1 SD 卡的文本读写实验

工程源码	|-ACZ702 v2.0 开发板 |-- SD_Card_Test
相关视频课程	无

章节导读

在本章节中，我们将探索如何将文本存储到 SD 卡，并简要概述 SD 卡的基本原理。本章的目标是使读者对 SD 卡操作有更深入地了解，并掌握基于 SD 卡的文本读写方法。

1.1 Micro SD 卡介绍

Micro SD Card，原名 Trans-flash Card（TF 卡），是一种极细小的快闪存储器卡。

(1) 分类

Micro SD 卡主要可以按照以下三个方面来分类：

1. 容量分类：按照容量，Micro SD 卡可以分为标准 Micro SD 卡、Micro SDHC 卡和 Micro SDXC 卡；它们最大容量分别为 2GB、32GB、2TB。

表 1-1 容量分类

类别	描述	最大容量
标准 Micro SD 卡	最小容量的 Micro SD 卡	2GB
Micro SDHC 卡	较大容量的 Micro SD 卡	32GB
Micro SDXC 卡	最大容量的 Micro SD 卡	2TB

2. 速度分类：按照速度，Micro SD 卡分为 Class 2、Class 4、Class 6 和 Class 10。数字代表卡的最小写入速度，比如，Class 2 的最小写入速度为 2MB/s，Class 10 的最小写入速度为 10MB/s。一般情况下，速度越快的卡，其价格也会相对越高。

表 1-2 速度分类

类别	描述	最小写入速度
Class 2	其数字代表卡的最小写入速度，速度相对较慢	最小写入速度为 2MB/s
Class 4	中等的写入速度	最小写入速度为 4MB/s
Class 6	较高的写入速度	最小写入速度为 6MB/s
Class 10	最高的写入速度，价格相对高	最小写入速度为 10MB/s

3. 用途分类：此外，针对特定的使用场景，还有部分 Micro SD 卡分为高耐久卡（如工业级 Micro SD 卡）、高速卡（如用于摄影或游戏的高速 Micro SD

卡）等。

表 1-3 用途分类

类别	描述
高耐久卡（如工业级 Micro SD 卡）	适应特殊环境，对耐久性和稳定性有特别要求的卡
高速卡（如用于摄影或游戏的高速 Micro SD 卡）	适合需要快速写入和读取的场景

(2) 引脚

Micro SD 卡共有 8 个引脚，如下所示：

1. 电源引脚（VCC）：提供电源给卡，工作电压是 2.7-3.6V。

2. 接地引脚（GND）：用于建立共地。

3. 数据线 0 引脚（DAT0）：用于数据传输。在 SPI 模式下，此引脚充当 MISO（主输入，从输出）。

4. 数据线 1 引脚（DAT1）：用于数据传输。

5. 数据线 2 引脚（DAT2）：用于数据传输。在 SPI 模式下，此引脚不使用。

6. 数据线 3 引脚（DAT3/CS）：用于数据传输。在 SPI 模式下，此引脚充当片选（Chip Select）。

7. 命令线引脚（CMD）：用于发送命令给卡。在 SPI 模式下，此引脚充当数据输入（Data In）。

8. 时钟线引脚（CLK）：提供同步的时钟信号。

图 1-1 Micro SD 卡

1.2 TXT 文本

TXT 文本通常作为一种简单的文本数据格式，广泛被用于存储和处理不含

任何格式化的纯文本数据。在本次实验中，TXT 文本将作为实验数据的输出格式之一，用于记录和保存实验数据及结果。TXT 的格式具有相对较高的通用性和可读性，能够方便进行数据的查看、转移和处理。

1.3 文件系统

(1) 什么是文件系统？

文件系统是电脑中的一种存储方法，它规定了数据如何在存储设备中组织和定位。文件系统主要负责文件的存储、检索和管理等工作，包括文件的创建、读取、写入、删除和搜索等服务。它为用户操作文件提供了一种容易理解的方式，同时管理和保护了存储设备上的数据。

常见的文件系统包括以下几种：

1. FAT（File Allocation Table）文件系统：这是早期的文件系统格式，主要适用于硬盘驱动器和闪存设备，包括 FAT12、FAT16 和 FAT32 等版本。

2. NTFS（New Technology File System）文件系统：这是 Windows NT 及其后续版本中使用的主要文件系统，具有高级功能，如数据恢复和访问控制等。

3. EXT（Extended File System）文件系统：这是 Linux 操作系统中使用的文件系统，其中 EXT4 是目前最常用的版本，具有文件系统日志、容量大、文件恢复等功能。

4. exFAT（Extended File Allocation Table）文件系统：这是微软开发的用于特殊设备（如 SD 卡和 USB 闪存驱动器）的文件系统，它移除了 FAT32 中的文件大小限制，并改进了性能。

5. FATFS：一种面向嵌入式系统的文件系统，下面着重讲解。

表 4 常见文件系统

文件系统	简介	应用环境
FAT	较早的文件系统格式	硬盘驱动器、闪存设备
NTFS	Windows 系统主流文件系统	Windows NT 及后续版本
EXT	Linux 系统常用文件系统	Linux 操作系统
exFAT	面向特殊存储设备	SD 卡、U 盘等
FATFS	面向嵌入式系统的文件系统	嵌入式操作系统

(2) FATFS 文件系统

FATFS 文件系统是一种免费、开源的适用于嵌入式系统的文件系统模块。主要用于支持 FAT32、FAT16、FAT12 和 exFAT 格式的不同存储介质，包括 SD 卡、MMC 卡、CF 卡、硬盘等。

以下是一些关于 FATFS 文件系统的基本特性：

1. 嵌入式：FATFS 文件系统是为嵌入式系统设计的，无需操作系统就能运行。

2. 多文件系统支持：支持 FAT32、FAT16、FAT12 以及 exFAT 等文件系统格式。

3. 支持长文件名：支持 FAT 文件系统中的长文件名。

4. 支持 UTF-8 编码：FATFS 文件系统可以处理 UTF-8 编码的文件名和路径名。

5. 代码小巧：代码已经被优化，尽可能减小了体积，适应资源有限的嵌入式系统。

6. 灵活的配置：可以根据应用需求来选择配置，比如支持只读配置、只支持某一种 FAT 文件系统等。

1.4 硬件逻辑系统设计

1.4.1 添加 IP 核

本次设计我们只使用 Zynq 处理系统 IP 核。

1.4.2 IP 核配置

首先，我们需要配置 Zynq IP 核。

(1) DDR

打开 DDR 配置界面，设置 DDR 型号为“MT41K128M16 JT-125”，如图 1-2 所示。

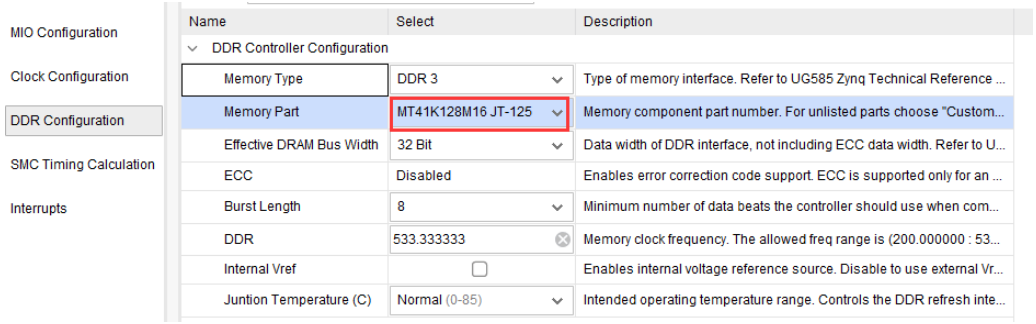

图 1-2 DDR 型号

(2) Clock Configuration

AXI UartLite IP 核的工作时钟应该低于 120MHz，因此这里我们打开 Clock

Configuration 界面，按照图 1-3 配置 PL 的时钟为 100MHz。

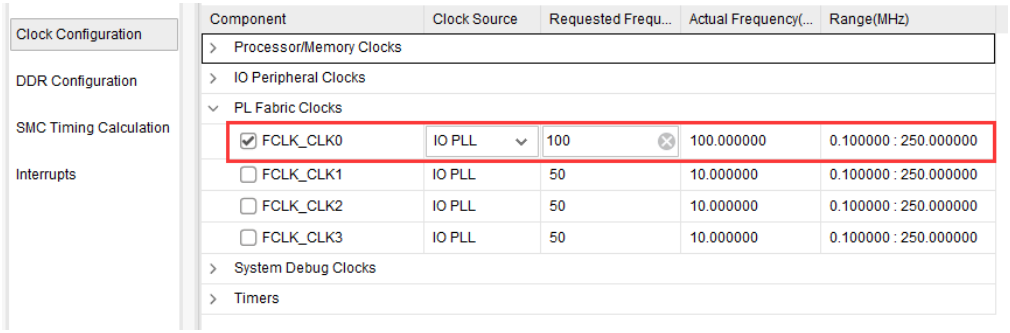

图 1-3 PL 时钟配置

(3) SD

勾选“SD 0”即可，其中 CD、WP、Power 都不需要勾选。

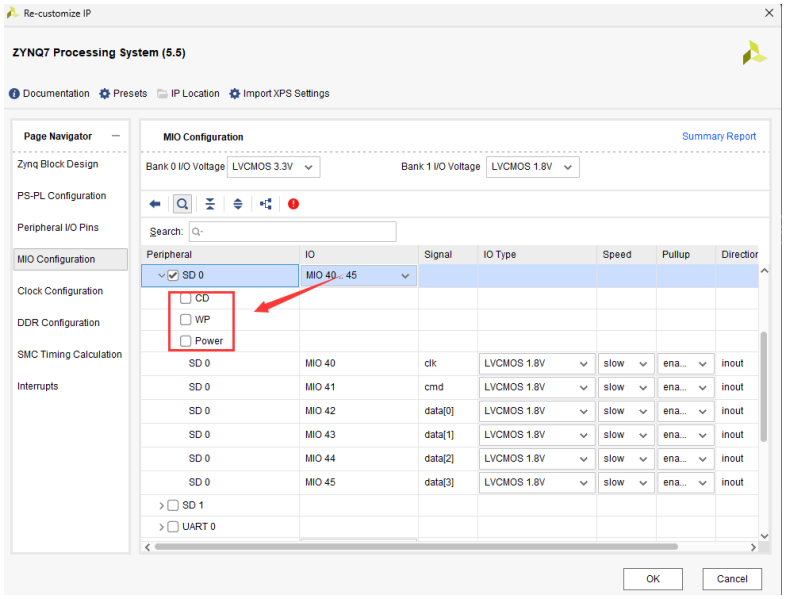

图 1-4 SD

(4) PS_UART

在通信过程中，会通过 PS 串口进行交互，所以这里我们还需要使能 PS 端串口外设，ACZ702 开发板上 PS 侧串口引脚对应 MIO48...49，外设使能如图 1-5 所示：

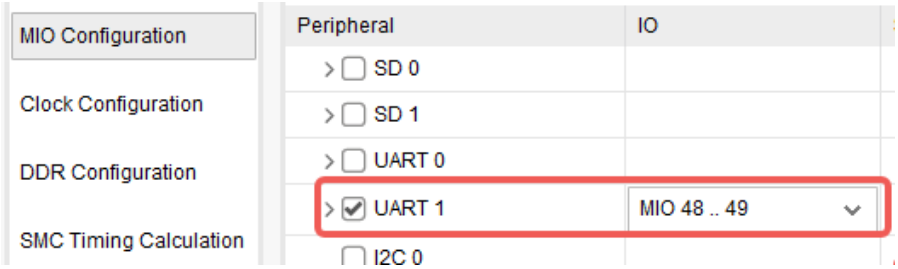

图 1-5 PS 端串口配置

1.4.3 导出引脚

接下来我们需要将 IP 核的引脚导出，点击上方的蓝色小字“Run Block Automation”，让系统自动帮我们导出引脚。此时软件会弹出弹窗，勾选“ALL Automation”，其余部分保持默认，点击 OK，软件便会帮我们导出。

1.4.4 端口连接

首先，如图 1-6 所示，将 FCLK_CLK0 和 M_AXI_GPIO_ACLK 连接。

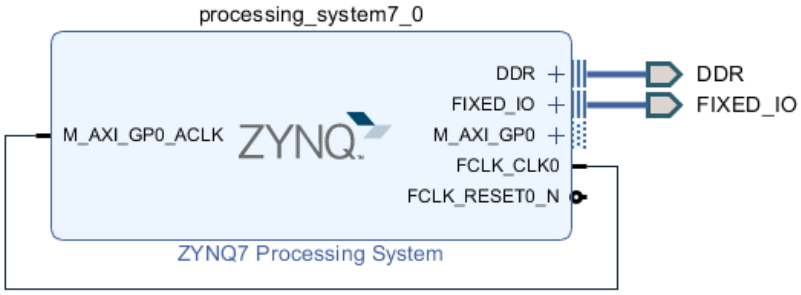

图 1-6 FCLK_CLK0

点击“Run Connection Automation”，并勾选弹窗中全部对象，点击 OK，等待自动连接。

最后，验证设计，出现“Validation successfu...”，说明无错误。

1.4.5 生成封装

点击 sources 资源栏下我们创建的 system 模块设计，单击右键，在展开的功能中选择“Generate Output Products...”生成输出。

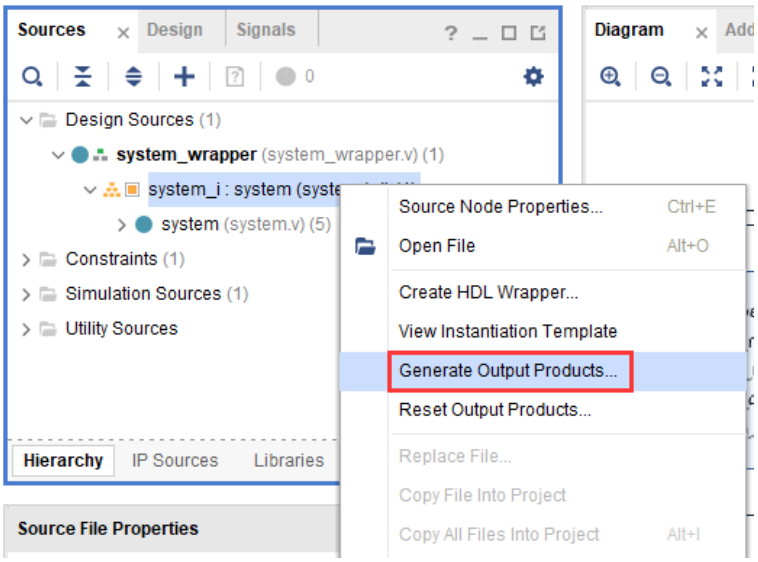

图 1-7 Generate Output Products

如图 1-8 所示，接下来软件会弹出生成输出前的设置界面。在合成选项栏直接选择“Out Of context per IP”即可，下方的“Number of jobs”选项选择最大值 16。设置完成后点击“Generate”开始生成。

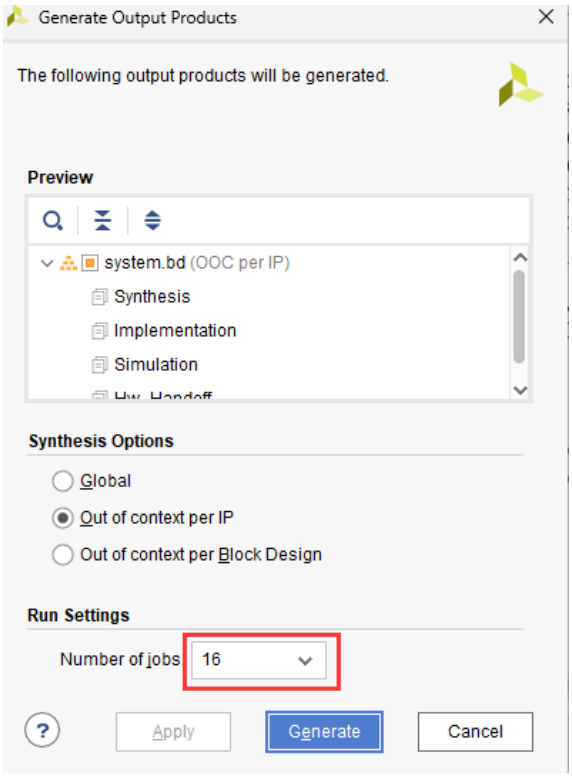

图 1-8 设置生成速度

继续右键单击 system 模块，在展开的功能中选择“Create HDL Wrapper...”创建 HDL 封装。

因为本次实验没有用到 PL 侧资源，所以无需管脚约束；直接生成比特流并将硬件资源描述文件导出到 SDK 即可。

接下来打开 SDK，开始 CPU 软件程序设计。

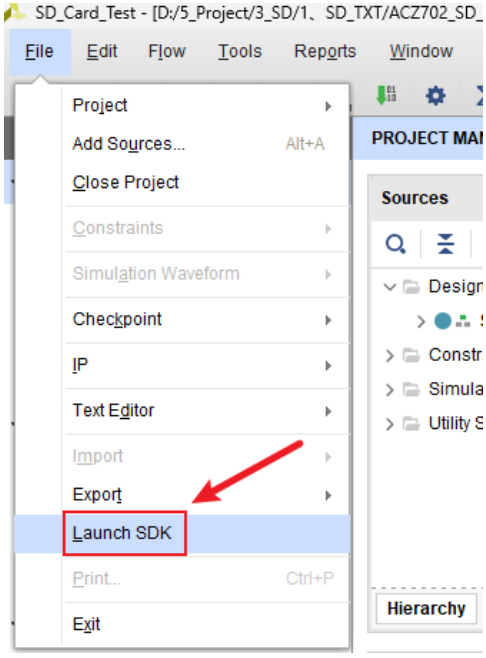

图 1-9 Launch SDK

1.5 CPU 软件程序设计

1.5.1 创建 SDK 工程

打开 SDK 后我们需要新建一个 SDK 工程，点击软件左上方的 File，在展开的功能栏中依次选择 New/Application Project。

接下来会进入工程创建界面。此时我们需要为工程命名，设置为“SD_Card_Test”。这里我们保持默认，点击 Next 进行下一步。

然后为工程选择模板，选择“Empty Application”，如图 1-10 所示，此时右边窗口便会出现当前工程模板的描述，点击 Finish 完成工程创建。

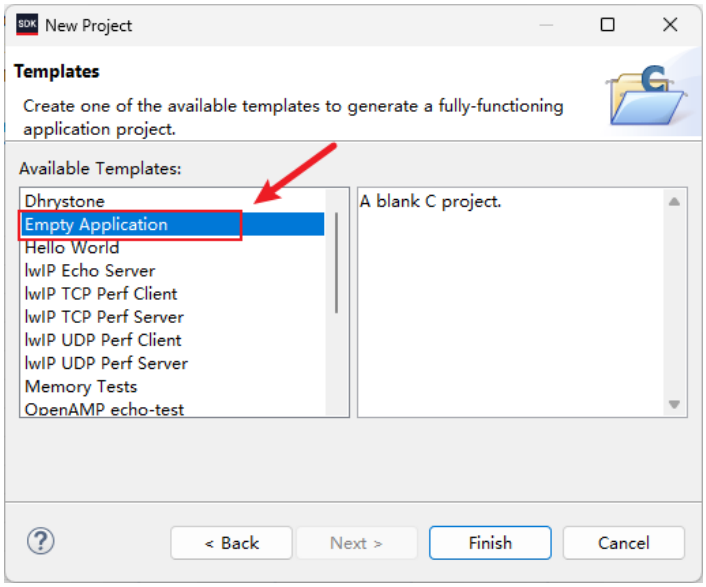

图 1-10 选择模板

1.5.2 添加 FATFS 库

首先，在“SD_Card_Test_bsp”上右键点击，选择“Board Support Package Settings”，如图 1-11 所示。

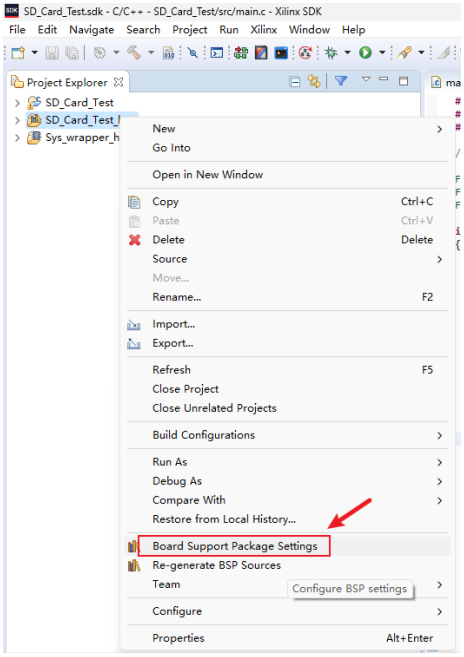

图 1-11 添加 FATFS 库

在新出现的界面上，勾选“xilffs”，如下图所示。

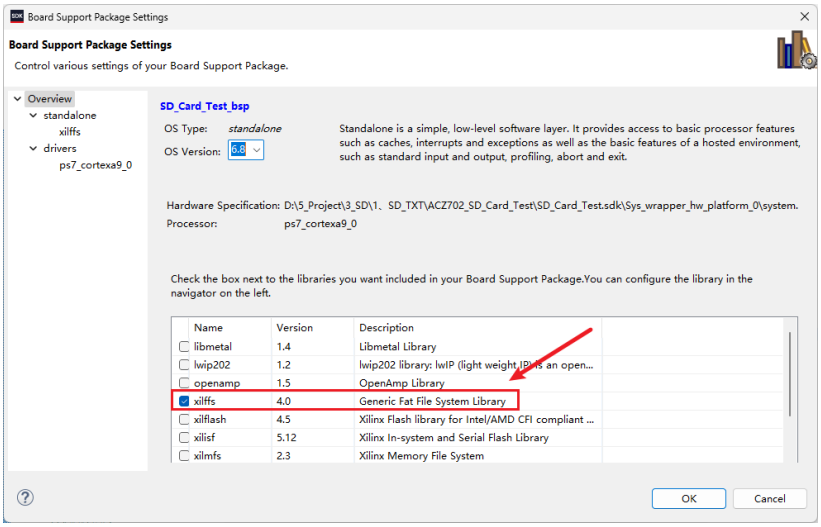

图 1-12 添加 xilffs

此时，在“standalone”下方的会被添加上“xilffs”；继续鼠标左键单击 use_lfn 的 Value 栏，将 false 修改为 true；“use_lfn”是一个配置选项，用来确定是否启用长文件名和识别小写字母。

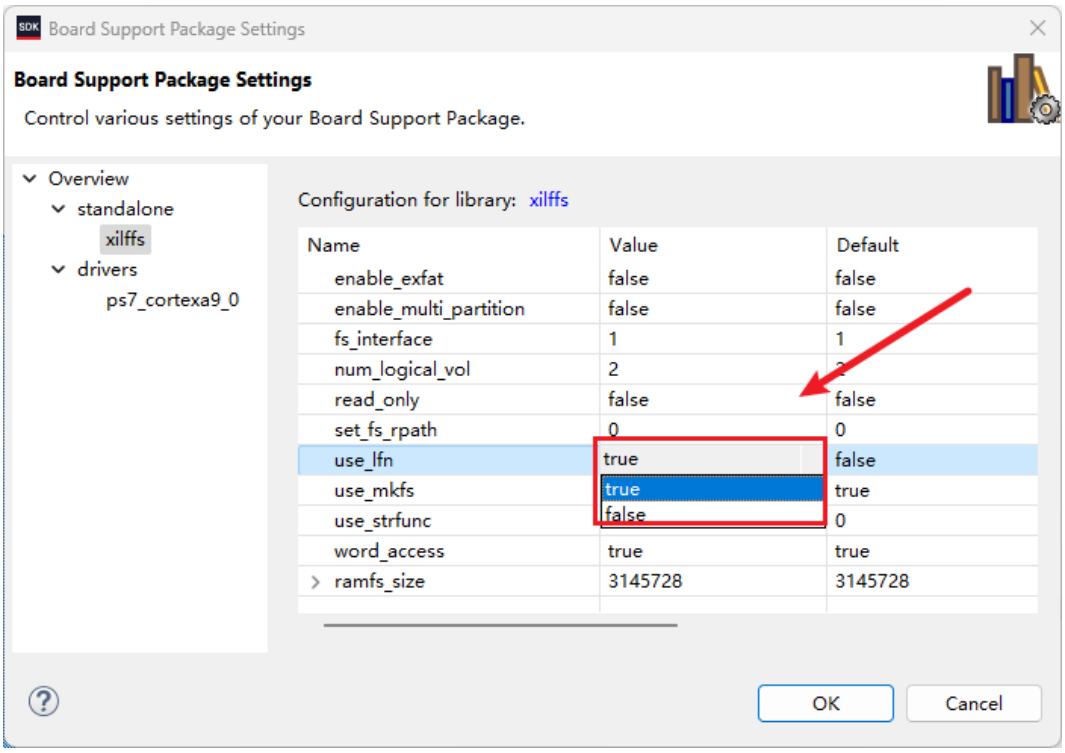

图 1-13 use_lfn 参数

1.5.3 添加用户代码

具体代码，可以参考例程，下面主要讲解说明：main.c

首先，通过 f_mount 函数，我们将 SD 卡挂载到文件系统中。如果挂载操作

失败，系统会返回相应的错误提示。

接下来，系统会提示用户输入新创建的文件名称。为了方便以后的操作，文件名称会自动添加".txt"后缀。然后，通过 f_open 函数，我们成功创建了一个新文件。

在此之后，系统再次提示用户输入想要写入的文本内容。同样的，我们通过 f_write 函数写入指定的内容至文件里。

此时，我们通过 f_lseek 函数将文件读写位置定位回文件的开头。然后，f_read 函数开始读取文件中的内容，并显示在屏幕上。

最后，我们使用 f_close 函数关闭文件，完成整个过程。（注意不要直接复制下列代码，可能出现格式错位）

```
#include "ff.h"
#include "stdio.h"
#include "string.h"

//在 SD 卡中创建一个文件，然后往文件里写入文本，最后读出文本内容
//本实验需要连接串口，输入文件名和写入的文本内容

FATFS fs;
FIL file;
FRESULT Res;

int main(void)
{
    char File_Name[128] = {0};
    char W_Str[128] = {0};
    char R_Str[128] = {0};

    //挂载 SD 卡
    Res = f_mount(&fs, "0:/", 1);
    if(Res != FR_OK) {
        //挂载失败
        printf("Mount failed to return a value of \"%d\" !\n",Res);
        return -1;
    }

    //输入创建文件的名称（不要带空格）
    printf("Please enter the name of the created file:\n");
    scanf("%s",File_Name);
    strcat(File_Name, ".txt");

    //创建文件
    f_open(&file, File_Name, FA_CREATE_ALWAYS | FA_WRITE | FA_READ);
```

```
    printf("The file named \"%s\" was created
         successfully!\n",File_Name);

    //输入想写进文件的内容（不要带空格）
    printf("Please enter what you want to write to:\n");
    scanf("%s",W_Str);

    //写文件
    f_write(&file, W_Str, strlen(W_Str), NULL);
    printf("\"%s\" was written to the \"%s\"
         file!\n",W_Str,File_Name);

    //定位到文件开头
    f_lseek (&file, 0);
    //读文件
    f_read (&file, R_Str, strlen(W_Str), NULL);
    printf("The document \"%s\" reads:\n %s \n",File_Name,R_Str);

    //关闭文件
    f_close (&file);

    return 0;
}
```

1.6 板级调试与验证

本次实验的板级验证阶段主要围绕以下任务进行：通过串口将文件内容保存到SD卡，并支持自定义命名。在完成sd卡写入后，会自动读取写入的内容，并打印到屏幕上。

系统所需硬件如下：

1. ACZ702 v2.0 开发板 x1
2. Type-c 线缆 x1
3. Micro SD 卡 x1

1.6.1 硬件连接

将 SD 卡插入到开发板中：

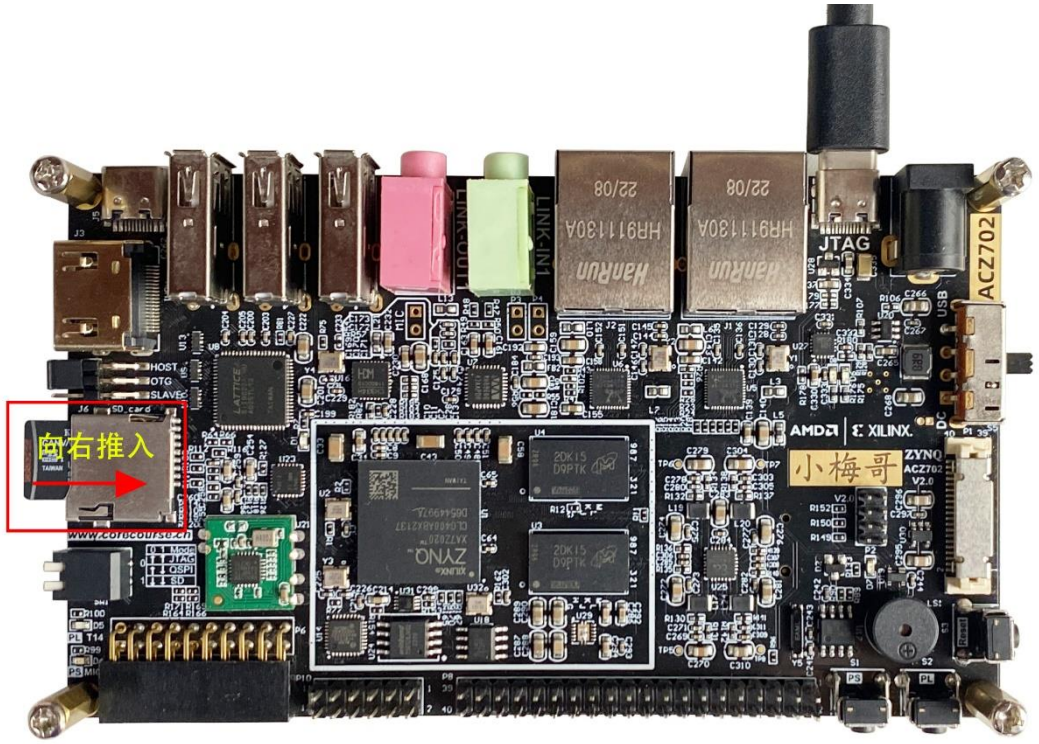

图 1-14 硬件连接图

设计对供电的要求不高，因此可以直接使用 type-c 供电。

1.6.2 下载验证

点击模板工程资源文件，将生成的烧录文件下载至开发板中如图 1-15 和图 1-16 所示，烧录流程如下：

1. 双击 GDB 或 System Debugger 新建配置任务，推荐使用 GDB。

2. 在右侧检查是否添加比特流文件和 PS 初始化脚本，通常软件会自动添加，如果没有，用户可以关闭并重新打开该界面或自己手动添加。

3. 确认下方三个选项都被勾选，这四个选项分别是系统复位、配置 FPGA、PS 初始化和 PL-PS 电平转换。后两个选项通常是默认勾选的，用户需要手动勾选前两项以确保 PL 部分能够被正确配置。

4. 点击上方的 Application 切换到应用界面。

5. 检查.elf 文件是否添加且烧录任务是否被选中。这里.elf 文件由软件编译产生，参与用户程序的运行。 SDK 默认情况下设置了自动编译，用户保存设计后软件便会编译并生成.elf 文件，所以当搜索不到.elf 文件时检查设计是否保存。

6. 点击“ Run”开始烧录

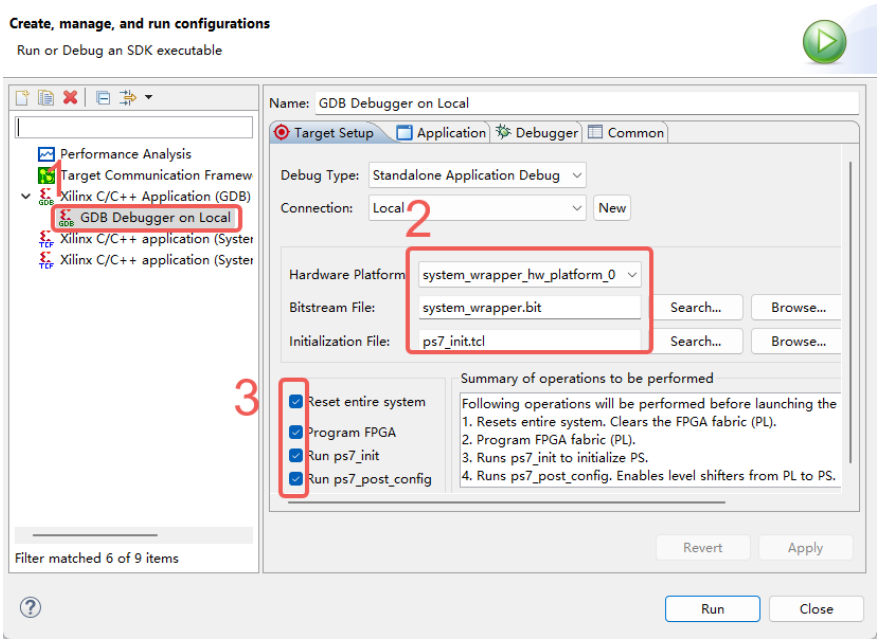

图 1-15 配置烧录任务

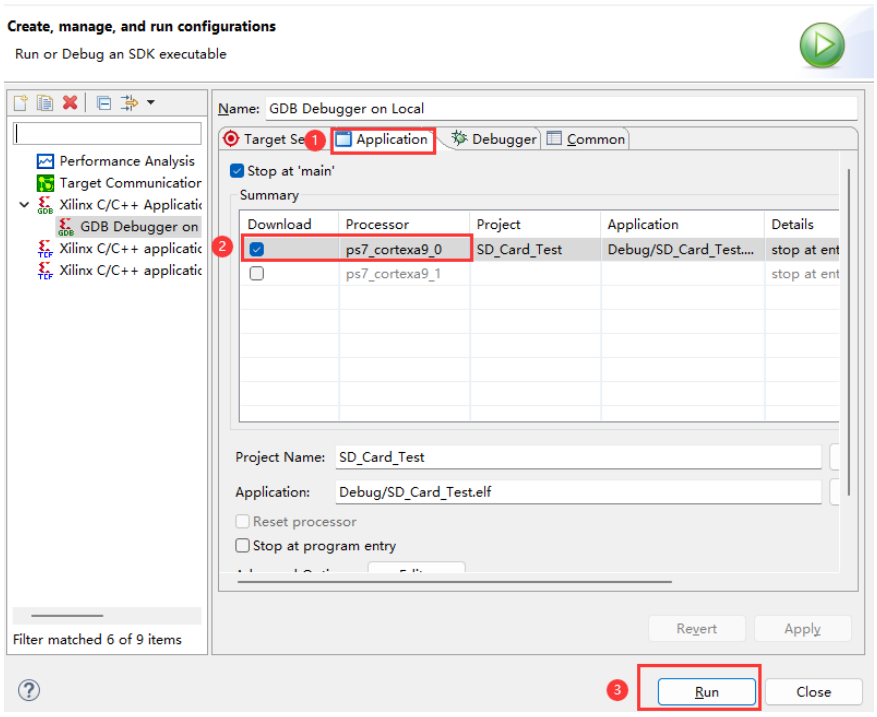

图 1-16 配置烧录任务

1.6.3 读写测试

打开工程，在 SDK 中运行程序。程序运行后会通过串口发送文本“Please enter the name of the created file:”

(1) 输入文件名

此时，回发数据，会被认为是文件命名操作；例如，要创建文件名 123.txt，那么可以通过串口软件发送“123”（注意：不要代空格）。

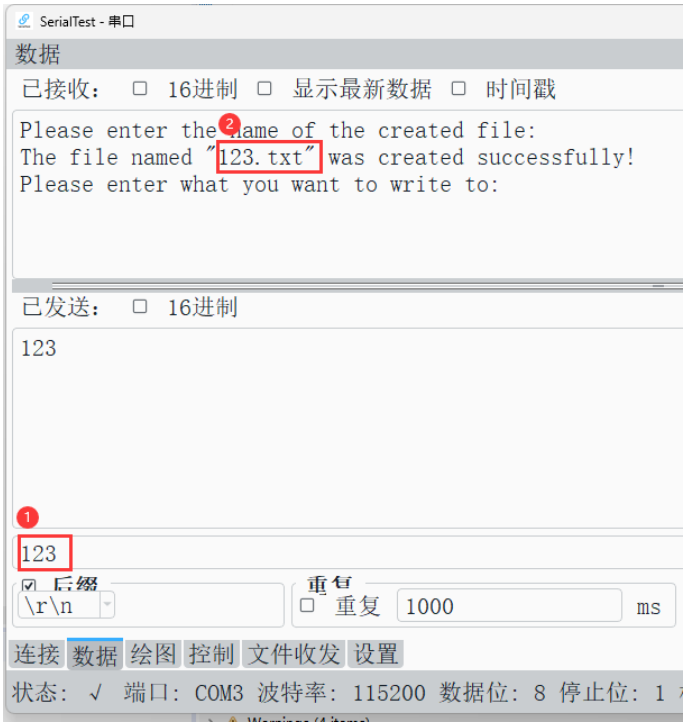

图 1-17 输入文件名

(2) 输入文件内容

如果出现提示“Please enter what you want to write to:”代表可以输入文件内容；如下图所示，输入“http://www.corecourse.cn/”，并回显打印，说明写入成功，已被保存到文件中。

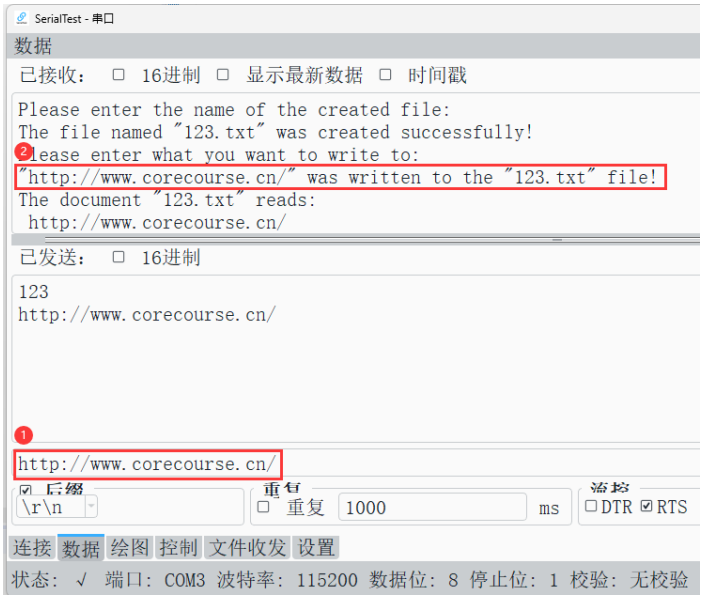

图 1-18 输入文件内容

(3) 读取文件内容

在发送文件内容后，如下图所示，会自动打印文件内容；如果打印的内容与输入的内容一致，说明文件读取测试成功。

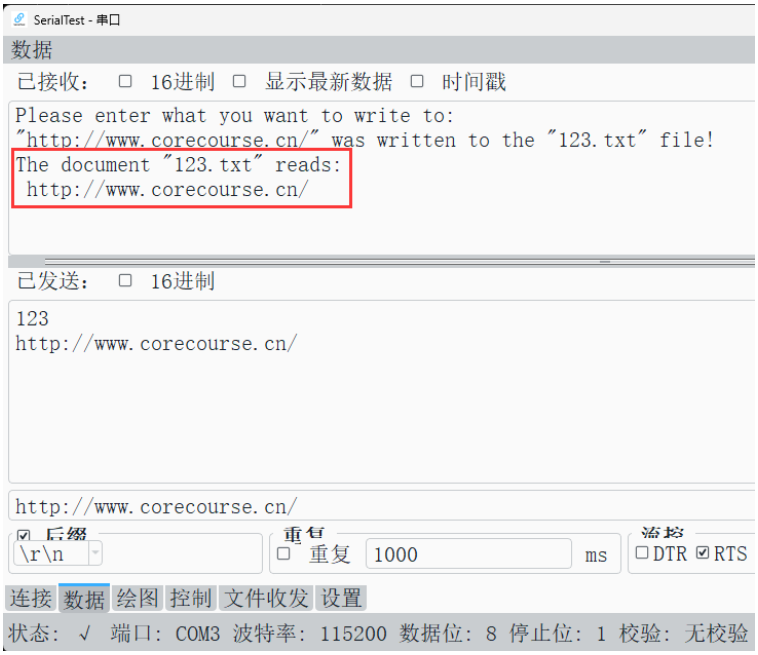

图 1-19 读取文件内容

(4) 文件检测

接下来从开发板上取下 SD 卡，并插入读卡器；然后连接到电脑 usb 上；如图 1-20 所示，此时可以看到该文件。

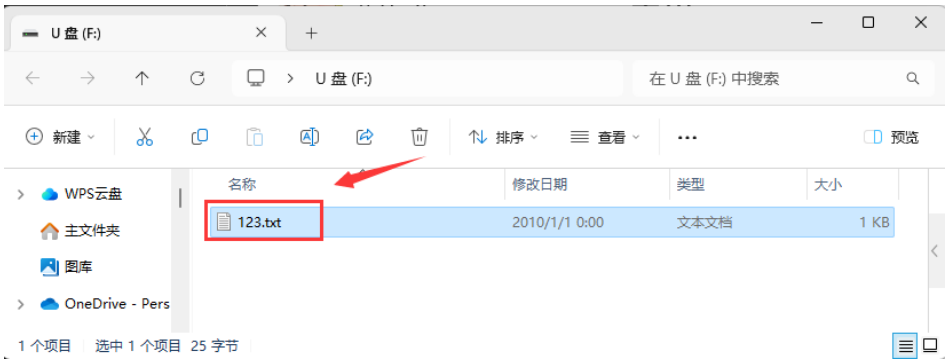

图 1-20 打开文件

继续打开“123.txt”文件，如图 1-21 所示，可以看到里面的内容，与刚刚发送的字符串一致。说明 SD 卡的读写测试实验成功。

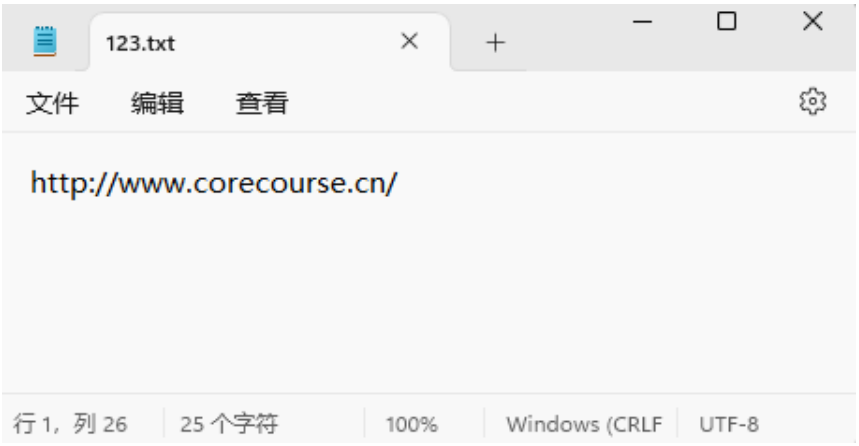

图 1-21 文件检测

1.7 总结

通过本次实验，我们深入理解并实践了 SD 卡的文本读写过程。从基本原理的探讨到具体实验步骤的执行，逐步实现了文本存储至 SD 卡，以及文本信息的读取过程。