uALFAT™ Tutorial

Step-by-step and hands-on

Rev 1.5

GHI Electronics, LLC
www.ghielectronics.com

Updated – May 8, 2006

Table of Contents

1. Before I receive my new uALFAT-SD? 4
1.1. Power Supply 4
1.2. RS232 Level Converter 4
1.3. Processor Prototyping Board 5
1.4. uPICFAT 5
2. New board has just arrived! 6
2.1. Placing 0.1" header 6
2.2. Connect the RS232 circuit 6
2.3. uALFAT-SD pin-out 7
2.4. Starting a Terminal Program 7
2.5. First power up 9
3. uALFAT Creates the First File 10
3.1. Running the Firmware 10
3.2. Initialize the SD card and create a folder 10
3.3. Writing to a file 11
3.4. Reading from a file 13
4. Updating uALFAT firmware 14
5. Using a microcontroller with SPI interface 16
5.1. Why SPI? 16
5.2. What is SPI? 16
5.3. More on SPI 17
5.4. Choosing a microcontroller 17
5.5. Ordering everything needed 18
5.6. Wiring the circuit board. 19
5.7. I am not using a PIC! 20
5.8. The uALFAT_lib library 20
5.9. uALFAT_lib library Functions 20
5.9.1. int8 GHI_InitializeSD(void); 20
5.9.2. int8 GHI_OpenFile(int8 filehandle, int8 * filename, int8 openmode); 21
5.9.3. int8 GHI_FlushFile(int8 filehandle); 21
5.9.4. int8 GHI_CloseFile(int8 filehandle); 21
5.9.5. int8 GHI_DeleteFile(int8 * filename); 21
5.9.6. int8 GHI_GetFileInfo(int8 * filename, int32 * size, int8 * attributes); 22
5.9.7. int8 GHI_InitGetFile(int8 * filename, int32 * size, int8 * attributes); 22
5.9.8. int8 GHI_GetNextFile(int8 * filename, int8 * fileext, int8 * attributes, int32 * size); 22
5.9.9. int8 GHI_SendWriteCommand(int8 filehandle, int32 datasize); 22
5.9.10. int8 GHI_GetReadAndWriteResults(int32 * actualdatasize) 23
5.9.11. int8 GHI_Send ReadCommand(int8 filehandle, int32 datasize, int8 filler); 23

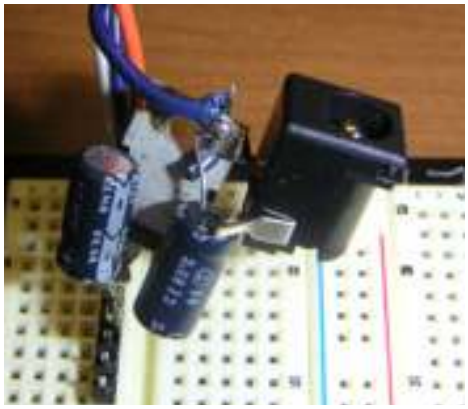
5.9.12. int8 GHI_ChangeDirectory(int8 * dirname);....24
5.9.13. int8 GHI_MakeDirectory(int8 * dirname);....24
5.9.14. int8 GHI_RemoveDirectory(int8 * dirname);....24
5.9.15. void GHI_StartQFormat(void);....24
5.9.16. void GHI_StartMediaStatistics (void);....24
5.9.17. int8 GHI_GetResultMediaStatistics(int32 * size, int32 * free);....25
5.9.18. int8 GHI_ReadSector(int8 filehandle, int32 sectornum, int8 ascii);....25
5.9.19. int8 GHI_WriteSector(int8 filehandle, int32 sectornum);....25
5.9.20. void GHI_uALFATSleep(void);....26
5.9.21. int8 GHI_uALFATWake(void);....26
5.9.22. int8 GHI_GetVersion(int8 * major, int8 * BCDminor);....26
5.9.23. int8 GHI_InitializeTime(int8 backup)....26
5.9.24. int8 GHI_GetTime(int32 * time)....27
5.9.25. int8 GHI_SetTime(int32 time)....27
5.10. Driver Functions....27
5.10.1. void GHI_Sleep(int16 ms);....27
5.10.2. int8 GHI_OpenInterface(void);....28
5.10.3. int8 GHI_CloseInterface(void);....28
5.10.4. int8 GHI_SetBaudRate(int32 baud);....28
5.10.5. int8 GHI_GetC(void);....29
5.10.6. void GHI_PutC(int8 ch);....29
5.10.7. void GHI_PutS(int8 * str);....29
5.10.8. int8 GHI_DataIsReady(void);....29
5.10.9. void GHI_ToggleWakePin(void);....30
6. From SPI to I2C....31
Similar to SPI, I2C is a serial interface. Even though it is similar, there are major differences on how communication happens. I2C uses 2 open-drain lines. The open drain pins are pulled-up through 2 resistors. The interface usually has one master and one or more slaves. The master starts communication by sending “start” signal followed by the slave address. Every slave is required to have a unique address on the I2C bus. Also, when the master sends the address, it sends a flag telling the slave it needs to read or write. When the master is done with data transfers, it sends “stop” signal....31
Here is a table with I2C wire connections on my board....31
7. uALFAT Error Codes....32

1. Before I receive my new uALFAT-SD?

Before you get the board, make sure you have 3.3V regulated voltage, RS232 level converter and a development board with a processor of your choice (PIC, AVR, basic stamp…etc.) The RS232 level converter is not required but it will ease the understanding of uALFAT commands.

1.1. Power Supply

If you have a variable bench power supply, you should be all set. Just set the voltage to 3.3V. Now, if you need to build your own, there is a bit more work involved. You will need any 3.3V regulator chips and 2 electrolytic capacitors. Your local store such as radio shack will have some regulators or you can order it online from companies like www.digikey.com. LD330 chip is a good example. You will need to add one capacitor on the input and one on the output. Look at your regulator's datasheet for more details.



IMPORTANT: Any voltage other than 3.3V may damage uALFAT chip and the SD card. This voids the warranty on the chip and the board.

1.2. RS232 Level Converter

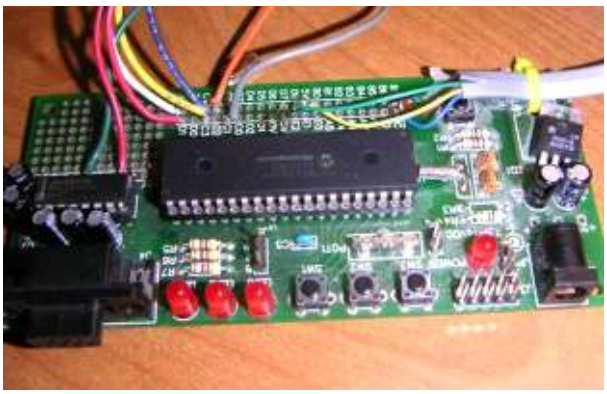
The easiest way around this is to buy a finished board. There are many out there for $10 to $20 but it is always more fun to build your own! These chips are available everywhere. Keep in mind that your source is 3.3V. If you use RS232 level converter chip that runs off 5V, you will need 2 different voltage sources. A good example is our ALFAT-DEV schematic on our website. Look it up and see how MAX3232 chip is connected.

The reason you need this chip is to connect uALFAT-SD directly to the serial port on your PC. If you want to skip this test then you don't need RS232 circuit but we highly recommend doing the PC connection test.

1.3. Processor Prototyping Board

You can use any processor of your choice. You will need UART, SPI or I2C for communication. Your processor can run at 5V or 3.3V because uALFAT is 5V tolerant but remember that your processor must accept 3.3V levels or you will have to add level shifters. Most micros do accept 3.3V levels, PIC for example.

In these examples, we used a PIC running on 5V. See picture to the right.



1.4. uPICFAT

The better option to start with uALFAT is to use uPICFAT board. This board comes with everything you need to test uALFAT and even make a whole project out of it. The board comes in a form of a kit that includes everything you need, even the power supply and the RS232 cable is included. On the software side, uPICFAT comes with a full

library and many examples.

We highly recommend starting up with uPICFAT.

2. New board has just arrived!

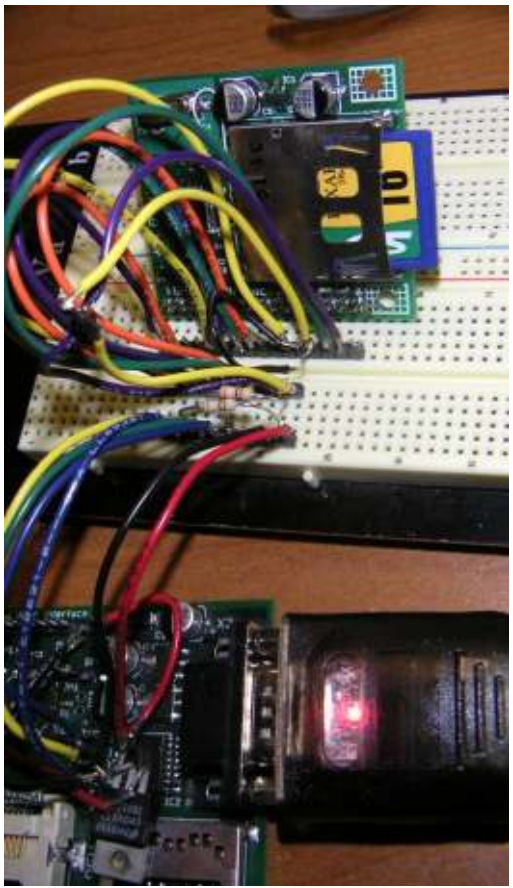
Congratulations, you have just received your uALFAT-SD board. You are excited to test it out and can't wait to create files. If you have purchased uPICFAT board then you can skip many of the following steps.

2.1. Placing 0.1" header

The first thing you want to test is the functionality of your hardware. First, connect wires to the needed pins. We recommend placing a 0.1" pin header on the board and then plug it in a bread board.

2.2. Connect the RS232 circuit

Now, connect power and your RS232 circuit and plug in your serial cable from your PC. We used, off the shelf, USB to serial cable in this test. Don't plug in the power yet. Lets take a look at the manual. We have everything setup except the mode select. uALFAT needs to know if you want it to run in UART, SPI or I2C mode.



SPI_SSEL#	SPI_SCK	Interface
0	0	UART
0	1	Skip boot loader
1	0	I2C

1	1	SPI

For UART, we need SPI_SCK and SPI_SSEL pins connected to ground. We used resistors to pull these pins to ground but you can safely connect them straight to ground.

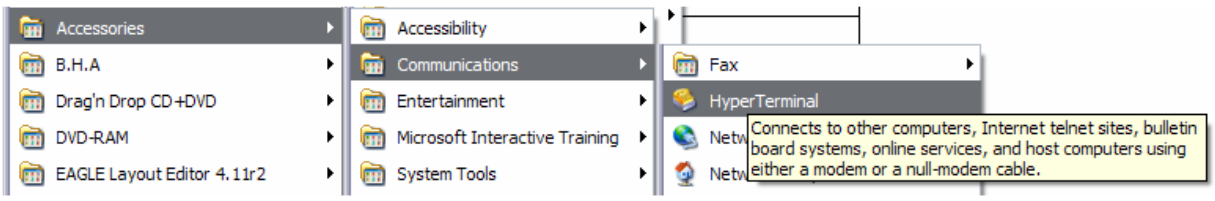
2.3. uALFAT-SD pin-out

Here is the pin out of uALFAT-SD board.

Pin #	Function
1	UART_TX/DATARDY
2	UART_RX/BUSY
3	I2C_SCL
4	I2C_SDA
5	SPI_SCK
6	SPI_MISO/UART_RTS
7	SPI_MOSI/UART_CTS
8	SPI_SSEL
9	MISC
10	CARD_DET (detect)
11	WAKE
12	VBAT
13	3.3 V power
14	RESET
15	GROUND
16	CARD_WP (write protect)

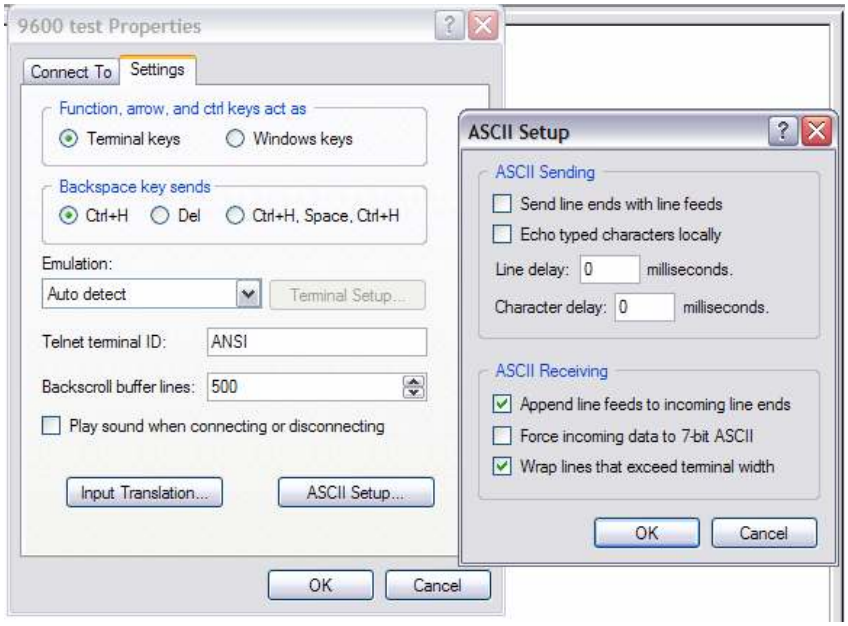
2.4. Starting a Terminal Program

You can use any terminal program such as “tera term” or “hyper terminal.” We will use hyper terminal since it comes already installed with windows OS.



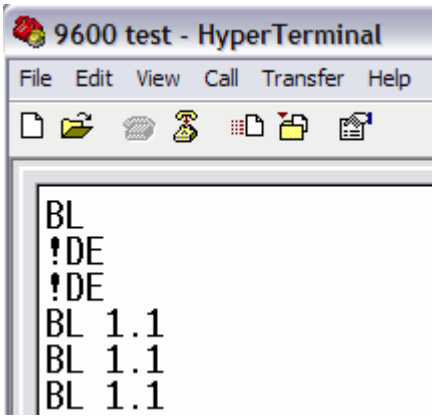
First select the COM port number if the port you want to use on your PC. Usually, it is COM1 if you have COM port on the mother board and COM4 if you are using USB serial cable. Now, we need to configure the terminal. You need to set it to 9600, no parity, 8 bits, 1 stop bit and hardware handshaking.

uALFAT returns "carriage return" at the end of every line. It will never return "line feed" For that, you need to append line feed to carriage returns. On hyper terminal, click on "file → properties" then click on "ASCII settings…" button. Make sure to change your settings to mach the picture below.



2.5. First power up

Finally, we are ready to power up the board. Power up the board, and take a look at the terminal program. You should see 'BL' on the screen. If you press 'V', you will get back the boot loader version number.



IMPORTANT: The boot loader doesn't accept lower case; therefore, 'v' will not work, it has to be upper case 'V'.

Pressing 'v' will return !DE and pressing 'V' will return BL 1.1
'!' symbol indicates an error and it is always followed by the error number. DE means "unknown command"

If you got this far, you are ready for some fun. If you can't see the responses as the picture above, double check your settings of the terminal and the voltages. Make sure you have CTS and RTS connected and if they are not used then CTS on uALFAT must be connected to ground the terminal "flow control" must be set to "none"

3. uALFAT Creates the First File

3.1. Running the Firmware

Press 'R' key to run the firmware. If you see 'BL' returning back then uALFAT is not loaded with any firmware or the firmware is corrupted. Read "Updating uALFAT Firmware" for details on how to load the firmware. If everything is okay you will see GHI Electronics banner and the version number of the firmware.

```
BL

 GHI Electronics, LLC
----------------------
    uALFAT(TM) 1.10
!00
```

At the end, you will see !00 indicating that uALFAT is ready for your commands. All uALFAT commands are one character or symbol and usually are followed by some parameters.

First thing we want to do is to enable echo. When you enable echo, uALFAT will echo back what you send it. The echo command is the '#' symbol. The parameter is 0 or 1. 0 will disable echo and 1 will enable echo. All commands must be followed by a space. Now you know how, send uALFAT a command to enable echo. This will be accomplished be sending "# 1". Note that you will not type the quotes and you will end by pressing the enter key on your keyboard (carriage return.) After that, uALFAT will echo back everything you type. You can use backspace to erase typos. Same as the boot loader, uALFAT will return '!' followed by the error number if it detected some error. The 'V' command returns the version number as you can see in the picture to the right.

```
!00
Data echoed back!
!68
V
uALFAT 1.10
!00
```

3.2. Initialize the SD card and create a folder

Insert any SD card in the connector and initialize it using the 'I' command. When done, display a list of files using '@' command followed by multiple 'N' commands. !4D is not really an error. It is an indication that there is no more files/folders to list.

Let us create a folder and name it "MYFOLDER" on the card. Use "M MYFOLDER" and remember to finish the

```
I
!00
@
!00
N
!00
UALFATFW.GHI 20 00005D30
!00
N
!4D
M MYFOLDER
!00
@
!00
N
!00
UALFATFW.GHI 20 00005D30
!00
N
!00
MYFOLDER.    10 00000000
!00
```

command by pressing the enter key. All the previous commands should execute without any errors.

Each file is listed with its attribute and size. Directories attribute is 0x10.

You have probably noticed that files are followed by the size in HEX and decimal. Folders are followed by <DIR>

Now, remove the card from uALFAT and plug it into the PC to verify the creation of the new folder.

3.3. *Writing to a file*

Now that you have a new folder, we want to write a file to go inside that folder. Keep in mind that you need to put the card back in uALFAT and you need to resend 'I' command to reinitialize the card.

To access an existing folder, use 'A' command. Let us send 'A MYFOLDER' and hit enter. You should get the prompt back without errors. Now try 'L' command again. This time you will see "dot" and "dot dot" folders. In FAT file system, "dot" is a pointer to the same folder and "dot dot" is a pointer to the

```
!00
@
!00
N
!00
.          .     10 00000000
!00
N
!00
..         .     10 00000000
!00
A ..
!00
@
!00
N
!00
UALFATFW.GHI 20 00005D30
!00
N
!00
MYFOLDER.     10 00000000
!00
```

parent folder. To go back and access the previous folder use ‘A ..’. You can also use ‘A \’ to access the root folder.

For the sake of this tutorial, we have created used the ‘L’ command that lists all entries in the current directory. This command will be removed in future releases and shouldn’t be used by users.

Make sure you have access to “MYFOLDER” and send open new file for write command. ‘O 2W>VOLTAGE.TXT’ this command will create a new file under file handle number 2. On uALFAT, you have 4 file handles, 0 to 3. List the files and see if the new file exists or not. As you can see the file size is 0.

```
O 2W>VOLTAGE.TXT
!00
L

. <DIR>
.. <DIR>
VOLTAGE.TXT 00000000 0!00
```

Witting data to a file is as easy as all other commands. You can send as many write commands as you wish. The write command ‘W 2>10’ will write 16 bytes (10 HEX = 16 decimal) to the open file. After uALFAT sees the command, it will respond with !00 if command was accepted. At this point, you can send any data you like to go in the file. Data can be ASCII or binary data. When uALFAT is done receiving all the data, it will send back the result of the write starting with how many bytes it was able to write and followed by the error code. Data is not echoed and this is why it is not showing in the snapshot to the right.

```
W 2>10
!00
$00000010
!00
L

. <DIR>
.. <DIR>
VOLTAGE.TXT 00000000 0!00
F 2
!00
L

. <DIR>
.. <DIR>
VOLTAGE.TXT 00000010 16!00
```

After the write, run the list command and you will notice that the file size is still zero. This is because uALFAT buffers the data internally until the write is a must. This speeds us the file processing and it prolong the life of your card. To force the data to be flushed to the card, use the flush command 'F'. Now, check the file size again and you will see the right file size.

Let us assume that we need to add more data to the file. Simply, send another write command. When finished close the file using 'C 2'. Move the file to your PC and make sure the data exists in the file. In our example, we put this string in the file "1234567890ABCDEF"

3.4. Reading from a file

Similar to write, read is very easily done. We start by opening a file but this time we open it for read, 'O 1R>VOLTAGE.TXT'. This time we will use handle number 1 to open the file. As you can see 'R 1Z>5' will read 5 bytes from the file opened by handle number 1. But what was the Z for? In case uALFAT had some issue while it was reading the file it still has to return the promised data.

```
O 1R>VOLTAGE.TXT
!00
R 1Z>5
!00
12345$00000005
!00
R 1Z>10
!00
67890abcdefZZZZZ$0000000B
!00
```

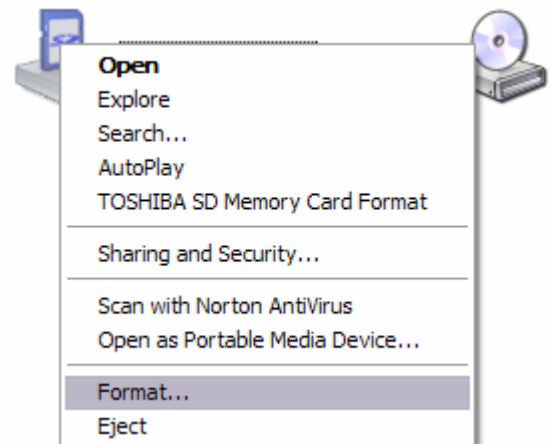
In this case, uALFAT will use the filler character to fill in the data. This can be a unique character in your system to indicate bad data before you even receive the error code. The character can be anything including '\0' (NULL)

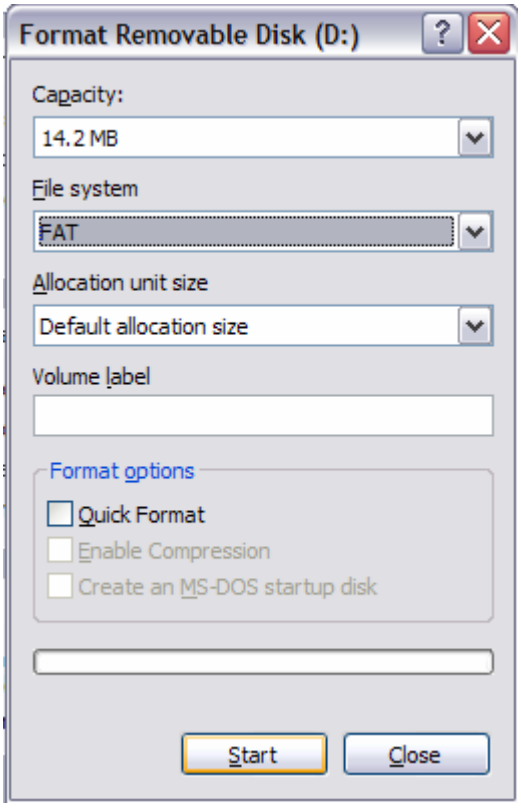
As you can see, the second read request was for 16 bytes but the file had only 11 bytes left in it (16-5) In this case uALFAT returned the promised 16 characters but the last 5 were the filler character, Z in this case.

If you wish to read a certain location in the file you can use the position command 'P' to place the file read pointer. In this example we will return the pointer to the beginning of the file using 'P 1>0', see above.

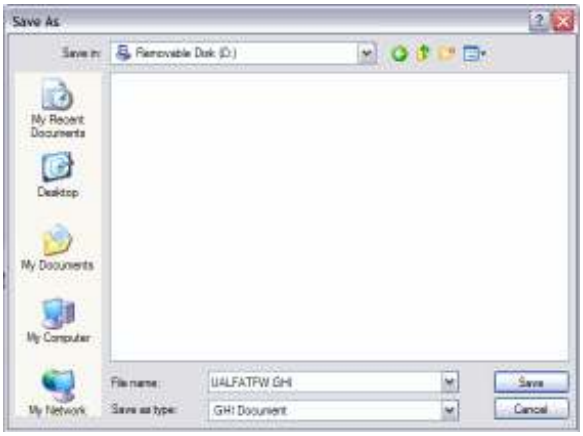
4. Updating uALFAT firmware

We are constantly enhancing uALFAT firmware and we want it to be extremely easy for you to update the firmware on your own. You r first step is to download the firmware from our website www.ghielectronics.com. You need to save that file on a formatted SD card.





Once the media is formatted, download the latest firmware from our website from uALFAT page and place the file on the memory card.



uALFAT Downloads

- User Manual
- Step by step Tutorial 1.5
- Code Library 0.4 beta
- Latest Firmware 1.xx
- LQFP48 package outline

Place the card in the connector of uALFAT-SD and reset the board. You should go back to the boot loader and see 'BL'. Loading the new firmware is accomplished by 'LOK' ok command. You will not see the command when you type it. Just makes sure you enter the three letters in upper case an in the right order. You will see uALFAT prompting for new sector write 'Wxx' where xx is the sector number. If any error happened, uALFAT will display the error number; otherwise, you will see !00 at the end of the firmware write.

```
W25
W26
W27
W28
W29
W2A
W2B
W2C
!00
```

When loading succeed, send 'R' to run the new firmware and check the version number

5. Using a microcontroller with SPI interface

5.1. Why SPI?

uALFAT runs on a standard interfaces (UART, SPI and I2C) that can be implemented in hardware or software. SPI will be the easiest to implement in software as you are simply shifting bits. Any modern microcontroller will have three of the supported interface. So if you have 3 of them, which one would you use? I would use UART because you only need 4 wires and even 2 are enough in some cases.

The problem is that most people already have UART used for debugging or to do other work. UART can't be connected to more than one device. Unlike UART, SPI and I2C can be connected to multiple devices. SPI is faster than I2C so let us use SPI.

5.2. What is SPI?

SPI is in a synchronous serial interface. Usually, SPI has one master and one or more slaves. Basically, on every side there is a shift register that has an input and an output and on every clock (SCK) edge, it will shift one bit out on the output and take a new bit from the input. After 8 clocks, the master and the one slave will have their buffers swapped. For example, if we have 11 in the master's buffer and 22 in the slave's buffer, the master generates 8 clocks and we will end up with 11 in the slave's buffer and 22 in the master's buffer.

The master uses a slave select pin (SSEL) that will tell the one slave that the clocks are meant for it. If there is a system with 4 slaves, there will 4 different SSEL lines going to the different slaves but one and only one SSEL line can be selected (low) at once.

The two signals used to transfer data to/from the master/slave are:

- MOSI: Master Out Slave In
- MISO: Master In Slave Out

5.3. More on SPI

uALFAT is slave SPI. On SPI, slaves have no way of informing the master of new data. This is why we added DATARDY. When this pin is high, the connected microcontroller (you) will know that uALFAT want to send data and you must read it. Another problem arises here. What if uALFAT is busy and it can't accept any data? This is what BUSY line is for. When BUSY is high, do not send any data to uALFAT.

This section is a bit confusing but the code should explain it more. Because of how SPI is defined (not limitation of uALFAT), you read and write at the same time, swap buffers. So what if you just want to read data from uALFAT but you have nothing to send to it? In this case you need to use uALFAT's software tokens. uALFAT uses Half Data Token (HDT) and No Data Token (NDT.)

When uALFAT has no data to send, it will send NDT. Your microcontroller should do the same. When you need to read from uALFAT, you will send NDT. uALFAT ignores incoming NDTs like nothing came in and your microcontroller should do the same. NDT is the value 255 decimal of 0xFF hexadecimal. In most applications, you will never need to send 0xFF to uALFAT but if you had to? This is when HDT becomes handy. HDT is the value 254 or 0xFE.

Assuming you are writing some binary data to a file and you had to use 0xFF. In this case, send HDT followed by NDT. NDT+HDT give us 0xFF. The same applies if you need to send HDT. HDT+HDT give us 0xFE.

Enough with boring stuff and let us do something more fun!

5.4. Choosing a microcontroller

You can use any microcontroller you like, including PIC16, PIC18, PIC24, dsPIC, AVR, Zilog, 8051, ARM, HC12, basic stamp…and many more!! Even if your microcontroller doesn't have any of the interfaces, there are many examples online on how to implement those interfaces in software using simple general I/Os.

I am going to be using my old PIC board. It has been helping me for the past 4 years so why not use it again? If you looked at the picture in the first page, you will notice it is not so pretty but it does the job well, I hope.

5.5. Ordering everything needed

If you decided you want to go with PIC, the easiest way is to use our uPICFAT board. Priced very low and ir runs out-of the box. The PIC compiler we used for the examples is C18 from microchip. You can download trial or free version directly from their website www.microchip.com

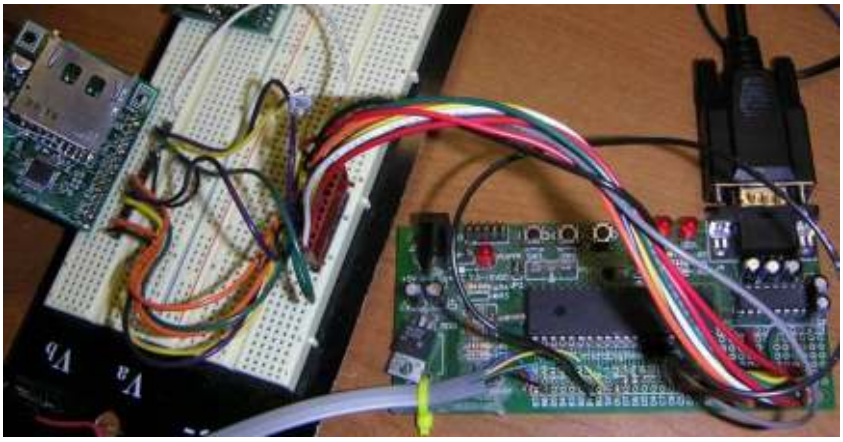
uPICFAT accepts uALFAT-SD and uALFAT-USB boards. uALFAT-USB is similar to uALFAT-SD but you read files from USB memory instead of SD cards. You can't buy uALFAT-USB yet but I can get one because I work for GHI Electronics ;-) I was told it will ship July 2006.

If you are implementing your own circuit, you will need to add 3.3V regulator. You can simply got to www.digikey.com and search for "regulator" Any regulator with 100mA or more will work just fine. Your local electronics store should have 3.3V regulator. Do not forget to add capacitors on the input and output of the regulator. **Always** carefully read the data sheet of whatever you are using.

This is it for the hardware. Now how do we write software? For PIC, go to www.microchip.com and download the free MPLAB. And also download the free C compiler for PIC MCC18.

Don't forget to go to www.ghielectronics.com and download the latest version of uALFAT library. It will be on the uALFAT main page under downloads.

5.6. Wiring the circuit board.



Not all pins need to be connected from uALFAT-SD board. For example, SD_DET is useful to detect the replacement of an SD card. For this simple tutorial, we will not use it. Also, the I2C pins are not needed at all.

This is a table of the pins and how they are connected. Please consult PIC18F458 or PIC18F452 for more details.

uALFAT-SD Pin #	uALFAT Function	Connected to
1	DATARDY	PIC RD2 (can be any pin)
2	BUSY	PIC RD1 (can be any pin)
3	I2C_SCL	Not connected
4	I2C_SDA	Not connected
5	SPI_SCK	PIC RC3/SCK
6	SPI_MISO	PIC RC4/SI
7	SPI_MOSI	PIC RC5/SO
8	SPI_SSEL	PIC RC1 (can be any pin)
9	MISC	Not connected
10	CARD_DET (detect)	Not connected
11	WAKE	Not connected
12	VBAT	Regulated 3.3V
13	3.3 V power	Regulated 3.3V
14	RESET	PIC RB1 (can be any pin)
15	GROUND	Ground
16	CARD_WP (write protect)	Not connected

5.7. I am not using a PIC!

Our uALFAT library (uALFAT_lib) is written in a layered manner. The low level file is the interface driver (GHI_inter.c). This can be written for any microcontroller and for any interface. You will have to write this. The second layer is the library core functions. These functions are written in pure 'C'. This file should work for any microcontroller (uALFA_lib.c) If you ever need to make any change on this file, let us know and we will update the library. You shouldn't need to make any change to the core file.

5.8. The uALFAT_lib library

When you download uALFAT_lib.zip file you will get a complete MPLAB project for MCC18 compiler. To get the library going to your system, you need to rewrite or modify GHI_inter.c. Use the file example.c as a starting point for you application. The final 'C' file is uALFAT_lib.c. You shouldn't need to make any changes on this file.

5.9. uALFAT_lib library Functions

First thing you need to know is we define our own variable types. These types are in "types.h" This file also includes the type of project options. Take a look at it.

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

5.9.1. int8 GHI_InitializeSD(void);

This function will initialize the SD card.

Return: uALFAT error code.

5.9.2. int8 GHI_OpenFile(int8 filehandle, int8 * filename, int8 openmode);

filehandle: uALFAT supports 4 file handles. This value can be 0 to 3. The handle must not be opened.
filename: and name but it has to comply with 8.3 standard (ex:FILE1234.TXT)
openmode: Three modes for opening files

- 'R' to open an existing file for read
- 'W' to open new file for read and if the file exists it will delete it first
- 'A' to open a file that existed and append new data to the end or create new file if it didn't exist.

Return: uALFAT error code.

5.9.3. int8 GHI_FlushFile(int8 filehandle);

Flushes all buffered data to the media.

filehandle: 0 to 3. The file has to be open for read.

Return: uALFAT error code.

5.9.4. int8 GHI_CloseFile(int8 filehandle);

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

filehandle: 0 to 3. The file has to be open already.

Return: uALFAT error code.

5.9.5. int8 GHI_DeleteFile(int8 * filename);

Delete a file from the media. The file must exist and it shouldn't be opened by any handle. uALFAT doesn't check if the file is opened by a handle.

filename: A null terminated string with the file name.

Return: uALFAT error code.

5.9.6. int8 GHI_GetFileInfo(int8 * filename, int32 * size, int8 * attributes);

Use this function to find a specific file with a known name. This will also return found directories. Use Attributes to determine the type of found entry.

Return: uALFAT error code.

5.9.7. int8 GHI_InitGetFile(int8 * filename, int32 * size, int8 * attributes);

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

Return: uALFAT error code.

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

In some applications, it would be useful to obtain a list of available files. Use GHI_InitGetFile once and then keep pooling GHI_GetNextFile until you have read all directories. You can access files in between GHI_GetNextFile calls.

Return: uALFAT error code.

5.9.9. int8 GHI_SendWriteCommand(int8 filehandle, int32 datasize);

Writing data to files happen 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 uALFAT at this point and you have to send all your data. This is why we recommend smaller data blocks.

4. When finished, uALFAT will return the results. Use GHI_GetWriteResults to query the write results.

filehandle: 0 to 3. The handle must be opened first.
datasize: How many bytes are needed.

Return: uALFAT error code.

5.9.10. int8 GHI_GetReadAndWriteResults(int32 * actualdatasize)

After sending a write or read request and finish processing the data, uALFAT 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 uALFAT and you have to flush the file before 100% of the data exist in the card.

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

Return: uALFAT error code.

5.9.11. int8 GHI_Send ReadCommand(int8 filehandle, int32 datasize, int8 filler);

Similar to writing data reading data from files happen in multiple stages.

5. Open a file for read
6. Send read request (GHI_SendReadCommand)
7. If previous command passed, uALFAT will return the data from the file. IT will send **"datasize"** bytes
8. When finished, uALFAT will return the results. Use GHI_GetWriteResults to query the write results.
9. If uALFAT failed to read the data from the file it will send back the bytes as **"filler"**. For example, if uALFAT said it can read 10 bytes but it was able to read only 8, it will send 8 bytes actual data from a file and 2 filler bytes.

filehandle: 0 to 3. The handle must be opened first.
datasize: How many bytes are needed.
filler: The filler can be any value of your choice.

Return: uALFAT error code.

5.9.12. int8 GHI_ChangeDirectory(int8 * dirname);

Changes the current access to a pre-existed directory (folder) on the connected media.

dirname: A null terminated string with the directory name.

Return: uALFAT error code.

5.9.13. int8 GHI_MakeDirectory(int8 * dirname);

Creates a new directory (folder) on the connected media. The name must be unique over the current directory span.

dirname: A null terminated string with the directory name.

Return: uALFAT error code.

5.9.14. int8 GHI_RemoveDirectory(int8 * dirname);

Removes a pre-existing directory (folder) from the connected media. The directory must be empty/

dirname: A null terminated string with the directory name.

Return: uALFAT error code.

5.9.15. void GHI_StartQFormat(void);

This function will completely wipe the files of the connected media. This function can take few seconds on a very large media. The function doesn't wait for the task to finish and returns immediately. Use GHI_GetResult to obtain the results.

5.9.16. void GHI_StartMediaStatistics (void);

Depending on the media size, this function could take couple seconds to finish. This function will obtain the media size in bytes and also the free bytes. Use GHI_GetResultMediaStatistics to obtain the values

5.9.17. int8 GHI_GetResultMediaStatistics(int32 * size, int32 * free);

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

size: A pointer to return the media size
free: A pointer to return the free space

Return: uALFAT error code.

5.9.18. int8 GHI_ReadSector(int8 filehandle, int32 sectornum, int8 ascii);

Rarely, you may with to read a sector directly from the media overriding the file system. The function requires a buffer internally to work and for that it uses one of the file handles. The file handle must be not is use.

filehandle: One of the free file handles
sectornum: the number of the sector you with to read
ascii: This is a flag. If it is true, uALFAT will return the data in ASCII HEX. If it is false, uALFAT will return the binary data as it sees it on the media.

Return: uALFAT error code.

5.9.19. int8 GHI_WriteSector(int8 filehandle, int32 sectornum);

You should never use this function unless you are well aware of what you are doing. uALFAT will take your data without any formatting whatsoever and place it in the sector.

filehandle: One of the free file handles
sectornum: the number of the sector you with to write

Return: uALFAT error code.

5.9.20. void GHI_uALFATSleep(void);

In low power applications, it is better to completely shut off the power of uALFAT and the SD card for zero power consumption. The Real Time Clock has a backup battery if there is a need to keep track of time.

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

The only way to wake uALFAT after this function is a power cycle or a toggle on WAKE pin

5.9.21. int8 GHI_uALFATWake(void);

This is used to wakeup uALFAT from sleep and obtains any error codes.

Return: uALFAT error code.

5.9.22. int8 GHI_GetVersion(int8 * major, int8 * BCDminor);

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

major: A pointer to return the major release number
Will be 1 for uALFAT with SD support and 2 for uALFAT with SD and USB support

BCDminor: This is a BCD number that represent the minor release.
For example: 0x12 = version x.12
0x32 = version x.32
0xDS= invalid value and will never happen!

Return: uALFAT error code.

5.9.23. int8 GHI_InitializeTime(int8 backup)

The Real Time Clock inside uALFAT is needed to set the correct dates on the saved files. There are 2 options for uALFAT 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 it is true, uALFAT will run the RTC on backup battery and when it is false it will run the RTC using the same power source and oscillator as core processor.

Return: uALFAT error code.

5.9.24. int8 GHI_GetTime(int32 * time)

If you need to obtain the time, to save to a file for example, use GHI_GetRTC.

time: A pointer to hold in the time value. The data is in the same format used by FAT file system and is defines in FATtime structure.

5.9.25. int8 GHI_SetTime(int32 time)

Similar to GHI_GetRTC but this one sets the RTC.

time: holds the time value to be set. The data is in the same format used by FAT file system and is defines in FATtime structure.

5.10. Driver Functions

uALFAT_lib is written to work on any processor but there will be a need to implement few driver functions that will help uALFAT_lib in using your processor. In our example code, the driver functions reside in GHI_inter.c file.

5.10.1. void GHI_Sleep(int16 ms);

In some situations, uALFAT_lib will require some delay on some task. All delays happen through GHI_Sleep. This function will return after x milliseconds. This function doesn't have to be accurate at all and can be

implemented using simple loops. You can test if your function is working right by simply toggling an LED and use 500 ms for delay.

```
Void BlinEveryOneSecond(void)
{
        LED=!LED; // toggle an LED
        // 500 ms low + 500 ms high = switch on every 1 second
        GHI_Sleep(500);
}
```

ms: how many millisecond to loop

5.10.2. int8 GHI_OpenInterface(void);

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

Note when we say interface from now on we will be referring to UART, SPI or I2C.

Return: 0 if okay or error code otherwise.

5.10.3. int8 GHI_CloseInterface(void);

Will close the interface. In most cases this won't be necessary.

Return: 0 if okay or error code otherwise.

5.10.4. int8 GHI_SetBaudRate(int32 baud);

This is needed only in the case of the interface is UART. It is a good practice to switch the baud rate to a faster one. uALFAT powers up with 9600 baud. This is very slow to what you can set the baud rate to.

baud: The baud is the rate of how many bit per seconds will be trnsfered.

Return: 0 if okay or error code otherwise.

5.10.5. int8 GHI_GetC(void);

When there is data ready in the interface receive buffer, GHI_GetC will fetch it and return it immediately. If there is no data ready, GHI_GetC will wait for data to be available. You can some timeout and 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.

5.10.6. void GHI_PutC(int8 ch);

If the interface is ready to transmit data, GHI_PutC will place a byte in it's transmit buffer. GHI_PutC will wait for the interface to become ready before it sends anything.

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

5.10.7. void GHI_PutS(int8 * str);

Very similar to GHI_PutS but GHI_PutS will traqnsfer a null terminated string to the interface. Be careful when you use a processor that has RAM and ROM pointers, a PIC for example. GHI_PutS will work with ram pointers only. Most processor use the same pointer for RAM and ROM, including your PC's processor.

str: a null terminated string to be transmitted to the interface.

Null terminated means that the least byte of the string must be zero.

5.10.8. int8 GHI_DataIsReady(void);

It is a good practice to check that there is some data in the receive buffer before using GHI_GetC so we will not lockup the code.

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

5.10.9. void GHI_ToggleWakePin(void);

uALFAT_lib doesn't know how to toggle the wake pin on your system. Implement this function to do so if you need to use the sleep mode.

6. From SPI to I2C

Similar to SPI, I2C is a serial interface. Even though it is similar, there are major differences on how communication happens. I2C uses 2 open-drain lines. The open drain pins are pulled-up through 2 resistors. The interface usually has one master and one or more slaves. The master starts communication by sending "start" signal followed by the slave address. Every slave is required to have a unique address on the I2C bus. Also, when the master sends the address, it sends a flag telling the slave it needs to read or write. When the master is done with data transfers, it sends "stop" signal.

Here is a table with I2C wire connections on my board

uALFAT-SD Pin #	uALFAT Function	Connected to
1	DATARDY	PIC RD2 (can be any pin)
2	BUSY	PIC RD1 (can be any pin)
3	I2C_SCL	PIC RC3/SCL
4	I2C_SDA	PIC RC4/SDA
5	SPI_SCK	Not Connected
6	SPI_MISO	Not connected
7	SPI_MOSI	Not connected
8	SPI_SSEL	PIC RC1 (can be any pin)
9	MISC	Not connected
10	CARD_DET (detect)	Not connected
11	WAKE	Not connected
12	VBAT	Regulated 3.3V
13	3.3 V power	Regulated 3.3V
14	RESET	PIC RB1 (can be any pin)
15	GROUND	Ground
16	CARD_WP (write protect)	Not connected

Don't forget to connect 2 resistors from SCL and SDA to power. On my test, I have 1K resistors and they are connected to 5V.

As far as using our library with PIC, you only need to use the I2C driver in "types.h"
#define _USE_GHI_INTR_PIC_I2C_

7. uALFAT Error Codes

Error Number	Description
0x00	No Error
0x01	ERROR_READ_SECTOR
0x02	ERROR_WRITE_SECTOR
0x03	ERROR_ERASE_SECTOR
0x11	ERROR_MBR_SIGNATURE_MISSMATCH
0x12	ERROR_BS_SIGNATURE_MISSMATCH
0x13	ERROR_SECTOR_SIZE_NOT_512
0x14	ERROR_FSINFO_SIGNATURE_MISSMATCH
0x21	ERROR_CLUSTER_OVER_RANGE
0x22	ERROR_CLUSTER_UNDER_RANGE
0x23	ERROR_NEXT_CLUSTER_VALUE_OVER_RANGE
0x24	ERROR_NEXT_CLUSTER_VALUE_UNDER_RANGE
0x25	ERROR_NO_FREE_CLUSTERS
0x31	ERROR_FILE_NAME_FORBIDDEN_CHAR
0x32	ERROR_FILE_NAME_DIR_NAME_OVER_8
0x33	ERROR_FILE_NAME_DIR_EXTENSION_OVER_3
0x34	ERROR_FILE_NAME_FIRST_CHAR_ZERO
0x35	ERROR_MEDIA_FULL
0x40	DIR_ENT_FOUND
0x41	DIR_ENT_NOT_FOUND
0x42	ERROR_FOLDER_IS_CORRUPTED_FIRST_CLUSTER
0x43	ERROR_FOLDER_IS_CORRUPTED_DIR_DOT_NOT_FOUND
0x44	ERROR_FOLDER_IS_CORRUPTED_DIR_DOTDOT_NOT_FOUND
0x45	ERROR_ROOT_DIRECTORY_IS_FULL
0x46	ERROR_OPEN_FOLDER_FILE
0x47	ERROR_WRTIE_TO_READ_MODE_FILE
0x48	ERROR_SEEK_REQUIER_READ_MODE
0x49	ERROR_INVALID_SEEK_POINTER
0x4A	ERROR_FOLDER_NOT_EMPTY
0x4B	ERROR_IS_NOT_FOLDER
0x4C	ERROR_READ_MODE_REQUIRED
0x4D	ERROR_END_OF_DIR_LIST
0x4E	ERROR_FILE_PARAMETERS
0x4F	ERROR_INVALID_HANDLE
0x61	ERROR_COMMANDER_BAD_COMMAND
0x62	ERROR_COMMANDER_STR_LEN_TOO_LONG
0x63	ERROR_COMMANDER_NAME_NOT_VALID
0x64	ERROR_COMMANDER_NUMBER_INVALID
0x65	ERROR_COMMANDER_WRITE_PARTIAL_FAILURE
0x66	ERROR_COMMANDER_UNKNOWN_MEDIA_LETTER
0x67	ERROR_COMMANDER_FAILED_TO_OPEN_MEDIA
0x68	ERROR_COMMANDER_INCORRECT_CMD_PARAMETER
0xFD	ERROR_COMMANDER_UNKNOWN_ERROR