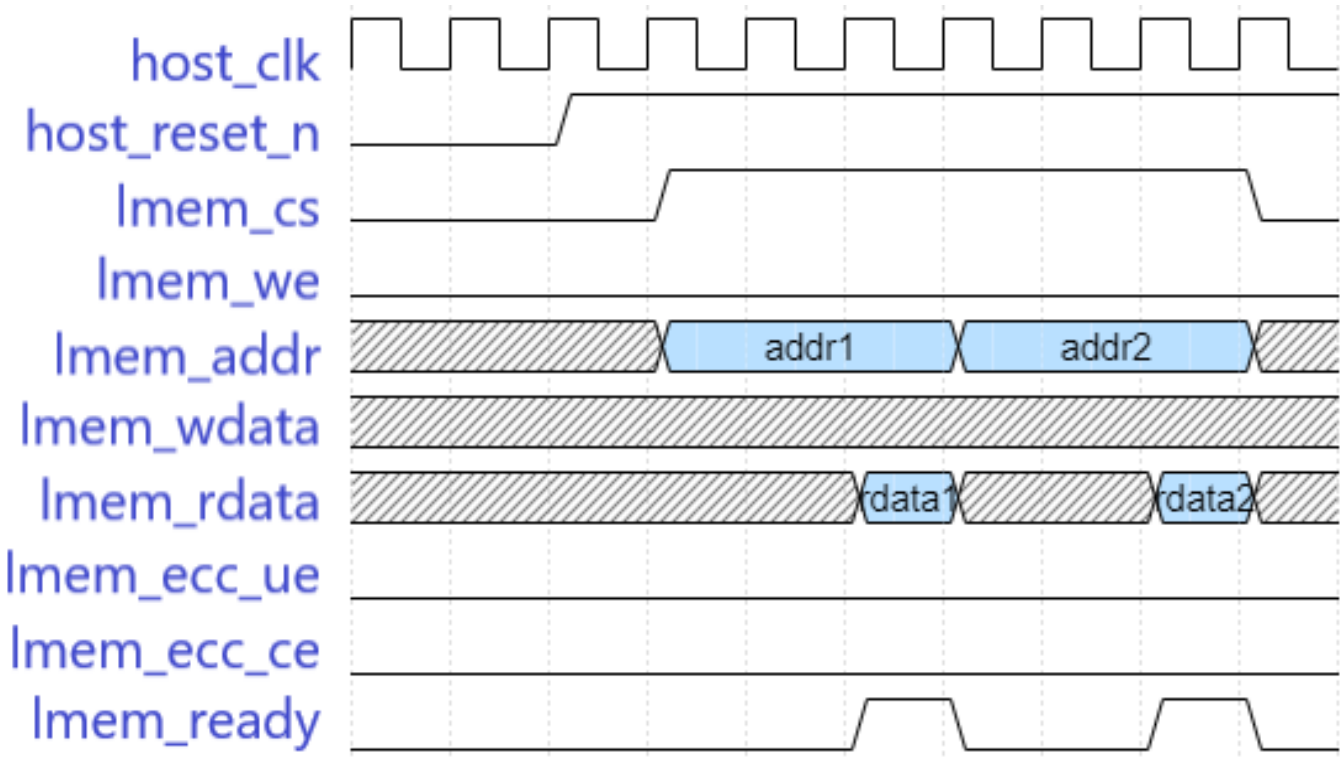
1.4.2.8.1.4 Read transaction with wait (ready delayed)



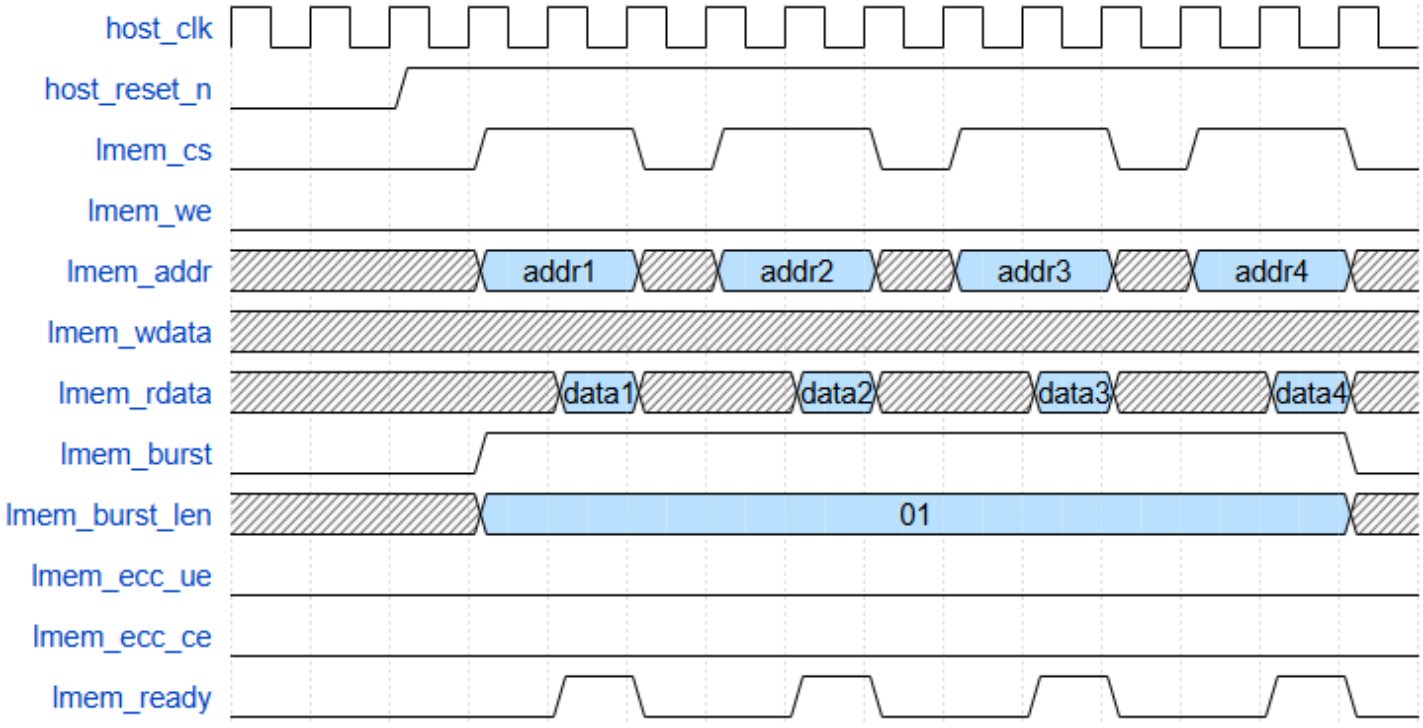
1.4.2.8.1.5 Back to Back Write with Wait State



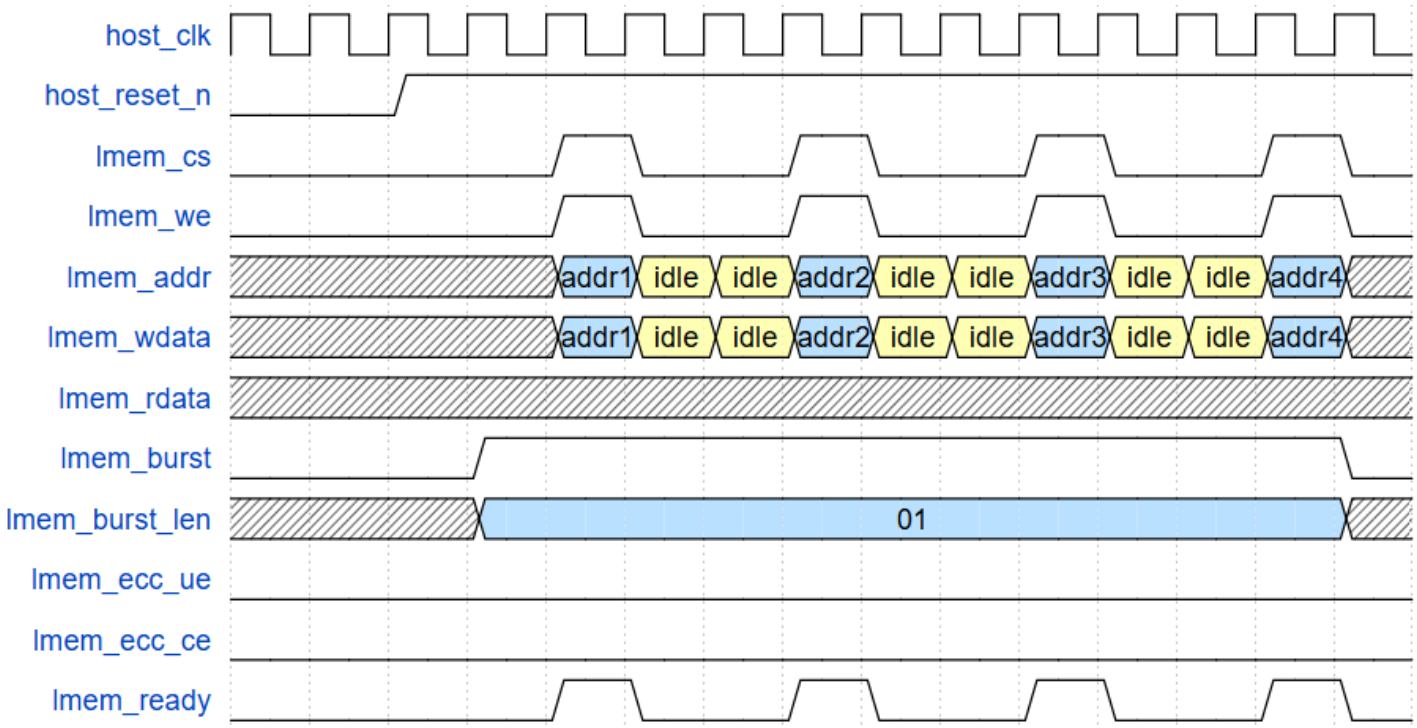
1.4.2.8.1.6 Back to Back Read with Wait State



1.4.2.8.1.7 Back to Back Read without Wait state and burst length=4



1.4.2.8.1.8 Back to Back Write without Wait State and burst length =4



During a burst access to LMEM, idle cycles are inserted between two consecutive addresses by the XS_CAN IP for internal buffering and read write data alignment. The number idle cycles vary based on whether host or CTDMA is performing the access.

1.4.2.9 HOST_AHB Slave Interface

Host communicates with XS_CAN via the top level AHB slave interface. This interface is compliant to AMBA2 protocol [5] intended for point-to-point connectivity, means one master connects to one slave. Refer the section Integration Guidelines in the reference [13] for the AHB slave interface integration requirements.
Burst accesses are supported only for LMEM accesses via AHB interface in Message Handler, this means for accesses to VBs and for transparent LMEM accesses (see Memory Map). For register accesses to all the modules via APB interface (AMBA4), burst accesses are not supported.
Note that some features of the AHB are not supported:
i) The read and write data width is always 32bits.i.e. HSIZE[2:0]=word
ii) Only OKAY and ERROR responses are supported. SPLIT and RETRY are not given.
iii) HBURST[2:0] = INCR, wrap4, wrap8 and wrap16 are not supported. If host issues these transfers, write accesses are ignored with OK response and read accesses are responded with default data 0xCAFECAFE and OK response.

Note that early burst termination feature of AHB burst is not supported by the IP i.e. HTRANS=NONSEQ and IDLE after the burst has started, is not supported. However master can issue BUSY transfers during the burst.
For burst accesses, host must ensure the correctness of the addresses issued for each beat. IP does not check this.
During reset, HOST_AHB_HREADYOUT is set to 1 by XS_CAN to avoid the bus from getting stalled.

Table 3. HOST AHB Slave Response (page 1 of 2)

Access Space	Burst Type	SIZE	RESP
Reserved Register space	SINGLE	8,16	Response= OKAY. Read data = 'hCAFECAFE and Write data is ignored
	SINGLE	32	Response = ERROR. Error interrupt HOST_ARA
	INCR, INCR4, INCR8, INCR16	8,16,32	Response= OKAY. Read data = 'hCAFECAFE and Write data is ignored

Table 3. HOST AHB Slave Response (page 2 of 2)

Access Space	Burst Type	SIZE	RESP
Valid Register address space	SINGLE	8,16	Response= OKAY. Read data = 'hCAFECAFE and Write data is ignored
	SINGLE	32	Response =OKAY with correct read data and write data accepted
	INCR, INCR4, INCR8, INCR16	8,16,32	Response= OKAY. Read data = 'hCAFECAFE and Write data is ignored
Reserved LMEM space	SINGLE/INCR/INCR4/ 8/16	32	Response= OKAY. Read data = 'hCAFECAFE and Write data is ignored. Error interrupt HOST_ARA
	SINGLE/INCR/INCR4/ 8/16	8,16	†Response= OKAY. Read data = 'hCAFECAFE and Write data is ignored. Safety interrupt MH_VB_ACCESS_VIOLATION
Valid LMEM space (TXFQ, TXPQ, RXFQ0/1, TEFQ, Transparent accesses)	SINGLE, INCR4, INCR8, INCR16	32	Response =OKAY with correct read data and write data accepted
	INCR	8,16,32	Response= OKAY. Read data = 'hCAFECAFE and Write data is ignored. Safety interrupt MH_VB_ACCESS_VIOLATION
	SINGLE, INCR4, INCR8, INCR16	8,16	Response= OKAY. Read data = 'hCAFECAFE and Write data is ignored. Safety interrupt MH_VB_ACCESS_VIOLATION
CTB	SINGLE, INCR, INCR4, INCR8	8,16,32	Response= OKAY. Read data = 'hCAFECAFE and Write data is ignored. Safety interrupt MH_VB_ACCESS_VIOLATION
	INCR16	32	Response =OKAY with correct read data and write data accepted
	INCR16	8,16	Response= OKAY. Read data = 'hCAFECAFE and Write data is ignored. Safety interrupt MH_VB_ACCESS_VIOLATION

1.4.2.10 LMEM Access Protection

The XS_CAN provides an “LMEM protection configuration” [LMEMPC] to configure LMEM protection for all accesses to the LMEM. The start address (**LMEMPC_START_ADDR**) and end address (**LMEMPC_END_ADDR**) both of 18 bit width are configured in 64 byte granularity. These are static inputs and must be provided before configuring and starting the XS_CAN. The values must remain unchanged during the operation of the IP.

XS_CAN allows accesses to LMEM only in the range of **LMEMPC_START_ADDR** and **LMEMPC_END_ADDR** addresses. Both addresses are included in this range. Since the start and end addresses are in 64 byte granularity an access is granted in the following address range: LMEMPC_START_ADDR to LMEMPC_END_ADDR. This granularity introduces a limitation where the entire LMEM space may not be fully utilized. For example, for a 4KB memory with start address 0x000, the usable range would be from 0x00 to 0xFC0 (included). Any access to an address greater than 0xFC0 will result in the setting of an error event flag and triggering of an interrupt. Depending on the source of access (host, TX Handler, RX handler, Transparent access), behavior of the IP is different. Refer Section 1.5.8 Error and Exception handling in [11] for details.

LMEM_PROTECT_EVENT, a 1-bit signal is signaled when any access outside the range of start and end address occurs. The signalling has a pulse of the length of 1 HOST_CLK cycles. By default LMEM_PROTECT_EVENT has the value 0. Transparent LMEM access addresses via VBM are also checked.

1.4.2.11 Cut Through Master DMA (CTDMA) Signals

The XS_CAN can be operated in CTM only in combination with the CTDMA. The XS_CAN does not support a system DMA to perform the block copy transfers.

The XS_CAN provides the necessary signals needed to operate the CTDMA add-on module. The CTDMA add on module is necessary to perform block copy transfer if XS_CAN is in Cut Through Mode. In CTM, the block copy needs to be performed at fast rate because of strict timing constraints dictated by the CAN bus. More details are given in the CTDMA specification [12].
The XS_CANXS_CAN provides the following signals to support the CTDMA module.

1) CTM_DESCRIPTOR (64bits): Cut Through Mode Descriptor signal output from XS_CAN consisting of the source and destination addresses of block copy transfer in CTM.
CTM_DESCRIPTOR[31:0]= MH.CTM_DESC_DEST
CTM_DESCRIPTOR[63:32]= MH.CTM_DESC_SRC

2) CTM_TX_WREQ(1 bit) and CTM_RX_RREQ (1 bit): Output Signals from XS_CAN to indicate the direction of transfer. CTM_TX_WREQ high indicates that block copy has to be fetched from SMEM and written into CTB via VBM. CTM_RX_RREQ high indicates that block copy has to be fetched from CTB via VBM and written into SMEM. Only one request is active at a time. Both CTM_TX_WREQ and CTM_RX_RREQ cannot be active simultaneously.

3) CTM_RESP (2 bits) :Response of the block copy transfer issued by CTDMA.
CTM_RESP=1: OK response
CTM_Resp=2: ERROR response
CTM_RESP=0,3 reserved

4) CTM_BC_ERR (1 bit) : Output signal from XS_CAN to indicate block copy transfer is canceled by the XS_CAN.

5) MH_EN (1 bit) : Output signal from XS_CAN indicating if MH is enabled or not. High indicates MH is enabled.

6) CTM_EN (1 bit) : Output signal from XS_CAN indicating if cut through mode is enabled or not. High indicates CTM is enabled.

1.4.2.12 Hardware Debug Port

The XS_CAN provides a 16 bit Hardware Debug Port (HDP), intended to be multiplexed to SoC output pins for debugging. Internal XS_CAN signals can be multiplexed to this interface and be observed via a logic analyzer.

The use of this feature requires deep knowledge of internal behavior of the XS_CAN and thus require support from the IP provider.

The internal signals are organized in pre-defined sets of signals The HDP signal set selection has 2 hierarchy levels. The first hierarchy level is selected by the register HDP_CTRL.HDP_SEL in Interrupt Controller module (IRC). If MH set of signals have to be observed at the HDP, HDP_SEL should be set to 0 and if PRT signals have to be observed at HDP, then HDP_SEL should be set to1.The following tables describe the signal sets. The second hierarchy level is inside the MH, where the user can select between different MH signal sets.

Table 4. HDP List

HDP [15:0]	HDP_SEL = 0 (MH Debug Port)	HDP_SEL = 1 (PRT Interface Signals)
15	MH_HDP[15]	TX_DU
14	MH_HDP[14]	RX_DO
13	MH_HDP[13]	BUS_OFF
12	MH_HDP[12]	E_PASSIVE
11	MH_HDP[11]	RX_TSS
10	MH_HDP[10]	BUS_ERR
9	MH_HDP[9]	TX_EVT
8	MH_HDP[8]	RX_EVT
7	MH_HDP[7]	STAT_ACT[1]
6	MH_HDP[6]	STAT_ACT[0]
5	MH_HDP[5]	XLT
4	MH_HDP[4]	D_RX
3	MH_HDP[3]	D_TX
2	MH_HDP[2]	SAMPLE_POINT
1	MH_HDP[1]	CAN_TX
0	MH_HDP[0]	CAN_CLK

Detailed description of the MH Debug Port can be found in the 'Trace and Debug' section 1.5.6.11 of the MH chapter in [11].

1.4.2.13 Interrupt Controller

The XS_CAN is equipped with a central interrupt controller (IRC) which performs the generation of the top level interrupts. Interrupts are categorized into 4- TX Functional, RX Functional, Error and Safety. IRC module captures all events of the MH and PRT and can be configured for each event individually to interrupt the HOST CPU.

1.4.3 Safety Mechanisms

Safety measures are implemented inside MH and PRT and not at the top level.

1.5 MH - Message Handler

1.5.1 Overview

The Message Handler submodule performs enqueuing and dequeuing of TX and RX messages in LMEM. It is designed to receive CAN messages for transmission from System Memory (SMEM) to LMEM and to send them to the PRT. On the other direction, it provides the received CAN messages at PRT to the LMEM and send to SMEM on request by the host.

All functions, concerning the storage and scheduling of CAN messages, are implemented in the Message Handler (MH). The TX path supports one TX FIFO Queue (TXFQ), one Priority Queue (TXPQ) with 32 PQ slots and one TX Event FIFO Queue (TEFQ) up to 63 elements. The RX path provides 2 RX FIFO Queues (RXFQ0, RXFQ1). There are two modes of operation in MH- Full Message Mode (FMM) and Cut Through Mode (CTM). The detailed functionality of both modes are explained in TX Handler and RX Handler chapters. The mode is defined by the host in the register MH_CFG.CTME.Note that if the input signal NO_SAFETY_NO_CTM is set to 1, MH cannot be configured in CTM and none of the safety mechanisms in MH can be enabled. Also the mode must not be changed during runtime.

MH also supports two Cut Through Buffers (CTB) in LMEM for storing messages in Cut Through Mode.Virtual addresses are provided for TXFQ, TXPQ, TEFQ, RXFQ0,RXFQ1 and CTB. The Virtual Buffer manager (VBM) module in MH converts virtual addresses to actual LMEM addresses. RX Filter elements are also stored in LMEM by the host via LMEM transparent access addresses. Refer section 1.4.1.2 XS_CAN MEMORY MAP for the addresses. RX Filtering provide methods to accept or reject CAN Messages and to determine the target RX FIFO for data storage.

The MH is configured and controlled by the host via HOST_AHB interface. CAN messages are transported between SMEM and Local Memory (LMEM) autonomously by an internal virtual buffer manager module, which is connected to HOST_AHB interface. The MH needs an LMEM, which is connected via LMEM interface. Depending on the chosen SoC integration, multiple XS_CAN can share the same LMEM.

1.5.2 Features

• AHB Slave Interface (compliant to AMBA 2 ARM Ltd protocol, see [5]):
– 32 bit data bus width
– Up to 256Kb addressable space with 18 bit address bus width
– Maximum burst length supported is 16
– Host accesses local memory via this slave interface
– Enqueuing and Dequeuing of TX and RX queues are done via this interface

• APB Slave Interface (compliant to AMBA 4 ARM Ltd protocol, see [6]):
– 32 bit data bus width
– 4Kb addressable space with 12 bit address bus width
– Configuration registers of MH are accessed via this slave interface

• LMEM Interface:
– Standard interface with 32 bit data bus width
– 18 bit address bus width for 256Kb addressable space
– Supports ready input signal from LMEM

• PRT Interface
– TX_MSG signals for TX path

– RX_MSG signals for RX path
• Two modes operation
– Full Message Mode (FMM)
– Cut Through Mode (CTM)
• Supports
– 1 TX FIFO queue
– Up to 2 RX FIFO queues
– 1 Priority Queue with a programmable number of slots, limited to 32
– 1 TX Event FIFO queue up to programmable number of 63 elements
– 2 Cut Through Buffers (CTB)
• Virtual address mapping for all queues and Cut Through Buffer via Virtual Buffer Manager (VBM) module
• RX message filtering with up to 127 filter elements and 254 Reference Value-Mask pairs
• CAN CC, CAN FD and CAN XL supported
• Provides the following safety features
– Data path parity protection
– Interface timeout protection at LMEM interface

1.5.3 Block Diagram

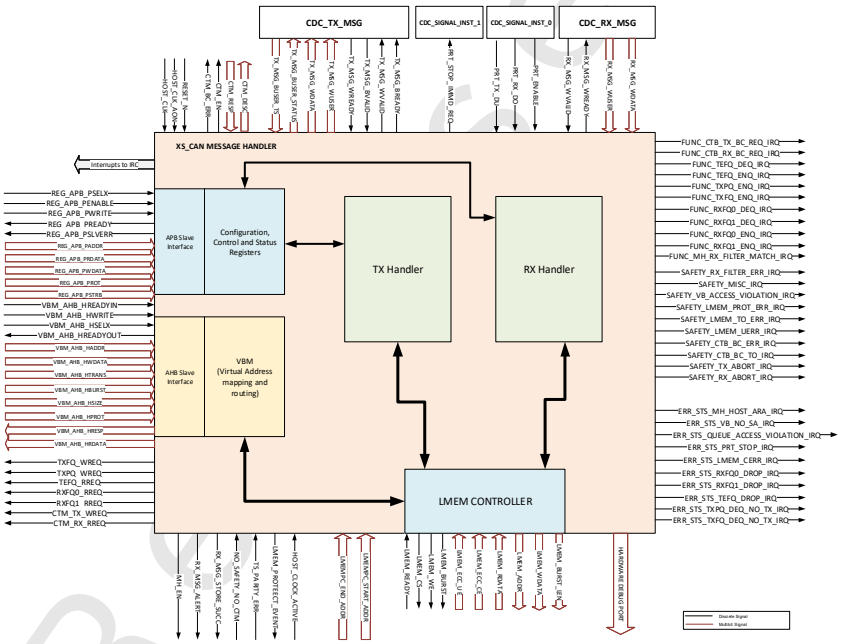


Figure 3. MH_TOP Block Diagram

1.5.4 Hardware Interface

Not Applicable.

1.5.5 Software Interface

1.5.5.1 MH MAP

1.5.5.1.1 Overview

Table 5. Address Blocks list

Address Block	Offset	Range	Description
xcans_mh_creg	0x0000	0x08FF	Usage: register Global MH control and status registers

Table 6. Registers overview (page 1 of 3)

Register name	Offset	Mode	Description
xcans_mh_creg			
XSCAN_VERSION	0x0000	R	Register size: 32 Release Identification Register
MH_CTRL	0x0004	RW	Register size: 32 Message Handler Control register
MH_CFG	0x0008	RW	Register size: 32 Message Handler Configuration register This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.
MH_STS0	0x000C	R	Register size: 32 Message Handler Status register
MH_STS1	0x0010	R	Register size: 32 Message Handler Status register.
MH_SFTY_CFG	0x0014	RW	Register size: 32 Message Handler Safety Configuration register This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.
LMEM_PROT	0x0018	R	Register size: 32 LMEM Protection Address register
RX_FILTER_CFG	0x001C	RW	Register size: 32 Global RX Filter configuration register This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.
RX_FILTER_LMEM	0x0020	RW	Register size: 32 RX Filter Start Address register This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.
TXFQ_LMEM_SA	0x0100	RW	Register size: 32 TXFQ Start Address register This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.
TXFQ_LMEM_EA	0x0104	RW	Register size: 32 TXFQ End Address register This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.
TXFQ_STS	0x0108	R	Register size: 32 TX FIFO Queue Status register
TXFQ_CFG	0x010C	RW	Register size: 32 TX FIFO Configuration register This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.
TXFQ_WPTR	0x0110	R	Register size: 32 TX FIFO Queue Write Pointer Register
TXFQ_RPTR	0x0114	R	Register size: 32 TX FIFO Queue Read Pointer Register
TXPQ_CFG	0x0200	RW	Register size: 32 TX Priority Queue configuration register This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.
TXPQ_STS0	0x0204	R	Register size: 32 TX Priority Queue Status 0 register
TXPQ_STS1	0x0208	R	Register size: 32 TX Priority Queue status 1 Register

Table 6. Registers overview (page 2 of 3)

Register name	Offset	Mode	Description
TXPQ_LMEM	0x020C	RW	Register size: 32 TX Priority Queue LMEM Address register This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.
TXPQ_WPTR	0x0210	R	Register size: 32 TX Priority Queue Write Pointer Register
TEFQ_LMEM	0x0300	RW	Register size: 32 TX Event FIFO LMEM Start Address Register This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.
TEFQ_CFG	0x0304	RW	Register size: 32 TX Event FIFO LMEM Configuration Register This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.
TEFQ_STS	0x0308	R	Register size: 32 TX Event FIFO Status register
TEFQ_WPTR	0x030C	R	Register size: 32 TX Event FIFO Queue Write Pointer Register
TEFQ_RPTR	0x0310	R	Register size: 32 TX Event FIFO Queue Read Pointer Register
CTB_LMEM	0x0400	RW	Register size: 32 Cut-Through Buffer start Address in LMEM Register This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.
CTM_DESC_SRC	0x0404	R	Register size: 32 Cut-Through Mode Source Address Register
CTM_DESC_DEST	0x0408	R	Register size: 32 Cut-Through Mode Destination Address Register
CTM_EVENT	0x040C	R	Register size: 32 Cut-Through Mode Event Register Event Flag to indicate software about active TX and RX block copy requests
RXFQ0_SA	0x0500	RW	Register size: 32 RX FIFO Queue 0 Start Address Register This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.
RXFQ0_EA	0x0504	RW	Register size: 32 RX FIFO Queue 0 End Address Register This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.
RXFQ0_RPTR	0x0508	RW	Register size: 32 RX FIFO Queue 0 Read Pointer Register
RXFQ0_WPTR	0x050C	R	Register size: 32 RX FIFO Queue 0 Write Pointer Register
RXFQ0_WRAP_PTR	0x0510	R	Register size: 32 RX FIFO Queue 0 Wrap Pointer in SMEM Register
RXFQ1_SA	0x0514	RW	Register size: 32 RX FIFO Queue 1 Start Address Register This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.
RXFQ1_EA	0x0518	RW	Register size: 32 RX FIFO Queue 1 End Address Register This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.
RXFQ1_RPTR	0x051C	RW	Register size: 32 RX FIFO Queue 1 Read Pointer Register

Table 6. Registers overview (page 3 of 3)

Register name	Offset	Mode	Description
RXFQ1_WPTR	0x0520	R	Register size: 32 RX FIFO Queue 1 Write Pointer Register
RXFQ1_WRAP_PTR	0x0524	R	Register size: 32 RX FIFO Queue 1 Wrap Pointer in SMEM Register
RXFQ_STS	0x0528	R	Register size: 32 RX FIFO Queue Status register
DEBUG_TEST_CTRL	0x0600	RW	Register size: 32 Debug Control register
TX_SCAN_WC	0x0604	R	Register size: 32 TX-SCAN winning candidates register This register gives the information on the winning candidate evaluated by the TX-Scan.
TX_SCAN_PC	0x0608	R	Register size: 32 TX-SCAN prepared candidates register This register gives the information on the prepared candidate which is in pipeline evaluated by the TX-Scan.
VBM_STATUS	0x060C	R	Register size: 32 Virtual Buffer Manager Status register

1.5.5.1.2 xcans_mh_creg Address Block

Table 7. XSCAN_VERSION register

Release Identification Register

Bit	Field	Mode	Initial Value	Description
11:8	MAJOR	R	0x1	Release name, used to identify the Major generation of the XS_CAN.
7:4	MINOR	R	0x1	Release name, used to identify the Minor generation of the XS_CAN.
3:0	PATCH	R	0x0	Release name, used to identify the Patch number of the XS_CAN.

Table 8. MH_CTRL register

Message Handler Control register

Bit	Field	Mode	Initial Value	Description
0	MH_ENABLE	R/W	0x0	If this bit is set to 1, it indicates that MH is enabled and is ready for operation. MH_ENABLE automatically is set to 1 by the IP if PRT is started and PRT enable output is set. While PRT is started, this value is read only. Only if PRT is not enabled/started, this bit becomes read-write for the host. Before MH_ENABLE is set to 1 by the host, all other configuration registers of MH must be written. After this bit is set, configuration registers are write protected.

Table 9. MH_CFG register (page 1 of 2)

Message Handler Configuration register
This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.

Bit	Field	Mode	Initial Value	Description
18:16	MAX_RETRANS	R/W	0x7	Maximum number of TX message re-transmissions. Different configurations are possible: 0 -> no re-transmission; 1 to 6 -> 1 to 6 re-transmissions; 7-> unlimited re-transmissions; Re-transmission means that a message is transmitted again, because the previous transmission attempt ended with an error on the CAN bus. Hint: This setting does not influence the number of re-arbitrations, i.e. arbitration attempts. Re-arbitrations means, the arbitration was lost and is retried.
8	CTME	R/W	0x0	When set to 1, the Cut-Through mode is active. This bit field register is accessible in write mode if the MH is not enabled(MH_CTRL.MH_ENABLE = 0) and the input signal no_safety_no_ctm is set to 0..
4	RXFQ1E	R/W	0x0	When set to 1 RXFQ1 is enabled.

Table 9. MH_CFG register (page 2 of 2)

Message Handler Configuration register
This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.

Bit	Field	Mode	Initial Value	Description
3	RXFQ0E	R/W	0x0	When set to 1 RXFQ0 is enabled.
2	TEFQE	R/W	0x0	When set to 1 TEFQ is enabled.
1	TXPQE	R/W	0x0	When set to 1 TXPQ is enabled.
0	TXFQE	R/W	0x0	When set to 1 TXFQ is enabled.

Table 10. MH_STS0 register

Message Handler Status register

Bit	Field	Mode	Initial Value	Description
30	TXFQ_ELEM_TOO_BIG	RC	0x0	Set when the message being enqueued into TXFQ is greater than TXFQ_CFG.MAX_ELEM_SIZE (FMM and CTM). The message DLC is used to determine the message size.
29	TXPQ_ELEM_TOO_BIG	RC	0x0	Set when the message being enqueued into TXPQ is greater than TXPQ_CFG.SLOT_SIZE (FMM and CTM). The message DLC is used to determine the message size.
28	TXFQ_FULL_WR	RC	0x0	Set when host tries to write to a full TXFQ(FMM and CTM)
27	TXPQ_FULL_WR	RC	0x0	Set when host tries to write to a full TXPQ(FMM and CTM)
26	TEFQ_EMPTY_RD	RC	0x0	Set when host tries to read from an empty TEFQ (FMM and CTM)
25	RXFQ0_EMPTY_RD	RC	0x0	Set when host tries to read from an empty RXFQ0 (This flag will be used in FMM only)
24	RXFQ1_EMPTY_RD	RC	0x0	Set when host tries to read from an empty RXFQ1(This flag will be used in FMM only)
23	CTB_RD_WO_REQ	RC	0x0	Set when host tries to read from CTB when there is no active block copy request i.e ctm_rx_rreq = 0 (CTM only)
22	CTB_WR_WO_REQ	RC	0x0	Set when host tries to write to CTB when there is no active block copy request i.e ctm_tx_wreq = 0 (CTM only)
21	TXFQ_VB_NO_SA	RC	0x0	Set when host issues another address instead of start address to TXFQ VB
20	TXPQ_VB_NO_SA	RC	0x0	Set when host issues another address instead of start address to TXPQ VB
19	TEFQ_VB_NO_SA	RC	0x0	Set when host issues another address instead of start address to TEFQ VB
18	RXFQ0_VB_NO_SA	RC	0x0	Set when host issues another address instead of start address to RXFQ0 VB
17	RXFQ1_VB_NO_SA	RC	0x0	Set when host issues another address instead of start address to RXFQ1 VB
16	CTB_VB_NO_SA	RC	0x0	Set when host issues another address instead of start address to CTB VB
1	CLOCK_ACTIVE	R	0x1	Status of HOST_CLK which is the core clock to MH: 0 = HOST_CLK off, 1 = HOST_CLK on.
0	PRT_ENABLE	R	0x0	Value of the ENABLE signal driven by the PRT. The PRT signalizes via ENABLE whether it is started (ENABLE = 1) and requires message handling or not (ENABLE = 0).

Table 11. MH_STS1 register (page 1 of 2)

Message Handler Status register.

Bit	Field	Mode	Initial Value	Description
17	TXFQ_VB_NONLIN_ACC	RC	0x0	Set when host issues non linear address after issuing start address for TXFQ
16	TXPQ_VB_NONLIN_ACC	RC	0x0	Set when host issues non linear address after issuing start address for TXPQ
15	TEFQ_VB_NONLIN_ACC	RC	0x0	Set when host issues non linear address after issuing start address for TEFQ

Table 11. MH_STS1 register (page 2 of 2)

Message Handler Status register.

Bit	Field	Mode	Initial Value	Description
14	RXFQ0_VB_NONLIN_ACC	RC	0x0	Set when host issues non linear address after issuing start address for RXFQ0
13	RXFQ1_VB_NONLIN_ACC	RC	0x0	Set when host issues non linear address after issuing start address for RXFQ1
12	CTB_VB_NONLIN_ACC	RC	0x0	Set when host issues non linear address after issuing start address for CTB
11	TXFQ_VB_RD	RC	0x0	Set when host tries to read from TXFQ VB
10	TXPQ_VB_RD	RC	0x0	Set when host tries to read from TXPQ VB
9	TEFQ_VB_WR	RC	0x0	Set when host tries to write to TEFQ VB
8	RXFQ0_VB_WR	RC	0x0	Set when host tries to write to RXFQ0 VB
7	RXFQ1_VB_WR	RC	0x0	Set when host tries to write to RXFQ1 VB
6	CTB_VB_TX_RD	RC	0x0	Set when host tries to read from CTB VB while the CTB VB is in write-only mode. (This is the case when CTM_TX_WREQ is active while a transmission is ongoing.)
5	CTB_VB_RX_WR	RC	0x0	Set when host tries to write to CTB VB while the CTB VB is in read-only mode. (this is the case when CTM_RX_RREQ is active while a reception is ongoing.)
4	TX_DP_SEQ_ERR	R	0x0	Set when there is an error or unexpected behavior at MH_PRT TX_MSG interface. PRT will be stopped. Restarting the PRT will reset this bit.
3	RX_DP_SEQ_ERR	R	0x0	Set when there is an error or unexpected behavior at MH_PRT RX_MSG interface.PRT will be stopped. Restarting the PRT will reset this bit.
2	TX_DP_PARITY_ERR	R	0x0	Set when there is parity mismatch in TX data path.PRT will be stopped and restarting PRT will reset this bit.
1	RX_DP_PARITY_ERR	R	0x0	Set when there is parity mismatch in RX data path.PRT will be stopped and restarting PRT will reset this bit.
0	TS_PARITY_ERR	R	0x0	Set when there is parity mismatch in Time Stamp in PRT.PRT will be stopped and restarting PRT will reset this bit.

Table 12. MH_SFTY_CFG register

Message Handler Safety Configuration register
This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.

Bit	Field	Mode	Initial Value	Description
15:8	LMEM_TIMEOUT_VAL	R/W	0x0	This value is used by the watchdog timer for the LMEM interface and defines the maximum number of timer ticks until a read or write access has to be completed. This value must be configured to the expected maximum latency on the LMEM interface. If this value is set to 0 and MH_SFTY_CFG.LMEM_TO_EN = 1 then the MEM_TO_ERR interrupt is triggered right away when accessing the LMEM.
2:1	PRESCALER	R/W	0x0	Prescaler used to generate the timer ticks for the watchdogs. . According to the value a different clock ratio can be selected: 0: clk divided by 32 1: clk divided by 64 2: clk divided by 128 3: clk divided by 256
0	LMEM_TO_EN	R/W	0x0	When set to 1, the watchdog for the LMEM interface is enabled, otherwise disabled.

Table 13. LMEM_PROT register

LMEM Protection Address register

Bit	Field	Mode	Initial Value	Description
31:20	EA	R	0x0	End address of the IP's LMEM space. This register field holds the input signal value LMEMPC_END_ADDR [17:6]. This least significant 6 bit of EA are 0, because the LMEMPC_END_ADDR has a 64 byte granularity.
15:4	SA	R	0x0	Start address of the IP's LMEM space. This register field holds the input signal value LMEMPC_START_ADDR [17:6]. This least significant 6 bit of SA are 0, because the LMEMPC_START_ADDR has a 64 byte granularity.

Table 14. RX_FILTER_CFG register

Global RX Filter configuration register

This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.

Bit	Field	Mode	Initial Value	Description
9	DEFAULT_FIFO	R/W	0x0	Number of the default RX FIFO to store RX message when AFAB=1 and/or ANMF=1 0= RXFQ0 is chosen as default Rx FIFO 1= RXFQ1 is chosen as default Rx FIFO
8	AFAB	R/W	0x0	(Accept Filter Abort) Messages for which filtering was aborted before finding a result are accepted and stored into default Rx FIFO when this bit is set 1, else they are rejected.
7	ANMF	R/W	0x0	(Accept Non-Matching Frames) Non-matching frames are accepted and stored into default Rx fifo when this bit is set 1, else they are rejected.
6:0	FE_NUM	R/W	0x0	Number of RX Filter elements for RX filtering. Allowed values are from 0 to 127 RX filter elements .If set to 0, then messages will be put into default Rx FIFO when ANMF=1, else they are dropped.

Table 15. RX_FILTER_LMEM register

RX Filter Start Address register

This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.

Bit	Field	Mode	Initial Value	Description
17:6	SA	R/W	0x0	Start address where the RX filter elements are defined in LMEM . This address value must always be 64byte aligned. The lower 6 bits (5:0) are hence 0 and not considered.

Table 16. TXFQ_LMEM_SA register

TXFQ Start Address register

This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.

Bit	Field	Mode	Initial Value	Description
17:6	ADD	R/W	0x0	Start address of the TXFQ in the LMEM.This address value must always be 64byte aligned. The lower 6 bits (5:0) are hence 0 and not considered.

Table 17. TXFQ_LMEM_EA register

TXFQ End Address register

This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.

Bit	Field	Mode	Initial Value	Description
17:6	ADD	R/W	0x0	End address of the TXFQ in the LMEM.This address value must always be 64byte aligned. The lower 6bits (5:0) are hence 0 and not considered.

Table 18. TXFQ_STS register

TX FIFO Queue Status register

Bit	Field	Mode	Initial Value	Description
8	FIFO_FULL	R	0x0	When set to 1,it indicates that the TXFQ is full .
7:0	FILL_LEVEL	R	0x0	FILL_LEVEL is the total number of messages in the TXFQ pending for transmission. Once a message is successfully enqueued FILL_LEVEL is incremented by 1 and when a message is dequeued for transmission it gets decremented by 1. Possible values are 0 to 255

Table 19. TXFQ_CFG register

TX FIFO Configuration register
This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.

Bit	Field	Mode	Initial Value	Description
5:0	MAX_ELEM_SIZE	R/W	0x0	Defines the maximum size of 1 TXFQ element that can be enqueued. The actual maximum element size is calculated as MAX_ELEM_SIZE*64 bytes. Allowed values are 0 to 63. If set to 0, TXFQ will be disabled.

Table 20. TXFQ_WPTR register

TX FIFO Queue Write Pointer Register

Bit	Field	Mode	Initial Value	Description
17:2	WPTR_ADD	R	0x0	Write pointer register shows the LMEM address of the last word of the last TXQE that was enqueued (written) successfully. The next TXQE starts from address WPTR_ADD+4. This address is updated by VBM in MH.This is a word aligned address, i.e., bits[1:0] are always 0.

Table 21. TXFQ_RPTR register

TX FIFO Queue Read Pointer Register

Bit	Field	Mode	Initial Value	Description
17:2	RPTR_ADD	R	0x0	Read pointer register shows the LMEM address of the last word of the last TXQE that was dequeued (read). The next TXQE to be dequeued starts at RPTR_ADD+4. This address is updated by TX Handler in MH. This is a word aligned address,.i.e bits[1:0] are always 0.

Table 22. TXPQ_CFG register

TX Priority Queue configuration register
This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.

Bit	Field	Mode	Initial Value	Description
17:8	SLOT_SIZE	R/W	0x0	Slot Size Defines the size of a TXPQ slot in LMEM. The configuration is in word granularity (4 byte). Actual slot size= SLOT_SIZE*4 byte. Allowed values are from 0 to 1023. If set to 0, TXPQ will be disabled.
7	TX_MSG_SEQE	R/W	0x0	Transmit Message Sequence Enable When enabled (1), messages in TXPQ that have the same priority are sent in a user defined sequence. The user can define the sequence via Message Marker [5:0] (part of header word T1). See details in according chapter. The header T1 is also read for TX SCAN for TXPQ messages if it is enabled (1).
5:0	SLOT_NUM	R/W	0x0	Slot Number Defines the number of TXPQ slots in LMEM. Allowed values are 0 to 32. Values larger than 32 will be interpreted as 32. If set to 0, TXPQ will be disabled.

Table 23. TXPQ_STS0 register

TX Priority Queue Status 0 register

Bit	Field	Mode	Initial Value	Description
31:0	SLOT_OCC	R	0x0	Slot Occupied When SLOT_OCC[n] = 1, the TXPQ slot n contains a TX message to be transmitted. The TX message in the slot n is in the LMEM and considered by the TX-Scan process. As long as SLOT_OCC[n] = 1, the TX message is not dequeued.

Table 24. TXPQ_STS1 register

TX Priority Queue status 1 Register

Bit	Field	Mode	Initial Value	Description
8	TXPQ_FULL	R	0x0	The value 1 indicates that all slots are occupied. And the TXPQ cannot store any new TX message.
5:0	FILL_LEVEL	R	0x0	FILL_LEVEL is the total number of messages in the TXPQ pending for transmission. Once a message is succesfully enqueued FILL_LEVEL is incremented by 1. When a message is dequeued, it gets decremented by 1. Possible values are 0 to 32

Table 25. TXPQ_LMEM register

TX Priority Queue LMEM Address register
This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.

Bit	Field	Mode	Initial Value	Description
17:6	SA	R/W	0x0	Start address of the TXPQ in LMEM. This address value must always be 64byte aligned. The lower 6 bits (5:0) are hence 0 and not considered. The end address of the TXPQ is not configured because the end of the queue can be determined by looking at its SA and Size. The TXPQ Size in byte is calculated as follows: TXPQ_CFG.SLOT_NUM*TXPQ_CFG.SLOT_SIZE * 4 byte.

Table 26. TXPQ_WPTR register

TX Priority Queue Write Pointer Register

Bit	Field	Mode	Initial Value	Description
17:2	WPTR_ADD	R	0x0	WPTR_ADD indicates the LMEM address corresponding to the last word of the previously enqueued TXQE. Word aligned address set at offset 2. Bits[1:0] are 0 and not considered. This register can be used for debug purpose.

Table 27. TEFQ_LMEM register

TX Event FIFO LMEM Start Address Register
This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.

Bit	Field	Mode	Initial Value	Description
17:6	SA	R/W	0x0	Start address of the TEFQ in LMEM .This address value must always be 64byte aligned. The lower 6 bits (5:0) are 0 and not considered. The end address of the TEFQ is not configured because the end of the queue can be determined by looking at its SA and Size. The TEFQ Size in byte is calculated as follows: TEFQ_CFG.TEQE_NUM*TEQE_size_in_byte. TEQE_size_in_byte is 16 bytes when TEFQ_CFG.TEQE_LARGE = 0 and 20 bytes when TEFQ_CFG.TEQE_LARGE = 1.

Table 28. TEFQ_CFG register

TX Event FIFO LMEM Configuration Register
This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.

Bit	Field	Mode	Initial Value	Description
8	TEQE_LARGE	R/W	0x0	Defines the size of a TEFQ element (TEQE). 1: TEQE size = 5 words 0: TEQE size = 4 words
5:0	TEQE_NUM	R/W	0x0	Maximum number of TEFQ elements that can be stored. Allowed values are 0 to 63. If set to 0, TEFQ is disabled.

Table 29. TEFQ_STS register

TX Event FIFO Status register

Bit	Field	Mode	Initial Value	Description
8	TEFQ_FULL	R	0x0	The value 1 indicates that the TEFQ is FULL and cannot store new elements.The value is cleared when there is a free space in TEFQ for one or more elements.
5:0	FILL_LEVEL	R	0x0	Indicates the current number of TEQEs present in the TEFQ. The value is incremented by one for each TEQE successfully enqueued into the TEFQ and decremented by one for each TEQE dequeued from the TEFQ. Possible values are 0 to 63.

Table 30. TEFQ_WPTR register

TX Event FIFO Queue Write Pointer Register

Bit	Field	Mode	Initial Value	Description
17:2	WPTR_ADD	R	0x0	This write pointer shows the LMEM address of the last word of the last TEQE that was enqueued (written) successfully. This is a word aligned address,.i.e bits[1:0] are always 0.

Table 31. TEFQ_RPTR register

TX Event FIFO Queue Read Pointer Register

Bit	Field	Mode	Initial Value	Description
17:2	RPTR_ADD	R	0x0	This read pointer shows the LMEM address of the last word of the last dequeued TEQE . This is a word aligned address,.i.e., bits[1:0] are always 0.

Table 32. CTB_LMEM register

Cut-Through Buffer start Address in LMEM Register
This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.

Bit	Field	Mode	Initial Value	Description
17:6	SA	R/W	0x0	Start address of the cut through buffer in LMEM. This address value must always be 64byte aligned. The lower 6 Bits (5:0) are hence 0 and not considered.

Table 33. CTM_DESC_SRC register

Cut-Through Mode Source Address Register

Bit	Field	Mode	Initial Value	Description
31:0	ADD	R	0x0	Source Address of cut-through descriptor .For TX direction, this will be SMEM address and for RX direction this will be CTB address. This value is valid when CTB_TX_WREQ = 1 or CTB_RX_RREQ = 1.

Table 34. CTM_DESC_DEST register

Cut-Through Mode Destination Address Register

Bit	Field	Mode	Initial Value	Description
31:0	ADD	R	0x0	Destination Address of cut-through descriptor.For TX direction, this will be CTB address and for RX direction this will be SMEM address

Table 35. CTM_EVENT register

Cut-Through Mode Event Register
Event Flag to indicate software about active TX and RX block copy requests

Bit	Field	Mode	Initial Value	Description
1	RX_BLOCK_COPY_REQ	R	0x0	A value of 1 shows that a block copy request is signaled via output port CTM_RX_RREQ. A value of 0 shows that there is no pending block copy for receive direction. This value is for debug purposes.
0	TX_BLOCK_COPY_REQ	R	0x0	A value of 1 shows that a block copy request is signaled via output port CTM_TX_WREQ. A value of 0 shows that there is no pending block copy for transmit direction. This value is for debug purposes.

Table 36. RXFQ0_SA register

RX FIFO Queue 0 Start Address Register
This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.

Bit	Field	Mode	Initial Value	Description
31:6	ADD	R/W	0x0	In FMM,ADD[17:0] stores the start address of RXFQ0 in LMEM. In CTM,ADD stores the start address of RXFQ0 in SMEM In both modes, the address must be 64byte aligned. The lower 6 bits(5:0) are hence 0 and not considered.

Table 37. RXFQ0_EA register

RX FIFO Queue 0 End Address Register
This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.

Bit	Field	Mode	Initial Value	Description
31:6	ADD	R/W	0x0	In FMM,ADD[17:0] stores the end address of RXFQ0 in LMEM. In CTM,ADD stores the end address of RXFQ0 in SMEM In both modes, the address must be 64byte aligned. The lower 6 bits(5:0) are hence 0 and not considered.

Table 38. RXFQ0_RPTR register

RX FIFO Queue 0 Read Pointer Register

Bit	Field	Mode	Initial Value	Description
31:2	ADD	R/W	0x0	In FMM, ADD[17:0] indicates the address in LMEM RXFQ0 from where the last word of the last dequeued (read) RXQE is read by the host. In FMM, this is updated by the IP and only for debugging purposes. This is word aligned address. In CTM, ADD indicates the address in SMEM RXFQ0 from where the last word of the last dequeued (read) RXQE is read by the host. In CTM, the host writes this value to inform the IP up to which address the RXFQ0 is dequeued. This is word aligned address.The ADD[1:0] bits are always 0 and not considered.

Table 39. RXFQ0_WPTR register

RX FIFO Queue 0 Write Pointer Register

Bit	Field	Mode	Initial Value	Description
31:2	ADD	R	0x0	In FMM, ADD[17:0] indicates the address in LMEM RXFQ0 where the last word of the last enqueued (write) RXQE is written by the MH. In CTM, ADD indicates the address in SMEM RXFQ0 where the last word of the last enqueued (write) RXQE is written by the CTDMA (performs SMEM accesses for the XS_CAN). This value is updated by the IP in FMM and CTM and is a word aligned address.The ADD[1:0] bits are always 0. This value is for debug purposes.

Table 40. RXFQ0_WRAP_PTR register

RX FIFO Queue 0 Wrap Pointer in SMEM Register

Bit	Field	Mode	Initial Value	Description
31:2	SMEM_ADD	R	0x0	In FMM the value is not defined. In CTM, SMEM_ADD indicates the address in SMEM at which RXFQ0 is wrapped around to store the next full RX message.This register is updated by the IP. The value is word aligned. The SMEM_ADD[1:0] bits are always 0.

Table 41. RXFQ1_SA register

RX FIFO Queue 1 Start Address Register
This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.

Bit	Field	Mode	Initial Value	Description
31:6	ADD	R/W	0x0	In FMM,ADD[17:0] stores the start address of RXFQ1 in LMEM. In CTM,ADD stores the start address of RXFQ1 in SMEM In both modes, the address must be 64byte aligned. The lower 6 bits(5:0) are always 0 and not considered.

Table 42. RXFQ1_EA register

RX FIFO Queue 1 End Address Register
This register is accessible in write mode when the MH is not started, otherwise it is only readable. See MH_CTRL.MH_ENABLE = 0.

Bit	Field	Mode	Initial Value	Description
31:6	ADD	R/W	0x0	In FMM,ADD[17:0] stores the end address of RXFQ1 in LMEM. In CTM,ADD stores the end address of RXFQ1 in SMEM In both modes, the address must be 64byte aligned. The lower 6 bits(5:0) are always 0 and not considered.

Table 43. RXFQ1_RPTR register

RX FIFO Queue 1 Read Pointer Register

Bit	Field	Mode	Initial Value	Description
31:2	ADD	R/W	0x0	In FMM, ADD[17:0] indicates the address in LMEM RXFQ1 from where the last word of the last dequeued (read) RXQE is read. In FMM, this is updated by the IP and only for debugging purposes. This is a word aligned address. In CTM, ADD indicates the address in SMEM RXFQ1 from where the last word of the last dequeued (read) RXQE is read by the host. In CTM, the host writes this value to inform the IP up to which address the RXFQ1 is dequeued. This is a word aligned address.The ADDR[1:0] bits are always 0 and not considered.

Table 44. RXFQ1_WPTR register

RX FIFO Queue 1 Write Pointer Register

Bit	Field	Mode	Initial Value	Description
31:2	ADD	R	0x0	In FMM, ADD[17:0] indicates the address in LMEM RXFQ1 where the last word of the last enqueued (write) RXQE is written by the MH. In CTM, ADD indicates the address in SMEM RXFQ1 where the last word of the last enqueued (write) RXQE is written by the CTDMA (performs SMEM accesses for this XS_CAN). This value is updated by the IP in FMM and CTM and is a word aligned address. The ADDR[1:0] bits are always 0. This value is for debug purposes.

Table 45. RXFQ1_WRAP_PTR register

RX FIFO Queue 1 Wrap Pointer in SMEM Register

Bit	Field	Mode	Initial Value	Description
31:2	SMEM_ADD	R	0x0	In FMM, the value is not defined. In CTM, SMEM_ADD indicates the address in SMEM at which RXFQ1 is wrapped around to store the next full RX message.This register is updated by the IP. This value is word aligned. The ADDR[1:0] bits are always 0

Table 46. RXFQ_STS register

RX FIFO Queue Status register

Bit	Field	Mode	Initial Value	Description
15:8	RXFQ1_FILL_LEVEL	R	0x0	FMM: Indicates the number of messages in RXFQ1 pending to be dequeued. Possible values are 0 to 255. CTM: Acts as a message count which gets reset at 255 and again starts to count up.
7:0	RXFQ0_FILL_LEVEL	R	0x0	FMM: Indicates the number of messages in RXFQ0 pending to be dequeued. Possible values are 0 to 255. CTM: Acts as a message count which gets reset at 255 and again starts to count up.

Table 47. DEBUG_TEST_CTRL register

Debug Control register

Bit	Field	Mode	Initial Value	Description
10:8	HDP_SEL	R/W	0x0	Defines the set of signals to be monitored on the HDP[15:0] bus signal interface.
0	HDP_EN	R/W	0x0	When set to 1, enable the hardware debug port to monitor MH internal signals.

Table 48. TX_SCAN_WC register

TX-SCAN winning candidates register
This register gives the information on the winning candidate evaluated by the TX-Scan.

Bit	Field	Mode	Initial Value	Description
31:16	FQ_PQ_ADD	R	0x0	LMEM address pointer to the winning candidate(TXFQ or TXPQ). This is for debug purposes.
6:2	PQSN	R	0x0	Slot number of the TXPQ if the winning candidate is from TXPQ. This is for debug purposes.
1	FQ_PQ	R	0x0	0: winning candidate is from TXFQ 1: winning candidate is from TXPQ
0	VALID	R	0x0	0: Contents of TX_SCAN_WC are not valid 1: Contents of TX_SCAN_WC are valid

Table 49. TX_SCAN_PC register

TX-SCAN prepared candidates register
This register gives the information on the prepared candidate which is in pipeline evaluated by the TX-Scan.

Bit	Field	Mode	Initial Value	Description
31:16	FQ_PQ_ADD	R	0x0	LMEM address pointer to the prepared candidate(TXFQ or TXPQ)
6:2	PQSN	R	0x0	Slot number of the TXPQ if the prepared candidate is from TXPQ. This is for debug purposes.
1	FQ_PQ	R	0x0	0: prepared candidate is from TXFQ 1: prepared candidate is from TXPQ
0	VALID	R	0x0	0: Contents of TX_SCAN_PC are not valid 1: Contents of TX_SCAN_PC are valid

Table 50. VBM_STATUS register (page 1 of 2)

Virtual Buffer Manager Status register

Bit	Field	Mode	Initial Value	Description
6	TXFQ_ENQ	R	0x0	1 indicates TXFQ enqueuing in progress
5	TXPQ_ENQ	R	0x0	1 indicates TXPQ enqueuing in progress
4	RXFQ0_DEQ	R	0x0	1 indicates RXFQ0 dequeuing in progress
3	RXFQ1_DEQ	R	0x0	1 indicates RXFQ1 dequeuing in progress

Table 50. VBM_STATUS register (page 2 of 2)

Virtual Buffer Manager Status register

Bit	Field	Mode	Initial Value	Description
2	TEFQ_DEQ	R	0x0	1 indicates TEFQ dequeuing in progress
1	CTB_ENQ	R	0x0	1 indicates CTB enqueuing in progress
0	CTB_DEQ	R	0x0	1 indicates CTB dequeuing in progress

1.5.6 Functional Description

The MH can manage up to one TX FIFO Queue, one TX Event FIFO Queue with up to 63 elements, up to 2 RX FIFO Queues and up to 32 TX priority queue slots. All accesses to LMEM are controlled by MH. The Virtual Buffer manager (VBM) inside MH handles enqueuing of TX messages into TXFQ and TXPQ slots and dequeuing of messages from RXFQs and TX Event FIFO Queue. TX Handler module inside MH handles the transmission of the enqueued messages to PRT. RX Handler module inside MH handles enqueuing of received messages from PRT into LMEM.
The LMEM controller module handles the arbitration and accesses to LMEM from all the other submodules in MH.

MH supports two modes of operation- Full Message Mode (FMM) and Cut Through Mode (CTM).

1.5.6.1 TX Message Header Definition

The TX messages stored into LMEM consists of header part and payload part. This section explains the format of TX message header and the way in which the header words must be stored into TXFQ or TXPQ.
The TX Message Header data structure depends on the CAN Frame Format (CAN CC, CAN FD, CAN XL) and the mode of operation (FMM or CTM).

1.5.6.1.1 TX Message Header for Full Message Mode (FMM)

The following tables describe the three data structures used for the headers in FMM.

Table 51. CAN CC TX Queue Element (TXFQ) Header

Tn	Bits	Name	Description/Constraints
T0	[31]	FDF	FD Format
	[30]	XLF	XL Format
	[29]	XTD	Extended Identifier
	[28:18]	BaseID [28:18]	Base ID
	[17:0]	ExtID [17:0]	Extended ID (applicable only for CEFF)
T1	[31]	Reserved	Not Applicable
	[30]	FIR	Fault Injection Request
	[29]	EFC (HOST)	Event FIFO Control: When set to 0, TX Event FIFO is not stored for the TX message. When set to 1, TX Event FIFO is stored for the TX message
	[28:27]	Reserved	Not Applicable
	[26]	RTR	Remote Transmission Request
	[25:20]	Reserved	Not Applicable
	[19:16]	DLC [3:0]	Data Length Code
	[15:0]	MM (HOST)	Message Marker (host)

Note: CAN CC frames (CBFF, CEFF) require **T0.FDF** = 0 and **T0.XLF** = 0. The header consists of T0 and T1.

Table 52. CAN FD TX Queue Element (TXFQ) Header

Tn	Bits	Name	Description/Constraints
T0	[31]	FDF	FD Format
	[30]	XLF	XL Format
	[29]	XTD	Extended Identifier
	[28:18]	BaseID [28:18]	Base ID
	[17:0]	ExtID [17:0]	Extended ID (applicable only for FEFF)
T1	[31]	Reserved	Not Applicable
	[30]	FIR	Fault Injection Request
	[29]	EFC (HOST)	Event FIFO Control: When set to 0, TX Event FIFO is not stored for the TX message. When set to 1, TX Event FIFO is stored for the TX message
	[28:27]	Reserved	Not Applicable
	[26]	0	Must be set to 0
	[25]	BRS	Bit Rate Switch
	[24:21]	Reserved	Not Applicable
	[20]	ESI	Error State Indicator
	[19:16]	DLC [3:0]	Data Length Code
	[15:0]	MM (HOST)	Message Marker (HOST)

Note: CAN FD frames (FBFF, FEFF) and CAN FD light commander frames require **T0.FDF** = 1 and **T0.XLF** = 0. The header consists of T0 and T1.

Table 53. CAN XL TX Queue Element (TXFQ) Header (FMM)

Tn	Bits	Name	Description/Constraints
T0	[31]	FDF	FD Format
	[30]	XLF	XL Format
	[29]	XTD	Extended Identifier
	[28:18]	Priority ID [28:18]	Priority ID (11 bit identifier for frames in CAN XL Frame Format(XLFF))
	[17]	RRS	Remote Request Substitution
	[16]	SEC	Simple Extended Content
	[15:8]	VCID [7:0]	Virtual CAN Network ID
	[7:0]	SDT [7:0]	SDU Type
T1	[31]	Reserved	Not Applicable
	[30]	FIR	Fault Injection Request
	[29]	EFC (HOST)	Event FIFO Control: When set to 0, TX Event FIFO is not stored for the TX message. When set to 1, TX Event FIFO is stored for the TX message
	[28:27]	Reserved	Not Applicable
	[26:16]	DLC-XL [10:0]	Data Length Code with CAN XL encoding
	[15:0]	MM (HOST)	Message Marker (HOST)
T2	[31:0]	AF [31:0]	Acceptance Field

Note: CAN XL frames (XLFF) require **T0.FDF** = 1, **T0.XLF** = 1 and **T0.XTD** = 0. The header consists of T0, T1 and T2.

There are two fields in the header which are not part of CAN protocol :
1) EFC bit: This bit must be decided by the host while storing the header. The TX event FIFO element will be created for a TX message only if this bit is set by the host.
2)MM(Message Marker) : This field is defined by the host while storing the header. MM is a 16bit ID for a message and the same MM value is copied into TX event FIFO Queue element to identify the status.

Note: The byte order of the Acceptance Field, delivered to the PRT, has to be reordered by the MH:

T2[7:0] => TX_MSG_DATA [31:24]
T2[15:8] => TX_MSG_DATA [23:16]
T2[23:16] => TX_MSG_DATA [15:8]
T2[31:24] => TX_MSG_DATA [7:0]

1.5.6.1.2 TX Message Header for Cut Through Mode (CTM)

The following tables describe the TX header formats in CTM.

Table 54. CAN CC TX Queue Element (TXFQ) Header

Tn	Bits	Name	Description/Constraints
T0	[31]	FDF	FD Format
	[30]	XLF	XL Format
	[29]	XTD	Extended Identifier
	[28:18]	BaseID [28:18]	Base ID
	[17:0]	ExtID [17:0]	Extended ID (applicable only for CEFF)
T1	[31]	Reserved	Not Applicable
	[30]	FIR	Fault Injection Request
	[29]	EFC (HOST)	Event FIFO Control: When set to 0, TX Event FIFO is not stored for the TX message. When set to 1, TX Event FIFO is stored for the TX message
	[28:27]	Reserved	Not Applicable
	[26]	RTR	Remote Transmission Request
	[25:20]	Reserved	Not Applicable
	[19:16]	DLC [3:0]	Data Length Code
	[15:0]	MM (HOST)	Message Marker (host)

Note: CAN CC frames (CBFF, CEFF) require **T0.FDF** = 0 and **T0.XLF** = 0. The header consists of T0 and T1.

Table 55. CAN FD TX Queue Element (TXFQ) Header

Tn	Bits	Name	Description/Constraints
T0	[31]	FDF	FD Format
	[30]	XLF	XL Format
	[29]	XTD	Extended Identifier
	[28:18]	BaseID [28:18]	Base ID
	[17:0]	ExtID [17:0]	Extended ID (applicable only for FEFF)
T1	[31]	Reserved	Not Applicable
	[30]	FIR	Fault Injection Request
	[29]	EFC (HOST)	Event FIFO Control: When set to 0, TX Event FIFO is not stored for the TX message. When set to 1, TX Event FIFO is stored for the TX message
	[28:27]	Reserved	Not Applicable
	[26]	0	Must be set to 0
	[25]	BRS	Bit Rate Switch
	[24:21]	Reserved	Not Applicable
	[20]	ESI	Error State Indicator
	[19:16]	DLC [3:0]	Data Length Code
	[15:0]	MM (HOST)	Message Marker (HOST)

Note: CAN FD frames (FBFF, FEFF) require **T0.FDF** = 1 and **T0.XLF** = 0. The header consists of T0 and T1.

Table 56. CAN XL TX Queue Element (TXFQ) Header (CTM)

Tn	Bits	Name	Description/Constraints
T0	[31]	FDF	FD Format
	[30]	XLF	XL Format
	[29]	XTD	Extended Identifier
	[28:18]	Priority ID [28:18]	Priority ID (11 bit identifier for frames in CAN XL Frame Format (XLFF))
	[17]	RRS	Remote Request Substitution
	[16]	SEC	Simple Extended Content
	[15:8]	VCID [7:0]	Virtual CAN Network ID
	[7:0]	SDT [7:0]	SDU Type
T1	[31]	Reserved	Not Applicable
	[30]	FIR	Fault Injection Request
	[29]	EFC (HOST)	Event FIFO Control: When set to 0, TX Event FIFO is not stored for the TX message. When set to 1, TX Event FIFO is stored for the TX message
	[28:27]	Reserved	Not Applicable
	[26:16]	DLC-XL [10:0]	Data Length Code with CAN XL encoding
	[15:0]	MM (HOST)	Message Marker (HOST)
T2	[31:0]	AF [31:0]	Acceptance Field
T3	[31:0]	TX_PAYLOAD_SMEM_PTR	TX payload SMEM pointer. This SMEM address points to the first payload word of the message.

Note: CAN XL frames (XLFF) require **T0.FDF** = 1, **T0.XLF** = 1 and **T0.XTD** = 0. The header consists of T0, T1,T2 and T3.

In CTM, CAN XL messages consist of header T3 which points to the location of the first word of the payload in SMEM.This address is used by TX Handler to fetch the payload to be stored into Cut Through Buffers if needed. The header T3 is also defined by host while storing the header and must be word aligned.i.e lower order two bits must be 0s.

Note: The byte order of the Acceptance Field, delivered to the PRT, has to be reordered by the MH:
T2[7:0] => TX_MSG_DATA [31:24]
T2[15:8] => TX_MSG_DATA [23:16]
T2[23:16] => TX_MSG_DATA [15:8]
T2[31:24] => TX_MSG_DATA [7:0]

The contents in word T3 are not sent to PRT for transmission. It is used only by Message Handler.

1.5.6.2 RX Message Header Definition

Messages received from the CAN Bus are stored in the LMEM, each consisting of a header followed by the payload. The header data structure depends on the CAN Frame Format (CAN CC, CAN FD, CAN, XL) used for this message on the CAN Bus. It can be identified by the header bits FDF and XLF. The following tables describe the three data structures used for the headers, consisting of the words R0, R1 and R2. The RX Message Header format is same in FMM and CTM.

Table 57. CAN CC RX Queue Element (RXQE) Header (page 1 of 2)

Rn	Bits	Name	Description/Constraints
R0	[31]	FDF	FD Format
	[30]	XLF	XL Format
	[29]	XTD	Extended Identifier
	[28:18]	BaseID [28:18]	Base ID (11 bit Identifier)
	[17:0]	ExtID [17:0]	Extended ID (29 bit Identifier) (applicable only for CEFF)

Table 57. CAN CC RX Queue Element (RXQE) Header (page 2 of 2)

Rn	Bits	Name	Description/Constraints
R1	[31:28]	Reserved	Not Used
	[27]	Reserved	Not Used
	[26]	RTR	Remote Transmission Request
	[25:20]	Reserved	Not Used
	[19:16]	DLC [3:0]	Data Length Code
	[15:12]	SN [3:0]	Sequence Number: order in which the message gets stored into the queue
	[11]	RX_IRQ	IRQ bit copied from the filter element which was matched for this message
	[10]	FAB	Filter Aborted: when set to 1, the RX filtering process was ended before completion without results
	[9]	ALERT	ALERT bit copied from the filter element which was matched for this message
	[8]	FM	Filter Match: When set to 1 one of the filter elements (defined by FIDX[7:0]) has detected a match
	[7]	Reserved	Not Used
	[6:0]	FIDX [6:0]	Filter Index: provide the information of the filter index which has been triggered

Note: CAN CC frames (CBFF, CEFF) can be identified by **R0.FDF** = 0 and **R0.XLF** = 0.

Table 58. CAN FD RX Queue Element (RXQE) Header

Rn	Bits	Name	Description/Constraints
R0	[31]	FDF	FD Format
	[30]	XLF	XL Format
	[29]	XTD	Extended Identifier
	[28:18]	BaseID [28:18]	Base ID (11 bit Identifier)
	[17:0]	ExtID [17:0]	Extended ID (29 bit Identifier) (applicable only for FEFF)
R1	[31:28]	Reserved	Not Used
	[27:26]	Reserved	Not Used
	[25]	BRS	Bit Rate Switch
	[24:21]	Reserved	Not Used
	[20]	ESI	Error State Indicator
	[19:16]	DLC [3:0]	Data Length Code
	[15:12]	SN [3:0]	Sequence Number: order in which the message gets stored into the queue
	[11]	RX_IRQ	IRQ bit copied from the filter element which was matched for this message
	[10]	FAB	Filter Aborted: when set to 1, the RX filtering process was ended before completion without results
	[9]	ALERT	ALERT bit copied from the filter element which was matched for this message
	[8]	FM	Filter Match: When set to 1 one of the filter elements (defined by FIDX[7:0]) has detected a match
	[7]	Reserved	Not Used
	[6:0]	FIDX [6:0]	Filter Index: provide the information of the filter index which has been triggered

Note: CAN FD frames (FBFF, FEFF) can be identified by **R0.FDF** = 1 and **R0.XLF** = 0.

Table 59. CAN XL RX Queue Element (RXQE) Header

Rn	Bits	Name	Description/Constraints
R0	[31]	FDF	FD Format
	[30]	XLF	XL Format
	[29]	XTD	Extended Identifier (is set to 0)
	[28:18]	BaseID [28:18]	Priority ID (11 bit Identifier for frames in CAN XL Frame Format (XLFF))
	[17]	RRS	Remote Request Substitution
	[16]	SEC	Simple Extended Content
	[15:8]	VCID [7:0]	Virtual CAN Network ID
	[7:0]	SDT [7:0]	SDU Type
R1	[31:28]	Reserved	Not Used
	[27]	Reserved	Not Used
	[26:16]	DLC-XL [10:0]	Data Length Code with CAN XL encoding
	[15:12]	SN [3:0]	Sequence Number: order in which the message gets stored into the queue
	[11]	RX_IRQ	IRQ bit copied from the filter element which was matched for this message
	[10]	FAB	Filter Aborted: when set to 1, the RX filtering process was ended before completion without results
	[9]	ALERT	ALERT bit copied from the filter element which was matched for this message
	[8]	FM	Filter Match: When set to 1 one of the filter elements (defined by FIDX[7:0]) has detected a match
	[7]	Reserved	Not Used
	[6:0]	FIDX [6:0]	Filter Index: provide the information of the filter index which has been triggered
R2	[31:0]	AF [31:0]	Acceptance Field

Note: CAN XL frames (XLFF) could be identified by **R0.FDF** = 1 and **R0.XLF** = 1.
Note: The byte order of the Acceptance Field, delivered from the PRT, has to be reordered by the MH:
R2[7:0] <= RX_MSG_DATA [31:24]
R2[15:8] <= RX_MSG_DATA [23:16]
R2[23:16] <= RX_MSG_DATA [15:8]
R2[31:24] <= RX_MSG_DATA [7:0]

1.5.6.3 TX Event Element Header Definition

The XS_CAN supports 1 TX Event FIFO Queue, named TEFQ. The TEFQ elements are called TEQE. It stores one TEQE to the TEFQ per TX message transmitted if EFC (event FIFO control bit) bit in the TXQE header is enabled. It functions the same for FMM and CTM.

The following table describes the header format for CC CAN and CAN FD in TEQE when **TEFQ_CFG.TEQE_LARGE** is enabled.

Table 60. CAN CC and CAN FD TX Event FIFO Queue Element (TEQE) Header (TEQE_LARGE = 1) (page 1 of 2)

En	Bits	Name	Description/Constraints
E0	[31]	FDF	FD Format
	[30]	XLF	XL Format
	[29]	XTD	Extended Identifier
	[28:18]	BaseID [28:18]	Base ID (11 bit Identifier)
	[17:0]	ExtID [17:0]	Extended ID (29 bit Identifier)

Table 60. CAN CC and CAN FD TX Event FIFO Queue Element (TEQE) Header (TEQE_LARGE = 1) (page 2 of 2)

En	Bits	Name	Description/Constraints
E1	[31]	SFPQ	Source EF/PQ. When 0, indicates FQ else PQ
	[30:29]	TXS [1:0]	Transmit Status:
			00->no information
			01->message sent successfully,
			10->message transmission attempt aborted after reaching the "retransmission count" limit (see ISO/DIS11898-2024),
			11->message skipped due to HFI (Header Format Invalid)
	[28]	TSPE	Time Stamp Parity Error
	[27]	Reserved	Not Applicable
	[26]	RTR or 0	RTR for CAN CC / 0 for CAN FD
	[25]	BRS or Reserved	BRS for CAN FD / Not Applicable for CAN CC
	[24:21]	Reserved	Not Applicable
	[20]	ESI or Reserved	ESI for CAN FD / Not Applicable for CAN CC
	[19:16]	DLC [3:0]	Data Length Code
	[15:0]	MM [15:0]	Message Marker
E2	[31:16]	TS [31:16]	Lower order Time Stamp
	[15:0]	TS [15:0]	Lower order Time Stamp
E3	[31:16]	TS [63:48]	Higher order Time Stamp
	[15:0]	TS [47:32]	Higher order Time Stamp
E4	[31:0]	Reserved	Not Applicable

The header format for CAN XL in TEQE when TEFQ_CFG.TEQE_LARGE is enabled is given in the below table.

Table 61. CAN XL TX Event FIFO Queue Element (TEQE) Header (TEQE_LARGE = 1) (page 1 of 2)

En	Bits	Name	Description/Constraints
E0	[31]	FDF	FD Format
	[30]	XLF	XL Format
	[29]	XTD	Extended Identifier
	[28:18]	Priority ID [28:18]	Priority ID (11 bit identifier for frames in CAN XL Frame Format(XLFF))
	[17]	RRS	Remote Request Substitution
	[16]	SEC	Simple Extended Content
	[15:8]	VCID [7:0]	Virtual CAN Network ID
	[7:0]	SDT [7:0]	SDU Type
E1	[31]	SFPQ	Source EF/PQ. When 0, indicates FQ else PQ
	[30:29]	TXS [1:0]	Transmit Status:
			00->no information
			01->message sent successfully,
			10->message transmission attempt aborted after reaching the "retransmission count" limit (see ISO/DIS11898-2024),
			11->message skipped due to HFI (Header Format Invalid)
	[28]	TSPE	Time Stamp Parity Error
	[27]	Reserved	Not Applicable
	[26:16]	DLC - XL [10:0]	Data Length Code with CAN XL encoding
	[15:0]	MM [15:0]	Message Marker
E2	[31:0]	AF [31:0]	Acceptance Field

Table 61. CAN XL TX Event FIFO Queue Element (TEQE) Header (TEQE_LARGE = 1) (page 2 of 2)

En	Bits	Name	Description/Constraints
E3	[31:16]	TS [31:16]	Lower order Time Stamp
	[15:0]	TS [15:0]	Lower order Time Stamp
E4	[31:16]	TS [63:48]	Higher order Time Stamp
	[15:0]	TS [47:32]	Higher order Time Stamp

For TEFQ_CFG.TEQE_LARGE when disabled, the header format for CC CAN and CAN FD is given as:

Table 62. CC CAN and CAN FD TX Event FIFO Queue Element (TEQE) Header (TEQE_LARGE = 0)

En	Bits	Name	Description/Constraints
E0	[31]	FDF	FD Format
	[30]	XLF	XL Format
	[29]	XTD	Extended Identifier
	[28:18]	BaseID [28:18]	Base ID (11 bit Identifier)
	[17:0]	ExtID [17:0]	Extended ID (29 bit Identifier)
E1	[31]	SFPQ	Source EF/PQ. When 0, indicates FQ else PQ
	[30:29]	TXS [1:0]	Transmit Status:
			00->no information
			01->message sent successfully,
			10->message transmission attempt aborted after reaching the "retransmission count" limit (see ISO/DIS11898-2024),
			11->message skipped due to HFI (Header Format Invalid)
	[28]	TSPE	Time Stamp Parity Error
	[27]	Reserved	Not Applicable
	[26]	RTR or 0	RTR for CAN CC / 0 for CAN FD
	[25]	BRS or Reserved	BRS for CAN FD / Not Applicable for CAN CC
	[24:21]	Reserved	Not Applicable
	[20]	ESI or Reserved	ESI for CAN FD / Not Applicable for CAN CC
	[19:16]	DLC [3:0]	Data Length Code
	[15:0]	MM [15:0]	Message Marker
E2	[31:16]	TS [31:16]	Lower order Time Stamp
	[15:0]	TS [15:0]	Lower order Time Stamp
E3	[31:16]	TS [63:48]	Higher order Time Stamp
	[15:0]	TS [47:32]	Higher order Time Stamp

In case of CAN XL, when TEFQ_CFG.TEQE_LARGE is disabled, the header format is shown in the following figure:

Table 63. CAN XL TX Event FIFO Queue Element (TEQE) Header (TEQE_LARGE = 0) (page 1 of 2)

En	Bits	Name	Description/Constraints
E0	[31]	FDF	FD Format
	[30]	XLF	XL Format
	[29]	XTD	Extended Identifier
	[28:18]	Priority ID [28:18]	Priority ID (11 bit identifier for frames in CAN XL Frame Format(XLFF))
	[17]	RRS	Remote Request Substitution
	[16]	SEC	Simple Extended Content
	[15:8]	VCID [7:0]	Virtual CAN Network ID
	[7:0]	SDT [7:0]	SDU Type

Table 63. CAN XL TX Event FIFO Queue Element (TEQE) Header (TEQE_LARGE = 0) (page 2 of 2)

En	Bits	Name	Description/Constraints
E1	[31]	SFPQ	Source EF/PQ. When 0, indicates FQ else PQ
	[30:29]	TXS [1:0]	Transmit Status:
			00->no information
			01->message sent successfully,
			10->message transmission attempt aborted after reaching the "retransmission count" limit (see ISO/DIS11898-2024),
			11->message skipped due to HFI (Header Format Invalid)
	[28]	TSPE	Time Stamp Parity Error
	[27]	Reserved	Not Applicable
	[26:16]	DLC - XL [10:0]	Data Length Code with CAN XL encoding
	[15:0]	MM [15:0]	Message Marker
E2	[31:16]	TS [31:16]	Lower order Time Stamp
	[15:0]	TS [15:0]	Lower order Time Stamp
E3	[31:16]	TS [63:48]	Higher order Time Stamp
	[15:0]	TS [47:32]	Higher order Time Stamp

1.5.6.4 LMEM Configuration

This diagram shows an example of how different queues can be configured in LMEM.

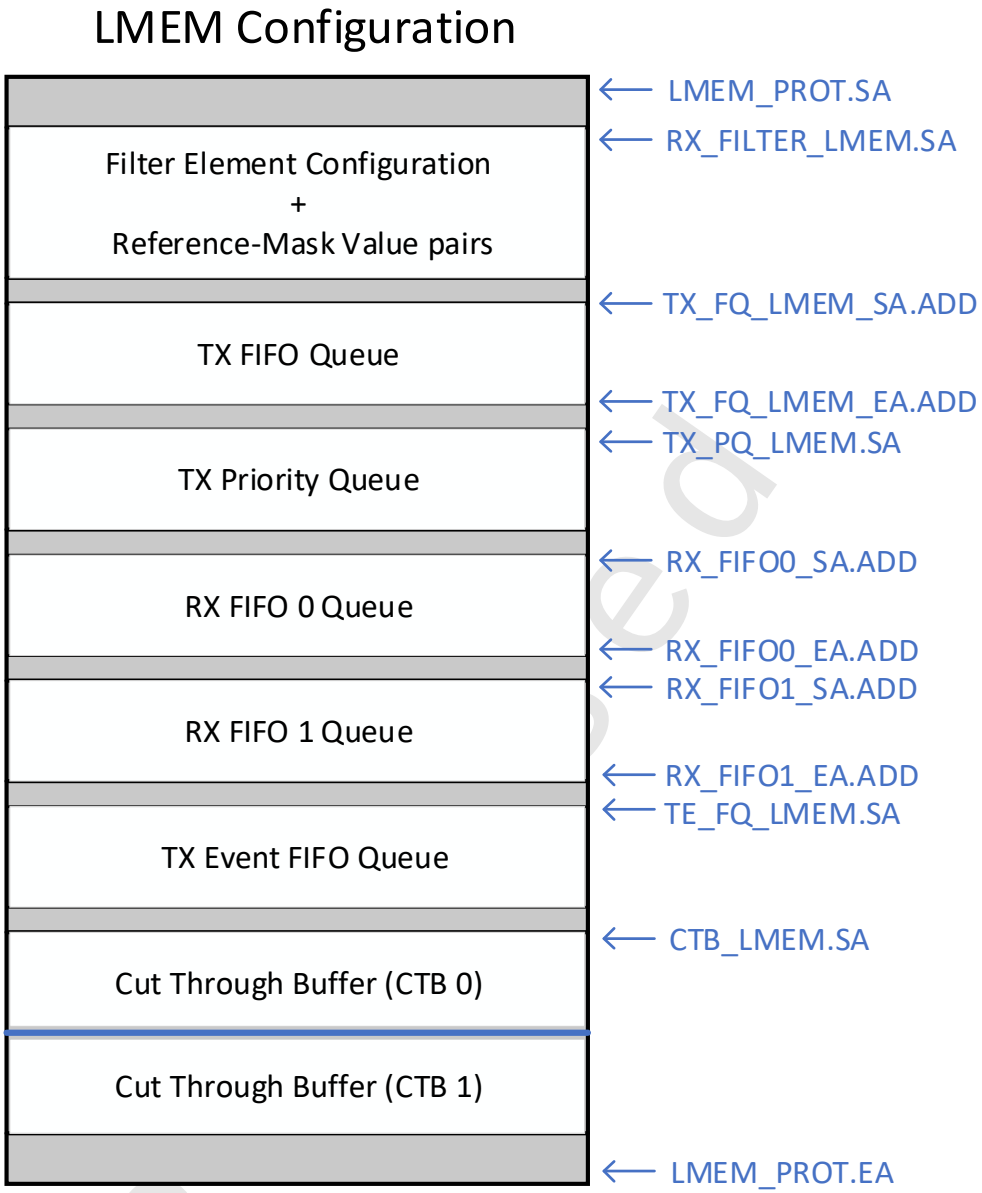


Figure 4. LMEM Configuration Diagram

1.5.6.4.1 RX Filter Elements

The RX filter elements consists of the Filter Element Configuration (FEC) and its one or two reference - mask pairs (REFP). The REFP has consists of 32 bit reference value and 32 bit mask value. Host must store the RX Filter elements via the transparent access address at AHB interface. Note that MH need not be enabled for transparent accesses.Refer section 1.4.1.2 XS_CAN Memory Map for the address range. Virtual Buffers are not present for RX Filter element storage.A more detailed section is described in RX Filter section (1.5.6.7.4).

1.5.6.4.2 TXFQ - TX FIFO Queue

The XS_CAN supports 1 TX FIFO Queue (TXFQ) defined in LMEM. A TX FIFO Queue holds TX messages to be sent in order to the PRT. Each message inside TXFQ is called a TX Queue Element (TXQE). Each TXQE consists of header and the payload.

The IP does not support overwriting of messages in the TXFQ. The IP provides access to the TXFQ via the AHB slave interface through virtual address in VBM. While a TXQE is getting enqueued into TXFQ, the message can wrap around without overwriting pending messages.

The TXFQ is enabled when **MH_CFG.TXFQE** is set to 1. The XS_CAN provides a register **TXFQ_LMEM_SA.ADD** to configure the start address and **TXFQ_LMEM_EA.ADD** to configure the end address of TXFQ in LMEM.

The address values are always 64 bytes aligned.
Example: For a TXFQ size of 64 bytes, if TXFQ_LMEM_SA.ADD = 0x0000, then **TXFQ_LMEM_EA.ADD** shall be programmed as 0x0040 (64byte aligned boundary) and not 0x003C. The next queue, if configured, shall start from the next 64-byte aligned boundary which is 0x0080. The space from 0x0044 to 0x007C will remain unused in this example.

TXFQ_CFG.MAX_ELEM_SIZE[5:0] is the TX FIFO Configuration register which defines the maximum size of 1 TXQE which includes the header and payload. The total size of TXFQ should be greater than or equal to **TXFQ_CFG.MAX_ELEM_SIZE** else it is not considered as a valid configuration. Also XS_CAN IP does not generate TXFQ_WREQ if MAX_ELEM_SIZE=0.

The status registers for TXFQ are given as:
TXFQ_STS.FIFO_FULL:- when set to 1, indicates TXFQ is full. Empty condition can be detected by reading the fill level register TXFQ_STS.FILL_LEVEL
TXFQ_STS.FILL_LEVEL:- It is a read-only register which stores the fill level of TXFQ. The maximum fill level of the TXFQ is 255. It indicates the total number of messages in the TXFQ pending for transmission. Once a message is successfully enqueued into FIFO, **TXFQ_STS.FILL_LEVEL** is incremented by 1 and decremented by 1 when a message is successfully dequeued by transmission on the bus or removed from the queue due to HFI or maximum retransmission count reached. If the maximum fill level is attained, TXQE is no longer enqueued, even when there is empty memory space left. Status register fields are updated one clock after the interrupt TX functional interrupt TXFQ_ENQ gets generated from MH.

With every Enqueuing process, the TXFQ write pointer **TXFQ_WPTR.WPTR_ADD** is updated and with every dequeuing process, the TXFQ read pointer **TXFQ_RPTR.RPTR_ADD** is updated.
The TX FIFO Queue Write Pointer register **TXFQ_WPTR.WPTR_ADD** points to the location where the last word of TXQE is enqueued and the next TXQE starts at address wptr+4. This address is updated by VBM in MH.
The TX FIFO Queue Read Pointer register **TXFQ_RPTR.RPTR_ADD** points to the location corresponding to the last word of TXQE is read by TX Handler for transmission to PRT. This address is updated by TX Handler in MH.

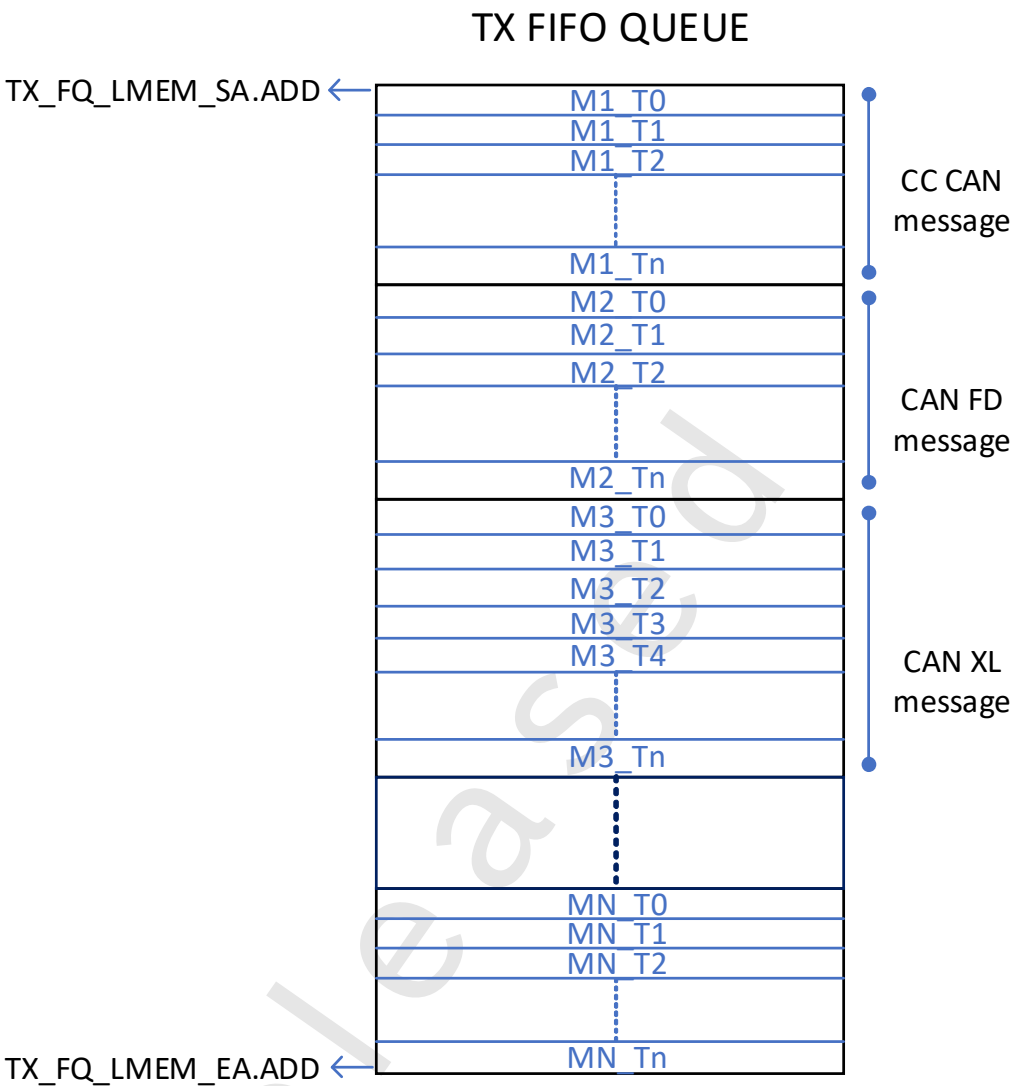


Figure 5. TX FIFO Queue Configuration Example

1.5.6.4.2.1 FMM - Full Message Mode

The XS_CAN IP stores the full message into LMEM in FMM. The TXQEs are stored back to back in the TXFQ to use LMEM efficiently.
There are 2 words of header for CAN CC and CAN FD whereas 3 words of header for CAN XL.

The diagram given above shows an example representation of how messages can be enqueued into TXFQ.
M1 (CAN CC), M2 (CAN FD) and M3 (CAN XL) are TX messages 1, 2 and 3 respectively.
For FMM, M1_T0 and M1_T1 are headers and M1_T2 onwards is payload in case of CAN CC and M2_T0 and M2_T1 are headers and M2_T2 onwards is payload in case of CAN FD. In case of CAN XL, M3_T0, M3_T1 and M3_T2 are all headers whereas M3_T3 onwards it is payload.
Refer section 1.5.6.1 for TX header format description.

1.5.6.4.2.2 CTM - Cut Through Mode

The XS_CAN IP stores full CAN CC and CAN FD messages as TXQE continuously in TXFQ in CTM including the header and complete payload. In case of CAN XL messages, header (16 bytes)and payload up to 64 bytes is stored into TXFQ. Remaining payload, beyond 64 bytes will be buffered into 2 cut through buffers of 64 bytes each.

In CTM, M1_T0 and M1_T1 are headers and M1_T2 is payload in case of CAN CC and M2_T0 and M2_T1 are headers and M2_T2 is payload in case of CAN FD. Whereas, in case of CAN XL, M3_T0, M3_T1 and M3_T2 are all headers, M3_T3 is SMEM pointer (word aligned address i.e. lower order 2 bits must be 0) and M3_T4 onwards it is payload. SMEM

pointer depicts the location of first payload word of the TX message (M3) in System Memory (SMEM) and should be word aligned value. Refer section 1.5.6.1 for TX header format description.

1.5.6.4.3 TXPQ - TX Priority Queue

The XS_CAN IP supports 1 TX Priority Queue (TXPQ). It supports up to 32 priority queue slots and the number of slots are user configurable.
The user does not have the flexibility to choose the slot to store the message which is to be enqueued. The VBM in MH will direct the Priority Queue message in ascending order of slot index.

The IP provides access to the TXPQ via the AHB slave interface via VBM virtual address mapping.
Each message inside TXPQ is called a TX Queue Element (TXQE). Each TXQE consists of header and the payload. The maximum size of the message stored depends on the size of PQ slot configured. If the host writes a TXQE larger than TX slot size, the messages are discarded.

The TXPQ registers are configured before MH is started and cannot be modified in between operation.
The register **MH_CFG.TXPQE** when set to 1 enables the TXPQ. The IP provides a register **TXPQ_LMEM.SA** to configure the TXPQ start address in the LMEM. The address value is always 64 bytes aligned.

The TXPQ Configuration register **TXPQ_CFG.SLOT_SIZE** defines the size of each priority queue slots in LMEM and the size is mentioned in granularity of 4 bytes for efficient use of LMEM. All slots have the same size.
TXPQ_CFG.SLOT_NUM is a TXPQ configuration register which is responsible to configure the number of slots in TXPQ, user can store up to 32 slots where each slot stores one TXQE.

The start address of Slot n where n is in {1 to 32} in TXPQ is calculated as:
(TXPQ_LMEM.SA)+ (n* TXPQ_CFG.SLOT_SIZE).
End address of each slot (n) is Start address of slot (n+1)-4

Static registers for TXPQ are given as:
TXPQ_STS0.SLOT_OCC: When SLOT_OCC[n] = 1, the TXPQ slot n is busy which means that the TX message in slot n is being loaded in LMEM and considered by TX-SCAN. The message attached to the slot n has not been sent yet as long as this bit remains high.
TXPQ_STS1.TXPQ_FULL: When set to 1, indicates all slots are occupied and cannot store any new message. Empty condition of TXPQ can be detected by reading the fill level register **TXPQ_STS1.FILL_LEVEL**.
TXPQ_STS1.FILL_LEVEL: It defines the total number of messages in the TXPQ pending for transmission. The maximum allowed fill level is same as the number of slots configured by the user. Once a message is successfully enqueued, **TXPQ_STS1.FILL_LEVEL** is incremented by 1 and decremented by 1 when a message is successfully dequeued. Once maximum fill level is attained, no more elements are enqueued.Status register fields are updated one clock after the interrupt TX functional interrupt TXPQ_ENQ gets generated from MH

TXPQ_WPTR.WPTR_ADD is TXPQ Write Pointer register which indicates the LMEM address corresponding to the last word of the previously enqueued TXQE.

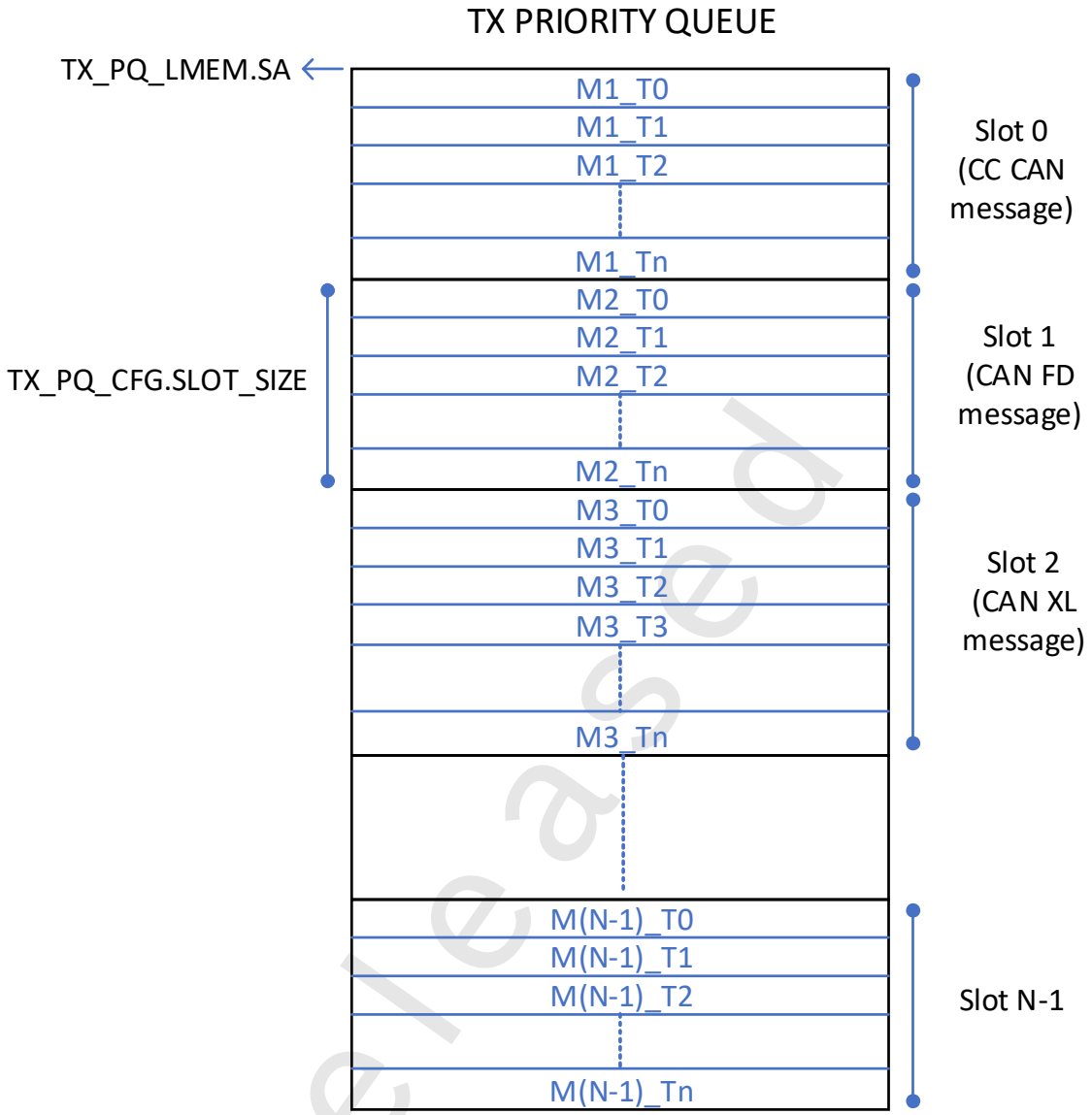


Figure 6. TX Priority Queue Configuration

1.5.6.4.3.1 FMM - Full Message Mode

The XS_CAN IP stores the full message into LMEM as TXQE though the maximum size of the message stored depends on the size of PQ slot configured.
There are 2 words of header for CAN CC and CAN FD and 3 words of header for CAN XL.
TXPQ does not store TXQEs back to back like in TXFQ. If the message size is less than the slot size, the remaining free space is not used and next message is stored from beginning of the next slot.

The figure mentioned above shows an example representation of how messages are enqueued into slots in TXPQ.
M1 (CAN CC), M2(CAN FD), M3 (CAN XL) and M(N-1) are TX messages of slots 1, 2, 3 and N-1 respectively where N is the number of slots in TXPQ (TXPQ_CFG.SLOT_NUM).
For FMM, in slot 0, M1_T0, M1_T1 are headers and M1_T2 onwards is payload in case of CAN CC and M2_T0, M2_T1 are headers and M2_T2 onwards is payload for slot 1 in case of CAN FD. In case of CAN XL in slot 2, M3_T0, M3_T1 and M3_T2 are all headers whereas M3_T3 onwards it is payload.
Refer section 1.5.6.1 for TX header format description.

1.5.6.4.3.2 CTM - Cut Through Mode

In CTM, host can store full CAN CC and CAN FD messages into TXPQ in LMEM including the header and payload. In case of CAN XL frames, 4 words of header and upto 64bytes of payload can be stored into one TXPQ slot. if the size of the message is greater than 64 bytes, then the remaining payload, from 65th byte has to be buffered into two cut through

buffers. The cut through buffers are 64 bytes in size each. The transfer of the payload beyond 64bytes is always done in 64byte blocks and this is called a block copy transfer. More details are given in TX Handler chapter.

In CTM, as shown in slot 0, M1_T0 and M1_T1 are headers and M1_T2 is payload in case of CAN CC and M2_T0, M2_T1 are headers and M2_T2 is payload in case of CAN FD for slot 1. Whereas in slot 2, in case of CAN XL, M3_T0, M3_T1 and M3_T2 are all headers, M3_T3 is SMEM pointer (word aligned i.e. last two bits must be 0) and M3_T4 onwards it is payload. SMEM pointer depicts the start location of the payload of the TX message (M3) in System Memory (SMEM). Refer section 1.5.6.1 for TX header format description.

1.5.6.4.4 RXFQ - RX FIFO Queue

The XS_CAN IP supports 2 RX FIFO Queues, namely RXFQ0 and RXFQ1. RXFQ contains the RX messages which are received after RX filtering. For FMM, it is defined in LMEM whereas it is defined in SMEM in case of CTM.

The received messages are stored back-to-back into RXFQ as RX Queue Elements (RXQE). A RXQE consists of Rx message header, payload data and timestamp. The RX message header contains 4 bit message sequence number which is incremented every time a new message is enqueued and transferred to LMEM. This is done to check the order of the messages as a safety mechanism. The last two words enqueued into the RXFQs are always the timestamp values.

The RXFQ0 is enabled when **MH_CFG.RXFQ0E** is set to 1 and RXFQ1 is enabled when **MH_CFG.RXFQ1E** is set to 1. The XS_CAN IP provides a register **RXFQ0_SA.ADD** for RXFQ0 to configure the start address of RXFQ0 and **RXFQ1_SA.ADD** for RXFQ1 to configure the start address of RXFQ1. RXFQ End Address register **RXFQ0_EA.ADD** (RXFQ0) and **RXFQ1_EA.ADD** (RXFQ1) stores the end address of RXFQ0 and RXFQ1 respectively.The addresses are always 64 byte aligned.
Example: For getting 64byte sized RXFQ0/1, if the **RXFQ0/1_SA.ADD**=0x0100, the **RXFQ0/1_EA.ADD** must be 0x0140 and not 0x013C. The next queue, if configured, shall start from the next 64-byte aligned boundary which is 0x0180. The space from 0x0144 to 0x017C will remain unused in this example.

RXFQ_STS.RXFQ0_FILL_LEVEL and **RXFQ_STS.RXFQ1_FILL_LEVEL** are RXFQ Status registers which indicates the number of messages in RXFQ0 and RXFQ1 pending to be dequeued respectively. The maximum fill level is 255 elements. It gets incremented every time a new message is successfully enqueued into LMEM and is decremented when its dequeued from LMEM. Once the maximum level is attained, no more messages are enqueued even when there is space remaining in FIFO. Status register fields are updated one clock after the interrupt RX functional interrupt RXFQx_DEQ/RXFQx_ENQ gets generated from MH.

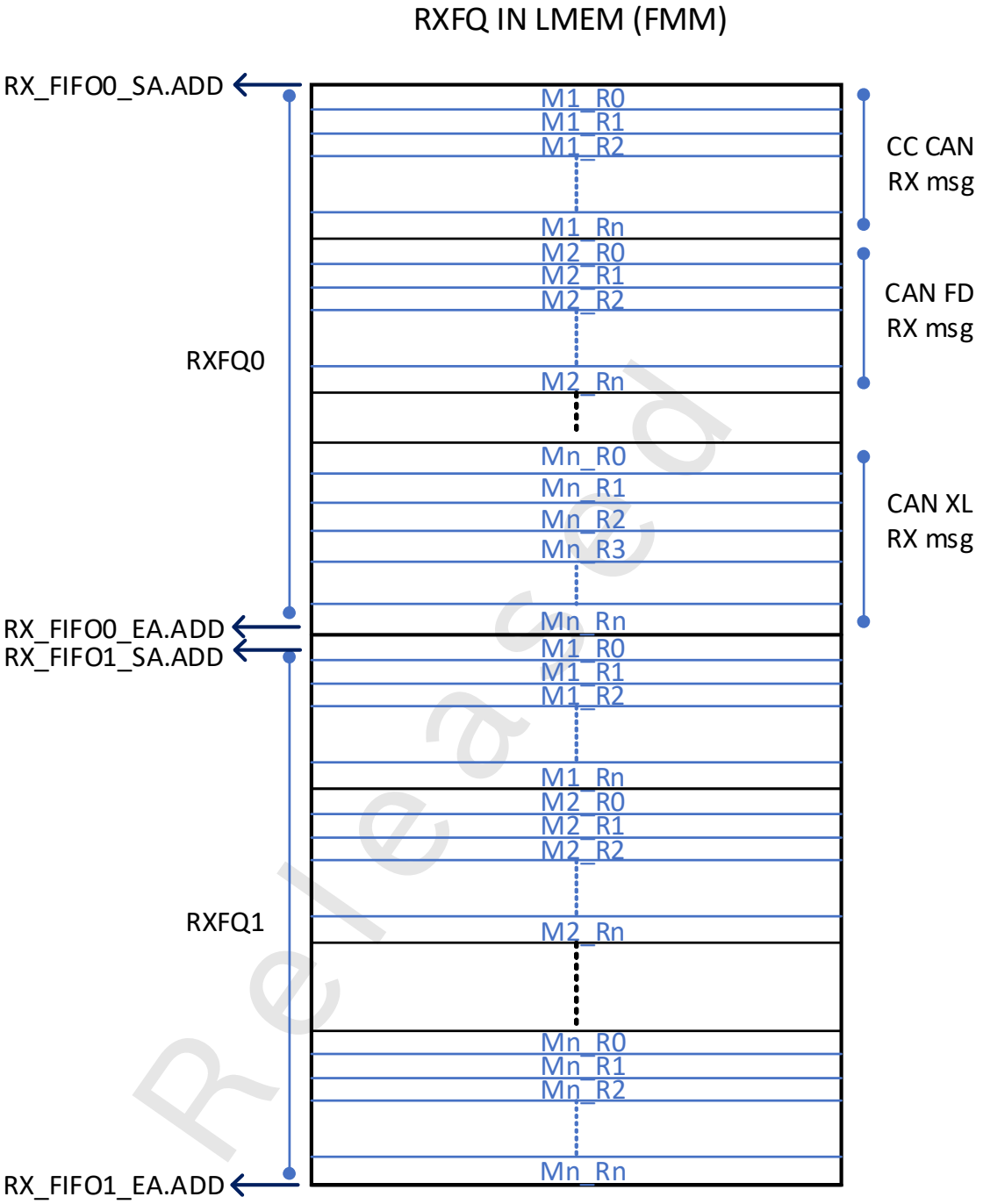


Figure 7. RXFQ Configuration in FMM

1.5.6.4.4.1 FMM - Full Message Mode

In FMM, the two RXFQs are defined in LMEM. There are 2 words of header for CAN CC and CAN FD and 3 words of header for CAN XL.

The RXFQ0 and RXFQ1 start address registers **RXFQ0_SA.ADD** and **RXFQ1_SA.ADD** and end address registers **RXFQ0_EA.ADD** and **RXFQ1_EA.ADD** stores the start and end addresses in LMEM in case of FMM.

The IP stores RX messages back-to-back into each of the RXFQ and supports wrapping around of messages in LMEM.

The diagram given above shows an example representation of how messages can be enqueued into RXFQ in LMEM in case of FMM. More details are given in section 1.5.6.2 for RX Header format description.

1.5.6.4.4.2 CTM - Cut Through Mode

In CTM, the two RXFQs are defined in SMEM and not in LMEM. The start and end address registers stores the addresses of both RXFQs in SMEM. The address are 64bytes aligned.

The received messages are directly buffered in 2 Cut Through Buffers (CTB) of 64 bytes size each and transferred to the RXFQs space in SMEM. More details are given in section 1.5.6.2 for RX Header format description.

1.5.6.4.5 TEFQ - TX Event FIFO Queue

The XS_CAN IP supports 1 TX Event FIFO Queue, named TEFQ. The TEFQ elements are called TEQE. It stores one TEQE to the TEFQ per TX message transmitted if EFC (event FIFO control bit) bit in the TXQE header is enabled. It functions the same for FMM and CTM.

The TEFQ is enabled when **MH_CFG.TEFQE** is set to 1. The IP provides a register **TEFQ_LMEM.SA** to configure the TEFQ start address where the TEQE are defined in LMEM. The address is always 64 bytes aligned.

The TEFQ LMEM Configuration register **TEFQ_CFG.TEQE_LARGE** decides the size of TEQE. When it is set to 1, the TEQE is of 5 words else it is of 4 words for all frame formats.
TEFQ_CFG.TEQE_NUM is also a TEFQ LMEM Configuration register which defines the maximum number (63 elements) of TEQE that can be stored in LMEM. The end address of TEFQ must not extend beyond the LMEMPC_END ADDR. For invalid FIFO configurations, IP behavior is not defined.

The TEFQ Status registers are as follows:
TEFQ_STS.TEFQ_FULL: When set to 1, the TEFQ has reached maximum number of elements.
TEFQ_STS.FILL_LEVEL: It defines the number of TEQE enqueued successfully in TEFQ.
Status register fields are updated one clock after the occurrence of the corresponding event in MH.

The TEFQ Write Pointer register **TEFQ_WPTR.WPTR_ADD** is updated with every enqueuing process and with every dequeuing process, the read pointer **TEFQ_RPTR.RPTR_ADD** is updated.
TEFQ_WPTR.WPTR_ADD points to the location in LMEM corresponding to the last word of previously enqueued TEQE whereas TEFQ_RPTR.RPTR_ADD indicates the location in LMEM corresponding to the last word of previously dequeued TEQE.

In the figure mentioned below for TEFQ Configuration in LMEM, it is segregated on the basis of the TEFQ LMEM Configuration register **TEFQ_CFG.TEQE_LARGE**. When it is set to 1, there are 5 words in TEFQ [E0, E1, E2, E3, E4] wherein in case of CC CAN and CAN FD, E2 has header (T2) whereas in case of CAN XL, E2 has Acceptance Field (AF). And when **TEFQ_CFG.TEQE_LARGE** is disabled, E4 word is reserved. Refer section 1.5.6.3 for TX Event Header format description.

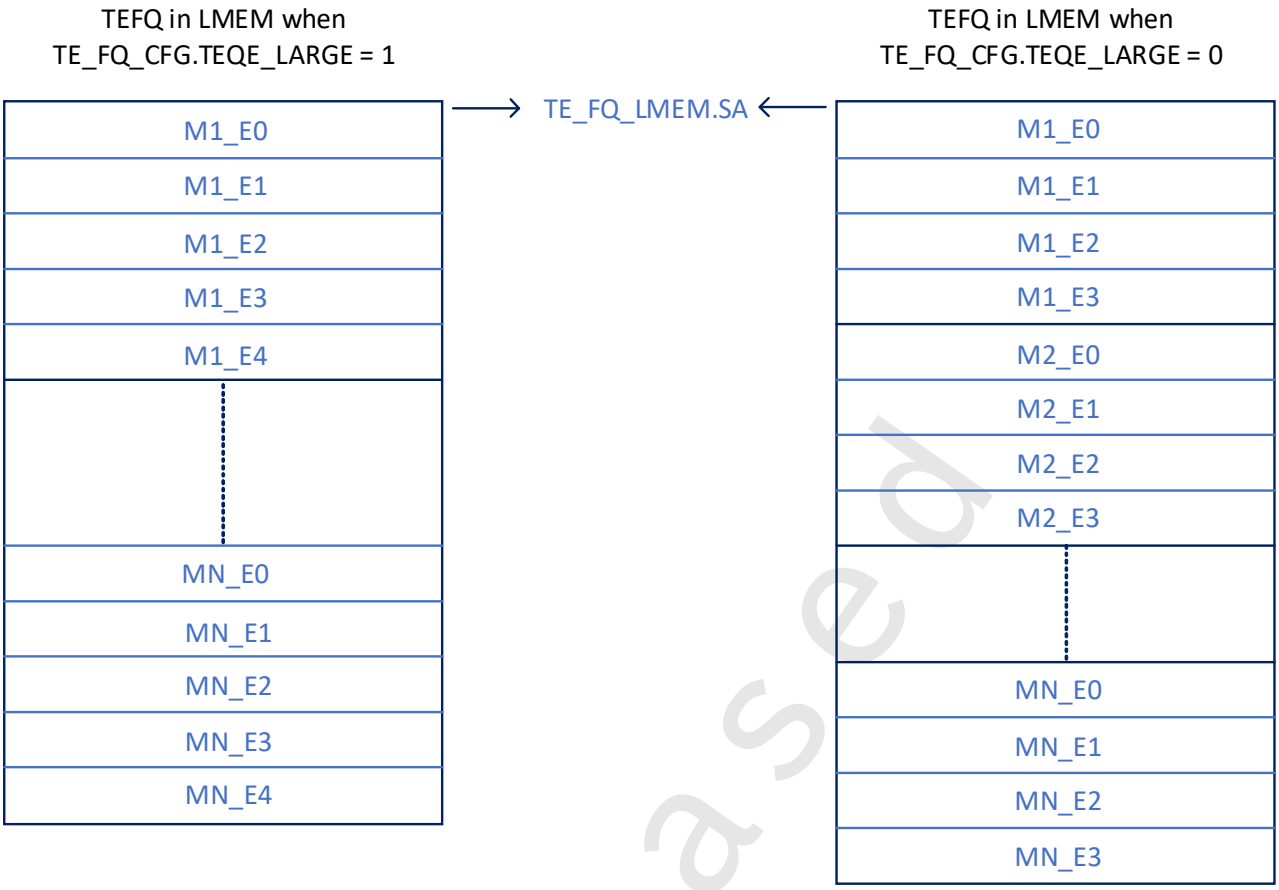


Figure 8. TX Event FIFO Configuration in LMEM

1.5.6.4.6 CTB - Cut Through Buffers

The XS_CAN IP supports 2 Cut Through Buffers of 64 bytes each in LMEM. It provides access to CTB in CTM only. The IP provides a register **CTB_LMEM.SA** to configure the start address of the first CTB in LMEM. The second CTB is in continuation to the first CTB. The address is always 64 bytes aligned. Access to CTB in FMM is not allowed.

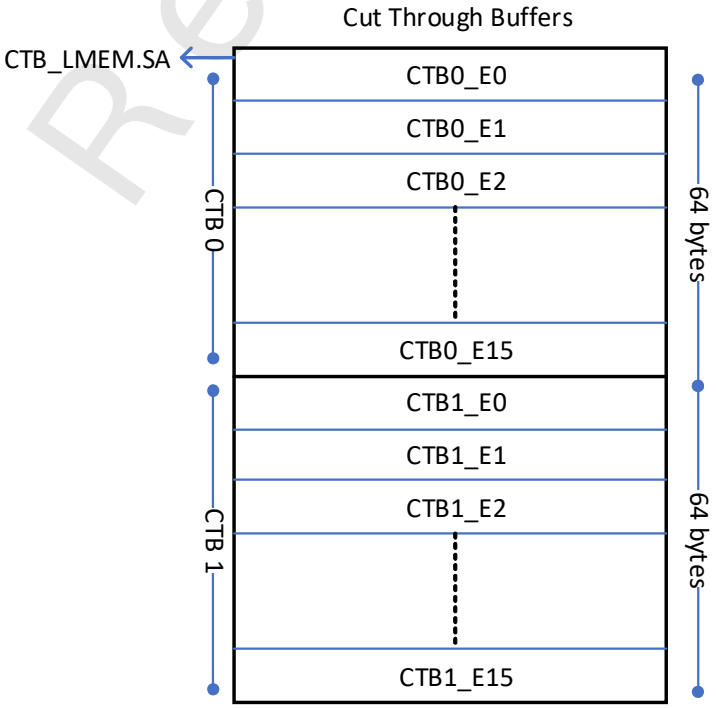


Figure 9. CTB Configuration

1.5.6.5 VBM - Virtual Buffer Manager

The virtual buffer manager (VBM) module is a part of the message handler which supports the enqueuing and dequeuing of CAN messages and RX Filter elements in LMEM. Host accesses the LMEM via the AHB slave interface of VBM. For each type of queue, a virtual buffer is defined.
The following block diagram shows the structure of Virtual Buffer Manager (VBM).

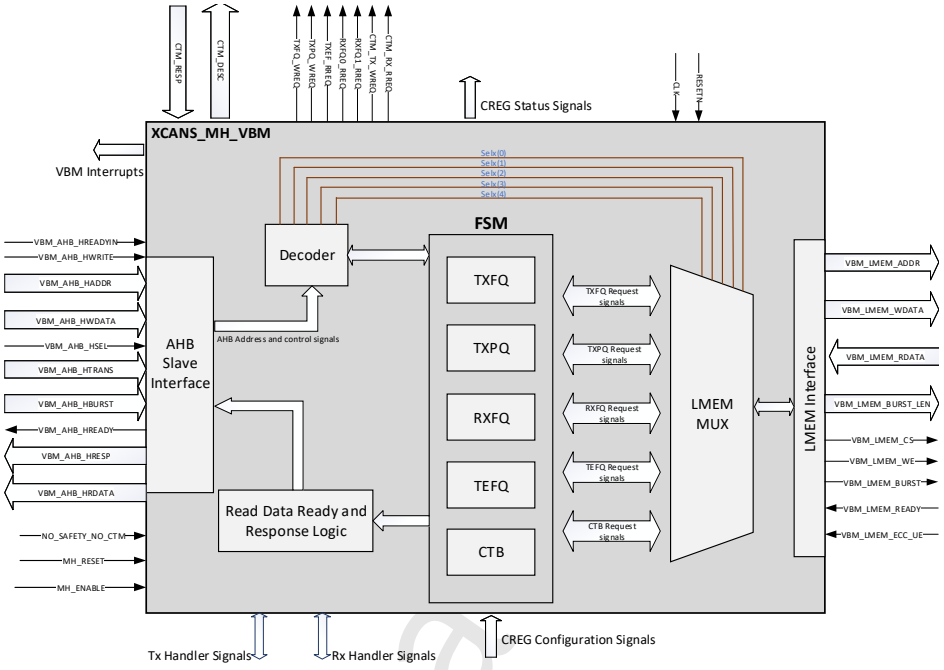


Figure 10. VBM Block Diagram

1.5.6.5.1 Virtual Buffers (FMM and CTM)

To ease the LMEM accesses for the host, virtual buffers (VB) are defined for TXFQ, TXPQ, RXFQ0, RXFQ1, TEFQ and CTB. Virtual buffer address is a fixed address range equivalent to the size of one largest message that can be stored into the queue in FMM and CTM. All virtual addresses are 64byte aligned.There is a start address, trigger address and end address corresponding to each virtual buffer. The virtual buffer address map is defined in the table 64 for FMM and table 65 for CTM. VBM accepts the transactions to virtual buffers when the **MH_CTRL.ENABLE** bit is set. However, the transparent access to LMEM is possible without setting the **MH_CTRL.ENABLE** bit. The configuration of **MH_CTRL.ENABLE** bit and the dependency to the input signal *PRT_ENABLE* from PRT module is explained in section 1.5.6.10 PRT ENABLE and MH_ENABLE dependency. Virtual buffers are mainly for LMEM accesses and not for the register accesses of MH, PRT and IRC. All registers are accessed via APB slave interface of MH.

Table 64. VBM VIRTUAL ADDRESS MAP (FMM)

Start location Address	End location Address	Symbol	Name
0x01000	0x01808	TXFQ	TX FIFO Queue
0x0180C	0x01FFC	reserved	reserved
0x02000	0x02808	TXPQ	TX Priority Queue
0x0280C	0x02FFC	reserved	reserved
0x03000	0x03810	RXFQ0	RX FIFO 0 Queue
0x03814	0x03FFC	reserved	reserved
0x04000	0x04810	RXFQ1	RX FIFO 1 Queue
0x04814	0x04FFC	reserved	reserved
0x05000	0x05010	TEFQ	TX Event FIFO Queue
0x05014	0x3FFFC	reserved	reserved
0x40000	0x7FFFC	LMEM TA	LMEM Transparent Access

Table 65. VBM VIRTUAL ADDRESS MAP (CTM)

Start Location Address	End Location Address	Symbol	Name
0x01000	0x01808	TXFQ	TX FIFO Queue
0x0180C	0x01FFC	reserved	reserved
0x02000	0x02808	TXPQ	TX Priority Queue
0x0280C	0x02FFC	reserved	reserved
0x03000	0x03810	RXFQ0	RX FIFO 0 Queue (reserved in CTM)
0x03814	0x03FFC	reserved	reserved
0x04000	0x04810	RXFQ1	RX FIFO 1 Queue (reserved in CTM)
0x04814	0x04FFC	reserved	reserved
0x05000	0x05010	TEFQ	TX Event FIFO Queue
0x05014	0x05FFC	reserved	reserved
0x06000	0x0603C	CTB	Cut Through Buffer
0x06040	0x3FFFC	reserved	reserved
0x40000	0x7FFFC	LMEM TA	LMEM Transparent Access

1.5.6.5.1.1 Accessing Order

The host issues addresses in linear incrementing order (4 byte alignment) starting from the start address till the trigger address. Any other order of addresses issued is considered as a nonlinear access and the behavior is explained in Error and Exception Handling section.
The following diagram shows the valid, acceptable, and reserved address space of a virtual buffer.

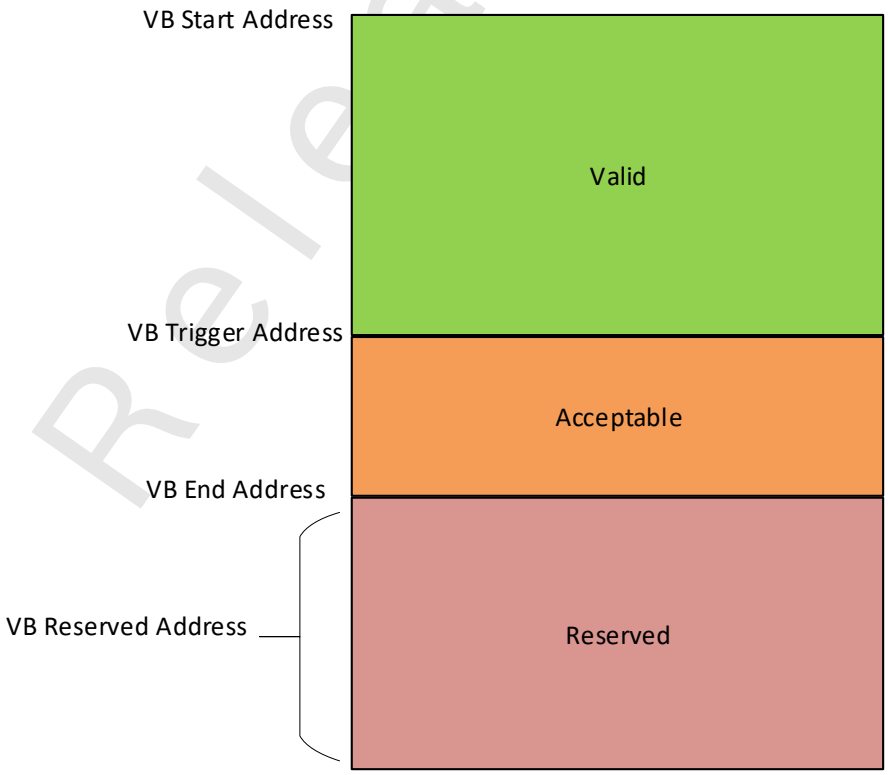


Figure 11. Address Space of a Virtual Buffer (VB)

To start the enqueue and dequeue of a message, the host issues the start address of the VB. Trigger address corresponds to the last word of the message. Calculation of the trigger address is explained in later sections. Trigger address and end address can be same if the message has the maximum payload possible which is a CAN XL with 2KB payload.

While enqueuing, if the DLC is such that the last beat of the burst transfer consists of invalid data byte along with valid byte of data, the full 4byte data is stored into LMEM by VBM. MH sends this full word to PRT and PRT discards the invalid data during transmission.

Note: If the host issues a burst transfer that extends the addresses into reserved range of Virtual Addresses (Refer VBM Virtual Address Map), the read transfers are discarded with OK response, a predefined read data = 0xCAFECAFE is given back to the host. Writes are ignored. Refer the section Error and Exception Handling to know more about the error scenarios and the corresponding behavior of VBM.

1.5.6.5.1.2 TX FIFO Queue Virtual Buffer (0x01000 - 0x01808)

This virtual buffer points to the TXFQ tail element in LMEM.i.e, the location where the next message should be stored into TXFQ. The start address of TXFQ virtual buffer is 0x01000. In FMM, end address is 0x01808 which corresponds to the maximum possible CAN message size i.e CAN XL message with 2KB payload and 3 words header. In CTM, the maximum size of a TXQE that can be stored in TXFQ is header plus 64 bytes payload. So the end address for TXFQ in CTM is 0x0104C. This means entire CAN CC and CAN FD messages can be stored in TXFQ. But in case of XL messages, header plus 64bytes payload of the message can be stored in TXFQ. Remaining payload must be buffered into 2 cut through buffers before sending to PRT.

For starting the enqueuing of a new message into TXFQ, the first word T0 of the message should be written to the start address 0x01000. This triggers the VBM to start the enqueue process into TXFQ. TXFQ handling part of the VBM converts the virtual buffer address to the actual address in LMEM where the next message should be enqueued. When a message gets enqueued into the queue for the first time, the actual address corresponding to the VB start address (0x01000) is **TXFQ_LMEM.SA**. Next location will be **TXFQ_LMEM.SA**+4 and so on. Only write accesses are allowed to TXFQ virtual buffer.

Consecutive words of the message is written into 0x1004, 0x1008 and so on in a linear order. VBM calculates the trigger address based on the type of frame (FDF and XLF bits in T0), RTR and DLC bits in T1. Trigger address corresponds to the last word of the message being enqueued.

For example, a CAN FD frame with DLC=4, trigger address will be for word T2(0x1008). A CAN XL frame with DLC=7, trigger address will be for wordT4(0x1010).

After the trigger address is accessed, the next expected address is the start address. In case if this is violated, status flags are updated based on the type of violation. Refer the registers MH_STS0,MH_STS1 or INTERRUPTS chapter for details on this.

In FMM, for a CAN XL with 2 KB payload, trigger address for TXFQ = 0x01808 (3 words header + 2 KB payload), remaining payload.
In CTM, for a CAN XL with 2 KB payload, trigger address for TXFQ = 0x0104C (4 words header + 64 bytes payload), remaining payload to be stored in CTB.

After enqueuing of a message into TXFQ, the following registers are updated by VBM.

1) Write pointer register **TXFQ_WPTR.WPTR_ADD** with the actual LMEM address corresponding to the last word of TXQE that is enqueued.
2) The fill level of the TXFQ is incremented by one and is updated into the register **TXFQ_STS.FILL_LEVEL**.

VBM also keeps track of the amount of space in TXFQ for storing the incoming messages.
In FMM, if the size of the transmitted message (header + payload) is greater than the value configured in the register **TXFQ_MAX_ELEM_SIZE.SIZE**, then the message will be dropped and interrupt is raised for the transaction. Next address expected will be again the start address of the virtual buffer. Also, in case if the remaining space in TXFQ is less than **TXFQ_MAX_ELEM_SIZE.SIZE**, VBM will raise the TXFQ_FULL status in **TXFQ_STS.FIFO_FULL** register field and virtual buffer access is disabled. Note that wrap around of messages are possible in TXFQ enqueue. If there is not enough space in the bottom of the queue equal to TXFQ_MAX_ELEM_SIZE.SIZE and If there is free space in the beginning of the queue there by making the total space in the TXFQ greater than or equal to TXFQ_MAX_ELEM_SIZE.SIZE then the message is stored with wrapping around. In CTM, the max element size check is performed only if the value configured in the **register TXFQ_MAX_ELEM_SIZE.SIZE** is less than or equal to 1, i.e. 64bytes. If the max element size>1, then check

is not performed since the maximum element size gets capped at 64bytes+header in case of CTM.
TXFQ_STS.FIFO_FULL is not raised if TXFQ_MAX_ELEM_SIZE.SIZE=0. TXFQ is considered disabled in this case.

1.5.6.5.1.3 TX Priority Queue Virtual Buffer (0x02000 - 0x02808)

This virtual buffer points to the TXPQ tail element in LMEM.i.e, the slot where the next message should be stored into TXPQ. The start address of TXPQ virtual buffer is 0x02000. In FMM, the end address is 0x02808 which corresponds to the maximum possible CAN message size i.e. CAN XL message with 2KB payload and 3 words header. But it is to be noted that the actual size of a TXQE that can be stored into TXPQ depends on the slot size defined in **TXPQ_CFG.SLOT_SIZE** register.

In CTM, the maximum size of a TXQE that can be stored in TXFQ is header plus 64 bytes payload which is 20 words. So the end address for TXFQ in CTM is 0x0204C and the value configured for **TXPQ_CFG.SLOT_SIZE** should not be greater than 20 words. This means entire CAN CC and CAN FD messages can be stored in TXFQ. But in case of XL messages, header plus 64bytes payload of the message can be stored in TXPQ. Remaining payload must be buffered into 2 cut through buffers before sending to PRT.

For starting the enqueuing of a new message into TXPQ, the first word T0 of the message should be written to the start address 0x02000. This triggers the VBM to start the enqueue process into TXPQ. TXPQ handling part of the VBM converts the virtual buffer address to the actual address in LMEM where the next message should be enqueued. When a message gets enqueued into the queue for the first time, the actual address corresponding to the VB start address 0x02000 is **TXPQ_LMEM.SA**. End address is calculated from the number of slots (**TXPQ_CFG.SLOT_NUM**) and the size of each slot (**TXPQ_CFG.SLOT_SIZE**).

The VBM enqueues the message in linear incrementing order of slot index. For example, if number of slots = 32, first message gets enqueued into slot, next one into slot 2 and so on till slot number 32. Host does not have the flexibility to choose the slot index for storing the message.

Consecutive words of the message are written into 0x2004,0x2008 and so on in a linear order. VBM calculates the trigger address based on the type of frame (FDF and XLF bits in T0), RTR and DLC bits in T1. Trigger address corresponds to the last word of the message being enqueued.

For example, a CAN FD frame with DLC = 4, trigger address is for word T2(0x2008). A CAN XL frame with DLC = 7, trigger address is for word T4(0x2010).

After the trigger address is accessed, the next expected address is the start address. In case if this is violated, status flags are updated based on the type of violation. Refer the registers MH_STS0,MH_STS1 or INTERRUPTS chapter for details on this.

After enqueuing of a message into TXPQ slot n, the following registers are updated.

1) Write pointer register **TXPQ_WPTR.WPTR_ADD** with the actual LMEM address corresponding to the last word of TXQE that is enqueued.
2) The slot occupied status into the register **TXPQ_STS0.SLOT_OCC[n]**.
3) The fill level of the TXPQ is incremented by one and is update into the register **TXPQ_STS1.FILL_LEVEL**.

VBM also keeps track of the amount of space in TXPQ for storing the incoming messages. In FMM, if the DLC (part of word T1) of the received message is greater than the value configured in the register **TXPQ_CFG.SLOT_SIZE**, then the message is dropped and interrupt is raised for the transaction. In CTM, this check is done only if the value configured in TXPQ_CFG.SLOT_SIZE < 128bytes since the maximum slot size gets capped at 64bytes+header in case of CTM. Next address expected is again the start address of the virtual buffer. Also, in case if all slots are full, VBM sets the status in **TXPQ_STS1.TXPQ_FULL** register field and virtual buffer access is disabled. Any further accesses to a full TXPQ are discarded. TXPQ_STS1.TXPQ_FULL is not set if **TXPQ_CFG.SLOT_SIZE**=0.

1.5.6.5.1.4 RX FIFO Queue 0 Virtual Buffer (0x03000 - 0x03810)

This virtual buffer points to the RXFQ0 head element in LMEM.i. e, the slot where the next message should be read from RXFQ0. The start address of RXFQ0 virtual buffer is 0x03000 for starting the dequeuing of a message. In FMM, the end

address for an RXQE is 0x03810 which corresponds to 3 words header, 2KB payload and 2 words timestamp. In CTM, RXFQ0 VB is disabled and only cut through buffers are used for buffering the received messages.
RXFQ handling part of the VBM converts the virtual buffer address to the actual address in LMEM from where the next message is dequeued.

When a message gets dequeued from the queue for the first time, the actual address corresponding to the VB start address 0x03000 is **RXFQ0_SA.ADD**. Next location will be **RXFQ0_SA.ADD**+4 and so on. Only read accesses are allowed from RXFQ virtual buffer.

Consecutive words of the message are read from 0x3004,0x3008 and so on in a linear order. VBM calculates the trigger address based on the type of frame (FDF and XLF bits in R0), RTR and DLC bits in R1. Trigger address corresponds to the last word of the message being dequeued.

For example, a CAN FD frame with DLC=4, trigger address will be for the last word of timestamp(0x3010). A CAN XL frame with DLC=7, trigger address will be 0x3018 (3 word header+2 words payload+2 words timestamp).
For a CAN XL with 2KB payload, Trigger address for RXFQ0=0x03810 (3 words header +2 KB payload+2 words of time stamp).

After the trigger address is accessed, the next expected address is the start address. In case if this is violated, status flags will be updated based on the type of violation. Refer the register MH_STS0,MH_STS1 or INTERRUPTS chapter for details on this.

After dequeuing of a message from RXFQ0, the following registers are updated.
1) Read pointer register **RXFQ0_RPTR.ADD** with the actual LMEM address corresponding to the last word of RXQE that is dequeued.
2) The fill level of RXFQ0 is decremented by one and is updated into the register **RXFQ_STS.RXFQ0_FILL_LEVEL**.

If RXFQ0 is empty virtual buffer access will be disabled. Any further accesses to an emptyRXFQ0 is discarded with default read data 0xCAFECAFE and ok response. Corresponding interrupt is also raised.

1.5.6.5.1.5 RX FIFO Queue 1 Virtual Buffer (0x04000 - 0x04810)

This virtual buffer points to the RXFQ1 head element in LMEM.i. e, the slot where the next message should be read from RXFQ1. The start address of RXFQ1 virtual buffer is 0x04000 for starting the dequeuing of a message. In FMM, the end address for an RXQE is 0x04810 which corresponds to 3 words header,2KB payload and 2 words timestamp. In CTM, RXFQ1 VB is disabled and only cut through buffers are used for buffering the received messages. RXFQ handling part of the VBM converts the virtual buffer address to the actual address in LMEM from where the next message should be dequeued.

When a message gets dequeued from the queue for the first time, the actual address corresponding to the VB start address 0x04000 is **RXFQ1_SA.ADD**. Next location will be **RXFQ1_SA.ADD** +4 and so on. Only read accesses are allowed from RXFQ virtual buffer.

Consecutive words of the message are read from 0x04004,0x04008 and so on in a linear order. VBM calculates the trigger address based on the type of frame (FDF and XLF bits in R0), RTR and DLC bits in R1. Trigger address corresponds to the last word of the message being dequeued.

For example, a CAN FD frame with DLC=4, trigger address will be for word R2(0x04010) which includes the last word of timestamp. A CAN XL frame with DLC=7, trigger address will be for word R4(0x04018).

For a CAN XL with 2KB payload, Trigger address for RXFQ1=0x04810 (3 words header +2 KB payload+2 words of time stamp).

After the trigger address is accessed, the next expected address is the start address. In case if this is violated, status flags will be updated based on the type of violation. Refer the registers MH_STS0,MH_STS1 or INTERRUPTS chapter for details on this.

After dequeuing of a message from RXFQ1, the following registers are updated.

1) Read pointer register **RXFQ1_RPTR.ADD** with the actual LMEM address corresponding to the last word of RXQE that is dequeued.
2) The fill level of RXFQ1 is decremented by one and is updated into the register **RXFQ_STS.RXFQ1_FILL_LEVEL**.

If RXFQ1 is empty virtual buffer read access will be disabled. Any further accesses to an empty RXFQ1 is discarded.

1.5.6.5.1.6 TX Event FIFO Queue Virtual Buffer

This virtual buffer points to the TEFQ head element in LMEM.i. e, the slot where the next message should be read from TEFQ. The start address of TEFQ virtual buffer is 0x05000 for starting the dequeuing of a message. TEFQ handling part of the VBM converts the virtual buffer address to the actual address in LMEM from where the next TEQE should be dequeued. TEFQ handling is the same in FMM and CTM.

When a TEQE gets dequeued from the queue for the first time, the actual address corresponding to the VB start address 0x05000 is **TEFQ_LMEM.SA**.

Consecutive words of the message are to be read by the host from 0x05004,0x05008 and so on in a linear order. VBM calculates the trigger address based on the **TEFQ_CFG.TEQE_LARGE** register configuration. If **TEFQ_CFG.TEQE_LARGE**=1, then there are 5 words in an element, else 4 words. Refer chapter 1.5.6.3 for TEQE format. When **TEFQ_CFG.TEQE_LARGE**=1, trigger address is 0x05010 else trigger address is 0x0500C.

After the trigger address is accessed, the next expected address is the start address. In case if this is violated, status flags will be updated based on the type of violation. Refer the registers MH_STS0,MH_STS1 or INTERRUPTS chapter for details on this.

After dequeuing of a message from TEFQ, the following registers are updated by VBM.
1) Read pointer register **TEFQ_RPTR.RPTR_ADD** with the actual LMEM address corresponding to the last word of TEQE that is dequeued.
2) The fill level of TEFQ is decremented by one and is updated into the register **TEFQ_STS.FILL_LEVEL**.

If TEFQ is empty, virtual buffer read access will be disabled. Any further accesses to an empty TEFQ is discarded.Refer section Error and Exception handling for more details.

1.5.6.5.1.7 Cut Through Buffer (CTB) Virtual Buffer (0x06000 - 0x0603C)

Cut Through buffers are used only in CTM and are disabled in FMM. In CTM, the same cut through buffers is shared for TX and RX directions.

1) **TX Direction**

In TX direction, CTB virtual buffer points to the start of the empty cut through buffer to which CAN XL payload beyond 64bytes must be enqueued. This size of transfer is always in 64 byte blocks and is called block copy transfer in TX direction. For TX, the block copy is from SMEM to LMEM. The start address of CTB VB to start enqueuing is 0x06000. As already explained in section 1.5.6.4.6, there are two cut through buffers (CTB 0 and CTB1) of size 64bytes each and the CTB VB start address can point to either one of the buffer based on the availability. CTB handling part of the VBM converts the virtual buffer address to the actual address in LMEM to which the next word must be enqueued.

For starting the enqueuing of a block into CTB, start address 0x06000 must be accessed. This triggers the VBM to start the enqueue process into CTB0 or CTB1 based on which is free. CTB handling part of the VBM converts the virtual buffer address to the actual address in LMEM where the next message should be enqueued. When a message gets enqueued into the queue for the first time, the actual address corresponding to the VB start address 0x06000 is **CTB_LMEM.SA**. Next location will be **CTB_LMEM.SA**+4 and so on. Only write accesses are allowed to CTB virtual buffer in TX direction.

2) **RX Direction**

In RX direction, CTB virtual buffer points to the start of the cut through buffer from which received CAN message must be dequeued. This size of transfer is always in 64 byte blocks and is called block copy transfer in RX direction. For RX, the block copy is from LMEM to SMEM. The start address of CTB VB to start dequeuing is 0x06000. As already explained in section 1.5.6.4.6, there are two cut through buffers (CTB 0 and CTB1) of size 64bytes each and the CTB VB start address

can point to either one of the buffer based on which buffer needs to be read by the host. CTB handling part of the VBM converts the virtual buffer address to the actual address in LMEM from which the next word must be dequeued.
When a message is read, the virtual buffer address 0x06000 maps to CTB_LMEM.SA, which is the start address of the selected cut-through buffer in LMEM. Subsequent words are read sequentially from CTB_LMEM.SA + 4, CTB_LMEM.SA + 8, and so on.
Only read accesses are permitted to the CTB Virtual Buffer in the RX direction. Data is read linearly from the virtual addresses 0x06000, 0x06004, 0x06008, ..., up to 0x0603C. Since the block size is fixed at 64 bytes, the trigger address is the same as the end address (0x0603C).
If there is no valid data to be read from any of the buffer, virtual buffer access will be disabled. Any further accesses to an empty CTB is discarded. Refer section Error and Exception handling for more details

After the trigger address is accessed, the next expected address is the start address. In case if this violated, status flags will be updated based on the type of violation. Refer the registers **MH_STS0, MH_STS1** or INTERRUPTS chapter for details on this.

1.5.6.5.1.8 LMEM Transparent Access

Host can access the full 256KB LMEM address space using the address range 0x40000 to 0x7FFFC.But the actual memory space allotted to an XS_CAN IP is indicated by the LMEMPC_START_ADDR and LMEMPC_END_ADDR values. Transparent accesses outside this range of start and end address will result in generation of LMEM_PROTECT_EVENT output.

Host must store the RX filter elements (Filter element configuration and reference value-mask pair) in the LMEM via transparent access addresses. Virtual buffers are not available for storing RX filter elements. Both read and write operations are allowed in this address range. MH need not be enabled before performing this access but HOST_CLK should be active to enable this access. LMEM transparent handling is the same in FMM and CTM.

1.5.6.5.2 Transfer Request Signals (FMM and CTM)

VBM provides the following requests to the host and CTDMA to perform transfers between SMEM and LMEM. It is to be noted that these requests are generated when MH is enabled.i.e, **MH_CFG.ENABLE** bit should be high.

a) *TXFQ_WREQ*: This TXFQ write request is raised by VBM when TXFQ is not full and can accept new message. After the request is raised, from the next cycle of HOST_CLK, VBM expects TXFQ virtual buffer start address.Once the TXFQ virtual buffer start address (0x01000) is received at VBM AHB interface, this request will be pulled low by VBM till the trigger address is reached and again raised only if TXFQ is not full after the message is enqueued. If the user configures TXFQ_CFG.MAX_ELEM_SIZE as 0, TXFQ_WREQ is not raised.

b) *TXPQ_WREQ*: The TXPQ write request is raised by VBM if there is a free slot in the TXPQ and new message can be written into this slot. After the request is raised, from the next cycle of HOST_CLK, VBM expects TXPQ virtual buffer start address.Once the TXPQ virtual buffer start address (0x02000) is received at VBM AHB interface, this request will be pulled low by VBM till the trigger address is reached and again raised only if TXPQ is not full. If the user configures TXPQ_CFG.SLOT_SIZE or TXPQ_CFG.SLOT_NUM as 0, TXPQ_WREQ is not raised.

c) *RXFQ0_RREQ*: The RXFQ0 read request is raised by VBM if the RXFQ0 is not empty and there is valid data to be read by the host. After the request is raised, from the next cycle of HOST_CLK, VBM expects RXFQ0 virtual buffer start address.Once the RXFQ0 virtual buffer start address (0x03000) is received at VBM AHB interface, this request will be pulled low by VBM till the trigger address is reached and again raised only if RXFQ0 is not empty after the message is dequeued.
Note: **RXFQ0_RREQ** is raised only in FMM as RXFQ0 is disabled in LMEM in CTM.

d) *RXFQ1_RREQ*: The RXFQ1 read request is raised by VBM if the RXFQ1 is not empty and there is valid data to be read by the host. After the request is raised, from the next cycle of HOST_CLK, VBM expects RXFQ1 virtual buffer start address. Once the RXFQ1 virtual buffer start address (0x04000) is received at VBM AHB interface, this request will be pulled low by VBM and again raised only if RXFQ1 is not empty after the message is dequeued.
Note: **RXFQ1_RREQ** is raised only in FMM as RXFQ1 is disabled in LMEM in CTM.

e) *TXEF_RREQ*: The TX Event FIFO read request is raised if TEFQ is not empty and there is a TEQE to be read by the host. After the request is raised, from the next cycle of HOST_CLK, VBM expects TEFQ virtual buffer start address. Once the TEFQ virtual buffer start address (0x05000) is received at VBM AHB interface, this request will be pulled low by VBM and again raised only if TEFQ is not empty after the element is dequeued.

f) *CTM_TX_WREQ (to CTDMA)*: The Cut Through Buffer write request is raised by VBM in CTM if 64byte block of CAN XL message has to be transferred from SMEM to CTB for transmission. After the request is raised, from the next clock cycle, CTB start address is expected.Upon receiving the CTB virtual buffer start address (0x06000) at VBM AHB interface, this request will be pulled low by VBM and again raised only if CTB is empty once the block gets transmitted successfully.
If the host wants to know the source address and destination address of the block copy transfer, the information is stored into the cut through descriptor registers CTM_DESC_SRC, CTM_DESC_DEST by the IP. More details are given in the section 1.5.7 Cut Through Descriptors and block copy.

Note: This request is raised only in CTM and not in FMM. The transfer is always done in blocks of size 64 bytes by the CTDMA.

g) *CTM_RX_RREQ* (to CTDMA): The Cut Through Buffer read request is raised by VBM in CTM for transferring all received messages from CTB in 64 byte blocks. The request is raised when at least one CTB has valid data to be transferred. After the request is raised, from the next cycle of HOST_CLK, VBM expects CTB virtual buffer start address. Upon receiving the CTB virtual buffer start address (0x06000) at VBM AHB interface, this request will be pulled low by VBM and again raised only if the previous transfer was completed and there is next block of data in CTB to be transmitted.
If the host wants to know the source address and destination address of the block copy transfer the information is stored into the cut through descriptor registers CTM_DESC_SRC, CTM_DESC_DEST by the IP. More details are given in the section 1.5.7 Cut Through Descriptors and block copy.

Note: This request is raised only in CTM and not in FMM. The transfer is always done in blocks of size 64 bytes by the IP. If the payload size is less than 64byte, zeros are appended by the IP while storing into CTB.

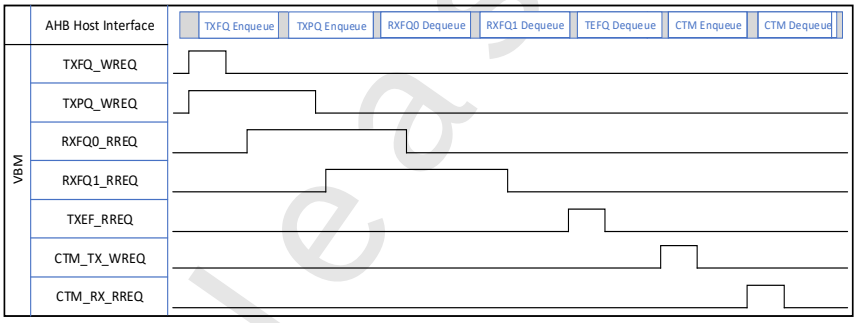


Figure 12. VBM TIMING

1.5.6.5.3 LMEM Request Generation

VBM generates the accesses to LMEM based on the request from the different queue FSMs. Host issues read write accesses to LMEM through virtual address as explained in section 1.5.6.5.1.VBM receives decoded virtual address issued by the host and generates the select signal (SELX[6]). This select signal selects between the request outputs of six FSMs as given below to be transmitted to LMEM_CTRL:
TX FQ Enqueue
TX PQ Enqueue
RX FQ0, RX FQ1 Dequeue
TX EF Dequeue
CTM Enqueue and Dequeue

At a time only one queue can be accessed by the host. AHB protocol does not support parallel read and writes. Hence, only one request to LMEM_CTRL is active and served by the VBM.

After the request is granted, corresponding address and control signals are placed at the LMEM interface of VBM. In case of an enqueue to LMEM, actual LMEM address, wdata (from *VBM_AHB_HWDATA*), cs and we signals are placed to LMEM_CONTROLLER. Once the ready from LMEM is high, access is considered complete. In case of a read, actual LMEM address along with cs and we signals are placed. Read data is sampled by VBM when ready from LMEM is high and directly propagated into RDATA of host AHB interface (*VBM_AHB_HRDATA*) for RX FQ0/1, TX EF and CTB Dequeue.
The ready received from LMEM_CTRL is propagated back as VBM_AHB_HREADY to the host to release the AHB bus after completion of a transaction. In case of uncorrected ECC error in a read access (indicated by *VBM_LMEM_ECC_UE*

= '1'), ahb_hresp is given as ERROR to the host. If an access to LMEM is successful, OK response is given by VBM back to the host.

Based on the type of transfer, the output signals VBM_LMEM_BURST and VBM_LMEM_BURST_LEN are also updated by the VBM. For burst accesses by host, VBM_LMEM_BURST=1 and the length of the burst is indicated as VBM_LMEM_BURST_LEN = 01 for INCR4, 10 for INCR8 and 11 for INCR16.
For single access, VBM_LMEM_BURST=0 and VBM_LMEM_BURST_LEN = 00

1.5.6.5.4 Interleaved Accesses in VBM

VBM supports interleaved accesses to different queues. For example, when TXFQ/TXPQ enqueuing is ongoing, host can perform RXFQ/TEFQ dequeue in between or vice versa. Note that VB addresses to each queue must be issued in a linear order and also a burst transfer is an atomic transfer which cannot be split into multiple transfers by the host. Each burst request is completed before serving the next request at VBM interface.

1.5.6.5.5 VBM Interrupts and Status flags

VBM updates the following flags into MH_STS0 register. Flags are updated within 5 host clock cycles.

1) TXFQ_ELEM_TOO_BIG- Set when DLC of the message getting enqueued into TXFQ is greater than TXFQ_CFG.MAX_ELEM_SIZE in FMM and CTM. This condition also triggers the pulse interrupt ERR_STS_QUEUE_VIOLATION_IRQ.

2) TXPQ_ELEM_TOO_BIG- Set when DLC of the message getting enqueued into TXPQ is greater than TXPQ_CFG.SLOT_SIZE in FMM and CTM. This condition also triggers the pulse interrupt ERR_STS_QUEUE_VIOLATION_IRQ.

3) TXFQ_FULL_WR- Set when host tries to write to a full TXFQ in FMM and CTM. This condition also triggers the pulse interrupt ERR_STS_QUEUE_VIOLATION_IRQ.

4) TXPQ_FULL_WR- Set when host tries to write to a full TXFQ in FMM and CTM. This condition also triggers the pulse interrupt ERR_STS_QUEUE_VIOLATION_IRQ.

5) TEFQ_EMPTY_RD- Set when host tries to read from an empty TEFQ in FMM and CTM.This condition also triggers the pulse interrupt ERR_STS_QUEUE_VIOLATION_IRQ.

6) RXFQ0_EMPTY_RD- Set when host tries to read from an empty RXFQ0 in FMM.In CTM, this is not applicable since RXFQs in LMEM is disabled and is located in SMEM.This condition also triggers the pulse interrupt ERR_STS_QUEUE_VIOLATION_IRQ.

7) RXFQ1_EMPTY_RD- Set when host tries to read from an empty RXFQ1 in FMM.In CTM, this is not applicable since RXFQs in LMEM is disabled and is located in SMEM.This condition also triggers the pulse interrupt ERR_STS_QUEUE_VIOLATION_IRQ.

8) CTB_RD_WO_REQ- Set when host tries to read from a CTB without an active read request, in CTM.This condition also triggers the pulse interrupt ERR_STS_QUEUE_VIOLATION_IRQ.

9) CTB_WR_WO_REQ- Set when host tries to write to a CTB without an active write request, in CTM.This condition also triggers the pulse interrupt ERR_STS_QUEUE_VIOLATION_IRQ.

10) TXFQ_VB_NO_SA- Set when host issues another address within the VB address range instead of the expected start address to TXFQ VB in FMM and CTM.This condition triggers the pulse interrupt ERR_STS_VB_NO_SA_IRQ.

11) TXPQ_VB_NO_SA- Set when host issues another address within the VB address range instead of the expected start address to TXPQ VB in FMM and CTM.This condition triggers the pulse interrupt ERR_STS_VB_NO_SA_IRQ.

12) TEFQ_VB_NO_SA- Set when host issues another address within the VB address range instead of the expected start address to TEFQ VB in FMM and CTM.This condition triggers the pulse interrupt ERR_STS_VB_NO_SA_IRQ.

13) RXFQ0_VB_NO_SA- Set when host issues another address within the VB address range instead of the expected start address to RXFQ0 VB in FMM. In CTM, this is not applicable since RXFQs in LMEM is disabled and is located in SMEM.This condition triggers the pulse interrupt ERR_STS_VB_NO_SA_IRQ.

14) RXFQ1_VB_NO_SA- Set when host issues another address within the VB address range instead of the expected start address to RXFQ1 VB in FMM.In CTM, this is not applicable since RXFQs in LMEM is disabled and is located in SMEM.This condition triggers the pulse interrupt ERR_STS_VB_NO_SA_IRQ.

15) CTB_VB_NO_SA- Set when host issues another address within the VB address range instead of the expected start address to CTB VB in CTM.This condition triggers the pulse interrupt ERR_STS_VB_NO_SA_IRQ.

VBM resets the corresponding FSM when an illegal access mentioned above occurs and continue to expect start address. Each of the flag mentioned above is reset with a read access to MH_STS0 register.

VBM updates the following flags into MH_STS1 register within 5 host clock cycles

1) TXFQ_VB_NONLIN_ACC- Set when host issues non linear address after issuing the start address to TXFQ VB in FMM and CTM. This condition triggers the pulse interrupt SAFETY_VB_ACCESS_VIOLATION_IRQ.

2) TXPQ_VB_NONLIN_ACC- Set when host issues non linear address after issuing the start address to TXPQ VB in FMM and CTM. This condition triggers the pulse interrupt SAFETY_VB_ACCESS_VIOLATION_IRQ.

3) TEFQ_VB_NONLIN_ACC- Set when host issues non linear address after issuing the start address to TEFQ VB in FMM and CTM. This condition triggers the pulse interrupt SAFETY_VB_ACCESS_VIOLATION_IRQ.

4) RXFQ0_VB_NONLIN_ACC- Set when host issues non linear address after issuing the start address to RXFQ0 VB in FMM. In CTM, this is not applicable since RXFQs in LMEM is disabled and is located in SMEM.This condition triggers the pulse interrupt SAFETY_VB_ACCESS_VIOLATION_IRQ.

5) RXFQ1_VB_NONLIN_ACC- Set when host issues non linear address after issuing the start address to RXFQ1 VB in FMM. In CTM, this is not applicable since RXFQs in LMEM is disabled and is located in SMEM. This condition triggers the pulse interrupt SAFETY_VB_ACCESS_VIOLATION_IRQ.

6) CTB_VB_NONLIN_ACC- Set when host issues non linear address after issuing the start address to CTB VB in FMM. This condition triggers the pulse interrupt SAFETY_VB_ACCESS_VIOLATION_IRQ.

7) TXFQ_VB_RD- Set when host issues AHB read access to TXFQ VB in FMM and CTM. Host has permission to only write to TXFQ Virtual buffer. This condition triggers the pulse interrupt SAFETY_VB_ACCESS_VIOLATION_IRQ.

8) TXPQ_VB_RD- Set when host issues AHB read access to TXPQ VB in FMM and CTM. Host has permission to only write to TXPQ Virtual buffer. This condition triggers the pulse interrupt SAFETY_VB_ACCESS_VIOLATION_IRQ.

9) TEFQ_VB_WR- Set when host issues AHB write access to TEFQ VB in FMM and CTM. Host has permission to only read from TEFQ Virtual buffer. This condition triggers the pulse interrupt SAFETY_VB_ACCESS_VIOLATION_IRQ.

10) RXFQ0_VB_WR- Set when host issues AHB write access to RXFQ0 VB in FMM. Host has permission to only read from RXFQ0 Virtual buffer. This condition triggers the pulse interrupt SAFETY_VB_ACCESS_VIOLATION_IRQ.

11) RXFQ1_VB_WR- Set when host issues AHB write access to RXFQ1 VB in FMM. Host has permission to only read from RXFQ1 Virtual buffer. This condition triggers the pulse interrupt SAFETY_VB_ACCESS_VIOLATION_IRQ.

12) CTB_VB_TX_RD- Set when host issues AHB read access to CTB in TX direction in CTM. Host has permission to only write to CTB in TX path in CTM. This condition triggers the pulse interrupt SAFETY_VB_ACCESS_VIOLATION_IRQ.

13) CTB_VB_RX_WR- Set when host issues AHB write access to CTB in RX direction in CTM. Host has permission to only read from CTB in RX path in CTM. This condition triggers the pulse interrupt SAFETY_VB_ACCESS_VIOLATION_IRQ.

VBM resets the corresponding FSM when an illegal access mentioned above occurs and continue to expect start address. Each of the flag mentioned above is reset with a read access to MH_STS1 register.

Refer section 1.5.6.9 for details on all interrupts generated by MH.

1.5.6.6 TX Message Handler

TX Handler module handles all the functionalities related to the transmission of a message to the PRT and the response back from PRT at PRT TX_MSG interface. The messages stored in TXFQ and TXPQ are first scanned to identify the highest priority message and this message is sent out to PRT for transmission on the CAN bus. This process is called TX-SCAN. Once the arbitration is won on the bus, TX Handler fetches the remaining payload from the selected queue and sends to the PRT. The response or feedback related to the transmitted message is stored back into TX Event FIFO queues by TX Handler.

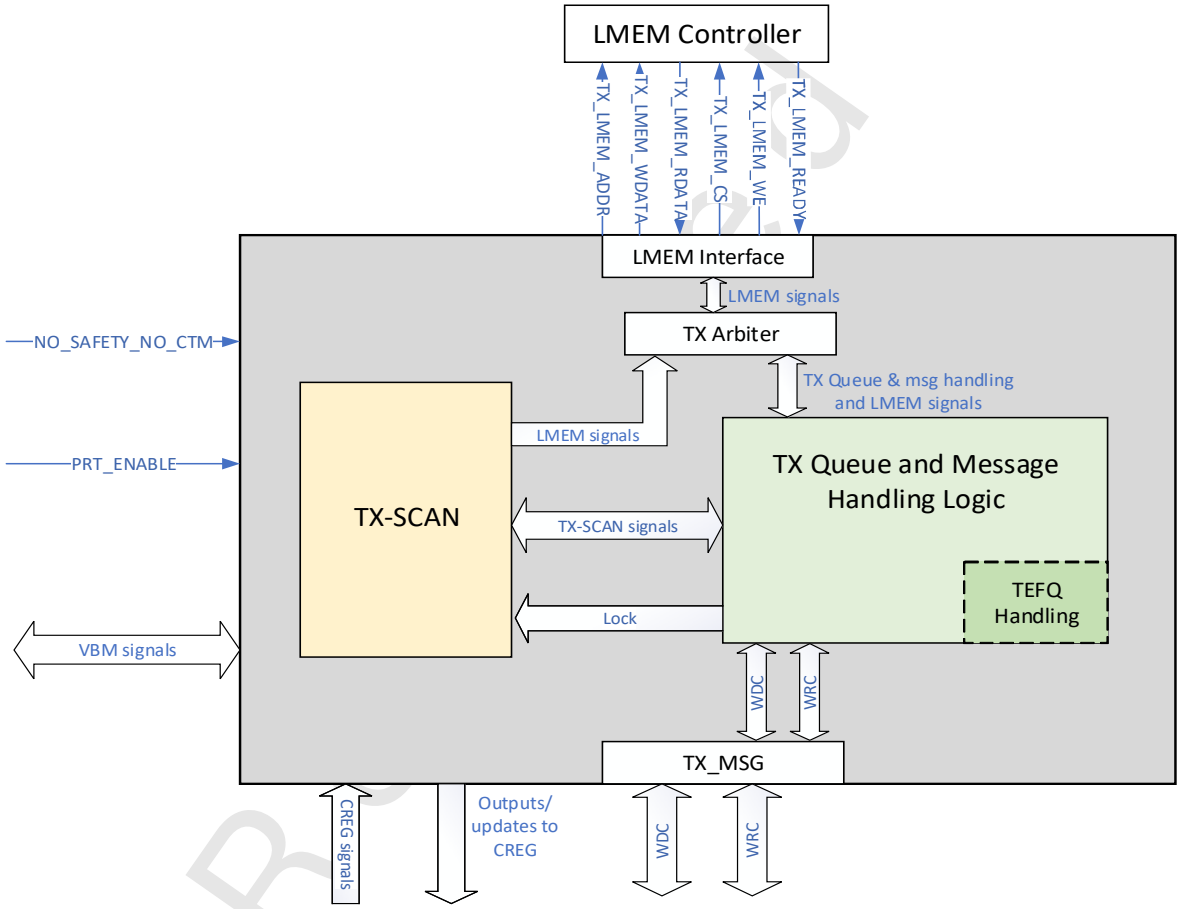


Figure 13. TXH Block Diagram

1.5.6.6.1 TX-SCAN

TX scan is the process of identifying the highest priority message to be transmitted from the enqueued messages in LMEM. While the Elements of the Tx FIFO Queue are to be transmitted in the order they were enqueued, the head Element of that FIFO (TXFQ_RPTR) is treated like an Element of the priority queue when the TX handler evaluates the transmit sequence, based on their relative priorities. For this internal arbitration, the priority should be the same as for the arbitration between CAN frames on the CAN bus, meaning that the frame with the lowest ID has the highest priority.

The elements considered for scan are the entire TXPQ contents and the head element from TXFQ. The element just after the head element in TXFQ is also considered for scanning under some specific scenarios which is explained later. TX Scan logic consists of two parallel processes. One process, TX_SCAN fetches the headers from the LMEM queues and prepares a Transient Buffer containing an intermediate scan result. The next process TX_BUFFER_MANAGER compares the transient buffer results with Prepared Candidate buffer which holds the second highest priority result and Winning Candidate Buffer which holds the highest priority message. The winning candidate buffer consists of the address and header of the message to be transmitted. The TX queue message handling logic takes this address and issues LMEM read transactions to continue with the fetching of the message and issue to PRT.

The table below shows how the 32bit priority value is calculated for CAN CC, CAN FD and CAN XL. The fields XLF, FDF, XTD, SRR and ID (defined in CAN protocol [1] and [2]) are used to determine the priority value of a given TX message. The priority value is computed for every TX message and then compared with each other to identify the highest priority message (the lowest value gives the highest priority message to transmit). For CAN CC, data frames and remote frames are not differentiated during TX-Scan.

Table 66. TX-SCAN Prioritization

CAN Protocol	Protocol Selection			Priority Value						
	XLF (T0[30])	FDF (T0[31])	XTD (T0[29])	31 down to 21	20	19	18	17	16 down to 1	0
Classical CAN	0	0	0	T0[28:18] (Base ID[10:0])	0	0 (XTD)	0 (FDF)	0	16'b0	0
Classical CAN (extended ID)	0	0	1	T0[28:18] (Base ID[10:0])	1 (SRR)	1 (XTD)	T0[17:0] (Identifier Extension[17:0])			0
CAN FD	0	1	0	T0[28:18] (Base ID[10:0])	0 (RRS)	0 (XTD)	1 (FDF)	0	16'b0	0
CAN FD (extended ID)	0	1	1	T0[28:18] (Base ID[10:0])	1 (SRR)	1 (XTD)	T0[17:0] (Identifier Extension[17:0])			0 (RRS)
CAN XL	1	X	X	T0[28:18] (Priority ID[10:0])	T0[17] (RRS)	0 (XTD)	1 (FDF)	1 (XLF)	16'b0	0

The selection of the TX message is done in the following order, TX Priority Queue slots from 0 to 31, followed by the head element (the element enqueued first) of TX FIFO Queue. If there are no messages in TXPQ (i.e. **TXPQ_STS1.FILL_LEVEL**=0), the elements enqueued into TXFQ are transmitted in the order of storage. The first message enqueued is selected by the scan algorithm followed by the second message.

Priority Calculation for messages in TXPQ with same Priority Value

This section explains how XS_CAN IP handles the message selection in case of messages with the same priority value in TXPQ. For calculating the Priority value, the header T0 of the TXFQ head element is fetched. In case of TXPQ, the configuration bit TXPQ_CFG.TX_MSG_SEQ_ENABLE decides whether to fetch T0 and T1 both. If this bit is set, then T1 is also fetched to use the Message Marker[5:0] bits to define the transmission order in case of same priority value for TXPQ messages.

TXPQ_CFG.TX_MSG_SEQ_ENABLE is disabled
In case TXPQ_CFG.TX_MSG_SEQ_ENABLE is disabled, the XS_CAN selects the TX messages with same priority value in the order of the TX scan. This means a TX messages with a lower TXPQ slot number is transmitted first, followed by the TXFQ head element as last. Since the VBM of the XS_CAN assigns the slot numbers dynamically during enqueue process, the transmit order for TX messages with same priority value will be practically random on the CAN bus.

TXPQ_CFG.TX_MSG_SEQ_ENABLE is enabled
In case TXPQ_CFG.TX_MSG_SEQ_ENABLE is enabled, the order of TX message transmission with the same priority value can be controlled via Message Marker[5:0] value. This means transmission order does not depend on the slot number anymore.

The lower 6bits of Message Marker, Message Marker[5:0],in header T1 are considered for determining the transmission sequence if more than one message in TXPQ share the same priority value. Message Marker[5:0] are called Seq_ID and defines the sequence of the messages. Seq_ID is an integer number in the range of 0 to 63. M=22 is the maximum number of elements in the TXPQ with the same priority value. If TXPQ_CFG.SLOT_NUM <= M, the algorithm is always functional. If TXPQ_CFG.SLOT_NUM > M then the user must assure, that never more than M TXPQ slots have the same

priority value. If more than M TX messages with same priority value are enqueued simultaneously, the order of transmission of these TX messages is not guaranteed to be according to the Seq_ID.
If a TXFQ element has the same priority value as a TXPQ element, then the TXPQ element is transmitted first, independent of the Seq_ID.
The Seq_ID values of the TX messages with same priority value must be numbered sequentially (+1) increasing. After Seq_ID 63 the Seq_ID 0 follows. If there is no return to zero in the Seq_ID values in the TXPQ, the messages are transmitted relative to each other in the order of their Slot_ID values, starting with the lowest Seq_ID value. Their internal priority is not determined by their positions in the TXPQ, but by their Seq_ID value. However, if there is a return to zero in the Seq_ID values the underlying algorithm has to detect this and send the messages with larger Seq_ID first. To detect this wrap-around case, the Seq_ID values are divided into three groups, the upper group C of X to the maximum value of Seq_ID, the lower group A from zero to Y, and the group B with the remaining Seq_ID elements.
In case of the wrap-around all Seq_ID are in either group A or C. The messages with Seq_ID in the upper group C are transmitted first, then the lower group A elements are transmitted. Host must ensure that the TXPQ messages with the same priority value are enqueued with sequentially (+1) increasing sequence numbers Seq_ID. If the message first enqueued to TXPQ has Seq_ID=N, then message enqueued next must have the Seq_ID=N+1 and so on. After Seq_ID=63 the Seq_ID=0 follows. Apart from the possible return to zero case, the messages with the same priority value are transmitted in the relative order of their Seq_ID.
Algorithm description to determine the sequence of messages with same priority value
To be able to recognize, a return to zero in the Seq_ID values, the value range of Seq_ID is divided into three ranges. The lower range_A (starting with 0) and the upper range (ending with maximum of Seq_ID) contain each (M-1) Seq_ID values. The middle range_B must contain at least (M-1) values and contains in the XS_CAN M values. In total all three ranges A, B, and C together contain 64 values: (M-1)+M+(M-1)= 3x22-2= 64. Because there are significantly more Seq_ID values than elements in the TXPQ and because all Seq_ID values for a specific priority value (manly defined by the CAN identifier) are sequential (+1), this ensures that at any time the Seq_ID values of a specific priority value in the TXPQ can never be spread over all three ranges A, B, and C, but only over two ranges. If the numerical values of a sequence of Seq_ID values in the TXPQ are in range_A and range_C, this sequence contains a return to zero, and can be detected by the TX scan algorithm. In the latter case, the messages with the Seq_ID values in range_C are sent before range_A to maintain the sequential order.

If Seq_ID <21, it is in range_A.
If Seq_ID > 42, it is in range_C
If Seq_ID is >=21 and <= 42, its in range_B.

The Tx-Scan algorithm needs some registers and signals, given below:
TXFQ_STS.FILL_LEVEL TX FIFO Queue fill level
TXFQ_RPTR TX FQ Read pointer

TXPQ_CFG.SLOT_NUM TX PQ Number of Slots
TXPQ_STS0 TX Priority Queue Status register
TXPQ_STS1 TX Priority Queue Status register

There are 4 internal buffers used by the TX Scan processes
1) LMEM Candidate Buffer (LCB): Current message details which is being fetched from LMEM for comparison.
2)Transient Candidate Buffer (TCB): Intermediate scan result after one round of scan is finished
3) Prepared Candidate Buffer (PCB): Second highest priority message details.
4) Winning candidate buffer (WCB): Highest priority message details.

LMEM buffer and Transient Buffer are in the TX_SCAN process. Prepared Candidate Buffer and Winning Candidate Buffer are in TX_BUFFER_MANAGER process.

All the buffers are registers of the same type consisting of two words:
i) First word is the header T0 of the message(32bits).
ii) Second word consists of the 16bit address of the start of the message along with one bit P_F to identify if the message is from PQ or FQ and also a valid flag (VALID) bit that shows whether the element is valid and pending for transaction. If the register bit TXPQ_CFG.TX_MSG_SEQE is set by the host, then for messages read from TXPQ, the lower order 6bits of Message Marker from word T1 are also stored.

The below table shows the format of the contents of the LCB and TCB buffer.

Table 67. LCB and TCB Buffer

T0 word (32 bits)				
6 bits Message Marker if TX_PQ_CFG.TX_MSG_SEQ_ENABLE = 1 and message is from TXPQ	TX_PQ slot num if P_F = 1	VALID (0 = Not pending for transaction, 1 = pending for transaction)	P_F (0 = TXFQ element, 1 = TXPQ element)	16 bit address corresponding to T0

The below table shows the format of the contents of the WCB and PCB buffer.

Table 68. WCB and PCB Buffer

T0 Word (32 bits)			
TX_PQ slot num if P_F = 1	VALID (0 = Not pending for transaction, 1 = pending for transaction)	P_F (0 = TXFQ element, 1 = TXPQ element)	16 bit address corresponding to T0

A new TX scan process gets triggered based on the conditions given below.

1) A new TX message enqueued into TX PQ slot.
2) A message dequeued from TXPQ or TXFQ due to any one of the 3 possible reasons: message transmission at PRT, dropping after N retransmissions and drop due to HFI.
3) Contents from TCB moved to PCB.
4) Contents from PCB moved to WCB
5) TXFQ fill level becomes greater than 0 from 0 value.

NB: A new message enqueued to TXFQ while fill level >1 does not trigger a scan.
As soon as the first message is enqueued into TXPQ successfully, scan gets triggered and this message is selected by default for transmission. If PRT is enabled during an ongoing TX Scan process which was triggered due to enqueuing of a message, already selected winning candidate will be sent for transmission even though the last enqueued message is of higher priority.
If the trigger happens while the TX-Scan is already running, a trigger pending flag is set in the TX_SCAN process. The current TX-Scan completes and is started again to take this new trigger.

A re-transmission counter defines the number of re-transmissions allowed to the same TX message when this one is unsuccessful. For every trial of the same TX message, the re-transmission counter is incremented and compared to a maximum value defined in the **MH_CFG.MAX_RETRANS[2:0]**. If the counter exceeds the limit, the current TX message will no longer be considered and is skipped, the next TX message is taken instead. The re-transmission counter is set back to 0 when a new TX message is selected. There is the option to define an unlimited number of trials for TX messages. The maximum number of re-transmissions is defined by the register **MH_CFG.MAX_RETRANS[2:0]** and covers the maximum value defined in 11898-1:2024.

Several options are defined:
• 0: No re-transmission
• 1 to 6: 1 to 6 re-transmissions
• 7: Unlimited re-transmissions (default value)

The two processes in the TX scan algorithm are explained below in flow chart:-

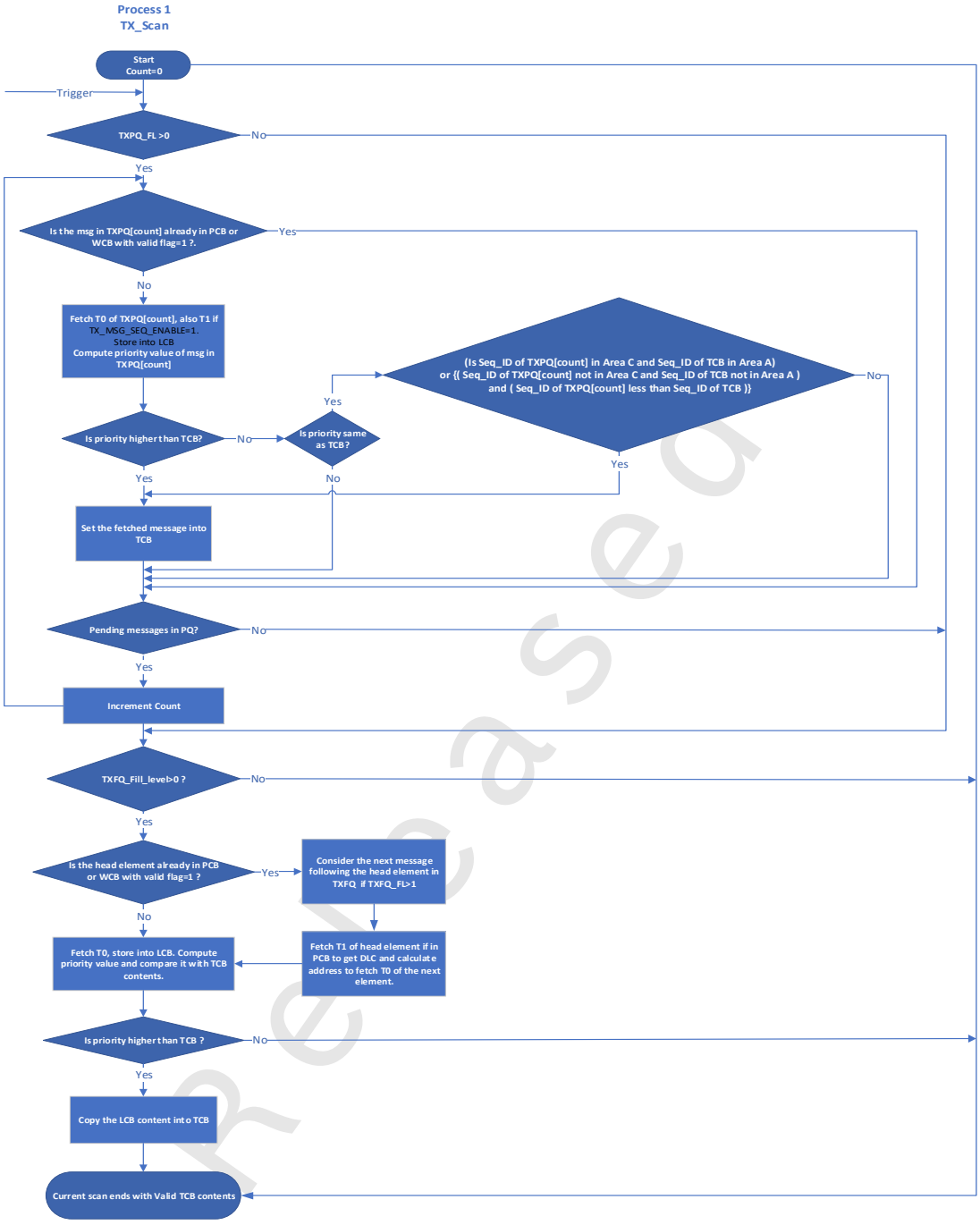


Figure 14. TX SCAN Flowchart

The process TX-Scan gets started every time there is a trigger. After this process is completed, the transient Buffer Content becomes Valid. This result is used by the process TX_Buffer_manager which runs concurrently with TX-Scan process.

Below is the flowchart showcasing the process of TX_Buffer_manager.

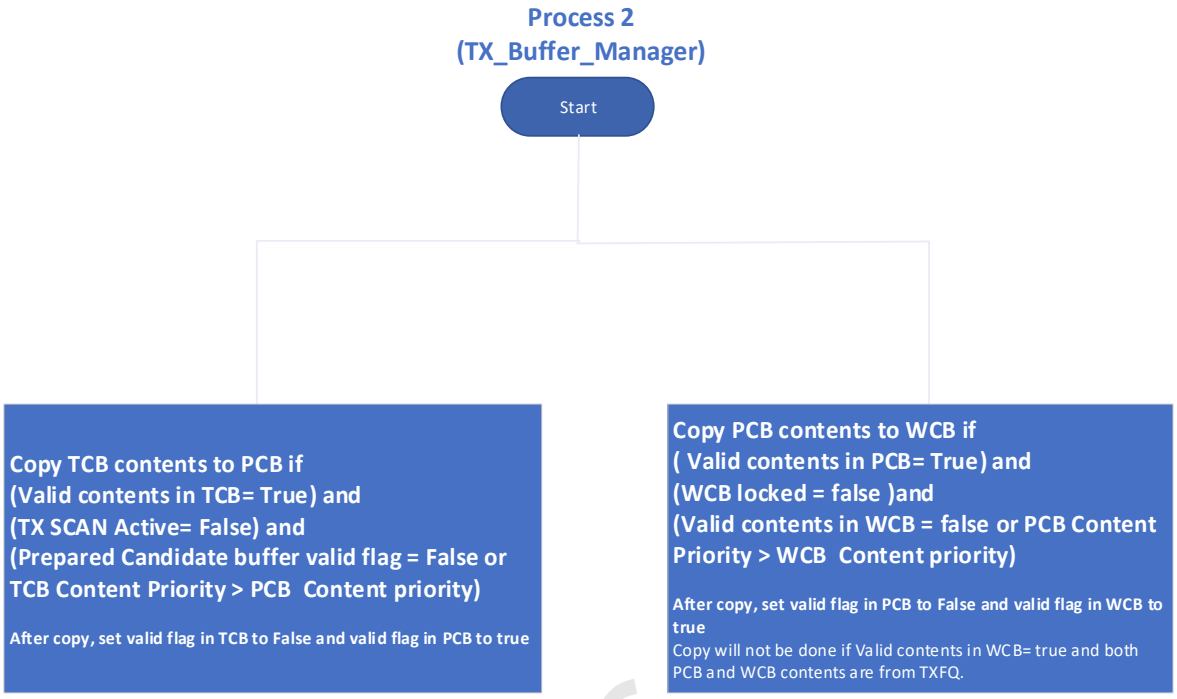


Figure 15. TX_Buffer_manager Flowchart

TX Buffer manager process manages the contents inside Prepared Candidate Buffer (PCB) and Winning Candidate buffer (WCB).

After scan is complete and WCB and PCB are in place, the address of the winning candidate from WCB is taken by TX queue message handler process and the header T1 of the message is read from LMEM. The words T0 and T1 are issued to PRT for arbitration. While a transmit phase is going on, the contents in WCB is locked by TX Handler until one of the conditions mentioned below occur.

1) Message successfully transmitted (ACK response from PRT)- In this case, the contents of WCB are replaced by contents of PCB and contents of TCB are moved to PCB if valid. Scan is triggered again to find the content for TCB.

2) HFI error from PRT- In this case, the message is dropped from transmission and not considered for the further scan. The contents of WCB are replaced by contents of PCB and scan is triggered again.

3) Arbitration lost- In this case, the WCB contents are compared with PCB content again. If PCB priority is greater than WCB priority, the contents of PCB are copied to WCB. Otherwise, the same message is retried again. Retransmission counter is not incremented in this case.

4) Restart response from PRT- The WCB is retried for transmission to PRT if it is still the higher priority message and the retransmission counter is incremented. If not, the PCB contents are copied into WCB and the new message is sent for transmission and retransmission counter is reset.

5) Message dropped after N retransmissions-- In this case, the message is dropped from transmission and not considered for the further scan. The contents of WCB are replaced by contents of PCB and scan is triggered again.

6) Data Underrun Code from PRT- If PRT gives DU code at TX_MSG_BUSER_STATUS, MH stops the current message transmission and contents of WCB are unlocked. WCB is retried for transmission to PRT if it is still the higher priority message and the retransmission counter is incremented. If not, the PCB contents are copied into WCB and the new message is sent for transmission and retransmission counter is reset.

WCB contents are set to invalid by TX queue message handler only on successful transmission or on message drop scenarios. Note that TXFQ messages cannot overtake each other. They must be transmitted in the order of storing. In case if WCB content is the head element of TXFQ and the WCB is locked for transmission, TX Scan considers the second TXFQ element also during scan. If the result of the scan is in such a way that the contents of PCB is also TXFQ element

with higher priority than the WCB contents, then PCB contents are not allowed to be copied into WCB unless the content in WCB becomes invalid.ie. from a successful transmission or message drop.
For fetching the headers from TXPQ and TXFQ, TX_SCAN process performs read from LMEM. In case of TXPQ message read the address is calculated from the slot occupied status, the start address of TXPQ and the size of each slot. In case of TXFQ the read address issued for the first time is same as the TXFQ_LMEM_SA.ADD and from the next message onwards the address is TXFQ_RPTR.RPTR_ADD+4.

The worst-case time to fetch all the headers depend on the total number of PQ slots defined and the setting of the register bit TXPQ_CFG.TX_MSG_SEQE.

The processing time for one TX -Scan run can be defined as:
TX-Scan processing time (us) = Lr * (1 + Nbpqs(T0)+ Nbpqs(T1)) * (1/CLK (MHz) where 1 = Number of TX FIFO Queues, Nbpqs(T0) = T0 fetch from Number of TX Priority Queue slots active, Nbpqs(T1) = T1 fetch from Number of TX Priority Queue slots active, Lr = read latency from LMEM defined in number of HOST CLK clock cycles.Considering a maximum of 32 TX Priority Queue slots running at the same time and TX_MSG_SEQE=1, this leads to (considering the previous formula): Max TX SCAN duration (us) = Lr * 65 * (1/CLK (MHz).

As an example, HOST CLK = 160MHz, Lr = 10 cycles leads to a Max delay TX message selection equal to 4.06 us. It is important to note that, in case of a TX message already in transmission, the newer highest priority message will have to wait for the current one to finish. Thus, the overall delay to have this message on the CAN bus may change according to the CAN protocol, the payload data size and bit rate.

1.5.6.6.2 Interfaces

TX Handler module has an LMEM interface to issue LMEM read write instructions. The TX_MSG interface interacts with PRT module. Other miscellaneous signals include CREG inputs and outputs, VBM inputs and outputs and interrupts.

1) TX_MSG PRT interface

Once the highest priority message is identified by TX-SCAN process, TX Handler starts the transmission to PRT via TX_MSG interface. The TX-SCAN output provides the address of the message and the header T0. TX Handler takes the address and reads the header T1 of the message from LMEM. The words T0 and T1 are provided to the TX_MSG interface of the PRT for arbitration. Based on the responses from PRT, further read access to LMEM is issued. More details on TX_MSG PRT interface is available in section 1.7 MH-PRT interface.

2) LMEM interface

TX Handler issues read and write requests to LMEM via this interface. Read accesses are done for getting TXFQ and TXPQ messages for transmission and write access is performed for storing elements into TX Event FIFO Queue. The signals at LMEM interface are:
TXH_LMEM_ADDR - 16 bits LMEM address (Top LMEM interface address width is 18bits. Lower two bits are always 0 for word aligned addressed and hence only 16bits are considered in TX LMEM interface)
TXH_LMEM_WDATA - 32 bit write data
TXH_LMEM_RDATA - 32 bit read data
TXH_LMEM_CS - Chip Select; 1 - active transaction, 0 - inactive
TXH_LMEM_WE - Write Enable; 0 - read, 1 - write

3) Miscellaneous signals

TX Handler functionality in both FMM and CTM is explained in the sections below.

1.5.6.6.3 FMM Description

1) TX-SCAN

TX-SCAN for FMM is explained in the section 1.5.6.6.1.

2) TX Arbiter

TX Arbiter module arbitrates between LMEM access requests from TX-SCAN logic and TX queue and message handler logic. TX SCAN places read requests to LMEM for fetching the headers for TX SCAN process. TX queue and message

handler logic places read and write requests to LMEM based on the state it is in. To access messages from TXFQ and TXPQ, read is issued and for storing the elements into TX Event FIFO Queue, write is issued. TX arbiter checks the incoming requests and performs a dynamic cyclic arbitration process. In the first cycle of arbitration, TX Scan request is checked first and is granted access to LMEM if active. If TX queue and message handler logic request is also active in the same cycle, it will be granted access once the TX scan request is completed.

In the next arbitration cycle, if both requests are active, then TX queue and message handler logic request gets priority and is served first. Upon completion of this request, TX Scan request will be served. This mechanism is like the LMEM controller arbitration logic. Once a request is selected by the arbiter, the corresponding memory signals are placed at the LMEM interface till ready from LMEM is high. The ready signal from LMEM is treated as the grant signal to the requesting modules. Ready is passed to the requesting module to indicate that the requested transfer is complete. There are no internal buffers in the arbiter. The requesting modules must hold the request till it is granted.

3) Data Flow
MH and PRT must be enabled for TX handler to start functioning. There should be at least one successfully enqueued message to be transmitted in TXFQ or TXPQ for TX SCAN to start and to proceed with transmission at PRT. In case of TXPQ, this is known from the TXPQ_STS0.SLOT_OCC and for TXFQ, this is known from the TXFQ_STS.FILL_LEVEL.The words of a TX message are represented as T0, T1...upto Tn where Tn is the last word.

a) TX Handler starts with TX scan process if there are enqueued messages in TXFQ and /or TXPQ. Based on the fill level value in the register **TXFQ_STS.FILL_LEVEL**, TX handler identifies if there is a message pending for transmission in TXFQ. In case of TXPQ, the slot occupied register **TXPQ_STS0.SLOT_OCC** is used for identifying the number of messages in TXPQ. If TXPQ does not have any message, automatically the first element enqueued into TXFQ gets the priority and is identified as the winning candidate. If there are messages in TXPQ, then scan is done to identify the highest priority message. After the scan is finished, the highest priority message is identified and the contents of WCB are given along with valid signal to TX queue and message handler logic.

b) Once valid is received from TX SCAN logic, TX queue and message handling logic reads and the contents of WCB to identify the address from where message should be fetched. Also the lock signal is set high to TX scan logic to indicate that the WCB message is under transmission and should not be modified by the scan. T0 of the message is already fetched and available as part of WCB contents. T1 is fetched from either TXFQ or TXPQ based on the address and stored into a local buffer. This is needed for forming the TEQE element to be stored into TX event fifo queue. From the DLC in T1, the number of words of payload to be fetched and transmitted is calculated.

First T0 is given to TX_MSG.WDC channel and PRT gives the response R0 back on TX_MSG.WRC channel.
T1 can placed immediately after T0 is accepted by PRT (**TX_MSG_WDC.TX_MSGWREADY**=1). And T2 fetch can be initiated by the handler. No need to wait for the response R0 to come in this case. One outstanding transaction is supported at PRT interface.

After T1 is placed, TX handler waits for the response R1 from PRT. Depending on the response from PRT, the next action is taken.
If R1=Ok, that means arbitration is won and the message can be continued.

c) After T1 is fetched, T2 fetch is parallelly initiated by the TX queue and message handling logic without waiting for the response from PRT. If it's a CAN XL message, T2 is also buffered locally for forming TEQE element later. T2 is issued to PRT and the transfer is complete when PRT is ready to accept it. This is repeated for the remaining words of the message till the last word is fetched and transmitted.

d) PRT issues timestamp values after the completion of payload and this is stored into TEQE element.
Note that TEQE for a message will be created only if EFC bit in the TX message header is set. If this bit is not set, TEQE is not formed and stored into TX Event FIFO queue. Refer section 1.5.6.3 for TEQE format.
According to the response from PRT, several actions are taken:

- RESTART: The current TX message will be resent according to the setting defined into the **MH_CFG.MAX_RETRANS**[2:0] bit field. The assigned TX FIFO Queue message or TX Priority Queue slot is aborted from an MH point of view, but the PRT will complete its current frame for resynchronization. The current TX message will still be considered for the next TX-Scan. In case it has not the highest priority it will be replaced.
- HFI (Header Format Invalid): The assigned TX FIFO Queue message is skipped, or the TX Priority Queue slot is set as inactive. The TX message won't be considered in the following TX-SCAN run.

- USOS (Unexpected Start Of Sequence): The MH and PRT are no more synchronized. The MH is stopped and the interrupt *DP_SEQ_ERR* is triggered to the system to notify the issue.
- DU (Data Underrun): The assigned TX FIFO Queue message or Priority Queue slot is discarded from an MH point of view, but the PRT will complete the transfer with a CRC error, to invalidate the current CAN frame. Despite the current TX message is discarded by the MH, it will still be considered in the following TX-SCAN runs and so can be resent according to the setting defined into the **MH_CFG.MAX_RETRANS**[2:0] bit field. In case it has not the highest priority it will be replaced.

Note: For CAN XL messages, T2 byte need to be reshuffled before being provided to the TX_MSG interface. Only the T2 word is managed this way and is provide to the TX_MSG data bus as defined below:
T2 [7:0] => TX_MSG_DATA [31:24]
T2 [15:8] => TX_MSG_DATA [23:16]
T2 [23:16] => TX_MSG_DATA [15:8]
T2 [31:24] => TX_MSG_DATA [7:0]

For T0 and T1 words it would be a one-to-one mapping as defined below:
T1 [31:0] => TX_MSG_DATA [31:0]
T0 [31:0] => TX_MSG_DATA [31:0]
The pulse interrupt *FUNC_TEFQ_ENQ_IRQ* is raised to IRC by TX handler to indicate a successful enqueue.

The following registers are updated by TX handler after the completion of a transmission.

1. If the transfer is done for a TXQE in TXFQ, then **TXFQ_STS.FILL_LEVEL** is decremented by one and **TXFQ_RPTR.RPTR_ADD** is updated with the location of the last word of the transmitted TXQE. If the transmission is not successful, these registers are not updated, and they hold the previous values.
2. if the transfer is done for a TXQE in TXPQ, then the corresponding slot occupied bit in **TXPQ_STS0.SLOT_OCC** register is reset, and the fill level is decremented by one in **TXPQ_STS1.FILL_LEVEL** register. If the transmission is not successful, these registers are not updated, and they hold the previous values.
3. If an element is enqueued successfully into TX event FIFO, the **TEFQ_STS.TEFQ_FULL** is set if the TEFQ is full, **TEFQ_STS.FILL_LEVEL** is incremented by one and **TEFQ_WPTR.WPTR_ADD** is updated with the location corresponding to the last word of the element just enqueued. In case if the transmission is not successful, if EFC bit is enabled in the TX header, then TEQE is created for that message and the status of transmission is marked in word E1 for that message. In case if TEFQ is full while trying to enqueue, the TEQE is dropped and the interrupt ERR_STS_TEFQ_DROP_IRQ is triggered to IRC.

1.5.6.6.4 CTM Description

In CTM mode, the payload size of a message that can be stored into TXFQ and TXPQ is limited to 64 bytes.

Maximum supported message size of TXQE in TXFQ and TXPQ = header + 64 bytes payload.

This means that an entire CAN CC and CAN FD message can be stored as TXQE in CTM into TXFQ and TXPQ, since the maximum payload supported for CAN CC is 8 bytes and CAN FD is 64bytes. For a CAN XL TXQE with payload greater than 64bytes, up to 64bytes of the message can be stored initially into TXFQ or TXPQ. If this message wins arbitration on the bus during transmission, then XS_CAN IP raises a request to the CTDMA to provide the remaining payload as 64byte blocks. This 64byte block is copied into cut through buffers (CTB0 and CTB1) in an alternating manner.i.e. first 64byte block goes to for e.g., CTB0, the next 64byte block goes into CTB1 and is repeated till the entire message is transferred. Detailed explanation on how this is done is given in section Data flow and Descriptor handling.

1) TX-SCAN
TX-Scan process is the same in FMM and CTM, refer section 1.5.6.6.1 TX-Scan for details.

2) TX Arbiter
TX arbitration is the same in FMM and CTM. Refer section 1.5.6.6.2 TX Arbiter for details.

3) Data Flow and Descriptor Handling
The data flow is the same in CTM and FMM in case of CAN CC and CAN FD and CAN XL with maximum payload of 64bytes. Refer to section 1.5.6.6.2 for details. The functionality of TX Handler in CTM differs from FMM only in case of a CAN XL message with payload greater than 64bytes. This is explained below.

1. TX scan has identified a CAN XL message from TXFQ or TXPQ as the highest priority for transmission.
2. TX queue and message handler process takes the address from WCB and issues read of the header T1.
3. Once T1 is received, DLC is read and payload length is calculated. At the same time, headers T0 and T1 are provided to PRT for starting the transmission. T2 fetch is initiated and kept ready by TX Handler. Note that for a can XL message in CTM mode, T0, T1 and T2 are the headers and T3 is the SMEM pointer i.e. the address in SMEM where the first word(T0) of the message is stored.
4. If payload is greater than 64bytes, then TX handler recognizes that the corresponding queue (TXFQ or TXPQ) contains up to 64bytes of payload and the remaining payload needs to be fetched from SMEM.
5. TX handler requests for the next 64byte block of the payload from SMEM under the following conditions
   a. Only if bus arbitration on the CAN bus is won for this message
   b. There is at least one CTB which is free.
   c. There is not pending block copy request active at VBM (either from RX handler or TX handler)

CTM_TX_BC_REQ pulse interrupt is raised by the TX Handler and indication is given to VBM to assert the CTB write signal *CTM_TX_WREQ* at the output. The cut through descriptors are updated with the corresponding source and destination address values. The addresses written into the cut through descriptors are 32bit values.

CTM_DESC_SRC = SMEM address calculated from SMEM pointer (word T3) and the number of block copies done for this message already.

CTM_DESC_DEST = virtual address for CTB.

Refer section 1.5.7 for more information on TX Cut Through Descriptors and Block Copy.

The pulse interrupt *CTB_BC_TX_REQ* is also raised to the IRC. CTDMA reads the descriptor signals to perform block copy. After a block copy is completed, CTDMA gives the response *CTM_RESP* to XS_CAN IP, active for one host clock cycle. Value 1 indicates OK response and 2 indicates ERROR response. 0 and 3 are reserved. If *CTM_RESP*=2, then the message under transmission is aborted and block copy transfer is terminated. Note that if XS_CAN itself aborts the transmission before CTM_RESP is received, then CTM_BC_ERR output pulse is asserted. CTDMA will not send any response in this case. XS_CAN IP can move to the next message transmission or reception.

Block copy is always done in 64byte size i.e CTDMA always reads 64bytes from SMEM to be transferred to CTB in LMEM.If the size of the payload is not a multiple of 64, the last block accessed from SMEM may overlap with the next message's data. VBM enqueues the entire 64bytes into the buffer even if it consists of invalid data. The first block copy data is stored into the available CTB. If both CTBs are free, CTB0 is used first and then CTB1. There can be a case where the previous operation performed by the IP was reception of a message in cut through mode and both the cut through buffers still has data left to be read by the host. In this case TX handler cannot issue a tx block copy request for the CAN XL TX message even though it has won arbitration on the CAN bus.

CTDMA must finish the following steps before the first 64 byte gets transmitted from the TXFQ or TXPQ

- Finish pending RX block copy request for the remaining received data
- Perform the TX block copy for the CAN XL message which is undergoing transmission.

6) While TX block copy is ongoing at VBM, the remaining words from the available 64bytes of the message are fetched from the corresponding queue in LMEM and sent to TX_MSG interface for PRT to continue the transmission on the CAN bus.

7) The TX block copy transfer by host at VBM and the TXQE fetch from LMEM for transmission at PRT happen in parallel. Once the first 64byte payload is transferred to PRT, the remaining payload is fetched from the first enqueued CTB. Note that TX handler fetches only the valid data from the 64byte block in the CTB. This is continued till the entire message gets transmitted at PRT.

8) PRT issues timestamp values after the completion of payload and this is stored into TEQE element.

Note that TEQE for a message will be created only if EFC bit in the TX message header is set. If this bit is not set, TEQE is not formed and stored into TX Event FIFO Queue. Refer section 1.5.6.3 for TEQE format.

According to the response from PRT, several actions are taken:

1. RESTART: The current TX message will be resent according to the setting defined into the **MH_CFG.MAX_RETRANS**[2:0] bit field. The assigned TX FIFO Queue message or TX Priority Queue slot is aborted from an MH point of view, but the PRT will complete its current frame for resynchronization. The current TX message will still be considered for the next TX-Scan. In case it has not the highest priority it will be replaced.
2. HFI (Header Format Invalid): The assigned TX FIFO Queue message is skipped, or the TX Priority Queue slot is set as inactive. The TX message won't be considered in the following TX-SCAN run.
3. USOS (Unexpected Start of Sequence): The MH and PRT are no more synchronized. The MH is stopped and the interrupt *DP_SEQ_ERR* is triggered to the system to notify the issue.
4. DU (Data Underrun): The assigned TX FIFO Queue message or Priority Queue slot is discarded from an MH point of view, but the PRT will complete the transfer with a CRC error, to invalidate the current CAN frame. Despite the current TX message is discarded by the MH, it will still be considered in the following TX-SCAN runs and so can be resent according to the setting defined into the **MH_CFG.MAX_RETRANS**[2:0] bit field. In case it has not the highest priority it will be replaced.

Note: For CAN XL messages, T2 byte need to be reshuffled before being provided to the TX_MSG interface. Only the T2 word is managed this way and is provide to the TX_MSG data bus as defined below:
T2 [7:0] => TX_MSG_DATA [31:24]
T2 [15:8] => TX_MSG_DATA [23:16]
T2 [23:16] => TX_MSG_DATA [15:8]
T2 [31:24] => TX_MSG_DATA [7:0]

For T0 and T1 words it would be a one-to-one mapping as defined below:
T1 [31:0] => TX_MSG_DATA [31:0]
T0 [31:0] => TX_MSG_DATA [31:0]

The pulse interrupt *FUNC_TEFQ_ENQ_IRQ* is raised to IRC by TX handler to indicate a successful enqueue.

The following registers are updated by TX handler after the completion of a transmission.

1. If the transfer is done for a TXQE in TXFQ, then **TXFQ_STS.FILL_LEVEL** is decremented by one and **TXFQ_RPTR.RPTR_ADD** is updated with the location of the last word of the transmitted TXQE. If the transmission is not successful, these registers are not updated, and they hold the previous values.
2. if the transfer is done for a TXQE in TXPQ, then the corresponding slot occupied bit in **TXPQ_STS0.SLOT_OCC** register is reset, and the fill level is decremented by one in **TXPQ_STS1.FILL_LEVEL** register. If the transmission is not successful, these registers are not updated, and they hold the previous values.
3. If an element is enqueued successfully into TX event FIFO, the **TEFQ_STS.TEFQ_FULL** is set if the TEFQ is full, **TEFQ_STS.FILL_LEVEL** is incremented by one and **TEFQ_WPTR.WPTR_ADD** is updated with the location corresponding to the last word of the element just enqueued. In case if the transmission is not successful, if EFC bit is enabled in the TX header, then TEQE is created for that message and the status of transmission is marked in word E1 for that message. If TEFQ is already full and there is no space for a new TEQE to be stored then TX handler raises the interrupt ERR_STS_TEFQ_DROP_IRQ.

1.5.6.7 RX Message Handler

RX handler manages the storing of received messages in RXFQ0 and RXFQ1 in FMM and in CTBs in CTM.

In FMM, all the received messages are stored into either RXFQ0 or RXFQ1 based on the configuration or result of filtering. Note that the received words are initially stored into both RXFQ0 and RXFQ1 if both fifos are enabled, since the target FIFO is not known until filtering gets finished. Once filtering results are known with the target FIFO, the write pointer corresponding to that FIFO is updated and the other FIFO contents are discarded. Host gets the information about availability of messages in RXFQs via interrupts and read-write pointers. If one RXFQ is disabled and the other is enabled, then the message storage starts for the enabled RXFQ. If the filtering result gives the disabled RXFQ as target FQ, then this message is discarded and the corresponding ERR_STS_RXFQx_DROP_IRQ is triggered.

In CTM, all the received messages are stored into the two cut through buffers. RXFQs are located in system memory (SMEM) and RXFQs in LMEM are disabled in CTM. The messages buffered in CTBs are transferred to SMEM RXFQ in blocks of 64bytes. The target RXFQ in SMEM must be known before one CTB block gets full.i.e before one 64byte block is ready for copy. More details are given in the later sections.

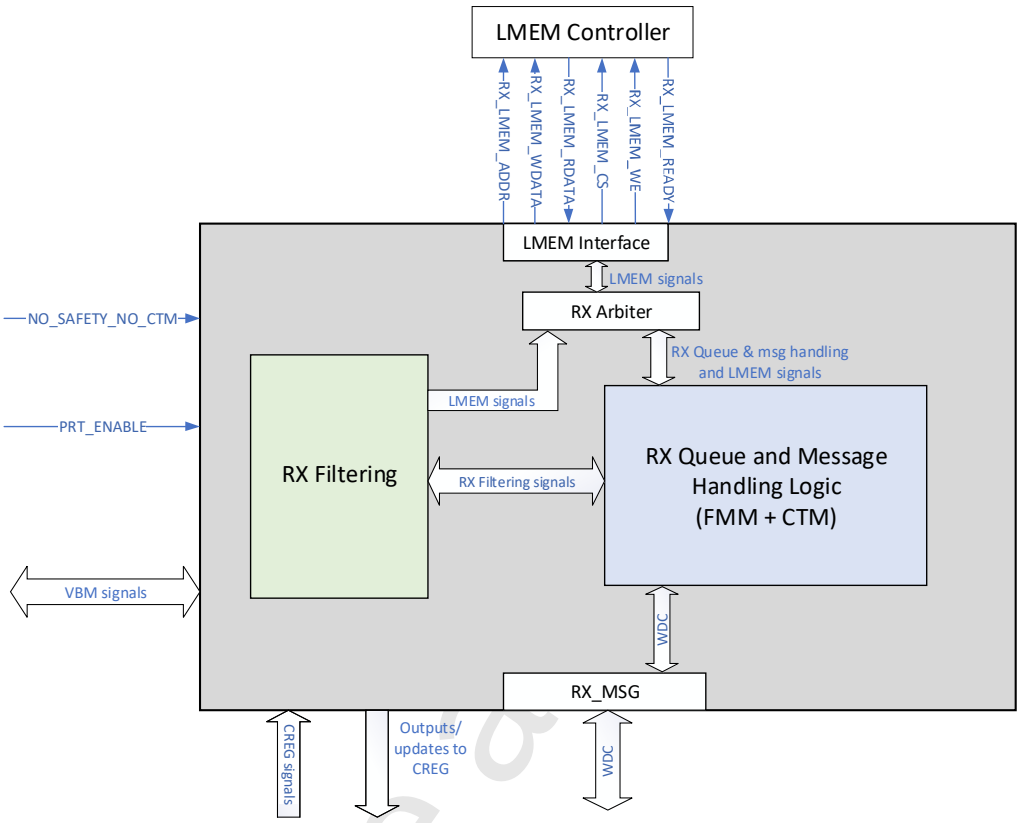


Figure 16. RXH Block Diagram

1.5.6.7.1 Interfaces

RX handler module has an LMEM interface to issue LMEM read and write instructions. The RX_MSG interface interacts with PRT module. Other miscellaneous signals include CREG inputs and outputs, VBM inputs and outputs and interrupts.

1) RX_MSG PRT Interface

The messages received at PRT are sent to MH via RX_MSG interface. There is only one channel at this interface, Write Data Channel (WDC).More details on RX_MSG PRT interface is available in section 1.7 MH-PRT interface.

2) LMEM Interface

RX Handler issues write requests to LMEM for storing the message and read requests for filtering via this interface. The signals at LMEM interface are
RXH_MEM_ADDR - 16 bits LMEM address
RXH_MEM_WDATA - 32 bits write data
RXH_MEM_RDATA - 32 bit read data
RXH_MEM_CS - Chip select; 1 - active transaction, 0 - inactive
RXH_MEM_WE - write enable; 0 - read, 1 - write

1.5.6.7.2 FMM Description

1) RX Filtering

Refer section 1.5.6.7.4 for RX Filtering section

2) RX Arbiter

RX arbiter processes two requests:

1. Read requests from RX Filtering process to LMEM for fetching RX filter elements.
2. Write requests from RX queue and message handling process for storing the message into RXFQs.

RX arbiter checks the incoming requests and performs a dynamic cyclic arbitration process. In the first cycle of arbitration, RX filtering request is checked first and is granted access to LMEM if active. If RX queue and message handler logic request is also active in the same cycle, it will be granted access once the RX filter request is completed.

In the next arbitration cycle, if both requests are active, then RX queue and message handler logic request gets priority and is served first i.e RXFQ0 write is performed followed by RXFQ1 write. Upon completion of this request, RX filter request will be served. This mechanism is like the LMEM controller arbitration logic. Once a request is selected by the arbiter, the corresponding memory signals are placed at the LMEM interface till ready from LMEM is high. The ready signal from LMEM is treated as the grant signal to the requesting modules. Ready is passed to the requesting module to indicate that the requested transfer is complete. There are no internal buffers in the arbiter. The requesting modules must hold the request till it is granted.

3) Data Flow

This section explains the data flow for the reception of a message M. The words of the message are named as R0,R1…upto Rn for ease of explanation. In FMM(**MH_CFG.CTME**=0), the RX handler starts accepting messages from PRT under the following conditions:

1. At least one word space in RXFQ0 or RXFQ1.
2. MH and PRT must be enabled.
3. At least one RXFQ must be enabled in CREG.

If any of the above condition is not true, then RX Handler still accepts the words from PRT, but are not stored. The message gets discarded and both RXFQ0_drop, RXFQ1_drop interrupts are raised.

After MH is enabled, RXFQx_RPTR.ADD and RXFQx_WPTR.ADD registers are updated with the respective start addresses from RXFQx_SA. This is done by RX Handler.

The message reception is successfully completed when the last two words of the time stamp (TS0 and TS1) are received at RX_MSG interface and stored into LMEM.

Refer to chapter 1.7.3 RX_MSG interface for details on the signals and handshaking.

Data flow in RX handler is explained in the following steps:

1. RX handler waits for start of sequence (sos) code from PRT at *RX_MSG_WUSER* signal. The first received word R0 is buffered locally. R0 is sent to RX filtering process to start filtering and stored into the enabled RXFQ. If both RXFQs are enabled, and is not full, then message is stored into both since target FIFO is not known yet. Write pointers for both FIFO queues are maintained locally in RX queue and message handling logic. Both these pointers are incremented every time a word is written into the queues. The write address issued to LMEM controller is calculated from the read pointer corresponding to the RXFQ.i.e **RXFQx_RPTR.ADD+4**. If there is no space in any of the enabled fifos then the received message is dropped. Corresponding interrupt is also raised i.e. ERR_STS_RXFQ0_DROP_IRQ or ERR_STS_RXFQ1_DROP_IRQ or both if filtering results are not known
2. The second received word R1 is stored locally but is not needed for filtering. The total length of the message is calculated from the DLC in R1 by the RX queue and message handling logic. This count is used for issuing the no of write transactions to RXFQs. R1 is stored twice into RXFQs, because the filtering results also need to be updated into the header R1 (FAB, FM etc.).
3. The next word received R2 is also stored locally and can be used by RX filtering process if any of the filter element configuration requires a comparison with R2. Write access is issued to RXFQ0 and RXFQ1 if both are enabled, else to only the enabled RXFQ, for storing R2.
4. The received words are written into the enabled queues till the result of filtering is known. As soon as the filtering results are known, the following actions are taken:
   a. Stop updating the local write pointer for the non-target FIFO and reset it back to the previous RXFQ write pointer address from CREG.

b. Continue updating the local write pointer of the target RXFQ till last word of timestamp is received.

5. If the filtering results are not known before the first word(R0) of the next received message, then the filtering process is treated as aborted without any result. In this case there are two possibilities:
   a. If the **RX_FILTER_CFG.AFAB** bit is set by the user, the message gets stored into the default FIFO set in the register **RX_FILTER_CFG.DEFAULT_FIFO**.i.e the write pointer in CREG of the default FIFO is updated. Header R1 is also updated with the FAB=1 status. When R0 of the next message is sent by PRT to MH, RX Handler does not give ready to PRT till header R1 of the aborted message is updated and written into the default FIFO. For short messages, this may result in data overflow condition in PRT if LMEM is being accessed with a burst access. The interrupt *SAFETY_RX_FILTER_ERR_IRQ* is triggered by the MH.
   b. If the **RX_FILTER_CFG.AFAB** bit is not set by the user, the current message for which filtering was ongoing, gets discarded and write pointers in CREG are not updated. First word R0 of the next message is accepted immediately.
6. When reception and filtering process is completed successfully, the last word received TS1 gets stored into the target RXFQ. The second word R1 header is updated with the results below from the filtering and stored into the target RXFQ. Refer 1.5.6.2 for RX message header details.
   a. The message sequence number (SN) - Indicates the ordering of the message in the queue. SN value is Incremented for every new message that gets enqueued and wraps back to 0 after value 15.
   b. Filter abort status (FAB)- If filtering could not be finished in time for this message, this bit is set in the header R1.
   c. Filter Match status (FM)- If a filter match has been found, then FM bit is set in the header R1.
   d. Filter ID number (FIDX)- The ID of the filter element which matched. If FM bit is 0, then the FIDX value is invalid.

If filtering is completed without a match and RX_FILTER_CFG.ANMF=0, then the current message for which filtering was going on, is dropped. No interrupt is triggered in this case. If filtering is completed without a match and RX_FILTER_CFG.ANMF=1, the message gets stored into the default fifo set in the register RX_FILTER_CFG.DEFAULT_FIFO.

The write pointer of the target RXFQ in CREG (**RXFQx_WPTR.ADD**) is updated by RX handler. It points to the location of the TS1. The fill level of the target RXFQ is incremented by one and updated into the register **RXFQ_STS**. Upto 255 messages can be stored in each RXFQ and once fill level becomes 255, no more messages will be stored into that fifo. Messages are not accepted from PRT which results in data overflow in PRT. A pulse interrupt on RXFQ0_ENQ or RXFQ1_ENQ is generated by RX handler to indicate successful enqueue. The received messages are read by the host via VBM AHB interface by issuing the virtual buffer addresses corresponding to the RXFQ. Refer VBM section 1.5.6.5 for details on dequeuing and the registers updated. If the RX FIFOs become empty with read and write pointers not pointing to the start address, then the next received message will be enqueued at the next address i.e.RXFQx_WPTR.ADD+4. But if the RX FIFOs become empty with read and write pointers pointing to the start addresses, then the next received message will be enqueued at the RXFQx_WPTR.ADD itself which is same as the start address.

Note that each message stored in the RXFQs is called an RX Queue element (RXQE). RXQEs are stored back-to-back into an RXFQ and wrapping around is possible in FMM. i.e., if there is not enough space to accommodate a full message from the write pointer location till the RXFQ end address, some part of the message can be stored at the bottom and the remaining from the start of the FIFO if there is space left.

1.5.6.7.3 CTM Description

This section explains the RX handler functions in Cut Through Mode. CTDMA Add-on module is needed for the IP to work in CTM mode.

Note: If CTM is enabled, RXFQs are in SMEM and not in LMEM. Host must enable the required RXFQs using the configuration bits MH_CFG.RXFQ0E and MH_CFG.RXFQ1E. The registers used to store the start address, end address, read pointer, write pointer for the RXFQs hold the addresses in SMEM.

The messages received are stored into the two cut through buffers (CTB0 and CTB1) as 64 bytes blocks.

When the last word of timestamp TS1 is written into the buffer and the full buffer is still not occupied fully, the remaining space can contain old or invalid data. For example, if the payload of the incoming CAN XL message is 200bytes, and assuming CTB0 and CTB1 are free initially, first 3 words of header(12bytes) +52bytes payload get stored into CTB0. Next 64bytes gets stored into CTB1. Next 64bytes get stored into CTB0 and the remaining 8bytes get stored into CTB0. RX handler does not store anything to the remaining 56byte space. It can contain old/invalid data from the previous block. The

CTB read is always done as 64byte blocks (burst length=16) which means there could be invalid data in it. But the write pointer for the target RXFQ is updated to the address that contains the last valid data.

1) RX Filtering

RX filtering process is the same in FMM and CTM. Refer section 1.5.6.7.4 RX Filter for details.

2) RX Arbiter

RX Arbiter logic is the same in FMM and CTM. Refer section 1.5.6.7.2 RX Arbiter for details.

3) Data Flow

In CTM, at least one cut through buffer should be free for RX handler to start accepting the received messages from PRT. Until then the RX queue and message handling logic will not give ready to PRT at RX_MSG interface.

This section explains the data flow for the reception of a message M. The words of the message are named as M0,M1…upto Mn. In CTM(**MH_CFG.CTME**=1), the RX handler starts accepting messages from PRT under the following conditions.

1. At least one CTB is free.
2. MH and PRT must be enabled.

If any of the above condition is not true, then MH will not assert **RX_MSG.RX_MSG_WREADY** to PRT indicating MH is not ready to accept data.
After MH is enabled, RXFQx_RPTR.ADD and RXFQx_WPTR.ADD registers are updated with the respective start addresses from RXFQx_SA. This is done by RX Handler. Note that RX Handler updates the start address to RXFQx_RPTR.ADD only once when MH is enabled. After that, host must update the address accordingly every time when a message is read out from the SMEM.

The message reception is successfully completed when the last two words of the time stamp (TS0 and TS1) are received at RX_MSG interface and stored into CTB.

The data flow for CTM description is explained in the following steps:

1. RX handler waits for start of sequence (sos) code from PRT at *RX_MSG_WUSER* signal. The first received word R0 is buffered locally and sent to RX filtering process to start filtering. R0 is also stored into the cut through buffer which is free at the same time. The start address for storing the word R0 to CTB is either **CTB_LMEM.SA**(CTB0) or **CTB_LMEM.SA+64**bytes (CTB1) based on the availability. Let us take CTB0 in this case.
2. The second received word R1 is stored locally but is not needed for filtering. The total length of the message is calculated from the DLC in R1 by the RX queue and message handling logic. This count is used for issuing the number of write transactions to CTB0. R1 is not stored into CTB0 now, because the filtering results also need to be updated into the header R1 (FAB, FM etc.). R1 write is issued after one CTB0 gets full.
3. The next word received R2 is also stored locally before storing to CTB0 and can be used by RX filtering process if any of the filter element configuration requires a comparison with R2.
4. In CTM, it is expected that the result of filtering is known before the write request comes for the next buffer, i.e CTB1. As soon as the filtering results are known following actions are taken: (RX message header R1 is updated with the following fields and kept ready.
   a. The message sequence number (SN) - Indicates the ordering of the message in the queue. SN value is Incremented for every new message that gets enqueued and wraps back to 0 after value 15.
   b. Filter abort status (FAB) - If filtering could not be finished in time for this message, this bit is set.
   c. Filter Match status (FM) - If a filter match has been found, then FM bit is set
   d. Filter ID number (FIDX) - The ID of the filter element which matched. If FM bit is 0, then the FIDX value is invalid.
5. If the filtering results are not known before the write request comes for the next cut through buffer, then the filtering process is treated as aborted without any result. In this case there are two possibilities:

a. If the **RX_FILTER_CFG.AFAB** bit is set by the user, the message gets stored into the default FIFO set in the register **RX_FILTER_CFG.DEFAULT_FIFO**.i.e the write pointer in CREG of the default FIFO is updated. Header R1 is also updated with the FAB=1 status. When next word is sent by PRT to MH, RX Handler does not give ready to PRT till header R1 of the aborted message is updated and written into the default FIFO. For short messages, this may result in data overflow condition in PRT if LMEM is being accessed with a burst access. The interrupt *SAFETY_RX_FILTER_ERR_IRQ* is triggered by the MH.

b. If the **RX_FILTER_CFG.AFAB** bit is not set by the user, the current message for which filtering was ongoing, gets discarded and write pointers in CREG are not updated. First word R0 of the next message is accepted immediately.

6. When filtering process is completed successfully for the current CTB0 (64 bytes), target RXFQ in SMEM is known to RX handler. RX handler performs the following:

Case 1: Target RX fifo is enabled and check the remaining space in the target RXFQ i.e. check the space between current write pointer and the end address of the FIFO. If the space is enough to fit the entire RX message (in multiples of 64bytes), then proceed to issue block copy request and there is no pending block copy request (TX or RX from the previous block copy), CTB read request (*CTM_RX_RREQ*) is raised by VBM for the CTB block copy. The CT descriptors are updated with the respective source and destination addresses by RX Handler. The addresses written into the cut through descriptors are 32bit values.

CTM_DESC_SRC = virtual address for CTB.

CTM_DESC_DEST = target RXFQ address in SMEM.

Refer section 1.5.7 for more information on Cut Through Descriptors.

The pulse interrupt *CTB_BC_RX_REQ* is also raised to the IRC. CTDMA reads the descriptor signals to perform block copy. CTDMA reads the 64byte block from CTB and performs write to the target RXFQ in SMEM.Block copy transfer must be done in 1 burst access of length 16. After a block copy is completed, response CTM_RESP is given to XS_CAN IP by CTDMA for one host clock cycle.Value 1 indicates ok response and 2 indicates error response. 0 and 3 are reserved. If CTM_RESP=2, then the block copy transfer is terminated and the message under reception is discarded. In this case, data overflow does not happen in PRT. Safety_rx_Abort interrupt is raised for the discarded message The write pointer of the target FIFO **RXFQx_WPTR.ADD** is updated by VBM only after the last block is stored i.e. after completing one full message. Even if the last block consists of invalid data along with the valid data, write pointer is updated with respect to the location where valid data is stored, which is the higher order 32bits of timestamp.

The remaining words are getting stored into the CTB1 while CTDMA reads out the contents from CTB0. If CTB1 gets full and the CTB0 block copy is still ongoing, before updating the CT Descriptors, RX handler waits till the CTB0 copy is finished or there is an overflow condition in the CTB whichever is the earliest. This process of storing into the two cut through buffers in an alternating manner continues till the entire message gets stored including the 64 bit timestamp. There is no need for RX Handler to calculate the space in the target RXFQ before every block copy in this case as it is already understood that the full message can be fit into the FIFO.
Once the time stamp values are also written into the current running CTB, reception is completed at RX Handler. CTDMA reads out the last set of 64byte block from this CTB and if CTM_RESP=1 marks the completion of the reception process for the IP.
In case 1, **RXFQx_WRAP_PTR.SMEM_ADD**is not updated by the RX Handler since there is no need to wrap around and check for space from the beginning. In this scenario, the **RXFQx_WRAP_PTR.SMEM_ADD** content is same as the address in **RXFQx_EA.ADD**.

Case 2: If the entire message cannot fit into the bottom of the target RXFQ, then RX handler checks if the there is at least one block (64 byte) space in the beginning of the RXFQ. To do this check, space between **RXFQx_RPTR.ADD** and **RXFQx_SA.ADD** (start address) is calculated by RX Handler. If there is at least 64byte space, and there is no previous pending block copy requests, RX handler indicates to VBM to issue the CTB service request. In this case, the block is not getting stored based on the current write pointer address in **RXFQx_WPTR.ADD**. The **RXFQx_SA.ADD** serves as temporary write pointer since block gets stored from the beginning. The actual write pointer value is copied into the wrap pointer register of the target RXFQ i.e **RXFQx_WRAP_PTR.SMEM_ADD**. This is the address which holds the last valid data in SMEM RXFQx. The space between **RXFQx_WRAP_PTR.SMEM_ADD** and **RXFQx_EA.ADD** is invalid and does not contain any valid data. This space becomes valid when host keeps reading from SMEM and the **RXFQx_RPTR.ADD** reaches the **RXFQx_WRAP_PTR.SMEM_ADD** address and goes further to the beginning of the FIFO. After this happens, **RXFQx_WRAP_PTR.SMEM_ADD** is set to **RXFQx_EA.ADD** to show that wrap around is not active anymore.

CTB read request (*CTB_RX_RREQ*) is raised by VBM for the CTB block copy. The CT descriptors are updated with the respective source and destination addresses by RX Handler. The addresses written into the cut through descriptors are 32bit values.

CTM_DESC_SRC = virtual address for CTB.

CTM_DESC_DEST = target RXFQ address in SMEM.

Refer section 1.5.7 for more information on Cut Through Descriptors.

The pulse interrupt *CTB_BC_RX_REQ* is also raised to the IRC by RX Handler. CTDMA reads the descriptor signals to to perform block copy. After a block copy is completed, response CTM_RESP is given to XS_CAN IP by CTDMA for one host clock cycle. Value 1 indicates ok response and 2 indicates error response. 0 and 3 are reserved. If CTM_RESP=2, then the block copy transfer is dropped and the message under reception is discarded.

The remaining words are getting stored into the CTB1 while CTDMA reads out the contents from CTB0. If CTB1 gets full and the CTB0 block copy is still ongoing, before updating the CT Descriptors, RX handler waits till the CTB0 copy is finished or there is an overflow condition in the CTB whichever is the earliest. This process of storing into the two cut through buffers in an alternating manner continues till the entire message gets stored including the 64 bit timestamp. Note that, before raising the request for the remaining block copies, RX Handler every time calculates the space in the target RXFQ between the write pointer and the read pointer and check if a block can fit. At any point during calculation, if there is no space for a block to fit, then the target RXFQ is considered full, and the entire message is dropped. The corresponding pulse interrupt either *ERR_STS_RXFQ0_DROP_IRQ* or *ERR_STS_RXFQ1_DROP_IRQ* is raised by RX Handler to IRC.

Once the time stamp values are also written into the current running CTB, reception is completed at RX Handler. CTDMA reads out the last set of 64byte block from this CTB and if CTM_RESP=1 marks the completion of the reception process for the IP.

If both the cases explained above are not possible, then the target RXFQ in SMEM is considered full and the message is dropped. The corresponding pulse interrupt either *ERR_STS_RXFQ0_DROP_IRQ* or *ERR_STS_RXFQ1_DROP_IRQ* is raised by RX Handler to IRC.
Once one full message is stored into target RXFQ in SMEM, the write pointer is updated by the VBM in the corresponding status register **RXFQx_WPTR.ADD**. Even if the last block consists of invalid data along with the valid data, write pointer is updated with respect to the location where valid data is stored, which is the higher order 32bits of timestamp. Once host reads out one full message from either of the RXFQ in SMEM, the read pointer is updated by the host in the corresponding status register **RXFQx_RPTR.ADD**. This read pointer address is used by the RX Handler to calculate the available size in RXFQx before updating the CT descriptors. So the host must ensure that **RXFQx_RPTR.ADD** is updated every time one message is read out by the host. This pointer points to the address of the last valid word of the message which was read out. The fill level of the target RXFQ is incremented by one by VBM and updated into the register **RXFQ_STS**. Once the fill level reaches 255, it is reset and starts from 0 again, which is not the case in FMM. In CTM, the fill level acts as a message counter. If the fill level hits 255, it resets to 0 upon storing a new message and increments thereafter with each new entry.

1.5.6.7.4 RX Filter

RX filtering is done to filter all incoming messages based on their identifiers. Filters can be configured to accept or reject messages with specific IDs, thereby reducing the number of messages that need to be processed by the software.
One Filter Element consists of a filter element configuration (FEC) and its associated one or two Reference-mask Pairs (REFP). Each reference pair consists of a 32bit reference value and a 32 bit mask value.
The registers used for RX filtering are **RX_FILTER_CFG** and **RX_FILTER_LMEM.SA**.

1.5.6.7.4.1 Filter Element Configuration

The IP supports up to 127 Filter Element Configurations. Each FEC is 8 bits in size. The register field **RX_FILTER_CFG.FE_NUM** defines the total number of FEC in LMEM. The register field **RX_FILTER_CFG.ANMF** indicates whether to accept the non matching messages. The register field **RX_FILTER_CFG.AFAB** indicates whether to accept and store messages even if the filtering process is aborted before it gets completed. The register field **RX_FILTER_CFG.DEFAULT_FIFO** indicates the default FIFO to which the messages that are not matching and/or aborted are stored if the bit **RX_FILTER_CFG.ANMF** bit and/or **RX_FILTER_CFG.AFAB** is set. If RX_FILTER_CFG.FE_NUM=0, then messages are accepted into default fifo if RX_FILTER_CFG.ANMF is set to 1.

FEC consists of the following bits:

1) Enable disable bit which tells the MH whether to use that filter element for filtering or not. The XS_CAN IP supports enabling and disabling of the filter element anytime while the IP is in operation.

2) Accept Reject bit to decide whether to accept or reject the frame in case of a match.

3) Interrupt enable bit to generate an interrupt in case of a match. Interrupt is triggered once the match happens.

4) Target FIFO bit to select the FIFO for storage. Target fifo set in FEC must be enabled. Disabled RX FIFO should not be configured as target FIFO.

5) ALERT bit to tag the message as security relevant message. If this bit is set and there is a match, an output signal RX_MSG_ALERT is generated and is reset once the message is enqueued successfully. In case if the message is rejected on match and not enqueued, RX_MSG_ALERT is still raised for one host clock cycle.

6) Number of comparison bit to decide whether to perform 1 or 2 comparisons

7) Word Index (2 bits) to indicate which word to be compared with the reference value-mask.

The table below shows the FEC bits order and their description.

Table 69. Filter Element Configuration

Bit Position	Name	Description
0	EN	If this bit is set, comparison is performed for this filter element. If this bit is not set, the filter element is skipped and the next FEC is fetched.
1	FIFO	This bit indicates the RXFQ to which the message must be stored.. 0 - RXFQ0. 1 - RXFQ1
2	IRQ	0 - Interrupt is not generated on match. 1 - Interrupt is generated on match
3	ALERT	This is a tag bit to identify security incident messages. This bit is also copied into the RX message header when the message is stored into the respective RXFQ.. 0 - RX_MSG_ALERT output is not generated on match. 1 - RX_MSG_ALERT output is generated on match
4	AR	This bit indicates whether the message shall be accepted or rejected on match. 0 - Accept if message matches. 1 - Reject if message matches
5	NC	This bit indicates the number of comparisons to be performed on the received message.. 0 - one comparison. 1 - two comparison
6	WI0	Word Index 0 bit indicates which word of the received message has to be considered for the first comparison. 0 - R0. 1 - R2. Note: If received frame is Classical CAN remote frame or Classical CAN or CAN FD frame with no payload, then WI0=1 is an invalid setting. Current filter element will be skipped and next FEC will be fetched even if NC is set to 1 (two comparisons).
7	WI1	Word Index 0 bit indicates which word of the received message has to be considered for the second comparison. 0 - R0. 1 - R2

1.5.6.7.4.2 Reference and Mask Pair

Each filter element is associated with one or two reference pairs (REFP). Each reference pair shall consist of a 32 bit reference value and a 32 bit mask value.

The list of REFPs shall start at a 4 byte aligned address to the Filter Element Configuration (FEC).All the pairs of reference value and mask are appended after the RX filter elements section in LMEM.

There shall be a fixed association between the FEC and their REFP(s).i.e The FECs shall be stored back to back in continuous way in the LMEM starting at the RX Filter SA. The REFPs shall be stored back to back in continuous way in the LMEM starting at the next word address after the last FEC. The first REFP value mask pair is for the first filter element, 2nd REFP value mask pair is for the second filter element, so on and so forth.If there is only one comparison, only one REFP will stored into LMEM and the next location is allocated to the REFP of the next filter element in the order.

Each FEC has one bit (user-configurable) which indicates the number of comparisons wherein bit 0 indicates 1 comparison and bit value 1 indicates 2 comparisons. When FEC is configured for 2 comparisons and the received message is CAN CC or CAN FD with 0 data bytes, 2nd comparison is not done and the filter does not match, the next filter is fetched.
FEC supports 2 bits (word index WI0 and WI1) to indicate the RX message word (R0 or R2 (for CAN XL)/ RD0 (for CC CAN and CAN FD) or both) used for comparison.
When bit WI0 is 0, it indicates R0 and bit value 1 indicates R2/RD0. Similarly, when bit WI1 is 0, it indicates R0 and bit value 1 indicates R2/RD0. If WI0 or WI1 = 1, then the process of Rx filtering is on hold waiting till R2/RD0 is received.

There are two scenarios, in case of one comparison, the filter match occurs when the REFP value matches with either R0 or R2/RD0 of the RX Message after applying the mask. In 2 comparison, word index plays a role in filter matching. When both comparisons match, WI0 is not equal to WI1, the filter matches; and when one of the comparisons match, WI0 is equal to WI1, the filter matches.

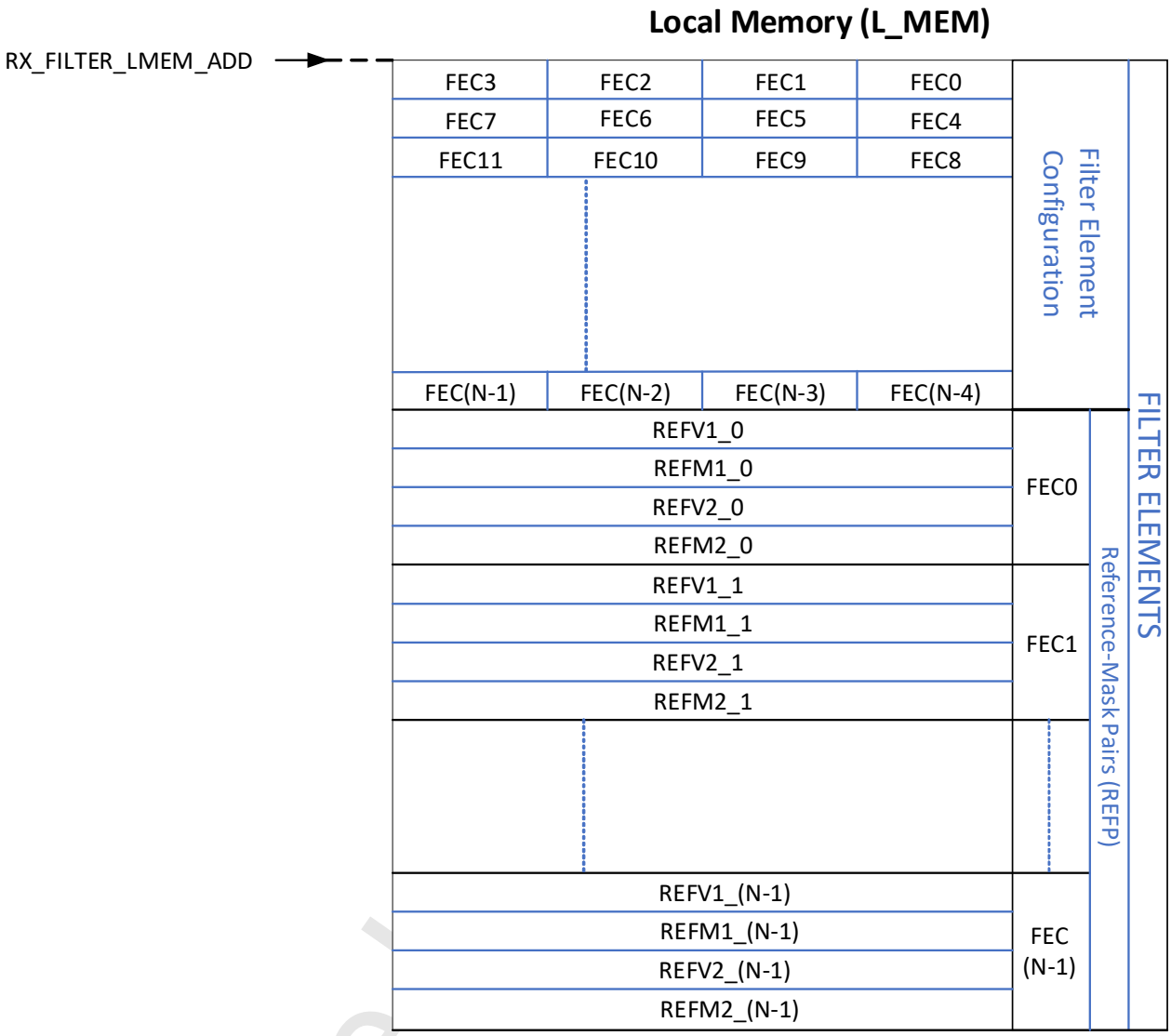


Figure 17. RX Filter Elements in LMEM having 2 comparisons

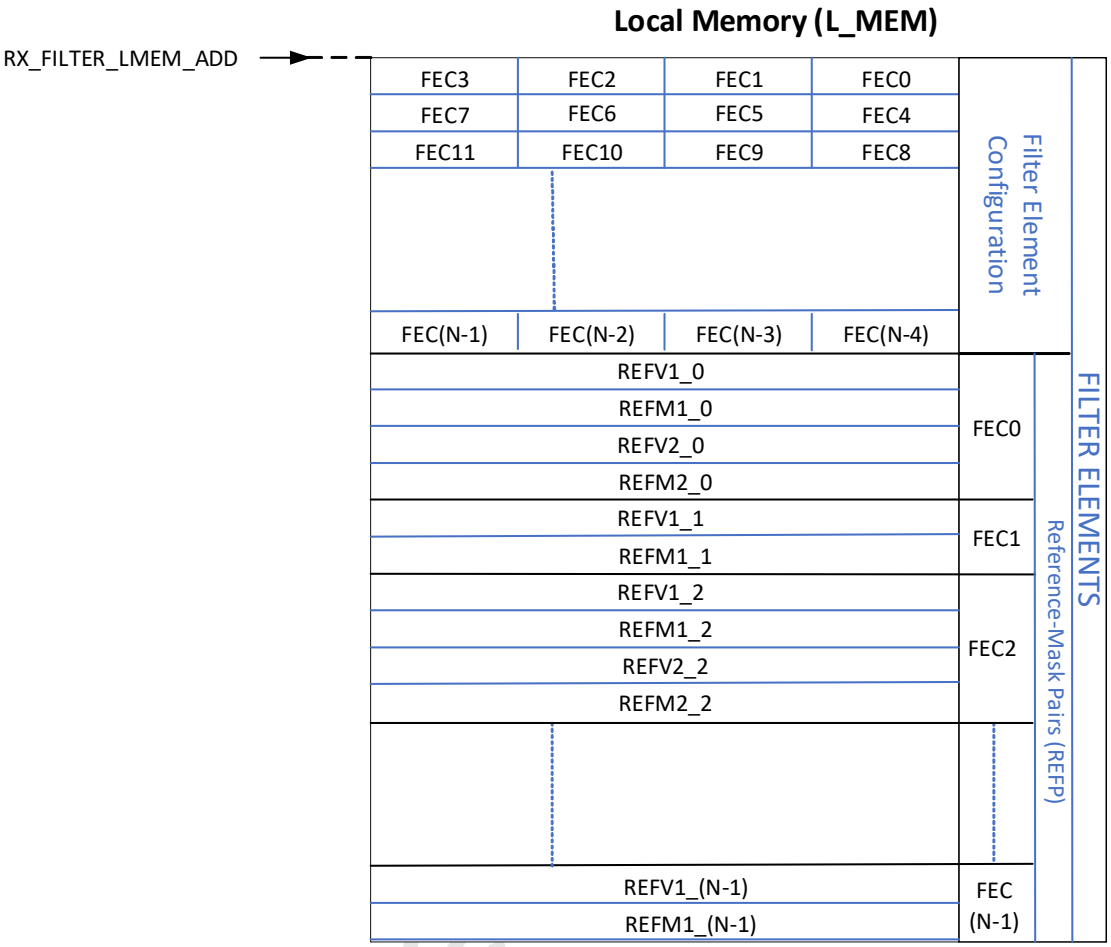


Figure 18. RX Filter Elements in LMEM having both 1 and 2 comparisons

1.5.6.7.4.3 RX Filtering Steps

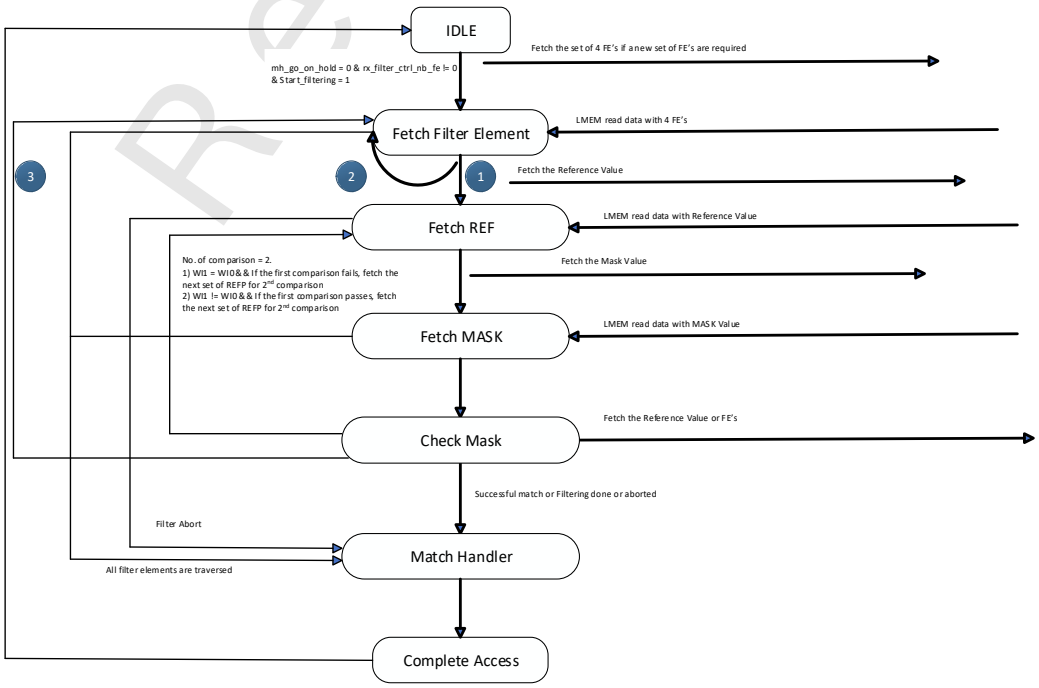


Figure 19. RX Filtering

Each time an RX message is received, the Rx filtering is triggered and will start the following sequence:

1. Fetch the first filter element from LMEM.
2. If the conditions listed below are met, the RX filter element will be discarded, and the next filter element be fetched (if there is one available). In all other cases, go to next step:
   a. If number of comparison is 1, then WI0 = 1 - Remote Frame or empty CC/FD Frame
   b. If number of comparison is 2, then WI0 or WI1 = 1 - Remote Frame or empty CC/FD Frame
   c. If bit 0 of FEC = '0'
3. The following scenarios can occur after applying the mask and comparing with the reference values:
   a. When number of comparison is 1 -> If there is a match, the RX filter looks at the AR bit to identify what to do with the RX message. It would then be accepted or rejected, and the RX filter stops. If there is no match, the RX filter keeps going with the next element, if there is one available, otherwise it stops.
   b. When number of comparison is 2 -> If WI0! = WI1, the match will occur only when both the comparisons are successful, the RX filter looks at the AR bit to identify what to do with the RX message. It would then be accepted or rejected, and the RX filter stops. If there is no match, the RX filter keeps going with the next element, if there is one available, otherwise it stops.
   c. When number of comparison is 2 -> If WI0 = WI1, the match will occur when either of the two comparisons is successful, then the RX filter looks at the AR bit to identify what to do with the RX message. It would then be accepted or rejected, and the RX filter stops. If there is no match, the RX filter keeps going with the next element, if there is one available, otherwise it stops.

The formula to be used for comparison is result = (Ri XOR REFP.value) AND REFP.mask. (Ri can be R0 or R2/RD0 based on the FEC). If result == 0x0 this comparison matches. The bit positions with zeros in the reference pair mask are ignored and the bit positions with ones in the mask are considered for comparison.

RX Filtering process is complete if there is an outcome. It can be either a match occurred or no match occurred after going through the entire list. If the filtering process is not completed on time with a result, then its a filtering abort condition. If the filtering is aborted without an end result, then decisions to store the message received or to discard it depends on the configuration in the register RX_FILTER_CFG.
If RX_FILTER_CFG.AFAB bit is set to 1, the current message in reception is accepted even if there is a filtering abort condition and is stored into the default FIFO configured in RX_FILTER_CFG.DEFAULT_FIFO. Else this message is discarded. After storing the message, RX Handler generates the 1 bit output signal RX_MSG_STORE_SUCC for 1 HOST_CLK cycle.

If RX_FILTER_CFG.ANMF bit is set by the host, then the frames that do not match after the filtering is completed, are also accepted. These frames are stored into the default FIFO configured in RX_FILTER_CFG.DEFAULT_FIFO. FM bit in the RX message header is set to 0 in this case and the FIDX bits in the header are 0 and not considered. If RX_FILTER_CFG.ANMF bit is not set, the messages that do not find a match are discarded.

1.5.6.7.4.4 RX Message Header Updates

Once filtering is completed, the following bits stored are modified in RX Message Header R1:

Table 70. Modified fields of RX Message Header R1

Bit Name	Bit Position	Description
SN	[15:12]	Sequence Number: order in which the message gets stored into the queue
RX_IRQ	11	IRQ bit copied from the filter element configuration which matched for this message
FAB	10	Filter Abort: when set to 1, the RX filtering process for this message ended before completing without a result
ALERT	9	ALERT: When set to 1, the RX message filtered belongs to a group of messages tagged as ALERT
FM	8	Filter Match: When set to 1 one of the filter elements (defined by FIDX[6:0]) has detected a match
FIDX	[6:0]	ID of the filter element that matched

1.5.6.7.4.5 RX Filter Processing Time

For the calculation of RX filter element fetching time, the following assumptions are made.
1.LMEM is not busy and responds immediately to the read requests.
2.LMEM controller is not tending to other requests and grants the RX filter element fetch requests without delay.

With the above assumptions, the read latency to fetch one word of FEC is Lrf, the read latency to fetch the Reference Value -Mask pair is Lrvm (defined in number of clk cycles).
One word fetched from LMEM consists of the 4 FECs. So total LMEM accesses to fetch all the FECs is Ceil((Nbfe1c+Nbfe2c)/4,1) where Nbfe1c=Number of FECs with 1 comparison, Nbfe2c = Number of FECs with 2 comparisons.

RX Filter computation time for doing comparison (us)= (Nbfe1c+2*Nbfe2c)*n*CLK_PERIOD where n=no. of clocks for one comparison
RX filter access time is computed as follow:
RX Filter processing time (us) = (Ceil((Nbfe1c+Nbfe2c)/4,1)*Lrf+ (Nbfe1c*2)*Lrvm+(Nbfe2c*4)*Lrvm+n) *CLK_PERIOD

Overall RX Filtering time (us) = RX Filter computation time + RX Filter processing time

1.5.6.7.5 RXFQ0/RXFQ1 Drop Interrupts

This section explains the conditions that result in dropping of messages and generation of RXFQ0/RXFQ1 drop interrupts. This is applicable for both FMM and CTM.

Table 71. RXFQ0/RXFQ1 Drop Interrupts

Condition	Target FIFO	RXFQ0 Status	RXFQ1 Status	IRQ
Message reception started/ongoing.	Not Known	Disabled	Disabled	err_sts_rxfq0_drop. err_sts_rxfq1_drop
		Disabled	Enabled and Full..	
		Enabled and Full..	Disabled	
		Enabled and Full	Enabled and Full	
	RXFQ1	Disabled	Enabled and Full	err_sts_rxfq1_drop
	RXFQ0	Disabled	Enabled	err_sts_rxfq0_drop
	RXFQ0	Enabled and Full	Disabled	err_sts_rxfq0_drop
	RXFQ1	Enabled	Disabled	err_sts_rxfq1_drop

In FMM, an RXFQ is considered full if there is no space left for a word to be stored or if the fill level =255, either of the two conditions. In CTM, if both the conditions below are true then RXFQ is considered full:
a) There is not enough space to fit the full message in multiples of 64bytes in the target RXFQ between write pointer and end address.
b) There is not even one block (64 byte) space from the start address to the read pointer.

The interrupts err_sts_rxfq0_drop and err_sts_rxfq1_drop are raised only due to functional errors (FIFO full or FIFO disabled). Message drop due to safety issues, for example data overflow in PRT, are signaled via safety_rx_abort interrupt.

1.5.6.7.6 RX Abort Interrupt

This section explains the conditions that result in dropping of messages and generation of safety_rx_abort interrupt.

Table 72. RX Abort Interrupt

1	Filtering not completed on time, AFAB bit=0	safety_rx_abort
2	PRT sends RX_MSG_WUSER codes "101" (Data Overflow),"110" (Reserved), "111" (Reserved) during reception	
3	MH- PRT Sequence Error during reception	
4	PRT ENABLE going low during message reception	

1.5.6.7.7 Minimum size of RXFQ in CTM

In CTM, the messages are written into the RXFQ in 64byte blocks. Received messages (RXFE) are stored in continuous locations and cannot be wrapped. So, to be able to store an RXFE of size RXFE_maxsize, the RXFE must either fit between RXFQx_WPTR and RXFQx_EA, or RXFQx_SA and RXFQx_RPTR. If the RXFQ size is not large enough, chances of dropping frames are more. The formula below gives the minimum RXFQ size to be configured based on the maximum RX message size defined by the user (RXFE_maxsize) including the headers and timestamp.
RXFQ_size_min = (RXFQx_EA-RXFQx_SA+1)*64 where
RXFQx_EA = RXFQx_SA + round_up_to_integer (RXFE_maxsize/64byte)*2-1

Example for CAN CC, 8 byte message: RXFQx_EA = RXFQx_SA + round_up_to_integer(24byte/64byte) * 2 - 1= RXFQx_SA + 1=> RXFQ_size_min = 2*64 byte = 128 byte

Example for CAN FD, 64 byte message: RXFQx_EA = RXFQx_SA + round_up_to_integer(80byte/64byte) * 2 - 1= RXFQx_SA + 3 => RXFQ_size_min = 4*64 byte = 256 byte

Example for CAN XL, 256 byte message: RXFQx_EA = RXFQx_SA + round_up_to_integer(272byte/64byte) * 2 - 1= RXFQx_SA + 9 => RXFQ_size_min = 10*64 byte = 640 byte

Example for CAN XL, 2048 byte message: RXFQx_EA = RXFQx_SA + round_up_to_integer(2068byte/64byte) * 2 -1 = RXFQx_SA + 65 => RXFQ_size_min = 66*64 byte = 4224 byte

1.5.6.8 Local Memory Controller

LMEM controller handles all the read and write requests to LMEM raised by submodules inside MH.

LMEM read and write requests are placed at LMEM controller as inputs by VBM, TX Handler and RX Handler. Accesses are granted based on a dynamic cyclic arbitration logic inside the controller. Once request is granted, the corresponding address and control signals are placed at LMEM interface as outputs. The ready signal from LMEM is used as acknowledgment to indicate completion of the transfer. The following sections explain in detail the functionality of the controller.

The below given figure is the block diagram for the LMEM Controller.

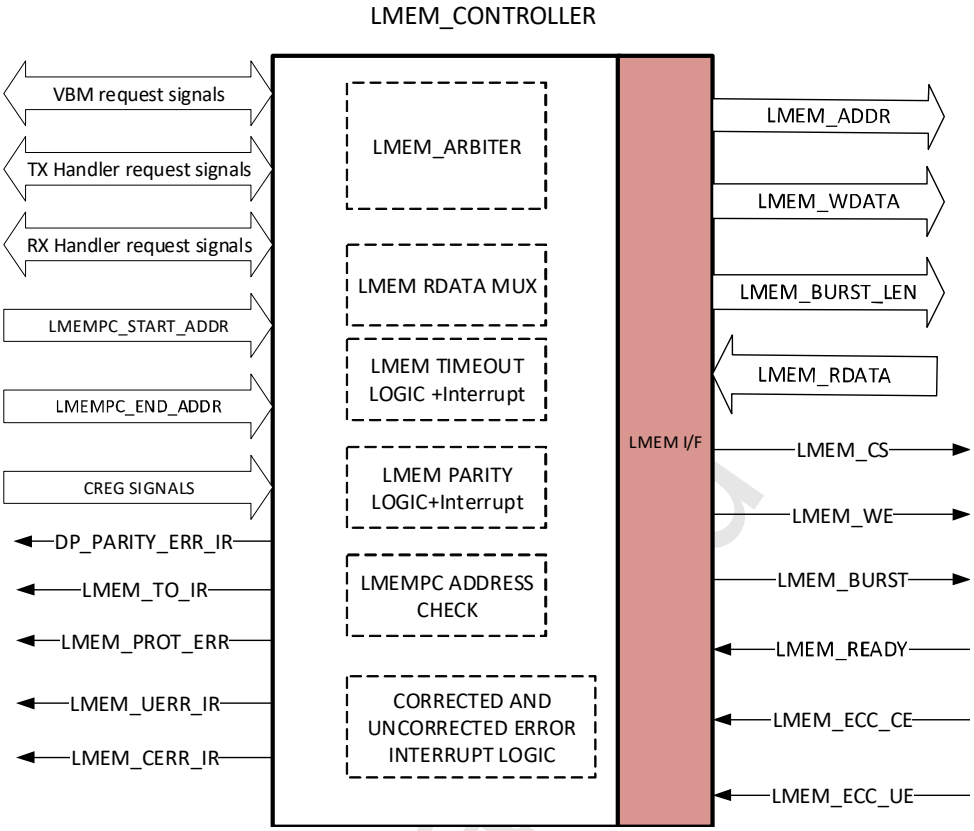


Figure 20. LMEM Controller Block Diagram

1.5.6.8.1 LMEM Arbiter

The arbiter logic inside LMEM controller arbitrates through all the requests and grants access one by one in a cyclic order in a weighted round robin manner. RX Handler requests have a weight of 2, TX Handler and VBM requests have weight 1. This means that two RX Handler requests, if active, are served back to back before proceeding to the next request in order. Signals coming to the arbiter are as follows:

1) VBM signals: **VBM_LMEM_ADDR**, **VBM_LMEM_WDATA**, **VBM_LMEM_RDATA**, **VBM_LMEM_CS**, **VBM_LMEM_WE**, **VBM_LMEM_READY**, VBM_LMEM_BURST,VBM_LMEM_BURST_LEN.

2) TX Handler signals: **TXH_LMEM_ADDR**, **TXH_LMEM_WDATA**, **TXH_LMEM_RDATA**, **TXH_LMEM_CS**, **TXH_LMEM_WE**, **TXH_LMEM_READY**.

3) RX Handler signals: **RXH_LMEM_ADDR**, **RXH_LMEM_WDATA**, **RXH_LMEM_RDATA**, **RXH_LMEM_CS**, **RXH_LMEM_WE**, **RXH_LMEM_READY**.

In the first arbitration cycle, the order of checking is VBM request first followed by TXH request and last RXH request. LMEM arbiter checks for VBM request first. If there is a request from VBM, it will be given access first. If VBM request is not active, TX Handler request will be served if active, else RX Handler request gets the slot if it is active. If more than one request is active in the first cycle, the one that comes first in the order is served. The other one must wait for their turns in the next cycles. The ready signal to each module is asserted only when access is granted and LMEM ready is high to complete the transfer. Note that when VBM gets the slot and VBM_LMEM_BURST=1, then the arbiter grants access to the next request only after the burst transfer is completed. Burst transfer requests are placed only by VBM and not by RXH and TXH.

After the first cycle, arbiter moves to the second cycle and the order of checking is TXH request first followed by RXH request and VBM request.
In the third cycle, order of check is RXH request first, followed by VBM request and TXH request at the last.
This kind of mechanism ensures fair arbitration to all requests and makes sure that all requests are served.

1.5.6.8.1.1 Write Operation

After arbitration, the selected address, and the data to be written is placed on the **LMEM_ADDR** and **LMEM_WDATA** line respectively. The chip select signal **LMEM_CS** and **LMEM_WE** are asserted high. These signals are retained till the **LMEM_READY** output from LMEM is high. A write operation is completed when ready is asserted by the LMEM indicating it has taken the data and **LMEM_CS** is pulled low in the next cycle if there is no more transaction. As soon as write signals are placed at the interface, the LMEM timeout counter starts to count down. The timeout setting is explained in section 1.5.6.12. If ready is not received at the LMEM interface before the timeout happens, **SAFETY_LMEM_TO_ERR_IRQ** single pulse interrupt is raised for one host clk period. Refer exception handling chapter 1.5.8 for the consequences and MH behavior.

1.5.6.8.1.2 Read Operation

After arbitration, the selected address is placed on the **LMEM_ADDR** line. The chip select signal **LMEM_CS** is asserted high and **LMEM_WE** is pulled low to indicate a read access. The read data at **LMEM_RDATA** is sampled by MH when **LMEM_READY** is high. A read operation is completed when ready is asserted by the LMEM indicating it has taken the data and **LMEM_CS** is deasserted in the next cycle. As soon as read signals are placed at the interface, the LMEM timeout counter starts to count down. The timeout setting is explained in section 1.5.6.12. If ready is not received at the LMEM interface before the timeout happens, **SAFETY_LMEM_TO_ERR_IRQ** pulse interrupt is raised for one host clk period. Refer exception handling chapter 1.5.8 for the consequences and MH behavior.

N.B :Note that VBM accesses are excluded from LMEM timeouts.i.e if the write or read access is from host, the timer is not activated. This is because VBM accesses are always done by the host and it is assumed that there is a master time out at the host which tracks the time for this access.

1.5.6.8.2 LMEM Error Handling Sideband Signals

It is assumed that the external LMEM implements safety mechanism to protect data. The safety protection implemented in the LMEM could either report error when reading a data (Single Error Detection) or be able to correct it in some cases (Single Error Correction and Double Error Detection for example). To address those two options, two input side band signals *LMEM_ECC_UE* (uncorrectable error) and *LMEM_ECC_CE* (correctable error), are provided to the IP.

Error Detection Only: When a corrupted data is read from the LMEM, a pulse of one HOST clock cycle must be generated on the *LMEM_ECC_UE* input signal. This input means that the read data is corrupted but not corrected. LMEM controller generates an interrupt pulse *SAFETY_LMEM_UERR_IRQ* to IRC for one host clk period. The *LMEM_ECC_UE* input signal is synchronous to HOST_CLK at the LMEM interface.

Error Detection and Correction: When a corrupted data is read from the LMEM but is corrected, a pulse of one HOST clock cycle must be generated on the *LMEM_ECC_CE* input signal. LMEM controller generates an interrupt pulse *ERR_STS_LMEM_CERR_IRQ* for one host clk period to IRC. The *LMEM_ECC_CE* input signal must be synchronous to the HOST_CLK at LMEM interface.

It is recommended to provide *LMEM_ECC_CE/LMEM_ECC_UE* signal with LMEM_READY which is given for that particular read access. However IP still accepts it even if it is asynchronous to LMEM_READY.

1.5.6.8.3 LMEMPC Check

The allowed address space of LMEM for one instance of XS_CAN IP can be provided as static input signals to the IP. **LMEMPC_START_ADDR** and **LMEMPC_END_ADDR** input signals define this range. If an access to an address outside this range is issued to LMEM, LMEM controller triggers an interrupt pulse **SAFETY_LMEM_PROT_ERR_IRQ** to IRC.

1.5.6.8.4 LMEM Parity Check

Datapath parity safety mechanism is implemented at LMEM controller interface for both TX and RX direction. Note that this feature is disabled if the input signal **NO_SAFETY_NO_CTM** is set to 1.

1.5.6.8.4.1 Parity Check and Removal of Parity bits:

Any write access to LMEM is done after checking for parity in write data path and parity bits are removed before sending the data to LMEM. This is applicable for TXFQ enqueuing, TXPQ enqueuing, LMEM transparent write access, RXFQ0/1 enqueuing in FMM, TEFQ enqueuing and CTB enqueuing in CTM. The parity calculation and insertion for the write data are done at other entry points in the IP (AHB host interface and RX_MSG MH-PRT interface). In case of parity mismatch, write transaction is not sent to LMEM.

1.5.6.8.4.2 Parity Calculation and Insertion of Parity Bits:

After a read access from LMEM, parity is calculated per byte of the read data and appended to the msb of read data before transmitting to the requested module. This is applicable for TXFQ dequeuing, TXPQ dequeuing, LMEM transparent read access, RXFQ0/1 dequeuing in FMM, TEFQ dequeuing and CTB dequeuing in CTM. The parity check and removal for the read data are done at other exit points in the IP (AHB host interface and RX_MSG MH-PRT interface).

In case of a parity mismatch in either TX or RX direction, safety interrupt **DP_PARITY_ERR** is generated by the LMEM controller. Interrupt is one host clock pulse signal.The source flags are updated into the register **MH_STS1.TX_DP_PARITY_ERR** and **MH_STS1.RX_DP_PARITY_ERR**.

1.5.6.8.5 Message Handler LMEM Interface

The signals at MH LMEM interface for memory operation are given below:

LMEM_ADDR (16 bits) - read and write address
LMEM_RDATA(32 bits) - read data from LMEM
LMEM_WDATA(32 bits) - write data to LMEM
LMEM_WE(1 bit) - Write enable; 1 - write, 0 - read
LMEM_CS(1 bit) - chip select
LMEM_READY(1 bit) - ready from LMEM
LMEM_BURST (1bit)- Burst access or single access
LMEM_BURST_LEN (2 bits)- 00- single, 01-INCR4,10-INCR8,11-INCR16

1.5.6.9 MH Interrupts

This section explains the interrupts raised by MH to IRC. If not mentioned specifically, the interrupt generation is same for FMM and CTM. Note that all interrupts and flags are generated within a latency of 5 host clock cycles of detecting the cause.

1.5.6.9.1 TX Functional Interrupts

1) *FUNC_TXFQ_ENQ_IRQ*: This interrupt is raised by VBM when one message is successfully enqueued into TXFQ. This is a pulse interrupt valid for one host clock cycle.

2) *FUNC_TXPQ_ENQ_IRQ*: This interrupt is raised by VBM when one message is successfully enqueued into TXPQ. This is a pulse interrupt valid for one host clock cycle.

3) *FUNC_TEFQ_ENQ_IRQ*: This interrupt is raised by TX Handler when one element is successfully enqueued into TEFQ. This is a pulse interrupt valid for one host clock cycle.

4) *FUNC_TEFQ_DEQ_IRQ*: This interrupt is raised by VBM when one element is successfully dequeued from TEFQ. This is a pulse interrupt valid for one host clock cycle.

5) *FUNC_CTB_TX_BC_REQ_IRQ*: This interrupt is raised by TX Handler to indicate to the host to perform a block copy in TX direction.This is a pulse interrupt valid for one host clock cycle and is generated only in CTM.The source flag associated with this event is stored into register **CTM_EVENT.TX_BLOCK_COPY_REQ**. This bit is auto cleared when block copy is finished.

1.5.6.9.2 RX Functional Interrupts

1) *FUNC_RXFQ0_ENQ_IRQ*: This interrupt is raised by RX Handler when one element is successfully enqueued into RXFQ0. This is a pulse interrupt valid for one host clock cycle. In FMM, this is generated when message is enqueued in LMEM and in CTM, this is generated after enqueue in SMEM.

2) *FUNC_RXFQ1_ENQ_IRQ*: This interrupt is raised by RX Handler when one element is successfully enqueued into RXFQ1. This is a pulse interrupt valid for one host clock cycle.In FMM, this is generated when message is enqueued in LMEM and in CTM, this is generated after enqueue in SMEM.

3) *FUNC_RXFQ0_DEQ_IRQ*: This interrupt is raised by VBM when one message is successfully dequeued from RXFQ0. This is a pulse interrupt valid for one host clock cycle and is generated only in FMM.

4) *FUNC_RXFQ1_DEQ_IRQ*: This interrupt is raised by VBM when one message is successfully dequeued from RXFQ1. This is a pulse interrupt valid for one host clock cycle and is generated only in FMM.

5) *FUNC_CTB_RX_BC_REQ_IRQ*: This interrupt is raised by RX Handler to indicate to the host to perform a block copy in RX direction.This is a pulse interrupt valid for one host clock cycle and is generated only in CTM.This is a pulse interrupt valid for one host clock cycle and is generated only in CTM.The source flag associated with this event is stored into register **CTM_EVENT.RX_BLOCK_COPY_REQ**. This bit is auto cleared when block copy is finished. In case if one cut through buffer is already full and read request is active, this interrupt will not be generated even if the next buffer gets full. Once the pending block copy request is served, interrupt is raised for the next full block.

6) *FUNC_MH_RX_FILTER_MATCH_IRQ*: This interrupt is generated by RX Handler if the IRQ bit in FEC is set and filtering resulted in a match. This is a pulse interrupt valid for one host clock cycle.

1.5.6.9.3 Error Interrupts

1) *ERR_STS_TXFQ_DEQ_NO_TX_IRQ*: This interrupt is raised by TX Handler if a message gets dequeued from TXFQ in LMEM but is not transmitted at the bus due to some error. This is a pulse interrupt valid for one host clock cycle

2) *ERR_STS_TXPQ_DEQ_NO_TX_IRQ*: This interrupt is raised by TX Handler if a message gets dequeued from TXPQ in LMEM but is not transmitted at the bus due to some error. This is a pulse interrupt valid for one host clock cycle

3) *ERR_STS_TEFQ_DROP_IRQ*: This interrupt is raised by TX Handler if an element cannot be stored into TEFQ due to full condition.This is a pulse interrupt valid for one host clock cycle.

4) *ERR_STS_RXFQ0_DROP_IRQ*: This interrupt is raised by TX Handler if a message cannot be stored into RXFQ0 due to full condition.Full condition can also mean that the RXFQ0 configurations are not done properly by the host.This is a pulse interrupt valid for one host clock cycle. In FMM, this is generated when message is enqueued into full RXFQ0 in LMEM and in CTM, this is generated when message is enqueued into full RXFQ0 in SMEM.

5) *ERR_STS_RXFQ1_DROP_IRQ*: This interrupt is raised by TX Handler if a message cannot be stored into RXFQ1 due to full condition.Full condition can also mean that the RXFQ1 configurations are not done properly by the host.This is a pulse interrupt valid for one host clock cycle. In FMM, this is generated when message is enqueued into full RXFQ1 in LMEM and in CTM, this is generated when message is enqueued into full RXFQ1 in SMEM.

6) *ERR_STS_MH_HOST_ARA_IRQ*: This interrupt is triggered by VBM to indicate that host is trying to access either a reserved area of a virtual buffer address or a reserved MH register address.

7) *ERR_STS_LMEM_CERR_IRQ*: This interrupt is generated by LMEM Controller module if the input signal *LMEM_ECC_CE* is asserted indicating there is a correctable error detected at the LMEM interface.This is a pulse interrupt valid for one host clock cycle.

8) *ERR_STS_QUEUE_ACCESS_VIOLATION_IRQ*: This interrupt is generated by VBM in case of a wrong access to any of the queues in LMEM. The source flags corresponding to this interrupt are updated into the register MH_STS0 bits 30 down to 22. The register MH_STS0 is a read on clear register and gets updated one clock after the interrupt is generated in MH. The interrupt *ERR_STS_QUEUE_VIOLATION_IRQ* is a pulse interrupt valid for one host clock cycle. Note that in case of access to a disabled queue also this interrupt is triggered.
Also note that this interrupt gets triggered if RXFQs are accessed in CTM and CTBs are accessed in FMM. No flags are provided for this two scenarios.

9) *ERR_STS_VB_NO_SA_IRQ*: This interrupt is generated by VBM when host issues an address within VB address range instead of the start address which is expected. After the trigger address is received at VBM, next expected address in always the start address for any virtual buffer.The source flags corresponding to this interrupt are updated into the register MH_STS0 bits 21down to 16. The register MH_STS0 is a read on clear register which gets updated one clock after the interrupt is generated in MH. The interrupt is a pulse interrupt valid for one host clock cycle.

10) *ERR_STS_PRT_STOP_IRQ*: This interrupt is triggered by MH if PRT is stopped and PRT_ENABLE input to MH goes low.

1.5.6.9.4 Safety Interrupts

1) *SAFETY_VB_ACCESS_VIOLATION_IRQ*: This interrupt is triggered by VBM in case of a virtual buffer access violation by host. The source flags corresponding to this interrupt are updated into the register MH_STS1 bits 23 down to 5. These source flags are cleared on reading the register. This is a pulse interrupt valid for one host clock cycle.

2) *SAFETY_LMEM_UERR_IRQ*: This interrupt is triggered by LMEM Controller if the input signal *LMEM_ECC_UE* is asserted indicating there is an uncorrectable error detected at the LMEM interface.This is a pulse interrupt valid for one host clock cycle.

3) *SAFETY_LMEM_TO_ERR_IRQ*: This interrupt is triggered by LMEM Controller if there is a timeout at the LMEM interface. This is a pulse interrupt valid for one host clock cycle.

4) *SAFETY_LMEM_PROT_ERR_IRQ*: This interrupt is triggered by LMEM Controller if there is an access to LMEM outside the LMEMPC address range. This is a pulse interrupt valid for one host clock cycle.

5) *SAFETY_RX_FILTER_ERR_IRQ*: This interrupt is triggered by RX Handler if filtering process was aborted before completion.This is a pulse interrupt valid for one host clock cycle.

6) *SAFETY_MISC_IRQ*: This interrupt is triggered by various submodules for different safety errors detected. The source flags corresponding to this interrupt are updated into the register MH_STS1 bits 4 down to 0. These source flags are cleared on reading the register. This is a pulse interrupt valid for one host clock cycle.

7) *SAFETY_CTB_BC_TO_IRQ*: This interrupt is triggered by TX handler or RX Handler if there is a timeout during block copy in CTM. There is no timer to configure this timeout. In case of TX, this interrupt is triggered if there is data underrun in MH while fetching from the cut through buffers and in case of RX, this is triggered if there is data overflow in MH while storing into cut through buffers. This is a pulse interrupt valid for one host clock cycle.

8)SAFETY_CTB_BC_ERR_IRQ: This interrupt is triggered by MH if an error response is received at CTM_RESP input signal for an ongoing block copy transfer.

9) *SAFETY_RX_ABORT_IRQ*: This interrupt is generated by RX Handler if a message in reception is aborted by MH. This is a pulse interrupt valid for one host clock cycle.

10) *SAFETY_TX_ABORT_IRQ*: This interrupt is generated by TX Handler if a message in transmission is aborted .by MH. This is a pulse interrupt valid for one host clock cycle.

1.5.6.10 PRT ENABLE and MH_ENABLE Dependency

The PRT module can be started by writing the command 1 to CTRL.STRT. After the PRT is started, the ENABLE output signal to MH goes high. This ENABLE signal from PRT sets the bit MH_CTRL.MH_ENABLE. This means that MH is enabled by default if PRT is started. The signal MH_CTRL.MH_ENABLE going to high from low is same as soft reset to MH. All logic inside MH gets reset with this. The status registers are also reset in the next clock cycle. The configuration registers written by the host retain their values. But the host must read the configuration registers and ensure the correctness before enabling the MH again. The register configuration bit MH_CTRL.MH_ENABLE is not writable by the host if PRT ENABLE is set. Only when *PRT_ENABLE* goes low, the register MH_CTRL.MH_ENABLE becomes read-write.
Also note that when PRT is stopped by software and *PRT_ENABLE* goes low, MH_CTRL.MH_ENABLE does not go low by default. MH VBM is still running. If MH is to be disabled, host must do this by writing 0. If host disables MH in the middle of a process which is already started in the IP, chances are that some output signals might be generated and status register updates can also happen. This is normal behavior. The host can enable the MH without starting the PRT by setting the MH_CTRL.MH_ENABLE bit. For accessing LMEM, the MH must be enabled. Virtual Buffer accesses are possible only if MH is enabled. Behavior of MH when PRT is stopped immediately is explained Error and Exception handling section.

1.5.6.11 Trace and Debug

1.5.6.11.1 Hardware Debug Port

The 16 bit HDP bus provides some visibility to internal signals to debug the MH. By default, there is not activity on the HDP bus.

To enable the toggling of the HW signal on the HDP bus, set the **DEBUG_TEST_CTRL.HDP_EN** bit to 1 and the selected set to be monitored on the HDP using the **DEBUG_TEST_CTRL.HDP_SEL[2:0]**.

To disable the Hardware Debug Port monitoring, do the previous set with **DEBUG_TEST_CTRL.HDP_EN** bit to 0.
Up to 8 sets can be defined using the **DEBUG_TEST_CTRL.HDP_SEL[2:0]** bit field. When the value is set to n the set n is selected in the HDP bus.

Here below are the description of sets available on the MH HDP bus.

Table 73. HDP Set 0 and 1

HDP[15:0]	set 0(Queue Interrupts)	set 1(Queues Access)
15	HOST_CLK	HOST_CLK
14	reserved	reserved
13	CTB_ENQ	reserved
12	CTB_DEQ	reserved
11	reserved	reserved
10	reserved	reserved
9	FUNC_CTB_RX_BC_REQ_IRQ	reserved
8	FUNC_CTB_TX_BC_REQ_IRQ	reserved
7	FUNC_RXFQ1_DEQ_IRQ	reserved
6	FUNC_RXFQ0_DEQ_IRQ	reserved
5	FUNC_RXFQ1_ENQ_IRQ	ERR_STS_RXFQ1_DROP_IRQ
4	FUNC_RXFQ0_ENQ_IRQ	ERR_STS_RXFQ0_DROP_IRQ
3	FUNC_TEFQ_DEQ_IRQ	ERR_STS_TEFQ_DROP_IRQ
2	FUNC_TEFQ_ENQ_IRQ	ERR_STS_TXPQ_DEQ_NO_TX_IRQ
1	FUNC_TXPQ_ENQ_IRQ	ERR_STS_TXFQ_DEQ_NO_TX_IRQ
0	FUNC_TXFQ_ENQ_IRQ	ERR_STS_QUEUE_ACCESS_VIOLATION_IRQ

Table 74. HDP Set 2 and 3

HDP[15:0]	set 2(Error IR and flags)	set 3(LMEM IF-MH APB)
15	HOST_CLK	HOST_CLK
14	reserved	MH_ENABLE
13	CTB_VB_NONLIN_ACC	HOST_CLK_AON
12	SAFETY_RX_FILTER_ERR_IRQ	SAFETY_TX_ABORT_IRQ
11	ERR_STS_PRT_STOP_IRQ	SAFETY_RX_ABORT_IRQ
10	SAFETY_VB_ACCESS_VIOLATION_IRQ	ERR_STS_LMEM_CERR_IRQ
9	SAFETY_LMEM_PROT_ERR_IRQ	LMEM_ECC_CE
8	SAFETY_LMEM_TO_ERR_IRQ	LMEM_ECC_UE
7	SAFETY_LMEM_UERR_IRQ	LMEM_CS
6	SAFETY_HOST_ARA_IRQ	LMEM_WE
5	SAFETY_CTB_BC_TO_IRQ	LMEM_READY
4	reserved	REG_APB_PSLVERR
3	reserved	REG_APB_PREADY
2	TX_DP_PARITY_ERR	REG_APB_PWRITE
1	RX_DP_PARITY_ERR	REG_APB_PENABLE
0	TS_PARITY_ERR	REG_APB_PSELX

Table 75. HDP Set 4 and 5

HDP[15:0]	set 4(MH-PRT interface)	set 5(TX-SCAN)
15	HOST_CLK	HOST_CLK
14	TX_MSG_WVALID	reserved
13	TX_MSG_WUSER[1]	reserved
12	TX_MSG_WUSER[0]	reserved
11	TX_MSG_WREADY	reserved
10	TX_MSG_BVALID	reserved
9	TX_MSG_BUSER_STATUS[2]	reserved
8	TX_MSG_BUSER_STATUS[1]	reserved
7	TX_MSG_BUSER_STATUS[0]	reserved
6	TX_MSG_BREADY	reserved
5	RX_MSG_WVALID	reserved
4	RX_MSG_WUSER[2]	reserved
3	RX_MSG_WUSER[1]	reserved
2	RX_MSG_WUSER[0]	reserved
1	RX_MSG_WREADY	reserved
0	PRT_ENABLE	reserved

Table 76. HDP Set 6 and 7

HDP[15:0]	set 6(VBM AHB)	set 7(DMA req+queues)
15	HOST_CLK	HOST_CLK
14	LMEM_CS	reserved
13	LMEM_WE	CTM_RREQ
12	LMEM_READY	CTM_WREQ
11	VBM_AHB_HRESP_0	CTM_RESP_1
10	VBM_AHB_HREADYOUT	CTM_RESP_0
9	VBM_AHB_HSIZE_2	FUNC_TXFQ_ENQ_IRQ
8	VBM_AHB_HSIZE_1	FUNC_TXPQ_ENQ_IRQ
7	VBM_AHB_HSIZE_0	FUNC_RXFQ0_DEQ_IRQ
6	VBM_AHB_HBURST_2	FUNC_RXFQ1_DEQ_IRQ
5	VBM_AHB_HBURST_1	FUNC_TEFQ_DEQ_IRQ
4	VBM_AHB_HBURST_0	TXFQ_WREQ
3	VBM_AHB_HTRANS_1	TXPQ_WREQ
2	VBM_AHB_HTRANS_0	TXEF_RREQ
1	VBM_AHB_HSELX	RXFQ0_RREQ
0	VBM_AHB_HWRITE	RXFQ1_RREQ

1.5.6.12 Timeout Settings

One timeout counter is provided in XS_CAN IP.

LMEM Interface timeout counter for LMEM read write accesses.

A common prescaler is used to set the reference clock for all timeout counters, see MH_SFTY_CFG.PRESCALER register. According to the value defined in the MH_SFTY_CFG.PRESCALER register, the timeout counter reference clock is either CLK/32, CLK/64, CLK/128 or CLK/256.

In order to set the timeout value, use the following registers:
• Set the value in the MH_SFTY_CFG.LMEM_TIMEOUT _VAL register for the LMEM interface and enable the counter by setting MH_SFTY_CFG.LMEM_TIMEOUT _EN.

As the HOST_CLK and prescaler are common to all timeout counters, once the HOST_CLK is defined and the prescaler set, only a defined range is possible. The table below provides the possible range for different timeout (according to the CLK clock frequency and clock ratio). Timeout can get triggered anytime between (Timeout_val-1)*clock ratio to (Timeout_val) * clock ratio, measured in host clock cycles.

Table 77. LMEM Interface Timeout Configuration

HOST CLK	PRESCALER		Timeout Step (us)	LMEM Interface Timeout range (us)	
320/160/80/40 MHz	Value	CLK Ratio		Max	Min
				255	0
40	0	32	0.8	204	0
	1	64	1.6	408	0
	2	128	3.2	816	0
	3	256	6.4	1632	0
80	0	32	0.4	102	0
	1	64	0.8	204	0
	2	128	1.6	408	0
	3	256	3.2	816	0
160	0	32	0.2	51	0
	1	64	0.4	102	0
	2	128	0.8	204	0
	3	256	1.6	408	0
320	0	32	0.1	25.5	0
	1	64	0.2	51	0
	2	128	0.4	102	0
	3	256	0.8	204	0

Timeout Calculation examples:

LMEM Timeout value (us)= (1/(HOST_CLK (MHz)) * (ratio defined in MH_SFTY_CFG.PRESCALER)*MH_SFTY_CFG. LMEM_TIMEOUT_VAL.
For example, if HOST_CLK=80MHz, MH_SFTY_CFG.PRESCALER=2, MH_SFTY_CFG. LMEM_TIMEOUT_VAL=128,

LMEM timeout in us=(1/80)*128*128

1.5.7 Cut Through Descriptors and Block Copy

In Cut Through Mode, CTDMA must perform write transfers to the two CTBs in case of TX and read transfers from CTBs in case of RX. There are two Cut-Through buffers of 64bytes each and the access is to be done in an alternate manner. This section explains the use of Cut Through Descriptors in performing block copy transfers from and to CTBs. Cut Through Descriptors are a set of registers provided by the IP to help the host in performing these transfers. It is important to note that even though CTD are part of MH CREG, they can be read via register address map. There are 2 registers in this set and the same CTDs are used for both TX and RX.

1. **CTM_DESC_SRC** (Cut Through Mode Descriptor Source Address): This is a read only register that holds the source address from which data should be read by the host. In case of TX direction, this points to a location in SMEM since data is transferred from SMEM to LMEM and in case of RX, this holds the virtual address of CTB since data is transferred from CTB(LMEM) to SMEM. This address is a 32bit value.
2. **CTM_DESC_DEST** (Cut Through Mode Descriptor Destination Address): This is a read only register that holds the destination address to which data should be written by the host. In case of TX direction, this holds the virtual address of the CTB since data is transferred from SMEM to LMEM and in case of RX, this points to a location in SMEM since data is transferred from CTB (LMEM) to SMEM.

1.5.7.1 CTD updates in TX direction

In case of TX, CAN XL with payload size greater than 64bytes, the payload beyond 64bytes need to be stored into the CTBs once the message wins arbitration on the CAN bus. When the VBM raises request for a block copy transfer in TX direction, the TX Handler would have already updated the contents of the source and destination descriptors.

For TX, **CTM_DESC_SRC** = SMEM address calculated from the SMEM pointer in the TX message header T3.

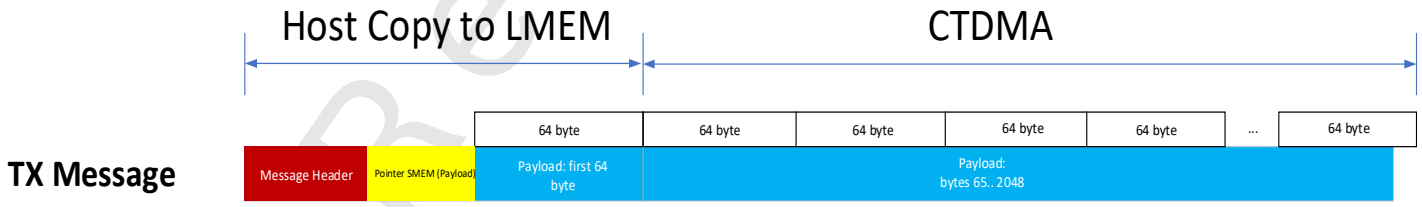


Figure 21. TX in CTM

The SMEM pointer in the TX message header T3 is a 4byte (word) aligned address that points to the address in SMEM where the payload for that CAN XL message starts. From this address, the location of the 65th byte is calculated and updated into the CTM_DESC_SRC register for the first block copy. Block copy is always in 64byte size. For the subsequent block copies, source address is calculated based on the remaining payload. If the payload size is not a multiple of 64byte, host must append some data to the remaining words to make the block copy in size of 64. The entire 64byte block is enqueued into CTB by VBM, even if it consists of invalid data.

CTM_DESC_DEST = CTB Virtual address i.e 0x06000. The destination address is the CTB virtual address in case of TX.

An example scenario is explained below.

Consider a CAN XL message with payload size 200bytes. The SMEM pointer address in T3 is a 4byte(word) aligned address 0xAAAA_11C0. This address points to the first word of the payload in SMEM. The IP needs to calculate the location where the 65th byte is stored.i.e 0xAAAA_11C0+0x40 =0xAAAA_1200.

For the first block copy transfer,
CTM_DESC_SRC = 0xAAAA_1200 **CTM_DESC_DEST** = 0x06000

These CTDs are updated by the TX Handler and block copy request is triggered by VBM if there is no pending previous block copy transfer. Host can read the CTDs from CREG using register address map and perform the block copy.
For the XS_CAN IP, what is important is the write issued to the CTBs. IP expects the payload word by word along with the incrementing addresses 0x06000, 0x06004, up to 0x0603C which is the trigger address. Accessing the trigger address completes one block copy. Totally 128 bytes are now read from SMEM (first 64bytes in TXFQ/TXPQ and the next 64bytes in one of the CTBs.). Remaining 72 bytes are to be read from SMEM. Once the first block copy is finished and there is at least one CTB which is free VBM again raises the block copy request and TX Handler updates the CTD.

For this block copy transfer,
CTM_DESC_SRC = 0xAAAA_1200+0x40=0xAAAA_1240
CTM_DESC_DEST = 0x06000(Unchanged)

The IP expects write transactions starting from 0x06000 till 0x0603C which completes the second block copy of 64bytes.

One more block copy transfer request must be initiated by the IP for copying the remaining last 8 bytes of payload from SMEM. Note that even though the remaining payload is not equal to 64bytes, host must append some data to the remaining bits and make the transfer size equal to 64bytes.

For this last block copy transfer,
CTM_DESC_SRC = 0xAAAA_1200+0x40+0x40= 0xAAAA_1280**CTM_DESC_DEST** = 0x06000(Unchanged)

The IP expects write transactions starting from 0x06000 till 0x0603C which completes the last block copy of 64bytes.

1.5.7.1.1 Block Copy Timeout Scenarios in TX

In TX, block copy timeout happens when a block copy request is not completed on time resulting in data underrun condition in PRT. Block copy is considered complete when CTM_RESP is received from CTDMA for the CTM_TX_WREQ raised. If the response is not received by the time data is needed from CTB, this results in timeout and data underrun. Safety Interrupts raised by XS_CAN are MH_CTB_BC_TO, PRT_TX_DU and MH_TX_ABORT. CTM_BC_ERR output is also raised if CTM_RESP is not received by the time data underrun happens.

1.5.7.2 CTD updates in RX direction

In case of RX, all received messages in CTM are buffered into CTBs irrespective of the type of message. MH starts to receive words from PRT at RX_MSG interface and are stored into the first available CTB. One full CTB must be free to start storing. If no CTB is available when the first word is received at the RX_MSG interface, the message is dropped. When one CTB is filled with data, VBM raises *CTM_RX_RREQ* for the CTDMA to start copying from CTB to SMEM. Cut Through Descriptors are also updated at the same time by RX Handler with the relevant information.

For RX, **CTM_DESC_SRC** = CTB Virtual address i.e 0x06000. The source address is the CTB virtual address in case of RX.

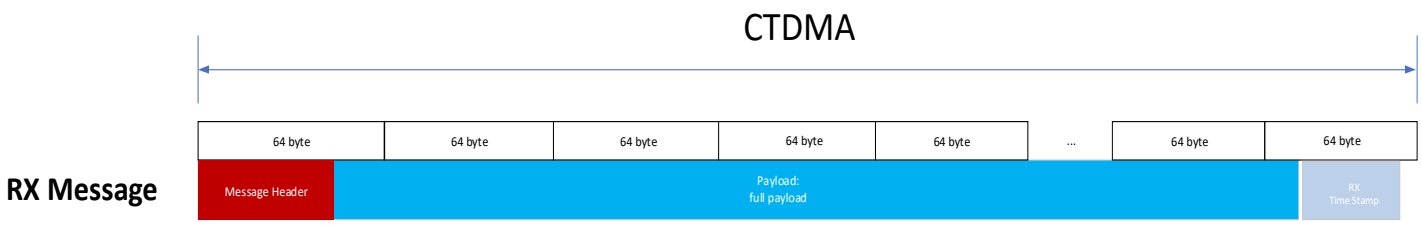


Figure 22. RX in CTM

CTM_DESC_DEST = Target RXFQ SMEM address calculated from RXFQx details from CREG where x is 0 or 1. The destination address is the RXFQ in SMEM. Based on the result of filtering, target RXFQ must be known to the RX Handler by the time the storing gets completed into the current CTB. CTM_DESC_DEST is calculated based on the **RXFQx_SA**, **RXFQx_EA**, **RXFQx_RPTR**, **RXFQx_WPTR** and **RX_FQx_WRAP_PTR**.

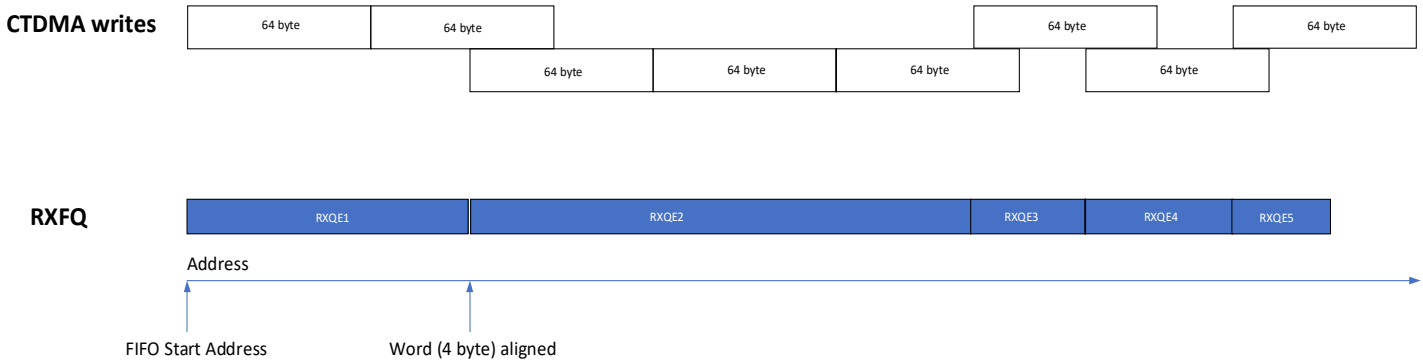


Figure 23. CAN message storage in SMEM (RXFQ)

An example scenario is explained below:

Consider a CAN XL message with total size 200bytes (header plus payload). Assume target FIFO is RXFQ0. The RXFQ0 start address is 64byte aligned address 0xAAAA_11C0.

For the first block copy transfer,
CTM_DESC_SRC = 0x06000
CTM_DESC_DEST = 0xAAAA_11C0

These CTDs are updated by the RX Handler and block copy request is triggered by VBM if there is no pending previous block copy transfer. Host can read the CTDs from CREG using register address map and perform the block copy.

For the XS_CAN IP, what is important is the read issued from CTBs. Host can issue a burst read access along with the incrementing addresses 0x06000, 0x06004, up to 0x0603C which is the trigger address. Accessing the trigger address completes one block copy. Totally 64 bytes are now written into SMEM. Remaining 136 bytes are to be read from CTB. Once the first block copy is finished and there is at least one CTB which is full VBM again raises the block copy request and RX Handler updates the CTD.

For this block copy transfer,
CTM_DESC_SRC = 0x06000(Unchanged)**CTM_DESC_DEST** = 0xAAAA_11C0+0x40=0xAAAA_1200

The IP expects write transactions starting from 0x06000 till 0x0603C which completes the second block copy of 64bytes.

Two more block copy transfer request must be initiated by the IP for copying the remaining last 72 bytes of payload to SMEM. Note that even though the remaining payload is not a multiple of 64bytes, VBM append 0s to the remaining bits and make the transfer size equal to 64bytes.

For this last block copy transfer,
CTM_DESC_SRC = x06000(Unchanged)
CTM_DESC_DEST = 0xAAAA_1200+0x40+0x40+0x40= 0xAAAA_12C0

The IP expects write transactions starting from 0x06000 till 0x0603C which completes the last block copy of 64bytes.

1.5.7.2.1 Block copy timeout scenarios in RX

Block copy timeout happens in RX if a CTB is not available for storing an incoming word from PRT. The word can belong to a new message or it can be a part of a message which already exists in one of the CTBs. If an RX block copy request raised by XS_CAN is not served by CTDMA on time (i.e. both buffers are full when there is a need for the buffer), then one of the following scenarios can occur:

1) The current message for which block copy request is raised takes more than 2 buffer space, and cannot be stored further. As a result, data over flow happens in PRT and the message is aborted. Safety Interrupts raised are PRT_RX_DO, MH_CTB_BC_TO and MH_RX_ABORT. If the same message for which request was raised gets dropped and data overflow happens before CTM_RESP, then CTM_BC_ERR output is also raised to indicate to CTDMA that the requested block copy is canceled. CTM_RX_RREQ is also pulled low.

2) A new message does not get space, data overflow happens in PRT and the message is aborted. Safety Interrupts raised are PRT_RX_DO, MH_CTB_BC_TO and MH_RX_ABORT. CTM_BC_ERR output is not raised in this case since a different message is aborted and the block copy request raised is still valid.

1.5.8 Error and Exception Handling in MH

This section describes the different error scenarios and the actions taken by MH for the same.Note that the behavior of the IP when not configured properly by the host cannot be defined. It is up to the host to define proper values into the CREG to ensure correct working of the IP. Such error cases are not described here.

1.5.8.1 Timeout at LMEM interface

This section explains the causes and actions of MH in case of a timeout at LMEM interface.

Cause: TX Handler and RX Handler accesses to LMEM

TX Handle rand RX handler can access LMEM for various operations like TX Scan, RX Filtering, TX message dequeue, RX message enqueue, TX Event FIFO Queue enqueue etc. As soon as the request is placed at LMEM interface, counter starts counting down while waiting for the ready from LMEM.

Action: If any of the access by TX Handler and RX Handler causes LMEM timeout, then the module that raised the request continues to wait in that state, waiting for the ready from LMEM. Timeout counter is not reset and round robin arbitration also do not take place in the arbiters inside TX Handler, RX Handler and LMEM controller.

Interrupts and Flags raised:
SAFETY_LMEM_TO_ERR_IRQ interrupt is raised to inform that a timeout has occurred.

Note that VBM accesses to LMEM do not result in timeout.Only TX Handler and RX handler accesses are considered for timeout calculations.

1.5.8.2 Uncorrectable error at LMEM interface

This section explains the causes and actions of MH in case of uncorrectable error at LMEM interface.

Cause 1: External Host read access to LMEM

Host read access to LMEM like read from RXFQ0, RXFQ1, TEFQ or transparent access read, via VBM resulting in setting of uncorrected error input (LMEM_ECC_UE =1).

Action: MH generates the output pulse PRT_STOP_IMMD_REQ instructing PRT to stop immediately. MH enable in the register MH_CTRL.MH_ENABLE is pulled low by MH. MH will hold its states till MH_ENABLE is made 1 by Host writing register or indirectly from PRT_ENABLE going high. If uncorrected error is received during a VBM read access, MH sends HOST_AHB_HRESP = ERROR back to the host with default read data 0xCAFECAFE. ERROR response lasts for two host clock cycles. If uncorrected error is received during CTB read access i.e. before CTM_RESP is received, the output signal CTM_BC_ERR is also generated for 1 HOST_CLK cycle. CTDMA does not send any further response if CTM_BC_ERR is raised by XS_CAN. Only transparent accesses are allowed during the time MH_ENABLE=0.

Cause 2: Internal module read access to LMEM

LMEM read accesses performed by TX Handler or RX handler modules resulting in setting of uncorrected error input (LMEM_ECC_UE =1).

Action: MH generates the output pulse PRT_STOP_IMMD_REQ instructing PRT to stop immediately. MH enable in the register MH_CTRL.MH_ENABLE is pulled low by MH. MH will hold its states till MH_ENABLE is made 1 by Host writing register or indirectly from PRT_ENABLE going high. HOST_AHB_HRESP is not applicable in this case since it is an internal access. Only transparent accesses are allowed during the time MH_ENABLE=0.

Interrupts and Flags Raised:

SAFETY_LMEM_UERR_IRQ interrupt is raised in both the cases to indicate that uncorrectable error at LMEM is received.

1.5.8.3 Correctable error at LMEM interface

This section explains the causes and actions of MH in case of correctable error at LMEM interface.

Cause: Read access to LMEM by host and internal modules of MH

Host read access from LMEM via VBM and internal read accesses performed by RX handler and TX Handler resulting in setting of correctable error input (LMEM_ECC_CE = 1).

Action: MH proceeds with the read request. Read data from LMEM is provided to the requested module with OK response at HOST_AHB_HRESP.

Interrupts and Flags raised:
ERR_STS_LMEM_CERR_IRQ interrupt is raised to indicate that correctable error at LMEM is received.

1.5.8.4 RX Filtering Error

This section explains the causes and actions of MH in case of RX filtering not finished on time.

Cause: RX filtering process takes longer time to finish
In FMM, before the first word of next message is received at RX_MSG interface, RX filtering should be completed and results known. If this is not available, then RX filtering is not completed on time in FMM. In CTM, RX filtering process should be completed and results known by the time write request comes for the next cut through buffer. If this is not available, then RX filtering is not completed on time in CTM.

Action: MH action depends on the configuration bit RX_FILTER_CFG.AFAB.

FMM:
a) If this bit is set by the host, the received message is stored into the default fifo set in the RX_FILTER_CFG.DEFAULT_FIFO. RX message header R1 is updated with AFAB=1.
b) If this bit is not set by the host, current message is dropped and SAFETY_RX_ABORT_IRQ is set. Note that PRT message reception is not aborted.The RX_MSG interface still accepts the words from PRT but is not stored into LMEM in this case.

CTM:
a) If this bit is set by the host, the current message is stored into the default fifo set in the RX_FILTER_CFG.DEFAULT_FIFO. RX message header R1 is updated with AFAB=1. Block copy request is raised to continue with the current message reception.
b) If this bit is not set by the host, the current message under reception is dropped and SAFETY_RX_ABORT_IRQ is set. Note that PRT message reception is not aborted.The RX_MSG interface still accepts the words from PRT but is not stored into LMEM in this case

Interrupts and Flags raised:
SAFETY_RX_FILTER_ERR_IRQ is raised in all cases of RX filter error irrespective of whether the message is stored or not.

1.5.8.5 TX and RX Datapath Parity Error

This section explains the causes and actions of MH in case of data path parity error in TX and RX.

Cause: Parity mismatch in TX and RX Datapath.
There are three places where parity is checked in MH: VBM AHB interface before sending data to host, LMEM interface before writing data to LMEM, TX_MSG interface before sending to PRT for transmission.Refer Safety Mechanisms chapter in MH for more details.

Action: Anytime when there is parity mismatch in case of TX and RX data path, MH generates the output pulse PRT_STOP_IMMD_REQ instructing PRT to stop immediately. MH enable in the register MH_CTRL.MH_ENABLE is pulled low by MH. MH will hold its states till MH_ENABLE is made 1 by host or indirectly from PRT_ENABLE going high. If

data path parity mismatch happens during host write to LMEM via VBM (including LMEM Transparent access), HOST_AHB_HRESP=OK is given but the write to LMEM is discarded. If data path parity mismatch happens during host read from LMEM via VBM (including LMEM Transparent access), HOST_AHB_HRESP=ERROR is given with default read data 0xCAFECAFE. ERROR response lasts for two host clock cycles. Only transparent accesses are allowed during the time when MH_ENABLE=0.
If parity mismatch happens in the write path to LMEM by RX Handler, write transaction for the corrupted data is not issued at LMEM. If parity mismatch happens in the write path to PRT by TX Handler, write transaction for the corrupted data is not issued at TX_MSG interface.

Interrupts and Flags raised:
SAFETY_MISC_IRQ is triggered in case of a parity mismatch error. In case of parity mismatch in TX direction, the flag MH_STS1.TX_DP_PARITY_ERR is set and in case of RX direction parity mismatch, the flag MH_STS1.RX_DP_PARITY_ERR is set. In case of data path parity error during LMEM transparent access, MH_STS1.TX_DP_PARITY_ERR is set for write to LMEM and MH_STS1.RX_DP_PARITY_ERR is set for read from LMEM. The flags are reset only if MH is reset. Read access to MH_STS1 register will not clear data path parity error flags. Refer Chapter 1.6 PRT for interrupts and flags raised by PRT when stopped immediately.

1.5.8.6 Timestamp Parity Error

This section explains the causes and actions of MH in case of timestamp parity error in TX and RX.

Cause: Time stamp parity calculation is done in PRT module and in case of a parity mismatch, pulse on input TS_PARITY_ERR is set by PRT to MH. This is applicable in both TX and RX directions.

Action: In case of a timestamp parity mismatch in TX direction, bit 28 TS_PARITY_ERR, in word E1 of TX Event Fifo Queue element for that TX message is set to 1 while storing. In case of RX direction timestamp parity mismatch, the received message is dropped.

Interrupts and Flags raised:
SAFETY_MISC_IRQ is triggered in case of a timestamp parity mismatch for both TX and RX. The flag MH_STS1.TS_PARITY_ERR is also set in both TX and RX direction. The flag is reset only if PRT is restarted. Read access to MH_STS1 register will not clear TS parity error flag.

1.5.8.7 MH-PRT Sequence error

This section explains the causes and actions of MH in case of sequence error between MH and PRT.

Cause: Sequence error occur when there is unexpected error at TX_MSG and RX_MSG interfaces at MH which results in synchronization issues with PRT.
RX_MSG Interface : Sequence error is triggered in RX_MSG interface under following conditions:
i) When RX_MSG is waiting for SoS and any other code is received at RX_MSG_WUSER signal.
ii) When RX_MSG is waiting for word R1 and the RX_MSG_WUSER sends SOS, TS1 or TS2.
iii) When RX_MSG is waiting for word R2 and the RX_MSG_WUSER sends SOS, TS1 or TS2.
iv) When RX_MSG is waiting for TS1, RX_MSG_WUSER sends SOS, OK or TS2.
v) When RX_MSG is waiting for TS2, RX_MSG_WUSER sends SOS, OK or TS1.

TX_MSG Interface : Sequence error is triggered in TX_MSG interface under following conditions:
i) When TX_MSG is waiting for R0 and PRT sends code word other than OK or Wait4SOS.
ii) When TX_MSG is waiting for R1 and PRT sends code word other than OK,HFI,ARBLOST,RESTART or ACK
iii) When TX_MSG is waiting for Rn (n>1) and PRT sends code word other than OK,RESTART,DU or ACK. In case of R1=HFI or ARBLOST followed by R2=WAIT4SOS, sequence error is not triggered.

Action: MH generates the output pulse PRT_STOP_IMMD_REQ instructing PRT to stop immediately. MH enable in the register MH_CTRL.MH_ENABLE is pulled low by MH. MH will hold its states till MH_ENABLE is made 1 by host or indirectly from PRT_ENABLE going high. Only transparent accesses are allowed during the time MH_ENABLE=0. Pending LMEM request from TX Handler or RX Handler are also terminated, i.e LMEM_CS is pulled low if not served before the sequence error occurs.

Interrupts and Flags raised:
SAFETY_MISC_IRQ is triggered in case of sequence error. The flags MH_STS1.TX_DP_SEQ_ERR is set in case of error at TX_MSG interface and MH_STS1.RX_DP_SEQ_ERR is set in case of error at RX_MSG interface.

1.5.8.8 Error response during block copy transfer

This section explains the causes and actions of MH in case of error response received at CTM_RESP input during a block copy transfer.

Cause: In CTM, an error in the block copy transfer at SMEM, for both TX and RX direction is indicated by the host to the IP by giving the response at input signal CTM_RESP. CTM_RESP=2 indicates there has been an error in the block copy operation and the host could not complete the transfer.

Action: If the error is received during TX, the block copy transfer is dropped and the message transmission on the CAN bus is aborted. If the error is received during RX, the block copy transfer is aborted and the message under reception is discarded.

Interrupts and Flags raised:
The interrupt SAFETY_TX_ABORT_IRQ is raised for TX and SAFETY_RX_ABORT_IRQ is raised for RX. Also for both TX and RX, the safety interrupt SAFETY_CTB_BC_ERR_IRQ is raised by MH.

1.5.8.9 Timeout during block copy transfer

This section explains the cause and actions of MH in case of a timeout during block copy transfer in CTM.

Cause: In CTM, block copy timeout in TX directions occurs when there is no valid data in CTB to be transferred to PRT on time for the message currently under transmission to continue without data underrun.
Block copy timeout in RX direction occurs when both CTBs are occupied with pending blocks to be transferred to SMEM and there is no space for the incoming word to be stored. The incoming word can be from a new message or part of the same message which is there in the buffers. If there is no free buffer to store this new word, then the entire message is dropped.

Action: In case of a block copy timeout in TX, MH raises safety interrupt MH_CTB_BC_TO as well as the output signal CTM_BC_ERR. The CTM_TX_WREQ block copy request is pulled low. Eventually data underrun happens at PRT as a result and the message is aborted or discarded. XS_CAN IP does not wait for CTM_RESP in this case if not already received.
In case of a block copy timeout in RX, data over flow happens in PRT and the message which did not get space in the buffer, is aborted. CTM_BC_ERR output is raised if the same message gets dropped for which CTM_RX_RREQ was raised. XS_CAN IP does not wait for CTM_RESP if CTM_BC_ERR output is raised.

Interrupts and Flags raised:
TX Path: MH_CTB_BC_TO, MH_TX_ABORT, PRT_TX_DU.
RX Path: MH_CTB_BC_TO, MH_RX_ABORT, PRT_RX_DO.

1.5.8.10 Start address not received for Virtual Buffers

This section explains the cause and actions of MH in case if start address for virtual buffer is not received when expected.

Cause: In both FMM and CTM, for starting enqueue and dequeue of a message when the requests are active, host has to issue the start address of the corresponding virtual buffer. If host sends any other random address within the VB address range other than the start address of VB when the IP is expecting a start address, this error scenario occurs. This is also applicable for AHB burst transfers crossing the trigger address and extending to acceptable address range of the virtual buffer.

Action: Write operation to a random address other than start address is ignored with OK response and read operation from a random address other than start address is responded with default data 0xCAFECAFE and OK response.

Interrupts and Flags raised:
The interrupt ERR_STS_VB_NO_SA_IRQ is raised. The flags set are TXFQ_VB_NO_SA,TXPQ_VB_NO_SA,TEFQ_VB_NO_SA,RXFQ0_VB_NO_SA,RXFQ1_VB_NO_SA, CTB_VB_NO_SA

in MH_STS0 register based on the VB accessed. Note that this interrupt is raised only if the enqueue and dequeue requests from VBM are active, i.e there are valid contents to be enqueued and dequeued.
Exception: In FMM, if the host crosses the trigger address while reading the last message in the RX FIFOs i.e. no more valid message to be dequeued, ERR_STS_QUEUE_ACCESS_VIOLATION interrupt is raised instead of the ERR_STS_VB_NO_SA interrupt. Flags set are RXFQ0_EMPTY_RD or RXFQ1_EMPTY_RD in MH_STS0 register.

1.5.8.11 Virtual Buffer access violations

This section explains the cause and actions of MH in case if virtual buffer accesses are violated..

Cause 1: Read to write only buffers are write to read only buffers.
TXFQ and TXPQ are write only queues and any read access to the virtual buffers of TXFQ and TXPQ results in VB access violation. Similarly, RXFQ0, RXFQ1 and TEFQ are read only queues and any write access to these queues results in VB access violation.

Action: Write operation to a read only queue is ignored with OK response and read operation to a write only queue is responded with default data 0xCAFECAFE and OK response.

Interrupts and Flags raised:
The interrupt SAFETY_VB_ACCESS_VIOLATION_IRQ is raised. The flags corresponding to the queue accessed is also set in MH_STS1 register.
Read from TXFQ -> MH_STS1.TXFQ_VB_RD
Read from TXPQ -> MH_STS1.TXPQ_VB_RD
Write to TEFQ ->MH_STS1.TEFQ_VB_WR
Write to RXFQ0 -> MH_STS1.RXFQ0_VB_WR
Write to RXFQ1 -> MH_STS1.RXFQ1_VB_WR
Note that these flags are raised irrespective of whether the queues are enabled or not.

Cause 2: Read from CTB in TX direction when the write request is active and Write to CTB in RX direction when read request is active

Host must read from CTB in RX direction when block copy request is active for RX and host must write to CTB in TX direction when block copy request is active for TX. If this is violated then this error scenario occurs.

Action: Write operation to CTB is ignored with OK response and read operation from CTB is responded with default data 0xCAFECAFE and OK response.

Interrupts and Flags raised:
The interrupt SAFETY_VB_ACCESS_VIOLATION_IRQ is raised. The flags corresponding to the direction is also set in MH_STS1 register, MH_STS1.CTB_VB_TX_RD and MH_STS1.CTB_VB_RX_WR.

Cause 3: Non linear access to Virtual Buffers after start address is issued

Host must issue virtual buffer addresses in linear sequential order. If the host issues any other random address after issuing the start address, within VB address range, then this is a case of VB access violation.

Action: Write operation is ignored with OK response and enqueue is canceled and read operation is responded with default data 0xCAFECAFE and OK response. Message dequeue is canceled. In case of CTB, behavior is the same and the block copy is terminated for that message. The output pulse signal CTM_BC_ERR is also raised by VBM for both TX and RX in CTM. CTDMA does not send any CTM_RESP in this case.

Interrupts and Flags raised:
The interrupt SAFETY_VB_ACCESS_VIOLATION_IRQ is raised. The flags corresponding to the direction is also set in MH_STS1 register, TXFQ_VB_NONLIN_ACC,TXPQ_VB_NONLIN_ACC,TEFQ_VB_NONLIN_ACC, RXFQ0_VB_NONLIN_ACC, RXFQ1_VB_NONLIN_ACC, CTB_VB_NONLIN_ACC.

Cause 4: Cut Through buffer read with Burst length not equal to 16
When CTB read or write request is active, transaction of burst length not equal to 16 is discarded by MH. CTB read and write transfers are always of burst length=16.

Action: Write operation to CTB is ignored with OK response and read operation from CTB is responded with default data 0xCAFECAFE and OK response.

Interrupts and Flags raised:
The interrupt SAFETY_VB_ACCESS_VIOLATION_IRQ is raised. No flags raised.

1.5.8.12 Queue Access Violations

This section explains the causes and actions of MH in case of queue access violations.

Cause 1: Enqueue element too big for TXFQ and TXPQ.
In FMM, if the message getting enqueued has a size greater than the value configured in TXFQ_CFG.MAX_ELEM_SIZE register in case of TXFQ and TXPQ_CFG.SLOT_SIZE in case of TXPQ, then this error scenario occurs.
In CTM, the check is performed only if TXFQ_CFG.MAX_ELEM_SIZE <'d2 for TXFQ and TXPQ_CFG.SLOT_SIZE <'d20 in case of TXPQ.

Action: The message enqueue is aborted with HOST_AHB_HRESP= OK response. The state in VBM goes back to waiting for start address.

Interrupts and Flags raised:
The interrupt ERR_STS_QUEUE_ACCESS_VIOLATION is raised for both TXFQ and TXPQ. The corresponding flags set are MH_STS0.TXFQ_ELEM_TOO_BIG in case of TXFQ and MH_STS0.TXPQ_ELEM_TOO_BIG in case of TXPQ.

Cause 2: Write to full FIFO and read from empty FIFO. (No requests from VBM)
If the host tries to write to a full TXFQ or TXPQ or read from empty RXFQs and TEFQ, then this scenario occurs. There are no requests from VBM for enqueue and dequeue.
Action: Writes are ignored with HOST_AHB_HRESP=OK response and reads are responded with default read data 0xCAFECAFE and HOST_AHB_HRESP=OK.

Interrupts and flags raised:
The interrupt ERR_STS_QUEUE_ACCESS_VIOLATION_IRQ is raised . The corresponding flags are set according to the queue which is accessed. Note that these flags are raised only if host issues start address of virtual buffers.

TXFQ-> MH_STS0.TXFQ_FULL_WR
TXPQ-> MH_STS0.TXPQ_FULL_WR
RXFQ0-> MH_STS0.RXFQ0_EMPTY_RD
RXFQ1-> MH_STS0.RXFQ1_EMPTY_RD
TEFQ-> MH_STS0.TEFQ_EMPTY_RD

If host issues any other address in the virtual address range other than the start address, then only interrupt ERR_STS_QUEUE_ACCESS_VIOLATION_IRQ is raised, MH_STS0 flags are not set.

Cause 3: CTB read and CTB write (No request from VBM, CTM_RX_RREQ=0/CTM_TX_WREQ=0)
Host must perform read from CTB in RX only if there is valid block to be transferred to SMEM and also perform a write to CTB in TX only if there is need for a block for transmission and at least one block is free i.e. CTM_TX_WREQ or CTM_RX_RREQ is active. If this is violated, then this error scenario occurs. This case is applicable only if host issues the correct CTB VB start address.

Action: Writes are ignored with HOST_AHB_HRESP=OK response and reads are responded with default read data 0xCAFECAFE and HOST_AHB_HRESP=OK.

Interrupts and flags raised:
The interrupt ERR_STS_QUEUE_ACCESS_VIOLATION is raised and the flag set is MH_STS0.CTB_RD_WO_REQ/MH_STS0.CTB_WR_WO_REQ

Note that if host issues any other address other than start address of CTB VB, only ERR_STS_QUEUE_ACCESS_VIOLATION interrupt is raised and flags are not set.

Cause 4: Read write to queues that are disabled by user or mode

Host must not perform any access to queues that are disabled. Host must enable the fifo queues by setting the bits TXFQE,TXPQE,TEFQE,RXFQ0E,RXFQ1E,CTME in MH_CFG. Also, in FMM, CTB accesses are not allowed and in CTM, accesses to RXFQs are not allowed. If this is violated then queue access violation occurs, irrespective of whether RXFQs are enabled or disabled.

Action: Writes are ignored with HOST_AHB_HRESP=OK response and reads are responded with default read data 0xCAFECAFE and HOST_AHB_HRESP=OK.

Interrupts and Flags raised:
The interrupt ERR_STS_QUEUE_ACCESS_VIOLATION is raised

Cause 5: Access to MH when MH disabled

Host must access virtual buffers only when MH is enabled. Only transparent accesses are allowed when MH_ENABLE=0. If the host accesses any VB address when MH_ENABLE=0, then this also results in Queue access violation.

Action: Write operation to VB is ignored with OK response and read operation from VB is responded with default data 0xCAFECAFE and OK response.

Interrupts and Flags raised:
The interrupt ERR_STS_QUEUE_ACCESS_VIOLATION is raised. No flags raised.

1.5.8.13 Host accesses to reserved CREG address (Address range 0x00000- 0x 00FFC)

This section explains the cause and actions of MH in case if host accesses reserved register addresses.

Cause: In both FMM and CTM, host must not access any reserved addresses in the register memory area. Any access to a reserved address results in error scenario.

Action: Write transaction is ignored with ERROR response and read operation from a reserved address is responded with default data 0xCAFECAFE and ERROR response.

Interrupts and Flags raised:
The interrupt ERR_STS_MH_HOST_ARA_IRQ is raised.

1.5.8.14 Message Dequeued from TXFQ/TXPQ but not transmitted

This section explains the cause and actions of MH when a message needs to be removed from the queue without successfully transmission on the bus

Cause: If there is HFI (Header format Invalid) response from PRT at TX_MSG or the maximum retransmission limit is reached for a message, the message must be dequeued or removed from the queue and should not be considered for transmission again.

Action: Message is dequeued from the corresponding queue (TXFQ or TXPQ). TEQE is generated if requested by the message, with transmit status field updated according to the reason for dequeue. Refer TX Event FIFO Queue section for details.

Interrupts and Flags raised:
The interrupt corresponding to the queue from which the message is dequeued is raised. For TXFQ, ERR_STS_TXFQ_DEQ_NO_TX_IRQ is raised and for TXPQ, ERR_STS_TXPQ_DEQ_NO_TX is raised.

1.5.8.15 Immediate STOP of PRT resulting in PRT_ENABLE going low

This section explains the cause and actions of MH when PRT_ENABLE input signal goes low in the middle of operation.

Cause: If IMMD STOP command is written into PRT, PRT stops the operation immediately and pull the PRT_ENABLE output to MH, low.

Action: During TX: In FMM, if transmission is going on in MH when PRT_ENABLE goes low, then MH TX Handler handling the MH_PRT communication, goes to idle state. The TXFQ or TXPQ read pointer is not decremented since the message is not transmitted yet. The message under transmission is again considered for the next transmission based on the TX SCAN results. If the same message is transmitted again when PRT is restarted, retransmission counter is incremented. TX SCAN and VBM can continue when PRT_ENABLE is low.
In CTM, the same behavior mentioned above is applicable, also CTM_TX_WREQ raised is invalidated and the block copy data received at AHB for already raised request is ignored.
During RX: In FMM, if reception is going on in MH when PRT_ENABLE goes low, then MH RX Handler handling the MH_PRT communication, goes to idle state. The message under reception is discarded. RXFQx write pointer is not incremented since the message is dropped. RX Filtering if started is also aborted. In CTM, the same behavior mentioned above is applicable, also CTM_RX_RREQ raised is invalidated and the block copy data read at AHB for already raised request is ignored.

Interrupts and Flags raised:
MH raises SAFETY_RX_ABORT_IRQ in case of RX and SAFETY_TX_ABORT_IRQ in case of TX.The register bit MH_STS0.PRT_ENABLE goes to 0 from 1.

1.5.8.16 LMEM Protected range accesses

This section explains the cause and actions of MH when an access is done in the LMEM protected range addresses .i.e. address outside the LMEM_PC_START_ADDR and LMEM_PC_END_ADDR range.

Cause: Possible sources of LMEM accesses are:
1) Read or Write by VBM, TX Handler and RX Handler- Host access in virtual address range, or internal module TX Handler and RX Handler can issue read or write transactions at LMEM.
2) LMEM transparent access by host- Host accesses in LMEM transparent range

Action: In case if LMEM access by VBM, RX Handler or TX Handler falls in the protected range, MH and PRT are immediately stopped and there is no recovery unless XS_CAN IP is reset by the host. If the last address of a read burst access by host falls in protected area, host must discard the entire burst.
In case of LMEM transparent access resulting in protected range address, MH and PRT are not stopped.
In all cases, write transaction is ignored with OK response and read transfer is given OK response with default read data 0xCAFECAFE.

Interrupts and Flags raised:
MH raises SAFETY_LMEM_PROT_ERR_IRQ in case of protected range accesses. The output signal LMEM_PROTECT_EVENT (high for one HOST_CLK period) is also generated at the top level. Except for LMEM transparent accesses, PRT_STOP_IMMD_REQ is also generated by MH for other LMEM accesses.

1.5.8.17 TX Data Underrun in PRT due to high LMEM latency

This section explains the cause and actions when a data underrun scenario happens in PRT due to LMEM_READY signal latency.

Cause: During transmission, TX data underrun can occur in PRT if the time taken to fetch the data from the LMEM is too long (FMM and CTM).

Action: When the MH receives a *PRT_TX_DU* interrupt, it sends a dummy word to the PRT to obtain the DU response. The remaining words are not transmitted to the PRT. The same message can be tried for retransmission if applicable. Pending LMEM read request from TX Handler is also terminated, i.e LMEM_CS is pulled low if not served before the error occurs.

Interrupts and Flags raised:
Safety interrupt *PRT_TX_DU*

1.5.9 Application Information

This section provides some general information on MH performance, flags and status registers to look at and some guidelines related to cluster mode.

1.5.9.1 Cluster

The same LMEM can be shared across several Message Handler, but several points need to be highlighted. A trade-off needs to be found to ensure every MH will get enough time to complete their RX Filter process as well as their TX-Scan for a given LMEM bandwidth.

• The worst scenario on RX path is defined when all MH in a cluster is receiving an RX message at the same time. Therefore, it is essential to ensure the available bandwidth on the LMEM can support the RX filter process from all concurrent MH. Several measures can be taken to lower the bandwidth for a given value: limit the number of RX filter elements and the number of comparison (1 or 2) per filter element.

• The worst scenario on TX path is defined by all TX FIFO queues active for every MH as well as new TX messages being added to all TX Priority Queue slots. As one message is added or sent at a time for every MH, the impact of the TX-Scan may be limited but may play an important role by generating more arbitration occurrences.

• The read latency to access the LMEM is a common factor for all MH and should be as low as possible. This access time is driven the overall performances when in cluster mode.

1.5.9.2 Performances

Several processing times have a direct impact on the overall MH performances,
The minimum MH core clock frequency is driven by several parameters:
• The maximum number of RX filter elements to support
• The CAN CC, CAN FD, and CAN XL bit rates (Arbitration and Data Phase)
• The LMEM read latency
• The maximum number of TX Priority Queue slots
To estimate the minimum core clock frequency to set, please refer to excel file [4]. One must keep in mind that the computed value is a minimum. Other clock frequency constraints may require a higher clock speed on the MH.

1.5.9.3 TX SCAN

The TX-Scan does compute the highest priority message from amongst TX FIFO queue head element and the TX Priority Queue slots. The processing time is mainly linked to the number of TX Priority Queue slots being set active. The higher the number of slots, the higher the bandwidth from the LMEM. The sooner the result is known the better the expected transmission order is. For more detail on TX-Scan refer to the TX-Scan chapter in TX Handler Section.

1.5.9.4 RX FILTER

The process of filtering is started as soon as the first RX message header data is received. The Classical CAN with a low bit rate does provide more margin to complete the RX filtering in time. The critical path is defined when receiving CAN FD frame with no payload data.
The HOST_CLK clock defines the RX Filtering performance, i.e. the amount of RX Filter comparisons that can be done for a worst case message (short and high bit rate).
The XS_CAN supports a large number for RX Filters. However, typically applications need up to 32 or up to 64 RX Filters. Integrator can dimension the HOST_CLK frequency to support e.g. 32 to 64 RX Filters with 2 comparisons under worst case conditions. In this case, the order of the RX Filter element configurations in the LMEM is not relevant, because dimensioning is done for the worst case.
It is recommended to use the following rules to sort the RX Filter element configurations.
This brings two advantages :
(a) reduction of the filtering time for critical cases
(b) increase of the number of supported RX Filter configurations.

1. First order: Short Data Field before long data field

2. Second order: First FD, second XL, third CC

3. Third order:

1. One comparison with R0
2. One comparison with R2
3. Two comparisons with R0/R2 before two comparisons

Example for a mixed network with FD and XL messages:
FD messages with short payload of up to 8 byte and one comparison, sorted by data field size
XL messages with short payload of up to 8 byte and one comparison, sorted by data field size
XL messages with short payload of up to 8 byte and two comparison, sorted by data field size
FD messages with longer payload
XL messages with longer payload

1.5.9.5 Address Alignment

1) TXFQ start and end addresses programmed into **MH.TXFQ_LMEM_SA** and **MH.TXFQ_LMEM_EA** register must be 64bytes aligned.
2) TXPQ start address programmed into **MH.TXPQ_LMEM** must be 64bytes aligned.
3) TXPQ slot size programmed into **MH.TXPQ_CFG** must be in granularity of 4bytes (1 word)
4) RXFQ0/1 start and end addresses programmed into **MH.RXFQx_SA** and **MH.RXFQx_EA** must be 64bytes aligned.
5) TEFQ start address programmed into **MH.TEFQ_LMEM** must be 64byte aligned
6) CTB start address programmed into **MH.CTB_LMEM** must be 64bytes aligned.

1.5.10 Detailed Design Information

1.5.10.1 Port Description

Table 78. Port List (page 1 of 4)

Signal	Bit Width	I/O	Description	Active Level	Clock Domain
CLOCKS					
HOST_CLK	1	I	Host Clock	na	na
HOST_CLK_AON	1	I	Clock for APB interface	na	na
RESETS					
RESET_N	1	I	Reset core	Low	HOST_CLK
INTERRUPTS					
FUNCTIONAL INTERRUPTS					
FUNC_CTB_TX_BC_REQ_IRQ	1	O	Indicates a block copy transer pending in TX direction	High	HOST_CLK
FUNC_CTB_RX_BC_REQ_IRQ	1	O	Indicates a block copy transer pending in RX direction	High	HOST_CLK
FUNC_TEFQ_DEQ_IRQ	1	O	Indicates a message is dequeued from TEFQ	High	HOST_CLK
FUNC_TEFQ_ENQ_IRQ	1	O	Indicates a message is enqueued from TEFQ	High	HOST_CLK
FUNC_TXPQ_ENQ_IRQ	1	O	Indicates a message is enqueued from TXPQ	High	HOST_CLK
FUNC_TXFQ_ENQ_IRQ	1	O	Indicates a message is enqueued from TXFQ	High	HOST_CLK
FUNC_RXFQ0_DEQ_IRQ	1	O	Indicates a message is dequeued from RXFQ0	High	HOST_CLK
FUNC_RXFQ1_DEQ_IRQ	1	O	Indicates a message is dequeued from RXFQ1	High	HOST_CLK
FUNC_RXFQ0_ENQ_IRQ	1	O	Indicates a message is enqueued from RXFQ0	High	HOST_CLK
FUNC_RXFQ1_ENQ_IRQ	1	O	Indicates a message is enqueued from RXFQ1	High	HOST_CLK
SAFETY INTERRUPTS					
SAFETY_RX_FILTER_ERR_IRQ	1	O	Error detected while RX filtering in progress	High	HOST_CLK
SAEFTY_MISC_IRQ	1	O	Different safety issues	High	HOST_CLK

Table 78. Port List (page 2 of 4)

Signal	Bit Width	I/O	Description	Active Level	Clock Domain
SAFETY_VB_ACCESS_VIOLATION_IRQ	1	O	Indicates there is a violation in VB access	High	HOST_CLK
SAFETY_TX_ABORT_IRQ	1	O	TX message aborted	High	HOST_CLK
SAFETY_RX_ABORT_IRQ	1	O	RX message aborted	High	HOST_CLK
SAFETY_LMEM_PROT_ERR_IRQ	1	O	Indicates host access to an LMEM address outside the range of LMEM start and end addresses	High	HOST_CLK
SAFETY_LMEM_TO_ERR_IRQ	1	O	Indicates a timeout at LMEM interface	High	HOST_CLK
SAFETY_LMEM_UERR_IRQ	1	O	Uncorrectable error is detected at LMEM interface	High	HOST_CLK
SAFETY_CTB_BC_TO_IRQ	1	O	Indicates timeout for block copy request.i.e block copy not finished in time before data underrun or overflow at CTB	High	HOST_CLK
SAFETY_CTB_BC_ERR_IRQ	1	O	Indicates that error response is received at CTM_RESP input signal	High	HOST_CLK
ERROR INTERRUPTS					
ERR_STS_VB_NO_SA_IRQ	1	O	Indicates MH has received another address instead of VB start address	High	HOST_CLK
ERR_STS_QUEUE_ACCESS_VIOLATION_IRQ	1	O	Indicates there has been a violation in accessing the queues	High	HOST_CLK
ERR_STS_PRT_STOP_IRQ	1	O	MH indicates that the PRT is stopped	High	HOST_CLK
ERR_STS_LMEM_CERR_IRQ	1	O	Correctable error is detected at the LMEM interface	High	HOST_CLK
ERR_STS_RXFQ0_DROP_IRQ	1	O	Indicates a new message to be stored into RXFQ0 is dropped	High	HOST_CLK
ERR_STS_RXFQ1_DROP_IRQ	1	O	Indicates a new message to be stored into RXFQ1 is dropped	High	HOST_CLK
ERR_STS_TEFQ_DROP_IRQ	1	O	Indicates a new TEQE is dropped	High	HOST_CLK
ERR_STS_TXPQ_DEQ_NO_TX_IRQ	1	O	Indicates a message is dequeued from TXPQ but is not transmitted	High	HOST_CLK
ERR_STS_TXFQ_DEQ_NO_TX_IRQ	1	O	Indicates a message is dequeued from TXFQ but is not transmitted	High	HOST_CLK
ERR_STS_MH_HOST_ARA_IRQ	1	O	Host accesses to reserved addresses	High	HOST_CLK
DMA REQUEST SIGNALS					
TXFQ_WREQ	1	O	TX FIFO Queue interrupt from VBM	High	HOST_CLK
TXPQ_WREQ	1	O	TX Priority Queue interrupt from VBM	High	HOST_CLK
TXEF_RREQ	1	O	TX Event FIFO Queue interrupt from VBM	High	HOST_CLK
RXFQ0_RREQ	1	O	RX FIFO Queue 0 interrupt from VBM	High	HOST_CLK
RXFQ1_RREQ	1	O	RX FIFO Queue 1 interrupt from VBM	High	HOST_CLK
CTM_TX_WREQ	1	O	Cut-Through Buffer interrupt for TX	High	HOST_CLK
CTM_RX_RREQ	1	O	Cut-Through Buffer interrupt for RX	High	HOST_CLK
AHB INTERFACE					
VBM_AHB_HWRITE	1	I	Read or write bit	na	HOST_CLK
VBM_AHB_HADDR	19	I	Address	na	HOST_CLK
VBM_AHB_HWDATA	32	I	Write data	na	HOST_CLK
VBM_AHB_HSELX	1	I	Select	High	HOST_CLK
VBM_AHB_HTRANS	2	I	Transfer type	na	HOST_CLK
VBM_AHB_HBURST	3	I	Burst type	na	HOST_CLK
VBM_AHB_HSIZE	3	I	Burst size	na	HOST_CLK

Table 78. Port List (page 3 of 4)

Signal	Bit Width	I/O	Description	Active Level	Clock Domain
VBM_AHB_HPROT	4	I	Protection type	na	HOST_CLK
VBM_AHB_HREADYOUT	1	O	Ready	High	HOST_CLK
VBM_AHB_HREADYIN	1	I	Output of top mux to indicate extension of data phase by bridge	High	HOST_CLK
VBM_AHB_HRESP	2	O	Response	na	HOST_CLK
VBM_AHB_HRDATA	32	O	Read data	na	HOST_CLK
DESCRIPTOR PORTS (CT MODE)					
CTM_RESP	2	I	Cut-through block copy completion response	na	HOST_CLK
CTM_EN	1	O	CTM Enable	na	HOST_CLK
CTM_BC_ERR	1	O	CTM Block Copy Error output from MH	na	HOST_CLK
CTM_DESC	64	O	Descriptor for Source, Destination, Size	na	HOST_CLK
TX MSG INTERFACE					
TX_MSG_WVALID	1	O	Write data valid	High	HOST_CLK
TX_MSG_WDATA	32	O	Write data	na	HOST_CLK
TX_MSG_WUSER	2	O	Write user code to control the sequence	na	HOST_CLK
TX_MSG_WREADY	1	I	Write ready from PRT	High	HOST_CLK
TX_MSG_BVALID	1	I	Response data valid	High	HOST_CLK
TX_MSG_BUSER_STATUS	3	I	Response user data status	na	HOST_CLK
TX_MSG_BUSER_TS	64	I	Response user data timestamp	na	HOST_CLK
TX_MSG_BREADY	1	O	Response ready from MH	High	HOST_CLK
RX MSG INTERFACE					
RX_MSG_WVALID	1	I	Write data valid	High	HOST_CLK
RX_MSG_WDATA	32	I	Write data valid	na	HOST_CLK
RX_MSG_WUSER	3	I	Write user data	na	HOST_CLK
RX_MSG_WREADY	1	O	Write ready	High	HOST_CLK
MISC SIGNALS					
MH_EN	1	O	MH Enable	High	HOST_CLK
PRT_STOP_IMMD_REQ	1	O	Instructs PRT to perform an immediate stop	High	HOST_CLK
PRT_TX_DU	1	O	Indicates there is a Data Underrun in PRT during TX message transmission	High	HOST_CLK
PRT_RX_DO	1	I	Indicates there is a Data Overflow in PRT during RX message reception	High	HOST_CLK
PRT_ENABLE	1	I	Serial CAN Bus Communication enabled	High	HOST_CLK
HOST_CLOCK_ACTIVE	1	I	Must be set when the MH core clock is active	High	HOST_CLK
LMEMPC_START_ADDR	18	I	LMEM protection config start address	na	ASYNC
LMEMPC_END_ADDR	18	I	LMEM protection config end address	na	ASYNC
LMEM_PROTECT_EVENT	1	O	LMEM protection event	Pulse High	HOST_CLK
TS_PARITY_ERR	1	I	Time Stamp Parity Error for TX and RX Path; it is checked when NO_SAFETY_NO_CTM is 0	Pulse High	HOST_CLK
NO_SAFETY_NO_CTM	1	I	When set to 1, Time Stamp Parity check is disabled in PRT	High	na
RX_MSG_STORE_SUCC	1	O	Is set to 1 when RX messages are successfully stored in an RX FIFO	Pulse High	HOST_CLK
RX_MSG_ALERT	1	O	Set to 1 when incoming message matches with a filter with ALERT bit set to1	High	HOST_CLK

Table 78. Port List (page 4 of 4)

Signal	Bit Width	I/O	Description	Active Level	Clock Domain
LMEM INTERFACE					
LMEM_READY	1	I	When set to 0, Bus is busy and when it is 1, the transaction is complete	High	HOST_CLK
LMEM_ECC_UE	1	I	ECC uncorrected error	Pulse High	HOST_CLK
LMEM_ECC_CE	1	I	ECC corrected error	Pulse High	HOST_CLK
LMEM_RDATA	32	I	Read data	na	HOST_CLK
LMEM_ADDR	16	O	Address	na	HOST_CLK
LMEM_WDATA	32	O	Write data	na	HOST_CLK
LMEM_CS	1	O	Chip select	High	HOST_CLK
LMEM_WE	1	O	Write enable	na	HOST_CLK
LMEM_BURST	1	O	LMEM burst	na	HOST_CLK
LMEM_BURST_LEN	2	O	LMEM burst length	na	HOST_CLK
HOST APB INTERFACE					
REG_APB_PADDR	12	I	Address	na	HOST_CLK_AON
REG_APB_PSELX	1	I	Select	High	HOST_CLK_AON
REG_APB_PENABLE	1	I	Enable	High	HOST_CLK_AON
REG_APB_PWRITE	1	I	Read or write bit	High	HOST_CLK_AON
REG_APB_PRDATA	32	O	Read data	na	HOST_CLK_AON
REG_APB_PWDATA	32	I	Write data	na	HOST_CLK_AON
REG_APB_PREADY	1	O	Used to extend an APB transfer by the Completer	High	HOST_CLK_AON
REG_APB_PSLVERR	1	O	Transfer error	High	HOST_CLK_AON
REG_APB_PPROT	3	I	Protection type	na	HOST_CLK_AON
REG_APB_PSTRB	4	I	Write strobe	High	HOST_CLK_AON
HARDWARE DEBUG PORT					
HDP	16	O	Hardware Debug Port	na	na

1.6 PRT - Protocol Controller

This document describes the PRT, the CAN XL Protocol Controller, and its interfaces with the host controller, a message handler frontend, and the CAN transceiver.

1.6.1 Overview

The PRT is a CAN XL Protocol Controller that can be integrated into different CAN modules. The PRT performs CAN communication as specified in ISO 11898-1:2024 (CAN CC, CAN FD and CAN XL) [1]. The bitrate can be configured to values up to 20MBit/s at a clock speed of 160MHz, depending on the resources of the message handler and on the used semiconductor technology. For the connection to the physical layer, additional transceiver hardware is required, which is connected via GPIO ports or may be integrated into the CAN module (see chapter “Transceiver Interface”). The PRT does

not provide internal buffering of frames, so that data has to be transferred by IP internal Message Busses in 32 bit slices in real-time while (de)-serialization on the CAN Bus. Thus, single data transfers at the internal Message Busses are closely time-synchronized to the schedule at the CAN bus.
ISO 11898-1 specifies the Information Processing Time (IPT). The XS_CAN has an IPT of zero, meaning the data for the next bit is available at the first clock edge after the sample point.

1.6.2 Features

- CAN CC, CAN FD and CAN XL as specified in ISO 11898-1:2024
- CAN CC bit rate up to 1Mbps
- Arbitration phase bit rate up to 1Mbps for CAN FD and for CAN XL
- CAN FD data phase bit rate up to 8Mbps at a clock speed of 80MHz or 160MHz
- CAN XL data phase bit rate up to 20Mbps at a clock speed of 160MHz
- CAN FD Light Commander data phase bit rate up to 8Mbps
- APB 4 slave interface (HOST_APB) (compliant to AMBA 4 ARM Ltd protocol, see [6])
- 32 bit data bus width interface
- 512 bytes address space register, 8 bit address bus width
- Dedicated Timebase interface

1.6.3 Block Diagram

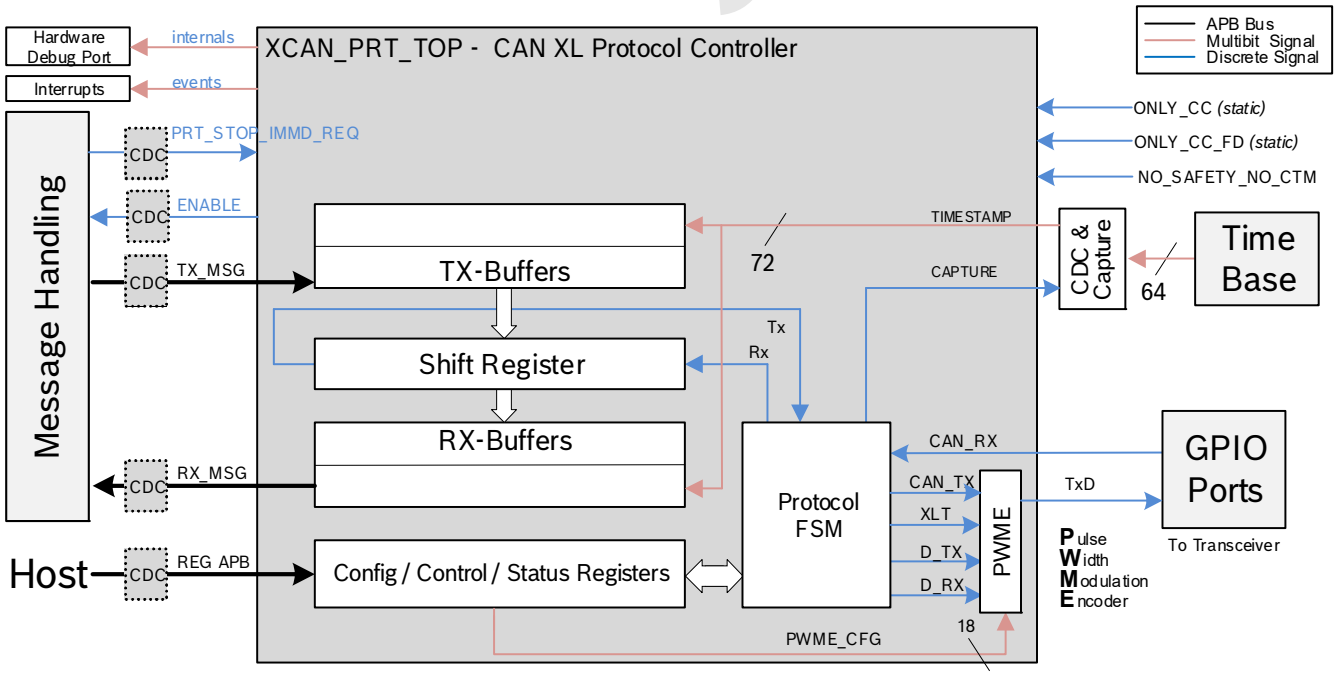


Figure 24. PRT Block Diagram

Figure above shows principle of the PRT's functions. When the PRT and the message handler operate in different clock domains, they are connected via CDC modules. The Time Base is captured inside a CDC module, triggered by the PRT's output signal CAPTURE.

The PRT consists of the CAN Protocol FSM, an Rx/Tx Shift Register, a set of interface registers for configuration, control, and status information as well as interfaces to the message handler for received messages and for messages to be transmitted. The PRT is not designed to store complete messages, there is only transient caching for two memory words for each direction during reception or transmission.

A separate message handler is needed for the storage of whole messages as well as for functions like acceptance filtering, sorting of received messages into specific message buffers, and ordering the sequence of messages that are

requested for transmission. Messages are streamed between message handler and PRT as sequences or 32 bit data words.

The host accesses the PRT's registers via REG_APB, an AMBA APB 4 interface, for configuration, control, and status information.

Note that the PRT submodules in the block diagram are named "xcan_" and not "xcans" in order to align with the RTL code which is reused from XCAN project.

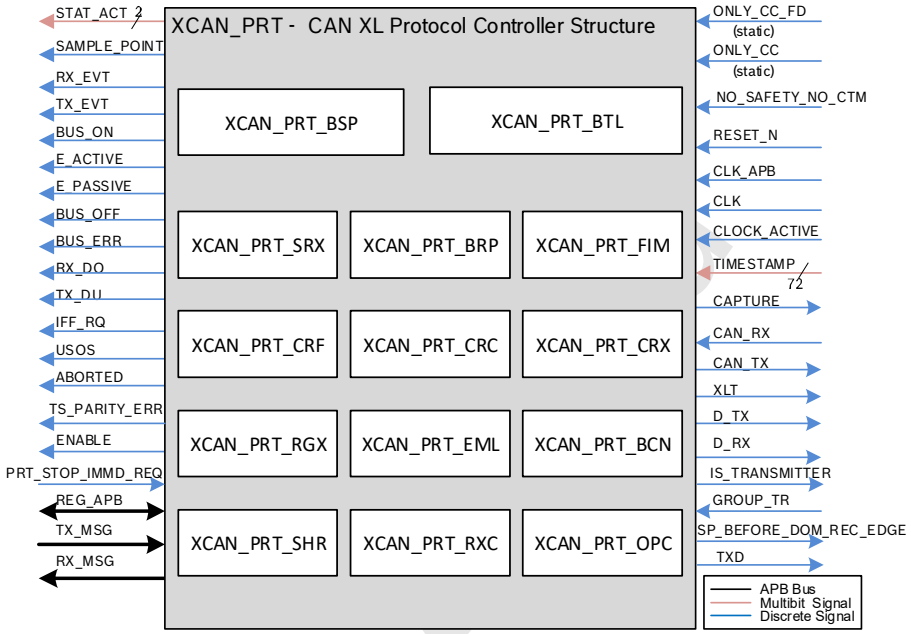


Figure 25. PRT Internal Structure

XCAN_PRT_BSP is the Bit Stream Processor, a sequencer controlling the data stream between transmit and receive shift register and the bus line. The BSP also controls the BTL, BCN, OPC, CRF, CRC, CRX, and EML such that the processes of reception, arbitration, transmission, and error signaling are performed according to the CAN protocol specification. The BSP also provides signals to the RXC indicating when a valid message was received and also to the SHR when the transmit sequence finishes after a successful transmission. Note that the automatic retransmission of messages which have been corrupted by noise or other error external error conditions on the bus line is not handled by the BSP.

XCAN_PRT_BTL is the Bit Timing Logic, monitoring the bus line input CAN_RXD and handling the busline related bit timing according to the CAN protocol. The BTL synchronises on a recessive to dominant busline transition at Start of Frame (hard synchronisation) and on any further recessive to dominant busline transition, if the CAN controller itself does not transmit a dominant bit (resynchronisation). The BTL uses three sets of configurable time segments to compensate for the propagation delay time and for phase shifts and to define the position of the Sample Point in the three bit times for CAN CC, CAN FD, and CAN XL.

XCAN_PRT_SRX synchronizes the asynchronous bus line input CAN_RXD to the CAN clock using a 2- FF synchronizer structure (user-replaceable module) before the signal is used inside the PRT.

XCAN_PRT_BRP is the Bit Rate Prescaler that generates the CAN time quantum from the CAN clock period and the NBTP.BRP configuration.

XCAN_PRT_FIM is the Fault Injection Module as described in the section 1.6.6.12. XCAN_PRT_CRF, XCAN_PRT_CRC, and XCAN_PRT_CRX are the Cyclic Redundancy Check modules for the CAN FD (CRF), CAN CC (CRC) and the CAN XL (CRX) frame formats. These modules divide the data stream by the code generator polynomials in order to generate the CRC sequences to be transmitted after the data bytes and to check the CRC sequences of incoming messages.

XCAN_PRT_RGX is the Register block, containing all readable and writable registers as well as the APB interface control.

XCAN_PRT_EML is the Error Management Logic, responsible for the fault confinement of the CAN device. Its counters, the Receive Error Counter and the Transmit Error Counter, are incremented and decremented by commands from the Bit

Stream Processor. According to the values of the error counters, the CAN controller is set into the states error active, error passive and busoff. The CAN controller is error active, if both error counters are below the error passive limit of 128. It is error passive, if at least one of the error counters equals or exceeds 128. It goes busoff, if the Transmit Error Counter equals or exceeds the busoff limit of 256. The device remains in this state, until the busoff recovery sequence is finished (see CAN specification).

XCAN_PRT_BCN is the Bit Counter, controlling the bit stuffing function and the length of the segments of the CAN frames.

XCAN_PRT_SHR contains the Shift Register that converts between parallel data words and the serial CAN bit stream as well as the two-stage Tx-buffers that are loaded by the MH via the TX_MSG interface as well as the TX_MSG interface itself. This interface is controlled by the MH.

XCAN_PRT_RXC contains the two-stage Rx-buffers that are loaded from the Shift Register and transferred to the MH via the RX_MSG interface as well as the RX_MSG interface itself. This interface is controlled by the PRT.

XCAN_PRT_OPC is the Output Controller that provides the output signals CAN_TX, XLT, D_TX, and D_RX to the transceiver and to the PWME module.

1.6.4 Hardware Interface

Table 79. PRT HW Interface

Pin	Name	I/O	Description
CAN_RX		I	Serial CAN receive input Transceiver
TXD		I	Serial CAN transmit output to Transceiver/PWME

1.6.5 Software Interface

1.6.5.1 PRT MAP

1.6.5.1.1 Overview

Table 80. Address Blocks list

Address Block	Offset	Range	Description
PRT_STATUS	0x0000	0x0020	Usage: register Status information of the PRT
PRT_EVENT	0x0020	0x0020	Usage: register Event information of the PRT
PRT_CONTROL	0x0040	0x0020	Usage: register Control of the PRT during runtime
PRT_CONFIGURATION	0x0060	0x0020	Usage: register Configuration of the PRT before runtime

Table 81. Registers overview (page 1 of 3)

Register name	Offset	Mode	Description
PRT_STATUS			
ENDN	0x0000	R	Register size: 32 Endianness Test Register
STAT0	0x0004	R	Register size: 32 PRT Status 0 Register
STAT1	0x0008	R	Register size: 32 PRT Status 1 Register
STATISTIC_COUNTER	0x000C	R	Register size: 32 PRT Statistic Counter Register

Table 81. Registers overview (page 2 of 3)

Register name	Offset	Mode	Description
PRT_EVENT			
EVNT	0x0020	RW	Register size: 32 Event Status Flags Register The EVNT Register contains event status flags. The flags is set by the PRT when specific events occur. A software reset clears all flags. A host write access to this register, writing a 1 to a specific flag, clears that flag. When a host write access occurs concurrently with a set condition for a flag, the flag is set.
PRT_CONTROL			
LOCK	0x0040	W	Register size: 32 Unlock Sequence Register Writing a sequence of specific data words enables the activation of control commands in the registers CTRL and FIMC.
CTRL	0x0044	W	Register size: 32 Control Register Writing to this register controls the CAN protocol operation. When writing to this register, only one of the four bits TEST, SRES , STRT, or STOP may be written to 1. Multiple CTRL register bits can be set simultaneously; however, such write access has no functional impact on the RTL, and the PRT state remains unchanged. The bit IMMD may be written to 1 together with the bit STOP, but not together with one of the other bits.
FIMC	0x0048	RW	Register size: 32 Fault Injection Module Control Register Writing the fault injection position number requires the application of the test mode key sequence before writing to FIMC.
TEST	0x004C	RW	Register size: 32 Hardware Test Functions Register This register is writable after the hardware test mode functions are enabled by writing the test mode key sequence to LOCK and CTRL registers. While the hardware test mode functions are not enabled, this register is read-only. The hardware test mode functions are disabled and cleared by the software reset of the PRT.
PRT_CONFIGURATION			
MODE	0x0060	RW	Register size: 32 Operating Mode Register Configuration register that is writable while the CAN communication is stopped and that is read-only after the CAN communication is started. This register defines separate operating mode options. The four configuration bits FDOE, XLOE, EFDI, and XLTR, are interrelated according to table Frame Formats defined in Operating Mode chapter.
NBTP	0x0064	RW	Register size: 32 Arbitration Phase Nominal Bit Timing Register Configuration register that is writable while the CAN communication is stopped and that is read-only after the CAN communication is started. This register defines the Nominal Bit Timing as defined in [1].

Table 81. Registers overview (page 3 of 3)

Register name	Offset	Mode	Description
DBTP	0x0068	RW	Register size: 32 CAN FD Data Phase Bit Timing Register Configuration register that is writable while the CAN communication is stopped and that is read-only after the CAN communication is started. This register defines the FD Data Phase Bit Timing as defined in [1].
XBTP	0x006C	RW	Register size: 32 CAN XL Data Phase Bit Timing Register Configuration register that is writable while the CAN communication is stopped and is read-only after the CAN communication is started. This register defines the XL Data Phase Bit Timing as defined in [2].
PCFG	0x0070	RW	Register size: 32 PWME Configuration Register Configuration register that is writable while the CAN communication is stopped and is read-only after the CAN communication is started. This register defines the parameters needed for the PWM coding (as described in [3]) in the PWME module for CAN XL transceivers with switchable operating modes

1.6.5.1.2 PRT_STATUS Address Block

Table 82. ENDN register

Endianness Test Register

Bit	Field	Mode	Initial Value	Description
31:0	ETV	R	0x87654321	The purpose of this register is to identify the beginning of the PRT address map in a memory dump and to check the proper endianness data byte mapping when the data word is routed through different busses.

Table 83. STAT0 register (page 1 of 2)

PRT Status 0 Register

Bit	Field	Mode	Initial Value	Description
31:24	TEC	R	0x0	The CAN protocol Transmit Error Counter. A software reset does not change the value in this register. When the Bus-Off recovery sequence is finished, the error counter TEC is cleared. When the increment TEC+8 would result in a value > 255 (carry-flag), the TEC is kept unchanged, but BO is set.
23	RP	R	0x0	The Passive flag of the CAN protocol Receive Error Counter. This flag is set on an error condition that would have caused an increment of the Receive Error Counter to a value beyond its 7 bit range.
22:16	REC	R	0x0	The CAN protocol Receive Error Counter. A software reset does not change the value in this register. When the Bus-Off recovery sequence is finished, the error counter REC is cleared. The REC is a 7-bit-counter, together with the Error-Passive flag EP. When the increment REC+1 or REC+8 would result in a value > 127 (carry-flag), the REC is kept unchanged, but EP is set. When EP is set but REC is below 127 and further errors are detected with an REC+1 condition, the REC will be incremented until it reaches 127. At the reception of a valid message, the REC-1 decrements the actual value of the REC by one AND clears the Error-Passive flag EP.
15:8	TDCV	R	0x0	Transmitter Delay Compensation delay value. A software reset clears the TDV bit field to 0x00. This register shows the sum of the measured delay plus the configured offset, giving the position of the secondary sample point. It is updated for each frame transmission that includes a data phase.
7	BO	R	0x0	This node is in Bus-Off state. This flag is set on an error condition that would have caused an increment of the Transmit Error Counter to a value beyond its 8 bit range. When the PRT enters Bus-Off state, BO is set to 1 and CAN protocol operation is stopped. When the Bus-Off recovery sequence is finished, BO is cleared.
6	EP	R	0x0	This node is in Error-Passive state. When the Bus-Off recovery sequence is finished, the EP bit is cleared.

Table 83. STAT0 register (page 2 of 2)

PRT Status 0 Register

Bit	Field	Mode	Initial Value	Description
5	FIMA	R	0x0	Fault Injection Module Activated, see Safety Measures chapter
4	CLKA	R	0x1	The actual value of the CLOCK_ACTIVE input signal, see Starting and Stopping the Module chapter. As the clock must be active when a reset is performed, the default value should be 1.
3	STP	R	0x0	Waiting for end of actual message after STOP command, see Starting and Stopping the Module chapter
2	INT	R	0x0	This node is integrating into CAN bus traffic
1:0	ACT	R	0x0	The current activity of this node: 0b00: inactive state 0b01: Idle 0b10: Receiver 0b11: Transmitter When the CAN protocol operation is stopped, ACT changes to 0b00 and INT changes to 0. When the CAN protocol operation is started, INT is set to 1, but ACT remains at 0b00 until the CAN protocol bus idle detection condition is met, then it changes to 0b01 and INT changes to 0. When the CAN protocol operation is started while BO is set, the PRT remains in integrating state (INT=1 and ACT=0b00) until the Bus-Off recovery sequence is finished, then it changes to 0b01 and INT changes to 0. When PRT detects a protocol exception event (see [1], chapter 10.9.5), ACT changes to 0b00 and INT changes to 1 until the CAN protocol bus idle detection condition is met, then ACT changes to 0b01 and INT changes to 0. ACT changes from 0b01 to 0b10 when the PRT has received a Start-of-Frame from the CAN bus. ACT changes from 0b01 to 0b11 when the PRT has sent a Start-of-Frame to the CAN bus. ACT changes from 0b11 to 0b10 when the PRT loses arbitration during a transmission. ACT changes from 0b10 to 0b01 or from 0b11 to 0b01 when the PRT detects the second bit of intermission (see [1], chapter 10.4.6.2) to be recessive.

Table 84. STAT1 register

PRT Status 1 Register

Bit	Field	Mode	Initial Value	Description
7	TSS_EN	R	0x0	This bit shows the actual value of the TSS_EN input port.
6	RX_TSS	R	0x0	Indicates whether this CAN IP Module is currently in a mode where it suppresses the sending of ACK bits and PWM coding at its CAN transmit output port. RX_TSS is set when the following condition is met: 1) The Transceiver Sharing Switch mode is enabled (input TSS_EN = '1') AND 2) There is at least one transmitter in the transceiver sharing group (input GROUP_TR = '1') AND 3) This CAN IP Module is currently in receiver state. RX_TSS is cleared when the received frame ends or when an error condition is detected.
5:3	TX_FSM_STATUS	R	0x0	Rx FSM Status of xcan_prt_shr RTL
2:0	RX_FSM_STATUS	R	0x0	Rx FSM Status of xcan_prt_rxc RTL

Table 85. STATISTIC_COUNTER register

PRT Statistic Counter Register

Bit	Field	Mode	Initial Value	Description
31:24	RX_UNSUCC	R	0x0	For each reception of a message that is not successful, but ends in an error, the counter is incremented once.
23:16	RX_SUCC	R	0x0	For each valid message received this counter is incremented.
15:8	TX_UNSUCC	R	0x0	For each unsuccessful transmission attempt of a message this counter is incremented once. Unsuccessful transmission attempt means that the transmission was started on the CAN bus but ends with an error. ARBLOST and HFI do not trigger the increment for the TX_UNSUCC counter. ARBLOST is not considered as an error, HFI prevents the start of the transmission on the CAN bus.
7:0	TX_SUCC	R	0x0	For each valid TX message received this counter is incremented.

1.6.5.1.3 PRT_EVENT Address Block

Table 86. EVNT register

Event Status Flags Register
The EVNT Register contains event status flags. The flags is set by the PRT when specific events occur. A software reset clears all flags. A host write access to this register, writing a 1 to a specific flag, clears that flag. When a host write access occurs concurrently with a set condition for a flag, the flag is set.

Bit	Field	Mode	Initial Value	Description
15	TX_PARITY_ERR_TS	R/W1C	0x0	Parity Error in TS of Tx MSG
14	RX_PARITY_ERR_TS	R/W1C	0x0	Parity Error in TS of Rx MSG
13	ABO	R/W1C	0x0	TX_MSG sequence stopped by TX_MSG_WUSER code ABORT
12	IFR	R/W1C	0x0	Invalid Frame Format requested in TX_MSG
11	USO	R/W1C	0x0	Unexpected Start of Sequence during TX_MSG sequence detected
10	DU	R/W1C	0x0	Underrun condition in TX_MSG sequence detected
9	PXE	R/W1C	0x0	Protocol Exception Event occurred
8	TXF	R/W1C	0x0	Frame transmitted
7	RXF	R/W1C	0x0	Frame received
6	DO	R/W1C	0x0	Overflow condition in RX_MSG sequence detected
5	STE	R/W1C	0x0	Stuff Error
4	FRE	R/W1C	0x0	Form Error or the condition of CAN error counting rule f)
3	AKE	R/W1C	0x0	Acknowledge Error
2	B1E	R/W1C	0x0	Bit1 Error: During the transmission of a message (with the exception of the arbitration field), the PRT wanted to send a recessive bit (logical value 1), but the monitored CAN bus value was dominant.
1	B0E	R/W1C	0x0	Bit0 Error: The PRT wanted to send a dominant bit (logical value 0), but the monitored CAN bus value was recessive. During Bus-Off recovery, B0E is also set each time a sequence of 11 recessive bits has been monitored, enabling the CPU to readily check whether the CAN bus is stuck at dominant or continuously disturbed, and to monitor the proceeding of the Bus-Off recovery sequence.
0	CRE	R/W1C	0x0	CRC Error

1.6.5.1.4 PRT_CONTROL Address Block

Table 87. LOCK register

Unlock Sequence Register
Writing a sequence of specific data words enables the activation of control commands in the registers CTRL and FIMC.

Bit	Field	Mode	Initial Value	Description
31:16	TMK	W	0x0	Test Mode Key
15:0	ULK	W	0x0	Unlock Key

Table 88. CTRL register

Control Register
Writing to this register controls the CAN protocol operation.
When writing to this register, only one of the four bits TEST, SRES , STRT, or STOP may be written to 1. Multiple CTRL register bits can be set simultaneously; however, such write access has no functional impact on the RTL, and the PRT state remains unchanged. The bit IMMD may be written to 1 together with the bit STOP, but not together with one of the other bits.

Bit	Field	Mode	Initial Value	Description
12	TEST	W	0x0	Enable Test Mode. The Test Mode Key must be used prior to write to this bit field.
8	SRES	W	0x0	Software Reset. When the CAN protocol operation is stopped, the software reset of all state machines of the PRT (excluding the error-counters and the error-states) is triggered by writing 1 to CTRL.SRES. No unlocking sequence is required. A software reset will not be executed while the CAN protocol operation is started.
4	STRT	W	0x0	Start CAN protocol operation.
1	IMMD	W	0x0	Stop CAN protocol operation immediately. The Unlock Key must be used prior to write to this bit . This bit is only effective when being set together with the bit STOP.
0	STOP	W	0x0	Stop CAN protocol operation. The Unlock Key must be used prior to write to this bit field. When not set together with bit IMMD the PRT waits for an ongoing CAN message to finish before stopping CAN protocol operation.

Table 89. FIMC register

Fault Injection Module Control Register
Writing the fault injection position number requires the application of the test mode key sequence before writing to FIMC.

Bit	Field	Mode	Initial Value	Description
14:0	FIP	R/W	0x0	Fault Injection Position. Writing to FIMC while MODE.FIME is set and the PRT is started activates the Fault Injection Module FIM (see Safety Measures chapter). While the FIM is activated, the value of FIMC.FIP is protected from further write accesses until the FIM is de-activated again.

Table 90. TEST register (page 1 of 2)

Hardware Test Functions Register
This register is writable after the hardware test mode functions are enabled by writing the test mode key sequence to LOCK and CTRL registers. While the hardware test mode functions are not enabled, this register is read-only. The hardware test mode functions are disabled and cleared by the software reset of the PRT.

Bit	Field	Mode	Initial Value	Description
28	TS_PARITY_ERR	W	0x0	Writing a 1 to the bit field triggers the related interrupt line, this bit is auto-cleared
27	BUS_OFF	W	0x0	Writing a 1 to the bit field triggers the related interrupt line, this bit is auto-cleared
26	BUS_ON	W	0x0	Writing a 1 to the bit field triggers the related interrupt line, this bit is auto-cleared
25	E_PASSIVE	W	0x0	Writing a 1 to the bit field triggers the related interrupt line, this bit is auto-cleared
24	E_ACTIVE	W	0x0	Writing a 1 to the bit field triggers the related interrupt line, this bit is auto-cleared
23	BUS_ERR	W	0x0	Writing a 1 to the bit field triggers the related interrupt line, this bit is auto-cleared
22	RX_EVT	W	0x0	Writing a 1 to the bit field triggers the related interrupt line, this bit is auto-cleared
21	TX_EVT	W	0x0	Writing a 1 to the bit field triggers the related interrupt line, this bit is auto-cleared
20	IFF_RQ	W	0x0	Writing a 1 to the bit field triggers the related interrupt line, this bit is auto-cleared

Table 90. TEST register (page 2 of 2)

Hardware Test Functions Register
This register is writable after the hardware test mode functions are enabled by writing the test mode key sequence to LOCK and CTRL registers. While the hardware test mode functions are not enabled, this register is read-only. The hardware test mode functions are disabled and cleared by the software reset of the PRT.

Bit	Field	Mode	Initial Value	Description
19	RX_DO	W	0x0	Writing a 1 to the bit field triggers the related interrupt line, this bit is auto-cleared. It is recommended to test this IR while PRT is stopped. If PRT is started at this point in time, then additionally to the tested IR (IRC.SAFETY_RAW.PRT_RX_DO) also the IR IRC.SAFETY_RAW.MH_RX_ABORT is set.
18	TX_DU	W	0x0	Writing a 1 to the bit field triggers the related interrupt line, this bit is auto-cleared. It is recommended to test this IR while PRT is stopped. If PRT is started at this point in time, then additionally the following things will happen: a) IRC.SAFETY_RAW.MH_SAFETY_MISC is set, along with MH.STS1.TX_DP_SEQ_ERR as IR reason. b) PRT is stopped. c) IRC.ERR_STS_RAW.PRT_STOP is set HINT: The additional behavior occurs, because IRC.SAFETY_RAW.PRT_TX_DU leads in the MH to the detection of an unexpected sequence at MH_PRT TX_MSG interface.
17	USOS	W	0x0	Writing a 1 to the bit field triggers the related interrupt line, this bit is auto-cleared
16	ABORTED	W	0x0	Writing a 1 to the bit field triggers the related interrupt line, this bit is auto-cleared
15	HWT	R	0x0	This status flag HWT shows whether the hardware test mode functions are enabled, set to 1 means enable.
5:4	TXC	R/W	0x0	Control the bit value driven at CAN_TX 0b00: Normal function of CAN TX 0b01: Normal function of CAN TX. CAN RX is ignored (for message look back mode) 0b10: CAN TX output set to 0 0b11: CAN TX output set to 1
3	RXD	R	0x1	Bit value seen at CAN_RX. The CAN_RX input (output signal of the transceiver) is always readable through this bit.
0	LBCK	R/W	0x0	Enable the Message Loop-Back mode, see chapter Trace and Debug.

1.6.5.1.5 PRT_CONFIGURATION Address Block

Table 91. MODE register (page 1 of 2)

Operating Mode Register
Configuration register that is writable while the CAN communication is stopped and that is read-only after the CAN communication is started. This register defines separate operating mode options. The four configuration bits FDOE, XLOE, EFDI, and XLTR, are interrelated according to table Frame Formats defined in Operating Mode chapter.

Bit	Field	Mode	Initial Value	Description
12	LCHB	R/W	0x0	Light Commander mode for Higher Bit rates {DMON, NACK, EOFF}
11	FIME	R/W	0x0	Fault Injection Module Enable, see Safety Measures chapter
10	EFDI	R/W	0x0	Error Flag Disable, 1 means Error Signaling is disabled as defined in [2] and the error counters REC and TEC are not incremented.. When this bit is set, only CAN XL frames are transmitted and received dominant FDF or XLF bits are treated as form errors.
9	XLTR	R/W	0x0	XL Transceiver Connected
8	SFS	R/W	0x0	Time stamp position: Start of Frame Stamping 1: Timestamps captured at the start of a frame 0: Timestamps captured at the end of a frame
7	RSTR	R/W	0x0	Restricted Mode Enabled as defined in [1]

Table 91. MODE register (page 2 of 2)

Operating Mode Register
Configuration register that is writable while the CAN communication is stopped and that is read-only after the CAN communication is started. This register defines separate operating mode options. The four configuration bits FDOE, XLOE, EFDI, and XLTR, are interrelated according to table Frame Formats defined in Operating Mode chapter.

Bit	Field	Mode	Initial Value	Description
6	MON	R/W	0x0	Monitoring Mode Enabled as defined in [1]
5	TXP	R/W	0x0	Transmit Pause. If this bit is set, the PRT pauses for two CAN bit times before starting the next transmission after itself has successfully transmitted a frame
4	EFBI	R/W	0x0	Edge Filtering during Bus Integration. If this bit is set, the PRT requires two consecutive dominant tq to detect an edge causing the reset of the bit counter for the detection of the idle condition.
3	PXHD	R/W	0x0	Protocol Exception Handling Disabled
2	TDCE	R/W	0x0	Transmitter Delay Compensation Enabled as defined in [1]
1	XLOE	R/W	0x0	XL Frame Format enabled. When set to 0, node behaves according to ISO11898-1:2024, no arbitration during FDF bit. When set to 1, node behaves according to ISO11898-1:2024-1, arbitration during FDF bit and XLF bit. This bit cannot be set to 1 when one of the static inputs ONLY_CC or ONLY_CC_FD is set. Setting XLOE without setting FDOE is an invalid configuration.
0	FDOE	R/W	0x0	FD Frame Format enabled. When set to 1, node is FD enabled according to ISO11898-1:2024. When set to 0, node is FD tolerant according to ISO11898-1:2024 (only CAN Classic frames used). This bit cannot be set to 1 when the static input ONLY_CC is set.

Table 92. NBTP register

Arbitration Phase Nominal Bit Timing Register
Configuration register that is writable while the CAN communication is stopped and that is read-only after the CAN communication is started. This register defines the Nominal Bit Timing as defined in [1].

Bit	Field	Mode	Initial Value	Description
29:25	BRP	R/W	0x0	Bit Rate Prescaler. Valid values for the Bit Rate Prescaler BRP are 0x00-0x1F. This value defines the length of the Time Quantum TQ for all three bit time configurations. The actual interpretation of this value is that the TQ is (BRP + 1) CLK periods long
24:16	NTSEG1	R/W	0x0	Nominal Prop_Seg and Phase_Seg1. Valid values for NTSEG1 are 0x01-0x1FF. This value defines the sum of Prop_Seg(N) and Phase_Seg1(N). The actual interpretation of this value is that these segments together are (NTSEG1 + 1) TQ long.
14:8	NTSEG2	R/W	0x0	Nominal Phase_Seg2. Valid values for NTSEG2 are 0x01-0x7F. This value defines the length of Phase_Seg2(N). The actual interpretation of this value is that the phase buffer segment 2 is (NTSEG2 + 1) TQ long.
6:0	NSJW	R/W	0x0	Nominal SJW. Valid values for the Nominal Synchronization Jump Width NSJW are 0x00-0x7F. The actual interpretation of this value is that the Nominal Synchronization Jump Width is (NSJW + 1) TQ long.

Table 93. DBTP register

CAN FD Data Phase Bit Timing Register
Configuration register that is writable while the CAN communication is stopped and that is read-only after the CAN communication is started. This register defines the FD Data Phase Bit Timing as defined in [1].

Bit	Field	Mode	Initial Value	Description
31:24	DTDCO	R/W	0x0	Transmitter Delay Compensation Offset for FD frames. Valid values for the FD Transmitter Delay Compensation Offset DTDCO is 0x00-0xFF. This configuration defines the distance between the measured delay from CAN_TX to CAN_RX and the secondary sample point SSP, measured in CLK periods. This value is used when transmitting a CAN FD frame
23:16	DTSEG1	R/W	0x0	FD data phase Prop_Seg and Phase_Seg1. Valid values for DTSEG1 are 0x00-0xFF. This value defines the sum of Prop_Seg(D) and Phase_Seg1(D). The actual interpretation of this value is that these segments together are (DTSEG1 + 1) TQ long
14:8	DTSEG2	R/W	0x0	FD data phase Phase_Seg2. Valid values for DTSEG2 are 0x01-0x7F. This value defines the length of Phase_Seg2(D). The actual interpretation of this value is that the phase buffer segment 2 is (DTSEG2 + 1) TQ long.
6:0	DSJW	R/W	0x0	FD data phase SJW. Valid values for the FD data phase Synchronization Jump Width DSJW are 0x00-0x7F. The actual interpretation of this value is that the FD data phase Synchronization Jump Width is (DSJW + 1) TQ long.

Table 94. XBTP register

CAN XL Data Phase Bit Timing Register
Configuration register that is writable while the CAN communication is stopped and is read-only after the CAN communication is started. This register defines the XL Data Phase Bit Timing as defined in [2].

Bit	Field	Mode	Initial Value	Description
31:24	XTDCO	R/W	0x0	Transmitter Delay Compensation Offset for XL frames. Valid values for the XL Transmitter Delay Compensation Offset XTDCO is 0x00-0xFF. This configuration defines the distance between the measured delay from CAN_TX to CAN_RX and the secondary sample point SSP, measured in CLK periods. This value is used when transmitting a CAN XL frame.
23:16	XTSEG1	R/W	0x0	XL data phase Prop_Seg and Phase_Seg1. Valid values for XTSEG1 are 0x00-0xFF. This value defines the sum of Prop_Seg(X) and Phase_Seg1(X). The actual interpretation of this value is that these segments together are (XTSEG1 + 1) TQ long
14:8	XTSEG2	R/W	0x0	XL data phase Phase_Seg2. Valid values for XTSEG2 are 0x01-0x7F. This value defines the length of Phase_Seg2(X). The actual interpretation of this value is that the phase buffer segment 2 is (XTSEG2 + 1) TQ long
6:0	XSJW	R/W	0x0	XL data phase SJW. Valid values for the XL data phase Synchronization Jump Width XSJW are 0x00-0x7F. The actual interpretation of this value is that the XL data phase Synchronization Jump Width is (XSJW + 1) TQ long

Table 95. PCFG register

PWME Configuration Register
Configuration register that is writable while the CAN communication is stopped and is read-only after the CAN communication is started. This register defines the parameters needed for the PWM coding (as described in [3]) in the PWME module for CAN XL transceivers with switchable operating modes

Bit	Field	Mode	Initial Value	Description
21:16	PWMO	R/W	0x0	The actual interpretation of this value is that the PWM offset length is PWMO clock cycles long.
13:8	PWML	R/W	0x0	The actual interpretation of this value is that the PWM long phase length is (PWML + 1) clock cycles long.
5:0	PWMS	R/W	0x0	The actual interpretation of this value is that the PWM short phase length is (PWMS + 1) clock cycles long.

1.6.6 Functional Description

The PRT does not provide an internal time base for time stamping; for that purpose, an external time base needs to be connected (see Hardware Timestamping chapter).

The PRT does provide several interrupt outputs that signal, with a high-pulse of one CLK period length, the occurrence of specific internal events. For test purposes, the TEST register has a Generate Interrupt Pulse function GIP so that in hardware test mode HWT an interrupt output pulse can also be triggered by writing a 1 to the corresponding TEST register bit.

Table 96. PRT Interrupts

Interrupt	TEST bit	Activated when
TS_PARITY_ERR	28	TimeStamp Parity mismatch
BUS_OFF	27	Stop of CAN module, either after CTRL.STOP command written or stop due to entering Bus_Off state (STAT.BO=1)
BUS_ON	26	Signals starting of CAN module, is set immediately after CTRL.STRT command is written (this is also the case if CAN module was in Bus_Off state (STAT.BO=1))
E_PASSIVE	25	Switching from Error-Active to Error-Passive
E_ACTIVE	24	Switching from Error-Passive to Error-Active
BUS_ERR	23	Error detected on CAN bus or Protocol Exception Event detected
RX_EVT	22	Received a valid message
TX_EVT	21	Successfully transmitted a message
IFF_RQ	20	MH Requests a Message with Invalid Frame Format in Header
RX_DO	19	Data Overflow condition in RX_MSG sequence detected
TX_DU	18	Data Underrun condition in TX_MSG sequence detected
USOS	17	Unexpected Start of Sequence during TX_MSG sequence detected
ABORTED	16	TX_MSG sequence stopped by TX_MSG_WUSER code ABORT

The PRT outputs internal status information, optionally to be connected to a hardware debug port
SAMPLE_POINT:This is the CAN Sample Point
STAT_ACT: This is the actual 2-bit-value of register STAT.ACT (see register description)

1.6.6.1 CAN FD light Commander for higher bit rates

CAN FD Light is a concept of price-sensitive sensor and actuator networks that can be used under automotive conditions. The commander node is a CAN FD node compliant to ISO 11898-1:2024, see [1]. A CAN FD light responder node is an implementation based on a subset of the CAN FD data link layer functionality as specified in ISO 11898-1:2024 Annex A. CAN FD Light communication uses a commander/responder - scheme. For more information, please refer to [10] CiA 604-3.
Since CAN FD Light uses the same data rate for the arbitration and data phase the maximum CAN FD Light bit rate is determined by the maximum bit rate achievable in the arbitration phase and by the response time of the ACK-bit. Additionally, the used physical layer transceiver limits the data rate.

The bus monitoring is the main bit rate limiting factor. The commander node monitors if the value on the bus is the same as the one it transmits. Bus monitoring is not possible anymore if the bit time is too short. CAN FD Light responders do not monitor the bus while they are transmitting.

Another bit rate limiting factor is that the commander node requires after frame transmission the correct reception of an ACK-bit sent by at least one node in the network, which may not be received correctly if the ACK-slot is too short. CAN FD Light nodes send the ACK-bit, but they do not require it after they have transmitted their frame. Sending the ACK-bit can be optionally switched off depending on the implementation.

CAN FD light requires neither bus monitoring since arbitration and error frames are not used nor the ACK response because of the commander-responder operation. Therefore, higher data rates in a CAN FD Light network can be achieved if the commander does not use bus monitoring and does not require the confirmation of the ACK-bit after having transmitted a frame.

Since CAN FD Light does not arbitrate and does not need error frames the wired-AND functionality of the CAN physical layer is not needed, which allows the use of active push-pull transceivers (e.g. like being used during the data phase of CAN XL).

In conclusion the following features of the commander node may be disabled to achieve higher data rates:
• turn off bus monitoring
• disable the need of the ACK response by the commander
• disable sending error and overload frames
• disable synchronization in transmitter state

CAN FD light commander mode in PRT can be enabled by setting the bit LCHB in PRT MODE register.

1.6.6.2 Statistic Counters

The PRT provides four readable statistic counters (8 bits each) to count the respective events:

STATISTIC_COUNTER.TX_SUCC [8]: TX_SUCC counter increments at every successful transmission on the bus.
STATISTIC_COUNTER.TX_UNSUCC [8]: For each unsuccessful transmission attempt of a message this counter is incremented once. Unsuccessful transmission attempt means that the transmission has started on the CAN bus, but ends with an error. Losing arbitration is not regarded as an error and does not cause an increment of this counter. In case of HFI, the PRT does not start the transmission attempt on the CAN bus and therefore it does not cause an increment of this counter. Following errors are taken for incrementing this counter.

ACK error: Absence of dominant bit during the ACK slot.

Bit Error: Bit value that is monitored differs from the bit value sent.

STATISTIC_COUNTER.RX_SUCC [8]: RX_SUCC counter increments at every successful reception of message from the bus.
STATISTIC_COUNTER.RX_UNSUCC [8]: For each reception of a message that is not successful, but ends in an error, the counter is increments once. This counter is triggered by following errors:

Stuff Error: sixth consecutive equal bit level in a frame field that is coded by the method of dynamic bit stuffing. In CAN FD and in CAN XL frames the receivers also count the number of dynamic stuff bits in a frame and check it with the stuff bit-count number value transmitted in that frame.

CRC error: CRC calculated from the receiver based on the frame field values differs from CRC sent by the transmitter. There are two types of CRC errors: PCRC error and FCRC error as there are two independent CRCs used in CAN XL MAC sub-layer frame.

Form error: fixed form bit field contains one or more illegal bits, or when a fixed stuff bit in an XL Frame is not at its expected value.
Note that the fill level values in **MH.RXFQ_STATUS** register and the values in **STATISTIC_COUNTER.RX_SUCC** may not align. It could happen that the STATISTIC_COUNTER.RX_SUCC has incremented but the fill level of RXFQx has not incremented. STATISTIC_COUNTER.RX_SUCC indicates that a message is successfully received by the PRT. It does not indicate that the same message is successfully stored into the queue. The message can get dropped in MH due to several reasons like fifo being full, filtering process abort, filter no match etc.

1.6.6.3 Transceiver Sharing Switch (TSS) Support

The purpose of a TSS is to allow several CAN modules sharing one CAN transceiver, saving pins to connect to CAN transceivers themselves. The TSS is not part of any CAN module, it is a separate entity. Several CAN modules together with several TSSs may be integrated into one IC.
PRT supports the following signals for interaction with TSS:

1) TSS_EN (Status bit) Transceiver Sharing Switch Enable
STAT1.TSS_EN shows the status of the input port TSS_EN. It indicates that the suppression of the ACK bit transmission and PWM encoding is enabled. Suppression occurs only in Receiver state when requested by the external transceiver sharing logic via the input signal GROUP_TR.

2) IS_TRANSMITTER (output)

XS_CAN sets IS_TRANSMITTER=1 when it is in Transmitter state. See STAT0.ACT status.

3) GROUP_TR (input)
GROUP_TR=1 informs the XS_CAN that XS_CAN and the local transmitter share one transceiver.

4) CAN_TX (output)
When XS_CAN is in Receiver state AND GROUP_TR=1 AND TSS_EN=1 then the XS_CAN suppresses the ACK bit transmission and PWM encoding.
Local ACK bit suppression is required to disallow local ACK while local transmitters are sending. This allows to detect if external nodes are connected.

5) CAN_RX (input)
No impact

1.6.6.4 Fingerprinting Support

The XS_CAN IP provides a 1 bit output signal *SP_BEFORE_DOM_REC_EDGE* which stays high for one CAN_CLK period. It is generated for both transmission and reception of CAN messages during the arbitration phase. In each message format, the pulse is triggered at definite bits in the frame as given below.

CAN CC Base Frame (CBFF) - (remote or data message): r0 bit in control field
CAN CC Extended Frame (CEFF) - (remote or data message): r0 bit in control field
CAN FD Base Frame Format (FBFF): res bit in control field
CAN FD Extended Frame Format (FEFF): res bit in control field
CAN XL Frame Format (XLFF): resXL bit in control field

1.6.6.5 PRT static configuration

The two inputs *ONLY_CC* and *ONLY_CC_FD* are intended to be connected to static signals (either hard-wired or OTP), thereby permanently restricting the PRT's function to older versions of the CAN protocol, see also [1].

The function of the PRT is restricted to the CAN CC frame format (CAN FD tolerant implementation of ISO 11898 1:2024) when *ONLY_CC* = 1 or when the configuration bit **MODE.FDOE** is not set by the host.

The function of the PRT is restricted to the frame formats CAN CC and CAN FD (full implementation of ISO 11898 1:2024) when *ONLY_CC_FD* = 1 or when the configuration bit **MODE.XLOE** is not set by the host.

With the exception of **MODE.XLOE** and **MODE.FDOE**, *ONLY_CC* and *ONLY_CC_FD* have no impact on the behavior of the other configuration registers.
They can change **MODE.XLOE** and **MODE.FDOE** to read-only.
- *ONLY_CC* = 1 means **MODE.FDOE**=0 and is not writable by Software
MODE.XLOE=0 and is not writable by Software
- *ONLY_CC_FD* = 1 means **MODE.XLOE** and is not writable by Software

1.6.6.6 Host Access to Protocol Controller

The host is connected to the REG_APB interface of the PRT. The host is the APB master, the PRT is the APB slave. The host is able to control the PRT's operating mode, to write its configuration data and to read the PRT's status. REG_APB is implemented as an AMBA APB 4 slave interface (compliant to AMBA 4 ARM Ltd protocol, see [6]). The host must write all configuration registers to valid values before starting the PRT's CAN operation.

The registers FIMC and TEST are writable after the hardware test mode functions are enabled by writing the test mode key sequence to LOCK and CTRL registers. If the hardware test mode functions are not enabled, this register is read-only. Write Access to FIMC and TEST registers without performing the test mode key, results in OK response, but the data is not written. Note that there is no interrupt generated to indicate this.

• Any access to registers, either read or write, must use a 32bit aligned address, otherwise the transaction is ignored and no IR/flag is set.
• When an access is performed to a non-mapped register in the address range, IP gives ERROR response to the host along with the interrupt HOST_ARA_IRQ IP if enabled in IRC. Write transactions are ignored and read accesses are given the default data 0xCAFECAFE.

• When a read access to write-only registers or a write access to read-only registers is performed, IP gives ERROR response to the host along with the interrupt HOST_ARA_IRQ IP if enabled in IRC. Write transactions are ignored and read accesses are given the default data 0xCAFECAFE.

1.6.6.7 Software Reset

The software reset is triggered by writing 1 to **CTRL.SRES** when the CAN protocol operation is stopped. This does not require an unlocking sequence. The software reset must not be executed when **CTRL.SRES** is written while the CAN protocol operation is started. The software reset resets all state machines of the PRT (excluding the error-counters and the error-states) and clears the following readable registers: **STAT.TDCV**, **STAT.FIMA**, **FIMC.FIP**, **TEST.HWT**, **TEST.TXC**, **TEST.LBCK**, and all flags of EVNT. The configuration registers are not changed by a software reset.

1.6.6.8 Operating Mode

The operating mode is defined using the MODE register and only when the CAN communication is stopped, otherwise the register is read only. The register MODE defines separate operating mode options.

The four configuration bits FDOE, XLOE, EFDI, and XLTR, and are interrelated according to the table below.

Table: Frame Formats

Table 97. Frame Formats (page 1 of 2)

FDOE	XLOE	XLTR	EFDI	Description
0	0	X	0	Operating only in Classical CAN frame format. When FDOE is not set, the PRT shall be restricted to the Classical CAN frame format. In this case, when PXHD is set, the PRT shall accept both recessive and dominant bits as received reserved bits. When PXHD is not set, the PRT shall treat a recessive first reserved bit as a Protocol Exception condition and shall enter the Protocol Exception State as defined in [1] and it shall set the flag EVNT.PXE. FDOE shall be static at 0 while the input signal ONLY_CC is 1.
0	1	X	X	Invalid configuration
0	X	X	1	Invalid configuration
X	0	X	1	Invalid configuration
1	0	X	0	Operating in Classical CAN and CAN FD frame fromat. When FDOE is set, the PRT shall be able to transmit and to receive Classical CAN frames and CAN FD frames as defined in [1]. When XLOE is not set, the PRT shall not be able to transmit or to receive CAN XL frames as efined in [2]. When FDOE is set but not XLOE,PXHD defines the PRT's reaction on a recessive reserved but following the recessive FDF bit in a CAN FD frame. In this case, when PXHD is set, the PRT shall treat this condition as a Form Error. When PXHD is not set, the PRT shall treat this condition as a Protocol Exception condition and shall enter the Protocol Exception State as defined in [1] and it shall set the flag EVNT.PXE, XLOE shall be static at 0 while the input signals ONLY_CC_FD or ONLY_CC are at 1.
1	1	0	0	Operating in all frame formats, without XL transceiver. When FDOE and XLOE are both set and EFDI is not set, the PRT shall be able to transmit and to receive Classical CAN frames and CAN FD frames as defined in [1] and it shall be able to transmit and to receive CAN XL frames as defined in [2]. When both FDOE and XLOE are set, PXHD defines the PRT's reaction on a recessive reserved bit following the recessive XLF bit in a CAN XL frame. In this case, when PXHD is set, the PRT shall treat this condition as a Form Error. When PXHD is not set, the PRT shall treat this condition as a Protocol Exception State as defined in [2] and it shall set the flag EVNT.PXE. When EFDI is not set, the PRT shall send error lags as defined in [1].

Table 97. Frame Formats (page 2 of 2)

FDOE	XLOE	XLTR	EFDI	Description
1	1	0	1	Operating in XL frame format only, without XL transceiver, error frames are . disabled for all communication. It shall be an invalid configuration to set EFDI without setting both FDOE and XLOE. When XLTR is not set, the PRT shall not control the operating mode of the transceiver.
1	1	1	0	Invalid configuration
1	1	1	1	Operating in XL frame format only, enabling XL transceiver, error frames are disabled for all communication. When XLTR is set together with FDOE and XLOE, the PRT shall control the PWME to the transceiver In order to switch the operating mode of the transceiver at the beginning and at the end of the CAN XL data phase, as defined in [2]. When EFDI is set together with FDOE and XLoe, the PRT shall not send error flags and it shall not change its trasmit error counter or its receive error counter. When an error condition occurs, the PRT shall, instead of sending an error flag, enter Protocol Exception State, like in Restricted Mode, as defined in [2].

1.6.6.9 Starting and Stopping the Module

When the CAN protocol operation is stopped, the ENABLE output is at 0 and the **TX_MSG** and **RX_MSG** interfaces are not active, but the REG_APB interface remains fully functional.

The PRT is started by writing 1 to **CTRL_START**. This does not require an unlocking sequence. After the start command, the PRT waits for the occurrence of a sequence of 11 consecutive recessive bits (the idle condition of ISO 11898-1:1995) to finish its integration into the CAN communication on the CAN bus line. When the PRT has detected the idle condition and has no pending transmission request, it switches into idle state. When the PRT has detected the idle condition and has a pending transmission request, the PRT starts the transmission in the following bit. When the PRT sees a dominant bit on entering idle state, it immediately becomes receiver of that frame.

There are two options to stop the CAN protocol operation under software control, one that waits for the completion of an ongoing message transfer and one that stops the operation immediately.

The two options use different variants of the **CTRL_STOP** command. In the first variant, only the STOP bit is written to 1. In the second variant, both the **CTRL_STOP** bit and the **CTRL_IMMD** bit are written to 1 at the same time. Both variants require the application of the unlock key sequence, followed by a write access to the CTRL register. The three consecutive write accesses may not be interrupted by other accesses via **REG_APB**.

The first variant of the **STOP** command is asserted this way:
Step Write data to Register at Address
1: Write 0x1234 to **LOCK.ULK** 0x40
2: Write 0x4321to **LOCK.ULK** 0x40
3: Write 1to **CTRL.STOP** 0x44

The unlock is only for 1 write access. After 1 write access, it will automatically lock again.First, write the 1st unlock key word. Then write the 2nd unlock key word, immediately followed by the write to the control register.Writing the 1st unlock key word sets a 1st flipflop.After the 1st flipflop has been set, writing the 2nd unlock key word sets a 2nd flipflop. When both flip flops are set, the control register may be modified.Please note that any register access after the 2 key flip flops have been set, clears the 2 key flipflops.When the 1st key flipflop has been set, and that is followed by any access that is not writing the 2nd unlock key, the 1st key flipflop is cleared again.

The PRT's reaction to the first variant of the STOP command depends on its current activity (**STAT.ACT**). When the current activity of this node is Idle, it switches its activity to its inactive state immediately and stop all CAN operation. When the current activity is either Receiver or Transmitter, it continues that activity, and it sets the status flag **STAT.STP** to show that it is waiting for end of the actual message after a **CTRL.STOP** command. If no TX_MSG sequence is already started, the PRT clears TX_MSG_WREADY and keep it cleared until all CAN operation is stopped. As soon as the current reception or transmission is finished (either successfully or in failure) the PRT reports the result of that transfer to the MH, clears the status flag **STAT.STP**, clears ENABLE, switches its activity to its inactive state, and stops all CAN operation. The PRT does not start another reception or transmission until it is started again.

The second variant of the **STOP** command is asserted this way:
StepWrite data to Register at Address
1:Write 0x1234to **LOCK.ULK** 0x40
2:Write 0x4321to **LOCK.ULK** 0x40
3:Write 1to **CTRL.IMMD** and 1 to **CTRL.STOP** 0x44
4:Write 1to **CTRL.SRES** 0x44
The PRT's reaction to the second variant of the STOP command is to switch its activity to its inactive state immediately, to stop all CAN operation, and to set its **CAN_TX** output to 1 and to clear **ENABLE**. When the current activity was Transmitter, the PRT aborts that transmission. When the current activity was Receiver, it aborts that reception. Both interfaces with the MH are reset, outstanding transactions are discontinued. If this happens while an RX_MSG sequence was ongoing, this sequence is discontinued with the ABORT code. The PRT does not start another reception or transmission until it is started again.

When the PRT is stopped, it sets the **ENABLE** output to 0.
The CAN protocol operation stops automatically on the following conditions:
When the node enters the CAN protocol's Bus-Off state, unexpected Start of Sequence transaction during TX_MSG sequence detected.

Note: Detecting an Unexpected Start of Sequence (USOS) indicates a mis-synchronization between PRT and MH that requires a restart.

When the CAN protocol operation is stopped, it is started by writing 1 to **CTRL.STRT**. This does not require an unlocking sequence.

When the CAN protocol operation was stopped because the PRT entered the CAN Bus_Off state, the start command (writing 1 to **CTRL.STRT**) causes the PRT to perform the CAN Bus_Off Recovery Sequence before it is again able to participate in CAN communication.

The CAN Bus_Off Recovery Sequence (see ISO 11898-1:2024) cannot be shortened by starting or stopping the PRT. If the PRT goes Bus_Off, it will set **STAT.BO** bit and it will, of its own accord, stop all bus activities. Once the PRT has been started again, the PRT clears **STAT.TEC**, **STAT.RP**, **STAT.REC** and **STAT.EP** but it will keep **STAT.BO**.
The PRT will then wait for 129 occurrences of Bus Idle (129 * 11 consecutive recessive bits) before resuming normal operation. The PRT uses the Receive Error Counter (**STAT.REC**) to count the occurrences of Bus Idle. Additionally, each time a sequence of 11 recessive bits has been monitored, a Bit0 Error (**EVNT.B0E**) is reported, enabling the host to readily check-up whether the CAN bus is stuck at dominant or continuously disturbed and to monitor the progress of the Bus_Off recovery sequence. When the last-but-one sequence of 11 recessive bits has been monitored, **STAT.RP**, and **STAT.EP** are set and **STAT.REC** is at 0x7F. When the last sequence of 11 recessive bits has been monitored, the end of the Bus_Off recovery sequence is reached and **STAT.TEC**, **STAT.RP**, **STAT.REC**, **STAT.EP** and **STAT.BO** will all be reset. The PRT switches into idle state.

1.6.6.10 Reaction on Exceptions at the TX_MSG and RX_MSG Interfaces

1.6.6.10.1 MH Requests a Message with Invalid Frame Format in Header

This is detected when the requested transmit frame format is disabled in the configuration register (see **MODE.FDOE**, **MODE.EFDI**, or **MODE.XLOE**) or there is an internal contradiction in the header content of that frame. On the detection of this condition, the PRT ends the **TX_MSG** sequence with the response code HFI, generate a pulse on the **IFF_RQ** interrupt output and it does not transmit that message.

1.6.6.10.2 MH Intentionally Aborts TX_MSG Sequence

If the ABORT command is given with the second transaction of the **TX_MSG** sequence, the PRT does not start the transmission. If the ABORT command is given after the second transaction of the **TX_MSG** sequence, the PRT sets an internal flag that causes the FCRC bits of that transmission to be transmitted inverted. This internal flag is cleared at the end of the transmission. In both cases, the PRT generates a pulse on the **ABORTED** interrupt output and the ongoing **TX_MSG** sequence is finished.

1.6.6.10.3 Data Underrun Condition in TX_MSG Sequence Detected

This is detected when the ongoing transmission on the CAN bus needed another **TX_MSG** data word but that word was not provided in time. On the detection of this condition, the PRT continues the transmission (to avoid disturbing the message schedule on the CAN bus), but the PRT generates a pulse on the **TX_DU** interrupt output and the PRT sets an

internal flag that causes the FCRC bits of that transmission to be transmitted inverted (to avoid the acceptance of a message containing invalid data). This internal flag is cleared at the end of the transmission. In case of a Data Underrun, the ongoing **TX_MSG** sequence finishes with the code DU when the MH transfers the missing data word. After the end of the transmission, TX_MSG_WREADY is asserted again to accept the following TX_MSG Sequence (starting again with W0).

1.6.6.10.4 Unexpected Start of Sequence Detected

This is detected when the PRT receives a **TX_MSG** transaction marked as Start of Sequence before a previously started **TX_MSG** sequence has ended. This indicates that MH and PRT operate out-of-phase. On the detection of this condition, the PRT generates a pulse on the **USOS** interrupt output and the PRT stops CAN protocol operation. Afterwards, the PRT needs to be restarted under software control by writing 1 to **CTRL.STRT**.

1.6.6.10.5 Data Overflow Condition in RX_MSG Sequence Detected

This is detected when, during an ongoing reception, the MH has not acknowledged an **RX_MSG** data word in time. On the detection of this condition, the PRT generates a pulse on the **RX_DO** interrupt output and the PRT ends such an RX_MSG sequence via a subsequent transfer with code DO. This transfer with code DO must be acknowledged by the MH before the PRT can start a new **RX_MSG** sequence.

1.6.6.11 Controlling the Module's Clock Input

The PRT has two clock inputs, **CLK** and **CLK_APB**. **CLK** is the clock input of the PRT excluding its REG_APB interface, while **CLK_APB** is the clock input of the PRT's REG_APB module. Both clocks are synchronous to each other, driven from the same source. The difference between the two clocks is that **CLK_APB** must be always active (to keep the REG_APB interface operational), but **CLK** may be switched off (gated) while the PRT is stopped, e.g., when no CAN communication is needed.

The recommended clock frequency for CAN XL operation is 160 MHz.
The recommended clock frequency for CAN FD operation only is 80 MHz.
The function of the PRT does not depend on a particular duty-cycle of the clock, i.e., it reacts only on rising clock edges.
The duration of the clock high pulse may vary between 10% and 90% of the clock period during operation.

The PRT's input signal **CLOCK_ACTIVE** shows whether the clock input **CLK** of the PRT is active. The actual value of the **CLOCK_ACTIVE** input signal is always readable from the status bit **STAT_CLKA**, even when CLK is not active. The signal **CLOCK_ACTIVE** does not control the PRT's function, it provides only status information, coming from a clock multiplexer outside of the PRT.

While **CLK** is not active, the PRT has no function and cannot be started. **CLK** must be reactivated before the PRT needs to be started again.

1.6.6.12 Fault Injection Module

The PRT's Fault Injection Module FIM shall be able to invert one specific bit in a transmitted frame in order to check the reaction of a CAN system to a faulty CAN frame. This specific bit shall be selected by FIMC.FIP.For this purpose, all bits in a frame shall be counted, including stuff bits and fixed format bits, starting with zero at the start of frame bit.
The FIM shall be enabled by setting MODE.FIME to 0b1. The FIM shall require a specific activation for each fault injection to be performed. When FIM is enabled, it shall be activated by writing the fault injection position number to FIMC.FIP once PRT is started. It shall not be possible to activate the FIM while the FIM is disabled (MODE.FIME is 0b0).
The status flag STAT.FIMA shall be set to 0b1 when the FIM is activated, it shall be set to 0b0 when the FIM is de-activated. While the FIM is activated, it shall not be possible to change to FIMC.FIP.

Each time a bit in a transmitted frame has been inverted by the FIM and each time the transmission of a message with W1.FIR = 0b1 (See section TX Message Header Definition in section 1.5 Message Handler) ended successfully without bit inversion, the FIM shall de-activate itself and STAT.FIMA shall be cleared.

The FIM shall invert bits only in transmitted frames for which in the second TX_MSG word the bit W1.FIR is set and when the FIM is activated before the second TX_MSG word is transferred to the PRT.

When the FIM is active and the transmission of a message with W1.FIR = 0b1 is requested, the specific bit numbered by FIMC.FIP shall be inverted during the transmission of that message. All bits in the transmitted frame, including format bits, dynamic and fixed stuff bits shall be considered when selecting a bit for inversion. The numbering of bits in a frame shall

start with FIMC.FIP = 0x0000 for the start-of-frame bit. No bit shall be inverted when the transmitted frame has less bits than indicated by FIMC.FIP.

Internally, the PRT shall treat each intentionally inverted bit as if it had been transmitted non-inverted, meaning it does not treat it as a bit error and it does not use the inverted value for CRC calculation or stuff bit generation.

1.6.6.13 Transceiver Interface

The CAN bus is usually implemented as a twisted-pair bus line, its bus wires called CAN_H and CAN_L. An analog CAN transceiver device is connected to the CAN bus, interfacing between the bidirectional CAN bus wires and the CAN protocol controller's unidirectional, digital serial input and output signals. The future CAN XL transceivers will need to switch between two operating modes during the transmission of a CAN XL message. The switching control is coded into signals between protocol controller and transceiver. The coding is implemented inside dedicated PWME module inside PRT ,see figure PRT Block Diagram.

The transceiver's RxD output is connected to the PRT's CAN_RX input. This asynchronous input signal is synchronized to the CAN clock by routing it through two synchronizer-FFs. This delay of two clock cycles is part of the input delay for the calculation of the propagation segment length of the CAN bit time. The CAN bit time configuration is only functional if the following conditions are fulfilled:
a) ((NTSEG1+1) x (BRP x CLK_period)) is larger than the transmitter loop delay
b) ((DTSEG1+1) x (BRP x CLK_period)) and ((XTSEG1+1) x (BRP x CLK_period)) are larger than the transmitter loop delay or transmitter delay compensation is enabled.

The connection from the PRT to the transceiver is through the PWME module (Pulse Width Modulation Encoder, specified in [2]). In CAN XL communication using a transceiver with switchable operating modes, the PWME controls the operating mode of the transceiver, to switch it into the CAN XL data phase modes for transmissions as well as receptions and back. The PWME_CFG configuration data consists of PCFG.PWMO, PCFG.PWML, and PCFG.PWMS, it is an 18-bit vector concatenating the three 6-bit vectors from PCFG.PWMO[5] down to PCFG.PWMS[0]. The appropriate switching times are signaled by the PRT via its outputs XLT (see [2], chapter 7.2.4), D_TX (see [2], chapter 7.2.5) and D_RX (see [2], chapter 7.2.6).

The PWME function is controlled by the following signals in the PRT: *PWME_CFG*, *XLT*, *D_TX*, *D_RX*, and *CAN_TX*.

1.6.6.13.1 PWME Module

The function and the configuration of the PWME module (Pulse Width Modulation Encoder) will be specified in [2].

According to the currently known PWME concept, the function will be as follows:
When transceiver mode switching is disabled (**MODE.XLTR** = 0) and outside a CAN XL frame's data phase, the PWME's *CAN_TX* input is directly connected to the PWME's TxD output.

When transceiver mode switching is enabled (**MODE.XLTR** = 1), the PWME's *CAN_TX* input signal is-coded during a CAN XL frame's data phase, to generate the PWME's TxD output signal.

The transceiver decodes the coded TxD signal, to extract both the signal to be transmitted on the CAN bus and the required transceiver operating mode.

The protocol controller signals the required transceiver operating mode to the PWME module via the three PWME input signals *XLT*, *D_Tx*, and *D_Rx*.

When transceiver mode switching is enabled (MODE.XLTR = 1), XLT is set to 1 from the XLF bit of a CAN XL frame until the end of that frame.

In transmitted CAN XL frames, D_Tx is set to 1 at the beginning of the ADH bit and to 0 at the beginning of the DAH bit or when that frame is disturbed.

In received CAN XL frames, D_Rx is set to 1 at the beginning of the ADH bit and to 0 at the beginning of the DAH bit or when that frame is disturbed.

In transmitted CAN FD frames, D_Tx is set to 1 at the beginning of the data phase and to 0 at the end of the data phase or when that frame is disturbed.

Configuration data needed for the correct PWM coding are provided via the PWME's PWME_CFG input.
When no transceiver mode switching is required, that allows an alternative (empty) implementation of the PWME module where its CAN_TX input is directly connected to its TxD output.

1.6.6.13.2 Connection to the Transceiver

The connection between a CAN module inside a µC and an external CAN transceiver may use dedicated pins, but is usually wired via selected GPIO ports, which are used as generic GPIO pins (see figure below) when the CAN-function is not needed.

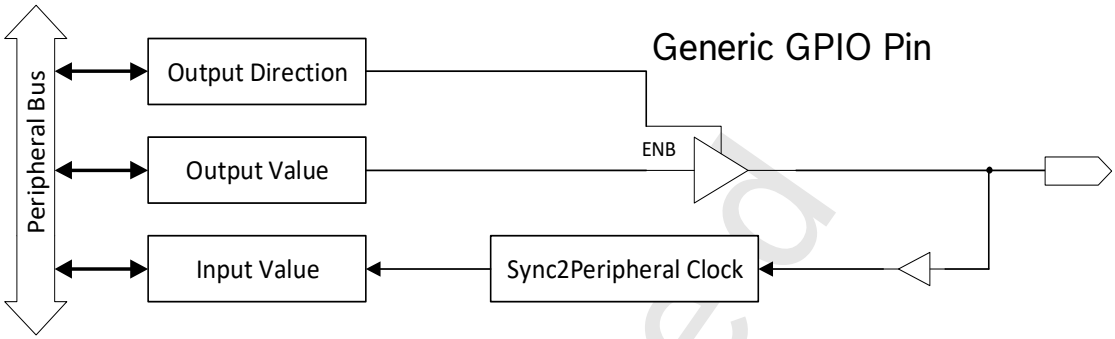


Figure 26. Generic GPIO Ports

General Purpose Input Output ports have a configurable direction, a writable output value, and a readable input value. The Direction FF enables the tristate output driver while the Output FF controls input of the output driver. The port's input value is synchronized to the clock of the peripheral logic in order to make it readable from the Input value FF. The direction, output, and input value FFs of several GPIO ports are usually collected in registers that are accessible via the µC's peripheral bus.

GPIO ports selected for CAN communication, to be connected to the transceiver's RxD and TxD pins, need two additional features, marked blue in figure below.

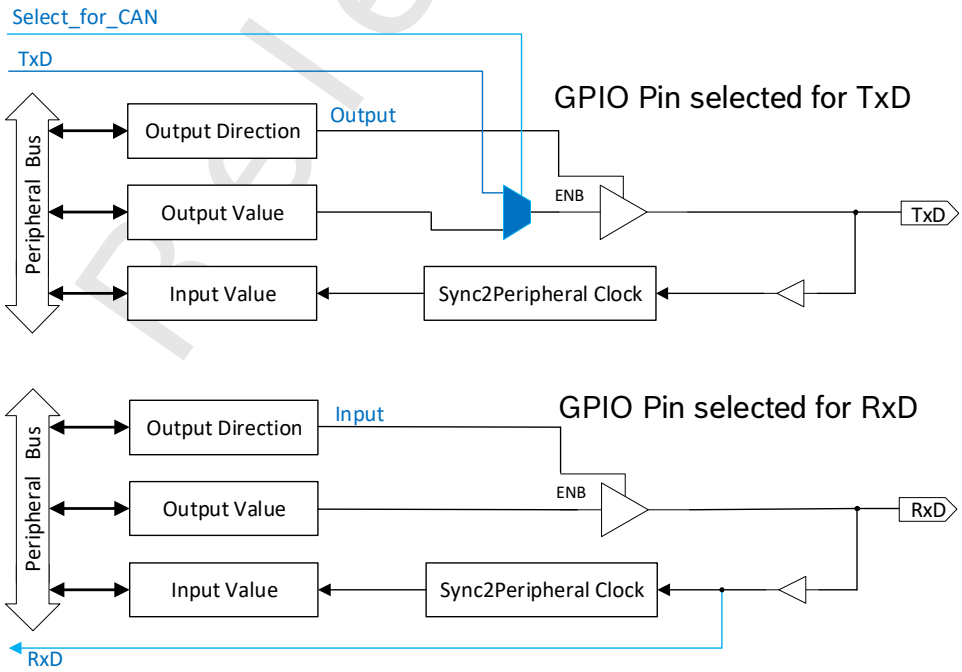


Figure 27. CAN XL Transceiver Interface Ports

The input value of the GPIO port selected as RxD is routed directly to the CAN-module's input CAN_RX, without synchronization to the peripheral clock. The direction of the GPIO port is set to input.

The direction of the GPIO port selected as TxD is set to output. A multiplexer in the GPIO logic allows the input of the TxD port's output driver to be directly driven by the PWME module's serial transmit output TxD or, when no PWME module is implemented, by the CAN module's serial transmit output CAN_TX.

When transceiver device and protocol controller are integrated together, the GPIO pins and the PWME module are omitted and the transceiver device is directly connected to the protocol controller's CAN_RX, CAN_TX, XLT, D_Rx and D_Tx pins.

1.6.6.14 Hardware Timestamping

1.6.6.14.1 Timestamping Function

Timestamps are captured for each transmitted or received message, captured at either the sample point of the start of frame bit of the message or at the sample point of the bit when the message becomes valid at the end of the frame. The capture position is defined by the configuration bit MODE.SFS.

When a timestamp is to be captured, the Protocol Controller toggles its capture output signal CAPTURE.

The Protocol Controller's input signal TIMESTAMP is a 64-bit value, captured outside the Protocol Controller in a dedicated CDC and capture module (see figure PRT Block Diagram).

Timestamps for transmitted messages are reported via TX_MSG after the message has been successfully transmitted.

Timestamps for received messages are reported via RX_MSG after the message has been successfully received.

1.6.6.14.2 Time Base Capturing and Clock Domain Crossing

The external time base is typically a timer inside the host controller, but it may also come from some other peripheral module. Since the host controller and other peripherals typically operate in other clock domains than the PRT, the functions of clock domain crossing and capturing is combined in a CDC & Capture module where the actual value of the time base (Host clock domain or timebase clock domain) is captured each time the PRT's CAPTURE output (CLK clock domain) toggles, see previous chapter. The captured value is provided to the PRT as the signal TIMESTAMP.

The Time Base is a 64-bit value, generated by one of the Host's timer modules, it is used for hardware timestamping of CAN messages. The Time Base need not to be in the CLK clock domain. A dedicated CDC and capture module (see Block Diagram) operate in the CAN clock domain. It synchronizes the Protocol Controller's CAPTURE output signal into the host clock (or timebase clock) domain, and it captures the Time Base each time the synchronized CAPTURE_C2H signal toggles. The captured Time Base value is provided, as the TIMESTAMP signal, to the Protocol Controller. The TIMESTAMP signal is updated within up to 16 clk cycles after the toggling of the CAPTURE output and it remains static until next time the CAPTURE output toggles.

An example for such a CDC and capture module is shown in figure below.

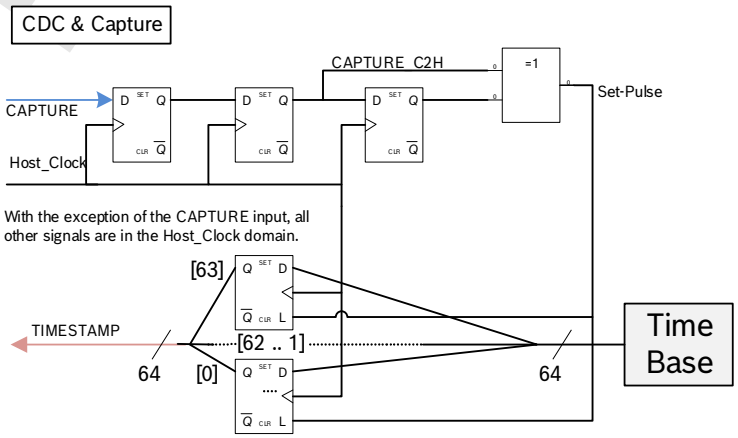


Figure 28. Example for Timebase CDC and Capture Module

1.6.6.15 Trace and Debug

The hardware test mode functions are disabled by the software reset of the PRT. The status flag TEST.HWT shows whether the hardware test mode functions are enabled.

When the hardware test mode functions are disabled, all bits of TEST are cleared with the exception of RXD. Enabling the hardware test mode functions (see chapter “Hardware Test Functions”) requires the application of the test mode key sequence. The test mode key sequence consists of three consecutive write accesses, not interrupted by other accesses via REG_APB:

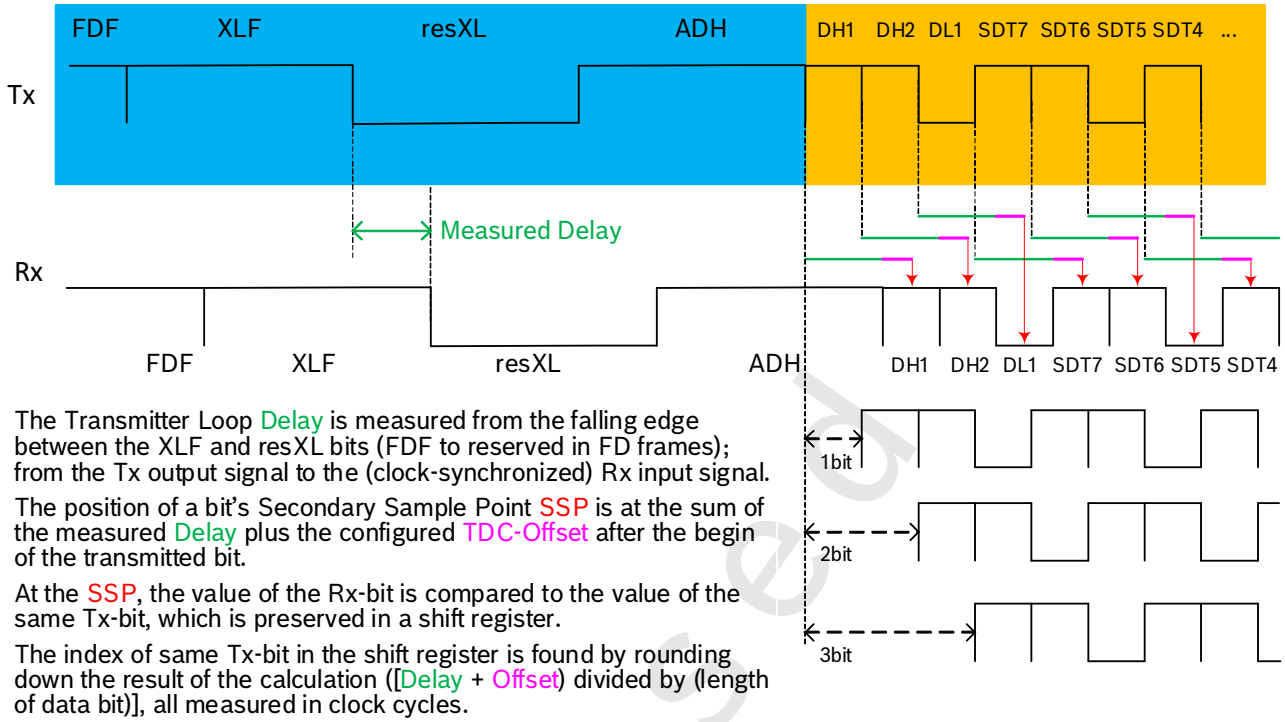
StepWrite data to Register at Address
1:Write 0x6789 to LOCK.TMK at Address 0x40
2:Write 0x9876 to LOCK.TM at Address K0x40
3:Write 1 to CTRL.TEST0x44
The hardware test mode functions enable the host to directly control the values driven at the transceiver interface pins, to read the actual transceiver RxD output, and to transmit messages in a loop-back mode where all transmitted messages are also reported through the RX_MSG interface as received messages.

The CAN_RX input (output signal of the transceiver) is always readable at RXD
The CAN_TX output control TXC offers four options:
0b00: Normal function of CAN_TX
0b01: Normal function of CAN_TX, CAN_RX is ignored (for message loop-back mode)
0b10: CAN_TX output set to 0 and XLT output set to 0
0b11: CAN_TX output set to 1 and XLT output set to 0
When LBCK is set, the PRT operates in the message loop-back mode. In message loop-back mode, the PRT reports transmit messages (requested through the TX_MSG interface) via RX_MSG as received messages. The transmit messages are encoded and decoded bitwise inside the PRT, but a transmitted message is treated as successfully transmitted even if it does not get ACK. When the host sets LBCK to 1, it also sets TXC to either 0b01 or to 0b11 to control whether messages transmitted in the message loop-back mode are visible at the transceiver pins. In the message loop-back mode with TXC set to a value > 0b00, the actual CAN_RX input (output signal of the transceiver) is ignored by the PRT.

When the host sets TXC=0b11 in the message loop-back mode, the PRT keeps the CAN_TX output at 1 and loops back its internal serial output signal to its internal serial input signal. With this configuration, the loopback transmission does not disturb a CAN bus system connected to its transceiver.

When the host sets TXC=0b01 in the message loop-back mode, the PRT drives the frame bits at its CAN_TX output. In this case, the PRT loops back its internal serial output signal to its internal serial input signal, so it is not able to perform an arbitration or to react on bit errors on the CAN bus.

1.6.7 Application Information



The Transmitter Loop Delay is measured from the falling edge between the XLF and resXL bits (FDF to reserved in FD frames); from the Tx output signal to the (clock-synchronized) Rx input signal.

The position of a bit's Secondary Sample Point SSP is at the sum of the measured Delay plus the configured TDC-Offset after the begin of the transmitted bit.

At the SSP, the value of the Rx-bit is compared to the value of the same Tx-bit, which is preserved in a shift register.

The index of same Tx-bit in the shift register is found by rounding down the result of the calculation ([Delay + Offset) divided by (length of data bit)], all measured in clock cycles.

Figure 29. Overview of Transmitter Delay Compensation

The Transmitter Delay Compensation TDC is needed for applications where the bit rate in the CAN FD or CAN XL data phase is so high (and therefore the data bit time so short) that the transmitter loop delay prevents the node from a meaningful bit error check at the Sample-Point.
The transmitter loop delay is the time from the CAN_TX output at the start of the transmitted bit through (if used) the PWM Encoder, then through the transceiver to the CAN bus and back through the transceiver to the CAN_RX input and through the input synchronization.
Any two CAN nodes may have a systematic phase-shift to each other by the amount of the sum of the transmitter loop delay and the bus line delay between the two nodes.
The CAN arbitration mechanism and the acknowledge signaling function require that the time from the start of a bit at the synchronization segment to the bit's Sample-Point must be at least twice that systematic phase shift. This defines the required length of the bit's propagation segment. The two phase-buffer segments and the resynchronization mechanism remain to compensate for the oscillator frequency differences between the nodes.
In CAN CC and in arbitration phase bit timing, the propagation segment is necessary and raises the minimum length of the bit time and therefore limits the achievable bit rate.
The CAN arbitration mechanism and the acknowledge signaling function are not used in the data phase of CAN FD or CAN XL, so in that phase no propagation segment is needed. When error signaling is enabled, the transmitter still needs to check for bit errors while operating in the data phase, this means that the time from the start of a bit at the synchronization segment to the bit's Sample-Point must be longer than the transmitter loop delay. For this calculation, the phase-buffer segment before the Sample-Point is included since here only the bits generated from the node's own local clock are regarded, no clock tolerances need to be considered.

When the transmitter loop delay is longer than the time from the start of the bit to the Sample-Point, the TDC mechanism needs to be enabled and the bit error check is then delayed to the time of the Secondary-Sample-Point, where the delayed input signal has arrived.

As shown in the TDC overview figure, the position of the Secondary-Sample-Point is not inside the transmitted bit, but inside one of the following Tx-bits. The TDC measures the delay in each transmitted FD or XL frame before it switches into the data phase. There may be small differences between successive measurement results due to changes in voltage, temperature, or input synchronization jitter. The measured delay (green line in the figure) shows the phase shift between the start of a transmitted bit at the Tx-output to the start of the same bit seen at the Rx-input of the PRT. The TDC offset (magenta line in the figure) needs to be configured to place the SSP at a position inside the same bit seen at the Rx-input. When the TDC detects a bit error at the SSP position, this information will be processed by the PRT at the following

regular Sample-Point. The optimum position of the SSP inside that bit depends on the analyzed properties of the physical layer. The length of the recessive and the dominant bits may become asymmetric due to signal reflections and the different driving strengths. When transceivers are used that drive more symmetric signal shapes (e.g., CAN SIC or CAN SIC XL types), the position of the SSP should be in the middle of the received bit. The TDC offset configuration must always be below the length of a data-phase bit time.

Usual CAN SIC transceivers have a maximum Tx-input to Rx-output delay of 190ns. Together with a digital delay of three clock periods, this results in a maximum transmitter loop delay of 209ns (160 MHz clock) or 228ns (80 MHz clock). Other transceivers, especially those including galvanic isolation, have longer delay times and require TDC even at lower bit rates.

CAN SIC transceivers are specified for bit rates up to 8 MBit/s (125ns bit time). For higher bit rates, CAN SIC XL transceivers are needed that operate in a dedicated XL mode during the data phase. TDC is not needed for these higher bit rates because the CAN XL protocol specifies that error signaling, and bit error checking are disabled when the transceiver is switched into the XL mode. So 125ns is the shortest bit time to be considered for the TDC mechanism.

At the Secondary-Sample-Point, the TDC mechanism compares the actual value of the Rx-input signal with a stored value of the Tx-output signal. For this purpose, the TDC stores the last nine transmitted bits in a shift register. To select the correct shift register cell (or the current Tx-bit), to be compared to the Rx-input value at the SSP, the TDC divides the SSP position (STAT.TDCV, number of clock periods after the start of the bit) by the length of one bit time in the data phase (calculated number of clock periods from the configuration of NBTP.BRP and DBTP or XBTP). This limits the transmitter loop delay that can be compensated to at most 10 bit times. With the shortest bit time of 125ns (8 MBit/s), this is 1250ns. At 5 MBit/s, the limit is 2000ns.

The second limitation for the maximum loop delay compensation comes from the calculation range. The TDC operates with a resolution of clock cycles in the range of 8 bit, so the latest SSP position is 255 clock cycles after the start of the Txbit,1593ns (160 MHz clock) or 3187ns (80 MHz clock). The maximum distance between two SSPs is also 255 clock cycles, so the maximum length of a data phase bit may not exceed the number 255. When TDC is used, NBTP.BRP must be configured to a value of 0x00 or 0x01.
The length of a CAN FD data bit time is (NBTP.BRP * (DBTP.TSEG1 + DBTP.TSEG2 + 3)) clock cycles.
The length of a CAN XL data bit time is (NBTP.BRP * (XBTP.TSEG1 + XBTP.TSEG2 + 3)) clock cycles.

The CAN XL protocol specification requires that node shall be able to compensate transmitter delays of at least 95 clock cycles, which is 593,75ns (160 MHz clock) or 1187.5ns (80 MHz clock).

When the transmitter loop delay is indeed at 95 clock cycles (upper range as required by protocol specification), this results in an upper limit for the configuration range of the TDC offset (DBTP.DTDCO or XBTP.XTDCO) to the number 160 (255 - 95). In systems with shorter transmitter loop delays, the TDC offset may be configured to a higher value. The TDC mechanism can compensate longer transmitter loop delays than 95 clock cycles, as long as the sum of the measured delay and the configured TDC offset does not exceed the number 255.

1.6.8 Verification and Validation Requirements

Not applicable.

1.6.9 Detailed Design Information

1.6.9.1 Port Description

Table 98. Port List (page 1 of 4)

Signal	Bit width	I/O	Description	Active level	Clock domain
CLOCK and RESET					
CLK	1	I	Clock input of the PRT	na	na
CLK_APB	1	I	Clock input of the PRTís REG_APB module, same domain as CLK	na	na
CLOCK_ACTIVE	1	I	Shows whether Clock input of the PRT is active	na	na

Table 98. Port List (page 2 of 4)

Signal	Bit width	I/O	Description	Active level	Clock domain
RESET_N	1	I	Module reset, externally synchronized to CLK domain	Low	CLK_APB
TIMESTAMP INTERFACE					
CAPTURE	1	O	Capture trigger for time base	na	CLK
TIMESTAMP	72	I	Time base captured as time stamp for CAN message	na	CLK
CONFIGURATION INTERFACE					
ONLY_CC	1	I	Restricts the PRT to Classical CAN frame format	High	na
ONLY_CC_FD	1	I	Restricts the PRT to Classical CAN and CAN FD frame formats	High	na
TSS_EN	1	I	Controls Transceiver Sharing Switch mode: enabled (1) or disable (0)	High	na
NO_SAFETY_NO_CTM	1	I	When set to 1, Time Stamp Parity check is disabled in PRT	High	na
TX_MSG INTERFACE, see [4]					
TX_MSG_WVALID	1	I	Write data valid	High	CLK
TX_MSG_WDATA	32	I	Write data	na	CLK
TX_MSG_WUSER	2	I	Write user code to control the sequence	na	CLK
TX_MSG_WREADY	1	O	Write ready from PRT	High	CLK
TX_MSG_BVALID	1	O	Response data valid	High	CLK
TX_MSG_BUSER_STATUS	3	O	Response user data status	na	CLK
TX_MSG_BUSER_TS	64	O	Response user data timestamp	na	CLK
TX_MSG_BREADY	1	I	Response ready from MH	High	CLK
RX_MSG INTERFACE, see [4]					
RX_MSG_WVALID	1	O	Write data valid	High	CLK
RX_MSG_WDATA	32	O	Write data	na	CLK
RX_MSG_WUSER	3	O	Write user data	na.	CLK
RX_MSG_WREADY	1	I	Write ready	High	CLK
MISC SIGNALS					
ENABLE	1	O	Serial CAN Bus Communication enabled	High	CLK
SP_BEFORE_DOM_REC_EDGE	1	O	Indicates the correct sampling of the arbitration phase	High Pulse	CLK
TRANSCEIVER INTERFACE					
CAN_RX	1	I	Serial CAN receive input from Transceiver	na	na
TXD	1	O	Serial CAN transmit output to Transceiver	Low	CLK
IS_TRANSMITTER	1	O	Current Transmitter of a CAN frame	High	CLK
GROUP_TR	1	I	Signals whether at least one of the CAN modules in the transceiver sharing group is a transmitter	High	CLK

Table 98. Port List (page 3 of 4)

Signal	Bit width	I/O	Description	Active level	Clock domain
INTERRUPT INTERFACE					
BUS_OFF	1	O	Stop of CAN module, either after CTRL.STOP command written or stop due to entering Bus_Off state (STAT.BO=1)	High pulse	CLK
BUS_ON	1	O	Signals starting of CAN module, is set immediately after CTRL.STRT command is written (this is also the case if CAN module was in Bus_Off state (STAT.BO=1))	High pulse	CLK
E_PASSIVE	1	O	Switching from Error-Active to Error-Passive	High pulse	CLK
E_ACTIVE	1	O	Switching from Error-Passive to Error-Active	High pulse	CLK
BUS_ERR	1	O	Error detected on CAN bus or Protocol Exception Event detected	High pulse	CLK
RX_EVT	1	O	Received a valid message	High pulse	CLK
TX_EVT	1	O	Successfully transmitted a message	High pulse	CLK
IFF_RQ	1	O	MH Requests a Message with Invalid Frame Format in Header	High pulse	CLK
RX_DO	1	O	Data Overflow condition in RX_MSG sequence detected	High pulse	CLK
TX_DU	1	O	Data Underrun condition in TX_MSG sequence detected	High pulse	CLK
USOS	1	O	Unexpected Start of Sequence during TX_MSG sequence detected	High pulse	CLK
PRT_STOP_IMMD_REQ	1	I	Indicates that PRT is stopped	High	CLK
ABORTED	1	O	TX_MSG sequence stopped by TX_MSG_WUSER code ABORT	High pulse	CLK
TS_PARITY_ERR_INT	1	O	Time Stamp Parity Error for TX and RX Path; it is checked when NO_SAFETY_NO_CTM is 0	High Pulse	CLK
HARDWARE DEBUG PORT					
CAN_TX	1	O	Serial CAN transmit output to PWME	Low	CLK
D_TX	1	O	Data Phase of transmitted frame active	High	CLK
D_RX	1	O	Data Phase of received frame active	High	CLK
XLT	1	O	Indicates an XL frame with transceiver mode switching is active	High	CLK
SAMPLE_POINT	1	O	CAN Sample Point	High pulse	CLK
RX_TSS	1	O	Status flag inidcating that XS_CAN IP is receiver when another XS_CAN IP sharing the same transceiver is transmitting.	High	CLK
STAT_ACT	2	O	Actual value of register STAT.ACT (see register description)	na	CLK
HOST INTERFACE (APB_REG)					
REG_APB_PADDR	8	I	Address	na	CLK_APB
REG_APB_PSEL	1	I	Select bit	High	CLK_APB

Table 98. Port List (page 4 of 4)

Signal	Bit width	I/O	Description	Active level	Clock domain
REG_APB_PENABLE	1	I	Enable bit	High	CLK_APB
REG_APB_PWRITE	1	I	Read or write bit	High	CLK_APB
REG_APB_PRDATA	32	O	Read data	na	CLK_APB
REG_APB_PWDATA	32	I	Write data	na	CLK_APB
REG_APB_PREADY	1	O	Used to extend an APB transfer by the Completer	High	CLK_APB
REG_APB_PSLVERR	1	O	Slave error	High	CLK_APB
REG_APB_PPROT	3	I	Protection unit support for reg_apb	na	CLK_APB
REG_APB_PSTRB	4	I	Write strobes for reg_apb	High	CLK_APB

1.6.9.2 Parameters

Table 99. PRT Parameters

Parameter name	Default value	Description/Constraints
NODE_G	1'b0	Used for simulation when using PRT internal monitor component xcan_prt_mon_trc.vhd.

1.6.10 PWME - Pulse Width Modulation Encoder

This document describes the PWME, the Pulse Width Modulation Encoder, and its interfaces with the CAN XL Protocol Controller and the CAN transceiver.

1.6.10.1 Overview

PWME is the Pulse Width Modulation Encoder build in the XS_CAN.

1.6.10.2 Features

PWM encoding as specified in [2]

1.6.10.3 Block Diagram

Figure “PWME overview” shows the PWME and its interfaces with the CAN XL Protocol Controller and the CAN transceiver.

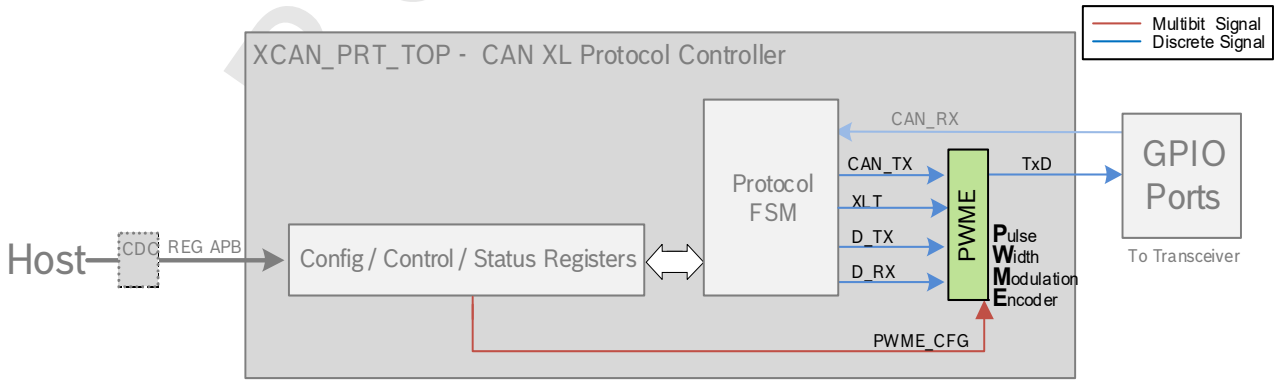


Figure 30. PWME Overview

1.6.10.4 Hardware Interface

Table 100. PWME Hardware Interface

Pin	Name	I/O	Description
TXD		O	Serial CAN transmit output to Transceiver

1.6.10.5 Software Interface

1.6.10.5.1 PWME Configuration (PWME_CFG)

The PWME_CFG contains the parameters needed for the PWM encoding (as defined in [2]) in the PWME module. PWME_CFG is assigned from PRT.PCFG register in PRT module.The PWME_CFG signal may not change its value while the PRT is started, i.e., while CAN frames can be received and transmitted.
PRT.PCFG[5:0] => PWME_CFG[5:0]
PRT.PCFG[13:8] => PWME_CFG[11:6]
PRT.PCFG[21:16] => PWME_CFG[17:12]

Table 101. PWME Configuration

Bits	Config	Description
[17:12]	PWMO[5:0]	PWM Offset
[11:6]	PWML[5:0]	PWM phase Long
[5:0]	PWMS[5:0]	PWM phase Short

Valid values for the PWM phase Short **PWMS** are 0x00-0x3F. The actual interpretation of this value is that the PWM short phase length is (**PWMS** + 1) clock cycles long.

Valid values for the PWM phase Long **PWML** are 0x00-0x3F. The actual interpretation of this value is that the PWM long phase length is (**PWML** + 1) clock cycles long.

The PWM symbol length is the sum of PWM short phase length and PWM long phase length (**PWMS** + **PWML**+ 2) clock cycles.

Valid values for the PWM Offset **PWMO** are 0x00-0x3F. PWMO shall always be smaller than the PWM symbol length (**PWMO** < **PWMS** + **PWML** + 2).

1.6.10.6 Functional Description

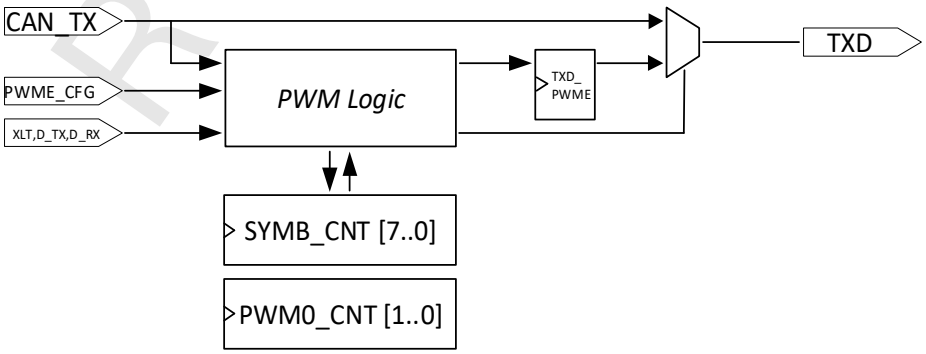


Figure 31. PWME Block Diagram

PWME implements the PWM encoding as specified in [2]. When transceiver mode switching is enabled, the PWME encodes the CAN_TX input signal during a CAN XL frame's data phase and during ADH bit, to generate the PWM encoded output signal TXD.

The output of the PWM Logic is registered by the Flip Flop TXD_PWME. An active Reset sets this Flip Flop TXD_PWME to one.
All Flip Flops in the PWME change their value with the rising edge of CLK.

1.6.10.6.1 Transparent Mode

While XLT is passive or both D_RX and D_TX are passive, the PWME interface behaves transparent between CAN_TX input and TXD output.

This Mode is independent of Reset.

1.6.10.6.2 PWM encoded Mode

While XLT is active, the TXD output is PWM encoded for a transmitting node while D_TX is active and for the receiving node while D_RX active.

The PWM encoded TXD output has one CLK cycle delay (internal processing delay) relative to the bit boundaries on CAN_TX input.

1.6.10.6.2.1 Transmitting Mode

When the PWME detects an edge from passive to active on D_TX and if XLT is active, then the PWME drives a LOW level on TXD for one PWM Offset time.

With expiration of the PWM Offset time the TXD output drives 2 consecutive PWM_0 symbols. From there onwards all following PWM symbols follow the CAN_TX input.

When D_TX is passive or XLT is passive the PWME switches back to transparent behavior regardless of the actual PWM phase.

1.6.10.6.2.2 Receiving Mode

When the PWME detects an edge from passive to active on D_RX and if XLT is active, then the PWME drives consecutive PWM_1 symbols without a leading Offset time.

When D_RX is passive or XLT is passive the PWME switches back to transparent behavior regardless of the actual PWM phase.

1.6.10.6.3 Multiplexer switching

To avoid glitches on the TXD output, the multiplexer before the TXD output shall switch only, when both input signals (CAN_TX and TXD_PWME) are stable.

In Transparent Mode the Flip Flop TXD_PWME takes over the value of CAN_TX.
Switching from CAN_TX Input to TXD_PWME Flip Flop (Receiving Node):
The select input of the multiplexer changes with the passive to active edge of D_RX.
During this first CLK cycle after the D_RX signal edge, the multiplexer input signals CAN_TX and TXD_PWME have both the logical value one.

Due to the internal processing delay of one CLK cycle, the signal TXD_PWME keeps its value constant for one more CLK cycle after the D_RX signal edge.

Switching from CAN_TX Input to TXD_PWME Flip Flop (Transmitting Node):
The select input of the multiplexer changes two CLK cycles after the edge between XLF-Bit to resXL-Bit. During this time the multiplexer input signals CAN_TX and TXD_PWME have both the logical value are zero.
The Flip Flop TXD_PWME takes over the value of CAN_TX (one CLK cycle delay) until up to the point in time, when PWM encoding starts.

PWM encoding starts earliest one CLK cycle after the following bit boundary.

Switch back to CAN_TX Input (Transparent):
The select input of the multiplexer changes one CLK cycle after the active to passive edge of D_RX or respectively D_TX.

At this time CAN_TX is stable and TXD_PWME is clamped to its previous value. The values of CAN_TX and TXD_PWME may differ.

1.6.10.7 Safety Mechanisms

Not applicable.

1.6.10.8 Application Information

Not applicable.

1.6.10.9 Integration Guidelines

Not applicable.

1.6.10.10 Verification and Validation Requirements

Not applicable.

1.7 MH-PRT Interface

1.7.1 Introduction

The CAN XL Protocol Controller (PRT) a kind of a bridge between the serial CAN Bus and the 32 bit Message Busses. The PRT does not provide internal buffering of frames, so that data has to be transferred by Message Busses in 32 bit slices in real-time while (de)-serialisation on the CAN Bus. Thus, single data transfers at the Message Busses are closely time-synchronised to the schedule at the CAN bus.

The Message Handler (MH) is required to relax the timing requirements of the Message Busses by further buffering of data. Usually such a MH will provide multiple FIFOs for CAN Frames received at the CAN Bus, and vice versa multiple FIFOs/Priority Queues for CAN Frames to be transmitted at the CAN Bus.

This document describes the interfacing between MH and PRT, consisting of the two Message Busses TX_MSG and RX_MSG and the discrete signal ENABLE. The other signals are not regarded in this specification and thus they are omitted.

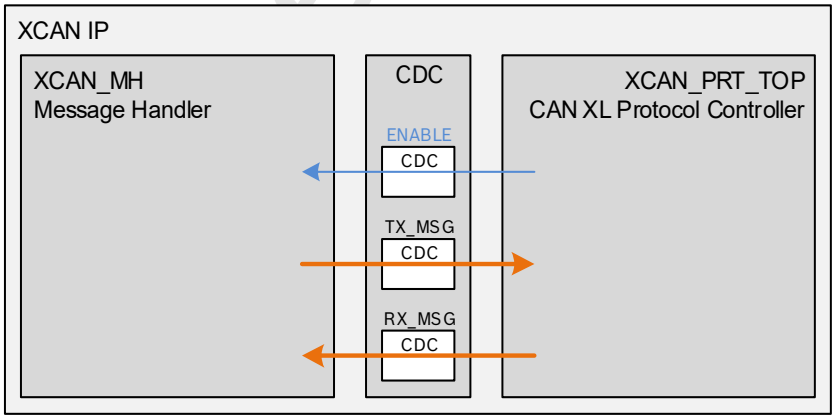


Figure 32. MH-PRT Block Diagram

Messages to be transmitted on CAN Bus are transported from the MH to the PRT via point-to-point TX_MSG Message Bus. Furthermore, status information is exchanged bidirectional, so that transmissions on the CAN Bus are completely handled by that single interface.

Received messages are transported from the PRT to the MH via the point-to-point RX_MSG Message Bus together with status information. Receptions on the CAN Bus are completely handled by that single interface.

The TX_MSG and RX_MSG Messages Busses are independent of each other, each working autonomously.
When MH and PRT belong to different clock domains, the Clock Domain Crossing should be done as depicted in Figure 1.

The signal ENABLE is driven by PRT and is used to enable or immediately abort communication via the Message Busses.

The timing calculations in this document are based on following assumptions:

- CAN CC: 1Mbps bit rate, clock tolerance df ≤ +/-0.3%
- CAN FD: 1Mbps arbitration bit rate & 5Mbps data bit rate, clock tolerance df ≤ +/- 0.3%
- CAN XL: 1Mbps arbitration bit rate & 15Mbps data bit rate, clock tolerance df ≤ +/- 0.2%

1.7.2 TX_MSG

Messages to be transmitted on CAN Bus are transported from the MH to the PRT via point-to-point TX_MSG Message Bus in a sequence of transactions. Each transaction is an atomic data package, consisting of the request to serialize that data to CAN Bus and the response, if serialisation of this data was successful or not. Therefore, the interface consists of two channels, a Write Data Channel and a Write Response Channel.

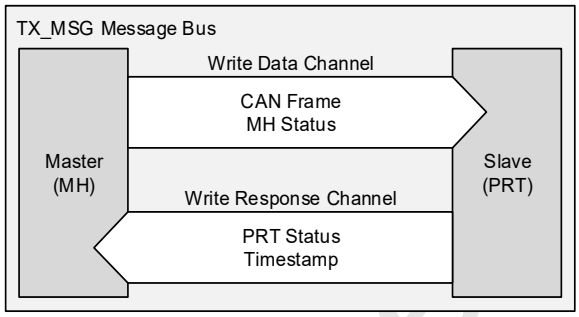


Figure 33. TX_MSG Message Bus

Only the MH initiates transactions. A transaction consists of one transfer on the Write Data Channel (the start of a transaction) and one transfer on the Write Response Channel which completes the transaction.

The assignment between Data and its dedicated Response is defined by order, there are no timing constraints between both channels.

The MH can initiate one new transaction, although there is one outstanding transaction, i.e. the current transaction has not been completed by a response.

The transport of a message requires a sequence of transactions. In the following chapters such TX_MSG sequences will be explained via chronograms. The next figure gives an introduction how information is illustrated for the Write Data Channel and Write Response Channel.

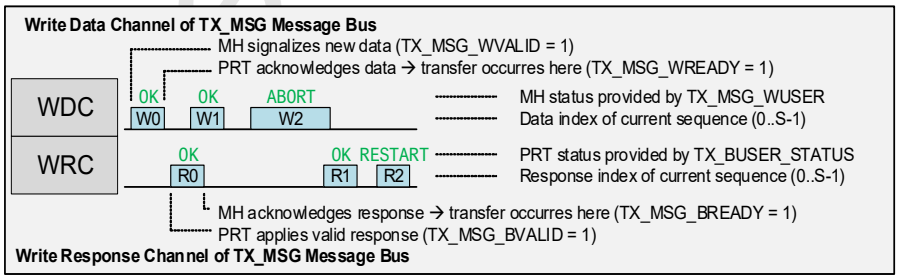


Figure 34. Introduction to TX_MSG Chronogram

1.7.2.1 Physical Interface

Table 102. Physical Interface of MH-PRT (page 1 of 2)

Port	MH	PRT	Active	W	Description
TX_MSG Write Data Channel					
TX_MSG_WVALID	OUT	IN	high	1	Write data valid
TX_MSG_WDATA	OUT	IN	na	32	Write data (frame word)
TX_MSG_WUSER	OUT	IN	na	2	Write user data (MH status information)
TX_MSG_WREADY	IN	OUT	high	1	Write data acknowledge

Table 102. Physical Interface of MH-PRT (page 2 of 2)

Port	MH	PRT	Active	W	Description
TX_MSG Write Response Channel					
TX_MSG_BVALID	IN	OUT	high	1	Response data valid
TX_MSG_BUSER_STATUS	IN	OUT	na	3	Response user data (PRT status reports)
TX_MSG_BUSER_TS	IN	OUT	na	64	Response user data (Timestamp)
TX_MSG_BREADY	OUT	IN	high	1	Response data acknowledge

All signals belong to the clock domain of the dedicated module, i.e. MH respective PRT. Optional clock domain crossing is considered as depicted in Figure Block diagram.

1.7.2.2 TX_MSG Write Data Channel

The TX_MSG Write Data Channel is used to transport information from MH to PRT via two vectors: TX_MSG_WDATA transports CAN Frame data in slices of 32 bit words and TX_MSG_WUSER provides related status information.

1.7.2.2.1 TX_MSG_WDATA

Depending on the CAN Frame Format, the first three TX_MSG_WDATA words of a sequence have different contents. The bit **T1.FIR** (Fault Injection Request) is not part of the CAN message and is used to control special functions of the PRT.

For CAN XL frames (XLFF):

Table 103. TX_MSG_WDATA (CAN XL)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T0	FDF=1	XLF=1	XTD=0	Priority ID[28:18]											RRS	SEC	VCID[7:0]								SDT[7:0]							
T1		FIR	EFC(host)	0		DLC-XL[10:0]									Message marker (HOST)																	
T2	AF[31:16]																AF[15:0]															

T1 Bit 31: **EFC - Event FIFO control bit** [0 - TX Event FIFO is not stored for this TX message; 1 - TX Event FIFO is stored for this TX message]
T1 Bit 15:0 - **Message Marker**
The third word T2 is used for the AF (Acceptance Field, a 32 bit value).

For CAN FD frames (FBDF, FEDF):-

Table 104. TX_MSG_WDATA (CAN FD)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T0	FDF=1	XLF=0	XTD	Base ID[28:18]											Extended ID[17:0]																	
T1		FIR	EFC(host)	0		0	BRS	ignored				ESI	DLC[3:0]				Message marker (HOST)															

For CAN CC frames (CBDF, CEDF, CBRF, CERF):-
EFC - Event FIFO control bit
0 - TX Event FIFO not stored for this TX message
1 - TX Event FIFO is stored for this TX message

Table 105. TX_MSG_WDATA (CAN CC)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T0	FDF=0	XLF=0	XTD	Base ID[28:18]											Extended ID[17:0]																	
T1		FIR	EFC(host	0		RTR	ignored						DLC[3:0]		Message marker (HOST)																	

1.7.2.2.2 TX_MSG_WUSER

The vector TX_MSG_WUSER provides status information related to the vector TX_MSG_WDATA using the following codes:

Table 106. TX_MSG_WUSER Codes

Code Value	Code Name	Description
0	OK	Code for transactions of an ongoing sequence.
1	RESERVED	Do not use, will be interpreted like ABORT until defined otherwise.
2	ABORT	Initiate abortion of current sequence.
3	SOS	Code for the first transaction of a sequence (start of sequence).

The following chronogram shows the usage of all TX_MSG_WUSER codes.

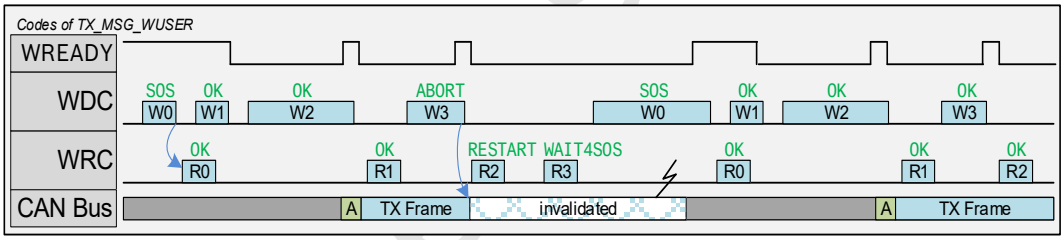


Figure 35. Codes of TX_MSG_WUSER

1.7.2.2.2.1 Handling of TX_MSG_WUSER

The code **ABORT** is allowed for all transactions except the first transaction of a sequence which requires the code **SOS**.

Since the abortion has an impact to the CAN Bus, it should only be used if a serious problem was detected by the MH during an ongoing sequence to prevent the successful transmission of a previously requested message, e.g. due to a descriptor CRC error in a subsequent descriptor.

In the figure above Codes of TX_MSG_WUSER, the MH starts a sequence with W0 using code **SOS** and requests for an abortion of the sequence with W3 using code **ABORT**.

1) Effect on TX_MSG Sequence

The transaction with the TX_MSG_WUSER code **ABORT** finishes an ongoing TX_MSG sequence.
The time of the response and the response code depend on when the **ABORT** command is given in relation to the requested transmission and to the PRT's ability to accept a new transaction.
If the **ABORT** command is given and accepted before the transmission was started on the CAN bus (e.g. during a reception), the transmission will not be started, and the response **RESTART** is given immediately.

If the **ABORT** command is given and accepted after the transmission was started on the CAN bus, but before the FCRC bits are transmitted, the transmission will be continued with inverted FCRC bits (as specified in 2.3.3.2) and the response **RESTART** is given immediately. The inversion of the FCRC bits makes this transmission invalid for all receivers, avoiding the processing of invalid data.

If the **ABORT** command is given and accepted after the FCRC bits were transmitted, but before the transmission is finished, the transmission will be continued. Even if this transmission is acknowledged on the CAN bus, the **ABORT** command gets the response **RESTART** immediately, the TX_MSG sequence will not get the response **ACK**.
When the already started transmission on the CAN bus is continued after the **ABORT** command is given and accepted, the PRT will not be able to accept further transactions while that transmission is still ongoing.

In all cases, if there are, after the response **RESTART**, still outstanding responses for previous transactions, they will each get the response **WAIT4SOS**, as for a spare transaction.

2) Actions of the PRT

If the **ABORT** command is given before the transmission was started on the CAN bus, the PRT will not start the transmission.
If the **ABORT** command is given after the transmission was started on the CAN bus, the PRT will set an internal flag that causes the FCRC bits to be transmitted inverted (as specified in 2.3.3.2). This internal flag is cleared at the end of the transmission.

3) Actions of the MH

MH will get feedback **RESTART**
MH shall not start new TX attempt for this message

1.7.2.3 TX_MSG Write Response Channel

The TX_MSG Write Response Channel is used by PRT to provide status feedback to the MH for each transaction. It will be provided when relevant status information inside PRT has been collected, that belongs to the current TX_MSG transaction. The status is transported via TX_MSG_buser_status and consist of the following codes:

Table 107. TX_MSG_BUSER_STATUS Codes

Code Value	Code Name	Description
0	OK	Code for transactions of an ongoing sequence.
1	ACK	Initiate stop of current sequence, due to transmission on CAN bus has been completed successfully.
2	RESTART	Initiate stop of current sequence, due to transmission on CAN bus was not successful, e.g. disturbed by error, did not get ACK.
3	USOS	Initiate stop of current sequence, due to USOS (Unexpected Start of Sequence).
4	HFI	Initiate stop of current sequence, due to HFI (Header Format Invalid). HFI can only be given by response R1.
5	DU	Initiate stop of current sequence, due to DU (Data Underrun).
6	WAIT4SOS	When PRT expects a transaction starting a new sequence but TX_MSG_WUSER is not code SOS, then the PRT responds with code WAIT4SOS (Wait for Start of Sequence) and discards the data of the transaction.
7	ARBLOST	Initiate stop of current sequence, due to arbitration on CAN Bus was lost or an RX_MSG sequence has reached the codes TS0 or ABORT, see Error! Reference source not found.. ARBLOST can only be given by response R1.

The timestamp of the CAN Frame is provided by vector TX_MSG_BUSER_TS when TX_MSG_BUSER_STATUS = **ACK**. For all other codes, TX_MSG_BUSER_TS is invalid.

1.7.2.3.1 Code HFI

The bits **T0.XLF**, **T0.FDF**, **T0.XTD** and **T1.RTR** define the CAN Frame Format. The following table shows all possible bit combinations.

Table 108. All Possible Formats

XLF	FDF	XTD	RTR	Frame	Description
1	1	0	-	XLFF	CAN XL Data frame
1	1	1	-		invalid
1	0	-	-		invalid
0	1	1	0	FEDF	CAN FD Extended Data Frame
0	1	0	0	FBDF	CAN FD Base Data Frame
0	1	-	1		invalid
0	0	1	1	CERF	Classical CAN Extended Remote Frame
0	0	1	0	CEDF	Classical CAN Extended Data Frame
0	0	0	1	CBRF	Classical CAN Base Remote Frame
0	0	0	0	CBDF	Classical CAN Base Data Frame

When the requested CAN Frame Format is not valid because e.g. disabled by the PRT's configuration, the PRT will initiate the stop of the current sequence by responding at R1 with code **HFI** and will not start a transmission on the CAN Bus.

The following chronogram shows the TX_MSG sequence for the case of **HFI** (Header Format Invalid).

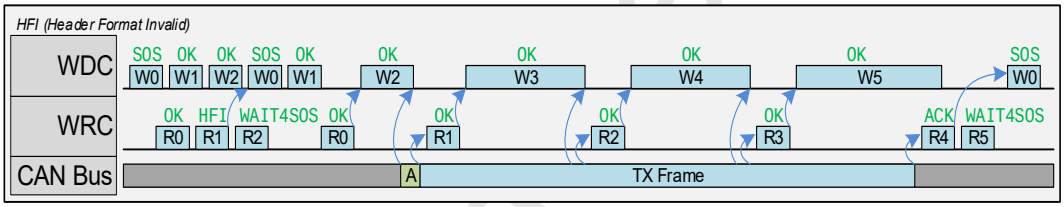


Figure 36. TX_MSG Sequence with Code HFI

The response at R1 with code HFI initiates the stop of current sequence, which triggers the MH to start a new sequence, if a TX Frame is available.

In this example the MH already started the next transaction by applying W2. Such an extra transaction will be responded with code WAIT4SOS and W2 is discarded by PRT. MH aborts MEM pending read access if any for the frame that resulted in **HFI** code by pulling LMEM_CS low.

Afterwards, a sequence with a successful transmission on the CAN Bus follows. Note that in this example the last data transaction W4 of the TX_MSG sequence is followed by an extra transaction (W5) that is discarded by PRT.

1.7.2.3.2 Code USOS Unexpected Start of Sequence

As the name tells, this condition is detected by the PRT when it gets a TX_MSG transaction with TX_MSG_WUSER = **SOS** (Start of Sequence) while another TX_MSG sequence is still ongoing. This can only happen when MH and PRT are mis-synchronized and causes the PRT to stop. The PRT stops operation, clears first TX_MSG_WREADY and then ENABLE.

MH resets its Retransmission Counter and stops feeding the PRT until ENABLE is set again. Restarting the PRT will re-synchronize the PRT to the MH.
The following chronogram shows the TX_MSG sequence for the case of an unexpected Start of Sequence.

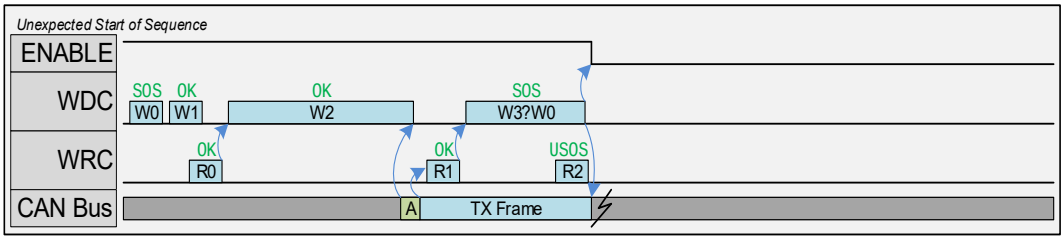


Figure 37. TX_MSG Sequence with Unexpected Start of Sequence

The sequence starts identical to other examples, then W3 is issued with code SOS, which is not allowed during an ongoing sequence.

When W3 is acknowledged, the PRT is stopped, i.e. it immediately stops CAN protocol operation. With this a partially sent message at the CAN bus will not be completed, preventing other nodes of using this frame.

1.7.2.3.3 Code DU Data Underrun

This error condition is detected when the PRT has started a transmission and reaches a point in that transmission where it needs to reload its transmit shift register with new data that have not yet been delivered by the MH. It can only occur when a transaction of the TX_MSG sequence is late (when e.g. the DMA access to the system RAM is delayed during high system load) or missing (when e.g. the MH fails).

The following chronogram shows the TX_MSG sequence for the case of a data underrun.

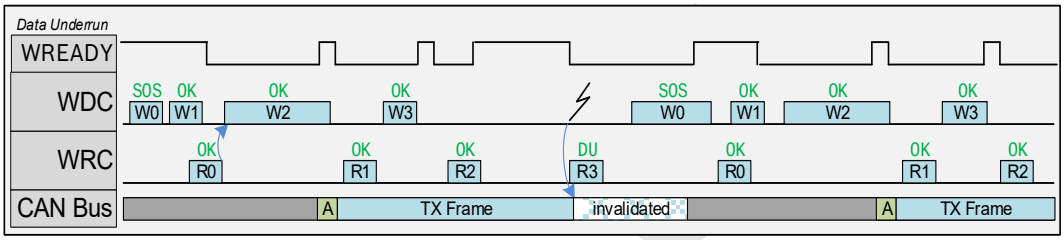


Figure 38. TX_MSG Sequence with case of Data Underrun

When MH provided W0 and W1, the PRT will attend the next arbitration on the CAN Bus. Once the transmission to the CAN Bus has been started, the data via TX_MSG is required within certain time windows, see chapter 2.5. In this example, W4 is not delivered in time, which is the underrun condition.

1.7.2.3.3.1 Effect on TX_MSG Sequence

Detection of the data underrun condition causes the PRT to immediately respond to the previous transaction with the TX_MSG_WUSER code DU (R3) which finishes the ongoing TX_MSG Sequence.

1.7.2.3.3.2 Actions of the PRT

The PRT clears TX_MSG_WREADY (to show that it is no longer able to accept further transactions) at the same time it gives the TX_MSG_WUSER code **DU**. The PRT continues the ongoing transmission even after it detects this error condition (to avoid disturbing the message schedule on the CAN bus), but it will set an internal flag that causes the FCRC bits to be transmitted inverted (to avoid the acceptance of a message containing invalid data). This internal flag is cleared at the end of the transmission and TX_MSG_WREADY is asserted again to accept the following TX_MSG Sequence (starting again with W0).

1.7.2.3.3.3 Actions of the MH

In case of **DU**, the MH:
• If a DU occurs due to a delay in receiving LMEM_READY for a data read operation, the corresponding LMEM read command is aborted, and a dummy word is transmitted to the PRT to obtain the DU code word from the PRT. MH Stops feeding PRT with remaining data.
• Counts **DU** as normal transmission attempt (equal to case: error on bus)
• Will behave like after usual **RESTART**• Provides same or next message to PRT

1.7.2.3.4 Codes OK, ACK and WAIT4SOS

The following chronogram shows a TX_MSG sequence for the case of a successful transmission of a CAN XL frame (XLFF) with 7 byte payload followed by one spare transaction.

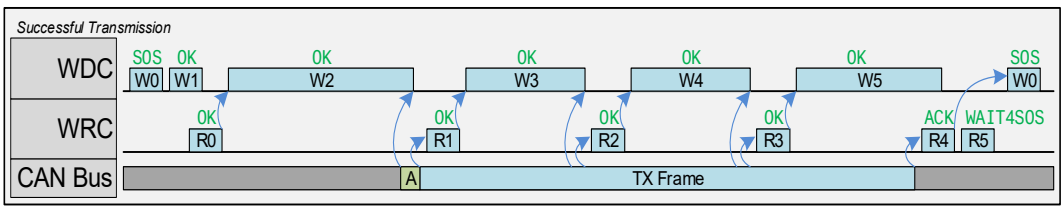


Figure 39. TX_MSG Sequence of a Successful Transmission

The MH provides W0 and W1 which contains the data needed by the PRT to attend an arbitration on the CAN Bus. Then the MH waits for R0 before it can start the next transaction by applying W2 (only one outstanding transaction is allowed). PRT will acknowledge W2 as soon as its local buffer gets free. When the arbitration on the CAN Bus was won, the PRT provides R1 with code **OK**. The response R3 is provided by the PRT as soon as the dedicated 32 bits are serialized to the CAN Bus and the result of the serialisation is known. The same is done for transaction 4. In this example W4 contains the last payload bytes of the CAN Frame.

The successful transmission on the CAN Bus will be signalized by R4 with code **ACK** together with the timestamp via vector **TX_MSG_BUSER_TS**. This initiates the stop of the current sequence and triggers the MH to start a new sequence when a CAN Message for transmission is available.

The MH may start transactions (considering one outstanding transaction) as long as the stop of the current sequence was not initiated. In this example, W4 provides the last payload data for the CAN Frame and transaction 5 is not required. The stop of the sequence will be initiated by R4 with code **ACK** and the extra transaction 5 (and all following extra transactions) will be respond with code **WAIT4SOS** and will be discarded by PRT.

Note: The code **WAIT4SOS** is intended for MH implementations that do not calculate the number of required TX_MSG transactions from frame format and DLC, but that e.g. always transfer the whole content of a transmit buffer. In that case, the extra transactions following the last payload word (until the next start of sequence) are responded with this code and will be discarded by the PRT.

1.7.2.3.5 Code RESTART

The following chronogram shows a TX_MSG sequence for the case of an unsuccessful transmission on the CAN Bus.

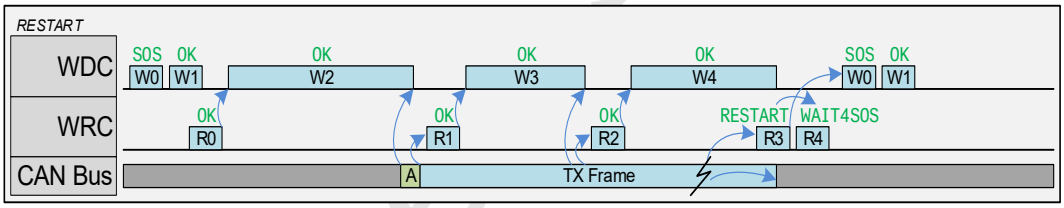


Figure 40. TX_MSG Sequence with Code RESTART

The sequence starts identical to last example, but the PRT aborts the serialisation during transaction 3 (W3/R3) due to a problem on the CAN Bus. This event is signalized by response R3 with status **RESTART**, which initiates the stop of the current TX_MSG sequence. This triggers the MH to start a new sequence if a TX Frame is available. MH aborts any pending LMEM read access for the frame which resulted in RESTART response by pulling LMEM_CS low.

In this example the MH already started the next transaction by applying W4. Such a transaction (and all following transactions belonging to the stopped TX_MSG sequence) are now considered extra transactions and are responded with code WAIT4SOS (R4) Extra transactions are discarded by the PRT.

1.7.2.3.6 Code ARBLOST

The following chronogram shows the TX_MSG sequence for the case of a lost arbitration at the CAN Bus.

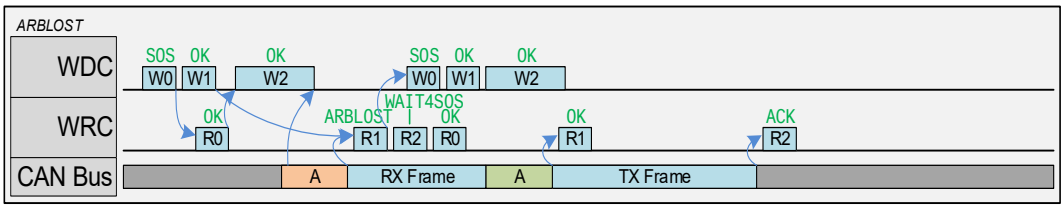


Figure 41. TX_MSG Sequence with Code ARBLOST

The MH transfers W0 and W1, then it waits for R0 before it may start the next transaction by transferring W2 (only one outstanding transaction is allowed).

When Arbitration was lost on the CAN Bus, the PRT responses at R1 with code **ARBLOST**, which initiates the stop of current sequence. This triggers the MH to start a new sequence if a TX Frame is available.

In this example the MH already started the next transaction 2. Such an extra transaction will be responded with code **WAIT4SOS** and W2 will be discarded by PRT.

Note: The code **ARBLOST** is given not only when a CAN bit arbitration was lost, but also when the transmission could not be started immediately because a reception had started earlier. This reception might delay the transmission for a very long time, e.g. when the received frame's DLC is large. For this case, the code **ARBLOST** is given when the reception ends successfully (**TS0**) or is disturbed by an error (**ABORT**). This gives the MH the opportunity to re-check which Tx-message then has the highest priority and should be transmitted next as well as to start a new TX_MSG sequence. This (maybe newly found) Tx-message will be started after Intermission on the CAN bus and may arbitrate with other possible messages on the CAN bus.

1.7.2.4 Shortest and Longest TX_MSG sequence

The TX_MSG sequence becomes shortest, when transmitting a CAN CC or CAN FD frames with no data bytes (Data Length Code = 0). Such a sequence consists of two transactions.

In this special case, R1 cannot be issued directly after arbitration on the CAN Bus like in all other cases. Instead it will be delayed until CAN Frame was completed and thus the result of transmission will be provided via R1.

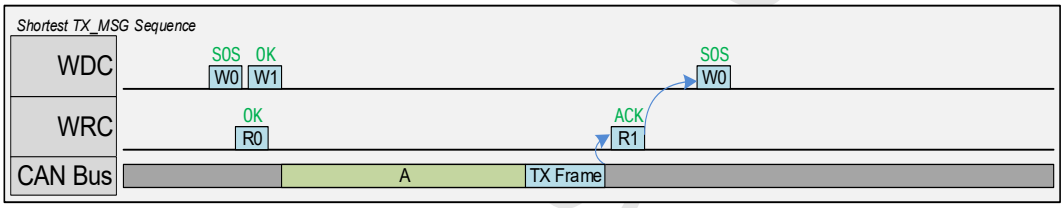


Figure 42. Shortest TX_MSG Sequence

The TX_MSG sequence becomes longest, when transmitting CAN XL Frame with maximum payload. Such a sequence consists of 515 transactions.

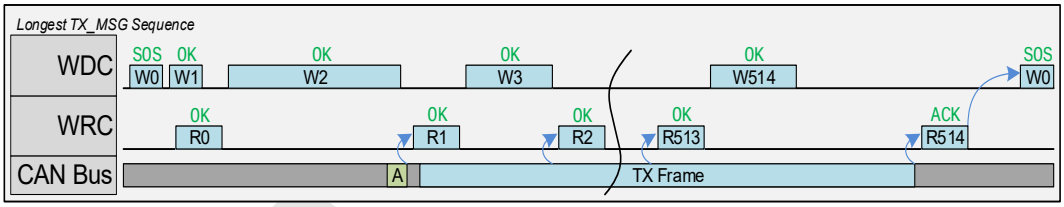


Figure 43. Longest TX_MSG Sequence

1.7.2.5 TX_MSG timing

This chapter shows the timing requirements of the PRT to exchange data via TX_MSG in time with the MH. All mentioned time budgets are the lowest possible time budgets. These times are determined by serialisation of the TX Element to the CAN Bus under worst case scenarios, i.e. where the dedicated time budget becomes smallest.

For better presentation, the scale between various transfers was relinquished.

1.7.2.5.1 Timing t_RS-W1

When a stop of the current TX_MSG sequence was initiated by response RS, the MH has to provide W0 and W1 for the next sequence within the time window **t_RS-W1 = 2 µs** to ensure, that PRT may attend the next arbitration on the CAN Bus, as long as TX Elements are available at the MH.

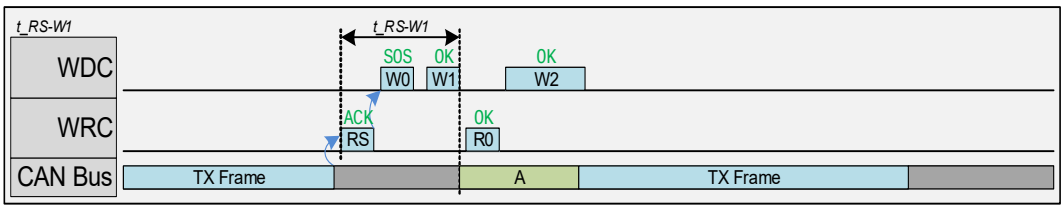


Figure 44. TX_MSG Sequence Timing t_RS-W1

1.7.2.5.2 Timing t_W1-W2 and t_W-W

The MH has to provide W2 within the time window **t_W1-W2 = 15 µs** after W1 transfer occurred. This time is defined by the first part of the CAN Frame (mainly the arbitration) until W2 is needed for serialisation to the CAN Bus.

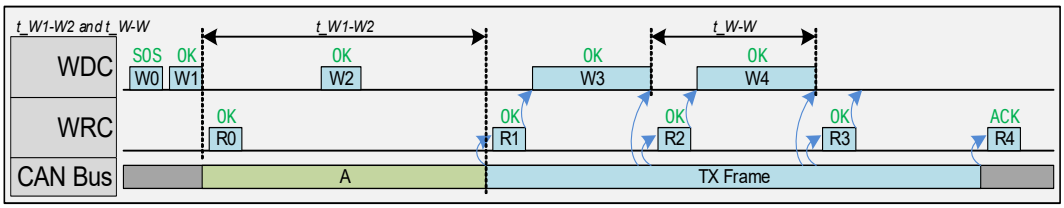


Figure 45. TX_MSG Sequence Timings t_W1-W2 and t_W-W

The time budget to apply all other data after W2 is **t_W-W = 2.13 µs**. The time is defined by serializing 32 bit with 15 Mbit/s on the CAN Bus.

1.7.2.5.3 Start of a new sequence by MH

The MH should do a priority scan whenever the content of a FIFOs or Priority Queue has been changed, e.g. the host scheduled a new TX Element, which happens asynchronous to the CAN Bus.

To base on the latest result of the priority scan, the TX_MSG sequence should start as close as possible to the point of time when the CAN Bus becomes available for transmission, i.e. when a previous CAN Frame finishes.

In following chronogram, the MH applies new data and the subsequent arbitration will be lost, and PRT starts the reception on the CAN Bus. The response R1 with code **ARBLOST** triggers the MH to start a new TX_MSG sequence.

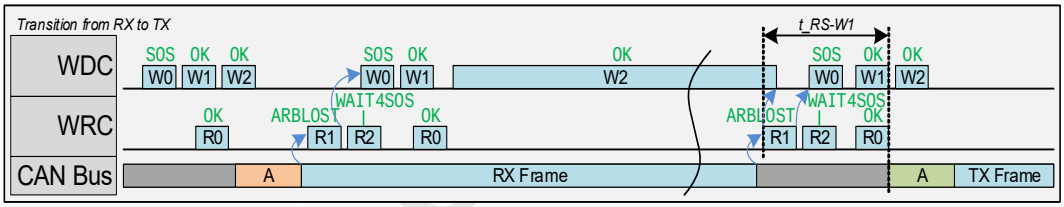


Figure 46. Transition from RX to TX

For this, the PRT will initiate the stop of a potentially started TX_MSG sequence via R1 with code **ARBLOST** when the RX_MSG sequence has reached the transfer with code **TS0** or **ABORT**. With this, the MH can start a new sequence, time synchronized to the CAN Bus, providing a TX Element based on the latest priority scan.

Note: The **ARBLOST** code triggers the MH to restart a TX_MSG sequence, or to start another sequence, it does not require the MH to trigger a new TX_SCAN.

The new start of a TX_MSG sequence gives the MH the opportunity to switch the Tx-message from the previous TX_MSG sequence to another Tx-message that may have been found by an intermediate TX_SCAN (which was triggered by other events).

1.7.3 RX_MSG

Received messages are transported from the PRT to the MH via the point-to-point RX_MSG Message Bus. Each transfer is an atomic data package, consisting of de-serialized data from the CAN Bus combined with the status of the de-serialisation.

The RX_MSG Message Bus consist of a Write Data Channel to transport the CAN Frame, status information of PRT and timestamp of CAN Frame to the MH. Status feedback from MH to PRT is not supported.

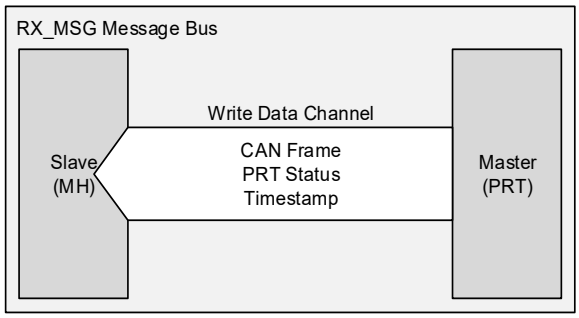


Figure 47. RX_MSG Message Bus

1.7.3.1 RX_MSG Physical Interface

Table 109. RX_MSG Physical Interface

Port	PRT	MH	Active	W	Description
RX_MSG Write Data Channel					
RX_MSG_WVALID	OUT	IN	high	1	Write data valid
RX_MSG_WDATA	OUT	IN	na	32	Write data (frame word & timestamp)
RX_MSG_WUSER	OUT	IN	na	3	Write user data (PRT status information)
RX_MSG_WREADY	IN	OUT	high	1	Write data acknowledge

All signals belong to the clock domain of the dedicated module, i.e. MH respective PRT.

1.7.3.2 RX_MSG Write Data Channel

The RX_MSG Write Data Channel is used, to transport information from PRT to MH via two vectors: RX_MSG_WDATA transports CAN Frame data and timestamp in slices of 32 bit words and RX_MSG_WUSER provides related status information.

1.7.3.2.1 RX_MSG_WDATA

Depending on the CAN Frame Format, the first three RX_MSG_WDATA words of a sequence have different contents.
For CAN XL frames (XLFF):

Table 110. RX_MSG_WDATA (CAN XL)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R0	FDF=0	XLF=1	XTD=0	Priority ID[28:18]											RRS	SEC	VCID[7:0]								SDT[7:0]							
R1	0x00				0x0	DLC-XL[10:0]											SN[3:0]				RX_IRQ	FAB	ALERT	FM	0x0	FIDX[6:0]						
R2	AF[31:16]																AF[15:0]]															

For CAN FD frames (FBDF, FEDF), the first two words have the following content:

Table 111. RX_MSG_WDATA (CAN FD)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R0	FDF=1	XLF=0	XTD	Base ID[28:18]											Extended ID[17:0]																	
R1	0x00				0x00		BRS	0x0				ESI	DLC[3:0]		SN[3:0]					RX_IRQ	FAB	ALERT	FM	0x0	FIDX[6:0]						

For CAN CC frames (CBDF, CEDF, CBRF, CBRF), the two words have the following content:

Table 112. RX_MSG_WDATA (CAN CC)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R0	FDF=0	XLF=0	XTD	Base ID[28:18]											Extended ID[17:0]																	
R1	0x00				0x0	RTR	0x0x						DLC[3:0]		SN[3:0]						RX_IRQ	FAB	ALERT	FM	0x0	FIDX[6:0]						

In the following chapters, RX_MSG sequences will be explained via chronograms. The next figure gives an introduction how information is illustrated for the Write Data Channel.

1.7.3.3 RX_MSG_WUSER

The vector RX_MSG_WUSER provides status information related to the vector RX_MSG_WDATA using the following codes:

Table 113. RX_MSG_WUSER Codes

Code Value	Code Name	Description
0	SOS	Code for the first transaction of a sequence (Start of Sequence). RX_MSG_WDATA contains M0 of the CAN Frame.
1	OK	Code for a transaction of an ongoing sequence. RX_MSG_WDATA contains data M(s) of CAN Frame.
2	TS0	Code for the second last transaction of a sequence for a received valid CAN message. RX_MSG_WDATA contains timestamp word 0.
3	TS1	Code for the last transaction of a sequence for a received valid CAN message. RX_MSG_WDATA contains timestamp word 1.
4	ABORT	Code for the last transaction of a sequence for a received invalid CAN message. RX_MSG_WDATA is invalid.
5	DO	Code for the last transaction of a sequence indicating a Data Overflow occurred at RX_MSG, i.e. previous transaction not acknowledged in time by MH via the signal RX_MSG_WREADY. RX_MSG_WDATA is invalid.
6	RESERVED	Do not use, will be interpreted like ABORT until defined otherwise.
7	RESERVED	Do not use, will be interpreted like ABORT until defined otherwise.

1.7.3.3.1 Codes SOS, OK, TS0 and TS1

The following chronogram shows the RX_MSG sequence for the case of a successful reception of a CAN XL frame (XLFF) with 11 byte payload.

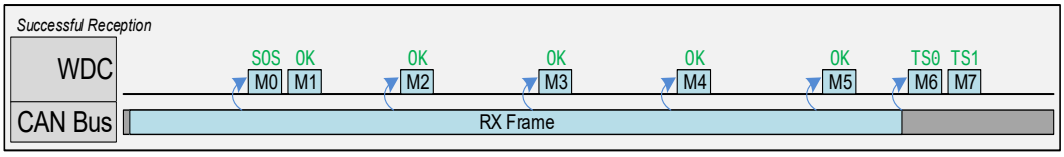


Figure 48. RX_MSG Sequence of a Successful Reception

The PRT starts a sequence at the Message Bus RX_MSG, by applying word M0 with code **SOS**, and M1 with code **OK**, when related information was de-serialized from the CAN Bus.

The PRT continues with transaction M2 which is the AF (Acceptance Field) for CAN XL frames (XLFF). The PRT continues with transactions M3 and M4, each containing 4 byte payload of the CAN Frame.

In this example, M5 contains the last three bytes of the 11 byte payload. Then PRT applies M6 and M7, containing the 64 bit timestamp sliced into two 32 bit transactions. Note that these last two transactions come with the codes **TS0** and **TS1**.

1.7.3.3.2 Code ABORT

The following chronogram shows the RX_MSG sequence for the case of an unsuccessful reception of a CAN frame.

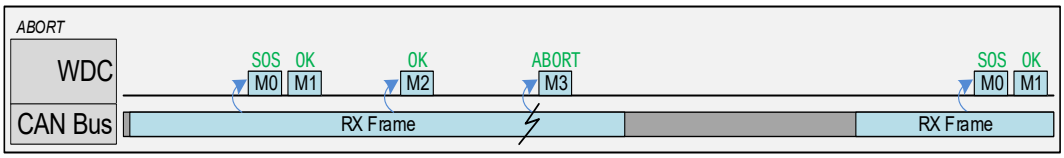


Figure 49. RX_MSG Sequence with Code ABORT

1.7.3.3.3 Code DO (Data Overflow)

The following chronogram shows the RX_MSG sequence for the case of a Data Overflow on the Message Bus RX_MSG.

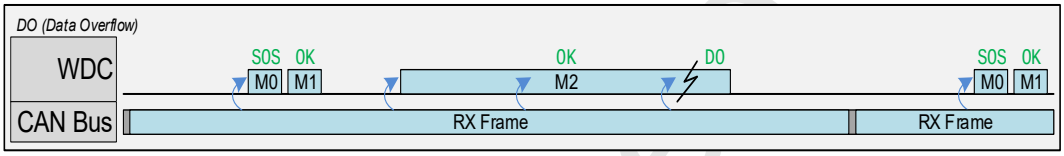


Figure 50. RX_MSG Sequence with Code DO (Data Overflow)

In this example, the M2 is not acknowledged in time, meaning the MH does not assert RX_MSG_WREADY. The PRT ends such a sequence by changing the RX_MSG_WUSER code from **OK** to **DO**. The data of such a sequence can be inconsistent and incomplete and thus must be discarded by the MH.

When the last two transactions of an RX_MSG sequence that come with the codes **TS0** and **TS1** are not acknowledged in time, meaning before the next frame starts on the CAN bus, the PRT ends such a sequence by changing the RX_MSG_WUSER code from **TS0** or **TS1** respectively to **DO**.

The MH needs to acknowledge the transaction that comes with the code **DO** before the PRT can start a new RX_MSG sequence. If a DO occurs due to a pending LMEM write access for a received frame word other than TS0/TS1, the MH aborts the pending LMEM write operation by driving LMEM_CS low.

1.7.3.4 Shortest and longest RX_MSG sequence

The RX_MSG sequence becomes shortest, when receiving a CAN CC or CAN FD Frame with no data bytes (Data Length Code = 0). Such a sequence consists of four transactions.

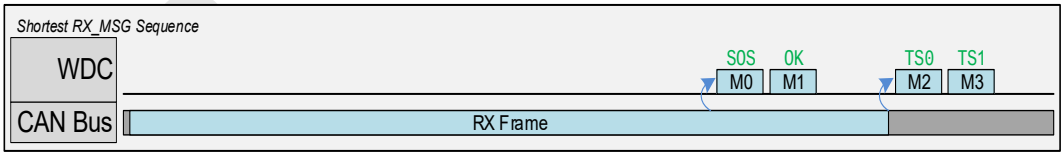


Figure 51. Shortest RX_MSG Sequence

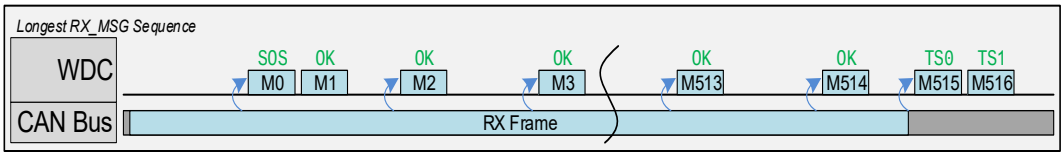


Figure 52. Longest RX_MSG Sequence

1.7.3.5 RX_MSG timing

This chapter shows the timing requirements of the PRT to exchange data via RX_MSG in time with the MH. All mentioned time budgets are the lowest possible time budgets. These times are determined by serialisation of the RX Element from the CAN Bus under worst case scenarios, i.e. where the dedicated time budget becomes smallest.

For better presentation, the scale between various transactions was relinquished.

Based on the bit rates and clock tolerances assumed in the introduction (see 1), and considering the possible shortening of the Intermission by an early SOF, the minimum CAN frame lengths for the calculation of worst case repetition rates are:

CAN CC remote frame: 46.00 µs
CAN CC data frame without payload: 47.00 µs
CAN FD frame without payload: 34.60 µs (28 arb.bits+33 data bits)
CAN XL frame with minimum payload: 41.93 µs (33 arb.bits+ 134 data bits).
The worst case frame repetition rate for a receiver is therefore a sequence of CAN FD Frames without data bytes coming from a transmitter with a clock of (1 +df) fnom while the receiver operates with a clock of (1 -df) fnom, resulting in a (perceived) repetition rate of one frame every 34.6µs × (1 – 2df) = 34.6 µs × (0.996) = 33.4616 µs

1.7.3.5.1 Timing t_M0-M0

The shortest interval for starting a new sequence at RX_MSG will occur when receiving consecutive CAN FD Frames with no data bytes and is **t_M0-M0** = 33.4616 µs.

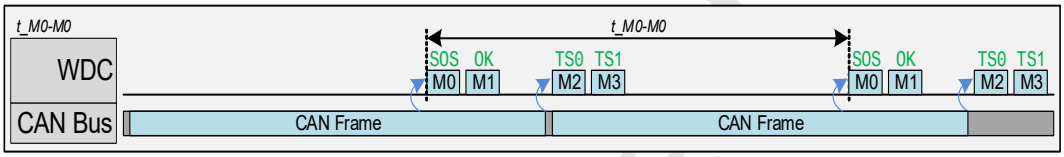


Figure 53. RX_MSG Sequence Timings t_M0-M0

The MH has to manage the storage of a RX_MSG transaction within that time.

1.7.3.5.2 Timing t_M-M and t_MS-M0

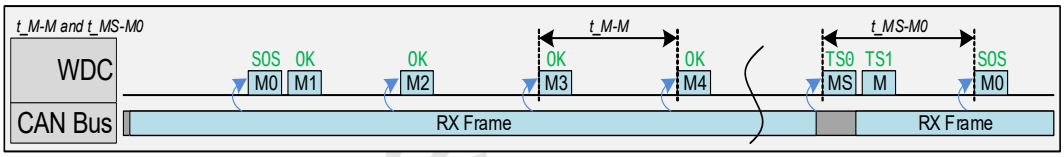


Figure 54. RX_MSG Sequence Timings t_M-M and t_MS-M0

When PRT signalized new data by asserting RX_MSG_WVALID, the MH has to acknowledge it by asserting RX_MSG_WREADY within **t_M-M = 2.13 µs**.

The time is defined by de-serializing 32 bit with 15 Mbit/s on the CAN Bus.
When a reception on the CAN Bus has ended, the last transactions will be issued by the PRT. The next sequence must be able to start within **t_MS-M0 = 17 µs**. This is the time budget for the MH to become ready for a new RX_MSG sequence.

The time is defined by CAN Bus and consist of two intermission bit times, SOF and the Arbitration Field with highest possible bit-rate = 1 Mbit/s.

1.7.4 Signal ENABLE

The PRT provides the signal ENABLE, which indicates if the Messages Busses have to be served (PRT is online), or not (PRT is offline).

If enable = 0, the Message Busses must be shut down on both sides (PRT and MH) in the following way:
• Ongoing transfers at a channel will be finished by the sink applying READY. Data can be discarded.
• A new transfer at a channel will not be started by the source.
• Outstanding transactions (TX_MSG only) will be discarded, i.e. source does not wait for responses and sink does not respond anymore.
• Ongoing sequences will be discarded.

If ENABLE changes from 0 to 1, the Message Busses start as after hardware reset, i.e. without memorization on previous sequences, transactions or transfers.

The signal ENABLE can be optionally used to control other functions of the MH, e.g. the write protection of registers which must not be configured during runtime.

1.7.5 Signal PRT_STOP_IMMD_REQ

The MH provides the pulse signal PRT_STOP_IMMD_REQ to instruct PRT to perform an immediate stop. The PRT's reaction to this signal is the same as writing a 1 to CTRL.STOP and CTRL.IMMD registers at the same time in PRT.
PRT must switch its activity to its inactive state immediately, to stop all CAN operation, and to set its CAN_TX output to 1 and to clear ENABLE.When the current activity was Transmitter, the PRT aborts that transmission. When the current activity was Receiver, it aborts that reception. Both interfaces with the MH are reset, outstanding transactions are discontinued. If this happens while an RX_MSG sequence was ongoing, this sequence is discontinued with the ABORT code. The PRT does not start another reception or transmission until it is started again.

The conditions where this input is generated by MH is mentioned in Error and Exception Handling Section.

1.7.6 Miscellaneous

1.7.6.1 Channel Handshake

Each channel of a Message Bus has its own two-way flow control, thus, both the source and sink can control the transfer rate.

The source signalizes new data to be transferred by asserting **VALID** (independent of **READY**) and keeps its information stable until the transfer occurs.

The Sink signalizes that it is able to accept data by asserting **READY** (independent of **VALID**).
The Transfer occurs (the data is accepted by the sink) when both **READY** and **VALID** are asserted at the same time.

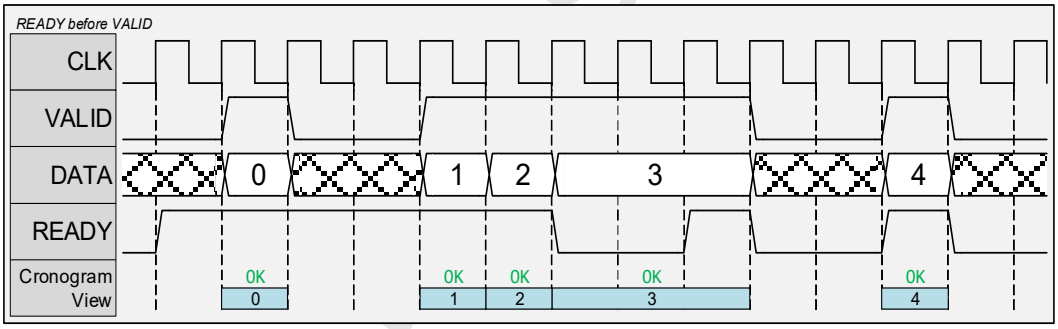


Figure 55. Channel Handshake Type: READY before VALID

At transfer 4 the sink becomes 'ready to accept data', and independently of this, the source signalizes 'new data' in the same clock cycle by chance. In this case, the transfer is done within one clock cycle.

1.7.6.1.1 Assumptions for Channel Handshake

- The sink may assert READY before VALID is asserted.
- The source may assert VALID before READY is asserted.
- The transfer occurs when VALID and READY are asserted together.
- VALID may be asserted for a next transfer in the next clock cycle.
- Inside source and sink interfaces, there must be no combinatorial paths between input and output signals.

1.7.6.1.2 Names of standardized bits and fields in CAN Elements

Table 114. Fields in CAN Elements (page 1 of 2)

Name	Description
Base ID	11 bit identifier for Classical CAN frames (CBDF, CEDF, CBRF, CERF) and CAN FD. frames (FBDF, FEDF)
Extended ID	29 bit identifier for Classical CAN frames (CBDF, CEDF, CBRF, CERF) and CAN FD. frames (FBDF, FEDF)
Priority ID	11 bit identifier for frames in the CAN XL Frame Format (XLFF)
XTD	Extended Identifier 0=11 bit ID, 1=29 bit ID for Classical CAN frames (CBDF, CEDF, CBRF, CERF) and CAN FD frames (FBDF, FEDF)

Table 114. Fields in CAN Elements (page 2 of 2)

Name	Description
FDF	FD Format
XLF	XL Format
BRS	Bit Rate Switch
DLC	Data Length Code
DLC-XL	Data Length Code with CAN XL encoding
SDT	SDU Type
VCID	Virtual CAN Network ID
RTR	Remote Transmission Request
RRS	Remote Request Subsitution
ESI	Error State Indicator
TS	Timestamp
FIR	Fault Injection Request

1.8 IRC - Interrupt Controller

1.8.1 Overview

The XS_CAN IP is equipped with a central interrupt controller (IRC). It captures all events of the MH and PRT and can be configured for each event individually to interrupt the HOST CPU. The events are organized in 4 categories, i.e., TX Functional, RX Functional, Error and safety. Functional Events can trigger the IRC output TX_FUNC_INT and RX_FUNC_INT. Error Events can trigger the IRC outputs ERR_INT and safety events trigger SAFETY_INT.

All 4 interrupt lines are level triggered. Logical 1 on the line shows an active interrupt. The output interrupt lines are triggered after two host clock cycles after the corresponding source gets activated at IRC raw registers.

1.8.2 IRC MAP

1.8.2.1 Overview

Table 115. Address Blocks list

Address Block	Offset	Range	Description
IRC_EVENT	0x0000	0x0010	Usage: register This address block provides registers to capture events of the MH and the PRT. The events are organized in three categories (one register for each category): Functional relevant, functional error relevant and safety relevant.
IRC_CONTROL	0x0010	0x0020	Usage: register This address block provides the registers for controlling the IRC.
IRC_CONFIGURATION	0x0030	0x0010	Usage: register Hardware configuration of the IRC
IRC_AUXILIARY	0x0040	0x00CF	Usage: register Auxiliary registers

Table 116. Registers overview (page 1 of 2)

Register name	Offset	Mode	Description
IRC_EVENT			
TX_FUNC_RAW	0x0000	R	Register size: 32 TX Functional raw event status register. This register provides information about the occurrence of TX functional relevant events inside the MH and the PRT. A flag is set when the related event is detected, independent of TX_FUNC_ENA. The flags remain set until the Host CPU clears them by writing a 1 to the corresponding bit position at register TX_FUNC_CLR.
RX_FUNC_RAW	0x0004	R	Register size: 32 RX Functional raw event status register. This register provides information about the occurrence of functional relevant events inside the MH and the PRT. A flag is set when the related event is detected, independent of RX_FUNC_ENA. The flags remain set until the Host CPU clears them by writing a 1 to the corresponding bit position at register RX_FUNC_CLR.
ERR_STS_RAW	0x0008	R	Register size: 32 Error raw event status register. This register provides information about the occurrence of functional error relevant events inside the MH and the PRT. A flag is set when the related event is detected, independent of ERR_STS_ENA. The flags remain set until the Host CPU clears them by writing a 1 to the corresponding bit position at register ERR_STS_CLR.
SAFETY_RAW	0x000C	R	Register size: 32 Safety raw event status register. This register provides information about the occurrence of safety relevant events inside the MH and the PRT. A flag is set when the related event is detected, independent of SAFETY_ENA. The flags remain set until the Host CPU clears them by writing a 1 to the corresponding bit position at register SAFETY_CLR.
IRC_CONTROL			
TX_FUNC_CLR	0x0010	W	Register size: 32 TX Functional raw event clear register. Writing a 1 to a certain bit position clears the corresponding bit of register TX_FUNC_RAW. Writing a 0 has no effect.
RX_FUNC_CLR	0x0014	W	Register size: 32 Functional raw event clear register. Writing a 1 to a certain bit position clears the corresponding bit of register RX_FUNC_RAW. Writing a 0 has no effect.
ERR_STS_CLR	0x0018	W	Register size: 32 Error raw event clear register. Writing a 1 to a certain bit position clears the corresponding bit of register ERR_STS_RAW. Writing a 0 has no effect.
SAFETY_CLR	0x001C	W	Register size: 32 Safety raw event clear register. Writing a 1 to a certain bit position clears the corresponding bit of register SAFETY_RAW. Writing a 0 has no effect.
TX_FUNC_ENA	0x0020	RW	Register size: 32 TX Functional raw event enable register. Any bit in the TX_FUNC_ENA register enables the corresponding bit in the TX_FUNC_RAW to trigger the interrupt line TX_FUNC_INT. The interrupt line gets active high, when at least one RAW/ENA pair is 1, e.g. TX_FUNC_RAW.MH_TX_FQ_IRQ = TX_FUNC_ENA.MH_TX_FQ_IRQ = 1
RX_FUNC_ENA	0x0024	RW	Register size: 32 RX Functional raw event enable register. Any bit in the RX_FUNC_ENA register enables the corresponding bit in the RX_FUNC_RAW to trigger the interrupt line RX_FUNC_INT. The interrupt line gets active high, when at least one RAW/ENA pair is 1, e.g. RX_FUNC_RAW.MH_RX_FQ0_IRQ = RX_FUNC_ENA.MH_RX_FQ0_IRQ = 1
ERR_STS_ENA	0x0028	RW	Register size: 32 Error raw event enable register. Any bit in the ERR_STS_ENA register enables the corresponding bit in the ERR_STS_RAW to trigger the interrupt line ERR_INT. The interrupt line gets active high, when at least one RAW/ENA pair is 1, e.g. ERR_STS_RAW.MH_TX_FQ_IRQ = ERR_STS_ENA.MH_TX_FQ_IRQ = 1

Table 116. Registers overview (page 2 of 2)

Register name	Offset	Mode	Description
SAFETY_ENA	0x002C	RW	Register size: 32 Safety raw event enable register. Any bit in the SAFETY_ENA register enables the corresponding bit in the SAFETY_RAW to trigger the interrupt line SAFETY_INT. The interrupt line gets active high, when at least one RAW/ENA pair is 1, e.g. SAFETY_RAW.MH_TX_FQ_IRQ = SAFETY_ENA.MH_TX_FQ_IRQ = 1
IRC_CONFIGURATION			
CAPTURING_MODE	0x0030	R	Register size: 32 IRC configuration register. This register shows the hardware configuration of the IRC concerning the capturing mode of the event inputs. The IP internal events signals coming from the MH and the PRT require an 'edge sensitive' capturing. That is why the value of this register is 0x7 and cannot be changed.
IRC_AUXILIARY			
HDP	0x0040	RW	Register size: 32 Hardware Debug Port control register

1.8.2.2 IRC_EVENT Address Block

Table 117. TX_FUNC_RAW register

TX Functional raw event status register. This register provides information about the occurrence of TX functional relevant events inside the MH and the PRT. A flag is set when the related event is detected, independent of TX_FUNC_ENA. The flags remain set until the Host CPU clears them by writing a 1 to the corresponding bit position at register TX_FUNC_CLR.

Bit	Field	Mode	Initial Value	Description
16	PRT_TX_EVT	R	0x0	PRT interrupt that indicates PRT transmitted a valid CAN message
4	MH_CTB_TX_BC_REQ	R	0x0	MH interrupt that indicates a block copy transfer pending in TX direction
3	MH_TEFQ_DEQ	R	0x0	MH interrupt that indicates a message is dequeued from TEFQ
2	MH_TEFQ_ENQ	R	0x0	MH interrupt that indicates a message is enqueued into TEFQ
1	MH_TXPQ_ENQ	R	0x0	MH interrupt that indicates a message is successfully enqueued into TXPQ
0	MH_TXFQ_ENQ	R	0x0	MH interrupt that indicates a message is successfully enqueued into TXFQ

Table 118. RX_FUNC_RAW register

RX Functional raw event status register. This register provides information about the occurrence of functional relevant events inside the MH and the PRT. A flag is set when the related event is detected, independent of RX_FUNC_ENA. The flags remain set until the Host CPU clears them by writing a 1 to the corresponding bit position at register RX_FUNC_CLR.

Bit	Field	Mode	Initial Value	Description
16	PRT_RX_EVT	R	0x0	PRT interrupt that indicates PRT received a valid CAN message
6	MH_RX_MSG_ALERT	R	0x0	MH interrupt that indicates message alert detected. This is generated only if ALERT bit in FEC is set to 1.
5	MH_RX_FILTER_MATCH	R	0x0	MH interrupt that indicates filter match detected. This is generated only if IRQ bit in FEC is set to 1.
4	MH_CTB_RX_BC_REQ	R	0x0	MH interrupt that indicates a block copy transfer pending in RX direction
3	MH_RXFQ1_DEQ	R	0x0	MH interrupt that indicates a message is dequeued from RXFQ1
2	MH_RXFQ0_DEQ	R	0x0	MH interrupt that indicates a message is dequeued from RXFQ0
1	MH_RXFQ1_ENQ	R	0x0	MH interrupt that indicates a message is enqueued into RXFQ1
0	MH_RXFQ0_ENQ	R	0x0	MH interrupt that indicates a message is enqueued into RXFQ0

Table 119. ERR_STS_RAW register

Error raw event status register. This register provides information about the occurrence of functional error relevant events inside the MH and the PRT. A flag is set when the related event is detected, independent of ERR_STS_ENA. The flags remain set until the Host CPU clears them by writing a 1 to the corresponding bit position at register ERR_STS_CLR.

Bit	Field	Mode	Initial Value	Description
22	PRT_BUS_ON	R	0x0	PRT interrupt that indicates starting of CAN module, which is set immediately after CTRL.STRT command is written (this is also the case if CAN module was in Bus_Off state (STAT.BO=1)).
21	PRT_E_ACTIVE	R	0x0	PRT interrupt that indicates PRT switched from Error-Passive to Error-Active state
20	PRT_BUS_OFF	R	0x0	PRT interrupt that indicates the stop of CAN module, either after CTRL.STOP command is written or stop due to entering Bus_Off state (STAT.BO=1).
19	PRT_E_PASSIVE	R	0x0	PRT switched from Error-Active to Error-Passive state.
18	PRT_BUS_ERR	R	0x0	PRT detected error on the CAN Bus.
17	PRT_IFF	R	0x0	PRT detected invalid Frame Format at TX_MSG.
16	PRT_STOP	R	0x0	Interrupt to indicate PRT is stopped i.e when PRT ENABLE goes from high to low
8	HOST_ARA	R	0x0	MH interrupt that indicates host access to reserved addresses (includes VB reserved address and register map reserved addresses)
7	MH_VB_NO_SA	R	0x0	MH interrupt to indicate MH has received another address instead of VB start address.See MH_STS0 bits 21 to 16 to know the source flags.
6	MH_QUEUE_ACCESS_VIOLATION	R	0x0	MH interrupt to indicate there has been a violation in accessing the queues. See MH_STS0 bits 30 to 22 to know the source flags.
5	MH_LMEM_CERR	R	0x0	MH interrupt that indicates a correctable error has been detected at the LMEM interface
4	MH_RXFQ1_DROP	R	0x0	MH interrupt that indicates a new message to be stored into RXFQ1 is dropped.
3	MH_RXFQ0_DROP	R	0x0	MH interrupt that indicates a new message to be stored into RXFQ0 is dropped.
2	MH_TEFQ_DROP	R	0x0	MH interrupt that indicates a new TEQE is dropped.
1	MH_TXPQ_DEQ_NO_TX	R	0x0	MH interrupt that indicates a message has been dequeued from TXPQ but is not transmitted.
0	MH_TXFQ_DEQ_NO_TX	R	0x0	MH interrupt that indicates a message has been dequeued from TXFQ but is not transmitted.

Table 120. SAFETY_RAW register (page 1 of 2)

Safety raw event status register. This register provides information about the occurrence of safety relevant events inside the MH and the PRT. A flag is set when the related event is detected, independent of SAFETY_ENA. The flags remain set until the Host CPU clears them by writing a 1 to the corresponding bit position at register SAFETY_CLR.

Bit	Field	Mode	Initial Value	Description
19	PRT_RX_DO	R	0x0	PRT detected overflow condition at RX_MSG sequence.
18	PRT_TX_DU	R	0x0	PRT detected underrun condition at TX_MSG sequence.
17	PRT_USOS	R	0x0	PRT detected unexpected Start of Sequence during TX_MSG sequence.
16	PRT_TX_ABORTED	R	0x0	PRT detected stop of TX_MSG sequence by TX_MSG_WUSER code ABORT.
9	MH_TX_ABORT	R	0x0	MH interrupt to indicate MH has to abort an ongoing transmission of a message to PRT
8	MH_RX_ABORT	R	0x0	MH interrupt to indicate MH has to abort an ongoing reception of a message from PRT.
7	MH_RX_FILTER_ERR	R	0x0	MH interrupt to indicate error in filtering process like filtering not completed on time or target RXFQ not enabled.

Table 120. SAFETY_RAW register (page 2 of 2)

Safety raw event status register. This register provides information about the occurrence of safety relevant events inside the MH and the PRT. A flag is set when the related event is detected, independent of SAFETY_ENA. The flags remain set until the Host CPU clears them by writing a 1 to the corresponding bit position at register SAFETY_CLR.

Bit	Field	Mode	Initial Value	Description
6	MH_SAFETY_MISC	R	0x0	Interrupt that indicates different safety issues like datapath parity error in MH, datapath sequence error in MH and Timestamp parity error in PRT. See MH_STS1 bits 4 to 0 for the source flags for this interrupt
5	MH_VB_ACCESS_VIOLATION	R	0x0	MH interrupt that indicates there is a violation in VB access(read to write only address and vice versa), See description MH_STS1 register bits 11 to 5 for the source flags for this interrupt
4	MH_LMEM_PROT_ERR	R	0x0	MH interrupt that indicates host access to an LMEM address outside the range of LMEM START ADDRESS and LMEM END ADDRESS
3	MH_LMEM_TO_ERR	R	0x0	MH interrupt that indicates a timeout at LMEM interface (includes read and write directions)
2	MH_LMEM_UERR	R	0x0	MH interrupt that indicates an uncorrectable error is detected at the LMEM interface
1	MH_CTB_BC_ERR	R	0x0	MH interrupt to indicate ongoing block copy is cancelled due to error response received at CTM_RESP input
0	MH_CTB_BC_TO	R	0x0	MH interrupt to indicate time out for block copy request.i.e block copy not finished in time before data underrun or overflow at CTB.

1.8.2.3 IRC_CONTROL Address Block

Table 121. TX_FUNC_CLR register

TX Functional raw event clear register. Writing a 1 to a certain bit position clears the corresponding bit of register TX_FUNC_RAW. Writing a 0 has no effect.

Bit	Field	Mode	Initial Value	Description
16	PRT_TX_EVT	W	0x0	Writing 1 clears the bit TX_FUNC_RAW.PRT_TX_EVT
4	MH_CTB_TX_BC_REQ	W	0x0	Writing 1 clears the bit TX_FUNC_RAW.MH_CTB_TX_BC_REQ
3	MH_TEFQ_DEQ	W	0x0	Writing 1 clears the bit TX_FUNC_RAW.MH_TEFQ_DEQ
2	MH_TEFQ_ENQ	W	0x0	Writing 1 clears the bit TX_FUNC_RAW.MH_TEFQ_ENQ
1	MH_TXPQ_ENQ	W	0x0	Writing 1 clears the bit TX_FUNC_RAW.MH_TXPQ_ENQ
0	MH_TXFQ_ENQ	W	0x0	Writing 1 clears the bit TX_FUNC_RAW.MH_TXFQ_ENQ

Table 122. RX_FUNC_CLR register

Functional raw event clear register. Writing a 1 to a certain bit position clears the corresponding bit of register RX_FUNC_RAW. Writing a 0 has no effect.

Bit	Field	Mode	Initial Value	Description
16	PRT_RX_EVT	W	0x0	Writing 1 clears the bit RX_FUNC_RAW.PRT_RX_EVT
6	MH_RX_MSG_ALERT	W	0x0	Writing 1 clears the bit RX_FUNC_RAW.MH_RX_MSG_ALERT
5	MH_RX_FILTER_MATCH	W	0x0	Writing 1 clears the bit RX_FUNC_RAW.MH_RX_FILTER_MATCH
4	MH_CTB_RX_BC_REQ	W	0x0	Writing 1 clears the bit RX_FUNC_RAW.MH_CTB_RX_BC_REQ
3	MH_RXFQ1_DEQ	W	0x0	Writing 1 clears the bit RX_FUNC_RAW.MH_RXFQ1_DEQ
2	MH_RXFQ0_DEQ	W	0x0	Writing 1 clears the bit RX_FUNC_RAW.MH_RXFQ0_DEQ
1	MH_RXFQ1_ENQ	W	0x0	Writing 1 clears the bit RX_FUNC_RAW.MH_RXFQ1_ENQ
0	MH_RXFQ0_ENQ	W	0x0	Writing 1 clears the bit RX_FUNC_RAW.MH_RXFQ0_ENQ

Table 123. ERR_STS_CLR register

Error raw event clear register. Writing a 1 to a certain bit position clears the corresponding bit of register ERR_STS_RAW. Writing a 0 has no effect.

Bit	Field	Mode	Initial Value	Description
22	PRT_BUS_ON	W	0x0	Writing 1 clears the bit ERR_STS_RAW.PRT_BUS_ON
21	PRT_E_ACTIVE	W	0x0	Writing 1 clears the bit ERR_STS_RAW.PRT_E_ACTIVE
20	PRT_BUS_OFF	W	0x0	Writing 1 clears the bit ERR_STS_RAW.PRT_BUS_OFF
19	PRT_E_PASSIVE	W	0x0	Writing 1 clears the bit ERR_STS_RAW.PRT_E_PASSIVE
18	PRT_BUS_ERR	W	0x0	Writing 1 clears the bit ERR_STS_RAW.PRT_BUS_ERR
17	PRT_IFF	W	0x0	Writing 1 clears the bit ERR_STS_RAW.PRT_IFF
16	PRT_STOP	W	0x0	Writing 1 clears bit ERR_STS_RAW.PRT_STOP
8	HOST_ARA	W	0x0	Writing 1 clears bit ERR_STS_RAW.HOST_ARA
7	MH_VB_NO_SA	W	0x0	Writing 1 clears the bit ERR_STS_RAW.MH_VB_NO_SA
6	MH_QUEUE_ACCESS_VIOLATION	W	0x0	Writing 1 clears the bit ERR_STS_RAW.MH_QUEUE_ACCESS_VIOLATION
5	MH_LMEM_CERR	W	0x0	Writing 1 clears the bit ERR_STS_RAW.MH_LMEM_CERR
4	MH_RXFQ1_DROP	W	0x0	Writing 1 clears the bit ERR_STS_RAW.MH_RXFQ1_DROP
3	MH_RXFQ0_DROP	W	0x0	Writing 1 clears the bit ERR_STS_RAW.MH_RXFQ0_DROP
2	MH_TEFQ_DROP	W	0x0	Writing 1 clears the bit ERR_STS_RAW.MH_TEFQ_DROP
1	MH_TXPQ_DEQ_NO_TX	W	0x0	Writing 1 clears the bit ERR_STS_RAW.MH_TXPQ_DEQ_NO_TX
0	MH_TXFQ_DEQ_NO_TX	W	0x0	Writing 1 clears the bit ERR_STS_RAW.MH_TXFQ_DEQ_NO_TX

Table 124. SAFETY_CLR register

Safety raw event clear register. Writing a 1 to a certain bit position clears the corresponding bit of register SAFETY_RAW. Writing a 0 has no effect.

Bit	Field	Mode	Initial Value	Description
19	PRT_RX_DO	W	0x0	Writing 1 clears bit SAFETY_RAW.PRT_RX_DO
18	PRT_TX_DU	W	0x0	Writing 1 clears bit SAFETY_RAW.PRT_TX_DU
17	PRT_USOS	W	0x0	Writing 1 clears bit SAFETY_RAW.PRT_USOS
16	PRT_TX_ABORTED	W	0x0	Writing 1 clears bit SAFETY_RAW.PRT_TX_ABORTED
9	MH_TX_ABORT	W	0x0	Writing 1 clears the bit SAFETY_RAW.MH_TX_ABORT
8	MH_RX_ABORT	W	0x0	Writing 1 clears the bit SAFETY_RAW.MH_RX_ABORT
7	MH_RX_FILTER_ERR	W	0x0	Writing 1 clears bit SAFETY_RAW.MH_RX_FILTER_ERR
6	MH_SAFETY_MISC	W	0x0	Writing 1 clears bit SAFETY_RAW.MH_SAFETY_MISC
5	MH_VB_ACCESS_VIOLATION	W	0x0	Writing 1 clears bit SAFETY_RAW.MH_VB_ACCESS_VIOLATION
4	MH_LMEM_PROT_ERR	W	0x0	Writing 1 clears bit SAFETY_RAW.MH_LMEM_PROT_ERR
3	MH_LMEM_TO_ERR	W	0x0	Writing 1 clears bit SAFETY_RAW.MH_LMEM_TO_ERR
2	MH_LMEM_UERR	W	0x0	Writing 1 clears bit SAFETY_RAW.MH_LMEM_UERR
1	MH_CTB_BC_ERR	W	0x0	Writing 1 clears bit SAFETY_RAW.MH_CTB_BC_ERR
0	MH_CTB_BC_TO	W	0x0	Writing 1 clears bit SAFETY_RAW.MH_CTB_BC_TO

Table 125. TX_FUNC_ENA register

TX Functional raw event enable register. Any bit in the TX_FUNC_ENA register enables the corresponding bit in the TX_FUNC_RAW to trigger the interrupt line TX_FUNC_INT. The interrupt line gets active high, when at least one RAW/ENA pair is 1, e.g. TX_FUNC_RAW.MH_TX_FQ_IRQ = TX_FUNC_ENA.MH_TX_FQ_IRQ = 1

Bit	Field	Mode	Initial Value	Description
16	PRT_TX_EVT	R/W	0x0	Enables TX_FUNC_RAW.PRT_TX_EVT to activate FUNC_TX_INT
4	MH_CTB_TX_BC_REQ	R/W	0x0	Enables TX_FUNC_RAW.MH_CTB_TX_BC_REQ to activate FUNC_TX_INT
3	MH_TEFQ_DEQ	R/W	0x0	Enables TX_FUNC_RAW.MH_TEFQ_DEQ to activate FUNC_TX_INT
2	MH_TEFQ_ENQ	R/W	0x0	Enables TX_FUNC_RAW.MH_TEFQ_ENQ to activate FUNC_TX_INT
1	MH_TXPQ_ENQ	R/W	0x0	Enables TX_FUNC_RAW.MH_TXPQ_ENQ to activate FUNC_TX_INT
0	MH_TXFQ_ENQ	R/W	0x0	Enables TX_FUNC_RAW.MH_TXFQ_ENQ to activate FUNC_TX_INT

Table 126. RX_FUNC_ENA register

RX Functional raw event enable register. Any bit in the RX_FUNC_ENA register enables the corresponding bit in the RX_FUNC_RAW to trigger the interrupt line RX_FUNC_INT. The interrupt line gets active high, when at least one RAW/ENA pair is 1, e.g. RX_FUNC_RAW.MH_RX_FQ0_IRQ = RX_FUNC_ENA.MH_RX_FQ0_IRQ = 1

Bit	Field	Mode	Initial Value	Description
16	PRT_RX_EVT	R/W	0x0	Enables RX_FUNC_RAW.PRT_RX_EVT to activate FUNC_RX_INT
6	MH_RX_MSG_ALERT	R/W	0x0	Enables RX_FUNC_RAW.MH_RX_MSG_ALERT to activate FUNC_RX_INT
5	MH_RX_FILTER_MATCH	R/W	0x0	Enables RX_FUNC_RAW.MH_RX_FILTER_MATCH to activate FUNC_RX_INT
4	MH_CTB_RX_BC_REQ	R/W	0x0	Enables RX_FUNC_RAW.MH_CTB_RX_BC_REQ to activate FUNC_RX_INT
3	MH_RXFQ1_DEQ	R/W	0x0	Enables RX_FUNC_RAW.MH_RXFQ1_DEQ to activate FUNC_RX_INT
2	MH_RXFQ0_DEQ	R/W	0x0	Enables RX_FUNC_RAW.MH_RXFQ0_DEQ to activate FUNC_RX_INT
1	MH_RXFQ1_ENQ	R/W	0x0	Enables RX_FUNC_RAW.MH_RXFQ1_ENQ to activate FUNC_RX_INT
0	MH_RXFQ0_ENQ	R/W	0x0	Enables RX_FUNC_RAW.MH_RXFQ0_ENQ to activate FUNC_RX_INT

Table 127. ERR_STS_ENA register (page 1 of 2)

Error raw event enable register. Any bit in the ERR_STS_ENA register enables the corresponding bit in the ERR_STS_RAW to trigger the interrupt line ERR_INT. The interrupt line gets active high, when at least one RAW/ENA pair is 1, e.g. ERR_STS_RAW.MH_TX_FQ_IRQ = ERR_STS_ENA.MH_TX_FQ_IRQ = 1

Bit	Field	Mode	Initial Value	Description
22	PRT_BUS_ON	R/W	0x0	Enables ERR_STS_RAW.PRT_BUS_ON to activate ERR_INT
21	PRT_E_ACTIVE	R/W	0x0	Enables ERR_STS_RAW.PRT_E_ACTIVE to activate ERR_INT
20	PRT_BUS_OFF	R/W	0x0	Enables ERR_STS_RAW.PRT_BUS_OFF to activate ERR_INT
19	PRT_E_PASSIVE	R/W	0x0	Enables ERR_STS_RAW.PRT_E_PASSIVE to activate ERR_INT
18	PRT_BUS_ERR	R/W	0x0	Enables ERR_STS_RAW.PRT_BUS_ERR to activate ERR_INT
17	PRT_IFF	R/W	0x0	Enables ERR_STS_RAW.PRT_IFF to activate ERR_INT
16	PRT_STOP	R/W	0x0	Enables ERR_STS_RAW.PRT_STOP to activate ERR_INT
8	HOST_ARA	R/W	0x0	Enables ERR_STS_RAW.HOST_ARA to activate ERR_INT
7	MH_VB_NO_SA	R/W	0x0	Enables ERR_STS_RAW.MH_VB_NO_SA to activate ERR_INT
6	MH_QUEUE_ACCESS_VIOLATION	R/W	0x0	Enables ERR_STS_RAW.MH_QUEUE_ACCESS_VIOLATION to activate ERR_INT
5	MH_LMEM_CERR	R/W	0x0	Enables ERR_STS_RAW.MH_LMEM_CERR to activate ERR_INT
4	MH_RXFQ1_DROP	R/W	0x0	Enables ERR_STS_RAW.MH_RXFQ1_DROP to activate ERR_INT
3	MH_RXFQ0_DROP	R/W	0x0	Enables ERR_STS_RAW.MH_RXFQ0_DROP to activate ERR_INT

Table 127. ERR_STS_ENA register (page 2 of 2)

Error raw event enable register. Any bit in the ERR_STS_ENA register enables the corresponding bit in the ERR_STS_RAW to trigger the interrupt line ERR_INT. The interrupt line gets active high, when at least one RAW/ENA pair is 1, e.g. ERR_STS_RAW.MH_TX_FQ_IRQ = ERR_STS_ENA.MH_TX_FQ_IRQ = 1

Bit	Field	Mode	Initial Value	Description
2	MH_TEFQ_DROP	R/W	0x0	Enables ERR_STS_RAW.MH_TEFQ_DROP to activate ERR_INT
1	MH_TXPQ_DEQ_NO_TX	R/W	0x0	Enables ERR_STS_RAW.MH_TXPQ_DEQ_NO_TX to activate ERR_INT
0	MH_TXFQ_DEQ_NO_TX	R/W	0x0	Enables ERR_STS_RAW.MH_TXFQ_DEQ_NO_TX to activate ERR_INT

Table 128. SAFETY_ENA register

Safety raw event enable register. Any bit in the SAFETY_ENA register enables the corresponding bit in the SAFETY_RAW to trigger the interrupt line SAFETY_INT. The interrupt line gets active high, when at least one RAW/ENA pair is 1, e.g. SAFETY_RAW.MH_TX_FQ_IRQ = SAFETY_ENA.MH_TX_FQ_IRQ = 1

Bit	Field	Mode	Initial Value	Description
19	PRT_RX_DO	R/W	0x0	Enables SAFETY_RAW.PRT_RX_DO to activate SAFETY_INT
18	PRT_TX_DU	R/W	0x0	Enables SAFETY_RAW.PRT_TX_DU to activate SAFETY_INT
17	PRT_USOS	R/W	0x0	Enables SAFETY_RAW.PRT_USOS to activate SAFETY_INT
16	PRT_TX_ABORTED	R/W	0x0	Enables SAFETY_RAW.PRT_TX_ABORTED to activate SAFETY_INT
9	MH_TX_ABORT	R/W	0x0	Enables SAFETY_RAW.MH_TX_ABORT to activate SAFETY_INT
8	MH_RX_ABORT	R/W	0x0	Enables SAFETY_RAW.MH_RX_ABORT to activate SAFETY_INT
7	MH_RX_FILTER_ERR	R/W	0x0	Enables SAFETY_RAW.MH_RX_FILTER_ERR to activate SAFETY_INT
6	MH_SAFETY_MISC	R/W	0x0	Enables SAFETY_RAW.MH_SAFETY_MISC to activate SAFETY_INT
5	MH_VB_ACCESS_VIOLATION	R/W	0x0	Enables SAFETY_RAW.MH_VB_ACCESS_VIOLATION to activate SAFETY_INT
4	MH_LMEM_PROT_ERR	R/W	0x0	Enables SAFETY_RAW.MH_LMEM_PROT_ERR to activate SAFETY_INT
3	MH_LMEM_TO_ERR	R/W	0x0	Enables SAFETY_RAW.MH_LMEM_TO_ERR to activate SAFETY_INT
2	MH_LMEM_UERR	R/W	0x0	Enables SAFETY_RAW.MH_LMEM_UERR to activate SAFETY_INT
1	MH_CTB_BC_ERR	R/W	0x0	Enables SAFETY_RAW.MH_CTB_BC_ERR to activate SAFETY_INT
0	MH_CTB_BC_TO	R/W	0x0	Enables SAFETY_RAW.MH_CTB_BC_TO to activate SAFETY_INT

1.8.2.4 IRC_CONFIGURATION Address Block

Table 129. CAPTURING_MODE register

IRC configuration register. This register shows the hardware configuration of the IRC concerning the capturing mode of the event inputs. The IP internal events signals coming from the MH and the PRT require an 'edge sensitive' capturing. That is why the value of this register is 0x7 and cannot be changed.

Bit	Field	Mode	Initial Value	Description
2	SAFETY	R	0x1	Capturing mode of SAFETY RAW register. 0 = Level sensitive 1 = Edge sensitive
1	ERR	R	0x1	Capturing mode of ERR RAW register. 0 = Level sensitive 1 = Edge sensitive
0	FUNC	R	0x1	Capturing mode of FUNC_RAW register. 0 = Level sensitive 1 = Edge sensitive

1.8.2.5 IRC_AUXILIARY Address Block

Table 130. HDP register

Hardware Debug Port control register

Bit	Field	Mode	Initial Value	Description
0	HDP_SEL	R/W	0x0	Select the driver of the Hardware Debug Port. See also chapter HDP. 0 = Message Handler 1 = Protocol Controller

1.8.3 Functional Description

The following table lists the categories and shows the related IRC registers together with the dedicated IRC outputs:

Table 131. IRC Events

Category	IRC Registers	IRC Output
Functional Event	TX_FUNC_RAW, RX_FUNC_RAW, TX_FUNC_CLR, RX_FUNC_CLR, TX_FUNC_ENA, RX_FUNC_ENA	TX_FUNC_INT, RX_FUNC_INT
Error Event	ERR_STS_RAW, ERR_STS_CLR, ERR_STS_ENA	ERR_INT
	SAFETY_RAW, SAFETY_CLR, SAFETY_ENA	SAFETY_INT

For each category, three registers are implemented: xxx_RAW, xxx_CLR, and xxx_ENA. To capture the events, the xxx_RAW registers are used. These registers provide information about the occurrence of events inside the MH and the PRT. A flag remain set until the HOST CPU clears them by writing a 1 to the corresponding bit position at register xxx_CLR.

The xxx_ENA registers control on bit level, whether a certain bit in the xxx_RAW register can activate the interrupt line xxx_INT. The interrupt line xxx_INT gets active high, when at least one RAW/ENA pair is 1, e.g., ERR_INT gets active high, when **ERR_STS_RAW.MH_RX_FILTER_ERR = ERR_STS_ENA.MH_RX_FILTER_ERR = 1.**
Interrupt generation from XXX_RAW register has higher priority over interrupt clearing. If the host writes to xxx_CLR at the same time when the corresponding xxx_RAW bit is getting updated, then xxx_RAW register bit remains set and will not be cleared.

In case of TX_FUNC_RAW events, MH_TXPQ_DEQ (TXPQ element dequeued successfully) and MH_TXFQ_DEQ (TXFQ element dequeued successfully) interrupt flags are not provided.This is indirectly understood from the combination of the two interrupts
->PRT_TX_EVT and MH_TXPQ_DEQ_NO_TX_IRQ for TXPQ
->PRT_TX_EVT and MH_TXFQ_DEQ_NO_TX_IRQ for TXFQ

Note that IRC works on HOST_CLK_AON clock.

Details provided by register definitions.
The following table shows the connection between the event outputs of the modules MH and PRT and the registers fields of the IRC.

1.8.3.1 IRC Flags and Source Ports

Table 132. IRC Flags and Source Ports (page 1 of 2)

Register	IRC Field Name	MH Port Name	PRT Port Name
TX_FUNC_RAW	PRT_TX_EVT	NA	TX_EVT
	MH_CTB_TX_BC_REQ	FUNC_CTB_TX_BC_REQ_IRQ	NA
	MH_TEFQ_DEQ	FUNC_TEFQ_DEQ_IRQ	NA
	MH_TEFQ_ENQ	FUNC_TEFQ_ENQ_IRQ	NA
	MH_TXPQ_ENQ	FUNC_TXPQ_ENQ_IRQ	NA
	MH_TXFQ_ENQ	FUNC_TXFQ_ENQ_IRQ	NA

Table 132. IRC Flags and Source Ports (page 2 of 2)

Register	IRC Field Name	MH Port Name	PRT Port Name
RX_FUNC_RAW	PRT_RX_EVT	NA	RX_EVT
	MH_RX_MSG_ALERT	FUNC_MH_RX_MSG_ALERT	NA
	MH_RX_FILTER_MATCH	FUNC_MH_RX_FILTER_MATCH_IRQ	NA
	MH_CTB_RX_BC_REQ	FUNC_CTB_RX_BC_REQ_IRQ	NA
	MH_RXFQ1_DEQ	FUNC_RXFQ1_DEQ_IRQ	NA
	MH_RXFQ0_DEQ	FUNC_RXFQ0_DEQ_IRQ	NA
	MH_RXFQ1_ENQ	FUNC_RXFQ1_ENQ_IRQ	NA
	MH_RXFQ0_ENQ	FUNC_RXFQ0_ENQ_IRQ	NA
ERR_STS_RAW	PRT_BUS_ON	NA	BUS_ON
	PRT_E_ACTIVE	NA	E_ACTIVE
	PRT_BUS_OFF	NA	BUS_OFF
	PRT_E_PASSIVE	NA	E_PASSIVE
	PRT_BUS_ERR	NA	BUS_ERR
	PRT_IFF	NA	IFF_RQ
	PRT_STOP	ERR_STS_PRT_STOP_IRQ	NA
	MH_VB_NO_SA	ERR_STS_VB_NO_SA_IRQ	NA
	MH_QUEUE_ACCESS_VIOLATION	ERR_STS_QUEUE_ACCESS_VIOLATION_IRQ	NA
	MH_LMEM_CERR	ERR_STS_LMEM_CERR_IRQ	NA
	MH_RXFQ1_DROP	ERR_STS_RXFQ1_DROP_IRQ	NA
	MH_RXFQ0_DROP	ERR_STS_RXFQ0_DROP_IRQ	NA
	MH_TEFQ_DROP	ERR_STS_TEFQ_DROP_IRQ	NA
	MH_TXPQ_DEQ_NO_TX	ERR_STS_TXPQ_DEQ_NO_TX_IRQ	NA
	HOST_ARA	NA	NA
	MH_TXFQ_DEQ_NO_TX	ERR_STS_TXFQ_DEQ_NO_TX_IRQ	NA
SAFETY_RAW	PRT_RX_DO	NA	RX_DO
	PRT_TX_DU	NA	TX_DU
	PRT_USOS	NA	USOS
	PRT_TX_ABORTED	NA	ABORTED
	MH_TX_ABORT	SAFETY_TX_ABORT_IRQ	NA
	MH_RX_ABORT	SAFETY_RX_ABORT_IRQ	NA
	MH_RX_FILTER_ERR	SAFETY_RX_FILTER_ERR_IRQ	NA
	MH_SAFETY_MISC	SAFETY_MISC_IRQ	NA
	MH_VB_ACCESS_VIOLATION	SAFETY_VB_ACCESS_VIOLATION_IRQ	NA
	MH_LMEM_PROT_ERR	SAFETY_LMEM_PROT_ERR_IRQ	NA
	MH_LMEM_TO_ERR	SAFETY_LMEM_TO_ERR_IRQ	NA
	MH_LMEM_UERR	SAFETY_LMEM_UERR_IRQ	NA
	MH_CTB_BC_ERR	SAFETY_CTB_BC_ERR_IRQ	NA
	MH_CTB_BC_TO	SAFETY_CTB_BC_TO_IRQ	NA

Note that HOST_ARA is an 'OR' of HOST_ARA from VBM and HOST_ARA interrupt from AHB to APB Bridge. HOST_ARA interrupt is raised by the bridge if there is an illegal access to any of the registers in IRC, MH and PRT. Refer section 1.4.2.1 AHB (Advanced High performance Bus) to APB (Advanced Peripheral Bus) Bridge for details.

1.9 Clock Domains and Resets

1.9.1 Clock Domains

The XS_CAN IP is split into three clock domains.
TIMEBASE Clock Domain for timebase CDC
CAN Clock Domain for PRT
HOST Clock Domain for all other modules

Only the rising edge of the clocks are considered for operation.

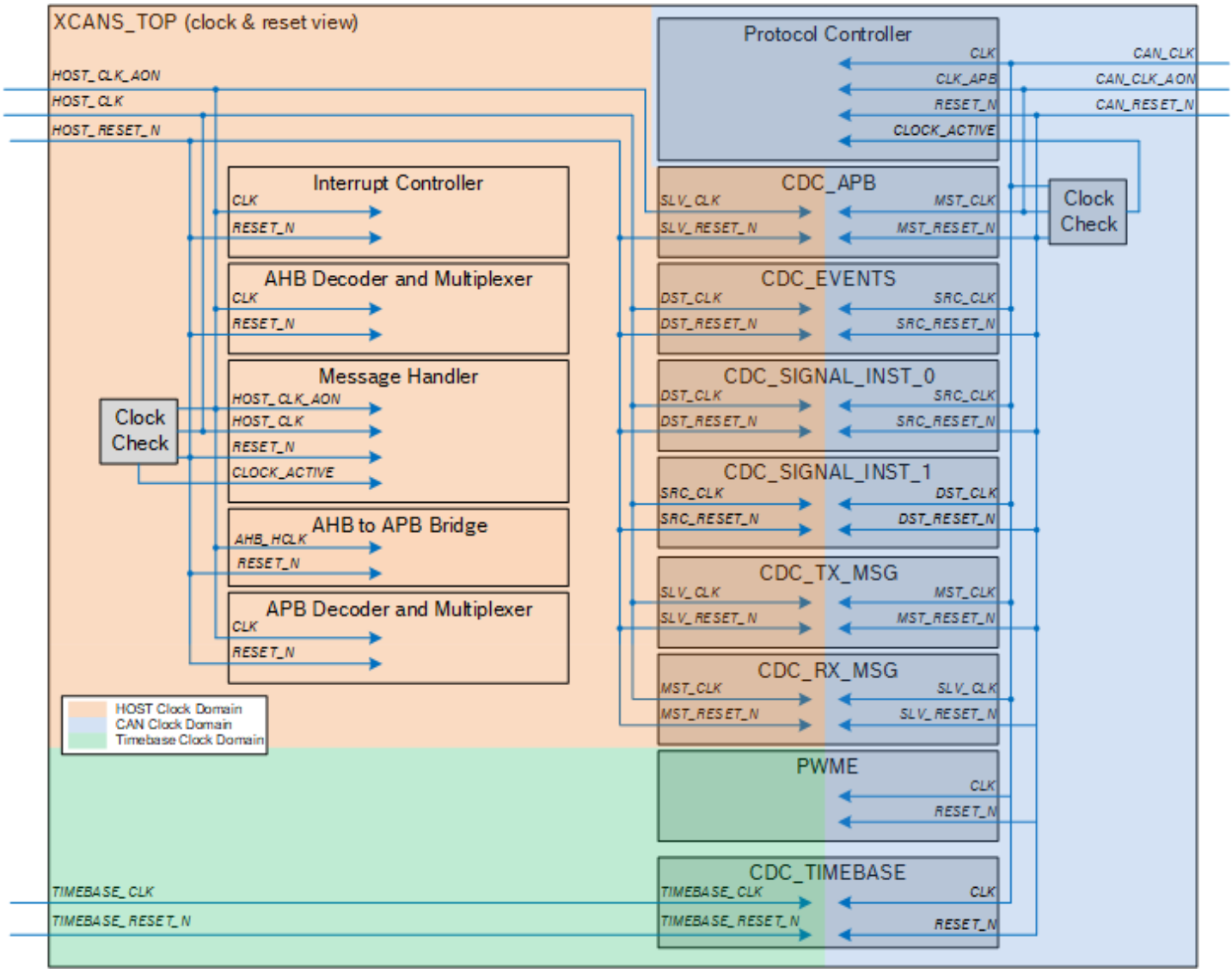


Figure 56. XS_CAN_TOP Clock & Reset

These three clock domains provide flexibility for XS_CAN IP integration. If less than three domains are required, domains can be bound together by using the same clock and reset sources.

The integrator should check, which embedded CDC modules become obsolete and should replace those by dummy architectures, to reduce gate count.

The IP supports the following clock rations (verified in simulation).
The allowed ratio of CAN_CLK:TIMEBASE_CLK is in the range 1:16 to 2:1.
The allowed ratio of CAN_CLK:HOST_CLK is in the range 1:5 and 4:1.
Any ratio within these ranges is allowed. For example, the CAN_CLK:HOST_CLK ratios 1:1, 1:2, 3:1, 3:2 are supported, since they are in the allowed range.

The absolute clock frequencies ranges, the IP was designed for, are listed below. Exceeding these ranges is allowed, if the clock ratios are in the allowed range. Verification and validation was carried out with these absolute frequencies.
HOST_CLK: 20 MHz to 320 MHz
CAN_CLK: 20 MHz to 160 MHz
TIMBASE_CLK: 10 MHz to 2560 MHz
CAN_CLK recommendations:
For CAN XL operation, CAN_CLK characteristics like frequency, tolerance, and jitter are given in CiA 612-1.
CiA 612-1 recommends to use for CAN XL data bit rates of up to 20Mbps a CAN clock of 160MHz.
For CAN FD operation, CAN_CLK characteristics like frequency and tolerance are given in CiA 601-3. CiA 601-3 recommends for CAN FD data bit rates of up to 8Mbps a CAN_CLK of 80MHz, and for CAN FD data bit rate limited devices up to 2 Mbps a CAN_CLK of 40MHz.

Table 133. Clock Frequency Verification Summary

	HOST_CLK	CAN_CLK	TIMEBASE_CLK
	Range		
Verified Frequency	20MHz to 320MHz	20MHz to 160MHz	10MHz to 2560MHz
Conceptually possible (Not Verified)	Below 20MHz and above 320MHz	Below 20MHz and above 160MHz	Below 20MHz and above 2560MHz

Integrator must check the host clock frequency requirements using the Excel sheet provided based on the use case.

All clock domains can be supplied with clocks which are asynchronous to each other. The XS_CAN IP internally handles all clock domain crossings, fully transparent for the user.

The XS_CAN IP gets two types of clock inputs. One type of clock that goes to the submodule's APB configuration interface (HOST_CLK_AON and CAN_CLK_AON) should always be on and never turned off during operation. The second type of clock (HOST_CLK and CAN_CLK) is for internal logic and could be switched off for power down mode. The clock to clock checker module checks the HOST_CLK and CAN_CLK and provides information to MH and PRT whether clock is on or off.
XXX_CLK and XXX_CLK_AON must be of the same frequency and phase aligned. If XXX_CLK is turned on after switching off during the power down phase, it must be aligned with XXX_CLK_AON without any change in frequency or phase. It is recommended to generate XXX_CLK and XXX_CLK_AON from the same source.

All internal Flip Flops of the XS_CAN IP take over the data triggered by the rising clock edge. The following chapters describe all CDC components used for internal clock domain crossings. All CDC components provide a hardware parameter SYNC_FF_COUNT_G, which defines the number of synchronization flip-flops in series. For the XS_CAN IP, the parameter is set to the default, which is two FFs in series. It is not recommended to change the value, because this has an impact for the latency of the CDC and therefore affects the IP function. This is the reason, why those parameters are not exposed at top level.

1.9.1.1 Behavior While Not Clocked

Here below is the XS_CAN behavior when its clocks are off:
• Accessing the HOST_AHB interface when the HOST_CLK_AON is inactive will result in a hang-up.
• Accessing the PRT through the APB interface, when the CAN_CLK_AON is inactive, should result in a timeout at the host side. There is no timer inside the IP for tracking this.

1.9.1.2 CDC Modules

1.9.1.2.1 CDC for PRT Events and PRT_STOP_IMMD_REQ signal from MH

Event signals of the XS_CAN that are driven by the Protocol Controller in the CAN clock domain and are captured by the Interrupt Controller located in the HOST clock domain. The signal PRT_STOP_IMMD_REQ is driven by the message handler from HOST clock domain and is used by the Protocol Controller in the CAN clock domain.The module CDC_EVENTS is used for this clock domain crossing.

The PRT provides single clock cycle active high pulses when an event occurs. The CDC ensures, that such pulses will be transferred to the destination and are again single clock cycle active high pulses.

The complete list of PRT events is shown in IRC register definition, i.e., the one with prefix PRT.

Figure 57. CDC_EVENTS

CDC_EVENTS makes the synchronization of all 13 event lines coming from the PRT. It is built from 13 instances of CDC_PULSE_FLOW, each synchronizing a single event line.
The following section provides details about the CDC_PULSE_FLOW.

1.9.1.2.1.1 CDC Pulse Flow

The CDC Pulse Flow design is a clock domain crossing component. It synchronizes incoming pulses at a source clock domain to a destination clock domain. Thereby the design can store incoming pulses while it is transferring a prior pulse.

1) Features

- Transfer of pulses from source block side to destination clock side
- The clock frequencies may have any ratio
- The clock may have any phase relation
- Storing of incoming pulses while the component is occupied with a prior pulse transfer
- The number of pulses the design is able to store can be defined by design parameters

2) Block Diagram

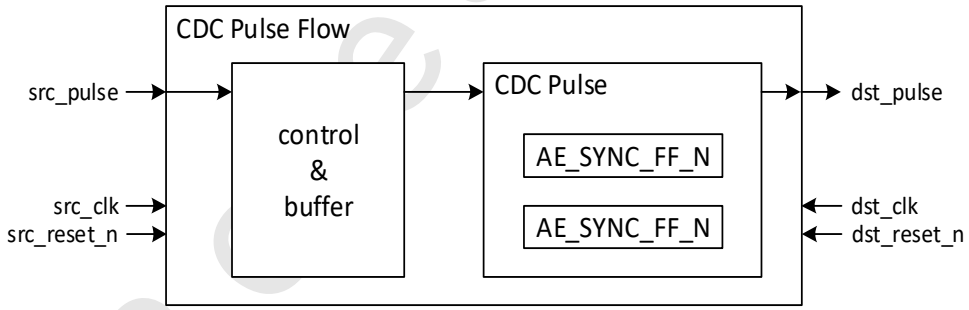


Figure 58. Block Diagram of CDC Pulse Flow

3) Hardware Interface

Not applicable.

4) Software Interface

Not applicable.

5) Functional Description

The design transfers incoming pulses from the source clock domain (SRC) to the destination clock domain (DST). A pulse is defined as a sequence of “0” – “1” – “0”. The input signal for the pulses at the source clock domain is SRC_PULSE. The output signal for the pulses synchronized to the destination clock domain is DST_PULSE.

If pulses arrive while the design is still transferring a prior pulse the new pulse(s) are stored.
Over two generic parameters PULSE_COUNT_WIDTH_G and PULSE_COUNT_MAX_G the amount of stored pulses can be defined.

If parameter values are set so that every incoming pulse over a defined time period can be stored (no counter overflow) all pulses at the source side are generated at the destination side, too. If parameter values are used that this is not the case, the design will transport as much as possible pulses, but the number of pulses at the destination side can be less than at the source side.

The following timing diagrams give two examples.
Setup of the first timing diagram:

- Clock frequency ratio (SRC:DST):1:2
- SRC_RESET_N and DST_RESET_N are active low
- A fast sequence of ten pulses followed by a single pulse later
- The design is able to store two pulses; design parameter PULSE_COUNT_MAX_G = 2

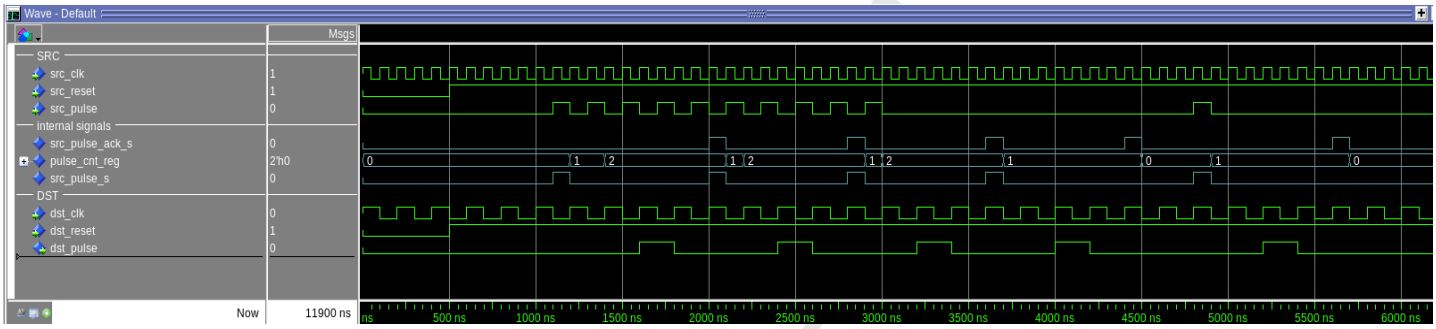


Figure 59. Waveform: PULSE_COUNT_MAX_G = 2

As shown in the timing diagram a fast pulse sequence of ten pulses arrives at signal SRC_PULSE followed by a single pulse some time later.

The internal signal PULSE_CNT_REG counts the stored pulses. It is increased if a new pulse arrives at the input SRC_PULSE without an acknowledge pulse (SRC_PULSE_ACK_S) which comes from the clock domain crossing block that a pulse has been transferred. The counter is decreased if an acknowledge pulse (SRC_PULSE_ACK_S) occurs without an input pulse at SRC_PULSE. In all other cases the counter stays unmodified.

It can been seen that in this implementation the design is not able to store all incoming pulses, but only two. Thus only four of the ten incoming pulses are transferred to the destination side. But it is guaranteed that after the last incoming pulse of such a fast pulse sequence there is always one or more pulses at the destination side.

Setup of the second timing diagram:

- Clock frequency ratio (SRC:DST):1:2 (as before)
- SRC_RESET_N and DST_RESET_N are active low
- A fast sequence of ten pulses followed by a single pulse later (as before)
- The design is able to store 16 pulses; design parameter PULSE_COUNT_MAX_G = 16

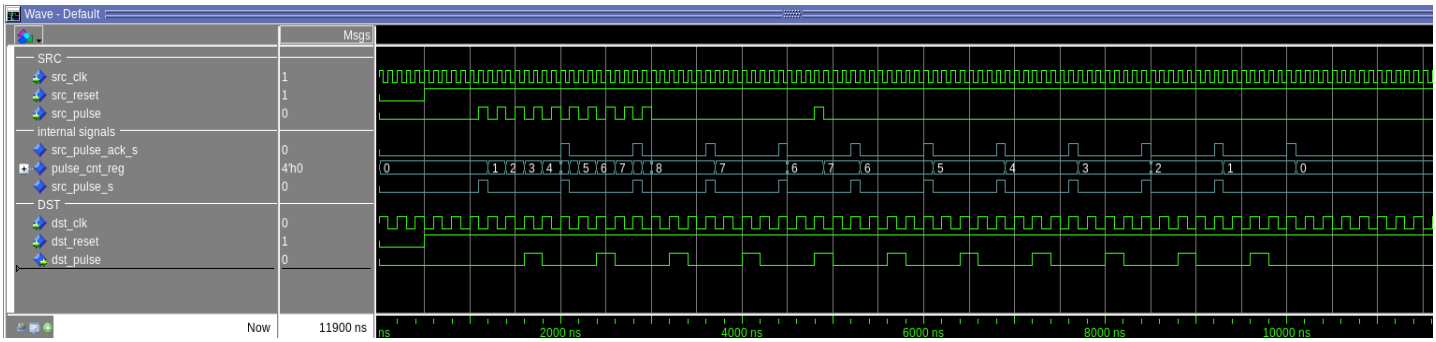


Figure 60. Waveform: PULSE_COUNT_MAX_G = 16

It can been seen that in this implementation the design is able transfer all incoming pulses to the destination clock domain. The delay depends to the clock ratio.

6) Safety Mechanisms
None.

7) Application Information

The design is able to work with any clock frequencies of source clock and destination clock.
The clocks frequencies may have any ratio.
The two clocks may be synchronous or asynchronous.
The active clock edge of the design is the rising edge.
The reset signals SRC_RESET_N and DST_RESET_N are asynchronous resets.
The reset inputs of both clock domains have to be asserted (set to low) at the same time, and each deasserted (set to high) synchronously to the dedicated domain clock.
The input signal SRC_PULSE shall be a single pulse signal. This means it is mandatory that the signal is always a '0'-'1'-'0' sequence.

8) Verification and Validation Requirements

It is recommended to do a consistency check or lint check.
The design has to be checked by a clock domain crossing check tool.
The design shall be simulated in the target design environment to check that it fulfills all application needs.

9) Detailed Design Information
The following figure shows the CDC Pulse Flow design in detail.

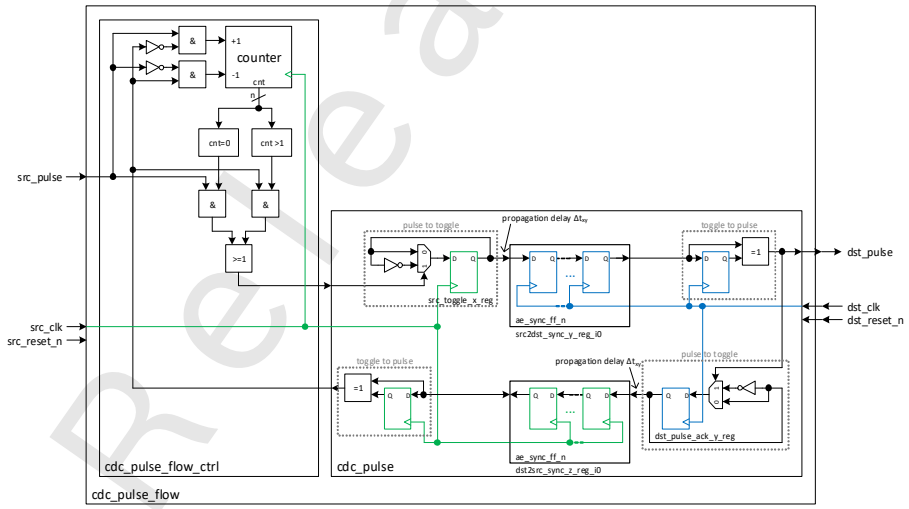


Figure 61. CDC Pulse Flow Design

a) Port Description

Table 134. CDC Pulse Flow Port Description

Signal	Bit Width	I/O	Description	Active Level	Clock Domain
SRC_CLK	1	I	Clock for SRC clock domain	rising edge	SRC
SRC_RESET_N	1	I	Asynchronous reset for SRC clock domain	low	SRC
SRC_PULSE	1	I	Input of pulse signal	high	SRC
DST_CLK	1	I	Clock for DST clcok domain	rising edge	DST
DST_RESET_N	1	I	Asynchronous reset for DST clock domain	low	DST
DST_PULSE	1	O	Output of pulse signal	high	DST

b) Parameters

Table 135. CDC Pulse Flow Parameters

Parameter name	Default Value	Description/Constraints
SYNC_FF_COUNT_G	2	number of synchronization flip-flops
PULSE_COUNT_WIDTH_G	2	width of pulse counter
PULSE_COUNT_MAX_G	2	number of pulses the design can store

c) Memory Needs
Not applicable.

d) Design Dimensioning Details
The generic parameters how many pulses can be stored shall fulfill the following:
PULSE_COUNT_MAX_G <= 2**PULSE_COUNT_WIDTH_G - 1

Flip-Flop count of the design:
PULSE_COUNT_WIDTH_G + 4 + 2*SYNC_FF_COUNT_G
With the generic parameter SYNC_FF_COUNT_G the number of synchronization flip-flops can be defined. Depending to the used ASIC technology and the mean time between failure requirements the number of synchronizer flip-flops can be adjusted.

The latency of a first pulse (this mean the design is not occupied by a prior pulse transfer) at the source side to the destination side is:
1 * SCR_CLK period + (1 + SYNC_FF_COUNT_G + (+0/-1)) * DST_CLK period

The latency until a next pulse can be transferred (design is occupied by a prior pulse to be transferred) can be up to:
(2 + SYNC_FF_COUNT_G + (+0/-1)) * SCR_CLK period + (1 + SYNC_FF_COUNT_G + (+0/-1)) * DST_CLK period

Note: In the formulas above the expression (+0/-1) is used to consider the uncertainty in the delay of the synchronization stage, which depends on the clock phases between SRC_CLK and DST_CLK.

e) Test Concept
Not applicable.

10) Integration Guidelines
The design includes the block AE_SYNC_FF_N. This block includes the synchronization flip-flops. The block can be replaced by the IP customer by its own synchronization flip-flop component.

a) False Path Constraints
With reference to the figure above the timing of the paths

- From register output SRC_TOGGLE_X_REG/Q to register data input SRC2DST_SYNC_Y_REG_I0/SYNC_REG(0)/D and
- From register output DST_PULSE_ACK_Y_REG/Q to register data input DST2SRC_SYNC_Z_REG_I0/SYNC_REG(0)/D

are not critical for setup time checks. Therefore, a false path constraint for synthesis to this flip-flops is recommended.

Following false paths are recommended:
set_false_path -from [find cell {*x_reg_reg} -hierarchy]
-to [find cell {*sync_y_reg_i0/sync_reg_reg(0)} -hierarchy]
set_false_path -from [find cell {*y_reg_reg} -hierarchy]
-to [find cell {*sync_z_reg_i0/sync_reg_reg(0)} -hierarchy]

b) Maximum Delay Constraints
To avoid a potential metastable state at the synchronizer flip-flops (SYNC_REG(0), ..., SYNC_REG (sync_ff_count_g-1)), the paths between two synchronizer flip-flops should be as short as possible (i.e., two flip-flops should be placed together as close as possible). At least for ASIC back-end flow and FPGA synthesis tools it is recommended to specify additionally a maximum delay on dedicated signal paths.

As a result, it is recommended to constraint a maximum path delay from flip-flop SYNC_REG(0) to SYNC_REG(1) and probably for all further synchronizer flip-flop pairs e.g., SYNC_REG(1) to SYNC_REG(2) and so on.

The propagation delay on the domain crossing paths should also be constrained. It is recommended that the maximum propagation delay does not exceeds the value of one destination clock period minus the setup time of destination Flip-flop on the following paths:

- From Flip-flop SRC_TOGGLE_X_REG to Flip-flop SRC2DST_SYNC_Y_REG_I0/SYNC_REG(0) (propagation delay Δtxy)
- From Flip-flop DST_PULSE_ACK_Y_REG to Flip-flop DST2SRC_SYNC_Z_REG_I0/REG_SYNC(0) (propagation delay Δtyz)

1.9.1.2.2 CDC for ENABLE

The ENABLE signal is driven by the Protocol Controller in the CAN clock domain and is used by the Message Handler located in the HOST clock domain.

The module CDC_SIGNAL is used for this clock domain crossing.

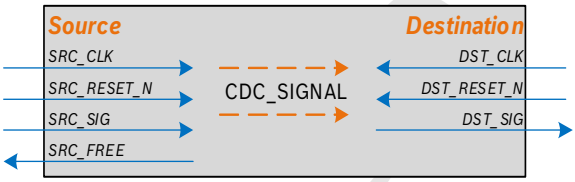


Figure 62. CDC_SIGNAL

The following section provides details about the embedded CDC component.

1.9.1.2.2.1 CDC Signal

The CDC Signal design is a clock domain crossing component. It synchronizes an input signal level at a source clock domain to a destination clock domain.

1) Features

The components has the following features:

- Transfer of signal level from source clock side to destination clock side
- The clock frequencies may have any ratio
- The clocks may have any phase relation

2) Block Diagram

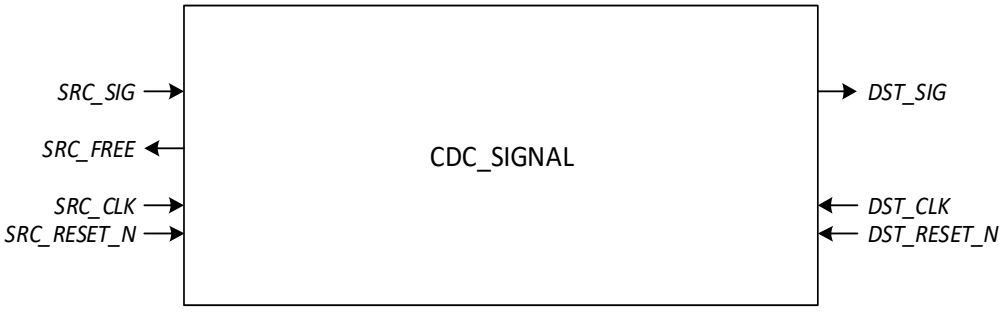


Figure 63. Block Diagram of CDC_SIGNAL

3) Hardware Interface

Not applicable.

4) Software Interface

Not applicable.

5) Functional Description

The design transfers the level of a signal from the source clock domain (SRC) to the destination clock domain (DST). The input signal is SRC_SIG. The output signal is DST_SIG.

An additional output signal SRC_FREE at the source clock domain shows if the design is free to transfer a new level change of SRC_SIG or still occupied to transfer the last level change. The usage of this signal is optional and depends to the application or design which instantiates this CDC component.

If the input signal SRC_SIG is changed while SRC_FREE is zero then it will took more time until the signal level change is transferred to the destination side (see 1.7, too).

It is mandatory that during SRC_FREE is zero the input signal isn't changed more than once. Otherwise it is undefined if such a short pulse is available at the destination side.

The following timing diagrams give some examples of signal level changes. For all examples in this documentation the generic parameter SYNC_FF_COUNT_G is set to 2.

Setup of the timing diagrams:

- Clock frequency ratio (SRC:DST):4:1 (10MHz:2,5MHz)

In the first timing diagram a relaxed timing of the input signal SRC_SIG is shown.

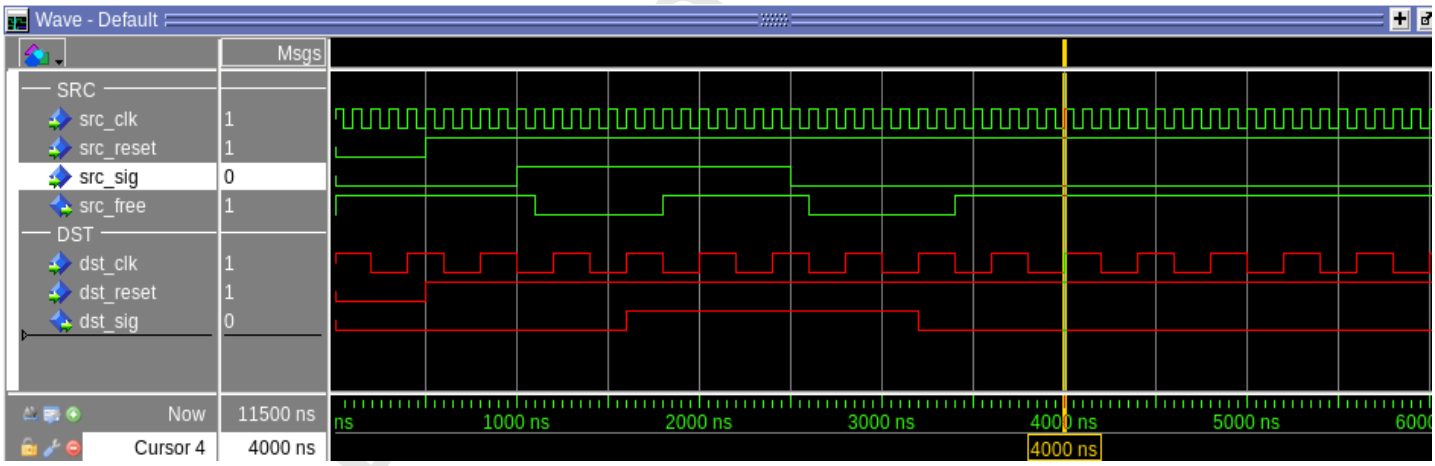


Figure 64. Relaxed Timing of SRC_SIG

Signal SRC_SIG changes its level from zero to one at 1000ns. One SRC_CLK cycle later at 1100ns the output signal SRC_FREE changes to zero to show that a transfer is ongoing. At 1600ns output signal DST_SIG changes its level to one. At 1800ns signal SRC_FREE changes to one again to show that the CDC component is free to transfer a new signal level.

At 2500ns a new signal level change at SRC_SIG now from one to zero occurs.
In the next timing diagram two consecutive signal level changes occurs as fast as possible (one source clock cycle between the edges).

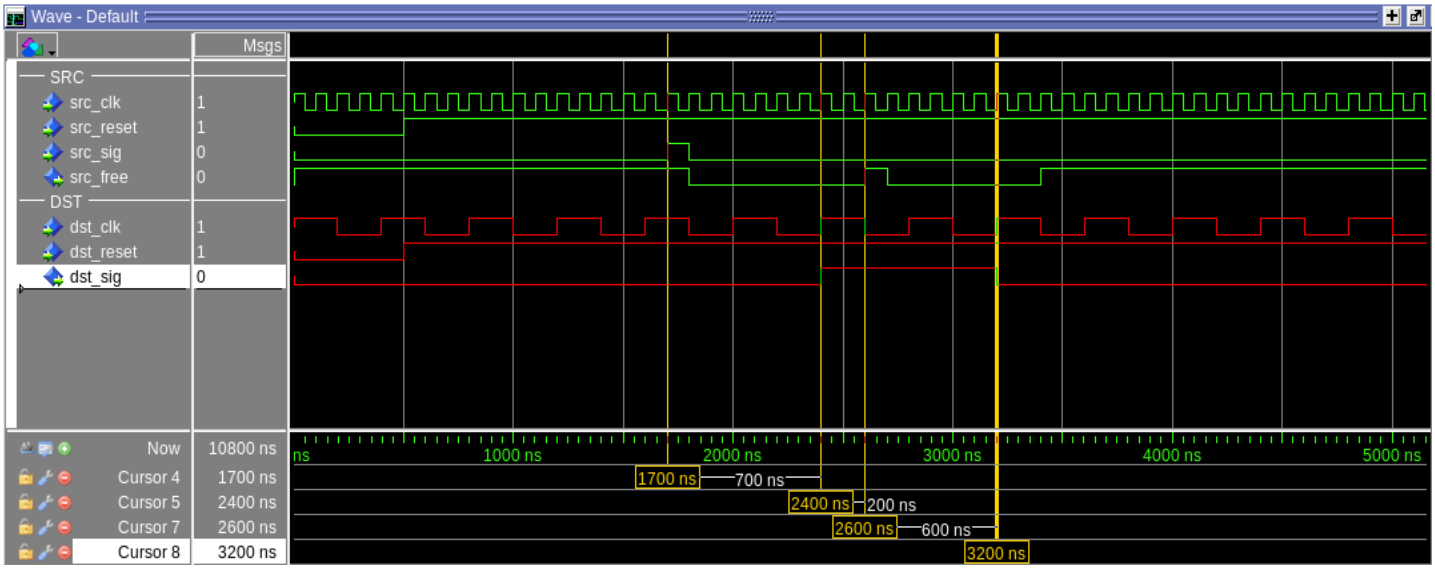


Figure 65. SRC_SIG at One Pulse High

The first change at 1700ns is processed as described above for the first timing diagram. But here in the second timing diagram a second signal level change at SRC_SIG just one source clock cycles later is done.

The CDC component transfers the first signal change which is shown by setting SRC_FREE to zero.
The second signal level change can't be processed at this time.

At 2400ns the first signal level change is available at output DST_SIG. Two source clock cycles (200ns) later at 2600ns the output signal SRC_FREE shows that the design is ready to transfer a new signal level change. This is the time the second signal level change is transferred to the destination side.

In the third timing diagram a situation is shown where not all signal level changes are transferred, because the changes arrive faster at the input than they can be propagated to the destination side due to the given clock ratio.

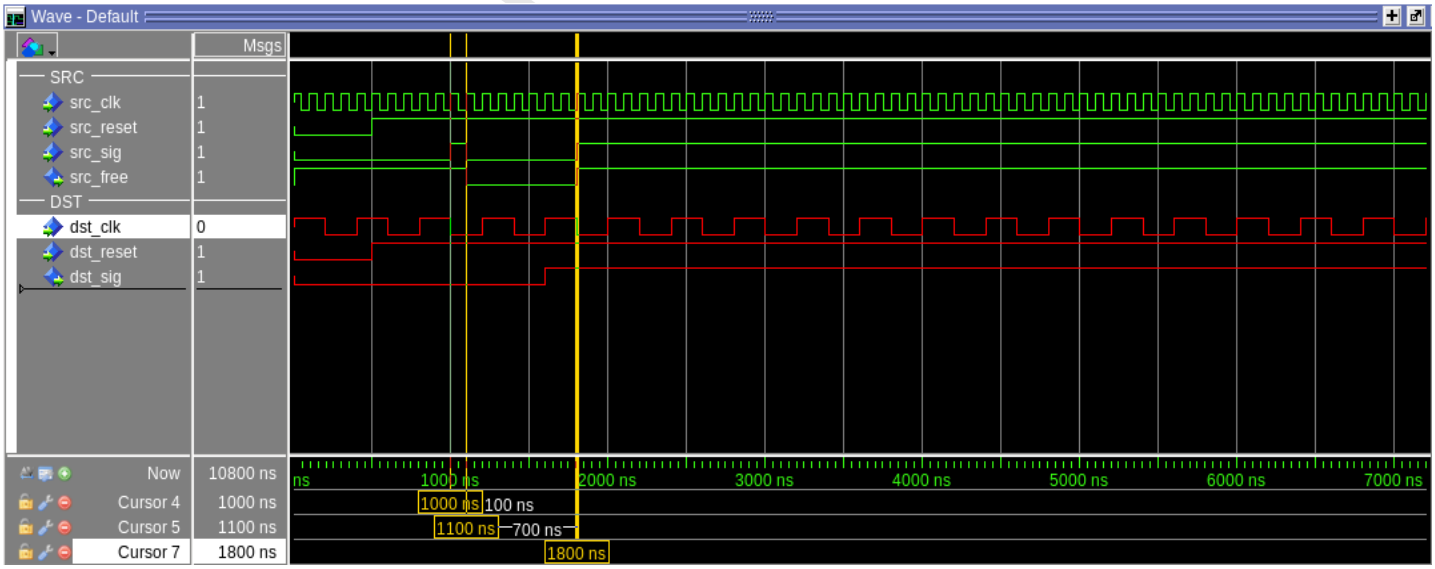


Figure 66. SRC_SIG with 3 Consecutive Changes

Three consecutive signal level changes occur at 1000ns, 1100ns and 1800ns. The second signal level change at 1100ns occurs during the design is transferring the first change. The 3rd signal level change at 1800ns occurs just when the CDC component gets free from the 1st change. Thus the 2nd change at 1100ns isn't seen and nothing has to be transferred, because the signal level after the 1st change and the 3rd change is the same. The timing diagram shows the maximum time a signal level at the input SRC_SIG isn't transferred to DST_SIG corresponding to the used clock periods of SRC_CLK and DST_CLK.

6) Safety Mechanisms

None.

7) Application Information

The design is able to work with any clock frequencies of source clock and destination clock.
The clocks frequencies may have any ratio.
The two clocks may be synchronous or asynchronous.
The active clock edge of the design is the rising edge.
The reset signals SRC_RESET_N and DST_RESET_N are asynchronous resets.
The reset inputs of both clock domains have to be asserted (set to low) at the same time, and each deasserted (set to high) synchronously to the dedicated domain clock.

The duration until a signal level change at SRC_SIG is transferred to DST_SIG is:
1 * SCR_CLK period + (SYNC_FF_COUNT_G (+0/-1)) * DST_CLK period

Note: In the formula above the expression (+0/-1) is used to consider the uncertainty in the delay of the synchronization stage, which depends on the clock phases between SRC_CLK and DST_CLK.

The duration until an SRC_FREE is zero after a signal level change is:
1 * SCR_CLK period

The duration until an SRC_FREE is one again after a signal level change is:
(1 + SYNC_FF_COUNT_G (+1)) * SCR_CLK period + (SYNC_FF_COUNT_G (+1)) * DST_CLK period

Note: In the formula above (+1) is used to show that there is an uncertainty of one additional SRC_CLK and DST_CLK clock period if the two clocks are not an integer multiple of each other (phase shift of the two clocks is always zero).

It is mandatory that during SRC_FREE is zero only one signal level change at SRC_SIG is done.
The time period during only one signal level change may occur at SRC_SIG depends to the two clock periods for source and destination side:
(SYNC_FF_COUNT_G + (+1))* SCR_CLK period + (SYNC_FF_COUNT_G + (+1)) * DST_CLK period

Note: In the formula above (+1) is used to show that there is an uncertainty of one additional SRC_CLK and/or DST_CLK clock period if the two clocks are not an integer multiple of each other (phase shift of the two clocks is always zero).

Note: This CDC component is not designed to transfer pulses (0-1-0 / 1-0-1). Therefore, another CDC component e.g., CDC Pulse or CDC Pulse Flow should be used.

8) Verification and Validation Requirements

It is recommended to do a consistency check or lint check.
The design has to be checked by a clock domain crossing check tool.
The design shall be simulated in the target design environment to check that it fulfills all application needs.

9) Detailed Design Information

The following figure shows the CDC Signal design in detail.

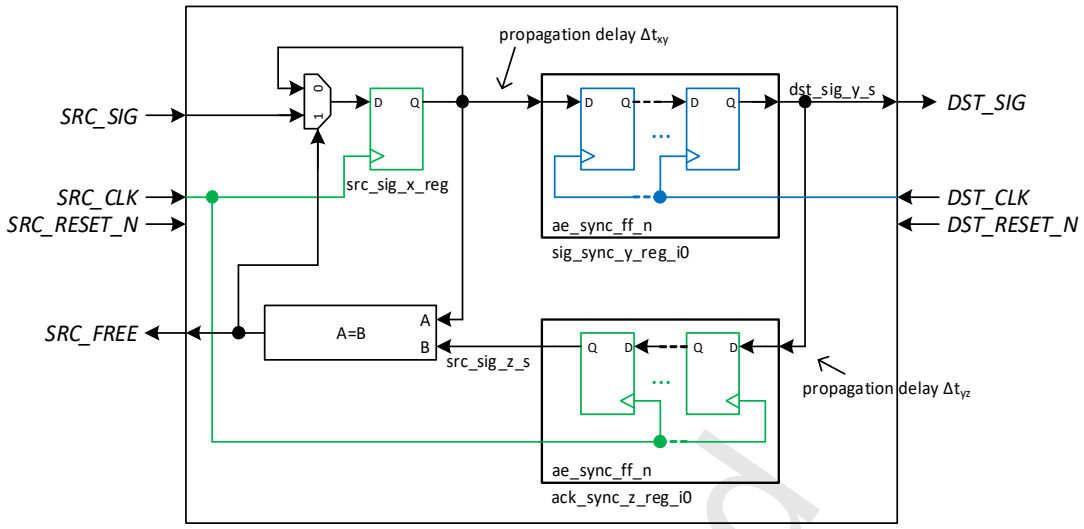


Figure 67. CDC_SIGNAL Flow Design

a) Port Description

Table 136. CDC Signal Port Description

Signal	Bit Width	I/O	Description	Active Level	Clock Domain
SRC_CLK	1	I	Clock for SRC clock domain	rising edge	SRC
SRC_RESET_N	1	I	Asynchronous reset for SRC clock domain	low	SRC
SRC_SIG	1	I	Signal for clock domain crossing (input)	high	SRC
SRC_FREE	1	O	Output signal which shows if component is free	high	SRC
DST_CLK	1	I	Clock for DST clock domain	rising edge	DST
DST_RESET_N	1	I	Asynchronous reset for DST clock domain	low	DST
DST_SIG	1	O	Signal for clock domain crossing (output)	high	DST

b) Parameters

Table 137. CDC Signal Parameters

Parameter name	Default Value	Description/Constraints
SYNC_FF_COUNT_G	2	number of synchronization flip-flops
OUTVAL4RESET_G	0	default value of all FFs in the design and of signal DST_SIG

c) Memory Needs
Not applicable.

d) Design Dimensioning Details
Flip-Flop count of the design:
1 + 4 * SYNC_FF_COUNT_G
With the generic parameter SYNC_FF_COUNT_G the number of synchronization flip-flops can be defined. Depending to the used ASIC technology and the mean time between failure requirements the number of synchronizer flip-flops can adjusted.

10) Integration Guidelines
The design includes the block AE_SYNC_FF. This block includes the synchronization flip-flops. The block can be replaced by the IP customer by its own synchronization flip-flop component.

a) False Path Constraints
With reference to the figure above the timing of the paths

- From register output SRC_SIG_X_REG/Q
- To register data input SIG_SYNC_Y_REG_I0/SYNC_REG(0)/D and
- From register output SIG_SYNC_Y_REG_I0/SYNC_REG(sync_ff_count_g-1)/Q
- Alias signal DST_ISIG_Y_S
- To register data input ACK_SYNC_Z_REG_I0/SYNC_REG(0)/D

are not critical for setup time checks. Therefore, a false path constraint for synthesis to this flip-flops is recommended.

Following false paths are recommended:
set_false_path -from [find cell {*x_reg_reg} -hierarchy]
– to [find cell {*sync_y_reg_i0/sync_reg_reg(0)} -hierarchy]
set_false_path -from [find cell {*sync_y_reg_i0/sync_reg_reg(sync_ff_count_g-1)} -hierarchy]
– to [find cell {*sync_z_reg_i0/sync_reg_reg(0)} -hierarchy]

b) Maximum Delay Constraint
To avoid a potential metastable state at the synchronizer flip-flops (SYNC_REG(0), ..., SYNC_REG(sync_ff_count_g-1)), the paths between two synchronizer flip-flops should be as short as possible (i.e., two flip-flops should be placed together as close as possible). At least for ASIC back-end flow and FPGA synthesis tools it is recommended to specify additionally a maximum delay on dedicated signal paths.

As a result, it is recommended to constraint a maximum path delay from flip-flop SYNC_REG(0) to SYNC_REG(1) and probably for all further synchronizer flip-flop pairs e.g., SYNC_REG(1) to SYNC_REG(2) and so on.

The propagation delay on the domain crossing paths should also be constrained. It is recommended that the maximum propagation delay does not exceeds the value of one destination clock period minus the setup time of destination Flip-flop on the following paths:

- From Flip-flop SRC_SIG_X_REG to Flip-flop REQ_SYNC_Y_REG_I0/SYNC_REG(0)
- (propagation delay Δtxy)
- From Flip-flop REQ_SYNC_Y_REG_I0/SYNC_REG(sync_ff_count_g-1) to Flip-flop ACK_SYNC_Z_REG_I0/SYNC_REG(0)
- (propagation delay Δtyz)

1.9.1.2.3 CDC for busses

The XS_CAN IP uses three different busses to interchange data between local modules, i.e., REG_APB, TX_MSG and RX_MSG. Each bus is divided into channels which all have a common behavior.

Such a channel has its own two-way flow control; thus, both the source and destination can control the transfer rate. The source signalizes new data to be transferred by asserting VALID=1 and keeps its information stable until the transfer occurs. The Transfer occurred when destination acknowledges by asserting READY=1.

All CDCs for the busses are based on one design, which is making the clock domain crossing for a single channel.

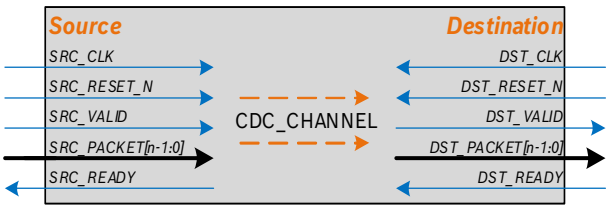


Figure 68. CDC_CHANNEL

The following table shows the latencies of the CDC_CHANNEL for different clock period ratios between HOST and CAN clock domain. CDC_CHANNEL parameter SYNC_FF_COUNT_G=2. The first column is the ratio of the clock period between HOST and CAN clock domain. The second column is the latency (unit is HOST clock cycles) to forward a transfer from HOST to the CAN clock domain, i.e., at CDC_CHANNEL: SRC_VALID=SRC_READY=1 until DST_VALID=1. It is relevant for the Write Data Channel of the CDC_TX_MSG. The third column is the latency (unit is HOST clock cycles) to forward a transfer from CAN to the HOST clock domain, i.e., at CDC_CHANNEL: SRC_VALID=SRC_READY=1 until

DST_VALID=1. It is relevant for the Write Response Channel of the CDC_TX_MSG and the Write Data Channel of the CDC_RX_MSG.

Table 138. CDC Channel Latency

Clock Period HOST:CAN	Latency [HOST cycles] HOST-> CAN	Latency [HOST cycles] CAN-> HOST
1:1	5	5
1:2	9	6
1:4	17	8

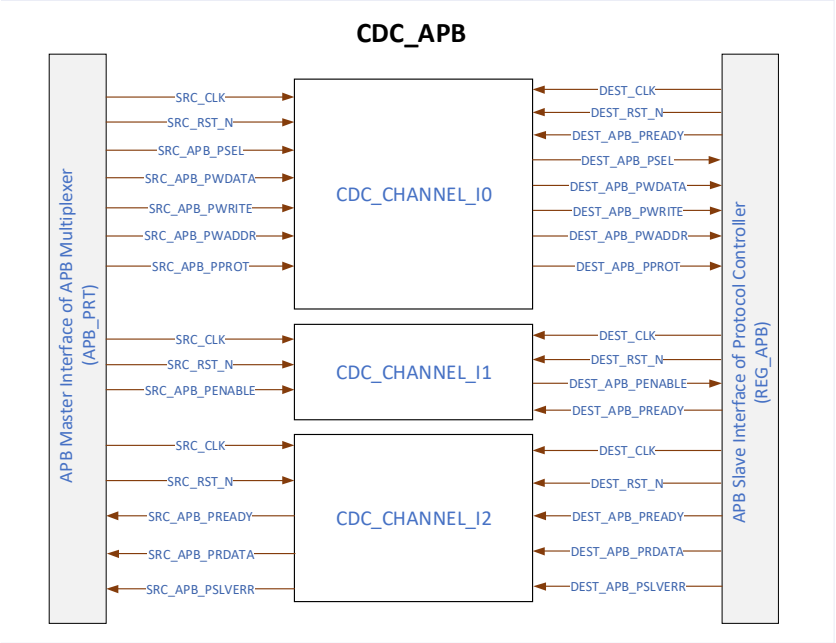


Figure 69. CDC_APB Block Diagram

The CDC_APB is used for the register interface REG_APB of the PRT and makes the clock domain crossing between HOST clock domain and CAN clock domain.

The signals coming from APB Multiplexer (APB master) pass through **CDC_APB** to the destination PRT (APB Slave). When a new transaction is initiated (PSEL and PENABLE are 1), **CDC_CHANNEL_I0** block performs the CDC operation for PSEL, PADDR, PWDATA, PWRITE and PPROT from *HOST_CLK_AON* domain to *CAN_CLK_AON* domain.

In APB, there has to be a 1 clock cycle delay between PSEL and PENABLE signal. As a result, **CDC_CHANNEL_I1** is used to perform the CDC for PENABLE signal.

The signals coming from the PRT to APB Mux PRADATA, PSLVERR, PREADY passes through **CDC_CHANNEL_I2** to undergo the CDC from *CAN_CLK_AON* domain to *HOST_CLK_AON* domain.

The following figure shows the application of the CDC_CHANNEL for the clock domain crossing of the TX_MSG interface. This component is named CDC_TX_MSG.

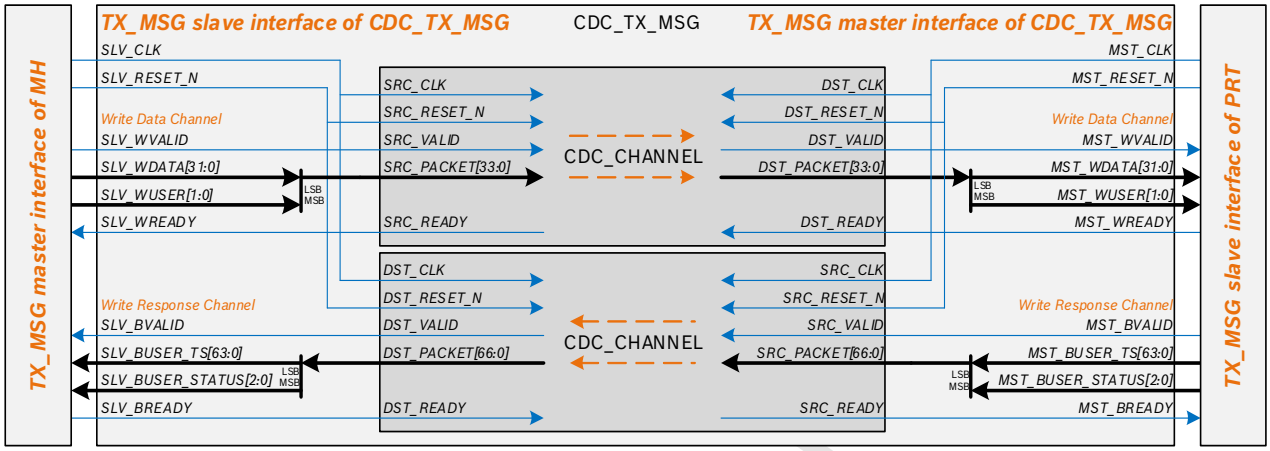


Figure 70. CDC_TX_MSG

The TX_MSG interface is driven by the Message Handler in the HOST clock domain and is connected to the PRT in the CAN clock domain.

The following figure shows the application of the CDC_CHANNEL for the clock domain crossing of the RX_MSG interface. This component is named CDC_RX_MSG.

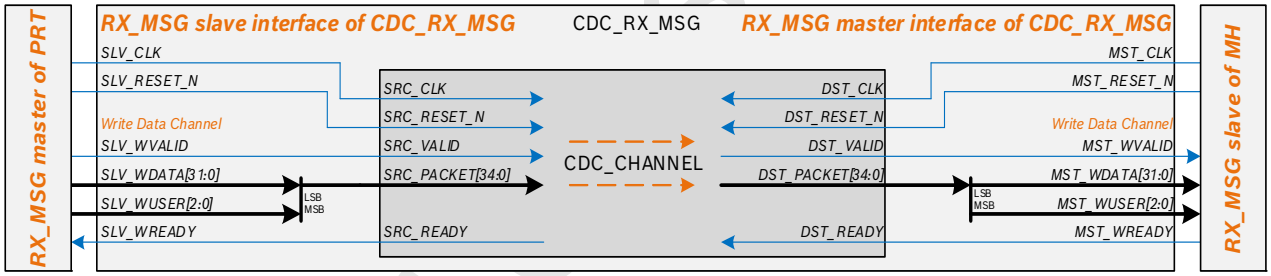


Figure 71. CDC_RX_MSG

The RX_MSG interface is driven by the PRT in the CAN clock domain and is connected to the Message Handler in the HOST clock domain.

The following section provides details about the embedded CDC component.

1.9.1.2.3.1 CDC Channel

The CDC_CHANNEL is a clock domain crossing component which transports a channel from a source clock domain (SRC) to a destination clock domain (DST).

A channel consists of a data packet (PACKET) and a two-way flow control (VALID and READY). The source signalizes a new packet to be transferred by asserting SRC_VALID=1 and keeps SRC_PACKET valid until the transfer occurs. The transfer occurred when SRC_VALID and SRC_READY are both one. This triggers an CDC_CHANNEL internal clock domain crossing and thus the packet is applied to the destination, i.e., DST_PACKET=SRC_PACKET and DST_VALID=1. The destination acknowledges the packet by asserting DST_READY. The transfer to the destination occurred when DST_VALID and DST_READY are both one.

A transfer at the source clock domain will always be propagated to the destination clock domain and can't be aborted. The CDC_CHANNEL could be used for channel based bus interfaces RX_MSG and TX_MSG.

1) Features
The component has the following features:

- Transport of one channel from a source clock domain to a destination clock domain
- The channel consists of a data packet (PACKET) and a two-way flow control (VALID and READY)

- The PACKET carries all data, sliced into one vector e.g., address, data, write-strobe, status
- The PACKET size is configurable by design parameter
- Double buffering (Source Packet Buffer, Destination Packet Buffer)
- The clock frequencies may have any ratio
- The clocks may have any phase relation

2) Block Diagram

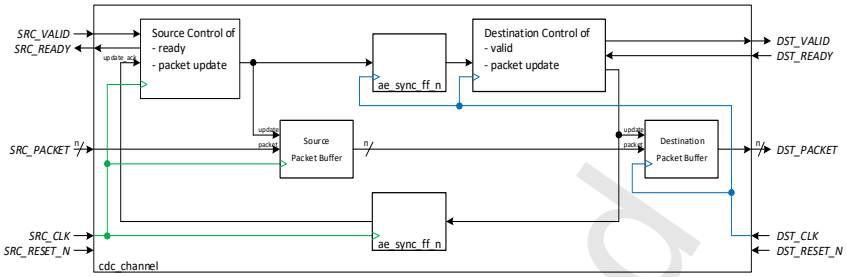


Figure 72. Block Diagram of CDC_CHANNEL

3) Hardware Interface

Not applicable.

4) Software Interface

Not applicable.

5) Functional Description

The CDC_CHANNEL is a clock domain crossing component which transports a channel from a source clock domain (SRC) to a destination clock domain (DST).

A channel consists of a data packet (PACKET) and a two-way flow control (VALID and READY). The CDC_CHANNEL signalizes SRC_READY=1 whenever Source Packet Buffer is empty. The source signalizes a new packet to be transferred by asserting SRC_VALID=1 together with a valid SRC_PACKET. The transfer into the Source Packet Buffer occurred when SRC_VALID and SRC_READY are both one.

When Source Packet Buffer contains new data, the CDC_CHANNEL starts an internal clock domain crossing into the destination clock domain. When done, the new data is stored into the Destination Packet Buffer as soon as the Destination Packet Buffer is empty. The storage event will be forwarded to the source clock domain in order to free-up the Source Packet Buffer, signalled by SRC_READY=1.

The Destination Packet Buffer asserts DST_VALID=1 whenever it contains new data. The data transfer to the destination occurred when destination signalizes DST_READY=1, i.e. DST_VALID and DST_READY are both one.

Due to the packet buffering in both clock domains the CDC_CHANNEL can store up to two transfers.
When both buffers are full, no more transfers are accepted from the source.

The following timing diagram shows an example.

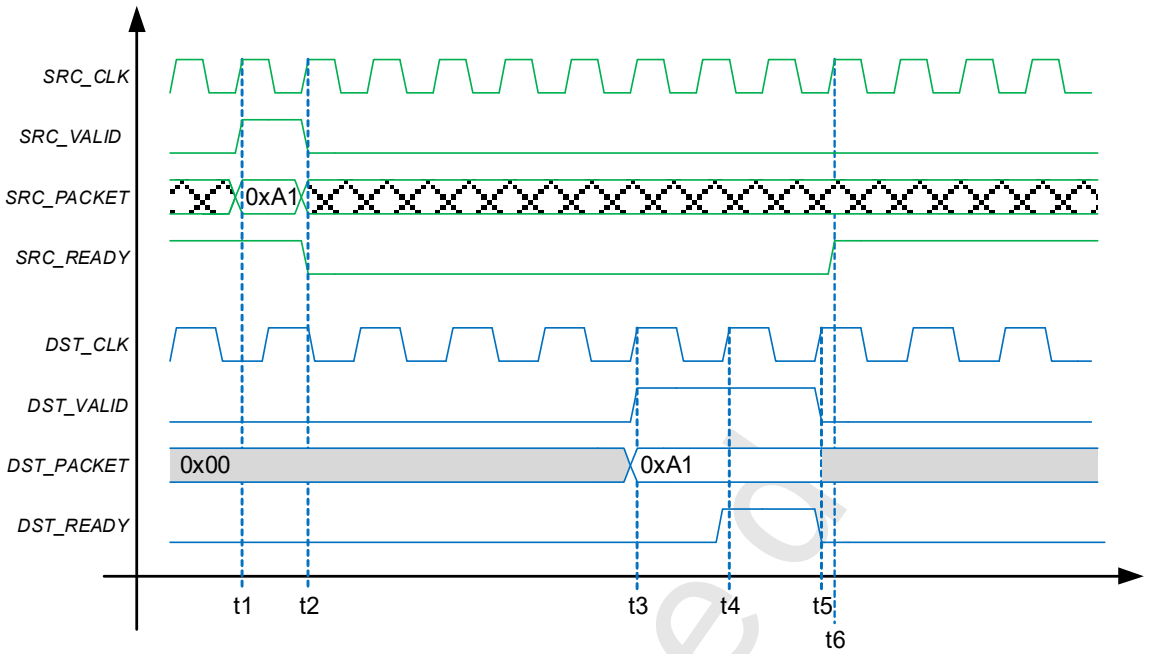


Figure 73. CDC_CHANNEL Timing Diagram

Note: Resets are not active (SRC_RESET_N=1, DST_RESET_N=1)
At t1 the access starts at the source clock domain by applying SRC_VALID and SRC_PACKET. Due to the fact that SRC_READY is already one, the transfer into Source Packet Buffer is done at t2.

At t3, the internal clock domain crossing into the Destination Packet Buffer has been finished and DST_VALID is set to one and DST_PACKET gets the value which SRC_PACKET had at t1. This data-handover will be signalized to the source clock domain to free-up the Source Packet Buffer, thus SRC_READY gets one at t5.

At t4 the destination rises DST_READY. One clock cycle later, at t5, DST_VALID is set to zero and the transfer at the destination clock domain is done. From now signal DST_PACKET is no longer valid, but stays unchanged until new data is available, which can happen at any time.

In the following some more accesses are shown. The clock frequency ratio (SRC:DST) for the example below is 1:4.

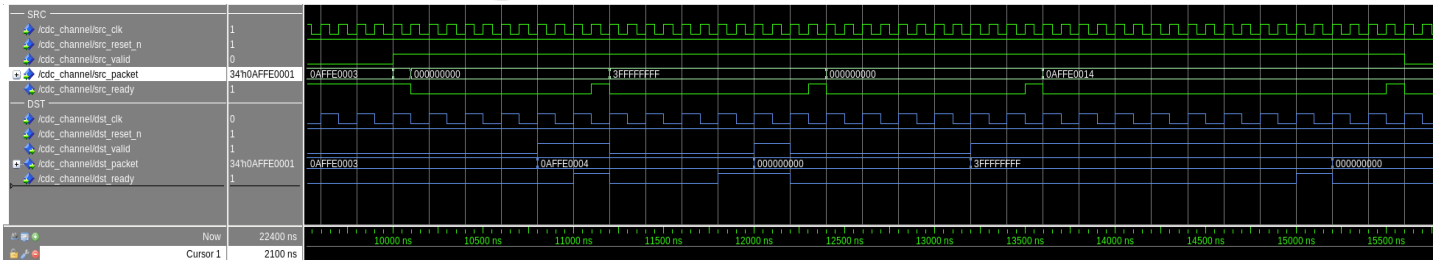


Figure 74. Waveform: Clock Frequency 1:4

The diagram shows five transfers at the source interface, where SRC_PACKET has the following values: 0x0AFFE004, 0x0, 0x3FFFFFFF, 0x0 and 0x0AFFE0014. The source provides data back-to-back, so signal SRC_VALID stays active while that access sequence. Every time a new transfer has been forwarded to the Destination Packet Buffer the source interface is ready for a new transfer (SRC_READY gets one).

At the destination interface, the acknowledge from the destination (DST_READY) will be varied.
At the first transfer DST_READY is set to one a single clock cycle after DST_VALID is set to one.
At the second transfer DST_READY is already set when DST_VALID is set to one.
At the third transfer DST_READY is set many clock cycles after DST_VALID. During this time the clock domain crossing of a new transfer has already been completed, i.e. the new packet is available in the destination clock domain for the

Destination Packet Buffer. Thus signal DST_VALID stays active at one when DST_READY is set, because one clock cycle later the new DST_PACKET is applied to the destination interface.

Note: The timing diagram doesn't show the 5th transfer at the destination interface.

6) Safety Mechanisms

None.

7) Application Information

The design is able to work with any clock frequencies of source clock and destination clock.
The clocks frequencies may have any ratio.
The two clocks may be synchronous or asynchronous.
The active clock edge of the design is the rising edge.
The reset signals SRC_RESET_N and DST_RESET_N are asynchronous resets, active-low.
The reset inputs of both clock domains have to be asserted (set to low) at the same time, and each deasserted (set to high) synchronously to the dedicated domain clock.

Signal SRC_READY is set by default (ready before valid).
A second transfer at the source interface is acknowledged as soon as a prior transfer is forwarded to the Destination Packet Buffer, i.e., independent of the transfer at the destination interface.

If a first transfer is pending at the destination clock domain and a second transfer is accepted at the source interface, the CDC_CHANNEL will not accept any further transfers.

8) Verification and Validation Requirements

It is recommended to do a consistency check or lint check.
The design has to be checked by a clock domain crossing check tool.
The design shall be simulated in the target design environment to check that it fulfills all application needs.

9) Detailed Design Information

The following figure shows the CDC_CHANNEL design in detail.

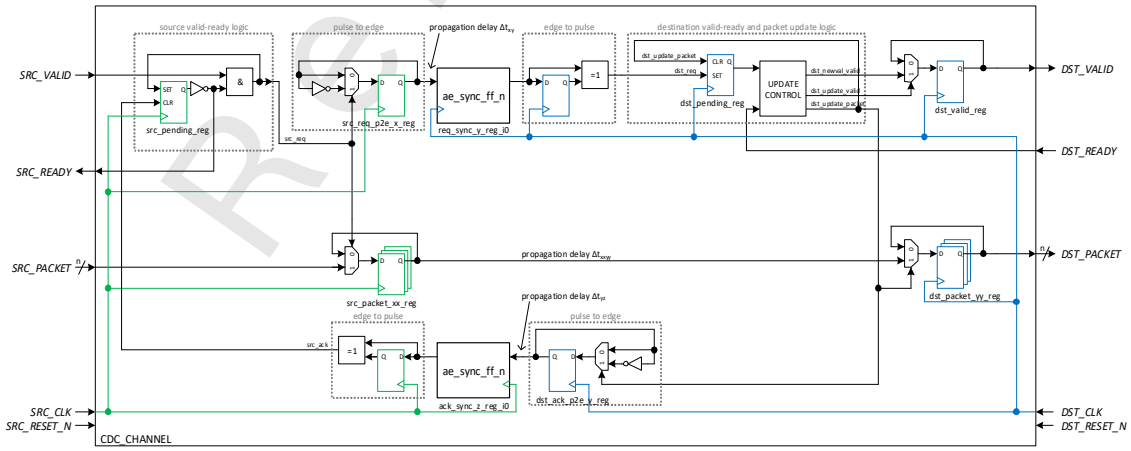


Figure 75. CDC_CHANNEL Flow Design

a) Port Description

Table 139. CDC Channel Port Description (page 1 of 2)

Signal	Bit Width	I/O	Description	Active Level	Clock Domain
SRC_CLK	1	I	Clock for SRC clock domain	rising edge	SRC
SRC_RESET_N	1	I	Asynchronous reset for SRC clock domain	low	SRC
SRC_VALID	1	I	Source valid	high	SRC

Table 139. CDC Channel Port Description (page 2 of 2)

Signal	Bit Width	I/O	Description	Active Level	Clock Domain
SRC_READY	1	O	Source ready	high	SRC
SRC_PACKET	PACKET_WIDTH_G	I	Source data packet	na	SRC
DST_CLK	1	I	Clock for DST clock domain	rising edge	DST
DST_RESET_N	1	I	Asynchronous reset for DST clock domain	low	DST
DST_VALID	1	O	Destination valid	high	DST
DST_READY	1	I	Destination ready	high	DST
DST_PACKET	PACKET_WIDTH_G	O	Destination data packet	na	DST

b) Parameters

Table 140. CDC Channel Parameters

Parameter name	Default Value	Description/Constraints
SYNC_FF_COUNT_G	2	number of synchronization flip-flops
PACKET_WIDTH_G	34	Width of SRC_PACKET and DST_PACKET

c) Memory Needs
Not applicable.

d) Design Dimensioning Details

Flip-Flop count of the design: 7 + 2 * PACKET_WIDTH_G + 2 * SYNC_FF_COUNT_G
The generic parameter PACKET_WIDTH_G is the width of signal SRC_PACKET and signal DST_PACKET. Corresponding to this the flip-flop count scales to buffer these signals.

With the generic parameter SYNC_FF_COUNT_G the number of synchronization flip-flops can be defined. Depending to the used ASIC technology and the mean time between failure requirements the number of synchronizer flip-flops can be adjusted.

The latency between the transfer at source interface (SRC_VALID=1 and SRC_READY=1) and data available at destination interface (DST_VALID=1) is:
(1 * SRC_CLK period) + ((SYNC_FF_COUNT_G + (+0/-1) + 2) * DST_CLK period)

The latency until an acknowledge for the successful transfer of a channel from the source clock domain to the destination clock domain (DST_VALID=1 and corresponding DST_PACKET) is transferred back to the source clock domain and clearing the pending flag and SRC_READY getting one is:
(SYNC_FF_COUNT_G + (+0/-1) + 2) * SRC_CLK period

Note: In the formulas above the expression (+0/-1) is used to consider the uncertainty in the delay of the synchronization stage, which depends on the clock phases between SRC_CLK and DST_CLK.

e) Test Concept
Not applicable.

10) Integration Guidelines

The design includes the block AE_SYNC_FF_N. This block includes the synchronization flip-flops. The block can be replaced by the semiconductor vendor by its own synchronization flip-flop component.

a) False Path Constraints
With reference to the figure above the timing of the paths

- From register output SRC_REQ_P2E_X_REG/Q

- To register data input REQ_SYNC_Y_REG_I0/SYNC_REG(0)/D and
- From register output DST_ACK_P2E_Y_REG/Q
- To register data input ACK_SYNC_Z_REG_I0/SYNC_REG(0)/D

are not critical for setup time checks. Therefore, a false path constraint for synthesis to this flip-flops is recommended.

Following false paths are recommended:
set_false_path -from [find cell {*x_reg_reg} -hierarchy]
- to [find cell {*sync_y_reg_i0/sync_reg_reg(0)} -hierarchy]
set_false_path -from [find cell {*y_reg_reg} -hierarchy]
- to [find cell {*sync_z_reg_i0/sync_reg_reg(0)} -hierarchy]

b) Multi Cycle Path Constraints
From the figure above it can be derived that the path delay from the registers SRC_PACKET_XX_REG of the source clock domain to the registers DST_PACKET_YY_REG of the destination clock domain shall not exceed the limit of three times the destination clock period (delay of SRC_REQ_P2E_X_REG to DST_PENDING_REG) minus the setup time of the destination register. Therefore, it is recommended to constraint for synthesis a multi cycle path of 3 for setup time related to the destination clock.

Following multi cycle paths are recommended:
for bus signals synchronized from src_clk domain to dst_clk domain:
set_multicycle_path 3 -setup -end
- from [find cell {*xx_reg_reg[*]} -hierarchy]
- to [find cell {*yy_reg_reg[*]} -hierarchy]

set_multicycle_path 1 -hold -start
-from [find cell {*xx_reg_reg[*]} -hierarchy]
-to [find cell {*yy_reg_reg[*]} -hierarchy]

c) Maximum Delay Constraints
To avoid a potential metastable state at the synchronizer flip-flops (SYNC_REG(0), ..., SYNC_REG(sync_ff_count_g-1)), the paths between two synchronizer flip-flops should be as short as possible (i.e., two flip-flops should be placed together as close as possible). At least for ASIC backend flow and FPGA synthesis tools it is recommended to specify additionally a maximum delay on dedicated signal paths.

As a result, it is recommended to constraint a maximum path delay from flip-flop SYNC_REG(0) to SYNC_REG(1) and probably for all further synchronizer flip-flop pairs e.g., SYNC_REG(1) to SYNC_REG(2) and so on.

The propagation delay on the domain crossing paths should also be constrained.
It is recommended that the maximum propagation delay does not exceeds the value of one destination clock period minus the setup time of destination Flip-flop on the following paths:

- From Flip-flop SRC_REQ_P2E_X_REG to Flip-Flop REQ_SYNC_Y_REG_I/SYNC_REG(0)
- (propagation delay Δtxy)
- From Flip-flop DST_ACK_P2E_Y_REG to FlipFlop ACK_SYNC_Z_REG_I/SYNC_REG(0)
- (propagation delay Δtyz)

The maximum propagation delay of three destination clock periods minus the setup time of destination FlipFlop shall not be exceeded on the following paths:

- From FlipFlops SRC_PACKET_XX_REG to FlipFlops DST_PACKET_YY_REG
- (propagation delay Δtxxyy)

d) Critical Combinational Logic
One critical design element is the multiplexer in front of the flip-flops of DST_PACKET_YY_REG. Because of the fact that some data of the multiplexer come from another clock domain (SRC_PACKET_XX_REG), it is in theory possible that depending on the design of the instantiated/implemented multiplexer a toggling input signal causes a toggling at the multiplexer output. If this toggling at the output occurs near the point in time of the clock edge of the flip-flops DST_PACKET_YY_REG, these flip-flops can fall into a metastable state or switch to an undesired output value.

To avoid potential problems, it is recommended to check in the backend flow that the instantiated multiplexer circuit in front of the destination clock domain flip-flops DST_PACKET_YY_REG is a glitch free design. The multiplexer feeding the DST_PACKET_YY_REG flip-flop is a user-replaceable module, allowing the IP customer to substitute it with their own multiplexer component.

1.9.1.2.4 CDC for Timebase

The PRT provides the MH a 64 bit timestamp for each transmitted or received message. The PRT uses the module CDC_TIMEBASE to capture the time from a SoC timebase. The SoC timebase may be in a separate clock domain.

For timestamping, the PRT tolerates a certain latency, i.e., the time between a request of the PRT for a new timestamp until the new timestamp value must be applied to the PRT. The following table shows the verified clock period ratios between CAN and TIMEBASE clock domain together with the resulting latencies of the CDC_TIMEBASE. Depending on the phase between CAN and TIMEBASE, the latency can be less, which is not critical.

Note: CDC_TIMEBASE parameter SYNC_FF_COUNT_G=2.
The first column is the ratio of the clock period between CAN and TIMEBASE clock domain.
The column T_CAPTURE shows the latency of a capture request in the CAN clock domain until the timestamp value is captured in the TIMEBASE clock domain.
The column T_ PROPAGATE shows the latency until the timestamp value in the TIMEBASE clock domain is propagated to the CAN clock domain.
The column T_TOTAL shows the latency overall, i.e., the latency of a capture request until the TIMESTAMP gets valid.

Table 141. CDC Timebase Latency

Clock Period CAN:TIMEBASE	T_CAPTURE [CAN cycles]	T_PROPAGATE [CAN cycles]	T_TOTAL [CAN cycles]
1:2	7	3	10
1:1	4	3	7
2:1	2,5	3	5,5
4:1	1,75	3	4,75

The following section provides details about the embedded CDC component.

1.9.1.2.4.1 CDC_TIMEBASE

The CDC_TIMEBASE design is a clock domain crossing component. It synchronizes an incoming capture request pulse at the CAN clock domain to the TIMEBASE clock domain where it is used to capture a timebase value. This timebase value snapshot alias timestamp is transferred back to the CAN clock domain.

1) Features

- The component has the following features: transfer of capture request from CAN clock domain to TIMEBASE clock domain
- Transfer of captured timebase value from TIMEBASE clock domain to CAN clock domain
- Valid signal to show if the timestamp at the CAN clock domain is valid and if a new capture may be requested
- The clock frequencies may have any ratio
- The clocks may have any phase relation

2) Block Diagram

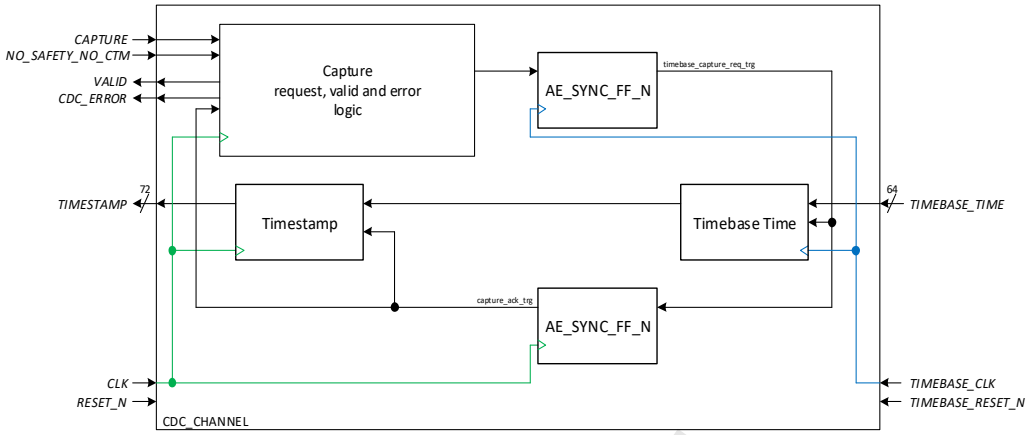


Figure 76. Block Diagram of CDC_TIMEBASE

3) Hardware Interface

Not applicable.

4) Software Interface

Not applicable.

5) Functional Description

The design transfers a capture request pulses from the CAN clock domain to the TIMEBASE clock domain. A pulse is defined as a sequence of '0'–'1'–'0'. The capture request signal is CAPTURE.

At the TIMEBASE clock domain the capture request pulse triggers a capturing of the timebase value from signal TIMEBASE_TIME. After that the captured timebase value is transferred to the CAN clock domain. If done, the signal VALID gets one, indicating that signal TIMESTAMP is now valid.

The signal VALID stays one until a new capture request, i.e. high-pulse at signal CAPTURE.
A capturing should only be requested while the CDC_TIMEBASE is idle, indicated by VALID=1. If a capturing is request while the CDC_TIMEBASE is occupied to fulfill a previous request, the error event signal CDC_ERROR is set for one clock cycle.

The following timing diagrams give some examples how CDC_TIMEBASE works.
Setup for the next timing diagrams:

- Clock Frequency ratio (SRCDST):1:4

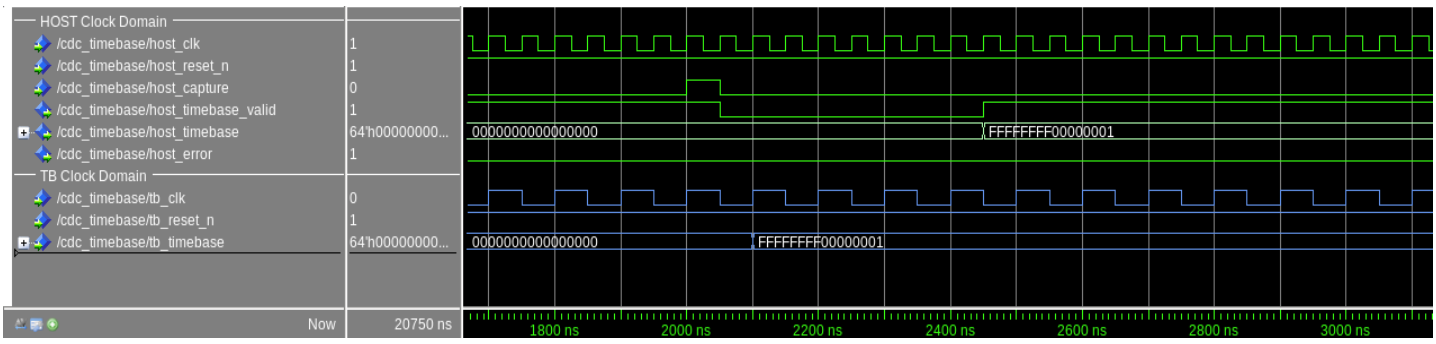


Figure 77. Waveform: Single CAPTURE Request

The timing diagram above shows a capture request at signal CAPTURE. One CLK cycle later the VALID signal is cleared ('0') to show that a new request should not be done and that the TIMESTAMP value might change any time and should not evaluated any more.

The capture process is finished when the timebase value TIMESTAMP is updated and signal VALID is set. Now a new capture request can be done.

The next timing diagram shows the case if a new capture request is done while a previous request isn't finished. The second capture request is done at 6900ns (signal CAPTURE is set for one clock cycle) while VALID is zero. At time 7000ns signal CDC_ERROR is set. (Screen-shot is not up to date, now a pulse is generated.)

It can be seen that the previous capture process is finished one clock cycle later.
A running capture process is not disturbed by a further capture request while signal VALID is zero. If signal VALID is zero and a capture request (CAPTURE) arrives it is undefined if this new capture request is executed.

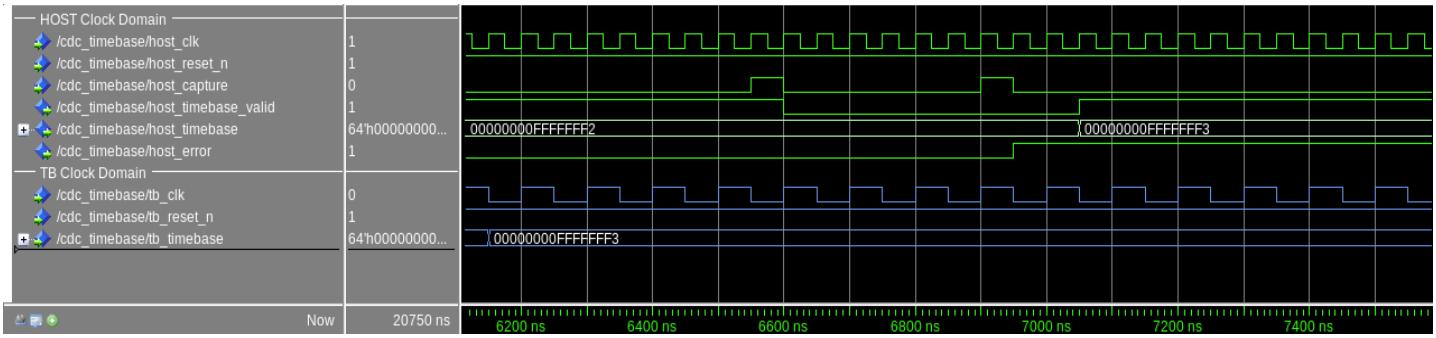


Figure 78. Waveform: Multiple CAPTURE Requests

6) Safety Mechanisms

None.

7) Application Information

The design is able to work with any clock frequencies of CAN clock (CLK) and Timebase clock (TIMEBASE_CLK).
The clocks frequencies may have any ratio.
The two clocks may be synchronous or asynchronous.
The active clock edge of the design is the rising edge.
The reset signals RESET_N and TIMEBASE_RESET_N are asynchronous resets.
The reset inputs of both clock domains have to be asserted (set to low) at the same time, and each deasserted (set to high) synchronously to the dedicated domain clock.

It is mandatory that signal CAPTURE is a single pulse. This means it shall be always a '0'–'1'–'0' sequence.
It is strongly recommended that signal CAPTURE is not set while signal VALID is zero. Otherwise, the corresponding capture request could be lost.

It is mandatory that signal TIMESTAMP is not evaluated if signal VALID is zero. Otherwise, a wrong value can be read.

If signal CAPTURE is set while signal VALID is zero, then event signal CDC_ERROR is set for one clock cycle. The usage of signal CDC_ERROR is optional.

8) Verification and Validation Requirements

It is recommended to do a consistency check or lint check.
The design has to be checked by a clock domain crossing check tool.
The design shall be simulated in the target design environment to check that it fulfills all application needs.

9) Detailed Design Information

The following figure shows the CDC_TIMEBASE design in detail.

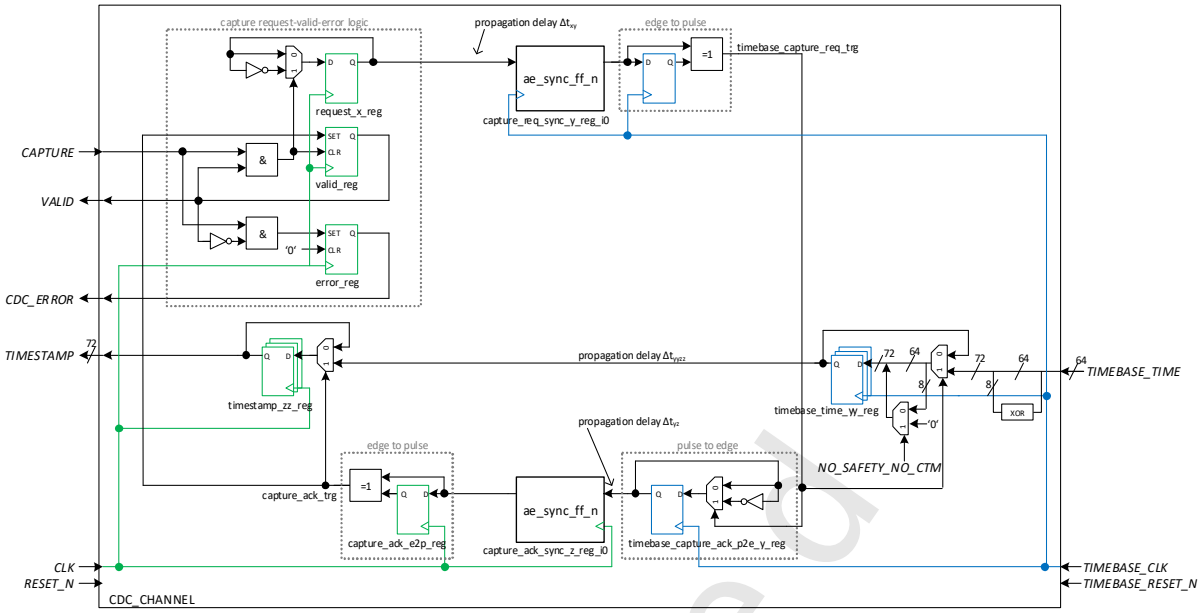


Figure 79. CDC_TIMEBASE Flow Design

a) Port Description

Table 142. CDC Timebase Port Description

Signal	Bit Width	I/O	Description	Active Level	Clock Domain
CLK	1	I	Clock for CAN clock domain	rising edge	CAN
RESET_N	1	I	Asynchronous reset for CAN clock domain	low	CAN
CAPTURE	1	I	Pulse to trigger capturing of timebase	high-pulse	CAN
NO_SAFETY_NO_CTM	1	I	When it is set to 1, Time Stamp parity generation is disabled in CDC_TIMEBASE	high	CAN
TIMESTAMP	72	O	Captured time	na	CAN
VALID	1	O	TIMESTAMP is valid	high	CAN
CDC_ERROR	1	O	Error signaling: New campture request while CDC is busy	high-pulse	CAN
TIMEBASE_CLK	1	I	Clock for TIMEBASE clock domain	rising edge	TIMEBASE
TIMEBASE_RESET_N	1	I	Asynchronous reset for TIMEBASE clock domain	low	TIMEBASE
TIMEBASE_TIME	64	I	Time vector provided by SoC timebase	na	TIMEBASE

b) Parameters

Table 143. CDC Timebase Parameters

Parameter name	Default Value	Description/Constraints
SYNC_FF_COUNT_G	2	number of synchronization flip-flops

c) Memory Needs
Not applicable.

d) Design Dimensioning Details
Flip-flop count of the design: 6 + 2 * 64 (timebase width) + 2 * SYNC_FF_COUNT_G
With the generic parameter SYNC_FF_COUNT_G the number of synchronization flip-flops can be defined. Depending to the used ASIC technology and the mean time between failure requirements the number of synchronizer flip-flops can be adjusted.

The latency of a capture request in the CAN clock domain until a timebase value is captured in the TIMEBASE clock domain is:
1 * CLK period + (SYNC_FF_COUNT_G + (+0/-1) + 1) * TIMEBASE_CLK period

The latency until the captured timebase value in the TIMEBASE clock domain is transferred to the CAN clock domain is:
(SYNC_FF_COUNT_G + (+0/-1) + 1) * CLK period

Note: In the formulas above the expression (+0/-1) is used to consider the uncertainty in the delay of the synchronization stage, which depends on the clock phases between CLK and TIMEBASE_CLK.

e) Test Concept
Not applicable.

10) Integration Guidelines
The design includes the block AE_SYNC_FF_N. This block includes the synchronization flip-flops. The block can be replaced by the IP customer by its own synchronization flip-flop component.

a) False Path Constraints
With reference to the figure above the timing of the paths

- From register output REQUEST_X_REG/Q
- To register data input CAPTURE_REQ_SYNC_Y_REG_I0/SYNC_REG(0)/D and
- From register output TIMEBASE_CAPTURE_ACK_P2E_Y_REG/Q
- To register data input CAPTURE_ACK_SYNC_Z_REG_I0/SYNC_REG(0)/D

are not critical for setup time checks. Therefore, a false path constraint for synthesis to this flip-flops is recommended.

Following false paths are recommended:
set_false_path -from [find cell {*x_reg_reg} -hierarchy]
- to [find cell {*sync_y_reg_i0/sync_reg_reg(0)} -hierarchy]
set_false_path -from [find cell {*y_reg_reg} -hierarchy]
- to [find cell {*sync_z_reg_i0/sync_reg_reg(0)} -hierarchy]

b) Multi Cycle Path Constraints
From the figure above it can be derived that the path delay from the bus register TIMEBASE_TIME_YY_REG of the TIMEBASE clock domain to the bus register TIMESTAMP_ZZ_REG of the CAN clock domain shall not exceed the limit of two times the CAN clock period (delay of TIMEBASE_CAPTURE_ACK_P2E_Y_REG through the synchronizer instance CAPTURE_ACK_SYNC_Z_REG_I0) minus the setup time of a synchronizer flip-flop. Therefore, it is recommended to constraint for synthesis a multi cycle path of 2 for setup time related to the CAN clock.

Following multi cycle paths are recommended:
for bus signals synchronized from timebase_clk clock domain to clk clock domain:
set_multicycle_path 2 -setup -end
- from [find cell {*yy_reg_reg[*]} -hierarchy]
- to [find cell {*zz_reg_reg[*]} -hierarchy]
set_multicycle_path 1 -hold -start
- from [find cell {*yy_reg_reg[*]} -hierarchy]
- to [find cell {*zz_reg_reg[*]} -hierarchy]

c) Maximum Delay Constraints
To avoid a potential metastable state at the synchronizer flip-flops (SYNC_REG(0), ..., SYNC_REG(sync_ff_count_g-1)), the paths between two synchronizer flip-flops should be as short as possible (i.e., two flip-flops should be placed together as close as possible). At least for ASIC back-end flow and FPGA synthesis tools it is recommended to specify additionally a maximum delay on dedicated signal paths.
As a result, it is recommended to constraint a maximum path delay from flip-flop SYNC_REG(0) to SYNC_REG(1) and probably for all further synchronizer flip-flop pairs e.g., SYNC_REG(1) to SYNC_REG(2) and so on.
The propagation delay on the domain crossing paths should also be constrained.
It is recommended that the maximum propagation delay does not exceeds the value of one destination clock period minus the setup time of destination flip-flop on the following paths:

- From Flip-flop REQUEST_X_REG
- To Flip-flop CAPTURE_REQ_SYNC_Y_REG_I0/SYNC_REG(0)
- (propagation delay Δtxy)
- From flip-flop TIMEBASE_CAPTURE_ACK_P2E_Y_REG
- To flip-flop CAPTURE_ACK_SYNC_Z_REG_I0/SYNC_REG(0)
- (propagation delay Δtyz)

The maximum propagation delay of three CAN clock periods minus the setup time of destination flip-flops shall not be exceeded on the following paths:

- From Flip-flops TIMEBASE_TIME_YY_REG to Flip-flops TIMESTMAP_ZZ_REG
- (propagation delay Δtxxyy)

e) Critical Combinational Logic
One critical design element is the multiplexer in front of the flip-flops of TIMESTAMP_ZZ_REG. Because of the fact that one input signal of the multiplexer comes from another clock domain (TIMEBASE_TIME_YY_REG), it is in theory possible that depending on the design of the instantiated/implemented multiplexer a toggling input signal causes a toggling at the multiplexer output.

If this toggling at the output occurs near the point in time of the clock edge of the flip-flops TIMESTAMP_ZZ_REG, these flip-flops can fall into a metastable state or switch to an undesired output value.

To avoid potential problems, it is recommended to check in the back-end flow that the instantiated multiplexer circuit in front of the CAN clock domain flip-flops TIMESTAMP_ZZ_REG is a glitch free design. The multiplexer feeding the TIMESTAMP_ZZ_REG flip-flop is a user-replaceable module, allowing the IP customer to substitute it with their own multiplexer component.

1.9.2 Resets

Each clock domain has its own single reset input (xxx_RESET_N). The XS_CAN does not embed reset synchronizers, i.e., the reset inputs are directly connected to the asynchronous reset inputs of each Flip Flop located in the dedicated clock domain.

The XS_CAN supports only a global reset over all clock domains. Thus, the reset inputs of all domains have to be asserted (set to low) at the same time, and each has to be de-asserted (set to high) synchronously to the dedicated domain clock.
The following figure shows the recommended reset circuit.

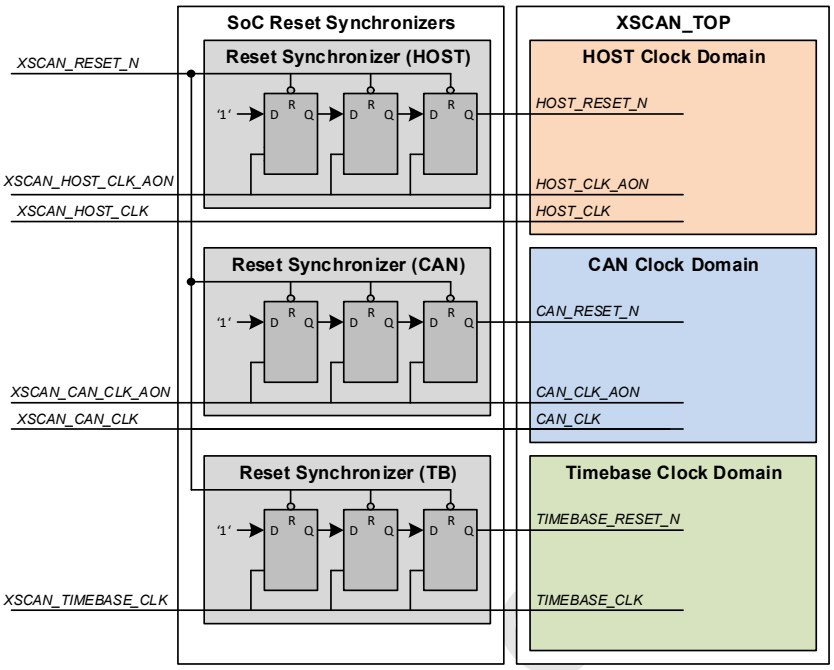


Figure 80. XS_CAN_TOP Reset Circuit

Following is the waveform for the reset circuit after synchronization:

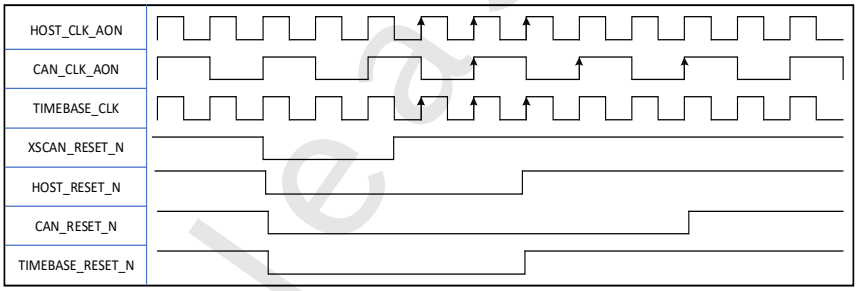


Figure 81. Reset Synchronization Waveform

1.9.2.1 Behavior While Reset Active

Here below is the XS_CAN behavior when its resets are active, or its clocks are off:
During reset(HOST_RESET_N=0, CAN_RESET_N=0), HOST_AHB_HREADYOUT is asserted high by the IP so that bus does not get hung. Accessing the IP during reset is not recommended and accessing the IP when the clocks are off result in hangup.

1.10 Application Information

1.10.1 Bit Rate and Performance

The bit rates (nominal bit rate, FD Data bit rate, XL Data bit rate) supported by the XS_CAN IP depend on the system parameters: host clock frequency, CAN clock frequency, number of RX filter elements, latency to the memories LMEM and SMEM, and number of XS_CANs connected to the same LMEM.
An excel-sheet [4] is provided with the XS_CAN IP to check if a specific combination of bit rates (nominal bit rate, FD Data bit rate, XL Data bit rate) and SMEM/LMEM latencies is functional under specific system parameters. Here below is an extract of a configuration set in the excel-sheet [4].

The XS_CAN IP module is set to support:
- Nominal bit rate: 0.8Mbps
- FD Data bit rate: 8Mbps
- XL Data bit rate: 20Mbps (CAN Clk Frequency 160MHz)

• RX Filter elements used: Up to 127 filter element configurations, with 1 or 2 comparisons each
• LMEM shared by 4 XS_CANs

Host clock frequency has dependency on the type of bust transfers used to access LMEM while message reception is ongoing. Refer the excel sheet for the minimum host clock frequency required for each type of burst to ensure that received frame is not dropped.

1.10.2 Time Stamping Offset

Due to internal propagation delays for the timestamping in the Protocol Controller (1 CAN clock cycle) and the clock domain crossing for the time capturing (n clock cycles, depending on the clock frequency ratio between the CAN and the TIMBASE domain) a certain offset has to be considered to derive the correct time.

These are the formulas for the offset:
TS_offset_max = 2 CAN_CLK period + 3 TIMEBASE_CLK period
TS_offset_min = 2 CAN_CLK period + 2 TIMEBASE_CLK period

This is the formula for the corrected timestamp:
Corrected Timestamp = Timestamp in RX message/ TX Event fifo queue- TS_offset

The following table shows offsets for typical clock frequencies:
The first two columns contain the clock frequencies of the CAN_CLK and the TIMEBASE_CLK, measured in MHz. The third and fourth column contain the minimum and maximum TS_offset of the captured timestamp in multiple of TIMEBASE_CLK cycles, for TX and RX. The last two columns show the interpretation of the TS_offset in nano seconds.

Table 144. Time Stamping Offset

CAN_CLK [MHz]	TIMEBASE_CLK [MHz]	TS_offset_min [TIMEBASE_CLK cycles]	TS_offset_max [TIMEBASE_CLK cycles]	TS_offset_min [ns]	TS_offset_max [ns]
80	80	4	5	50,00	62,50
80	160	6	7	37,50	43,75
80	240	8	9	33,33	37,50
80	320	10	11	31,25	34,38
160	80	3	4	37,50	50,00
160	160	4	5	25,00	31,25
160	320	6	7	18,75	21,88

1.10.3 Cut Through Mode with and without CTDMA add on module

Case 1: CTDMA integrated
XS_CAN must be used with CTDMA Add-on module if Cut Through Mode is enabled. All LMEM accesses from the host are to be done through AHB slave interface of CTDMA and the block copy transfers are performed by CTDMA.

Case 2a: CTDMA not integrated and XS_CAN IP used for normal operations.
XS_CAN IP operation in cut through mode without CTDMA is not productive enough and hence not recommended.

Case 2b:CTDMA not integrated and XS_CAN IP used for test purpose.
Block copy transfers in XS_CAN IP can be performed in CTM without CTDMA add on module with the following information:
i) Source and Destination addresses for block copy transfers have to be read from MH.CTM_DESC_SRC and MH.CTM_DESC_DEST registers
ii) Block copy events are to be read from MH.CTM_EVENT register or based on interrupts IRC
iii) *CTM_RESP* input to XS_CAN IP must be connected to a logic that can stimulate this input the same way as in CTDMA.
iv) CTM_BC_ERR output from XS_CAN IP must be connected to a logic to detect the cancellation of block copy by XSC_AN IP.

1.10.4 AHB burst length usage recommendation

This section gives information regarding the usage of bursts at AHB Slave interface of XS_CAN

1.10.4.1 XS_CAN Integrated without CTDMA

When XS_CAN is integrated without CTDMA, the use of AHB burst transfers on the host interface is not recommended. XS_CAN treats each burst transaction as an atomic operation. In access scenarios involving a high number of burst transfers, this behavior can reduce the number of LMEM access opportunities available to XS_CAN for executing its internal functions. System integrators must determine the minimum required HOST_CLK frequency using the excel sheet [4] to ensure correct XS_CAN functionality under the expected access patterns.

1.10.4.2 XS_CAN Integrated with CTDMA

When XS_CAN is integrated together with CTDMA, the use of AHB burst transfers is intended.The CTDMA throttling mechanism ensures that XS_CAN is always provided with sufficient LMEM bandwidth to execute its essential internal functions, even in the presence of burst-based host traffic. System integrators must determine the minimum required HOST_CLK frequency using the excel sheet [4].

1.10.5 Loopback Mode

Loopback mode is a feature to test the full transmit and receive path of one single XS_CAN. Section 1.6.6.15 in PRT describes the Loopback mode configuration and behavior in the PRT.The MH does not need a special configuration. The table below shows the configurations for which loopback is supported.

Table 145. Loopback Configuration

CAN messages payload size	FMM	CTM
<= 64 bytes	Loopback supported	Loopback supported
> 64 bytes	Loopback supported	Loopback NOT supported

Loopback mode is not supported for CAN XL frames with payloads greater than 64 bytes in CTM, as both the transmit and receive paths would require the use of cut-through buffers.

1.10.6 Supported CAN Variants in FMM and CTM

The following table gives an overview on which CAN variants are supported in which operation mode of the XS_CAN.

Table 146. Can Variants (page 1 of 2)

CAN Variant	FMM	CTM
CAN CC	Yes	Yes
CAN FD	Yes	Yes

Table 146. Can Variants (page 2 of 2)

CAN Variant	FMM	CTM
CAN XL	Yes	Yes
CAN FD light without FLCHB mode (PRT.MODE.LCHB=0)	Yes	Yes
CAN FD light with FLCHB mode (PRT.MODE.LCHB=1)	Yes	No

FLCHB means FD light commander high bit rate mode and is specified in CiA 604-4. In the XS_CAN this mode can be enabled/disabled by software via the configuration bit PRT.MODE.LCHB. The only CAN variant that is not supported in CTM is FD light in FLCHB mode. This is because, in FLCHB mode the length of a frame (in time) can be nearly 8 times shorter than without FLCHB mode. Due to this the block copies cannot be finished in time. Frames in FLCHB mode are much shorter in time, because all bits are sent with high bit rate of e.g. 8 Mbit/s, and not just the bits of the data phase.

1.11 Programming Guidelines

1.11.1 Start Operation

Steps to program and start the IP.

1. Ensure that all the clocks are up and running and the resets are de-asserted. HOST_CLK_AON must be turned ON for configuring MH and IRC registers and CAN_CLK_AON must be turned ON for configuring PRT registers. Wait for at least 20 clock cycles (HOST_CLK_AON) after reset before read write to MH and PRT registers.
2. Configure the registers in Message Handler, PRT and IRC. Refer Chapter Software Interface in MH,PRT and IRC sections in [11] for details.
3. Enable MH and start PRT. Enqueue messages only after PRT is started.

1.11.2 Stop Operation

Steps to stop the IP.

1. Stop PRT and MH by writing to registers. Details on how to stop PRT is given in section 1.6.6.5 Starting and Stopping the Module in [11]. MH can be stopped by writing to register MH_CTRL.MH_EN.

1.11.3 Power Down Mode

XS_CAN IP can be brought to power down state by stopping the clocks HOST_CLK, CAN_CLK and TIMEBASE_CLK externally from the system. There is no internal mechanism to do this in the IP. The clocks HOST_CLK_AON and CAN_CLK_AON are always ON clocks and should not be turned off during the operation of the IP.
The user must proceed in the following way
Step 1: Stop the IP by stopping the PRT and MH. This is done by writing to registers of the XS_CAN.
Step 2: Disable the clocks: HOST_CLK, CAN_CLK, TIMEBASE_CLK

1.12 Detailed Design Information

1.12.1 Design Dimensioning Information

1.12.2 Port Description

Table 147. Port List (page 1 of 3)

Signal	Bit Width	I/O	Description	Active Level	Clock Domain
CLOCKS					
HOST_CLK	1	I	Host Clock	rising edge	HOST_CLK
HOST_CLK_AON	1	I	Always ON Host Clock	rising edge	na
CAN_CLK	1	I	CAN Clock	rising edge	na
CAN_CLK_AON	1	I	Always ON CAN Clock	rising edge	na
TIMEBASE_CLK	1	I	Timebase Clock	rising edge	TIMEBASE_CLK
RESETS					
HOST_RESET_N	1	I	Asynchronous Top Level Reset	Low	HOST_CLK
CAN_RESET_N	1	I	Asynchronous Reset for PRT	Low	CAN_CLK
TIMEBASE_RESET_N	1	I	Asynchronous Reset for Timebase	Low	TIMEBASE_CLK
HOST AHB SLAVE INTERFACE					
HOST_AHB_HADDR	19	I	Address	na	HOST_CLK
HOST_AHB_HSELX	1	I	Select	High	HOST_CLK
HOST_AHB_HTRANS	2	I	Transfer type	na	HOST_CLK
HOST_AHB_HBURST	3	I	Burst type	na	HOST_CLK
HOST_AHB_HMASTLOCK	1	I	Master lock	na	HOST_CLK
HOST_AHB_HSIZE	3	I	Burst size	na	HOST_CLK
HOST_AHB_HWRITE	1	I	Read or write bit	na	HOST_CLK
HOST_AHB_HPROT	4	I	Protection type	na	HOST_CLK
HOST_AHB_HWDATA	32	I	Write data	na	HOST_CLK
HOST_AHB_HRDATA	32	O	Read data	na	HOST_CLK
HOST_AHB_HREADYOUT	1	O	Ready	High	HOST_CLK
HOST_AHB_HRESP	2	O	Response	na	HOST_CLK
DMA REQUEST SIGNALS					
TXFQ_WREQ	1	O	TX FIFO Queue write request from VBM	High	HOST_CLK
TXPQ_WREQ	1	O	TX Priority Queue write request from VBM	High	HOST_CLK
TXEF_RREQ	1	O	TX Event FIFO Queue read request from VBM	High	HOST_CLK
RXFQ0_RREQ	1	O	RX FIFO Queue 0 read request from VBM	High	HOST_CLK
RXFQ1_RREQ	1	O	RX FIFO Queue 1 read request from VBM	High	HOST_CLK
CTM_TX_WREQ	1	O	TX Cut-Through Buffer write request	High	HOST_CLK
CTM_RX_RREQ	1	O	RX Cut-Through Buffer read request	High	HOST_CLK
CTM_RESP	2	I	Cut-Through block copy completion response	na	HOST_CLK
CT MODE ADDITIONAL SIGNALS					
MH_EN	1	O	MH Enable	High	HOST_CLK
CTM_BC_ERR	1	O	CTM Block Copy Error output from MH	High	HOST_CLK
CTM_EN	1	O	CTM Enable	High	HOST_CLK
CTM_DESCRIPTOR	64	O	CT Descriptor Source and Destination addresses	na	HOST_CLK

Table 147. Port List (page 2 of 3)

Signal	Bit Width	I/O	Description	Active Level	Clock Domain
INTERRUPTS (LEVEL BASED)					
FUNC_TX_INT	1	O	TX Functional interrupts	High	HOST_CLK
FUNC_RX_INT	1	O	RX Functional interrupts	High	HOST_CLK
ERR_INT	1	O	Error interrupt	High	HOST_CLK
SAFETY_INT	1	O	Safety interrupt	High	HOST_CLK
LMEM PROTECTION					
LMEMPC_START_ADDR	18	I	LMEM Protection config start address	na	ASYNC
LMEMPC_END_ADDR	18	I	LMEM Protection config end address	na	ASYNC
LMEM_PROTECT_EVENT	1	O	LMEM Protect event	Pulse High	HOST_CLK
TRANSCEIVER INTERFACE					
TXD	1	O	Serial digital transmit to SoC pin TXD, see chapter Hardware interface	na	CAN_CLK
CAN_RX	1	I	Serial digital receive to SoC pin CAN_RX, see chapter Hardware interface	na	ASYNC
TIMEBASE INTERFACE					
TIMEBASE_TIME	64	I	Time value of external time base	na	TIMEBASE_CLK
LMEM INTERFACE					
LMEM_READY	1	I	When set to 0, Bus is busy and when it is 1, the transaction is complete	High	HOST_CLK
LMEM_ECC_UE	1	I	ECC uncorrectable error	Pulse High	HOST_CLK
LMEM_ECC_CE	1	I	ECC correctable error	Pulse High	HOST_CLK
LMEM_RDATA	32	I	Read data	na	HOST_CLK
LMEM_ADDR	18	O	Address	na	HOST_CLK
LMEM_WDATA	32	O	Write data	na	HOST_CLK
LMEM_CS	1	O	Chip select	Low	HOST_CLK
LMEM_WE	1	O	Write Enable	na	HOST_CLK
LMEM_BURST	1	O	LMEM Burst Signal	na	HOST_CLK
LMEM_BURST_LEN	2	O	LMEM Burst Length	na	HOST_CLK
STATIC SIGNALS (required for mode of operation)					
ONLY_CC	1	I	Restricts the PRT to Classical CAN frame format	High	na
ONLY_CC_FD	1	I	Restricts the PRT to Classical CAN and CAN FD frame formats	High	na
NO_SAFETY_NO_CTM	1	I	When set to 1, CTM and all safety features are disabled.	High	na
TSS_EN	1	I	Controls Transceiver Sharing Switch mode: enabled (1) or disabled (0)	High	na
PRT SIDEBAND SIGNALS					
SAMPLE_POINT	1	O	CAN Sample point; for debug and demonstration purpose	High	CAN_CLK
STAT_ACT	2	O	Actual value of register STAT.ACT	na	CAN_CLK
BUS_ERR	1	O	Error detected on CAN bus or Protocol Exception Event detected	High	CAN_CLK
TXD_NRZ_DEBUG	1	O	NRZ coded TX signal provided by PRT; for debug and demonstration purpose	High	CAN_CLK
IS_TRANSMITTER	1	O	Current Transmitter of a CAN frame	High	CAN_CLK

Table 147. Port List (page 3 of 3)

Signal	Bit Width	I/O	Description	Active Level	Clock Domain
GROUP_TR	1	I	Signals whether at least one of the CAN modules in the transceiver sharing group is a transmitter	High	CAN_CLK
SP_BEFORE_DOM_REC_EDGE	1	O	Indicates the correct sampling of the arbitration phase	Pulse High	CAN_CLK
MH Sideband signals					
RX_MSG_STORE_SUCC	1	O	Indicates the successful storage of RX messages in RX FIFO	Pulse High	HOST_CLK
RX_MSG_ALERT	1	O	Set to 1 when incoming message matches with a filter with ALERT bit set to1	High	HOST_CLK
HARDWARE DEBUG PORT					
HDP	16	O	Hardware Debug Port	na	na

1.13 Glossary

Table 148. Glossary (page 1 of 3)

Term/Acronym	Description
AF	Acceptance Field
ASIC	Application Specific IC
ASIL	Automotive Safety Integrity Level
BCD	Binary Coded Decimal
Buffer	1x Element in the FIFO
CAN	Controller Area Network
CAN CC	CAN Classic
CAN FD	CAN with Flexible Data rate
CAN FD light	A commander / responder protocol based on CAN FD
CAN XL	CAN with extended frame Length
CDC	Clock Domain Crossing
CiA	CAN in Automation
CiA 610-1	CiA CAN XL Protocol Standard
Commander	Commander (Master) in a Commander Responder Network
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check used for data consistency check
CRU	Clock Recovery Unit
CTB	Cut Through Buffer
CTD	Cut Through Descriptor
CTM	Cut Through Mode (CTM) means the IP store messages only partly in the LMEM
DFA	Dependent Failure Analysis
DIA	Development Interface Agreement
DLC	Data Length Code
DLC-XL	Data Length Code with CAN XL encoding
DMA	Direct Memory Access
VBM	Virtual Buffer Manager
DST	Destination (data destination)
E_MEM	External Memory accessible off chip
EMI	Electro-Magnetic Interference

Table 148. Glossary (page 2 of 3)

Term/Acronym	Description
ETXP	Enhanced Traffic Pause
FEC	Filter Element Configuration
FF(s)	Flip-Flop(s)
FE	Filter Element
FIFO	First In First Out
FIM	Fault Injection Module
FIR	Fault Injection Request
FM_PLL	Phase Lock Loop with Frequency Modulation to spread the noise spectrum
FMEDA	Failure Mode, Effects and Diagnostic Analysis
FMM	Full Message Mode (FMM) means the IP stores always full (complete) messages in LMEM
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input / Output
HDP	Hardware Debug Port
HFI	Header Format Invalid
HOST	This is the CPU which is hosting the X_CAN
HW	Hardware
IP	Intellectual property e.g., the XS_CAN
IR	Interrupt
IRC	Interrupt Controller
IRQ	Interrupt Request
ISO	International Standardization Organization
ISO 11898-1:2015	ISO Standard for CAN
ISO 11898-1:2024	ISO CAN protocol specification (CAN, CAN FD, CAN XL and CAN FD Light Responder)
ISO 11898-1:2024 Annex A	ISO CAN FD Light Responder Standard
ISO 21434	ISO Standard for Cybersecurity
ISO 21434	This document specifies engineering requirements for cybersecurity risk management regarding concept, product development, production,. operation, maintenance and decommissioning of electrical and electronic (E/E) systems in road vehicles, including their components and interfaces
L_MEM	Local Memory accessible on chip
LMEM	Local Memory
LMEMPC	LMEM protection configuration
LSB	Least Significant Bit
MEM Memory	Memory
MESSAGE	The information package to be transported on the CAN bus
MH	Message Handler
MSB	Most Significant Bit
MSG	Message, see MESSAGE
MUX	Multiplexer
NA	Not Applicable
PARITY	Parity bit used for data consistency check
PLL	Phase Locked Loop
PRT	Protocol Controller
PWM	Pulse Width Modulation
PWME	Pulse Width Modulation Encoder
PWML	PWM long phase length

Table 148. Glossary (page 3 of 3)

Term/Acronym	Description
PWMO	PWM time offset parameter
PWMS	PWM short phase length
QUEUE	Buffer e.g., for Messages
REFP	Reference Pairs
Responder	Responder (Slave) in a Commander Responder Network
RTL	Register Transfer Level (design abstraction level)
RX	Receive, e.g., reception of a frame on the CAN bus
RXFQ	Rx FIFO Queue
RxMsg	Rx Message
RxQE	RX FIFO Queue Element (RXQE)
S_MEM	System Memory. This memory could be an on-chip RAM or an E_MEM
SDT	SDU Type
SEC	Simple Extended Content
SEooC	Safety Element out of Context
SMEM	System Memory
SoC	System on Chip
SPI	Serial Peripheral Interface
SRAM	On chip Static RAM
SRC	Source (data source)
SW	Software
TEFQ	TX Event FIFO Queue
TEQE	TX Event Queue Element
TPE	TX Pause Enhanced
TSS	Transceiver Sharing Switch
Tx	Transmit
TX	Transmit, e.g., transmission of a frame on the CAN bus
Tx FIFO	dynamic message memory where the message are transmitted in the order of First in First out
Tx Pause	Delay of the next transmit message
Tx Queue	dynamic message memory where the message are transmitted in the order of their CAN priority
TX SCAN	SCAN Process used to select the TX message having the highest priority before sending it to the PRT
TXFQ	TX FIFO Queue
TXPQ	Tx Priority Queue
TXQE	TX Queue Element
VCID	Virtual CAN Network ID
VHDL	VHSIC Hardware Description Language
VHSIC	Very High-Speed Integrated Circuit
WORD	Data word used by X_CAN architecture = 32 bit = DWORD
X_CAN	Family of CAN IP supporting CAN XL protocol
X_CAND	X_CAN with embedded DMA
XS_CAN	Slim CAN XL IP

1.14 References

This document refers to the following documents:

Ref Author(s) Title
[1] ISO ISO 11898-1:2024 CAN data link layer and physical signaling

[2] CAN in Automation CiA 610-2 CAN XL Protocol Specification,
[3] (Not Applicable)
[4] MS/EEJ calc_min_host_clk_freq.xlsx
[5] AHB AMBA 2 ARM IHI 0011A
[6] APB AMBA 4 ARM IHI 0011A
[7] ISO ISO 26262-5:2018 Road vehicles - Functional safety
[8] IEC IEC/TR 62380:2004 Reliability data handbook - Universal model for reliability prediction of electronics components, PCBs and equipment
[9] AIAG & VDA AIAG & VDA FMEA Handbook - 1st Edition - June 2019
[10] CAN in Automation CiA 604-3 CAN FD Light
[11] XS_CAN User ManualUser Manual
[12] CTDMA User Manual
[13] XS_CAN Integration Guide
[14] XS_CAN_Local_Memory_size_calculator

1.15 User Manual Version History

Table 149. Local Change History (page 1 of 2)

Version	Date of Issue	Description
2.4	Aug 11th, 2026	Section 1.6.6.3 Transceiver Sharing Switch Support, TSS_EN description corrected. Document status changed to "Released".
2.3	July 13th, 2026	Section 1.5.6.11 Hardware Debug Port updated
2.2	May 14th, 2026	1) Section 1.5.7 Cut Through Descriptors and Block Copy updated. 2) Section 1.5.8.10 Start address not received for Virtual Buffers in Error and Exception Handling updated. 3) Section 1.6.5 PRT.TEST and PRT.MODE register descriptions updated. 4) Section 1.7.2.3 TX_MSG Write Response Channel in MH-PRT Interface updated. 5) Section 1.7.3.3 RX_MSG_WUSER in MH-PRT Interface updated
2.1	April 29th, 2026	1) Typo corrections.. 2) Section 1.4.2.9 Host AHB Slave Interface section updated.. 3) Section 1.6.6.6 Host Access to Protocol Controller updated for PRT.FIMC and PRT.TEST register exceptions.. 4) Section 1.5.6.4 LMEM Configuration updated for TXFQ and RXFQ.
2	April 1st, 2026	1) MH Software Interface section 1.5.5 - TEFQ_CFG register description updated.. 2) 1.4.2.8 LMEM Interface section updated . 3) 1.4.2.10 LMEM Access Protection section updated. 4) 1.10 Application Information section updated.
1.9	Feb 26th, 2026	1) MH Software Interface section 1.5.5 - LMEM_PROT register updated for 64 bytes and TXPQ_LMEM and TEFQ_LMEM register descriptions updated. 2) Section 1.4.2.9 Host AHB Slave Interface updated.
1.8	Feb 3rd, 2026	1) Port Description modified in section 1.6.9.1 for PRT and section 1.12.2 for TOP. . 2) Typo corrections. . 3) IRC Software Interface section 1.8.2 description updated for the TX_FUNC_RAW register. . 4) MH Software Interface section 1.5.5.1 description updated for the TXPQ_LMEM register and TEFQ_LMEM register.
1.7	Dec 18th, 2025	1) Updated section 1.10 Application Information with AHB Burst Length usage recommendation. 2) Updated section 1.9 Clock Domains and Reset with improved text. 3) Improved TX-SCAN diagram in section 1.5.6.6.1. 4) Updated 1.6.6.2 Statistics Counter updated with more information.
1.6	Nov 21st, 2025	1) Section 1.4 TOP and 1.5.5 MH Software Interface content updated with improved text and descriptions. 2) Section 1.5.8 Error and Exception Handling in MH updated. 3) Section 1.6.6.12 Fault Injection Module description updated. 4) Section 1.9 Clock Domains and Resets updated with Clock Frequency Verification summary table. 5) Multiplexer in CDC Channel and CDC for Timebase modified as a replaceable component.

Table 149. Local Change History (page 2 of 2)

Version	Date of Issue	Description
1.5	Sept 26th, 2025	1) RXFQ Fill level Description updated in section 1.5.5. 2) CTM in TX Handler under section 1.5.6.6 and RX Handler Description under section 1.5.6.7 updated. 3) Unaligned Register Access details updated. 4) Rx Abort generation table under section 1.5.6.7.7 updated. 5) LMEM Timing Diagram for back to back Read updated. 6) Register description updated for LOCK and CTRL registers in PRT under 1.6.5.
1.4	Aug 14th, 2025	1) MH, PRT Register description updated. 2) Error and Exception Handling scenario updated for Queue access violation. 3) TSS_EN Port description updated.
1.3	Jul 22th, 2025	1) MH VERSION register update. 2) PRT STAT1 register update. 3) PRT port list update.
1.2	Jun 20th, 2025	Added Drop and Abort interrupts table in section 1.5.6.7 RX Message Handler
1.1	May 16th, 2025	1) Explanation for Target field in RX Filter Element Configuration (FEC) updated. 2) Section 1.5.6.10 PRT_ENABLE and MH_ENABLE Dependency updated.
1.0	Apr 29th, 2025	IPT information added into PRT Overview
0.9	Apr 17th 2025	1) Register description updated for PRT STATISTIC_COUNTER register. 2) MH behavior updated for condition when filtering is completed without a match.
0.8	Mar 31st, 2025	1) MH portlist updated with the addition of PRT_TX_DU. 2) MH_STS0.CLOCK_ACTIVE bit initial value updated to 1. 3) RTR bit addition to TEFQ header. 4) LMEM_ECC_UE made asynchronous to LMEM_READY. 5) Programming Guidelines section update.
0.7	Mar 11th,2025	1) TX_SCAN Logic explanation update. 2) MH Error and Exception handling section update. 3) Programming Guidelines Section added.
0.6	Feb 19th, 2025	CCB ticket updates
0.5	Feb 13th, 2025	CCB ticket updates
0.4	Jan 17th 2025	CCB ticket updates
0.3	Dec 18th, 2024	CCB ticket updates
0.1	Nov 5th, 2024	Initial Version

1.16 Appendix

This section provides an overview of the changes introduced in XS_CAN r01d1.

Table 150. Summary of changes (page 1 of 2)

No	User Manual/Integration Guide section	Change details
1	User Manual Section 1.4.2.1: AHB (Advanced High performance Bus) to APB (Advanced Peripheral Bus) Bridge	AHB write access to registers PRT.FIMC and PRT.TEST without performing the required unlock sequence results in an AHB OK response. No interrupt is asserted.
2	User Manual section 1.4.2.9: HOST_AHB Slave Interface.	XS_CAN Integration requirement modification that the XS_CAN is not suitable for use in a standard multi-slave AHB bus architecture that relies on transfer pipelining across multiple slaves.
3	User Manual section 1.5.5: Register MH.LMEM_PROT	LMEM protection start and end addresses are 64-byte aligned
4	User Manual section 1.5.8.7: MH-PRT Sequence Error	Missing information added to MH-PRT Sequence Error description. ìIn case of R1 = HFI or ARBLOST followed by R2 = WAIT4SOS, sequence error is not triggered.î
5	User Manual section 1.6.5	TSS_EN bit added to register PRT.STAT1.
6	User Manual section 1.6.5	PRT.TEST register description modified for TX_DU and RX_DO bits.
7	User Manual section 1.6.5	TSSE bit removed from PRT.MODE register

Table 150. Summary of changes (page 2 of 2)

8	User Manual section 1.5.6.7.6: RX Abort Interrupt	Conditions to generate RX Abort Interrupt updated.
9	Integration Guide Section 1.6.3	External LMEM arbiter requirements updated inorder to handler LMEM request termination .
10	User Manual section 1.5.6.11: Trace and Debug	Message Handler HDP table updated.

1.17 Disclaimer

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

NO WARRANTIES: TO THE MAXIMUM EXTENT PERMITTED BY LAW, NEITHER THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, NOR ANY PERSON, EITHER EXPRESSLY OR IMPLICITLY, WARRANTS ANY ASPECT OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, INCLUDING ANY OUTPUT OR RESULTS OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO UNLESS AGREED TO IN WRITING. THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IS BEING PROVIDED "AS IS", WITHOUT ANY WARRANTY OF ANY TYPE OR NATURE, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND ANY WARRANTY THAT THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IS FREE FROM DEFECTS.
ASSUMPTION OF RISK: THE RISK OF ANY AND ALL LOSS, DAMAGE, OR UNSATISFACTORY PERFORMANCE OF THIS SPECIFICATION (RESPECTIVELY THE PRODUCTS MAKING USE OF IT IN PART OR AS A WHOLE), SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO RESTS WITH YOU AS THE USER. TO THE MAXIMUM EXTENT PERMITTED BY LAW, NEITHER THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, NOR ANY PERSON EITHER EXPRESSLY OR IMPLICITLY, MAKES ANY REPRESENTATION OR WARRANTY REGARDING THE APPROPRIATENESS OF THE USE, OUTPUT, OR RESULTS OF THE USE OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, BEING CURRENT OR OTHERWISE. NOR DO THEY HAVE ANY OBLIGATION TO CORRECT ERRORS, MAKE CHANGES, SUPPORT THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, DISTRIBUTE UPDATES, OR PROVIDE NOTIFICATION OF ANY ERROR OR DEFECT, KNOWN OR UNKNOWN. IF YOU RELY UPON THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, YOU DO SO AT YOUR OWN RISK, AND YOU ASSUME THE RESPONSIBILITY FOR THE RESULTS. SHOULD THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL LOSSES, INCLUDING, BUT NOT LIMITED TO, ANY NECESSARY SERVICING, REPAIR OR CORRECTION OF ANY PROPERTY INVOLVED TO THE MAXIMUM EXTEND PERMITTED BY LAW.
DISCLAIMER: IN NO EVENT, UNLESS REQUIRED BY LAW OR AGREED TO IN WRITING, SHALL THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS OR ANY PERSON BE LIABLE FOR ANY LOSS, EXPENSE OR DAMAGE, OF ANY TYPE OR NATURE ARISING OUT OF THE USE OF, OR INABILITY TO USE THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, INCLUDING, BUT NOT LIMITED TO, CLAIMS, SUITS OR CAUSES OF ACTION INVOLVING ALLEGED INFRINGEMENT OF COPYRIGHTS, PATENTS, TRADEMARKS, TRADE SECRETS, OR UNFAIR COMPETITION.
INDEMNIFICATION: TO THE MAXIMUM EXTEND PERMITTED BY LAW YOU AGREE TO INDEMNIFY AND HOLD HARMLESS THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, AND EMPLOYEES, AND ANY PERSON FROM AND AGAINST ALL CLAIMS, LIABILITIES, LOSSES, CAUSES OF ACTION, DAMAGES, JUDGMENTS, AND EXPENSES, INCLUDING THE REASONABLE COST OF ATTORNEYS' FEES AND COURT COSTS, FOR INJURIES OR DAMAGES TO THE PERSON OR PROPERTY OF THIRD PARTIES, INCLUDING, WITHOUT LIMITATIONS, CONSEQUENTIAL, DIRECT AND INDIRECT DAMAGES AND ANY ECONOMIC LOSSES, THAT ARISE OUT OF OR IN CONNECTION WITH YOUR USE, MODIFICATION, OR DISTRIBUTION OF THIS

SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, ITS OUTPUT, OR ANY ACCOMPANYING DOCUMENTATION.
GOVERNING LAW: THE RELATIONSHIP BETWEEN YOU AND ROBERT BOSCH GMBH SHALL BE GOVERNED SOLELY BY THE LAWS OF THE FEDERAL REPUBLIC OF GERMANY. THE STIPULATIONS OF INTERNATIONAL CONVENTIONS REGARDING THE INTERNATIONAL SALE OF GOODS SHALL NOT BE APPLICABLE. THE EXCLUSIVE LEGAL VENUE SHALL BE DUESSELDORF, GERMANY.
MANDATORY LAW SHALL BE UNAFFECTED BY THE FOREGOING PARAGRAPHS.
INTELLECTUAL PROPERTY OWNERS/COPYRIGHT OWNERS/CONTRIBUTORS: ROBERT BOSCH GMBH, ROBERT BOSCH PLATZ 1, 70839 GERLINGEN, GERMANY AND ITS LICENSORS.

Robert Bosch GmbH
Robert-Bosch-Platz 1
70839 Gerlingen-Schillerhöhe