Fujitsu Microelectronics Europe
Application Note

MCU-AN-3000243-E-V10

F²MC-16LX FAMILY

16-BIT MICROCONTROLLER

MB96330

USB MINI-HOST MASS STORAGE CLASS

APPLICATION NOTE

Revision History

Date	Issue
2008-06-30	V0.1; MWi; First version

This document contains 72 pages.

Warranty and Disclaimer

To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH restricts its warranties and its liability for **all products delivered free of charge** (eg. software include or header files, application examples, target boards, evaluation boards, engineering samples of IC's etc.), its performance and any consequential damages, on the use of the Product in accordance with (i) the terms of the License Agreement and the Sale and Purchase Agreement under which agreements the Product has been delivered, (ii) the technical descriptions and (iii) all accompanying written materials. In addition, to the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH disclaims all warranties and liabilities for the performance of the Product and any consequential damages in cases of unauthorised decompiling and/or reverse engineering and/or disassembling. **Note, all these products are intended and must only be used in an evaluation laboratory environment**.

1. Fujitsu Microelectronics Europe GmbH warrants that the Product will perform substantially in accordance with the accompanying written materials for a period of 90 days form the date of receipt by the customer. Concerning the hardware components of the Product, Fujitsu Microelectronics Europe GmbH warrants that the Product will be free from defects in material and workmanship under use and service as specified in the accompanying written materials for a duration of 1 year from the date of receipt by the customer.

2. Should a Product turn out to be defect, Fujitsu Microelectronics Europe GmbH´s entire liability and the customer´s exclusive remedy shall be, at Fujitsu Microelectronics Europe GmbH´s sole discretion, either return of the purchase price and the license fee, or replacement of the Product or parts thereof, if the Product is returned to Fujitsu Microelectronics Europe GmbH in original packing and without further defects resulting from the customer´s use or the transport. However, this warranty is excluded if the defect has resulted from an accident not attributable to Fujitsu Microelectronics Europe GmbH, or abuse or misapplication attributable to the customer or any other third party not relating to Fujitsu Microelectronics Europe GmbH.

3. To the maximum extent permitted by applicable law Fujitsu Microelectronics Europe GmbH disclaims all other warranties, whether expressed or implied, in particular, but not limited to, warranties of merchantability and fitness for a particular purpose for which the Product is not designated.

4. To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH´s and its suppliers´ liability is restricted to intention and gross negligence.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES

To the maximum extent permitted by applicable law, in no event shall Fujitsu Microelectronics Europe GmbH and its suppliers be liable for any damages whatsoever (including but without limitation, consequential and/or indirect damages for personal injury, assets of substantial value, loss of profits, interruption of business operation, loss of information, or any other monetary or pecuniary loss) arising from the use of the Product.

Should one of the above stipulations be or become invalid and/or unenforceable, the remaining stipulations shall stay in full effect

Contents

1 Introduction

This User Guide describes the implementation of a USB Mass Storage Class driver and gives an idea how to use the driver from application software. A short menu based user dialog allows to read and write files and list directory entries from a FAT16 formated USB device.

The software is developed primarily for Fujitsu MB96F338 microcontroller. It utilizes USB Mini Host functionality of MB96F338 processor. The driver makes use of the calls and interface provided by Thesycon's™ Fujitsu USB Mini Host Application Package "FUMA" that can be download from the Thesycon's website:
http://www.thesycon.de/eng/fuma.shtml

Also in this application note a FAT16 file system is used what can be downloaded from the developer's Website:
http://elm-chan.org/fsw/ff/00index_e.html.

The complete project with the complete source code can be downloaded at:
http://mcu.emea.fujitsu.com/mcu_product/mcu_all_software.htm
(*96330_usb_mass_storage_demo*)
Please always look for latest version. The source codes in this application note may be older than the latest software example.

2 Overview

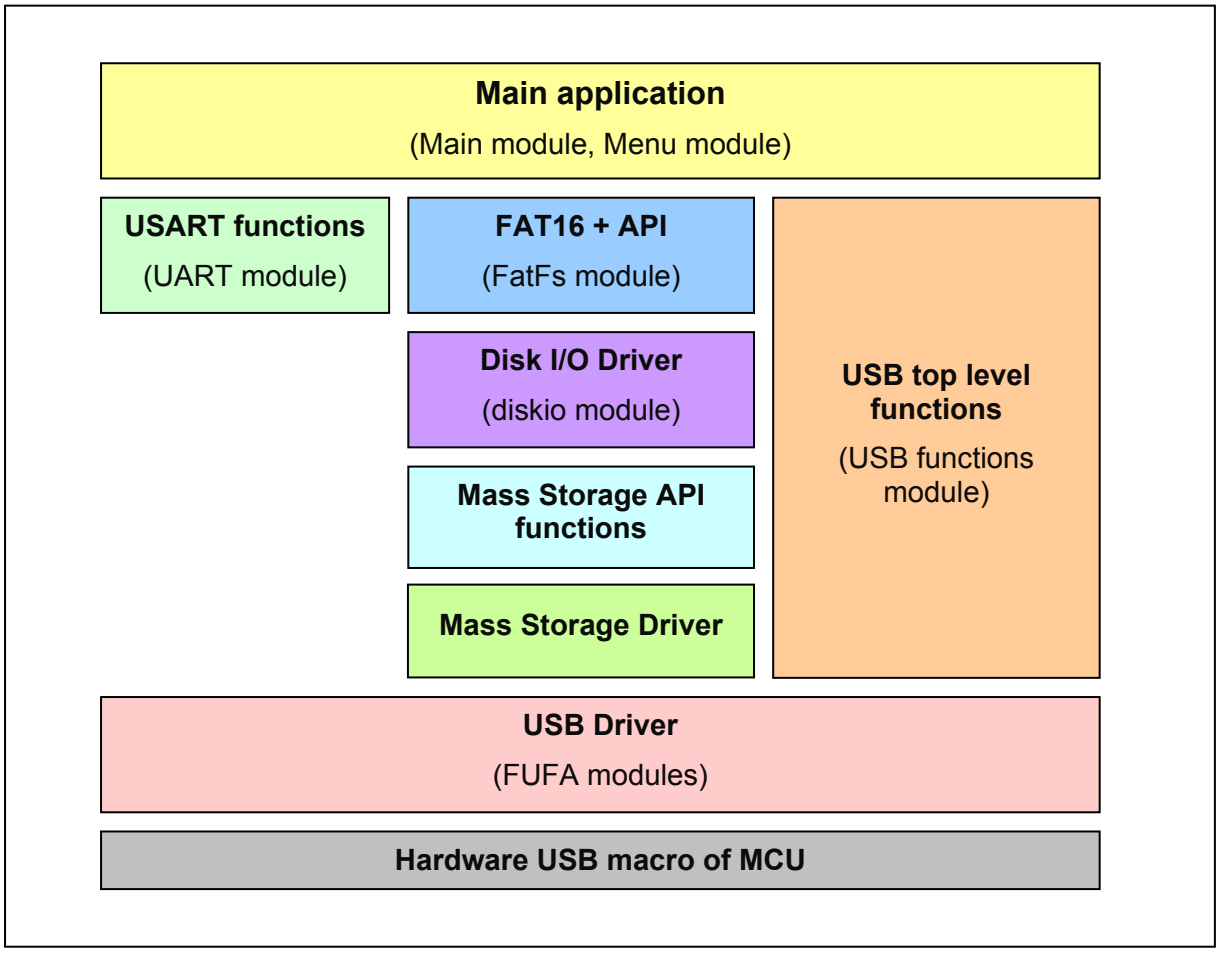
OVERVIEW OF THE USB MINI HOST MASS STORAGE CLASS PROJECT

2.1 Project Parts

The USB mini host mass storage project consist of the following software groups:

- USB mini host driver developed by Thesycon™ (*FUMA library*)
- FAT16 file system developed and provided by ELM-ChaN (*FatFs*)
- Mass storage class and API developed by Fujitsu
- Application (Menu and user dialog)

2.1.1 Project Block Diagram



2.1.2 USB Driver Files

The following files were taken from the Thesycon™ FUMA package and are copied in the subfolder *Src\usb*:

- *config.h* Application specific definitions (class, endpoint, etc.)
- *DLIST.H* Double linked list `struct` definitions
- *hw_support.c* Device specific code (e.g. Callback, USART, ...)
- *hw_support.h* Device specific definitions and prototypes
- *osal.h* Definitions for operating system abstraction layer
- *osal_impl.c* Code for operating system abstraction layer
- *osal_impl.h* Include file for operating system abstraction layer
- *t_dbgprint.c* Debug print support code
- *t_dbgprint.h* Debug print support include
- *t_trc_printf.c* `printf` trace function code implementation
- *t_trc_printf.h* `printf` trace function code implementation include
- *t_Utils.h* Utility include (e.g. `MIN`, `MAX`, `SWAP`, …)
- *tal_defs.h* Type definitions (e.g. `UCHAR`, `UINT`, ...)
- *umh_dbgprint.c* Debug print support for USB driver
- *umh_dbgprint.h* Debug print support for USB driver include
- *umh_device.c* Device specific driver routines
- *umh_device.h* Device specific driver routines include
- *umh_ep.c* Host endpoint object
- *umh_ep.h* Host endpoint object include
- *umh_event.c* USB event handler
- *umh_event.h* USB event handler include
- *umh_glob.h* USB host global definitions
- *umh_hal.c* USB host hardware abstraction layer (HAL)
- *umh_hal.h* USB host HAL include
- *umh_hal_dbgprint.c* Debug print support for USB host HAL
- *umh_hal_dbgprint.h* Debug print support for USB host HAL include
- *umh_haldef.h* USB host HAL definitions
- *umh_host.c* USB host token scheduler
- *umh_host.h* USB host token scheduler include
- *umh_lib.c* USB host main library module
- *umh_reg_def.h* Global definitions whether 16LX or 16FX is used
- *umh_status.h* Status code definitions of USB host library
- *UsbMiniHost.h* USB host library API
- *usbspec.h* Configuration and descriptor definitions
- *verdef.h* Copyright and version information

Also the following files from the FUFA package were copied to the *Src* folder:

- *app_global.h* Global include for application
- *app_hid_dbgprint.c* Implementation of debug print for application
- *app_hid_dbgprint.h* Include of debug print for application
- *app_hid_global.h* Include for application

2.1.3 FAT16 File System and Disk-I/O Driver

The following files for the FAT16 file system were used in the subfolder *Src\Filesystem*:

- *diskio.c* Disk I/O operation code
- *diskio.h* Disk I/O operation include

- *ff.c* FAT16 user API
- *ff.h* FAT16 user API include
- *integer.h* FAT16 definitions and global include

2.1.4 Mass Storage Class Driver

The mass storage class consists of the following files and are also contained in the subfolder *Src\mass_storage*:

- *storage.c* Mass storage class with SCSI functions
- *storage.h* Mass storage class include
- *storage_API.c* Mass storage class API
- *storage_API.h* Mass storage class API include
- *msc_config.h* Mass storage class definitions

2.1.5 Main Application and remaining USB Functions

In the *Src* folder the main application and some remaining USB functions are located:

- *MAIN.C* Main code, calls `MenuInit()` in *menu.c*
- *mb96338us.asm* Device specific I/O register address definitions
- *mb96338us.h* Device specific I/O register bit field definitions
- *menu.c* Actual main application including user dialog
- *menu.h* Main application include
- *START.ASM* Start and initialization code
- *uart.c* USART specific routines (`puts`, `getch`, …)
- *uart.h* USART specific include
- *usb_functions.c* USB specific application functions
- *usb_functions.h* USB specific application functions include
- *vectors.c* IRQ vector definition and initialization

Please note:

The module *usb_functions* and its include file contain USB callback functions and USB handlers according to the architecture of the FUMA USB driver which is close to the application main functions. These functions are USB device class dependant and have to be modified by the user for other purposes than using USB mass storage classes.

2.2 Restrictions

Due to the used components and to preserve a “not too complex” project, the discussed application has some restrictions as follows.

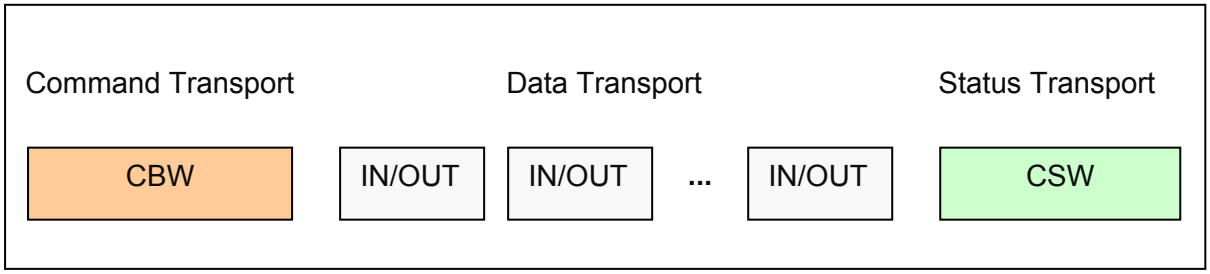
- No error handling (e.g. 8.3 filename violations) in the menu dialog
- Changing the root directory is not possible
- Using backspace in the dialog leads to undefined behavior

3 Mass Storage Class

DESCRIPTION OF THE MASS STORAGE CLASS

3.1 Transmission Protocol

For the mass storage class the Bulk-Only-Transport (*BOT*) is used. The protocol looks like the following illustration.



3.1.1 Command Block Wrapper (*CBW*)

The *CBW* consists of the following parts:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 – 3	dCBWSignature = 0x43425355							
4 – 7	dCBWTag							
8 – 11	dCBWDataTransferLength							
12	dCBWFlags							
13	*reserved*				bCBWLUN			
14	*reserved*			dCBWCBLength				
15 – 30	CBWCB							

The `dCBWSignature` is always `0x43425355`.

The `dCBWTag` is used for identifying the prior sent *CBW* to a *CSW*.

The `dCBWDataTransferLength` contains the number of data bytes to be transported.

The `dCBWFlags` indicates data transfer direction. Is bit 7 == 0, data is transferred from host to device.

The `dCBWLUN` contains the logical entity of the external disk (e.g. USB stick). Here it is always 0.

The `bCBWCBLength` is the actual length of the *CBW*.

The `CBWCB` holds the SCSI command for the external mass storage device (e.g. USB stick). Unused bytes remain `0x00`.

3.1.2 Command Status Wrapper (*CSW*)

The *CSW* consists of the following parts:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 – 3	dCSWSIgnature = 0x53425355							
4 – 7	dCSWTag							
8 – 11	dCSWDataReside							
12	dCSWStatus							

The `dCSWSignature` is always `0x53425355`.

The `dCSWTag` identifies a former sent *CBW*.

The `dCSWDataReside` shows the difference between `dCBWDataLengthTransfer` and actual data for or from the device.

The `dCSWStatus` shows the result of a command. `0` == command successful, `1` == command error, `2` == fatal error (USB reset should follow).

3.1.3 SCSI Commands

The SCSI commands are part of the `CBW` in the `CBWCB` field. The following table shows the commands which are used in this application:

Command	Code
Inquiry	0x12
Mode Sense	0x5A
Read(10)	0x28
Read Capacity	0x25
Request Sense	0x03
Test Unit Ready	0x00
Write(10)	0x2A

The definitions of all commands can be found in *storage_API.h* (7.11).

3.1.3.1 Inquiry

Using the *Inquiry* command, the attributes of a device is read. The attributes contain e.g. device identifications, SCSI command version, kind of disk, etc.

The *Inquiry* command consists of the following parts:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command Code: 0x12							
1	Number of logical unit to be accessed			*reserved*		Enable Vital Product Data (EVPD) = 0x00		
2	Page Code: 0x00							
3	*reserved*							
4	Allocation Length (Indicates how many *Inquiry* data bytes should be returned)							
5 – 11	*reserved*							

The *Inquiry* response within data part consists of the following:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	*reserved*			Peripheral Device Type (0x00 indicates direct access to device)				
1	Portable disk device	*reserved*						
2	ISO Version: 0x00		ECMA Version: 0x00			ANSI Version: 0x00		
3	*reserved*				Response Data Format: 0x01			
4	Additional Length (Number of bytes which follow)							
5 – 7	*reserved*							
8 – 15	Manufacturer Information							
16 – 31	Product Information							
32 – 35	Product Revision							

3.1.3.2 Mode Sense

With the *Mode Sense* command are used to read out further device settings and attributes. It contains e.g. information of write protection.

The command is built as follows:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command Code: 0x1A							
1	Number of logical unit to be accessed			*reserved*	DBD = 0x00	*reserved*		
2	Page Control*	Page Code: 0x00 = Header, 0x3F = all accessible pages						
3 – 6	*reserved*							
7 – 8	Allocation Length (Number of bytes reserved for answer)							
9 – 11	*reserved*							

* 00: return current Values
01: return changeable Values
10: return default Values
11: return saved Values

The *Mode Sense* response within data part consists of the following:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Mode Data Length (Number of bytes which follows)							
1	Medium Type Code (Standard = 0x00)							
2	WP	*reserved*		DPOFUA = 00b		*reserved*		
4 – 7	*reserved*							

3.1.3.3 Read(10)

With the *Read(10)* command, the connected mass storage device data can be read out.

The structure of the command is as follows:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command Code: 0x28							
1	LUN: Logical unit of disk			DPO = 0	FUA = 0	*reserved*		RelAdr = 0
2 – 5	LBA (Address of the logical block to read out)							
6	*reserved*							
7 – 8	Number of blocks to be read (1 block = 512 bytes)							
9 – 11	*reserved*							

The data part is contained of 64 bytes. If a block should be read, 8 x 64 bytes are transmitted before *CSW* request.

3.1.3.4 Read Capacity

Read Capacity is used to determine the size of the connected mass storage device. The response is the number of the last logical block and the number of bytes per block.

The command structure is as follows:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command Code: 0x25							
1	LUN: Logical unit of disk			*reserved*				RelAdr = 0
2 – 5	LBA (Address of the logical block to read out)							
6 - 7	*reserved*							
7 – 8	LUN: Logical entity of disk							PMI = 0
9	Control							

The response is as follows:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 - 3	Last logical block address							
4 – 7	Block length in bytes							

3.1.3.5 Request Sense

If the *CSW* response shows an error, the *Request Sense* command can be used to get detailed information about the error. It has the following structure in the *CBW*:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command Code: 0x03							
1	LUN: Logical unit of disk			*reserved*				
2 – 3	*reserved*							
4	Allocation Length (Number of bytes reserved for answer)							
5 – 11	*reserved*							

The response is shown in the following table:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Validity Bit	Error Code (0x70 if error is related to last command)						
1	*reserved*							
2	*reserved*				Sense Key Code			
3 – 6	Additional Length (Number of bytes which follow)							
5 – 11	*reserved*							

3.1.3.6 Test Unit Ready

The *Test Unit Ready* command is used to see the disk device's status and if it is read to been written to or read from. If the device is not ready a `1` is responded in *CSW*:*Status*.

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command Code: `0x00`							
1	LUN: Logical unit of disk			*reserved*				
2 – 11	*reserved*							

3.1.3.7 Write(10)

The *Write(10)* is the pendant to the *Read(10)* command. The following table shows the structure of this command:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command Code: `0x2A`							
1	LUN: Logical unit of disk			DPO = 0	FUA = 0	*reserved*		RelAdr = 0
2 – 5	LBA (Address of the logical block to read out)							
6	*reserved*							
7 – 8	Number of blocks to be read (1 block = 512 bytes)							
9 – 11	*reserved*							

The data part is contains 64 bytes. If a block of 512 bytes should be written, 8 x 64 bytes are transmitted before *CSW* request.

3.2 Transfer Details

For the transfers the *CBW* and *CSW* parts color coding is used as in the following tables:

CBW Parts	
	CBWSignature
	CBWTag
	CBWDataTransferLength
	CBWFlags
	CBWLUN
	CBWCBLength
	CBWCB

CSW Parts	
	CSWSignature
	CSWTag
	CSWDataReside
	CSWStatus

3.2.1 Inquiry (0x12)

CBW Data															
55	53	42	43	12	34	56	78	24	00	00	00	80	00	06	12
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Inquiry Data															
00	80	00	01	31	00	00	00	46	75	6A	69	74	73	75	20
4D	61	73	73	53	6F	72	61	67	65	20	20	20	20	20	20
31	2E	30	30												

Inquiry Data Parts	
	Direct access to medium: 0x00
	If Bit 7 is set: Portable disk drive
	ISO/ECMA/ANSI versions
	Response data format
	Additional length: Number of bytes which are following
	Reserved
	Manufacturer string: Fujitsu
	Product information: MassStorage
	Product revision: 1.00

CSW Data												
55	53	42	53	12	34	56	78	00	00	00	00	00

3.2.2 Read Capacity (0x25)

CBW Data															
55	53	42	43	12	34	56	78	08	00	00	00	80	00	0A	25
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Read Capacity Data							
00	07	9D	FF	00	00	02	00

Inquiry Data Parts	
	Last logical block address: 0x00079DFF = 499199d
	Block length: 0x200 = 512d

CSW Data												
55	53	42	53	12	34	56	78	00	00	00	00	00

CSW Status	
	0x00 == passed
	0x01 == failed
	0x10 == phase error

3.2.3 Request Sense (0x03)

CBW Data															
55	53	42	43	12	34	56	78	24	00	00	00	80	00	0C	03
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Request Sense Data															
70	00	05	00	00	00	00	0A	00	00	00	00	24	00	00	00
00	00														

Request Sense Data Parts	
	Bit 7 indicates error, remaining bits are error code
	Reserved
	Bit 7-4 reserved, Bit 3-0 Sense Key
	Additional information about error block address (not always supported)
	Additional Sense Length: 10 further bytes
	Additional Sense Code
	Additional Sense Code Qualifier

CSW Data												
55	53	42	53	12	34	56	78	00	00	00	00	00

3.2.4 Mode Sense (0x1A)

CBW Data															
55	53	42	43	12	34	56	78	24	00	00	00	80	00	06	1A
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Additional CBW Data	
	Logical unit number
	Page Control (bit 7-6): 00b: Return current values 01b: Return changeable values 10b: Return default values 11b: Return saved values Page Code (bit 5-0): 0x00: Header 0x3F: All accessible sides

Mode Sense Data															
00	0E	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Inquiry Data Parts	
	Number of bytes following (15)
	Type of medium: 0x00: Default medium (e.g. USB stick) 0x1E: 720KB formatted flexible disk 0x93: 1.25MB formatted flexible disk 0x94: 1.44MB formatted flexible disk
	Bit 7: 0 == writeable, 1 == write protected Bit 6-0: *reserved*
	reserved

CSW Data												
55	53	42	53	12	34	56	78	00	00	00	00	00

3.2.5 Test Unit Ready (0x00)

CBW Data															
55	53	42	43	12	34	56	78	24	00	00	00	80	00	06	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

CSW Data (Device ready)												
55	53	42	53	12	34	56	78	00	00	00	00	00

CSW Data (Device not ready)												
55	53	42	53	12	34	56	78	00	00	00	00	01

3.2.6 Write(10) (0x2A)

CBW Data															
55	53	42	43	12	34	56	78	00	02	00	00	80	00	06	2A
00	00	00	00	C3	00	00	01	00	00	00	00	00	00	00	

Additional CBW Data	
	LUN: Logical unit
	Logical block address (here block #0xC3)
	Number of blocks to be written (here: 1)

Here one block with 512 bytes is written. The USB endpoint holds 64 FIFO bytes, so that 8 packages have to be sent (8 x 64 = 512).

Write Data (64 bytes) package 1 - 8															
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

CSW Data												
55	53	42	53	12	34	56	78	00	00	00	00	00

3.2.7 Read(10) (0x28)

CBW Data															
55	53	42	43	12	34	56	78	00	02	00	00	80	00	06	28
00	00	00	00	C3	00	00	01	00	00	00	00	00	00	00	

Additional CBW Data	
	LUN: Logical unit
	Logical block address (here block #0xC3)
	Number of blocks to be written (here: 1)

Here one block with 512 bytes is read. The USB endpoint holds 64 FIFO bytes, so that 8 packages have to be received (8 x 64 = 512).

Write Data (64 bytes) package 1 - 8															
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

CSW Data												
55	53	42	53	12	34	56	78	00	00	00	00	00

4 Mass Storage Driver

SHORT DESCRIPTION OF THE MASS STORAGE DRIVER

4.1 API functions

The following functions are implemented:

```
int MSD_Read(BYTE *buff, DWORD BlockStart, BYTE BlockCnt);
int MSD_write(BYTE *buff, DWORD BlockStart, BYTE BlockCnt);
int MSD_Read_Capacity(void (*Buff));
int StorageInit(void);
int ModeSense(void);
```

4.1.1 StorageInit

This function should be called at the very first beginning after device enumeration and configuration done by the USB driver.

It is possible that some devices return a `0x01` in the *CSW* respond to the *Inquiry* command for the first time. Therefore the command *Request Sense* is repeated before performing error handling.

Return value is the status (no error == `0x00`).

4.1.2 MSD_Read

The Mass Storage Device Read command is used to read blocks of the medium. The needed parameters are the pointer to the read buffer, the start address of the block, and the number of blocks to be read. Return value is the status (no error == `0x00`).

4.1.3 MSD_Write

The Mass Storage Device Write command is used to write blocks to the medium. The needed parameters are the pointer to the write buffer, the start address of the block, and the number of blocks to be written. Return value is the status (no error == `0x00`).

4.1.4 ModeSense

This function is used to check if the connected device is write protected or not. Return value is the status. Parameter is the global structure `ModeSenseRespond_t`. The write protect bit indicates the protection status after the function call (write protection == `1`).

4.1.5 MS_Read_Capacity

With this function the size of the connected device can be obtained. Parameter is the pointer to the result value variable.

4.2 *CBW* and *CSW*

The *CBW* and *CSW* are declared as a type definition structure `CBW_t` and `CSW_t`. The declaration can be found in *storage.h* (7.9).

The transmission function for *CBW* is `int Send_CBW(STORAGE_INFO_t)` and the reception function for *CSW* is `int Get_CSW(STORAGE_INFO_t)` Both functions can be found in *storage.c*.

The type STORAGE_INFO_t is as follows and can be found in *storage_API.h* (7.11):

```
typedef struct _StorageInfo_t
{
   unsigned char STORAGE_CMD;
   unsigned int  BUFFER_SIZE;
   unsigned char *BUFFER;
   unsigned int  IN_EP;
   unsigned int  OUT_EP;
   unsigned long BLOCK_START;
   unsigned char BLOCK_CNT;
   unsigned int  STATUS;
   unsigned char SEND_CBW;
   unsigned char GET_CSW;
   unsigned char CSW_STATUS;
   unsigned int  BYTES_TRANSFERRED;
   unsigned int  ERROR_CODE;
   unsigned char MAX_LUN;
}STORAGE_INFO_t;
```

4.2.1 *CSW* Check

After a data exchange the function `int CheckCSW(STORAGE_INFO_t *gStorageInfo)` is used to check for possible errors or validity. The criteria for a *valid* transmission is:

- The *CSW* is 13 bytes long
- The *CSW* signature is equal to `0x53425355`
- The *CSW* tag must be equal to the *CBW* tag of the previous sent *CBW*

Also a *meaningful* data exchange can be detected. The criteria for this result is:

- The *CSW* status is `0x00` or `0x01`, and: *CSWDataresidue* is lower than or equal to *CBWDataTransferLength*, and: *CSWDataresidue* is not equal to `0x00`
- The *CSW* status is `0x02`

4.2.2 Data Exchange

For data exchange the mass storage class uses the function `int Transmit_Data(STORAGE_INFO_t *gStorageInfo)`. The function is capable to handle transmission and reception data.

4.2.3 Transmission Completion Check

Because every transmission needs a certain time and to avoid overlapping transfers, the function `void CheckForCompletion(unsigned char *NAKDetected)` checks for ongoing transfer.

For data transfers from host to function the variable `gOutTransferDone` and for the opposite direction the variable `gInTransferDone` are used. They are set to `0x01` before any transmission and set to `0x00` if the transmission has finished.

If a timeout (about 9 seconds) has occurred the variable `NAKDetected` is set to `0x01`.

Note:
Investigations has shown that from time to time several USB sticks “crashed” during a data transfer so that a *NAK* was sent back in this case. In this case the mass storage driver error handling sends a *USB Reset* to the connected device.

4.2.4 Bulk Only MS Reset

The specification of this class command stipulates its implementation. It is realized in the function `int BulkOnlyMSReset(void)`.

Note:
Not all USB mass storage devices support this command. In this case these devices enter the *STALL* mode. A workaround would be to send a *USB Reset* in this case. Therefore this mass storage driver does not use this function.

4.2.5 Get Max LUN

The specification of this class command stipulates its implementation. It is realized in the function `int GetMaxLun(STORAGE_INFO_t *gStorageInfo)`.

Note:
Some USB mass storage devices return a `0x00` although they contain more than one logical units. Also some USB mass storage devices enter the *STALL* mode after this command. A workaround would be to send a *USB Reset* in this case. Therefore this mass storage driver does not use this function and makes only use of logical unit number 0.

4.2.6 Test Unit Ready

To check if a connected device is ready, the function `int TestUnitReady (STORAGE_INFO_t *gStorageInfo)` is used.

Note:
Before the *CSW* reception, the driver waits approximately 2 ms using the function `UMH_StallExecutionUserCallback` which uses reload timer 0.

Please also note that some USB mass storage devices need up to 20 ms for internal check.

4.3 Error Handling

The mass storage class driver does not provide all possible error handling according to the USB specification. The main reason is to realize a compact driver for embedded systems to preserve ROM memory. Other reasons are the lack of all SCSI command implementations of the manufacturer of mass storage devices.

4.3.1 *NAK* Handshake

In case of transmission errors or ongoing *NAK* handshake the function `void ErrorHandler(unsigned int errCode, unsigned char *clrEponly)` is called. This error handler calls the function `void HardReset(void)` to perform a *USB Reset*. In this case the connected device is re-enumerated again.

If an endpoint goes into *STALL* mode, the function `int ClearInEndpoint(void)` or `int ClearOutEndpoint(void)` respectively is called. If this does not lead to success the function `int ResetRecovery(void)` is called.

5 Manual Changes

DESCRIPTION OF THE CHANGES OF THE DRIVER SOFTWARE

5.1 USB-FUMA

After copying the files listed in 2.1.2 to the project some changes are needed.

5.1.1 config.h

Please remove `PRINT_DEVICE_EVENTS` definition and replace the following definitions with new values as shown for declaration of mass storage devices.

```
// USB HID class specific defines for supported devices
#define DEMO_DEVICE_DATA_EP_ADDRE 0x81
#define DEMO_DEVICE_INTERFACE_CLASS     8    // Mass Storage Device = 8
#define DEMO_DEVICE_INTERFACE_SUB_CLASS 6    // SCSI Transparent Command Set = 6
#define DEMO_DEVICE_INTERFACE_PROTOCOL  0x50 // Bulk Only Transport = 0x50
#define DEMO_DEVICE_BNUM_ENDPOINTS      2    // Number of Endpoints = 2
```

5.1.2 hw_support.c

Registers of USART5 should be changed to USART1. Take care of port input enable register. Please also comment V_{BUS} and overcurrent enable ports, if they are not supported. These ports are used if an external USB bus power circuit is connected to the MCU.

```
  . . .

// Key code trace UART
static
void
Init_UART1( void )
{
  // set portpin SIN1/P08_5 to input
  DDR08_D5=0;
  PIER08_IE5=1;

  . . .

  void
  EnableVbus(void)
  {
       // VBUS_ENABLE_PORT=1;       // please uncomment if supported
  }

  void
  DisableVbus(void)
  {
       // VBUS_ENABLE_PORT=0;       // please uncomment if supported
  }

  BOOL
  IsOverCurrent(void)
  {
       // return !OVERCURRENT_PORT; // please uncomment if supported
       return 0;                    // please comment if supported
  }
```

5.1.3 hw_support.h

Set correct `LD1` and `LD2` port names or use `void*`. For the *SK-16FX-144PMC-USB* board it is:

```
//USB device connected
#define LD1_USB_LINK_LED_DIR  DDR00_D0
#define LD1_USB_LINK_LED      PDR00_P0
// Timer interrupt
#define LD2_TIMER_LED_DIR     DDR00_D3
#define LD2_TIMER_LED         PDR00_P3
```

5.2 File System

5.2.1 diskio.h

The case code for `ATA` and `MMC` can be removed completely, because only `USB` is used.

Write `stat = 0;` and `return stat;` within the curly braces of the switch query right behind `case usb:` of the function `disk_initialize`. Remove the variable `int result`.

Note: There is no need to initialize the USB-Mass-Storage before each write or read transaction. This is the reason for a static zero return from this function.

In the function `disk_status` remove the variable `int result`.

Note: There is also no need to check the disk for write protection, before each write or read transaction. If you want to use it call the Mass-Storage-API-Function `ModeSense()`.

Replace the function caller `USB_disk_read (buff, sector, count)` with `MSD_Read (buff, sector, count)`. This is the read function from the Mass-Storage-API.
After that, you have to do a little result code translation.

```
      if (result == 0x02)
      {
            res = RES_NOTRDY;
      }
      else if( result == 0x03)
      {
            res = RES_PARERR;
      }
      else
      {
            res = result;
      }
```

Explanation:

If the device is not ready, the function `MSD_Read` returns `0x02`. If an error occurred, the function returns a `0x03` error code. There must be a little translation because the Mass-Storage-Class uses different error codes as the FAT-Module.

Remove the switch cases `MMC` and `ATA` of the function `disk_write`. Replace the function caller `USB_disk_write (buff, sector, count)` with `MSD_Write (buff, sector, count)`. Now do the same as you did in the step above.

This step is a bit more complicate. Here is a working example how the function should look like.

```
DRESULT disk_ioctl (
      BYTE drv,       // Physical drive number (0..)
      BYTE ctrl,      // Control code
      void *buff      // Buffer to send/receive control data
)

{
    unsigned char tmp[4];
    unsigned long *ptmp;
    READ_CAPA_t ReadCapacity;
    DRESULT res;

    switch (drv)
    {
        case USB :

        res = MSD_Read_Capacity(&ReadCapacity);
        switch (ctrl)
        {
            case GET_SECTOR_COUNT : // Get number of sectors on the disk
                                    //   (DWORD)

               tmp[0] = (ReadCapacity.SectorCount >> 24);
               tmp[1] = (ReadCapacity.SectorCount >> 16);
               tmp[2] = (ReadCapacity.SectorCount >>  8);
               tmp[3] = (ReadCapacity.SectorCount);
               ptmp = (unsigned long*) &tmp[0];
               *(DWORD*)buff = *ptmp;
               return res;

             case GET_BLOCK_SIZE :  // Get erase block size in sectors (DWORD)

               tmp[0] = (ReadCapacity.SectorSize >> 24);
               tmp[1] = (ReadCapacity.SectorSize >> 16);
               tmp[2] = (ReadCapacity.SectorSize >>  8);
               tmp[3] = (ReadCapacity.SectorSize);
               ptmp = (unsigned long*) &tmp[0];
               *(DWORD*)buff = (*ptmp)/(512);
               return res;

             case CTRL_SYNC :       // Nothing to do
               return RES_OK;

             default:
               return RES_PARERR;
       }
       default:
       return RES_PARERR;
    }
}
```

This function checks the Mass-Storage-Capacity.

Add after the function `ioctl` a function that is called `get_fattime`. This function returns the system time. In that example a static date and time is returned, but you can also implement the time from a real-time-clock (RTC) or dcf77 receiver.

Note:
This is an example, to return the date 01.01.1981 and the time 01:01. About the time format, refer the FAT specification (8.2).

```
DWORD get_fattime (void)
{
       return 0x2210821;    // 01.01.1981 01:01
}
```

The following includes should be added:

```
#include "UsbMinihost.h"
#include "storage.h"
#include "storage_API.h"
```

5.2.2 diskio.h

Please add the following include due to changes in *integer.h* (s. below).

```
#include "tal_defs.h"
```

5.2.3 integer.h

Please comment the type definitions for

`UINT`, `CHAR`, `UCHAR`, `SHORT`, `USHORT`, `LONG`, and `ULONG`.

Otherwise they will conflict with the type definitions in *tal_defs.h* of the FUMA package.

6 File System

SHORT LIST OF THE FUNCTIONS OF THE FILE SYSTEM

6.1 User API

The FAT16 file system user API contains the following functions. Please be aware that this list depends on the used file system from ELM Chan. If the user wants to use a different FAT 16 driver, the following function names, functionality, and/or the used parameters may differ.

Here only the used functions of the discussed application note project are listed. For further details please refer to the download link in the application note appendix (8.2).

6.1.1 Create File System

The function `FRESULT f_mkfs (BYTE Drive, BYTE PartitioningRule, WORD AllocSize)` creates a file system on a connected mass storage device.

6.1.2 Register/Unregister a work area

To register or unregister a work area, the function `FRESULT f_mount (BYTE Drive, FATFS* FileSystemObject)` is used.

6.1.3 Open/Create File

`FRESULT f_open (FIL* FileObject, const char* FileName, BYTE ModeFlags)` opens a file object so that it can be accessed.

6.1.4 Remove File/Directory

The function `FRESULT f_unlink (const char* FileName)` removes a file or directory.

6.1.5 Read File

The function `FRESULT f_read (FIL* FileObject, void* Buffer, UINT ByteToRead, UINT* BytesRead)` is used to read data from a file.

6.1.6 Write File

`FRESULT f_write (FIL* FileObject, const void* Buffer, UINT ByteToWrite, UINT* ByteWritten)` writes data from a buffer to a file.

6.1.7 Get File Status

To test a file or directory path, the function `FRESULT f_stat (const char* FileName, FILINFO* FileInfo)` is used. The result is stored in the structure `FileInfo`.

6.1.8 Close File

To close a file, the function `FRESULT f_close (FIL* FileObject)` is used.

6.1.9 Rename File/Directory

`FRESULT f_rename (const char* OldName, const char* NewName)` renames a file or directory.

6.1.10 Make Directory

The function `FRESULT f_mkdir (const char* DirName)` creates a new directory.

6.1.11 Move R/W File Pointer

The function `FRESULT f_lseek (FIL* FileObject, DWORD Offset)` moves the file read/write pointer. It also can be used to append data to a file.

6.1.12 Get free Clusters

`FRESULT f_getfree (const char* Path, DWORD* Clusters, FATFS** FileSystemObject)` is used to get the number of free sectors per cluster.

6.1.13 Open Directory

`FRESULT f_opendir (DIR* DirObject, const char* DirName)` opens a (sub) directory.

6.1.14 Read Directory

`FRESULT f_readdir (DIR* DirObject, FILINFO* FileInfo)` reads the directory entries.

7 Source Codes

SOURCE CODES OF THE MASS STORAGE PROJECT

7.1 Main.c

Please note that the main function is only the entry point for USB enumeration. After successful enumeration, the function `MenuInit()` in the module *menu.c* is called.

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*                 (C) Fujitsu Microelectronics Europe GmbH                  */
/*---------------------------------------------------------------------------*/

#include <stdlib.h>
#include "mb96338us.h"

#include "usbspec.h"
#include "UsbMiniHost.h"
#include "osal.h"
#include "msc_config.h"

#include "stdlib.h"
#include "storage_API.h"
#include "usb_functions.h"

#include "app_global.h"
#include "config.h"

// For debugging UART1 is used
// Menu dialog on USART0

////////////// globals
int gConnectFlag;
int gConfigured;
int gStartEnumerate, gStartConfigure;

// all used handles for the demo application
UMH_HANDLE    gBulkInHandle;
UMH_HANDLE    gBulkOutHandle;
UCHAR         gBulkEpINAddr, gBulkEpOUTAddr;

extern unsigned long gTimer3;

#define TCCS_CP_INT_EN_MASK    0x0100
#define TCCS_STOP_MASK         0x0020
#define TCCS_SRCCLK_MASK       0 //timer clock is 24Mhz
#define TCCS_CLR_COUNTER_MATCH 0x0010

//*************************************************************
//                    M A I N ()
//*************************************************************

void main(void)
{
  UINT ByteRead =0;
  DWORD P1 = 0;

  int status;
```

```
UINT maxINPacketSize,maxOUTPacketSize;

gConnectFlag = FALSE;
gConfigured  = FALSE;

// endpoint handles
gBulkInHandle  = NULL;
gBulkOutHandle = NULL;

gStartEnumerate = TRUE;
gStartConfigure = TRUE;

initglobals();

HW_Init();

InitUART0();

// Set all trace masks for all used modules
// This application
Hid_SetTraceMask(0x07);

// Host hardware abstraction layer trace masks
UmhHal_SetTraceMask(0x07);

// USB host library
UMH_SetTraceMask(0x07);

DBGOUT(DBG_ERR,DbgPrint("\n*****************************\n"));
DBGOUT(DBG_ERR,DbgPrint("MB96338U USB Host Library\n"));
DBGOUT(DBG_ERR,DbgPrint("*****************************\n"));

__DI();
UMH_Init(StateCallBack);
__EI();

DBGOUT(DBG_ERR,DbgPrint("UMH initialized.\n"));

puts("\nFujitsu Microelectronics GmbH - USB-Host Mass Storage Demo.\n\n");
puts("USB Library initialized. Please connect USB-Stick.\n");

while(1)
{
  if(!gConnectFlag)
  {
    // wait for connection
    gConfigured=FALSE;

    if( gBulkInHandle)
    {
      // puts("gBulkInHandle\n");
      // remove the bulk in endpoint if exist
      status=UMH_RemoveEndpoint(&gBulkInHandle);
      if( status)
      {
        DBGOUT(DBG_ERR,DbgPrint("ERROR UMH_RemoveEndpoint:%d!\n",status));
      }
    }
    if( gBulkOutHandle)
    {
      // puts("gBulkOutHandleg\n");
      // remove the bulk Out endpoint if exist
      status=UMH_RemoveEndpoint(&gBulkOutHandle);
```

```
        if( status)
        {
          DBGOUT(DBG_ERR,DbgPrint("ERROR UMH_RemoveEndpoint:%d!\n",status));
        }
      }
      gStartEnumerate = TRUE;
      gStartConfigure = TRUE;
    }

    do{
      if( gConnectFlag )
      {
        if( gStartEnumerate )
        {
          gStartEnumerate = FALSE;
          DBGOUT(DBG_INFO,DbgPrint("Enumerate the USB device!\n"));
          // enumerate the device
          status = UMH_EnumerateDevice(DEVICE_ADDRESS,
                   (UMH_STATUS_COMPLETION*)NULL);
          // set the configuration
          if( status)
          {
            DBGOUT(DBG_ERR,DbgPrint("ERROR UMH_EnumerateDevice:%d!\n",status));
            gConnectFlag=FALSE;
            break;
          }
          Status = USBStickDetection(&gBulkEpINAddr,
                   &gBulkEpOUTAddr,&maxINPacketSize,&maxOUTPacketSize);
          if( status)
          {
            DBGOUT(DBG_ERR,DbgPrint("No USBStick detected.
Error:%d!\n",status));
            gConnectFlag = FALSE;
            break;
          }

          // add an endpoint only after the enumeration
          // if this endpoint is removed then it is only allowed to add them
          // only  after an USB reset or after then UMH_EnumerateDevice() call
          // because this endpoint is also initialised
          if( NULL == gBulkInHandle)
          {
           // add a bulk  endpoint
            status=UMH_AddEndpoint(
                    &gBulkInHandle,
                    gBulkEpINAddr,
                    maxINPacketSize,
                    0x00,          //this should be 0 for bulk transfer
                    (UMH_COMPLETION*)BulkInCompletion
                                   );
            DBGOUT(DBG_INFO,DbgPrint("IN-Endpoint added\n"));
            if( status)
            {
              DBGOUT(DBG_ERR,DbgPrint("ERROR UMH_AddEndpoint:Ep:0x%x
Status:%d!\n",(USHORT)gBulkEpINAddr,status));
              break;
            }

            if( NULL==gBulkOutHandle)
            {
            // add a bulk endpoint
```

```
            status = UMH_AddEndpoint(
                     &gBulkOutHandle,
                     gBulkEpOUTAddr,
                     maxOUTPacketSize,
                     0x00,          // this should be 0 for bulk transfer
                     (UMH_COMPLETION*)BulkOutCompletion
                                    );
            DBGOUT(DBG_INFO,DbgPrint("Out-Endpoint added\n"));
            if( status)
            {
              DBGOUT(DBG_ERR,DbgPrint("ERROR UMH_AddEndpoint:Ep:0x%x\
                       Status:%d!\n",(USHORT)gBulkEpOUTAddr,status));
              break;
            }
          }
        }

        if( gStartConfigure)
        {
          gStartConfigure = FALSE;
          status = UMH_SetConfiguration(CONFIGURATION_VALUE,
                   (UMH_STATUS_COMPLETION*)NULL);
          if( status)
          {
            DBGOUT(DBG_ERR,DbgPrint("ERROR UMH_SetConfiguration:%d!\n",status));
            gConnectFlag = FALSE;
            break;
          }
          else
          {
            DBGOUT(DBG_ERR,DbgPrint("Device is configured!\n",status));
            // device is configured
            gConfigured = TRUE;
          }
          if( UMH_STATUS_SUCCESS != StorageInit())
          {
             DBGOUT(DBG_INFO,DbgPrint("Initialization failed!!\n"));
          }
          else
          {
             DBGOUT(DBG_INFO,DbgPrint("Initialization successful!!\n"));
          }
        }
        break;
      }
    }while(0);

    // read or write data from device from here or in a other interrupt context
    if( gConfigured && gConnectFlag )
    {
       MenuInit();  // <----------- calls the main menu (main application)

       puts("\n Menu closed. Please reset for starting over.\n");
       while(1);
    }
  } // end while
}// end main
```

7.2 menu.c

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*                 (C) Fujitsu Microelectronics Europe GmbH                  */
/*---------------------------------------------------------------------------*/

#include <stdlib.h>
#include "mb96338us.h"
#include "menu.h"

#include "usbspec.h"
#include "usbminihost.h"
#include "osal.h"
#include "config.h"
#include "storage.h"
#include "storage_API.h"
#include "diskio.h"
#include "uart.h"
#include "ff.h"
#include "UsbMiniHost.h"
#include "string.h"

/*---------------------------------------------------------------------------*/
/* Globals                                                                   */
/*---------------------------------------------------------------------------*/
extern int gConnectFlag;
extern int gWakeUpFlag;

FIL ffile;
FATFS fatfs[1];
UCHAR RWBuffer[512];

/*---------------------------------------------------------------------------*/

void MenuInit(void)
{
   UCHAR MMExit=0, input;
   while(!MMExit)
   {
       puts("\n");
       puts("= --------------------------- =\n");
       puts("=          MAIN MENU          =\n");
       puts("= --------------------------- =\n");
       puts("= 0..Format USB-Stick          =\n");
       puts("= 1..Read                      =\n");
       puts("= 2..Create/Write              =\n");
       puts("= 3..Delete/Rename             =\n");
       puts("= 4..Dirlist                   =\n");
       puts("= 5..Disk Space                =\n");
       puts("= 6..EXIT                      =\n");
       puts("================================\n");
       puts("= Select function (press 0-6) =\n");
       puts("================================\n");
       puts("\n>");

       input = Inputhex(1);

       switch(input)
       {
           case 0:  FormatMenu();    ;break;
           case 1:  ReadMenu();      ;break;
           case 2:  CreateMenu();    ;break;
```

```
            case 3:  DeleteMenu();     ;break;
            case 4:  DirlistMenu();    ;break;
            case 5:  DiskSpace();      ;break;
            case 6:  MMExit=1;         ;break;
        }
    }
}

void FormatMenu(void)
{
    UCHAR SWExit=0, input;
    DWORD P1 = 0;
    f_mount((BYTE)P1, &fatfs[P1]);

    while(!SWExit)
    {
        puts("\n");
        puts("= --------------------------- =\n");
        puts("=          WARNING!!          =\n");
        puts("= Formatting will erase all   =\n");
        puts("= Data on this disk.          =\n");
        puts("= --------------------------- =\n");
        puts("= 1..FORMAT DISK              =\n");
        puts("= 2..CANCLE                   =\n");
        puts("================================\n");
        puts("= Select function (press 1-2) =\n");
        puts("================================\n");
        puts("\n>");

        input = Inputhex(1);

        switch(input)
        {
            case 1:  puts("\nPlease wait\n");
                     f_mkfs(0, 0, 2048); SWExit=1;break;
            case 2:  SWExit=1;                    break;
        }
    }
}


void ReadMenu(void)
{
    UCHAR SWExit=0, input;

    while(!SWExit)
    {
        puts("\n");
        puts("= --------------------------- =\n");
        puts("=           Read Menu         =\n");
        puts("= --------------------------- =\n");
        puts("= 1..Read File                =\n");
        puts("= 2..CANCLE                   =\n");
        puts("================================\n");
        puts("= Select function (press 1-2) =\n");
        puts("= For Help press 9            =\n");
        puts("================================\n");
```

```
        puts("\n>");
        input = Inputhex(1);

        switch(input)
        {
            case 1:  ReadFile();          SWExit=1 ;break;
            case 2:  SWExit=1                      ;break;
            case 9:  Help()                        ;break;
        }
    }
}

void CreateMenu(void)
{
   UCHAR SWExit=0, input;

   while(!SWExit)
   {
        puts("\n");
        puts("= ---------------------------- =\n");
        puts("=          Create Menu         =\n");
        puts("= ---------------------------- =\n");
        puts("= 1..Create subfolder          =\n");
        puts("= 2..Create file               =\n");
        puts("= 3..Write to existing file    =\n");
        puts("= 4..CANCLE                    =\n");
        puts("================================\n");
        puts("= Select function (press 1-4) =\n");
        puts("= For Help press 9             =\n");
        puts("================================\n");
        puts("\n>");
        input = Inputhex(1);

        switch(input)
        {
            case 1:  CreateSubfolder();SWExit=1 ;break;
            case 2:  CreateFile();SWExit=1      ;break;
            case 3:  WriteExist();SWExit=1      ;break;
            case 4:  SWExit=1                   ;break;
            case 9:  Help();                    ;break;
        }
    }
}

void DeleteMenu(void)
{
   UCHAR SWExit=0, input;

   while(!SWExit)
   {
        puts("\n");
        puts("= ---------------------------- =\n");
        puts("=         Delete Menu          =\n");
        puts("= ---------------------------- =\n");
        puts("= 1..Delete Folder/File        =\n");
        puts("= 2..Rename Folder/File        =\n");
        puts("= 3..Cancle                    =\n");
        puts("================================\n");
        puts("= Select function (press 1-3) =\n");
        puts("= For Help press 9             =\n");
        puts("================================\n");
        puts("\n>");
        input = Inputhex(1);
```

```
        switch(input)
        {
             case 1:  DelFoldFile();     SWExit=1 ;break;
             case 2:  RenFoldFile();     SWExit=1 ;break;
             case 3:  SWExit=1                    ;break;
             case 9:  Help();                     ;break;
        }
    }
}


void DirlistMenu(void)
{
   UCHAR SWExit=0, input;

   while(!SWExit)
   {
       puts("\n");
       puts("= --------------------------- =\n");
       puts("=    Directory listing Menu   =\n");
       puts("= --------------------------- =\n");
       puts("= 1..List root folder          =\n");
       puts("= 2..List other folder         =\n");
       puts("= 3..Cancle                    =\n");
       puts("================================\n");
       puts("= Select function (press 1-3) =\n");
       puts("================================\n");
       puts("\n>");
       input = Inputhex(1);

       switch(input)
       {
           case 1:  scan_files ("");     SWExit=1 ;break;
           case 2:  DirListOther();      SWExit=1 ;break;
           case 3:  SWExit=1                      ;break;
       }
   }
}

void DirListOther(void)
{
   char *BufferPtr = NULL;

   puts("\n");
   puts("= --------------------------- =\n");
   puts("= Type in Directory to list:");
   BufferPtr = (char*) receive_line_echo(NULL);
   scan_files ((char*) BufferPtr);
   puts("\n");
}

void ReadFile(void)
{
   FILINFO finfo;
   UINT ByteRead = 0;
   DWORD P1 = 0;
   char *BufferPtr = NULL;

   memset(RWBuffer,0,sizeof(RWBuffer));
   f_mount((BYTE)P1, &fatfs[P1]);
   puts("\n");
   puts("= --------------------------- =\n");
```

```
   BufferPtr = (char*) receive_line_echo(NULL);
   f_open(&ffile,(char*)  BufferPtr, FA_READ);
   f_stat((char*)  BufferPtr, &finfo);
   puts("\n");
   puts("FileSize: ");
   putdec(finfo.fsize);
   puts("\n");
   if (finfo.fsize > sizeof(RWBuffer))
   {
      ULONG i = 0;
      i = finfo.fsize;
      while (i > 0 )
      {
         f_read(&ffile, RWBuffer, (UINT) sizeof(RWBuffer), &ByteRead);
         puts((char *)RWBuffer);
         i = i - ByteRead;
         memset(RWBuffer,0,sizeof(RWBuffer));
      }
   }
   else
   {
      f_read(&ffile, RWBuffer, (UINT) sizeof(RWBuffer), &ByteRead);
      puts((char *)RWBuffer);
      memset(RWBuffer, 0, sizeof(RWBuffer));
   }

   f_close(&ffile);
}


void CreateSubfolder(void)
{
   DWORD P1 = 0;
   char *BufferPtr = NULL;

   f_mount((BYTE)P1, &fatfs[P1]);
   puts("\n");
   puts("= --------------------------- =\n");
   puts("= Type in subfolder name:");
   BufferPtr = (char*) receive_line_echo(NULL);
   puts("\n");
   f_mkdir((char*) BufferPtr);


}

void CreateFile(void)
{
    char *BufferPtr = NULL;
    UINT s2;
    DWORD P1;
    UINT cnt;

    P1=0;
    f_mount((BYTE)P1, &fatfs[P1]);
    puts("\n");
    puts("= --------------------------- =\n");
    puts("= Type in file name:");
    BufferPtr = (char*) receive_line_echo(&cnt);
    puts("\n");
    f_open(&ffile,(char*)  BufferPtr, FA_CREATE_ALWAYS | FA_WRITE);
```

```
    puts("= Type in some text:");
    BufferPtr = (char*) receive_line_echo(&cnt);
    puts("\n");
    f_write(&ffile,(char*) BufferPtr,cnt,&s2);
    f_close(&ffile);
}

// Adding some text to an existing file
void WriteExist(void)
{
    char *BufferPtr = NULL;
    UINT s2;
    DWORD P1;
    UINT cnt;

    P1=0;
    f_mount((BYTE)P1, &fatfs[P1]);
    puts("\n");
    puts("= --------------------------- =\n");
    puts("= Type in file name:");
    BufferPtr = (char*) receive_line_echo(&cnt);
    puts("\n");
    f_open(&ffile,(char*)  BufferPtr, FA_OPEN_EXISTING | FA_WRITE);
    f_lseek(&ffile, ffile.fsize);             //move pointer to the end of file
    puts("= Type in some text:");
    BufferPtr = (char*) receive_line_echo(&cnt);
    puts("\n");
    f_write(&ffile,(char*) BufferPtr, cnt, &s2);
    f_close(&ffile);
}

void DelFoldFile(void)
{

  DWORD P1 = 0;
  char *BufferPtr = NULL;

  f_mount((BYTE)P1, &fatfs[P1]);
  puts("\n");
  puts("= --------------------------- =\n");
  puts("= Type in File- or Foldername name:");
  BufferPtr = (char*) receive_line_echo(NULL);
  puts("\n");
  f_unlink((char*) BufferPtr);
}

void RenFoldFile(void)
{
  char OldNameBuff[12];
  DWORD P1 = 0;
  char *BufferPtr = NULL;

  f_mount((BYTE)P1, &fatfs[P1]);
  puts("\n");
  puts("= --------------------------- =\n");
  puts("= Type in file- or foldername to rename: ");
  BufferPtr = (char*) receive_line_echo(NULL);
  strncpy(OldNameBuff, BufferPtr,12);
  puts("\n");
  puts("= Type in new file- or foldername: ");
  BufferPtr = (char*) receive_line_echo(NULL);
  puts("\n");
```

```
  f_rename((char*) OldNameBuff, (char*) BufferPtr);
}


void DiskSpace(void)
{
  FATFS *fs;
  DWORD P1 = 0;
  DWORD clust = 0;
  DWORD total = 0;
  DWORD free = 0;

  f_mount((BYTE)P1, &fatfs[P1]);
  puts("\nPlease wait\n");
  f_getfree ("", &clust, &fs);
  total = (fs->max_clust - 2) * fs->sects_clust / 2;
  free = clust * fs->sects_clust / 2;
  puts("\n");
  puts("= --------------------------- =\n");
  putdec(total / 1024);
  puts(" MB total Disk space\n");
  putdec(free / 1024);
  puts(" MB free Disk space\n");
  puts("= --------------------------- =\n");
}


void scan_files (char* path)
{
    DWORD P1;
    FILINFO finfo;
    DIR dirs;
    int i;

    P1=0;
    f_mount((BYTE)P1, &fatfs[P1]);

    if (f_opendir(&dirs, path) == FR_OK)
    {
        i = strlen(path);
        while ((f_readdir(&dirs, &finfo) == FR_OK) && finfo.fname[0])
        {
            if (finfo.fattrib & AM_DIR)
            {

                puts("\n");
                puts((char*)finfo.fname);
                puts("\n");
            }
            else
            {
                puts("\n");
                puts((char*)finfo.fname);
                puts("     ");
                putdec(finfo.fsize);
                puts(" ");
                puts("byte");
                puts("\n");
            }
        }
    }
}
```

```
void Help(void)
{
   UCHAR SWExit = 0, input;

   While (!SWExit)
   {
       puts("\n");
       puts("= --------------------------- =\n");
       puts("=           HELP              =\n");
       puts("= --------------------------- =\n");
       puts("= To read, create or delete   =\n");
       puts("= a file in a subfolder, type =\n");
       puts("= in the subfolders name in   =\n");
       puts("= front of the filename.      =\n");
       puts("= e.g. sub1/test.txt          =\n");
       puts("==============================\n");
       puts("= Press 9 to return           =\n");
       puts("==============================\n");
       puts("\n>");
       input = Inputhex(1);

       switch (input)
       {
           case 9:  SWExit=1             ;break;
       }
   }
}
```

7.3 menu.h

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*                 (C) Fujitsu Microelectronics Europe GmbH                  */
/*---------------------------------------------------------------------------*/


#ifndef _MENU_H_
#define _MENU_H_

void MenuInit(void);
void FormatMenu(void);
void ReadMenu(void);
void CreateMenu(void);
void DeleteMenu(void);
void CreateSubfolder(void);
void CreateFile(void);
void DelFoldFile(void);
void ReadFile(void);
void RenFoldFile(void);
void WriteExist(void);
void DiskSpace(void);
void scan_files (char* path);
void DirlistMenu(void);
void DirListOther(void);
void Help(void);

#endif // _MENU_H_
```

7.4 uart.c

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*                 (C) Fujitsu Microelectronics Europe GmbH                  */
/*---------------------------------------------------------------------------*/

#include "mb96338us.h"
#include "uart.h"

/*===========================================================================*/

// global variables
char line_buffer[255];
char *line_buffer_ptr;

// constants
const char ASCII[] = "0123456789ABCDEF";

/*****************************************************************************/
/* UART0 Communication routines */
/*****************************************************************************/

void InitUART0( void )
{
  PIER08 |= 0x04; // enable SIN0
  DDR08_D3 = 1;   // SOT0 = output

  BGR0 = 415;     // 115.2kbaud @ 48 MHz
  SCR0 = 0x17;    // 8N1
  SMR0 = 0x0d;    // enable SOT0, Reset, normal mode
  SSR0 = 0x00;    // LSB first
}

void putch (char ch)     // sends a char
{
   while (!SSR0_TDRE);   // wait for transmit buffer empty
   TDR0 = ch;            // put ch into buffer
}

void puts(char *buf)
{
      while (*buf != '\0')
      {
      if (*buf == '\n') putch('\r');
      putch(*buf++);     // send every char of string
    }
}

void puthex(unsigned long n, unsigned char digits)
{
    unsigned char i, ch, div = 0;

    puts("0x");                  // hex string
    div = (digits - 1) << 2;     // init shift divisor

    for (i = 0; i < digits; i++)
    {
         ch = (n >> div) & 0xF;  // get hex-digit value
        putch(ASCII[ch]);        // prompt to terminal as ASCII
        div -= 4;                // next digit shift
    }
}
```

```
void putdec(unsigned long x)
{
       int i;
       char buf[9];

       buf[9] = '\0';                    // end sign of string

       for (i = 8; i > 0; i--)
       {
          buf[i - 1] = ASCII[x % 10];
          x = x / 10;
       }

       for (i = 0; buf[i] == '0'; i++)   // no print of zero
       {
          buf[i] = ' ';
    }
    puts(buf);                           // send string
}

char getch (void)
{
       while(!SSR0_RDRF);                // wait for data in buffer
       if (SSR0_ORE || SSR0_PE)
       {                                 // overrun or parity error
         return (-1);
       } else

       return (RDR0);                    // return char
}

unsigned long ASCIItobin(unsigned char k)
{
  char d = (char) -1;
  if ((k > 47) & (k < 58)) d = k - 48;  // 0..9
  if ((k > 64) & (k < 71)) d = k - 55;  // A..F
  if ((k > 96) & (k < 103)) d = k - 87; // a..f
  return d;
}

void receive_line(void)
{
  unsigned char ch;
  unsigned short i = 0;

       do {
              ch = getch();
              if((ch == '\r') | (ch == '\n')) break;
              line_buffer[i++] = ch;
       } while(1);

   line_buffer[i] = '\0';
   line_buffer_ptr = line_buffer + i;
}

int receive_line_echo(unsigned int *cnt)
{

  unsigned char ch;
  unsigned short i = 0;
  memset(line_buffer, 0, sizeof(line_buffer));
```

```
       do {
              ch = getch();
         putch(ch);
         if((ch == 13) | (ch == 27)) break;
         line_buffer[i++] = ch;
   } while(1);

   line_buffer[i] = '\0';
   line_buffer_ptr = line_buffer + i;
   *cnt = i;
   //BufferPtr = (char)  &line_buffer;
   return (int) &line_buffer;
}

int scan_line(char *str)
{
    line_buffer_ptr = line_buffer;

    while((int)*line_buffer_ptr==(int)*str)
    {
         if((int)*str == '\0') return(0);
         line_buffer_ptr++;
         str++;
    }

    if((int)*str == '\0') return(1);
    return(2);
}

unsigned long Inputhex(unsigned char digits)
{
   unsigned long number = 0, digit = 0;
   unsigned char abort = 0, mlt = 0, i, key;

   mlt = (4  *(digits - 1));  // 20 for 6 hex-digit numers, 4 for 2 hex-digits
   for (i = 0; i < digits; i++)
   {
     digit = -1;
     while ((digit == -1) & (!abort)) // input next valid digit
     {
       key = getch();              // wait for next key input (ASCII)
       putch(key);
       if (key == 27) abort = 1;  // ESC aborts
       digit = ASCIItobin(key);   // convert to number
       if (digit == -1) putch(8); // backspace if not valid
     }
     number+= (digit << mlt);     // add digit value to number
     mlt - =4;                    // next digit shift
   }

   if (!abort) return number;     // return input value
   else
   {
      puts("\n\n input cancled \n");
      return -1;                  // return abort indicator
   }
}

char getkey(char LKey, char HKey)
{
   char key;

   do                             // input next valid digit
   {
```

```
      key = upcase(getch());      // wait for next key input (0-9,A-Z,a-z)
      if (key == 27)
      {
        puts("\r>>> cancel input \n");
        return -1;                // return with ESC aborts
      }

      if ( (key < LKey) || (key > HKey) )
      {
        /* undefinded key pressed */
        puts("\r>>> key not defined \r");
      }
      else
      {
        puts("\r>                    \r");
        putch(key);
        return key;               // return input value
      }

    } while(1);
}

char upcase(char k)
{
  char d = (char) -1;
  if ((k > 47) & (k < 58))  d = k;       // 0..9
  if ((k > 64) & (k < 71))  d = k;       // A..F
  if ((k > 96) & (k < 123)) d = k - 32; // a..f
  if (k == 27) d = k;                    // ESC
  return d;
}
```

7.5 uart.h

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*                 (C) Fujitsu Microelectronics Europe GmbH                  */
/*---------------------------------------------------------------------------*/

#ifndef _UART_H
#define _UART_H

void InitUART(void);
void receive_line(void);
int receive_line_echo(unsigned int *cnt);
int  scan_line(char *str);
unsigned long Inputhex(unsigned char digits);
unsigned long ASCIItobin(unsigned char k);
void puthex(unsigned long n, unsigned char digits);
void putdec(unsigned long x);
char getch(void);
void putch(char ch);
void puts(char *buf);
char getkey(char LKey, char HKey);
char upcase(char k);

#endif /* UART_H */
```

7.6 usb_functions.c

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*                 (C) Fujitsu Microelectronics Europe GmbH                  */
/*---------------------------------------------------------------------------*/

#include <stdlib.h>
#include "mb96338us.h"

#include "usbspec.h"
#include "usbminihost.h"
#include "osal.h"
#include "config.h"

#include "storage_API.h"

#include "diskio.h"
#include "usb_functions.h"
#include "app_global.h"

extern STORAGE_INFO_t gStorageInfo;                // extern from storage_API.c
extern UCHAR gBulkEpOUTAddr, gBulkEpINAddr;        // extern from main.c
extern UMH_HANDLE gBulkInHandle, gBulkOutHandle;   // extern from main.c
extern int gStartEnumerate, gStartConfigure;       // extern from main.c
extern int gConnectFlag;                           // extern from main.c
extern int gConfigured;                            // extern from main.c

volatile unsigned long gTimer2 = 0, gTimer3 = 0;
volatile unsigned long gTimer, gTransferSpeedTime;
volatile unsigned long gTotalWrite, gTotalRead, gCurrentWrite, gCurrentRead;
UCHAR         gSetupCompleteFlag;
UCHAR         gSetupStatus;
UINT          gSetupTransferred;
unsigned char gTempBuffer[200];
int gWakeUpFlag;
SETUP_DATA setup;

void initglobals(void)
{

  gWakeUpFlag     = FALSE;
  gConnectFlag    = FALSE;
  gConfigured     = FALSE;
  gBulkInHandle   = NULL;
  gBulkOutHandle  = NULL;

  gStartEnumerate = TRUE;
  gStartConfigure = TRUE;
}

int ResetRecovery(void)
{
   unsigned int status;
   status = BulkOnlyMSReset();
   status = ClearOutEndpoint();
   status = ClearInEndpoint();
   status = TestUnitReady(&gStorageInfo);
   return status;
}
```

```
int ClearInEndpoint(void)
{
  int status;

  ZERO_STRUCT(setup);
  setup.Bytes.Type=0x02;
  setup.Bytes.Request=0x01;
  setup.Words.Value=0x00;
  setup.Words.Index=USB_IN_DIRECTION | gBulkEpINAddr;
  setup.Words.Length=0x00;
  UMH_AbortTransfer(gBulkInHandle);
  status = UMH_SetupRequest((UCHAR*)&setup,(UCHAR*)NULL,(unsigned int*)NULL);
  UMH_ResetDataToggleBit(gBulkInHandle);
  return status;
}

int ClearOutEndpoint(void)
{
  int status;

  ZERO_STRUCT(setup);
  setup.Bytes.Type=0x02;
  setup.Bytes.Request=0x01;
  setup.Words.Value=0x00;
  setup.Words.Index=USB_OUT_DIRECTION | gBulkEpOUTAddr;
  setup.Words.Length=0x00;
  UMH_AbortTransfer(gBulkOutHandle);
  status = UMH_SetupRequest((UCHAR*)&setup,(UCHAR*)NULL,(unsigned int*)NULL);
  UMH_ResetDataToggleBit(gBulkOutHandle);
  return status;
}

void HardReset(void)
{
  HCNT0_URST=1;                          // Bus Reset
  UMH_StallExecutionUserCallback(100); // Wait 10ms
  gStartEnumerate=TRUE;
  gStartConfigure=TRUE;
  gConnectFlag=TRUE;
}

void GetEPStatus(unsigned char *InEpStatus, unsigned char *OutEpStatus)
{
  SETUP_DATA setup;
  int status;
  unsigned int transf=0;

  ZERO_STRUCT(setup);
  setup.Bytes.Type=0x82;
  setup.Bytes.Request=0x00;
  setup.Words.Value=0x00;
  setup.Words.Index=USB_IN_DIRECTION | gBulkEpINAddr;
  setup.Words.Length=0x02;
  status = UMH_SetupRequest((UCHAR*)&setup,(unsigned char*)InEpStatus,
          (unsigned int*)&transf);

  // puts("\n In EP Status: ");
  // putdec((unsigned long)*InEpStatus);

  ZERO_STRUCT(setup);
  setup.Bytes.Type=0x82;
  setup.Bytes.Request=0x00;
  setup.Words.Value=0x00;
```

```
  setup.Words.Index=USB_OUT_DIRECTION | gBulkEpOUTAddr;
  setup.Words.Length=0x02;
  status = UMH_SetupRequest((UCHAR*)&setup,(unsigned char*)OutEpStatus,
           (unsigned int*)&transf);
  // puts("\n Out EP Status: ");
  // putdec((unsigned long)*OutEpStatus);
}

void BulkInCompletion(UMH_STATUS Status, unsigned int BytesTransferred,
                      void* Context)
{
  Context = Context;

  // check the completion status
  if (Status)
  {
        DBGOUT(DBG_ERR,DbgPrint("ERROR BulkInCompletion:%d\n",Status));
  }
  else
  {
     DBGOUT(DBG_ERR,DbgPrint("Bulk.Out length:%d\n",BytesTransferred));
     // ProcessKbdPacket( gInterruptBuffer,BytesTransferred);
  }
  // start a new transfer if the device is in configured state and Status
  //   is not canceled
  if (Status == UMH_STATUS_CANCELED)
  {
     DBGOUT(DBG_ERR,DbgPrint("BulkOutCompletion stops because status \
                              UMH_STATUS_CANCELED!\n"));
     return;
  }
  // get the device state, another solution is to update a global flag
  // gConfigure that is set after successfully configuration
  // and reset if a the device is switched to suspend, after a USB
  // reset or disconnect event and set if a wakeup event occurs or the device
  // is configured

  gStorageInfo.STATUS = Status;
  //gBulkInComplete=TRUE;
  gStorageInfo.BYTES_TRANSFERRED = BytesTransferred;
  InTransferDone();
}

void BulkOutCompletion(UMH_STATUS Status, unsigned int BytesTransferred,
                       void* Context)
{
  Context=Context;

  // check the completion status
  if (Status)
  {
     DBGOUT(DBG_ERR,DbgPrint("ERROR BulkInCompletion:%d\n",Status));
  }
  else
  {
     DBGOUT(DBG_ERR,DbgPrint("Bulk.Out length:%d\n",BytesTransferred));
  }

  // start a new transfer if the device is in configured state and Status
  //   is not canceled
  if (Status == UMH_STATUS_CANCELED)
  {
```

```
    DBGOUT(DBG_ERR,DbgPrint("BulkOutCompletion stops because \
                            status UMH_STATUS_CANCELED!\n"));
    return;
  }
  // get the device state, another solution is to update a global flag
  // gConfigure that is set after successfully configuration and reset if a
  // the device is switched to suspend,after a USB reset or disconnect event
  // and set if a wakeup event occurs or the device is configured

  gStorageInfo.STATUS = Status;
  //gBulkOutComplete=TRUE;
  gStorageInfo.BYTES_TRANSFERRED = BytesTransferred;
  OutTransferDone();
}

int Transmit_MSDC_Data(unsigned char *Buffer, STORAGE_INFO_t *gStorageInfo)
{
   unsigned int status;

   if (gBulkInHandle && gBulkOutHandle)
   {

      if(gStorageInfo->SEND_CBW)
      {
         status = UMH_Transfer(gBulkOutHandle, NULL, Buffer,
                            gStorageInfo->BUFFER_SIZE);
         return status;
      }

      if (gStorageInfo->GET_CSW)
      {
         status = UMH_Transfer(gBulkInHandle, NULL, Buffer,
                            gStorageInfo->BUFFER_SIZE);
         return status;
      }

      switch(gStorageInfo->STORAGE_CMD)
      {

         case WRITE_MASS_STORAGE:
         case TEST_UNIT_READY:
         status = UMH_Transfer(gBulkOutHandle, NULL, Buffer,
                            gStorageInfo->BUFFER_SIZE);
         break;

         case MASS_STORAGE_REQUEST_SENSE:
         case MODE_SENSE:
         case READ_MASS_STORAGE_CAPACITY:
         case INQUIRY_MASS_STORAGE:
         case READ_MASS_STORAGE:
         status = UMH_Transfer(gBulkInHandle, NULL, Buffer,
                            gStorageInfo->BUFFER_SIZE);
         break;
       }
       return status;
   }

   DBGOUT(DBG_ERR,DbgPrint("No In or Out Handle found\n"));
   return UMH_STATUS_ERROR;

}
```

```
const void* GetNextDescriptor(const void* Desc)
{
   UCHAR bLength=*(UCHAR*)Desc;
   return ((UCHAR*)Desc)+bLength;
}

// Find InterfaceDescriptor search from the start of the configuration
// descriptor after a interface with the parameter Interfacenumber.
// params:
// Configuration: pointer to the start of the configuration descriptor
// Length:        IN: length of configuration descriptor
//                OUT:remaining byte length from the beginning of the
//                interfacedescriptor to the end of the configuration data
// Interfacenumber: interface number from 0 to n, this is not the
//                  interface index
// **Interface:     IN: - OUT: pointer to the Interface NULL if an error ocurred
// Return:
// UMH_STATUS_SUCCESS: no error
// UMH_STATUS_LENGTH:  length error
//
int FindInterfaceDescriptor(const UCHAR* Configuration,UINT * Length,
                            UCHAR Interfacenumber, const UCHAR **Interface)
{
  USHORT length = *Length;
  const UCHAR * desc=Configuration;
  const UCHAR *prevDesc;
  const USB_CONFIGURATION_DESCRIPTOR* conf;
  int InterfaceCt;

  *Interface = NULL;
  // check if this is a configuration descriptor
  conf = (USB_CONFIGURATION_DESCRIPTOR*)Configuration;

  if( conf->wTotalLength != *Length )
  {
    return UMH_STATUS_ERROR;
  }
  if( conf->bDescriptorType != USB_CONFIGURATION_DESCRIPTOR_TYPE)
  {
    return UMH_STATUS_INVALID_PARAM;
  }
  InterfaceCt=conf->bNumInterfaces;

  length -= *desc;  // search from the next desc.
  while(length)
  {
    prevDesc=desc;
    desc=GetNextDescriptor(prevDesc);
    length -= desc - prevDesc;

    if (desc==NULL)
    {
      return UMH_STATUS_ERROR;
    }
    if (*(desc +1) == USB_INTERFACE_DESCRIPTOR_TYPE )
    {
      if (*(desc + 2) == Interfacenumber )
      {
        *Interface=desc; // return the right Interface
        *Length=length;  // return the remaining length inclusive the length of
                         //   the interface desc.
        return UMH_STATUS_SUCCESS;
      }
    }
```

```
  }
  return UMH_STATUS_ERROR;
}

// USBStickDetection checks if the device is a USBstick with a specific class-
// code, sub-class code and protocol
int USBStickDetection (UCHAR *DataEndpointINAddress,
  UCHAR *DataEndpointOUTAddress, UINT *MaxINPacketSize, UINT *MaxOUTPacketSize)
{
  int status;
  UCHAR *buffer;
  const UCHAR *iface;
  const USB_ENDPOINT_DESCRIPTOR *epdesc;
  const USB_INTERFACE_DESCRIPTOR *idesc;
  SETUP_DATA setup;
  UINT remainingLength;
  ZERO_STRUCT(setup);

  buffer=gTempBuffer;
  setup.Bytes.Type= USB_IN_DIRECTION;         // get interface des
  setup.Bytes.Request=USB_REQ_GET_DESCRIPTOR;
  setup.Bytes.ValueH=USB_CONFIGURATION_DESCRIPTOR_TYPE;
  setup.Words.Length=sizeof(gTempBuffer);
  gSetupCompleteFlag=FALSE;
  status=UMH_SetupRequest(
    (UCHAR*)&setup,buffer,&remainingLength); // SetupCompletion
  if( status)
  {
    DBGOUT(DBG_ERR,DbgPrint("Error Setup-Request:%d!\n",status));
    return status;
  }
  // search interface with the number zero
  status=FindInterfaceDescriptor(buffer,&remainingLength,0,&iface);
  if( status)
  {
    DBGOUT(DBG_ERR,DbgPrint("Error FindInterfaceDescriptor:%d!\n",status));
    return status;
  }
  // check for mass storage class, subclass and interface protocol
  idesc=(USB_INTERFACE_DESCRIPTOR*)iface;
  if( idesc->bInterfaceClass != DEMO_DEVICE_INTERFACE_CLASS
    || idesc->bInterfaceSubClass != DEMO_DEVICE_INTERFACE_SUB_CLASS
    || idesc->bInterfaceProtocol != DEMO_DEVICE_INTERFACE_PROTOCOL
    )
    {
      DBGOUT(DBG_ERR,DbgPrint("Error Status invalid parameter.\n"));
      return UMH_STATUS_INVALID_PARAM;
  }
  // get the right endpoint number remaining length is inclusive the
  // interface desc. length
  epdesc=(USB_ENDPOINT_DESCRIPTOR*)iface;
  while( remainingLength)
  {
    epdesc = (USB_ENDPOINT_DESCRIPTOR*)GetNextDescriptor(epdesc);
    remainingLength -= epdesc->bLength;
    if ( epdesc->bDescriptorType == USB_ENDPOINT_DESCRIPTOR_TYPE )
    {
       // To check if it is an In or Out Endpoint
       if ((epdesc->bEndpointAddress & USB_IN_DIRECTION)  != 0 )
       {
              *DataEndpointINAddress = epdesc->bEndpointAddress;
              *MaxINPacketSize = epdesc->wMaxPacketSize;
       }
```

```
      else
      {
            *DataEndpointOUTAddress = epdesc->bEndpointAddress;
            *MaxOUTPacketSize = epdesc->wMaxPacketSize;
      }
   }

   epdesc = (USB_ENDPOINT_DESCRIPTOR*)GetNextDescriptor(epdesc);
   remainingLength -= epdesc->bLength;

   if ( epdesc->bDescriptorType==USB_ENDPOINT_DESCRIPTOR_TYPE )
   {

       // To check if it is an In or Out Endpoint
      if ((epdesc->bEndpointAddress & USB_IN_DIRECTION)  != 0 )
      {
            *DataEndpointINAddress =epdesc->bEndpointAddress;
            *MaxINPacketSize = epdesc->wMaxPacketSize;
      }
      else
      {
            *DataEndpointOUTAddress = epdesc->bEndpointAddress;
            *MaxOUTPacketSize = epdesc->wMaxPacketSize;
      }
   }

   //For mass storage devices it is usual to have 2 bulk Endpoints, it
   //is not necessary having more then these 2. Some USB FlashDrives
   //have in additional to the 2 bulk endpoints one interrupt in Endpoint.

   if(remainingLength == 0)
   {
      return UMH_STATUS_SUCCESS;
   }

   epdesc = (USB_ENDPOINT_DESCRIPTOR*)GetNextDescriptor(epdesc);
   remainingLength -= epdesc->bLength;

   if(epdesc->bEndpointAddress != 0x83) //if not interrupt in endpoint
   {
      DBGOUT(DBG_ERR,DbgPrint("Unknown Endpoint detected!"));
      return UMH_STATUS_ERROR;
   }
  }
  return UMH_STATUS_SUCCESS;
}

//  write a sequencenumber (two bytes) at the beginning of the buffer,
//  if the length is greater than MaxPacketSize then write
//  the sequencenumber at the start of every packet in the buffer

void UpdateSequenceNumber(unsigned char *buffer, unsigned short length,
                  unsigned short MaxPacketSize, unsigned short *SequenceNumber)
{
   int i;
   for(i = 0; i < length; i += MaxPacketSize)
   {
      *(unsigned short*)(buffer+i)=*SequenceNumber;
      *SequenceNumber+=1;
   }
}
```

```
// Fill a buffer with 0 to max. packet size
void InitBuffer(unsigned char *Buffer, unsigned short Length,
                unsigned short MaxPacketSize)
{
   unsigned char val;
   unsigned short ct=0;

   while( Length--)
   {
      if( (ct%MaxPacketSize) == 0)
      {
         val=0;
      }
      *Buffer++=val++;
      ct++;
   }
}

void StateCallBack( unsigned int Flags )
{
  switch( Flags )
  {
    case  UMH_EVENT_REMOVED:
      DBGOUT(DBG_ERR,DbgPrint("remove\n"));
      // if the deivce is removed then stop the transfer and wait for connection
      gConnectFlag=FALSE;
      break;
    case  UMH_EVENT_CONNECT:
      DBGOUT(DBG_ERR,DbgPrint("connect\n"));
      // after connect enumerate the device
      gConnectFlag=TRUE;
      break;
    case  UMH_EVENT_WAKEUP:
      // device now ready to transfer data if the device state is configured
      DBGOUT(DBG_ERR,DbgPrint("wakeup\n"));
      gWakeUpFlag=TRUE;
      break;
    default:
      DBGOUT(DBG_ERR,DbgPrint("ERROR StateCallBack:Invalid state\n"));
      break;
  }
}

// SetFeatureRemoteWakeUp is a sample for use of remote wakeup devices
// This function enables the remote wakeup capabality if the device supports
// the remote wakeup

int SetFeatureRemoteWakeUp(void)
{
  SETUP_DATA setup;
  UCHAR * buffer;
  int status;

  ZERO_STRUCT(setup);
  buffer=gTempBuffer;
  gSetupCompleteFlag=FALSE;
  setup.Bytes.Request=USB_REQ_SET_FEATURE;
  setup.Words.Value=FEATURE_REMOTE_WAKEUP;
  status=UMH_SetupRequest( (UCHAR*)&setup,NULL,NULL ); // SetupCompletion
  return status;
}
```

```
// max. execution time is 100us
void Idle(void)
{
  return;
}


// this function is called from the library to wait
// a specified time. This function must be implemented by the user
// with the fixed function name UmhStallExecutionCallback.
// Return: nothing
// parameter: time in 0.1 milliseconds units

void UMH_StallExecutionUserCallback(unsigned long time)
{
  unsigned long startTime=GetCurrentTime();

  while(1)
  {
    if (IsTimeOver(time, startTime))
      break;
    Idle();
  }
}
```

7.7 usb_functions.h

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*                 (C) Fujitsu Microelectronics Europe GmbH                  */
/*---------------------------------------------------------------------------*/

#ifndef _USB_FUNCTIONS_H_
#define _USB_FUNCTIONS_H_

// prototypes
void initglobals(void);
__interrupt void IRQ_PWCTimer(void);
void InitPWCTimer(void);
int ResetRecovery(void);
int ClearOutEndpoint(void);
int ClearInEndpoint(void);
void HardReset(void);
void GetEPStatus(unsigned char *InEpStatus, unsigned char *OutEpStatus);
void BulkOutCompletion(UMH_STATUS Status, unsigned int BytesTransferred,
                       void* Context);
void BulkInCompletion(UMH_STATUS Status, unsigned int BytesTransferred,
                      void* Context);
int Transmit_MSDC_Data(unsigned char *Buffer, STORAGE_INFO_t *gStorageInfo);
void SetupCompletion(UMH_STATUS Status, unsigned int BytesTransferred,
                     void* Context);
const void* GetNextDescriptor(const void* Desc);
int FindInterfaceDescriptor(const UCHAR* Configuration, UINT * Length,
                            UCHAR Interfacenumber, const UCHAR **Interface);
int USBStickDetection (UCHAR *DataEndpointINAddress,
                       UCHAR *DataEndpointOUTAddress,
                       UINT *MaxINPacketSize, UINT *MaxOUTPacketSize);
void UpdateSequenceNumber(unsigned char *buffer, unsigned short length,
                          unsigned short MaxPacketSize,
                          unsigned short *SequenceNumber);
void InitBuffer(unsigned char *Buffer,
                unsigned short Length, unsigned short MaxPacketSize);
void StateCallBack(unsigned int Flags);
int SetFeatureRemoteWakeUp(void);
unsigned long GetCurrentTime(void);
unsigned char IsTimeOver(unsigned long Time, unsigned long StartTime);
void Idle(void);
void UMH_StallExecutionUserCallback(unsigned long time);

#endif // _USB_FUNCTIONS_H_
```

7.8 storage.c

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*                 (C) Fujitsu Microelectronics Europe GmbH                  */
/*---------------------------------------------------------------------------*/

#include <stdlib.h>
#include "mb96338us.h"

#include "usbspec.h"
#include "usbminihost.h"
#include "osal.h"
#include "config.h"

#include "storage.h"
#include "diskio.h"
#include "storage_API.h"
#include "usb_functions.h"

#include "app_global.h"

/*---------------------------------------------------------------------------*/
/* variables                                                                 */
/*---------------------------------------------------------------------------*/

extern UMH_HANDLE gBulkOutHandle, gBulkInHandle;
extern unsigned long gTimer2;

READ_CAPA ReadCapacity;
CSW_t csw;
CBW_t cbw;

UCHAR gTransferDone=0, gInTransferDone=0, gOutTransferDone=0;

void ErrorHandler(unsigned int errCode, unsigned char *clrEponly )
{
 unsigned char InEpStatus=0, OutEpStatus=0;

 switch(errCode)
 {
   case UMH_STATUS_STALL:
   if (*clrEponly)
   {
      GetEPStatus(&InEpStatus, &OutEpStatus);
      if(InEpStatus)
      {
         ClearInEndpoint();
      }
      if (OutEpStatus)
      {
         ClearOutEndpoint();
      }
      *clrEponly=0;
      break;
   }
   ResetRecovery();
   break;

   default:
   HardReset();
   break;
   }
}
```

```
void CheckForCompletion(unsigned char *NAKDetected)
{
   gTimer2=0;
   do
   {
      if (gTimer2 >=90000) // if there is no Handshake 9seconds long,
                           //   something went wrong!
      {
         gOutTransferDone=0;
         gInTransferDone=0;
         *NAKDetected=1;   // There must be a circular NAK Handshake
      }
   }while(gOutTransferDone | gInTransferDone); // Wait for status information
                                               //   of previous transmission
}

// transfer done is called from the transfer completion function
void InTransferDone(void)
{
  gInTransferDone = 0;     // gInTransferDone = 0 means: transmission ended
}

void OutTransferDone(void)
{
  gOutTransferDone = 0;
}

int StorageCommandSend(STORAGE_INFO_t *gStorageInfo)
{
   unsigned int status;
   unsigned char NAKDetected=0;
   unsigned char clrEponly=0;
   unsigned int cnt=0;

   CheckForCompletion(&NAKDetected);
   gOutTransferDone = 1;               // Block transfer
   status = Send_CBW(gStorageInfo);
   if (status)
   {
      return status;
   }
   CheckForCompletion(&NAKDetected);
   if (gStorageInfo->STATUS | NAKDetected)
   {
      ErrorHandler(gStorageInfo->STATUS, &clrEponly );
      return UMH_STATUS_ERROR;
   }

   if(cbw.dCBWDataTransferLength != 0) // Jump in, if there is a data Transfer
   {
      if(gStorageInfo->STORAGE_CMD == WRITE_MASS_STORAGE)
      {
        gOutTransferDone = 1;
      }
      else
      {
        gInTransferDone = 1;
      }
      status = Transmit_Data(gStorageInfo);
      if (status)
      {
         return status;
      }
```

```
      CheckForCompletion(&NAKDetected);
      if(gStorageInfo->STATUS | NAKDetected)
      {
         clrEponly=1;
         ErrorHandler(gStorageInfo->STATUS, &clrEponly) ;
         NAKDetected=0;
      }
   }
   gInTransferDone = 1;
   status = Get_CSW(gStorageInfo);
   if (status)
   {
      return status;
   }

   CheckForCompletion(&NAKDetected);
   if (gStorageInfo->STATUS | NAKDetected)
   {
      clrEponly = 1;
      ErrorHandler(gStorageInfo->STATUS, &clrEponly);
      NAKDetected=0;
      status = Get_CSW(gStorageInfo);
      if (status)
      {
         return status;
      }
      CheckForCompletion(&NAKDetected);
      if (gStorageInfo->STATUS | NAKDetected)
      {
          ErrorHandler(gStorageInfo->STATUS, &clrEponly);
          return UMH_STATUS_ERROR;
      }
   }
   status = CheckCSW(gStorageInfo);
   return status;
}

int Send_CBW(STORAGE_INFO_t *gStorageInfo)
{
   unsigned int status;
   ZERO_STRUCT(cbw);
   gStorageInfo->SEND_CBW = 1;
   gStorageInfo->BUFFER_SIZE = 31; // CBW Size must always be 31 byte

   cbw.dCBWSignature = 0x43425355;
   cbw.dCBWTag = CBW_TAG;
   cbw.bCBWLUN = 0x00;
   cbw.CBWCB[1] = 0x00;
   cbw.CBWCB[2] = (0xff & (gStorageInfo->BLOCK_START >> 24));
   cbw.CBWCB[3] = (0xff & (gStorageInfo->BLOCK_START >> 16));
   cbw.CBWCB[4] = (0xff & (gStorageInfo->BLOCK_START >> 8));
   cbw.CBWCB[5] = (0xff & (gStorageInfo->BLOCK_START));
   cbw.CBWCB[6] = 0x00;
   cbw.CBWCB[7] = (UCHAR)(((USHORT)(gStorageInfo->BLOCK_CNT) >> 8) & 0xff);
   cbw.CBWCB[8] = (UCHAR)(((USHORT)(gStorageInfo->BLOCK_CNT) & 0xff));
   cbw.CBWCB[9] = 0x00;

   switch(gStorageInfo->STORAGE_CMD)
   {
      case READ_MASS_STORAGE:
      cbw.CBWCB[0] = gStorageInfo->STORAGE_CMD;
      cbw.dCBWDataTransferLength = 0x00000200;
      cbw.bCBWFlags = 0x80;
      cbw.bCBWCBLength = 0x0A;
      break;
```

```
        case WRITE_MASS_STORAGE:
        cbw.CBWCB[0] = gStorageInfo->STORAGE_CMD;
        cbw.dCBWDataTransferLength = 0x00000200;
        cbw.bCBWFlags = 0x00;
        cbw.bCBWCBLength = 0x0A;
        break;

        case INQUIRY_MASS_STORAGE:
        cbw.CBWCB[0] = gStorageInfo->STORAGE_CMD;
        cbw.dCBWDataTransferLength = 0x00000024;
        cbw.bCBWFlags = 0x80;
        cbw.bCBWCBLength = 0x06;
        cbw.CBWCB[4] = 0x24;
        break;

        case READ_MASS_STORAGE_CAPACITY:
        cbw.CBWCB[0] = gStorageInfo->STORAGE_CMD;
        cbw.dCBWDataTransferLength = 0x00000008;
        cbw.bCBWFlags = 0x80;
        cbw.bCBWCBLength = 0x0A ;
        break;

        case TEST_UNIT_READY:
        cbw.CBWCB[0] = gStorageInfo->STORAGE_CMD;
        cbw.dCBWDataTransferLength = 0x00000000;
        cbw.bCBWFlags = 0x00;
        cbw.bCBWCBLength = 0x06;
        break;

        case MASS_STORAGE_REQUEST_SENSE:
        cbw.CBWCB[0] = gStorageInfo->STORAGE_CMD;
        cbw.dCBWDataTransferLength = 0x00000012;
        cbw.bCBWFlags = 0x80;
        cbw.bCBWCBLength = 0x0C;
        cbw.CBWCB[4] = 0x12;
        break;

        case SEND_DIAGNOSTIC:
        cbw.CBWCB[0] = gStorageInfo->STORAGE_CMD;
        cbw.CBWCB[1] = 0x04;
        cbw.dCBWDataTransferLength = 0x00000000;
        cbw.bCBWCBLength = 0x0A;

        case MODE_SENSE:
        cbw.CBWCB[0] = gStorageInfo->STORAGE_CMD;
        cbw.dCBWDataTransferLength = 0x00000004;
        cbw.bCBWFlags = 0x80;
        cbw.bCBWCBLength = 0x06;
        cbw.CBWCB[2] = 0x3F;
        cbw.CBWCB[4] = 0x04;
    }

    status = Transmit_MSDC_Data((unsigned char *) &cbw, gStorageInfo);
    gStorageInfo->SEND_CBW = 0;
    return status;

}

int Transmit_Data(STORAGE_INFO_t *gStorageInfo)
{
    unsigned int status;
    gStorageInfo->BUFFER_SIZE = cbw.dCBWDataTransferLength;
    status = Transmit_MSDC_Data(gStorageInfo->BUFFER, gStorageInfo);
    return status;
}
```

```
int Get_CSW(STORAGE_INFO_t *gStorageInfo)
{
   unsigned int status;
   ZERO_STRUCT(csw);
   gStorageInfo->GET_CSW = 1;
   gStorageInfo->BUFFER_SIZE = 13; // CBW Size must always be 13 byte
   status = Transmit_MSDC_Data((unsigned char *) &csw, gStorageInfo);
   gStorageInfo->GET_CSW = 0;
   return status;
}

int CheckCSW(STORAGE_INFO_t *gStorageInfo)
{  // Check if CSW is valid
   unsigned char clrEponly=0;
   if ((csw.CSWTag != CBW_TAG) || (csw.CSWSignatur != 0x53425355)
       ||(gStorageInfo->BYTES_TRANSFERRED != 13 ))
   {
       ErrorHandler((unsigned int) NULL, &clrEponly);
       return UMH_STATUS_ERROR;
   }  // Check if CSW is meaningful
   if ((csw.CSWStatus == 2) || (csw.CSWDataResidue >>
       cbw.dCBWDataTransferLength))
   {
      DBGOUT(DBG_INFO,DbgPrint("Storage data transmission failed\n"));
      ErrorHandler((unsigned int) NULL, &clrEponly);
      return UMH_STATUS_ERROR;
   }
   if (csw.CSWStatus == 1)
   {
      gStorageInfo->CSW_STATUS = 1;
   }
   return UMH_STATUS_SUCCESS;
}

int BulkOnlyMSReset(void)
{
  int status;
  SETUP_DATA setup;
  ZERO_STRUCT(setup);
  setup.Bytes.Type=0x21;
  setup.Bytes.Request=0xFF;
  setup.Words.Value=0;
  setup.Words.Index=0;
  setup.Words.Length=0;

  status=UMH_SetupRequest( (UCHAR*)&setup, NULL, NULL );
  return status;
}

int StorageStatus(void)
{
   unsigned char caller;
   unsigned int flag;

   UMH_GetDeviceState((enum UMH_DEVICE_STATE*)&caller, &flag);
   if (caller == UMH_DEVICE_CONFIGURED)
   {
      DBGOUT(DBG_INFO,DbgPrint("O.K.!\n", caller));
      return 0;
   }
   else
   {
      DBGOUT(DBG_INFO,DbgPrint("Not O.K.!\n", caller));
      return 1;
   }
}
```

```
int TestUnitReady(STORAGE_INFO_t *gStorageInfo)
{
   unsigned int status;
   ZERO_STRUCT(gStorageInfo);
   gStorageInfo->STORAGE_CMD = TEST_UNIT_READY;

   status = Send_CBW(gStorageInfo);
   UMH_StallExecutionUserCallback(200);
   Get_CSW(gStorageInfo);

   return status;
}
```

7.9 storage.h

```
/*-----------------------------------------------------------------------------*/
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*                 (C) Fujitsu Microelectronics Europe GmbH                  */
/*-----------------------------------------------------------------------------*/

#ifndef _STORAGE_H_
#define _STORAGE_H_

#include "usbminihost.h"
#include "config.h"
#include "diskio.h"

// Structs

typedef struct _STORAGE_CBW
{
   ULONG dCBWSignature;
   ULONG dCBWTag;
   ULONG dCBWDataTransferLength;
   UCHAR bCBWFlags;
   UCHAR bCBWLUN;
   UCHAR bCBWCBLength;
   UCHAR CBWCB[16];
}CBW_t;

typedef struct _STORAGE_CSW
{
   ULONG CSWSignatur;
   ULONG CSWTag;
   ULONG CSWDataResidue;
   UCHAR CSWStatus;
}CSW_t;

// Response struct
typedef struct _INQUIRY_RESPONSE
{
   UCHAR Peripheral;
   UCHAR Removable;
   UCHAR Version;
   UCHAR ResponseDataFormat;
   UCHAR AdditionalLength;
   UCHAR Sccstp;
   UCHAR Bqueetc;
   UCHAR CmdQue;
   UCHAR VendorID[8];
   UCHAR ProductID[16];
   UCHAR ProductRev[4];
}InquiryResponse_t;


// 8 byte capacity value buffer
typedef struct _READ_CAPACITY
{
   ULONG SectorCount;
   ULONG SectorSize;
} READ_CAPA;
```

```
// Functions
typedef struct _RequestSenseResponse
{
   UINT ResponseCode :7;
   UINT VALID :1;
   UCHAR Obsolete;
   UINT SenseKey :4;
   UINT Resv :1;
   UINT ILI :1;
   UINT EOM :1;
   UINT FILEMARK :1;
   ULONG Information;
   UCHAR AddSenseLen;
   ULONG CmdSpecificInfo;
   UCHAR ASC;
   UCHAR ASCQ;
   UCHAR FRUC;
   UCHAR SenseKeySpecific[3];
}RequestSenseResponse_t;


// Struct for read data
typedef struct _Storage_Read
{
   UCHAR data[512];
}ReadWrite_t;


// Struct for write data
typedef struct _Storage_Write
{
   UCHAR data[10];
}Write_t;


// Prototypes
int StorageInquiry(void);
int StorageReadCapa(void (*Buff));
int StorageReady(void);
int StorageVerify(void);
int RequestSense(void);
int StorageInit(void);
void TransferDone(void);

// Read Write functions
int USB_disk_read(BYTE *buff, DWORD sector, BYTE count);
int USB_disk_write(BYTE *buff, DWORD sector, BYTE count);

#endif // _STORAGE_H_
```

7.10 storage_API.c

```
/*-----------------------------------------------------------------------------*/
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*                 (C) Fujitsu Microelectronics Europe GmbH                  */
/*-----------------------------------------------------------------------------*/

#include <stdlib.h>
#include "mb96338us.h"

#include "usbspec.h"
#include "usbminihost.h"
#include "osal.h"
#include "umh_dbgprint.h"

#include "storage_API.h"

#include "diskio.h"
#include "msc_config.h"

STORAGE_INFO_t gStorageInfo;

int MSD_Read(BYTE *buff, DWORD sector, BYTE count)
{
   unsigned int status;
   ZERO_STRUCT(gStorageInfo);
   gStorageInfo.STORAGE_CMD = READ_MASS_STORAGE;
   gStorageInfo.BUFFER = buff;
   gStorageInfo.BLOCK_START = sector;
   gStorageInfo.BLOCK_CNT = count;

   status = StorageCommandSend(&gStorageInfo);

   if (gStorageInfo.CSW_STATUS == 1)
   {
      struct RequestSenseResponse SenseResponse;
      ZERO_STRUCT(gStorageInfo);
      ZERO_STRUCT(SenseResponse);
      gStorageInfo.STORAGE_CMD = MASS_STORAGE_REQUEST_SENSE;
      gStorageInfo.BUFFER =(unsigned char *) &SenseResponse;
      status = StorageCommandSend(&gStorageInfo);

      if (status)
      {
         return status;
      }
      status = SenseResponse.SenseKey;
      return status;
   }
   return status;
}

int MSD_Write(BYTE *buff, DWORD sector, BYTE count)
{
   unsigned int status;
   ZERO_STRUCT(gStorageInfo);
   gStorageInfo.STORAGE_CMD = WRITE_MASS_STORAGE;
   gStorageInfo.BUFFER = buff;
   gStorageInfo.BLOCK_START = sector;
   gStorageInfo.BLOCK_CNT = count;

   status = StorageCommandSend(&gStorageInfo);

   if (gStorageInfo.CSW_STATUS == 1)
```

```
    {
        struct RequestSenseResponse SenseResponse;
        ZERO_STRUCT(gStorageInfo);
        ZERO_STRUCT(SenseResponse);
        gStorageInfo.STORAGE_CMD = MASS_STORAGE_REQUEST_SENSE;
        gStorageInfo.BUFFER = (unsigned char *) &SenseResponse;
        status = StorageCommandSend(&gStorageInfo);

        if (status)
        {
            return status;
        }

        if (SenseResponse.SenseKey == 0x7)
        {
            return 0x02;  // Write protect
        }

        status = SenseResponse.SenseKey;
        return status;
    }

    return status;
}

int MSD_Read_Capacity(void (*Buff))
{
    unsigned int status;
    ZERO_STRUCT(gStorageInfo);
    gStorageInfo.STORAGE_CMD = READ_MASS_STORAGE_CAPACITY;
    gStorageInfo.BUFFER = Buff;

    status = StorageCommandSend(&gStorageInfo);

    if (gStorageInfo.CSW_STATUS == 1)
    {
        struct RequestSenseResponse SenseResponse;
        ZERO_STRUCT(gStorageInfo);
        ZERO_STRUCT(SenseResponse);
        gStorageInfo.STORAGE_CMD = MASS_STORAGE_REQUEST_SENSE;
        gStorageInfo.BUFFER = (unsigned char *)&SenseResponse;
        status = StorageCommandSend(&gStorageInfo);

        if (status)
        {
            return status;
        }
        status = SenseResponse.SenseKey;
        return status;
    }
    return status;
}


int StorageInit(void)
{
    unsigned char cnt = 0;
    unsigned char CheckAgain = 1;
    struct INQUIRY_RESPONSE InquiryResponse;
    unsigned int status = 0;
    ZERO_STRUCT(gStorageInfo);
    ZERO_STRUCT(InquiryResponse);
    gStorageInfo.STORAGE_CMD = INQUIRY_MASS_STORAGE;
    gStorageInfo.BUFFER = (unsigned char *)&InquiryResponse;

    status = StorageCommandSend(&gStorageInfo);
```

```
    if (status)
    {
        return status;
    }

    while (CheckAgain)
    {
        if (cnt >= 2)
        {
            return UMH_STATUS_ERROR;
        }

        ZERO_STRUCT(gStorageInfo);
        gStorageInfo.STORAGE_CMD = TEST_UNIT_READY;
        status = StorageCommandSend(&gStorageInfo);
        if (status)
        {
            return status;
        }
        if (gStorageInfo.CSW_STATUS == 1)
        {
            struct RequestSenseResponse SenseResponse;
            ZERO_STRUCT(gStorageInfo);
            ZERO_STRUCT(SenseResponse);
            gStorageInfo.STORAGE_CMD = MASS_STORAGE_REQUEST_SENSE;
            gStorageInfo.BUFFER = (unsigned char *)&SenseResponse;
            status = StorageCommandSend(&gStorageInfo);
            if (status)
            {
                return status;
            }
            if (SenseResponse.SenseKey != 0x06)
            {
                CheckAgain = 0;
            }
            cnt++;
            status = SenseResponse.SenseKey;
        }
        else
        {
            CheckAgain = 0;
        }
    } // end of while
    status = StorageStatus();
    return status;
}

int ModeSense(void)
{
    unsigned int status;
    struct ModeSenseResponse ModeSense;

    ZERO_STRUCT(gStorageInfo);
    ZERO_STRUCT(ModeSense);
    gStorageInfo.STORAGE_CMD = MODE_SENSE;
    gStorageInfo.BUFFER = (unsigned char *)&ModeSense;
    status = StorageCommandSend(&gStorageInfo);

    if (ModeSense.WriteProtect ==1)
    {
        return UMH_STATUS_ERROR;
    }
    return status;
}
```

7.11 storage_API.h

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*                 (C) Fujitsu Microelectronics Europe GmbH                  */
/*---------------------------------------------------------------------------*/

#ifndef _STORAGE_API_H_
#define _STORAGE_API_H_

#include "diskio.h"

// Definitions

// UFI Command Descriptors (SCSI)
#define FORMAT_UNIT                     0x04
#define INQUIRY_MASS_STORAGE            0x12
#define MODE_SELECT                     0x55
#define MODE_SENSE                      0x5A
#define PREVENT_ALLOW_MEDIUM_REMOVAL    0x1E
#define READ_MASS_STORAGE               0x28
#define READ_MASS_STORAGE_CAPACITY      0x25
#define READ_FORMAT_CAPACITIES          0x23
#define MASS_STORAGE_REQUEST_SENSE      0x03
#define REZERO                          0x01
#define SEEK10                          0x2B
#define SEND_DIAGNOSTIC                 0x1D
#define START_STOP_UNIT                 0x1B
#define TEST_UNIT_READY                 0x00
#define VERIFY                          0x2F
#define WRITE_MASS_STORAGE              0x2A
#define WRITE12                         0xAA
#define WRITE_AND_VERIFY                0x2E

#define CBW_TAG                         0x1A2B3C4D


// Structs

typedef struct _StorageInfo_t
{
   unsigned char STORAGE_CMD;
   unsigned int  BUFFER_SIZE;
   unsigned char *BUFFER;
   unsigned int  IN_EP;
   unsigned int  OUT_EP;
   unsigned long BLOCK_START;
   unsigned char BLOCK_CNT;
   unsigned int  STATUS;
   unsigned char SEND_CBW;
   unsigned char GET_CSW;
   unsigned char CSW_STATUS;
   unsigned int  BYTES_TRANSFERRED;
   unsigned int  ERROR_CODE;
   unsigned char MAX_LUN;
}STORAGE_INFO_t;

struct INQUIRY_RESPONSE
{
   UCHAR Peripheral;
   UCHAR Removable;
   UCHAR Version;
   UCHAR ResponseDataFormat;
   UCHAR AdditionalLength;
```

```
    UCHAR Sccstp;
    UCHAR Bqueetc;
    UCHAR CmdQue;
    UCHAR VendorID[8];
    UCHAR ProductID[16];
    UCHAR ProductRev[4];
};

struct RequestSenseResponse
{
    UCHAR ResponseCode :7;
    UCHAR VALID :1;
    UCHAR Obsolete;
    UCHAR SenseKey :4;
    UCHAR Resv :1;
    UCHAR ILI :1;
    UCHAR EOM :1;
    UCHAR FILEMARK :1;
    ULONG Information;
    UCHAR AddSenseLen;
    ULONG CmdSpecificInfo;
    UCHAR ASC;
    UCHAR ASCQ;
    UCHAR FRUC;
    UCHAR SenseKeySpecific[3];
};

struct ModeSenseResponse
{
    UCHAR ModeDataLength;
    UCHAR MediumType;
    UCHAR Reserved1     :4;
    UCHAR DPOFUA        :1;
    UCHAR Reserved2     :2;
    UCHAR WriteProtect  :1;
    UCHAR BlockDescriptorLength;
};

int MSD_Read(BYTE *buff, DWORD BlockStart, BYTE BlockCnt);
int MSD_write(BYTE *buff, DWORD BlockStart, BYTE BlockCnt);
int MSD_Read_Capacity(void (*Buff));
int StorageInit(void);
int ModeSense(void);

#endif // _STORAGE_API_H_
```

7.12 msc_config.h

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*                 (C) Fujitsu Microelectronics Europe GmbH                  */
/*---------------------------------------------------------------------------*/

#ifndef __MSC_CONFIG_H__
#define __MSC_CONFIG_H__

/*************************************************************/
/*            Main specific definitions                      */
/*************************************************************/

/* fill memory */
#define FILL_MEMORY(mem,count,val)   OSALmemset((mem),(val),(count))

/* zero memory */
#define ZERO_MEMORY(mem,count)       OSALmemset((mem),0,(count))

/* zero local structure */
#define ZERO_STRUCT(s)               ZERO_MEMORY(&(s),sizeof(s))

/* zero whole array */
#define ZERO_ARRAY(s)                ZERO_MEMORY((s),sizeof(s))

/* compiler specific */
#define  BOOLEAN UCHAR

#ifdef yyyy
/* only to see all valid debug masks  */
DBG_ERR       0   /* errors*/
DBG_WARN      1   /* warnings */
DBG_FUNC      2   /* function names (entries with exits in some functions ) */
DBG_EP0       3   /* endpoint 0 setup request informations */
DBG_EP        4   /* all other endpoint informations */
DBG_INFO      5   /* not so important*/
DBG_DUMP_EP0  6   /* memory dumps from SETUP and vendor requests */
DBG_DUMP_EP   7   /* memory dumps from data endpoints */
DBG_SPECIAL   13

/* valid user debug traces */
USER_ERR  8
USER_WARN 9
USER_INFO 10
USER_FUNC 11
USER_DUMP 12
USER_SPECIAL 14
#endif

/*  define a new mask to see the interesting things in the usb debug library
    files and in the user program
*/

#define TRACE_LIB_ENABLE  1
#define TRACE_USER_ENABLE 1
#define TRACE_ONLY_LIB_ERRORS 0
#define TRACE_ONLY_USER_ERRORS 0

// dbg special is for handshake flags
#define LIB_DBG_MASK2      1<<DBG_ERR | 1<<DBG_WARN   // | 1<<DBG_SPECIAL
#define USER_DBG_MASK2     1<<USER_ERR | 1<<USER_WARN // | 1<<USER_SPECIAL
```

```
#if !TRACE_ONLY_LIB_ERRORS
#define LIB_DBG_MASK       LIB_DBG_MASK2 | 1<<DBG_WARN |  1<<DBG_INFO
             //| 1<<DBG_EP0 | 1<<DBG_EP |  1<<DBG_DUMP_EP0 | 1<<DBG_FUNC
#else
#define LIB_DBG_MASK       LIB_DBG_MASK2
#endif
#if !TRACE_ONLY_USER_ERRORS
#define USER_DBG_MASK1     USER_DBG_MASK2 | 1<<USER_INFO | 1<<USER_DUMP
             //| 1<<USER_FUNC
#else
#define USER_DBG_MASK1     USER_DBG_MASK2
#endif

#if (TRACE_LIB_ENABLE && TRACE_USER_ENABLE)
#define USER_DBG_MASK USER_DBG_MASK1 | LIB_DBG_MASK
#else
#if TRACE_LIB_ENABLE
#define USER_DBG_MASK LIB_DBG_MASK
#else
#if TRACE_USER_ENABLE
#define USER_DBG_MASK USER_DBG_MASK1
#else
#define USER_DBG_MASK 0
#endif
#endif
#endif

#define USR_PFX "USR:"  /* präfix during debug sump */
#endif
```

8 Additional Information

8.1 Internal Links

Information about FUJITSU Microcontrollers can be found on the following Internet page:
http://mcu.emea.fujitsu.com/

The software example related to this application note is:
96330_usb_mass_storage_demo
It can be found on the following Internet page:
http://mcu.emea.fujitsu.com/mcu_product/mcu_all_software.htm

8.2 External Links

Thesycon™ driver for USB mini-host:
http://www.thesycon.de/eng/fuma.shtml

Free of charge FAT16 file system:
http://elm-chan.org/fsw/ff/00index_e.html

Mass Storage Link (SCSI commands):
http://en.wikipedia.org/wiki/SCSI_commands

FAT16 description:
http://en.wikipedia.org/wiki/FAT16#File_Allocation_Table

