Part 1: EMMC and SD card test Experiment

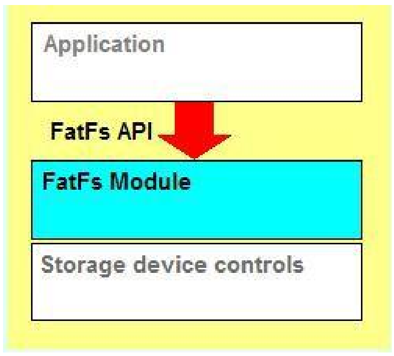
The experimental project is "**ps _emmc**"

This chapter introduces the use of FatFs file system module to write a file to EMMC and SD card, and read its content.

Part 1.1: Introduction to FatFs

FatFs is a general file system module used to implement FAT file system in small embedded systems. The writing of FatFs follows ANSI C, so it does not depend on the hardware platform. It can be embedded in cheap microcontrollers, such as 8051, PIC, AVR, SH, Z80, H8, ARM, etc., without any modification.

The application program calls the FatFs system module through the API function to control the storage devices of the SD card.



The FatFs system provides many API functions. Below we list the API functions that will be used in our routines.

f_mount: Register/Cancel a work area (Work Area)

f_open: Open/create a file

f_close: Close a file

f_read: Read file

f_write: Write file

For the introduction and description of API functions, you can refer to the following website for a deeper understanding. This website gives instructions and examples of each API function.

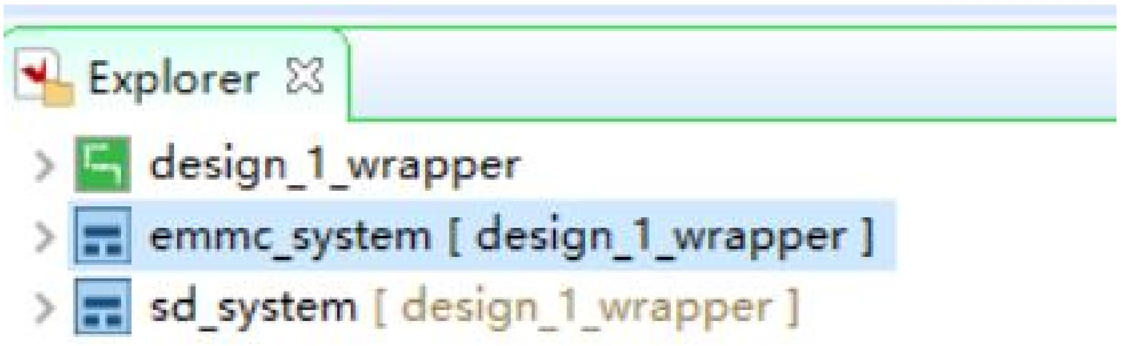
http://elm-chan.org/fsw/ff/00index_e.html

Part 1.2: Hardware Environment

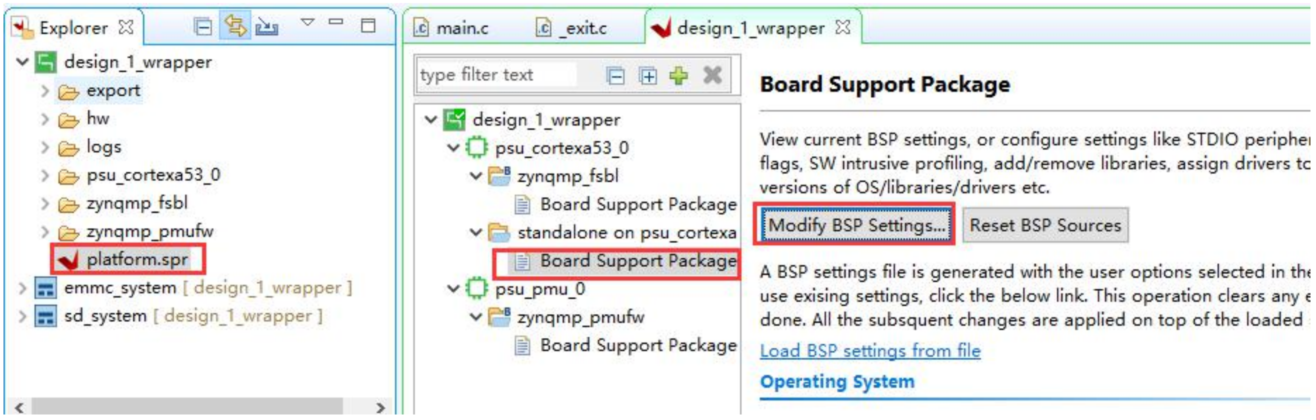
Use the "ps_hello" project to save another project. The SD card and EMMC have been configured in the project, and there is no need to configure it here. Export hardware information, not including bitstream

Part 1.3: Vitis Programming

1) The test program of EMMC and SD card is provided in the example, the program flow is: Register a workspace → Check status → Check if the driver is ready, if the file system exists, if not, create → Check if the directory file exists, if there is, delete it → Write to a file → Read the contents of the file.

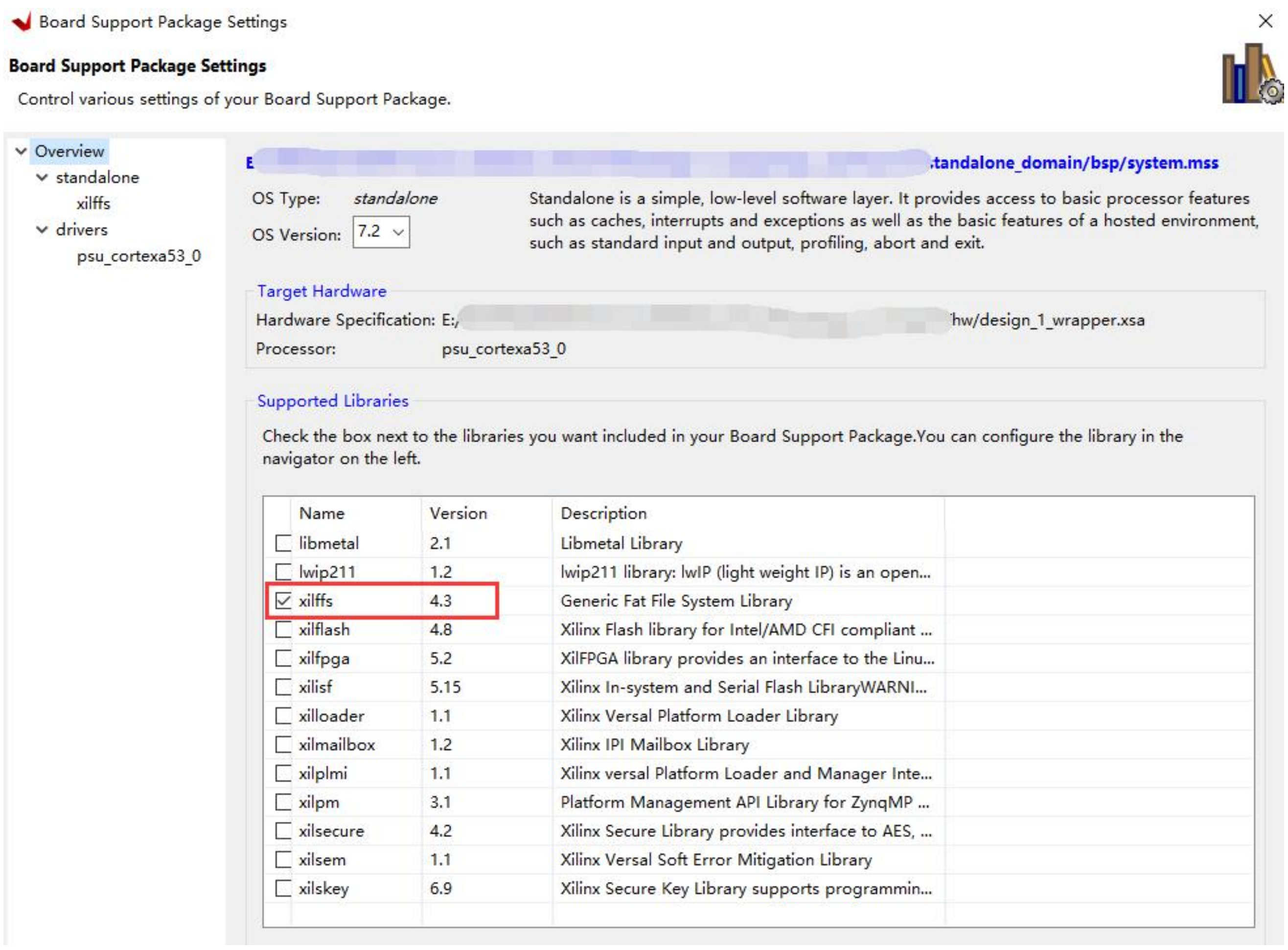

Explorer
design_1_wrapper
emmc_system [design_1_wrapper]
sd_system [design_1_wrapper]

2) Note that the BSP must check the xilffs library



View current BSP settings, or configure settings like STDIO periphe
flags, SW intrusive profiling, add/remove libraries, assign drivers tc
versions of OS/libraries/drivers etc.

A BSP settings file is generated with the user options selected in the
use exising settings, click the below link. This operation clears any e
done. All the subsequent changes are applied on top of the loaded



Board Support Package Settings

Board Support Package Settings

Control various settings of your Board Support Package.

- Overview
  - standalone
    - xilffs
  - drivers
    - psu_cortexa53_0

E[redacted]tandalone_domain/bsp/system.mss

OS Type: *standalone*

OS Version: 7.2

Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.

Target Hardware

Hardware Specification: E:/[redacted]hw/design_1_wrapper.xsa

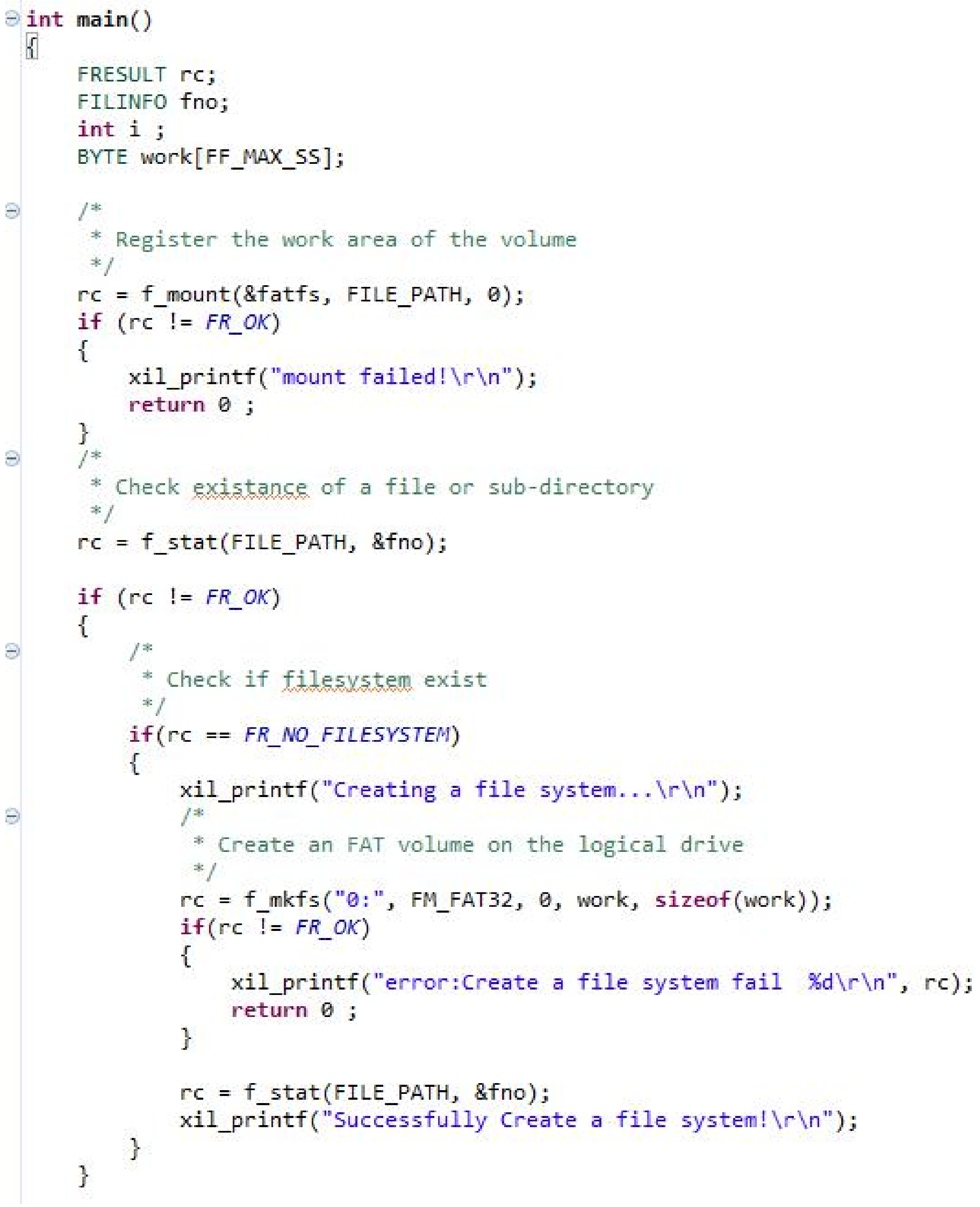
Processor: psu_cortexa53_0

Supported Libraries

Check the box next to the libraries you want included in your Board Support Package.You can configure the library in the navigator on the left.

	Name	Version	Description
☐	libmetal	2.1	Libmetal Library
☐	lwip211	1.2	lwip211 library: lwIP (light weight IP) is an open...
☑	xilffs	4.3	Generic Fat File System Library
☐	xilflash	4.8	Xilinx Flash library for Intel/AMD CFI compliant ...
☐	xilfpga	5.2	XilFPGA library provides an interface to the Linu...
☐	xilisf	5.15	Xilinx In-system and Serial Flash LibraryWARNI...
☐	xilloader	1.1	Xilinx Versal Platform Loader Library
☐	xilmailbox	1.2	Xilinx IPI Mailbox Library
☐	xilplmi	1.1	Xilinx versal Platform Loader and Manager Inte...
☐	xilpm	3.1	Platform Management API Library for ZynqMP ...
☐	xilsecure	4.2	Xilinx Secure Library provides interface to AES, ...
☐	xilsem	1.1	Xilinx Versal Soft Error Mitigation Library
☐	xilskey	6.9	Xilinx Secure Key Library supports programmin...

3) In the main function, register the workspace and check the file system. If there is no file system, you need to create a file system. Use the “f _mkfs” function. If it is SD0, the path is "0:". If it is SD1, the path is "1". , It takes a long time to create a file system, so be patient !



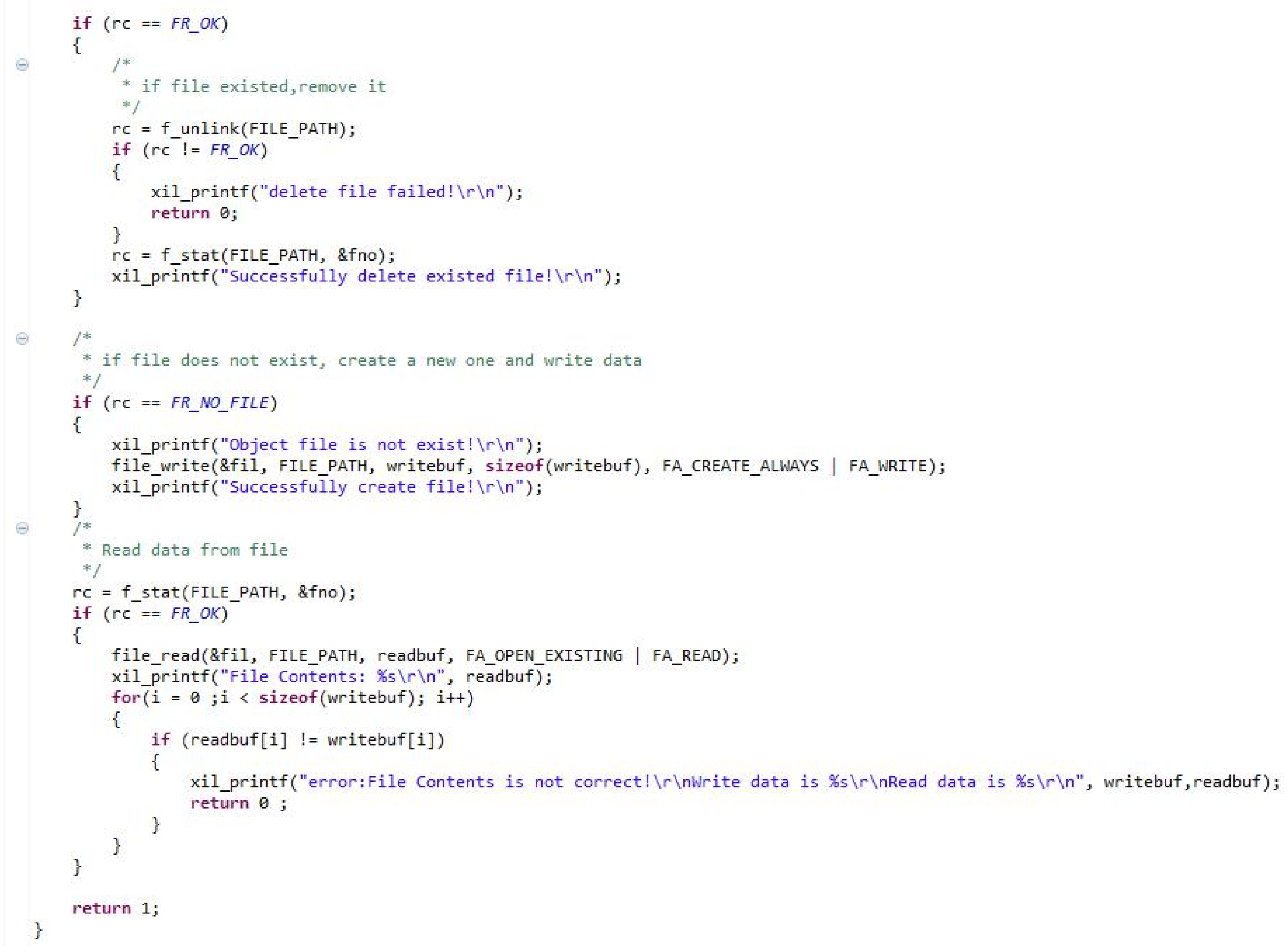
```
int main()
{
    FRESULT rc;
    FILINFO fno;
    int i ;
    BYTE work[FF_MAX_SS];

    /*
     * Register the work area of the volume
     */
    rc = f_mount(&fatfs, FILE_PATH, 0);
    if (rc != FR_OK)
    {
        xil_printf("mount failed!\r\n");
        return 0 ;
    }
    /*
     * Check existance of a file or sub-directory
     */
    rc = f_stat(FILE_PATH, &fno);

    if (rc != FR_OK)
    {
        /*
         * Check if filesystem exist
         */
        if(rc == FR_NO_FILESYSTEM)
        {
            xil_printf("Creating a file system...\r\n");
            /*
             * Create an FAT volume on the logical drive
             */
            rc = f_mkfs("0:", FM_FAT32, 0, work, sizeof(work));
            if(rc != FR_OK)
            {
                xil_printf("error:Create a file system fail  %d\r\n", rc);
                return 0 ;
            }

            rc = f_stat(FILE_PATH, &fno);
            xil_printf("Successfully Create a file system!\r\n");
        }
    }
```

4) Then judge whether the file exists, if it exists, delete it first, then write a file to it, and read it out again.

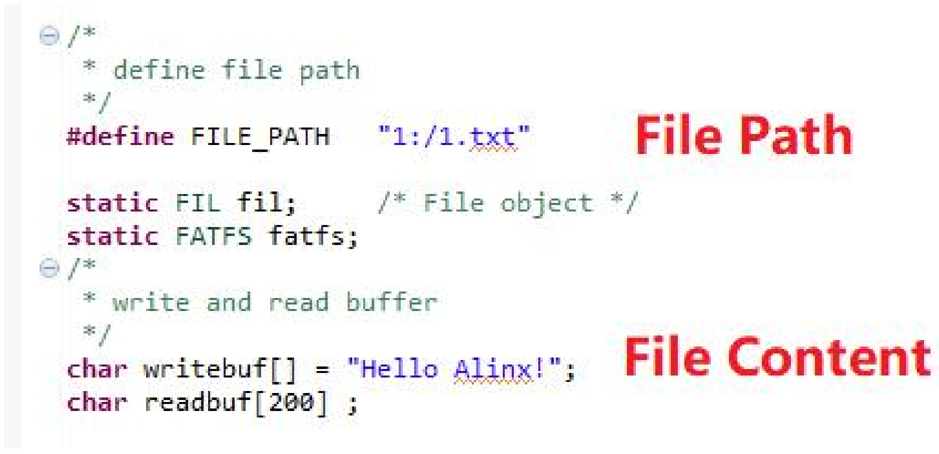


```
    if (rc == FR_OK)
    {
        /*
         * if file existed,remove it
         */
        rc = f_unlink(FILE_PATH);
        if (rc != FR_OK)
        {
            xil_printf("delete file failed!\r\n");
            return 0;
        }
        rc = f_stat(FILE_PATH, &fno);
        xil_printf("Successfully delete existed file!\r\n");
    }

    /*
     * if file does not exist, create a new one and write data
     */
    if (rc == FR_NO_FILE)
    {
        xil_printf("Object file is not exist!\r\n");
        file_write(&fil, FILE_PATH, writebuf, sizeof(writebuf), FA_CREATE_ALWAYS | FA_WRITE);
        xil_printf("Successfully create file!\r\n");
    }
    /*
     * Read data from file
     */
    rc = f_stat(FILE_PATH, &fno);
    if (rc == FR_OK)
    {
        file_read(&fil, FILE_PATH, readbuf, FA_OPEN_EXISTING | FA_READ);
        xil_printf("File Contents: %s\r\n", readbuf);
        for(i = 0 ;i < sizeof(writebuf); i++)
        {
            if (readbuf[i] != writebuf[i])
            {
                xil_printf("error:File Contents is not correct!\r\nWrite data is %s\r\nRead data is %s\r\n", writebuf,readbuf);
                return 0 ;
            }
        }
    }

    return 1;
}
```

5) Define the file path, file content, and file path. If SD0 is "0:/", if it is SD1, it is "1:/". For this project, SD0 is SD card, and SD1 is EMMC.



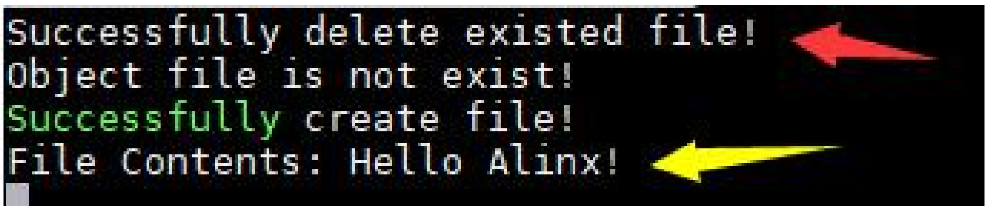
```
/*
 * define file path
 */
#define FILE_PATH   "1:/1.txt"

static FIL fil;     /* File object */
static FATFS fatfs;
/*
 * write and read buffer
 */
char writebuf[] = "Hello Alinx!";
char readbuf[200] ;
```

Part 1.4: Download and Debug

Part 1.4.1: EMMC test

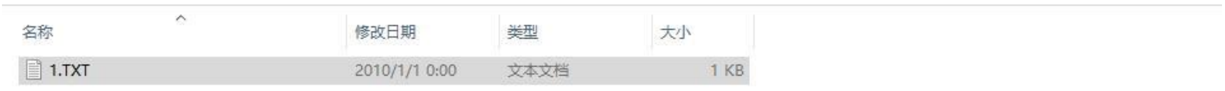
Download the EMMC program to the FPGA development board, the following information will be printed. If the directory file already exists in the EMMC, the first sentence will be printed, and the last sentence will be the content of the print file.



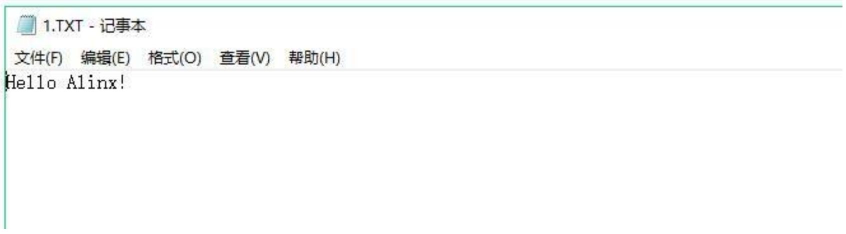
```
Successfully delete existed file!
Object file is not exist!
Successfully create file!
File Contents: Hello Alinx!
```

Part 1.4.2: SD card test

1) Download the SD card program to the FPGA development board in the same way, and the same content as the EMMC will be printed.
2) The SD card can be taken out and read on the computer with a card reader



名称	修改日期	类型	大小
1.TXT	2010/1/1 0:00	文本文档	1 KB



```
Hello Alinx!
```