南京理工大学

NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

计算机科学与工程学院

“嵌入式系统”实验报告书

题目：嵌入式系统作业 5——SPI

学号：922106840127
姓名：刘宇翔
成绩

日期：　　2025 年　　3 月　　30 日

1 题目要求

1. 题目设计要求

【1】作业目标：

（1）熟悉 SPI 工作原理；

（2）能够熟练 SPI 开发编程；

（3）能够使用 FLASH 进行数据读写。

【2】作业内容：

分别使用两种方法对开发板上的 W25Q64 进行读写，通过按键切换两种模式：

（1）按下 KEY_LEFT 键，处于 SPI 读写模式采用，SPI 部件对 W25Q64 的读写；

（2）按下 KEY_RIGHT 键，处于模拟 SPI 读写模式，采用 GPIO 管脚模拟 SPI 时序对 W25Q64 的读写；

以上两种方法均完成对 W25Q64 的初始化、数据写入和读出数据并开展校验，并分别计算出在读和写的速度（MB/s），并显示在 LCD 屏上。

【3】完成要求：

（1）两模式模式都要有，并且能自由切换，读写模式显示在 LCD 上，否则不给分；

（2）在不同模式下，都能对读写数据完成数据校验，并给出结果写 LCD 上；

（3）在不同模式下，分别完成读写速度测试，统计出读和写的速度结果写 LCD 上；

2. 拟实现的具体功能

本实验旨在实现两种 SPI 读写模式下对 W25Q64 FLASH 进行数据的初始化、写入、读取和校验，并在 LCD 屏上实时显示操作模式、数据校验结果以及读写速度。具体功能包括：

首先，通过硬件按键控制实现两种不同的读写模式切换。当用户按下 KEY_LEFT 按键时，系统进入基于 SPI 外设模块的读写模式，利用 MCU 内置 SPI 模块直接与 W25Q64 通信；当用户按下 KEY_RIGHT 按键时，系统则切换为利用 GPIO 管脚模拟 SPI 时序的方式进行 FLASH 操作。两种模式均能实时显示当前模式信息于 LCD 屏上，确保用户能够直观识别当前的操作状态。

其次，在每种模式下，系统首先对 W25Q64 进行初始化，配置相应的寄存器

和通信参数，确保 FLASH 处于可读写状态。接着系统进行数据写入操作，将预设的数据写入 FLASH 中，再进行数据读取操作以将写入的数据从 FLASH 中取出。读取的数据将与预先写入的数据进行逐一比对，实现数据校验功能。校验结果（成功或失败）会在 LCD 屏上明确显示，为后续调试和验证提供直观依据。

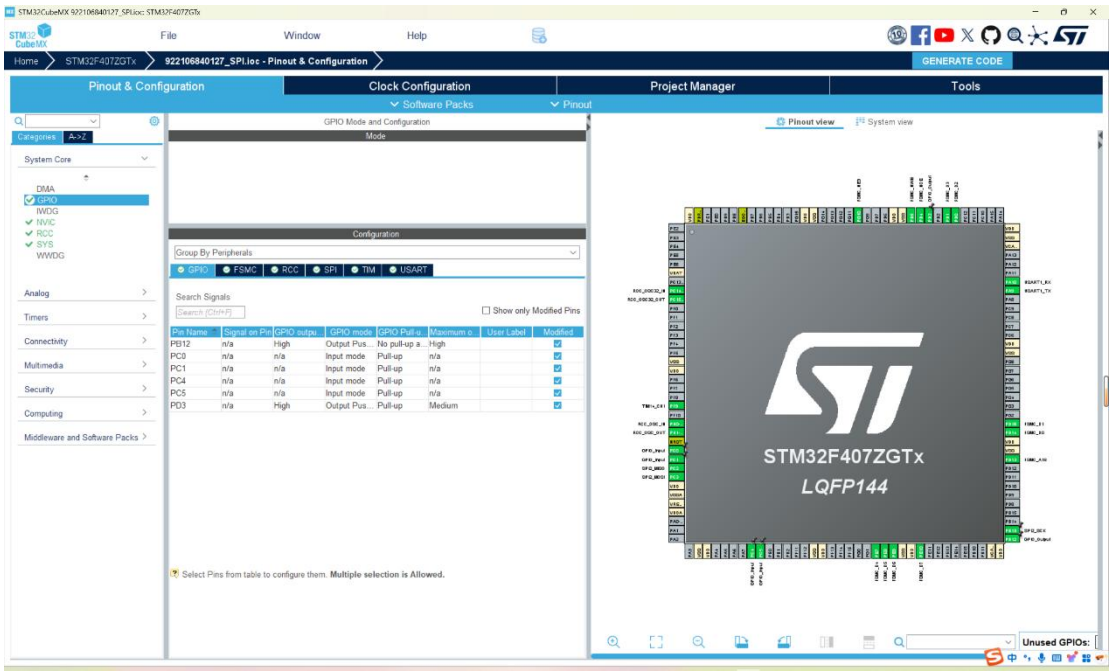
最后，系统在两种模式下还将分别进行读写速度测试。通过对数据传输过程计时，并计算出读写的传输速率（单位为 MB/s），这些测试数据同样会在 LCD 屏上实时更新显示，供用户参考比较。整体系统设计要求操作模式切换灵活、数据传输准确、校验有效、显示直观，以便用户能全面评估两种 SPI 实现方案的优缺点，并对 FLASH 存储器的性能有直观了解。

2 总体设计

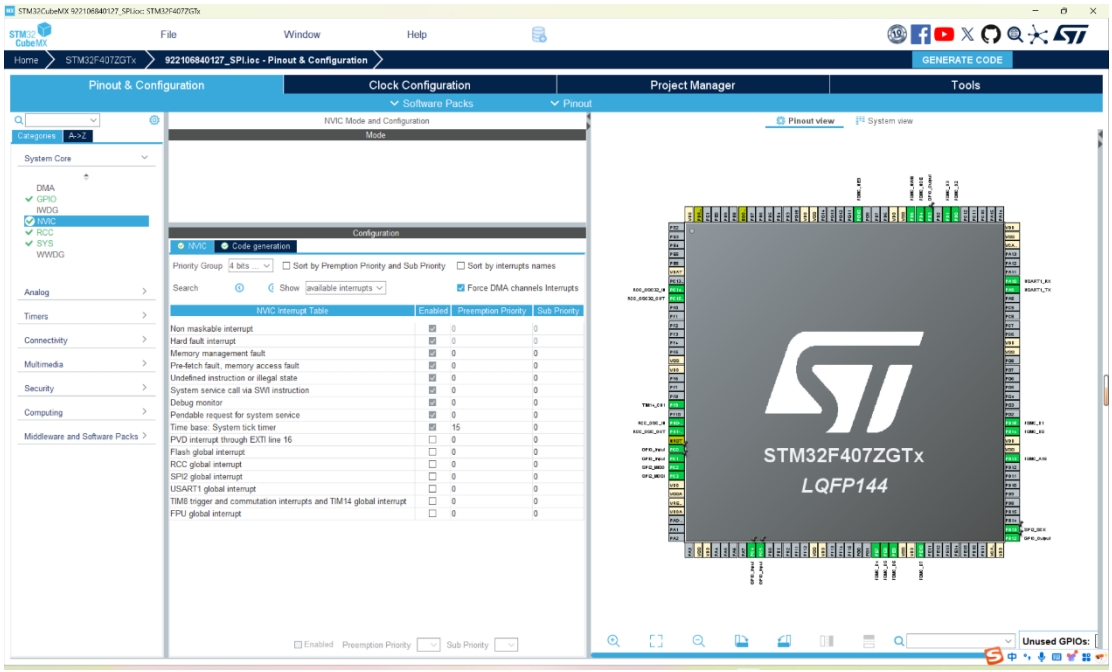
2.1 硬件设计

1.硬件设计思路

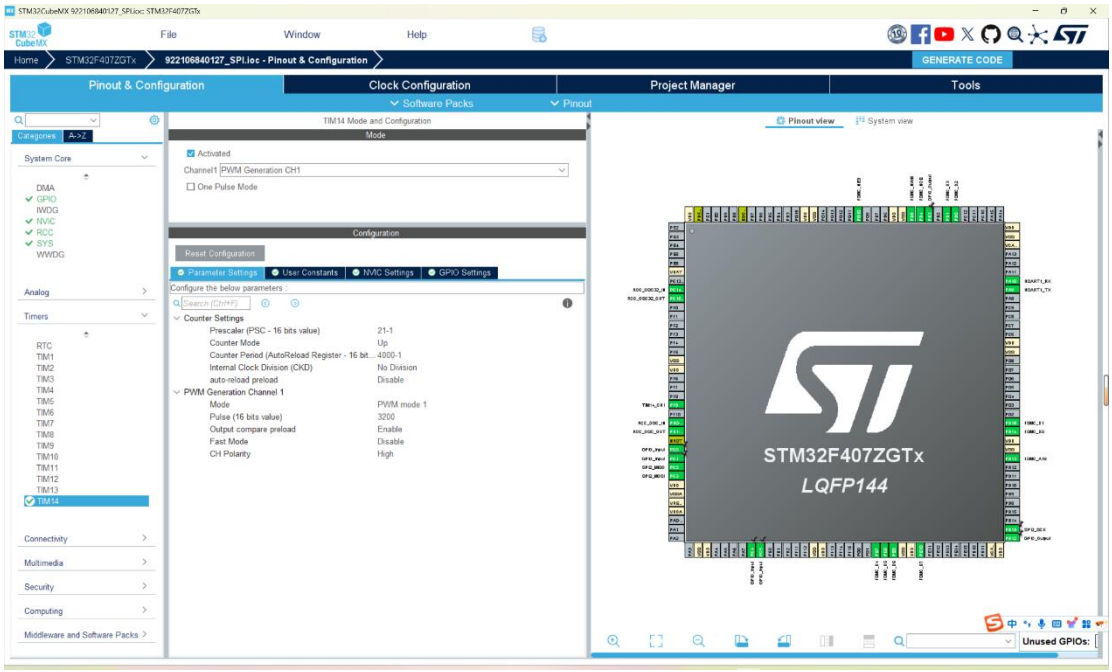
我通过查阅相关开发板原理构建硬件图和课本的参考指导确定了本次作业中会使用到的引脚与特殊设置的内容，例如 SPI,GPIO,USART 等相关设置，并对 LCD 的驱动进行了添加与相应的配置，完成整体的硬件设计，其中具体的代码工程配置如下：



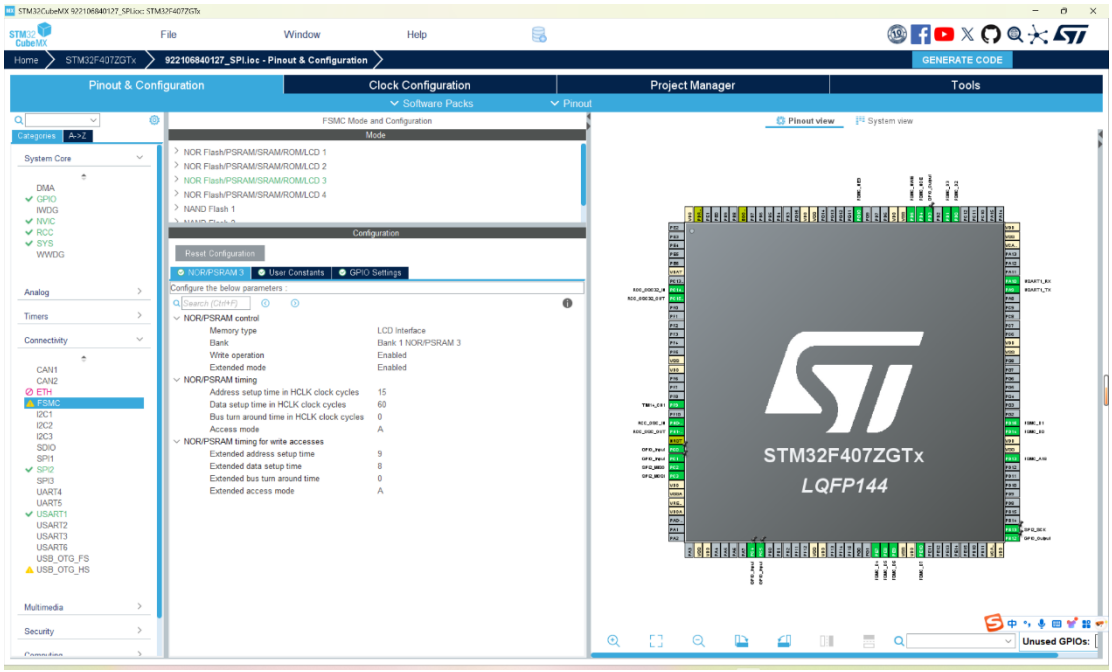
GPIO 端口的设置



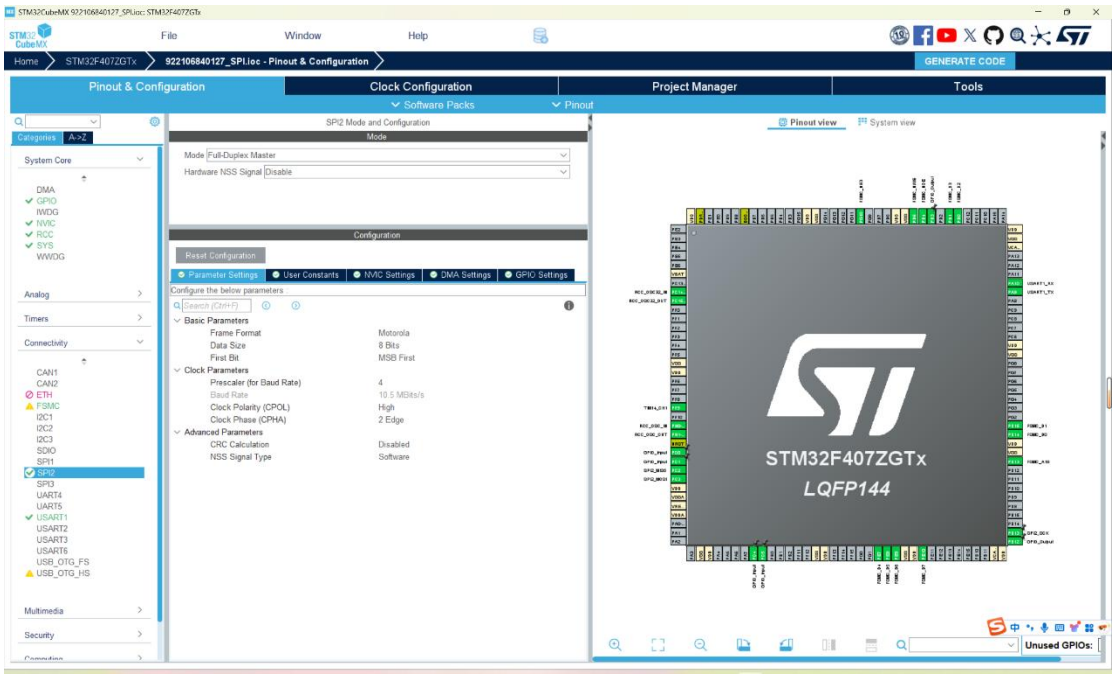
NVIC 的设置



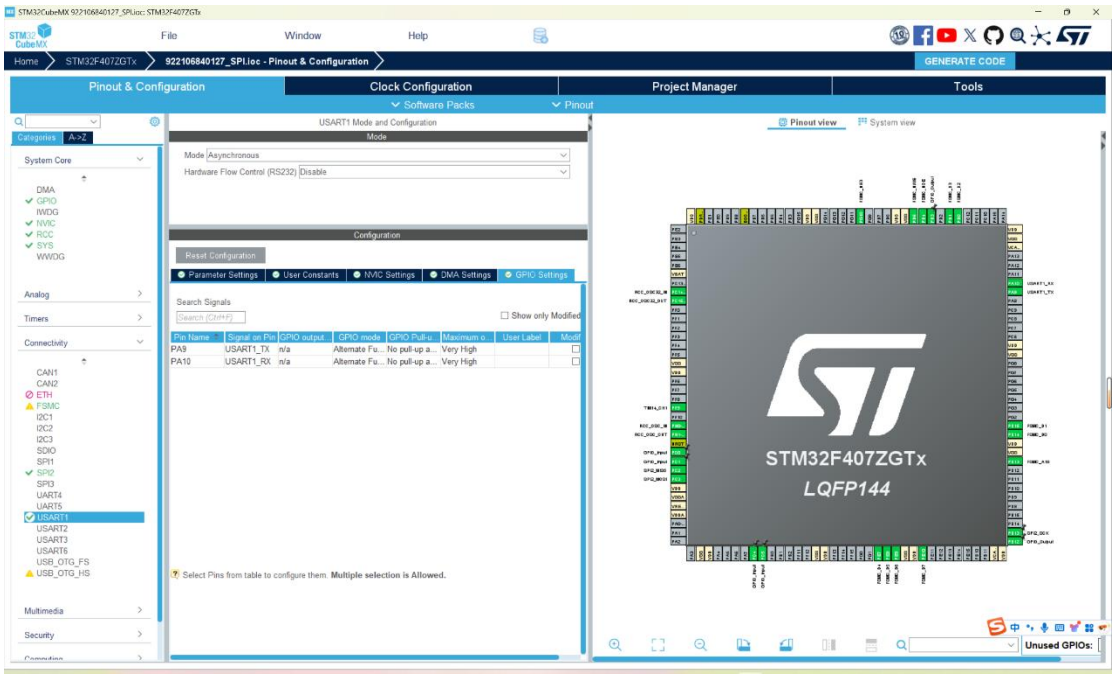
TIM14 的设置



FSMC_LCD 相关内容的设置（参考教材）



SPI 的数据传输相关内容的设置



USART 相关设置

此外，一些普遍的工程配置在本报告中不再展示（例如 RCC 时钟，整体工程的时钟配置，编译工具链选择 MDK-ARM V5.32 等）。

以上配置图即为我作为 STM32CUBEMX 进行的配置设置，设置后点击“Generate Code”生成初始化代码，继续进行接下来的代码软件设计部分。

2.2 软件设计

1．软件设计概述

本系统软件主要实现对 W25Q64 FLASH 的两种 SPI 读写方式（硬件 SPI 和 GPIO 模拟 SPI）的控制、数据传输、校验以及实时数据显示。软件设计采用直接操作外设寄存器和 HAL 库相结合的方法，整体分为以下几个模块：

（1） **系统初始化模块**

该模块负责系统整体的初始化工作，包括 MCU 初始化、系统时钟配置、GPIO、SPI、USART、FSMC 和定时器的初始化，同时完成 LCD 显示驱动的初始化。通过 FlashID 检测模块，系统确保 W25Q64 FLASH 正确连接并进入工作状态。

（2） **按键扫描及模式切换模块**

通过 Key_Read 函数检测按键状态，判断 KEY_LEFT 和 KEY_RIGHT 的按下情况，实现两种工作模式之间的自由切换。模块内通过软件消抖处理确保按键读

取的稳定性，并将模式切换信息及时反馈到主循环中，保证用户操作与系统响应的同步。

（3） **FLASH 操作及 SPI 读写模块**

该模块分别实现了基于硬件 SPI 和软件（GPIO 模拟）SPI 两种 FLASH 数据读写方式。硬件 SPI 模式调用标准 SPI 外设进行数据传输，而软件 SPI 模式则通过手动控制 GPIO 管脚生成 SPI 时序。两种模式均包含数据写入、读取、以及对数据完整性进行校验的功能，同时利用辅助函数计算校验码，确保数据传输的准确性。

（4） **数据传输计时与性能测试模块**

在读写过程中，通过 getCurrentMicros 等函数对传输过程计时，并计算出数据写入与读取的速率（KB/s）。测试结果作为性能指标，可用于对比硬件与软件两种 SPI 实现方式的效率。

（5） **LCD 显示与串口调试输出模块**

通过 LCD 驱动接口将当前模式、数据校验结果、读取数据和速度测试结果直观地显示在屏幕上；同时，利用 USART 串口调试功能，将关键操作的时间参数和传输速率通过串口输出，便于开发调试和系统性能评估。

总体来说，各模块协同工作，在有限硬件资源下实现了 FLASH 数据读写与校验、SPI 通信模式切换、性能测试和实时数据显示，保证了系统操作的稳定性和调试的便利性。

2. 软件流程分解

下面给出本实验的软件流程分解说明，其整体流程可分为以下几个阶段：

A. 初始阶段

开始 → 系统初始化

程序启动后首先进入系统初始化阶段。该阶段主要完成 MCU 和各外设的初始化工作，包括：

- 系统时钟配置、GPIO 端口初始化（用于按键、SPI、LCD 等外设）、USART 串口初始化、FSMC 及定时器初始化。
- LCD 显示驱动的初始化，确保显示模块可正常工作。
- FlashID 检测，循环检测 W25Q64 是否正确连接，直至读出正确的 ID 后

继续后续操作。

B. 主循环结构

进入主循环 → 持续检测按键状态

主循环中不断检测 KEY_LEFT 和 KEY_RIGHT 按键状态，判断是否切换 SPI 工作模式。每次检测均调用按键读取函数，并在检测到有效按键后设置标志位以便进行模式切换。

C. 模式选择与按键检测流程

- 读取按键状态：分别检测 KEY_LEFT 和 KEY_RIGHT 是否按下，按键按下后采用软件消抖处理，确保信号稳定。
- 模式判断：

  若检测到 KEY_LEFT 按下，则系统进入硬件 SPI 读写模式；

  若检测到 KEY_RIGHT 按下，则系统进入 GPIO 模拟 SPI 读写模式。
- 模式状态更新：通过标志位和变量记录当前工作模式，并及时刷新 LCD 显示当前模式信息。

D. FLASH 操作阶段

根据当前选择的模式，调用对应的 FLASH 操作任务函数（HD_SPI_Mode_Task 或 Soft_SPI_Mode_Task），实现以下步骤：

- **数据写入：**预设数据写入 Flash，同时记录写入开始和结束的时间。
- **数据读取：**从 Flash 中读取数据并记录读取时间。
- **数据校验：**计算写入数据和读取数据的校验码，对比校验结果，判断数据传输是否正确。

E. 数据处理与显示阶段

- 速度测试：根据写入和读取过程中所用的时间，计算出数据传输速率（KB/s 或 MB/s）。
- LCD 显示：将当前 SPI 工作模式、数据校验结果、实际读取数据以及读写速度测试结果实时显示在 LCD 屏上，便于用户直观观察。
- 串口输出：通过 USART 串口调试输出相关信息（如传输时间和速度），辅助调试和系统性能评估。

总体流程为：在系统初始化后进入主循环，不断检测按键并切换工作模式，在对

应模式下完成 FLASH 的读写、数据校验和速度测试，然后将所有结果实时显示和输出，最后返回主循环等待下一次按键触发。

3. 函数设计与功能实现

在本部分内容，我将选取部分关键函数，详细说明其设计思路和功能实现：

（1） Key_Read 函数

该函数用于检测按键状态，并通过简单的消抖处理确保按键输入的稳定性。函数内部分别检测 KEY_LEFT 和 KEY_RIGHT 对应的 GPIO 引脚电平，当检测到引脚为低电平时，首先调用 HAL_Delay 延时 50ms，再次确认按键状态，以防止因机械抖动产生误判。如果确认按键按下，则返回相应的键值，并设置 changed 标志，提示系统进行模式切换。此设计确保了按键触发的可靠性，同时避免重复触发。

（2） Soft_SPI_Transfer 函数

该函数是实现 GPIO 模拟 SPI 传输的核心，采用软件模拟方式完成数据位的串行传输。函数以 MSB 优先的方式依次处理 8 个位，首先根据 data 的最高位设置 MOSI 引脚的输出电平，然后通过短暂延时后，将 SCK 引脚拉高以产生时钟信号，再读取 MISO 引脚电平，将读取结果合并到 received 变量中，最后将 SCK 拉低，并左移 data 准备下一位传输。整个过程通过延时函数 Soft_SPI_Delay 控制时序，保证数据稳定传输。此函数的设计精髓在于通过软件精确模拟 SPI 时序，从而在硬件 SPI 不可用的情况下完成 FLASH 通信任务。

（3） HD_SPI_Mode_Task 与 Soft_SPI_Mode_Task 函数

这两个任务函数分别对应硬件 SPI 和软件（GPIO 模拟）SPI 的操作流程。它们实现了以下主要功能：

数据准备与写入：预设写入数据（例如连续字母），并通过 calculate_checksum 函数计算写入数据的校验码；同时记录写入操作的开始与结束时间，从而计算出写入的耗时。

数据读取与校验：在写入后延时等待 FLASH 完成内部操作，再次记录读取操作的时间区间，通过调用对应的读数据函数（W25QXX_Read 或 Soft_W25QXX_Read）获取 FLASH 中的数据，并计算读取数据的校验码，对比写入和读取数据是否一致。

性能测试与显示：根据计时结果计算数据传输速率，并将模式名称、数据校验结果、实际读出的数据以及读写速率显示到 LCD 屏上，同时通过 USART 串口输出调试信息。

这两个函数设计上保持了操作流程的一致性，仅在调用的底层 SPI 操作接口上有所区别，使得系统能够直观比较硬件 SPI 与软件 SPI 两种实现方式的性能差异。

（4） calculate_checksum 函数

该函数用于计算数据数组的校验和，通过遍历每个字节并进行累加，得到一个简单的 8 位校验码。此校验码在写入与读取操作后进行比对，以判断数据在传输过程中是否发生错误。设计上采用简单高效的累加算法，既节省资源又能满足校验需求。

通过上述关键函数的设计与实现，系统能够完成按键检测、SPI 数据传输、FLASH 读写校验以及性能测试等核心功能，并在 LCD 和串口上直观展示测试结果，保证了整个系统的可靠性和实时性。

4. μ vision 详细代码

```
#include "main.h"
#include "spi.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"
#include "fsmc.h"
#include "flash.h"
#include "stdio.h"
#include "./BSP/LCD/drv_lcd.h"
#include "./BSP/LCD/rttlogo.h"

#define BUFFER_SIZE 27
#define KEY_LEFT_PIN GPIO_PIN_0
#define KEY_RIGHT_PIN GPIO_PIN_4
#define KEY_PORT GPIOC

#define SCK_PIN GPIO_PIN_5
#define SCK_PORT GPIOA
#define MOSI_PIN GPIO_PIN_7
#define MOSI_PORT GPIOA
```

```
#define MISO_PIN GPIO_PIN_6
#define MISO_PORT GPIOA
#define CS_PIN GPIO_PIN_12
#define CS_PORT GPIOB

#define W25Q64_CMD_WRITE_ENABLE   0x06
#define W25Q64_CMD_READ_STATUS    0x05
#define W25Q64_CMD_READ_DATA      0x03
#define W25Q64_CMD_PAGE_PROGRAM   0x02

uint16_t    FlashID = 0;
uint8_t     WriteBuf[BUFFER_SIZE]="",ReadBuf[BUFFER_SIZE]="";
uint8_t write_checksum, read_checksum;
uint8_t current_mode = 0;
uint8_t changed=1;

void SystemClock_Config(void);

int fputc(int ch, FILE *f)
{
  HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 1000);
  return ch;
}

uint8_t calculate_checksum(uint8_t *data, uint16_t length) {
    uint8_t checksum = 0;
    for (uint16_t i = 0; i < length; i++) {
        checksum += data[i];
    }
    return checksum;
}

uint32_t get_time(void) {
    return HAL_GetTick();
}

uint32_t getCurrentMicros(void)
{
  uint32_t m0 = HAL_GetTick();
  __IO uint32_t u0 = SysTick->VAL;
  uint32_t m1 = HAL_GetTick();
  __IO uint32_t u1 = SysTick->VAL;
  const uint32_t tms = SysTick->LOAD + 1;

```

```
65.  if (m1 != m0) {
66.    return (m1 * 1000 + ((tms - u1) * 1000) / tms);
67.  } else {
68.    return (m0 * 1000 + ((tms - u0) * 1000) / tms);
69.  }
70.}
71.

72.uint8_t Key_Read(void) {
73.    uint8_t key = 0;
74.    if (HAL_GPIO_ReadPin(KEY_PORT, KEY_LEFT_PIN) == GPIO_PIN_RESET) {
75.        HAL_Delay(50); // ???
76.        if (HAL_GPIO_ReadPin(KEY_PORT, KEY_LEFT_PIN) ==
   GPIO_PIN_RESET) {
77.            key = 0;
78.          changed=1;
79.        }
80.    } else if (HAL_GPIO_ReadPin(KEY_PORT, KEY_RIGHT_PIN) ==
   GPIO_PIN_RESET) {
81.        HAL_Delay(50); // ???
82.        if (HAL_GPIO_ReadPin(KEY_PORT, KEY_RIGHT_PIN) ==
   GPIO_PIN_RESET) {
83.            key = 1;
84.          changed=1;
85.        }
86.    }
87.    return key;
88.}
89.
90.void Soft_SPI_Init(void)
91.{
92.    GPIO_InitTypeDef GPIO_InitStruct = {0};
93.
94.    /* ??GPIOA?GPIOB?? */
95.    __HAL_RCC_GPIOA_CLK_ENABLE();
96.    __HAL_RCC_GPIOB_CLK_ENABLE();  // CS????
97.
98.    /* ??SCK(PA5):???? */
99.    GPIO_InitStruct.Pin = SCK_PIN;
100.       GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
101.       GPIO_InitStruct.Pull = GPIO_NOPULL;
102.       GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
103.       HAL_GPIO_Init(SCK_PORT, &GPIO_InitStruct);
104.       HAL_GPIO_WritePin(SCK_PORT, SCK_PIN, GPIO_PIN_RESET);
105.
```

```
106.        /* ??MOSI(PA7):???? */
107.        GPIO_InitStruct.Pin = MOSI_PIN;
108.        HAL_GPIO_Init(MOSI_PORT, &GPIO_InitStruct);
109.        HAL_GPIO_WritePin(MOSI_PORT, MOSI_PIN, GPIO_PIN_RESET);
110.
111.        /* ??MISO(PA6):?? */
112.        GPIO_InitStruct.Pin = MISO_PIN;
113.        GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
114.        GPIO_InitStruct.Pull = GPIO_NOPULL;
115.        HAL_GPIO_Init(MISO_PORT, &GPIO_InitStruct);
116.    }
117.
118.    /**
119.      * @brief  ????,????SPI??,??1??
120.      */
121.    static void Soft_SPI_Delay(void)
122.    {
123.        HAL_Delay(1);
124.    }
125.
126.    /**
127.      * @brief  ??SPI??1????(MSB??),????????
128.      * @param  data ??????
129.      * @retval ??????
130.      */
131.    uint8_t Soft_SPI_Transfer(uint8_t data)
132.    {
133.        uint8_t received = 0;
134.        for (uint8_t i = 0; i < 8; i++)
135.        {
136.            /* ??data????MOSI */
137.            if(data & 0x80)
138.                HAL_GPIO_WritePin(MOSI_PORT, MOSI_PIN, GPIO_PIN_SET);
139.            else
140.                HAL_GPIO_WritePin(MOSI_PORT, MOSI_PIN,
    GPIO_PIN_RESET);
141.
142.            Soft_SPI_Delay();
143.
144.            /* SCK??? */
145.            HAL_GPIO_WritePin(SCK_PORT, SCK_PIN, GPIO_PIN_SET);
146.            Soft_SPI_Delay();
147.
148.            /* ??MISO?? */
```

```
149.            received <<= 1;
150.            if(HAL_GPIO_ReadPin(MISO_PORT, MISO_PIN) == GPIO_PIN_SET)
151.                received |= 0x01;
152.
153.            /* SCK??? */
154.            HAL_GPIO_WritePin(SCK_PORT, SCK_PIN, GPIO_PIN_RESET);
155.            Soft_SPI_Delay();
156.
157.            data <<= 1;
158.        }
159.        return received;
160.    }
161.
162.    /**
163.      * @brief  ????????W25Q64
164.      */
165.    void Soft_W25Q64_WriteEnable(void)
166.    {
167.        HAL_GPIO_WritePin(CS_PORT, CS_PIN, GPIO_PIN_RESET);
168.        Soft_SPI_Transfer(W25Q64_CMD_WRITE_ENABLE);
169.        HAL_GPIO_WritePin(CS_PORT, CS_PIN, GPIO_PIN_SET);
170.    }
171.
172.    /**
173.      * @brief  ??W25Q64?????
174.      * @retval ???????
175.      */
176.    uint8_t Soft_W25Q64_ReadStatus(void)
177.    {
178.        uint8_t status;
179.        HAL_GPIO_WritePin(CS_PORT, CS_PIN, GPIO_PIN_RESET);
180.        Soft_SPI_Transfer(W25Q64_CMD_READ_STATUS);
181.        status = Soft_SPI_Transfer(0xFF);  // ???????????
182.        HAL_GPIO_WritePin(CS_PORT, CS_PIN, GPIO_PIN_SET);
183.        return status;
184.    }
185.
186.    /**
187.      * @brief  ?W25Q64????
188.      * @param  addr ????(24?)
189.      * @param  buf ??????????
190.      * @param  len ??????
191.      */
```

```
192.    void Soft_W25Q64_ReadData(uint32_t addr, uint8_t *buf, uint16_t
   len)
193.    {
194.        HAL_GPIO_WritePin(CS_PORT, CS_PIN, GPIO_PIN_RESET);
195.        Soft_SPI_Transfer(W25Q64_CMD_READ_DATA);
196.        Soft_SPI_Transfer((addr >> 16) & 0xFF);
197.        Soft_SPI_Transfer((addr >> 8) & 0xFF);
198.        Soft_SPI_Transfer(addr & 0xFF);
199.        for(uint16_t i = 0; i < len; i++)
200.        {
201.            buf[i] = Soft_SPI_Transfer(0xFF);
202.        }
203.        HAL_GPIO_WritePin(CS_PORT, CS_PIN, GPIO_PIN_SET);
204.    }
205.
206.    /**
207.      * @brief  ?W25Q64???????
208.      * @param  addr ????(24?)
209.      * @param  buf ?????????
210.      * @param  len ?????(????????,???256??)
211.      */
212.    void Soft_W25Q64_PageProgram(uint32_t addr, uint8_t *buf,
   uint16_t len)
213.    {
214.        Soft_W25Q64_WriteEnable();
215.
216.        HAL_GPIO_WritePin(CS_PORT, CS_PIN, GPIO_PIN_RESET);
217.        Soft_SPI_Transfer(W25Q64_CMD_PAGE_PROGRAM);
218.        Soft_SPI_Transfer((addr >> 16) & 0xFF);
219.        Soft_SPI_Transfer((addr >> 8) & 0xFF);
220.        Soft_SPI_Transfer(addr & 0xFF);
221.
222.        for(uint16_t i = 0; i < len; i++)
223.        {
224.            Soft_SPI_Transfer(buf[i]);
225.        }
226.        HAL_GPIO_WritePin(CS_PORT, CS_PIN, GPIO_PIN_SET);
227.        while(Soft_W25Q64_ReadStatus() & 0x01)
228.        {
229.            Soft_SPI_Delay();
230.        }
231.    }
232.
```

```
233.    void Soft_W25QXX_Write(uint8_t *buf, uint32_t addr, uint16_t
   len)
234.    {
235.        Soft_W25Q64_PageProgram(addr, buf, len);
236.    }
237.
238.    void Soft_W25QXX_Read(uint8_t *buf, uint32_t addr, uint16_t len)
239.    {
240.        Soft_W25Q64_ReadData(addr, buf, len);
241.    }
242.
243.    void HD_SPI_Mode_Task(void){
244.
245.      uint32_t i;
246.      lcd_clear(WHITE);
247.          lcd_show_string(15, 15, 32,"HD SPI Mode");
248.
249.          //write
250.          for(i=0;i<26;i++) WriteBuf[i]=(i)%26+'A';
251.          WriteBuf[BUFFER_SIZE - 1] = '\0';
252.      write_checksum = calculate_checksum(WriteBuf, BUFFER_SIZE);
253.
254.      uint32_t write_start_time = getCurrentMicros();
255.      W25QXX_Write(WriteBuf,0,27);
256.      uint32_t write_end_time = getCurrentMicros();
257.      uint32_t write_duration = write_end_time - write_start_time;
258.
259.      printf("Write duration: %u us\n", write_duration);
260.
261.      HAL_Delay(1000);
262.      //read
263.      uint32_t read_start_time = getCurrentMicros();
264.      W25QXX_Read(ReadBuf,0,27);
265.      uint32_t read_end_time = getCurrentMicros();
266.      uint32_t read_duration = read_end_time - read_start_time;
267.
268.      printf("Read duration: %u us\n", read_duration);
269.
270.      HAL_Delay(1000);
271.      ReadBuf[BUFFER_SIZE - 1] = '\0';
272.
273.      read_checksum = calculate_checksum(ReadBuf, BUFFER_SIZE);
274.      //check
275.      if (write_checksum == read_checksum) {
```

```
276.           //LCD_ShowString(10, 139, 16, (uint8_t *)"Data Check
   Passed");
277.         lcd_show_string(20, 55, 16, "Data Check Passed");
278.       } else {
279.           //LCD_ShowString(10, 139, 16, (uint8_t *)"Data Check
   Failed");
280.         lcd_show_string(20, 55, 16, "Data Check Failed");
281.       }
282.
283.      lcd_show_string(20, 75, 16,"Read Data:");
284.      lcd_show_string(20, 95, 16,(const char *)ReadBuf);
285.       //speed
286.       float write_speed = (float)BUFFER_SIZE*1000 /
   write_duration ;
287.       float read_speed = (float)BUFFER_SIZE*1000 / read_duration ;
288.
289.       char write_speed_str[20];
290.       char read_speed_str[20];
291.       sprintf(write_speed_str, "Write: %.2f KB/s", write_speed);
292.       sprintf(read_speed_str, "Read: %.2f KB/s", read_speed);
293.
294.       lcd_show_string(20, 120, 16,"Speed showing activate:");
295.       lcd_show_string(20, 140, 16,write_speed_str);
296.       lcd_show_string(20, 170, 16,read_speed_str);
297.       lcd_show_string(20,200,16,"922106840127_SPI");
298.
299.       //printf
300.       printf("Write Speed: %.2f KB/s\n", write_speed);
301.       printf("Read Speed: %.2f KB/s\n", read_speed);
302.    }
303.
304.    void Soft_SPI_Mode_Task(void)
305.    {
306.        uint32_t i;
307.       lcd_clear(0xFFFF);  // ??WHITE????0xFFFF
308.       lcd_show_string(15, 15, 32, "GPIO SPI Mode");
309.       for(i = 0; i < 26; i++)
310.       {
311.           WriteBuf[i] = (i % 26) + 'A';
312.       }
313.       WriteBuf[BUFFER_SIZE - 1] = '\0';
314.       write_checksum = calculate_checksum(WriteBuf, BUFFER_SIZE);
315.       uint32_t write_start_time = getCurrentMicros();
316.       Soft_W25QXX_Write(WriteBuf, 0, 27);
```

```
317.
318.        uint32_t write_end_time = getCurrentMicros();
319.        uint32_t write_duration = write_end_time - write_start_time;
320.        printf("Write duration: %u us\n", write_duration);
321.        HAL_Delay(1000);
322.
323.        uint32_t read_start_time = getCurrentMicros();
324.        Soft_W25QXX_Read(ReadBuf, 0, 27);
325.        uint32_t read_end_time = getCurrentMicros();
326.        uint32_t read_duration = read_end_time - read_start_time;
327.        printf("Read duration: %u us\n", read_duration);
328.
329.        HAL_Delay(1000);
330.        ReadBuf[BUFFER_SIZE - 1] = '\0';
331.        read_checksum = calculate_checksum(ReadBuf, BUFFER_SIZE);
332.
333.        if (write_checksum == read_checksum)
334.        {
335.            lcd_show_string(20, 55, 16, "Data Check Passed");
336.        }
337.        else
338.        {
339.            lcd_show_string(20, 55, 16, "Data Check Failed");
340.        }
341.
342.        lcd_show_string(20, 75, 16, "Read Data:");
343.        lcd_show_string(20, 95, 16, (const char *)ReadBuf);
344.
345.        float write_speed = (float)BUFFER_SIZE * 1000 /
    write_duration;
346.        float read_speed  = (float)BUFFER_SIZE * 1000 /
    read_duration;
347.        char write_speed_str[20];
348.        char read_speed_str[20];
349.        sprintf(write_speed_str, "Write: %.2f KB/s", write_speed);
350.        sprintf(read_speed_str, "Read: %.2f KB/s", read_speed);
351.        lcd_show_string(20, 120, 16, "Speed:");
352.        lcd_show_string(20, 140, 16, write_speed_str);
353.        lcd_show_string(20, 160, 16, read_speed_str);
354.        lcd_show_string(20,200,16,"922106840127_SPI");
355.
356.        printf("Write Speed: %.2f KB/s\n", write_speed);
357.        printf("Read Speed: %.2f KB/s\n", read_speed);
358.    }
```

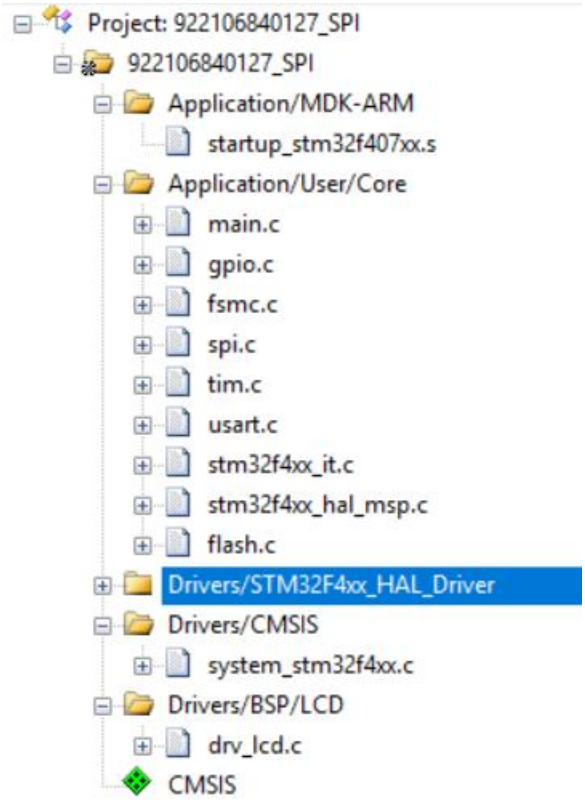
```
359.
360.    int main(void)
361.    {
362.      HAL_Init();
363.      SystemClock_Config();
364.
365.      MX_GPIO_Init();
366.      MX_SPI2_Init();
367.      MX_USART1_UART_Init();
368.
369.      MX_FSMC_Init();
370.      MX_TIM14_Init();
371.
372.      HAL_TIM_Base_Start_IT(&htim14);
373.      HAL_TIM_PWM_Start(&htim14, TIM_CHANNEL_1);
374.
375.      drv_lcd_init();
376.
377.      lcd_clear(WHITE);
378.      lcd_set_color(WHITE, BLACK);
379.
380.      FlashID = W25QXX_ReadID();
381.      while((FlashID = W25QXX_ReadID()) != W25Q64)
382.      {
383.        printf("W25Q128 Check Failed!!!!!!!!!! FlashID = 0x%X\n",
    FlashID); // ??????FlashID
384.        HAL_Delay(1000);
385.        FlashID = W25QXX_ReadID();
386.      }
387.      printf("Flash_ID = 0x%X\n", FlashID);
388.      while (1)
389.      {
390.        uint8_t key = Key_Read();
391.        if(changed==1 && key==0){
392.       HD_SPI_Mode_Task();
393.       changed=0;
394.        }else if(changed==1 && key==1){
395.          Soft_SPI_Mode_Task();
396.          changed = 0;
397.        }
398.      }
399.    }
400.    /**
401.      * @brief System Clock Configuration
```

```
402.      * @retval None
403.      */
404.    void SystemClock_Config(void)
405.    {
406.      RCC_OscInitTypeDef RCC_OscInitStruct = {0};
407.      RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
408.
409.      /** Configure the main internal regulator output voltage
410.      */
411.      __HAL_RCC_PWR_CLK_ENABLE();
412.      __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
413.
414.      /** Initializes the RCC Oscillators according to the specified
    parameters
415.      * in the RCC_OscInitTypeDef structure.
416.      */
417.      RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
418.      RCC_OscInitStruct.HSEState = RCC_HSE_ON;
419.      RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
420.      RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
421.      RCC_OscInitStruct.PLL.PLLM = 4;
422.      RCC_OscInitStruct.PLL.PLLN = 168;
423.      RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
424.      RCC_OscInitStruct.PLL.PLLQ = 4;
425.      if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
426.      {
427.        Error_Handler();
428.      }
429.
430.      /** Initializes the CPU, AHB and APB buses clocks
431.      */
432.      RCC_ClkInitStruct.ClockType =
    RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
433.                                    |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_P
    CLK2;
434.      RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
435.      RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
436.      RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
437.      RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
438.
439.      if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
    FLASH_LATENCY_5) != HAL_OK)
440.      {
441.        Error_Handler();
```

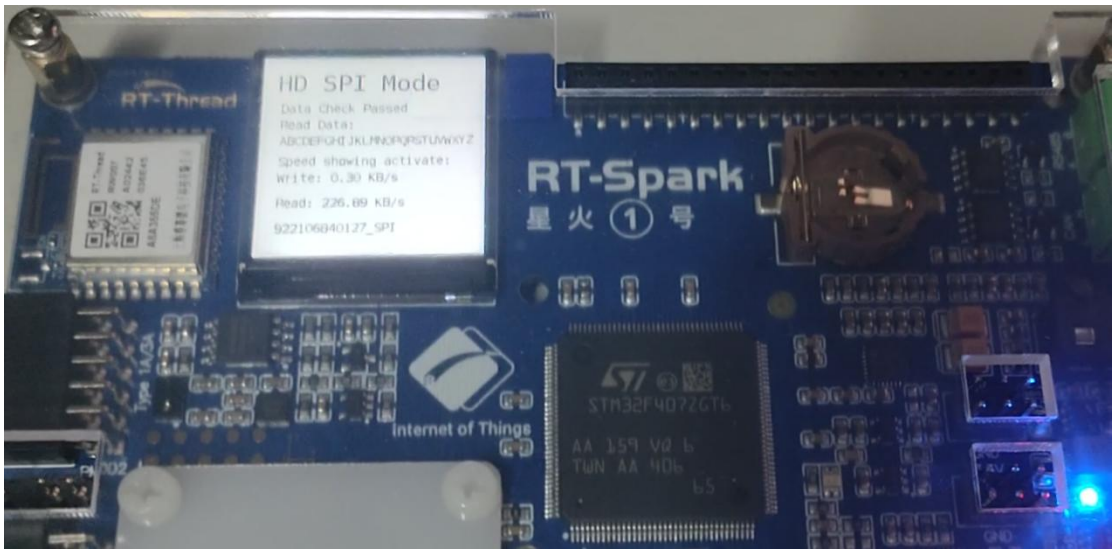
```
442.      }
443.    }
444.
445.    /* USER CODE BEGIN 4 */
446.
447.    /* USER CODE END 4 */
448.
449.    /**
450.      * @brief  This function is executed in case of error
   occurrence.
451.      * @retval None
452.      */
453.    void Error_Handler(void)
454.    {
455.      /* USER CODE BEGIN Error_Handler_Debug */
456.      /* User can add his own implementation to report the HAL error
   return state */
457.      __disable_irq();
458.      while (1)
459.      {
460.      }
461.      /* USER CODE END Error_Handler_Debug */
462.    }
463.
464.    #ifdef  USE_FULL_ASSERT
465.    /**
466.      * @brief  Reports the name of the source file and the source
   line number
467.      *         where the assert_param error has occurred.
468.      * @param  file: pointer to the source file name
469.      * @param  line: assert_param error line source number
470.      * @retval None
471.      */
472.    void assert_failed(uint8_t *file, uint32_t line)
473.    { }
474.    #endif /* USE_FULL_ASSERT */
475.
```

3 实验结果分析与总结

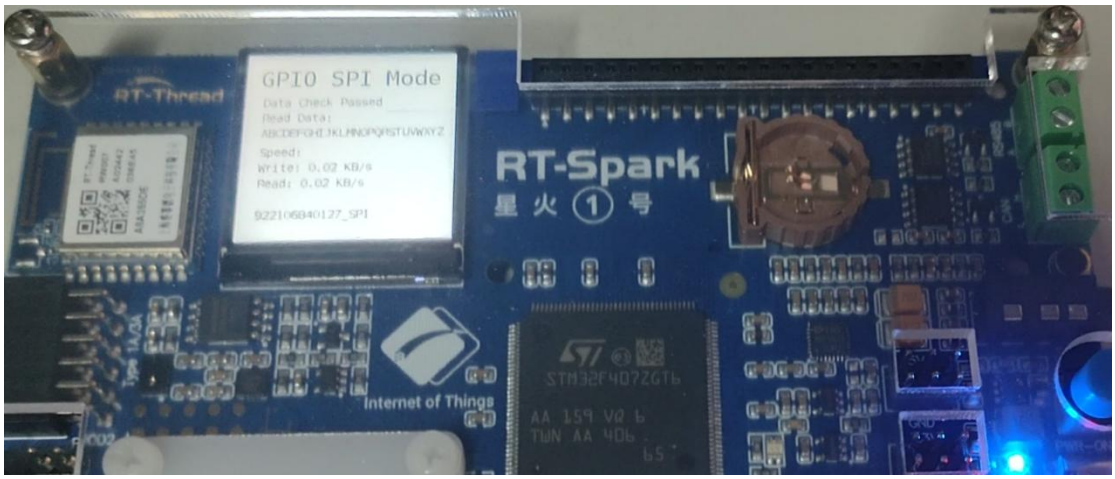
我经过对项目工程进行编译之后无报错，进行上板测试。以下为我的项目工程的文件结构：



本项目添加了点亮 LCD 所需的库函数与头文件，例如 Drivers/BSP/LCD 文件夹中存放的 drv_lcd.c 即为使用到的 LCD 驱动，通过软件编译之后我进行了上板测试，验证我的实验成果。



硬件模式下 SPI 的性能指标



软件模式（GPIO）模式下的性能指标

实验测试结果显示，在硬件 SPI 模式下，读速度高达 226.89 kB/s，而写速度仅为 0.30 kB/s；相比之下，软件 SPI 模式的读写速度均约为 0.20 kB/s。这一对比表明，硬件 SPI 利用内置外设实现高速数据读取，明显优于软件模拟模式；但在写操作方面，两者均因 FLASH 内部编程时序限制而较低，且硬件模式写速度虽稍高但仍处于低速水平。软件 SPI 模式由于额外的 GPIO 控制和延时，整体传输速率受到严重制约。总体而言，硬件模式在数据读取上具有明显优势，适合大批量数据高速读取，而写操作性能则受 FLASH 器件自身影响，两种模式均能确保数据传输可靠，但在性能要求较高的应用中，建议优先采用硬件 SPI 方案，并针对写入流程进行进一步优化。