Features

- USB Protocol
 - Based on the USB DFU class
 - Autobaud (AT90USBxxx 2, 4, 6, 8, 10, 12, 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

1. Description

AT90USB 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).



AT90USB128x AT90USB64x

USB DFU Bootloader Datasheet



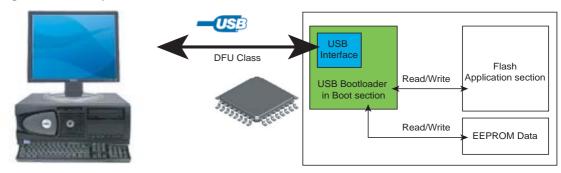


2. Bootloader Environment

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 AT90USB 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 (about 5K bytes), on AT90USBxxx it is configured to 8Kbytes.

Figure 2-1. Physical Environment



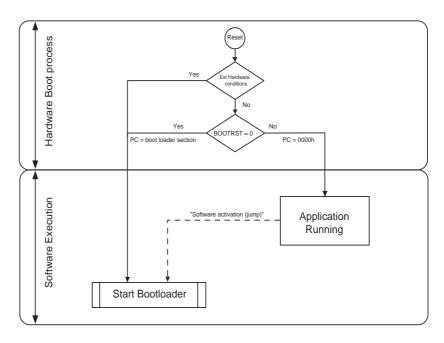
3. Bootloader Activation

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, 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.

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 1):

 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





bmRequestType	bRequest	wValue	wIndex	wLength	Data
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)
- A single functional descriptor

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. USB Parameters

Parameter	Atmel – AT90USB Bootloader		
Vendor ID	0x03EB		
Product ID	0x2FFB		
Release Number	0x0000		

 Table 4-3.
 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 (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
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 *blnterfaceNumber* field is always zero.

Table 4-4. 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.





4.3.2.2 DFU Functional Descriptor

Table 4-5. 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 73: reserved bit 2: device is able to communicate via USB after the manifestation phase 1 = yes, 0 = no, must see a 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 time-out in the DFU_DETACH request.
5	wTransferSize	2	Number	Maximum number of bytes that the device can accept per control- write transaction

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 22.

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 4-6.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 4-7.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





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 4-8. *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.

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	estType bRequest		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	wValue wIndex		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 4-9. 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





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
ld_prog_start	00h	start_address		end_address		Init FLASH programming
01h	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.

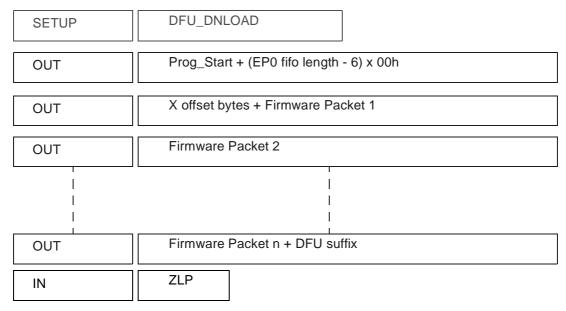
4.6.1.2 Firmware

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. $X = \text{start_address}$ [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.

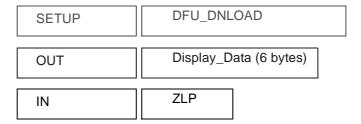
This operation is performed in 2 steps:

- 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.



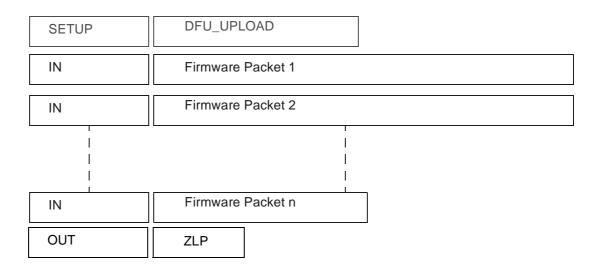
Command Identifier	data[0]	data[1]	data[2]	data[3]	data[4]	Description
	00h					Display FLASH Data
ld_display_data 03h	01h	start_a	address	end_a	ddress	Blank Check in FLASH
3311	02h					Display EEPROM Data

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).

SETUP	DFU_DNLOAD				
OUT	Read_command (2 bytes)				
IN	ZLP				

Command Identifier	data[0]	data[1]	data[2]	data[3]	data[4]	Description
		00h				Read Bootloader Version
	00h	01h				Read Device boot ID1
		02h				Read Device boot ID2
ld_read_command 05h	01h	30h				Read Manufacturer Code
00		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
ld_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.

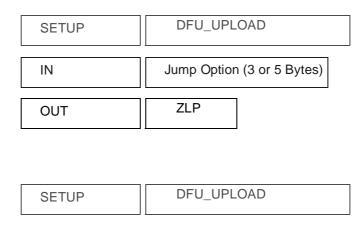
To start the application, the Host sends a DFU_DNLOAD request with the specified application start type in the data field (3 or 5 bytes).

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
Id_write_command		00h				Hardware reset
04h	03h	01h	ado	dress		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

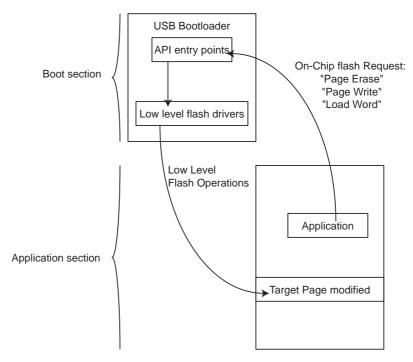
The AT90USB USB bootloader is located in the boot section of the on-chip flash memory, meanwhile the bootloader section is the unique memory location allowed to execute on-chip flash memory write operations (SPM instruction is decoded only in this section).

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 24.





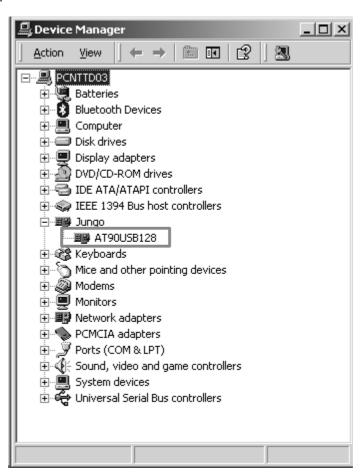
7. Using the USB bootloader for In System Programming

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

The following steps should be completed to allow the device starting DFU mode, and load the firmware file:

- 1. Install Flip software (Flip version 3.0 or above is required).
- 2. Connect the AT90USB USB interface to a PC.
- 3. Push the HWB (Hardware Bootloader) button.
- 4. Push the RST (Reset) button.
- 5. Release the RST button.
- 6. Release the HWB button.
- 7. If your hardware conditions explained above are correct, a new device detection wizard will be displayed. Please follow the instructions (the INF file is located in the USB subdirectory from Flip installation: "install path:\ATMEL\FLIP\FLIPx.x.x.\usb").
- 8. Check the Device Manager, and you should see the same icon (Jungo[®] icon) as shown in the figure below. If not start again from the step 2.

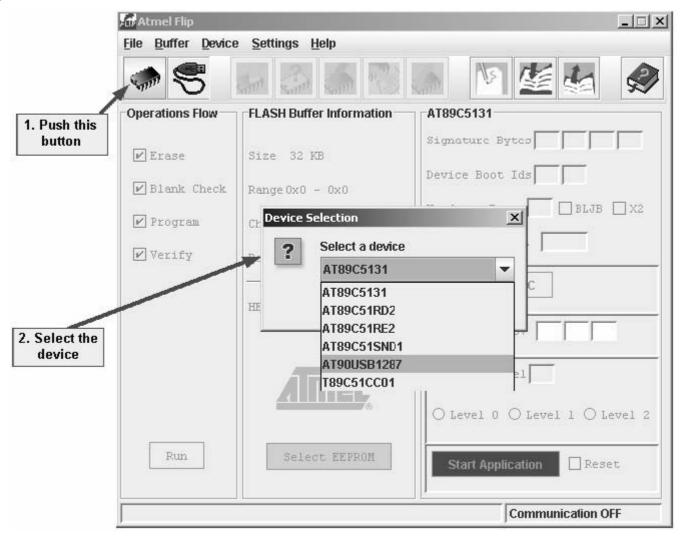
Figure 7-1. Device Manager



Once your device is in DFU mode, launch the Flip software and follow the instructions explained below:

1. Select AT90USB device

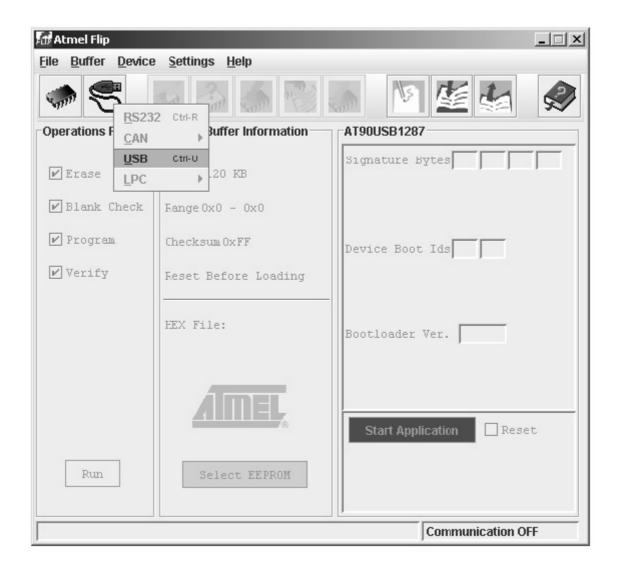
Figure 7-2. Device Selection





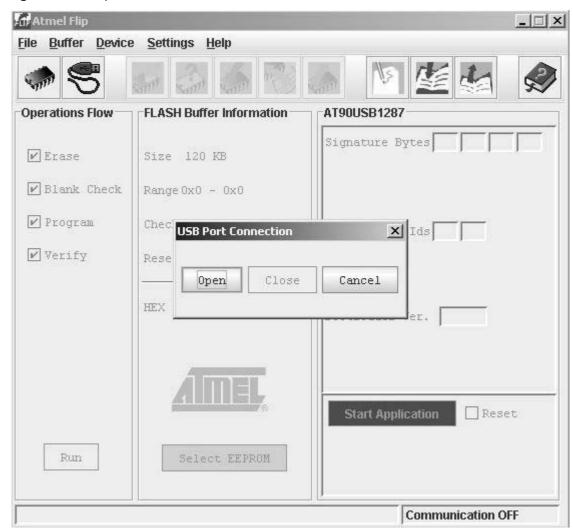
2. Select USB as communication mode

Figure 7-3. USB Communication Mode



3. Open the communication

Figure 7-4. Open the USB Communication



- 4. Choose the HEX file to load.
- 5. Program the HEX file (*Check Erase, Program* and *Verify*, then Push *Run* button).

Note: If you do not perform the "Erase" step, you will be prompted for security issues (it means you'll be unable to program the on-chip Flash memory until the part is not erased).



8. Appendix-A

Table 8-1. 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		and address		Init FLASH programming	
	01h	Start_a	address	end_address		Init EEPROM programming	
ld_display_data 03h	00h					Display FLASH Data	
	01h	start_address		end_address		Blank Check in FLASH	
	02h					Display EEPROM Data	
Id_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	
Id_change _base _address 06h	03h	00	"PP" Select "PP" 64kt page number		Select "PP" 64kBytes flash page number		

 Table 8-2.
 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

 Table 8-3.
 USB Parameters

Parameter	Atmel		
Vendor ID	0x03EB		
Product ID	0x2FFB		
Release Number	0x0000		



9. Appendix-B

```
; $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 lowlevel 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
; 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
; 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)
       RAMPZ, R18
 OUT
       R20,$05
               ; (1<<PGWRT) + (1<<SPMEN))
 LDI
 OUT SPMCSR, R20; argument 2 decides function (r18)
 SPM
               ;Store program memory
       WAIT_SPMEN ; Wait for SPMEN flag cleared
 RCALL
       flash_rww_enable
 RCALL
 RET
; NAME: flash_page_erase
;-----
       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
; 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
      R30,R16
               ; move adress to z pointer (R31=ZH R30=ZL)
 MOV
 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
       flash_rww_enable CAUTION DO NOT ACTIVATE HERE or
 ;RCALL
        you will loose the entire page buffer content !!!
 ;
 RET
```

```
; 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:
     WAIT_SPMEN ; Wait for SPMEN flag cleared
 MOV
      R31,R17
 MOV
      R30,R16
             ; move adress to z pointer (R31=ZH R30=ZL)
      RAMPZ, R18
 OUT
 LDI
      R20,$03
             ; (1<<PGERS) + (1<<SPMEN)))
 OUTSPMCSR, R20; argument 2 decides function (r18)
             ;Store program memory
 SPM
      WAIT SPMEN ; Wait for SPMEN flag cleared
 RCALL
 RCALL
      flash_rww_enable
 RET
; 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
      R20,$11
             ; (1<<WWSRE) + (1<<SPMEN)))
 OUT SPMCSR, R20; argument 2 decides function (r18)
 SPM
              ;Store program memory
      WAIT SPMEN ; Wait for SPMEN flag cleared
 RJMP
```





```
; NAME: flash read sig
:-----
; PARAMS:
; Return: R16: signature value
;------
; PURPOSE: Read harware signature byte. THe byte is selected trought the
; 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
              ; (1<<SPMEN) | (1<<SIGRD))
      R20,$21
 LDI
 OUT SPMCSR, R20; argument 2 decides function (r18)
 LPM
              ;Store program memory
      R16, R0
               ;Store return value (1byte->R16 register)
 MOV
      WAIT SPMEN ; Wait for SPMEN flag cleared
 RJMP
; NAME: flash_read_fuse
;------
; Return: R16: fuse value
; PURPOSE: Read fuse byte. The fuse byte is elected through the address
; 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
              ; (1<<SPMEN) | (1<<BLBSET))
      R20,$09
 LDI
 OUT SPMCSR, R20; argument 2 decides function (r18)
 LPM
              ;Store program memory
 MOV
                 ;Store return value (1byte->R16 register)
      R16, R0
```

```
RJMP
     WAIT SPMEN ; Wait for SPMEN flag cleared
* 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
*-----
* This function allows to load a word in the temporary flash buffer.
*-----
* EXAMPLE:
* fill temp buffer(data16, address);
*-----
* 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
    R0,R17
 MOV
           ; move data16 to reg 0 and 1
     R1,R16
 MOV
     R20, (1<<SPMEN)
 LDI
 OUT SPMCSR, R20; r18 decides function
 SPM
           ; Store program memory
 RJMP
     WAIT SPMEN ; Wait for SPMEN flag cleared
; NAME: lock wr bits
;-----
; PARAMS: R16: value to write
                 -----
; PURPOSE:
lock_wr_bits:
```





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

30

END



Atmel Corporation

2325 Orchard Parkway San Jose, CA 95131, USA 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

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, USA Tel: 1(408) 441-0311 Fax: 1(408) 436-4314

Microcontrollers

2325 Orchard Parkway San Jose, CA 95131, USA 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, USA

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, USA

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

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 providedotherwise, 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.

© Atmel Corporation 2006. All rights reserved. Atmel[®], logo and combinations thereof, are registered trademarks, and Everywhere You Are[®] are the trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.

