

USB Mass Storage Device Implementation with the AT89C5131A

*THE ERA OF THE FLOPPY DISK IS OVER!
THEY'RE TOO SLOW, TOO FRAGILE, NOT
ENOUGH CAPACITY...*

*USB IS NOW ON EVERY PC UNDER
WINDOWS OR LINUX, ON EVERY MAC.
AND YOU CAN VERY EASILY FIND USB
MASS STORAGE KEYS THAT ARE
AUTOMATICALLY RECOGNIZED BY
THE OPERATING SYSTEM, WITH HIGH
CAPACITIES AND EXCELLENT SPEED
PERFORMANCES. THIS ARTICLE
EXPLAINS THE ADVANTAGES OF
THE USB MASS STORAGE DEVICE
IMPLEMENTATION WITH THE
AT89C5131A.*

By: Laurent Guilhaumon, Atmel

The USB Mass Storage device has replaced the floppy disk and is used in many applications such as:

- USB mouse for laptop with Mass Storage integrated
- External Hard Drive
- Access to internal memory of a mobile phone or MP3 player
- Firmware Upgrade of a system
- Small capacity memory to store private information or preferences (Smart Card Reader, Control Access, Electronic Wallet...)

The aim of this article is to describe how to implement the USB Mass Storage class (with an example on the AT89C5131A product) and how the firmware delivered by Atmel works in this application. The microcontroller features a 32 Kbytes Flash, and the stack represents only 5 Kbytes of executable code without any memory management. This leaves enough program memory to implement many other application functionalities. It can even operate on the fully compatible AT89C5130 featuring 16 Kbytes of Flash.

What is USB Mass Storage

Inside the computer (figure 1), a lot of Mass Storage peripherals (the hard drive for example) can be accessed using SCSI command set. SCSI means "Small Computer System Interface". The USB Mass Storage class defines the USB wrapper of the SCSI commands. USB Mass Storage is no more than the SCSI command set for the USB. In order to access several memories at the same time, for a multiple-memory card reader for example, the Host selects the LUN (Logical Unit Number) whose SCSI command is addressed to.

Operating Systems and USB Mass Storage

Currently, most Operating Systems support USB and have built-in drivers to support the USB Mass Storage. This is very convenient for the final user who can now bring any file from his own computer to any other one.

For every Operating System, a USB Mass storage device is composed of:

- the Default Control Endpoint (0), to perform the enumeration process, to determine the number of LUN (Logical Unit Number), and to perform a reset recovery in case of mass storage error.
- a Bulk OUT Endpoint, used by the Host Controller to send the wrapped SCSI commands and to send the data to write inside the memory.
- a Bulk IN Endpoint, used by the Host Controller to read the wrapped SCSI status and to read data from the memory.

USB Mass Storage Device Firmware Stack Architecture

The USB Mass Storage Device Firmware architecture (figure 2) shows the architecture of the USB Mass Storage firmware stack.

The standard enumeration process (USB chapter 9 support) is performed by a generic function. The parameters and the descriptors are specific for each application. The USB Mass Storage class also requires specific request support.

You can see that the commands coming from the USB Host controller are first un-wrapped by the USB Mass Storage Protocol Decoder function before entering into the SCSI decoder. Each SCSI command is then decoded and transmitted to the appropriate memory through

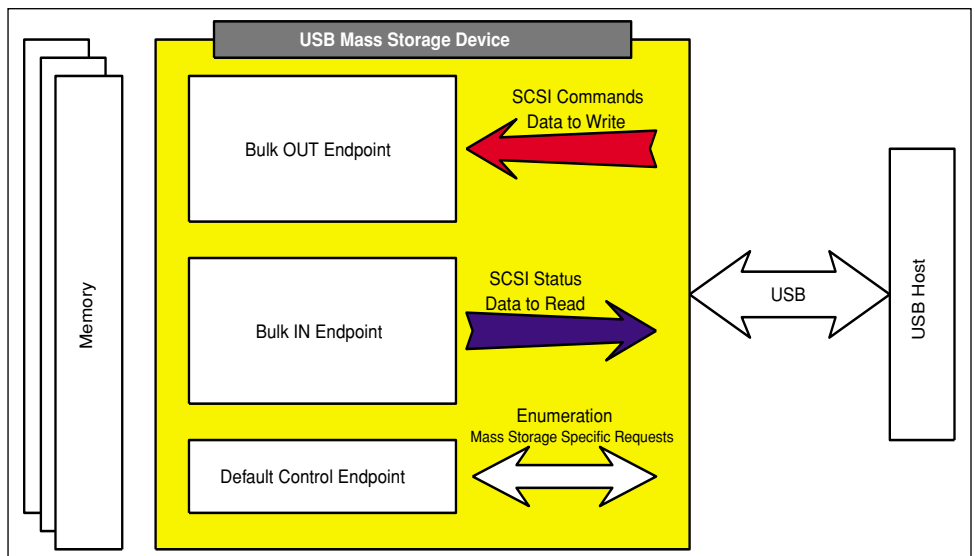


Figure 1: A Mass Storage Device seen by a USB Host

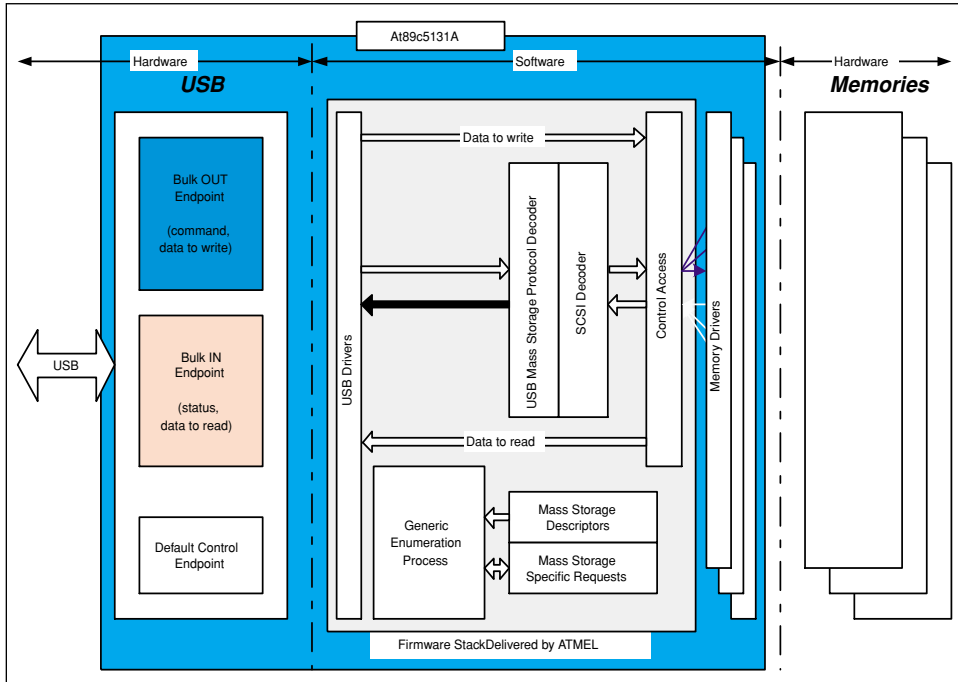
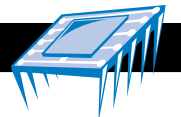


Figure 2: USB Mass Storage Device Firmware architecture

a command set (Read, Write, is memory present, is memory write protected...).

The memory answers are converted in SCSI status before being wrapped in USB CSW (Command Status Wrapper) and sent to the USB Host controller. Remember that because the USB bus is a one master bus (the

USB Host), each data transfer is initiated by the USB Host, following a specific Command-Data-Status flow. For more details regarding the Mass Storage flow, please refer to *USB Mass Storage Overview and USB Mass Storage Bulk-Only specification* on the USB-IF web site (www.usb.org).

Memory Organization Overview

From a USB Host point of view, the USB memory is organized in logical blocks (sectors) of 512 Bytes. The Host always addresses the memory with:

- the logical sector number
- the number of contiguous sectors to read/write.

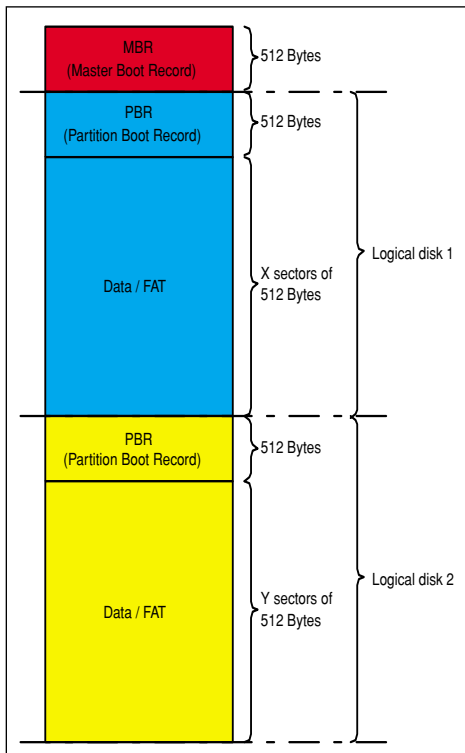


Figure 3: Memory logical organization overview

The MBR (Master Boot Record) is always located at address 0. It contains information regarding the whole memory and the number of logical partitions inside. In the case of a memory that has only 1 partition, the MBR is optional and can be replaced directly by the PBR (Partition Boot Record) that contains information regarding the logical partition and the entry point of the FAT (File Allocation Table).

Memory Management

As you can see on the USB Mass Storage Device Firmware architecture, each memory is interfaced to the Atmel firmware by a specific memory driver.

The USB Mass Storage Device firmware uses a set of 8 functions to read from the memory, to write into the memory, to determine the storage capacity, to test if the memory is present, busy or write protected. These 8 functions have to be implemented in order to support a memory.

The behavior of these functions is described in the application note. You will also find in annex of this application note, the algorithms to implement for the direct reading from USB to the memory and for the direct writing from the memory to USB. These algorithms support the standard mode and the Ping-Pong mode of the AT89C5131A endpoints. The Mass Storage Device performances will be increased if the Ping-Pong endpoints are used.

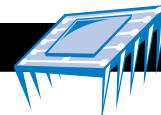
Integration of a Memory

The integration of a memory on the USB Mass Storage stack is performed in the `conf_access.h` file. The corresponding LUN has to be first set to ENABLE and the corresponding functions have to be defined. The USB Mass Storage stack supports up to 8 different LUN.

Here is an example (below) with the virtual memory sets as LUN_0:

```
// Active the Logical Unit
#define LUN_0          ENABLE
#define LUN_1          DISABLE
#define LUN_2          DISABLE
#define LUN_3          DISABLE
#define LUN_4          DISABLE
#define LUN_5          DISABLE
#define LUN_6          DISABLE
#define LUN_7          DISABLE

// LUN 0 DEFINE
#define LUN_0_INCLUDE "lib_mem\virtual_mem\virtual_mem.h"
#define Lun_0_test_unit_ready()    virtual_test_unit_ready()
#define Lun_0_read_capacity(nb_sect) virtual_read_capacity(nb_sect)
#define Lun_0_wr_protect()         virtual_wr_protect()
#define Lun_0_removal()           virtual_removal()
#define Lun_0_read_10(ad, sec)    virtual_read_10(ad, sec)
#define Lun_0_usb_read()          virtual_usb_read()
#define Lun_0_write_10(ad, sec)   virtual_write_10(ad, sec)
#define Lun_0_usb_write()         virtual_usb_write()
```



USB Configuration and Customizing

USB Clock

The USB clock is automatically configured using the FOSC value.

The FOSC value is the oscillator frequency, in kHz, and is defined in the *config.h* file. For example, the value of FOSC is set to 16000 for an oscillator frequency of 16 MHz.

Enumeration Parameters

The USB Mass Storage Device firmware stack is fully customizable. You can use your own strings of characters that will appear on the PC when plugging the USB Mass Storage device.

Vendor ID/Product ID

For an application based on the USB Mass Storage stack, the Vendor ID and the Product ID have to be changed with the ID assigned to the manufacturer by USB-IF.

The Vendor ID and Product ID are defined in the *conf_usb.h* file.

For this demo, these parameters are:

```
#define VENDOR_ID      0xEB03 // Atmel vendor ID = 03EBh
#define PRODUCT_ID    0x1320 // Product ID: 2013h = Mass Storage
```

Strings

The manufacturer can change the USB strings in the *conf_usb.h* file.

These strings are:

- Manufacturer Name
- Product Name
- Serial Number (at least 12 characters)

Do not forget to adapt the string length definitions according to the modifications:

```
#define USB_MN_LENGTH 5
#define USB_PN_LENGTH 28
#define USB_SN_LENGTH 13
```

Endpoint

If an application uses different Endpoints that those used in the demo, this can be done in the *conf_usb.h* file by changing these define:

```
#define EP_MS_IN      4
#define EP_MS_OUT     5
```

In such a case, do not forget to specify in the *conf_usb.h* file if the endpoints to use support the Ping-Pong mode or not:

```
#define NO_SUPPORT_USB_PING_PONG
```

or no define.

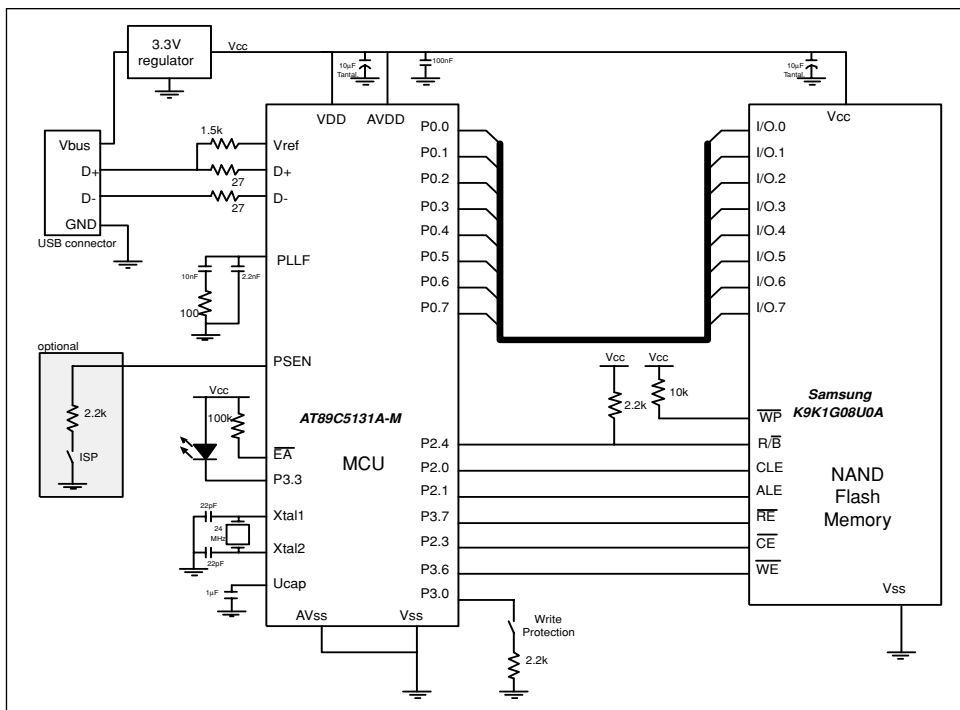


Figure 4: Implementation example with the AT89C5131A and a NAND Flash Memory

Implementation Example with a Nand Flash

The schematic above (figure 4) is an example of the USB Mass Storage hardware implementation supported by the Atmel firmware stack.

This example is based around the AT89C5131A micro-controller. This micro-controller integrates a USB macro which supports the Ping-Pong mode on several endpoints.

Conclusion

Building a USB Mass Storage key has never been so easy! Just customize the firmware, add your own memory driver (if not already supported by Atmel), plug, and play! This is also the easiest way to integrate in your product the USB Mass Storage as a complementary feature. The AT89C5131A introduces 6 versatile endpoints in addition of the Default Control Endpoint, and only 2 are used by the Mass Storage Class. You can declare additional interfaces and add USB Mass Storage to a mouse, a Bluetooth dongle, etc. No need of new PC driver.

More details can be found on the related application note available on the Atmel web site (www.atmel.com). The Mass Storage Device firmware stack for the AT89C5131 is also available on request.

STAFF BOX

Publisher: Glenn ImObersteg
Glenn@convergencepromotions.com

Managing Editor: Bob Henderson
Bob@convergencepromotions.com

Technical Editor: Markus Levy
Markus@convergencepromotions.com

Sales Manager: Mike Miller
mike@convergencepromotions.com

Production Manager: Dave Ramos
dbyd@garlic.com

Designer: Judy Gosse
Judy@convergencepromotions.com

This issue of the Atmel Applications Journal is published by Convergence Promotions. No portion of this publication may be reproduced in part or in whole without express permission, in writing, from the publisher. The contents of this publication are Copyright © Atmel Corporation 2005. All rights reserved. Atmel®, logo and combinations thereof are registered trademarks, and Everywhere You Are™ is the trademark of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others. All product names, specifications, prices and other information are subject to change without notice. The publisher takes no responsibility for false or misleading information or errors or omissions. Any comments may be addressed to the publisher, Glenn ImObersteg at glenn@convergencepromotions.com, or +1 (925) 516-6227.