FAT16/32 File System Driver for AVR

This library contains not only the basic read and write functions, but many other functions to help getting an embedded AVR FAT16/32 application up and running using BASCOM-AVR. It is able to mount a drives first partition whether it has a partition table or not, and access any directory.
The user of the library does not need to be concerned about FAT details, buffer status, and the type of drive, as it is all taken care of by the libraries.
Only short file names (8.3) are supported.
The mass storage device drivers are separate libraries, and currently have drivers for an ATA hard disk or compatible connected to general I/O ports and one for when it is memory mapped in XRAM space. The ATA hard disk library will also put the drive to sleep after a pre-determined time of no disk access.
A hardware driver for a MMC / SD card connected to the hardware SPI has also been written, as is one for a compact flash card in pc card memory mode
The real time clock may use either the hardware based DS1307 or use the software based BASCOM-AVR internal soft clock

Subs & Functions

Initdrive()
Initialise the AVR ports and or hardware SPI and initialise the mass storage device.
Initfat()
Initialise the FAT16/32 file system. Initfat will determine whether it has or has not got a partition table and determine the fat type.
If the fat type is FAT32, then fsinfo is read, to determine current count of free clusters, and the next free cluster. The next free cluster is checked to see if it is really free, and the real next free cluster is written back to fsinfo.
If the fat type is FAT16, then the fat table is scanned to determine current count of free clusters, and the next free cluster.
Flushdir2disk()
Flushes a modified file buffer to disk. This must be called after any of the “Write_dir_*” are used.
Drivemodel()as String
Returns a string containing the model of the drive (or card).
Next_file(direntry As Word , Byval Updown As Byte)
Finds the next of previous valid file in the current directory. If updown bit 7 is clear, then the next file number is returned via dir entry. If updown bit 7 is set, then the previous file number is returned via dir entry.
When looking up, and the last dir entry is reached, it will wrap and start looking for the next valid file from the start.
When looking down, and the first dir entry is reach, it will not wrap to the last.
The constant “Attrib_valid” determines a valid file
Next_file_match(file_mask As String , Byval Attrib_mask As Byte , Direntry As Word , Byval Updown As Byte)as Word
Returns the next or previous file number that matches the file mask OR matches the attribute mask, starting from dir entry. The file mask may contain wild cards, eg * and ?.
Find_file(dir_name_ext As String) As Word
Returns the file number of a file name that exactly matches. If file name does not have a dot at in character position 9, then it is implied. All characters are converted to upper case, and illegal characters stripped before attempting to find the file. Eg. “a^loNg*illeGal%file” will find the file “ALONGILL.EGA”, if it exists
Returns &HFFFF, if no file matches.
Create_file(name As String , Byval Attribute As Byte) As Word
Creates a file or directory with a short file name and the attribute, and returns its file number. If file name does not have a dot at in character position 9, then it is implied. All characters are converted to upper case, and illegal characters stripped before creating the file. Eg. “a^loNg*illeGal%file” will create the file or dir “ALONGILL.EGA”. If the attribute has the directory entry bit set, then a directory will be created. The current system time and date are used for the all the time and date fields.

Note that there is no check to see if the file already exists. Use Find_file, to see if the file already exists.

Delete_file(direntry As Word)
Deletes a file. Cannot delete a directory. If attempting to delete a directory, then the global variable “errorcode” will contain the constant “Del_dir_wrng”.

Open_file(direntry As Word)
Opens the file or directory. Will return the first valid file number, if a directory is opened.

Read_dir_name(direntry As Word) As String
Returns the name of the file (without it’s extension”.

Read_dir_ext(direntry As Word) As String
Returns the extension of the file.

Read_dir_name_ext(direntry As Word) As String
Returns the name and extension (if it has one) of the file, with the implied dot.

Read_dir_firstcluster(direntry As Word) As Long
Returns the first cluster of a file. (Internal routine, not intended for normal use)

Read_dir_filesize(direntry As Word) As Long
Returns the size of the file.

Read_dir_attr(direntry As Word) As Byte
Returns the attribute of the file.

Read_dir_wrttime(direntry As Word) As String
Returns the write time of the file using the BASCOM-AVR “Config Date” parameters.

Read_dir_wrtdate(direntry As Word) As String
Returns the write date of the file using the BASCOM-AVR “Config Date” parameters.

Read_dir_crttime(direntry As Word) As String
Returns the create time of the file using the BASCOM-AVR “Config Date” parameters.

Read_dir_crtdate(direntry As Word) As String
Returns the create date of the file using the BASCOM-AVR “Config Date” parameters.

Read_dir_lstaccdate(direntry As Word) As String
Returns the last access date of the file using the BASCOM-AVR “Config Date” parameters.

Read_dir_filesizestr(direntry As Word) As String
Returns a string rounded to the nearest kilo, Mega, or Gig of the file size. If the file is actually a directory, then the string “<DIR>” is returned.

Write_dir_name(dir_name As String , Direntry As Word)
Writes the name of the file. All characters are converted to upper case, and illegal characters stripped before writing the name.

Write_dir_ext(dir_ext As String , Direntry As Word)
Writes the extension of the file. All characters are converted to upper case, and illegal characters stripped before writing the extension.

Write_dir_name_ext(name_dot_ext As String , Direntry As Word)
Writes the name and extension of the file. All characters are converted to upper case, and illegal characters stripped before writing the name and extension. If file name does not have a dot at in character position 9, then it is implied.

Write_dir_filesize(direntry As Word)
Writes the current file offset to the file size field.

Write_dir_attr(attribute As Byte , Direntry As Word)
Writes the attribute to the file.

Write_dir_wrtdate(direntry As Word)
Writes the current system date to the write date field.

Write_dir_crtdate(direntry As Word)
Writes the current system date to the create date field.

Write_dir_lstaccdate(direntry As Word)
Writes the current system date to the last access date field.

Write_dir_wrttime(direntry As Word)
Writes the current system time to the write time field.

Write_dir_crttime(direntry As Word)
Writes the current system time to the create time field.

Write_dir_firstcluster(firstcluster As Long , Direntry As Word)
Writes the first cluster to the first cluster field. (Internal routine, not intended for normal use)

Random_access()

Adjusts file pointers to the current file offset. File offset can be set to any value from 0 to file size.

Read_random() As Byte

Returns a byte at the current file offset. File offset can be set to any value from 0 to file size. There is no assumption as to what file offset was in the previous call, or will be in the next call.

Read_sequential() As Byte

Returns a byte at the current file offset. File offset will be incremented by one, and then checked against file size. If the current file offset equals file size, then the end of file flag in File status is set. It is assumed that the next call will be file offset + 1.

Read_16()as Word

Returns a word at the current file offset. Calls Read_sequential() twice.

Read_32()as Long

Returns a long at the current file offset. Calls Read_sequential() four times.

Find_freefile() As Word

Returns the file number of the first free directory entry. A free directory entry, may have been unused or belonging to a deleted file.

Close_file

Closes a file.

If the file was only read, then just the last access date field is updated.

If the file was written to sequentially, then the file buffer, is flushed back to disk, the FAT is updated and flushed back to disk, the file size is updated, and the write date, write time, and last access date fields are updated.

If the file was written to randomly, then the file buffer, is flushed back to disk, and the write date, write time, and last access date fields are updated.

Append_file

Append file re-opens a closed file by removing the end of cluster chain in its FAT, and then random access to the last byte entry. It is then possible to write sequentially to the file.

Write_sequential(file_byte As Byte)

Writes a byte at the current file offset. File offset will be incremented by one. It is assumed that the next call will be file offset + 1.

Write_16(file_word As Word)

Writes a word at the current file offset. Calls “Write_sequential(file_byte As Byte)” twice.

Write_32(file_long As Word)

Writes a long at the current file offset. Calls “Write_sequential(file_byte As Byte)” four times.

Write_random(file_byte As Byte)

Writes a byte at the current file offset. File offset can be set to any value from 0 to file size. There is no assumption as to what file offset was in the previous call, or will be in the next call.

Open_path(path As String , Remain_path As String , Direntry As Word)

Opens a path, and returns a new file number via dir entry.

If part of the path does not exist, then remain path returns the portion of the path that was unable to be opened. The global variable “Errorcode” will also contain the constant Open_path_wrng, as a warning, so that the remaining path can be created if needed.

If a file was found rather than a directory, then Errorcode will contain Open_path_fnd_file_wrng.

If a directory was found rather than a file, then Errorcode will contain Open_path_fnd_dir_wrng.

If the whole path does exist, then remain path will return with an empty string.

It is possible to pass the same variable to path and remain path.

If the first character is \ (backslash) then the root directory is implied.

If the character after the name = \ then a directory is implied.

Eg.

\foo\ will open the, root\FOO directory.

\foo\foo.bar will open the, root\FOO\FOO.BAR file.

' FOO.BAR will open the, current directory FOO.BAR file.

\ will open the, root directory.

Create_path(path As String) As Word

Creates a whole path with a short file names and returns its file number.

The current system time and date are used for the all the time and date fields.

If a file was created, then global variable “Errorcode” will contain the constant Create_path_wrng as a warning, that the file must, at some stage be closed.
Note that there is no check to see if the path already exists. Use Open_path, to see if the path already exists, and only create the remaining path.
Eg.
\foo\ will create the, root\FOO directory.
\foo\foo.bar will create the, root\FOO\FOO.BAR file.
' FOO.BAR will create the, current directory FOO.BAR file.

Global Variables

ATA Driver with rotating media
Ticktock As Byte. System clock, goes up by 1 every second, wraps at 255 (4.25min)
Timetosleep As Byte. Contains the system time that the drive is due to sleep
ATA Driver without rotating media
None
SD / MMC driver
None.
FAT Driver
Plbabegin As Long. The LBA sector where the partition beings.
Bpb_secperclus As Byte. Bios Parameter Block Sectors per Cluster.
Bpb_resvdseccnt As Word. Bios Parameter Block Reserved Sector Count.
Bpb_numfats As Byte. Bios Parameter Block count of FAT data structures on vol. Usually 2.
Bpb_totsec As Long. Bios Parameter Block total count of sectors on the vol.
Bpb_fatz As Long. Bios Parameter Block Size (per one FAT table).
Bpb_rootclus As Long. Bios Parameter Block Root Cluster (FAT32 only). Usually 2.
Bpb_fsinfo As Long. Bios Parameter Block File System Info sector (FAT32 only.
Cntofclusters As Long. Count of (valid) Clusters on volume.
Dirlbabegin As Long. Dir LBA Begin (from start of disk, not volume).
Root_dir_sectors As Byte. Count of sectors occupied by a FAT16 root directory.
Bytesperclus As Word. Bytes Per Cluster
Fatlbabegin As Long. FAT LBA Begin (1st fat) (from start of disk, not volume).
Free_count As Long. Last known free cluster count.
Nxt_free As Long. Last known first free cluster
Fatloadedstart As Long. Contains the first fat sector loaded in the fat buffer.
Filenumofit As Byte. The number of sector iterations to fill the file buffer.
Fatstatus As Byte. Current fat status.
Errorcode As Byte. Error codes, also used by mass storage driver.
Dircluster As Long. The current dir cluster.
Dirstatus As Byte. Bit 7 = End Of File & bits 0 to 6 = the current count of sector iterations.
Dirclusterindex As Word The current dir cluster index.
Dirfirstcluster As Long. The first dir cluster of the dir.
Diroffset As Long. Directory byte counter.
Filecluster As Long. The current file cluster.
Filestatus As Byte. Bit 7 = End Of File & bits 0 to 6 = the current count of sector iterations.
Fileclusterindex As Word. The current file cluster index.
Filefirstcluster As Long. The first file cluster of the file.
Fileoffset As Long. File byte counter.
Filesize As Long. The size of the file.
Fileparentfilenumber As Word. The parent file file number.
Filebuffer(filebuffersize) As Byte. The file buffer, shared between file and directory.
Fatbuffer(fatbuffersize) As Byte. The FAT buffer.

User Definable Constants

ATA Driver
Drive_is_master 1 = strapped for Master, 0 = strapped for slave
Media_is_rotating 1 = media rotating, 0 = media is semiconductor (eg CF)
Maxtime. ' Max Time for drive to complete task (in seconds). (Only if media_is_rotating = 1)

Sleeptime. Put drive to sleep after Sleeptime sec of no access. (Only if media_is_rotating = 1)
The next 3 are only valid for ataonbus.lib
Atacs0. Chip Select 0 of ATA
Atacs1 Chip Select 1 of ATA
Atahi8 High 8 bits latch
The next 13 are only valid for ataonport.lib
Lo8port. The port that the low 8 bytes of the ATA are connected to.
Lo8pin. The pin that the low 8 bytes of the ATA are connected to.
Lo8dd. The data direction that the low 8 bytes of the ATA are connected to.
Hi8port. The port that the high8 bytes of the ATA are connected to.
Hi8pin. The pin that the high 8 bytes of the ATA are connected to.
Hi8dd. The data direction that the high 8 bytes of the ATA are connected to.
Contport The port that the control bits (A0-A2, CS0, CS1, WR, RD) are connected to.
Contdd. The data direction that the control bits are connected to.
A0bit. The bit of the contport where A0 is connected to.
' NOTE A1 must be connected to A0+1, and A2 must be connected to A0+2
Atawr The bit of the contport where write is connected to.
Atard The bit of the contport where read is connected to.
Atacs0 The bit of the contport where chip select 0 is connected to
Atacs1 The bit of the contport where chip select 1 is connected to.
Since chip select 1 is only used to issue a software reset during initialisation, if this feature is not required, make Atacs1 >= 8. If chip select 1 is not used, it must be pulled high.

SD / MMC Driver

Spiport The port on the AVR where the hardware SPI is located.
Spidd The data direction on the AVR where the hardware SPI is located.
Sdmmccs The pin of the port on the AVR where the Slave Select is located.

Compact Flash Driver

Lo8port. The port that the 8 bit data is connected to.
Lo8pin. The pin that the 8 bit data is connected to.
Lo8dd. The data direction that the 8 bit data is connected to.
Contport The port that the control bits (A0-A2, CE1 WE, OE) are connected to.
Contdd. The data direction that the control bits are connected to.
A0bit. The bit of the contport where A0 is connected to.
' NOTE A1 must be connected to A0+1, and A2 must be connected to A0+2
Cfwe The bit of the contport where write enable is connected to.
Cfoe The bit of the contport where output enable is connected to.
Cfce1 The bit of the contport where chip enable 1 is connected to

FAT 16 / FAT 32 driver

Fatbuffersectorsize The size of the FAT buffer in sectors.
Filebuffersectorsize The size of the file buffer in sectors. Must be a base 2 number, that is less than or equal to sectors per cluster. Make it 1 for MAX compatibility.
Attrib_valid. The file attribute that is considered to be a valid file for the routine Next_file.
Kilo. The value that is used by the routine Read_dir_filesizestr.

Code Snips

Creating a file

```
Input "filename " , filename                        'Get the user input
Filenumber = Find_file(filename)       'See if the file already exists
If Filenumber <> &HFFFF Then
   Print filename ; " exists"
   Call Delete_file(Filenumber)                  'delete the file first
End If
Filenumber = Create_file(filename , Attrib_archive)  'create the file
Call Open_file(filenumber)                          'open the created file
While Fileoffset < Max_size_of_file          'While less than Max size
   File_byte = Waitkey()              'Get a byte from the serial port
   If File_byte = 27 Then Exit While         'Bug out if CTLZ received
   Call Write_sequential(file_byte)'fill the file with received bytes
Wend
Call Close_file()                             'Close. Final flush to disk
```

Creating a path

```
Input "path " , Path                          'Get the user input
Call Open_path(path , Path , Filenumber)  'See if path already exists
If Errorcode = Open_path_wrng Then
   Print "creating " ; Path   'create the portion that does not exist
   Filenumber = Create_path(path)
End If
```

Print a text file

```
Call Open_file(filenumber)                              'open the file
While Filestatus.endoffileflag = 0          'While NOT at end of file
   Filebyte = Read_sequential()                           'Read a byte
   Print Chr(filebyte);                                 'Print the byte
Wend
Call Close_file()                          'Close the file when finished
```

Fatal Errors

All fatal errors are handled by simply jumping to a routine called “Errorhandle”, which must exist in the main application. The global variable “Errorcode” will contain a value that can be used to determine the cause of the fatal error. Values ranging from 1 to 20 are used for fatal mass storage driver errors. Values ranging from 21 to 40 are used for fatal fat16/32 driver errors.

Fat 16/32 error codes

Sig_chk_err = 21 Failed Master Boot Record, or Bios Parameter Block, or FSI signature check
Fatis12_err = 22 Fat12 is not supported
Nxt_clus_eoc_err = 23 Attempt to look for the next cluster while at end of cluster chain
Fill_buff_eof_err = 24 Attempt to fill buffer while at end of file
Fat_chain_err = 25 Next cluster contains an un allocated cluster.
Disk_full_err = 26 Disk is full
Flush_empty_buff_err = 27 Attempt to flush an empty buffer in random mode
Append_no_eoc_err = 28 Append did not find an end of cluster chain
Bytes_per_sec_err = 29 Bios Parameter Block Bytes per Sector not equal to 512
No_fat_loaded_err = 30 Attempt to force a fresh fat buffer, while fat buffer is modified
Open_file_err = 31 Attempt to open a volume ID or a long file name

ATA error codes

Time_out_err = 1 Time Out
Word_xfer_err = 2 Transferred words not equal to expected
Ata_status_err = 3 Status register has error bit set

SD / MMC error codes

Time_out_err = 1 Time Out
Ata_status_err = 3 Response indicates an error

Compact Flash error codes

Time_out_err = 1 Time Out
Word_xfer_err = 2 Transferred words not equal to expected
Ata_status_err = 3 Status register has error bit set