Features

- Protocol
  - USB Used as Physical Layer
  - Device Firmware Upgrade Class Compliant
  - USB Clock Auto-Configuration
- 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 (IAP)
  - Read/Write Flash and EEPROM Memories
  - Read Device ID
  - Block Erase
  - Read/Write Configuration Bytes
  - Bootloader Start

Description

This document describes the USB bootloader functionalities as well as the USB protocol to efficiently perform operations on the on-chip Flash (EEPROM) memories. Additional information on the AT89C5131A product can be found in the AT89C5131A datasheet and the AT89C5131A errata sheet available on the Atmel web site.

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

Bootloader Revision	Purpose of Modifications	Date
Revision 1.0.2	First release	25/03/2003
Revision 1.2.0	Bootloader improvement	20/03/2007

USB Microcontrollers

AT89C5131A USB Bootloader

4287E–USB–04/08

Functional Description

The AT89C5131A bootloader facilitates In-System Programming and In-Application Programming.

In-System Programming Capability (IAP)

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

In-Application Programming or Self Programming Capability (ISP)

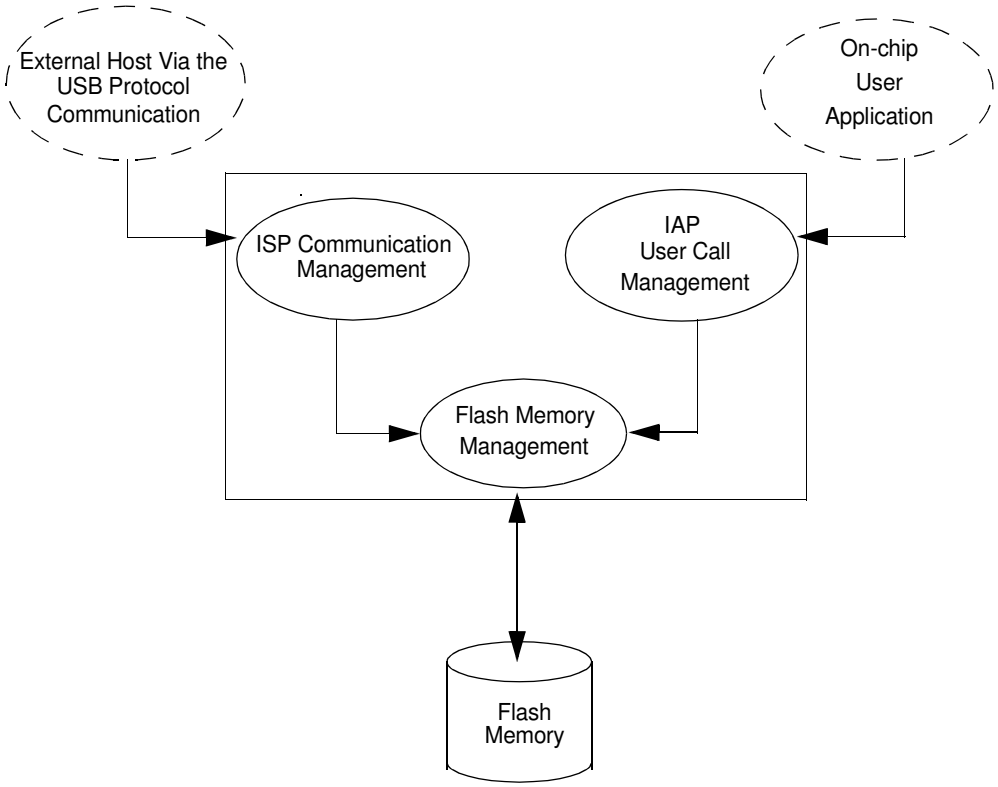
In-Application Programming (IAP) allows the reprogramming of a microcontroller on-chip Flash memory without removing it from the system and while the embedded application is running.

The USB bootloader contains some Application Programming Interface routines named API routines that allow 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 USB protocol (see section “Protocol”). This process translates serial communication frames (USB) 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 access to the Flash memory (performs read and write access).

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.

Mnemonic	Description	Default Value
BSB	Boot Status Byte	FFh
SBV	Software Boot Vector	FCh
SSB	Software Security Byte	FFh
EB	Extra Byte	FFh
P1_CF	Port 1 Configuration	FEh
P3_CF	Port 3 Configuration	FFh
P4_CF	Port 4 Configuration	FFh
Manufacturer		58h
Id1: Family code		D7h
Id2: Product Name		F7h
Id3: Product Revision		DFh

Mapping and Default Value of Hardware Security Byte

The 4 MSB of the Hardware Byte can be read/written by software (this area is called Fuse bits). The 4 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	OSCON1	U	Oscillator control (bit 1)
4	OSCON0	U	Oscillator control (bit 0)
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

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.
  The Bootloader returns an err_WRITE status.
  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 err_WRITE or an err_VENDOR status.

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	Any access allowed	Any access 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

In-System Programming

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

This section describes how to start the USB bootloader and the higher level protocol over the USB.

Boot Process

The bootloader can be activated in two ways:

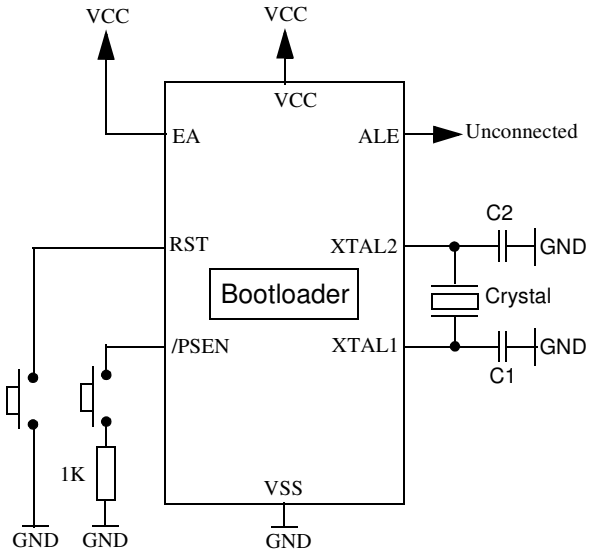
- Hardware conditions
- Regular boot process

Figure 3 and Figure 4 describe the boot process flows for low pin count and high pin count products.

High Pin Count Hardware Conditions

The Hardware conditions (EA = 1, PSEN = 0) during the $\overline{\text{RESET}}$ rising edge force the on-chip bootloader execution. In this way the bootloader can be carried out regardless of the user Flash memory content. It is recommended to pull the PSEN pin down to ground though a 1K resistor to prevent the PSEN pin from being damaged (see Figure 2 below).

Figure 2. ISP Hardware conditions



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 rising edge of reset signal. The hardware conditions are sampled at reset signal rising edge, thus they can be released at any time when reset input is high.

Low Pin Count Hardware Conditions

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 are 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 Figure 4 on page 9).

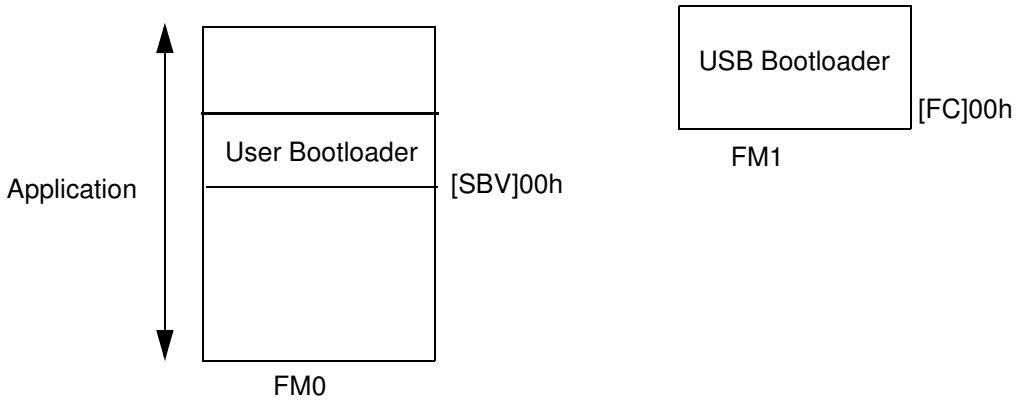
Note: The BLJB must be at 0 (programmed) to be able to restart the bootloader.
If the BLJB is equal to 1 (unprogrammed) only the hardware parallel programmer can change this bit (see AT89C5131A datasheet for more details).

Software Boot Vector

The default value [FF]00h is used in ISP mode. The boot address is, in this mode, the lowest adress of FM1 USB bootloader.

The Software Boot Vector (SBV) can be used to force 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”.



FLIP Software Program

FLIP is a PC software program running under Windows® 9x/Me/2000/XP and Linux® which can be used in ISP mode and which supports all Atmel C51 Flash microcontroller and USB protocol communication media.

The FLIP software program is free and is available from the Atmel web site.

Figure 3. High-pin Count Regular Boot Process

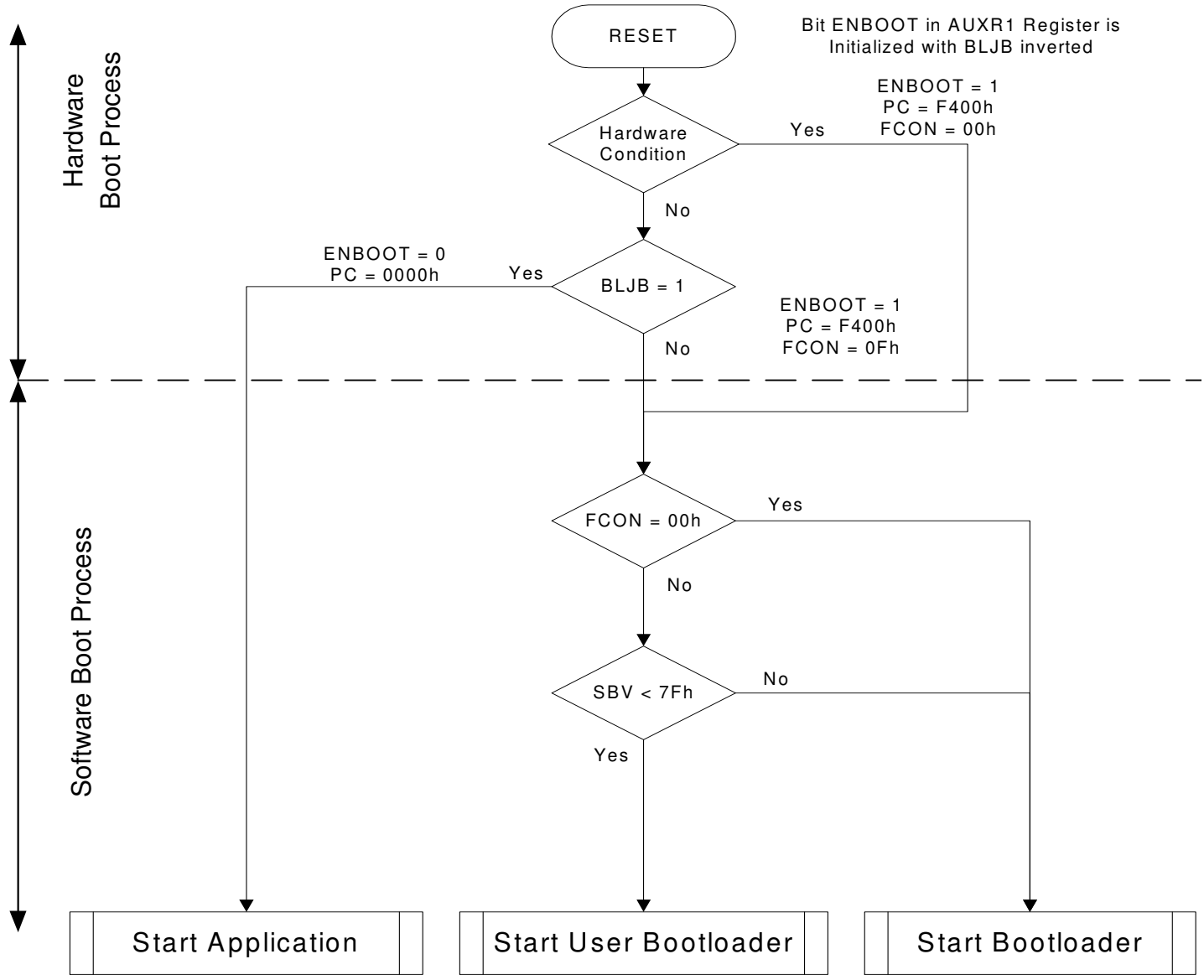
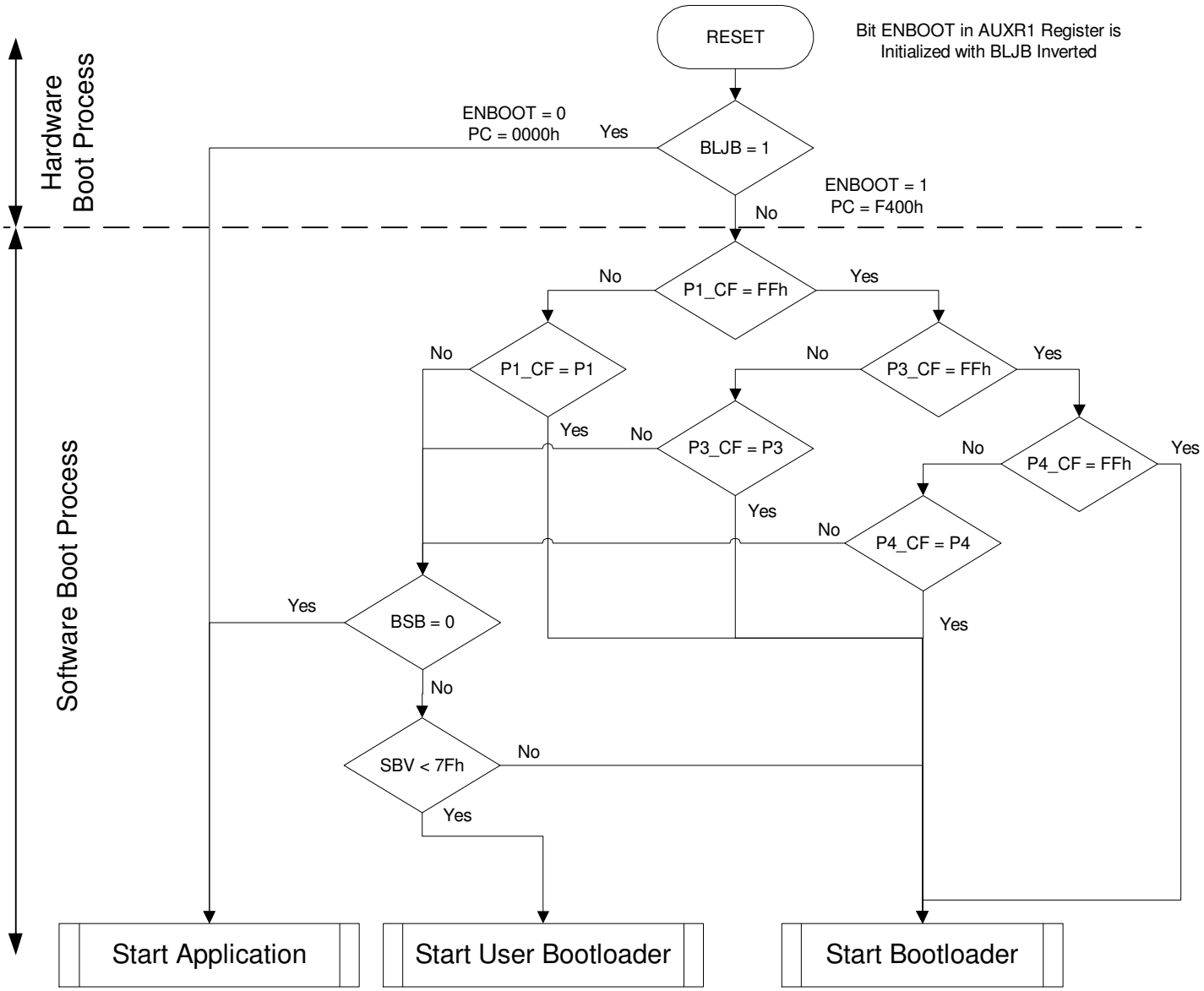


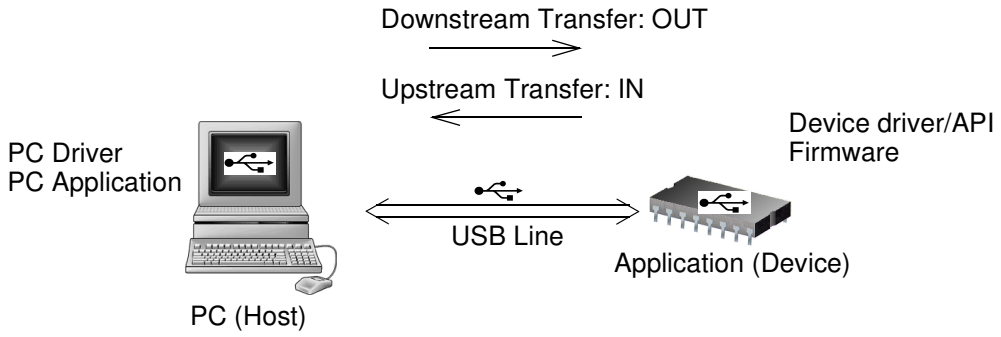
Figure 4. Low-pin Count Regular Boot Process



Physical Layer

The USB norm specifies all the transfers over the USB line. The USB specification also includes several CLASS and SUB-CLASS specifications. These stand-alone documents are used by the manufacturer to implement a USB link between a PC and a device supporting the In System Programming. Mostly, the USB specification is implemented by hardware (automatic reply, handshakes, timings, ...) and the USB Classes and SubClasses are implemented by software at a data level.

Figure 5. USB Bus Topography



The USB is used to transmit information that has the following configuration:

- USB DFU using the Default Control Endpoint only (endpoint 0) with a 32 bytes length.
- 48 MHz for USB controller: USB clock configuration performed by the bootloader

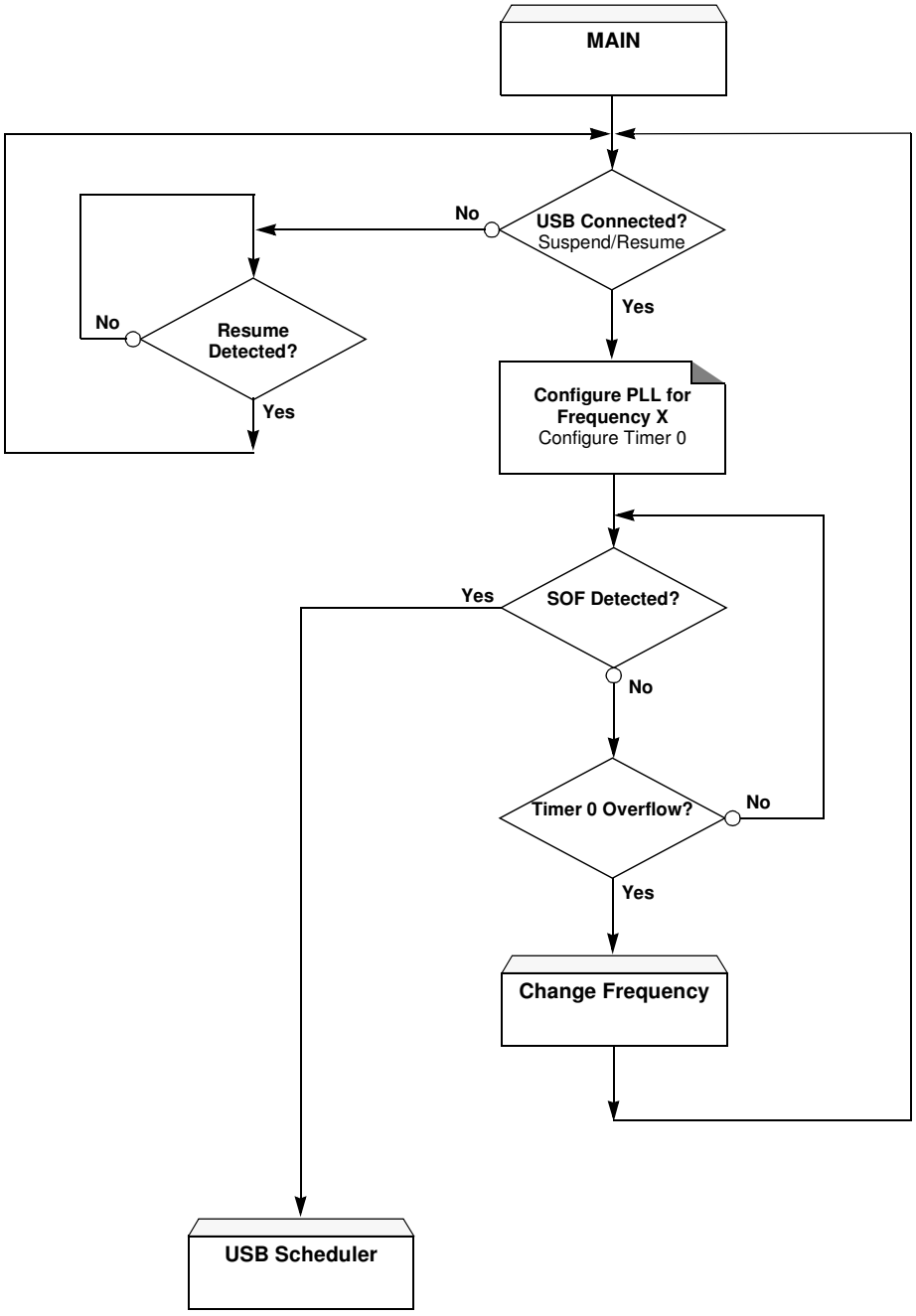
48 MHz Frequency Auto-Configuration

The bootloader includes a function which will automatically setup the PLL frequency (48MHz) versus the different XTAL configuration used on the application.

The table below shows the allowed frequencies compatible with the USB bootloader 48 MHz auto-generation.

	6 MHz	8 MHz	12 MHz	16 MHz	20 MHz	24 MHz	32 MHz	40 MHz	48 MHz
X1 or X2 Clock Modes	OK	OK	OK	OK	OK	OK	OK	OK	OK

Figure 6. 48 MHz Frequency Auto-Configuration



Protocol

Device Firmware Upgrade Introduction

Device Firmware Upgrade is the mechanism for accomplishing the task of upgrading the device firmware. Any class of USB device can exploit this capability by supporting the requirements specified in this document.

Because it is impractical for a device to concurrently perform both DFU operations and its normal run-time activities, those normal activities must cease for the duration of the DFU operations. Doing so means that the device must change its operating mode; i.e., a printer is **not** a printer while it is undergoing a firmware upgrade; it is a PROM programmer. However, a device that supports DFU is not capable of changing its mode of operation on its own. External (human or host operating system) intervention is required.

DFU Specific Requests

In addition of the USB standard requests, 7 DFU class-specific requests are employed to accomplish the upgrade operations (Table 1):

Table 1. DFU Class-specific Requests

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0010 0001b	DFU_DETACH (0)	wTimeout	Interface (4)	Zero	none
0010 0001b	DFU_DNLOAD (1)	wBlock	Interface (4)	Length	Firmware
1010 0001b	DFU_UPLOAD (2)	wBlock	Interface (4)	Length	Firmware
1010 0001b	DFU_GETSTATUS (3)	Zero	Interface (4)	6	Status
0010 0001b	DFU_CLRSTATUS (4)	Zero	Interface (4)	Zero	none
1010 0001b	DFU_GETSTATE (5)	Zero	Interface (4)	1	State
0010 0001b	DFU_ABORT (6)	Zero	Interface (4)	Zero	none

DFU Descriptors Set

The device exports the DFU descriptor set, which contains:

- A DFU device descriptor
- A single configuration descriptor
- A single interface descriptor (including descriptors for alternate settings, if present)
- A single functional descriptor

DFU Device Descriptor

This descriptor is only present in the DFU mode descriptor set. The DFU class code is reported in the *bDeviceClass* field of this descriptor.

Table 2. USB Parameters

Parameter	Atmel – AT89C5131A Bootloader
Vendor ID	0x03EB
Product ID	0x2FFD
Release Number	0x0000

Table 3. DFU Mode Device Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	12h	Size of this descriptor, in bytes
1	bDescriptorType	1	01h	DFU FUNCTIONAL descriptor type
2	bcdUSB	2	0100h	USB specification release number in binary coded decimal
4	bDeviceClass	1	FEh	Application Specific Class Code
5	bDeviceSubClass	1	01h	Device Firmware Upgrade Code
6	bDeviceProtocol	1	00h	The device does not use a class specific protocol on this interface
7	bMaxPacketSize0	1	32	Maximum packet size for endpoint zero
8	idVendor	2	03EBh	Vendor ID
10	idProduct	2	2FFDh	Product ID
12	bcdDevice	2	0x0000	Device release number in binary coded decimal
14	iManufacturer	1	0	Index of string descriptor
15	iProduct	1	0	Index of string descriptor
16	iSerialNumber	1	0	Index of string descriptor
17	bNumConfigurations	1	01h	One configuration only for DFU

DFU Configuration Descriptor

This descriptor is identical to the standard configuration descriptor described in the USB DFU specification version 1.0, with the exception that the *bNumInterfaces* field must contain the value 01h.

DFU Interface Descriptor

This is the descriptor for the only interface available when operating in DFU mode. Therefore, the value of the *bInterfaceNumber* field is always zero.

Table 4. DFU Mode Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	09h	Size of this descriptor, in bytes
1	bDescriptorType	1	04h	INTERFACE descriptor type
2	bInterfaceNumber	1	00h	Number of this interface
3	bAlternateSetting	1	00h	Alternate setting(1)
4	bNumEndpoints	1	00h	Only the control pipe is used
5	bInterfaceClass	1	FEh	Application Specific Class Code
6	bInterfaceSubClass	1	01h	Device Firmware Upgrade Code
7	bInterfaceProtocol	1	00h	The device doesn't use a class specific protocol on this interface
8	iInterface	1	00h	Index of the String descriptor for this interface

Note: 1. Alternate settings can be used by an application to access additional memory segments. In this case, it is suggested that each alternate setting employ a string descriptor to indicate the target memory segment; e.g., "EEPROM". Details concerning other possible uses of alternate settings are beyond the scope of this document. However, their use is intentionally not restricted because the authors anticipate that implementers will devise additional creative uses for alternate settings.

DFU Functional Descriptor

Table 5. DFU Functional Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	07h	Size of this descriptor, in bytes
1	bDescriptorType	1	21h	DFU FUNCTIONAL descriptor type
2	bmAttributes	1	Bit mask	DFU Attributes: bit 7..3: reserved bit 2: device is able to communicate via USB after Manifestation phase 1 = yes, 0 = no, must see bus reset bit 1: *bitCanUpload* : upload capable 1 = yes, 0 = no bit 0: *bitCanDnload* : download capable 1 = yes, 0 = no
3	wDetachTimeOut	2	Number	Time in milliseconds that the device will wait after receipt of the DFU_DETACH request. If this time elapses without a USB reset, the device will terminate the Reconfiguration phase and revert back to normal operation. This represents the maximum time that the device can wait (depending on its timers, ...). The Host may specify a shorter timeout in the DFU_DETACH request.
5	wTransferSize	2	Number	Maximum number of bytes that the device can accept per control-write transaction

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.

Device Status

Get Status

The Host employs the DFU_GETSTATUS request to facilitate synchronization with the device. This status gives information on the execution of the previous request: in progress/OK/Fail/...

bmRequestType	bRequest	wValue	wIndex	wLength	Data
1010 0001b	DFU_GETSTATUS (3)	Zero	Interface (4)	6	Status
0010 0001b	DFU_CLRSTATUS (4)	Zero	Interface (4)	Zero	none

The device responds to the DFU_GETSTATUS request with a payload packet containing the following data:

Table 6. DFU_GETSTATUS Response

Offset	Field	Size	Value	Description
0	bStatus	1	Number	An indication of the status resulting from the execution of the most recent request.
1	bwPollTimeOut	3	Number	Minimum time in milliseconds that the host should wait before sending a subsequent DFU_GETSTATUS. The purpose of this field is to allow the device to dynamically adjust the amount of time that the device expects the host to wait between the status phase of the next DFU_DNLOAD and the subsequent solicitation of the device's status via DFU_GETSTATUS.
4	bState	1	Number	An indication of the state that the device is going to enter immediately following transmission of this response.
5	iString	1	Index	Index of status description in string table.

Table 7. *bStatus* values

Status	Value	Description
OK	0x00	No error condition is present
errTARGET	0x01	File is not targeted for use by this device
errFILE	0x02	File is for this device but fails some vendor-specific verification test
errWRITE	0x03	Device id unable to write memory
errERASE	0x04	Memory erase function failed
errCHECK_ERASED	0x05	Memory erase check failed
errPROG	0x06	Program memory function failed
errVERIFY	0x07	Programmed memory failed verification
errADDRESS	0x08	Cannot program memory due to received address that is out of range
errNOTDONE	0x09	Received DFU_DNLOAD with *wLength* = 0, but device does not think it has all the data yet.
errFIRMWARE	0x0A	Device's firmware is corrupted. It cannot return to run-time operations
errVENDOR	0x0B	*iString* indicates a vendor-specific error

Table 7. *bStatus* values (Continued)

Status	Value	Description
errUSBR	0x0C	Device detected unexpected USB reset signaling
errPOR	0x0D	Device detected unexpected power on reset
errUNKNOWN	0x0E	Something went wrong, but the device does not know what it was
errSTALLEDPK	0x0F	Device stalled an unexpected request

Table 8. *bState* Values

State	Value	Description
appIDLE	0	Device is running its normal application
appDETACH	1	Device is running its normal application, has received the DFU_DETACH request, and is waiting for a USB reset
dfuIDLE	2	Device is operating in the DFU mode and is waiting for requests
dfuDNLOAD-SYNC	3	Device has received a block and is waiting for the Host to solicit the status via DFU_GETSTATUS
dfuDNBUSY	4	Device is programming a control-write block into its non volatile memories
dfuDNLOAD-IDLE	5	Device is processing a download operation. Expecting DFU_DNLOAD requests
dfuMANIFEST-SYNC	6	Device has received the final block of firmware from the Host and is waiting for receipt of DFU_GETSTATUS to begin the Manifestation phase or device has completed the Manifestation phase and is waiting for receipt of DFU_GETSTATUS.
dfuMANIFEST	7	Device is in the Manifestation phase.
dfuMANIFEST-WAIT-RESET	8	Device has programmed its memories and is waiting for a USB reset or a power on reset.
dfuUPLOAD-IDLE	9	The device is processing an upload operation. Expecting DFU_UPLOAD requests.
dfuERROR	10	An error has occurred. Awaiting the DFU_CLRSTATUS request.

Clear Status

Any time the device detects an error and reports an error indication status to the host in the response to a DFU_GETSTATUS request, it enters the dfuERROR state. The device cannot transition from the dfuERROR state, after reporting any error status, until after it has received a DFU_CLRSTATUS request. Upon receipt of DFU_CLRSTATUS, the device sets a status of OK and transitions to the dfuIDLE state. Only then is it able to transition to other states.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0010 0001b	DFU_CLRSTATUS (4)	Zero	Interface (4)	0	None

Device State

This request solicits a report about the state of the device. The state reported is the current state of the device with no change in state upon transmission of the response. The values specified in the *bState* field are identical to those reported in DFU_GETSTATUS.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
1010 0001b	DFU_GETSTATE (5)	Zero	Interface (4)	1	State

DFU_ABORT request

The DFU_ABORT request enables the device to exit from certain states and return to the DFU_IDLE state. The device sets the OK status on receipt of this request. For more information, see the corresponding state transition summary.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
1010 0001b	DFU_ABORT (6)	Zero	Interface (4)	0	None

Programming the Flash or EEPROM Data

The firmware image is downloaded via control-write transfers initiated by the DFU_DNLOAD class-specific request. The host sends between *bMaxPacketSize0* and *wTransferSize* bytes to the device in a control-write transfer. Following each downloaded block, the host solicits the device status with the DFU_GETSTATUS request.

As described in the USB DFU Specification, "Firmware images for specific devices are, by definition, vendor specific. It is therefore required that target addresses, record sizes, and all other information relative to supporting an upgrade are encapsulated within the firmware image file. It is the responsibility of the device manufacturer and the firmware developer to ensure that their devices can consume these encapsulated data. With the exception of the DFU file suffix, the content of the firmware image file is irrelevant to the host."

Firmware image:

- 32 bytes: Command
- X bytes: X is the number of byte (00h) added before the first significative byte of the firmware. The X number is calculated to align the beginning of the firmware with the flash page. X = start_address [32]. For example, if the start address is 00AFh (175d), X = 175 [32] = 15.
- The firmware
- The DFU Suffix on 16 Bytes.

Table 9. DFU File Suffix

Offset	Field	Size	Value	Description
-0	dwCRC	4	Number	The CRC of the entire file, excluding *dwCRC*
-4	bLength	1	16	The length of this DFU suffix including *dwCRC*
-5	ucDfuSignature	3	5 : 44h 6 : 46h 7 : 55h	The unique DFU signature field
-8	bcdDFU	2	BCD 0100h	DFU specification number
-10	idVendor	2	ID	The vendor ID associated with this file. Either FFFFh or must match device's vendor ID

Offset	Field	Size	Value	Description
-12	idProduct	2	ID	The product ID associated with this file. Either FFFFf or must match the device's product ID
-14	bcdDevice	2	BCD	The release number of the device associated with this file. Either FFFFh or a BCD firmware release or version number

Request From Host

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0010 0001b	DFU_DNLOAD (1)	wBlock	Interface (4)	Length	Firmware

Write Command

Command Identifier	data[0]	data[1]	data[2]	data[3]	data[4]	Description
ld_prog_start 01h	00h	start_address		end_address		Init FLASH programming
	01h					Init EEPROM programming

The write command is 6 bytes long. In order to reach the USB specification of the Control type transfers, the write command is completed with 26 (= 32 - 6) non-significant bytes. The total length of the command is then 32 bytes, which is the length of the Default Control Endpoint.

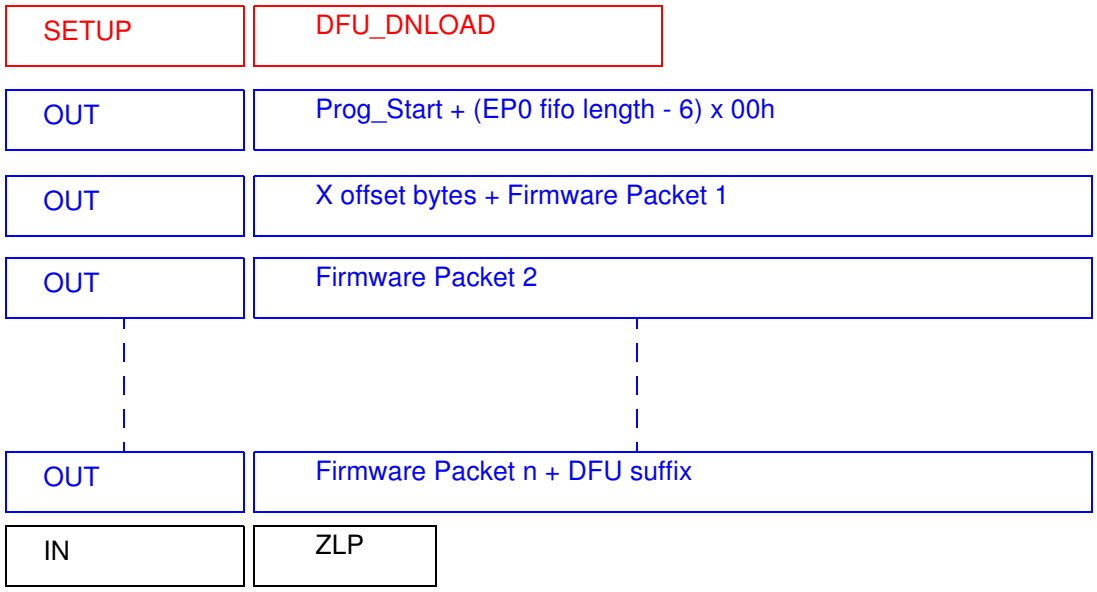
Firmware

The firmware can now be downloaded to the device. In order to be in accordance with the Flash page size (128 bytes), X non-significant bytes are added before the first byte to program. The X number is calculated to align the beginning of the firmware with the Flash page. X = start_address [32]. For example, if the start address is 00AFh (175d), X = 175 [32] = 15.

DFU Suffix

The DFU suffix of 16 bytes are added just after the last byte to program. This suffix is reserved for future use.

Figure 7. Example of Firmware Download Zero Length DFU_DNLOAD Request



The Host sends a DFU_DNLOAD request with the wLength field cleared to 0 to the device to indicate that it has completed transferring the firmware image file. This is the final payload packet of a download operation.

This operation should be preceded by a Long Jump address specification using the corresponding Flash command.

Answers from Bootloader

After each program request, the Host can request the device state and status by sending a DFU_GETSTATUS message.
If the device status indicates an error, the host can send a DFU_CLRSTATUS request to the device.

Reading the Flash or EEPROM Data

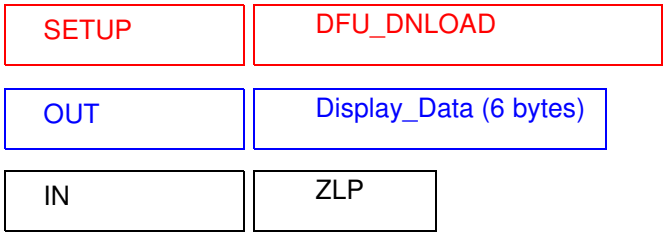
The flow described below allows the user to read data in the Flash memory or in the EEPROM data memory. A blank check command on the Flash memory is possible with this flow.

This operation is performed in 2 steps:

1. DFU_DNLOAD request with the read command (6 bytes)
2. DFU_UPLOAD request which correspond to the immediate previous command.

First Request from Host

The Host sends a DFU Download request with a Display command in the data field.



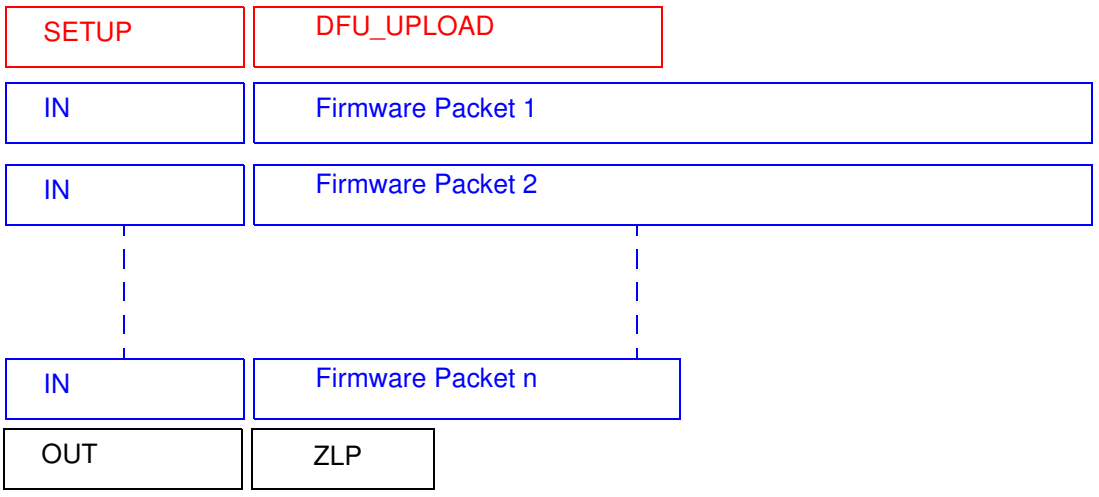
Command Identifier	data[0]	data[1]	data[2]	data[3]	data[4]	Description
ld_display_data 03h	00h	start_address		end_address		Display FLASH Data
	01h					Blank Check in FLASH
	02h					Display EEPROM Data

Second Request from Host

The Host sends a DFU Upload request.

Answers from the Device

The device send to the Host the firmware from the specified start address to the end address.



Answers from the Device to a Blank Check Command

The Host controller send a GET_STATUS request to the device. Once internal blank check has been completed, the device sends its status.

- If the device status is “OK”:
  the device memory is then blank and the device waits the next Host request.
- If the device status is “errCHECK_ERASED”:
  the device memory is not blank. The device waits for an DFU_UPLOAD request to send the first address where the byte is not 0xFF.

Programming Configuration Information

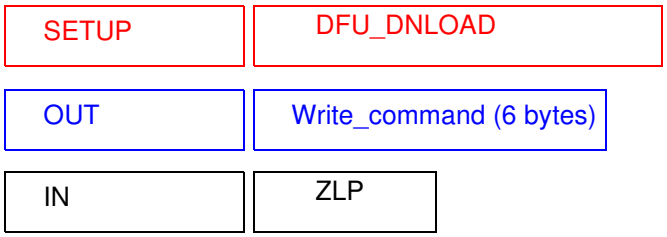
The flow described below allows the user to program Configuration Information regarding the bootloader functionality.

- Boot Process Configuration:
  - BSB
  - SBV
  - P1_CF, P3_CF and P4_CF
  - Fuse bits (BLJB, X2 and OSCON bits) (see section “Mapping and Default Value of Hardware Security Byte”)

Take care that the Program Fuse bit command programs the 4 Fuse bits at the same time.

Request from Host

To start the programming operation, the Host sends DFU_DNLOAD request with the Write command in the data field (6 bytes).



Command Identifier	data[0]	data[1]	data[2]	data[3]	data[4]	Description
Id_write_command 04h	01h	00h	Value			Write value in BSB
		01h				Write value in SBV
		02h				Write P1_CF
		03h				Write P3_CF
		04h				Write P4_CF
		05h				Write value in SSB
		06h				Write value in EB
	02h	00h	Value			Write value in Fuse (HSB)

Answers From Bootloader

The device has two possible answers to a DFU_GETSTATUS request:

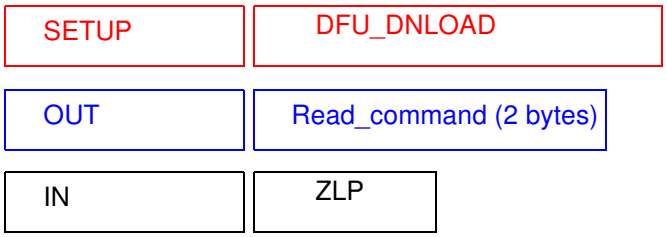
- If the chip is protected from program access, a “err_WRITE” status is returned to the Host.
- Otherwise, the device status is “OK“.

Reading Configuration Information or Manufacturer Information

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

Requests From Host

To start the programming operation, the Host sends DFU_DNLOAD request with the Read command in the data field (2 bytes).

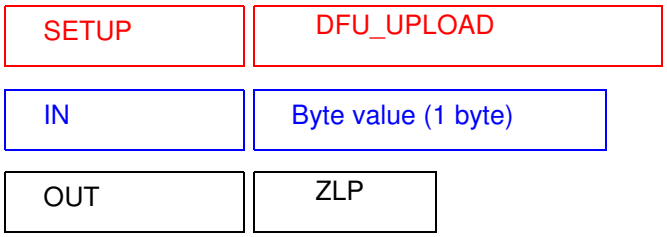


Command Identifier	data[0]	data[1]	data[2]	data[3]	data[4]	Description
ld_read_command 05h	00h	00h				Read Bootloader Version
		01h				Read Device boot ID1
		02h				Read Device boot ID2
	01h	00h				Read BSB
		01h				Read SBV
		02h				Read P1_CF
		03h				Read P3_CF
		04h				Read P4_CF
		05h				Read SSB
		06h				Read EB
		30h				Read Manufacturer Code
		31h				Read Family Code
		60h				Read Product Name
		61h				Read Product Revision
	02h	00h				Read HWB

Answers from Bootloader

The device has two possible answers to a DFU_GETSTATUS request:

- If the chip is protected from program access, an “err_VENDOR” status is returned to the Host.
- Otherwise, the device status is “OK“. The Host can send a DFU_UPLOAD request to the device in order the value of the requested field.



Erasing 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 (32 Kbytes) and sets some Configuration Bytes at 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 AT89C5131A:

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

Request from Host

To start the erasing operation, the Host sends a DFU_DNLOAD request with a Write Command in the data field (2 bytes).

Command Identifier	data[0]	data[1]	data[2]	data[3]	data[4]	Description
Id_write_command 04h	00h	00h				Erase block0 (0K to 8K)
		20h				Erase block1 (8K to 16K)
		40h				Erase block2 (16K to 32K)
		FFh				Full chip Erase (bits at FFh)

Answers from Bootloader

The device has two possible answers to a DFU_GETSTATUS request:

- If the chip is protected from program access, a “err_WRITE” status is returned to the Host.
- Otherwise, the device status is “OK“.

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

Starting the Application

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

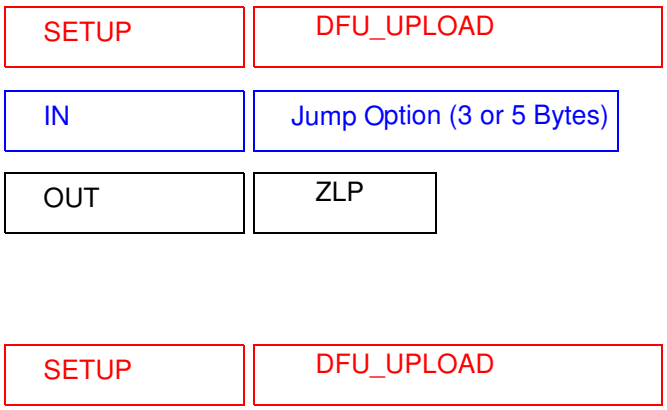
Two options are possible:

- 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.

To start the application, the Host sends a DFU_DNLOAD request with the specified application start type in the data field (3 or 5bytes).

This request is immediately followed by a second DFU_DNLOAD request with no data field to start the application with one of the 2 options.

Request From Host



Command Identifier	data[0]	data[1]	data[2]	data[3]	data[4]	Description
ld_write_command 04h	03h	00h				Hardware reset
		01h	address			LJMP address

Answer from Bootloader No answer is returned by the device.

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 package:

C Flash Drivers for the AT89C5131A for Keil® Compilers

The flash_api routines on the package work only with the USB 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 AT89C5131A 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 AT89C5131A 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 calls the __api_wr_code_byte or __api_wr_code_page see datasheet in section “Program/Code Memory”.
- Parameter settings

API Name	api_command	api_dph	api_dpl	api_value
__api_wr_code_byte __api_wr_code_page	0Dh			

- instruction: LCALL FFC0h.

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

Read Configuration and Manufacturer Information

- Parameter settings

API Name	api_command	api_dph	api_dpl	api_value
__api_rd_HSB	08h		00h	return HSB
__api_rd_BSB	05h		00h	return BSB
__api_rd_SBV	05h		01h	return SBV
__api_rd_P1_CF	05h		02h	return P1_CF
__api_rd_P3_CF	05h		03h	return P3_CF
__api_rd_P4_CF	05h		04h	return P4_CF
__api_rd_SSB	05h		05h	return SSB
__api_rd_EB	05h		06h	return EB
__api_rd_manufacturer	05h		30h	return manufacturer id
__api_rd_device_id1	05h		31h	return id1
__api_rd_device_id2	05h		60h	return id2
__api_rd_device_id3	05h		61h	return id3
__api_rd_bootloader_version	0Eh		00h	return value

- 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

Program Configuration Information

- Parameter settings

API Name	api_command	api_dph	api_dpl	api_value
__api_clr_BLJB	07h			(HSB & BFh) | 40h
__api_set_BLJB	07h			HSB & BFh
__api_clr_X2	07h			(HSB & 7Fh) | 80h
__api_set_X2	07h			HSB & 7Fh
__api_clr_OSCON1	07h			(HSB & DFh) | 20h
__api_set_OSCON1	07h			HSB & DFh
__api_clr_OSCON0	07h			(HSB & EFh) | 10h
__api_set_OSCON0	07h			HSB & EFh
__api_wr_BSB	04h		00h	value to write
__api_wr_SBV	04h		01h	value to write
__api_wr_P1_CF	04h		02h	value to write
__api_wr_P3_CF	04h		03h	value to write
__api_wr_P4_CF	04h		04h	value to write
__api_wr_SSB	04h		05h	value to write
__api_wr_EB	04h		06h	value to write

- instruction: LCALL FFC0h.

Notes: 1. See in the T89C51CC01 datasheet the time that a write operation takes.
2. No special resources are used by the bootloader during these operations.

Erasing the Flash

The AT89C5131A Flash memory is divided in several blocks:

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

API Name	api_command	api_dph	api_dpl	api_value
__api_erase_block0	00h	00h		
__api_erase_block1	00h	20h		
__api_erase_block2	00h	40h		

- instruction: LCALL FFC0h.

Notes: 1. See the AT89C5131A datasheet for the time that a write operation takes and this time must multiply by the number of pages.
2. No special resources are used by the bootloader during these operations.

Starting the 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 F400h

Appendix-A

Table 10. Summary of Frames from Host

Command Identifier	data[0]	data[1]	data[2]	data[3]	data[4]	Description
ld_prog_start 01h	00h	start_address		end_address		Init FLASH programming
	01h					Init EEPROM programming
ld_display_data 03h	00h	start_address		end_address		Display FLASH Data
	01h					Blank Check in FLASH
	02h					Display EEPROM Data
ld_write_command 04h	00h	00h				Erase block0 (0K to 8K)
		20h				Erase block1 (8K to 16K)
		40h				Erase block2 (16K to 32K)
		FFh				Full chip Erase (bits at FFh)
	01h	00h	Value			Write value in BSB
		01h				Write value in SBV
		02h				Write P1_CF
		03h				Write P3_CF
		04h				Write P4_CF
		05h				Write value in SSB
		06h				Write value in EB
	02h	00h	Value			Write value in Fuse (HSB)
	03h	00h				Hardware reset
		01h	address			LJMP address

Table 10. Summary of Frames from Host (Continued)

Command Identifier	data[0]	data[1]	data[2]	data[3]	data[4]	Description
ld_read_command 05h	00h	00h				Read Bootloader Version
		01h				Read Device boot ID1
		02h				Read Device boot ID2
	01h	00h				Read BSB
		01h				Read SBV
		02h				Read P1_CF
		03h				Read P3_CF
		04h				Read P4_CF
		05h				Read SSB
		06h				Read EB
		30h				Read Manufacturer Code
		31h				Read Family Code
		60h				Read Product Name
		61h				Read Product Revision
	02h	00h				Read HWB

Table 11. DFU Class-specific Requests

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0010 0001b	DFU_DETACH (0)	wTimeout	Interface (4)	Zero	none
0010 0001b	DFU_DNLOAD (1)	wBlock	Interface (4)	Length	Firmware
1010 0001b	DFU_UPLOAD (2)	wBlock	Interface (4)	Length	Firmware
1010 0001b	DFU_GETSTATUS (3)	Zero	Interface (4)	6	Status
0010 0001b	DFU_CLRSTATUS (4)	Zero	Interface (4)	Zero	none
1010 0001b	DFU_GETSTATE (5)	Zero	Interface (4)	1	State
0010 0001b	DFU_ABORT (6)	Zero	Interface (4)	Zero	none

Table 12. USB Parameters

Parameter	Atmel
Vendor ID	0x03EB
Product ID	0x2FFD
Release Number	0x0000

Table 13. Hardware Security Byte (HSB)

7	6	5	4	3	2	1	0
X2	BLJB	OSCON1	OSCON0		LB2	LB1	LB0

Appendix-2

Table 14. API Summary

Function Name	Bootloader Execution	api_command	api_dph	api_dpl	api_value
__api_rd_code_byte	no				
__api_wr_code_byte	yes	0Dh			
__api_wr_code_page	yes	0Dh			
__api_erase block0	yes	00h	00h		
__api_erase block1	yes	00h	20h		
__api_erase block2	yes	00h	40h		
__api_rd_HSB	yes	08h		00h	return value
__api_clr_BLJB	yes	07h			(HSB & BFh) | 40h
__api_set_BLJB	yes	07h			HSB & BFh
__api_clr_X2	yes	07h			(HSB & 7Fh) | 80h
__api_set_X2	yes	07h			HSB & 7Fh
__api_clr_OSCON1	yes	07h			(HSB & DFh) | 20h
__api_set_OSCON1	yes	07h			HSB & DFh
__api_clr_OSCON0	yes	07h			(HSB & EFh) | 10h
__api_set_OSCON0	yes	07h			HSB & EFh
__api_rd_BSB	yes	05h		00h	return value
__api_wr_BSB	yes	04h		00h	value
__api_rd_SBV	yes	05h		01h	return value
__api_wr_SBV	yes	04h		01h	value
__api_erase_SBV	yes	04h		01h	FFh
__api_rd_P1_CF	yes	05h		02h	return value
__api_wr_P1_CF	yes	04h		02h	value
__api_rd_P3_CF	yes	05h		03h	return value
__api_wr_P3_CF	yes	04h		03h	value
__api_rd_P4_CF	yes	05h		04h	return value
__api_wr_P4_CF	yes	04h		04h	value
__api_rd_SSB	yes	05h		05h	return value
__api_wr_SSB	yes	04h		05h	value
__api_rd_EB	yes	05h		06h	return value
__api_wr_EB	yes	04h		06h	value
__api_rd_manufacturer	yes	05h		30h	return value
__api_rd_device_id1	yes	05h		31h	return value

Table 14. API Summary (Continued)

Function Name	Bootloader Execution	api_command	api_dph	api_dpl	api_value
__api_wr_code_page	yes	01h			
__api_rd_device_id2	yes	05h		60h	return value
__api_rd_device_id3	yes	05h		61h	return value
__api_rd_bootloader_version	yes	0Eh		00h	return value
__api_eeprom_busy	no				
__api_rd_eeprom_byte	no				
__api_wr_eeprom_byte	no				
__api_start_bootloader	no				
__api_start_isp	no				

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
8051@atmel.com

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.