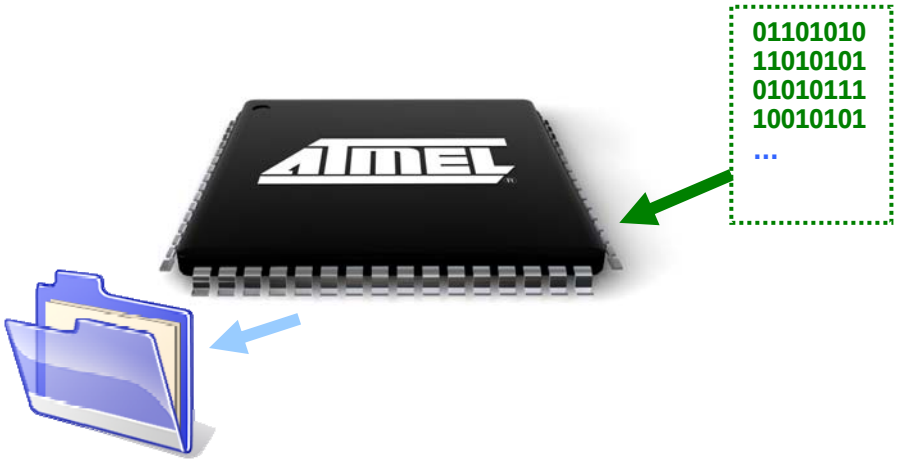
# AVR115: Data Logging with Atmel File System on ATmega32U4




AVR® Microcontrollers

Application Note

## 1 Introduction

Atmel® provides a File System management for AT90USBx and ATmegaxxUx parts. This File System allows performing data logging operation and this application note shows how to implement this feature using ATmega32U4 and EVK527 board. The "EVK527-series4-datalogging" is the firmware package related to AVR®115.

The reader should be familiar with AVR114 Application Note before reading this one.

Rev. 8202A-AVR-01/09

## 2 Hardware Requirements

The Data Logging example application requires the following hardware:

• AVR USB evaluation board ATEVK527 which includes:

- ATmega32U4
- DataFlash® (32Mbits)
- SD/MMC connector

• USB cable (Standard A to Mini B)

• PC running on Windows® (98SE, ME, 2000, XP), Linux® or MAC® OS with an USB 1.1 or 2.0 host

## 3 In-System programming and Device Firmware Upgrade

To program the device you can use one of the following methods:

• The JTAG interface using the JTAGICE mkII

• The SPI interface using the AVRISP mkII and JTAGICE mkII

• The USB interface thanks to the factory DFU bootloader and FLIP(1) software

• The parallel programming using the STK®500 or STK600

Please refer to FLIP(1) help content to see how to install the USB driver and program the device through the USB interface.

Note: 1. FLIP is software provided by Atmel to allow the user to program the Atmel devices through the USB interface (No external hardware required) thanks to the factory DFU bootloader.

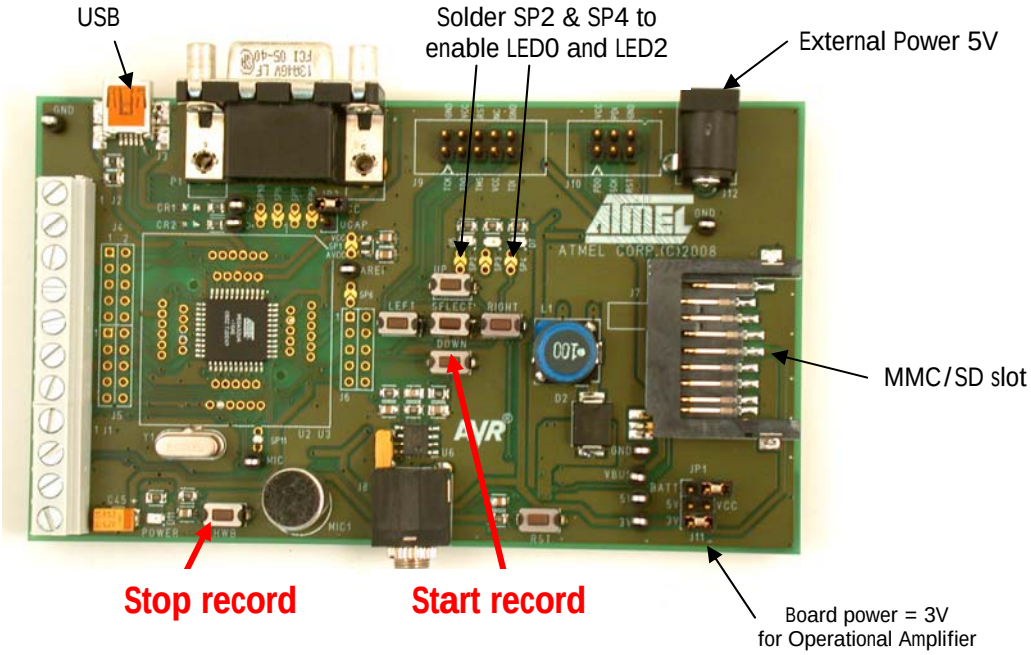
Note: **With JTAGICE MKII be careful with the box “erase before programming” in AVR Studio®. If checked, the DFU bootloader is deleted before to programming.**

# 4 Quick Start

Once your device is programmed with EVK527-series4-datalogging hex file, you can start the data logging demonstration:

1. Unplug the USB cable and plug a power cable (9 V)
2. Press key down to start record :
   - The log file is created either on the MMC/SD card if present, or on DataFlash memory
   - The LED**0 is turned on when the recording starts**
   - **The LED2 is turned on when an error occurs (Disk not present, Disk full, …)**
3. Press the key HWB or plug the USB cable to stop the recording.
4. Plug USB cable on your PC to run the U-Disk and to read the log file *"Disk:\log000\log000.bin"*

**Figure 4-1.** EVK521 Rev1




Note: This is the EVK527 default factory configuration except the SP2 & SP4 which must be sold.

**By default, the recorded data is only a digital number (16-bits) incremented and stored each 120 µs. One can change the record source via the software compilation options in *datalogging.c* file:**

```
#define LOG_ADCMIC_EXAMPLE    // From microphone ADC to a WAV file
#define LOG_ADCEXT_EXAMPLE    // From other ADC input to a BIN file
#define LOG_PIN_EXAMPLE       // Records pins states to a BIN file
#define LOG_ENUM_EXAMPLE      // Records counter value to a BIN file (Default)
```

# 5 Application

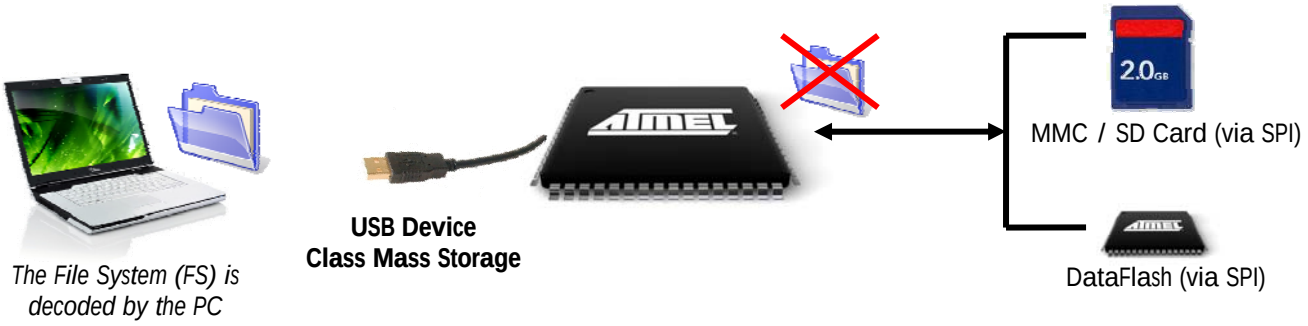
## 5.1 Behavior

The sample application provides two operating modes:

***Download mode:*** the user has to connect the kit to PC (removable disk "U-Disk") to be able to access to the log file written on the memories (DataFlash or SD/MMC card). In this mode, the embedded Atmel File System is not allowed to access to the memories.

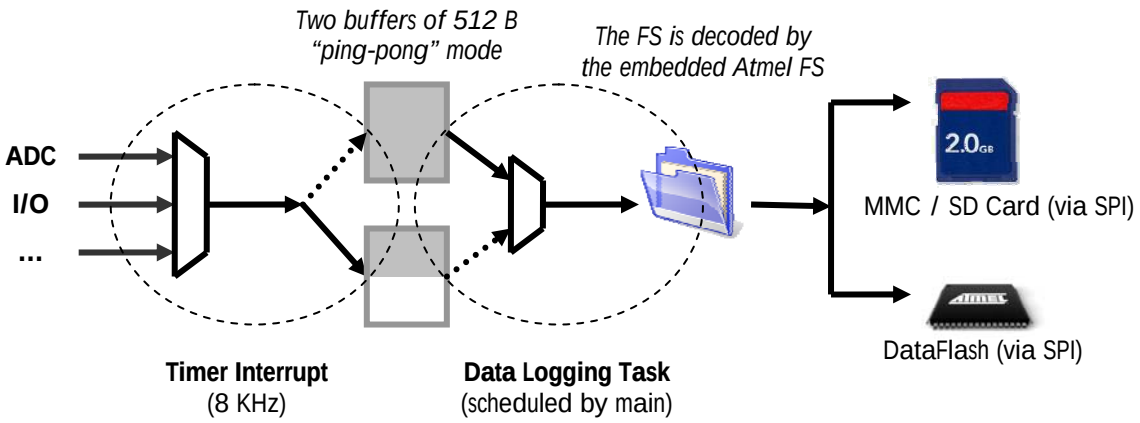
Note: To prevent data corruption, only one file system management may be active at a given time.

Figure 5-1. **Download mode (U-Disk)**




***Data logging mode:*** during this mode the data recording can be performed. The kit must be disconnected from the USB host and starting/stopping record actions are managed by EVK527 buttons (see Figure 4-1. **EVK521 Rev1**). When the data recording starts, a file is created in a memory. An 8 KHz interrupt timer samples the values from ADC or another source and writes them in a buffer. The full buffers are transferred in the file by the data logging task.

Figure 5-2. **Data Logging mode**



## 5.2 Firmware

This section explains only the File System management code and not the USB module. The following code samples are extracted from the ***data_logging.c*** file from "EVK527-series4-datalogging" package.

### 5.2.1 Enable/disable embedded FS

The File System module initialization and exit are managed by the *datalogging_task()*.

When the chip exits from USB Device mode, one can call "*nav_reset()*" to initialize the embedded Atmel File System.

When one wants to stop the embedded Atmel File System or when the USB Device mode starts, the "*nav_exit()*" must be called to flush cache information into the memories.

**Figure 5-3.** Example of datalogging_task() routine

```
void datalogging_task(void)
{
   // Change the data logging state
   if( Is_joy_down()             // If data logging started by user
   && (!Is_device_enumerated())) // and if USB Device mode stopped
   {
      if( !g_b_datalogging_running )
      {
         // Start data logging
         nav_reset();            //** Init File System
         g_b_datalogging_running = datalogging_start();
      }
   }
   if( Is_hwb()                  // If data logging stopped by user
   ||  Is_device_enumerated() )  // or if USB Device mode started
   {
      if( g_b_datalogging_running )
      {
         // Stop data logging
         g_b_datalogging_running = FALSE;
         datalogging_stop();
         nav_exit();             //** Exit of File System module
      }
   }

   // Execute data logging background task
   if( g_b_datalogging_running )
   {
      g_b_datalogging_running = datalogging_file_write_sector();
      if( !g_b_datalogging_running )
      {
         datalogging_stop();
         nav_exit();             //** Exit of File System module
      }
   }
}
```

### 5.2.2 Open disk

In this example, only one *navigator handle*[1] is needed because the application has only one disk exploration and only one file opened at the same time. In that case, we select the default navigator handle 0 *"nav_select(0)"*.

Note: 1. see Application Note AVR114 for more information about navigator handle

The algorithm to open a disk depends on the number of disks connected (see configuration in conf_access.h file). By default the example uses the DataFlash and MMC/SD driver and the following algorithm:

```
Try to mount MMC/SD disk and format if necessary
If error (disk not present, fail, …)
    Try to mount DataFlash disk and format if necessary
    If error (disk not present, fail, …)
        Abort data logging
```

**Figure 5-4.** datalogging_open_disk() routine

```
Bool datalogging_open_disk(void)
{
   U8 u8_i;
   nav_select(FS_NAV_ID_DEFAULT);

#if( (LUN_2 == ENABLE) && (LUN_3 == ENABLE)) // Configuration set in conf_access.h
   // Select and try to mount disk MMC/SD (lun 1) or DataFlash (lun 0)
   // Try first the MMC/SD
   for( u8_i=1; u8_i!=0xFF; u8_i-- )
#else
   // There is only one memory MMC/SD or DataFlash (lun 0)
   for( u8_i=0; u8_i!=0xFF; u8_i-- )
#endif
   {
      if( nav_drive_set(u8_i) )              // Select driver (not disk)
      {
         // Driver available then mount it
         if( !nav_partition_mount() )
         {
            // Error during the mount then check error status
            if( FS_ERR_NO_FORMAT != fs_g_status )
               continue;                     // Disk fails (not present, HW error, system error,
            // Disk no formated then format it
            if( !nav_drive_format(FS_FORMAT_DEFAULT) )
               continue;                     // Format fails
         }
         return TRUE;                        // Here disk mounted
      }
   }
   return FALSE;                             // No valid disk found
}
```

### 5.2.3 Create path file

This part creates the following path file “Disk:\logxxx\logxxx.bin”.

The *datalogging_create_path_file()* routine creates the directory “\logxxx\”. The “***nav_setcwd()***” routine searches and eventually creates the path (the third argument must be TRUE).

**Figure 5-5.** datalogging_create_path_file() routine

```
Bool datalogging_create_path_file(void)
{
   char ascii_name[15];
   U16 u16_dir_num = 0;

   if( !nav_dir_root() )
      return FALSE;           // Error FS

   while( u16_dir_num < 30 )  // The limitation of number of directories is just an example
   {
      sprintf( ascii_name, "./log%03d/", u16_dir_num);

      // Enter in sub directory and eventually create it if don't exist
      if( !nav_setcwd( ascii_name, FALSE, TRUE) )
         return FALSE;        // Error FS

      // Create a file
      if( datalogging_create_file() )
         return TRUE;         // File created

      // Here, the directory is full then go to parent directory to create the next sub directory
      if( !nav_dir_gotoparent() )
         return FALSE;        // Error FS
      u16_dir_num++;
   }
   return FALSE;              // Too many log directories and files
}
```

The *datalogging_create_file()* routine creates the file “*\logxxx.bin*” . This one limits the number of log file in a log directory to 10, because the exploration of a directory with many files may be too slow.

**Figure 5-6.** datalogging_create_file() routine

```
Bool datalogging_create_file(void)
{
   char ascii_name[15];
   U16 u16_file_num = 0;

   while( u16_file_num < 10 )
   {
      // Create file
      sprintf( ascii_name, "log%03d.bin", u16_file_num);
      if( nav_file_create( ascii_name ))
         return TRUE;   // Here, the file is created, closed and emptied
      // Error during creation then check error
      if( fs_g_status != FS_ERR_FILE_EXIST )
         return FALSE;  // Error FS (Disk full, directory full, ...)
      // The file exists then increment number in name
      u16_file_num++;
   }
   return FALSE;        // Too many log files in current directory
}
```

### 5.2.4 File space allocation

This section is not mandatory but allows increasing the data logging bandwidth. For more explanation, see §6.5 of AVR114 application note.

Note: At the end of data logging, the remaining allocated memory (size allocated - final size) is freed up when the file is closed.

**Figure 5-7.** datalogging_alloc_file_space() routine

```
Bool datalogging_alloc_file_space(void)
{
   Fs_file_segment g_recorder_seg;

   // Open the file created in write mode
   if( !file_open(FOPEN_MODE_W))
      return FALSE;

   // Define the size of segment to alloc (unit 512 B)
   // Note: you can alloc more in case of you don't know the total size
   g_recorder_seg.u16_size = FILE_ALLOC_SIZE;

   // Alloc in FAT a cluster list equal or inferior at segment size
   if( !file_write( &g_recorder_seg ))
   {
      file_close();
      return FALSE;
   }

   // If you want then you can check the mimimun size allocated
   if( g_recorder_seg.u16_size < FILE_ALLOC_SIZE_MIN )
   {
      file_close();
      nav_file_del(FALSE);
      return FALSE;
   }

   // Close/open file to reset size
   // Note: This sequence doesn't remove the previous FAT allocation
   file_close();                              // Closes file
   if( !file_open(FOPEN_MODE_W))              // Opens file in write mode and forces the size to 0
      return FALSE;
   return TRUE;   //** File open and FAT allocated
}
```

### 5.2.5 File filling

The “*file_write_buf()*“ is the best routine to fill a file. Using a multiple of 512 B as buffer size and current file position will give optimal speed performance, because the memory interface uses a block of 512 B.

Buffers (2 * 512 B) are filled by timer0 interrupt routine. The maximum data logging bandwidth on ATmega32U4 at 8 MHz is 18 KB/s. This value has been measured with the following example.

**Figure 5-8.** datalogging_file_write_sector() routine

```
Bool datalogging_file_write_sector(void)
{
   // !!!! Note :
   // if the written buffer size has a multiple of 512 B
   // and if the current file position is a multiple of 512 B
   // then the "file_write_buf()" routine is very efficient.
   if( g_b_buf_full[g_u8_cur_buf] )
   {
      if( !file_write_buf( &g_data_buf[g_u8_cur_buf*FS_SIZE_OF_SECTOR], FS_SIZE_OF_SECTOR ) )
         return FALSE;  // Error write
      g_b_buf_full[g_u8_cur_buf] = FALSE;
      // Now wait new buffer
      g_u8_cur_buf++;
      if( NB_DATA_BUF == g_u8_cur_buf )
        g_u8_cur_buf = 0;
   }
   return TRUE;
}
```

## Headquarters


***Atmel Corporation***
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600


## International


***Atmel Asia***
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

***Atmel Europe***
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

***Atmel Japan***
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581


## Product Contact

***Web Site***
www.atmel.com

***Technical Support***
avr@atmel.com

***Sales Contact***
www.atmel.com/contacts

***Literature Request***
www.atmel.com/literature