Features

- Protocol
  - CAN Used as a Physical Layer
  - 7 ISP CAN Identifiers
  - Relocatable ISP CAN Identifiers
  - Autobaud
- In-System Programming
  - Read/Write Flash and EEPROM Memories
  - Read Device ID
  - Full-chip Erase
  - Read/Write Configuration Bytes
  - Security Setting from ISP Command
  - Remote Application Start Command
- In-Application Programming/Self-Programming
  - Read/Write Flash and EEPROM Memories
  - Read Device ID
  - Block Erase
  - Read/Write Configuration Bytes
  - Bootloader Start

Description

This document describes the CAN bootloader functionalities as well as the CAN protocol to efficiently perform operations on the on-chip Flash (EEPROM) memories. Additional information on the T89C51CC01 product can be found in the T89C51CC01 datasheet and the T89C51CC01 Errata sheet available on the Atmel web site, www.atmel.com.

The bootloader software package (source code and binary) currently used for production is available from the Atmel web site.

<table>
<thead>
<tr>
<th>Bootloader Revision</th>
<th>Purpose of Modifications</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revision 1.0.4</td>
<td>First release</td>
<td>02/12/2001</td>
</tr>
<tr>
<td>Revision 1.2.1</td>
<td>Standardization of tasks in source program</td>
<td>19/03/2007</td>
</tr>
</tbody>
</table>
Functional Description

The T89C51CC01 Bootloader facilitates In-System Programming and In-Application Programming.

In-System Programming Capability

In-System Programming allows the user to program or reprogram a microcontroller on-chip Flash memory without removing it from the system and without the need of a pre-programmed application.

The CAN bootloader can manage a communication with a host through the CAN network. It can also access and perform requested operations on the on-chip Flash memory.

In-Application Programming or Self-Programming Capability

In-Application Programming (IAP) allows the reprogramming of a microcontroller on-chip Flash memory without removing it from the system and while the embedded application is running.

The CAN bootloader contains some Application Programming Interface routines named API routines allowing IAP by using the user’s firmware.

Block Diagram

This section describes the different parts of the bootloader. Figure 1 shows the on-chip bootloader and IAP processes.

**Figure 1.** Bootloader Process Description

![Block Diagram](image-url)
ISP Communication Management

The purpose of this process is to manage the communication and its protocol between the on-chip bootloader and an external device (host). The on-chip bootloader implements a CAN protocol (see Section “Protocol”, page 10). This process translates serial communication frames (CAN) into Flash memory accesses (read, write, erase...).

User Call Management

Several Application Program Interface (API) calls are available to the application program to selectively erase and program Flash pages. All calls are made through a common interface (API calls) included in the bootloader. The purpose of this process is to translate the application request into internal Flash Memory operations.

Flash Memory Management

This process manages low level accesses to the Flash memory (performs read and write accesses).

Bootloader Configuration

Configuration and Manufacturer Information

The following table lists Configuration and Manufacturer byte information used by the bootloader. This information can be accessed through a set of API or ISP commands.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSB</td>
<td>Boot Status Byte</td>
<td>FFh</td>
</tr>
<tr>
<td>SBV</td>
<td>Software Boot Vector</td>
<td>FCh</td>
</tr>
<tr>
<td>SSB</td>
<td>Software Security Byte</td>
<td>FFh</td>
</tr>
<tr>
<td>EB</td>
<td>Extra Byte</td>
<td>FFh</td>
</tr>
<tr>
<td>CANBT1</td>
<td>CAN Bit Timing 1</td>
<td>FFh</td>
</tr>
<tr>
<td>CANBT2</td>
<td>CAN Bit Timing 2</td>
<td>FFh</td>
</tr>
<tr>
<td>CANBT3</td>
<td>CAN Bit Timing 3</td>
<td>FFh</td>
</tr>
<tr>
<td>NNB</td>
<td>Node Number Byte</td>
<td>FFh</td>
</tr>
<tr>
<td>CRIS</td>
<td>CAN Relocatable Identifier Segment</td>
<td>00h</td>
</tr>
<tr>
<td>Manufacturer</td>
<td></td>
<td>58h</td>
</tr>
<tr>
<td>ID1: Family Code</td>
<td></td>
<td>D7h</td>
</tr>
<tr>
<td>ID2: Product Name</td>
<td></td>
<td>BBh</td>
</tr>
<tr>
<td>ID3: Product Revision</td>
<td></td>
<td>FFh</td>
</tr>
</tbody>
</table>
Mapping and Default Value of Hardware Security Byte

The 4 Most Significant Byte (MSB) of the Hardware Byte can be read/written by software (this area is called Fuse bits). The 4 (Least Significant Byte) LSB can only be read by software and written by hardware in parallel mode (with parallel programmer devices).

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Mnemonic</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>X2B</td>
<td>U</td>
<td>To start in x1 mode</td>
</tr>
<tr>
<td>6</td>
<td>BLJB</td>
<td>P</td>
<td>To map the boot area in code area between F800h-FFFFh</td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
<td>U</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
<td>U</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
<td>U</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>LB2</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>LB1</td>
<td>U</td>
<td>To lock the chip (see datasheet)</td>
</tr>
<tr>
<td>0</td>
<td>LB0</td>
<td>U</td>
<td></td>
</tr>
</tbody>
</table>

Note: 1. U: Unprogram = 1  
P: Program = 0

Security

The bootloader has Software Security Byte (SSB) to protect itself from user access or ISP access.

The Software Security Byte (SSB) protects from ISP accesses. The command ‘Program Software Security Bit’ can only write a higher priority level. There are three levels of security:

- **Level 0: NO_SECURITY (FFh)**  
  This is the default level.  
  From level 0, one can write level 1 or level 2.

- **Level 1: WRITE_SECURITY (FEh)**  
  In this level it is impossible to write in the Flash memory, BSB and SBV.  
  The Bootloader returns ID_ERROR message.  
  From level 1, one can write only level 2.

- **Level 2: RD_WR_SECURITY (FCh)**  
  Level 2 forbids all read and write accesses to/from the Flash memory.  
  The Bootloader returns ID_ERROR message.

Only a full chip erase command can reset the software security bits.

<table>
<thead>
<tr>
<th></th>
<th>Level 0</th>
<th>Level 1</th>
<th>Level 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash/EEPROM</td>
<td>Any access allowed</td>
<td>Read only access allowed</td>
<td>All access not allowed</td>
</tr>
<tr>
<td>Fuse bit</td>
<td>Any access allowed</td>
<td>Read only access allowed</td>
<td>All access not allowed</td>
</tr>
<tr>
<td>BSB &amp; SBV &amp; EB</td>
<td>Any access allowed</td>
<td>Read only access allowed</td>
<td>All access not allowed</td>
</tr>
<tr>
<td>SSB</td>
<td>Any access allowed</td>
<td>Write level2 allowed</td>
<td>Read only access allowed</td>
</tr>
<tr>
<td>Manufacturer info</td>
<td>Read only access allowed</td>
<td>Read only access allowed</td>
<td>All access not allowed</td>
</tr>
<tr>
<td>Bootloader info</td>
<td>Read only access allowed</td>
<td>Read only access allowed</td>
<td>All access not allowed</td>
</tr>
<tr>
<td>Erase block</td>
<td>Allowed</td>
<td>Not allowed</td>
<td>Not allowed</td>
</tr>
<tr>
<td>Full chip erase</td>
<td>Allowed</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td>Blank check</td>
<td>Allowed</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
</tbody>
</table>
**Software Boot Vector**

The Software Boot Vector (SBV) forces the execution of a user bootloader starting at address [SBV]00h in the application area (FM0).

The way to start this user bootloader is described in Section “Boot Process”.

**Figure 2. Software Boot Vector**

**FLIP Software Program**

FLIP is a PC software program running under Windows® 9x/2000/XP Windows NT® and LINUX® that supports all Atmel Flash microcontroller and CAN protocol communication media.

Several CAN dongles are supported by FLIP (for Windows).

This software program is available free of charge from the Atmel web site.
In-System Programming

ISP allows the user to program or reprogram a microcontroller’s on-chip Flash memory through the CAN network without removing it from the system and without the need of a pre-programmed application.

This section describes how to start the CAN bootloader and the higher level protocols over the CAN.

Boot Process

The bootloader can be activated in two ways:

• Hardware condition
• Regular boot process

Hardware Condition

The Hardware conditions (EA = 1, PSEN = 0) during the \text{RESET} falling edge force the on-chip bootloader execution. In this way the bootloader can be carried out whatever the user Flash memory content.

As PSEN is an output port in normal operating mode (running user application or bootloader code) after reset, it is recommended to release PSEN after falling edge of reset signal. The hardware conditions are sampled at reset signal falling edge, thus they can be released at any time when reset input is low.
Figure 3. Regular Boot Process

Hardware Boot Process

Software Boot Process

Start Application

Start User Bootloader

Start Bootloader

RESET

bit ENBOOT in AUXR1 Register is initialized with BLJB inverted

ENBOOT = 1
PC = F800h
FCON = 00h

ENBOOT = 1
PC = F800h
FCON = F0h

ENBOOT = 0
PC = 0000h

BLJB = 1

FCON = 00h

SBV < 7Fh

Yes

No

Yes

No

Yes

No

Yes

No
Physical Layer

The CAN is used to transmit information has the following configuration:
- Standard Frame CAN format 2.0A (identifier 11-bit)
- Frame: Data Frame
- Baud rate: autobaud is performed by the bootloader

CAN Controller Initialization

Two ways are possible to initialize the CAN controller:
- Use the software autobaud
- Use the user configuration stored in the CANBT1, CANBT2 and CANBT3

The selection between these two solutions is made with EB:
- EB = FFh: the autobaud is performed.
- EB not equal to FFh: the CANBT1:2:3 are used.

CANBT1:3 and EB can be modified by user through a set of API or with ISP commands.

The figure below describes the CAN controller flow.

Figure 4. CAN Controller Initialization
**CAN Autobaud**

The following table shows the auto baud performance for a point-to-point connection in X1 mode.

<table>
<thead>
<tr>
<th></th>
<th>8 MHz</th>
<th>11.059 MHz</th>
<th>12 MHz</th>
<th>16 MHz</th>
<th>20 MHz</th>
<th>22.1184 MHz</th>
<th>24 MHz</th>
<th>25 MHz</th>
<th>32 MHz</th>
<th>40 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>20K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>125K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>500K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: ‘–’ indicates an impossible configuration.

**CAN Autobaud Limitation**

The CAN Autobaud implemented in the bootloader is efficient only in point-to-point connection. Because in a point-to-point connection, the transmit CAN message is repeated until a hardware acknowledge is done by the receiver.

The bootloader can acknowledge an incoming CAN frame only if a configuration is found. This functionality is not guaranteed on a network with several CAN nodes.
Protocol

Generic CAN Frame
Description

- Identifier: Identifies the frame (or message). Only the standard mode (11-bit) is used.
- Control: Contains the DLC information (number of data in Data field) 4-bit.
- Data: Data field consists of zero to eight bytes. The interpretation within the frame depends on the Identifier field.

The CAN Protocol manages directly using hardware a checksum and an acknowledge.

Note: To describe the ISP CAN Protocol, we use Symbolic name for Identifier, but default values are given.

Command Description
This protocol allows to:
- Initiate the communication
- Program the Flash or EEPROM Data
- Read the Flash or EEPROM Data
- Program Configuration Information
- Read Configuration and Manufacturer Information
- Erase the Flash
- Start the application

Overview of the protocol is detailed in Appendix-A.

Several CAN message identifiers are defined to manage this protocol.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Command Effect</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_SELECT_NODE</td>
<td>Open/Close a communication with a node</td>
<td>[CRIS]0h</td>
</tr>
<tr>
<td>ID_PROG_START</td>
<td>Start a Flash/EEPROM programming</td>
<td>[CRIS]1h</td>
</tr>
<tr>
<td>ID_PROG_DATA</td>
<td>Data for Flash/EEPROM programming</td>
<td>[CRIS]2h</td>
</tr>
<tr>
<td>ID_DISPLAY_DATA</td>
<td>Display data</td>
<td>[CRIS]3h</td>
</tr>
<tr>
<td>ID_WRITE_COMMAND</td>
<td>Write in XAF, or Hardware Byte</td>
<td>[CRIS]4h</td>
</tr>
<tr>
<td>ID_READ_COMMAND</td>
<td>Read from XAF or Hardware Byte and special data</td>
<td>[CRIS]5h</td>
</tr>
<tr>
<td>ID_ERROR</td>
<td>Error message from bootloader only</td>
<td>[CRIS]6h</td>
</tr>
</tbody>
</table>

It is possible to allocate a new value for CAN ISP identifiers by writing the byte CRIS with the base value for the group of identifier.

The maximum value for CRIS is 7Fh and the default CRIS value is 00h.
Communication Initialization

The communication with a device (CAN node) must be opened prior to initiate any ISP communication.

To open communication with the device, the Host sends a “connecting” CAN message (ID_SELECT_NODE) with the node number (NNB) passed in parameter.

If the node number passed is equal to FFh then the CAN bootloader accepts the communication (Figure 6).

Otherwise the node number passed in parameter must be equal to the local Node Number (Figure 7).

Figure 6. First Connection

![Diagram of First Connection](image)

Figure 7. On Network Connection

![Diagram of On Network Connection](image)
Before opening a new communication with another device, the current device communication must be closed with its connecting CAN message (ID_SELECT_NODE).

**Request From Host**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Length</th>
<th>Data[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_SELECT_NODE</td>
<td>1</td>
<td>num_node</td>
</tr>
</tbody>
</table>

Note: num_node is the NNB (Node Number Byte) to which the Host wants to talk to.

**Answers From Bootloader**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Length</th>
<th>Data[0]</th>
<th>Data[1]</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_SELECT_NODE</td>
<td>2</td>
<td>boot_version</td>
<td>00h</td>
<td>Communication close</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01h</td>
<td>Communication open</td>
</tr>
</tbody>
</table>

Note: Data[0] contains the bootloader version.

If the communication is closed then all the others messages won’t be managed by bootloader.

**ID_SELECT_NODE Flow Description**

**Example**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Length</th>
<th>Data[0]</th>
<th>Data[1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST</td>
<td>01</td>
<td>FF</td>
<td></td>
</tr>
<tr>
<td>BOOTLOADER</td>
<td>02</td>
<td>01 01</td>
<td></td>
</tr>
</tbody>
</table>
Programming the Flash or EEPROM Data

The ID_PROG_START flow described below shows how to program data in the Flash memory or in the EEPROM data memory. This operation can be executed only with a device previously opened in communication.

1. The first step is to indicate which memory area (Flash or EEPROM data) is selected and the range address to program.
2. The second step is to transmit the data.

The bootloader programs on a page of 128 bytes basis when it is possible.

The host must take care of the following:
- The data to program transmitted within a CAN frame are in the same page.
- To transmit 8 data bytes in CAN message when it is possible

3. To start the programming operation, the Host sends a “start programming” CAN message (ID_PROG_START) with the area memory selected in data[0], the start address and the end address passed in parameter.

Requests from Host

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_PROG_START</td>
<td>5</td>
<td>00h</td>
<td>address_start</td>
<td></td>
<td></td>
<td>address_end</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>01h</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes:
1. Data[0] chooses the area to program:
   - 00h: Flash
   - 01h: EEPROM data
2. Address_start gives the start address of the programming command.
3. Address_end gives the last address of the programming command.

Answers from Bootloader

The device has two possible answers:
- If the chip is protected from program access an “Error” CAN message is sent (see Section “Error Message Description”, page 22).
- Otherwise an acknowledge is sent.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_PROG_START</td>
<td>0</td>
</tr>
</tbody>
</table>

The second step of the programming operation is to send data to program.

Request from Host

To send data to program, the Host sends a ‘programming data’ CAN message (ID_PROG_DATA) with up to 8 data by message and must wait for the answer of the device before sending the next data to program.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Length</th>
<th>Data[0]</th>
<th>...</th>
<th>Data[7]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_PROG_DATA</td>
<td>up to 8</td>
<td>x</td>
<td>...</td>
<td>x</td>
</tr>
</tbody>
</table>
The device has two possible answers:

- If the device is ready to receive new data, it sends a “programming data” CAN message (ID_PROG_DATA) with the result Command_new passed in parameter.
- If the device has finished the programming, it sends a “programming data” CAN message (ID_PROG_DATA) with the result Command_ok passed in parameter.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Length</th>
<th>Data[0]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_PROG_DATA</td>
<td>1</td>
<td>00h</td>
<td>Command OK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01h</td>
<td>Command fail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>02h</td>
<td>Command new data</td>
</tr>
</tbody>
</table>

**ID_PROG_DATA Flow Description**

**Host**
- **Send prog_start message with addresses**
  - **OR**
  - **Wait Prog Start**
  - **Command aborted**
- **Wait ERROR**
- **Send prog_data message with 8 datas**
  - **OR**
  - **Wait COMMAND_N**
  - **Wait COMMAND_OK**
  - **COMMAND FINISHED**

**Bootloader**
- **Send ProgStart ID_PROG_START Message**
  - **Wait Prog start**
  - **SSB = Level 0**
  - **Send ERROR**
- **Send prog_data message with 8 datas**
  - **Wait DATA prog**
  - **Column Latch Full**
  - **All bytes received**
  - **Wait Programming**
  - **All bytes received**
  - **Send COMMAND_NEW_DATA**
  - **Send COMMAND_OK**
  - **COMMAND FINISHED**
The ID_DISPLAY_DATA flow described below allows the user to read data in the Flash memory or in the EEPROM data memory. A blank check command on the Flash memory is possible with this flow.

This operation can be executed only with a device previously opened in communication.

To start the reading operation, the Host sends a “Display Data” CAN message (Id_display_data) with the area memory selected, the start address and the end address passed in parameter.

The device splits into block of 8 bytes data to transfer to the Host if the number of data to display is greater than 8 data bytes.

### Requests from Host

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_DISPLAY_DATA</td>
<td>5</td>
<td>00h</td>
<td>address_start</td>
<td></td>
<td>address_end</td>
<td></td>
</tr>
</tbody>
</table>

Notes:
1. Data[0] selects the area to read and the operation
   - 00h: Display Flash
   - 01h: Blank Check on the Flash
   - 02h: Display EEPROM data
2. The Address_start gives the start address to read.
3. The Address_end gives the last address to read.

### Answers from Bootloader

The device has two possible answers:

- If the chip is protected from read access an “Error” CAN message is sent (see Section “Error Message Description”, page 22).
- Otherwise:
  - for a display command the device starts to send the data up to 8 by frame to the host. For a blank check command, the device sends a result OK or the first address not erased.

Answer to a read command:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Length</th>
<th>Data[n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_DISPLAY_DATA</td>
<td>n</td>
<td>x</td>
</tr>
</tbody>
</table>
Answer to a blank check command:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Length</th>
<th>Data[0]</th>
<th>Data[1]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_DISPLAY_DATA</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>Blank Check OK</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>Address_start</td>
</tr>
</tbody>
</table>

Flow Description

Host

- Send Display_data Message with Addresses or Blank Check
- OR
- Wait ERROR
- COMMAND ABORTED
- OR
- Wait Data Display
- All Data Read
- COMMAND FINISHED
- OR
- Wait COMMAND_OK
- COMMAND FINISHED

Bootloader

- ID_DISPLAY_DATA Message
- Wait Display Data
- SSB = Level 2
- Send ERROR
- Blank Command
- Read Data
- All Data Read
- nb Max by Frame
- Send Data Read
- Verify Memory
- COMMAND FINISHED
- Blank Check
- Send COMMAND_OK
- COMMAND FINISHED
- Send COMMAND_OK
The ID_WRITE_COMMAND flow described below allows the user to program Configuration Information regarding the bootloader functionality.

This operation can be executed only with a device previously opened in communication.

The Configuration Information can be divided in two groups:

- **Boot Process Configuration:**
  - BSB
  - SBV
  - Fuse bits (BLJB and X2 bits) (see Section “Mapping and Default Value of Hardware Security Byte”, page 4)

- **CAN Protocol Configuration:**
  - BTC_1, BTC_2, BTC_3
  - SSB
  - EB
  - NNB
  - CRIS

Note: The CAN protocol configuration bytes are taken into account only after the next reset.

To start the programming operation, the Host sends a “write” CAN message (ID_WRITE_COMMAND) with the area selected, the value passed in parameter.

Take care that the Program Fuse bit command programs the 4 Fuse bits at the same time.
### Requests from Host

The device has two possible answers:

- If the chip is protected from program access an “Error” CAN message is sent (see Section “Error Message Description”, page 22).
- Otherwise an acknowledge “Command OK” is sent.

### Answers from Bootloader

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_WRITE_COMMAND</td>
<td>3</td>
<td>01h</td>
<td>value</td>
<td></td>
<td>write value in BSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00h</td>
<td></td>
<td></td>
<td>write value in SBV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01h</td>
<td></td>
<td></td>
<td>write value in SSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>05h</td>
<td></td>
<td></td>
<td>write value in EB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>06h</td>
<td></td>
<td></td>
<td>write value in BTC_1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1Ch</td>
<td></td>
<td></td>
<td>write value in BTC_2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1Dh</td>
<td></td>
<td></td>
<td>write value in BTC_3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1Eh</td>
<td></td>
<td></td>
<td>write value in NNB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1Fh</td>
<td></td>
<td></td>
<td>write value in CRIS</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>02h</td>
<td>00h</td>
<td>value</td>
<td>write value in Fuse bits</td>
</tr>
</tbody>
</table>

### ID_WRITE_COMMAND Flow Description

**Host**

- Send Write_Command
- OR
  - Wait ERROR_SECURITY
  - COMMAND ABORTED
- Wait COMMAND_OK
  - COMMAND FINISHED

**Bootloader**

- Wait Write_Command
- NO_SECURITY
- Send ERROR_SECURITY
- Write Data
- Send COMMAND_OK
  - COMMAND FINISHED
The ID_READ_COMMAND flow described below allows the user to read the configuration or manufacturer information. This operation can be executed only with a device previously opened in communication.

To start the reading operation, the Host sends a “Read Command” CAN message (ID_READ_COMMAND) with the information selected passed in data field.

### Requests from Host

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Length</th>
<th>Data[0]</th>
<th>Data[1]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ID_READ_COMMAND</strong></td>
<td>2</td>
<td>00h</td>
<td></td>
<td>Read Bootloader version</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>01h</td>
<td></td>
<td>Read Device ID1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>02h</td>
<td></td>
<td>Read Device ID2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>00h</td>
<td></td>
<td>Read BSB</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>01h</td>
<td></td>
<td>Read SBV</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>05h</td>
<td></td>
<td>Read SSB</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>06h</td>
<td></td>
<td>Read EB</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1Ch</td>
<td></td>
<td>Read BTC_1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1Dh</td>
<td></td>
<td>Read BTC_2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1 Eh</td>
<td></td>
<td>Read BTC_3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1Fh</td>
<td></td>
<td>Read NNB</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>20h</td>
<td></td>
<td>Read CRIS</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>30h</td>
<td></td>
<td>Read Manufacturer Code</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>31h</td>
<td></td>
<td>Read Family Code</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>60h</td>
<td></td>
<td>Read Product Name</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>61h</td>
<td></td>
<td>Read Product Revision</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>02h</td>
<td>00h</td>
<td>Read HSB (Fuse bits)</td>
</tr>
</tbody>
</table>
The device has two possible answers:

- If the chip is protected from read access an “Error” CAN message is sent (see Section “Error Message Description”).
- Otherwise:
  the device answers with a Read Answer CAN message (ID_READ_COMMAND).

### Flow Description

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Length</th>
<th>Data[n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_READ_COMMAND</td>
<td>1</td>
<td>value</td>
</tr>
</tbody>
</table>

### Example

**Read Bootloader Version**
```
   identifier control data
  HOST   Id_read_command  02 00 00
 BOOTLOADER  Id_read_command  01 55 // Bootloader version 55h
```

**Read SBV**
```
   identifier control data
  HOST   Id_read_command  02 01 01
 BOOTLOADER  Id_read_command  01 F5 // SBV = F5h
```

**Read Fuse bit**
```
   identifier control data
  HOST   Id_read_command  01 02
 BOOTLOADER  Id_read_command  01 F0 // Fuse bit = F0h
```
Erasing the Flash
The ID_WRITE_COMMAND flow described below allows the user to erase the Flash memory.
This operation can be executed only with a device previously opened in communication.

Two modes of Flash erasing are possible:
- Full Chip erase
- Block erase

The Full Chip erase command erases the whole Flash (32 Kbytes) and sets some Configuration Bytes to their default values:
- BSB = FFh
- SBV = FFh
- SSB = FFh (NO_SECURITY)

The Block erase command erases only a part of the Flash.
Three Blocks are defined in the T89C51CC01:
- block0 (from 0000h to 1FFFh)
- block1 (from 2000h to 3FFFh)
- block2 (from 4000h to 7FFFh)

To start the erasing operation, the Host sends a “write” CAN message (ID_WRITE_COMMAND).

Requests from Host

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Length</th>
<th>Data[0]</th>
<th>Data[1]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_WRITE_COMMAND</td>
<td>2</td>
<td>00h</td>
<td></td>
<td>00h  Erase block0 (0K to 8K)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20h  Erase block1 (8K to 16K)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>40h  Erase block2 (16K to 32K)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>FFh  Full chip erase</td>
</tr>
</tbody>
</table>

Answers from Bootloader
As the Program Configuration Information flows, the erase block command has two possible answers:
- If the chip is protected from program access an “Error” CAN message is sent (see Section “Error Message Description”, page 22).
- Otherwise an acknowledge is sent.

The full chip erase is always executed whatever the Software Security Byte value is.
On a full chip erase command an acknowledge “Command OK” is sent.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Length</th>
<th>Data[0]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_WRITE_COMMAND</td>
<td>1</td>
<td>00h</td>
<td>Command OK</td>
</tr>
</tbody>
</table>

Example

Full Chip Erase

<table>
<thead>
<tr>
<th>Identifier control data</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST</td>
</tr>
<tr>
<td>BOOTLOADER</td>
</tr>
</tbody>
</table>
Starting the Application

The ID_WRITE_COMMAND flow described below allows to start the application directly from the bootloader upon a specific command reception.

This operation can be executed only with a device previously opened in communication.

Two options are possible:

- Start the application with a reset pulse generation (using watchdog).
  When the device receives this command, the watchdog is enabled and the bootloader enters a waiting loop until the watchdog resets the device.
  Take care that if an external reset chip is used, the reset pulse in output may be wrong and in this case the reset sequence is not correctly executed.

- Start the application without reset
  A jump at the address 0000h is used to start the application without reset.

To start the application, the Host sends a “Start Application” CAN message (ID_WRITE_COMMAND) with the corresponding option passed in parameter.

Requests from Host

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_WRITE_COMMAND</td>
<td>2</td>
<td>03h</td>
<td>00h</td>
<td>-</td>
<td>-</td>
<td>Start Application with a reset pulse generation</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>01h</td>
<td>address</td>
<td></td>
<td></td>
<td>Start Application with a jump at “address”</td>
</tr>
</tbody>
</table>

Answer from Bootloader

No answer is returned by the device.

Example

Start application

identifier control data

HOST

BOOTLOADER

No answer

Error Message Description

The error message is implemented to report when an action required is not possible.

- At the moment only the security error is implemented and only the device can answer this kind of CAN message (ID_ERROR).

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Length</th>
<th>Data[0]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_ERROR</td>
<td>1</td>
<td>00h</td>
<td>Software Security Error</td>
</tr>
</tbody>
</table>
In-Application Programming/Self-programming

The IAP allows to reprogram a microcontroller on-chip Flash memory without removing it from the system and while the embedded application is running.

The user application can call Application Programming Interface (API) routines allowing IAP. These API are executed by the bootloader.

To call the corresponding API, the user must use a set of Flash_api routines which can be linked with the application.

Example of Flash_api routines are available on the Atmel web site on the software application note:

*C Flash Drivers for the T89C51CC01CA for Keil Compilers*

The Flash_api routines on the package work only with the CAN bootloader.

The Flash_api routines are listed in Appendix-B.

API Call

Process

The application selects an API by setting the 4 variables available when the Flash_api library is linked to the application.

These four variables are located in RAM at fixed address:

- api_command: 1Ch
- api_value: 1Dh
- api_dph: 1Eh
- api_dpl: 1Fh

All calls are made through a common interface “USER_CALL” at the address FFC0h.

The jump at the USER_CALL must be done by LCALL instruction to be able to comeback in the application.

Before jump at the USER_CALL, the bit ENBOOT in AUXR1 register must be set.

Constraints

The interrupts are not disabled by the bootloader.

Interrupts must be disabled by user prior to jump to the USER_CALL, then re-enabled when returning.

Interrupts must also be disabled before accessing EEPROM data then re-enabled after.

The user must take care of hardware watchdog before launching a Flash operation.

For more information regarding the Flash writing time see the T89C51CC01 datasheet.
API Commands

Several types of APIs are available:
- Read/Program Flash and EEPROM data Memory
- Read Configuration and Manufacturer Information
- Program Configuration Information
- Erase Flash
- Start bootloader

Read/Program Flash and EEPROM Data Memory

All routines to access EEPROM data are managed directly from the application without using bootloader resources.

The bootloader is not used to read the Flash memory.

For more details on these routines see the T89C51CC01 datasheet sections “Program/Code Memory” and “EEPROM Data Memory”

Two routines are available to program the Flash:
- __api_wr_code_byte
- __api_wr_code_page

- The application program loads the column latches of the Flash then calls the __api_wr_code_byte or __api_wr_code_page see datasheet in section “Program/Code Memory”.

Parameter Settings

<table>
<thead>
<tr>
<th>API Name</th>
<th>api_command</th>
<th>api_dph</th>
<th>api_dpl</th>
<th>api_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>__api_wr_code_byte</td>
<td>0Dh</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>__api_wr_code_page</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Instruction: LCALL FFC0h.

Note: No special resources are used by the bootloader during this operation.

Read Configuration and Manufacturer Information

- Parameter Settings

<table>
<thead>
<tr>
<th>API Name</th>
<th>api_command</th>
<th>api_dph</th>
<th>api_dpl</th>
<th>api_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>__api_rd_HSB</td>
<td>08h</td>
<td>-</td>
<td>00h</td>
<td>return HSB</td>
</tr>
<tr>
<td>__api_rd_BSB</td>
<td>05h</td>
<td>-</td>
<td>00h</td>
<td>return BSB</td>
</tr>
<tr>
<td>__api_rd_SBV</td>
<td>05h</td>
<td>-</td>
<td>01h</td>
<td>return SBV</td>
</tr>
<tr>
<td>__api_rd_SSB</td>
<td>05h</td>
<td>-</td>
<td>05h</td>
<td>return SSB</td>
</tr>
<tr>
<td>__api_rd_EB</td>
<td>05h</td>
<td>-</td>
<td>06h</td>
<td>return EB</td>
</tr>
<tr>
<td>__api_rd_CANBTC1</td>
<td>05h</td>
<td>-</td>
<td>1Ch</td>
<td>return CANBTC1</td>
</tr>
<tr>
<td>__api_rd_CANBTC2</td>
<td>05h</td>
<td>-</td>
<td>1Dh</td>
<td>return CANBTC2</td>
</tr>
<tr>
<td>__api_rd_CANBTC3</td>
<td>05h</td>
<td>-</td>
<td>1Eh</td>
<td>return CANBTC3</td>
</tr>
<tr>
<td>__api_rd_NNB</td>
<td>05h</td>
<td>-</td>
<td>1Fh</td>
<td>return NNB</td>
</tr>
<tr>
<td>__api_rd_CRIS</td>
<td>05h</td>
<td>-</td>
<td>20h</td>
<td>return CRIS</td>
</tr>
<tr>
<td>__api_rd_manufacturer</td>
<td>05h</td>
<td>-</td>
<td>30h</td>
<td>return manufacturer id</td>
</tr>
<tr>
<td>__api_rd_device_id1</td>
<td>05h</td>
<td>-</td>
<td>31h</td>
<td>return id1</td>
</tr>
</tbody>
</table>
**Program Configuration Information**

- **Instruction**: LCALL FFC0h.
- At the complete API execution by the bootloader, the value to read is in the api_value variable.

  **Note**: No special resources are used by the bootloader during this operation.

### Parameter Settings

<table>
<thead>
<tr>
<th>API Name</th>
<th>api_command</th>
<th>api_dph</th>
<th>api_dpl</th>
<th>api_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>__api_rd_device_id2</td>
<td>05h</td>
<td>-</td>
<td>60h</td>
<td>return id2</td>
</tr>
<tr>
<td>__api_rd_device_id3</td>
<td>05h</td>
<td>-</td>
<td>61h</td>
<td>return id3</td>
</tr>
<tr>
<td>__api_rd_bootloader_version</td>
<td>0Eh</td>
<td>-</td>
<td>00h</td>
<td>return value</td>
</tr>
</tbody>
</table>

### Erasing the Flash

The T89C51CC01 Flash memory is divided in three blocks of 8K Bytes:

- **Block 0**: from address 0000h to 1FFFh
- **Block 1**: from address 2000h to 3FFFh
- **Block 2**: from address 4000h to 7FFFh

These three blocks contain 128 pages.

### Parameter Settings

<table>
<thead>
<tr>
<th>API Name</th>
<th>api_command</th>
<th>api_dph</th>
<th>api_dpl</th>
<th>api_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>__api_erase_block0</td>
<td>00h</td>
<td>00h</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Starting the Bootloader

There are two start bootloader routines possible:
1. This routine allows to start at the beginning of the bootloader or after a reset. After calling this routine the regular boot process is performed and the communication must be opened before any action.
   - No special parameter setting
   - Set bit ENBOOT in AUXR1 register
   - Instruction: LJUMP or LCALL at address F800h
2. This routine allows to start the bootloader with the CAN bit configuration of the application and start with the state "communication open". That means the bootloader will return the message "ID_SELECT_NODE" with the field com port open.
   - No special parameter setting
   - Set bit ENBOOT in AUXR1 register
   - Instruction: LJUMP or LCALL at address FF00h

---

<table>
<thead>
<tr>
<th>API Name</th>
<th>api_command</th>
<th>api_dph</th>
<th>api_dpl</th>
<th>api_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>__api_erase_block1</td>
<td>00h</td>
<td>20h</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>__api_erase_block2</td>
<td>00h</td>
<td>40h</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
# Appendix-A

## Table 1. Summary of Frames from Host

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Id_select_node</td>
<td>1</td>
<td>num node</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Open/Close communication</td>
</tr>
<tr>
<td>Id_prog_start</td>
<td>5</td>
<td>00h</td>
<td>start_address</td>
<td>end_address</td>
<td>Init Flash programming</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CRIS:0h)</td>
<td></td>
<td>01h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Init EEPROM programming</td>
</tr>
<tr>
<td>Id_prog_data</td>
<td>n</td>
<td>data[0:8]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Data to program</td>
</tr>
<tr>
<td>(CRIS:2h)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Id_display_data</td>
<td>5</td>
<td>00h</td>
<td>start_address</td>
<td>end_address</td>
<td>Display Flash Data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CRIS:3h)</td>
<td></td>
<td>01h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Blank Check in Flash</td>
</tr>
<tr>
<td></td>
<td></td>
<td>02h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Display EEPROM Data</td>
</tr>
<tr>
<td>Id_write_command</td>
<td>2</td>
<td>00h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Erase block0 (0K to 8K)</td>
</tr>
<tr>
<td>(CRIS:4h)</td>
<td></td>
<td>20h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Erase block1 (8K to 16K)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Erase block2 (16K to 32K)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FFh</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Full-chip Erase</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>01h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Write value in BSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Write value in SBV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>05h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Write value in SSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>06h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Write value in EB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1Ch</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Write BTC_1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1Dh</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Write BTC_2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1Eh</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Write BTC_3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1Fh</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Write NNB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Write CRIS</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>02h</td>
<td>00h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Write value in Fuse (HSB)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>03h</td>
<td>00h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Start Application with Hardware Reset</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>01h</td>
<td>address</td>
<td>-</td>
<td></td>
<td></td>
<td>Start Application by LJMP address</td>
</tr>
</tbody>
</table>
### Table 1. Summary of Frames from Host (Continued)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Id_read_command</td>
<td>2</td>
<td>00h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Read Bootloader Version</td>
</tr>
<tr>
<td>(CRIS:5h)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>01h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Read Device ID1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>02h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Read Device ID2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Id_read_command</td>
<td>2</td>
<td>00h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Read BSB</td>
</tr>
<tr>
<td>(CRIS:5h)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>01h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Read SBV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>05h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Read SSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>06h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Read EB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>30h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Read Manufacturer Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>31h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Read Family Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>60h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Read Product Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>61h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Read Product Revision</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1Ch</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Read BTC_1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1Dh</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Read BTC_2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1Eh</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Read BTC_3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1Fh</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Read NNB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>20h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Read CRIS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>02h</td>
<td>00h</td>
<td>-</td>
<td>-</td>
<td>Read HSB</td>
</tr>
</tbody>
</table>

### Table 2. Summary of Frames from Target (Bootloader)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Id_select_node</td>
<td>2</td>
<td>Boot version</td>
<td>00h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>communication close</td>
</tr>
<tr>
<td>(CRIS:0h)</td>
<td></td>
<td></td>
<td>01h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>communication open</td>
</tr>
<tr>
<td>Id_prog_start</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Command OK</td>
</tr>
<tr>
<td>(CIRS:1h)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Id_prog_data</td>
<td>1</td>
<td>00h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Command OK</td>
</tr>
<tr>
<td>(CRIS:2h)</td>
<td></td>
<td></td>
<td>01h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Command fail</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>02h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Command New Data</td>
</tr>
<tr>
<td>Id_display_data</td>
<td>n</td>
<td>n data (n = 0 to 8)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Data read</td>
</tr>
<tr>
<td>(CRIS:3h)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Blank Check OK</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>first address not blank</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Blank Check fail</td>
</tr>
<tr>
<td>Id_write_command</td>
<td>1</td>
<td>00h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Command OK</td>
</tr>
<tr>
<td>(CIRS:4h)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Id_read_command</td>
<td>1</td>
<td>Value</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Read Value</td>
</tr>
<tr>
<td>(CRIS:5h)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2. Summary of Frames from Target (Bootloader) (Continued)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Id_error (CRIS:6h)</td>
<td>1</td>
<td>00h</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Software Security Error</td>
</tr>
</tbody>
</table>
### Appendix-B

#### Table 3. API Summary

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Bootloader Execution</th>
<th>api_command</th>
<th>api_dph</th>
<th>api_dpl</th>
<th>api_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>__api_rd_code_byte</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>__api_wr_code_byte</td>
<td>yes</td>
<td>0Dh</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>__api_wr_code_page</td>
<td>yes</td>
<td>0Dh</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>__api_erase block0</td>
<td>yes</td>
<td>00h</td>
<td>00h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>__api_erase block1</td>
<td>yes</td>
<td>00h</td>
<td>20h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>__api_erase block2</td>
<td>yes</td>
<td>00h</td>
<td>40h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>__api_rd_HSB</td>
<td>yes</td>
<td>08h</td>
<td>-</td>
<td>00h</td>
<td>return value</td>
</tr>
<tr>
<td>__api_clr_BLJB</td>
<td>yes</td>
<td>07h</td>
<td>-</td>
<td>-</td>
<td>(HSB &amp; BFh)</td>
</tr>
<tr>
<td>__api_set_BLJB</td>
<td>yes</td>
<td>07h</td>
<td>-</td>
<td>-</td>
<td>HSB &amp; BFh</td>
</tr>
<tr>
<td>__api_clr_X2</td>
<td>yes</td>
<td>07h</td>
<td>-</td>
<td>-</td>
<td>(HSB &amp; 7Fh)</td>
</tr>
<tr>
<td>__api_set_X2</td>
<td>yes</td>
<td>07h</td>
<td>-</td>
<td>-</td>
<td>HSB &amp; 7Fh</td>
</tr>
<tr>
<td>__api_rd_BSB</td>
<td>yes</td>
<td>05h</td>
<td>-</td>
<td>00h</td>
<td>return value</td>
</tr>
<tr>
<td>__api_wr_BSB</td>
<td>yes</td>
<td>04h</td>
<td>-</td>
<td>00h</td>
<td>value</td>
</tr>
<tr>
<td>__api_rd_SBV</td>
<td>yes</td>
<td>05h</td>
<td>-</td>
<td>01h</td>
<td>return value</td>
</tr>
<tr>
<td>__api_wr_SBV</td>
<td>yes</td>
<td>04h</td>
<td>-</td>
<td>01h</td>
<td>value</td>
</tr>
<tr>
<td>__api_erase_SBV</td>
<td>yes</td>
<td>04h</td>
<td>-</td>
<td>01h</td>
<td>FFh</td>
</tr>
<tr>
<td>__api_rd_SSB</td>
<td>yes</td>
<td>05h</td>
<td>-</td>
<td>05h</td>
<td>return value</td>
</tr>
<tr>
<td>__api_wr_SSB</td>
<td>yes</td>
<td>04h</td>
<td>-</td>
<td>05h</td>
<td>value</td>
</tr>
<tr>
<td>__api_rd_EB</td>
<td>yes</td>
<td>05h</td>
<td>-</td>
<td>06h</td>
<td>return value</td>
</tr>
<tr>
<td>__api_wr_EB</td>
<td>yes</td>
<td>04h</td>
<td>-</td>
<td>06h</td>
<td>value</td>
</tr>
<tr>
<td>__api_rd_CANBTC1</td>
<td>yes</td>
<td>05h</td>
<td>-</td>
<td>1Ch</td>
<td>return value</td>
</tr>
<tr>
<td>__api_wr_CANBTC1</td>
<td>yes</td>
<td>04h</td>
<td>-</td>
<td>1Ch</td>
<td>value</td>
</tr>
<tr>
<td>__api_rd_CANBTC2</td>
<td>yes</td>
<td>05h</td>
<td>-</td>
<td>1Dh</td>
<td>return value</td>
</tr>
<tr>
<td>__api_wr_CANBTC2</td>
<td>yes</td>
<td>04h</td>
<td>-</td>
<td>1Dh</td>
<td>value</td>
</tr>
<tr>
<td>__api_rd_CANBTC3</td>
<td>yes</td>
<td>05h</td>
<td>-</td>
<td>1Eh</td>
<td>return value</td>
</tr>
<tr>
<td>__api_wr_CANBTC3</td>
<td>yes</td>
<td>04h</td>
<td>-</td>
<td>1Eh</td>
<td>value</td>
</tr>
<tr>
<td>__api_rd_NNB</td>
<td>yes</td>
<td>05h</td>
<td>-</td>
<td>1Fh</td>
<td>return value</td>
</tr>
<tr>
<td>__api_wr_NNB</td>
<td>yes</td>
<td>04h</td>
<td>-</td>
<td>1Fh</td>
<td>value</td>
</tr>
<tr>
<td>__api_rd_CRIS</td>
<td>yes</td>
<td>05h</td>
<td>-</td>
<td>20h</td>
<td>return value</td>
</tr>
<tr>
<td>__api_wr_CRIS</td>
<td>yes</td>
<td>04h</td>
<td>-</td>
<td>20h</td>
<td>value</td>
</tr>
<tr>
<td>__api_rd_manufacturer</td>
<td>yes</td>
<td>05h</td>
<td>-</td>
<td>30h</td>
<td>return value</td>
</tr>
<tr>
<td>__api_rd_device_id1</td>
<td>yes</td>
<td>05h</td>
<td>-</td>
<td>31h</td>
<td>return value</td>
</tr>
</tbody>
</table>
Table 3. API Summary (Continued)

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Bootloader Execution</th>
<th>api_command</th>
<th>api_dph</th>
<th>api_dpl</th>
<th>api_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>__api_rd_device_id2</td>
<td>yes</td>
<td>05h</td>
<td>-</td>
<td>60h</td>
<td>return value</td>
</tr>
<tr>
<td>__api_rd_device_id3</td>
<td>yes</td>
<td>05h</td>
<td>-</td>
<td>61h</td>
<td>return value</td>
</tr>
<tr>
<td>__api_rd_bootloader_version</td>
<td>yes</td>
<td>0Eh</td>
<td>-</td>
<td>00h</td>
<td>return value</td>
</tr>
<tr>
<td>__api_eeprom_busy</td>
<td>no</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>__api_rd_eeprom_byte</td>
<td>no</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>__api_wr_eeprom_byte</td>
<td>no</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>__api_start_bootloader</td>
<td>no</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>__api_start_isp</td>
<td>no</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3. API Summary (Continued)

Document
Revision
History

Changes from 4210C - 12/03 to 4210D - 05/06

1. Changes to full chip erase command.

Changes from 4210D - 05/06 to 4210E - 03/08

1. Update of Bootloader version.