

Features

- Protocol
 - UART Used as a Physical Layer
 - Based on the Intel Hex-type Records
 - 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 UART bootloader functionalities as well as the serial protocol to efficiently perform operations on the on-chip Flash memory. Additional information on the T89C5115 product can be found in the T89C5115 datasheet and the T89C5115 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.

| Bootloader Revision | Purpose of Modifications | Date |
|---------------------|---|------------|
| Revisions 1.0.0 | First release | 03/12/2001 |
| Revision 1.2.1 | Standardization of tasks in source program. | 19/03/2007 |



**80C51
Microcontrollers**

**T89C5115
UART
Bootloader**



Functional Description

The T89C5115 Bootloader facilitates In-System Programming and In-Application Programming.

In-System Programming (ISP) Capability

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

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

In-Application Programming or Self-programming capability

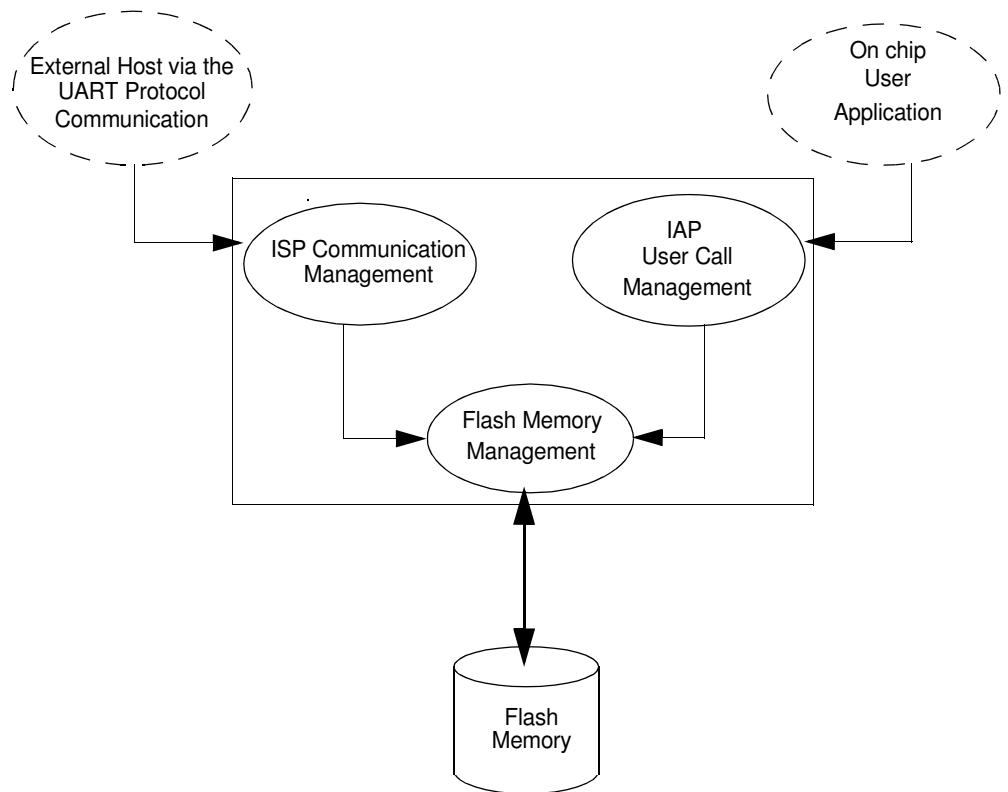
In-Application Programming (IAP) allows the reprogramming of a microcontroller's on-chip Flash memory without removing it from the system and while the embedded application is running.

The UART 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



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 Serial protocol (see Section "Protocol", page 9). This process translates serial communication frames (UART) 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.

| Mnemonic | Description | Default Value |
|-----------------------|------------------------|---------------|
| BSB | Boot Status Byte | FFh |
| SBV | Software Boot Vector | FCh |
| P1_CF | Port 1 Configuration | FEh |
| P3_CF | Port 3 Configuration | FFh |
| P4_CF | Port 4 Configuration | FFh |
| SSB | Software Security Byte | FFh |
| EB | Extra Byte | FFh |
| Manufacturer | | 58h |
| ID1: Family code | | D7h |
| ID2: Product Name | | BBh |
| ID3: Product Revision | | FFh |

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 Less Significant Byte (LSB) can only be read by software and written by hardware in parallel mode (with parallel programmer devices).

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

Note: 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 an 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 an error message.

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

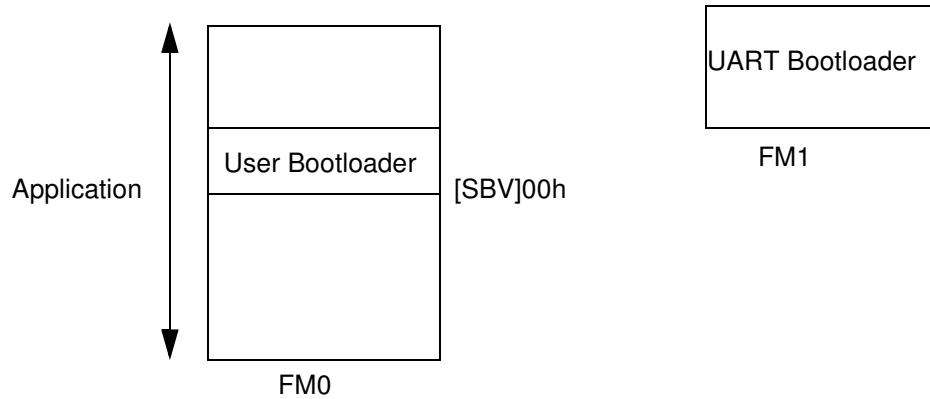
| | Level 0 | Level 1 | Level 2 |
|-------------------|--------------------------|--------------------------|--------------------------|
| Flash/EEPROM | Any access allowed | Read only access allowed | All access not allowed |
| Fuse bit | Any access allowed | Read only access allowed | All access not allowed |
| BSB & SBV & EB | Any access allowed | Read only access allowed | All access not allowed |
| SSB | Any access allowed | Write level2 allowed | Read only access allowed |
| Manufacturer info | Read only access allowed | Read only access allowed | Read only access allowed |
| Bootloader info | Read only access allowed | Read only access allowed | Read only access allowed |
| Erase block | Allowed | Not allowed | Not allowed |
| Full chip erase | Allowed | Allowed | Allowed |
| Blank check | Allowed | Allowed | Allowed |

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 the 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.

This software program is available free of charge from the Atmel web site.

In-System Programming

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

This section describes how to start the UART bootloader and all higher level protocol over the serial line.

Boot Process

The bootloader can be activated in two ways:

- Hardware condition
- Regular boot process

Hardware Condition

The Hardware Condition forces the bootloader execution from reset.

The default factory Hardware Condition is assigned to port P1.

- P1 must be equal to FEh

In order to offer the best flexibility, the user can define its own Hardware Condition on one of the following Ports:

- Port1
- Port3
- Port4 (only bit0 and bit1)

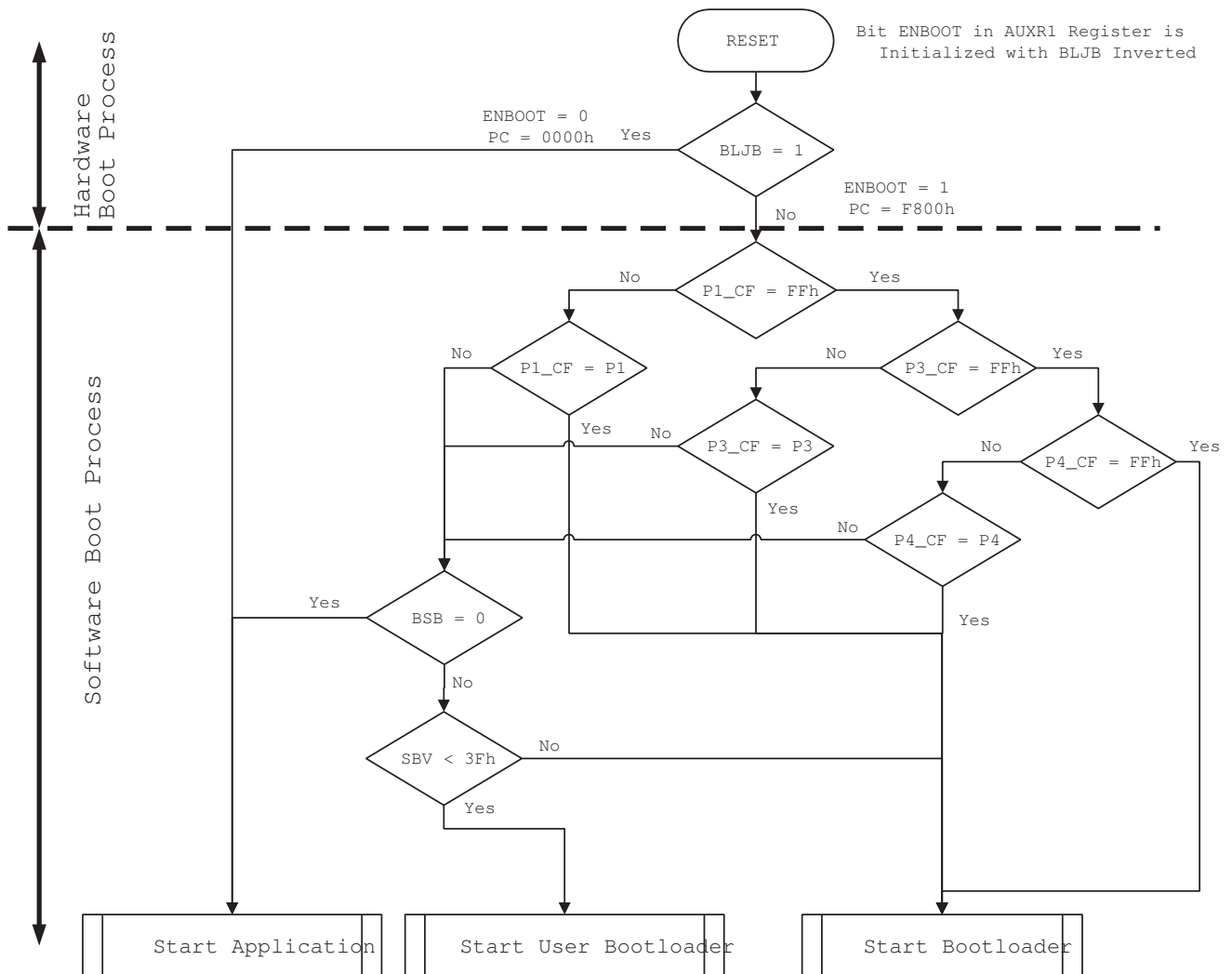
The Hardware Condition configuration is stored in three bytes called P1_CF, P3_CF, P4_CF.

These bytes can be modified by the user through a set of API or through an ISP command.

There is a priority between P1_CF, P3_CF and P4_CF (see boot process diagram).

Note: The BLJB must be at 0 (programmed) to be able to restart the bootloader.
If the BLJB is equal to 1 (unprogrammed) only the hardware parallel programmer can change this bit (see T89C5115 Datasheet for more detail).

Figure 3. Regular Boot Process



Physical Layer

The UART used to transmit information has the following configuration:

- Character: 8-bit data
- Parity: none
- Stop: 2 bit
- Flow control: none
- Baud rate: autobaud is performed by the bootloader to compute the baud rate chosen by the host.

Frame Description

The Serial Protocol is based on the Intel Hex-type records.

Intel Hex records consist of ASCII characters used to represent hexadecimal values and are summarized below.

Table 1. Intel Hex Type Frame

| Record Mark ':' | Record length | Load Offset | Record Type | Data or Info | Checksum |
|-----------------|---------------|-------------|-------------|--------------|----------|
| 1 byte | 1 byte | 2 bytes | 1 bytes | n byte | 1 byte |

- Record Mark:
 - Record Mark is the start of frame. This field must contain ':'.
- Record length:
 - Record length specifies the number of Bytes of information or data which follows the Record Type field of the record.
- Load Offset:
 - Load Offset specifies the 16-bit starting load offset of the data Bytes, therefore this field is used only for Data Program Record.
- Record Type:
 - Record Type specifies the command type. This field is used to interpret the remaining information within the frame.
- Data/Info:
 - Data/Info is a variable length field. It consists of zero or more Bytes encoded as pairs of hexadecimal digits. The meaning of data depends on the Record Type.
- Checksum:
 - The two's complement of the 8-bit Bytes that result from converting each pair of ASCII hexadecimal digits to one Byte of binary, and including the Record Length field to and including the last Byte of the Data/Info field. Therefore, the sum of all the ASCII pairs in a record after converting to binary, from the Record Length field to and including the Checksum field, is zero.

Protocol

Overview

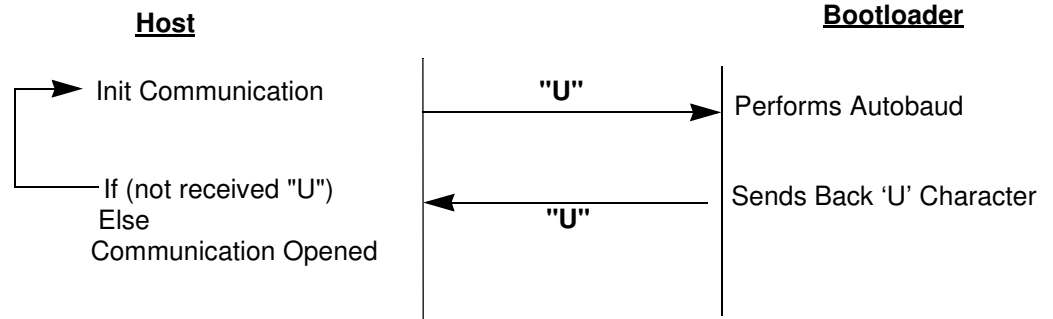
An initialization step must be performed after each Reset. After a microcontroller reset, the bootloader waits for an autobaud sequence (see Section “Autobaud Performances”).

When the communication is initialized, the protocol depends on the record type issued by the host.

Communication Initialization

The host initiates the communication by sending a 'U' character to help the bootloader to compute the baudrate (autobaud).

Figure 4. Initialization



Autobaud Performances

The bootloader supports a wide range of baud rates. It is also adaptable to a wide range of oscillator frequencies. This is accomplished by measuring the bit-time of a single bit in a received character. This information is then used to program the baud rate in terms of timer counts based on the oscillator frequency. Table 2 shows the autobaud capabilities.

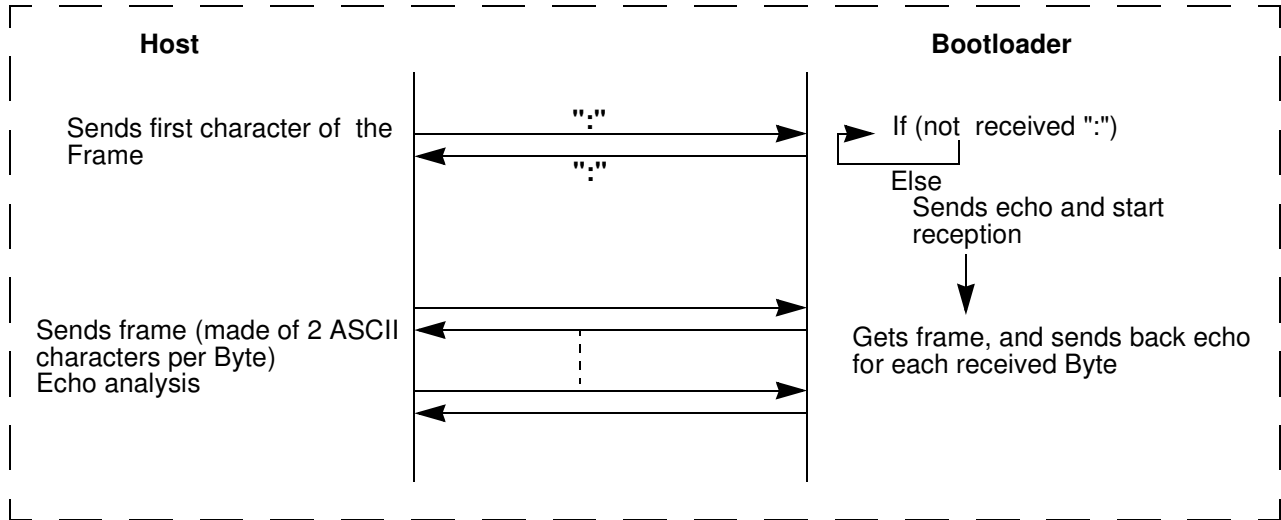
Table 2. Autobaud Performances

| Frequency (MHz) Baudrate (kHz) | 1.8432 | 2 | 2.4576 | 3 | 3.6864 | 4 | 5 | 6 | 7.3728 |
|-----------------------------------|--------|----|---------|----|--------|----|----|----|--------|
| 2400 | OK | OK | OK | OK | OK | OK | OK | OK | OK |
| 4800 | OK | - | OK | OK | OK | OK | OK | OK | OK |
| 9600 | OK | - | OK | OK | OK | OK | OK | OK | OK |
| 19200 | OK | - | OK | OK | OK | - | - | OK | OK |
| 38400 | - | - | OK | - | OK | - | OK | OK | OK |
| 57600 | - | - | - | - | OK | - | - | - | OK |
| 115200 | - | - | - | - | - | - | - | - | OK |
| Frequency (MHz) Baudrate (kHz) | 8 | 10 | 11.0592 | 12 | 14.746 | 16 | 20 | 24 | 26.6 |
| 2400 | OK | OK | OK | OK | OK | OK | OK | OK | OK |
| 4800 | OK | OK | OK | OK | OK | OK | OK | OK | OK |
| 9600 | OK | OK | OK | OK | OK | OK | OK | OK | OK |
| 19200 | OK | OK | OK | OK | OK | OK | OK | OK | OK |
| 38400 | - | - | OK | OK | OK | OK | OK | OK | OK |
| 57600 | - | - | OK | - | OK | OK | OK | OK | OK |
| 115200 | - | - | OK | - | OK | - | - | - | - |

Command Data Stream Protocol

All commands are sent using the same flow. Each frame sent by the host is echoed by the bootloader.

Figure 5. Command Flow



Programming the Flash or EEPROM data

The flow described below shows how to program data in the Flash memory or in the EEPROM data memory.

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

The host must take care that:

- The data to program transmitted within a frame are in the same page.

Requests from Host

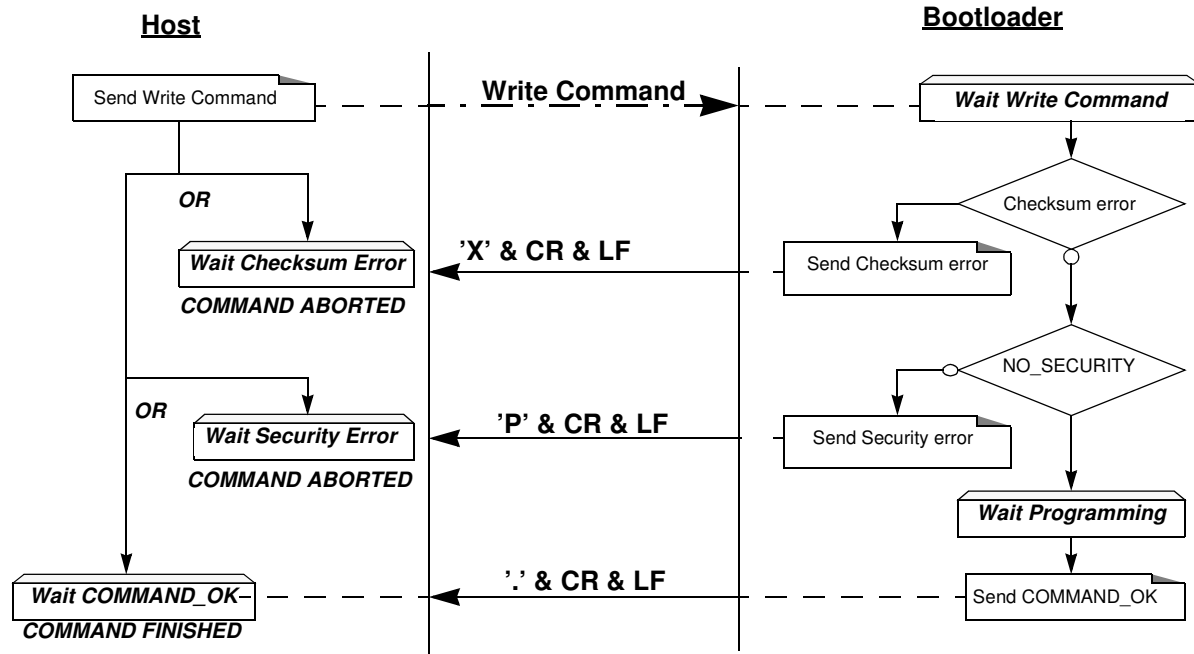
| Command Name | Record Type | Load Offset | Record Length | Data[0] | ... | Data[127] |
|---------------------|-------------|---------------|---------------|---------|-----|-----------|
| Program Flash | 00h | start address | nb of Data | x | ... | x |
| Program EEPROM Data | 07h | start address | nb of Data | x | ... | x |

Answers from Bootloader

The bootloader answers with:

- '.' & 'CR' & 'LF' when the data are programmed
- 'X' & 'CR' & 'LF' if the checksum is wrong
- 'P' & 'CR' & 'LF' if the Security is set

Flow Description



Example

Programming Data (write 55h at address 0010h in the Flash)

```

HOST          : 01 0010 00 55 9A
BOOT LOADER   : 01 0010 00 55 9A . CR LF
    
```

Read the Flash or EEPROM Data

The flow described below allows the user to read data in the Flash memory or in the EEPROM data memory. A blank check command is possible with this flow.

The device splits into blocks of 16 bytes the data to transfer to the Host if the number of data to display is greater than 16 data bytes.

Requests from Host

| Command Name | Record Type | Load Offset | Record Length | Data[0] | Data[1] | Data[2] | Data[3] | Data[4] |
|----------------------|-------------|-------------|---------------|---------------|---------|---------|---------|---------|
| Read Flash | 04h | x | 05h | start address | | | | 00h |
| Blank check on Flash | | | | | | | | 01h |
| Read EEPROM Data | | | | | | | | 02h |

Note: The field 'Load offset' is not used.

Answers from Bootloader

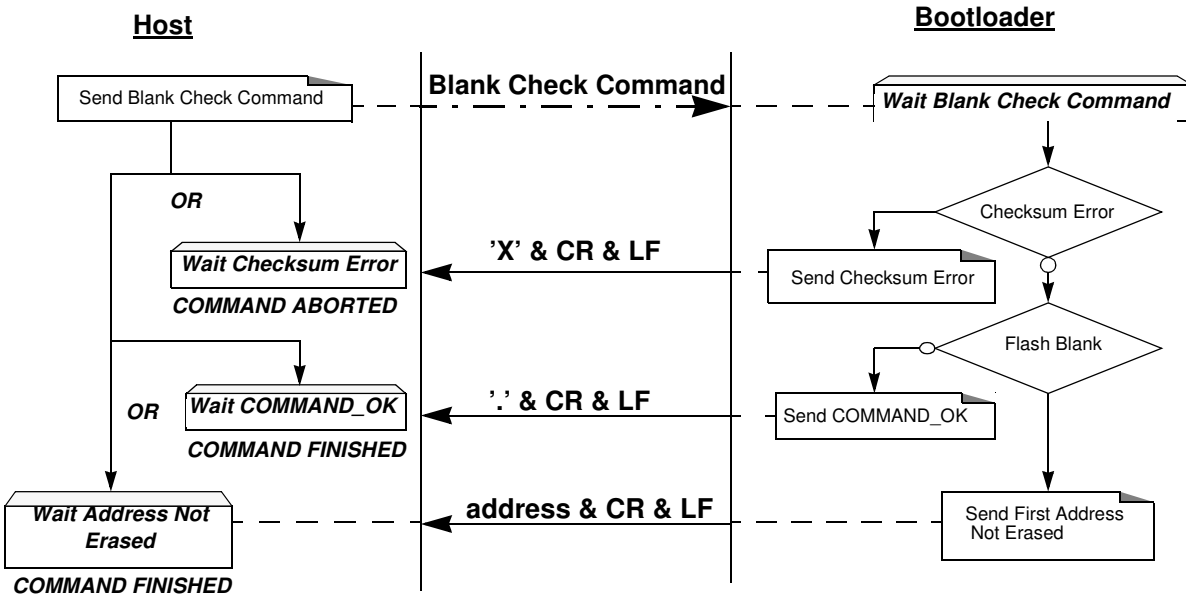
The bootloader answers to a read Flash or EEPROM Data memory command:

- 'Address = data' & 'CR' & 'LF' up to 16 data by line.
- 'X' & 'CR' & 'LF' if the checksum is wrong
- 'L' & 'CR' & 'LF' if the Security is set

The bootloader answers to blank check command:

- '.' & 'CR' & 'LF' when the blank check is ok
- 'First Address wrong' & 'CR' & 'LF' when the blank check is fail
- 'X' & 'CR' & 'LF' if the checksum is wrong
- 'P' & 'CR' & 'LF' if the Security is set

Flow Description: Blank Check Command



Example

Blank Check ok

```

HOST      : 05 0000 04 0000 7FFF 01 78
BOOT LOADER : 05 0000 04 0000 7FFF 01 78 . CR LF
    
```

Blank Check ko at address xxxx

```

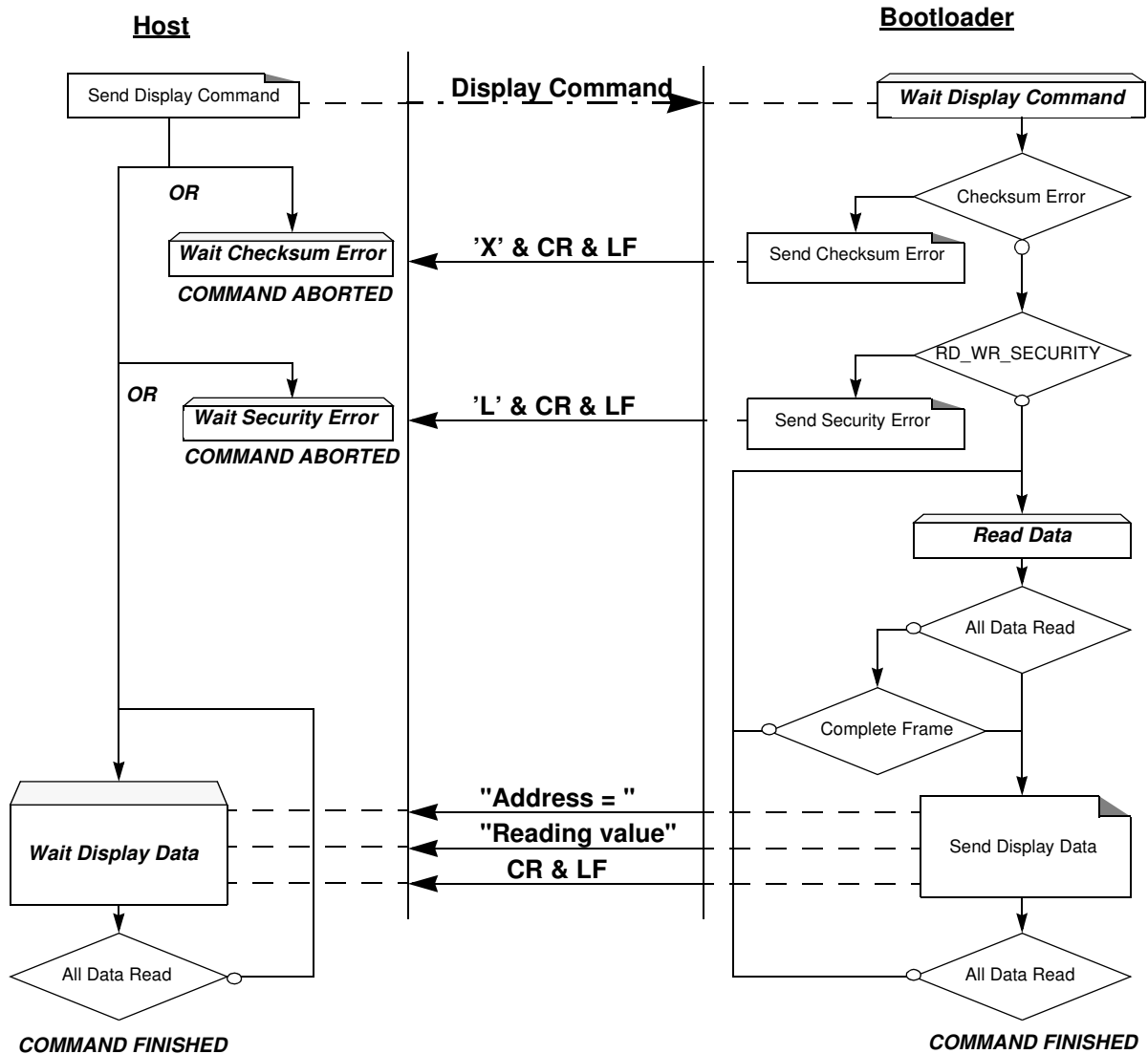
HOST      : 05 0000 04 0000 7FFF 01 78
BOOT LOADER : 05 0000 04 0000 7FFF 01 78 xxxx CR LF
    
```

Blank Check with checksum error

```

HOST      : 05 0000 04 0000 7FFF 01 70
BOOT LOADER : 05 0000 04 0000 7FFF 01 70 X CR LF CR LF
    
```

Flow Description: Read Command



Example

Display data from address 0000h to 0020h

```

HOST          : 05 0000 04 0000 0020 00 D7
BOOT LOADER   : 05 0000 04 0000 0020 00 D7
BOOT LOADER   0000=-----data----- CR LF   (16 data)
BOOT LOADER   0010=-----data----- CR LF   (16 data)
BOOT LOADER   0020=data CR LF                   ( 1 data)
  
```

Program Configuration Information

The flow described below allows the user to program Configuration Information regarding the bootloader functionality.

The Boot Process Configuration:

- BSB
- SBV
- P1_CF, P3_CF, P4_CF
- Fuse bits (BLJB and X2 bits) (see Section “Mapping and Default Value of Hardware Security Byte”)
- SSB
- EB

Requests from Host

| Command Name | Record Type | Load Offset | Record Length | Data[0] | Data[1] | Data[2] | |
|--------------------|-------------|-------------|---------------|---------|---------|---------|-------|
| Erase SBV & BSB | 03h | x | 02h | 04h | 00h | - | |
| Program SSB level1 | | | 02h | 05h | 00h | 00h | - |
| Program SSB level2 | | | | | | 01h | - |
| Program BSB | | | 03h | 06h | 06h | 00h | value |
| Program SBV | | | | | | 01h | |
| Program P1_CF | | | | | | 02h | |
| Program P3_CF | | | | | | 03h | |
| Program P4_CF | | | | | | 04h | |
| Program EB | | | | | | 06h | |
| Program bit BLJB | | | | | | 03h | |
| Program bit X2 | | | 08h | | | | |

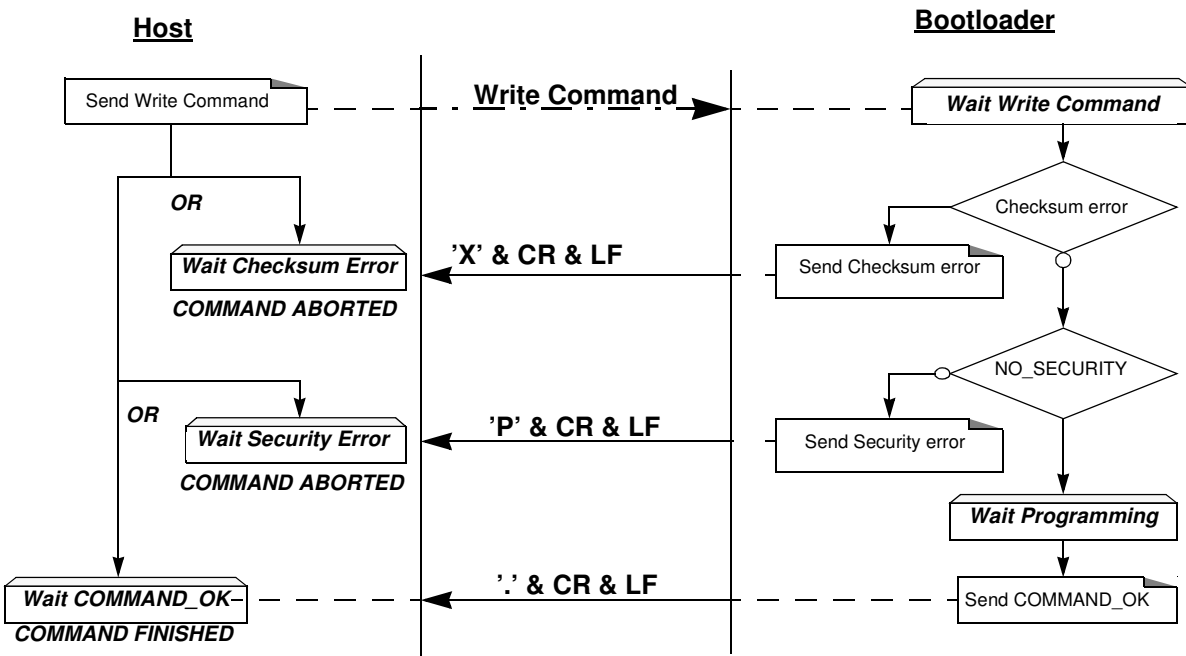
- Note:
1. The field 'Load Offset' is not used.
 2. To program the BLJB and X2 bit the 'bit value' is 00h or 01h.

Answers from Bootloader

The bootloader answers with:

- '.' & 'CR' & 'LF' when the value is programmed
- 'X' & 'CR' & 'LF' if the checksum is wrong
- 'P' & 'CR' & 'LF' if the Security is set

Flow Description



Example

Programming Atmel function (write SSB to level 2)

```

HOST          : 02 0000 03 05 01 F5
BOOT LOADER   : 02 0000 03 05 01 F5. CR LF
    
```

Writing Frame (write BSB to 55h)

```

HOST          : 03 0000 03 06 00 55 9F
BOOT LOADER   : 03 0000 03 06 00 55 9F . CR LF
    
```

Read Configuration Information or Manufacturer Information

The flow described below allows the user to read the configuration or manufacturer information.

Requests from Host

| Command Name | Record Type | Load Offset | Record Length | Data[0] | Data[1] |
|-------------------------|-------------|-------------|---------------|---------|---------|
| Read Manufacturer Code | 05h | x | 02h | 00h | 00h |
| Read Family Code | | | | | 01h |
| Read Product Name | | | | | 02h |
| Read Product Revision | | | | | 03h |
| Read SSB | | | | 07h | 00h |
| Read BSB | | | | | 01h |
| Read SBV | | | | | 02h |
| Read P1_CF | | | | | 03h |
| Read P3_CF | | | | | 04h |
| Read P4_CF | | | | | 05h |
| Read EB | | | | 06h | |
| Read HSB (Fuse bit) | | | | 0Bh | 00h |
| Read Device ID1 | | | | 0Eh | 00h |
| Read Device ID2 | | | | | 01h |
| Read Bootloader version | | | | 0Fh | 00h |

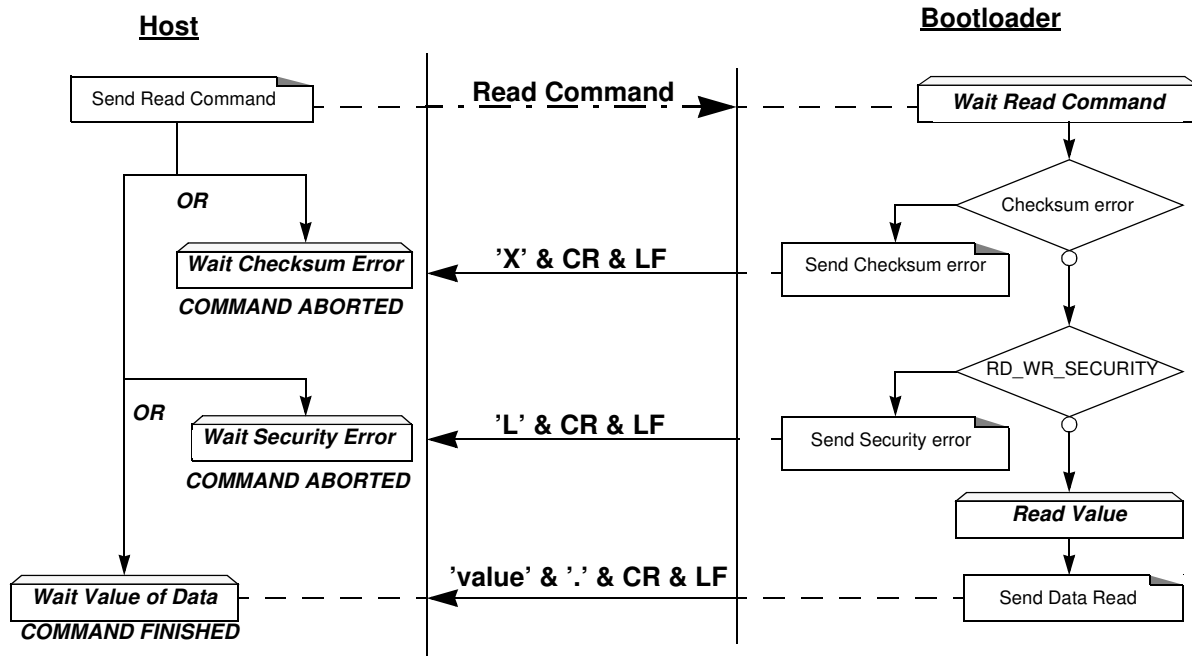
Note: The field 'Load Offset' is not used

Answers from Bootloader

The bootloader answers with:

- 'value' & '.' & 'CR' & 'LF' when the value is programmed
- 'X' & 'CR' & 'LF' if the checksum is wrong
- 'P' & 'CR' & 'LF' if the Security is set

Flow Description



Example

Read function (read SBV)

```

HOST          : 02 0000 05 07 02 F0
BOOT LOADER   : 02 0000 05 07 02 F0 Value . CR LF
    
```

Atmel Read function (read Bootloader version)

```

HOST          : 02 0000 01 02 00 FB
BOOT LOADER   : 02 0000 01 02 00 FB Value . CR LF
    
```

Erase the Flash

The flow described below allows the user to erase the Flash memory.

Two modes of Flash erasing are possible:

- Full-chip erase
- Block erase

The Full Chip erase command erases the whole Flash (16 Kbytes) and sets some Configuration Bytes to their default values:

- BSB = FFh
- SBV = FCh
- SSB = FFh (NO_SECURITY)

The full chip erase is always executed whatever the Software Security Byte value is.

Note: Take care that the full chip erase execution takes few seconds (128 pages)

The Block erase command erases only a part of the Flash.

Two Blocks are defined in the T89C5115:

- block0 (from 0000h to 1FFFh)
- block1 (from 2000h to 3FFFh)

Requests from Host

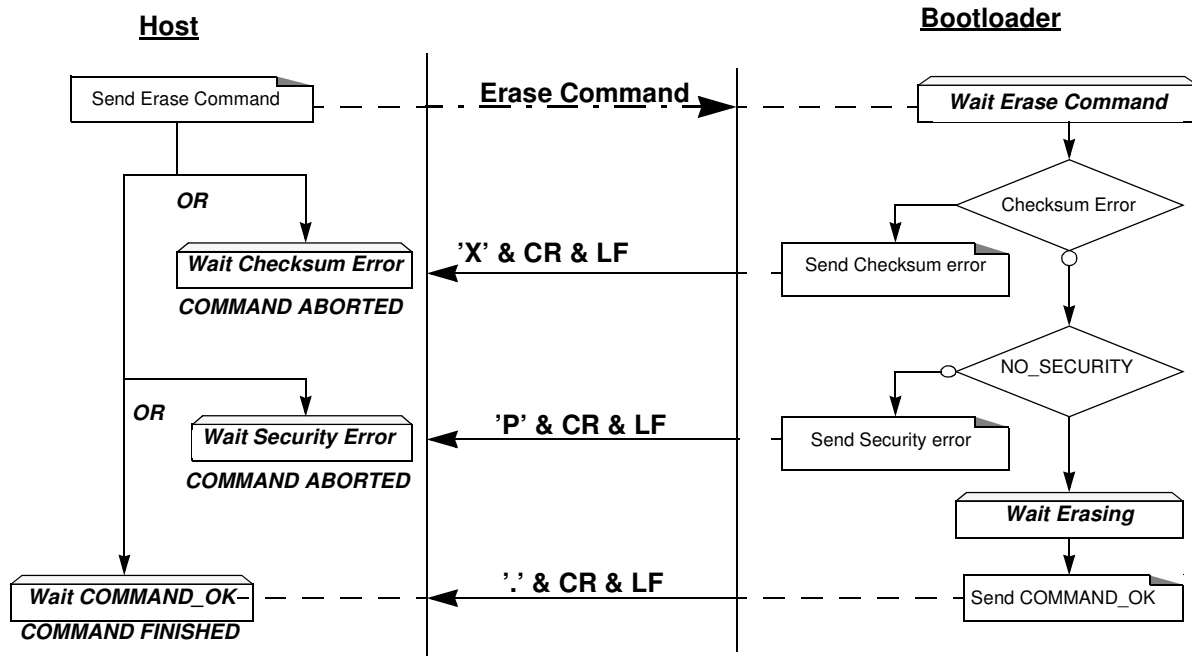
| Command Name | Record Type | Load Offset | Record Length | Data[0] | Data[1] |
|--------------------------|-------------|-------------|---------------|---------|---------|
| Erase block0 (0k to 8k) | 03h | x | 02h | 01h | 00h |
| Erase block1 (8k to 16k) | | | | | 20h |
| Full chip erase | | | 01h | 07h | - |

Answers from Bootloader

As the Program Configuration Information flows, the erase block command has three possible answers:

- '.' & 'CR' & 'LF' when the data are programmed
- 'X' & 'CR' & 'LF' if the checksum is wrong
- 'P' & 'CR' & 'LF' if the Security is set

Flow Description



Example

Full Chip Erase

```

HOST          : 01 0000 03 07 F5
BOOT LOADER   : 01 0000 03 07 F5 . CR LF
  
```

Erase Block1(8k to 16k)

```

HOST          : 02 0000 03 01 20 DA
BOOT LOADER   : 02 0000 03 01 20 DA . CR LF
  
```

Start the Application

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

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.

Requests from Host

| Command Name | Record Type | Load Offset | Record Length | Data[0] | Data[1] | Data[2] | Data[3] |
|---|-------------|-------------|---------------|---------|---------|---------|---------|
| Start application with a reset pulse generation | 03h | x | 02h | 03h | 00h | - | - |
| Start application with a jump at 'address' | | | 04h | | 01h | Address | |

Answer from Bootloader No answer is returned by the device.

Example

Start Application with reset pulse

```
HOST          : 02 0000 03 03 00 F8
BOOT LOADER   : 02 0000 03 03 00 F8
```

Start Application without reset at address 0000h

```
HOST          : 04 0000 03 03 01 00 00 F5
BOOT LOADER   : 04 0000 03 03 01 00 00 F5
```

In-Application Programming/Self-programming

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

The user application can call some 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 T89C5115

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

The Flash_api routines are listed in APPENDIX-2.

API Call

Process

The application selects an API by setting R1, ACC, DPTR0 and DPTR1 registers.

All calls are made through a common interface 'USER_CALL' at the address FFF0h.

The jump at the USER_CALL must be done by LCALL instruction to be able to comeback into 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 T89C5115 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.

To read the Flash memory the bootloader is not involved.

For more details on these routines see the T89C5115 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 load the column latches of the Flash then call the `__api_wr_code_byte` or `__api_wr_code_page` see data sheet in section 'Program/Code Memory'.
- Parameter Settings

| API_name | R1 | DPTR0 | DPTR1 | Acc |
|---------------------------------|-----|--|--|---------------------------|
| <code>__api_wr_code_byte</code> | 02h | Address in Flash memory to write | - | Value to write |
| <code>__api_wr_code_page</code> | 09h | Address of the first Byte to program in the Flash memory | Address in XRAM of the first data to program | Number of Byte to program |

- instruction: LCALL FFF0h.

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

Read Configuration and Manufacturer Information

- Parameter Settings

| API_name | R1 | DPTR0 | DPTR1 | Acc |
|--|-----|-------|-------|------------------------|
| <code>__api_rd_HSB</code> | 0Bh | 0000h | x | return HSB |
| <code>__api_rd_BSB</code> | 07h | 0001h | x | return BSB |
| <code>__api_rd_SBV</code> | 07h | 0002h | x | return SBV |
| <code>__api_rd_SSB</code> | 07h | 0000h | x | return SSB |
| <code>__api_rd_EB</code> | 07h | 0006h | x | return EB |
| <code>__api_rd_manufacturer</code> | 00h | 0000h | x | return manufacturer id |
| <code>__api_rd_device_id1</code> | 00h | 0001h | x | return id1 |
| <code>__api_rd_device_id2</code> | 00h | 0002h | x | return id2 |
| <code>__api_rd_device_id3</code> | 00h | 0003h | x | return id3 |
| <code>__api_rd_bootloader_version</code> | 0Fh | 0000h | x | return value |

- Instruction: LCALL FFF0h.
- 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

Program Configuration Information

- Parameter Settings

| API Name | R1 | DPTR0 | DPTR1 | Acc |
|----------------------------------|-----|-------|-------|----------------|
| <code>__api_set_X2</code> | 0Ah | 0008h | x | 00h |
| <code>__api_clr_X2</code> | 0Ah | 0008h | x | 01h |
| <code>__api_set_BLJB</code> | 0Ah | 0004h | x | 00h |
| <code>__api_clr_BLJB</code> | 0Ah | 0004h | x | 01h |
| <code>__api_wr_BSB</code> | 06h | 0000h | x | value to write |
| <code>__api_wr_SBV</code> | 06h | 0001h | x | value to write |
| <code>__api_wr_EB</code> | 06h | 0006h | x | value to write |
| <code>__api_wr_SSB_LEVEL0</code> | 05h | FFh | x | x |
| <code>__api_wr_SSB_LEVEL1</code> | 05h | FEh | x | x |

| API Name | R1 | DPTR0 | DPTR1 | Acc |
|---------------------|-----|-------|-------|-----|
| __api_wr_SSB_LEVEL2 | 05h | FCh | x | x |

- Instruction: LCALL FFF0h.

Note: 1. See in the T89C5115 data sheet the time that a write operation takes.
2. No special resources are used by the bootloader during these operations

Erase Flash

The T89C5115 Flash memory is divided in to two blocks:

Block 0: from address 0000h to 1FFFh

Block 1: from address 2000h to 3FFFh

These two blocks contain 64 pages.

- Parameter Settings

| API_name | R1 | DPTR0 | DPTR1 | Acc |
|--------------------|-----|-------|-------|-----|
| __api_erase_block0 | 01h | 0000h | x | x |
| __api_erase_block1 | | 2000h | x | x |

- Instruction: LCALL FFF0h.

Note: 1. See the T89C5115 data sheet for the time that a write operation takes, this time must be multiplied by the number of pages.
2. No special resources are used by the bootloader during these operations

Start Bootloader

This routine allows to start at the beginning of the bootloader as 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

Appendix-A

Table 3. Summary of Frames From Host

| Command | Record Type | Record Length | Offset | Data[0] | Data[1] | Data[2] | Data[3] | Data[4] |
|---|-------------|------------------------|---------------|---------|---------------|-----------|-------------|---------|
| Program Nb Data Byte in Flash. | 00h | nb of data (up to 80h) | start address | x | x | x | x | x |
| Erase block0 (0000h-1FFFh) | 03h | 02h | x | 01h | 00h | - | - | - |
| Erase block1 (2000h-3FFFh) | | 02h | x | 01h | 20h | - | - | - |
| Start application with a reset pulse generation | | 02h | x | 03h | 00h | - | - | - |
| Start application with a jump at 'address' | | 04h | x | | 01h | address | | - |
| Erase SBV & BSB | | 02h | x | 04h | 00h | - | - | - |
| Program SSB level 1 | | | x | 05h | 00h | - | - | - |
| Program SSB level 2 | | | x | | 01h | - | - | - |
| Program BSB | | 03h | x | 06h | 00h | value | - | - |
| Program SBV | | | x | | 01h | value | - | - |
| Program P1_CF | | | x | | 02h | value | - | - |
| Program P3_CF | | | x | | 03h | value | - | - |
| Program P4_CF | | | x | | 04h | value | - | - |
| Program EB | | | x | | 06h | value | - | - |
| Full Chip Erase | | 01h | x | 07h | - | - | - | - |
| Program bit BLJB | | 03h | x | 0Ah | 04h | bit value | - | - |
| Program bit X2 | | | x | | 08h | bit value | - | - |
| Read Flash | | 04h | 05h | x | Start Address | | End Address | |
| Blank Check | 01h | | | | | | | |
| Read EEPROM Data | 02h | | | | | | | |

Table 3. Summary of Frames From Host (Continued)

| Command | Record Type | Record Length | Offset | Data[0] | Data[1] | Data[2] | Data[3] | Data[4] |
|--------------------------------|-------------|---------------------------|------------------|---------|---------|---------|---------|---------|
| Read Manufacturer Code | 05h | 02h | x | 00h | 00h | - | - | - |
| Read Family Code | | | | | 01h | - | - | - |
| Read Product Name | | | | | 02h | - | - | - |
| Read Product Revision | | | | | 03h | - | - | - |
| Read SSB | | | | 07h | 00h | - | - | - |
| Read BSB | | | | | 01h | - | - | - |
| Read SBV | | | | | 02h | - | - | - |
| Read P1_CF | | | | | 03h | - | - | - |
| Read P3_CF | | | | | 04h | - | - | - |
| Read P4_CF | | | | | 05h | - | - | - |
| Read EB | | | | 06h | - | - | - | |
| Read Hardware Byte | | | | 0Bh | 00h | - | - | - |
| Read Device Boot ID1 | | | | 0Eh | 00h | - | - | - |
| Read Device Boot ID2 | | | | | 01h | - | - | - |
| Read Bootloader Version | | | | 0Fh | 00h | - | - | - |
| Program Nb Data byte in EEPROM | 00h | nb of data (up to 80h) | start address | x | x | x | x | x |

Appendix-B

Table 4. API Summary

| Function Name | Bootloader Execution | R1 | DPTR0 | DPTR1 | Acc |
|-----------------------------|----------------------|-----|--|--|---------------------------|
| __api_rd_code_byte | no | | | | |
| __api_wr_code_byte | yes | 02h | Address in Flash memory to write | - | Value to write |
| __api_wr_code_page | yes | 09h | Address of the first Byte to program in the Flash memory | Address in XRAM of the first data to program | Number of Byte to program |
| __api_erase_block0 | yes | 01h | 0000h | x | x |
| __api_erase_block1 | yes | 01h | 2000h | x | x |
| __api_rd_HSB | yes | 0Bh | 0000h | x | return value |
| __api_set_X2 | yes | 0Ah | 0008h | x | 00h |
| __api_clr_X2 | yes | 0Ah | 0008h | x | 01h |
| __api_set_BLJB | yes | 0Ah | 0004h | x | 00h |
| __api_clr_BLJB | yes | 0Ah | 0004h | x | 01h |
| __api_rd_BSB | yes | 07h | 0001h | x | return value |
| __api_wr_BSB | yes | 06h | 0000h | x | value |
| __api_rd_SBV | yes | 07h | 0002h | x | return value |
| __api_wr_SBV | yes | 06h | 0001h | x | value |
| __api_erase_SBV | yes | 06h | 0001h | x | FCh |
| __api_rd_SSB | yes | 07h | 0000h | x | return value |
| __api_wr_SSB_level0 | yes | 05h | 00FFh | x | x |
| __api_wr_SSB_level1 | yes | 05h | 00FEh | x | x |
| __api_wr_SSB_level2 | yes | 05h | 00FCh | x | x |
| __api_rd_EB | yes | 07h | 0006h | x | return value |
| __api_wr_EB | yes | 06h | 0006h | x | value |
| __api_rd_manufacturer | yes | 00h | 0000h | x | return value |
| __api_rd_device_id1 | yes | 00h | 0001h | x | return value |
| __api_rd_device_id2 | yes | 00h | 0002h | x | return value |
| __api_rd_device_id3 | yes | 00h | 0003h | x | return value |
| __api_rd_bootloader_version | yes | 0Fh | 0000h | x | return value |
| __api_eeeprom_busy | no | | | | |
| __api_rd_eeeprom_byte | no | | | | |
| __api_wr_eeeprom_byte | no | | | | |
| __api_start_bootloader | no | | | | |

Datasheet Change Log

**4225A - 03/03 to
4225B 12/03**

1. Added bit stop for the UART protocol.

**4225B 12/03 to
4225C 03/08**

1. Updated Bootloader version.



Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
Enter Product Line E-mail

Sales Contact
www.atmel.com/contacts

Literature Requests
www.atmel.com/literature

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN ATMEL'S TERMS AND CONDITIONS OF SALE LOCATED ON ATMEL'S WEB SITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© 2008 Atmel Corporation. All rights reserved. Atmel®, logo and combinations thereof, and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.