JesFs – BlackBox

A Flight Recorder for Home Use

Many technical devices work reliable for years and years and no one cares about them. But if they fail, there is always the question: WHAT WENT WRONG?

This is, where the JesFs (Jo's Embedded Serial File System) could become extremely useful:

JesFs is was especially designed as Ultra-Low-Power and extremely robust File System for all kinds of Embedded Processors (https://github.com/joembedded/JesFs).

As written in the JesFs manual, I wrote JesFs for my very own needs as Embedded Engineer and I use it daily! JesFs is reliable, allows Secured Bootloaders with Secured Updates "Over the Air", has a very low memory footprint and a lot of nice other features, like "History Logging" for Diagnostics:

History Logging

A typical data logger has a large memory and its primary purpose is to fill this memory with as much as possible data. But on many embedded devices only a limited amount of memory is available for historic data.

There are 2 problems:

→ "Ring Buffers" are difficult to implement with Flash Memory

→ It must be ensured, that only a limited amount of memory ist used for the History

JesFs has an almost ideal solution for this, called "Unclosed Files": JesFs allows to append data to files as long as they have not been closed. More Details in the JesFs docs (Link above).

Implementation of a History Log with JesFs

The first step is to decide the amount of memory JesFs is allowed to use. Often some kBytes are more than enough. The second step is to implement a system, that works with two Files, so always at least one File is completely full with historic data.

Let's call it Files "Data.pri" and "Data.sec":

- Data is always appended to "Data.pri"

- Between appending data, the Embedded System could sleep, Reset or do whatever you want.

- If the size of "Data.pri" is larger then a certain amount, "Data.pri" is renamed as "Data.sec"

- So there will always be historical data, just read "Data.pri" first and "Data.sec" afterwards

- And: this system guarantees, that only a small amount of the total memory is used for logging.

Demo Sourcecodes (2 solutions)

JesFs was primarily designed for Embedded Processors (like the CPUs from the SimpleLink series from CPU with embedded Radio, WiFi or Bluetooth 5.0). But it runs also on any Desktop PC, so it is easy to evaluate or to test.
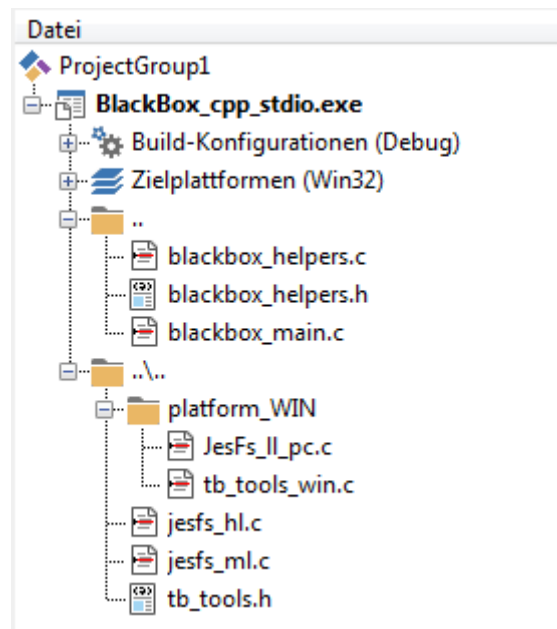
Solution 1: A test plattform for PC with C++ - Builder

A very convenient system is the free (as state 07/2019) **Embarcadero® C++ - Builder Community Edition** (but it will compile on almost any other C-compiler too).

Simply copy all source files from https://github.com/joembedded/JesFs into a

"Command Line with C"-project with tis structure:

Project Tree for C++ - Builder.

As Low Level Driver 'JesFs_ll_pc.c' is used to simulate a Serial Flash

Now, "Blackbox_main.c" should compile with no errors.

Solution 2: SEGGER Embedded Studio (SES)

The Nordic nRF52840 DK (PCA10056) contains a large 8 Mbyte serial Flash.

A complete Project (built with **nRF5 SDK v16.0.0**) is included in
`.../`**`JesFsDemo`**`/usecase_BlackBox/platform_nRF52_pca10056/`
                                        `ses/BlackBox_pca10056.emProject`

Because SES works best with relative pathes, copy the project into your nRF5 SDK v16.0.0
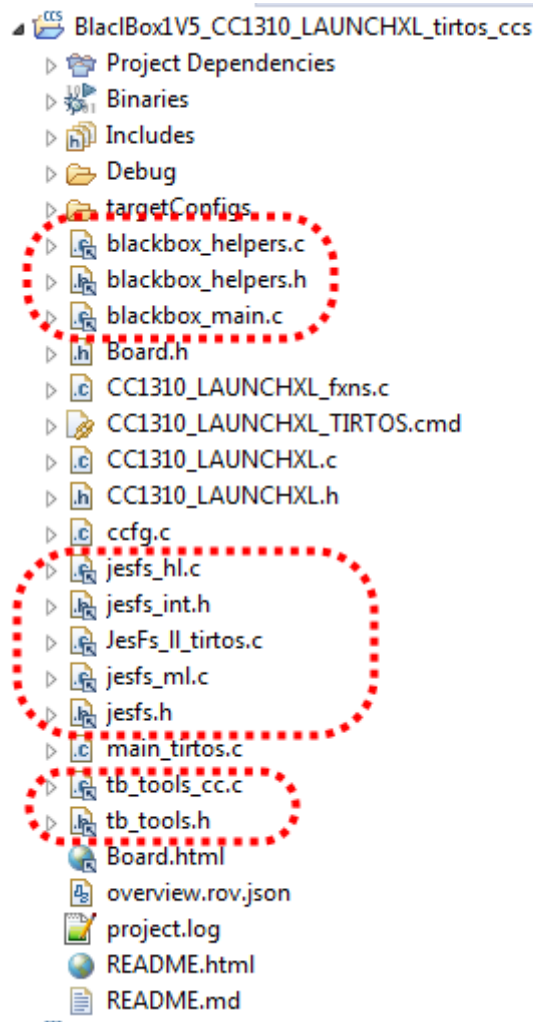installation directory. In my case the SDK is installed in:

`C:/nordic/nRF5_SDK_16.0.0_98a08e2`

and the BlackBox Demo is is installed in:

`C:/nordic/nRF5_SDK_16.0.0_98a08e2/own_projects/test/`**`JesFsDemo`**`/…`

Solution 3: CCS

The TI-Launchpads for SimpleLink CPUs contain a on-board Serial Flash with 1MByte. For developing Software for the SimpleLink CPU TI offers the free **CCS Code Composer Studio**

Project Tree for CCS and a CC1310-Launchpad. Use the demo project 'empty' to start and remove the unnecessary files, then add the files/folders in the red boxes.
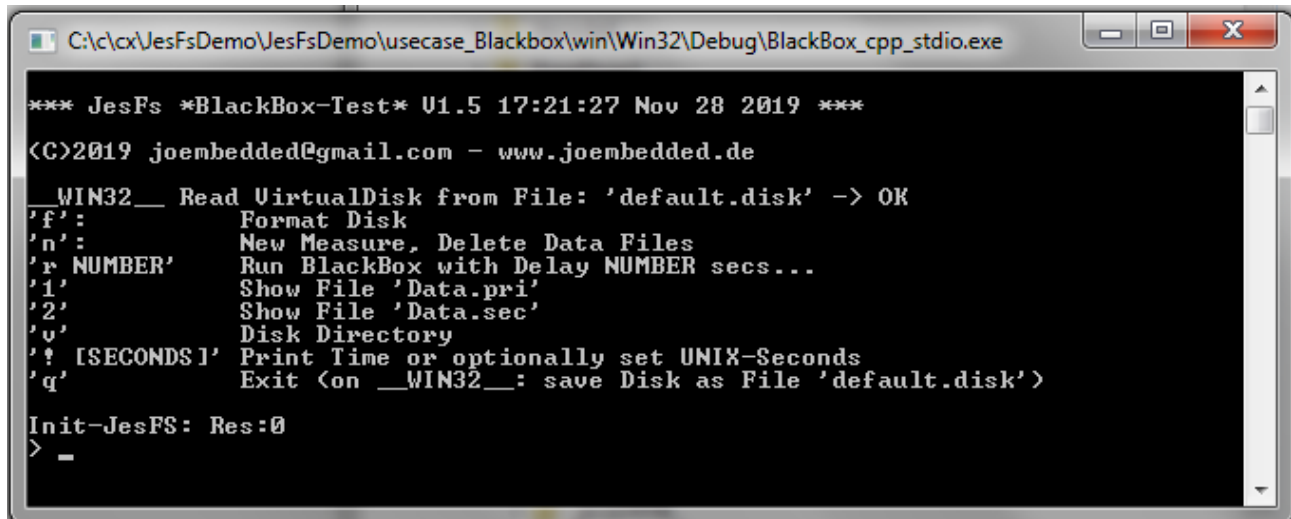
It is important to add **CC13XX_CC26XX** *as Preprocessor Definition (Project Options -> Preprocessor Definitions)*

The Launchpads offer 2 virtual COM-ports. One of them is connected to RX/TX on the Launchpad. Use a terminal program to communicate at 115200 Baud, "8N1" (set to "CR" on <Enter>).

Running the Project

On the TI-Launchpad the Project will directly use the on-board Serial Flash.

The PC simulation uses a file "default.disk" to simulate a non-volatile Flash. Since after the first start, this simulated or real existen Flash is not formated, the demo will report it:
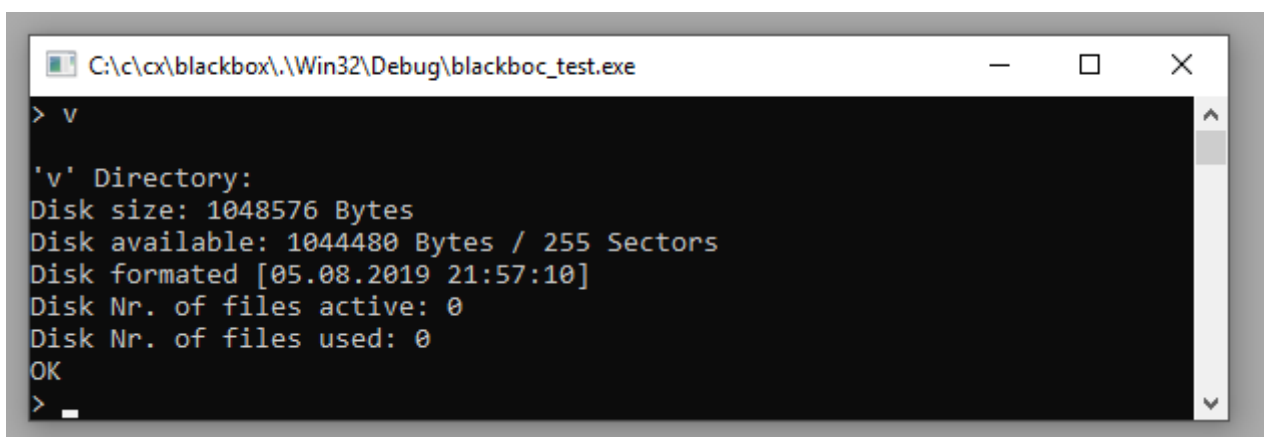

"-203" means: "File not found" and "-108" means: "Unformated" (You can find all error coded in "jesFs.h").

Step 1: Format the disk with "f"

After this, a formated disk is stored on disk (the default size is the same as on the TI-Launchpads for the Simple-Link CPUs)

Step 2: Check the Disk with "v"
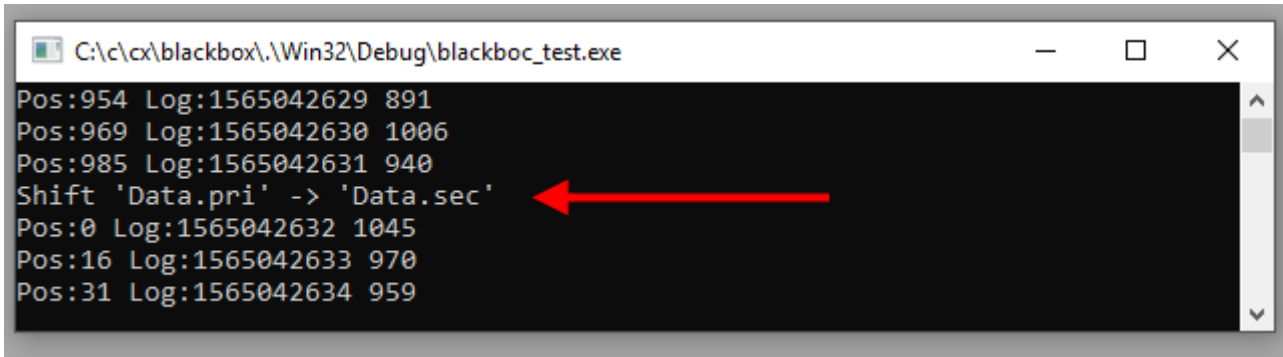
This will give you a directory listing of the JesFs disk:

Here you an see: 1 MB are available
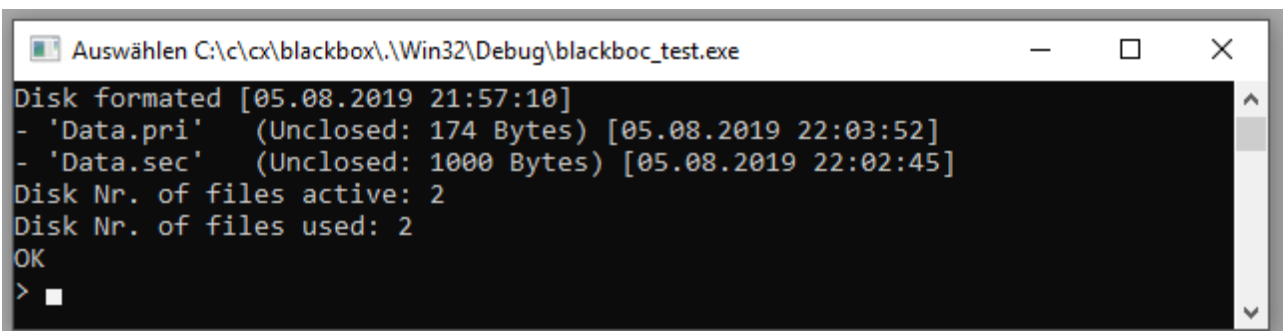
Step 3: Type "r 1"

This will add each second (parameter "1") one line of (sample) data to the history, The default (minimum) history length is set to 1000 (Bytes), so will notice the file shift after ca. 1000 Bytes:

Hit any Key to stop the measure!

Step 4: Show the data with "v":

The older data are in "Data.sec", the newer in "Data.pri".

Step 5: Display the data with "1" and "2"

With "1" for "Data.pri" and "2" for "Data.sec". Both together give the history.

Step 6: Start a new Measure with "n".

Other commands:

'!' shows the current UNIX Timestamp (seconds since 1.1.1970 (UTC))

'!' + number sets the UNIX Timestamp (only on Launchpads)

'q' exits the software on PC, on the Launchpad the behaviour might depend on the debugger.

Source Codes

"Blackbox_main.c" contains some helper functions, but the History logger is completely realised in only a few lines of C:

```c
/********************************************************************
* run_blackbox(asec)
* Take a record each asec secs, until Data.pri is >= HISTORY,
* then shift it to Data.sec and delete Data.pri.
* This demo uses "unclosed Files", which is very useful here.
* Run recoder loop *FOREVER* or until user hits key
********************************************************************/
int16_t run_blackbox(uint32_t delay_secs){
     FS_DESC fs_desc, fs_desc_sec;    // 2 JesFs file descriptors
     uint32_t asecs;
     uint16_t len;
     int16_t res;
     for(;;){    // Forever
          // modify a random value and get time (UNIX seconds)
          value+=(rand()&255)-128;    // Move sample value
          asecs=fs_get_secs();

          // Build the data we want to save: Time + Value
          len=sprintf(sbuffer,"%u %d\n",asecs,value);
          // Filesystem may be sleeping (= UltraLowPowerMode), WAKE fast
          res=fs_start(FS_START_RESTART);
          if(res) break;

          // Flags (see docu): Create File if not exists and open in
                                                        RAW-mode,
          // in RAW-Mode file is not truncated if existing
          res=fs_open(&fs_desc,"Data.pri",SF_OPEN_CREATE|SF_OPEN_RAW);
          if(res) break;

          fs_read(&fs_desc,NULL,0xFFFFFFFF); // (dummy) read as much as
                                                        possible
          // Show what will be written:
          tb_printf("Pos:%u Log:%s",fs_desc.file_len,sbuffer);

          // write the new data (ASCII) to the file
          res=fs_write(&fs_desc,sbuffer,len);
          if(res) break;

          // Now make a file shift if more data than HISTORY
          if(fs_desc.file_len>= HISTORY){
               tb_printf("Shift 'Data.pri' -> 'Data.sec'\n");
               // Optionally delete and (create in any case) backup
                                                             file
               res=fs_open(&fs_desc_sec,"Data.sec",SF_OPEN_CREATE);
               if(res) break;
```

```
                    // rename (full) data file to secondary file
                    res=fs_rename(&fs_desc,&fs_desc_sec);
                    if(res) break;

            }

            //---------- dooze --------
            fs_deepsleep(); // Set Filesystem to UltraLowPowerMode
            sleep(delay_secs); // Allow UltraLowPowerMode for CPU (UART
                                                        still on)
            if(tb_kbhit()) break;          // End
        }
        while(tb_kbhit()) tb_getc();
        tb_printf("Result: %d\n\n",res);    // 0: OK, else see 'jesfs.h'
        fs_start(FS_START_RESTART);           // Wake Filesystem, but
        return res;                           // Return last result
}
```

Results

Using JesFs is very easy. Feel free to use it for your own own projects or port it to any other processors.

***