XMC4000

32-bit Microcontroller Series for Industrial Applications

CAN Bootstrap Loader (BSL)

AP32269

Application Note

Scope and purpose

The on-chip firmware stored in the BootROM is the first software to be executed by the CPU after reset. The BootROM of the XMC4000 family provides a number of different boot modes including the CAN Bootstrap Loader (BSL) mode.

Using the CAN Bootstrap Loader it is possible to load code or data into the internal RAM of the device via the MultiCAN interface. This document describes how to program user code into the internal flash of XMC4000 devices via the CAN BSL.

Applicable Products

- XMC4000 Microcontroller Family

References

The example code “XMC4000 – CAN Bootstrap Loader (BSL) – Example Code” can be downloaded from **www.infineon.com/XMC**.

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

Table of Contents

1 CAN BSL in the XMC4000 family

A built-in CAN Bootstrap Loader (BSL) is implemented in the Infineon XMC4000 family of devices.

The CAN BSL provides a mechanism to load program code or data via the MultiCAN module into the PSRAM of a device, and to start executing the code loaded from a PSRAM address (address of the first transmitted byte).

The CAN BSL is an integrated mechanism that can be selected via port configuration during a system start after a Power-on Reset, or through write configuration registers, and trigger then a software reset.

The CAN BSL uses a CAN node 0 or 1 interface and configures the MultiCAN module to the baud rate of the **host** device.

Once communication has been established, the BSL receives a defined number of messages (the number defined in the **host**) for downloading the code or data. The received code or data is sequentially written to the PSRAM.

Once the download has finished, the program code that has been loaded is executed and the BSL is terminated.

The complete load sequence is based on the following three CAN standard frames:

- Initialization Frame
  - sent by the external **host** to XMC4000
- Acknowledge Frame
  - sent by the XMC4000 to the external host
- Data Frames
  - sent by the external **host** to the XMC4000

The initialization frame is used in the XMC4000 device for baud rate detection.

After successful baud rate detection, the XMC4000 reports to the external host via the acknowledge frame.

Data frames can then be transmitted from the host to the XMC device. Data frames are always transmitted in 8 bytes.

Table 1 CAN Bootstrap Loader Frames

Frame Type	Parameter	Description
Initialization Frame	Identifier	11-bit ID=0x555
	DLC=8	
	Data byte 1-0	baud rate detection pattern (0x5555)
	Data byte 3-2	acknowledge message identifier ACK_ID (use bits [13:0], bit 13=IDE)
	Data byte 5-4	Data Message Count MSG_CNT For example; max. MSG_CNT=0x0800 for $C000_H$-$FFFF_H$ (16KBytes), see **section 1.2**
	Data byte 7-6	Data Message Identifier MSG_ID

Frame Type	Parameter	Description
Acknowledge Frame	Identifier	ACK_ID as received bytes [3:2] of the Initialization Frame
	DLC=8	
	Data byte 1-0	BTR register value in XMC4000
	Data byte 3-2	Data bytes [3.2] of the Initialization Frame
	Data byte 5-4	No error: copy of init. frame byte 5-4 Error: 0xADDE for error state
	Data byte 7-6	No error: copy of init. frame byte 7-6 Error: 0xEFBE for error state
Data Frame	Identifier	Data Message Identifier MSG_ID as sent by data bytes [7:6] of the Initialization Frame
	DLC=8	
	Data byte 7-0	Data Bytes, assigned to increasing destination (PSRAM) addresses

1.1 Entering CAN BSL mode

The XMC4000 enters CAN BSL mode, when the CAN BSL mode is selected within the startup configuration on receipt of a reset signal.

Bit field HWCON=11_B in the STCON register indicates the CAN BSL mode.

Bit field HWCON is updated either by hardware or software after a System Reset:

- Hardware
  - with a pull-up/down resistor when the configuration pins TCK, TMS have the value 10_B at Power-on reset, the value of TCK and (not) TMS is latched into bit field HWCON[1:0] in register STCON, and SWCON[9:8] is a copy of HWCON[1:0] after Power-on Reset.
- Software
  - when 0011_B is written into bit field STCON.SWCON [11:8] and a System Reset is triggered with AIRCR. SYSRESETREQ =1 (Software Boot configuration selection).

Note:

1. The System Reset SYSRESET can be triggered from various sources, such as a software controlled CPU reset control register AIRCR, a watchdog time-out-triggered reset (WDT) or a system trap triggered reset such as a non-maskable interrupt (NMI) for parity error.
2. The System Reset resets almost all logic in the core domain. The only exceptions in the core domain are RCU Registers and Debug Logic.

MultiCAN initialization with CAN BSL

- CAN node 0/1 of the MultiCAN module is enabled.
- P1.5 and P1.4 are defined as CAN Rx pins.
  - The signal on P1.5 and P1.4 will be monitored.
  - If the initialization frame has been detected on P1.4, then P1.5 is configured as a CAN Tx pin and CAN node 1 is deactivated.
  - If the initialization frame has been detected on P1.5, then P1.4 is configured as a CAN Tx pin and CAN node 0 is deactivated.
- Message Object 0 (MO0) is configured as Receive Object to receive the initialization and data frame.
- Message Object 1 (MO1) is configured as Transmit Object to transmit the acknowledge frame.
- In CAN BSL the system clock is provided by OSC_HP circuitry (supplied by an external oscillator or an external clock source).
  - $f_{PERIPH}=f_{OSCHP}$ is selected by SCU_OSCHPCTRL.MODE=00_B. SCU_SYSCLKCR.SYSSEL=1_B.

Note: For transfer rates of 1mbps, the user must ensure that the external clock runs at 10MHz at least.

During the initialization phase

- The Bit Timing Mode is used for baud rate detection and analysis of the bit timing.
- The Analyzer Mode is used to monitor the CAN frame without active participation.
- All Acceptance Mask bits of the receive object MO0 are ON.

After the initialization phase

- The Analyzer Mode is switched off. The bit timing mode for CAN node 0/1 does not change.
- BTR is set with the calculated value and SJW=11_B.
- All Acceptance Mask bits of the receive object MO0 are ON.

1.2 Loading the User Code

The sequence for loading the user code is:

- The code is downloaded starting at the PSRAM address ($1FFFC000_H$ in XMC4400).
  - Consecutive data bytes are stored at incrementing addresses.
- After receipt of the last CAN data frame, the Bootstrap Loader sequence is finished and executes a jump to the PSRAM address ($1FFFC000_H$ in XMC4400).
- The size of the downloaded application is only limited by the size of PSRAM on the device.
  - XMC4400 has 16K PSRAM ($1FFFC000_H$ - $1FFFFFFF_H$). The maximum CAN message count (MSG_CNT) in the initialization frame is 2048=0x0800.

1.3 Exiting CAN BSL

After the Bootstrap Loader has been activated, debugger access is disabled.

The debug system is released automatically when the BSL terminates after having received a set number of messages from the host (the number of messages is defined via a variable in the host).

The CAN BSL is also aborted after a Power On Reset, if the non-BSL port configuration has been used.

2 General System Description

As **host** device, one set of XMC4400 CPU + COM_ETH kit, is connected to a PC via a JTAG/UART interface. The XMC4400 should be in the normal boot mode.

As **slave** device, another set of XMC4400 CPU board + COM_ETH kit is connected to the **host** via a CAN bus. The XMC4400 should be in active CAN BSL mode.

This description is of the software for the **host** board. The host device takes control of the slave device via the CAN bus, while commands are received and process status changes are reported via the JTAG/UART interface to the PC COM port.

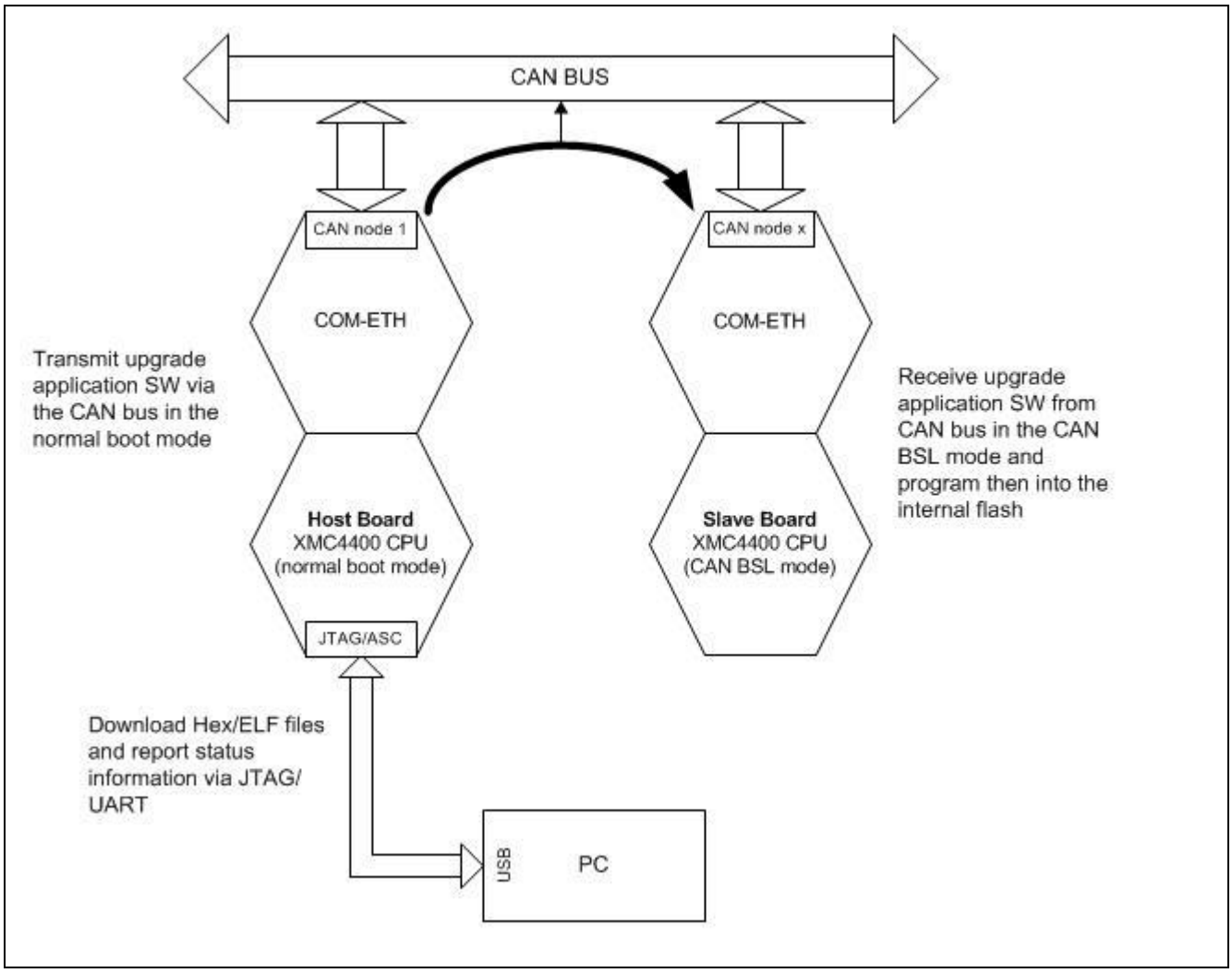

Figure 1 System Overview

There are three different pieces of code (DAVE3 projects) for the **host** device associated with this example:

- Transmitter Program: XMC4400_MasterCAN
  - Initializes the MultiCAN module in the **host** device (including the CAN transmit pins and configuring the initialization frame) to establish the communication with the slave.

    CPU_44A_V2 kit: CAN1_TxPin=P1.12; CAN1_RxPin=P1.4;
  - Initializes the USIC module and configures U0C0 as the ASC interface to communicate with the PC COM port.

    CPU_44A_V2 kit: U0C0_TxPin=P1.7; U0C0_RxPin=P1.5;
  - Modify standard I/O library function to give command and display status information via PC COM.
  - Defines a port pin P1.8 to indicate error status
  - Code should be compiled for the location of the internal flash at address $0C000000_H$.
- Receiver Program: XMC4400_CANLoader
  - Must first be downloaded to the slave device in active CAN BSL mode. After a successful download, this code will be executed afterwards. The startup file has been modified; see “startup_XMC4400.s”.
  - It contains flash driver and commands for a ‘handshake’ between the host and slave devices.
  - Defines a port pin P5.2 to signal the process status on the slave board.
  - Code should be compiled for the location of the internal PSRAM, at address $01FFFC000_H$;, see project link file “DAVE3_1_10_XMC4400_CANLoader_PSRAM.id”.
- Application Software: XMC4400_AppSW
  - This is the upgrade software.
  - In our example we create a simple LED (P1.8) toggle program.
  - Code should be compiled for the location of the internal flash, at address $0C040000_H$. see project link file “DAVE3_1_10_XMC4400_AppSW_16K.id”

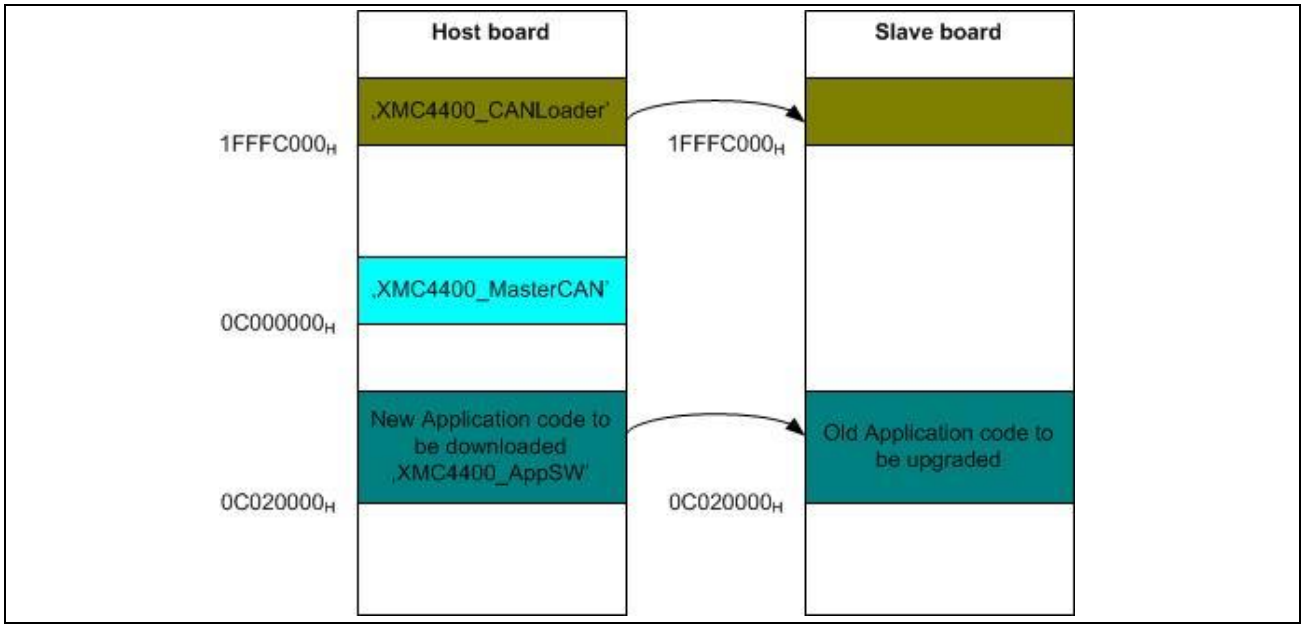

Figure 2 Memory Map

2.1 Host Board

One method of connecting a PC to a microcontroller is to use the PC COM port to download a generated hex file into the microcontroller. In this case the microcontroller must be in active UART BSL mode.

*Note: Infineon provides the freeware **Memtool** to support on-chip flash programming.*

Another method is to use the JTAG interface. In this application example, the host board is used as a 'USB-to-CAN bridge' to download and program the XMC4400 internal flash from the PC.

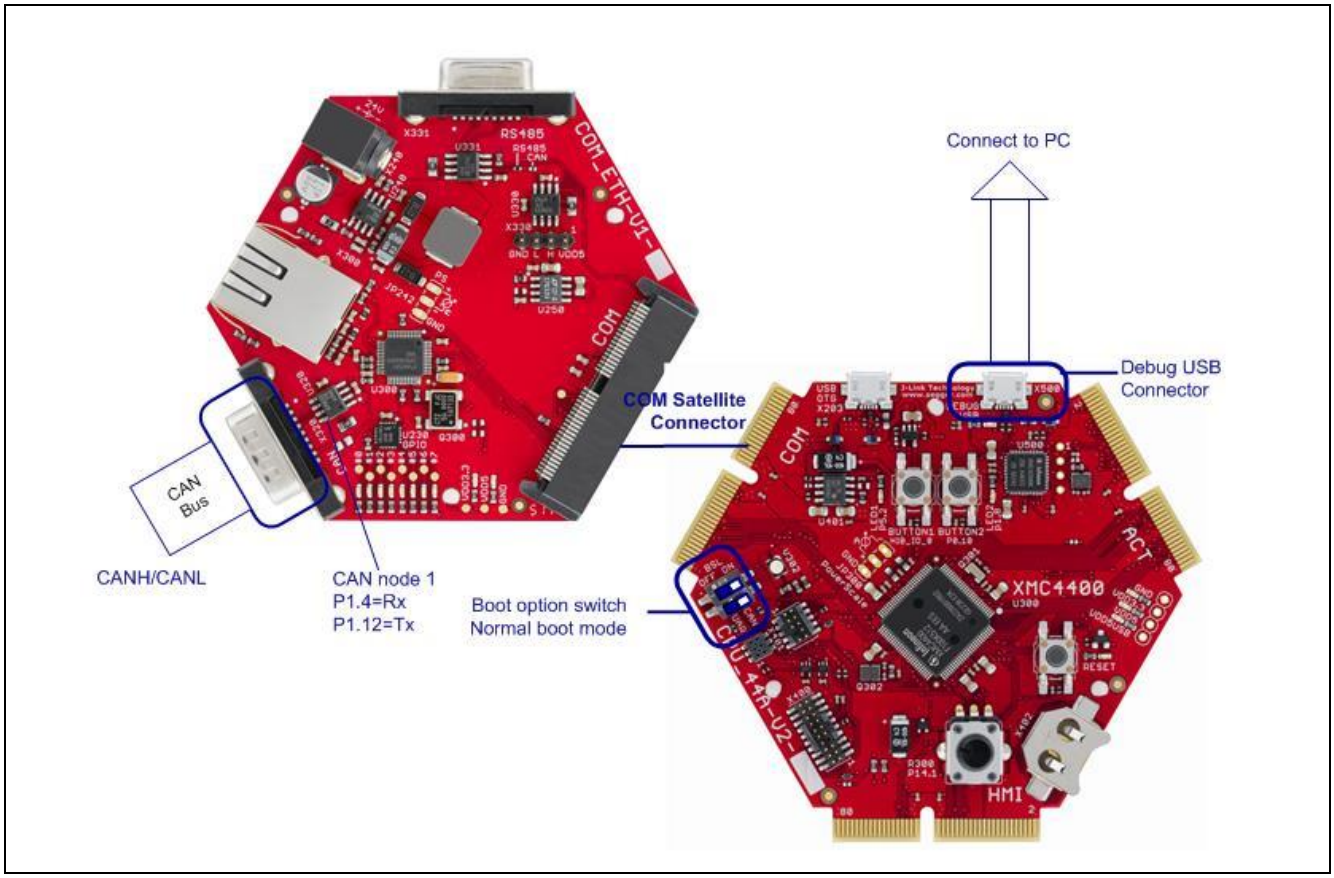

Figure 3 Host Board (CPU_V44A_V2 + COM_ETH_V1)

The debug USB connector is on the On Host CPU kit is. Inside there is an XMC4200 chip with USB plug. This interface provides board power supply, serial wire debug and full duplex UART communication via a USB virtual COM. the serial channel is connected to pins P1.7 and P1.5 of the XMC4400 USIC module.

To start the serial interface either Windows HyperTerminal software, or freeware like MTTTY, can be used.

DAVE™3 integrates code generation, flash programming and debugging.

On **Host COM_ETH kit** the CAN DB-9 connector signals CAN_TXD/CAN_RXD lead to CPU kit CAN1_Tx=P1.12/CAN1_Rx=P1.4 via COM satellite connector Pin28/Pin30. In SW this CAN node is initialized to communicate with slave board. Signal connection of Infineon XMC4000 application kit see please in **section 7.2**.

To start this application, SW pin CANL and pin CANH on **host board** and **slave board** (see **Figure 5**) must be connected together.

2.1.1 Transmitter Program (Project File: XMC4400_MasterCAN)

The Transmitter program is the main program for the **host** board. It controls the interface between the development environment (the PC) and the CAN bus.
The **Host board** has a 12MHz quartz crystal for the input oscillator circuit. At startup, through a function call, the system frequency is changed to 120MHz via the internal PLL configuration and Pin 1.15 is defined for system clock output and can be enabled via macro SCU_CLOCKOUT_SETUP.
The Transmitter program uses the communication peripherals of the microcontroller:

- The USIC0 channel 0 (Tx=1.7, Rx=P1.5) is configured as UART (115200Baud) to communicate serially to the HyperTerminal on the PC.
- The MultiCAN module CAN node 1 (Tx=P1.12, Rx=P1.4) is configured with 500Kbaud to communicate to the slave.
- Message Object 0 is defined as Receive object.
- Message Object 1 is defined as Transmit object.

In the CAN initialization routine the Initialization Frame is configured as per the description in **Table 1**. No interrupt is defined in the CAN or USIC module (polling system).

After initialization, master board sends CAN initialization frame to slave to try to establish the CAN communication between master and slave board.

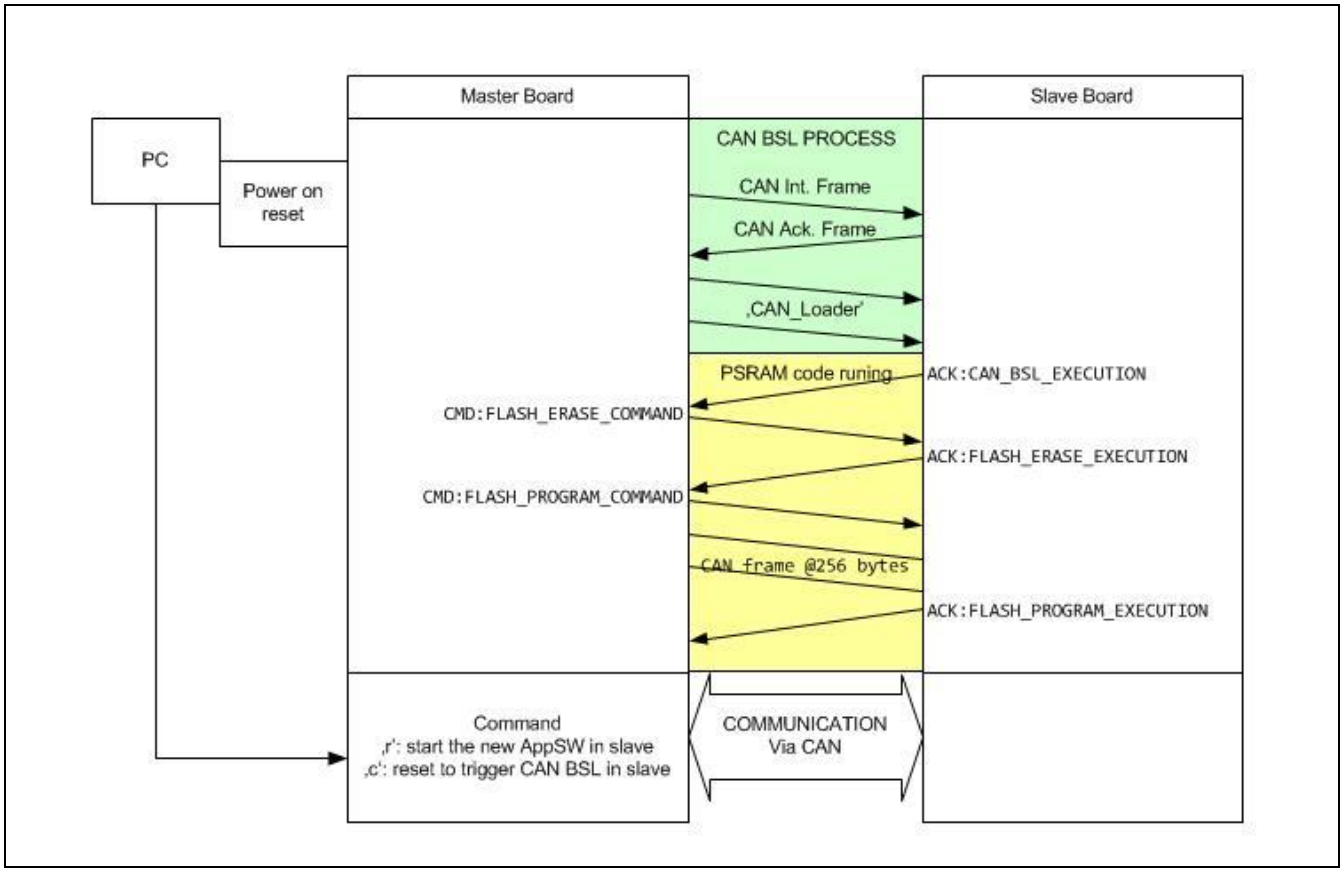

Figure 4 Data Communication on the CAN Bus

After a successful CAN data exchanging during CAN BSL process in the slave board, which can be seen with reporting on the terminal program communicated via the PC COM port, several commands can be selected to control the handshake between the host device and the slave (see **Section 4** for details).

2.1.2 Receiver Program (Project File: XMC4400_CANLoader)

The Receiver program is a piece of code for the slave device. Its role is to program the application program (received via the CAN bus) into the internal flash. The Receiver program must be downloaded into PSRAM via CAN BSL mode.

The built-in CAN BSL only provides a mechanism to load code via the CAN bus into the PSRAM. To program the internal flash of XMC4400 devices, the flash driver (software code of flash command sequences, such as 'erase' and 'program' for example), must be implemented and included in the **XMC4400_CANLoader** project file and be downloaded to the device.

The Receiver program is composed of two main parts:

- Flash Driver
  - The flash programming routines, such as sector erase and page programming, used to program the flash memory.
- Communication Sequence
  - Between the host and the slave, including signal handling (the commands of the CAN message object) for the handshake.

Pin5.2 is defined as GPIO output to signal error state in the slave board.

16Kbytes are currently defined for the Receiver program 'XMC4400_CANLoader.hex' in this application note ($1FFFC000_H$ – $1FFFFFFF_H$).

Blinking LED (Pin5.2) on the slave board indicates that the CAN BSL mode has been terminated and that the code downloaded to PSRAM is executing. The software then checks the receive information of the CAN frame in cycle polling (the MultiCAN module and system clock are not reconfigured and the Tx/Rx message object and the baud rate parameter are not changed), while the software waits for a CAN message command from the host.

In the current version only two commands (see **Table 6**) are implemented:

- A command to start the new downloaded Application SW in slave board
- A command to enter the CAN BSL mode in slave board again via a system reset (see section 1.1).

2.1.3 Application Software (Project File: XMC4400_AppSW)

An application program is required to upgrade the internal flash of the device. This application note includes a simple application program (XMC4400_AppSW).

The example code is intended to be straightforward. Pin1.8 is defined as GPIO output pin and starting the application program lights up its LED.

2.2 Slave Board

Any Infineon Easy Kit equipped with a target device listed at the start of this document, can be used as a **slave** board. Signal connections in different kit versions are listed in **Table 8**. The following figure shows the layout overview of the Slave board using XMC4400 CPU_44A_V2 kit. Some hardware modifications must be made, as indicated in the following figure.

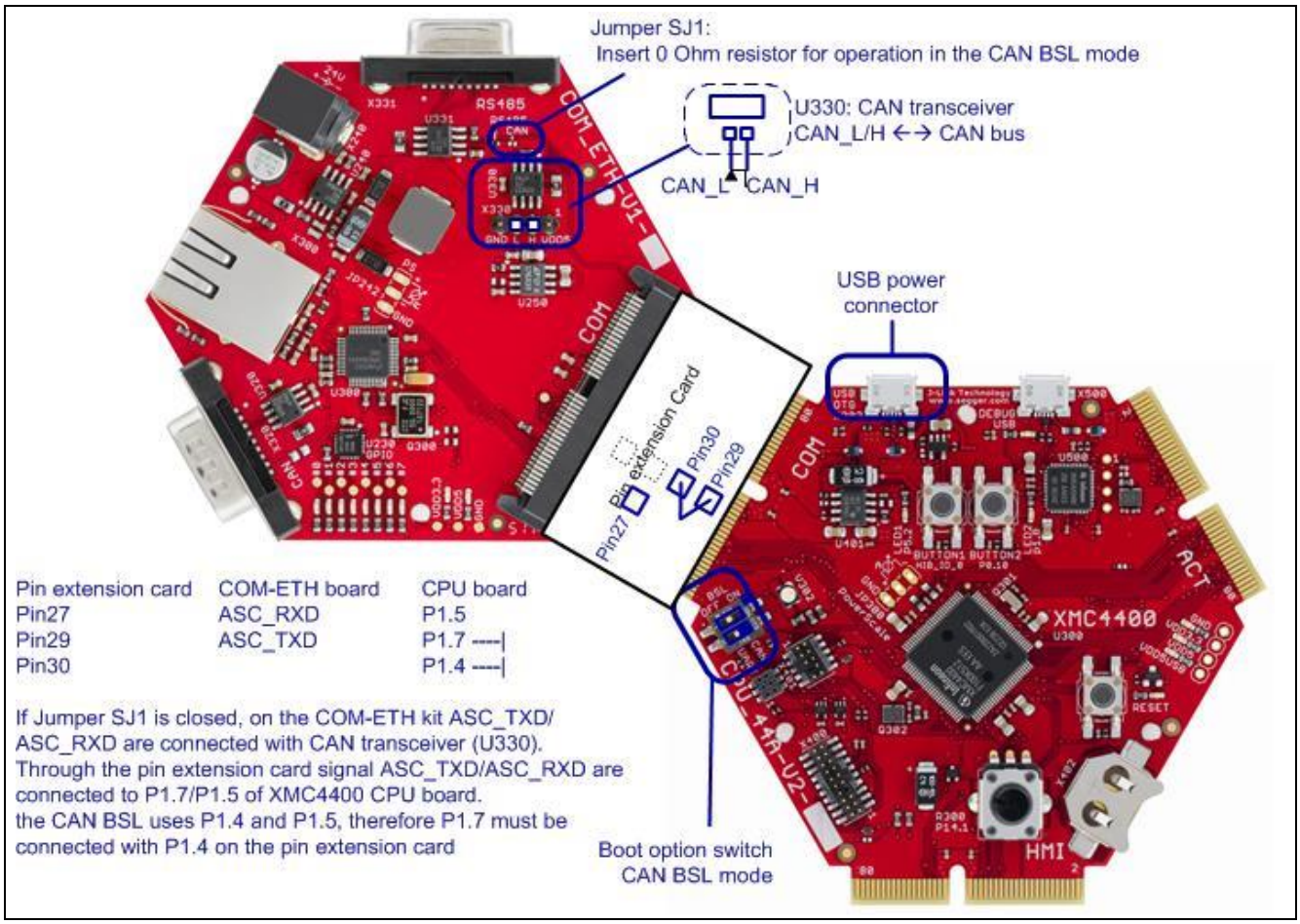

Figure 5 Slave Board (XMC4400 CPU + COM-ETH kit Layout Overview)

The CAN BSL requires a ‘point-to-point’ connection with the host; i.e. the XMC4400 must be the only CAN node connected to the network. If the CANanylzer tool is connected on the CAN bus for testing, the in tool settings menu “no-ACK” must be selected.

A system clock with at least 10 MHz is required for the CAN Bootstrap Loader operation for transfer rates of 1 Mbps.

The XMC4000 family has a 1:1 direct drive clock in the CAN BSL mode and on all XMC4000 CPU kits of the XMC4000 Application Kit series an external 12MHz crystal is integrated.

3 XMC4000 Internal Flash Programming

This chapter is divided into two sections:

- The Internal Flash Memory
- Flash Driver

3.1 The Internal Flash Memory

The XMC4000 devices (see **Table 1**) have up to 1Mbyte of on-chip flash memory. The physical memory is mapped in the non-cached address space at 08000000_H. In XMC4000 the on-chip flash can also be accessed via cacheable address space $0C000000_H$, for benchmarking purposes for example. But flash program and erase operations must be executed to the non-cacheable address space.

The on-chip flash memory in XMC4400 consists of a maximum of 4 physical sectors:

PS0: 64K

PS4: 64K

PS8: 128K

PS9: 256K

Each physical sector is isolated from each other, and they can be separately erased and programmed (in blocks of 256 bytes=page size). A physical sector is the largest erase unit.

The flash memory supports read and write protection, where protection is implemented by logical sector. The on-chip flash memory consists of 10 logical sectors:

S0…S7: 16K

S8: 128K

S9: 256K

The ‘erase’ operation can be used on a logical sector (S0…S9) or a complete physical sector (PS0, PS4, PS8, PS9). The ’program’ operation is applied only per page (256 bytes).

The flash memory can be read and write protected, with protection configured by UCBx programming.

Read protection always globally covers the whole flash memory range (512K). The bit field PROCONx.RPRO indicates the status.

Write protection can be globally enabled when Read protection is applied, or it can be specified for each logical sector (S0…S9). Registers PROCONx (bit S9L…S0L) indicate the status.

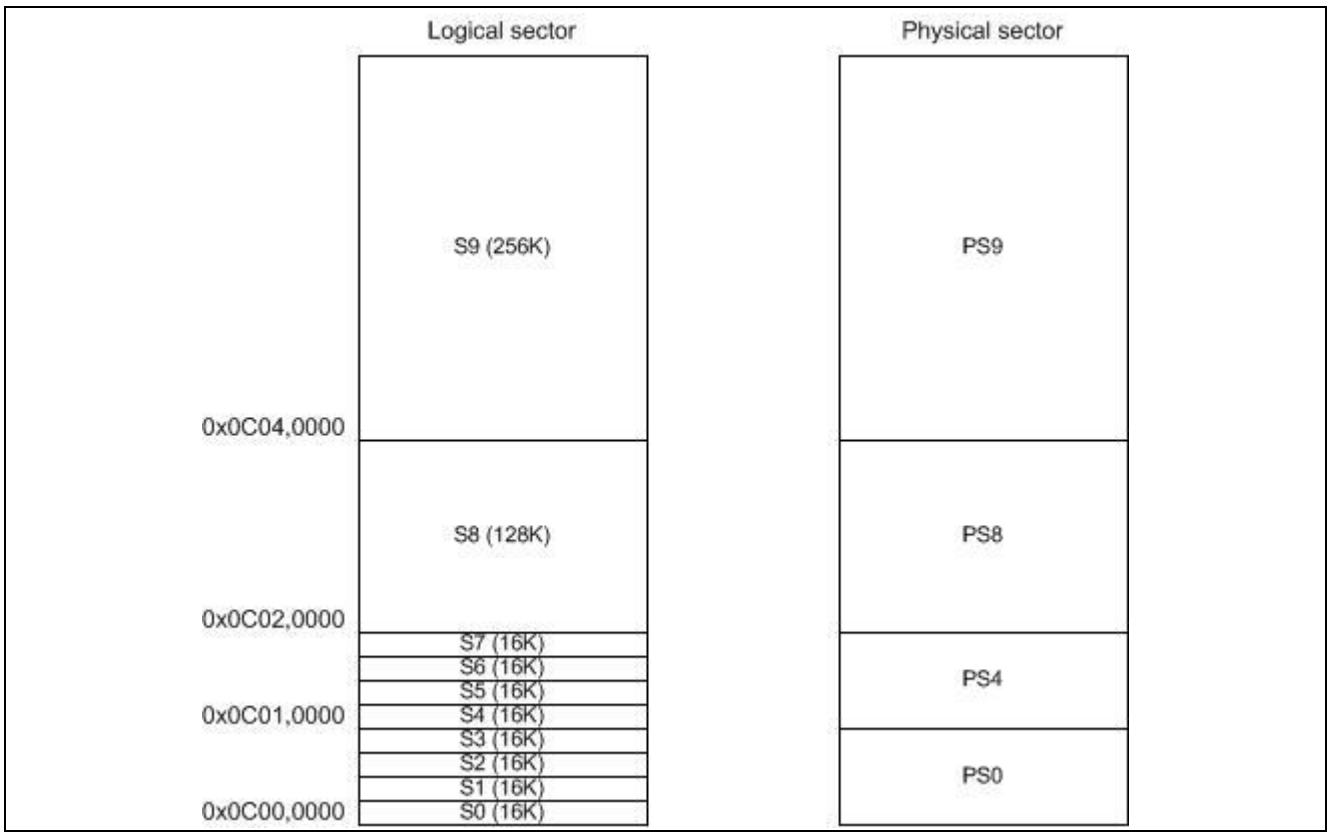

Figure 6 Logical Sectors in XMC4400 (max. 512K Flash)

Note: The XMC4000 devices all have different flash sizes and different allocations of logical sectors. For more details about flash memory size, location, protection and operating modes, please refer to the appropriate User's Manual or Data Sheet.

3.2 Flash Driver

To perform a 'program' operation on the flash memory, it is necessary to write some command sequences. The following table gives the possible operations and their command definitions as used in this document:

Table 2 Command Sequence Definitions (Programming and Erasing)

Command sequence	1. cycle	2. cycle	3. cycle	4. cycle	5. cycle	6. cycle	BUSY
Erase logical Sector	.5554$_H$, AA$_H$	.AAA8$_H$, 55$_H$	.5554$_H$, 80$_H$	.5554$_H$, AA$_H$	.AAA8$_H$, 55$_H$	**SA**$_H$, 30$_H$	yes
Erase physical Sector	.5554$_H$, AA$_H$	.AAA8$_H$, 55$_H$	.5554$_H$, 80$_H$	.5554$_H$, AA$_H$	.AAA8$_H$, 55$_H$	**SA**$_H$, 40$_H$	yes
Enter Page	.5554$_H$, 50$_H$						no
Load Page WORD	.55F0$_H$, **WD**$_H$	.55F0$_H$, **WD**$_H$					no

Program Page	.5554$_H$, AA$_H$	.AAA8$_H$, 55$_H$	.5554$_H$, A0$_H$	**PA**$_H$ AA$_H$			yes
Clear status	.5554$_H$, F5$_H$						yes

Note:

1. ***WD**=32-bit write data*
2. ***PA**=absolute start address of the flash page*
3. ***SA**=absolute start address of the flash logical sector*
4. *All of the command sequences given above, except for Load Page Word and Enter Page, set the FSR.BUSY flag for the duration of their operation. Software must check the BUSY bit after the command instructions.*

The main part of the software XMC4400_CAN_Loader implements flash routines which provide the flash functionality listed in the table. However the user can still add other features as required, such as the ’enable read protection’ operation, or ’disable read protection’ for example.

4 Communication Protocol

After downloading the XMC4400_CAN_Loader code to the XMC4400 device PSRAM, the CAN Bootstrap Loader terminates and the XMC4400 device will start executing the code in PSRAM ($1FFFC000_H$).

The acknowledge CAN_BSL_EXECUTION (see **Figure 4**) from the **slave** board to the **host** board, signals communication is established. Afterwards flash sector defined for the new application software will be erased and programmed with CAN frame data sent by the host board. The FLASH_PROGRAM_EXECUTION acknowledge signals that the user code has successfully finished.

The host board then waits for commands from the PC to execute the new code or start CAN BSL mode again. These commands (process commands) are forwarded using the CAN messages of the host board to slave. These can be modified for any specific requirements.

4.1 Process Commands

4.1.1 Sector Erasing Operation

The Sector Erasing Operation process command.

Table 3 Process Command: Sector Erasing Operation

byte 7	byte 6	byte 5	byte 4	byte 3	byte 2	byte 1	byte 0
							CMD (0xC5)

- ASCII for the terminal program: not implemented
- Physical Sector address: $0C0200000_H$
- Command code: CMD=0xC5

The **slave** board sends byte_0=0x5F and byte_1=0x00(ok) or 0x01(error) to the host.

4.1.2 Flash Page Programming Operation

The Flash Page Programming Operation process command.

Table 4 Process Command: Flash Page Programming Operation

byte 7	byte 6	byte 5	byte 4	byte 3	byte 2	byte 1	byte 0
				Page Address			**CMD (0xC3)**

- ASCII for the terminal program: not implemented
- Page address (24 bits): $0C020000_H$
- Command code: CMD= 0xC3

After receipt of this process command, the XMC4400 **slave** board initializes the flash address and enters a loop state, waiting to receive 256 bytes of data from the host (8 CAN data frames within 8 bytes). This page will then be programmed.

Table 5 Data Frame Format

byte 7	byte 6	byte 5	byte 4	byte 3	byte 2	byte 1	byte 0
Data	Data	Data	Data	Data	Data	Data	Data

The XMC4400 **slave** board sends byte_0=page number and byte_1=0x00(ok) or 0x01(error) to indicate the status for each page programming, followed by an acknowledge 0x3F.

4.1.3 Running the Updated SW

Command to start running the new user code.

Table 6 Process Command: Application Reset for the Normal boot mode Start

byte 7	byte 6	byte 5	byte 4	byte 3	byte 2	byte 1	byte 0
							CMD (0xC1)

- ASCII for the terminal program: ’r’
- Command code: CMD=0xC1

This command allows the user to directly start the downloaded software stored at $0C020000_H$.

4.1.4 Application Reset for the CAN BSL Mode

Command to start the CAN BSL mode again via system reset.

Table 7 Process Command: Application Reset for the CAN BSL Mode

byte 7	byte 6	byte 5	byte 4	byte 3	byte 2	byte 1	byte 0
							CMD (0xC0)

- ASCII for the terminal program: ’c’
- Command code: CMD=0xC0

This command allows the user to start the CAN Bootstrap Loader in **slave** mode again.

After this command, the user can give a power-on reset on the **host** board to repeat the complete process.

5 Implementing the example described

All hardware should be connected as shown in **Figure 1**.

- Host board:
  - Connect CPU kit with COM_ETH kit
  - connect the on board USB connector (for power supply and debug tool) to the PC USB port
- Slave board:
  - Insert 0 Ohm resistor on jumper SJ1 on slave COM_ETH kit
  - Connect CPU kit and COM_ETH kit via the pin extension connector
  - Connect Pin29 with Pin30 on the pin extension connector
  - connect USB power connector (only for power supply) to the PC USB port
- Between host and slave board:
  - CAN cable CANH/CANL

The two boards have to be configured first:

- Set the slave board to active CAN BSL mode:
  - boot option switch in **Figure 5** BSL(TMS)=ON(0), CAN/UART(TCK)=CAN(1)
- Set the host board to active in normal boot mode:
  - boot option switch in **Figure 3** BSL(TMS)=OFF(1), CAN/UART(TCK)=don't' care

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

- Start DAVE tool and download the three pieces of code (two located in the flash and one located in the PSRAM) to the host board.

Note: Because the DAVE (IFX GDB Debugging) places its own flash driver in the PSRAM at address $1FFFC000_H$, the PSRAM code XMC4400_CANLoader must be downloaded in the final stages to avoid being overwritten by the DAVE programming.

- Start the HyperTerminal program (115200Baud) and connect virtual COM port
- make a power-on reset on the slave board
- make a power-on reset on the host board

After power-on reset on the **host** board the transmitter program XMC4400_MasterCAN starts. The XMC4400_CANLoader starts to be loaded to the slave board via CAN bus.

On the **slave** board the received data is stored in PSRAM, starting at address $1FFFC000_H$.

After receipt of the last CAN message object, the CAN BSL terminates and the code loaded to PSRAM is started, followed by sector erasing and programming.

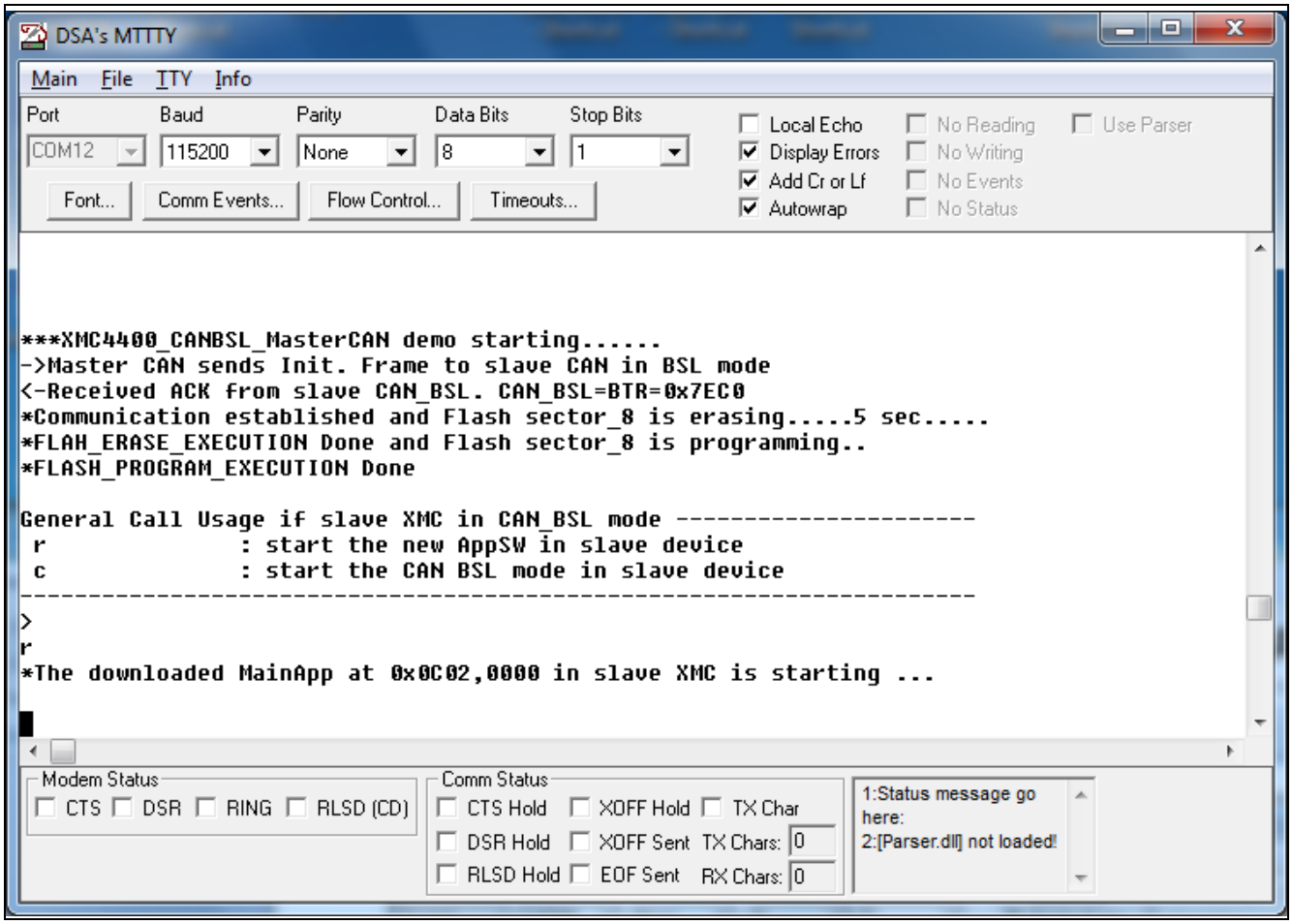

Figure 7 Terminal Report

To execute the XMC440_AppSW code after flash programming on the **slave**, enter the key '**r**'. The LED of Pin 1.8 will blink to indicate that the code in flash is running.

6 Running with two XMC4500 CPU kits

To run the CAN BSL in the XMC4500, please refer to the supplied XMC4500 project files, which have been tested on the XMC4500_AC step.

Note: Among other differences, the XMC4500 has a different PSRAM start address from the XMC4400.

Because pins P1.4/P1.5 (Pin27/Pin29 on the pin connector) of the CPU_45A_V3 kit lead to CANH/CANL on the COM_ETH kit, no cable connection on the pin connector is required, in comparison with XMC4400 tests. Also, it is only required to switch ON Jumper SJ1 on the COM_ETH kit, in this case the CAN node 0 is used in the CAN BSL.

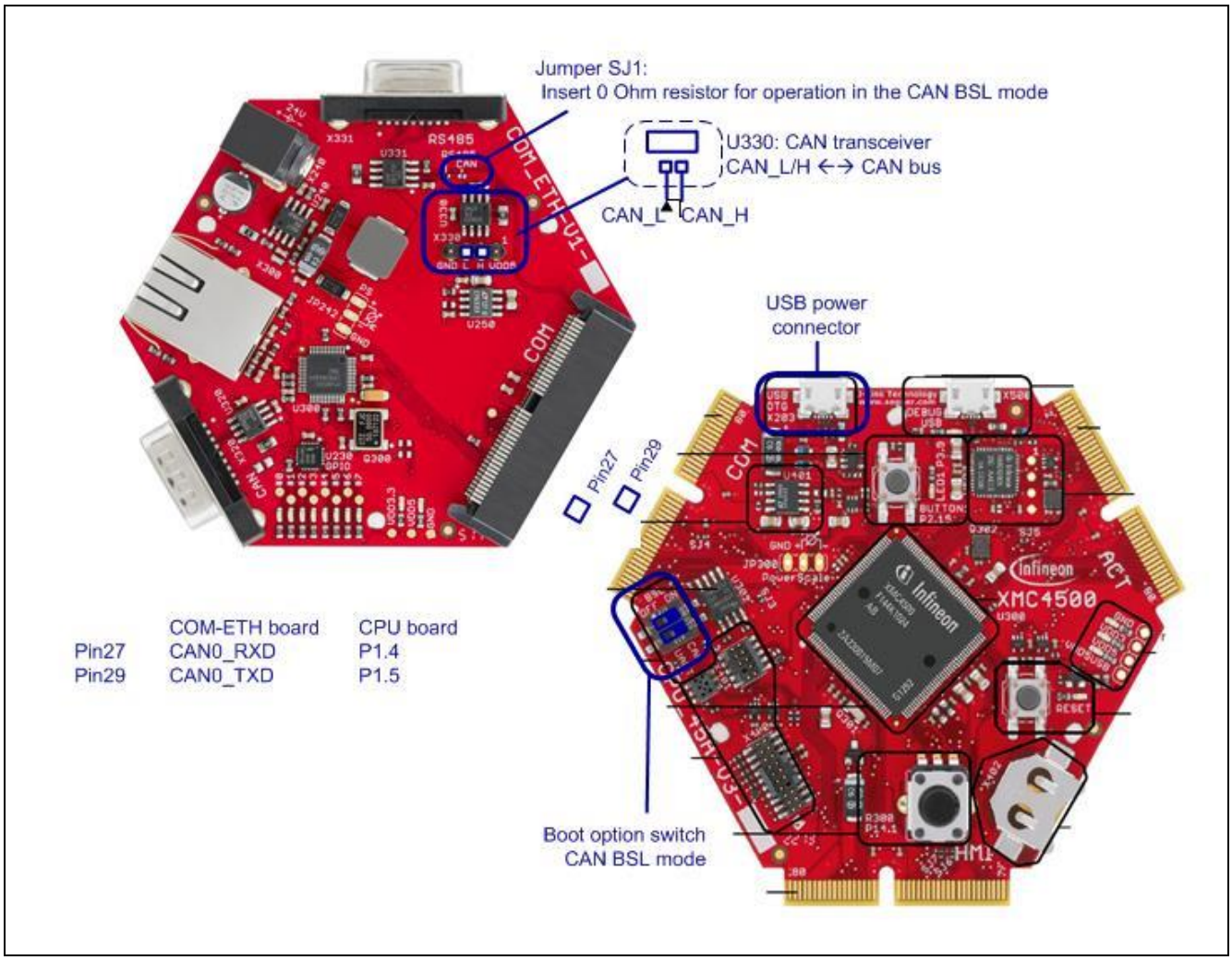

Figure 8 Slave Board (XMC4500 CPU + COM-ETH kit Layout Overview)

7 Appendix

7.1 Hardware Bugs

There are two hardware bugs to be aware of:

- STARTUP_CM.001
- USIC_AI.007

STARTUP_CM.001: CAN Bootstrap Loader

Description:

- The oscillator start-up detection by device firmware does not check for a required stable frequency lock. It is not therefore possible to support an entire spectrum of standard XTAL input frequencies in the CAN BSL boot mode. As a result the device may not answer the initial CAN frame.

Workaround:

- None

Note: This bug is in ;
XMC4500_AA / _AB, and XMC4400 / XMC4200 / XMC4100_AA
but fixed in;
XMC4500_AC, and XMC4400 / XMC4200 / XMC4100_AB.

USIC_AI.007

Description:

- Protocol-related argument and error bits in the RBUFSR register contain incorrect values following a received data word.

Note:

1. *Please see the errata sheet for further details.*
2. *This bug is in XMC4500 AA / AB / AC steps.*
3. *The following derivatives are not affected: XMC4400 / XMC4200 / XMC4100.*
4. *This bug must be considered in project file XM4400_MasterCAN. In the UART protocol, when a data word is received, an alternate receive event (PSR.AIF) may be indicated instead of a receive event (PSR.RIF) even though parity mode is disabled*

7.2 Kit Information

Infineon provides several **XMC4000 Application kits**, interface of UART for terminal program and CAN use different pins (hard-wired) on different board versions.

Table 8 XMC4000 Application Kit Signal Connections

host board kits			
	CAN node (in the normal boot mode)	**Debug USB interface (TTTY)**	**LED**
CPU_45A_V3 (XMC4500_AC)	CAN connector is not available on CPU_45A_V3 CAN pins must be connected via COM pin connector CAN2_TxD=P1.9 => Pin28 CAN2_RxD=P1.8 => Pin30	U0C0 Rx=P1.4 Tx=P1.5	P3.9
CPU_44A_V2 (XMC4400_AB)	CAN connector is not available on CPU_44A_V2 CAN pins must be connected via COM pin connector CAN1_TxD=P1.12 => Pin28 CAN1_RxD=P1.4 =>Pin30	U0C0 Rx=P1.5 Tx=P1.7	P1.8
CPU_42A_V1 (XMC4200_AA)	CAN connector is available on CPU_42A_V1: CAN transceiver with CAN connector (**DE-9**) CAN1_TxD=P1.5 CAN1_RxD=P1.4	U0C0 Tx=P1.5 Rx=P1.4	P2.1
	Note: P1.4 and P1.5 are routed to both CAN and UART interface, on these pins the UART function is overlaid with CAN function. AppNote SW cannot run on this kit		
COM pin connector	Pin connector between CPU kit and COM_ETH kit		
COM_ETH_V1	CAN transceiver with CAN connector (**DE-9**) CAN signal: Pin28=CAN_TXD Pin30=CAN_RXD		
slave board kits			
	CAN node (in the CAN BSL mode) pins: P1.4 and P1.5		**LED**
CPU_45A_V3 (XMC4500_AC)	Leading P1.5/P1.4 to CANL/CANH on **COM_ETH** via **COM pin connector** P1.4=Pin27 → CANH P1.5=Pin29 → CANL CAN_Node1 is used in the CAN BSL		P3.9

CPU_44A_V2 (XMC4400_AB)	Leading P1.5/P1.4 to CANH/CANL on **COM_ETH** via **COM pin connector** P1.4=Pin30 P1.5=Pin27 → CANH P1.7=Pin29 → CANL *Note: connect Pin29/Pin30 on the COM pin connector to lead P1.4 to CANL* CAN_Node0 is used in the CAN BSL	P1.8
CPU_42A_V1	*Note:XMC4200_AA is mounted, this kit is not tested due to HW bug* ***STARTUP_CM.001*** *this kit.*	
COM pin connector	Pin connector between CPU kit and COM_ETH kit	
COM_ETH_V1	**CAN transceiver with CAN connector X330 via Jumper SJ1: ON** CAN Signal: CANH=Pin27='ASC_RXD' CANL=Pin29='ASC_TXD'	

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

8 Revision History

Major changes since the last revision

Page or Reference	Description of change
V1.0, 2014-9	Initial Version

www.infineon.com

Edition 2014-09

Published by

Infineon Technologies AG

81726 Munich, Germany

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

Email: erratum@infineon.com

Document reference

AP32269

Legal Disclaimer

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

Information

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

Warnings

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