Mega MP3 Player

Brian Tucker

Sam Roundy

Mike Ballard

Mega MP3 Player

Brian Tucker, Sam Roundy, Mike Ballard

University of Utah

10.December.2004

Abstract

As technology races forward and digital toys and gadgets flood the market, MP3 players are becoming more and more popular. One reason for this is the desire to store the most amount of music possible on a limited amount of disk space. MP3's are compressed music files that take up a 10th of the disk space that normal audio files consume. Although home systems can store infinite amounts of data it is complicated to have much more than 100 GB of storage for portable use. Since the technology does not exist ,to our knowledge, to reduce the size of music files the Mega MP3 Player will employ 80+Gb of disk space for automobiles giving the user more than enough space to store and play their music collections.

1. Introduction

1.1 Background Research

1.1.1 MP3File Format

For simplicity purposes MPEG Layer I/II format 3 (MP3) will be the only file format type that the Mega MP3 Player will support. The purpose of the Mega MP3 Player is to store as many songs as possible. MP3 file format provides the smallest audio file format on the market. Thus we have decided to only use MP3 file format for our player. As an addition to our MP3 player we will tack on a device number to each MP3 file so our player will be able to have a unique way of distinguishing all the files in the system.

1.1.2 MP3 Players

We had originally planned to develop an MP3 player for an automobile. This was scratched when we realized there was too much involved to get it to work in the time allotted and also that we went with a development kit. So we eventually just built a normal MP3 player that reads from a typical CP HDD.

1.2 Purpose/Design Goals

1.2.1 Store +80G Files

Our player has the capability to play any size of HDD on the market, from 300MB up to 300GB. It all depends on how much storage the user wants and how much they can afford.

1.2.2 Playlist

Our player supports Playlists. The user sets up the playlists from their computer and transfers the playlists to the MP3 player just as they would normal files.

2. User Interface

2.1 Inputs

2.1.1 Keypad

The keypad allows for controlling the song playing on the MP3 player as well as the ability to manage and edit the songs on any of the different sources. There are three different music storage devices which the keypad input will control. They are: MMC Flash, Onboard Flash and IDE/HDD.

The development board came with basic play functionality for the Onboard Flash mode and we added play functionality for the other 2 modes as well as the ability to create play lists.

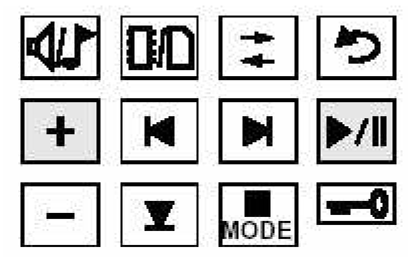


Figure 2.1 – Keypad Assignment

Figure 2.1 above shows the layout of the keypad buttons on the development board. The mapping of the buttons is the same for each source mode and is shown in Table 2.1 below:

Table 2.1 Keypad Button Mapping

🔊	Sound Control - Change Volume, Bass & Treble
💾	Memory Control - Choose memory source; Onboard, MMC or IDE
⏭	Next Song - Play next song in current directory
⏮	Previous Song – Play previous song in directory
▶/II	Play/Pause – Start playing current song, pause current song, and enter play list
■ MODE	Stop/Mode – Stop playing current song, change between format mode & play mode
+	Increase Control – Increase Volume, Bass or Treble
−	Decrease Control – Decrease Volume, Bass or Treble
↺	Repeat All – Repeat playing from the beginning after last song. Also exits current playlist
⇄	Repeat Song – Repeats current song

⏬	Erase – Hold for more than 3 seconds in stop state will delete current song from storage device
🔑	Keypad Lock Control – Lock or unlock the keypad

2.1.2 USB/PC

The USB interface allows the user to load MP3 or WMA files onto any of the three storage devices using a PC with the Windows operating system. The MP3 Player is setup as a plug and play device that is recognized as a removable storage device by the OS. To load a song, follow the steps below:

1. First verify the MP3 player is turned on, in the stop state and in the correct memory source mode.(MMC, Onboard or IDE) Also make sure the Computer is turned on and running Windows 98/NT or XP.
2. Next connect the USB cable from the PC to the MP3 player. Windows should automatically detect the MP3 player and show whichever storage device you've selected as a removable disk.
3. To verify that the storage device has been recognized, open My Computer and look under the Hard Disk Drives section for a newly added removable disk. Once the disk appears you can add songs to it by opening the disk and then dragging or dropping songs onto it or copying and pasting them.
4. To create a play list, simply create a folder on the removable drive and then add the songs you want on the play list to that folder.
5. Once you've added all the songs you desire to the selected storage device. You can then stop the removable device by clicking the green arrow on the bottom right of the task bar and then disconnect the USB cable. This should return the MP3 player to play mode. Once in play mode the songs should show up on the selected device for playing.

2.2 Outputs

2.2.1 Speaker

The audio output jack is located on the front left hand side of the development board. You can connect headphones or any speaker arrangement you desire. The MP3 player does have built in volume control for song amplification, but if you desire more output you can use an external amp as well.

2.2.2 LCD

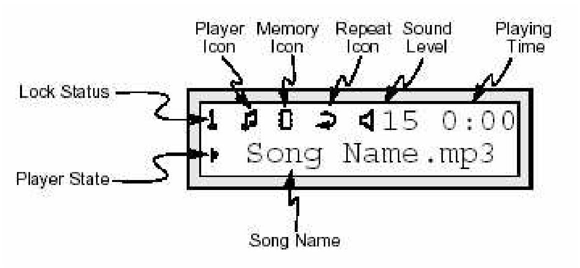


Figure 2.1 – Display in Song Playing Mode

The LCD display shown in Figure 2.2 above provides all song information while in play mode, as well as different icons for other modes and settings available. The mapping of the icons and LCD symbols are labeled above. Table 2.2 below shows the meanings for the different Memory & Player State icons.

Table 2.2 – Selected Memory Status and Song Player States

▯	On-Board Memory
🗋	MMC Flash Memory
🗎	IDE Hard Drive Memory
♪	Song – stop state, song selected
🗀	Playlist/Directory - stop state, play list or directory selected
▶	Play - selected song playing
⏸	Pause - current song paused
✖	Error – Selected memory is not formatted

2.3 Storage

2.3.1 Hard Drive

Selecting a Hard Drive

The MP3 player supports IDE hard drives of any size. However, we recommend that the hard drive run at a minimum of 5400 rpm for smooth song playing. We also ran into a buffer under run issue when using an older hard drive with a very small disk cache, so we'd recommend a new drive with a large enough cache to avoid the 'chirping effect'.

Connecting the Hard Drive

The development board didn't originally support IDE hard drives, so we had to create an IDE adaptor for connecting the hard drive to the 50 pin J6 extension on the development board. With the drive connected using the adaptor, you can then power the hard drive with a PC power supply or any 5 & 12 V sources and a female power adaptor. The jumper settings on the drive are usually left with the jumper not connected.

2.3.2 MMC Flash Card

You can use any size MMC card. The card is simply plug_and_play, to use simply select MMC mode using the keypad indicated in section 2.1.

2.4 Playlist

2.4.1 Setting It Up

See section 2.1.2 step 4.

2.4.2 Accessing and Playing

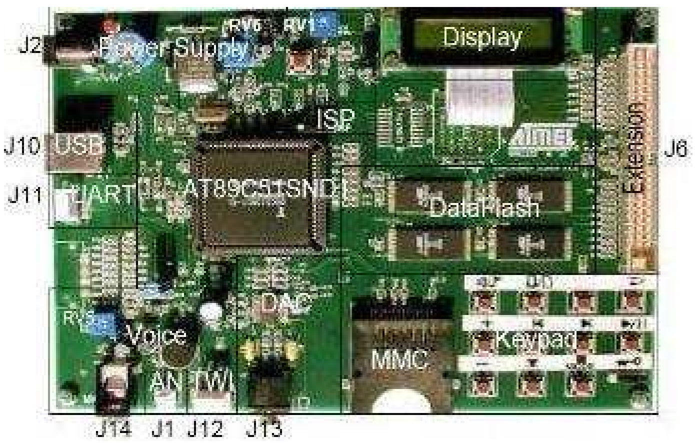
Once the play list has been loaded onto the storage device and the player is back to play mode, you'll notice it starts in the root directory, use the previous song and next song buttons to scroll through the different play lists(directories) and songs located in the root directory. You can distinguish between the two by the song and play list icons shown on the display.(see Table 2.2) Once selected use the play button to enter the play list and then to start playing the first song in the play list. You can exit the play list by hitting the stop button and then the repeat all button. (see Table 2.1)

3. Description and Design

3.1 Development Board

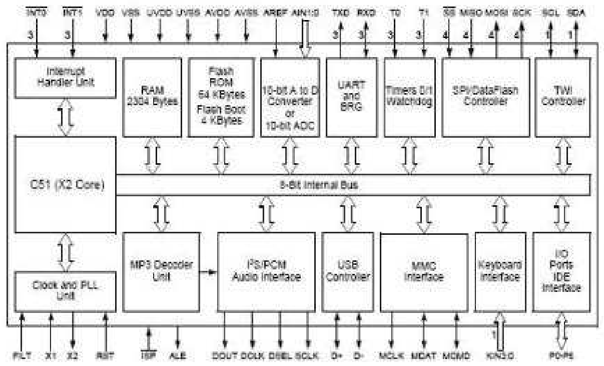
3.1.1 Layout

The board layout is shown below.



3.1.2 Controller

All of the control signals are handled inside the AT89C51SND1 microprocessor. The block diagram below shows the interface between the C51 and the other components.



On top of housing the code, the C51 microprocessor also includes an oscillator, PLL, and timer registers to control the clock. There are also special function registers for use with interrupts, system management, ports, and various memory and I/O devices. The decoding of the MP3 files is handled internally in the C51.

3.1.3 IDE and Resistors

The development board provides a 50 pin IDE adapter interface. In order to use the 50 pin connector, we had to map the control signals from the board to the standard 40 pin IDE ribbon cable which we used to connect the hard drive. Also, various 100 Ohm resistors and 0 Ohm jumpers had to be placed to connect the signal traces from the C51 to the output pins of the IDE interface slot.

3.1.4 MMC Flash Card

The actual design for implementing the flash card was made simple by the fact that the development board came with both the MMC card reader built into it and all of the low level drivers provided. The main design issue was modifying the existing firmware so that you could select the MMC card as a memory source, as well as being able to both have all the playing functionality of the onboard memory as well as the ability to create play lists. The other major part was setting it up for song loading and management from a PC over the USB connection. All this was added to the existing source code for the player. (See Appendix A for MMC Card Firmware Source Code)

3.2 Flip and Keil

3.2.1 Compiling

We wrote code using C. Keil makes a compiler called uvision2. We used uvision2 to compile our C code into the hex form that the chip requires. Uvision2 is made specifically for microcontrollers such as the Atmel 8951 chip that we use. We had to use a trial version that we downloaded from keil.com. This version was not big enough to compile both the HDD and the MMC into one hex file so we could use both functionalities at the same time. We would have had to purchase the real compiler for around $1300. We decided against this because of our lack of funds.

3.2.2 Loading to Board

After getting our hex code from our compiler we used a program called FLIP to download it to our development board. FLIP is made by Atmel and is made specifically for writing to Atmels products. As we were using one of their products FLIP made it easy for us to download to our board. There is a switch called jsp1 on the board that must be switched to the off position in order for FLIP to download to the board. It took us a while to figure this out so at first we were having troubles.

3.3 Hard Drive

3.3.1 IDE Adapter

The adapter consisted of a 50 pin SCSI ribbon cable that plugged into the development board IDE adapter connector. Each individual wire was mapped to the corresponding pin of a 40 pin IDE connector. The hard drive power was generated by two external power supplies since there was no where to access the necessary voltage levels on the development board.

3.3.2 HDD Selection

Our mp3 player only supports 40 pin IDE Hard Drives. The key to quality music is to make sure that the HDD is a newer model. We were originally using an older model and there was chirping in the sound. We think this is because the buffer on the older HDD we were using was too small but have not solidified that assumption. The HDD jumper setting should be set to master or single.

3.4 MMC Card

3.4.1 Implementation

See section 3.1.4

4. Log of Design Decisions

4.1 Spring

This project originally was meant to be a car stereo extension, which allowed the user to add his entire MP3 music library to the built in hard drive over USB. None of us had ever designed an MP3 player before. As a matter of fact looking back on it, we were really naive in the fact that the scope of our original project, even though it sounded cool, was way outside of our means and ability to accomplish in just one semester. At the time though, we thought we could pull it off, so we continued on through the planning stages. We started with a Motorola chip for micro-controller and basically using separate venders for each major component. Luckily for us, another group was originally doing a similar project so they were also researching parts and helped lead us to Atmel by the end of the semester. Once we came across the Atmel C51 chip, which is designed specifically for using in an MP3 player, we decided to use that instead. And that was our status at the end of the Spring semester. Again what now was another lucky break, but at the time seemed unlucky was just after the CS 3992 course ended, one of our group members, Fetah Basic, decided to leave the group and do his own project which he needed to do for his job anyways. This left us in a position, where there was no way the two remaining members, Mike Ballard & Sam Roundy, would be able to finish the project during the Fall Semester. So we now had the Summer to decide if we were still going to do the project and how it would be possible to get done with just the 2 of us.

4.2 Summer

Our log during the summer is where we made a lot of major decisions. In early June after reviewing the Atmel chip we decided that the easiest way to go about implementing any design was to go with the development board that is offered by Atmel. This board supports the Atmel chip we were using and also USB and IDE interfacing which is what we were going for. After the loss of a teammate we thought that two of us could implement our mp3 player easily using the Atmel development board. Deciding to go with this board was a major decision because we were stuck with the LCD that came with it and the user button interface that came with it. This changed the way we could implement LCD output and user button input.

4.3 Fall

Initially, we intended to use a laptop hard drive for its size advantages. Yet, the 50 pin connector would have made it difficult to map to the 44 pin laptop IDE connector. For this reason, we chose to use a normal desktop hard drive. We also ran into trouble with our compiler when we tried to incorporate all the functionality into one hex file. Our compiler allowed us to compile files of a certain size but with the code for the ide and the mmc interface combined, we exceeded this limit. We chose to demo the working hard drive version since we believed it better displayed our design intentions.

5. Purchase vs. Design

5.1 Development Board

The major purchase of our project was the development board. After reviewing several suppliers we decided to purchase the cheapest one we could find. Digikey offered the board that we selected at the cheapest price on the market for $700. We decided that purchasing a development board as opposed to doing our own board design would be easier and we could focus more on having a working mp3 player. Also, another reason we decided to purchase instead of build was that one of our original team members decided to back out and do a different project. Thus we were down to two team members and we were not sure that between the both of us we would be able to finish our project.

5.2 Cable/HDD

We would have preferred to purchase the 50 to 40 pin cable adapter but were unable to find anything.

5.3 Code

Most of our time this past semester wasn't used in actually writing new code, but rather, going through the source code provided as well as all of the other documentation provided with the development board, to see how the whole system was put together, so that we could modify it. We probably learned the most from the amount of time all three of us spent poring over the existing documentation and source code for the MP3 player.

As I said before, we were adding onto existing code which already provided play functionality, but in some ways modifying and changing somebody else's code is just as hard or harder than starting from scratch. Lucky for us, Atmel was good at providing extensive documentation as well as commenting their code each step of the way. They also provided great tech support if we ever got stumped along the way.

Another part of the development kit which made it possible for us to finish the project, was the development tools provided with the board. Atmel included not only Flip, which made it easy to load a hex file onto the C51 controller, but they also gave us a great 8051 C compiler for writing, debugging and compiling the code into a loadable hex file. The other software tool included, which we didn't have a lot of time to use but which is extremely powerful, was the Jungo software tools for creating USB and other component drivers which interface with the Windows OS.

6. Bibliography

Our sources are taken from the following websites:

http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2925

http://www.atmel.com/dyn/products/product_card.asp?part_id=2598

http://www.keil.com/

http://www.digikey.com/

7. Conclusion

7.1 Problems

Overall I would say that we didn't have as many problems as were projected. One major problem we encountered was the first development board. When we were trying to get it to read from the MMC card it was not linking to the PC at all. After exhaustive troubleshooting we shipped it back to Atmel to have them take a look at it. Sure enough, the LCD was drawing too much power. We were worried that it was our fault that the board was bad and that Digikey would not return it and give us new one. But much to our surprise they were willing to do so and we shipped the bad board back to Digikey and received a new one.

Another problem we encountered while designing the mp3 player using the HDD was that our music sounded really chirpy. We thought this was a buffer error in our code but when we changed that it still chirped. Then we switched to a more state-of-the-art HDD and the chirping stopped. So we figure that the older HDD had to slow of a buffer for the amount of reading we were doing. We did try to remedy this problem by using slower bit-rate mp3s and that worked but the quality of sound was horrible.

7.2 Analysis

The development process went fairly smoothly. Where we faced the most problems was in the shortcomings of the development board. With enough time, and money, a better fit could have been built to match our design intentions. For example, a lot of our trouble came with trying to connect the hard drive rather than to set up the code for the hard drive interface. If we had designed the system from the ground up, we would have had a 44 pin connector on board for IDE connections. Also, we spent a lot of time communicating with ATMEL to try and figure out how the human interface with the board worked. Ideally, we would not have had any questions about these basic functionalities had we started from scratch and not with the development board. The rest of the design process went very smoothly as we were able to meet often and regularly to work on and to update each other on our progress.

7.3 Improvements

If we had more time to work on the project, some improvements we would make would be, to take the project from its current status as an elaborate Ipod and adapt it so it would meet our original goal of being a car stereo extension which allows the user to hold his entire music library in his car stereo. To accomplish this, we've discussed adding the ability to transfer songs directly between the hard drive and MMC card for song transfer. We also talked about adapting the Player to read from portable USB data storage devices which could also be used for song transfer between the users PC and the car stereo. The biggest change though would be the packaging of the player itself, so that it would fit into a car stereo and run off of the car's battery. Then we would find funding for our product, and create a company call MSB

8. Acknowledgments

Throughout this past semester, from the time we received the first defective board until now; Atmel has provided great tech support. We'd like to especially acknowledge the help of our Atmel tech support contact named Patrice Graziotin. If it weren't for his great help and patience, I doubt we would have been able to ever complete the project.

We'd also like to acknowledge the tech support at Digi-Key who sold us the development board. They were really good about replacing the defective board at the start of the semester in a timely manner making it possible for us to move on with our project and get it done by the end of the semester.

Last but not least we'd like to thank Al Davis for making a great course and learning experience. We all had a lot of fun and really enjoyed ourselves. Thanks for helping us overcome the obstacles we ran into and for all you taught us.

Appendix A

```
/*****************************************************************************
* NAME:         project.h
*-----------------------------------------------------------------------------

*-----------------------------------------------------------------------------
* PURPOSE:
* This file contains the project configuration definition. This file is
specific
* to your project.
******************************************************************************
/

#ifndef _PROJECT_H_
#define _PROJECT_H_


/*_____ I N C L U D E S
____________________________________________________*/


/*_____ M A C R O S
________________________________________________________*/

/*----- Library Configuration -----*/

#define   IDE_SOFT_RESET  0
#define   IDE_HARD_RESET  1


/* IHM */
#define MEM_SELECT_NEXT       0
#define MEM_NO_CHANGE         1

/* Define here your project. For example, your project can be :
    Nand Flash and Multimedia Card      Use of FAT file system
    HDD drive only                      Use of FAT32 file system
    HDD drive with Multimedia Card      Use of FAT file system
    CDR only                            Use of ISO9660 file system
    HDD drive with CDR                  Use of FAT32 and ISO9660 file system
    Nand Flash with CDR                 Use of FAT32 and ISO9660 file system
*/


/*****************************************************************************
**********/
/*                                   MEDIA CONFIGURATION
*/
/* Define here the media used in your project
*/
/* For IDE devices, note that is a HDD is present, this one a to be selected
as master*/
/* If only one IDE device is defined
*/
```

```
/****************************************************************************
**********/
#ifdef CONF_HDD
/*                           MEDIA DEFINITION
*/
#define MEM_CHIP_TYPE   CHIP_HDD            /* _DF, _NF, _HDD or _NONE
*/
#define MEM_CARD_TYPE   CARD_NONE           /* _MMC, _SD, _SMC, _CF, _CDR or
_NONE */
/*                            FILE SYSTEM
*/
#define MEM_CHIP_FS     FS_FAT_32           /* _FAT_12_16  _FAT_32  _ISO
_NONE     */
#define MEM_CARD_FS     FS_NONE             /* _FAT_12_16  _FAT_32  _ISO
_NONE     */
#define ATA_CONF        HDD                 /* HDD_CDR  or HDD only or CDR
only    */
#endif

#ifdef CONF_CDR
/*                           MEDIA DEFINITION
*/
#define MEM_CHIP_TYPE   CHIP_NONE           /* _DF, _NF, _HDD or _NONE
*/
#define MEM_CARD_TYPE   CARD_CDR            /* _MMC, _SD, _SMC, _CF, _CDR or
_NONE */
/*                            FILE SYSTEM
*/
#define MEM_CHIP_FS     FS_NONE             /* _FAT_12_16  _FAT_32  _ISO
_NONE     */
#define MEM_CARD_FS     FS_ISO              /* _FAT_12_16  _FAT_32  _ISO
_NONE     */
#define ATA_CONF        CDR                 /* HDD_CDR  or HDD only or CDR
only    */
#endif

#ifdef CONF_HDD_CDR
/*                           MEDIA DEFINITION
*/
#define MEM_CHIP_TYPE   CHIP_HDD           /* _DF, _NF, _HDD or _NONE
*/
#define MEM_CARD_TYPE   CARD_CDR           /* _MMC, _SD, _SMC, _CF, _CDR or
_NONE  */
/*                            FILE SYSTEM
*/
#define MEM_CHIP_FS     FS_FAT_32          /* _FAT_12_16  _FAT_32  _ISO
_NONE      */
#define MEM_CARD_FS     FS_ISO             /* _FAT_12_16  _FAT_32  _ISO
_NONE      */
#define ATA_CONF        HDD_CDR            /* HDD_CDR  or HDD only or CDR
only     */
#endif

#ifdef CONF_CF
/*                           MEDIA DEFINITION
*/
```

```
#define MEM_CHIP_TYPE   CHIP_NONE           /* _DF, _NF, _HDD or _NONE
*/
#define MEM_CARD_TYPE   CARD_CF             /* _MMC, _SD, _SMC, _CF, _CDR or
_NONE  */
/*                             FILE SYSTEM
*/
#define MEM_CHIP_FS     FS_NONE             /* _FAT_12_16  _FAT_32  _ISO
_NONE      */
#define MEM_CARD_FS     FS_FAT_12_16        /* _FAT_12_16  _FAT_32  _ISO
_NONE      */
#define ATA_CONF        CF                  /* HDD_CDR  or HDD only or CDR
only     */
#endif


//#define IDE_INIT_TYPE   IDE_HARD_RESET
#define  IDE_INIT_TYPE   IDE_SOFT_RESET

/* Configuration allowed:
    Master      Slave
    HDD         CDR
    CDR         -
    HDD         -
*/

/*****************************************************************************
**********/
/* Define here the file system used in your project
*/
/* Be careful : if MEM_CHIP_TYPE = CHIP_NONE or MEM_CARD_TYPE = CARD_NONE,
*/
/* selected FS must be definedas FS_NONE
*/
/*****************************************************************************
**********/

/* Mass Storage Configuration */
#ifdef CONF_HDD_CDR
  #define MS_MAX_LUN                (1)           /* number of logical unit -
1 */
#else
  #define MS_MAX_LUN                (0)           /* number of logical unit -
1 */
#endif


/*****************************************************************************
**********/
//#define CARD_ERROR_POLICY       MEM_NO_CHANGE       /* _SELECT_NEXT or
_NO_CHANGE */
#define CARD_ERROR_POLICY         MEM_SELECT_NEXT       /* _SELECT_NEXT or
_NO_CHANGE */

/*_____ D E C L A R A T I O N
______________________________________________*/

                                          #endif    /* _PROJECT_H_ */
```

```
/***************************************************************************
* NAME:         config.h
*---------------------------------------------------------------------------

*---------------------------------------------------------------------------
* PURPOSE:
* This file contains the system configuration definition
****************************************************************************
/

#ifndef _CONFIG_H_
#define _CONFIG_H_


/*_____ I N C L U D E S
____________________________________________________*/

#include "lib_mcu\compiler.h"       /* compiler definitions */
#include "lib_mcu\regsnd1.h"        /* component declaration */
#include "lib_mcu\extsnd1.h"        /* component extended declaration */
#include "project.h"                        /* project definitions */

/*_____ M A C R O S
________________________________________________________*/

/*----- Library Configuration -----*/

/* Player Mode */
#define PLAYER_PLAY_MODE      PLAY_DIR      /* _DIR or _DISK */
#define PLAYER_PLAY_START     START_PLAY    /* _PLAY or _STOP */
#define PLAYER_PLAY_LOOP      PLAY_NO_LOOP  /* _LOOP or _NO_LOOP */


/* Timer Driver */
#define T0_X2           TRUE                /* FT0_IN always Fosc/2 */
#define T1_X2           TRUE                /* FT1_IN always Fosc/2 */

/* Keyboard Driver */
#define KBD_EXIT_PD     FALSE               /* Exit power down configuration
*/
#define LOCK_ROW        P1_2                /* ROW of lock key */
#define KEY_LOCK        KEY_2_3             /* Keypad Locked */

/* MP3 Driver */
#define MP3_CRC_SKIP    FALSE               /* CRC error configuration */

/* USB Driver */
#define USB_PU_CTRL     FALSE               /* no pull-up control */
#define VENDOR_ID       0xEB03              /* Atmel vendor ID = 03EBh */
#define PRODUCT_ID      0x0220              /* Atmel Product ID: 2002h */
#define RELEASE_NUMBER  0x0001

/* USB Mass Storage ! unicode char */
#define USB_MANUFACTURER_NAME {'A'<<8,'T'<<8,'M'<<8,'E'<<8,'L'<<8}
#define USB_MN_LENGTH         5
```

```
#define USB_PRODUCT_NAME      {'S'<<8,'N'<<8,'D'<<8,'1'<<8,'
'<<8,'S'<<8,'t'<<8,\
                               'o'<<8,'r'<<8,'a'<<8,'g'<<8,'e'<<8}
#define USB_PN_LENGTH         12

#define USB_SERIAL_NUMBER
{'4'<<8,'7'<<8,'1'<<8,'0'<<8,'7'<<8,'6'<<8,'5'<<8,\
                               '0'<<8,'6'<<8,'6'<<8,'4'<<8,'5'<<8,'1'<<8}
#define USB_SN_LENGTH         13

/* SCSI Mass Storage */
#define SBC_VENDOR_ID         {'G','e','n','e','r','i','c',' '}
#define SBC_PRODUCT_ID        {'S','N','D','1',' ','M','P','3','
','P','l','a','y','e','r',' '}
#define SBC_REVISION_ID       {'0','.','0','0'}

/*----- System Definition -----*/

/* Clock */
#define X2_MODE                            /* X2 mode */
//#undef X2_MODE                             /* no X2 mode */

#define FOSC            16000              /* oscillator frequency (KHz) */

#ifdef X2_MODE
    #define FPER        FOSC
#else
    #define FPER        FOSC/2
#endif

/* Scheduler Definitions */
#define Scheduler_task_1_init   mode_task_init
#define Scheduler_task_2_init   usb_task_init
#define Scheduler_task_3_init   mem_task_init
#define Scheduler_task_4_init   kbd_task_init
#define Scheduler_task_5_init   disp_task_init

#define Scheduler_task_1        mode_task
#define Scheduler_task_2        usb_task
#define Scheduler_task_3        mem_task
#define Scheduler_task_4        kbd_task
#define Scheduler_task_5        disp_task

#define scheduler_btick_0       gl_cpt_tick
#define scheduler_btick_1       gl_kbd_tick
#define scheduler_btick_2       gl_mem_tick

#define SCHEDULER_TYPE        SCHEDULER_FREE  /* _TIMED, _TASK, _FREE */
#define Scheduler_time_init     sch_time_init

#define SCHEDULER_TICK  10                 /* unit is ms */

/* ADC Frequency */
#define SAMPLING_PERIOD 0.125              /* unit is ms */
#define SECT_PER_SECOND (2/SAMPLING_PERIOD) /* 512 bytes sector size */

/* Interrupt Priorities */
```

```
#define EX0_PRIO              ((Byte)0)
#define T0_PRIO               ((Byte)3)
#define EX1_PRIO              ((Byte)0)
#define T1_PRIO               ((Byte)2)
#define MP3_PRIO              ((Byte)2)
#define AUD_PRIO              ((Byte)2)
#define MMC_PRIO              ((Byte)0)
#define I2C_PRIO              ((Byte)0)
#define SPI_PRIO              ((Byte)0)
#define ADC_PRIO              ((Byte)0)
#define KBD_PRIO              ((Byte)0)
#define USB_PRIO              ((Byte)1)


/*----- Firmware Definition -----*/
#define GL_BUF_SIZE           (256)



/* Chip Version */
/* AT89C51SND1C = 0x84 */
#define CHIP_VERSION          ((Byte)0x84)

/* Firmware Version */
#define SYS_VERSION           "V2.1.2"



/*_____ D E F I N I T I O N
________________________________________________*/

/* FAT Format Structure */
typedef struct
{
  Uint16 nb_cylinder;
  Byte   nb_head;
  Byte   nb_sector;
  Byte   nb_hidden_sector;
  Byte   nb_sector_per_cluster;
} s_format;


/*_____ D E C L A R A T I O N
______________________________________________*/


#endif    /* _CONFIG_H_ */
```

```
/*****************************************************************************
* NAME:         main.c
*----------------------------------------------------------------------------

*----------------------------------------------------------------------------
* PURPOSE:
* This is the demonstration software for T8xC51SND1
*****************************************************************************
/

/*_____ I N C L U D E S
____________________________________________________*/

#include "config.h"                          /* system definition */
#include "lib_mcu\c51_drv.h"                 /* c51 driver definition */
#include "scheduler.h"                       /* scheduler definition */


/*_____ M A C R O S
________________________________________________________*/


/*_____ D E F I N I T I O N
________________________________________________*/


/*_____ D E C L A R A T I O N
______________________________________________*/

extern  void    in_system_prog (void);

static  void    main (void);


/*F***************************************************************************
* NAME: main
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*----------------------------------------------------------------------------
* PURPOSE:
*   Main user routine
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
void main (void)
{
  in_system_prog();                          /* check ISP execution request */
  Enable_interrupt();                        /* global enable */

  #ifdef X2_MODE
  Set_x2_mode();
  #endif

  Scheduler();                               /* endless scheduler execution */
}


/*****************************************************************************
* NAME:         sbc.h
*----------------------------------------------------------------------------

*----------------------------------------------------------------------------
* PURPOSE:
* This file contains the SCSI command browsers definition
*
* NOTES:
* - sbc_wr_busy & sbc_rd_busy flags are set to TRUE in mass storage libraries
*   when write or read is in progress.
*   These events must be set to FALSE by the display task
*****************************************************************************
/

#ifndef _SBC_H_
#define _SBC_H_

/*_____ I N C L U D E S
____________________________________________________*/


/*_____ M A C R O S
________________________________________________________*/

#if MEM_CHIP_TYPE == CHIP_DF
  #define Sbc_chip_init     sbc_df_init      /* dataflash memory chip */
  #define Sbc_command_chip sbc_command_df
#elif MEM_CHIP_TYPE == CHIP_NF
  #define Sbc_chip_init     sbc_nf_init      /* nand flash memory chip */
  #define Sbc_command_chip  sbc_command_nf
#elif MEM_CHIP_TYPE == CHIP_HDD
  #define Sbc_chip_init     sbc_hdd_init     /* hard disk memory chip */
  #define Sbc_command_chip  sbc_command_hdd
#elif MEM_CHIP_TYPE == CHIP_NONE
  #define Sbc_chip_init()                    /* no memory chip */
  #define Sbc_command_chip()
#else
  #error unsupported MEM_CHIP_TYPE defined in config.h
#endif

#if MEM_CARD_TYPE == CARD_MMC
  #define Sbc_card_init     sbc_mmc_init     /* multimediacard card */
  #define Sbc_command_card  sbc_command_mmc
#elif MEM_CARD_TYPE == CARD_SD
  #define Sbc_card_init     sbc_sd_init      /* secure digital card */
  #define Sbc_command_card  sbc_command_sd
```

```
#elif MEM_CARD_TYPE == CARD_CF
  #define Sbc_card_init     sbc_cf_init     /* compactflash card */
  #define Sbc_command_card  sbc_command_cf
#elif MEM_CARD_TYPE == CARD_SMC
  #define Sbc_card_init    sbc_smc_init    /* smarmedia card */
  #define Sbc_command_card  sbc_command_smc
#elif MEM_CARD_TYPE == CARD_CDR
  #define Sbc_card_init     sbc_atapi_init    /* compactflash card */
  #define Sbc_command_card  sbc_command_atapi
#elif MEM_CARD_TYPE == CARD_DF
  #define Sbc_card_init     sbc_df_init    /* dataflash card */
  #define Sbc_command_card  sbc_command_df
#elif MEM_CARD_TYPE == CARD_SD_DFC
  extern void   sbc_command_sd();
  extern bit    sbc_sd_init();
  extern void   sbc_command_df();
  extern bit    sbc_df_init();
  #define Sbc_card_init()     ( (gl_sd_dfc == MEM_DFC) ? sbc_df_init()    :
sbc_sd_init())
  #define Sbc_command_card()  ( (gl_sd_dfc == MEM_DFC) ? sbc_command_df() :
sbc_command_sd())
#elif MEM_CARD_TYPE == CARD_NONE
  #define Sbc_card_init()                   /* no card */
  #define Sbc_command_card()
#else
  #error unsupported MEM_CARD_TYPE defined in config.h
#endif


/*_____ D E F I N I T I O N
________________________________________________*/


/*_____ D E C L A R A T I O N
______________________________________________*/


#endif  /* _SBC_H_ */


/*****************************************************************************
* NAME:         hdd.c
*----------------------------------------------------------------------------

*----------------------------------------------------------------------------
* PURPOSE:
* This file contains the high level ATA routines
*****************************************************************************
/

/*_____ I N C L U D E S
____________________________________________________*/

#include "config.h"                           /* system configuration */
#include "board.h"                            /* board definition  */
#include "lib_mcu\usb\usb_drv.h"              /* usb driver definition */
#include "ide_drv.h"                          /* ide definition */
```

```
#include "hdd.h"


/*_____ M A C R O S
________________________________________________________*/


/*_____ D E F I N I T I O N
________________________________________________*/

extern  data    Uint32  gl_ptr_mem;               /* memory data pointer
*/

extern  data    Uint16 gl_ide_cpt_page;

extern  bdata   bit ide_parity;

/* IDE registers */
/*                            name                    |  RD# = 0       |   WR#
= 0   | */
extern  xdata Byte volatile ide_data;           /*  | RD data        | WR
data     | */
extern  xdata Byte volatile ide_error;          /*  | Error          |
Features    | */
extern  xdata Byte volatile ide_sector_count;   /*  | Sect. count    | Sect.
count | */
extern  xdata Byte volatile ide_sector_number;  /*  | Sect. No       | Sect.
No    | */
extern  xdata Byte volatile ide_cylinder_low;   /*  | Cyl. Low       | Cyl.
Low    | */
extern  xdata Byte volatile ide_cylinder_high;  /*  | Cyl. High      | Cyl.
High   | */
extern  xdata Byte volatile ide_drive_head;     /*  | Drive/Head     |
Drive/Head  | */
extern  xdata Byte volatile ide_status;         /*  | Status         |
Command     | */
extern  xdata Byte volatile ide_alt_status;     /*  | Alt status     | Device
Ctrl | */

code  Uint32 hdd_block_size = Hdd_block_size();

xdata Byte previous_op;
xdata Byte error;
xdata Uint32  hdd_mem_size;             /* HDD size in sector     */




/*_____ D E C L A R A T I O N
______________________________________________*/

/*F**************************************************************************
* NAME: hdd_read_open
*----------------------------------------------------------------------------
* PARAMS:
*   sect: address of the logic sector to read (size 512 bytes)
*   size : number of sector
*   global: gl_ptr_mem
*
* return:
*   open status:  OK: open done
*                 KO: open not done
*----------------------------------------------------------------------------
* PURPOSE:
*   Open memory card in read mode (read block)
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*
*   drive/head register     -> LBA 27-24
*   cylinder high register  -> LBA 23-16
*   cylinder low register   -> LBA 15-8
*   sector number register  -> LBA 7-0
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
bit hdd_read_open (Uint32 sect, Byte nb_sector)
{
  if (Ide_notbsy_drq() == OK)
  {
    if (previous_op == 0)
    {
      while (Ide_notbsy_drq() == OK)
        ide_data;
    }
    else
    {
      if (previous_op == 1)
      {
        while (Ide_notbsy_drq() == OK)
        {
          DAT16H = 0xFF;
          ide_data = 0xFF;
        }
      }
    }
  }

  while (Ide_notbsy_notdrq() == KO);                                /* Wait
for BSY = 0 */
  Ide_set_drive_head( ATA_LBA_MODE + (((Byte*)&sect)[0] & 0x0F));   /* LBA
27-24 */

  gl_ide_cpt_page = 0;

  ide_parity = 0;
  previous_op = 0;

  while (Ide_notbsy_notdrq() == KO);                  /* Wait for BSY = 0
and DRQ = 0 */
  Ide_set_sector_count ( nb_sector );                 /* sending parameters
*/
```

```
  Ide_set_sector_number(((Byte*)&sect)[3]);           /* LBA 7-0    */
  Ide_set_cylinder_low (((Byte*)&sect)[2]);           /* LBA 15-8   */
  Ide_set_cylinder_high(((Byte*)&sect)[1]);           /* LBA 23-16  */
  Ide_send_command(ATA_CMD_READ_SECTOR);              /* send command */
  gl_ptr_mem = sect;                                  /* Initialize the
global byte counter */
  while (ide_alt_status & IDE_BSY);
  if (ide_status & IDE_ERR)
  {
    return KO;
  }
  if (!(ide_status & IDE_DRQ))
    return KO;
  return OK;
}


/*F**************************************************************************
* NAME: hdd_read_close
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*----------------------------------------------------------------------------
* PURPOSE:
*   read close
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*
*----------------------------------------------------------------------------
* REQUIREMENTS:
****************************************************************************
/
void hdd_read_close(void)
{
  while (ide_alt_status & IDE_BSY);
  while (ide_alt_status & IDE_DRQ)
    ide_data;
  ide_status; /* clear pending interrupt */
}

/*F*************************************************************************
* NAME: hdd_read_byte
*---------------------------------------------------------------------------
* PARAMS:
*   global: gl_ptr_mem
*
* return:
*   Data read from media
*----------------------------------------------------------------------------
* PURPOSE:
*   Byte read function
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
```

```
* NOTE:
*
*----------------------------------------------------------------------------
* REQUIREMENTS:
****************************************************************************/
Byte hdd_read_byte(void)
{
  if (((Byte*)&gl_ide_cpt_page)[0] == 0x02)
  {
    gl_ptr_mem++;
    while (ide_alt_status & IDE_BSY);
    ide_status;                     /* clear pending interrupt */
    hdd_read_open(gl_ptr_mem, 1);   /* open the next sector */
  }
  gl_ide_cpt_page++;
  return ide_read_databyte();
}


/*F*************************************************************************
* NAME: hdd_read_long_big_endian
*---------------------------------------------------------------------------
* PARAMS:
*   global: gl_ptr_mem
*
* return:
*   Data read from media
*----------------------------------------------------------------------------
* PURPOSE:
*   Long word big endian coded read function
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*
*----------------------------------------------------------------------------
* REQUIREMENTS:
****************************************************************************/
Uint32 hdd_read_long_big_endian(void)
{
Uint32 temp;
  if (((Byte*)&gl_ide_cpt_page)[0] == 0x02)
  {
    gl_ptr_mem++;
    while (ide_alt_status & IDE_BSY);
    ide_status;     /* clear pending inteerupt */
    hdd_read_open(gl_ptr_mem, 1); /* open the next sector */
  }
  ((Byte*)&temp)[3] = ide_data;
  ((Byte*)&temp)[2] = DAT16H;
  ((Byte*)&temp)[1] = ide_data;
  ((Byte*)&temp)[0] = DAT16H;
  gl_ide_cpt_page += 4;
  return temp;
}

/*F*************************************************************************
```

```
* NAME: hdd_read_one_sector
*----------------------------------------------------------------------------
* PARAMS:
*   global: gl_ptr_mem
*
* return:
*
*-----------------------------------------------------------------------------
* PURPOSE:
*   Read one sector from media to buffer
*-----------------------------------------------------------------------------
* EXAMPLE:
*-----------------------------------------------------------------------------
* NOTE:
*
*-----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************/
void hdd_read_one_sector(void)
{
  if (((Byte*)&gl_ide_cpt_page)[0] == 0x02)
  {
    gl_ptr_mem++;
    while (ide_alt_status & IDE_BSY);
    ide_status;     /* clear pending inteerupt */
    hdd_read_open(gl_ptr_mem, 1); /* open the next sector */
  }
  ata_load_sector();
  ((Byte*)&gl_ide_cpt_page)[0] = 0x02;
}


/*F***************************************************************************
* NAME: hdd_read_sector
*-----------------------------------------------------------------------------
* PARAMS:
*   global: gl_ptr_mem
*
* return: OK read done
*         KO read failure
*-----------------------------------------------------------------------------
* PURPOSE:
*   This function is an optimized function that writes 512 bytes from HDD
*   to USB controller
*-----------------------------------------------------------------------------
* EXAMPLE:
*-----------------------------------------------------------------------------
* NOTE:
*-----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
bit hdd_read_sector (Uint16 nb_sector)
{
Byte i;

#ifndef NO_SUPPORT_USB_PING_PONG
```

```
  bit   begin_ping_pong;
    begin_ping_pong = TRUE;
#endif

  do
  {
    for (i = 8; i != 0; i--)
    {
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);
      Usb_write_byte(ide_data); Usb_write_byte(DAT16H);

  #ifdef NO_SUPPORT_USB_PING_PONG
      Usb_set_TXRDY();                      /* start usb transfer */
      while (!Usb_tx_complete());           /* wait end of transfer */
      Usb_clear_TXCMPL();                   /* ack transfer */
  #else
      if (begin_ping_pong)
      {
        begin_ping_pong = FALSE;
      }
      else
      {
        while (!Usb_tx_complete());         /* wait end of transfer */
        Usb_clear_TXCMPL();                 /* ack transfer */
      }
      Usb_set_TXRDY();                      /* start usb transfer */
  #endif
```

```
    }

    gl_ptr_mem++;
    nb_sector--;
    while (ide_alt_status & IDE_BSY);
    ide_status; /* cleat pending interrupt */
    if ( ((Byte*)&nb_sector)[1] != 0)
    {
      while (Ide_notbsy_drq() == KO);
    }
    if ( (((Byte*)&nb_sector)[0] !=0) && ((Byte*)&nb_sector)[1]==0)
    {
      hdd_read_open (gl_ptr_mem, (Byte) nb_sector);   /* open the next sector
*/
    }
  }
  while (nb_sector != 0);

  #ifndef NO_SUPPORT_USB_PING_PONG
    while (!Usb_tx_complete());             /* wait end of last transfer */
    Usb_clear_TXCMPL();                     /* ack transfer */
  #endif

  return OK;
}


/*F**************************************************************************
* NAME: hdd_write_open
*----------------------------------------------------------------------------
* PARAMS:
*   sect: address of the the next write data
*   size: number of sector
*   global: gl_ptr_mem
*
* return:
*   status: OK: open done
*           KO: open not done
*----------------------------------------------------------------------------
* PURPOSE:
*   Open media in write mode (write block)
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*   drive/head register     -> LBA 27-24
*   cylinder high register  -> LBA 23-16
*   cylinder low register   -> LBA 15-8
*   sector number register  -> LBA 7-0
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
bit hdd_write_open (Uint32 sect, Byte nb_sector)
{
Byte dummy;
```

```
  if (Ide_notbsy_drq() == OK)
  {
    if (previous_op == 0)
    {
      while (Ide_notbsy_drq() == OK)
        dummy = ide_data;
    }
    else
    {
      if (previous_op == 1)
      {
        while (Ide_notbsy_drq() == OK)
        {
          DAT16H = 0xFF;
          ide_data = 0xFF;
        }
      }
    }
  }

  while (Ide_notbsy_notdrq() == KO);                            /* Wait for
BSY = 0 */
  Ide_set_drive_head(ATA_LBA_MODE + (((Byte*)&sect)[0] & 0x0F));       /* LBA
27-24 */
  previous_op = 1;

  ide_parity = 0;
  gl_ide_cpt_page = 0;


  while (Ide_notbsy_notdrq() == KO);                      /* Wait for BSY = 0
and DRQ = 0 */
  Ide_set_sector_count ( nb_sector );                             /* sending
parameters  */
  Ide_set_sector_number(((Byte*)&sect)[3]);               /* LBA 7-0    */
  Ide_set_cylinder_low (((Byte*)&sect)[2]);               /* LBA 15-8   */
  Ide_set_cylinder_high(((Byte*)&sect)[1]);               /* LBA 23-16  */
  Ide_send_command(ATA_CMD_WRITE_SECTOR);                 /* send command*/
  gl_ptr_mem = sect;                                      /* Update global
memory pointer */

  if (ide_status & IDE_ERR)
  {
    error |= 0x02;
  }

  while (Ide_notbsy_drq() == KO);                        /* Wait for BSY = 0
and DRQ = 1 */
  return OK;
}


/*F**************************************************************************
* NAME: hdd_write_close
*----------------------------------------------------------------------------
* PARAMS:
```

```
*
*
* return:
*----------------------------------------------------------------------------
* PURPOSE:
*   Media write close
*   finish programming end of block
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
void hdd_write_close (void)
{
  while (((Byte*)&gl_ide_cpt_page)[0] != 0x02)
  {
    ide_write_databyte(0xFF);
    gl_ide_cpt_page++;
  }
  while (ide_alt_status & IDE_BSY);
  ide_status; /* clear pending interrupt */
}


/*F**************************************************************************
* NAME: hdd_write_byte
*----------------------------------------------------------------------------
* PARAMS:
*   b: data to write
*   global: gl_ptr_mem
*
* return:
*   write status: OK: write done
*                 KO: write not done
*----------------------------------------------------------------------------
* PURPOSE:
*   byte write function
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
bit hdd_write_byte(Byte b)
{
  if (((Byte*)&gl_ide_cpt_page)[0] == 0x02)
  {
    gl_ptr_mem++;
    while (ide_alt_status & IDE_BSY);
    ide_status;  /* clear pending interrupt */
    hdd_write_open(gl_ptr_mem, 1); /*  open at next sector */
  }
  gl_ide_cpt_page++;
  ide_write_databyte(b);
  return OK;
}


/*F**************************************************************************
* NAME: hdd_write_one_sector
*----------------------------------------------------------------------------
* PARAMS:
*   global: gl_ptr_mem
*
* return:
*
*----------------------------------------------------------------------------
* PURPOSE:
*   write one sector from buffer
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************/
void hdd_write_one_sector(void)
{
  if (((Byte*)&gl_ide_cpt_page)[0] == 0x02)
  {
    gl_ptr_mem++;
    while (ide_alt_status & IDE_BSY);
    ide_status;  /* clear pending interrupt */
    hdd_write_open(gl_ptr_mem, 1); /*  open at next sector */
  }
  ata_download_sector();
  ((Byte*)&gl_ide_cpt_page)[0] = 0x02;
}


/*F***************************************************************************
* NAME: hdd_write_sector
*----------------------------------------------------------------------------
* PARAMS:
*   global: gl_ptr_mem
*
* return:
*   write status: OK: write done
*                 KO: write not done
*----------------------------------------------------------------------------
* PURPOSE:
*   This function is an optimized function that writes 512 bytes from USB
*   controller to HDD card
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
```

```
* REQUIREMENTS:
*****************************************************************************
/
bit hdd_write_sector (Uint16 nb_sector)
{
  do
  {
    while (!Usb_rx_complete());    /* wait end of reception */
    ata_download_data_usb();       /* 1 */
    Usb_clear_RXOUT_PP();          /* usb read acknowledgement */
    while (!Usb_rx_complete());    /* wait end of reception */
    ata_download_data_usb();       /* 2 */
    Usb_clear_RXOUT_PP();          /* usb read acknowledgement */
    while (!Usb_rx_complete());    /* wait end of reception */
    ata_download_data_usb();       /* 3 */
    Usb_clear_RXOUT_PP();          /* usb read acknowledgement */
    while (!Usb_rx_complete());    /* wait end of reception */
    ata_download_data_usb();       /* 4 */
    Usb_clear_RXOUT_PP();          /* usb read acknowledgement */
    while (!Usb_rx_complete());    /* wait end of reception */
    ata_download_data_usb();       /* 5 */
    Usb_clear_RXOUT_PP();          /* usb read acknowledgement */
    while (!Usb_rx_complete());    /* wait end of reception */
    ata_download_data_usb();       /* 6 */
    Usb_clear_RXOUT_PP();          /* usb read acknowledgement */
    while (!Usb_rx_complete());    /* wait end of reception */
    ata_download_data_usb();       /* 7 */
    Usb_clear_RXOUT_PP();          /* usb read acknowledgement */
    while (!Usb_rx_complete());    /* wait end of reception */
    ata_download_data_usb();       /* 8 */
    Usb_clear_RXOUT_PP();          /* usb read acknowledgement */

    nb_sector--;
    while (ide_alt_status & IDE_BSY);
    ide_status; /* clear pending interrupt */
    if (((Byte*)&nb_sector)[1] != 0)
    {
      while (Ide_notbsy_drq() == KO);
    }
    gl_ptr_mem++;
    if ( (((Byte*)&nb_sector)[0]!=0) && ((Byte*)&nb_sector)[1]==0)
    {
      hdd_write_open (gl_ptr_mem, (Byte) nb_sector);   /* open the next
sector */
    }
  }
  while (nb_sector != 0);
  return OK;
}


/*F**************************************************************************
* NAME: hdd_check_presence
*----------------------------------------------------------------------------
* PARAMS:
*
*
```

```
* return:
*
*----------------------------------------------------------------------------
* PURPOSE:
*
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
bit hdd_check_presence(void)
{
  return OK;
}


/*F**************************************************************************
* NAME: hdd_get_capacity
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*
*----------------------------------------------------------------------------
* PURPOSE:
*   This function return the number of sector in LBA mode.
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
Uint32 hdd_get_capacity(void)
{
  return hdd_mem_size;
}

/*F**************************************************************************
* NAME: hdd_format
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*
*----------------------------------------------------------------------------
* PURPOSE:
*   This function is called by the fat_format function and returns a pointer
*   to a table containing the format parameters.
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
```

```
* NOTE:
*
*-----------------------------------------------------------------------------
* REQUIREMENTS:
******************************************************************************
/
s_format  xdata *hdd_format(void)
{
Byte i, temp;
Uint16 dummy;
xdata s_format  format;

  format.nb_cylinder =  0x00;          /* Number of cylinders */
  format.nb_head =      0x00;          /* Number of heads */
  format.nb_sector =    0x00;          /* Number of sector/track */
  format.nb_hidden_sector = 0x00;

  while (Ide_notbsy() == KO);      /* Wait for  BSY = 0 */

  Ide_set_drive_head (ATA_LBA_MODE);    /* Select Drive / Head */

  while (Ide_notbsy_drdy() == KO);

  Ide_send_command(ATA_CMD_IDENTIFY_DRIVE);/* send command */

  while ( Ide_notbsy_drq() == KO );  /* Check if data request bit set */

  for (i = 54; i != 0; i--)
  {
    dummy = ide_read_dataword();       /* dummy reads */
  }

  format.nb_cylinder =  ide_read_dataword();          /* Number of cylinders
*/
  format.nb_head =      ide_read_dataword();          /* Number of heads */
  format.nb_sector =    ide_read_dataword();          /* Number of
sector/track */
  format.nb_hidden_sector = 32;

  /* cluster size determination */
  if (hdd_mem_size <= 532480)         /* up to 260Mb, 0.5K cluster FAT32 */
    temp = 1;
  else
    if (hdd_mem_size <= 16777216)     /* up to 8Gb, 4K cluster FAT32 */
      temp = 8;
    else
      if (hdd_mem_size <= 33554432)   /* up to 16Gb, 8K cluster FAT32 */
        temp = 16;
      else
        if (hdd_mem_size <= 67108864) /* up to 32Gb, 16K cluster FAT32 */
          temp = 32;
        else
          temp = 64;                  /* disks greater than 32Gb, 32K cluster
size */

  format.nb_sector_per_cluster = temp;
  i = 54;
  while (Ide_notbsy_drq() == OK)
  {
    dummy = ide_read_dataword();       /* dummy reads */
    i++;
  }
  return &format;
}

/*****************************************************************************
* NAME:         ide_drv.c
*----------------------------------------------------------------------------

*----------------------------------------------------------------------------
* PURPOSE:
* This file contains the low level HDD/CDR routines
*****************************************************************************
/

/*_____ I N C L U D E S
____________________________________________________*/

#include "config.h"                        /* system configuration */
#include "board.h"                         /* board definition */
#include "ide_drv.h"                       /* IDE definition */



/*_____ M A C R O S
________________________________________________________*/


/*_____ D E F I N I T I O N
________________________________________________*/

/*                           IDE registers                          */
/*                  name                                            |  RD# =
0    |   WR# = 0   | */
xdata Byte volatile ide_data       At(IDE_DATA_ADDRESS);          /*| RD data
| WR data     | */
xdata Byte volatile ide_error      At(IDE_ERROR_ADDRESS);         /*| Error
| Features    | */
xdata Byte volatile ide_sector_count        At(IDE_SECTOR_COUNT_ADDRESS);
/*| Sect. count | Sect. count | */
xdata Byte volatile ide_sector_number       At(IDE_SECTOR_NUMBER_ADDRESS);
/*| Sect. No    | Sect. No    | */
xdata Byte volatile ide_cylinder_low        At(IDE_CYLINDER_LOW_ADDRESS);
/*| Cyl. Low    | Cyl. Low    | */
xdata Byte volatile ide_cylinder_high       At(IDE_CYLINDER_HIGH_ADDRESS);
/*| Cyl. High   | Cyl. High   | */
xdata Byte volatile ide_drive_head          At(IDE_DRIVE_HEAD_ADDRESS);
/*| Drive/Head  | Drive/Head  | */
xdata Byte volatile ide_status     At(IDE_STATUS_ADDRESS);        /*| Status
| Command     | */
xdata Byte volatile ide_alt_status At(IDE_ALT_STATUS_ADDRESS);    /*| Alt
status  | Device Ctrl | */

bdata bit ide_parity;
```

```
data Uint16 gl_ide_cpt_page;


/*_____ D E C L A R A T I O N
______________________________________________*/

/*F**************************************************************************
* NAME: ide_read_dataword
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*   Word Data read from media
*----------------------------------------------------------------------------
* PURPOSE:
*   Word read function
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
****************************************************************************/
Uint16 ide_read_dataword (void)
{
Uint16 tmp;

  ((Byte*)&tmp)[1] = ide_data;
  ((Byte*)&tmp)[0] = DAT16H;
  return tmp;
}

/*F**************************************************************************
* NAME: ide_read_8_dataword
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*   Word Data read from media
*----------------------------------------------------------------------------
* PURPOSE:
*   Word read function
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
****************************************************************************/
Uint16 ide_read_8_dataword (void)
{
Uint16 tmp;

  ((Byte*)&tmp)[1] = ide_data;
  ((Byte*)&tmp)[0] = ide_data;
  return tmp;
```

```
}

/*F*************************************************************************
* NAME: ide_read_databyte
*--------------------------------------------------------------------------
* PARAMS:
*
* return:
*   Byte Data read from media
*---------------------------------------------------------------------------
* PURPOSE:
*   Byte read function
*---------------------------------------------------------------------------
* EXAMPLE:
*---------------------------------------------------------------------------
* NOTE:
*---------------------------------------------------------------------------
* REQUIREMENTS:
**************************************************************************/
Byte ide_read_databyte (void)
{
static Byte high;
Byte low;
  if (ide_parity == 0)
  {
    ide_parity = 1;
    low = ide_data;
    high = DAT16H;
    return low;
  }
  else
  {
    ide_parity = 0;
    return high;
  }
}


/*F*************************************************************************
* NAME: ide_write_databyte
*--------------------------------------------------------------------------
* PARAMS:
*
* return:
*   Byte Data write to media
*---------------------------------------------------------------------------
* PURPOSE:
*   Byte write function
*---------------------------------------------------------------------------
* EXAMPLE:
*---------------------------------------------------------------------------
* NOTE:
*---------------------------------------------------------------------------
* REQUIREMENTS:
**************************************************************************/
void ide_write_databyte (Byte d)
{
```

```
static Byte temp;

  if (ide_parity == 0)
  {
    ide_parity = 1;
    temp = d; /* High order byte -> next time, will be stored in ACC */
  }
  else
  {
    ide_parity = 0;
    DAT16H = d;
    ide_data = temp;      /* Low order byte */
  }
}


/*****************************************************************************
* NAME:         mmc_drv.c
*----------------------------------------------------------------------------

*----------------------------------------------------------------------------
* PURPOSE:
* This file contains the MMC driver routines
*
* NOTES:
* Driver Configuration:
*   - None
* Global Variables:
*   - None
*****************************************************************************
/

/*_____ I N C L U D E S
____________________________________________________*/

#include "config.h"                           /* system configuration */
#include "mmc_drv.h"                          /* mmc driver definition */


/*_____ M A C R O S
________________________________________________________*/


/*_____ D E F I N I T I O N
________________________________________________*/

static  Byte    mmc_state;
static  bit     mmc_ready;                    /* MMC in prog state */


/*_____ D E C L A R A T I O N
______________________________________________*/

void    mmc_set_prio (Byte);
void    mmc_send_cmd (Byte, Uint32, Byte);
bit     mmc_check_response (void);
```

```
/*F**************************************************************************
* NAME: mmc_set_prio
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*----------------------------------------------------------------------------
* PURPOSE:
*   Set the MMC controller priority interrupt
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
* ram/xram:
* cycle:
* stack:
* code:
*****************************************************************************
/
void mmc_set_prio (Byte priority)
{
  if ((priority == 1) || (priority == 3))     /* set LSB priority bit */
  {
    IPL1 |=  MSK_ESPI;
  }
  if ((priority == 2) || (priority == 3))     /* set MSB priority bit */
  {
    IPH1 |= MSK_ESPI;
  }
}


/*F**************************************************************************
* NAME: mmc_send_cmd
*----------------------------------------------------------------------------
* PARAMS:
*   index:    command index
*   argument: argument (32 bits) of the command to send
*   response: expected response to the command to send
*
* return:
*----------------------------------------------------------------------------
* PURPOSE:
*   Send a command on the bus
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*   The fifo lock flag is not tested it is under firmware responsability to
*   take care of inter-command delays
*----------------------------------------------------------------------------
* REQUIREMENTS:
* ram/xram:
```

```
* cycle:
* stack:
* code:
****************************************************************************
/
void mmc_send_cmd (Byte index, Uint32 argument, Byte response)
{
  MMCMD = index;
  MMCMD = ((Byte*)&argument)[0];
  MMCMD = ((Byte*)&argument)[1];
  MMCMD = ((Byte*)&argument)[2];
  MMCMD = ((Byte*)&argument)[3];

  switch (response)
  {
    case MMC_RESP_R1:
    case MMC_RESP_R4:
    case MMC_RESP_R5:
    {
      MMCON0 |= MSK_RFMT;                   /* set 48 bits response */
      MMCON0 &= ~MSK_CRCDIS;                /* set response with CRC7 */
      MMCON1 |= (MSK_CMDEN|MSK_RESPEN);
      MMCON1 &= ~(MSK_CMDEN|MSK_RESPEN);    /* send command with response */
      break;
    }
    case MMC_RESP_R2:
    {
      MMCON0 &= ~(MSK_RFMT | MSK_CRCDIS);   /* set 136 bits response with
CRC7 */
      MMCON1 |= (MSK_CMDEN|MSK_RESPEN);
      MMCON1 &= ~(MSK_CMDEN|MSK_RESPEN);    /* send command with response */
      break;
    }
    case MMC_RESP_R3:
    {
      MMCON0 |= (MSK_RFMT | MSK_CRCDIS);    /* set 48 bits response without
CRC7 */
      MMCON1 |= (MSK_CMDEN|MSK_RESPEN);
      MMCON1 &= ~(MSK_CMDEN|MSK_RESPEN);    /* send command with response */
      break;
    }
    case MMC_NO_RESP:
    default:
    {
      MMCON1 |= MSK_CMDEN;
      MMCON1 &= ~MSK_CMDEN;                 /* send command without response
*/
      break;
    }
  }
}


/*F**************************************************************************
* NAME: mmc_send_scmd
*----------------------------------------------------------------------------
* PARAMS:
```

```
*   index:    command index
*   response: expected response to the command to send
*
* return:
*----------------------------------------------------------------------------
* PURPOSE:
*   Send a short command on the bus
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*   The fifo lock flag is not tested it is under firmware responsability to
*   take care of inter-command delays
*----------------------------------------------------------------------------
* REQUIREMENTS:
* ram/xram:
* cycle:
* stack:
* code:
*****************************************************************************
/
void mmc_send_scmd (Byte index, Byte response)
{
  MMCMD = index;
  MMCMD = (Byte)MMC_NO_ARG;
  MMCMD = (Byte)MMC_NO_ARG;
  MMCMD = (Byte)MMC_NO_ARG;
  MMCMD = (Byte)MMC_NO_ARG;

  switch (response)
  {
    case MMC_RESP_R1:
    case MMC_RESP_R4:
    case MMC_RESP_R5:
    case SD_RESP_R6:
    {
      MMCON0 |= MSK_RFMT;                    /* set 48 bits response */
      MMCON0 &= ~MSK_CRCDIS;                 /* set response with CRC7 */
      MMCON1 |= (MSK_CMDEN|MSK_RESPEN);
      MMCON1 &= ~(MSK_CMDEN|MSK_RESPEN);    /* send command with response */
      break;
    }
    case MMC_RESP_R2:
    {
      MMCON0 &= ~(MSK_RFMT | MSK_CRCDIS);   /* set 136 bits response with
CRC7 */
      MMCON1 |= (MSK_CMDEN|MSK_RESPEN);
      MMCON1 &= ~(MSK_CMDEN|MSK_RESPEN);    /* send command with response */
      break;
    }
    case MMC_RESP_R3:
    {
      MMCON0 |= (MSK_RFMT | MSK_CRCDIS);    /* set 48 bits response without
CRC7 */
      MMCON1 |= (MSK_CMDEN|MSK_RESPEN);
      MMCON1 &= ~(MSK_CMDEN|MSK_RESPEN);    /* send command with response */
      break;
    }
    case MMC_NO_RESP:
    default:
    {
      MMCON1 |= MSK_CMDEN;
      MMCON1 &= ~MSK_CMDEN;                 /* send command without response
*/
      break;
    }
  }
}


/*F**************************************************************************
* NAME: mmc_check_response
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*   MMC_ERR_RESP: no response or bad format received
*   MMC_RESP_OK:  response received
*----------------------------------------------------------------------------
* PURPOSE:
*   Check command response
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
* ram/xram:
* cycle:
* stack:
* code:
*****************************************************************************
/
bit mmc_check_response (void)
{
  if (Mmc_response_received())
  { /* response received */
    if ((MMCON0 & MSK_CRCDIS) != 0)
    {                                         /* CRC7 not computed */
      if ((MMSTA & MSK_RESPFS) != 0)
      {
        return (MMC_RESP_OK);
      }
      else
      {
        return (MMC_ERR_RESP);                /* format error */
      }
    }
    else
    {                                         /* CRC7 computed */
      if ((MMSTA & (MSK_RESPFS | MSK_CRC7S)) == (MSK_RESPFS | MSK_CRC7S))
      {
        return (MMC_RESP_OK);
      }
      else
      {
        return (MMC_ERR_RESP);                /* format or CRC7 error */
      }
    }
  }
  else
  { /* no response received */
    return (MMC_ERR_RESP);
  }
}


/*F***************************************************************************
* NAME: mmc_read_response
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*   4-byte argument of the response
*----------------------------------------------------------------------------
* PURPOSE:
*   Read command argument response
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
* ram/xram:
* cycle:
* stack:
* code:
*****************************************************************************
/
Uint32 mmc_read_response (void)
{
Uint32  argument;

  ((Byte*)&argument)[0] = MMCMD;    /* dummy index read */
  ((Byte*)&argument)[0] = MMCMD;
  ((Byte*)&argument)[1] = MMCMD;
  ((Byte*)&argument)[2] = MMCMD;
  ((Byte*)&argument)[3] = MMCMD;

  return argument;
}

/*****************************************************************************
* NAME:         usb_drv.c
*----------------------------------------------------------------------------

*----------------------------------------------------------------------------
* PURPOSE:
* This file contains the USB driver routines
*
* NOTES:
```

```
* Driver Configuration (see config.h):
*   - VENDOR_ID              enum vendor ID delivered by USB organisation
*   - PRODUCT_ID             enum product number
*   - RELEASE_NUMBER         enum release number
*   - USB_MANUFACTURER_NAME  mass storage manufacturer string (unicode)
*   - USB_MN_LENGTH          mass storage manufacturer string length
*   - USB_PRODUCT_NAME       mass storage product name string (unicode)
*   - USB_PN_LENGTH          mass storage product name string length
*   - USB_SERIAL_NUMBER      mass storage product serial nb string (unicode)
*   - USB_SN_LENGTH          mass storage product serial nb string length
*****************************************************************************
/

/*_____ I N C L U D E S
____________________________________________________*/

#include "config.h"                       /* system configuration */
#include "usb_drv.h"                      /* usb driver definition */


/*_____ M A C R O S
________________________________________________________*/


/*_____ D E F I N I T I O N
________________________________________________*/

code struct usb_st_device_descriptor usb_device_descriptor =
  {
    sizeof(usb_device_descriptor), DEVICE, 0x1001, 0, 0, 0,
EP_CONTROL_LENGTH,
    VENDOR_ID, PRODUCT_ID, RELEASE_NUMBER, MAN_INDEX, PROD_INDEX, SN_INDEX, 1
  };

code struct usb_st_manufacturer usb_manufacturer =
  {
    sizeof(usb_manufacturer), STRING,
    USB_MANUFACTURER_NAME
  };

code struct usb_st_product usb_product =
  {
    sizeof(usb_product), STRING,
    USB_PRODUCT_NAME
  };

code struct usb_st_serial_number usb_serial_number =
  {
    sizeof(usb_serial_number), STRING,
    USB_SERIAL_NUMBER
  };

code struct usb_st_language_descriptor usb_language =
  {
    sizeof(usb_language), STRING, 0x0904
  };
```

```
code struct
  {
    struct usb_st_configuration_descriptor  cfg;
    struct usb_st_interface_descriptor      ifc;
    struct usb_st_endpoint_descriptor       ep1;
    struct usb_st_endpoint_descriptor       ep2;
  }
  usb_configuration =
    {
      { 9, CONFIGURATION, sizeof(usb_configuration) << 8, 1, 1, 0,
USB_CONFIG_BUSPOWERED, 0x32},
      { 9, INTERFACE, 0, 0, 2, 0x08, 0x06, 0x50, 0 },
      { 7, ENDPOINT, 0x81, 0x02, EP_IN_LENGTH << 8, 0 },
      { 7, ENDPOINT, 0x02, 0x02, EP_OUT_LENGTH << 8, 0 }
    };

static  bdata bit     zlp;
static  idata Byte    endpoint_status[3];

static  idata Byte    *pbuffer;
static  idata Byte    bmRequestType;

/*_____ D E C L A R A T I O N
______________________________________________*/

extern  void    usb_mass_storage_reset (void);
extern  void    usb_mass_storage_get_lun (void);

static  void    usb_get_descriptor (void);
static  Byte*   send_ep0_packet (Byte *, Byte);
static  void    usb_read_request (void);
static  void    usb_set_address (void);
static  void    usb_set_configuration (void);
static  void    usb_clear_feature (void);
static  void    usb_set_feature (void);
static  void    usb_get_status (void);
static  void    usb_get_configuration (void);


/*F**************************************************************************
* NAME: usb_init
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*----------------------------------------------------------------------------
* PURPOSE:
* This function initializes the USB controller and resets the endpoints
FIFOs.
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
```

```
void usb_init (void)
{
  Usb_enable();                             /* enable USB */
  UEPRST = 0x07;                            /* Reset EP 0, 1 and 2 */
  UEPRST = 0x00;
  endpoint_status[EP_CONTROL] = 0x00;
  endpoint_status[EP_IN] = 0x00;
  endpoint_status[EP_OUT] = 0x00;
  Usb_select_ep(EP_CONTROL);                /* control endpoint config */
  UEPCONX = CONTROL;
}


/*F*****************************************************************************
* NAME: usb_ep_init
*-----------------------------------------------------------------------------
* PARAMS:
*
* return:
*-----------------------------------------------------------------------------
* PURPOSE:
* This function configures the endpoints.
*-----------------------------------------------------------------------------
* EXAMPLE:
*-----------------------------------------------------------------------------
* NOTE:
*-----------------------------------------------------------------------------
* REQUIREMENTS:
*******************************************************************************
/
void usb_ep_init (void)
{
  Usb_select_ep(EP_CONTROL);
  UEPCONX = CONTROL;
  Usb_select_ep(EP_IN);                     /* endpoints configuration */
  UEPCONX = BULK_IN ;
  Usb_select_ep(EP_OUT);
  UEPCONX = BULK_OUT;
  UEPRST = 0x07;
  UEPRST = 0x00;
}


/*F*****************************************************************************
* NAME: usb_send_ep0_packet
*-----------------------------------------------------------------------------
* PARAMS:
* *tbuf:        address of the first data to send
* data_length:  number of bytes to send
*
* return:       address of the next byte to send
*-----------------------------------------------------------------------------
* PURPOSE:
* This function sends the data over the default control endpoint.
*-----------------------------------------------------------------------------
* EXAMPLE:
*-----------------------------------------------------------------------------
```

```
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
Byte* send_ep0_packet (Byte *tbuf, Byte data_length)
{
Byte i;

  Usb_select_ep(EP_CONTROL);
  for (i = data_length; i != 0 ; i--, tbuf++)
  {
    Usb_write_byte(*tbuf);
  }
  Usb_set_TXRDY();                          /* Send packet */
  return tbuf;
}


/*F**************************************************************************
* NAME: usb_enumeration_process
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*----------------------------------------------------------------------------
* PURPOSE:
* This function manages the enumeration process
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
void usb_enumeration_process (void)
{
  Usb_select_ep(EP_CONTROL);
  usb_read_request();
}


/*F**************************************************************************
* NAME: usb_read_request
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*----------------------------------------------------------------------------
* PURPOSE:
* This function reads the SETUP request sent to the default control endpoint
* and  the appropriate function. When exiting of the usb_read_request
* function, the device is ready to manage the next request.
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
```

```
* NOTE: list of supported requests:
*               GET_DESCRIPTOR
*               GET_CONFIGURATION
*               SET_ADDRESS
*               SET_CONFIGURATION
*               CLEAR_FEATURE
*               SET_FEATURE
*               GET_STATUS
*               GET_MAX_LUN
*               MASS_STORAGE_RESET
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
void usb_read_request (void)
{
  bmRequestType = Usb_read_byte();          /* read bmRequestType */

  switch (Usb_read_byte())                  /* test the bRequest value */
  {
    case GET_DESCRIPTOR:
      usb_get_descriptor();
      break;
    case GET_CONFIGURATION:
      usb_get_configuration();
      break;
    case SET_ADDRESS:
      usb_set_address();
      break;
    case SET_CONFIGURATION:
      usb_set_configuration();
      break;
    case CLEAR_FEATURE:
      usb_clear_feature();
      break;
    case SET_FEATURE:
      usb_set_feature();
      break;
    case GET_STATUS:
      usb_get_status();
      break;
    case GET_MAX_LUN:
      usb_mass_storage_get_lun();
      break;
    case MASS_STORAGE_RESET:
      usb_mass_storage_reset();
      break;
    default:
      Usb_clear_RXSETUP();
      Usb_set_STALLRQ();
      while (!Usb_STALL_sent());
      Usb_clear_STALLRQ();
      Usb_clear_STALLED();
      break;
  }
}
```

```
/*F**************************************************************************
* NAME: usb_set_address
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*----------------------------------------------------------------------------
* PURPOSE:
* This function manages the SET_ADDRESS request. The new address is stored
* in the USBADDR register
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
void usb_set_address (void)
{
Byte add;

  Usb_clear_DIR();
  add = Usb_read_byte();                    /* store the LSB of wValue =
address */
  UEPRST = 0x01 ;
  UEPRST = 0x00 ;
  Usb_clear_RXSETUP();
  Usb_set_TXRDY();                          /* send a ZLP for STATUS phase */
  Usb_set_FADDEN();
  while (!(Usb_tx_complete()));
  Usb_clear_TXCMPL();
  Usb_configure_address(add);
}


/*F**************************************************************************
* NAME: usb_set_configuration
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*----------------------------------------------------------------------------
* PURPOSE:
* This function manages the SET_CONFIGURATION request.
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
void usb_set_configuration (void)
{
Uchar configuration_number;
```

```
  configuration_number = Usb_read_byte();
  Usb_clear_DIR();
  Usb_clear_RXSETUP();
  Usb_set_TXRDY();                          /* send a ZLP for STATUS phase */
  if (configuration_number == 0)
  {
    Usb_clear_CONFG();
  }
  else
  {
    Usb_set_CONFG();
  }
  while (!Usb_tx_complete());
  Usb_clear_TXCMPL();
  Usb_select_ep(EP_IN);                     /* endpoints configuration */
  UEPCONX = BULK_IN ;
  Usb_select_ep(EP_OUT);
  UEPCONX = BULK_OUT;
}


/*F**************************************************************************
* NAME: usb_get_descriptor
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*----------------------------------------------------------------------------
* PURPOSE:
* This function manages the GET_DESCRIPTOR request.
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
void usb_get_descriptor (void)
{
Byte    data_to_transfer;
Uint16  wLength;
Byte    descriptor_type;
Byte    string_type;

  Usb_set_DIR();                            /* set out on EP0 */
  zlp = FALSE;                              /* no zero length packet */

  string_type = Usb_read_byte();            /* read LSB of wValue */
  descriptor_type = Usb_read_byte();        /* read MSB of wValue */
  switch (descriptor_type)
  {
    case DEVICE:
    {
      data_to_transfer = sizeof (usb_device_descriptor);
      pbuffer = &(usb_device_descriptor.bLength);
      break;
    }

    case CONFIGURATION:
    {
      data_to_transfer = sizeof (usb_configuration);
      pbuffer = &(usb_configuration.cfg.bLength);
      break;
    }

    case STRING:
    {
      switch (string_type)
      {
        case LANG_ID:
        {
          data_to_transfer = sizeof (usb_language);
          pbuffer = &(usb_language.bLength);
          break;
        }
        case MAN_INDEX:
        {
          data_to_transfer = sizeof (usb_manufacturer);
          pbuffer = &(usb_manufacturer.bLength);
          break;
        }
        case PROD_INDEX:
        {
          data_to_transfer = sizeof (usb_product);
          pbuffer = &(usb_product.bLength);
          break;
        }
        case SN_INDEX:
        {
          data_to_transfer = sizeof (usb_serial_number);
          pbuffer = &(usb_serial_number.bLength);
          break;
        }
        default:
        {
          Usb_clear_RXSETUP();
          while (!Usb_setup_received())
          {
            Usb_set_STALLRQ();              /* send stall */
            while ((!(Usb_STALL_sent())) && (Usb_setup_received()));
            Usb_clear_STALLED();
            Usb_clear_STALLRQ();
          }
          Usb_clear_DIR();                  /* set in on EP0 */
          return;
        }
      }
      break;
    }

    default:
    {
      Usb_clear_RXSETUP();
      while (!Usb_setup_received())
      {
        Usb_set_STALLRQ();                  /* send stall */
        while ((!(Usb_STALL_sent())) && (Usb_setup_received()));
        Usb_clear_STALLED();
        Usb_clear_STALLRQ();
      }
      Usb_clear_DIR();                      /* set in on EP0 */
      return;
    }
  }

  ACC = Usb_read_byte();                    /* don't care of wIndex field */
  ACC = Usb_read_byte();
  ((Byte*)&wLength)[1] = Usb_read_byte();   /* read wLength */
  ((Byte*)&wLength)[0] = Usb_read_byte();
  if (wLength > data_to_transfer)
  {
    if ((data_to_transfer % EP_CONTROL_LENGTH) == 0)
    {
      zlp = TRUE;                           /* send a zero length packet */
    }
    else
    {
      zlp = FALSE;                          /* no need of zero length packet
*/
    }
  }
  else
  {
    data_to_transfer = (Byte)wLength;       /* send only requested number of
data */
  }
  Usb_clear_RXSETUP() ;                     /* clear the receive setup flag
*/

  while (data_to_transfer > EP_CONTROL_LENGTH)
  {
    pbuffer = send_ep0_packet(pbuffer, EP_CONTROL_LENGTH);
    data_to_transfer -= EP_CONTROL_LENGTH;
    while ((!(Usb_rx_complete())) && (!(Usb_tx_complete())));
    if ((Usb_rx_complete()))                /* if no cancel from USB Host */
    {
      Usb_clear_RXOUT();
      return;
    }
    if (Usb_tx_complete())
    {
      Usb_clear_TXCMPL();
    }
  }
  /* send last data packet */
  pbuffer = send_ep0_packet(pbuffer, data_to_transfer);
  data_to_transfer = 0;
  while ((!(Usb_rx_complete())) && (!(Usb_tx_complete())));
  if ((Usb_rx_complete()))                  /* if no cancel from USB Host */
  {
    Usb_clear_RXOUT();
    return;
  }
  if (Usb_tx_complete())
  {
    Usb_clear_TXCMPL();
  }
  if (zlp == TRUE)
  {
    send_ep0_packet(pbuffer, 0);
    while ((!(Usb_rx_complete())) && (!(Usb_tx_complete())));
    if ((Usb_rx_complete()))              /* if no cancel from USB Host */
    {
      Usb_clear_RXOUT();
      return;
    }
    if (Usb_tx_complete())
    {
      Usb_clear_TXCMPL();
    }
  }
  while ((!(Usb_rx_complete())) && (!(Usb_setup_received())));
  if (Usb_setup_received())
  {
    return;
  }

  if (Usb_rx_complete())
  {
    Usb_clear_DIR();                        /* set in on EP0 */
    Usb_clear_RXOUT();
  }
}


/*F**************************************************************************
* NAME: usb_get_configuration
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*----------------------------------------------------------------------------
* PURPOSE:
* This function manages the GET_CONFIGURATION request.
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
****************************************************************************
/
void usb_get_configuration (void)
{
  Usb_clear_RXSETUP();
  Usb_set_DIR();
  if (USBCON & MSK_CONFG)
  {
    Usb_write_byte(1);
  }
  else
  {
    Usb_write_byte(0);
  }

  Usb_set_TXRDY();
  while (!(Usb_tx_complete()));
  Usb_clear_TXCMPL();
  while (!(Usb_rx_complete()));
  Usb_clear_RXOUT();
  Usb_clear_DIR();
}


/*F**************************************************************************
* NAME: usb_get_status
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*----------------------------------------------------------------------------
* PURPOSE:
* This function manages the GET_STATUS request.
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
void usb_get_status (void)
{
Byte wIndex;

  ACC = Usb_read_byte();                        /* dummy read */
  ACC = Usb_read_byte();                        /* dummy read */
  wIndex = Usb_read_byte();
  Usb_clear_RXSETUP();
  Usb_set_DIR();
  switch(bmRequestType)
  {
    case REQUEST_DEVICE_STATUS:
    {
      Usb_write_byte(SELF_POWERED);
      break;
    }
    case REQUEST_INTERFACE_STATUS:
    {
      Usb_write_byte(0x00);
      break;
    }
    case REQUEST_ENDPOINT_STATUS:
```

```
    {
      wIndex = wIndex & MSK_EP_DIR;
      Usb_write_byte(endpoint_status[wIndex]);
      break;
    }
  }
  Usb_write_byte(0x00);
  Usb_set_TXRDY();
  while ((!(Usb_tx_complete())) || (Usb_setup_received()));
  Usb_clear_TXCMPL();
  while ((!(Usb_rx_complete())) || (Usb_setup_received()));
  Usb_clear_RXOUT();
  Usb_clear_DIR();
}


/*F**************************************************************************
* NAME: usb_set_feature
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*----------------------------------------------------------------------------
* PURPOSE:
* This function manages the SET_FEATURE request.
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
void usb_set_feature (void)
{
  if (bmRequestType == ZERO_TYPE)
  {
    Usb_clear_RXSETUP();
    Usb_set_STALLRQ();
    while (!(Usb_STALL_sent()));
    Usb_clear_STALLRQ();
  }
  if (bmRequestType == INTERFACE_TYPE)
  {
    Usb_clear_RXSETUP();
    Usb_set_STALLRQ();
    while (!(Usb_STALL_sent()));
    Usb_clear_STALLRQ();
  }
  if (bmRequestType == ENDPOINT_TYPE)
  {
    if (Usb_read_byte() == 0x00)
    {
      ACC = Usb_read_byte();                /* dummy read */
      switch (Usb_read_byte())              /* check wIndex */
      {
        case ENDPOINT_1:
```

```
        {
          Usb_select_ep(EP_IN);
          Usb_set_STALLRQ();
          Usb_select_ep(EP_CONTROL);
          endpoint_status[1] = 0x01;
          Usb_clear_RXSETUP();
          Usb_set_TXRDY();
          while (!(Usb_tx_complete()));
          Usb_clear_TXCMPL();
          break;
        }
        case ENDPOINT_2:
        {
          Usb_select_ep(EP_OUT);
          Usb_set_STALLRQ();
          Usb_select_ep(EP_CONTROL);
          endpoint_status[2] = 0x01;
          Usb_clear_RXSETUP();
          Usb_set_TXRDY();
          while (!(Usb_tx_complete()));
          Usb_clear_TXCMPL();
          break;
        }
        default:
        {
          Usb_clear_RXSETUP();
          Usb_set_STALLRQ();
          while (!(Usb_STALL_sent()));
          Usb_clear_STALLRQ();
          break;
        }
      }
    }
  }
}


/*F***************************************************************************
* NAME: usb_clear_feature
*----------------------------------------------------------------------------
* PARAMS:
*
* return:
*----------------------------------------------------------------------------
* PURPOSE:
* This function manages the SET_FEATURE request.
*----------------------------------------------------------------------------
* EXAMPLE:
*----------------------------------------------------------------------------
* NOTE:
*----------------------------------------------------------------------------
* REQUIREMENTS:
*****************************************************************************
/
void usb_clear_feature (void)
{
  if (bmRequestType == ZERO_TYPE)
```

```
{
  Usb_clear_RXSETUP();
  Usb_set_STALLRQ();
  while (!(Usb_STALL_sent()));
  Usb_clear_STALLRQ();
}
if (bmRequestType == INTERFACE_TYPE)
{
  Usb_clear_RXSETUP();
  Usb_set_STALLRQ();
  while (!(Usb_STALL_sent()));
  Usb_clear_STALLRQ();
}
if (bmRequestType == ENDPOINT_TYPE)
{
  if (Usb_read_byte() == 0x00)
  {
    ACC = Usb_read_byte();                /* dummy read */
    switch (Usb_read_byte())              /* check wIndex */
    {
      case ENDPOINT_1:
      {
        Usb_select_ep(EP_IN);
            if(Usb_STALL_requested())
            {
          Usb_clear_STALLRQ();
            }
            if(Usb_STALL_sent())
            {
              Usb_clear_STALLED();
        }
        UEPRST = 0x02;
        UEPRST = 0x00;
        Usb_select_ep(EP_CONTROL);
        endpoint_status[EP_IN] = 0x00;
        Usb_clear_RXSETUP();
        Usb_set_TXRDY();
        while (!(Usb_tx_complete()));
        Usb_clear_TXCMPL();
        break;
      }
      case ENDPOINT_2:
      {
        Usb_select_ep(EP_OUT);
            if(Usb_STALL_requested())
            {
          Usb_clear_STALLRQ();
            }
            if(Usb_STALL_sent())
            {
              Usb_clear_STALLED();
        }
            UEPRST = 0x04;
        UEPRST = 0x00;
        Usb_select_ep(EP_CONTROL);
        endpoint_status[EP_OUT] = 0x00;
        Usb_clear_RXSETUP();
```

```
          Usb_set_TXRDY();
          while (!(Usb_tx_complete()));
          Usb_clear_TXCMPL();
          break;
        }
        case ENDPOINT_0:
        {
          Usb_clear_RXSETUP();
          Usb_set_TXRDY();
          while (!(Usb_tx_complete()));
          Usb_clear_TXCMPL();
          break;
        }
        default:
        {
          Usb_clear_RXSETUP();
          Usb_set_STALLRQ();
          while (!(Usb_STALL_sent()));
          Usb_clear_STALLRQ();
          break;
        }
      }
    }
  }
}
```