µALFAT™ Library Manual

Rev. 1.04 Date: July 15, 2008 Library Manual

Document Information

Information	Description
Abstract	This document covers complete information about µALFAT™ Library.

GHI Electronics

Table of Contents

1. µALFAT™Library

The provided library is just an example of how communications with µALFAT and the commands are handled. The library does **not** support all features provided by µALFAT. It can be used as a startup project but it must be ported to the target processor and changes must be made where needed. The library works out-of-the-box on USBFAT development system which is targeted for PIC micro controller. However, all functions are written in 'C' language, so they are highly portable to virtually any platform. When you download the library, you will get a complete MPLAB project for MCC18 compiler to get your system going. If you are using a micro other than PIC, use the file PIC_example.c as a starting point for you application. The base library file is "µALFAT_lib.c".

µALFAT User Manual **should** be consulted in all situation; it contains all details. The library functions are based on commands and structures provided in the User Manual.

2. µALFAT™Library Functions

Few data types are used. They are defined in “types.h”. This file also includes the type of project and other options.

Type	Limits	Byte Count
int8	0 to 255	1
int16	0 to 65535	2
int32	0 to 4,294,967,295	4

Note: All functions return an int8 which is an error code provided in µALFAT User Manual, unless, otherwise indicated.

2.1. Boot Loader Functions

These functions uses µALFAT boot loader. There is no need to use them unless updating the firmware.

Note: The Boot Loader sends data over USART, so it will not be possible to see the data when using I2C or SPI interface.

1. void GHI_SendLoadFirmwareFromSDCommand(void)

- Implements µALFAT command “X S”.
- This function commands µALFAT to update the firmware from an SD card.
- µALFATFW.GHI must be present on the media. This file holds the firmware.
- After successful file transferring, the new firmware executes and µALFAT sends the banner and version number of the loaded firmware.

2. void GHI_SendLoadFirmwareFromUSBCommand(void)

- Implements µALFAT command “X U”.
- This function commands µALFAT to update the firmware from a USB.
- µALFATFW.GHI must be present on the media. This file holds the firmware.
- After successful file transferring, the new firmware executes and µALFAT sends the banner and version number of the loaded firmware.

2.2. Firmware Functions

1. int16 GHI_GetLibraryVersion(void)

Gets the library version number in BCD format.

Ex: If the return value is 0x205, the version number is 2.05

2. int8 GHI_GetVersion(int8 * major, int8 * BCDminor)

Implements µALFAT command "V".

Obtaining the latest version is very useful and very important. Always keep µALFAT updated with latest firmware.

Major: Major release number.

BCDminor: This is a BCD number that represent the minor release.

For example:

0x12 is version x.12

0x32 is version x.32

3. int8 GHI_InitializeSD(void)

Implements µALFAT command "I". This function will initialize the SD card.

Must be called before accessing a newly attached storage media device.

4. int8 GHI_InitializeUSB(void)

Implements µALFAT command "U". This function will initialize the USB storage media.

Must be called before accessing a newly attached storage media device.

5. int8 GHI_OpenFile(int8 filehandle, int8 * filename, int8 openmode)

Implements µALFAT command "O".

filehandle: µALFAT supports 4 file handles. This value can be 0 to 3.

filename: file name

openmode: One ACSII character, represents three modes for opening files:

- 'R' to open an existing file for read
- 'W' to open new file for write and if the file already exists it will be deleted and re-written.
- 'A' to append to a file or create new file if it does not exist.

6. int8 GHI_FlushFile(int8 filehandle)

Implements µALFAT command “F”.

Flushes all buffered data to the media.

filehandle: 0 to 3

7. int8 GHI_CloseFile(int8 filehandle)

Implements µALFAT command “C”.

Flushes all buffered data to the media and closes the file handle.

filehandle: 0 to 3

8. int8 GHI_SendWriteCommand(int8 filehandle, int32 datasize)

Implements µALFAT command “W”.

Writing data to files is done in multiple stages:

1. Open a file for write or append
2. Send write request (GHI_SendWriteCommand)
3. If previous command passed, send your data. You can’t stop µALFAT at this point and you have to send all your data. This is why we recommend smaller data blocks.
4. When finished, µALFAT will return the results. Use GHI_GetWriteResults to query the write results.

filehandle: 0 to 3. The handle must be opened first.

datasize: Count of bytes needed.

9. int8 GHI_GetReadAndWriteResults(int32 * actualdatasize)

Gets µALFAT command “W”, “R” results.

After sending a write or read request and processing the data, µALFAT will try its best to process all the data. In some case, the file write or read could fail, if the media is full, for example. This function tells you how many bytes the read or write command was able to process.

After writing, some data maybe buffered inside µALFAT and you have to flush the file before 100% of the data exist in the card.

actualdatasize: A pointer to int32 that returns how many bytes were actually written/read. In most cases the returned value is the same as “datasize” requested for write or read operation.

10. int8 GHI_Send ReadCommand(int8 filehandle, int32 datasize, int8 filler)

Implements µALFAT command “R”.

Reading is done in multiple stages:

1. Open a file for read
2. Send read request (GHI_SendReadCommand)
3. If previous command passed, µALFAT will return the data from the file. It will send “datasize” bytes
4. When finished, µALFAT will return the results. Use GHI_GetWriteResults to query the read results.
5. If µALFAT failed to read the data from the file it will send back the bytes as “filler”. For example, if µALFAT was only able to read 8 out of 10 requested bytes, it will send 8 bytes actual data from the file and 2 filler bytes.

filehandle: 0 to 3. The handle must be opened first.

datasize: Count of bytes needed.

filler: The filler can be any value of your choice.

11. int8 GHI_ReadFile(int8 handle, int8 *buffer, int32 size, int8 filler, int32 *actualdatasize)

This is an optional wrapper function to do all the steps of reading a file through a buffer.

- _USE_READWRITE_WRAPERS_ Must be defined before using this function, it appears at the top of µALFAT_lib.h.
- comment this definition to reduce the library code size.

handle: 0 to 3. The handle must be opened first.

buffer: This should be provided by the user to hold the read data.

size: Size of data to read

filler: The filler can be any value of your choice.

actualdatasize: A pointer to int32 that returns how many bytes were actually read. In most cases the returned value is the same as “size” in the read request function.

12. int8 GHI_WriteFile(int8 handle, int8 *buffer, int32 size, int32 *actualdatasize)

This is an optional wrapper function to do all the steps of writing a file through a buffer.

- _USE_READWRITE_WRAPERS_ Must be defined before using this function, it appears at the top of µALFAT_lib.h

- comment this definition to reduce the library code size.

handle: 0 to 3. The handle must be opened first.

buffer: This should be provided by the user to get the data from.

size: Size of data to write

actualdatasize: A pointer to int32 that returns how many bytes were actually written. In most cases the returned value is the same as "size" in the write request function.

13. int8 GHI_SeekFile(int8 filehandle, int32 newposition)

Implements µALFAT command "P". Sets the file pointer to new position.

filehandle: 0 to 3. The handle must be opened first.

newposition: New position

14. int8 GHI_DeleteFile(int8 * filename)

Implements µALFAT command "D".

Deletes a file from the media. The file must exist and it should not be opened by any handle. µALFAT doesn't check if the file is opened by a handle.

filename: A null terminated string with the file name

15. int8 GHI_GetFileInfo(int8 * filename, int32 * size, int8 * attributes)

Implements µALFAT command "?".

Use this function to find a specific file with a known name.

filename: A null terminated string with the file name

size: This pointer stores the file size

attributes: File Attributes are one byte standard Attribute structure in FAT system, see Appendix.

16. int8 GHI_InitGetFile(void)

Implements µALFAT command "@".

This function must be called before using GetNextFile to return the pointer to the first entry in a directory list.

17. int8 GHI_GetNextFile(int8 * filename, int8 * fileext, int8 * attributes, int32 * size)

Implements µALFAT command "N".

To get the directory list, use GHI_InitGetFile once and then keep pooling GHI_GetNextFile until you have read all the needed directories. You cannot access files in between GHI_GetNextFile calls.

filename: A null terminated string with the file name

fileext: A null terminated string with the file extension

attributes: File Attributes are one byte standard Attribute structure in FAT system, see Appendix.

size: The file size in bytes

Note: ERROR_END_OF_DIR_LIST value is returned if the entire directory list is read.

18. int8 GHI_ChangeDirectory(int8 * dirname)

Implements µALFAT command "A".

Changes the current working directory.

dirname: A null terminated string with the directory name.

19. int8 GHI_MakeDirectory(int8 * dirname)

Implements µALFAT command "M".

Creates a new directory (folder) on the connected media.

dirname: A null terminated string with the directory name.

20. void GHI_StartMediaStatistics (void)

Implements µALFAT command "K".

Depending on the media size, this function can take several seconds to finish. This function will obtain the media size and free size in sectors, where each sector is 512 bytes. Use GHI_GetResultMediaStatistics to obtain the values.

21. int8 GHI_GetResultMediaStatistics(int32 * size, int32 * free)

Gets µALFAT command "K" results.

After sending GHI_StartMediaStatistics, use this function to obtain the size of the media and the free space size.

size: Media size

free: Free space

22. int8 GHI_µALFATPowerMode(int8 mode, int16 baud)

Implements µALFAT command “Z F”/”Z R”.

- Sets µALFAT in full or reduced power modes.
- µALFAT power up in full power mode.

mode: This can be ‘F’ for full power or ‘R’ for reduced power mode.

baud: The baudrate at which µALFAT will operate.

23. void GHI_µALFATHibernate(void)

Implements µALFAT command “Z H”.

In case you need to go in low power mode but keep the file handles open, you need to use GHI_µALFATHibernate

The only way to wake µALFAT after using this function, is with a power cycle or a toggle on the WAKE pin int8 GHI_µALFATWake(void); This is used to wakeup µALFAT from sleep and obtains any error codes.

24. int8 GHI_InitializeTime(int8 backup)

Implements µALFAT command “T”.

The Real Time Clock inside µALFAT is needed to set the correct dates on the saved files. There are 2 options for µALFAT RTC. It can be run using the same power source and oscillator as the processor core or it can run off 32Khz clock and a backup battery. Use this function when you need to switch between the 2 options.

backup: A flag that is when true, µALFAT will run the RTC from the backup battery and when it is false, it will run the RTC using the same power source and oscillator as the core processor.

25. int8 GHI_GetTime(int32 * time)

Implements µALFAT command “G X”.

If you need to obtain the time, to save it to a file for example, use this function.

time: A pointer to hold the time value. The data is in the same format used by FAT file system, see Appendix.

26. int8 GHI_GetFormattedTime(int8 *buffer)

Implements µALFAT command “G F”.

To get the time in a formatted way.

buffer: A null-terminated buffer that will hold the time. It must be at least 22 bytes. **Ex:** 05/12/2009 – 02:50:34

Note: This function is a little slower than GHI_GetTime(time).

27. int8 GHI_SetTime(int32 time)

Implements μALFAT command "S".

Sets the RTC.

time: Holds the time value to be set. The data is in the same format used by FAT file system, see Appendix.

28. int32 GetFATTimeStructure(int32 year, int32 month, int32 day, int32 hours, int32 minutes, int32 seconds)

- This function takes the time as parameters and returns a 32-Bit variable that holds the FAT Time Structure.
- The arguments are self explanatory.

Return: 32-Bit variable that holds the time. See Appendix.

2.3. Driver Functions

μALFAT_lib is written to work on any processor but there will be a need to implement a few driver functions that will help μALFAT_lib in using your processor. Examples of the driver functions for UART, SPI and I2C on a PIC are provided. The library does not know anything about your processor or the interface you want to use. For example, When the library wants to send a byte to μALFAT, it will call GHI_PutC function. Therefore, the user **must** provide the appropriate driver functions.

The term interface, used here, refers to SPI/UART/I2C

1. void GHI_Sleep(int16 ms)

In some situations, μALFAT_lib will require some delay on some task. All delays happen through GHI_Sleep. This function will return after "ms" milliseconds but it doesn't have to be accurate.

ms: how many millisecond to loop

2. int8 GHI_OpenInterface(void)

The library does not need this function but you can use it at power up to initialize the interface. The interface can be UART, SPI or I2C.

3. int8 GHI_CloseInterface(void)

Closes the interface, if necessary.

4. int8 GHI_SetBaudRate(int32 baud)

This is needed only if the interface is UART. It is a good practice to switch the baud rate to a faster one. µALFAT powers up with 9600 baud.

baud: Baudrate

5. int8 GHI_GetC(void)

When there is data ready in the interface receive buffer, GHI_GetC will fetch it and return immediately. If there is no data ready, GHI_GetC will wait for data to be available. You can have a timeout and a return 0 if there is no data for a long time to prevent code lockups.

The implementation can simply pool the interface or it can read a FIFO that is filled by the interface's interrupt routine.

Return: A character (byte) from the interface when ready.

6. void GHI_PutC(int8 ch)

If the interface is ready to transmit data, GHI_PutC will place a byte in the transmit buffer.

ch: A character (byte) to be transferred to the interface when ready.

7. void GHI_PutS(int8 * str)

Transfers a null terminated string to the interface. Be careful when you use a processor that has RAM and ROM pointers, for example, a PIC. GHI_PutS will work with Ram pointers only. Most processors use the same pointers for RAM and ROM, including a PC processor.

str: A null terminated string to be transmitted.

8. int8 GHI_DataIsReady(void)

It is a good practice to check if there is some data in the receive buffer before using GHI_GetC to avoid lockups.

Return: 1 if data is ready and 0 if there is no data ready.

9. void GHI_ToggleWakePin(void)

µALFAT_lib does not know how to toggle the wake pin on your system. Implement this function to do so if you need to use the sleep mode.

3. Examples

Note:

- The following code require proper set-up in the program.
- For the sake of simplicity, the code does not check for error codes. It is highly recommended that you do.

3.1. Simple Write/Read Process

```
// Write 2 bytes to a file
int8 file_name[] = “FILE.EXT”, buffer[3];
int32 size;
GHI_InitializeUSB(); // Assuming using USB
GHI_OpenFile(0, file_name, ‘W’); // open a file for write using handle 0
GHI_SendWriteCommand(0, 2); // write 2 bytes to handle 0
GHI_PutC(‘V’); // sending 2 bytes
GHI_PutC(‘A’);
GHI_GetReadAndWriteResults(&size); // after this, the size should be 2
GHI_CloseFile(0); // release the handle
// writing done, let’s read it
GHI_OpenFile(0, file_name, ‘R’); // re-open the same file for read
GHI_SendReadCommand(0, 3, ‘*’); // read 3 bytes using ‘*’ as a filler
GHI_GetC(&buffer[0]); // read three bytes
GHI_GetC(&buffer[1]);
GHI_GetC(&buffer[2]);
/*
/* Now the buffer should look like this:
/* buffer[0]  ‘V’
/* buffer[1]  ‘A’
/* buffer[2]  ‘*’
*/
GHI_GetReadAndWriteResults(&size); // after this, the size should be 2
GHI_CloseFile(0); // release the handle
```

3.2. Enumerating Files

```
// read directory list
int8 file_name[16], file_ext[4], attributes;
```

```
int32 size;
GHI_InitializeSD(); // assuming using an SD card
GHI_InitGetFile(); // Start the list
While( GHI_GetNextFile(file_name, file_ext, &attribute, &size) !=
ERROR_END_OF_DIR_LIST ) // loop until the end
{
   printf("%s\n", file_name); // just print the file name
}
```

3.3. Real Time Clock

```
// …
int32 seconds, minutes, hours, day, month, years;
int32 time;
int8 buffer[32];
// set the time parameters
seconds = 40;
minutes = 30;
hours = 4;
day = 11;
month = 8;
years = 2006;
// Set the time into 32bit structure
time = GetFATTimeStructure(years, month, day, hours, minutes, seconds);
GHI_InitTime(0); // initialize with '0' "No backup battery", use '1' if any
GHI_SetTime(time);// set the time
// After 1 hour, 20 minutes
GHI_GetFormattedTime(buffer); // read the time, you can also use
// GHI_GetTime(&time);
printf("%s\n", buffer); // print it
/*
/* the output should look like this:
/* 08/11/2006 – 05:50:40
*/
```

Appendix A

1. FAT attributes structure

7	6	5	4	3	2	1	0
Reserved		Archive	Folder	Volume ID	System	Hidden	Read Only

2. FAT Time structure

Bits(s)	Field	Description
31..25	Year1980	Years since 1980
24..21	Month	1..12
20..16	Day	1..31
15..11	Hour	0..23
10..5	Minute	0..59
4..0	Second2	Seconds divided by 2 (0..30)

DISCLAIMER