Application Note, V1.00, June 2007

AP08062

XC886MultiCanController

Stand-Alone MultiCAN Controller (SAMC)
User's Guide

Microcontrollers

Never stop thinking

Edition 2007-06-17

Published by
Infineon Technologies AG
81726 München, Germany

© Infineon Technologies AG 2007.
All Rights Reserved.

LEGAL DISCLAIMER

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

Information

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

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

AP08062		
Revision History:	2007-06	V1.00
Previous Version:	none	

Page	Subjects (major changes since last revision)
V1.00	Initial Release

We Listen to Your Comments
Any information within this document that you feel is wrong, unclear or missing at all?
Your feedback will help us to continuously improve the quality of this document.
Please send your proposal (including a reference to this document) to:
mcdocu.comments@infineon.com

Author: Chris Wunderlich

Infineon Technologies North America Corp.

Table of Contents Page

1 Stand-Alone XC886MultiCanController Device 6
1.1 Introduction 6
1.2 Architectural Overview 6
1.3 Application Fields 6
1.4 Basic Requirements of the XC886MultiCanController device 6
1.5 Firmware Code 7
1.5.1 Keil Software 7
1.5.1.1 Keil Debugger 7
1.5.1.2 HiTop Debugger for XC800 7
1.5.1.3 FLOAD UART Downloader 8
1.6 Pin Definition and Functions 8
1.6.1 Pin Configuration 8
1.7 Device Description 10
1.7.1 Clock Generation Unit 10
1.7.2 PLL Mode 11
1.8 Power Management 11
1.8.1 Overview 11
1.8.2 Power-Down Mode 12
1.9 Port Control 12
1.9.1 Mode Selections 12
1.9.2 Input Port Configuration 13
1.9.2.1 Special action configured to an input pin 13
1.9.2.2 Port pin used as a Trigger for a CAN Message Transfer 13
1.9.3 Output Port Configuration 13
1.9.4 Spare I/O 13

2 Serial Interfaces 13
2.1 Communication via the Universal Asynchronous Receivers/Transmitters (UART) 13
2.2 Communication via the Synchronous Serial Channel (SSC) 14
2.2.1 SSC in Slave Mode 15
2.2.2 Transfer length 15
2.2.2.1 Handshake 15
2.2.3 Baud Rate Generation 15
2.2.4 Register Address Map 16
2.2.5 Error Handling 16

3 MultiCAN Module Description 17
3.1 MultiCAN Register Address Map 17

4 Application Program Interface (API) 18
4.1 Mailbox Layout (2+20 bytes): 18
4.2 Structure for a 32-bit Little Endian machine (TriCore) 21
4.3 API to communicate between host and slave controllers 22
4.3.1 API 0: NOP 23
4.3.1.1 MTSR Packet Description 23
4.3.1.2 MRST Packet Description 23
4.3.2 API 1: SetCanChannelOnOff 24
4.3.2.1 MTSR Packet Description 24
4.3.2.2 MRST Packet Description 24
4.3.3 API 2: GetCanChannelOnOff 25
4.3.3.1 MTSR Packet Description 25
4.3.3.2 MRST Packet Description 25
4.3.4 API 3: SetCanIrqOnOff 26
4.3.4.1 MTSR Packet Description 26
4.3.4.2 MRST Packet Description 26
4.3.5 API 4: GetCanIrqOnOff 27
4.3.5.1 MTSR Packet Description 27
4.3.5.2 MRST Packet Description 27
4.3.6 API 5: SetCanCounter 28

Table of Contents **Page**

4.3.6.1 MTSR Packet Description 28
4.3.6.2 MRST Packet Description 28
4.3.7 API 6: GetCanCounter 29
4.3.7.1 MTSR Packet Description 29
4.3.7.2 MRST Packet Description 29
4.3.8 API 7: GetCanIrqStatus....30
4.3.8.1 MTSR Packet Description 30
4.3.8.2 MRST Packet Description 30
4.3.9 API 8: SetCanObject 31
4.3.9.1 MTSR Packet Description 31
4.3.9.2 MRST Packet Description 31
4.3.10 API 9: GetCanObject....32
4.3.10.1 MTSR Packet Description 32
4.3.10.2 MRST Packet Description 32
4.3.11 API 10: SetCanBitRate....33
4.3.11.1 MTSR Packet Description 33
4.3.11.2 MRST Packet Description 33
4.3.12 API 11: GetCanBitRate 34
4.3.12.1 MTSR Packet Description 34
4.3.12.2 MRST Packet Description 34
4.3.13 API 12: SetCanRegData 35
4.3.13.1 MTSR Packet Description 35
4.3.13.2 MRST Packet Description 35
4.3.14 API 13: GetCanRegData....36
4.3.14.1 MTSR Packet Description 36
4.3.14.2 MRST Packet Description 36
4.3.15 API 14: SetCpuClock 37
4.3.15.1 MTSR Packet Description 37
4.3.15.2 MRST Packet Description 37
4.3.16 API 15: GetCpuClock 38
4.3.16.1 MTSR Packet Description 38
4.3.16.2 MRST Packet Description 38
4.4 Data flow example (SSC mode)....39
4.5 Bandwidth calculations (SPI) 40
4.6 Logic Analyzer Diagrams of some transfers 40
4.6.1 Transmit a message on the CAN bus 40
4.6.2 Receive a message from the CAN bus....41
4.6.3 Read data from a register on MultiCAN 41

1 Stand-Alone XC886MultiCanController Device

1.1 Introduction

This is a stand-alone device (SAx-XC886CM-8FF with firmware) that is capable of providing a connection with two independent Controller Area Networks (CAN). A user configurable mechanism for communication with a host controller is provided.

The purpose of this application note is to provide a framework (specification) for communication and configuration of the MultiCAN module on the XC886 microcontroller. This document will refer to this device as the XC886MultiCanController.

1.2 Architectural Overview

The Stand-Alone XC886MultiCanController provides several sub modules to control the data flow and to configure the peripheral function:

- Two serial interface channels are implemented for the communication with a host device CPU to read and write the XC886MultiCanController's internal registers for initial configuration and control during normal operation.
  - Synchronous Serial Channel (SSC), an alternate name would be Synchronous Peripheral Interface (SPI).
  - Universal Asynchronous Receiver / Transmitter (UART)
  - The interface selection is done via a MODE pin, which can be directly connected to the supply voltage or via pull-up/down resistors (of about 10-47 kOhm)
- Both communication channels are based on byte transfers.
- Initialization mechanism for all MultiCAN registers can be configured via internal Flash memory or via serial commands.
- The Port Control unit can be used to select the required functionality of the port pins operating as a communication channel, interrupt request line or general purpose I/O.
- The internal power saving modes can be used to reduce current consumption.
- Interrupt requests generated by the XC886MultiCanController can be sent to the external host via output pins.

1.3 Application Fields

The Stand-Alone MultiCAN device "XC886MultiCanController" can be used in applications requiring one or two independent CAN nodes. Message objects are built with double-chained lists that are assigned by the user to one or the other nodes. The FIFO and gateway features minimize the CPU load for the message handling and lead to an improved real-time behavior. The access to the internal registers is handled via a serial interface that can be adapted to a large variety of applications. The clock generation can be controlled either by the XC886MultiCanController or by the system it is connected to.

1.4 Basic Requirements of the XC886MultiCanController device

- Compliant with the ISO 11898
- CAN functionality according to CAN specification V2.0 B active
- Dedicated control registers for each CAN node
- Data transfer rates up to 1 Mbit/s
- Full-CAN functionality: A set of 32 message objects that can be individually
  - Allocated (assigned) to either CAN node

- Configured as transmit or receive object
- Setup to handle frames with 11-bit, 29-bit identifier or both

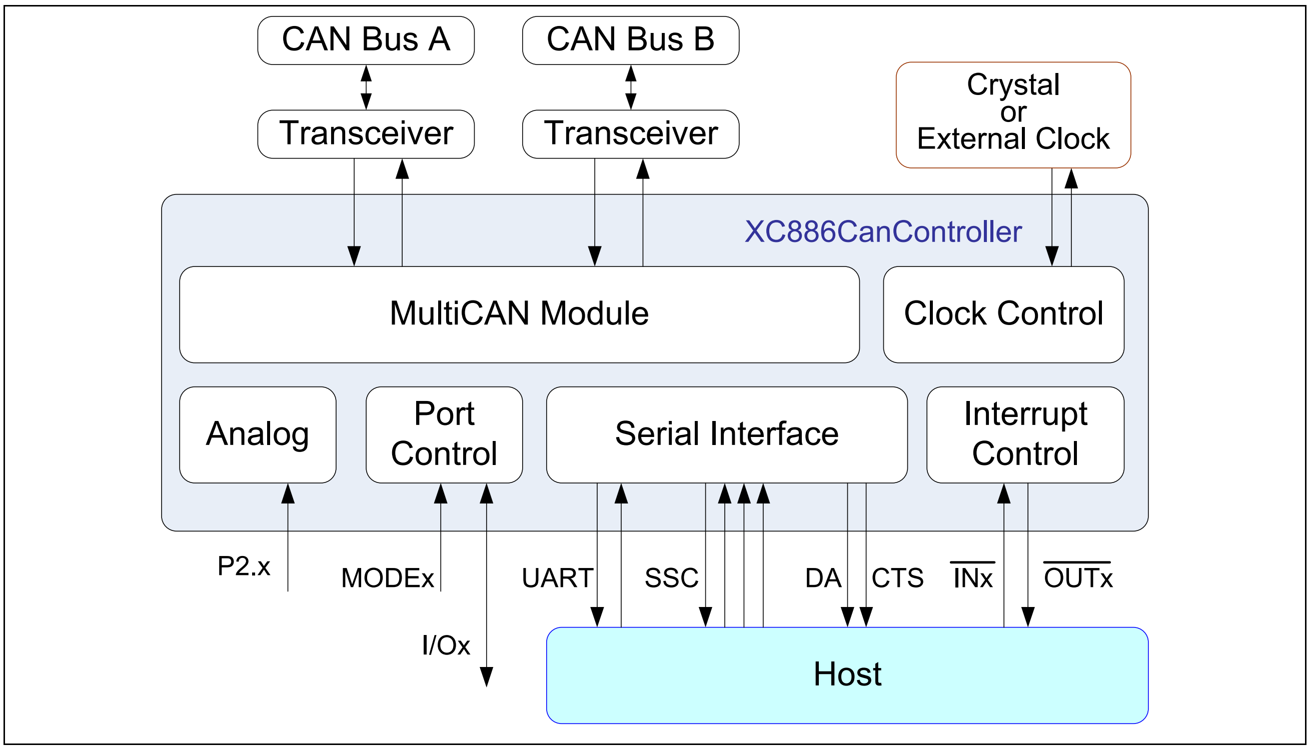


Figure 1 Port Control Unit

1.5 Firmware Code

1.5.1 Keil Software

This application note provides the full source code and project configuration created with the Keil tool chain. The Keil version used is v8.08 and exceeds the demo limit and requires a full license to modify the code. The hex file is provided if you want to try the firmware without modification.

For more information about the Keil please visit their web site: http://www.keil.com

1.5.1.1 Keil Debugger

Although the size of the code exceeds the 2k demo limit for the Keil Software, the integrated Keil debugger can still be used to download the code via the On-Chip Debug Support (OCDS) which is a modified JTAG interface. The Keil ULINK, Infineon Wiggler Box or Hitex Tantino can be used to connect the target hardware (e.g. an evaluation board) with the PC. With the demo version of the Keil tools, debugging is not possible. With the full version of the Keil C51 tools, the integrated simulator can be used for debugging without any real hardware.

1.5.1.2 HiTop Debugger for XC800

For debugging the firmware the user can use the Hitex HiTOP52-XC800

For more information about Hitex please visit their web site: http://www.hitex.com

1.5.1.3 FLOAD UART Downloader

The free FLOAD tool from Infineon can also be used to download the hex file via the UART interface. The FLOAD tool can be found on the Infineon web site: http://www.infineon.com

1.6 Pin Definition and Functions

1.6.1 Pin Configuration

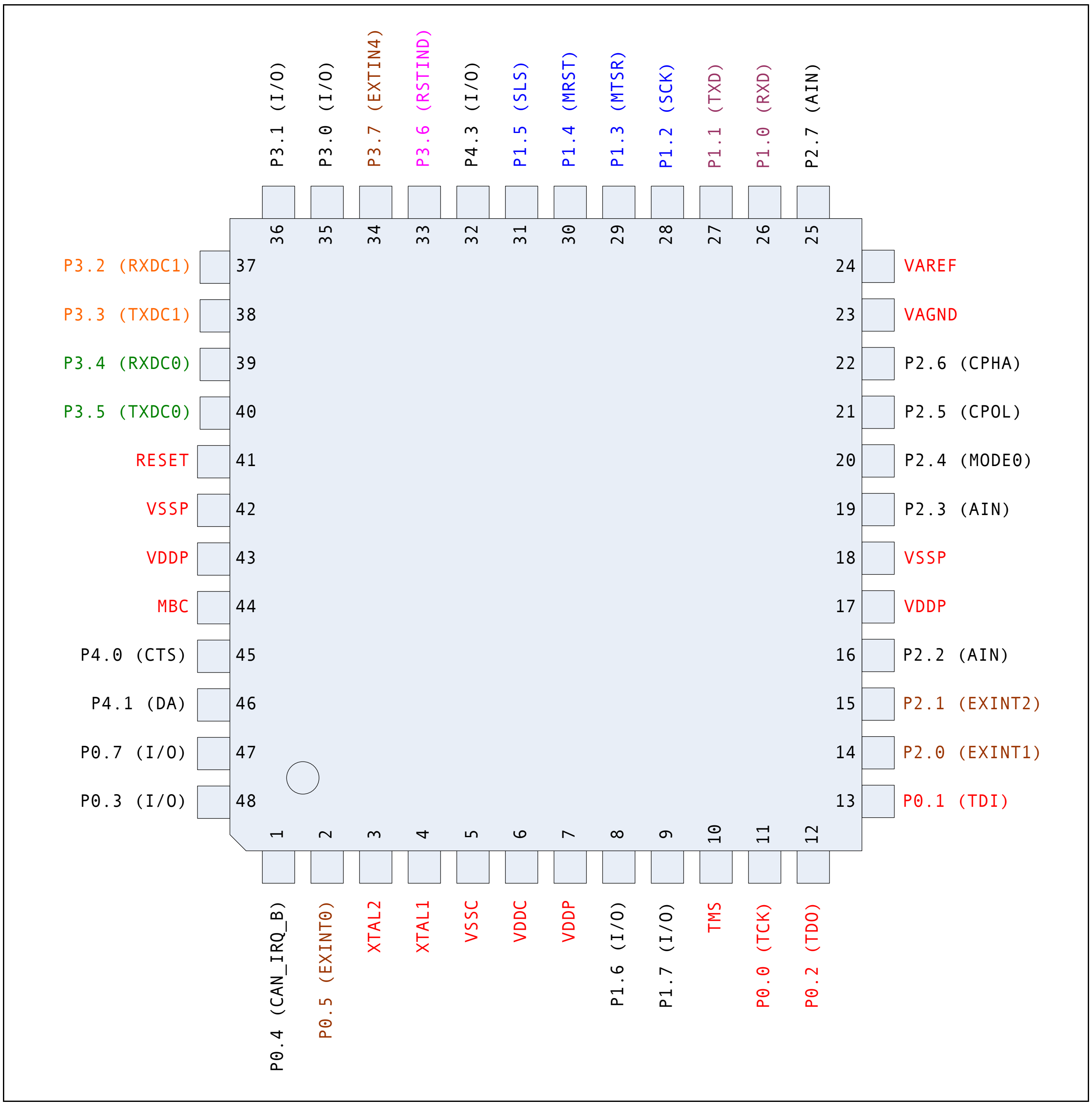


Figure 2 XC886MultiCanController Pin Configuration

The pin functions of the XC886 are described in Table 1.

Table 1 Pin Definition and Function

Label	Name	Pin	Reset State	I/O	Description
CAN_IRQ_B	P0.4	1	Hi-Z	O	Output (active low) to notify an external host that an action has occurred internally to the XC886
EXINT0	P0.5	2	Hi-Z	I/O	User configurable I/O Can be configured to wake up from power down mode
XTAL2	-	3	Hi-Z	O	External Oscillator Output
XTAL1	-	4	Hi-Z	I	External Oscillator Input
VSSC	-	5	-	-	Core Supply Ground
VDDC	-	6	-	-	Core Supply Output (2.5V)
VDDP	-	7, 17, 43	-	-	I/O Port Supply (3.3V or 5V)
-	P1.6	8	PU	I/O	User configurable pin
-	P1.7	9	PU	I/O	User configurable pin
TMS		10	PD	I	Test Mode Select
TCK	P0.0	11	Hi-Z	I	JTAG Clock Input
TDO	P0.2	12	PU	O	JTAG Serial Data Output
TDI	P0.1	13	Hi-Z	I	JTAG Serial Data Input
EXINT1	P2.0	14	Hi-Z	I	Analog or Digital Input
EXINT2	P2.1	15	Hi-Z	I	Analog or Digital Input
-	P2.2	16	Hi-Z	I	Analog or Digital Input
VSSP	-	18, 42	-	-	I/O Port Ground
-	P2.3	19	Hi-Z	I	Analog or Digital Input
COMM_MODE	P2.4	20	Hi-Z	I	Communication mode (Serial Interface Selection)
CPOL	P2.5	21	Hi-Z	I	SPI clock polarity configuration pin.
CPHA	P2.6	22	Hi-Z	I	SPI clock phase configuration pin.
VAGND	-	23	-	-	Analog Reference Voltage
VAREF	-	24	-	-	Analog Reference Ground
-	P2.7	25	Hi-Z	I	Analog or Digital Input
RXD	P1.0	26	PU	I	UART Receive Input
TXD	P1.1	27	PU	O	UART Transmit Output
SCK	P1.2	28	PU	I	SSC Slave Clock Input
MTSR	P1.3	29	PU	I	SSC Slave Receive Input
MRST	P1.4	30	PU	O	SSC Slave Transmit Output
SLS	P1.5	31	PU	I	SSC Slave Select Input
I/O	P4.3	32	Hi-Z	I/O	User configurable I/O or Output Trigger
RSTIND	P3.6	33	PD	O	Reset indication for internal reset condition in microcontroller
EXINT4	P3.7	34	Hi-Z	I/O	User configurable I/O
-	P3.0	35	Hi-Z	I/O	User configurable I/O
-	P3.1	36	Hi-Z	I/O	User configurable I/O

Label	Name	Pin	Reset State	I/O	Description
RXDC1	P3.2	37	Hi-Z	I	MCAN Node 1 Receiver Input
TXDC1	P3.3	38	Hi-Z	O	MCAN Node 1 Transmitter Output
RXDC0	P3.4	39	Hi-Z	I	MCAN Node 0 Receiver Input
TXDC0	P3.5	40	Hi-Z	O	MCAN Node 0 Transmitter Output
RESET	-	41	PU	I	Reset Input
MBC	-	44	PU	I	Monitor & Bootstrap Loader Control
CTS	P4.0	45	Hi-Z	O	Clear To Send 1: The host controller can send a new message frame to the XC886 (slave) 0: Busy or Processing a previous message
DA	P4.1	46	Hi-Z	O	Data Available 1: There is a new message frame ready to be read from the XC886 (slave) 0: No messages pending for the host to read
-	P0.7	47	PU	I/O	User configurable Output
-	P0.3	48	Hi-Z	I/O	User configurable Output

1.7 Device Description

1.7.1 Clock Generation Unit

The Clock Generation Unit (CGU) in the XC886 consists of an oscillator circuit and a Phase-Locked Loop (PLL). In the XC886, the oscillator can be from either of these two sources: the on-chip oscillator (9.6 MHz) or the external oscillator (3 MHz to 12 MHz). The term “oscillator” is used to refer to both on-chip oscillator and external oscillator, unless otherwise stated. After the reset, the on-chip oscillator will be used by default. The external oscillator can be selected via software. The PLL can convert a low-frequency external clock signal from the oscillator circuit to a high-speed internal clock for maximum performance.

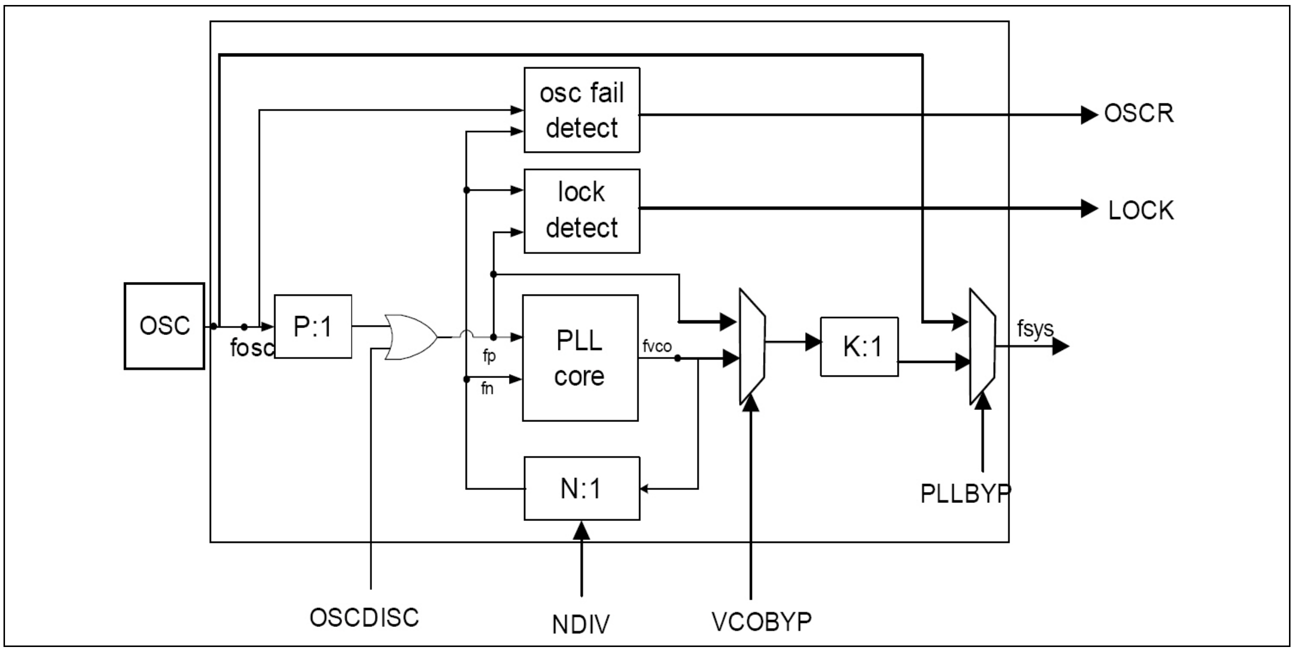


Figure 3 CGU Block Diagram

When the XC886 is powered up, the PLL is disconnected from the oscillator and will run at its VCO base frequency. After the EVR is stable, provided the oscillator is running, the PLL will be connected and the continuous lock detection will ensure that the PLL starts functioning. Once reset has been released, bit OSCR will be set to 1 if the oscillator is running and bit LOCK will be set to 1 if the PLL is locked.

Table 2 XTAL1 Input Characteristics (Operating Conditions apply)

Parameter	Symbol	Limit Values		Unit
		min.	max.	
Digital core supply voltage	V_{DDC}	2.3	2.7	V
Input low voltage at XTAL1	V_{ILX}	V_{SS} - 0.5	0.3 × V_{DDC}	V
Input high voltage at XTAL1	V_{IHX}	0.7 × V_{DDC}	V_{DDC} + 0.5	V

1.7.2 PLL Mode

The system clock is derived from the oscillator clock, divided by the P factor, multiplied by the N factor, and divided by the K factor. Both VCO bypass and PLL bypass must be inactive for this PLL mode. This is the mode the XC886MultiCanController will be configured as the run mode. The fCPU is programmed to run at 24MHz. Within the software a calibration value (CRYSTAL_FREQ) will be user selectable (see **Table 3**), with the default value assumed to be an 8MHz external crystal.

For the XC886MultiCanController , the value of P is fixed to 1. In order to obtain the required fsys, the value of N and K can be selected by bits NDIV and KDIV respectively for different oscillator inputs. The output frequency must always be configured for 96 MHz.

The clock frequency of the MultiCAN peripheral is running at 48MHz.

Table 3 CRYSTAL_FREQ Selections

Label	Oscillator	Frequency	N	P	K	fsys	fcpu
_4MHZ	External	4 MHz	48	1	2	96 MHz	24 MHz
_6MHZ	External	6 MHz	32	1	2	96 MHz	24 MHz
_8MHZ	External	8 MHz	24	1	2	96 MHz	24 MHz
_9_6MHZ	On-Chip	9.6 MHz	20	1	2	96 MHz	24 MHz
_12MHZ	External	12 MHz	16	1	2	96 MHz	24 MHz

1.8 Power Management

1.8.1 Overview

The XC886MultiCanController power-management system allows software to configure the various processing units so that they automatically adjust to draw the minimum necessary power for the application.

There are four power-management modes: Active Mode, Idle Mode, Slow Down Mode, and Power Down Mode. The operation of each system component in each of these states can be configured by software. The power management modes provide flexible reduction of power consumption through a combination of techniques, including:

- Stopping the CPU clock
- Stopping the clocks of other system components individually
- Clock-speed reduction of some peripheral components
- Power-down of the entire system with fast restart capability

1.8.2 Power-Down Mode

In the power-down mode, the oscillator and the PLL are turned off. The FLASH is put into the power-down mode. The main voltage regulator is switched off, but the low power voltage regulator is still operating. Therefore, all functions of the microcontroller are stopped and only the contents of the FLASH, on-chip RAM, XRAM, and the SFRs are maintained. The port pins hold the logical state they had when the power-down mode was activated.

In power-down mode, the clock is turned off. Hence, it cannot be awakened by an interrupt or the Watchdog Timer. It will be awakened only when it receives an external wake up signal or reset signal.

Power down can be selected via port pin (if previously enabled by software) or by software command via API call.

Exiting Power-Down Mode

Power down-mode can be exited in two ways:

- The EXINT0 pin detects a falling edge, or
- The RXD pin detects a falling edge

1.9 Port Control

1.9.1 Mode Selections

Pin MODE selects whether the on-chip SSC or the UART interface is used as the communication channel. A logic level 0 immediately after RESET is released the UART module is enabled. Likewise a logic level 1 activates the SSC module.

Note: Any configuration related to the state after a rising edge of RESET is performed by software as the normal startup sequence and the user needs to ensure that the pin state remains stable until this process has ended.

The logic state of pin MODE0 during a rising edge on the RESET pin determines the access mode:

- If the UART is selected:
  - MODE0=0 configures the on-chip UART.
- If the SSC is selected:
  - MODE0=1 configures the on-chip SSC as slave device, which requires an external SSC operating in master mode as the communication partner (host device).

Table 4 User Configuration Pins

Label	Pin	I/O	Description

P2.3 (COMM_MODE)	20	I	Serial Interface Selection.
			MODE=0, UART Mode Active.
			MODE=1, SSC Mode Active.

P2.4 (CPOL)	21	I	SPI clock polarity configuration pin.
			CPOL=0, SCLK active low.
			CPOL=1, SCLK active high.

P2.5 (CPHA)	22	I	SPI clock phase configuration pin. CPHA=0, shift data on leading edge, latch data on trailing edge. CPHA=1, latch data on leading edge, shift data on trailing edge.

After latching the initial states of MODE, CPOL and CPHA with the rising edge of the reset signal, the pins can be used as regular inputs (or analog).

1.9.2 Input Port Configuration

1.9.2.1 Special action configured to an input pin

It's possible to configure different types of actions based on the edge/level of certain port pins. Some options could be to go into sleep mode, change modes of operation and trigger CAN transfers or filters.

The determination of available IN (EXINTx) pins is TBD.

1.9.2.2 Port pin used as a Trigger for a CAN Message Transfer

When register TBD is loaded with TBD, a falling edge detected on pin(s) TBD generates a message transfer via the CAN bus by setting control bit TXRQ in the corresponding message object. If the respective message object is configured for transmit operation, a data frame will be transmitted. A configuration as a receive message object causes a remote frame to be sent out.

1.9.3 Output Port Configuration

The XC886MultiCanController has 34 port pins organized into five parallel ports, Port 0 (P0) to Port 4 (P4). Each pin has a pair of internal pull-up and pull-down devices that can be individually enabled or disabled. Ports P0, P1, P3 and P4 are bidirectional and can be used as general purpose input/output (GPIO) or to perform alternate input/output functions for the on-chip peripherals. When configured as an output, the open drain mode can be selected. Port P2 is an input-only port, providing general purpose input functions, alternate input functions for the on-chip peripherals, and also analog inputs for the Analog-to-Digital Converter (ADC).

The determination of available OUT pins is TBD.

If pin OUTx is configured as interrupt output via bit field TBD in register TBD, a logic level 0 indicates an interrupt request to the external host device. The interrupt line will be active if there is a new pending interrupt request for interrupt node 0 according to the selection made by control bit TBD in register TBD.

1.9.4 Spare I/O

The remaining I/O pins are controlled by port control logic and can be used as I/O extension. These lines can be read or written by the serial channel or by CAN messages (if defined by the API). Furthermore, these lines can be programmed as additional interrupt output lines in order to increase the number of independent interrupts.

2 Serial Interfaces

2.1 Communication via the Universal Asynchronous Receivers/Transmitters (UART)

UART Features:

- Full-duplex asynchronous modes
- 8-bit data frames, no parity, 1 start bit, one stop bit, LSB first
- Binary data transfer based on a defined protocol
- Baud rate is 115200

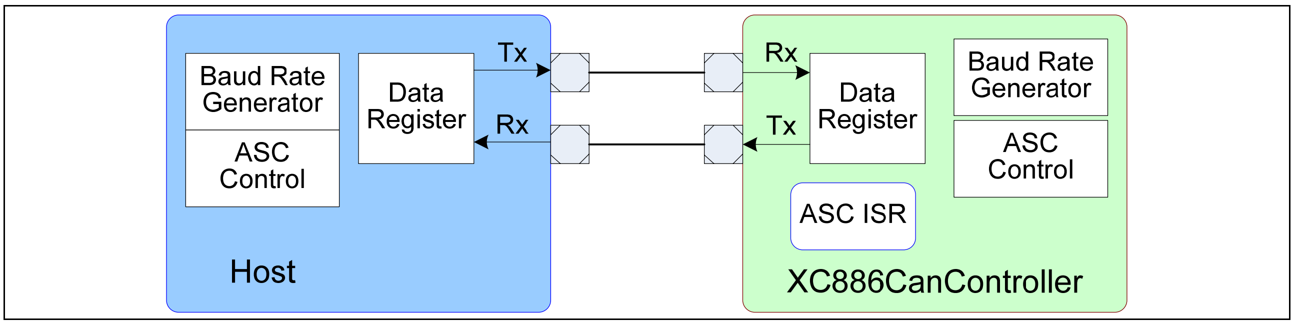


Figure 4 UART Configuration

The message packet used for the UART will consist of a Header byte and a "Packet Cnt" (number of bytes contained within the total message packet). Then the API packet is inserted ending with a "Chksum" byte (a two's complement checksum for all bytes within the message packet) (see **Figure 5**).

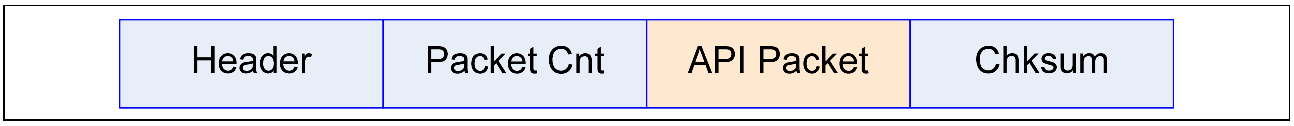


Figure 5 UART Message Packet

2.2 Communication via the Synchronous Serial Channel (SSC)

SSC Features (Slave Mode):

- Full-duplex synchronous Slave mode
- Master mode is not allowed for communicating to a host
- 8-bit data frames, MSB first
- Phase and Clock Polarity is according to pins CPHA and CPOL (see **Table 4**)
- fixed baud rate (1 Mbit @ 24MHz fCPU clock)
- SSC is equivalent to an SPI

The reset value of the baud rate generator (SSCBRG) is 0x000B expecting a clock signal of 1 MHz at the SCK pin given the internal fCPU is equal 24 MHz on the XC886MultiCanController device.

Note: In order to avoid undefined states during power-up, it is recommended to add pull resistors (about 47-100 kOhm) to the power supply on the SLS and SCK lines.

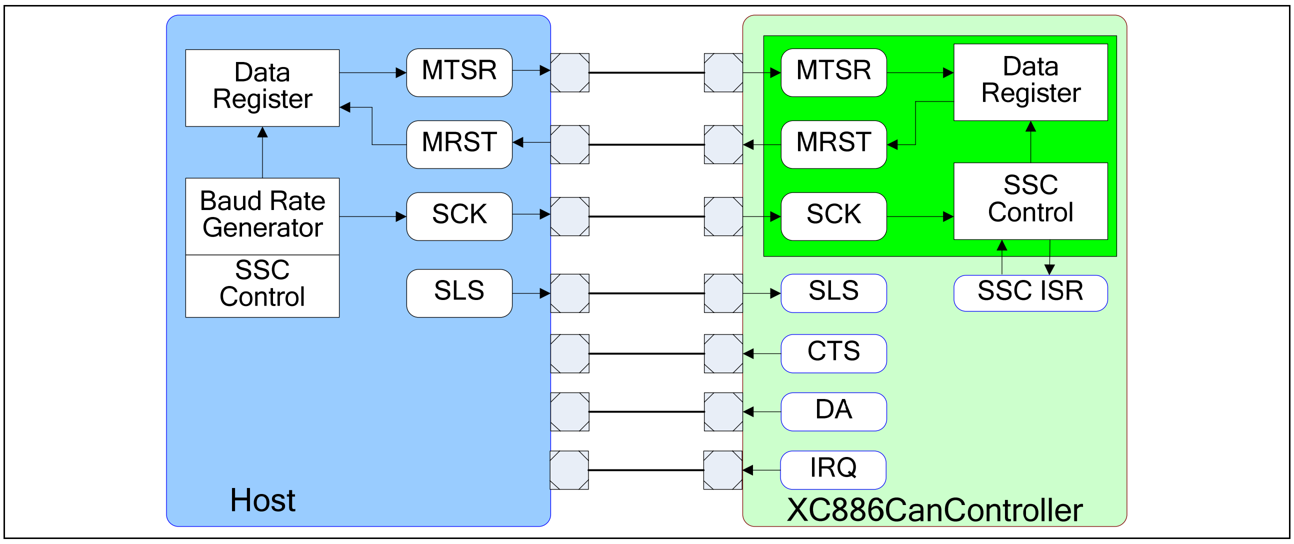


Figure 6 SSC Slave Configuration

2.2.1 SSC in Slave Mode

The XC886MultiCanController device can be easily connected to an external host device via a serial channel. This mode is selected by pins MODE=0 and CPOL and CPHA levels (see **Table 4**) at the rising edge of the RESET signal. In slave mode, the on-chip SSC can be connected to an external SSC (Host) in master mode according to **Table 4**.

The SSC module of the XC886MultiCanController is equivalent to a standard four-line SPI-compatible interface. The XC886MultiCanController **does not implement a true slave select signal** meaning the clock is always active and the user must dedicate the SPI channel (or at least ensure that all transfers are 8-bits in length). The user can also choose to externally gate the SCK with the SLS signal.

The host can only begin a new message transfer if the CTS signal is at a high level (logic ‘1’). Upon the transmission of a message stream, the data is expected or returned by the XC886MultiCanController as long as the SLS pin is held on a low level (logic ‘0’) and the SCK pin is provided with a clock signal (maximum of 22 bytes). The signal SLS has to be set to a high level (logic ‘1’) by the host to complete the message transfer. The XC886MultiCanController device will drive the signal CTS low during the last byte of the transfer. This indicates the transfer is complete from XC886MultiCanController perspective. The XC886MultiCanController will drive the CTS signal high (logic ‘1’) when it has processed the message from the host.

Note: The host is not allowed to initiate any message transfers unless the CTS signal is set (logic ‘1’).

2.2.2 Transfer length

The first byte transmitted by the external master (host) SSC after the activation of the SLS signal, represents the API command. All transfers on the SSC interface are of a fixed length (22 bytes) and support full-duplex operation (once the corresponding buffer has been filled). This type of transfer was chosen to support standard DMA Block transfers controlled by a host controller.

2.2.2.1 Handshake

The XC886MultiCanController will configure two port pins as status outputs for the host to read. The two pins named “Clear To Send” (CTS) and “Data Available” (DA). These pins are used for hardware flow control between the host and slave controllers.

CTS	“high” the host controller can send a new message frame to the XC886MultiCanController (slave), “low” – otherwise
DA	“high” there is a new message frame ready to be read from the XC886MultiCanController (slave) “low” – otherwise

2.2.3 Baud Rate Generation

The synchronous serial channel SSC has its own dedicated 16-bit baud-rate generator with 16-bit reload capability. **Figure 7** shows the baud-rate generator.

The baud rate of the SSC is fixed (at 1Mbit) and there is no API call to allow for modification by the host.

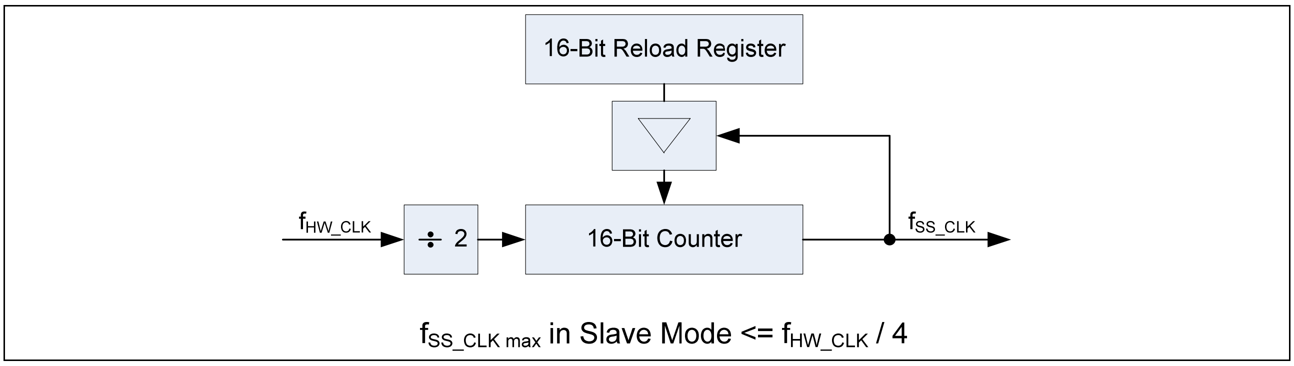


Figure 7 SSC Baud-rate Generator

The baud-rate generator is clocked with the module clock fPCLK. The timer counts downwards. Register BR is the dual-function Baud-rate Generator/Reload register. Reading BR, while the SSC is enabled, returns the contents of the timer. Reading BR, while the SSC is disabled, returns the programmed reload value. In this mode, the desired reload value can be written to BR.

The formulas below calculate either the resulting baud rate for a given reload value or the required reload value for a given baud rate:

$$\text{Baud rate} = \frac{f_{HW\_CLK}}{2 * (<BR> + 1)} \qquad BR = \frac{f_{HW\_CLK}}{2 * \text{Baud rate}} - 1$$

The maximum baud rate that can be achieved when using a module clock of 24 MHz is 12 MBaud in master mode (with <BR> = 0000H) or 6 MBaud in slave mode (with <BR> = 0001H).

2.2.4 Register Address Map

Table 5 Summary of Registers

Register Symbol	Register Name	Address	Reset Value
RMAP = 0			
PISEL	Port Input Select Register	$A9_H$	00_H
CONL	Control Register Low	AA_H	00_H
CONH	Control Register High	AB_H	00_H
TBL	Transmitter Buffer Register Low	AC_H	00_H
RBL	Receiver Buffer Register Low	AD_H	00_H
BRL	Baud rate Timer Reload Register Low	AE_H	00_H
BRH	Baud rate Timer Reload Register High	AF_H	00_H

2.2.5 Error Handling

TDB

3 MultiCAN Module Description

3.1 MultiCAN Register Address Map

All Kernel registers, implemented for controlling the MultiCAN in the XC886MultiCanController, are summarized in **Table 6**; detailed information about each register is provided in the respective module description chapter in the XC886/8 User's Manual.

Note: *Accesses to addresses which are not specified as registers in the following register address map are forbidden.*

To decode the address of the MultiCAN kernel registers, at least an 14-bit address line is needed. As the MultiCAN registers are 32-bit wide (4 Bytes), the address lines A[1:0] are not needed for decoding and are tied to "00". The address lines A[13:2] are implemented and they are programmed from the register bits CA2 to CA9 in the register ADL and CA10 to CA13 in the register ADH. The address registers need to be programmed before accessing the MultiCAN registers.

Note: *This means you need to shift right the offset value by 2 for the correct address in the MultiCAN register. Example, writing to register NCR0 address would be an offset address of 200h but the value written to the MultiCAN would be 0x200 >> 2 = 0x80.*

Table 6 Summary of Registers for MultiCAN

Register Symbol	Register Name	Offset Address	Reset Value
MultiCAN Global Module Registers			
LISTm	List Register m	$0100_H + m \times 4_H$	$XXXX\ XXXX_H$
MSPNDk	Message Pending Register k	$0120_H + k \times 4_H$	
MSIDk	Message Index Register k	$0140_H + k \times 4_H$	
MSIMASK	Message Index Mask Register	$01C0_H$	
PANCTR	Panel Control Register	$01C4_H$	
MCR	Module Control Register	$01C8_H$	
MITR	Module Interrupt Trigger Register	$01CC_H$	
Node Registers			
NCRx	Node x Control Register	$0200_H + x \times 100_H$	$XXXX\ XXXX_H$
NSRx	Node x Status Register	$0204_H + x \times 100_H$	
NIPRx	Node x Interrupt Pointer Register	$0208_H + x \times 100_H$	
NPCRx	Node x Port Control Register	$020C_H + x \times 100_H$	
NBTRx	Node x Bit Timing Register	$0210_H + x \times 100_H$	
NECNTx	Node x Error Counter Register	$0214_H + x \times 100_H$	
NFCRx	Node x Frame Counter Register	$0218_H + x \times 100_H$	
Message Object Registers			
MOFCRn	Message Object n Function Control	$1000_H + n \times 20_H$	$XXXX\ XXXX_H$
MOFGPRn	Message Object n FIFO/Gateway Pointer	$1004_H + n \times 20_H$	
MOIPRn	Interrupt Pointer	$1008_H + n \times 20_H$	
MOAMRn	Acceptance Mask	$100C_H + n \times 20_H$	
MODATALn	Data Register Low (0-3)	$1010_H + n \times 20_H$	
MODATAHn	Data Register High (4-7)	$1014_H + n \times 20_H$	
MOARn	Arbitration (ID)	$1018_H + n \times 20_H$	
MOCTRn MOSTATn	Message Object n Control Reg. Message Object n Status Reg.	$101C_H + n \times 20_H$	$1F1E\ 0000_H$

Summary of Registers

Register Symbol	Register Name	Address	Reset Value
RMAP = 0			
ADCON	CAN Address/Data Control	D8H	00H
ADL	CAN Address Low	D9H	00H
ADH	CAN Address High	DAH	00H
DATA0	CAN Data Register 0	DBH	00H
DATA1	CAN Data Register 1	DCH	00H
DATA2	CAN Data Register 2	DDH	00H
DATA3	CAN Data Register 3	DEH	00H

4 Application Program Interface (API)

The Application Program Interface (API) is a set of routines that the Host application uses to request and receive lower-level services performed on the XC886MultiCanController.

A simplistic configuration would have the host controller generally see the content of just two receive mailboxes (one for each CAN channel) and populate information in two transmit mailboxes (one for each CAN channel). The logical mailbox structure seen by the host controller matches very closely to the hardware register implementation used on the XC886MultiCanController. This results in reduced software overhead of the data management within the XC886MultiCanController firmware. The mailbox format defines the packet definition for the CAN frame data exchanged between the host and slave controllers over the SPI interface (GetCanObject / SetCanObject).

4.1 Mailbox Layout (2+20 bytes):

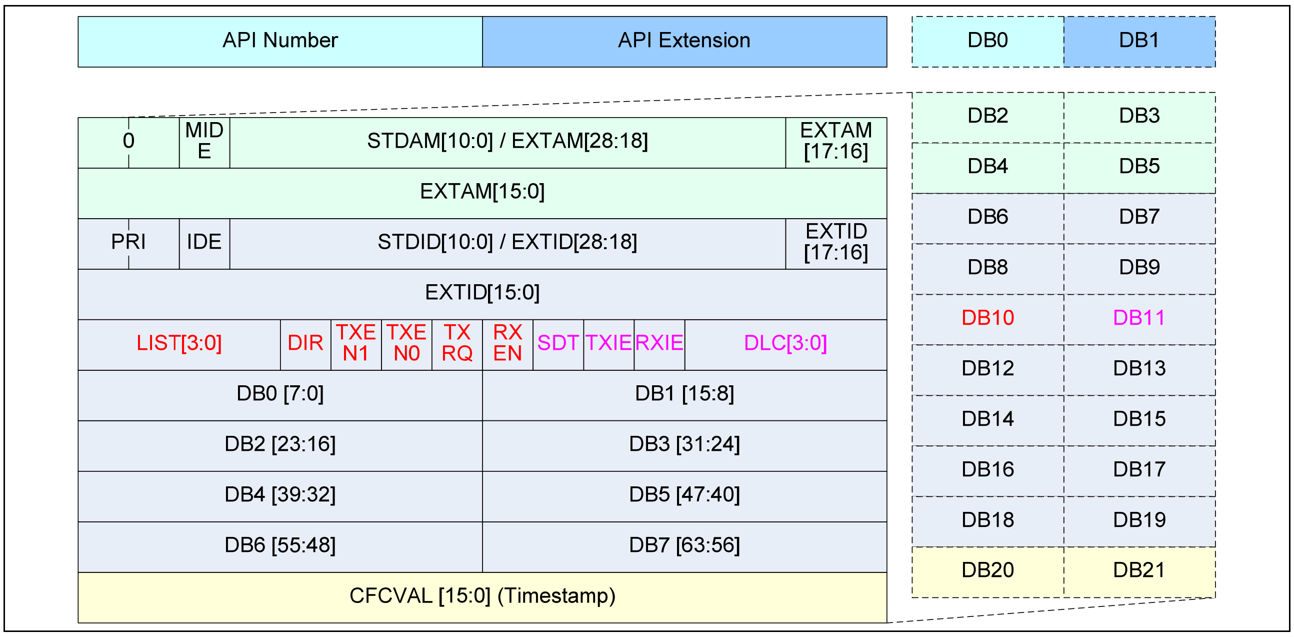


Figure 8 Mailbox Layout

Table 7 Mailbox Registers

Field	Bits	Description
MIDE	157	**Acceptance Mask bit for Message IDE bit** 0 This message object accepts the reception of both standard and extended frames 1 This message object only receives frames with a matching IDE bit
STDAM/EXTAM	[156:128]	**Acceptance Mask for CAN Message Identifier** Identifier of a standard message (xID[28:18]) or an extended message (xID[28:0]). For standard identifiers bits xID[17:0] are “don’t care”.
PRI	[127:126]	**Priority Class** PRI assigns one of the four priority classes 0, 1, 2, 3 to the message object, with lower PRI number meaning higher priority. Message objects with lower PRI value always win acceptance filtering for frame reception and transmission over message objects with higher PRI value. Acceptance filtering based on identifier/mask and list position is only performed between message objects of the same priority class. PRI also defines the acceptance filtering method for transmission: 00 Time Triggered CAN (Do Not Use) 01 Transmit acceptance filtering is based on the list order 00 Transmit acceptance filtering is based on the CAN identifier 11 Transmit acceptance filtering is based on the list order
IDE	125	**CAN IDE bit of Message Object** 0 Standard frame with 11-bit identifier 1 Extended frame with 29-bit identifier
STDID/EXTID	[124:96]	**CAN Identifier of Message Object** Identifier of a standard message (xID[28:18]) or an extended message (xID[28:0]). For standard identifiers bits xID[17:0] are “don’t care”.
LIST	[95:92]	**List Allocation** This field indicates the list to which the message object is allocated 0000 Unallocated object 0001 Allocated to list 1 (CAN channel 1) 0010 Allocated to list 2 (CAN channel 2) else reserved
DIR	91	**Message Direction** 0 Receive Object. With TXRQ = 1 a remote frame with the identifier of the message object is scheduled for transmission. On reception of a data frame with matching identifier the message is stored in this message object. 1 Transmit Object. If TXRQ = 1 this message object is scheduled for transmission as a data frame. On reception of a remote frame with matching identifier the TXRQ bit is set.
TXEN1	90	**Transmit Enable 1** The message object may only be transmitted if both TXEN0 and TXEN1 are set. TXEN1 is used by the MultiCAN module for selecting the active message object in transmit FIFO’s.
TXEN0	89	**Transmit Enable 0** The message object may only be transmitted if both TXEN0 and TXEN1 are set. The user may clear TXEN0 in order to inhibit the transmission of a message that is currently updated or to disable automatic response of remote frames.
TXRQ	88	**Transmit Request** 0 No transmission of message object n is requested. 1 Transmission of message object n on the CAN bus is requested. The transmit request becomes valid only if TXRQ, TXEN0, TXEN1 and MSGVAL are set. TXRQ is set by hardware if a matching remote frame has been received correctly. TXRQ is reset by hardware if message object n has been transmitted successfully and NEWDAT is not set again by software.
RXEN	87	**Receive Enable** The message object may only receive a frame from the CAN bus if RXEN is set. The message object does not match any receiving frame if RXEN is not set. RXEN is only evaluated for receive acceptance filtering.
SDT	86	**Single Data Transfer** If SDT = 1 and message object n is not a FIFO base object, then MSGVAL is reset when this object has taken part in a successful data transfer (receive or transmit).

Field	Bits	Description
TXIE	85	**Transmit Interrupt Enable** TXIE enables the message transmit interrupt of message object n. This interrupt is generated after the transmission of a CAN message. 0 Message transmit interrupt is disabled. 1 Message transmit interrupt is enabled. Bit field MOIPRn.TXINP selects the interrupt output line which becomes activated at this type of interrupt.
RXIE	84	**Receive Interrupt Enable** RXIE enables the message receive interrupt of message object n. This interrupt is generated after reception of a CAN message (independent of whether the CAN message is received directly or indirectly via a gateway action). 0 Message receive interrupt is disabled. 1 Message receive interrupt is enabled. Bit field MOIPRn.RXINP selects the interrupt output line which becomes activated at this type of interrupt.
DLC	[83:80]	**Data Length Code** Valid values for the data length are 0 to 8. DLC>8 leads to 8 data bytes, but the DLC code is not truncated upon reception or transmission of CAN frames.
DB0	[79:72]	**CAN Data Byte 0 [7:0]**
DB1	[71:64]	**CAN Data Byte 1 [15:8]**
DB2	[63:56]	**CAN Data Byte 2 [23:16]**
DB3	[55:48]	**CAN Data Byte 3 [31:24]**
DB4	[47:40]	**CAN Data Byte 4 [39:32]**
DB5	[39:32]	**CAN Data Byte 5 [47:40]**
DB6	[31:24]	**CAN Data Byte 6 [55:48]**
DB7	[23:16]	**CAN Data Byte 7 [63:56]**
CFCVAL	[15:0]	**Time Stamp** In CAN bit time

4.2 Structure for a 32-bit Little Endian machine (TriCore)

```
typedef struct {
  unsigned int  w      :29;
  unsigned int  mide   :1;
  unsigned int         :2;
} CAN_OBJ_AM_EXT_TYPE;

typedef struct {
  unsigned int  w      :29;
  unsigned int  ide    :1;
  unsigned int  pri    :2;
} CAN_OBJ_ID_EXT_TYPE;

typedef struct {
  unsigned int         :18;
  unsigned int  w      :11;
  unsigned int  mide   :1;
  unsigned int         :2;
} CAN_OBJ_AM_STD_TYPE;

typedef struct {
  unsigned int         :18;
  unsigned int  w      :11;
  unsigned int  ide    :1;
  unsigned int  pri    :2;
} CAN_OBJ_ID_STD_TYPE;

typedef struct {
  unsigned int  dlc    :4;
  unsigned int  rxie   :1;
  unsigned int  txie   :1;
  unsigned int  sdt    :1;
  unsigned int  rxen   :1;
  unsigned int  txrq   :1;
  unsigned int  txen0  :1;
  unsigned int  txen1  :1;
  unsigned int  dir    :1;
  unsigned int  list   :4;
} CAN_OBJ_CFG_TYPE;

typedef struct {
  union {
    U32 w;
    CAN_OBJ_AM_STD_TYPE std;
    CAN_OBJ_AM_EXT_TYPE ext;
  } am;                /* acceptance mask */

  union {
    U32 w;
    CAN_OBJ_ID_STD_TYPE std;
    CAN_OBJ_ID_EXT_TYPE ext;
  } id;                /* identifier */

  union {
    U16 hw;
    CAN_OBJ_CFG_TYPE b;
  } cfg;               /* object configuration */

  U8  db[8];           /* data bytes */
  U16 timestamp;       /* object time stamp */
} CAN_MAILBOX_TYPE;
```

4.3 API to communicate between host and slave controllers

Table 8, lists the available messages for communication between the host and the slave (XC886MultiCanController). The list can be expanded to meet future needs.

Table 8 (API) Description with byte lengths for the messages

Name	SSC (SPI) Bytes	UART Bytes	
		In	Out
API 0: NOP	22	4	4
API 1: SetCanChannelOnOff	22	6	-
API 2: GetCanChannelOnOff	22	5	6
API 3: SetCanIrqOnOff	22	5	-
API 4: GetCanIrqOnOff	22	5	6
API 5: SetCanCounter	22	9	-
API 6: GetCanCounter	22	5	9
API 7: GetCanIrqStatus	22	5	5
API 8: SetCanObject	22	25	-
API 9: GetCanObject	22	5	25
API 10: SetCanBitRate	22	7	-
API 11: GetCanBitRate	22	5	7
API 12: SetCanRegData	22	11	-
API 13: GetCanRegData	22	7	10
API 14: SetCpuClock	22	5	-
API 15: GetCpuClock	22	4	5

The SSC transfers are always of a fixed length (22 bytes) and support full-duplex communication. A two byte header begins the transaction and is followed by specific mailbox data (20 bytes) releated to the header.

Note: For SSC transfers all bytes depend on the type of message and will be "don't care". The API will always fill from the zero byte position out to the length specified by the message API. The software does not explicitly clear unused mailbox data to zero.

*Note: For UART transfers the user can omit all dummy transfers and instead must wrap the API message frame within the UART message packet as shown in **Figure 5**.*

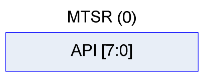
Note: API's 7 and 14 are not implemented in the accompanying source code for the XC886MultiCanController. Only the framework for these API's exists in the source code.

4.3.1 API 0: NOP

Description:

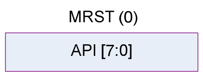
This means No OPeration and the user should ignore the message as the data is not valid.

4.3.1.1 MTSR Packet Description



Field	Byte	Bits	Description
API	0	[6:0]	**API Function ID** 0x00 No OPeration
-	*1-21*	*[7:0]*	***Dummy transfer***

4.3.1.2 MRST Packet Description



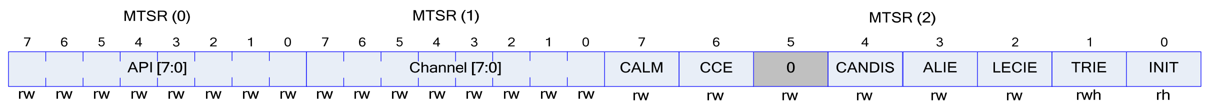
Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID** 0x00 No OPeration
-	*1-21*	*[7:0]*	***Unused data***

4.3.2 API 1: SetCanChannelOnOff

Description:

Enable / Disable CAN channel 1 (2)

4.3.2.1 MTSR Packet Description



Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID** 0x01 Enable / Disable CAN channel
CH	1	[7:0]	**Channel** 0 Channel 1 1 Channel 2
INIT	2	[0]	**Node Initialization** 0 Module is initialized (messages with participate) 1 Module is inactive or in Bus off state
TRIE	2	[1]	**Transfer Interrupt Enable** 0 Transfer interrupt is disabled 1 Transfer interrupt is enabled
LECIE	2	[2]	**LEC Indicated Error Interrupt Enable** 0 Last error code interrupt is disabled 1 Last error code interrupt is enabled
ALIE	2	[3]	**Alert Interrupt Enable** 0 Alert interrupt is disabled 1 Alert interrupt is enabled
CANDIS	2	[4]	**CAN Disable** Setting this bit disables the CAN node. The CAN node first waits until it is bus-idle or bus-off. Then bit INIT is automatically set, and an alert interrupt is generated if bit ALIE is set
CCE	2	[6]	**Configuration Change Enable** 0 The Bit Timing Register, the Port Control Register, and the Error Counter register may only be read. All attempts to modify them are ignored. 1 The Bit Timing Register, the Port Control Register, and the Error Counter Register may be read and written
CALM	2	[7]	**CAN Analyze Mode** If this bit is set, then the CAN node operates in Analyze Mode. This means that messages may be received, but not transmitted. No acknowledge is sent on the CAN bus upon frame reception. Active-error flags are sent recessive instead of dominant. The transmit line is continuously held at recessive (1) level. Bit CALM can be written only while bit INIT is set.
-	*3-21*	*[7:0]*	***Dummy Transfer***

4.3.2.2 MRST Packet Description

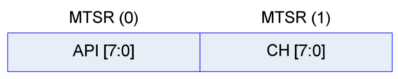
There is no transmit packet for this API

4.3.3 API 2: GetCanChannelOnOff

Description:

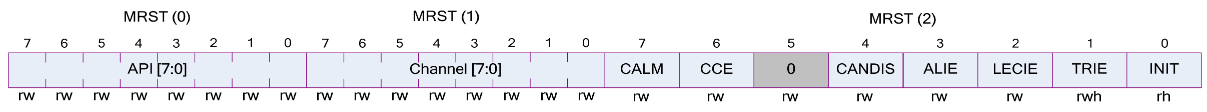
Get CAN channel 1 (2) Status

4.3.3.1 MTSR Packet Description



Field	Byte	Bits	Description
API	0	[6:0]	**API Function ID** 0x02 Read CAN channel status (enabled or disabled)
CH	1	[7:0]	**Channel** 1 Channel 1 2 Channel 2 else reserved
-	*2-21*	*[7:0]*	***Dummy transfer***

4.3.3.2 MRST Packet Description



Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID** 0x02 Read CAN channel status
CH	1	[7:0]	**Channel** 0 Channel 1 1 Channel 2
INIT	2	[0]	**Node Initialization** 0 Module is initialized (messages with participate) 1 Module is inactive or in Bus off state
TRIE	2	[1]	**Transfer Interrupt Enable** 0 Transfer interrupt is disabled 1 Transfer interrupt is enabled
LECIE	2	[2]	**LEC Indicated Error Interrupt Enable** 0 Last error code interrupt is disabled 1 Last error code interrupt is enabled
ALIE	2	[3]	**Alert Interrupt Enable** 0 Alert interrupt is disabled 1 Alert interrupt is enabled
CANDIS	2	[4]	**CAN Disable** Setting this bit disables the CAN node. The CAN node first waits until it is bus-idle or bus-off. Then bit INIT is automatically set, and an alert interrupt is generated if bit ALIE is set
CCE	2	[6]	**Configuration Change Enable** 0 The Bit Timing Register, the Port Control Register, and the Error Counter register may only be read. All attempts to modify them are ignored. 1 The Bit Timing Register, the Port Control Register, and the Error Counter Register may be read and written
CALM	2	[7]	**CAN Analyze Mode** If this bit is set, then the CAN node operates in Analyze Mode. This means that messages may be received, but not transmitted. No acknowledge is sent on the CAN bus upon frame reception. Active-error flags are sent recessive instead of dominant. The transmit line is continuously held at recessive (1) level. Bit CALM can be written only while bit INIT is set.
-	*3-21*	*[7:0]*	***Unused data***

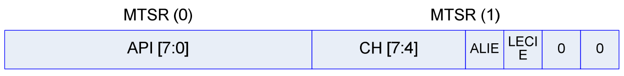
4.3.4 API 3: SetCanIrqOnOff

Description:

Enable / Disable CAN channel 1 (2) Interrupts

Note: Transmit and Receive interrupts are individually enabled via the mailbox for the corresponding object.

4.3.4.1 MTSR Packet Description



Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID**
			0x03 Enable / Disable CAN channel interrupts
CH	1	[7:4]	**Channel**
			1 Channel 1
			2 Channel 2
			else reserved
ALIE	1	3	**Alert Interrupt Enable**
			0 disable
			1 enable
LECIE	1	2	**Last Error Code indicated Error Interrupt Enable**
			0 disable
			1 enable
-	1	1	**Always write 0**
-	1	0	**Always write 0**
-	*2-21*	*[7:0]*	***Dummy transfer***

4.3.4.2 MRST Packet Description

There is no transmit packet for this API

4.3.5 API 4: GetCanIrqOnOff

Description:

Get status of CAN channel 1 (2) Interrupt

Note: Transmit and Receive interrupts are individually enabled via the mailbox for the corresponding object.

4.3.5.1 MTSR Packet Description

MTSR (0)	MTSR (1)
API [7:0]	CH [7:0]

Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID** 0x04 Read CAN channel interrupt status
CH	1	[7:0]	**Channel** 1 Channel 1 2 Channel 2 else reserved
-	*2-21*	*[7:0]*	***Dummy transfer***

4.3.5.2 MRST Packet Description

MRST (0)	MRST (1)	MRST (2)				
API [7:0]	CH [7:4]	-	ALIE	LECIE	-	-

Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID** 0x04 Read CAN channel interrupt status
CH	1	[7:0]	**Channel** 1 Channel 1 2 Channel 2 else reserved
-	2	[7:4]	**Not used**
ALIE	2	3	**Alert Interrupt** 0 disabled 1 enabled
LECIE	2	2	**Last Error Code indicated Error Interrupt** 0 disabled 1 enabled
-	2	1	**Not Used**
-	2	0	**Not Used**
-	*3-21*	*[7:0]*	***Unused data***

4.3.6 API 5: SetCanCounter

Description:

Write CAN Channel Counter

4.3.6.1 MTSR Packet Description

MTSR (0)	MTSR (1)		MTSR (2)	MTSR (3)	MTSR (4)	MTSR (5)
API [7:0]	CH [7:4]	CNTY [3:0]	DB3 [31:24]	DB2 [23:16]	DB1 [15:8]	DB0 [7:0]

Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID** 0x05 Write CAN channel counter
CH	1	[7:4]	**Channel** 1 Channel 1 2 Channel 2 else reserved
CNTY	1	[3:0]	**Counter Type** 1 CAN frames received 2 CAN frames transmitted 3 CAN errors received 4 CAN errors transmitted 5 CAN Alert errors 6 CAN LEC errors else reserved
DB3	2	[7:0]	**Counter Data Byte 3 [31:24]**
DB2	3	[7:0]	**Counter Data Byte 2 [23:16]**
DB1	4	[7:0]	**Counter Data Byte 1 [15:8]**
DB0	5	[7:0]	**Counter Data Byte 0 [7:0]**
-	*6-21*	*[7:0]*	***Dummy transfer***

4.3.6.2 MRST Packet Description

There is no transmit packet for this API

4.3.7 API 6: GetCanCounter

Description:

Read CAN Channel Counter

4.3.7.1 MTSR Packet Description

MTSR (0)	MTSR (1)	
API [7:0]	CH [7:4]	CNTY [3:0]

Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID**
			0x06 Read CAN channel counter
CH	1	[7:4]	**Channel**
			1 Channel 1
			2 Channel 2
			else reserved
CNTY	1	[3:0]	**Counter Type**
			1 CAN frames received
			2 CAN frames transmitted
			3 CAN errors received
			4 CAN errors transmitted
			5 CAN Alert errors
			6 CAN LEC errors
			else reserved
-	*2-21*	*[7:0]*	***Dummy transfer***

4.3.7.2 MRST Packet Description

MRST (0)	MRST (1)		MRST (2)	MRST (3)	MRST (4)	MRST (5)
API [7:0]	CH [7:4]	CNTY [3:0]	DB3 [31:24]	DB2 [23:16]	DB1 [15:8]	DB0 [7:0]

Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID**
			0x06 Read CAN channel counter
CH	1	[7:4]	**Channel**
			1 Channel 1
			2 Channel 2
			else reserved
CNTY	1	[3:0]	**Counter Type**
			1 CAN frames received
			2 CAN frames transmitted
			3 CAN errors received
			4 CAN errors transmitted
			else reserved
DB3	2	[7:0]	**Counter Data Byte 3 [31:24]**
DB2	3	[7:0]	**Counter Data Byte 2 [23:16]**
DB1	4	[7:0]	**Counter Data Byte 1 [15:8]**
DB0	5	[7:0]	**Counter Data Byte 0 [7:0]**
-	*6-21*	*[7:0]*	***Unused data***

4.3.8 API 7: GetCanIrqStatus

Description:

Get status of CAN channel 1 (2) interrupts

Not implemented in application note

4.3.8.1 MTSR Packet Description

MTSR (0)	MTSR (1)
API [7:0]	CH [7:0]

Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID** 0x07 CAN channel source interrupt status
CH	1	[7:0]	**Channel** 1 Channel 1 2 Channel 2 else reserved
-	*2-21*	*[7:0]*	***Dummy transfer***

4.3.8.2 MRST Packet Description

MRST (0)	MRST (1)				
API [7:0]	CH [7:4]	RXIE	TXIE	ALIE	LECIE

Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID** 0x07 Read CAN channel interrupt status
CH	1	[7:4]	**Channel** 1 Channel 1 2 Channel 2 else reserved
RXIE	1	3	**Receive Interrupt Pending** 0 false 1 true Cleared by slave firmware once the message object has been read by the host
TXIE	1	2	**Transmit Interrupt Pending** 0 false 1 true Cleared by slave firmware once the message object has been successfully transmitted on the CAN bus.
ALIE	1	1	**Alert Interrupt Pending** 0 false 1 true Cleared by slave firmware TBD
LECIE	1	0	**Last Error Code indicated Error Interrupt Pending** 0 false 1 true Cleared by slave firmware TBD
-	*2-21*	*[7:0]*	***Unused data***

4.3.9 API 8: SetCanObject

Description:

Write CAN channel mailbox data

For the description of the mailbox data see **Table 7** Mailbox Registers

4.3.9.1 MTSR Packet Description

MTSR (0)	MTSR (1)	MTSR (2)	MTSR (21)
API [7:0]	OBJ [7:0]	Mailbox MSB	Mailbox LSB

Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID (DB0)** 0x08 Write CAN mailbox data (message object)
OBJ	1	[7:0]	**Message Object Number (DB1)**
-	2	[7:0]	**Mailbox MSB (DB2)**
-	…	…	**…**
-	21	[7:0]	**Mailbox LSB (DB21)**

4.3.9.2 MRST Packet Description

There is no transmit packet for this API

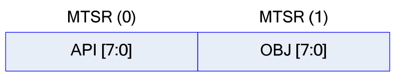
4.3.10 API 9: GetCanObject

Description:

Read CAN channel mailbox data

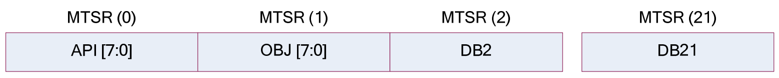
For the description of the mailbox data see **Table 7** Mailbox Registers

4.3.10.1 MTSR Packet Description



Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID** 0x09 Read CAN mailbox data
OBJ	1	[7:0]	**Message Object Number**
-	*2-21*	*[7:0]*	***Dummy transfer***

4.3.10.2 MRST Packet Description

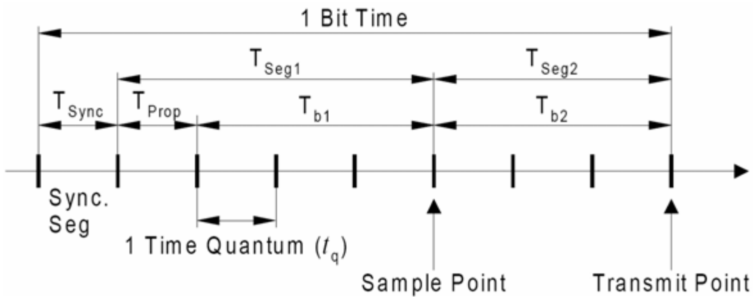


Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID (DB0)** 0x09 Read CAN mailbox data
OBJ	1	[7:0]	**Message Object Number (DB1)**
-	2	[7:0]	**Mailbox MSB (DB2)**
-	…	…	**…**
-	21	[7:0]	**Mailbox LSB (DB21)**

4.3.11 API 10: SetCanBitRate

Description:

Write CAN channel (Node Bit Timing Register NBTR)



t_q	= (BRP + 1) / fCAN if DIV8 = 0		(fCAN = 48MHz)
	= 8 × (BRP+1) / fCAN if DIV8 = 1		
T_{Sync}	= 1 × t_q		
T_{Seg1}	= (TSEG1 + 1) × t_q	(min. 3 t_q)	
T_{Seg2}	= (TSEG2 + 1) × t_q	(min. 2 t_q)	

$T_{SJW} = (SJW + 1) \times t_q$
$T_{Seg1} \geq T_{SJW} + T_{prop}$
$T_{Seg2} \geq T_{SJW}$

bit time = $T_{Sync} + T_{Seg1} + T_{Seg2}$ (min. 8 t_q)

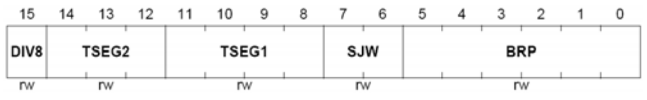
Forumla for CAN baud rate (bit time):

baud rate = fCAN / ((BRP + 1) x ($T_{Sync} + T_{Seg1} + T_{Seg2}$))

Example for 250 Kbaud: BTR = 0x494B

4.3.11.1 MTSR Packet Description

MTSR (0)	MTSR (1)	MTSR (2)	MTSR (3)
API [7:0]	CH [7:0]	NBTRH[15:8]	NBTRL[7:0]



Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID** 0x0A Write CAN channel (Node Bit Timing Register NBTR)
CH	1	[7:0]	**Channel** 1 Channel 1 2 Channel 2 else reserved
NBTRH	2	[7:0]	**NBTR [15:8]**
NBTRL	3	[7:0]	**NBTR [7:0]**
-	*4-21*	*[7:0]*	***Dummy transfer***

4.3.11.2 MRST Packet Description

There is no transmit packet for this API

4.3.12 API 11: GetCanBitRate

Description:

Read CAN channel (Node Bit Timing Register NBTR). See API 10: SetCanBitRate for definition of the NBTR value.

4.3.12.1 MTSR Packet Description

MTSR (0)	MTSR (1)
API [7:0]	CH [7:0]

Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID** 0x0B Read CAN channel (Node Bit Timing Register NBTR)
CH	1	[7:0]	**Channel** 1 Channel 1 2 Channel 2 else reserved
-	*2-21*	*[7:0]*	***Dummy transfer***

4.3.12.2 MRST Packet Description

MRST (0)	MRST (1)	MRST (2)	MTSR (3)
API [7:0]	CH [7:0]	NBTRH[15:8]	NBTRL[7:0]

Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID** 0x0B Read CAN channel (Node Bit Timing Register NBTR)
CH	1	[7:0]	**Channel** 1 Channel 1 2 Channel 2 else reserved
NBTRH	2	[7:0]	**NBTR [15:8]**
NBTRL	3	[7:0]	**NBTR [7:0]**
-	*2-21*	*[7:0]*	***Unused data***

For description of NBTR bits see API 10: SetCanBitRate

4.3.13 API 12: SetCanRegData

Description:

Write CAN channel register (32-bits)

4.3.13.1 MTSR Packet Description

MTSR (0)	MTSR (1)	MTSR (2)	MTSR (3)	MTSR (4)	MTSR (5)	MTSR (6)	MTSR (7)
API [7:0]	Data3[31:24]	Data2[23:16]	Data1[15:8]	Data0[7:0]	ADDRH[13:10]	ADDRL[9:2]	ADCON [7:0]

Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID** 0x0C Write CAN channel register (32-bits)
DATA3	1	[7:0]	**CAN Data [31:24]**
DATA2	2	[7:0]	**CAN Data [23:16]**
DATA1	3	[7:0]	**CAN Data [15:8]**
DATA0	4	[7:0]	**CAN Data [7:0]**
-	5	[7:4]	**Must be zero**
ADH	5	[4:0]	**CAN Address [13:10]**
ADL	6	[7:0]	**CAN Address [9:2]**
ADCON	7	[7:0]	**CAN Address Control Register**
-	*8-21*	*[7:0]*	***Dummy transfer***

4.3.13.2 MRST Packet Description

There is no transmit packet for this API

4.3.14 API 13: GetCanRegData

Description:

Read CAN channel register (32-bits)

4.3.14.1 MTSR Packet Description

MTSR (0)	MTSR (1)	MTSR (2)	MTSR (3)
API [7:0]	ADDRH[13:10]	ADDRL[9:2]	ADCON [7:0]

Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID** 0x0D Read CAN channel register (32-bits)
-	1	[7:4]	**Must be zero**
ADH	1	[4:0]	**CAN Address [13:10]**
ADL	2	[7:0]	**CAN Address [9:2]**
ADCON	3	[7:0]	**CAN Address Control Register**
-	*4-21*	*[7:0]*	***Dummy transfer***

4.3.14.2 MRST Packet Description

MRST (0)	MRST (1)	MRST (2)	MTSR (3)	MTSR (4)	MTSR (5)	MTSR (6)
API [7:0]	Data3[31:24]	Data2[23:16]	Data1[15:8]	Data0[7:0]	ADDRH[13:10]	ADDRL[9:2]

Field	Byte	Bits	Description
API	1	[7:0]	**API Function ID** 0x0D Read CAN channel register (32-bits)
DATA3	2	[7:0]	**CAN Data [31:24]**
DATA2	3	[7:0]	**CAN Data [23:16]**
DATA1	4	[7:0]	**CAN Data [15:8]**
DATA0	5	[7:0]	**CAN Data [7:0]**
ADH	6	[4:0]	**CAN Address [13:10]**
ADL	7	[7:0]	**CAN Address [9:2]**
-	*8-21*	*[7:0]*	***Unused data***

4.3.15 API 14: SetCpuClock

Description:

Change the CPU Clock Frequency

Not implemented in application note

4.3.15.1 MTSR Packet Description

MTSR (0)	MTSR (1)
API [7:0]	CLKSEL

Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID** 0x0E Change the CPU Clock Frequency
CLKSEL	1	[7:0]	**CPU Clock Frequency Selection** 0 9.6MHz internal oscillator (default) 1 4 MHz external oscillator 2 6 MHz external oscillator 3 8 MHz external oscillator 4 12 MHz external oscillator else reserved
-	*2-21*	*[7:0]*	***Dummy transfer***

4.3.15.2 MRST Packet Description

There is no transmit packet for this API

4.3.16 API 15: GetCpuClock

Description:

Read CPU Clock Frequency

4.3.16.1 MTSR Packet Description

MTSR (0)
API [7:0]

Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID** 0x0F Read the CPU Clock Frequency
-	*4-21*	*[7:0]*	***Dummy transfer***

4.3.16.2 MRST Packet Description

MRST (0)								MRST (1)							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
API [7:0]								NDIV				VCOBYP	OSCDISC	RESLD	LOCK
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rwh	rh

Field	Byte	Bits	Description
API	0	[7:0]	**API Function ID** 0x0F Read the CPU Clock Frequency
NDIV	1	[7:4]	**PLL N-Divider** 0000 N = 10 0001 N = 12 0010 N = 13 0011 N = 14 0100 N = 15 0101 N = 16 0110 N = 17 0111 N = 18 1000 N = 19 1001 N = 20 1010 N = 24 1011 N = 30 1100 N = 32 1101 N = 36 1110 N = 40 1111 N = 48`
VCOBYP	1	3	**PLL VCO Bypass Mode Select** 0 Normal operation (default) 1 VCO bypass mode (PLL output clock is derived from input clock divided by P- and K-dividers).
OSCDSC	1	2	**Oscillator Disconnect** 0 Oscillator is connected to the PLL. 1 Oscillator is disconnected from the PLL.
RESLD	1	1	**Restart Lock Detection** Setting this bit will reset the PLL lock status flag and restart the lock detection. This bit will automatically be reset to 0 and thus always be read back as 0. 0 No effect 1 Reset lock flag and restart lock detection
LOCK	1	0	**Restart Lock Detection** PLL Lock Status Flag 0 PLL is not locked. 1 PLL is locked.
-	*2-21*	*[7:0]*	***Unused data***

4.4 Data flow example (SSC mode)

1) The Host waits for the XC886MultiCanController to be ready; the hardware should have a weak pull down resistor connected to the CTS pin of the XC886. Once the XC886MultiCanController is ready it will drive the CTS pin high (logic ‘1’).

2) Host controller initializes the XC886MultiCanController:

• SetCanBitRate(1, 0x494B)	- set CAN channel 1 to 250 Kbaud
• SetCanBitRate(2, 0x494B)	- set CAN channel 2 to 250 Kbaud
• SetCanObject(MAILBOX_9)	- set mailbox (object 9 receive, CAN channel 1)
• SetCanObject(MAILBOX_25)	- set mailbox (object 25 receive, CAN channel 2)

Total number of bytes transferred:

4 (transfers) * 22 (bytes for each transfer) = 88 bytes

3) Host controller wishes to transmit a CAN frame on the XC886MultiCanController’s CAN channel 1

- MailBox_8 will be used for a transmit object on CAN channel 1.

If “CTS” signal is high (logic ‘1’) then the host sends the message SetCanObject(MAILBOX_8)

The XC886MultiCanController will drive the CTS line low (logic ‘0’) indicating that a message has been received and queued for transmission as a CAN message.

Once the XC886MultiCanController has configured the CAN object, the XC886MultiCanController will assert the CTS pin (logic ‘1’). This indicates it’s ready to accept another message from the host.

Total number of bytes transferred: 22 bytes

4) Host controller receives a CAN frame from XC886MultiCanController (either channel).

The XC886MultiCanController receives a message object according to its acceptance mask.

The “DA” line is driven high (logic ‘1’) by the XC886MultiCanController indicating its transfer buffer has data ready for the host to read.

The Interrupt line from XC886MultiCanController to host controller is driven low (logic ‘0’) by the XC886MultiCanController.

The host controller’s interrupt service routine checks if both the “DA” and “CTS” signals are high (logic ‘1’) and then performs a message transfer with the XC886MultiCanController.

Total number of bytes transferred: 22 bytes (note: the send queue on the XC886MultiCanController is double buffered and may result in two transfers to get the received message (44 bytes).

4.5 Bandwidth calculations (SPI)

Worst case bus loading would be receiving simultaneous message objects on both nodes at 100% bus utilization (with zero data bytes and no additional stuff bits).

The needed SPI throughput for 22 bytes in a packet (176 bits) is:

For a standard message object this would occur within 46-bit time quanta.

((176) /46) * 2 = 7.65 Mbit

For an extended message object this would occur within 67-bit time quanta.

((176) / 67) * 2 = 5.25 Mbit

Assuming 8 data bytes (and no additional stuff bits):

((176) / (46+64)) * 2 = 3.20 Mbit for a Standard object

((176) / (67+64)) * 2 = 2.69 Mbit for an Extended object

Note: assuming both CAN buses are operating at 1MBit, SPI transfer overhead is not included

For 250Kbaud CAN buses simply divide the 1MBaud values by 4.

Additional messages can be added to reduce the host/slave packet lengths.

4.6 Logic Analyzer Diagrams of some transfers

4.6.1 Transmit a message on the CAN bus

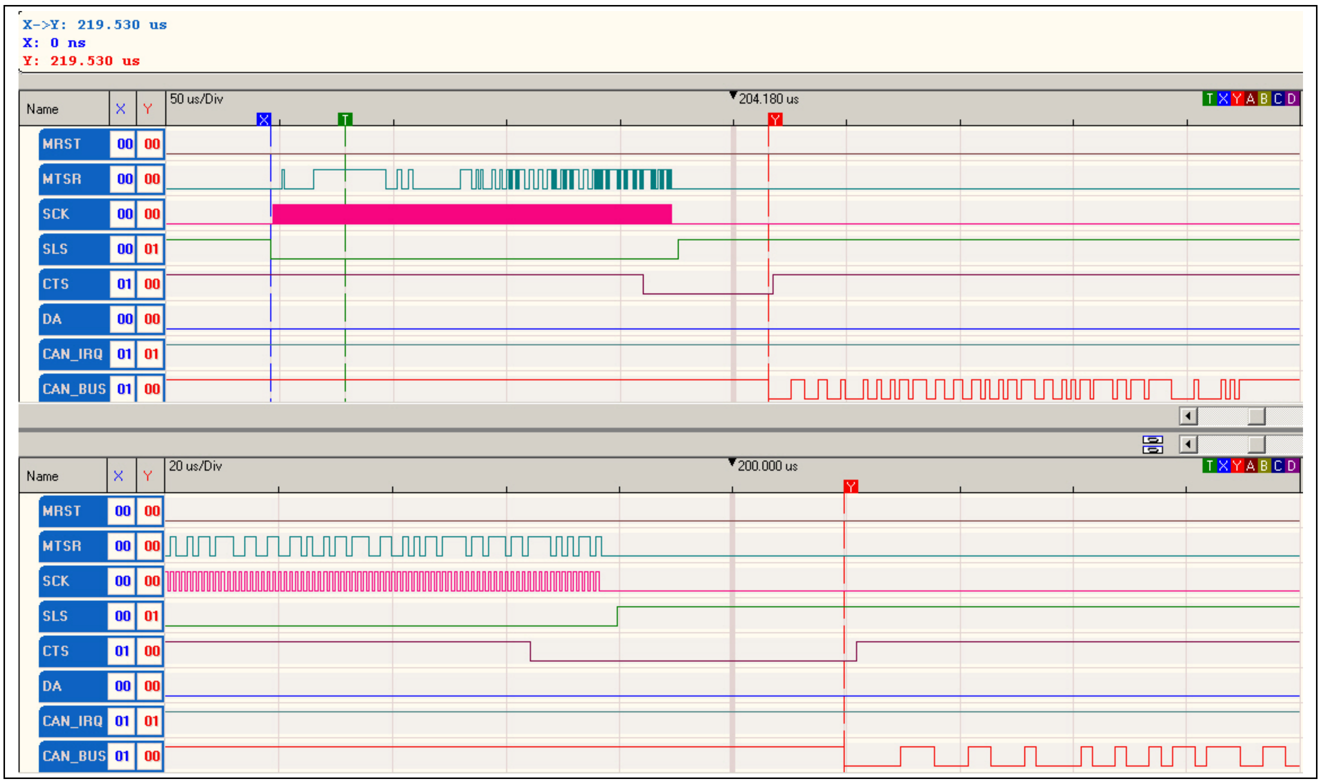


Figure 9 CAN Transmit

4.6.2 Receive a message from the CAN bus

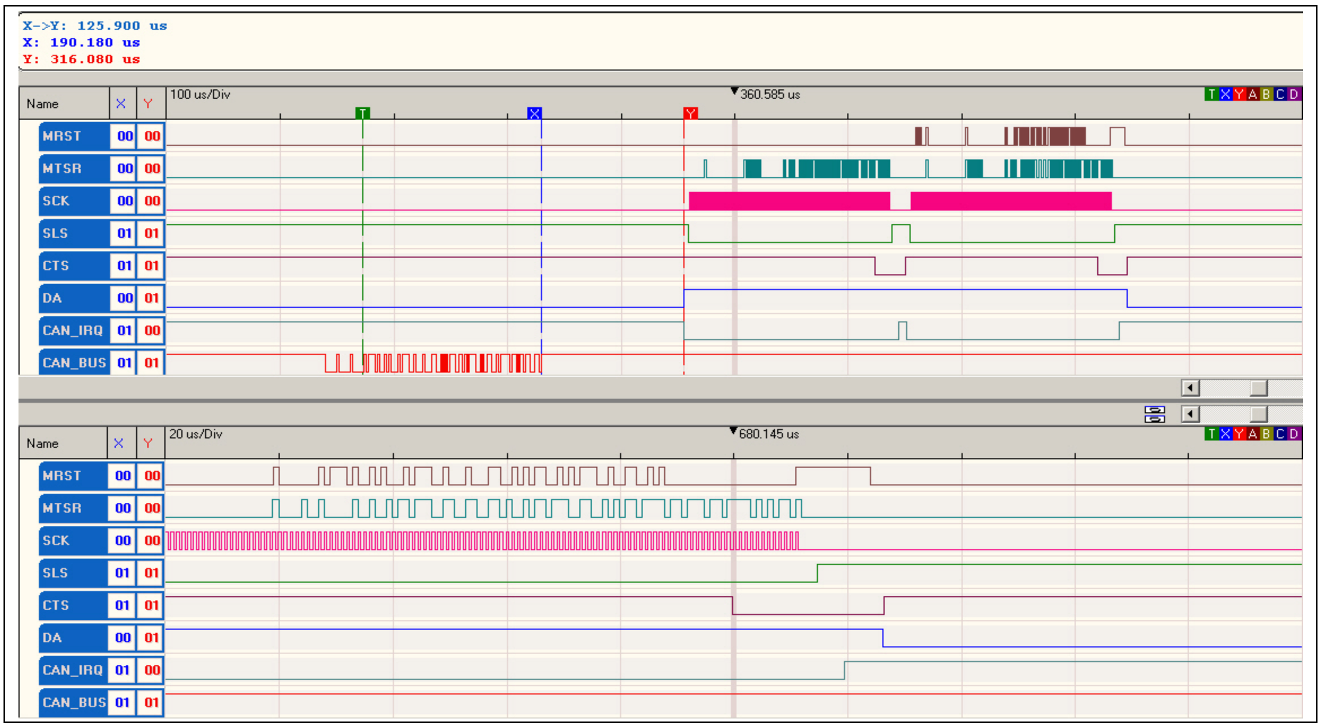


Figure 10 CAN Receive

4.6.3 Read data from a register on MultiCAN

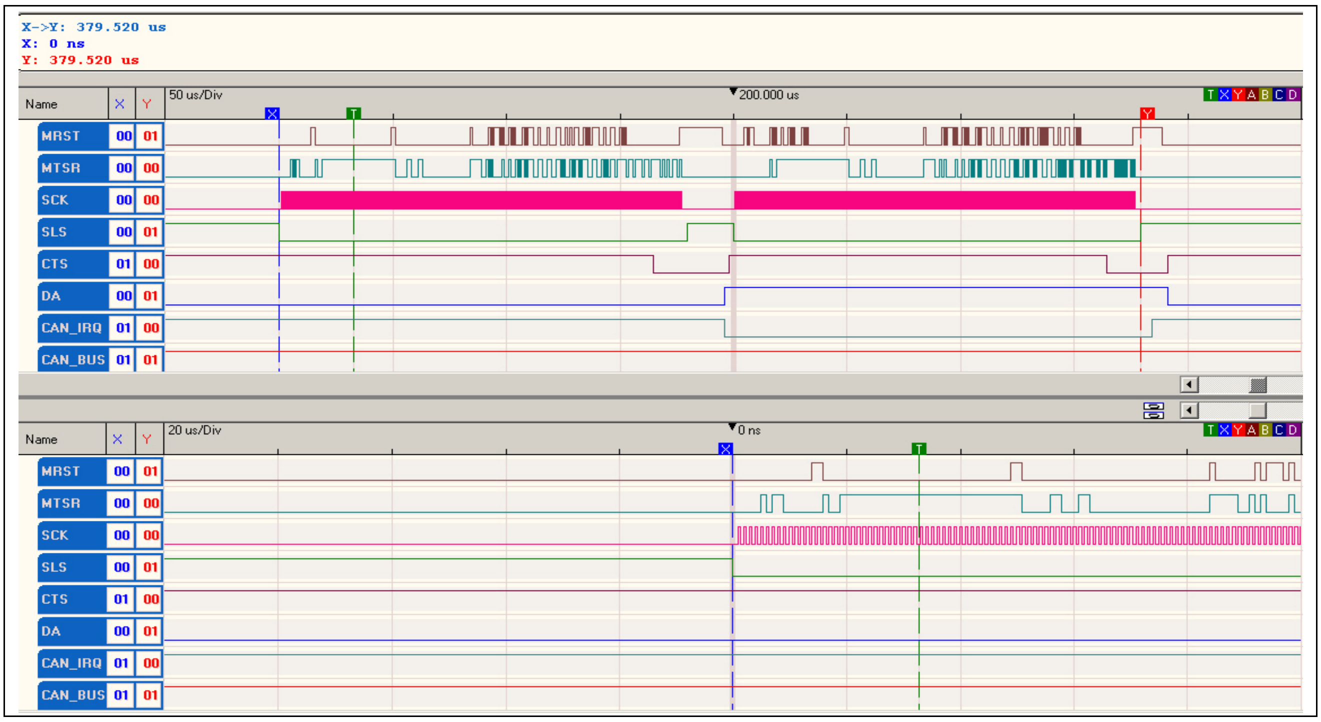


Figure 11 Read

http://www.infineon.com

Published by Infineon Technologies AG