Features

- **Protocol**
  - **UART Used as a Physical Layer**
  - **Based on the Intel Hex-type Records**
  - **Autobaud**
- **In-System Programming**
  - **Read/Write Flash Memory**
  - **Read Device IDs**
  - **Block Erase**
  - **Full-chip Erase**
  - **Read/Write Configuration Bytes**
  - **Security Setting From ISP Command**
  - **Remote Application Start Command**
- **In-Application Programming/Self-Programming**
  - **Read/Write Flash Memory**
  - **Read Device IDs**
  - **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 for the AT89C51SND1 product can be found in the AT89C51SND1 data sheet and the AT89C51SND1 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
Revision 1.0.0	New release increasing programming speed	June 2002
Revision 1.1.0	Bug fix in boot process	October 2002

MP3 Microcontrollers

AT89C51SND1 UART Bootloader

4241B–MP3–07/04

Functional Description

The AT89C51SND1 bootloader facilitates In-System Programming and In-Application Programming.

In-System Programming Capability

In-System Programming (ISP) 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 a 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

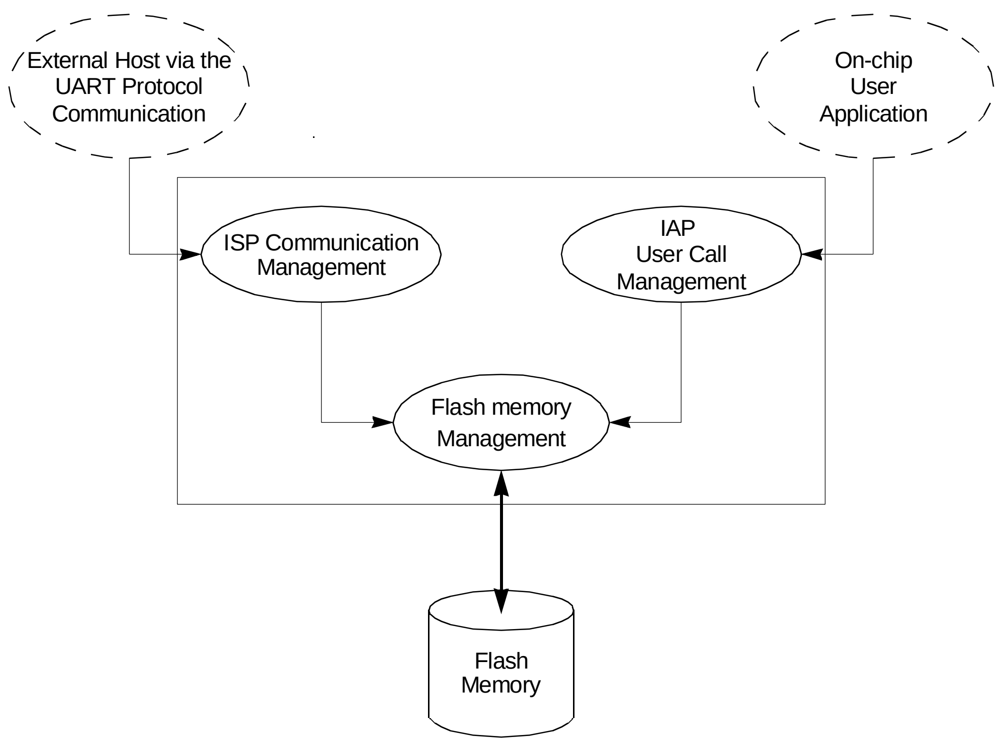
In-Application Programming (IAP) allows the reprogramming of a microcontroller's on-chip Flash memory without removing it from the system and while the embedded application is running.

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, etc.).

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 table below lists configuration and manufacturer byte information used by the bootloader. This information can be accessed through a set of API or ISP commands.

Table 1. Configuration and Munfacturer Byte Information

Mnemonic	Description	Default Value
BSB	Boot Status Byte	FFh
SBV	Software Boot Vector	F0h
SSB	Software Security Byte	FCh
Manufacturer		58h
ID1: Family code		D7h
ID2: Product Name		ECh
ID3: Product Revision		FFh

Mapping and Default Value of Hardware Security Byte

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

Table 2. Mapping and Default Value of HSB

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 F000h-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: Unprogrammed = 1, P: Program = 0

Software Security Byte

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)
  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)
  This is the default level.
  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.

Table 3. Software Security Byte Levels

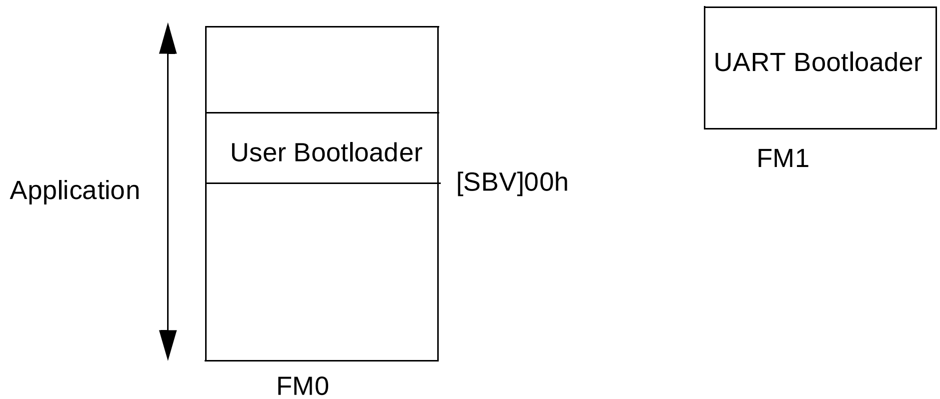
	Level 0	**Level 1**	**Level 2**
Flash	Any access allowed	Read only access allowed	All access not allowed
Fuse bit	Any access allowed	Read only access allowed	All access not allowed
BSB & SBV	Any access allowed	Read only access allowed	All access not allowed
SSB	Any access allowed	Write level2 allowed	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 "Regular Boot Process", page 7.

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 microcontrollers.

This free software program is available on 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 the higher level protocol over the serial line.

Bootloader Execution

As internal C51 code space is limited to 64K Bytes, some mechanisms are implemented to allow boot memory to be mapped in the code space for execution at addresses from F000h to FFFFh. The boot memory is enabled by setting the ENBOOT bit in AUXR1. The three ways to set this bit are detailed below.

Software Boot Mapping

The software way to set ENBOOT consists in writing to AUXR1 from the user's software. This enables bootloader or API routines execution.

Hardware Condition Boot Mapping

The hardware condition is based on the ISP# pin. When driving this pin to low level, the chip reset sets ENBOOT and forces the reset vector to F000h instead of 0000h in order to execute the bootloader software.

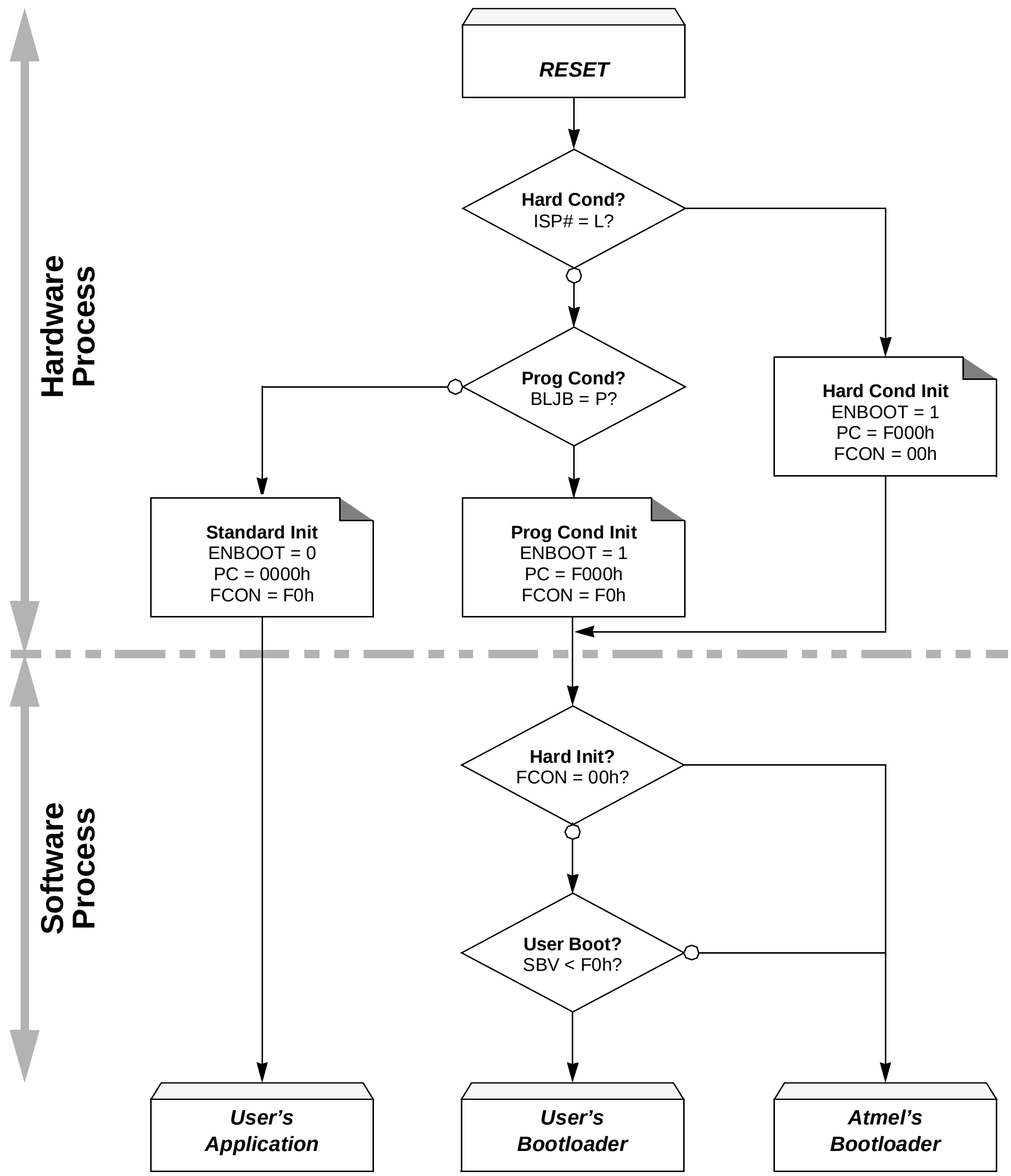
As shown in Figure 3, the hardware condition always allows In-System recovery when user's memory has been corrupted.

Programmed Condition Boot Mapping

The programmed condition is based on the Bootloader Jump Bit (BLJB) in HSB. As shown in Figure 3, this bit is programmed (by hardware or software programming mode), the chip reset set ENBOOT and forces the reset vector to F000h instead of 0000h, in order to execute the bootloader software.

Regular Boot Process

Figure 3. Boot Process Algorithm



Physical Layer

The UART used to transmit information has the following configuration:

- Character: 8-bit data
- Parity: none
- Stop: 1 bit
- Flow control: none
- Baud rate: auto baud 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 in Table 4.

Table 4. Intel Hex Type Frame

Record Mark ':'	Record length	Load Offset	Record Type	Data or Info	Checksum
1 byte	1 byte	2 bytes	1 byte	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.
- 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 include the Record Length field to the last Byte of the Data/Info field inclusive. Therefore, the sum of all the ASCII pairs in a record after converting to binary, from the Record Length field to and the Checksum field inclusive, is zero.

Protocol

Overview

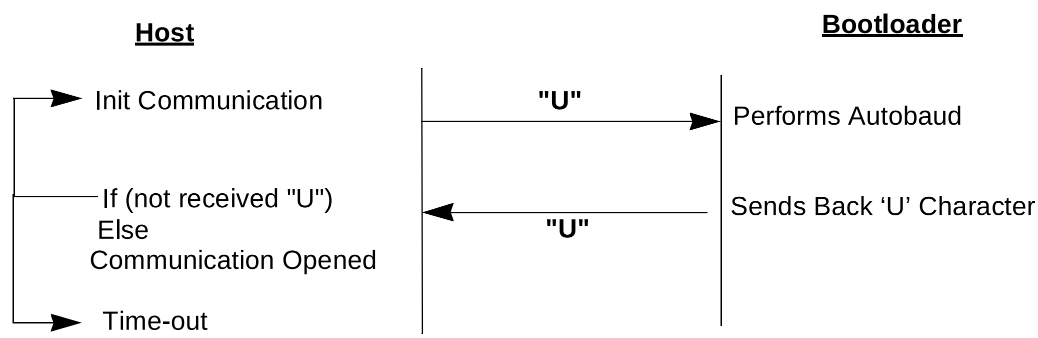
An initialization step must be performed after each Reset. After microcontroller reset, the bootloader waits for an auto baud sequence (see Section "Autobaud Performances", page 9).

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 baud rate (auto baud).

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 5 shows the auto baud capabilities.

Table 5. Autobaud Performances

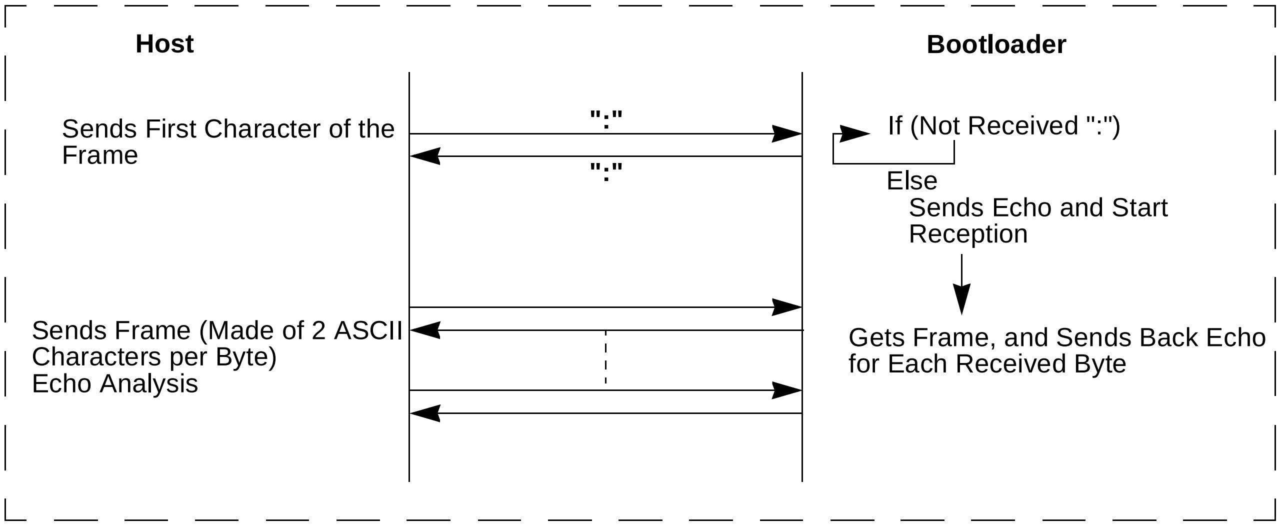
	F_{OSC} = 12 MHz		F_{OSC} = 16 MHz		F_{OSC} = 20 MHz	
Baudrate	**Status**	**Error%**	**Status**	**Error%**	**Status**	**Error%**
9600	OK	0.16	OK	0.16	OK	0.16
19200	OK	0.16	OK	0.16	OK	0.16
38400	OK/KO[1]	2.34	OK	0.16	OK	1.36
57600	OK	0.16	OK/KO[1]	2.12	OK	1.36
115200			OK/KO[1]	3.55	OK	1.36

Note: 1. Depending on the host, error values may lead to unsupported baudrate.

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 Data

The flow described in Figure 6 shows how to program data in the Flash 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

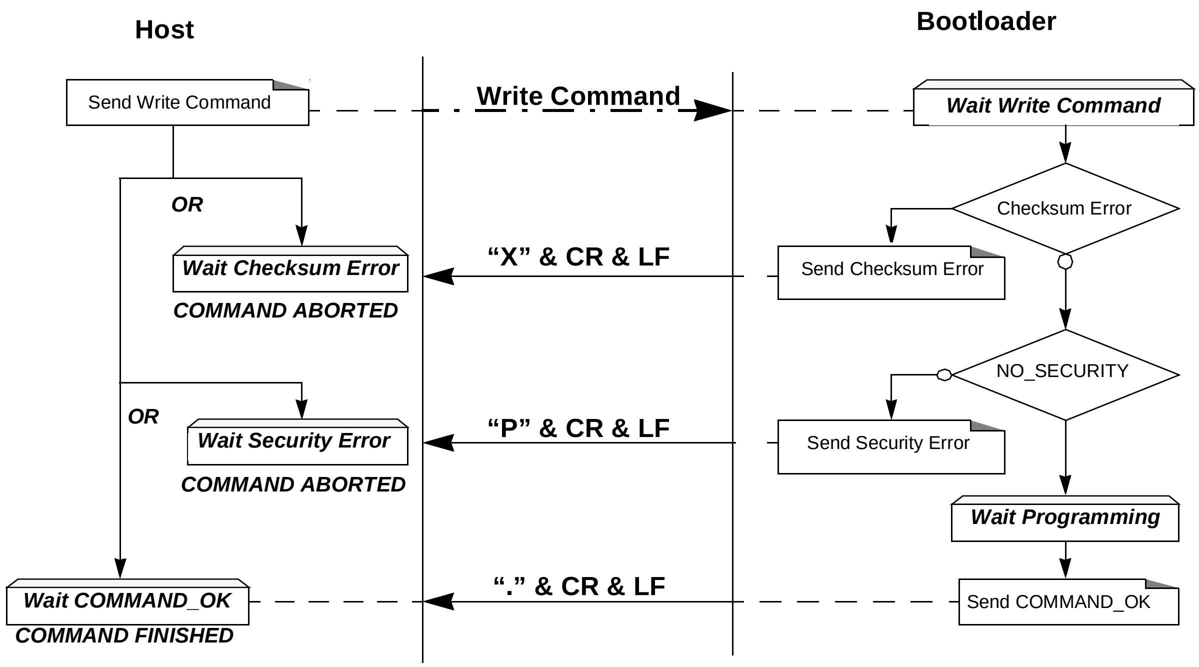
Answers from Bootloader

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

Figure 6. Programming Command



Programming Example

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

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

Reading the Flash Data

The flow described in Figure 7 allows the user to read data in the Flash 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		End Address		00h
Blank check on Flash								01h

Note: The field “Load offset” is not used.

Answers from Bootloader

The bootloader answers to a read Flash data memory command:

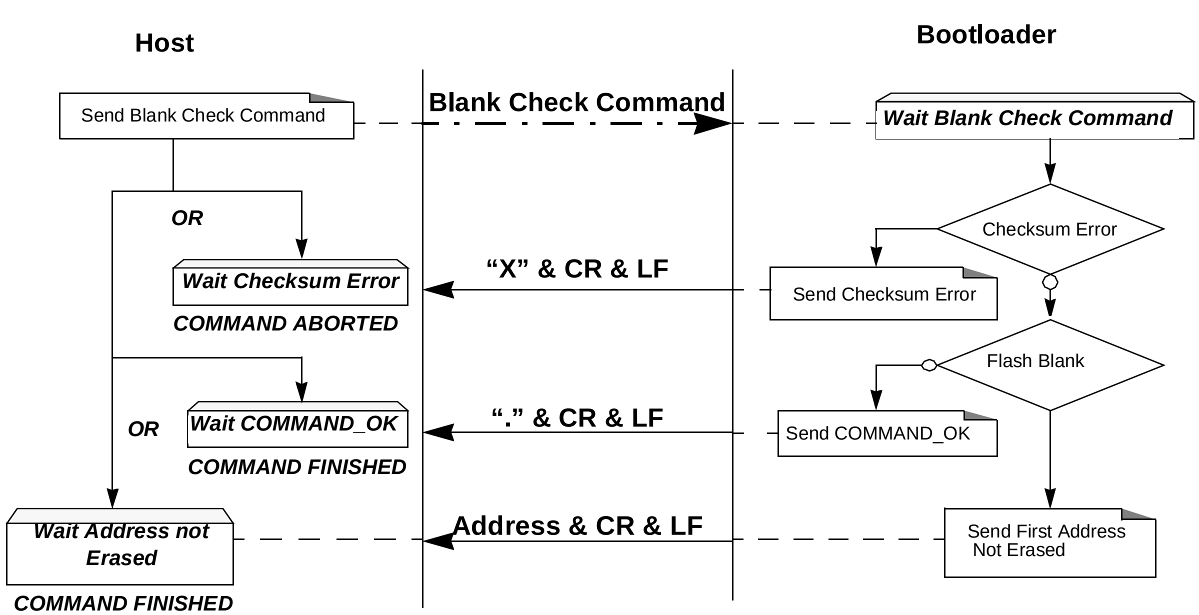
- "Address = data " & "CR" & "LF"
  up to 16 data by line.
- "X" & "CR" & "LF" if the checksum is wrong
- "L" & "CR" & "LF" if the Security is set

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

Figure 7. Blank Check Command



Blank Check Example

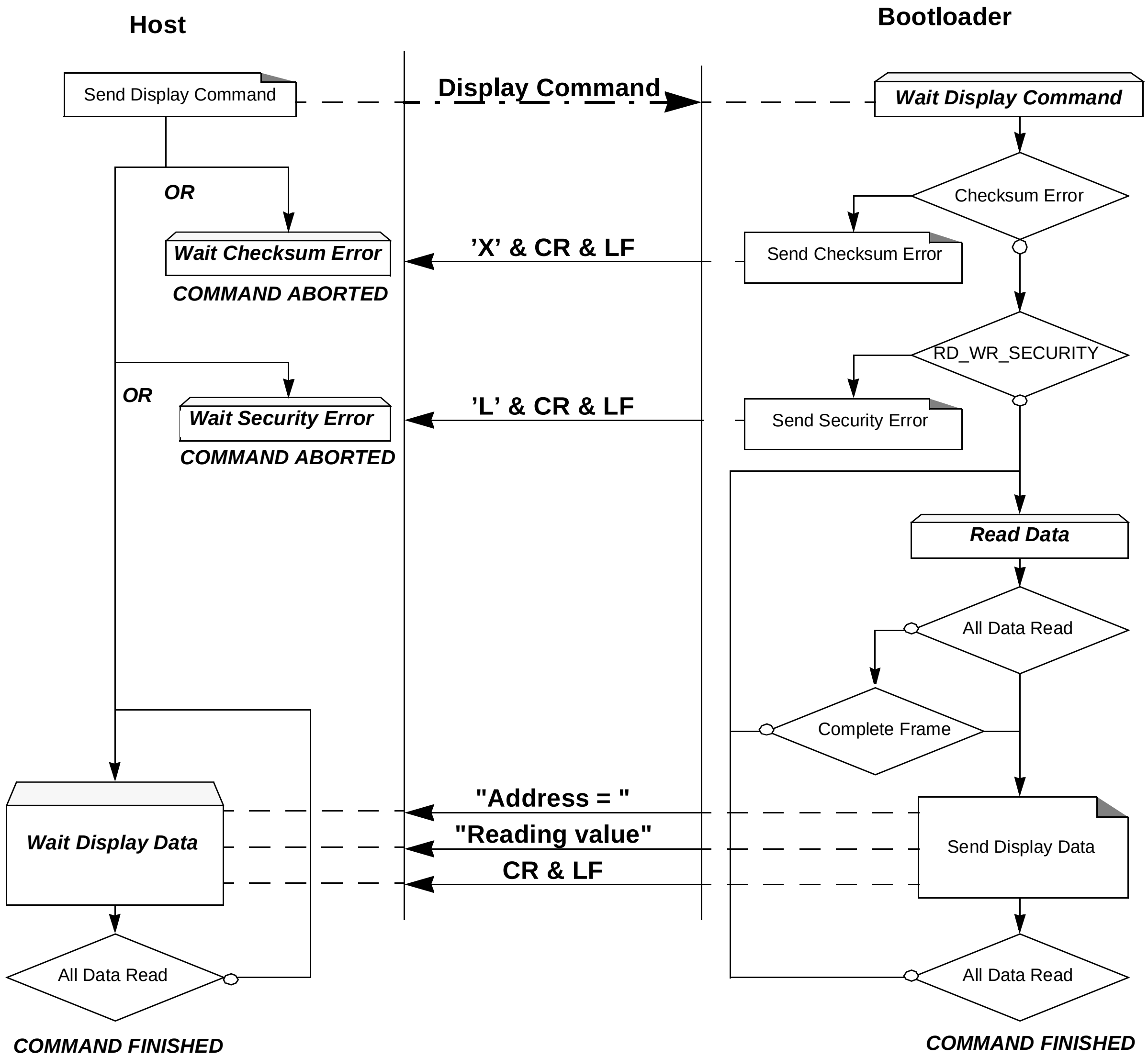
```
Blank Check ok
HOST          : 05 0000 04 0000 7FFF 01 78
BOOTLOADER    : 05 0000 04 0000 7FFF 01 78 . CR LF

Blank Check ok at address xxxx
HOST          : 05 0000 04 0000 7FFF 01 78
BOOTLOADER    : 05 0000 04 0000 7FFF 01 78 xxxx CR LF

Blank Check with checksum error
HOST          : 05 0000 04 0000 7FFF 01 70
BOOTLOADER    : 05 0000 04 0000 7FFF 01 70 X CR LF CR LF
```

Flow Description

Figure 8. Read Command Flow



Blank Check Example

```
Display data from address 0000h to 0020h
HOST          : 05 0000 04 0000 0020 00 D7
BOOTLOADER    : 05 0000 04 0000 0020 00 D7
BOOTLOADER    0000=-----data------ CR LF   (16 data)
BOOTLOADER    0010=-----data------ CR LF   (16 data)
BOOTLOADER    0020=data CR LF              (1 data)
```

Program Configuration Information

The flow described in Figure 9 allows the user to program Configuration Information regarding the bootloader functionality.

The Boot Process Configuration:

BSB
SBV
Fuse bits (BLJB and X2 bits) (see Section "Mapping and Default Value of Hardware Security Byte", page 3)
SSB

Requests from Host

<table>
<tr><th>Command Name</th><th>Record Type</th><th>Load Offset</th><th>Record Length</th><th>Data[0]</th><th>Data[1]</th><th>Data[2]</th></tr>
<tr><td>Erase SBV & BSB</td><td rowspan="7">03h</td><td rowspan="7">x</td><td>02h</td><td>04h</td><td>00h</td><td></td></tr>
<tr><td>Program SSB level1</td><td rowspan="2">02h</td><td rowspan="2">05h</td><td>00h</td><td></td></tr>
<tr><td>Program SSB level2</td><td>01h</td><td></td></tr>
<tr><td>Program BSB</td><td rowspan="2">03h</td><td rowspan="2">06h</td><td>00h</td><td rowspan="2">value</td></tr>
<tr><td>Program SBV</td><td>01h</td></tr>
<tr><td>Program bit BLJB</td><td rowspan="2">03h</td><td rowspan="2">0Ah</td><td>04h</td><td rowspan="2">bit value</td></tr>
<tr><td>Program bit X2</td><td>08h</td></tr>
</table>

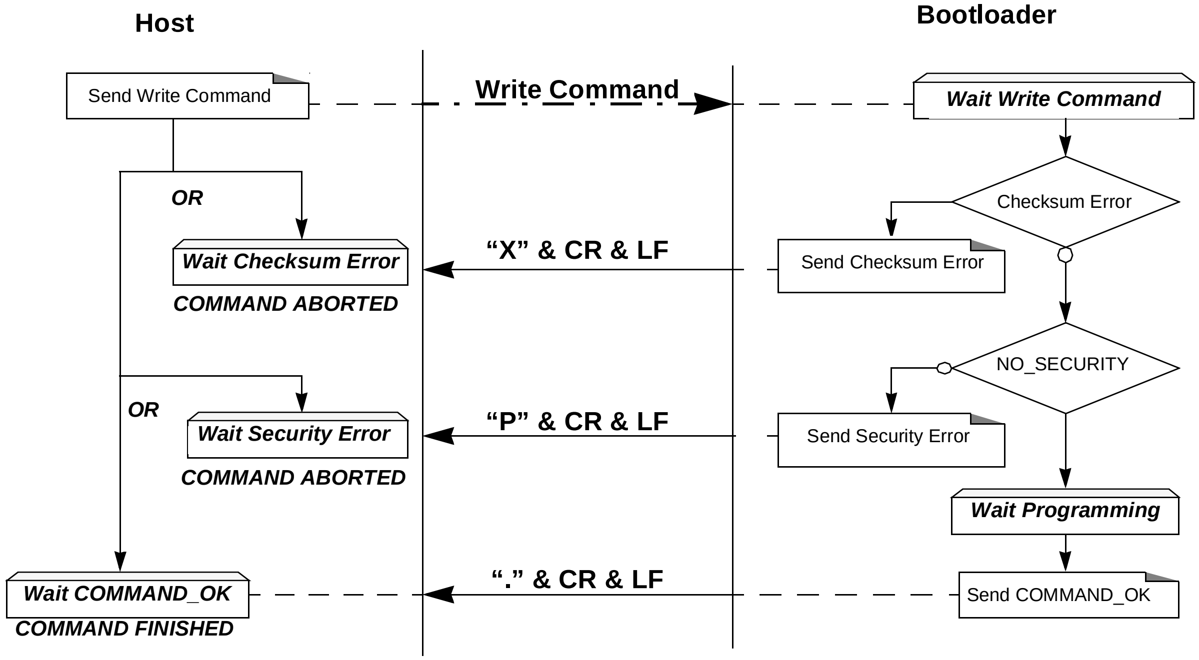
Note: 1. The field "Load Offset" is not used
2. To program the BLJB and X2 bit the "bit value" is 00h or 01h.

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

Figure 9. Write Command Flow



Program Configuration Example

<u>`Programming Atmel function (write SSB to level 2)`</u>

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

<u>`Writing Frame (write BSB to 55h)`</u>

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

Read Configuration Information or Manufacturer Information

The flow described in Figure 10 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					00h
Read Family Code				00h	01h
Read Product Name					02h
Read Product Revision					03h
Read SSB					00h
Read BSB	05h	x	02h	07h	01h
Read SBV					02h
Read HSB (Fuse bit)				0Bh	00h
Read Device ID1				0Eh	00h
Read Device ID2					01h
Read bootloader version				0Fh	00h

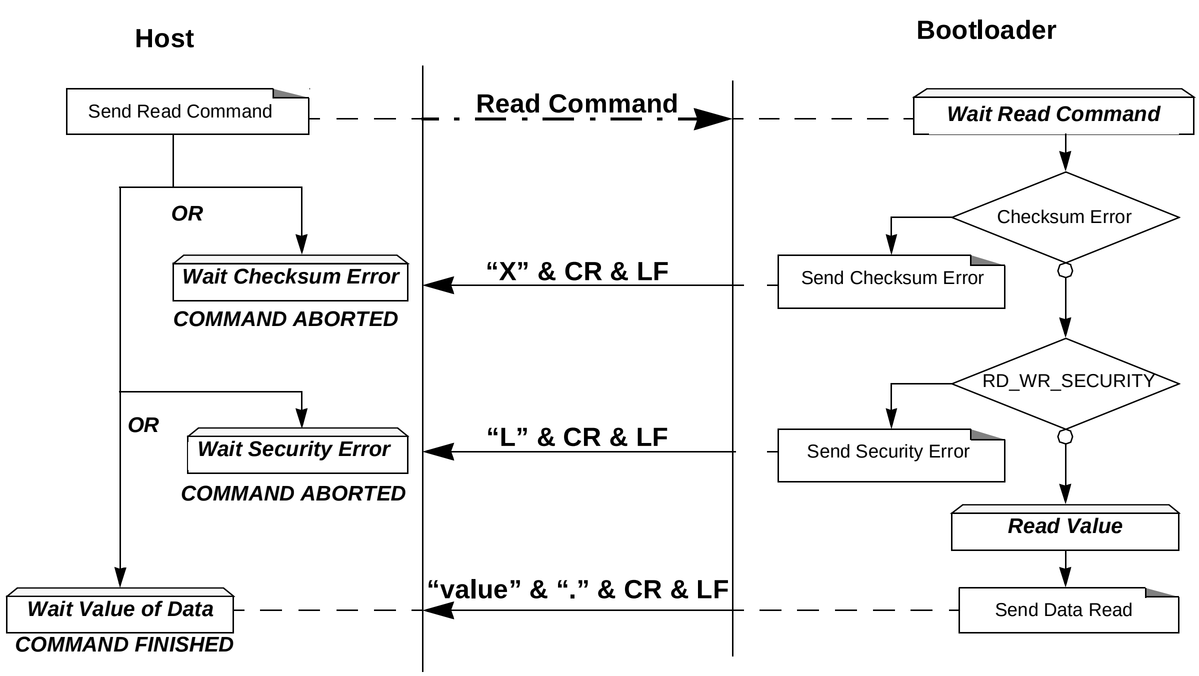
Note: The field “Load Offset” is not used.

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

Figure 10. Read Command



Read Example

```
Read function (read SBV)
HOST         : 02 0000 05 07 02 F0
BOOTLOADER   : 02 0000 05 07 02 F0 Value . CR LF
Atmel Read function (read bootloader version)
HOST         : 02 0000 01 02 00 FB
BOOTLOADER   : 02 0000 01 02 00 FB Value . CR LF
```

Erase the Flash

The flow described in Figure 11 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 and sets some Configuration Bytes at their default values:

- BSB = FFh
- SBV = F0h
- SSB = FFh (NO_SECURITY)

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

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

Four Blocks are defined in the AT89C51SND1:

- block0 (From 0000h to 1FFFh)
- block1 (From 2000h to 3FFFh)
- block2 (From 4000h to 7FFFh)
- block3 (From 8000h to FFFFh)

Requests from Host

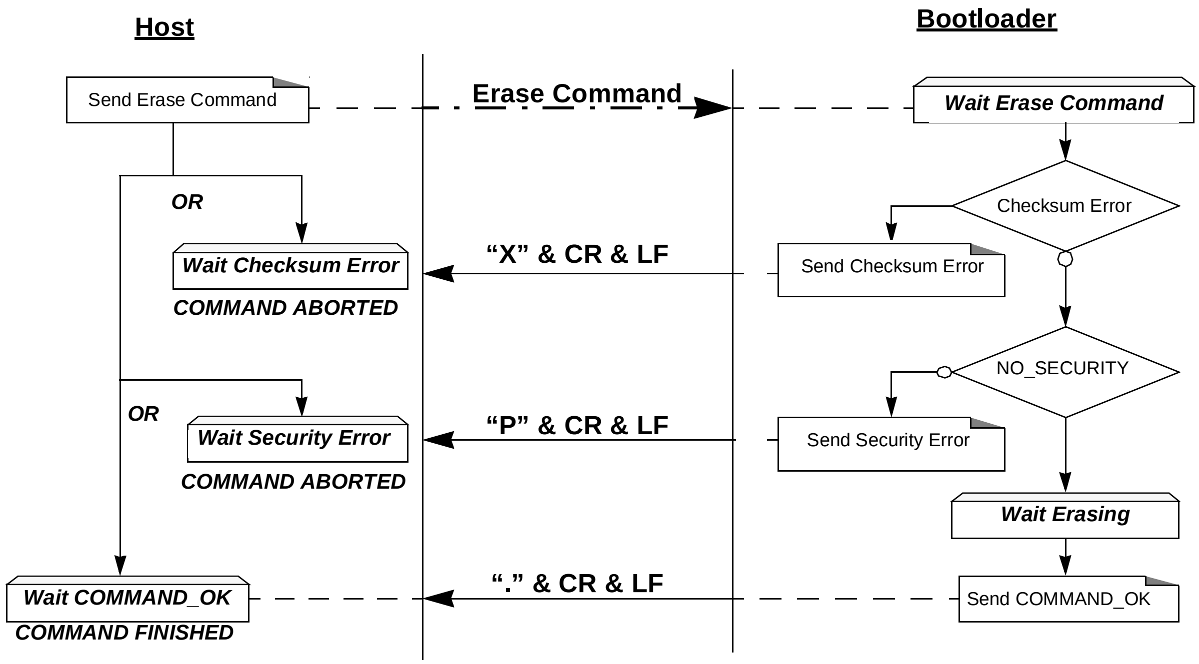
Command Name	Record Type	Load Offset	Record Length	Data[0]	Data[1]
Erase block0 (0k to 8k)	03h	x	02h	01h	00h
Erase block1 (8k to 16k)					20h
Erase block2 (16k to 32k)					40h
Erase block2 (32k to 64k)					80h
Full chip erase			01h	07h	-

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

Figure 11. Erase Command



Example

Full Chip Erase

HOST	: 01 0000 03 07 F5
BOOTLOADER	: 01 0000 03 07 F5 . CR LF

Erase Block1(8k to 16k)

HOST	: 02 0000 03 01 20 DA
BOOTLOADER	: 02 0000 03 01 20 DA . CR LF

Start the Application

The command described below allows to start the application directly from the boot-loader 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

<table>
<tr><th>Command Name</th><th>Record Type</th><th>Load Offset</th><th>Record Length</th><th>Data[0]</th><th>Data[1]</th><th>Data[2]</th><th>Data[3]</th></tr>
<tr><td>Start application with a reset pulse generation</td><td rowspan="2">03h</td><td rowspan="2">x</td><td>02h</td><td rowspan="2">03h</td><td>00h</td><td></td><td></td></tr>
<tr><td>Start application with a jump at “address”</td><td>04h</td><td>01h</td><td colspan="2">Address</td></tr>
</table>

Answer from Bootloader

No answer is returned by the device.

Start Application Example

```
Start Application with reset pulse

HOST          : 02 0000 03 03 00 F8
BOOTLOADER    : 02 0000 03 03 00 F8

Start Application without reset at address 0000h

HOST          : 04 0000 03 03 01 00 00 F5
BOOTLOADER    : 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 AT89C51SND1.

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

The flash_api routines are listed in APPENDIX B.

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 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.

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

For more information regarding the Flash writing time refer to the AT89C51SND1 datasheet.

API Commands

Several types of APIs are available:

- Read/Program Flash Data memory
- Read Configuration and Manufacturer Information
- Program Configuration Information
- Erase Flash
- Start bootloader

Read/Program Flash Memory

All routines to access Flash 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 AT89C51SND1 Datasheet sections "Program/Code 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

API_name	R1	DPTR0	DPTR1	ACC
__api_wr_code_byte	02h	Address in Flash memory to write		Value to write
__api_wr_code_page	09h	Address of the first Byte to program in the Flash memory	Address in XRAM of the first data to program	Number of Bytes 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
__api_rd_HSB	0Bh	0000h	x	return HSB
__api_rd_BSB	07h	0001h	x	return BSB
__api_rd_SBV	07h	0002h	x	return SBV
__api_rd_SSB	07h	0000h	x	return SSB
__api_rd_manufacturer	00h	0000h	x	return manufacturer id
__api_rd_device_id1	00h	0001h	x	return id1
__api_rd_device_id2	00h	0002h	x	return id2
__api_rd_device_id3	00h	0003h	x	return id3
__api_rd_bootloader_version	0Fh	0000h	x	return version 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
__api_set_X2	0Ah	0008h	x	00h
__api_clr_X2	0Ah	0008h	x	01h
__api_set_BLJB	0Ah	0004h	x	00h
__api_clr_BLJB	0Ah	0004h	x	01h
__api_wr_BSB	06h	0000h	x	value to write
__api_wr_SBV	06h	0001h	x	value to write
__api_wr_SSB_LEVEL0	05h	FFh	x	x

- Parameter Settings (Continued)

API_name	R1	DPTR0	DPTR1	ACC
__api_wr_SSB_LEVEL1	05h	FEh	x	x
__api_wr_SSB_LEVEL2	05h	FCh	x	x

- Instruction: LCALL FFF0h.

Note: 1. Refer to the AT89C51SND1 datasheet for information on write operation timing.
2. No special resources are used by the bootloader during these operations.

Erase Flash

The AT89C51SND1 Flash memory is divided in four blocks:

Block 0: from address 0000h to 1FFFh (64 pages)

Block 1: from address 2000h to 3FFFh (64 pages)

Block 2: from address 4000h to 7FFFh (128 pages)

Block 3: from address 8000h to FFFFh (256 pages)

- Parameter Settings

API_name	R1	DPTR0	DPTR1	ACC
__api_erase_block0	01h	0000h	x	x
__api_erase_block1		2000h	x	x
__api_erase_block2		4000h	x	x
__api_erase_block3		8000h	x	x

- Instruction: LCALL FFF0h.

Note: 1. Refer to the AT89C51SND1 datasheet for information on write operation timing and multiply this timing 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 F000h

Appendix A

Table 6. Summary of Frames From Host

<table>
<tr><th>Command</th><th>Record Type</th><th>Record Length</th><th>Offset</th><th>Data[0]</th><th>Data[1]</th><th>Data[2]</th><th>Data[3]</th><th>Data[4]</th></tr>
<tr><td>Program Nb Data Byte in Flash.</td><td>00h</td><td>nb of data (up to 128)</td><td>start address</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td></tr>
<tr><td>Erase block0 (0000h-1FFFh)</td><td rowspan="14">03h</td><td rowspan="4">02h</td><td rowspan="4">x</td><td rowspan="4">01h</td><td>00h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Erase block1 (2000h-3FFFh)</td><td>20h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Erase block2 (4000h-7FFFh)</td><td>40h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Erase block3 (8000h-FFFFh)</td><td>80h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Start application with a reset pulse generation</td><td>02h</td><td>x</td><td rowspan="2">03h</td><td>00h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Start application with a jump at “address”</td><td>04h</td><td>x</td><td>01h</td><td colspan="2">address</td><td>–</td></tr>
<tr><td>Erase SBV & BSB</td><td rowspan="3">02h</td><td>x</td><td>04h</td><td>00h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Program SSB level 1</td><td>x</td><td rowspan="2">05h</td><td>00h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Program SSB level 2</td><td>x</td><td>01h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Program BSB</td><td rowspan="2">03h</td><td>x</td><td rowspan="2">06h</td><td>00h</td><td>value</td><td>–</td><td>–</td></tr>
<tr><td>Program SBV</td><td>x</td><td>01h</td><td>value</td><td>–</td><td>–</td></tr>
<tr><td>Full Chip Erase</td><td>01h</td><td>x</td><td>07h</td><td>-</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Program bit BLJB</td><td rowspan="2">03h</td><td>x</td><td rowspan="2">0Ah</td><td>04h</td><td>bit value</td><td>–</td><td>–</td></tr>
<tr><td>Program bit X2</td><td>x</td><td>08h</td><td>bit value</td><td>–</td><td>–</td></tr>
<tr><td>Read Flash</td><td rowspan="2">04h</td><td rowspan="2">05h</td><td rowspan="2">x</td><td colspan="2" rowspan="2">Start Address</td><td colspan="2" rowspan="2">End Address</td><td>00h</td></tr>
<tr><td>Blank Check</td><td>01h</td></tr>
<tr><td>Read Manufacturer Code</td><td rowspan="11">05h</td><td rowspan="11">02h</td><td rowspan="11">x</td><td rowspan="4">00h</td><td>00h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Read Family Code</td><td>01h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Read Product Name</td><td>02h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Read Product Revision</td><td>03h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Read SSB</td><td rowspan="3">07h</td><td>00h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Read BSB</td><td>01h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Read SBV</td><td>02h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Read Hardware Byte</td><td>0Bh</td><td>00h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Read Device Boot ID1</td><td rowspan="2">0Eh</td><td>00h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Read Device Boot ID2</td><td>01h</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>Read bootloader Version</td><td>0Fh</td><td>00h</td><td>–</td><td>–</td><td>–</td></tr>
</table>

Appendix B

Table 7. 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 first Byte to program in 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_erase_block2	yes	01h	4000h	x	x
__api_erase_block3	yes	01h	8000h	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_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_start_bootloader	no	–	–	–	–

Atmel Corporation

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

Regional Headquarters

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

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

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

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

Printed on recycled paper.

4241B–MP3–07/04 /xM