FAT 16/32 File System Driver for ATMEL AVR

Version 1.00, April 26, 2004
Developed by Angelo Bannack and Giordano Bruno Wolaniuk
{angelo@earmazem.com.br}, {giordano@earmazem.com.br}

This document describes the FAT16/32 File System Driver for ATMEL AVR. This file system driver was written because our needs to read and write a hard drive using a microcontroller. We look at Internet but we don't found a good code to do this, specifically in ATMEL AVR microcontroller series. The mostly codes found were written to mp3 players, and only read data (don't write) from FAT file system and mix mp3 specific functions with printf functions. We found a few companies that have some FAT codes to microcontrollers but with expensive prices. So we decide to do our own code and share it to all community. It's a real generic library. You can use this to do an mp3 player, or a data logger, or anything else your mind tells you. All the code was written following the Microsoft Specification in their document "Microsoft Extensible Firmware Initiative FAT32 File System Specification[1]" and there are some limitation because the nature of microcontrollers and their limitations about memory and speed. Use this code at your own risk. If you find any problems in this code, feel free to tell us. We reminder you that it's distributed under the GNU Public License, so you can use and distribute it to anyone, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty.

[1] See the original document at http://www.microsoft.com/hwdev/download/hardware/fatgen103.doc

Contents

Notational Conventions in this Document

Numbers that have the characters “0x” at the beginning of them are hexadecimal (base 16) numbers.

Any numbers that do not have the characters “0x” at the beginning are decimal (base 10) numbers.

The code fragments in this document are written in the ‘C’ programming language.

General Comments

All of the FAT file systems were originally developed for the IBM PC machine architecture. The importance of this is that FAT file system on disk data structure is all “little endian.” If we look at one 32-bit FAT entry stored on disk as a series of four 8-bit bytes—the first being byte[0] and the last being byte[4]—here is where the 32 bits numbered 00 through 31 are (00 being the least significant bit):

```
byte[3]        3 3 2 2 2 2 2 2
               1 0 9 8 7 6 5 4

byte[2]        2 2 2 2 1 1 1 1
               3 2 1 0 9 8 7 6

byte[1]        1 1 1 1 1 1 0 0
               5 4 3 2 1 0 9 8

byte[0]        0 0 0 0 0 0 0 0
                7 6 5 4 3 2 1 0
```

This is important if your machine is a “big endian” machine, because you will have to translate between big and little endian as you move data to and from the disk. Note that ATMEL AVR series are 8 bits microcontrollers so the compiler is responsible for use little or big endian. The WinAVR is little endian by default. If you want to use another kind of compiler, make sure that it’s compiling like a little endian mode.

Requirements

To use this library you will have to use an ATMEL[2] AVR microcontroller with at least:
- 15 Kbytes FLASH Memory;
- 1.2 Kbytes RAM Memory (2 512Kbytes buffer + other variables);
- 3 eight bits PORTS to ATA interface (only 2 exclusive PINs – IDE_RD and IDE_WR);

The code was written in WinAVR C and compiled using the WinAVR[3] 20040404.

In our tests we use an ATMega128[4] connected directly to the ATA dispositive.

If you want to use this library only for read files in the FAT file system, like in an mp3 player, for instance, you can remove the entire write and create files routines. You can do this defining a constant ATA_READ_ONLY in *ataconf.h*, like the code below:

```
#define ATA_READ_ONLY
```

Then all the routines used to write files and create new files and directories will be removed from the source code in compilation time. But, note that the RAM memory usage will be no reduced significantly.

The most significant part of the RAM memory is used to 2 (two) buffers with 512 bytes each, for keep data sector and fat sector caches. The fat sector cache could share the data sector memory but with time restrictions and with a lot of modifications in the software. We don't recommend this modification.

[2] ATMEL: www.atmel.com
[3] WinAVR: http://winavr.sourceforge.net
[4] ATMega128: See the specifications at http://www.atmel.com/dyn/products/product_card.asp?part_id=2018

Specifications

The complete code has 9 files:

1. *ata.c*	IDE-ATA interface driver for hard disks code
2. *ata.h*	ATA specification include file
3. *ataconf.h*	Configuration ATA file, memory sector buffer and PIN descriptions
4. *fat.c*	FAT16/32 file system driver for ATMEL AVR code
5. *fat.h*	FAT specification include file
6. *fatconf.h*	Configuration FAT file, memory FAT cache buffer
7. *fattime.c*	FAT time driver code, time functions
8. *fattime.h*	FATTIME specification include file
9. *global.h*	AVRlib project global include file

Ata Files

The *ata.c*, *ata.h* and *ataconf.h* files are responsible to interface the ATA dispositive.

Ataconf.h

In *ataconf.h* are defined the sector buffer with 512 bytes to keep the sector data read from the ATA dispositive. This buffer can be addressed to an external memory buffer, changing his address.

All the pins are defined in this file, and you can change it according to your hardware needs

```
#define DDR_DATAL         DDRC         // Define the Direction PORT to DATA High
#define DDR_DATAH         DDRB         // Define the Direction PORT to DATA Low
#define PORT_DATAL        PORTC        // Define the PORT to DATA High
#define PORT_DATAH        PORTB        // Define the PORT to DATA Low
#define PIN_DATAL         PINC         // Define the PIN to DATA High
#define PIN_DATAH         PINB         // Define the PIN to DATA Low

#define PORT_ADDR         PORTA        // Define the PORT to Address
#define DDR_ADDR          DDRA         // Define the PORT to Address

#define PORT_IDE_RD       PORTA        // Define the PORT to IDE_RD
#define PIN_IDE_RD        7            // Define the PIN number to IDE_RD

#define PORT_IDE_WR       PORTA        // Define the PORT to IDE_WR
#define PIN_IDE_WR        6            // Define the PIN number to IDE_WR
```

Don’t change the ATA Registers, only if you know the correspondent pin address in PORT_ADDR.

Ata.c and *Ata.h*

These files define and implement the ATA read and write routines. The main functions that are used by *fat.c* file are:

```
unsigned char ataReadSectors
       (unsigned char Drive,
        unsigned long lba,
        unsigned char *Buffer,
        unsigned long *SectorInCache);
```

Read one sector (512 bytes) from the ATA dispositive. It's necessary to inform the Drive number, the lba address to be read, the buffer address to keep the readed data, and the variable address that informs the sector address in buffer cache

```
unsigned char ataWriteSectors
       (unsigned char Drive,
        unsigned long lba,
        unsigned char *Buffer);
```

Write one sector (512 bytes) to the ATA dispositive. It's necessary to inform the Drive number, the lba address to be write, the buffer address that have the data to be written.

The routines below can be used by your source code:

```
void ataInit (void);
```

Gives a software reset into the ATA dispositive and gets all the hardware parameters from it. It's necessary to call this routine in the begging of main software.

```
unsigned long ataGetSizeInSectors
       (void);
```

Returns the ATA dispositive number of sectors

```
unsigned long ataGetSize (void);
```

Returns the ATA dispositive size in bytes

```
char *ataGetModel (void);
```

Returns the ATA dispositive model string

```
void ataSetDrivePowerMode
       (unsigned char DriveNo,
        unsigned char mode,
        unsigned char timeout);
```

Sets the Power Mode to the drive. It's necessary to inform the drive number, the mode, and the timeout. To the hardware.
The possible modes are:
ATA_DISKMODE_SPINDOWN
ATA_DISKMODE_SPINUP
ATA_DISKMODE_SETTIMEOUT
ATA_DISKMODE_SLEEP

All the other functions are used internally only and don't have to be called.

Comments

An important thing here is that the ATA access functions were optimized to an ATMega128 with a 16MHz source clock and reading an ATA66 dispositive. So there are a few "NOPs" that were used like a delay. If you use a slower source clock, this won't be a problem, but if the source clock were bigger than this, so you may need to put extras "NOPs" in ataReadDataBuffer, ataWriteDataBuffer, ataReadByte and ataWriteByte functions defined in *ata.c* file. See the time diagram in the ATA specification[5] to a correct access time.

[5] ATA 2 Specification: See the official document at http://www.t13.org/project/d0948r4c.pdf

Fat Files

The *fat.c*, *fat.h*, *fatconf.h*, *fattime.c* and *fattime.h* files are responsible to the FAT file system driver.

Fatconf.h

In *fatconf.h* are defined the sector buffer with 512 bytes to keep the sector data read from the ATA dispositive. This buffer is defined like the buffer address defined in *ataconf.h*. In this file is defined too the fat sector buffer with 512 bytes to keep the sector fat read from the ATA dispositive. This buffer can be addressed to an external memory buffer, changing his address.

Fat.c and *Fat.h*

These files define and implement the FAT file system driver.
There is an important structure defined in *fat.h* that is useful in a dir routine, it is the *direntry* struct.

```
struct direntry {
      unsigned char       deName[8];                // filename, blank filled
      #define             SLOT_EMPTY          0x00  // slot has never been used
      #define             SLOT_E5             0x05  // the real value is 0xe5
      #define             SLOT_DELETED        0xe5  // file in this slot deleted
      unsigned char       deExtension[3];           // extension, blank filled
      unsigned char       deAttributes;             // file attributes
      #define             ATTR_NORMAL         0x00  // normal file
      #define             ATTR_READONLY       0x01  // file is readonly
      #define             ATTR_HIDDEN         0x02  // file is hidden
      #define             ATTR_SYSTEM         0x04  // file is a system file
      #define             ATTR_VOLUME         0x08  // entry is a volume label
      #define             ATTR_LONG_FILENAME  0x0f  // this is a long filename
      #define             ATTR_DIRECTORY      0x10  // entry is a directory name
      #define             ATTR_ARCHIVE        0x20  // file is new or modified
      unsigned char       deLowerCase;              // NT VFAT lower case flags
      #define             LCASE_BASE          0x08  // filename in lower case
      #define             LCASE_EXT           0x10  // extension in lower case
      unsigned char       deCHundredth;             // hundredth of seconds
      unsigned char       deCTime[2];               // create time
      unsigned char       deCDate[2];               // create date
      unsigned char       deADate[2];               // access date
      unsigned int        deHighClust;              // high bytes of cluster
      unsigned char       deMTime[2];               // last update time
      unsigned char       deMDate[2];               // last update date
      unsigned int        deStartCluster;           // starting cluster of file
      unsigned long       deFileSize;               // size of file in bytes
};
```

You can do a loop reading the entire directory only checking the deName[0] to see if it is equal to SLOT_EMPTY. The SLOT_EMPTY marks the end of the directory. You can use this structure to print out all the needed information about the files.

The date stamp is a 16-bit field that is basically a date relative to the MS-DOS epoch of 01/01/1980. here is the format (bit0 is the LSB of the 16-bit word, bit 15 is the MSB of the 16-bit word):

> Bits 0–4: Day of month, valid value range 1-31 inclusive.
> Bits 5–8: Month of year, 1 = January, valid value range 1–12 inclusive.
> Bits 9–15: Count of years from 1980, valid value range 0–127 inclusive (1980–2107).

The time stamp is a 16-bit field that has a granularity of 2 seconds. Here is the format (bit 0 is the LSB of the 16-bit word, bit 15 is the MSB of the 16-bit word).

> Bits 0–4: 2-second count, valid value range 0–29 inclusive (0 – 58 seconds).
> Bits 5–10: Minutes, valid value range 0–59 inclusive.
> Bits 11–15: Hours, valid value range 0–23 inclusive.

The current directory is controlled by an internal variable *currentDirCluster*. When you call the *fatInit* routine, this variable is started with the cluster address of the root directory in FAT. If you want to change the current directory, do a *fatCddir* using a directory name in the parameter field. The *currentDirCluster* variable will be changed to the cluster address of this directory. You can use the *fatGetCurDirCluster* to get the actual cluster address.

This struct is used too to define the TFILE struct:

```
typedef struct{
	struct direntry de;		// Information about the file opened
	unsigned int currentSector;	// Actual sector address in memory
	unsigned char *buffer;		// buffer pointer to memory (cache sector)
	unsigned long bytePointer;	// byte pointer to the actual byte
	unsigned char sectorHasChanged;	// TRUE if the sector has changed
}TFILE;
```

The TFILE struct is used to permit the read and write in files. When a file is opened with the fatFopen function, this struct is filled with the direntry information about the file. This struct keeps the current sector in memory; a pointer to the data sector in memory; a byte pointer to the actual character, used by fatFgetc and fatFputc functions; and a flag to tell the fClose function if the sector in memory has changed and needs to be writed in the FAT file system before the file be closed.

Another important struct, used to read partition information, is the *partrecord* struct.

```
// Partition Type used in the partition record
#define PART_TYPE_UNKNOWN        0x00
#define PART_TYPE_FAT12          0x01
#define PART_TYPE_XENIX          0x02
#define PART_TYPE_DOSFAT16       0x04
#define PART_TYPE_EXTDOS         0x05
#define PART_TYPE_FAT16          0x06
```

```
#define PART_TYPE_NTFS           0x07
#define PART_TYPE_FAT32          0x0B
#define PART_TYPE_FAT32LBA       0x0C
#define PART_TYPE_FAT16LBA       0x0E
#define PART_TYPE_EXTDOSLBA      0x0F
#define PART_TYPE_ONTRACK        0x33
#define PART_TYPE_NOVELL         0x40
#define PART_TYPE_PCIX           0x4B
#define PART_TYPE_PHOENIXSAVE    0xA0
#define PART_TYPE_CPM            0xDB
#define PART_TYPE_DBFS           0xE0
#define PART_TYPE_BBT            0xFF

struct partrecord // length 16 bytes
{
       unsigned char prIsActive;       // 0x80 indicates active partition
       unsigned char prStartHead;      // starting head for partition
       unsigned int  prStartCylSect;   // starting cylinder and sector
       unsigned char prPartType;       // partition type (see above)
       unsigned char prEndHead;        // ending head for this partition
       unsigned int  prEndCylSect;     // ending cylinder and sector
       unsigned long prStartLBA;       // first LBA sector for this partition
       unsigned long prSize;           // size of this partition
};
```

This struct is filled in fatInit routine, and show important information about the FAT file system. The most important field in this struct is the prPartType, in wich we can discovery if the ATA dispositive is formatted like a FAT16 or FAT32.

Functions

The main functions that can be used by your source code are:

```
unsigned char fatInit (void);
```

Get FAT info from ATA dispositive and initialize internal variables. It's necessary to call this routine on the begging of the main software.

```
unsigned int fatClusterSize (void);
```

Return the number of sectors in a disk cluster.

```
unsigned long fatGetFirstDirCluster
       (void);
```

Return the first dir entry cluster in FAT. This is useful when you want to do a fatDir in the root directory.

```
unsigned char *fatDir
       (unsigned long cluster,
        unsigned long offset);
```

Return the sector with direntries info starting in the parameter cluster, with offset sectors from the begining cluster sector. You can do a dir in the entire fat directory doing a loop in offset, started in 0, and with no restrictions to end. In each offset read gives a fatDir and if the direntry structure start with a SLOT_EMPTY mark, so the entire directory was returned.

```
struct direntry   *fatGetFileInfo
(struct direntry *rde, char
*shortName);
```

This routine is usefull if you know the filename and you want to get information about it. This routine will return a direntry struct filled with the file information, and will return NULL if the file was not found in the current directory.

```
char *fatGetVolLabel (void);
```

Return the FAT volume name read in fatInit

```
struct partrecord *fatGetPartInfo
       (void);
```

Return the partition information read in fatInit routine. This routine returns a *partrecord* struct.

```
unsigned int fatGetSecPerClust (void);
```

Return the number of Sectors per Cluster read in fatInit. This is only for information.

```
unsigned long fatGetFirstFATSector
       (void);
```

Return the sector address of the first FAT in the ATA dispositive. This is only for information.

```
unsigned long
fatGetFirstFAT2Sector (void);
```

Return the sector address of the second FAT in the ATA dispositive. This is only for information.

```
unsigned long fatGetFirstDataSector
       (void);
```

Return the sector address of the data field in FAT. This is only for information.

```
unsigned long fatGetNumClusters (void);
```

Return the total number of clusters in the ATA dispositive. This is only for information.

```
unsigned char fatCddir (char *path);
```

Change the current directory. Only one level path, For example, you can do fatCddir(“test”), but not fatCddir(“\test\test2”). Use two calls to this function in this case: fatCddir(“test”); fatCddir(“test2”);

```
TFILE *fatFopen (char *shortName);
```

Open a file to read and write. The File struct is filled and the Sector Buffer in memory is filled with the first sector of the file opened. Remember that to read one file you need to call this function first, and then use the TFILE struct returned in fatFgetc, fatFputc, fatFseek, fatFeof, fatFflush and fatFclose functions.

```
char fatFgetc (TFILE *fp);
```

Get the next character from file, and actualize the byte pointer in the TFILE struct.

```
unsigned int fatFseek
       (TFILE *fp,
        unsigned long offSet,
        unsigned char mode);
```

Find a byte position in the file and load the corresponded sector in the buffer memory. The possible modes are:

- SEEK_CUR: the offset is counted from the current position of the file pointer;
- SEEK_SET: the offset is counted from the beggining of the file;
- SEEK_END: the offset is counted from the end of file to back.

```
unsigned char fatFeof (TFILE *fp);
```

Return TRUE if the byte pointer points to the end of the file. This is useful to do a loop to read an entire file.

```
unsigned long fatGetCurDirCluster
       (void);
```

Return the current directory cluster number. This function is useful to use like a parameter to do a fatDir in the current directory. Example:

```
for (os=0; os++;)
       fatDir(fatGetCurDirCluster(),os);
```

```
unsigned char fatMkdir (char *path);
```

Create a new directory on the current directory. Only creates the directory, don’t change the current dir. This function will return TRUE if the directory was successfully created, and FALSE otherwise.

```
unsigned char fatRename
       (char *oldShortName,
        char *newShortName);
```

Change the name of a directory or file. Gives the old name and the new name of the file or directory. This function will return TRUE if the file or directory was successfully renamed, and FALSE otherwise.

```
unsigned char fatRemove
       (char *shortName);
```

Remove a file or directory. This function will look for a file in the current directory. It will return TRUE if the file or directory was successfully removed, and FALSE otherwise. If a directory has any files, it won’t be removed. Call fatRemoveAll in the current directory before this operation.

```
TFILE *fatFcreate (char *shortName);
```

Create a file and open it in the current directory. If the file already exist this function will return NULL. Otherwise this function will create the file in the current directory and will call the fatFopen function, returning a TFILE struct.

```
unsigned char fatFclose (TFILE *fp);
```

Write the current file to the FAT file system and refresh the file size, if necessary. You need to use this function before open another file and before a fatDir use.

```
unsigned char fatFflush (TFILE *fp);
```

Write the current sector file to hard disk, if necessary. This function is called by the fatFclose function, and you can call it in any time to force a buffer write to the disk.

```
unsigned char fatFputc
       (TFILE *fp, char c);
```

Write a character to the file. Return FALSE if the disk is full, and TRUE otherwise. Note that this function will only write the character in the internal buffer, and won’t write it to the disk. The write is executed only when the character belongs to a new sector, or in a call to fatFflush or fatFclose.

```
void fatRemoveAll (void);
```

Remove all files in the current directory. You can use this function before a fatRemove in a directory.

All the other functions are used internally only and don’t have to be called.

FatTime.c and *FatTime.h*

These files define the data and time structures and only one function, fatGetCurTime, that is used in a file create and close to update the current time in file. The programmer needs to change this function to get the clock from your hardware project. If you don’t have a clock, just set one unique time and data in the fatGetCurTime return, or let the default date an time.

The TTime struct is defined in *fattime.h*.

```
typedef struct
{
	unsigned char day;
	unsigned char month;
	unsigned int  year;
	unsigned char hour;
	unsigned char minutes;
	unsigned char seconds;
}TTime;
```

The date is divided in three fields: day, month and year.

Day: valid range from 1 to 31;
Month: valid range from 1 to 12;
Year: valid range from 0 to 65535.

Note that the year is an integer field that have to be filled with the correct year number. For example, if the year is 2004, this field have to be filled with the integer 2004, or 0x07D4 in hexadecimal format. The FAT file system only uses an eight bit field to the year, but the internal fat functions will correct to this form.

The time is divided in three fields too: hour, minutes and seconds.

Hour: valid range from 0 to 23;
Minutes: valid range from 0 to 59;
Seconds: valid range from 0 to 59;

In FAT file system, the seconds field is a five bit field, so they are divided by two before be written into the disk. The programmer cannot be worried about this.

Comments

This library and specially the *fatInit* function were prepared to be used by hard disks that have a MBR (Master Boot Record) in the first sector. Other types of media, like Memory Disks, Memory Sticks and Compact Flash don't have de MBR in the first sector. The first sector in this case is the Boot Sector, and contains information about the FAT file System, like size FAT, number of FATs, sectors per track and number of heads. To use this library to read a media like this, you need to change the *fatInit* function to ignore the MBR and read the BPB directly. To determine the FAT type, the library read a byte in the MBR. So you have to change this to use the number of cluster informations to determine the FAT type. See the "Microsoft Extensible Firmware Initiative FAT32 File System Specification[6]" to more details. These implementations will be done in a new version of this library.

[6] See the original document at http://www.microsoft.com/hwdev/download/hardware/fatgen103.doc

Other Files

The complete library includes a *global.h* file, that is used to define de TRUE and FALSE statements, and to configure the CPU clock speed (F_CPU).

The CPU clock speed is used in an internal *ata.c* delay function, to correct access the ATA dispositive. You have to define your CPU clock speed here to guarantee the correct ATA dispositive access. Read the ATA Comments chapter in page 7 to more details about this.

Limitations

Because the restrictions of memory and speed, this FAT code have a little limitations:

- Only short names can be used: long names access was not implemented because they are totally compatible with short names, and because the use in microcontrollers will significantly decrease the speed performance and will increase the FLASH requirements.

- Only one file opened at same time: because the RAM memory limitations we limitate to only one file opened. If you want to read another file, you will have to close the opened file first. If a file is opened when another has no closed yet, the sector data in RAM will be losted. The programmer needs to guarantee that two files were not opened in same time and that the close file function has been called before a file opened to write was no more necessary.

- No fatDir when a file is opened: when a file is opened, the programmer cannot call the fatDir routine, because the sector buffer is filled with the file information, and the dir routine will destroy the data file.

- Filenames in upper case mode: all the filenames are written in upper case mode in the FAT file system.

- Only master dispositive: only the master dispositive will be interfaced with this code. If you need to use the slave dispositive, change the DRIVE0 constant in *ata.h* file to 1

```
#define DRIVE0      1
```

The simultaneous use of master and slave ATA dispositives was note implemented.

- Low speed: the speed will be limited by your source clock. In our tests we could transfer up to 800Kbytes/s in an ATMega128 with a 16MHz crystal oscillator.

Performance

In our tests we could transfer up to 800 Kbytes/s using an ATMega128 with a 16MHz crystal oscllator, and compiled with WinAVR 20040404 with optimize mode ‘s’.

The speed could be increased if you interface the ATA dispositive in 16 bits mode, like a memory interface to the microcontroller, so the memory hardware inside the microcontroller will do all the transfers and the microcontroller will be free to do all the others operations. We don’t use it because it’s necessary to use some extra hardware latches to do this, and for simplification purposes, because we have price restrictions in our project and the speed was no problem. But it’s factible and simple to do it.

Hardware Schematic

In page 20 you can find the hardware schematic used to test this library. The hardware uses only an ATMega128 microcontroller and an IDE/ATA connector. It's included a serial port to tests purposes.

The table below shows the entire signal interface to an IDE/ATA dispositive and the ATMEL Pin used to interface it.

Table 1 – IDE/ATA Signal Interface

Description	ATMEL Pin	IDE Pin	Acronym
Reset	VDD	1	/IDE_RESET
Ground	GND	2	GND
Data bus bit 7	PC7	3	IDE_D7
Data bus bit 8	PB0	4	IDE_D8
Data bus bit 6	PC6	5	IDE_D6
Data bus bit 9	PB1	6	IDE_D9
Data bus bit 5	PC5	7	IDE_D5
Data bus bit 10	PB2	8	IDE_D10
Data bus bit 4	PC4	9	IDE_D4
Data bus bit 11	PB3	10	IDE_D11
Data bus bit 3	PC3	11	IDE_D3
Data bus bit 12	PB4	12	IDE_D12
Data bus bit 2	PC2	13	IDE_D2
Data bus bit 13	PB5	14	IDE_D13
Data bus bit 1	PC1	15	IDE_D1
Data bus bit 14	PB6	16	IDE_D14
Data bus bit 0	PC0	17	IDE_D0
Data bus bit 15	PB7	18	IDE_D15
Ground	GND	19	GND
(keypin)	NC	20	
DMA Request	VDD	21	VDD
Ground	GND	22	GND
I/O Write	PA6	23	IDE_WR
Ground	GND	24	GND
I/O Read	PA7	25	IDE_RD
Ground	GND	26	GND
I/O Ready	NC	27	
Spindle Sync or Cable Select	NC	28	
DMA Acknowledge	VDD	29	VDD
Ground	GND	30	GND
Interrupt Request	PA5	31	IDE_IRQ
16 Bit I/O	NC	32	
Device Address Bit 1	PA1	33	IDE_A1
Passed Diagnostics	NC	34	
Device Address Bit 0	PA0	35	IDE_A0
Device Address Bit 2	PA2	36	IDE_A2
Chip Select 0	PA4	37	IDE_CS0
Chip Select 1	PA3	38	IDE_CS1
Device Active or Slave Present	NC	39	
Ground	GND	40	GND

Below you can see the power interface to an IDE/ATA dispositive. This is the hard disk view.

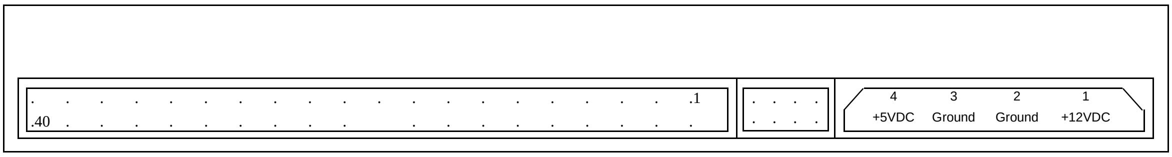
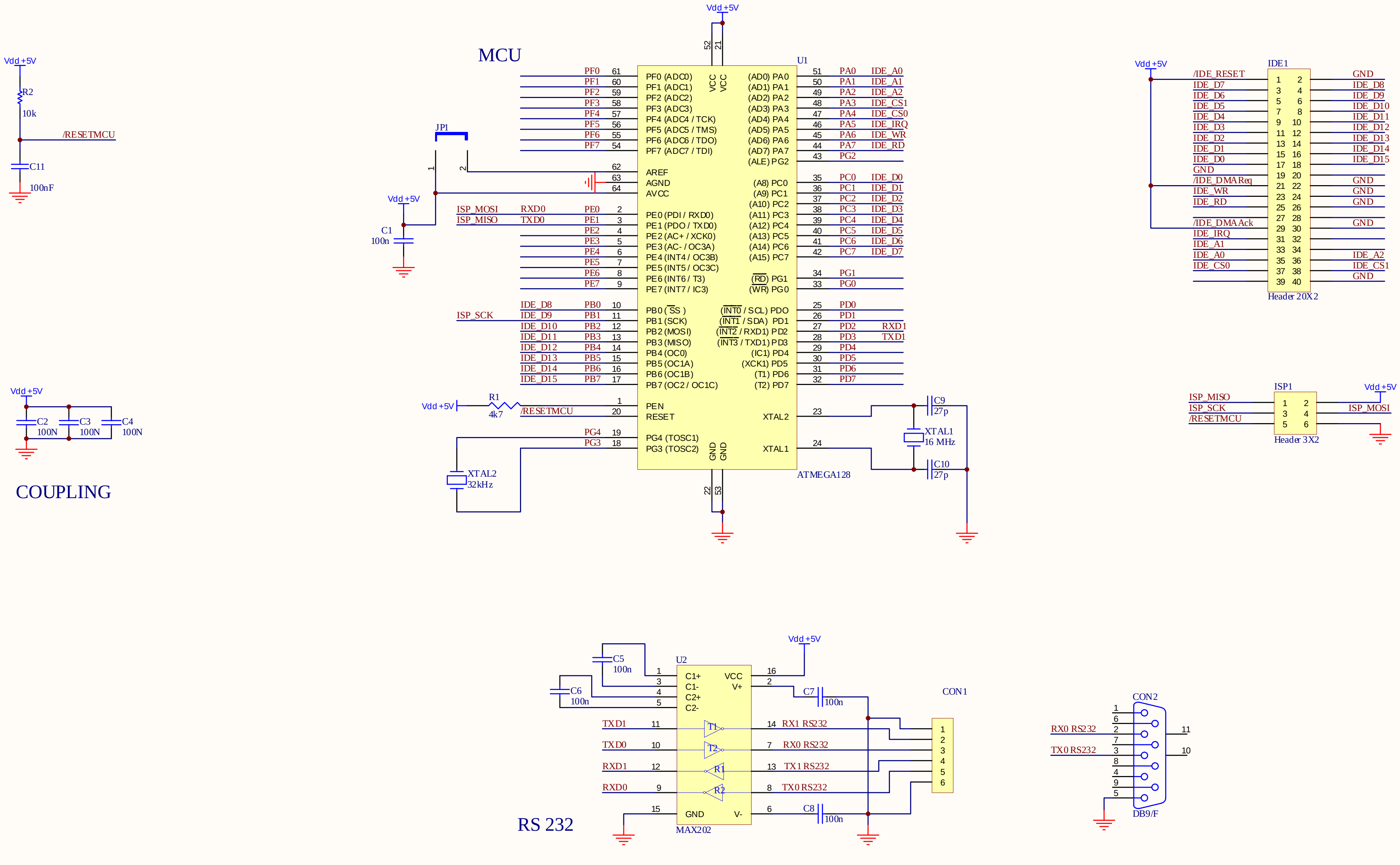


Figure 1 – IDE/ATA Interface

Table 2 – IDE/ATA Power Line

Power line designation	Pin Number	Default Color
+12 Volts	1	Yellow
Ground	2	Black
Ground	3	Black
+5 Volts	4	Red



www.earmazem.com

FAT 16/32 File System Driver for ATMEL AVR

Angelo Bannack, Giordano Bruno Wolaniuk

Size	FCSM No.	DWG No.	Rev
A3		1	a
Scale	26 april 2004	Sheet	

How to Test

In the package is included a *main.c* file. This file was written to be used with the hardware described in the above page. To use it, you need to connect the hardware to a PC serial port, and using your favorite serial program, configure the serial port to 115.200 bps, 8 bits, no parity, 1 stop bit. Connect a FAT16 or FAT32 formatted hard disk to the ATA/IDE interface. Write the file *main.hex*, included in the package, to the ATMega128, using a STK500 hardware test or another hex programmer like Pony Prog[7].

The program will execute the following procedure:

1. All the information about the hard disk and the FAT file system will be showed;
2. The root directory will be listed;
3. A new *DIR1* directory will be created;
4. The root directory will be listed;
5. The current directory will be changed to the new *DIR1* created;
6. The *DIR1* directory will be listed;
7. A *readme.txt* file with no content will be created;
8. The *DIR1* directory will be listed;
9. The *readme.txt* file will be filled with the message “Testing 123”;
10. The *readme.txt* file will be entirely read;
11. The *readme.txt* file will be renamed to *test.txt*;
12. The *DIR1* directory will be listed;
13. All the files in the *DIR1* directory will be erased;
14. The *DIR1* directory will be listed;
15. The current directory will be changed to the root directory;
16. The root directory will be listed;
17. The *DIR1* directory will be removed;
18. The root directory will be listed.

[7] Pony Prog: See more information about Pony Prog in http://www.lancos.com/prog.html

GNU GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the

rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM