Application Note, V 1.1, November 2008

AP32132

TriCore AUDO-F Flash Download Using Bootstrap Loader

Microcontrollers

Never stop thinking

Edition 2008-11

Published by Infineon Technologies AG,
St.-Martin-Strasse 53,
81669 München, Germany

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. 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.

TC1767, TC1797, TC1736

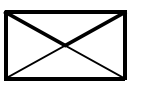
Revision History: V 1.1, 2008-11

Previous Version(s): none

Version	Subjects (major changes since last revision)
V1.0	Initial release for TC1767, TC1797 and TC1736
V1.1	Additional support for flash programming via CAN interface

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

1 Introduction

The TriCore microcontrollers of the AUDO Future (AUDO-F) family **TC1767**, **TC1797** and **TC1736** have a built-in Bootstrap Loading (BSL) mechanism that can be used for flash programming (readers can refer to the BootROM chapter of the User's Manual). However, the TriCore family does not provide any hard coded Bootstrap Loader routines for flash programming (small programs embedded in the BootROM to carry out flash functions, e.g. writing, reading, erasing, verification, etc.). Thus, a flash loader program providing flash programming routines must be implemented by the user.

In TriCore family, Asynchronous Serial Interface (**ASC**) BSL and Controller Area Network (**CAN**) BSL are supported. This example will demonstrate Bootstrap Loading using **both** interfaces.

The target device is connected to a PC via one of the interfaces. The flash loader system demonstrated in this application note consists of two parts:

- The flash loader program is sent to the target device using the built-in Bootstrap Loading mechanism. Once sent and executed, the flash loader program establishes a communication protocol to receive commands from a HOST program (a program running on the PC that controls the flash programming of the target device).
- The HOST program running on a PC uses the communication protocol defined by the flash loader. It sends flash programming commands and the code bytes to be programmed. The HOST program may vary with the specific application it is used for. Thus, the HOST program in this application note is considered to be an example.

The flash loader programs for ASC and CAN BSL are developed for two arbitrary toolchains:

- **Tasking VX-toolset for Tricore v3.0r1**(http://www.tasking.com/tricore).
- **HighTec GNU Toolchain for Tricore v3.4.5.1** (http://www.hightec-rt.com.

The project files for both toolchains provided in this example are completely independent from each other. The user can choose either toolchain.

As an example flash program, the project **LED_Blinking**, which toggles some LEDs controlled by Port 5, is provided for both toolchains as well. The file **LED_Blinking.hex** can be downloaded to flash memory.

Note: Depending on the application, toggling Port 5 of the target device might not always be suitable.

The **TriLoad** HOST program is developed in **Microsoft Visual C++ 6.0**. TriLoad supports both the ASC and CAN interface. TriLoad also supports flash programming for TriCore devices other than the AUDO-F family. Upon program start, the user must specify which device shall be programmed.

Introduction

In general, this example includes the following source code, which will be introduced in detail in later sections.

- In the folder **.\Tasking\Loader2**, ASC BSL Loader 2 (both the source files and the HEX file developed using TASKING VX-TriCore Toolset) is provided.
- In the folder **.\GNU\Loader2**, ASC BSL Loader 2 (both the source files and the HEX file developed using the HighTec GNU TriCore Compiler) is provided.
- In the folder **.\Tasking\Loader3**, ASC BSL Loader 3 (both the source files and the HEX file developed using TASKING VX-TriCore Toolset) is provided.
- In the folder **.\GNU\Loader3**, ASC BSL Loader 3 (both the source files and the HEX file developed using the HighTec GNU TriCore Compiler) is provided.
- In the folder **.\Tasking\CANLoader**, CAN BSL Loader (both the source files and the HEX file developed using TASKING VX-TriCore Toolset) is provided.
- In the folder **.\GNU\CANLoader**, CAN BSL Loader (both the source files and the HEX file developed using the HighTec GNU TriCore Compiler) is provided.
- In the folder **.\Tasking\LED_Blinking** the flash example program (both the source files and the HEX file developed using the TASKING VX-TriCore Toolset) is provided.
- In the folder **.\GNU\LED_Blinking** the flash example program (both the source files and the HEX file developed using the HighTec GNU TriCore Compiler) is provided.
- In the folder **.\Tasking\LED_Blinking_SPRAM** the SPRAM example code (both the source files and the HEX file developed using the TASKING VX-TriCore Toolset) is provided.
- In the folder **.\GNU\LED_Blinking_SPRAM** the SPRAM example code (both the source files and the HEX file developed using the HighTec GNU TriCore Compiler) is provided.
- In the folder **.\TriLoad** an example HOST program that demonstrates the whole process of flash programming. The project files can be compiled with **Microsoft Visual C++ 6.0**.

2 ASC Bootstrap Loading

The communication between PC and the target device is established via the ASC interface. **Figure 2-1** shows a hardware setup for this application, in which the following two pins are used as RxD and TxD, respectively.

- receive pin RxD at pin P3.0 (TC1767, TC1736) or P5.0 (TC1797) respectively
- transmit pin TxD at pin P3.1(TC1767, TC1736) or P5.1 (TC1797) respectively

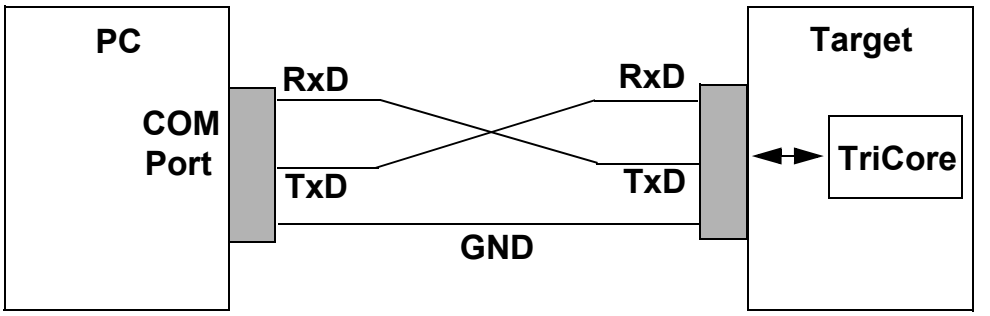


Figure 2-1 The connection between a PC and the target system for TriCore Bootstrap Loading

The flash loader itself is divided into two parts: **Loader 2**[1)] and **Loader 3**. The bootloader procedure is shown in **Figure 2-2**.

1) The built-in Bootstrap Loading mechanism handles the first interaction between PC and target device and can be considered as Loader 1.

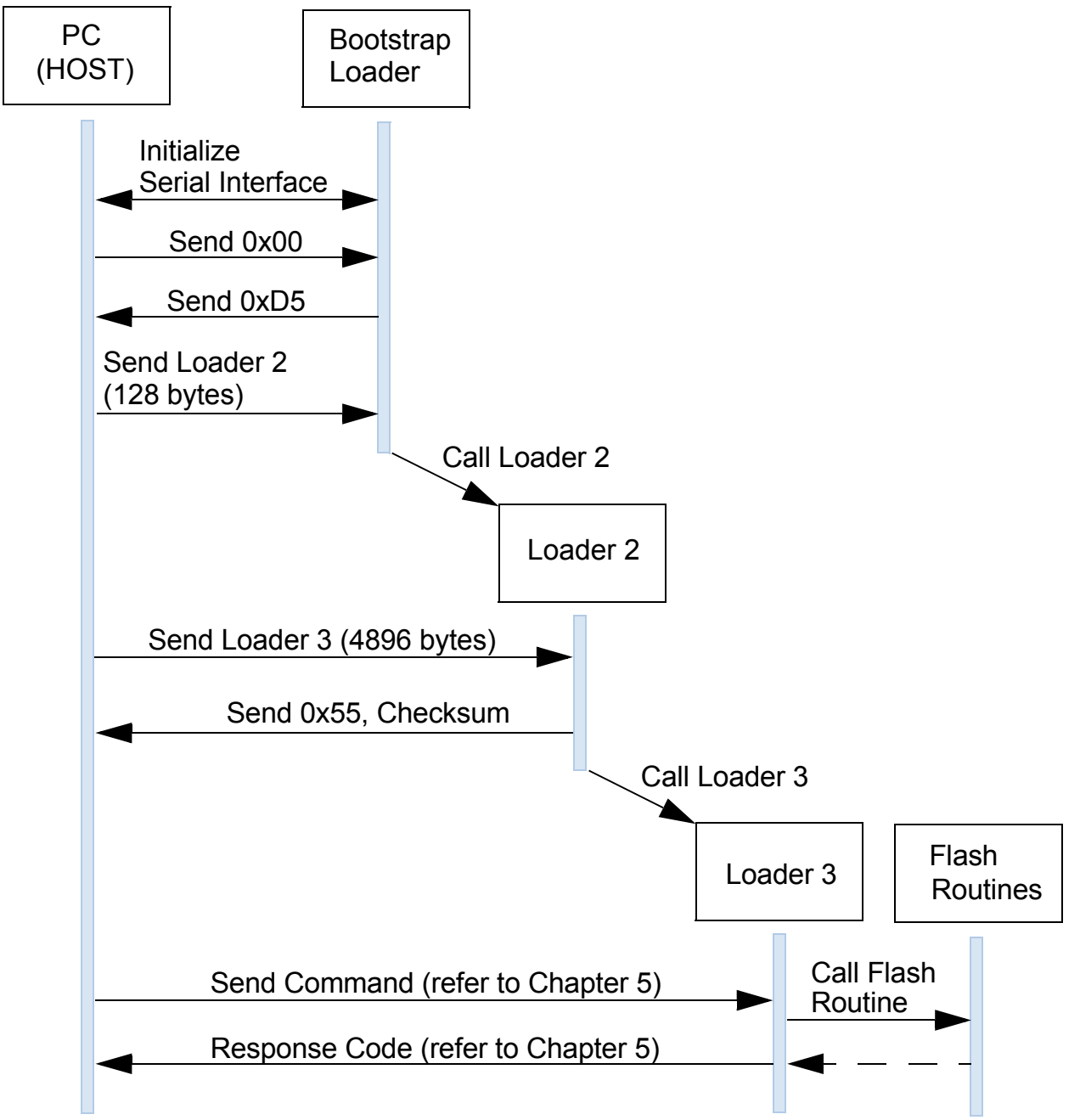


Figure 2-2 The bootloader procedure for flash programming

To run this program, the first step is to make the target device enter BSL mode.

ASC Bootstrap Loading

ASC Bootstrap Loader mode is entered upon a device reset, if the following values are applied at the configuration pins P0[7:0] of Port 0:

P0[7:0] = 10101xx0

The configuration pins are usually connected to a DIP switch on the TriCore board. Assuming that P0.7 is connected to switch pin 1 and the remaining pins accordingly (as for TriBoard, EasyKit and EBeam board), the DIP switch configuration looks as follows:

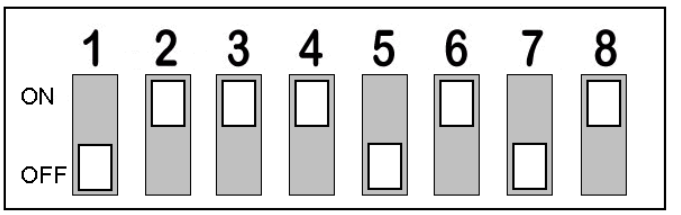


Figure 2-3 DIP switch configuration on the TriCore board for ASC BSL. “On” means high level signal at the pin.

After entering Bootstrap Loader mode, the device switches the clock system from initial **PLL Freerunning Mode** (VCO base frequency) to **Prescaler Mode** with a frequency divider of 1. Hence the system frequency becomes equal to the frequency of an external crystal which must be obligatorily connected between XTAL1/XTAL2 pins, if a Bootstrap Loader mode is selected upon power-on. The crystal frequency **must be at least 10 MHz**.

Further on, the HOST sends **0x00**. Based on this byte, the baud rate used by the PC will be automatically detected by the target device. The TriCore device supports baud rates of up to **115200 bits/s**. The ASC interface will be initialized for **8 data bits** and **1 stop bit**. Once the baud rate is detected and the ASC interface is configured, **0xD5** is sent back to the PC in case of success.

Then the Bootstrap Loader enters a loop and waits to receive exactly 128 bytes from the HOST. These 128 bytes represent the secondary loader (**Loader 2**) and will be stored at the beginning of PMI Scratchpad RAM (SPRAM, base address **0xD4000000**).

Once Loader 2 received, the BootROM jumps to the start address of the secondary loader: Loader 2 is executed. In this application note Loader 2 is stored in the file **loader2.hex**. Its functionality is to receive further bytes from the PC and store them in SPRAM (following its own code section at address **0xD4000080**).

The file **loader3.hex** contains this further received code (**Loader 3**). After Loader 3 is downloaded to SPRAM and executed, it will first establish the communication between PC and the target device and then carry out flash operations.

2.1 Loader 2

Before the Loader 2 can receive further bytes from the PC, a basic device initialization needs to be done. Due to the size constraint of 128 bytes, this startup code must be as small as possible.

Once the device jumps to address **0xD4000000**, its configuration status is as follows:

- The ENDINIT bit is cleared[1]. System control registers that are protected by the ENDINIT feature can be modified.
- The watchdog timer is enabled, which means that the device will be reset if the watchdog timer is not disabled within a certain period of time.
- The serial interface ASC0 is configured. The baud rate is the same as calculated by the BootROM code.
- The clock system has been reset from Precaler Mode to PLL Freerunning Mode. Thus, the device runs with a different clock frequency than the frequency used for baud rate calculation, which means that the actual baud rate does not match anymore the baud rate used by the PC COM interface.
- Stack pointers and Context Save Areas (CSA) are not initialized.
- Interrupt and trap vectors are not defined.

Based on the above conditions, Loader 2 does the following initialization:

- The clock system is reset to Prescaler Mode since the baud rate calculation of the ASC0 interface was based on the clock frequency in Prescaler Mode. The frequency divider that decreases the system frequency is disabled by setting it to 0. The according clock system registers can be modified since the ENDINIT bit is cleared.
- The watchdog timer is disabled. The watchdog timer register can be modified as well since the ENDINIT bit is cleared.
- The ENDINIT bit is set.

Subsequently the code enters a loop waiting to receive exactly 4896[2] (0x1320, size of Loader 3) bytes which are stored in SPRAM starting from address **0xD4000080**. Each byte written to memory is read back and the XOR sum with the previous bytes is calculated.

After reception of the 4896 bytes, the Loader 2 sends 0x55 and XOR checksum to the PC. Finally a jump to address 0xD4000080 is performed in order to execute Loader 3.

The entire code is contained in the files **Loader2.c** (Tasking) and **Loader2.s** (GNU).

1) Some system control registers are protected by the ENDINIT feature. These registers can only be modified, if the ENDINIT bit is cleared. Please refer to the ENDINIT function description in the User's Manual.

2) The actual code size of Loader 3 is less than 4896 bytes. Please refer to **Chapter 2.2** for further details.

2.1.1 Tasking Project Settings

Since the code size of Loader 2 is limited to 128 bytes, the startup code automatically created by Tasking must be replaced by the user startup code (**function __initdevice**).

The code does neither define any stack, nor initializes the stack pointer. Hence, the usage of function calls is not possible. Therefore functions are defined as **inline**.

Beside the default configuration the Tasking project settings need to be configured as follows:

- C/C++ Build -> Processor -> AUDO Future Family -> Check **TC1767**[1)]
- C/C++ Build -> Settings ->
- C/C++ Compiler -> Allocation -> Threshold for putting data in __near: **0**
- C/C++ Compiler -> Optimization -> Optimization level: **0 - None**
- Linker -> Output Format: Check **Generate Intel Hex format file**, Size of addresses: **4**
- Linker -> Libraries: Uncheck **Link default libraries**
- Linker -> Miscellaneous: Uncheck **Include debugger synchronization utility**

The Linker Script Language file **Loader2.lsl** defines 128 bytes in SPRAM memory of type **rom** starting from address **0xD4000000**. This meets the size constraint of 128 bytes required by the target device and the user will be informed of an exceedance already during compilation.

The reset start address is set to **0xD4000000**.

1) In the case that another AUDO-F device is used, the same setting TC1767 applies.

2.1.2 GNU Project Settings

The HighTec GNU settings for the Loader 2 project define one build target **RAM**. The output file **Loader2.elf** is created in the subdirectory **RAM**. If build target **RAM** does not exist, the user must create it to comply with the following project settings.

Beside the default configuration the build options for this build target must be configured as follows:

- RAM -> Compiler settings: Check **Do not link against the default crt0.s**
- RAM -> Compiler settings: Check **Do not link against standard system startup files**
- RAM -> Compiler settings -> Check **Optimize generated code (for size)**
- RAM -> Compiler settings -> Check **Tricore 1767**[1)]
- RAM -> Linker settings -> Other linker options, add line:**-Wl,Loader2.ld -nocrt0 -nostartfiles**
- RAM -> Linker settings -> Other linker options, add line: **-mcpu=tc1767**[1)]
- RAM -> Linker settings -> Other linker options, add line: **-T Loader2.ld**
- RAM -> Linker settings -> Other linker options, add line: **-Wl,-Map,mapfile.lst**
- RAM -> Pre/post build steps -> Post-build steps: **tricore-objcopy -O ihex RAM/Loader2.elf RAM/Loader2.hex**
- RAM -> Pre/post build steps -> Post-build steps: **tricore-objdump -t RAM/Loader2.elf**

The final output file **Loader2.hex** is created in the subdirectory .**RAM**.

The linker description file **Loader2.ld** in the project's root directory defines the entire available memory of the TC1767[1)] device.

The only memory used is the SPRAM code memory **0xD4000000 - 0xD4000080**. The startup section is located at address **0xD4000000**.

1) In the case that another AUDO-F device is used, the same setting applies.

2.2 Loader 3

Loader 3 implements the flash routines and establishes the communication between PC and the target device. Since Loader 2 provides only a simple initialization of the device, the following further initialization steps are done at startup of Loader 3:

- Set the stack pointers for user and interrupt stack,
- initialize the call depth counter,
- initialize the CSA list.

These steps permit the usage of regular function calls. It is implemented in the file **ctr0.s** which is a modified version of the default HighTec startup code. For the Tasking variant of Loader 3, the startup code is contained in the file **cstart.c**.

The main part of Loader 3 (**main.c**) implements flash routines providing the following features:

- Erase flash sectors[1)],
- program flash pages[1)],
- verify a programmed flash page,
- protect PFlash[1)],
- program SPRAM memory,
- execute flash user code starting from address **0xA0000000**,
- execute SPRAM user code starting from address **0xD4001400**.

The flash protection enables a write protection of PFlash. Erase or program attempts result in a protection error, if flash is protected. Upon receiving the protection command, the protection status of the flash is checked. Unprotected flash memory will be protected using two 32bit user-passwords. Protected flash memory will be unprotected using the same passwords. Protection of DFlash is not possible.

Warning: For AUDO-F devices, the flash protection and unprotection can be performed up to 4 times only.

For erasing and programming flash, the sector and page address must be specified respectively. An invalid address (e.g. an address that is not within the flash boundaries) results in an address error. The memory organization for TC1767, TC1797 and TC1736 is described in **Chapter 4**.

Flash user code is executed starting from the PFlash base address **0xA0000000**. Since Loader 2 and Loader 3 occupy the first 0x1400 bytes in SPRAM, programming SPRAM is only possible starting from address **0xD4001400**. Thus, SPRAM user code is executed starting from this address.

Loader 3 defines a communication protocol to receive commands from the PC. Based on the command received, the corresponding flash routine is executed. The communication structure is described in **Chapter 5**.

1) Please refer to **Chapter 4**, Flash Memory Organization

2.2.1 Tasking Project Settings

Beside the default configuration the Tasking project settings for Loader 3 need to be configured as follows:

- C/C++ Build -> Processor -> AUDO Future Family -> Check **TC1767**[1)]
- C/C++ Build -> Settings ->
- C/C++ Compiler -> Allocation -> Threshold for putting data in __near: **0**
- C/C++ Compiler -> Optimization -> Optimization level: **1 - Optimize**
- C/C++ Compiler -> Optimization -> Trade-off between speed and size: **Level4 - Size**
- Linker -> Output Format: Check **Generate Intel Hex format file**, Size of addresses: **4**
- Linker -> Libraries: Uncheck **Link default libraries**
- Linker -> Miscellaneous: Uncheck **Include debugger synchronization utility**

The Linker Script Language file **Loader3.lsl** defines 4896 (0x1320) bytes in SPRAM memory of type **rom** starting from address **0xD4000080** and 68 Kbytes in LDRAM of type **ram** starting from address **0xD0000000**. CSA, stack, heap and global variables are located in LDRAM.

The reset start address is set to **0xD4000080**.

Note: The actual code size of Loader 3 is less than the assumed 0x1320 bytes, which permits changes of the code. If a changed Loader 3 exceeds the size of 0x1320 bytes, Loader 2 must be adapted to this size. A new starting address for SPRAM user code (see Chapter 5.4) must be taken care of.

1) In the case that another AUDO-F device is used, the same setting applies.

2.2.2 GNU Project Settings

The HighTec GNU settings for Loader 3 project define one build target **RAM**. The output file **Loader3.elf** is created in the subdirectory **RAM**. If build target **RAM** does not exist, the user must create it to comply with the following project settings.

Beside the default configuration the build options for this build target must be configured as follows:

- RAM -> Compiler settings: Check **Do not link against the default crt0.s**
- RAM -> Compiler settings: Check **Do not link against standard system startup files**
- RAM -> Compiler settings -> Check **Optimize generated code (for size)**
- RAM -> Compiler settings -> Check **Tricore 1767**[1)]
- RAM -> Linker settings -> Other linker options, add line: **-Wl,Loader3.ld -nocrt0 -nostartfiles**
- RAM -> Linker settings -> Other linker options, add line: **-mcpu=tc1767**[1)]
- RAM -> Linker settings -> Other linker options, add line: **-T Loader3.ld**
- RAM -> Linker settings -> Other linker options, add line: **-Wl,-Map,mapfile.lst**
- RAM -> Pre/post build steps -> Post-build steps: **tricore-objcopy -O ihex RAM/Loader3.elf RAM/Loader3.hex**
- RAM -> Pre/post build steps -> Post-build steps: **tricore-objdump -t RAM/Loader3.elf**

The final output file **Loader3.hex** is created in the subdirectory .**RAM**.

The linker description file **Loader3.ld** in the project's root directory defines the entire available memory of the TC1767[1)] device.

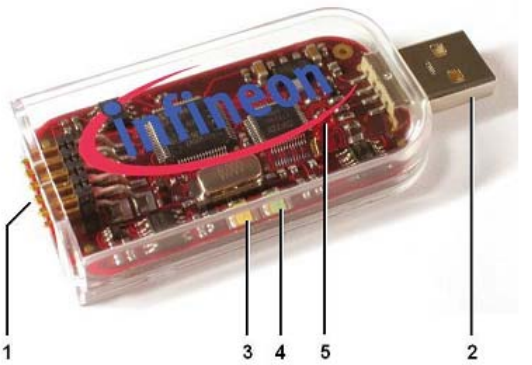
The only memory used is the SPRAM code memory **0xD4000080 - 0xD4001400** and the internal LDRAM with a size of 68 Kbytes starting from address **0xD0000000**. CSA, stack, heap and global variables are located in LDRAM.

Note: The actual code size of Loader 3 is less than the assumed 0x1320 bytes, which permits changes of the code. If a changed Loader 3 exceeds the size of 0x1320 bytes, Loader 2 must be adapted to this size. A new starting address for SPRAM user code (see Chapter 5.4) must be taken care of.

1) In the case that another AUDO-F device is used, the same setting applies.

3 CAN Bootstrap Loading

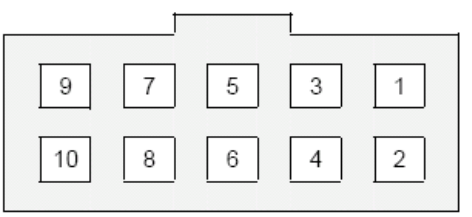
The communication between PC and the target device is established via the CAN interface. Since the regular PC does not have any CAN-Bus interface, a USB-to-CAN bridge is used. The TriLoad HOST program example uses the Infineon XC164CM UCAN start kit for this purpose.



1 Target CAN Connector
2 USB Connector
3 COM LED (yellow)
4 RUN LED (green)
5 USER LED (red)

Figure 3-1 Infineon XC164CM UCAN start kit used as USB-to-CAN bridge

The target device is connected to the start kit via the Target CAN Connector.



1 +5V	6 GND
2 DIO3	7 CAN1H
3 CAN2L	8 CAN1L
4 CAN2H	9 DIO2
5 AIN1	10 DIO1

Figure 3-2 Target CAN Connector of the XC164CM UCAN start kit

The following pins must be connected:

- CAN1L of the start kit to the CAN0L pin of the target device board.
- CAN1H of the start kit to the CAN0H pin of the target device board.

Figure 3-3 shows a hardware setup for this application.

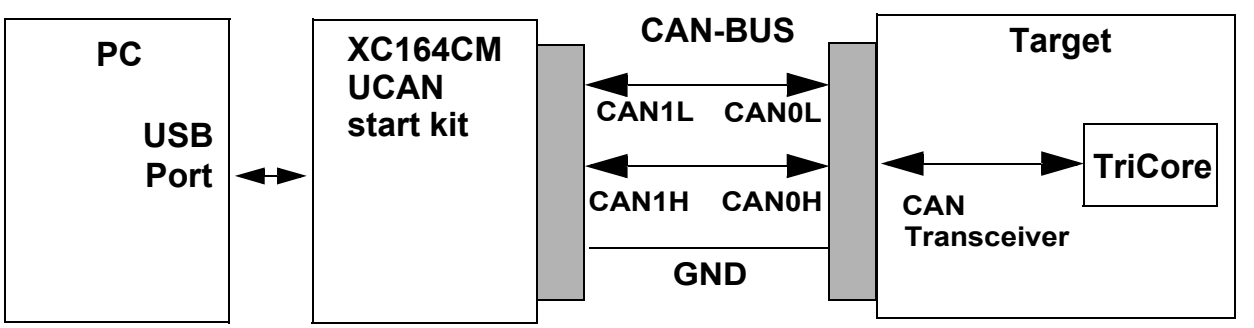


Figure 3-3 Hardware setup for flash programming using TriLoad

The USB-to-CAN transceiver in the XC164CM UCAN start kit is automatically started by TriLoad. The blinking red LED indicates the running application. The start kit should be the only USB device connected to the PC.

Note: The TriLoad example code is developed for the XC164CM UCAN start kit only. If another USB-to-CAN bridge is used, the TriLoad routines to send and receive CAN messages must be adapted.

The flash loader program **CANLoader** is independent from the USB-to-CAN bridge. The bootloader procedure is shown in **Figure 3-4**.

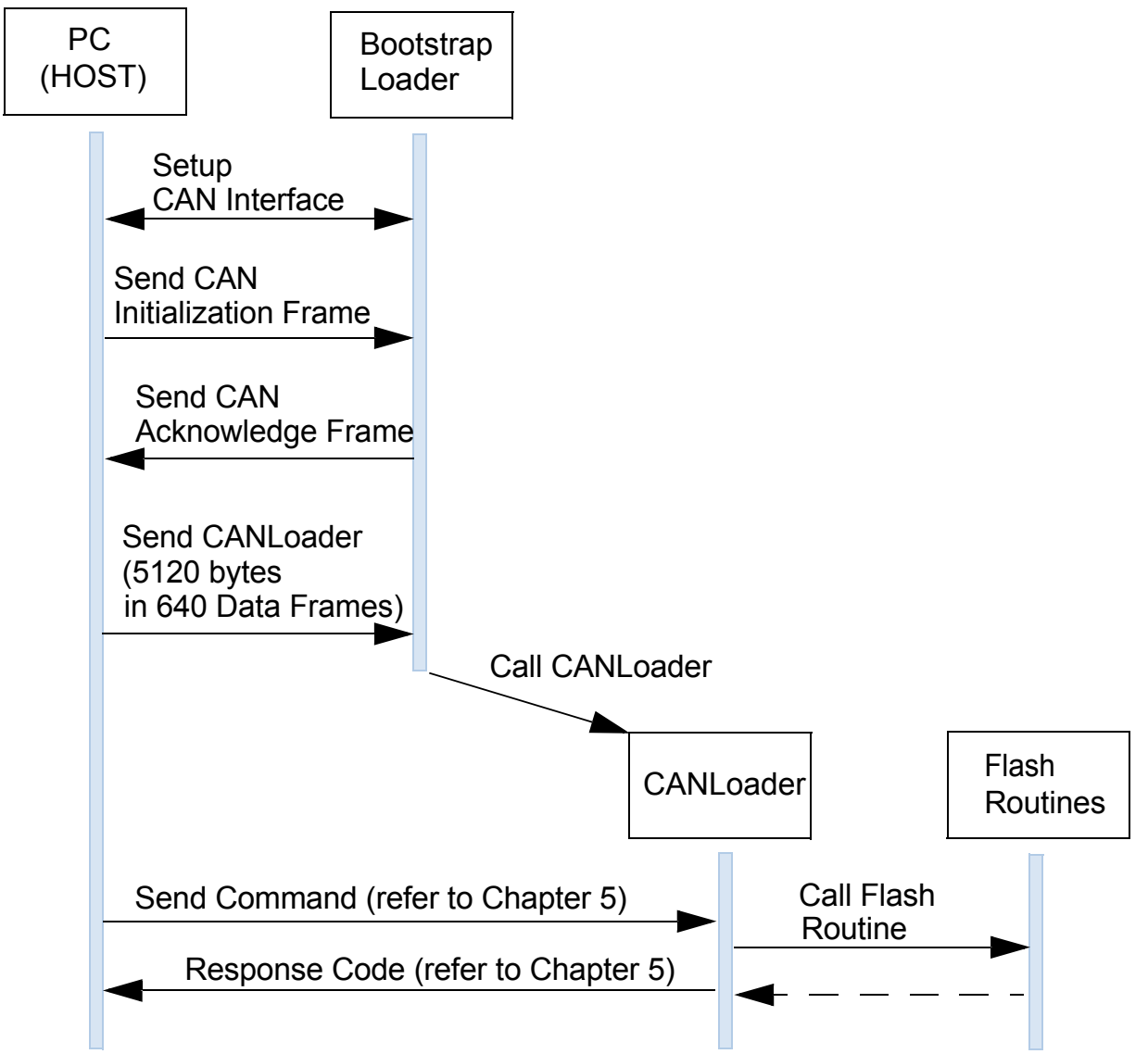


Figure 3-4 The bootloader procedure for flash programming

CAN Bootstrap Loader mode is entered upon a device reset, if the following values are applied at the configuration pins P0[7:0] of Port 0:

P0[7:0] = 010xxxx0

The configuration pins are usually connected to a DIP switch on the TriCore board. Assuming that P0.0 is connected to switch pin 1 and the remaining pins accordingly (as for TriBoard, EasyKit and EBeam board), the DIP switch configuration looks as follows:

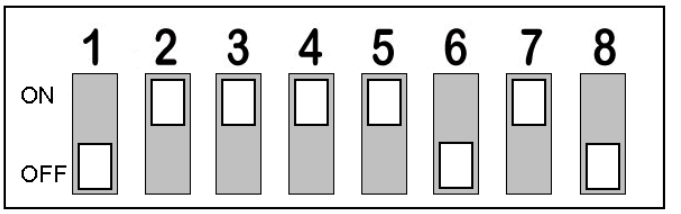


Figure 3-5 DIP switch configuration on the TriCore board for CAN BSL. “On” means high level signal at the pin.

After entering Bootstrap Loader mode, the device switches the clock system from initial **PLL Freerunning Mode** (VCO base frequency) to **Prescaler Mode** with a frequency divider of 1. Hence the system frequency becomes equal to the frequency of an external crystal which must be obligatorily connected between XTAL1/XTAL2 pins, if a Bootstrap Loader mode is selected upon power-on. The crystal frequency **must be at least 10 MHz**.

Further on, the HOST sends the **Initialization CAN Frame** to the device. The CAN-Bus baud rate used by the HOST is automatically detected based on this frame.

Initialization Frame

Parameter	Description
Identifier	11-bit, don’t care
DLC = 8	Data Length Code: 8 bytes within this CAN frame
Data Byte 0	0x55
Data Byte 1	0x55
Data Byte 2	Acknowledge Message Identifier ACKID, low byte
Data Byte 3	Acknowledge Message Identifier ACKID, high byte
Data Byte 4	Data Message Count DMSGC, low byte
Data Byte 5	Data Message Count DMSGC, high byte
Data Byte 6	Data Message Identifier DMSGID, low byte
Data Byte 7	Data Message Identifier DMSGID, high byte

Data Message Count **DMSGC** specifies the number of **CAN Data Frames** sent subsequently to the device.

Data Message Identifier **DMSGID** specifies the identifier that all subsequent CAN Data Frames must carry. The identifier will be internally stored in the device and every

incoming CAN Data Frame will be checked for the same identifier. Only 11 bit out of the 16 bit are stored internally as identifier:

The upper 3 bits of the 16-bit-word will be disregarded and the remaining word will be right-shifted by 2. This yields the 11-bit-identifier.

After reception of a correct initialization frame, the device sends back the **Acknowledge Frame**.

Acknowledge Frame

Parameter	Description
Identifier	Acknowledge Message Identifier ACKID as received by data bytes [3:2] of the initialization frame
DLC = 4	Data Length Code: 4 bytes within this CAN frame
Data Byte 0/1	Contents of bit-timing register
Data Byte 2/3	Copy of acknowledge identifier from initialization frame

After the device has sent the acknowledge frame, it enters a loop waiting to receive exactly the number of CAN Data Frames specified by DMSGC. Each data frame carries 8 bytes of data content.

Data Frame

Parameter	Description
Identifier	Data Message Identifier **DMSGID** as sent by data bytes [7:6] of the initialization frame and transformed as described above
DLC = 8	Data Length Code: 8 bytes within this CAN frame
Data Byte 0..7	Data bytes, assigned to increasing destination addresses in **SPRAM**

DMSGC specifies the number of 640 data frames (5120 bytes). These 5120[1)] (0x1400) bytes represent the **CANLoader** program and will be stored at the beginning of PMI Scratchpad RAM (SPRAM, base address **0xD4000000**).

Once CANLoader received, the BootROM jumps to its start address: CANLoader is executed. In this application note, CANLoader is stored in the file **CANLoader.hex**.

Note: The UCAN USB-to-CAN bridge does not provide any high-speed CAN bus connection. The programming procedure using UCAN is slower compared to ASC BSL.

1) The actual code size of CANLoader is less than 5120 bytes. Please refer to **Chapter 3.1.1** for further details.

3.1 CANLoader

Before the CANLoader can provide flash programming functionality, a further device initialization needs to be done. Once the device jumps to address **0xD4000000**, its configuration status is as follows:

- The ENDINIT bit is cleared[1]. System control registers that are protected by the ENDINIT feature can be modified.
- The watchdog timer is enabled, which means that the device will be reset if the watchdog timer is not disabled within a certain period of time.
- The CAN interface CAN0 is configured. The CAN baud rate is the same as calculated by the BootROM code.
- The clock system has been reset from Precaler Mode to PLL Freerunning Mode. Thus, the device runs with a different clock frequency than the frequency used for baud rate calculation, which means that the actual baud rate does not match anymore the baud rate used by the HOST CAN interface.
- Stack pointers and Context Save Areas (CSA) are not initialized.
- Interrupt and trap vectors are not defined.

Based on the above conditions, CANLoader does the following initialization:

- The clock system is reset to Prescaler Mode since the baud rate calculation of the CAN0 interface was based on the clock frequency in Prescaler Mode. The frequency divider that decreases the system frequency is disabled by setting it to 0.
- The watchdog timer is disabled.
- The ENDINIT bit is set.
- Stack pointers for user and interrupt stack are set.
- The call depth counter is initialized.
- The CSA list is initialized.

These steps are implemented in the file **ctr0.s** which is a modified version of the default HighTec startup code. For the Tasking variant of CANLoader, the startup code is basically contained in the file **cstart.c**. The main part of CANLoader (**main.c**) implements flash routines providing the following features:

- Erase flash sectors[2],
- program flash pages[2],
- verify a programmed flash page,
- protect PFlash[2],

1) Some system control registers are protected by the ENDINIT feature. These registers can only be modified, if the ENDINIT bit is cleared. Please refer to the ENDINIT function description in the User's Manual.

2) Please refer to **Chapter 4**, Flash Memory Organization

- program SPRAM memory,
- execute flash user code starting from address **0xA0000000**,
- execute SPRAM user code starting from address **0xD4001400**.

The flash protection enables a write protection of PFlash. Erase or program attempts result in a protection error, if flash is protected. Upon receiving the protection command, the protection status of the flash is checked. Unprotected flash memory will be protected using two 32bit user-passwords. Protected flash memory will be unprotected using the same passwords. Protection of DFlash is not possible.

Warning: For AUDO-F devices, the flash protection and unprotection can be performed up to 4 times only.

For erasing and programming flash, the sector and page address must be specified respectively. An invalid address (e.g. an address that is not within the flash boundaries) results in an address error. The memory organization for TC1767, TC1797 and TC1736 is described in **Chapter 4**.

Flash user code is executed starting from the PFlash base address **0xA0000000**. Since CANLoader occupies the first 0x1400 bytes in SPRAM, programming SPRAM is only possible starting from address **0xD4001400**. Thus, SPRAM user code is executed starting from this address.

CANLoader defines a communication protocol to receive commands from the PC. Based on the command received, the corresponding flash routine is executed. The communication structure is described in **Chapter 5**.

3.1.1 Tasking Project Settings

Beside the default configuration the Tasking project settings for CANLoader need to be configured as follows:

- C/C++ Build -> Processor -> AUDO Future Family -> Check **TC1767**[1)]
- C/C++ Build -> Settings ->
- C/C++ Compiler -> Allocation -> Threshold for putting data in __near: **0**
- C/C++ Compiler -> Optimization -> Optimization level: **2 - Optimize more**
- C/C++ Compiler -> Optimization -> Trade-off between speed and size: **Level4 - Size**
- Linker -> Output Format: Check **Generate Intel Hex format file**, Size of addresses: **4**
- Linker -> Libraries: Uncheck **Link default libraries**
- Linker -> Miscellaneous: Uncheck **Include debugger synchronization utility**

The Linker Script Language file **CANLoader.lsl** defines 5120 (0x1400) bytes in SPRAM memory of type **rom** starting from address **0xD4000000** and 68 Kbytes in LDRAM of type **ram** starting from address **0xD0000000**. CSA, stack, heap and global variables are located in LDRAM.

The reset start address is set to **0xD4000000**.

Note: The actual code size of CANLoader is less than the assumed 0x1400 bytes, which permits changes of the code. If a changed CANLoader exceeds the size of 0x1400 bytes, a new starting address for SPRAM user code (see Chapter 5.4) must be taken care of.

1) In the case that another AUDO-F device is used, the same setting applies.

3.1.2 GNU Project Settings

The HighTec GNU settings for CANLoader project define one build target **RAM**. The output file **CANLoader.elf** is created in the subdirectory **RAM**. If build target **RAM** does not exist, the user must create it to comply with the following project settings.

Beside the default configuration the build options for this build target must be configured as follows:

- RAM -> Compiler settings: Check **Do not link against the default crt0.s**
- RAM -> Compiler settings: Check **Do not link against standard system startup files**
- RAM -> Compiler settings -> Check **Optimize generated code (for size)**
- RAM -> Compiler settings -> Check **Tricore 1767**[1)]
- RAM -> Linker settings -> Other linker options, add line: **-Wl,CANLoader.ld -nocrt0 -nostartfiles**
- RAM -> Linker settings -> Other linker options, add line: **-mcpu=tc1767**[1)]
- RAM -> Linker settings -> Other linker options, add line: **-T CANLoader.ld**
- RAM -> Linker settings -> Other linker options, add line: **-Wl,-Map,mapfile.lst**
- RAM -> Pre/post build steps -> Post-build steps: **tricore-objcopy -O ihex RAM/CANLoader.elf RAM/CANLoader.hex**
- RAM -> Pre/post build steps -> Post-build steps: **tricore-objdump -t RAM/CANLoader.elf**

The final output file **CANLoader.hex** is created in the subdirectory .**RAM**.

The linker description file **CANLoader.ld** in the project's root directory defines the entire available memory of the TC1767[1)] device.

The only memory used is the SPRAM code memory **0xD4000000 - 0xD4001400** and the internal LDRAM with a size of 68 Kbytes starting from address **0xD0000000**. CSA, stack, heap and global variables are located in LDRAM.

Note: The actual code size of CANLoader is less than the assumed 0x1400 bytes, which permits changes of the code. If a changed CANLoader exceeds the size of 0x1400 bytes, a new starting address for SPRAM user code (see Chapter 5.4) must be taken care of.

1) In the case that another AUDO-F device is used, the same setting applies.

4 Flash Memory Organization

The devices of the AUDO-F family have at least one **Program Memory Unit** (**PMU0**). The following memories belong to the Program Memory Unit:

- **PFlash**: Flash memory for code or constant data (called Program Flash)
- **DFlash**: additional flash memory used for emulation of EEPROM data (called Data Flash)

PFlash and DFlash memories are characterized by their sector architecture and by their page structure. Sectors are flash memory partitions of different sizes. The flash modules and sectorization of the AUDO-F devices are shown in the following tables.

Flash erasure is sector-wise. Sectors are subdivided into pages. Flash memory programming is page-wise. A PFlash page contains 256 bytes. A DFlash page contains 128 bytes.

4.1 TC1767

In TC1767, the flash module **PFlash0** includes 2 MB of PFlash memory.

PFlash0 sector	Address range	Size in bytes
0	0xA0000000 - 0xA0003FFF	0x4000
1	0xA0004000 - 0xA0007FFF	0x4000
2	0xA0008000 - 0xA000DFFF	0x4000
3	0xA000C000 - 0xA000FFFF	0x4000
4	0xA0010000 - 0xA0013FFF	0x4000
5	0xA0014000 - 0xA0017FFF	0x4000
6	0xA0018000 - 0xA001DFFF	0x4000
7	0xA001C000 - 0xA001FFFF	0x4000
8	0xA0020000 - 0xA003FFFF	0x20000
9	0xA0040000 - 0xA007FFFF	0x40000
10	0xA0080000 - 0xA00DFFFF	0x40000
11	0xA00C0000 - 0xA00FFFFF	0x40000
12	0xA0100000 - 0xA013FFFF	0x40000
13	0xA0140000 - 0xA017FFFF	0x40000
14	0xA0180000 - 0xA01DFFFF	0x40000
15	0xA01C0000 - 0xA01FFFFF	0x40000

DFlash includes 64 Kbyte of additional data flash memory.

DFlash sector	Address range	Size in bytes
0	0xAFE00000 - 0xAFE07FFF	0x8000
1	0xAFE10000 - 0xAFE17FFF	0x8000

The SPRAM memory is not subdivided into sectors or pages and can be programmed byte-by-byte.

TC1767 includes 24 Kbytes of SPRAM in an address range of 0xD4000000 - 0xD4005FFF.

4.2 TC1797

In TC1797, the flash module **PFlash0** includes 2 MB of PFlash memory.

PFlash0 sector	Address range	Size in bytes
0	0xA0000000 - 0xA0003FFF	0x4000
1	0xA0004000 - 0xA0007FFF	0x4000
2	0xA0008000 - 0xA000DFFF	0x4000
3	0xA000C000 - 0xA000FFFF	0x4000
4	0xA0010000 - 0xA0013FFF	0x4000
5	0xA0014000 - 0xA0017FFF	0x4000
6	0xA0018000 - 0xA001DFFF	0x4000
7	0xA001C000 - 0xA001FFFF	0x4000
8	0xA0020000 - 0xA003FFFF	0x20000
9	0xA0040000 - 0xA007FFFF	0x40000
10	0xA0080000 - 0xA00DFFFF	0x40000
11	0xA00C0000 - 0xA00FFFFF	0x40000
12	0xA0100000 - 0xA013FFFF	0x40000
13	0xA0140000 - 0xA017FFFF	0x40000
14	0xA0180000 - 0xA01DFFFF	0x40000
15	0xA01C0000 - 0xA01FFFFF	0x40000

DFlash includes 64 Kbyte of additional data flash memory.

DFlash sector	Address range	Size in bytes
0	0xAFE00000 - 0xAFE07FFF	0x8000
1	0xAFE10000 - 0xAFE17FFF	0x8000

In addition to flash module PFlash0, TC1797 includes a second flash module **PFlash1**, which belongs to Program Memory Unit **PMU1**.

PFlash1 sector	Address range	Size in bytes
0	0xA0200000 - 0xA0203FFF	0x4000
1	0xA0204000 - 0xA0207FFF	0x4000
2	0xA0208000 - 0xA020DFFF	0x4000
3	0xA020C000 - 0xA020FFFF	0x4000
4	0xA0210000 - 0xA0213FFF	0x4000
5	0xA0214000 - 0xA0217FFF	0x4000
6	0xA0218000 - 0xA021DFFF	0x4000
7	0xA021C000 - 0xA021FFFF	0x4000
8	0xA0220000 - 0xA023FFFF	0x20000
9	0xA0240000 - 0xA027FFFF	0x40000
10	0xA0280000 - 0xA02DFFFF	0x40000
11	0xA02C0000 - 0xA02FFFFF	0x40000
12	0xA0300000 - 0xA033FFFF	0x40000
13	0xA0340000 - 0xA037FFFF	0x40000
14	0xA0380000 - 0xA03DFFFF	0x40000
15	0xA03C0000 - 0xA03FFFFF	0x40000

The SPRAM memory is not subdivided into sectors or pages and can be programmed byte-by-byte.

TC1797 includes 40 Kbytes of SPRAM in an address range of 0xD4000000 - 0xD4009FFF.

4.3 TC1736

In TC1736, the flash module **PFlash0** includes 1MB of PFlash memory.

PFlash0 sector	Address range	Size in bytes
0	0xA0000000 - 0xA0003FFF	0x4000
1	0xA0004000 - 0xA0007FFF	0x4000
2	0xA0008000 - 0xA000DFFF	0x4000
3	0xA000C000 - 0xA000FFFF	0x4000
4	0xA0010000 - 0xA0013FFF	0x4000
5	0xA0014000 - 0xA0017FFF	0x4000
6	0xA0018000 - 0xA001DFFF	0x4000
7	0xA001C000 - 0xA001FFFF	0x4000
8	0xA0020000 - 0xA003FFFF	0x20000
9	0xA0040000 - 0xA007FFFF	0x40000
10	0xA0080000 - 0xA00DFFFF	0x40000
11	0xA00C0000 - 0xA00FFFFF	0x40000

DFlash includes 32 Kbyte of additional data flash memory.

DFlash sector	Address range	Size in bytes
0	0xAFE00000 - 0xAFE03FFF	0x4000
1	0xAFE10000 - 0xAFE13FFF	0x4000

The SPRAM memory is not subdivided into sectors or pages and can be programmed byte-by-byte.

TC1736 includes 8 Kbytes of SPRAM in an address range of 0xD4000000 - 0xD4001FFF.

5 Communication Protocol

The **flash loader programs** Loader 3 or CANLoader establish a communication structure to receive commands from the PC HOST. The HOST sends commands via transfer blocks. Three types of blocks are defined:

1) Header Block

Byte 0	Byte 1	Bytes 2...14	Byte 15
Block Type (0x00)	**Mode**	**Mode-specific content**	**Checksum**

The header block has a length of 16 bytes.

2) Data Block

Byte 0	Byte 1	Bytes 2...257	Bytes 258...262	Byte 263
Block Type (0x01)	**Verifi-cation option**	**256 data bytes**	**Not Used**	**Checksum**

The data block has a length of 264 bytes.

3) EOT Block

Byte 0	Bytes 1...14	Byte 15
Block Type (0x02)	**Not Used**	**Checksum**

The EOT block has a length of 16 bytes.

The action required by the HOST is indicated in the **Mode** byte of the header block. The flash loader program waits to receive a valid header block and performs the corresponding action. The correct reception of a block is judged by its checksum which is calculated as follows:

The XOR sum of all block bytes excluding block type byte and checksum byte itself.

The different modes specify the flash routines that will be executed by the loader. The modes and their corresponding communication protocol are described as follows.

In ASC BSL mode, all block bytes are sent at once via the UART interface. In CAN BSL mode, each block to be sent must be split into 8-byte-parts and sent in a sequence of **CAN Data Frames**. This yields 2 CAN frames for Header and EOT block and 33 CAN frames for a Data block. The CAN Data Frames must carry the same identifier as specified in **DMSGID** (refer to **Chapter 3**).

5.1 Mode 0: Program Flash Page

Header Block

Byte 0	Byte 1	Byte 2...5	Byte 6...14	Byte 15
Block Type (0x00)	**Mode (0x00)**	**Page Address**	**Not Used**	**Checksum**

PageAddress (32bit): Address of the flash page to be programmed. The address must be 256-byte-aligned (128-byte-aligned for DFlash) and in a valid range (see **Chapter 4**). Otherwise an address error will occur. Byte 2 indicates the highest byte while Byte 5 indicates the lowest byte.

After reception of the header block, the device sends either **0x55** as acknowledgement or an error code in case of an invalid block.

The loader enters a loop waiting to receive the subsequent data blocks in the following format. The loop is **terminated by sending an EOT block** to the target device.

Data Block

Byte 0	Byte 1	Bytes 2...257	Bytes 258...262	Byte 263
Block Type (0x01)	**Verifi-cation option**	**256 code bytes**	**Not Used**	**Checksum**

VerificationOption: Set this byte to **0x01** to request a verification of the programmed page bytes. If this byte is **0x00**, no verification is performed.

Code bytes: Page content.

Since a DFlash page contains only 128 bytes, the second 128 bytes are irrelevant and not used in case of DFlash programming.

After each received data block, the device sends either **0x55** to the PC as acknowledgement or an error code.

EOT Block

Byte 0	Bytes 1...14	Byte 15
Block Type (0x02)	**Not Used**	**Checksum**

After each received EOT, block the device sends either **0x55** to the PC as acknowledgement or an error code

5.2 Mode 1: Execute User Program in PFlash

Header Block

Byte 0	Byte 1	Byte 2...14	Byte 15
Block Type (0x00)	**Mode (0x01)**	**Not Used**	**Check sum**

The command causes a jump to the flash base address **0xA0000000**. The device will exit BSL mode after sending **0x55** as acknowledgement.

5.3 Mode 2: Program SPRAM

Header Block

Byte 0	Byte 1	Bytes 2...5	Byte 6...14	Byte 15
Block Type (0x00)	**Mode (0x02)**	**Address**	**Not Used**	**Checksum**

Address (32bit): Starting address of the SPRAM section to be programmed. The address must be 4-byte-aligned and in a valid range (see **Chapter 4**). Otherwise an address error will occur. Byte 2 indicates the highest byte while Byte 5 indicates the lowest byte.

After reception of the header block, the device sends **0x55** as acknowledgement or an error code in case of an invalid block.

The loader enters a loop waiting to receive the subsequent data blocks in the following format. The loop is **terminated by sending an EOT block** to the target device.

Data Block

Byte 0	Byte 1	Bytes 2...257	Bytes 258...262	Byte 263
Block Type (0x01)	**0x00**	**256 code bytes**	**Not Used**	**Checksum**

Code bytes: Data content.

The data content of SPRAM is not verified.

After each received data block the device sends **0x55** to the PC as acknowledgement or the according error code.

EOT Block

Byte 0	Bytes 1...14	Byte 15
Block Type (0x02)	**Not Used**	**Checksum**

After each received EOT block, the device sends either **0x55** to the PC as acknowledgement or the according error code.

5.4 Mode 3: Execute User Program in SPRAM

Header Block

Byte 0	Byte 1	Byte 2...14	Byte 15
Block Type (0x00)	**Mode (0x03)**	**Not Used**	**Check sum**

The command causes a jump to the SPRAM user code base address **0xD4001400**. The device will exit BSL mode after sending **0x55** as acknowledgement.

5.5 Mode 4: Erase Flash Sector

Header Block

Byte 0	Byte 1	Bytes 2...5	Bytes 6...9	Bytes 10...14	Byte 15
Block Type (0x00)	**Mode (0x04)**	**Sector Address**	**Sector Size**	**Not Used**	**Check sum**

SectorAddress (32bit): Address of the flash sector to be erased. The address must be a valid sector address (see **Chapter 4**), an address error will occur otherwise. Byte 2 indicates the highest address byte while Byte 5 indicates the lowest byte.

SectorSize (32bit): Size of the flash sector to be erased. The size must be a valid sector size (see **Chapter 4**). Byte 6 indicates the highest address byte while Byte 9 indicates the lowest byte.

The device sends either **0x55** to the PC as acknowledgement or an error code.

5.6 Mode 6: Protect / Unprotect PFlash

Header Block

Byte 0	Byte 1	Bytes 2...5	Byte 6...9	Byte 10	Byte 11...12	Byte 13-14	Byte 15
Block Type (0x00)	**Mode (0x06)**	**User Password1**	**User Password2**	**Flash Module**	**Protection Config**	**Not Used**	**Check sum**

UserPassword1 (32bit): First user password. Byte 2 indicates the highest byte while Byte 5 indicates the lowest byte.

UserPassword2 (32bit): Second user password. Byte 6 indicates the highest byte while Byte 9 indicates the lowest byte.

FlashModule: PFlash module to be protected:

0	PFlash0
1	PFlash1
X	PFlashX

ProtectionConfig (16bit): Selection of the flash sectors to be protected. The protection configuration word has the following structure:

ProtectionConfig bit scheme

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	S15/ S14	S13/ S12	S11/ S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0

Sn = 0: Sector n will not be protected.

Sn = 1: Sector n will be protected.

Sector 10 - 15 can only be protected in pairs.

Note: Not all AUDO-F devices have 16 PFlash sectors. In the case that sector n does not exist, Bit Sn should be set to 0. Please refer to Chapter 4 for detailed information about the flash sectorization.

After sending an acknowledgement, the device needs to be reset. All erase or program commands sent to a flash-protected device will cause a protection error.

If the PFlash is unprotected, it will be protected after sending this header block. The same block sent with the same passwords to a flash-protected device will unprotect the PFlash. Protection of DFlash is not possible.

Warning: For AUDO-F devices, the flash protection and unprotection can be performed up to 4 times only.

5.7 Response Code to the HOST

The flash loader program will let the HOST know whether a block has been successfully received and whether the requested flash routine has been successfully executed by sending out a response code.

Response Code	Description
0x55	Acknowledgement, no error
0xFF	Invalid block type
0xFE	Invalid mode
0xFD	Checksum error
0xFC	Invalid address
0xFB	Error during flash erasing
0xFA	Error during flash programming
0xF9	Verification error
0xF8	Protection error

6 TriLoad - HOST Program Example

The TriLoad HOST program developed in C++ uses the above communication structure (**Chapter 5**). The file **TriLoad_API.cpp** contains the API for direct communication with Loader 3 or CANLoader.

The API includes the following functions:

API Function	Description
init_uart	Initialize PC COM interface
init_ucan_uart	Initialize PC COM interface and the UCAN XC164CM USB-to-CAN bridge
init_ASC_BSL	Initialize ASC BSL
send_loader2	Send the Loader 2
send_loader3	Send the Loader 3
send_CANinit_frame	Send the CAN Initialization Frame
send_CANdata_frame	Send a CAN Data Frame
send_CANLoader	Send the CANLoader
bl_send_header	Send header block via ASC interface
blCAN_send_header	Send header block via CAN interface
bl_send_data	Send data block via ASC interface
blCAN_send_data	Send data block via CAN interface
bl_send_EOT	Send EOT block via ASC interface
blCAN_send_EOT	Send EOT block via CAN interface
bl_erase_flash	Erase PFlash/ DFlash sectors
bl_download_pflash	Download code to PFlash
bl_download_dflash	Download code to DFlash
bl_download_spram	Download code to SPRAM
make_flash_image	Create a flash image from HEX file
make_flash_hexfile	Generate a dummy HEX file to fill the entire flash. The code is not executable.

The main program (**TriLoad.cpp**) initializes ASC or CAN BSL and sends Loader 2 and Loader 3 or CANLoader respectively to the target device.

The user must specify the HEX file to be downloaded and the target memory for programming.

An example HEX file is provided for each memory type:

- PFlash (led_blinking.hex)
- DFlash (DFlash_data.hex)
- SPRAM (led_blinking_SPRAM.hex)

If PFlash is protected, the user needs to enter two correct passwords to unprotect the flash. Then the user code is downloaded to PFlash and the flash is protected if desired. Finally the user can execute the downloaded code from either PFlash or SPRAM.

The flash erasing procedure is shown in **Figure 6-1**. The procedure is implemented in the function **bl_erase_flash()**.

The PFlash programming procedure is shown in **Figure 6-2**. The procedure is implemented in the function **bl_download_pflash()**.

The procedures for DFlash and SPRAM programming are implemented accordingly.

Note: TriLoad also supports programming of TriCore devices other than the AUDO-F family. Upon program start, the user must specify which device shall be programmed. The HEX files according to supported TriCore families are included in subfolders (e.g. AUDO-F, AUDO-NG) in the project folder.

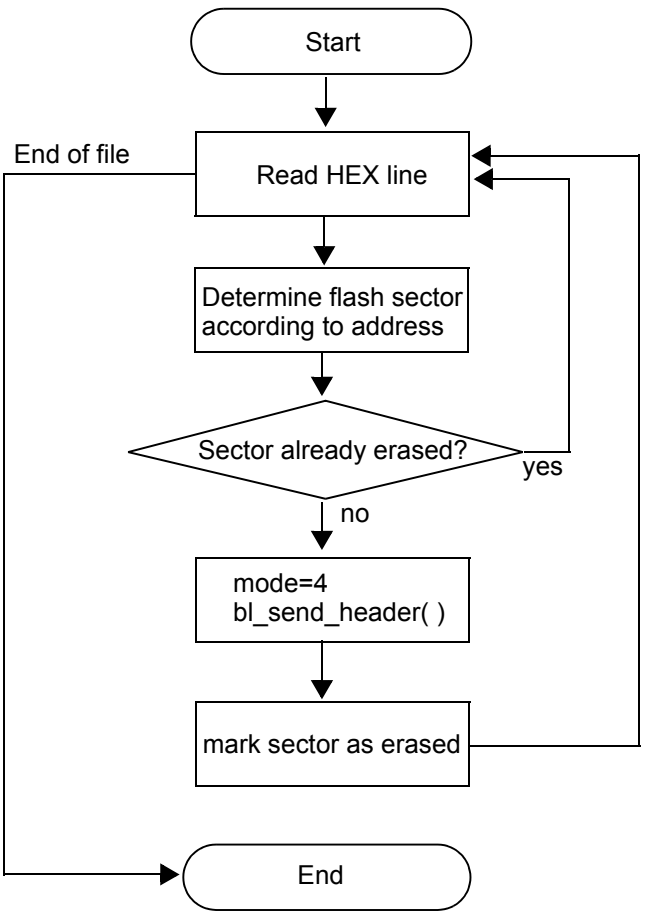


Figure 6-1 Flash erasing procedure implemented in bl_erase_flash()

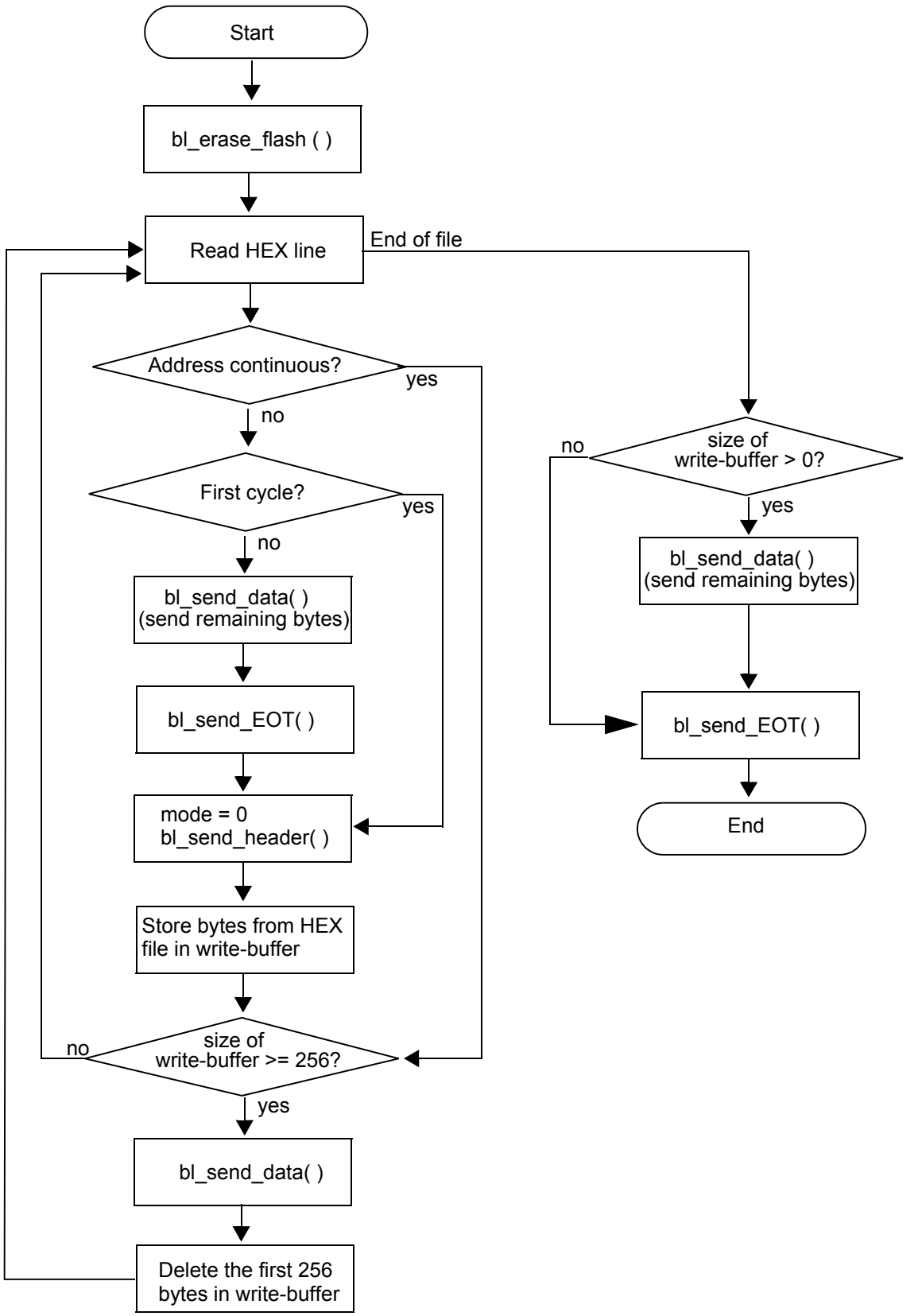


Figure 6-2 Programming procedure implemented in bl_download_pflash()

7 List of Provided Files

The following project files are provided in this application note.

7.1 Tasking VX-toolset for Tricore v3.0r1

Loader 2 (.\Tasking\Loader2):

- Loader2.c
- Loader2.lsl
- tc1767.lsl
- .\Debug\ Loader2.hex
- additional files automatically generated by Tasking

Loader 3 (.\Tasking\Loader3):

- cstart.c
- cstart.h
- main.c
- Loader3.lsl
- tc1767.lsl
- .\Debug\ Loader3.hex
- additional files automatically generated by Tasking

CANLoader (.\Tasking\CANLoader):

- cstart.c
- cstart.h
- main.c
- CANLoader.lsl
- tc1767.lsl
- .\Debug\ CANLoader.hex
- additional files automatically generated by Tasking

LED_Blinking (.\Tasking\LED_Blinking):

- cstart.c
- cstart.h
- main.c
- LED_Blinking.lsl
- tc1767.lsl
- .\Debug\ LED_Blinking.hex
- additional files automatically generated by Tasking

LED_Blinking_SPRAM (.\Tasking\LED_Blinking_SPRAM):

- cstart.c
- cstart.h
- main.c
- LED_Blinking_SPRAM.lsl

- tc1767.lsl
- .\Debug\ LED_Blinking_SPRAM.hex
- additional files automatically generated by Tasking

7.2 HighTec GNU Toolchain for Tricore v3.4.5.1

Loader 2 (.\GNU\Loader2):

- .\src\ Loader2.s
- .\src\ reg176x.h
- Loader2.ld
- .\RAM\ Loader2.hex
- additional files automatically generated by HighTec

Loader 3 (.\GNU\Loader3):

- .\src\ crt0.s
- .\src\ main.c
- .\src\ reg176x.h
- Loader3.ld
- .\RAM\ Loader3.hex
- additional files automatically generated by HighTec

CANLoader (.\GNU\CANLoader):

- .\src\ crt0.s
- .\src\ main.c
- .\src\ reg176x.h
- CANLoader.ld
- .\RAM\ CANLoader.hex
- additional files automatically generated by HighTec

LED_Blinking (.\GNU\LED_Blinking):

- .\src\ main.c
- .\src\ reg176x.h
- LED_Blinking.ld
- .\ROM\ LED_Blinking.hex
- additional files automatically generated by HighTec

LED_Blinking_SPRAM (.\GNU\LED_Blinking_SPRAM):

- .\src\ crt0.s
- .\src\ main.c
- .\src\ reg176x.h
- LED_Blinking_SPRAM.ld
- .\RAM\ LED_Blinking_SPRAM.hex
- additional files automatically generated by HighTec

7.3 Microsoft Visual C++ 6.0

TriLoad v1.2, Example HOST program (.\TriLoad):

The source files are included in a Microsoft Visual C++ 6.0 project.

- TriLoad.cpp
- TriLoad_API.cpp
- TriLoad_API.h
- Device_Memory.h
- .\AUDO-F\ Loader2.hex (needs to be in the C++ project folder)
- .\AUDO-F\ Loader3.hex (needs to be in the C++ project folder)
- .\AUDO-F\ CANLoader.hex (needs to be in the C++ project folder)
- .\AUDO-F\ LED_Blinking.hex (needs to be in the C++ project folder)
- .\AUDO-F\ LED_Blinking_SPRAM.hex (needs to be in the C++ project folder)
- .\AUDO-F\ DFlash_data.hex (needs to be in the C++ project folder)
- FTCJTAG.lib
- FTCJTAG.dll
- FTCJTAG.h
- FTD2XX.dll
- additional files automatically generated by Microsoft Visual C++

8 Reference Documents

Document	Description	Location
TC1767 User's Manual	User's Manual for the TC1767 device	http://www.infineon.com
TC1797 User's Manual	User's Manual for the TC1797 device	http://www.infineon.com
TC1736 User's Manual	User's Manual for the TC1736 device	http://www.infineon.com
ap3208231_tc176x_examples_collection.pdf	Collection of software examples for TC176x devices	http://www.infineon.com
tc_v131_instructionset_v__138.pdf	Instruction set for TriCore V1.3 and V1.3.1 architecture	http://www.infineon.com
Infineon FLASH Samples.pdf	Reference samples for programming Infineon on-chip flash memory devices	http://www.infineon.com
U-CAN-XC164CM-SystemDescription.pdf	System description of the UCAN XC164CM start kit, USB-to-CAN bridge	http://www.infineon.com

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG