Features

- **USB Protocol**
  - **Based on the USB DFU class**
  - **Autobaud (8/16 MHz crystal)**
- **In-System Programming**
  - **Read/Write Flash and EEPROM on-chip memories**
  - **Read Device ID**
  - **Full chip Erase**
  - **Start application command**
- **In-Application Programming**
  - **Software Entry-points for on-chip flash drivers**

USB DFU Bootloader Datasheet

AT90USB128x
AT90USB64x
AT90USB162
AT90USB82
ATmega32U4
ATmega16U4

1. Description

The 8bits mega AVR with USB interface devices are factory configured with a USB bootloader located in the on-chip flash boot section of the controller.

This USB bootloader allows to perform In-System Programming from an USB host controller without removing the part from the system or without a pre-programmed application, and without any external programming interface.

This document describes the USB bootloader functionalities as well as the serial protocol to efficiently perform operations on the on chip Flash memories (Flash and EEPROM).

7618C–AVR–07/08

2. Bootloader Environment

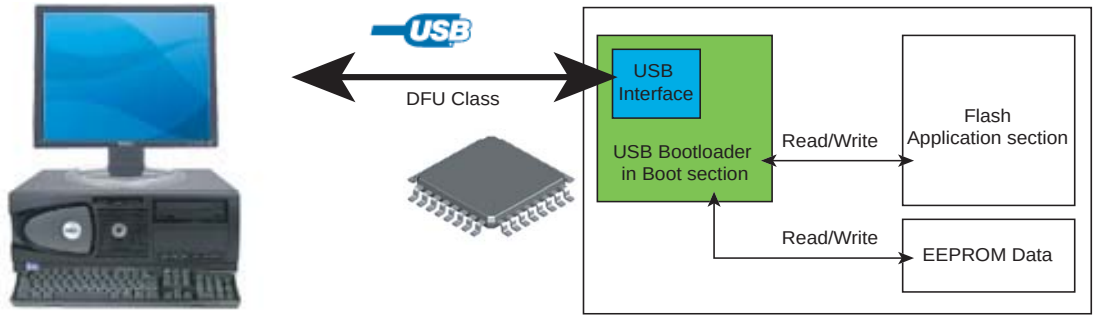
The bootloader is located in the boot section of the on-chip Flash memory, it manages the USB communication protocol and performs read/write operations to the on-chip memories (Flash/EEPROM).

The USB bootloader is loaded in the “Bootloader Flash Section” of the on-chip Flash memory. The size of the bootloader flash section must be larger than the bootloader size.USB products are factory configured with the default on-chip USB bootloader and the required bootsection configuration.

Table 2-1. USB Bootloader Parameters

Product	Flash Bootsection Size Configuration	VID / PID	Bootloader Start Address (word address)
AT90USB1287	4 KWord	0x03EB / 0x2FFB	0xf000
AT90USB1286			
AT90USB647	2 KWord	0x03EB / 0x2FF9	0x7800
AT90USB646			
AT90USB162		0x03EB / 0x2FFA	0x1800
AT90USB82		0x03EB / 0x2FF7	0x0800
ATmega32U4		0x03EB / 0x2FF4	0x3800
ATmega16U4		0x03EB / 0x2FF3	0x0800

Figure 2-1. Physical Environment



3. Bootloader Activation

As specified in the AT90USB datasheet, the bootloader can be activated by one of the following conditions:

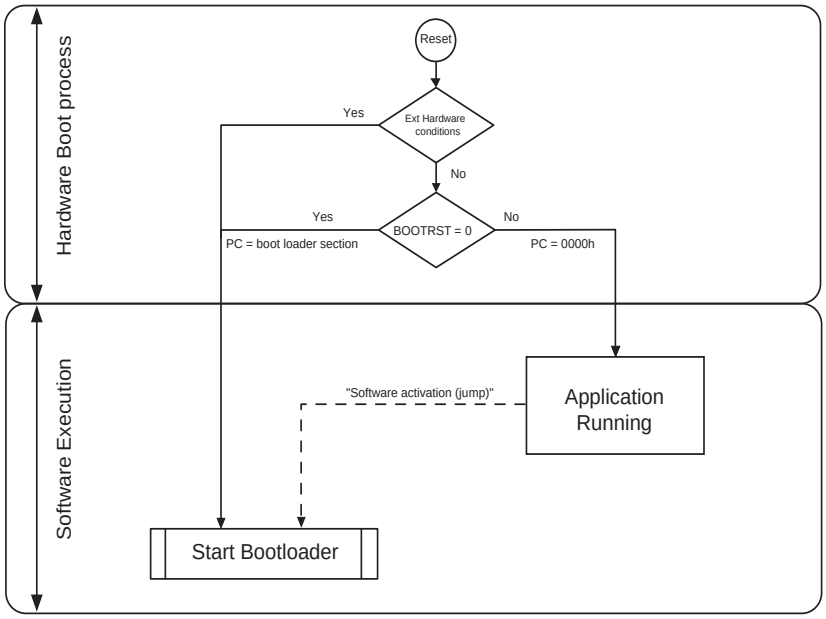
- **Regular application execution:** A jump or call from the user application program. This may be initiated by a trigger such as a command received via USB, USART or SPI and decoded by the application.

- **Boot Reset Fuse** The Boot Reset Fuse (BOOTRST) can be programmed so that the Reset Vector points to the Boot Flash section start address after reset. Once the user code is loaded, a bootloader command (“start application”) can start executing the application code. Note that the fuses cannot be changed by the MCU itself. This means that once the Boot Reset Fuse is programmed, the Reset Vector will always point to the Bootloader Reset and the fuse can only be changed through the serial or parallel programming interface. The BOOTRST fuse is not active in the default factory configuration.

- **External Hardware conditions** The Hardware Boot Enable fuse (HWBE) can be programmed so that upon special hardware conditions under reset, the bootloader execution is forced after reset.

These different conditions are summarized in Figure 3-1 on page 3.

Figure 3-1. Boot Process



4. Protocol

4.1 Device Firmware Upgrade Introduction

Device Firmware Upgrade (DFU) is the mechanism implemented to perform device firmware modifications. Any USB device can exploit this capability by supporting the requirements specified in this document.

Because it is unpractical for a device to concurrently perform both DFU operations and its normal run-time activities, these 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.

4.2 DFU Specific Requests

In addition to the USB standard requests, 7 DFU class-specific requests are used to accomplish the upgrade operations:

Table 4-1. 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

4.3 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)

4.3.1 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 4-2. 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 (limited to 32 due to Host side driver)
8	idVendor	2	03EBh	Vendor ID
10	idProduct	2	2FFBh	Product ID
12	bcdDevice	2	0x0000	Device release number in binary coded decimal
14	iManufacturer	1	0	Index of string descriptor

Offset	Field	Size	Value	Description
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

4.3.2 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.

4.3.2.1 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 4-3. DFU Mode Interface Description

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 does not 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 implements will devise additional creative uses for alternate settings.

4.4 Commands Description

The protocol implemented in the AT90USB bootloader 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” on page 18.

4.5 Device Status

4.5.1 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 4-4. 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 4-5. *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

Status	Value	Description
errVENDOR	0x0B	*iString* indicates a vendor-specific error
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 4-6. *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.

4.5.2 Clear Status

Each time the device detects and reports an error indication status to the host in response to a DFU_GETSTATUS request, it enters the dfuERROR state. After reporting any error status, the device can not leave the dfuERROR state, until it has received a DFU_CLRSTATUS request. Upon receipt of DFU_CLRSTATUS, the device sets status to OK and move to the dfuIDLE state. Once the device is in the dfuIDLE state it is then able to move to other states.

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

4.5.3 Device State

The state reported is the current state of the device up to 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

4.5.4 DFU_ABORT request

The DFU_ABORT request forces the device to exit from any other state 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

4.6 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 process 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 significant 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 4-7. 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

Offset	Field	Size	Value	Description
-10	idVendor	2	ID	The vendor ID associated with this file. Either FFFFh or must match device's vendor ID
-12	idProduct	2	ID	The product ID associated with this file. Either FFFFh 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

4.6.1 Request From Host

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

4.6.1.1 Write Command

Command Identifier	data[0]	data[1]	data[2]	data[3]	data[4]	Description
Id_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 meet with 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.

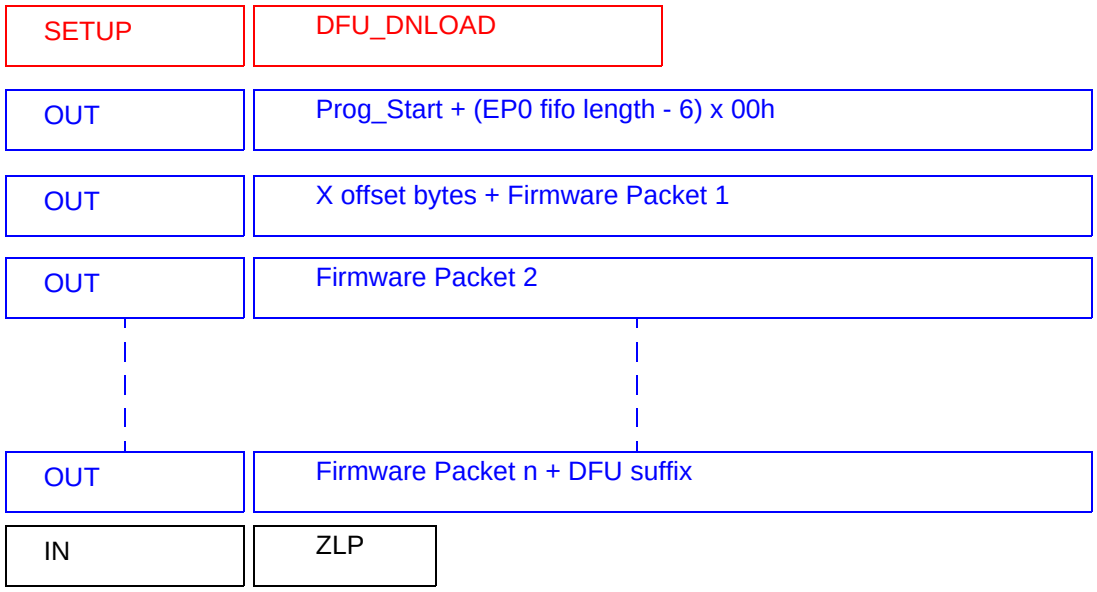
4.6.1.2 Firmware

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.

4.6.1.3 DFU Suffix

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

Figure 4-1. Example of Firmware Download Zero Length DFU_DNLOAD Request



The Host sends a DFU_DNLOAD request with Zero Length Packet (ZLP) to indicate that it has completed transferring the firmware image file. This is the final payload packet of a download operation.

4.6.1.4 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 must send a DFU_CLRSTATUS request to the device.

4.7 Reading the Flash or EEPROM Data

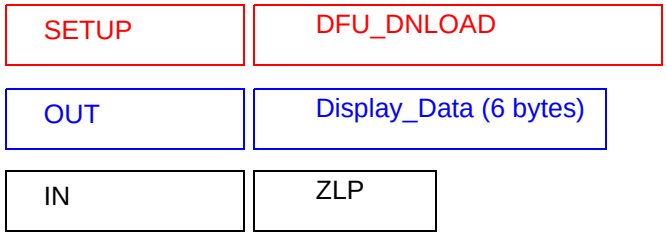
The flow described below allows the user to read data in the Flash memory or in the EEPROM data memory. A blank check command on the Flash memory is possible with this flow.

This operation is performed in 2 steps:

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

4.7.1 First Request from Host

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



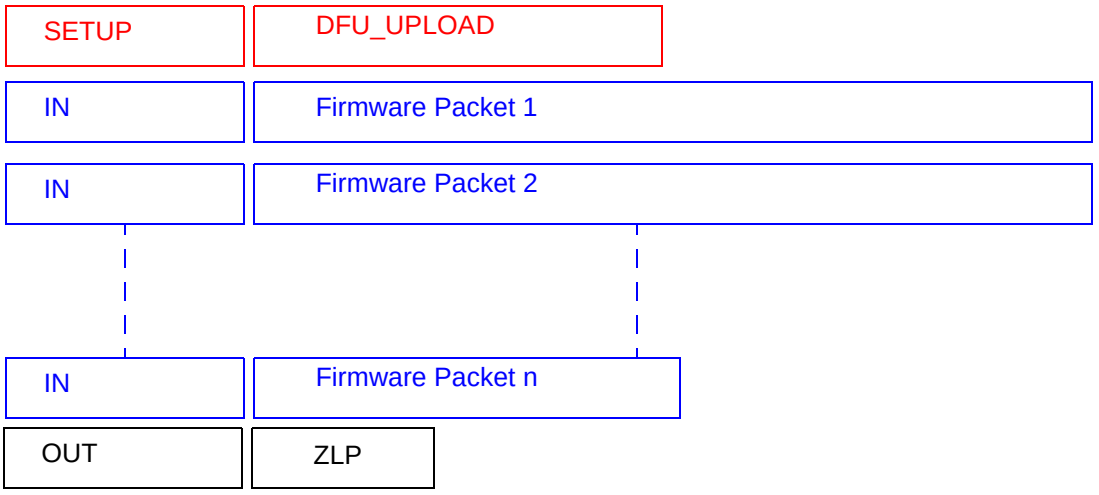
Command Identifier	data[0]	data[1]	data[2]	data[3]	data[4]	Description
Id_display_data 03h	00h	start_address		end_address		Display FLASH Data
	01h					Blank Check in FLASH
	02h					Display EEPROM Data

4.7.2 Second Request from Host

The Host sends a DFU Upload request.

4.7.3 Answers from the Device

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



4.7.4 Answers from the Device to a Blank Check Command

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

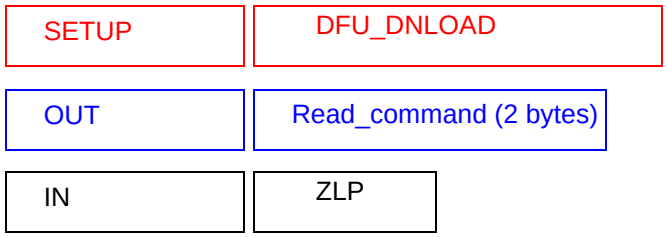
- If the device status is “OK”:
  the device memory is then blank and the device waits for the next Host request.
- If the device status is “errCHECK_ERASED”:
  the device memory is not blank. The device waits for an DFU_UPLOAD request to send the first address where the byte is not 0xFF.

4.8 Reading Configuration Information or Manufacturer Information

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

4.8.1 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
Id_read_command 05h	00h	00h				Read Bootloader Version
		01h				Read Device boot ID1
		02h				Read Device boot ID2
	01h	30h				Read Manufacturer Code
		31h				Read Family Code
		60h				Read Product Name
		61h				Read Product Revision

4.8.2 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 to get the value of the requested field.

SETUP	DFU_UPLOAD
IN	Byte value (1 byte)
OUT	ZLP

4.9 Erasing the Flash

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

The Full Chip erase command erases the whole Flash.

4.9.1 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	FFh				Full chip Erase (bits at FFh)

4.9.2 Answers from Bootloader

The device has two possible answers to a DFU_GETSTATUS request:

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

4.10 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 an internal hardware reset 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.

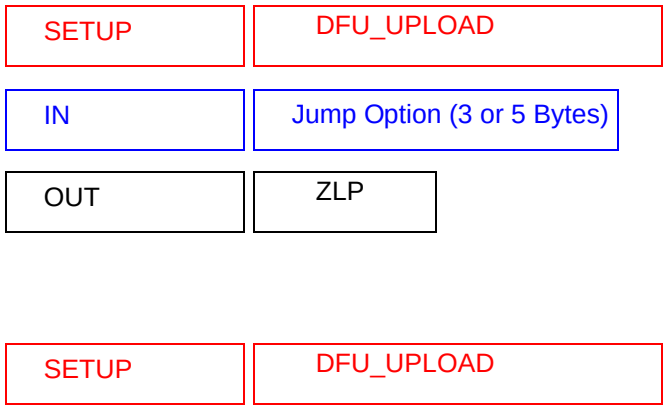
- Start the application without reset.
  A jump at the address 0000h is used to start the application without reset.

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

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.

Important note:
The bootloader performs a watchdog reset to generate the “hardware reset” that allows to execute the application section. After a watchdog reset occurs, the AVR watchdog is still running, thus the application should take care to disable watchdog at program start-up (otherwise the application that does not manage the hardware watchdog will run in an infinite reset loop).

4.11 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

4.12 Answer from Bootloader

No answer is returned by the device.

5. Security

When the USB bootloader connection is initiated, the bootloader automatically enters a read/write software security mode (independent of the product lock bits settings). This allows to protect the on-chip flash content from read/write access over the USB interface. Thus the only DFU command allowed after a USB bootloader connection is a “Full Chip Erase” command.

After this “Full Chip Erase” has been received and properly executed, all DFU commands are allowed, and thus the on-chip flash can be reprogrammed and verified.

6. Accessing the Low level Flash Drivers

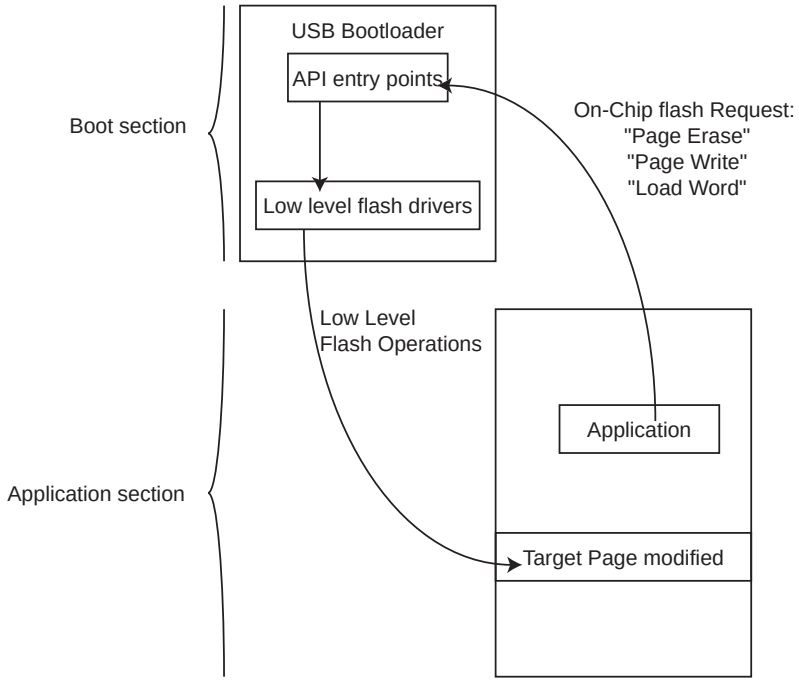
The AT90USB USB bootloader is located in the boot section of the on-chip flash memory, meanwhile the bootloader section is the unique memory location allowed to execute on-chip flash memory write operations (SPM instruction is decoded only in this section).

Thus applications which require on-chip flash write access can perform calls to specific entry points located in the USB bootloader.

The USB bootloader provides several Application Programming Interfaces (API) that allows the application to access low level flash drivers located in the boot section. These APIs allow the following operations:

- Page Erase
- Page Write
- Load word in the temporary page buffer

Figure 6-1. USB bootloader API calls



The API are located at absolute addresses in the USB bootloader firmware and accept specific registers values as parameters. These parameters are compatible with a C compiler calling convention and thus can be called directly with function pointer declared as in the example below:

C Code Example

```
#if (FLASH_END==0x1FFFF) //128K bytes parts
   #define  LAST_BOOT_ENTRY   0xFFFE
#elif (FLASH_END==0xFFFF)//64K bytes parts
   #define  LAST_BOOT_ENTRY   0x7FFE
#else
   #error You must define FLASH_END in bytes.
#endif

// These functions pointers are used to call functions entry points in bootloader
void (*boot_flash_page_erase_and_write)(unsigned long adr)=(void (*)(unsigned
long))(LAST_BOOT_ENTRY-12);
U8 (*boot_flash_read_sig) (unsigned long adr)=(U8 (*)(unsigned
long))(LAST_BOOT_ENTRY-10);
U8 (*boot_flash_read_fuse) (unsigned long adr)=(U8 (*)(unsigned
long))(LAST_BOOT_ENTRY-8);
void (*boot_flash_fill_temp_buffer) (unsigned int data,unsigned int adr)=(void
(*)(unsigned int, unsigned int))(LAST_BOOT_ENTRY-6);
void (*boot_flash_prg_page) (unsigned long adr)=(void (*)(unsigned
long))(LAST_BOOT_ENTRY-4);
void (*boot_flash_page_erase) (unsigned long adr)=(void (*)(unsigned
long))(LAST_BOOT_ENTRY-2);
void (*boot_lock_wr_bits) (unsigned char val)=(void (*)(unsigned
char))(LAST_BOOT_ENTRY);

// This function writes 0x55AA @ 0x1200 in the on-flash calling flash drivers located
in USB bootloader
void basic_flash_access(void)
{
  unsigned long address;
  unsigned int temp16;
  temp16=0x55AA;
  address=0x12000;
  (*boot_flash_fill_temp_buffer)(temp16,address);
  (*boot_flash_page_erase)(address);
  (*boot_flash_prg_page)(address);
}
```

The full assembly code for the flash API drivers is given in “Appendix-B” on page 20.

7. Using the USB bootloader for In System Programming

Flip software is the PC side application used to communicate with the USB bootloader (Flip is available for free on the Atmel website).

For detailed instructions about using Flip and USB bootloader, please refer to AVR282: “USB Firmware Upgrade for AT90USB” (doc 7769).

8. Bootloader History

The following table shows the different bootloader revision and associated changes.

Table 8-1. USB Bootloader History

Product	Bootloader Revision	Changes
AT90USB1287 AT90USB1286 AT90USB647 AT90USB646	1.0.1	Initial Revision
AT90USB162 AT90USB82	1.0.0	Initial Revision
	1.0.1	Allow to use 16MHz cristal with 3.3V power supply and CKDIV8 fuse.
	1.0.5	Improved USB autobaud process
ATmega32U4 ATmega16U4	1.0.0	Initial Revision

9. Appendix-A

Table 9-1. 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	FFh				Full chip Erase (bits at FFh)
	03h	00h				Hardware reset
		01h	address			LJMP address
ld_read_command 05h	00h	00h				Read Bootloader Version
		01h				Read Device boot ID1
		02h				Read Device boot ID2
	01h	30h				Read Manufacturer Code
		31h				Read Family Code
		60h				Read Product Name
		61h				Read Product Revision
ld_change _base _address 06h	03h	00	“PP”			Select “PP” 64kBytes flash page number

Table 9-2. 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

10. Appendix-B

```
;*A***************************************************************************
**
; $RCSfile: flash_boot_drv.s90,v $
;---------------------------------------------------------------------------
--
; Copyright (c) Atmel.
;---------------------------------------------------------------------------
--
; RELEASE:      $Name:  $
; REVISION:     $Revision: 1.7 $
; FILE_CVSID:   $Id: flash_boot_drv.s90,v 1.7 2005/10/03 15:50:12 $
;---------------------------------------------------------------------------
--
; PURPOSE:
; This file contains the low level driver for the flash access
;***************************************************************************
**
NAMEflash_drv(16)

;_____ I N C L U D E S
_____________________________________________________

#define ASM_INCLUDE
#include "config.h"

;***************************************************************************
**
; This is the absolute table entry points for low level flash drivers
; This table defines the entry points that can be called
; from the application section to perform on-chip flash operations:
;
;  entry_flash_page_erase_and_write:
;        R18:17:R16: The byte address of the page
;
;  entry_flash_fill_temp_buffer:
;        data16 :  R16/R17: word to load in the temporary buffer.
;        address:  R18/R19: address of the word in the temp. buffer.
;
;  entry_flash_prg_page:
;        R18:17:R16: The byte address of the page
;
;  entry_flash_page_erase:
;        R18:17:R16: The byte address of the page
;
;***************************************************************************
**
ASEG FLASH_END-0x0001B
entry_flash_page_erase_and_write:
```

```
   JMP flash_page_erase_and_write
entry_flash_read_sig:
   JMP  flash_read_sig
entry_flash_read_fuse:
   JMP  flash_read_fuse
entry_flash_fill_temp_buffer:
   JMP  flash_fill_temp_buffer
entry_flash_prg_page:
   JMP  flash_prg_page
entry_flash_page_erase:
   JMP  flash_page_erase_public
entry_lock_wr_bits:
   JMP  lock_wr_bits

RSEGBOOT

;*F*****************************************************************
**
; NAME: flash_page_erase_and_write
;-----------------------------------------------------------------------
--
; PARAMS:  R18:17:R16: The byte address of the page
;-----------------------------------------------------------------------
--
; PURPOSE: This function can be called for the user appplication
; This function performs an erase operation of the selected target page and
; the launch the prog sequence of the same target page.
; This function allows to save the 256 bytes software temporary buffer in
; the application section
;*****************************************************************
**
flash_page_erase_and_write:
   PUSH  R18
   RCALL flash_page_erase
   POP   R18
   RCALL flash_prg_page
   RET

;*F*****************************************************************
**
; NAME: flash_prg_page
;-----------------------------------------------------------------------
--
; PARAMS: R18:17:R16: The byte address of the page
;-----------------------------------------------------------------------
--
; PURPOSE:  Launch the prog sequence of the target page
```

```
;*************************************************************************
**
flash_prg_page:
  RCALL   WAIT_SPMEN  ;Wait for SPMEN flag cleared
  MOV     R31,R17
  MOV     R30,R16     ;move adress to z pointer (R31=ZH R30=ZL)
  OUT     RAMPZ, R18
  LDI     R20,$05     ;(1<<PGWRT) + (1<<SPMEN))
  OUT SPMCSR,R20; argument 2 decides function (r18)
  SPM                 ;Store program memory
  RCALL    WAIT_SPMEN  ;Wait for SPMEN flag cleared
  RCALL   flash_rww_enable
  RET

;*F*************************************************************************
**
; NAME: flash_page_erase
;-------------------------------------------------------------------------
--
; PARAMS:   R18:17:R16: The byte address of the page
;-------------------------------------------------------------------------
--
; PURPOSE:  Launch the erase sequence of the target page
;-------------------------------------------------------------------------
--
; NOTE:  This function does nt set the RWWSE bit after erase. Thus it does
not
; erase the hardware temporary temp buffer.
; This function is for bootloader usage
;-------------------------------------------------------------------------
--
; REQUIREMENTS:
;*************************************************************************
**
 flash_page_erase:
  RCALL   WAIT_SPMEN  ;Wait for SPMEN flag cleared
  MOV     R31,R17
  MOV     R30,R16     ;move adress to z pointer (R31=ZH R30=ZL)
  OUT     RAMPZ, R18
  LDI     R20,$03     ;(1<<PGERS) + (1<<SPMEN)))
  OUT SPMCSR, R20; argument 2 decides function (r18)
  SPM                 ;Store program memory
  RCALL    WAIT_SPMEN  ;Wait for SPMEN flag cleared
  ;RCALL   flash_rww_enable CAUTION DO NOT ACTIVATE HERE or
  ;        you will loose the entire page buffer content !!!
  RET
```

```
;*F*****************************************************************
**
; NAME: flash_page_erase_public
;-----------------------------------------------------------------------
--
; PARAMS:  R18:17:R16: The byte address of the page
;-----------------------------------------------------------------------
--
; PURPOSE:  Launch the erase sequence of the target page
;-----------------------------------------------------------------------
--
; NOTE:  !!!!This function set the RWWSE bit after erase. Thus it
; erase the hardware temporary temp buffer after page erase
;*****************************************************************
**
flash_page_erase_public:
  RCALL   WAIT_SPMEN  ;Wait for SPMEN flag cleared
  MOV     R31,R17
  MOV     R30,R16     ;move adress to z pointer (R31=ZH R30=ZL)
  OUT     RAMPZ, R18
  LDI     R20,$03     ;(1<<PGERS) + (1<<SPMEN)))
  OUTSPMCSR, R20; argument 2 decides function (r18)
  SPM                 ;Store program memory
  RCALL    WAIT_SPMEN  ;Wait for SPMEN flag cleared
  RCALL   flash_rww_enable
  RET

;*F*****************************************************************
**
; NAME: flash_rww_enable
;-----------------------------------------------------------------------
--
; PARAMS:  none
;-----------------------------------------------------------------------
--
; PURPOSE:  Set RWSE bit. It allows to execute code in the application
section
; after a flash prog (erase or write page)
;*****************************************************************
**
  flash_rww_enable:
  RCALL   WAIT_SPMEN  ;Wait for SPMEN flag cleared
  LDI     R20,$11     ;(1<<WWSRE) + (1<<SPMEN)))
  OUT SPMCSR, R20 ; argument 2 decides function (r18)
  SPM                 ;Store program memory
  RJMP    WAIT_SPMEN  ;Wait for SPMEN flag cleared
```

```
;*F*****************************************************************
**
; NAME: flash_read_sig
;-----------------------------------------------------------------------
--
; PARAMS:
; Return: R16: signature value
;-----------------------------------------------------------------------
--
; PURPOSE:  Read harware signature byte. THe byte is selected trought the
addr
; passed as argument (see product data sheet)
;*****************************************************************
**
flash_read_sig:
  RCALL   WAIT_SPMEN  ;Wait for SPMEN flag cleared
  MOV     R31,R17
  MOV     R30,R16     ;move adress to z pointer (R31=ZH R30=ZL)
  OUT     RAMPZ, R18
  LDI     R20,$21     ;(1<<SPMEN) | (1<<SIGRD))
  OUT SPMCSR, R20; argument 2 decides function (r18)
  LPM                 ;Store program memory
  MOV     R16, R0      ;Store return value (1byte->R16 register)
  RJMP    WAIT_SPMEN  ;Wait for SPMEN flag cleared


;*F*****************************************************************
**
; NAME: flash_read_fuse
;-----------------------------------------------------------------------
--
; Return: R16: fuse value
;-----------------------------------------------------------------------
--
; PURPOSE: Read fuse byte. The fuse byte is elected through the address
passed
; as argument (See product datasheet for addr value)
;*****************************************************************
**
flash_read_fuse:
  RCALL   WAIT_SPMEN  ;Wait for SPMEN flag cleared
  MOV     R31,R17
  MOV     R30,R16     ;move adress to z pointer (R31=ZH R30=ZL)
  OUT     RAMPZ, R18
  LDI     R20,$09     ;(1<<SPMEN) | (1<<BLBSET))
  OUT SPMCSR, R20; argument 2 decides function (r18)
  LPM                 ;Store program memory
  MOV     R16, R0        ;Store return value (1byte->R16 register)
```

```
  RJMP    WAIT_SPMEN  ;Wait for SPMEN flag cleared

/*F****************************************************************************
**
* NAME: flash_fill_temp_buffer
*-------------------------------------------------------------------------------
--
* PARAMS:
* data16 :  R16/R17: word to load in the temporary buffer.
* address:  R18/R19: address of the word.
* return:  none
*-------------------------------------------------------------------------------
--
* PURPOSE:
* This function allows to load a word in the temporary flash buffer.
*-------------------------------------------------------------------------------
--
* EXAMPLE:
* fill_temp_buffer(data16, address);
*-------------------------------------------------------------------------------
--
* NOTE:
* the first paramater used the registers R16, R17
* The second parameter used the registers R18, R19
*******************************************************************************
**/
flash_fill_temp_buffer:
  MOV     R31,R19     ;move adress to z pointer (R31=ZH R30=ZL)
  MOV     R30,R18
  MOV     R0,R17      ;move data16 to reg 0 and 1
  MOV     R1,R16
  LDI     R20,(1<<SPMEN)
  OUT SPMCSR, R20; r18 decides function
  SPM                ; Store program memory
  RJMP   WAIT_SPMEN   ; Wait for SPMEN flag cleared

;*F****************************************************************************
**
; NAME: lock_wr_bits
;-------------------------------------------------------------------------------
--
; PARAMS:   R16: value to write
;-------------------------------------------------------------------------------
--
; PURPOSE:
;*******************************************************************************
**
lock_wr_bits:
```

```
  RCALL   WAIT_SPMEN  ; Wait for SPMEN flag cleared
  MOV     R0,R16
  LDI     R18,((1<<BLBSET)|(1<<SPMEN))
  OUT SPMCSR, R18 ; r18 decides function
  SPM                 ; write lockbits
  RJMP   WAIT_SPMEN   ; Wait for SPMEN flag cleared

;*F*****************************************************************************
**
; NAME: wait_spmen
;-----------------------------------------------------------------------------
--
; PARAMS:    none
;-----------------------------------------------------------------------------
--
; PURPOSE:   Performs an active wait on SPME flag
;*****************************************************************************
**
WAIT_SPMEN:
  MOVR0, R18
  INR18, SPMCSR   ; get SPMCR into r18
  SBRC   R18,SPMEN
  RJMP   WAIT_SPMEN    ; Wait for SPMEN flag cleared
  MOVR18, R0
  RET

END
```

11. Document Revision History

11.1 7618B 03/08

1. Removed references to DFU Functional Descriptor throughout the document.

11.2 7618C 07/08

1. Update for AT90USB162/82, AT90USB64x, ATmega32U4 and ATmega16U4.
2. Update bootloader revision history.

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
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Requests
www.atmel.com/literature