Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по дисциплине «Микропроцессорные системы»
на тему:

RAID-массив

Студент _____ _____ Д.А. Шестаков
(Группа) (Подпись, дата) (И.О. Фамилия)

Руководитель _____ И.Б. Трамов
(Подпись, дата) (И.О. Фамилия)

2023 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ИУ6
_____ А.В. Пролетарский
«2» сентября 2023 г.

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине Микропроцессорные системы

Студент группы ИУ6-72Б

Шестаков Даниил Алексеевич

Тема курсовой работы: RAID-массив

Направленность курсовой работы: учебная

Источник тематики (кафедра, предприятие, НИР): кафедра

График выполнения работы: 25% – 4 нед., 50% – 8 нед., 75% – 12 нед., 100% – 16 нед.

Техническое задание:

Разработать МК-систему для хранения данных в формате RAID-массива. Предусмотреть возможность выбора режима хранения данных: RAID 0 — RAID 5. Определить достоинства и недостатки работы RAID-массива в каждом из рассматриваемых режимов.

Предусмотреть команды для записи временных данных с указанием времени жизни.

Выводить на ЖК-дисплей статистическую информацию: текущий режим хранения, общий объем массива, объем занятой и свободной памяти, скорость записи.

Управление работой МК-системы осуществлять с ПЭВМ и телефона.

Выбрать наиболее оптимальный вариант МК. Выбор обосновать.

Разработать схему, алгоритмы и программу. Отладить проект в симуляторе или на макете. Оценить потребляемую мощность. Описать принципы и технологию программирования используемого микроконтроллера.

Оформление курсовой работы:

1. Расчетно-пояснительная записка на 30-35 листах формата А4.
2. Перечень графического материала:
   а) схема электрическая функциональная;
   б) схема электрическая принципиальная.

Дата выдачи задания: «4» сентября 2023 г. Дата защиты: «20» декабря 2023 г.

Руководитель курсовой работы	04.09.2023 (Подпись, дата)	И.Б. Трамов (И.О.Фамилия)
Студент	04.09.2023 (Подпись, дата)	Д.Д. Шестаков (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах; один выдается студенту, второй хранится на кафедре

РЕФЕРАТ

РПЗ 120 страниц, 49 рисунков, 5 таблиц, 12 источников, 2 приложения.

МИКРОКОНТРОЛЛЕР, СИСТЕМА, RAID, ПОСЛЕДОВАТЕЛЬНАЯ FLASH-ПАМЯТЬ.

Объектом разработки является RAID-массив, объединяющий несколько запоминающих устройств в логический модуль для повышения отказоустойчивости и (или) производительности.

Цель работы – создание функционального устройства ограниченной сложности, модель устройства и разработка необходимой документации на объект разработки.

Поставленная цель достигается посредством использования Proteus 8.

В процессе работы над курсовым проектом решаются следующие задачи: выбор МК и драйвера обмена данных, создание функциональной и принципиальной схем системы, расчет потребляемой мощности устройства, разработка алгоритма управления и соответствующей программы МК, а также написание сопутствующей документации.

Оглавление

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

МК – микроконтроллер.

ТЗ – техническое задание.

Proteus 8 — пакет программ для автоматизированного проектирования (САПР) электронных схем.

RAID – Redundant Array of Independent Disks – это технология объединения двух и более накопителей в единый логический элемент с целью повышения производительности и (или) отказоустойчивости отдельно взятого элемента массива.

UART – Universal asynchronous receiver/transmitter – последовательный универсальный синхронный/асинхронный приемопередатчик.

SPI – Serial Peripheral Interface – интерфейс для связи МК с другими внешними устройствами.

FLASH-память с последовательным интерфейсом – энергонезависимое запоминающее устройство, выполненное в виде отдельной микросхемы, взаимодействие с которой происходит через SPI.

LED-дисплей – устройство отображения и передачи визуальной информации, в котором каждой точкой — пикселем — является один или несколько полупроводниковых светодиодов

ВВЕДЕНИЕ

В данной работе производится разработка RAID-массива для объединения нескольких запоминающих устройств в логический модуль для повышения отказоустойчивости и (или) производительности.

В процессе выполнения работы проведён анализ технического задания, создана концепция устройства, разработаны электрические схемы, построен алгоритм и управляющая программа для МК, выполнено интерактивное моделирование устройства.

Система состоит из МК; кнопки RESET; LED-дисплея для отображения информации о текущем режиме хранения, общем объеме массива, объеме занятой и свободной памяти, скорости записи; 4 микросхем FLASH-памяти с последовательным интерфейсом, которые выступаю в виде запоминающих устройств в RAID-массиве; виртуального терминала для симуляции управления RAID-массивом с ПЭВМ.

Актуальность разрабатываемого устройства для хранения данных в виде RAID-массива заключается в том, что в современном информационном мире, где цифровые данные играют ключевую роль, обеспечение производительности и отказоустойчивости устройств, хранящих эти данные, является критическим аспектом. Эта система предоставляет пользователям возможность выбора режима хранения данных, что позволяет под разные задачи выбирать подходящий режим.

1 Конструкторская часть

1.1 Анализ требований и принцип работы системы

Исходя из требований, изложенных в техническом задании, можно сделать вывод, что задачей работы устройства является запись на внешние запоминающие устройства (в данном случае микросхемы FLASH-памяти с последовательным интерфейсом) данных, способ распределения и время жизни которых вводятся с ПЭВМ.

Помимо этого, у нас должна выводиться на ЖК-дисплей статическая информация: текущий режим хранения, общий объем массива, объем занятой и свободной памяти, скорость записи.

У пользователя также есть возможность нажать кнопку «Reset», что перезапустит программу выполнения на микроконтроллере.

Всего программа поддерживает пять режимов хранения данных: RAID 0, 1, 3, 4, 5.

RAID 0 наиболее производительный и наименее защищенный из всех режимов. Данные разбиваются на блоки пропорционально количеству запоминающих устройств, что приводит к более высокой пропускной способности. Высокая производительность данной структуры обеспечивается параллельной записью и отсутствием избыточного копирования. Отказ любого запоминающего устройства в массиве приводит к потере всех данных.

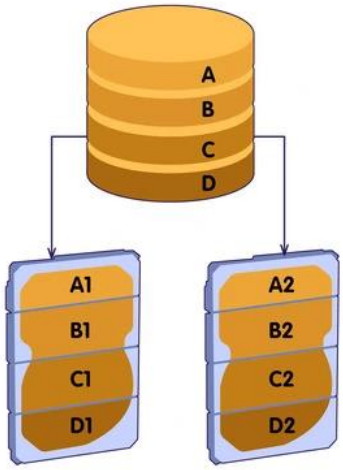


Рисунок 1 – Распределение данных в RAID 0

RAID 1 - mirroring - зеркальное отражение двух запоминающих устройств (два запоминающих устройства являются полными копиями друг друга). Избыточность структуры данного массива обеспечивает его высокую отказоустойчивость. Массив отличается высокой себестоимостью и низкой производительностью.

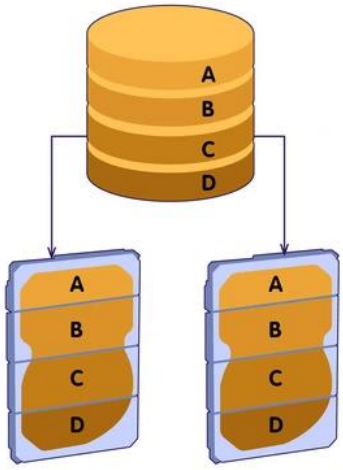


Рисунок 2 – Распределение данных в RAID 1

В массиве RAID 3 из ***n*** запоминающих устройств данные разбиваются на байты и распределяются по ***n - 1*** запоминающим устройствам. Ещё одно

запоминающее устройство используется для хранения блоков чётности. В случае повреждения данных на одном из запоминающих устройств возможно восстановление с помощью операции XOR.

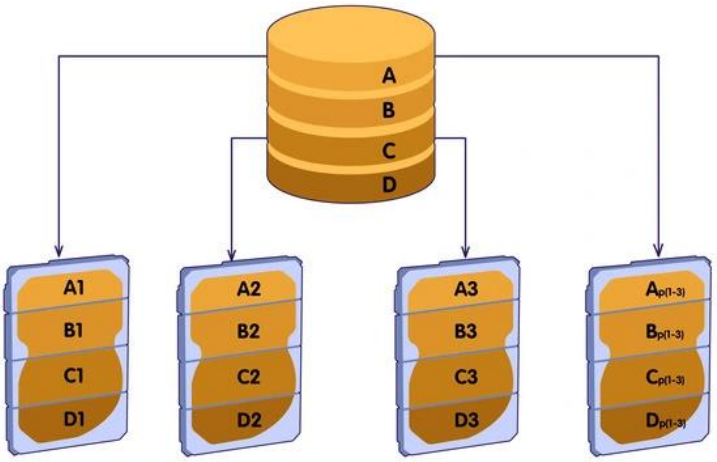


Рисунок 3 – Распределение данных в RAID 3

RAID 4 похож на RAID 3, но отличается от него тем, что данные разбиваются на блоки, а не на байты. Таким образом, удалось отчасти «победить» проблему низкой скорости передачи данных небольшого объёма. Запись же производится медленно из-за того, что чётность для блока генерируется при записи и записывается на единственный диск.

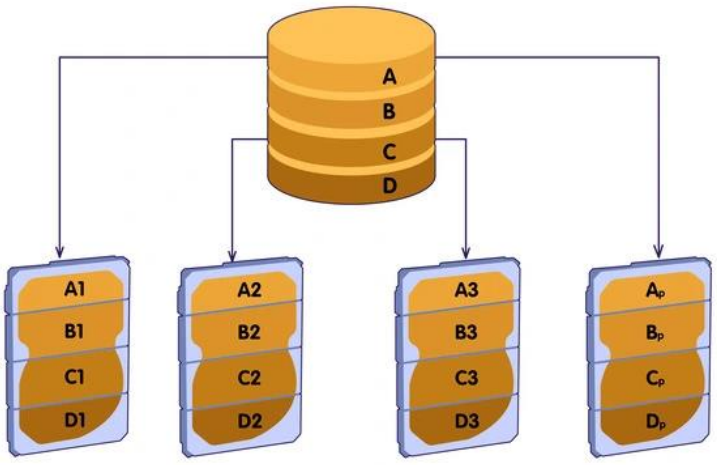


Рисунок 4 – Распределение данных в RAID 4

В RAID 5 используются блоки четности и чередование как в RAID 3 и RAID 4, но объем под хранение блоков четности распределяется по всему массиву. Это дает прирост в скорости записи, так как операции теперь можно производить параллельно. Количество накопителей в массиве начинается с трех. Для хранения контрольных сумм выделяется объем, равный объему одного накопителя. RAID 5 наиболее распространен и используется в файловых серверах, серверах общего хранения, серверах резервного копирования, работе с потоковыми данными и других средах, требующих хорошей производительности.

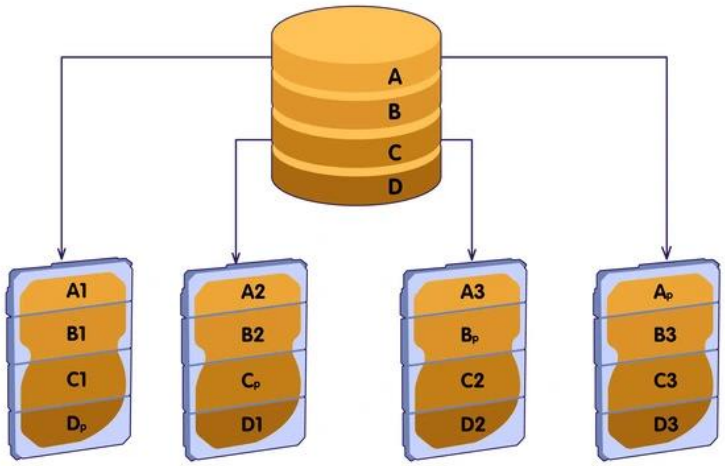


Рисунок 5 – Распределение данных в RAID 5

Разработанная структурная RAID-массива представлена на рисунке 6.

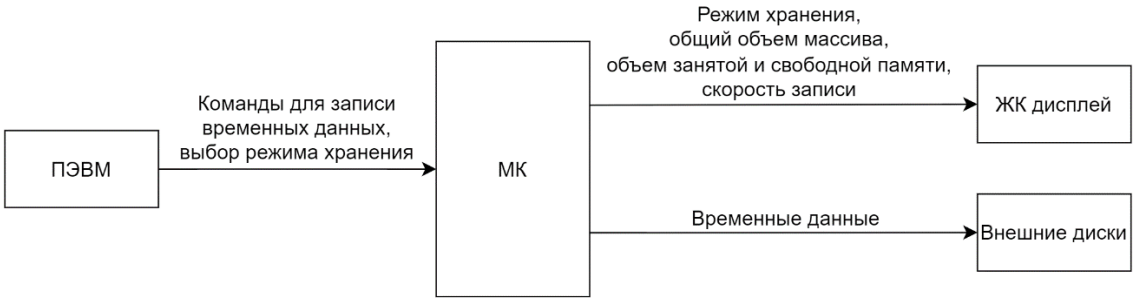


Рисунок 6 – Структурная схема устройства

1.2 Проектирование функциональной схемы

В этом разделе приведено функциональное описание работы системы и проектирование функциональной схемы.

1.2.1 Микроконтроллер STM32F103C8T6

Основным элементом разрабатываемого устройства является микроконтроллер (МК). Существует множество семейств МК, для разработки выберем из тех, что являются основными [2]:

– 8051 – это 8-битное семейство МК от компании Intel.

– PIC – это серия МК, разработанная компанией Microchip;

– AVR – это серия МК разработанная компанией Atmel;

– ARM – одним из семейств процессоров на базе архитектуры RISC, разработанным компанией Advanced RISC Machines.

Сравнение семейств показано в таблице 1.

Таблица 1 – Сравнение семейств МК

Критерий	8051	PIC	AVR	ARM
Разрядность	8 бит	8/16/32 бит	8/32 бит	32 бит, иногда 64 бит
Интерфейсы	UART, USART, SPI, I2C	PIC, UART, USART, LIN, CAN, Ethernet, SPI, I2S	UART, USART, SPI, I2C, иногда CAN, USB, Ethernet	UART, USART, LIN, I2C, SPI, CAN, USB, Ethernet, I2S, DSP, SAI, IrDA, FATFS
Скорость	12 тактов на инструкцию	4 такта на инструкцию	1 такт на инструкцию	1 такт на инструкцию
Память	ROM, SRAM, FLASH	SRAM, FLASH	Flash, SRAM, EEPROM	Flash, SDRAM, EEPROM

Энергопо-требление	Среднее	Низкое	Низкое	Низкое
Объем FLASH памяти	До 128 Кб	До 512 Кб	До 256 Кб	До 2056 Кб

Было выбрано семейство ARM, так как для разрабатываемой системы нужна высокая скорость работы интерфейсов для отрисовки текста на ЖК-дисплее и работы с внешней FLASH-памятью. Кроме того, для программы потенциально понадобится большой объем памяти, чтобы вместить все алгоритмы режимов хранения, работу с внешней FLASH-памятью.

ARM включает в себя немалое количество семейств, поэтому рассмотрим только основные

1. STM32, имеющие следующие характеристики:
– Flash-память до 2056 Кбайт;
– RAM до 1,4 Мбайт;
– Максимальная частота ядра до 480 МГц;
– число пинов (ножек) ввода-вывода 16–64;
– самый разнообразный набор периферии
2. NXP, имеющие следующие характеристики:
– FLASH до 2048 Кбайт;
– RAM до 8096 Кбайт;
– Максимальная частота ядра до 360 МГц;
– число пинов ввода-вывода 16-64;
– самый разнообразный набор периферии
3. Toshiba, имеющие следующие характеристики:
– FLASH до 1,5Мбайт;
– RAM до 514 Кбайт;
– Максимальная частота ядра до 120 МГц;

– самый разнообразный набор периферии

Выберем подсемейство STM32 от ST Microelectronics, так как у них самая активная поддержка сообщества, что поможет использовать некоторые готовые решения. Кроме того, мы имели дело с представителем этого подсемейства в рамках лабораторных работ курса «Микропроцессорные системы», что также является плюсом при выборе.

В подсемействе STM32 семейства ARM был выбран МК STM32F103C8T6, обладающий всем необходимым функционалом для реализации проекта:

– 2 интерфейса SPI для работы с внешней FLASH-памятью и для ЖК-дисплея;

– интерфейс UART для ПЭВМ(виртуального терминала);

– 20 Кбайт RAM;

– 4 таймера, которые могут быть использованы для отслеживание времени жизни данных;

– 64 Кбайта FLASH-памяти;

– Возможность назначить внешнее прерывание практически на любой PIN;

– частота работы до 72 Мгц.

А также с данным МК уже есть опыт работы, что упростит разработку, и не потребует траты времени на изучение функционала МК.

Это экономичный 32-разрядный микроконтроллер, основанный на RISC архитектуре. STM32F103C8T6 обеспечивает производительность 1 миллион операций в секунду на 1 МГц синхронизации за счет выполнения большинства инструкций за один машинный цикл и позволяет оптимизировать потребление энергии за счет изменения частоты синхронизации. Структурная схема МК показана на рисунке 7 и УГО на рисунке 8 [3].

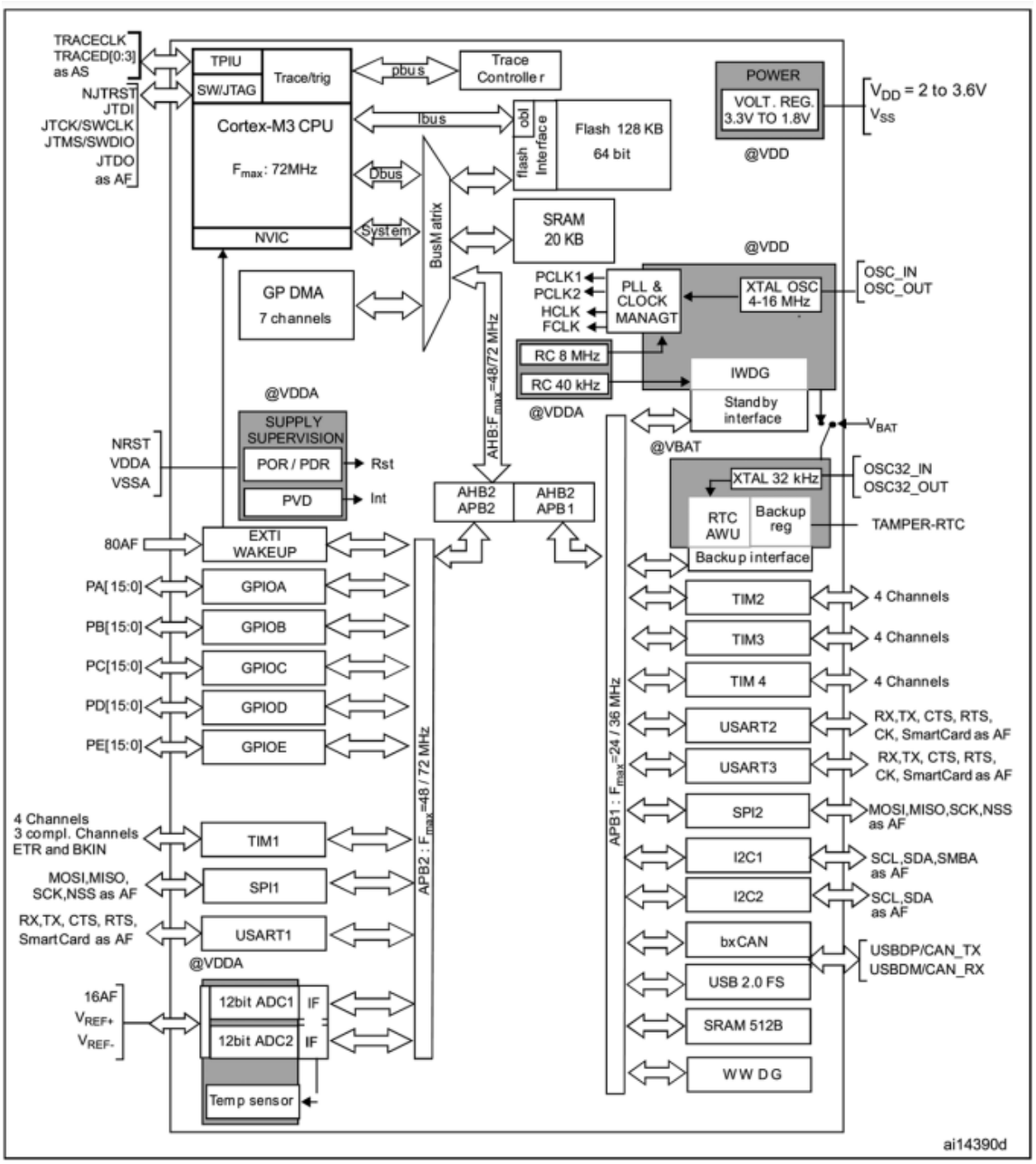


Рисунок 7 – Структурная схема МК STM32F103C8T6

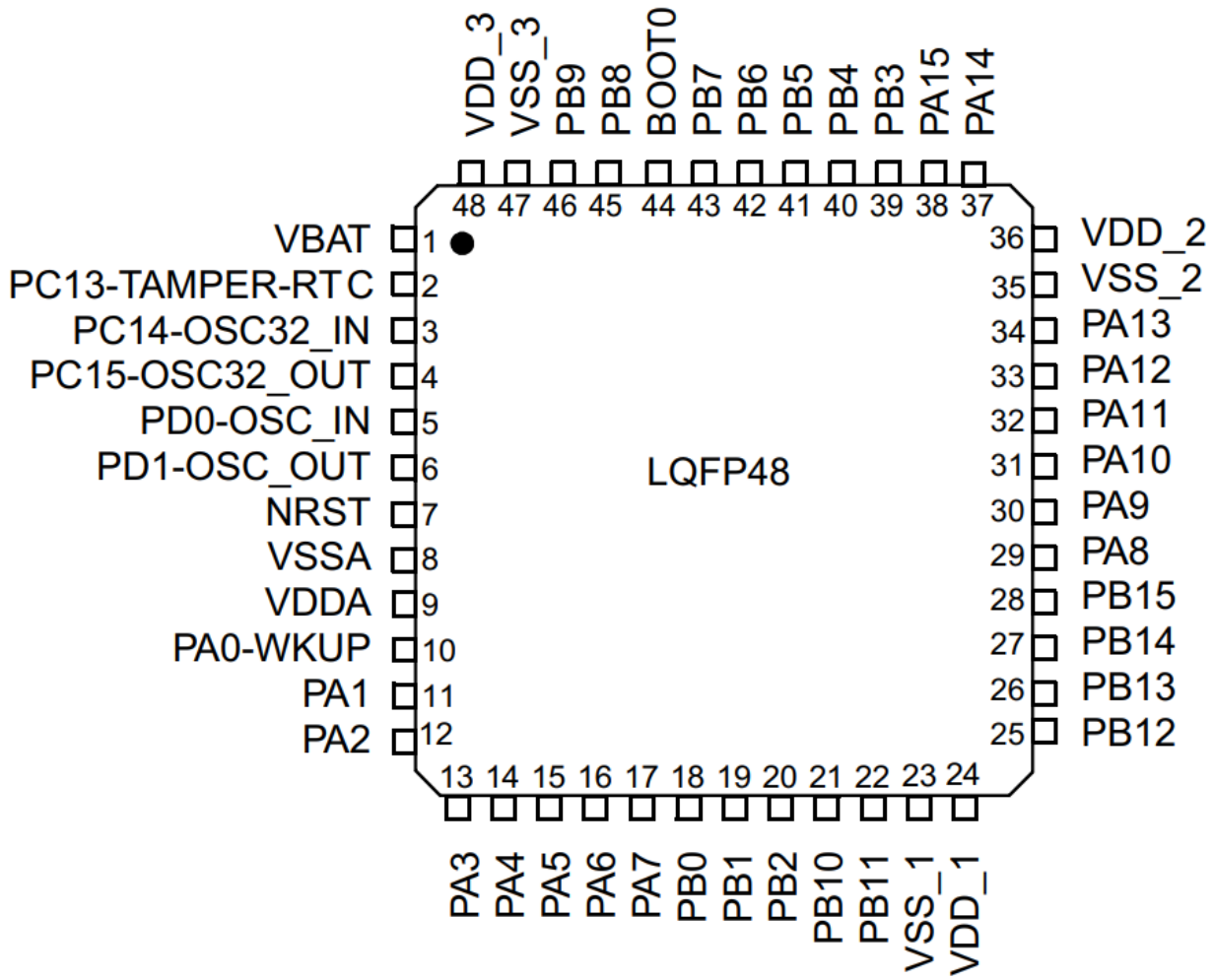


Рисунок 8 – УГО МК STM32F103C8T6

Он обладает следующими характеристиками:

1. Архитектура и производительность:

– Процессор Cortex-M3 от ARM с частотой до 72 МГц.

– 32-битная архитектура с набором команд Thumb-2 для эффективной работы.

2. Память:

– 64 КБ флеш-памяти для программного кода.

– 20 КБ ОЗУ (SRAM) для хранения данных.

– Возможность расширения памяти с использованием внешних устройств.

3. Периферийные устройства:

– Несколько портов GPIO (General-Purpose Input/Output) для подключения и управления внешними устройствами.

– USART, SPI, I2C и другие интерфейсы для обмена данными с внешними устройствами.

– АЦП (аналогово-цифровой преобразователь) для измерения аналоговых сигналов.

4. Интерфейсы и коммуникации:

– USB-интерфейс для обмена данными с компьютером или другими устройствами.

– Возможность работы с различными протоколами связи, такими как CAN (Controller Area Network), Ethernet и другими.

5. Прочие особенности:

– Встроенные таймеры и счетчики для управления временем и частотой.

– Низкое энергопотребление в режиме ожидания.

– Защита от переполнения стека и ошибок программирования.

6. Программирование и разработка:

– Поддержка различных интегрированных сред разработки (IDE), таких как Keil, STM32CubeIDE, и других.

– Обширная документация, примеры кода и библиотеки для упрощения разработки.

7. Применение:

– Широко используется в различных приложениях, включая промышленные системы управления, автоматизацию, умные устройства, медицинское оборудование, робототехнику и многое другое.

8. Напряжение питания: 2– 3.6В.

1.2.1.1 Используемые элементы

Для функционирования RAID-массива в МК STM32F103C8T6 задействованы не все элементы его архитектуры. Выделим и опишем те, что используются во время функционирования схемы.

– Порт A – использованные пины и их назначение описано в пункте 1.2.3.

– Указатель стека – играет важную роль в организации стека, используемого для управления вызовами подпрограмм. Указатель стека используется для сохранения адреса возврата и регистров при вызове функций. Это обеспечивает корректный возврат из функций и поддерживает структуру вызовов функций.

– Регистры общего назначения – предназначены для хранения операндов арифметико-логических операций, а также адресов или отдельных компонентов адресов ячеек памяти.

– АЛУ – выполняет арифметические и логические операции, обеспечивает выполнение базовых математических операций и манипуляций с битами.

– Память SRAM – статическая память МК, хранящая объявленные переменные.

– Память Flash – память МК, хранящая загруженную в него программу.

– Программный счетчик – указывает на следующую по испольнению команду.

– Регистры команд – содержит исполняемую в настоящий момент команду(или следующую), то есть команду, адресуемую счетчиком команд.

– Декодер – выделяет код операции и операнды команды и далее вызывает микропрограмму, исполняющую данную команду.

– Сигналы управления – нужны для синхронизации обработки данных.

– Логика программирования – устанавливает логику того, как будет вшита программа в МК.

– Генератор – генератор тактовых импульсов. Необходим для синхронизации работы МК.

– Управление синхронизацией и сбросом – обрабатывает тактовые сигналы и отвечает за сброс состояния МК.

– Прерывания – обрабатывает внешние прерывания и прерывания периферийных устройств МК (таймеров, портов и т.д.). В устройстве используются прерывания с таймеров для отслеживания времени жизни данных.

– UART (Universal Asynchronous Receiver Transmitter) – интерфейс, при помощи которого происходит передача команд управления в МК из ПЭВМ.

– SPI (Serial Peripheral Interface) – интерфейс для связи МК с другими внешними устройствами. В устройстве используется для взаимодействия с внешней FLASH-памятью, прошивки МК и вывода данных на жидкокристаллический дисплей.

– Таймеры – МК содержит в себе четыре 16-ти разрядных таймеров (TIM1, TIM2, TIM3, TIM4). В устройстве используется только один канал таймера TIM2 для отслеживания времени жизни данных.

1.2.1.2 Распределение портов

МК STM32F103C8T6 содержит пять портов – A, B, C, D и E. Опишем назначение тех, что используются в данной системе для её функционирования.

Порт A:

– PA1 – CS-пин (Chip Select) для первой микросхемы последовательной FLASH-памяти;

– PA2 – CS-пин (Chip Select) для второй микросхемы последовательной FLASH-памяти;

– PA3 – CS-пин (Chip Select) для третьей микросхемы последовательной FLASH-памяти;

– PA4 – CS-пин (Chip Select) для четвертой микросхемы последовательной FLASH-памяти;

– PA5 – тактовый сигнал (SCK) для SPI для LCD-дисплея и последовательной FLASH-памяти;

– PA6 – MISO-пин для SPI для последовательной FLASH-памяти;

– PA7 – MOSI-пин для SPI для LCD-дисплея и последовательной FLASH-памяти;

– PA8 – отправка данных или команд на дисплей;

– PA9 – отправка данных по UART на ПЭВМ;

– PA10 – прием данных по UART от ПЭВМ;

– PA11 – CS-пин (Chip Select) для LCD-дисплея;

– PA13 – SWDIO-пин для программатора ST-LINK V2;

– PA14 – SWCLK-пин для программатора ST-LINK V2.

1.2.1.3 Организация памяти

Схема организации памяти МК STM32F103C8T6 показана на рисунке 9.

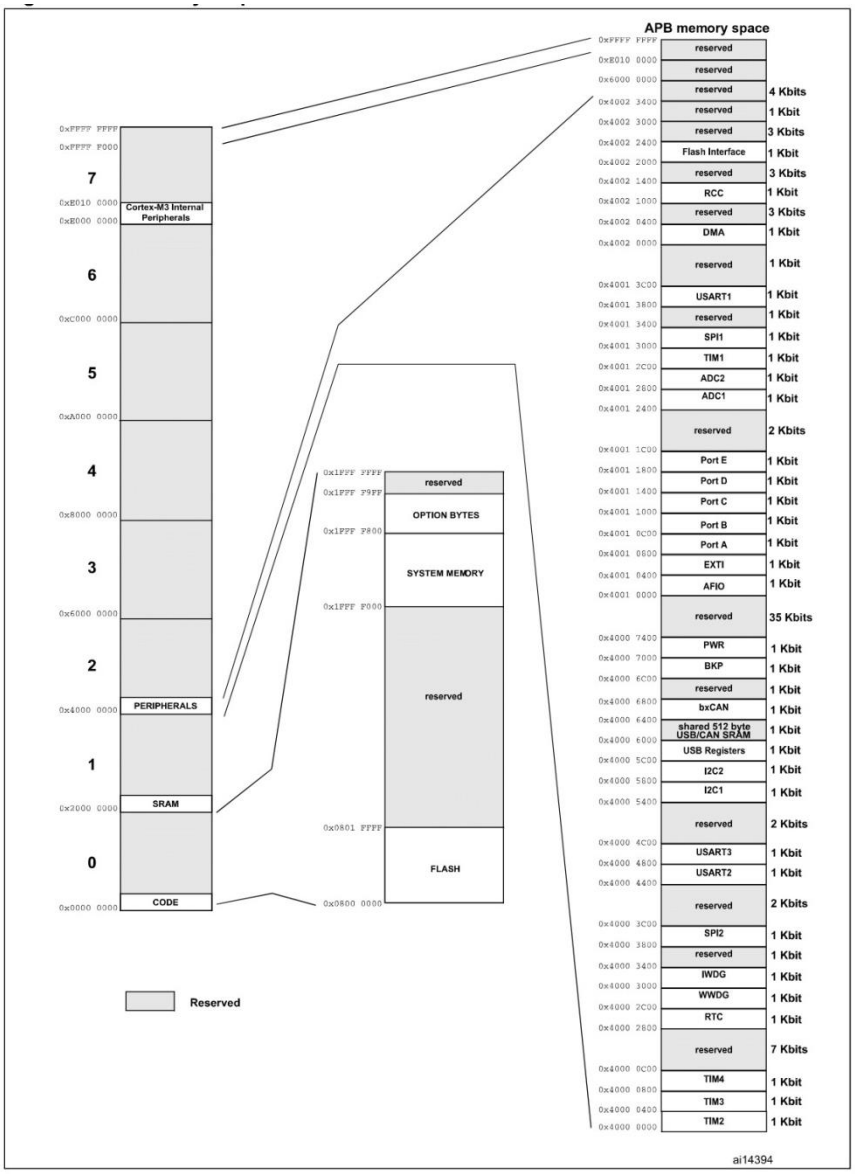


Рисунок 9 – Организация памяти МК STM32F103C8T6

1.2.2 Прием данных от ПЭВМ

Приём данных от ПЭВМ происходит через драйвер MAX232. MAX232 – интегральная схема, преобразующая сигналы последовательного порта RS-232 в цифровые сигналы.

RS-232 – стандарт физического уровня для синхронного и асинхронного интерфейса (USART и UART). Обеспечивает передачу данных и некоторых специальных сигналов между терминалом и устройством приема. Сигнал, поступающий от интерфейса RS-232, через преобразователь передается в микроконтроллер на вход RxD.

К внешнему устройству MAX232 подключен через разъем DB-9. На схеме условное обозначение – XP2.

Внутреннее изображение MAX232 показано на рисунке 10. Назначение пинов описано в таблице 2 [4].

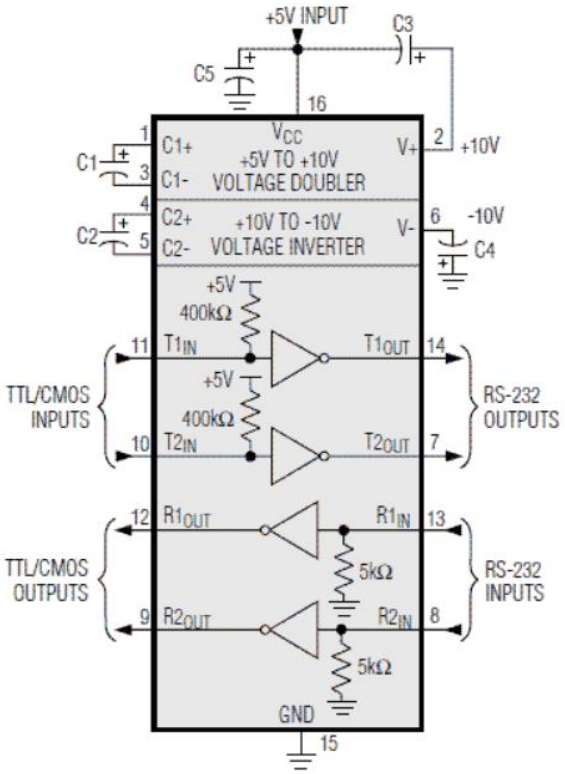


Рисунок 10 – Преобразователь MAX232

Таблица 2 - Назначение пинов MAX232

Номер	Имя	Тип	Описание
1	C1+	–	Положительный вывод C1 для подключения конденсатора
2	VS+	O	Выход положительного заряда для накопительного конденсатора
3	C1-	–	Отрицательный вывод C1 для подключения конденсатора
4	C2+	–	Положительный вывод C2 для подключения конденсатора

Продолжение таблицы 2

Номер	Имя	Тип	Описание
5	C2-	–	Отрицательный вывод C2 для подключения конденсатора
6	VS-	O	Выход отрицательного заряда для накопительного конденсатора
7, 14	T2OUT, T1OUT	O	Вывод данных по линии RS232
8, 13	R2IN, R1IN	I	Ввод данных по линии RS232
9, 12	R2OUT, R1OUT	O	Вывод логических данных
10, 11	T2IN, T1IN	I	Ввод логических данных
15	GND	–	Земля
16	Vcc	–	Напряжение питания, подключение к внешнему источнику питания 5 В

Когда микросхема MAX232 получает на вход логический "0" от внешнего устройства, она преобразует его в напряжение от +5 до +15В, а когда получает логическую "1" - преобразует её в напряжение от -5 до -15В, и по тому же принципу выполняет обратные преобразования от RS-232 к внешнему устройству.

1.2.3 Настройка USART для взаимодействия с ПЭВМ

Интерфейс USART (Universal Synchronous Asynchronous Receiver Transmitter) в микроконтроллерах STM32 представляет собой универсальный последовательный интерфейс, который может работать в режиме синхронной или асинхронной передачи данных. Он обеспечивает возможность обмена данными между микроконтроллером и другими устройствами, такими как датчики, модули связи и периферийные устройства.

USART в STM32 поддерживает передачу данных через одну линию для приема (RX) и одну для передачи (TX). Он также может работать в

полудуплексном режиме, когда одна линия используется для передачи и приема данных.

USART может настраиваться на разные скорости передачи данных (бодрейты), количество бит данных, контроль четности, стоповые биты и другие параметры через специальные регистры микроконтроллера. Это обеспечивает гибкость в настройке передачи данных в соответствии с требованиями конкретного приложения.

USART был выбран для использования, так как это достаточно быстрый интерфейс, его использование это классический ход при работе с терминалом.

В разрабатываемой системе USART используется в асинхронном режиме для вывода текста на виртуальный терминал и для чтения эталонной контрольной суммы с виртуального терминала, который выступает в роли ПЭВМ. Рассмотрим настройку USART для этого конкретного приложения и регистры, с помощью которых это делается.

Настройка USART в разрабатываемой системе показана на рисунке 11.

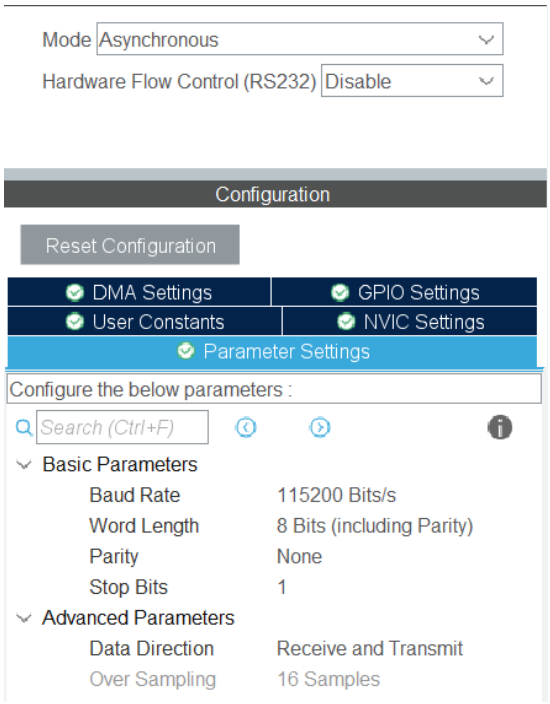


Рисунок 11 – Настройка USART

Таким образом, USART используется в асинхронном режиме, контроль сигнала CTS/RTS отключен, baud rate – 115200 бит/с, длина каждой посылки – 8 бит, включая бит четности, контроль четности отключен, используется один стоп-бит, оверсемплинг в режиме 16-семплирования.

Настройка USART по указанным выше параметрам показана в листинге 1.

Листинг 1 – Настройка USART

```
static void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
}
```

Оверсемплинг в USART относится к технике, используемой для приема данных в асинхронном режиме. Эта техника помогает улучшить точность синхронизации битов данных, особенно при работе с высокими скоростями передачи данных.

Оверсемплинг подразумевает выбор частоты сэмплирования (число раз, которое система измеряет состояние входного сигнала за определенный промежуток времени) значительно выше, чем минимально необходимая частота для корректного считывания данных.

В USART для асинхронной передачи, оверсемплинг обычно используется для более точного определения момента прихода каждого бита данных. К примеру, в режиме 16-семплирования (16x oversampling), каждый бит данных будет сэмплироваться 16 раз за период передачи, что улучшает точность считывания данных и помогает бороться с потерей или искажением сигнала в условиях шумов или неполадок в канале связи.

Эта техника позволяет повысить устойчивость и надежность приема данных по USART, особенно при работе на высоких скоростях передачи данных или в условиях, где возможны помехи или искажения сигнала.

Всего существует 7 регистров, связанных с настройкой и работой USART: USART_SR (Status register), USART_DR (Data register), USART_BRR (Baud rate register), USART_CR1 (Control register 1), USART_CR2 (Control register 2), USART_CR3 (Control register 3), USART_GTPR (Guard time and prescaler register). Ниже будут описаны все регистры кроме неиспользованных регистров для настройки.

Начнем с настройки USART. Для этого используются control-регистры и регистр управления скоростью передачи. Начнем с USART_CR1. Его изображение представлено на рисунке 12.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 12 – Регистр USART_CR1

Описание регистра:

UE: USART enable - включить USART (включается установкой бита в 1).

M: Word length - длина слова, задаёт количество бит данных в одном фрейме. Бит не должен модифицироваться в процессе обмена данными (это касается как передачи, так и приёма). 0 - 1 старт-бит, 8 бит данных, n стоп-битов; 1 - 1 старт-бит, 9 бит данных, n стоп-битов. Примечание. Бит чётности считается битом данных.

WAKE: Wakeup method - метод пробуждения USART. 0 - "линия свободна" (Idle line); 1- адресная метка.

PCE: Parity control enable - включить аппаратный контроль чётности (генерация бита чётности при передаче данных и проверка в принимаемых данных).

PS: Parity selection - выбор метода контроля чётности. Выбор происходит после завершения передачи/приёма текущего байта. 0 - контроль на чётность; 1 - контроль на нечётность.

PEIE: PE interrupt enable - разрешение прерывания от PE. 0 – прерывание запрещено; 1 – генерируется прерывание от USART, когда USART_SR.PE==1.

TXEIE: TXE interrupt enable - разрешение прерывания от TXE. 0 – прерывание запрещено; 1 – генерируется прерывание от USART, когда USART_SR.TXE==1.

TCIE: Transmission complete interrupt enable - разрешение прерывания после завершения передачи. 0 – прерывание запрещено; 1 – генерируется прерывание от USART, когда USART_SR.TC==1.

RXNEIE: RXNE interrupt enable - разрешение прерывания от RXNE. 0: прерывание запрещено; 1: генерируется прерывание от USART, когда USART_SR.ORE==1 или USART_SR.RXNE==1.

IDLEIE: IDLE interrupt enable - разрешение прерывания при обнаружении, что "линия свободна" (Idle line). 0: прерывание запрещено; 1: генерируется прерывание от USART, когда USART_SR.IDLE==1.

TE: Transmitter enable - включить передатчик USART (включается установкой бита в 1).

RE: Receiver enable - включить приёмник USART (включается установкой бита в 1). После установки бита, приёмник начинает поиск старт-бита во входном сигнале.

RWU: Receiver wakeup - переводит USART в тихий режим. Этот бит устанавливается и сбрасывается программно, а также может сбрасываться аппаратно при обнаружении пробуждающей последовательности.

SBK: Send break - отправить Break посылку. Бит может быть установлен и сброшен программно. Его необходимо программно установить в 1 для формирования Break посылки, он будет сброшен аппаратно во время

формирования stop-бита в Break фрейме. 0: Break-символ не передаётся; 1: Break-символ будет передан.

Теперь опишем регистр BRR, с помощью которого контролируется скорость передачи данных через USART. Регистр представлен на рисунке 13.

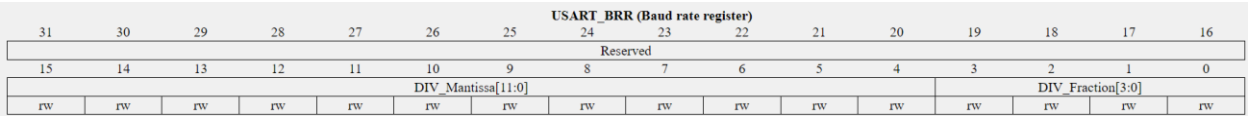


Рисунок 13 – Регистр BRR

DIV_Mantissa[11:0]: mantissa of USARTDIV - целая часть коэффициента деления делителя частоты.

DIV_Fraction[3:0]: fraction of USARTDIV - дробная часть коэффициента деления. В режиме с OVER8==1 в битовом поле DIV_Fraction[3:0] старший бит [3] не используется и должен быть сброшен.

С помощью регистра USART_BRR задаётся скорость передачи - одновременно как для приёмника USART, так и для передатчика. На рисунке 14 представлена схема, показывающая, как именно высчитывается скорость передачи.

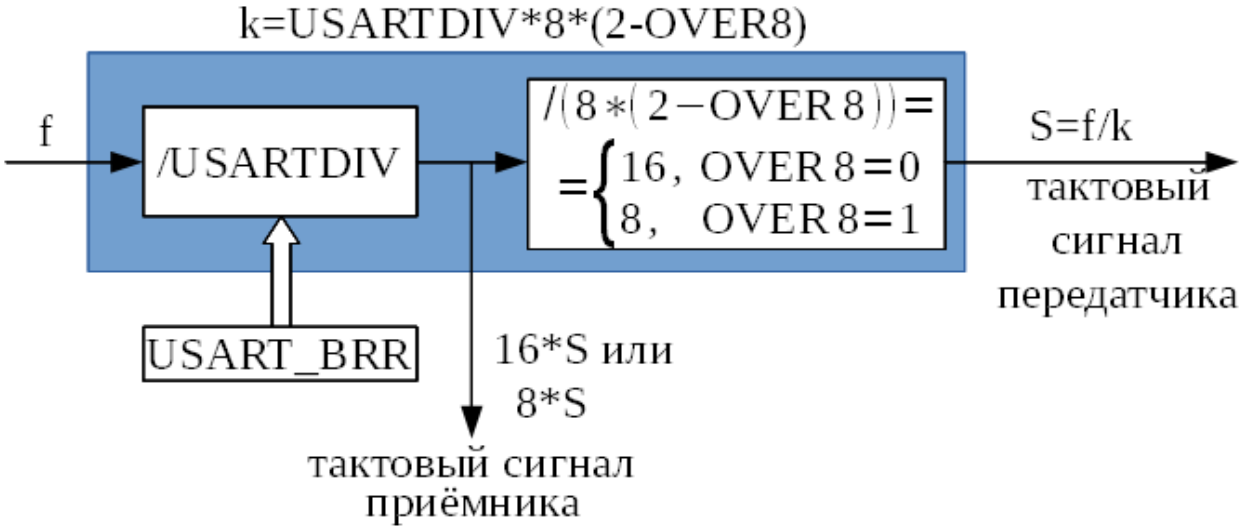


Рисунок 14 – Вычисление скорости приема и передачи

В данной системе было принято решение использовать baud rate = 115200, поэтому был выставлен USART_BRR = 69. Проверим: 8000000 / 69 = 115942 ~ 115200.

Далее рассмотрим USART_DR – регистр, через который передаются непосредственно данные. Он представлен на рисунке 15.

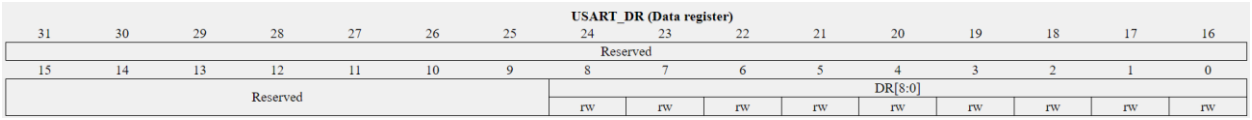


Рисунок 15 – Регистр данных

DR[8:0]: Data value - регистр данных. Содержит полученный или передаваемый символ, в зависимости от того, производится чтение из него или запись в регистр. Регистр выполняет двойную функцию за счёт того, что он является составным, он объединяет в себе два регистра: один для передачи (TDR) и один для приёма (RDR). TDR обеспечивает загрузку данных в выходной сдвигающий регистр, сдвигающий регистр преобразует загруженное в него слово в последовательную форму. Получаемые в последовательной форме данные накапливаются в приёмном сдвигающем регистре, когда фрейм получен полностью, данные из сдвигающего регистра передаются в регистр RDR, который реализует параллельный интерфейс между внутренней шиной микроконтроллера и входным сдвигающим регистром.

Когда осуществляется передача данных с включённым контролем чётности (USART_CR1.PCE==1), старший бит, записываемый в регистр USART_DR (бит [7] или [8], в зависимости от выбранной длины слова, см. USART_CR1.M), не учитывается. Он замещается вычисленным битом чётности.

При получении данных с включённым контролем чётности, при чтении из USART_DR будем получать значение, содержащее полученный бит чётности.

Последний рассматриваемый регистр в USART – USART_SR(status register). Он представлен на рисунке 16.

USART_SR (Status register)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

Рисунок 16 – Регистр статуса

CTS: CTS flag - флаг изменения состояния nCTS. Устанавливается аппаратно, когда происходит переключение сигнала на входе nCTS. Если установлен бит CTSIE (USART_CR3.CTSIE==1), то при установке флага генерируется прерывание. Флаг сбрасывается программно записью 0.

LBD: LIN break detection flag - флаг приёма посылки Break. Устанавливается аппаратно при обнаружении посылки Break на входе; если установлен бит LBDIE (USART_CR3.LBDIE==1), то генерируется прерывание. Флаг срсывается программно записью 0.

TXE: Transmit data register empty - флаг устанавливается аппаратно, когда содержимое регистра передаваемых данных TDR пересылается в сдвигающий регистр (доступ к TDR осуществляется путём записи в регистр USART_DR). Если установлен бит TXEIE (USART_CR1.TXEIE==1), генерируется прерывание. Флаг сбрасывается путём записи в регистр USART_DR.

TC: Transmission complete - флаг завершения передачи, устанавливается аппаратно, если передача фрейма завершена, и флаг TXE установлен (т.е. регистр передаваемых данных пуст, больше нет данных для передачи). Если USART_CR1.TCIE==1, то при установке флага генерируется прерывание. Флаг сбрасывается программно последовательностью действий: чтение регистра USART_SR, затем запись в USART_DR. Также бит может быть сброшен записью в него 0. Примечание. После сброса этот бит установлен.

RXNE: Read data register not empty - регистр данных для чтения не пуст. Флаг устанавливается аппаратно, когда содержимое принимающего сдвигающего регистра передаётся в регистр принимаемых данных RDR. Если USART_CR1.RXNEIE==1, при этом генерируется прерывание. Флаг

сбрасывается чтением из регистра USART_DR. Также бит может быть сброшен записью в него 0.

IDLE: IDLE line detected - линия свободна. Флаг устанавливается аппаратно, если обнаружено что линия свободна. Это происходит, если получен целый фрейм единиц. При этом генерируется прерывание, если USART_CR1.IDLEIE==1. Флаг сбрасывается программно последовательностью действий: чтение регистра USART_SR с последующим чтением из регистра USART_DR.

ORE: Overrun error - ошибка переполнения. Флаг устанавливается аппаратно, когда слово, полученное в сдвигающей регистр готово к перемещению в регистр принимаемых данных RDR, но RXNE==1 (регистр RDR не пуст, содержит ещё не прочитанные из него принятые USART данные). Если USART_CR1.RXNEIE==1, то при установке флага генерируется исключение. Флаг сбрасывается программно последовательностью действий: чтение из регистра USART_SR с последующим чтением из USART_DR.

NF: Noise detected flag - флаг устанавливается аппаратно при обнаружении шума в полученном фрейме. Сбрасывается программно последовательностью действий: чтение из регистра USART_SR, затем чтение из регистра USART_DR.

FE: Framing error - ошибка фрейма. Флаг устанавливается аппаратно в случае нарушения синхронизации, чрезмерного шума в линии, при обнаружении символа Break. Флаг сбрасывается программно последовательностью действий: чтение из регистра USART_SR, затем чтение из регистра USART_DR. Примечание. В отношении генерации прерывания этот флаг полностью аналогичен флагу NF.

PE: Parity error - ошибка чётности. Флаг устанавливается аппаратно, когда в принятом фрейме обнаружена ошибка чётности (если контроль чётности включён). Если USART_CR1.PEIE==1, то генерируется

прерывание. Флаг сбрасывается программно последовательностью действий: чтение из регистра USART_SR, затем чтение либо запись регистра USART_DR. Перед сбросом флага, программа должна дождаться установки флага RXNE (регистр данных для чтения не пуст).

1.2.4 LCD-дисплей ST7735

Для выбора дисплея в первую очередь необходимо рассчитать достаточный размер экрана. Так как был выбран TFT ЖК-дисплей то его размер (или разрешение) измеряется в пикселях. Так как взаимодействие с дисплеем должно быть удобно для пользователя, необходимо подобрать легко-читаемый размер шрифта.

Размер шрифта 10x6 пикселей на символ является подходящим по читаемости и занимает достаточно места чтобы уместить таблицу в 49 двузначных цифр.

Для выбора размера экрана проведем расчет в пикселях.

В ширину необходимо максимум 20 символов шрифта 10x6, в высоту – 8 символов шрифта 10x6 и межстрочный интервал по 5 пикселей.

Тогда в высоту необходимо 7 * 10 + 5 * 6 = 100 пикселей, в ширину – 20 * 6 = 120 пикселей

Получилось, что достаточный размер экрана 100x120 пикселей. Вариант поменьше взять нельзя, из вариантов побольше наиболее подходящий – 160x128 пикселей. Наиболее популярные контроллеры дисплеев такого разрешения, поддерживающие работу с МК семейства STM32 – ILI9163, ST7735. Сравнительный анализ дисплеев приведен в таблице 3.

Таблица 3 – Сравнительный анализ контроллеров дисплеев

Критерий	**ST7735**	**ILI9163**
Цветовая глубина	16 бит	16 бит
Интерфейсы	SPI, I2C	SPI
Цена	Ниже ILI9163	Выше ST7735
Размер	1,8 Дюймовый	1,8 Дюймовый

Как можно увидеть, контроллеры и дисплеи достаточно схожи между собой. В силу меньшей стоимости ST7735 был выбран он. ST7735 – это однокристальный контроллер/драйвер для графического TFT ЖК-дисплея. Он может выполнять операции чтения/записи данных в оперативной памяти дисплея без внешнего тактового сигнала для минимизации энергопотребления [5].

Основные пины взаимодействия дисплея:

– IM2 – выбор шины параллельного и последовательного интерфейса – при установке в 1 параллельный, при 0 – последовательный.

– IM1, IM0 – выбор типа параллельного интерфейса. В таблице 4 представлены возможные значения.

Таблица 4 – Типы параллельного интерфейса

IM1	**IM0**	**Параллельный интерфейс**
0	0	8 бит
0	1	16 бит
1	0	9 бит
1	1	18 бит

– SPI4W – 0 при трех линиях SPI, 1 при четырех линиях.

– RESX – сигнал перезапустит устройство и нужно его использовать для правильной инициализации устройства.

– CSX – пин выбора микроконтроллера устройства, работает по низкому сигналу.

– D/CX – пин выбора данных или команды на интерфейсе микроконтроллера дисплея. При 1 – данные или параметры, при 0 – команды. При SPI используется как SCL.

– RDX – дает возможность считать при включенном параллельном интерфейсе в микроконтроллере.

– WRX(D/CX) – дает возможность писать при включенном параллельном интерфейсе в микроконтроллере. При 4 линейном SPI используется как D/CX.

– D[17:0] – используются как шины отправки данных параллельного интерфейса микроконтроллера. D0 это сигнал входа/выхода при последовательном интерфейсе. При последовательном интерфейсе сигналы D[17:1] не используются.

– TE – пин вывода для синхронизации микроконтроллера с частотой устройства, активируемый программно командой перезапуска.

– OSC – контролирующий пин вывода внутреннего тактового генератора, активируемый программно командой перезапуска.

Пины выбора режима дисплея:

– EXTC – использование режима расширенных команд. При 0 используются обычные команды, при 1 расширенный набор команд NVM.

– GM1, GM0 – пины выбора разрешения. При обоих пинах в состоянии 1 – разрешение 132x162, при обоих 0 – 128x160.

– SRGB – пин настройки порядка фильтров цветов RGB. В устройстве не важен.

– SMX/SMY – пины, отвечающие за направление вывода на дисплей. По умолчанию началом экрана считается левый верхний угол.

– LCM – пин выбора типа кристалла, белый при 0 и черный при 1.

– GS – пин изменения гаммы. Оставлен по умолчанию.

– TESEL – пин используется для изменения вывода TE сигнала. Работает только при GM[1:0] = 00 и при 0 выводит номер строки из 162, при 1 номер строки из 160.

1.2.6 FLASH-память M45PE16 с последовательным интерфейсом SPI

В качестве микросхем последовательной FLASH-памяти были выбраны микросхемы M45PE16, поскольку она есть в библиотеке Proteus 8 и обладает наибольшей памятью по сравнению с аналогичными микросхемами.

M45PE16 имеет объем 16 Мбит, максимальная тактовая частота 50 МГц. Память может быть записана или запрограммирована от 1 до 256 байт за раз с помощью инструкции ***“Page Write”*** или ***“Page Program”***. Инструкция записи страницы состоит из интегрированного цикла стирания страницы, за которым следует цикл программирования страницы. Память организована в виде 32 секторов, каждый из которых содержит 256 страниц. Ширина каждой страницы составляет 256 байт. Таким образом, всю память можно рассматривать как состоящую из 8192 страниц, или 2 097 152 байтов. Память можно стирать по странице за раз, используя инструкцию ***“Page Erase”***, или по сектору за раз, используя инструкцию ***“Sector Erase”***.

На рисунке 17 изображена логическая диаграмма микросхемы.

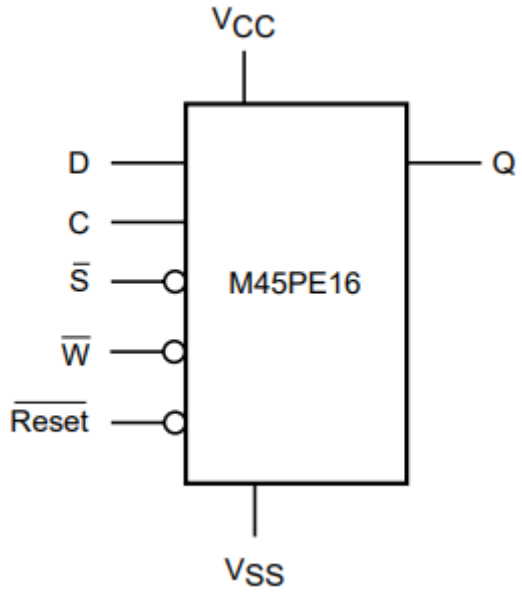


Рисунок 17 – Логическая диаграмма микросхемы M45PE16

У M45PE16 есть следующие пины взаимодействия:

– D (serial data input) – этот входной сигнал используется для последовательной передачи данных в микросхему. На него поступают инструкции, адреса и данные для программирования. Значения фиксируются по нарастающему фронту сигнала Serial Clock (C).

– Q (serial data output) – этот выходной сигнал используется для последовательной передачи данных из микросхемы. Данные передаются по по спадающему фронту сигнала Serial Clock (C).

– C (serial clock) – этот входной сигнал обеспечивает синхронизацию последовательного интерфейса. Инструкции, адреса или данные, поступающие на вход данных (D), фиксируются по нарастающему фронту сигнала Serial Clock (C). Данные на выходе данных (Q) изменяются после спада фронта сигнала Serial Clock (C).

– $\overline{S}$ (chip select) – это сигнал для активации микросхемы. Когда этот входной сигнал имеет высокий уровень, то устройство не активировано, а выход данных (Q) имеет высокий импеданс. Если не выполняется внутренний цикл чтения, программирования, стирания или записи, устройство будет находиться в режиме ожидания. Передача низкого уровня

сигнала активирует микросхему, переводя его в режим активного питания. После включения питания перед началом выполнения любой команды требуется падающий фронт сигнала на Chip Select ($\overline{S}$).

– Reset ($\overline{Reset}$) – вход Reset ($\overline{Reset}$) обеспечивает аппаратный сброс памяти. Когда на вход Reset ($\overline{Reset}$) подается высокий сигнал, память находится в нормальном рабочем режиме. Когда на вход Reset ($\overline{Reset}$) подается низкий уровень, устройство переходит в режим сброса. В этом режиме выход Q является высокоимпедансным. Если в момент подачи низкого уровня сигнала Reset ($\overline{Reset}$) выполняется внутренняя операция (цикл записи, стирания или программирования), устройство переходит в режим сброса и любой текущий цикл записи, программирования или стирания прерывается. Адресованные данные могут быть потеряны.

– Write Protect ($\overline{W}$) – этот входной сигнал переводит устройство в режим аппаратной защиты, когда Write Protect ($\overline{W}$) подключен к V_{ss}, в результате чего первые 256 страниц памяти становятся доступными только для чтения, защищая их от операций записи, программирования и стирания. Когда Write Protect ($\overline{W}$) подключен к V_{cc}, первые 256 страниц памяти ведут себя как остальные страницы памяти.

– V_{cc} – напряжение питания.

– V_{ss} – земля.

На рисунке 18 представлена схема микросхемы.

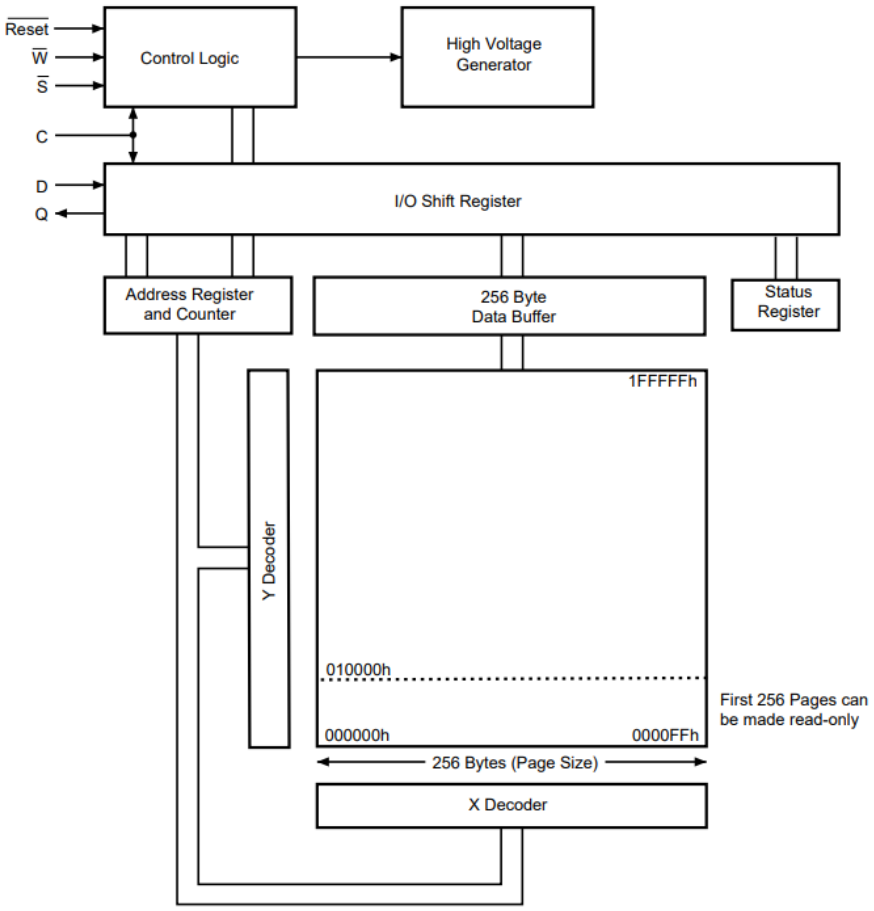


Рисунок 18 – Схема FLASH-памяти M45PE16

На рисунке 19 изображена организация памяти M45PE16.

Sector	Address Range	
	Start	End
31	001F 0000	001F FFFF
30	001E 0000	001E FFFF
29	001D 0000	001D FFFF
28	001C 0000	001C FFFF
27	001B 0000	001B FFFF
26	001A 0000	001A FFFF
25	0019 0000	0019 FFFF
24	0018 0000	0018 FFFF
23	0017 0000	0017 FFFF
22	0016 0000	0016 FFFF
21	0015 0000	0015 FFFF
20	0014 0000	0014 FFFF
19	0013 0000	0013 FFFF
18	0012 0000	0012 FFFF
17	0011 0000	0011 FFFF
16	0010 0000	0010 FFFF
15	000F 0000	000F FFFF
14	000E 0000	000E FFFF
13	000D 0000	000D FFFF
12	000C 0000	000C FFFF
11	000B 0000	000B FFFF
10	000A 0000	000A FFFF
9	0009 0000	0009 FFFF
8	0008 0000	0008 FFFF
7	0007 0000	0007 FFFF
6	0006 0000	0006 FFFF
5	0005 0000	0005 FFFF
4	0004 0000	0004 FFFF
3	0003 0000	0003 FFFF
2	0002 0000	0002 FFFF
1	0001 0000	0001 FFFF
0	0000 0000	0000 FFFF

Рисунок 19 – Организация памяти M45PE16

На рисунке 20 представлен набор команд, используемых для взаимодействия с микросхемой.

Instruction	Description	One-byte Instruction Code		Address Bytes	Dummy Bytes	Data Bytes
WREN	Write Enable	0000 0110	06h	0	0	0
WRDI	Write Disable	0000 0100	04h	0	0	0
RDID	Read Identification	1001 1111	9Fh	0	0	1 to 3
RDSR	Read Status Register	0000 0101	05h	0	0	1 to ∞
READ	Read Data Bytes	0000 0011	03h	3	0	1 to ∞
FAST_READ	Read Data Bytes at Higher Speed	0000 1011	0Bh	3	1	1 to ∞
PW	Page Write	0000 1010	0Ah	3	0	1 to 256
PP	Page Program	0000 0010	02h	3	0	1 to 256
PE	Page Erase	1101 1011	DBh	3	0	0
SE	Sector Erase	1101 1000	D8h	3	0	0
DP	Deep Power-down	1011 1001	B9h	0	0	0
RDP	Release from Deep Power-down	1010 1011	ABh	0	0	0

Рисунок 20 – Набор команд M45PE16

1.2.5 Настройка SPI для взаимодействия с LCD-дисплеем и последовательной FLASH-памятью

Интерфейс SPI (Serial Peripheral Interface – последовательный периферийный интерфейс) является высокоскоростным синхронным последовательным интерфейсом. Он обеспечивает обмен данными между микроконтроллером и различными периферийными устройствами, такими как АЦП, ЦАП, цифровые потенциометры, карты памяти, другие микросхемы и микроконтроллеры.

МК STM32F103C8T6 содержит два интерфейса SPI, которые обеспечивают передачу данных на частотах до 18 МГц. Один интерфейс SPI расположен на низкоскоростной шине APB1, работающей на тактовой частоте до 36 МГц, а другой – на высокоскоростной шине периферийных устройств APB2, которая работает на тактовой частоте до 72 МГц. Для увеличения эффективности передачи данных в микроконтроллере выделено два канала DMA. По интерфейсу SPI можно связать ведущий микроконтроллер с одним или несколькими ведомыми устройствами.

Одно из устройств должно быть определено ведущим (мастер), а остальные – ведомыми (подчинённые). Связь между устройствами осуществляется с помощью следующих линий связи:

– MOSI – выход данных для ведущего или вход данных для ведомого устройства;

– MISO – вход данных для ведущего или выход данных для ведомого устройства;

– SCK – сигнал общей синхронизации интерфейса.

Существует четыре режима передачи данных по SPI, которые определяются полярностью и фазой тактового сигнала. Отличие режимов заключается в том, что активным уровнем сигнала синхронизации может

быть единичный или нулевой потенциал, а запись данных может производиться по фронту или спаду импульса данного синхросигнала. Эти режимы интерфейса обозначаются цифрами 0, 1, 2 и 3. На рисунке 21 представлена диаграмма всех перечисленных режимов работы интерфейса SPI.

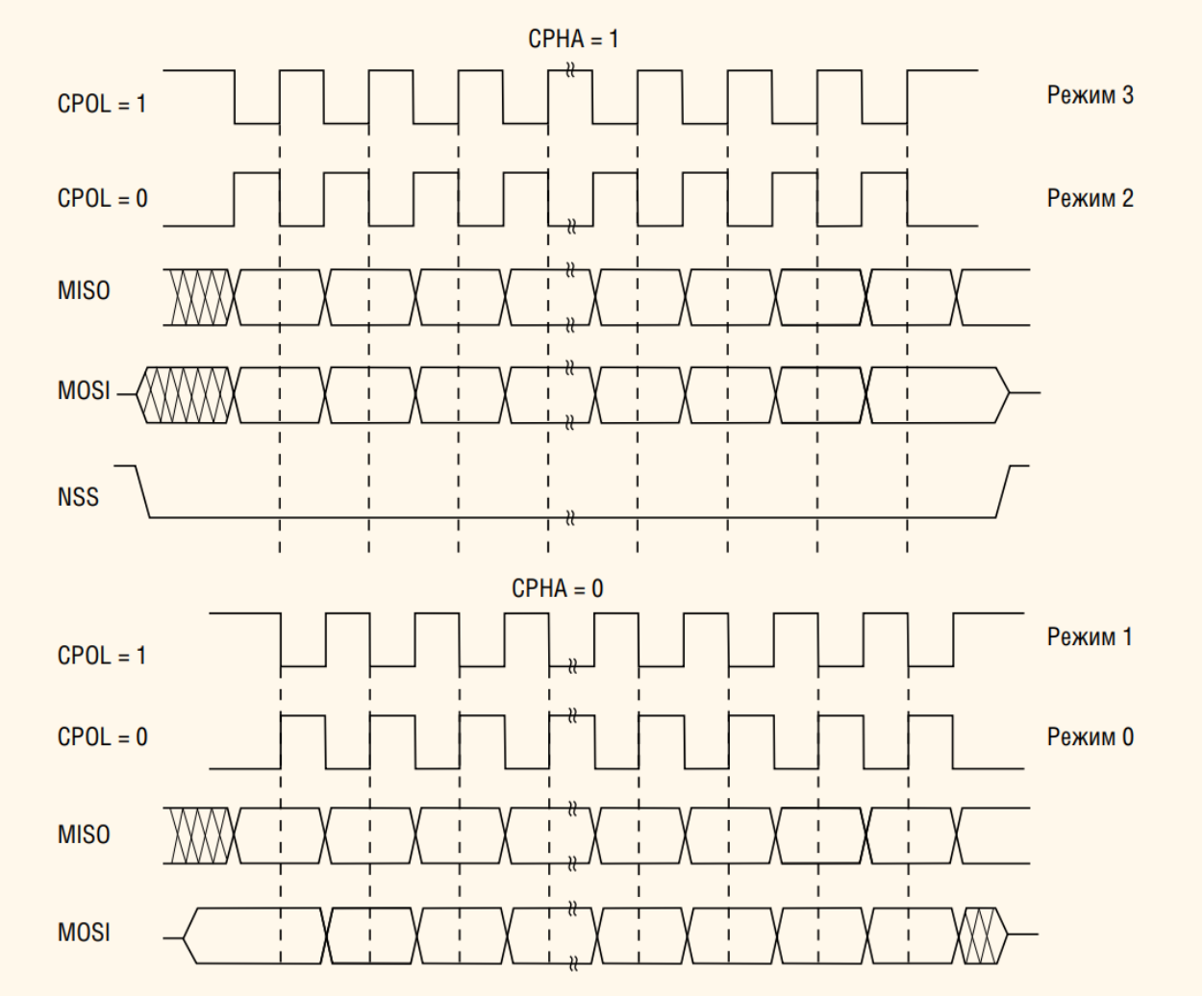


Рисунок 21 – Диаграмма режимов работы интерфейса SPI

Микроконтроллер позволяет для каждого интерфейса SPI задать полярность и фазу тактового сигнала, определяя тем самым режим его работы. Кроме того, для микроконтроллера можно установить формат передачи данных 8-разрядными или 16-разрядными словами и определить порядок передачи данных – старшим или младшим битом вперёд. Это

позволяет микроконтроллеру с помощью обоих интерфейсов SPI обмениваться информацией с любыми другими SPI-устройствами.

В данном проекте SPI используется для связи с LCD-дисплеем и с последовательной FLASH-памятью. Настройка SPI показана на рисунке 22.

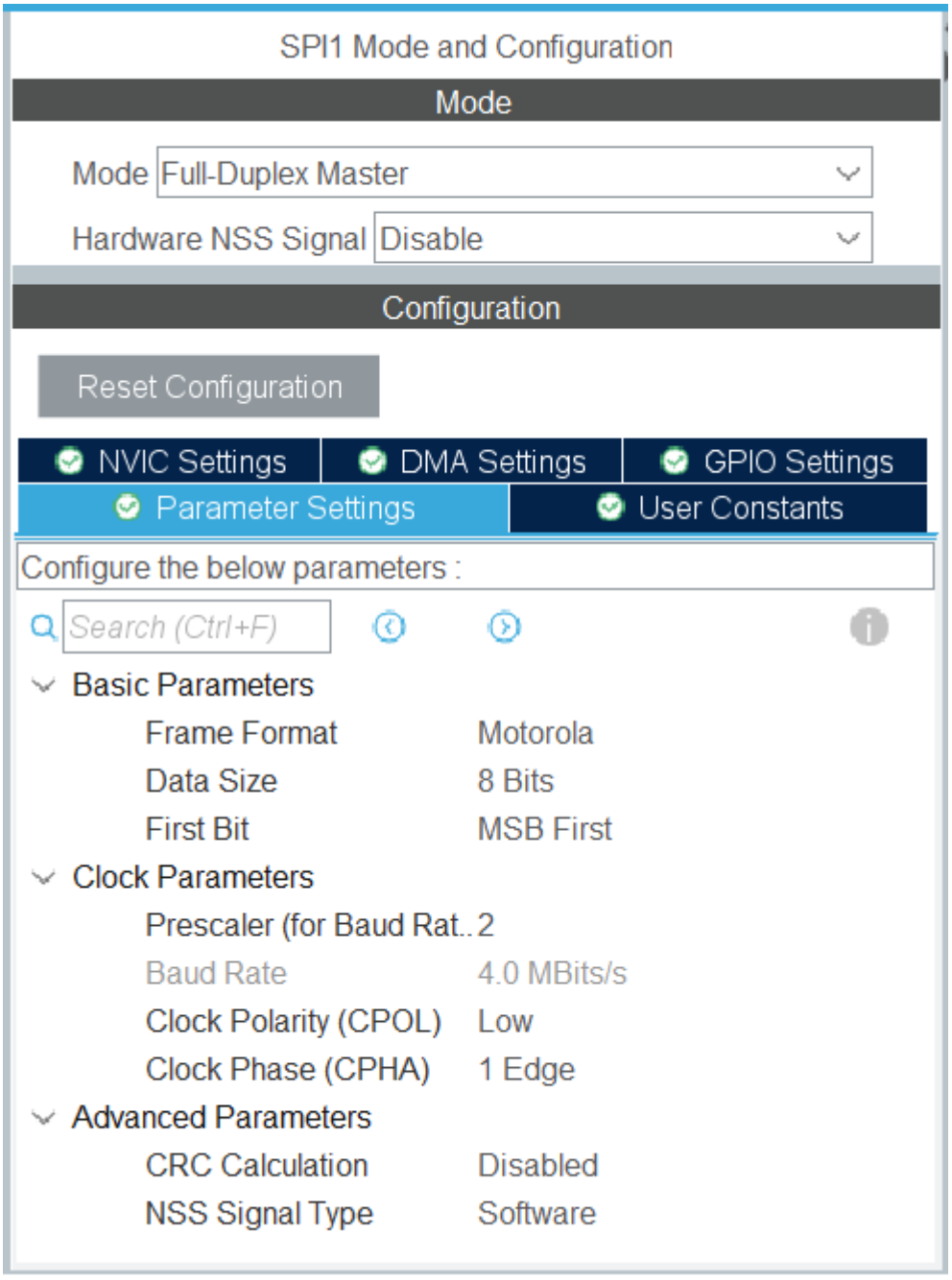


Рисунок 22 – Настройка SPI1 для взаимодействия с дисплеем

Таким образом, SPI1 работает только в режиме приема/передачи, размер посылки – 8бит, прескейлер – 2, активным уровнем сигнала

синхронизации является нулевой потенциал, а запись данных может производится по фронту импульса данного синхросигнала.

Настройка SPI1 в коде по указанным выше параметрам показана в листинге 2.

Листинг 2 – Настройка SPI1

```
static void MX_SPI1_Init(void)
{
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
}
```

Рассмотрим внутреннюю архитектуру SPI микроконтроллера STM32, которая представлена на рисунке 23.

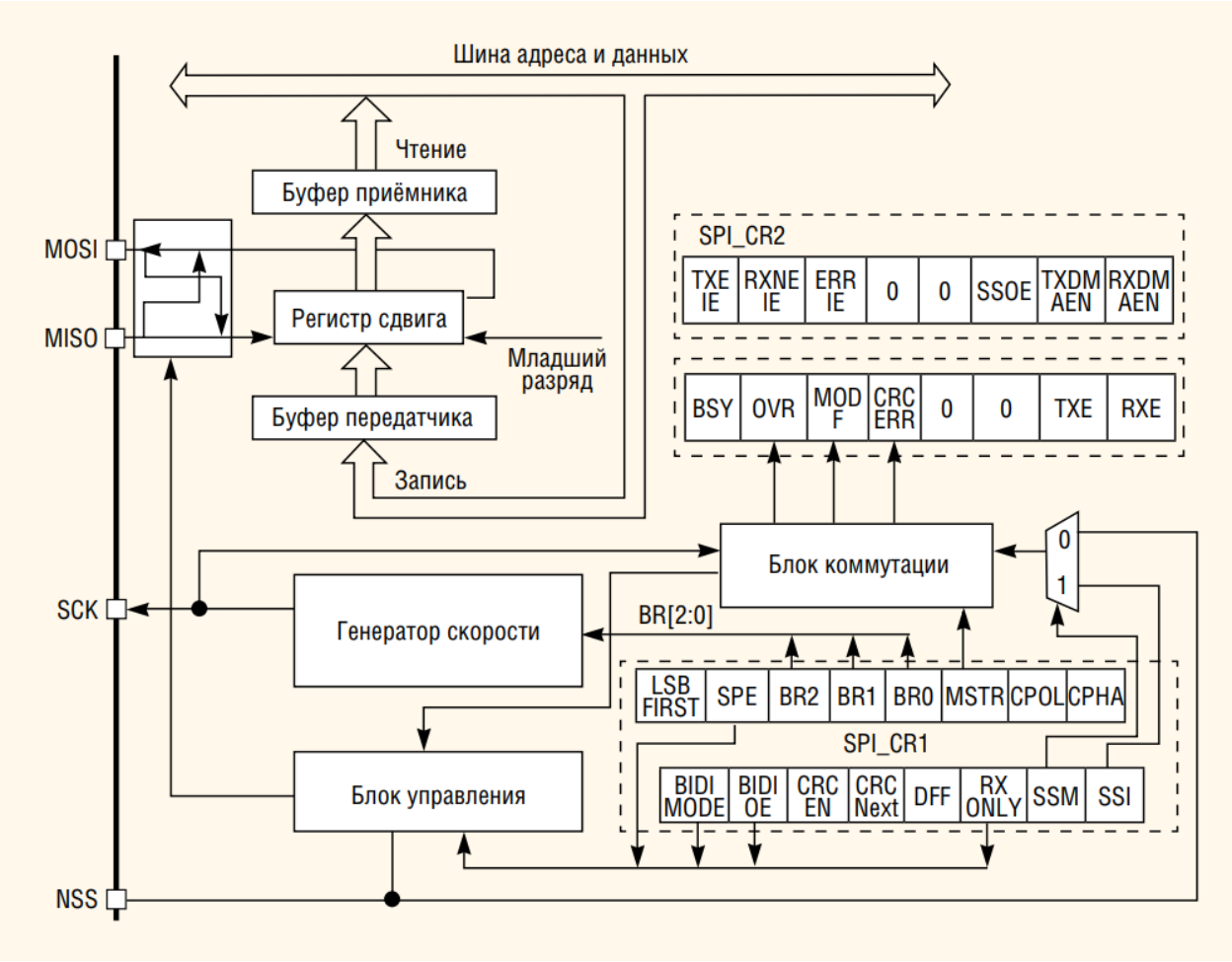


Рисунок 23 – Архитектура SPI МК семейства STM32

Регистр сдвига представляет собой основной регистр, через который передаются и принимаются данные. Если интерфейс SPI работает в режиме ведущего устройства, то вход этого сдвигового регистра соединён с выводом MISO, а выход – с выводом MOSI.

В режиме ведомого устройства происходит обратное переключение, которое регулирует блок управления. Для передачи данных их необходимо записать в регистр передатчика. Принятые данные читаются из регистра приёмника.

Для программы существует один регистр с именем SPI_DR. При чтении этого регистра происходит обращение к регистру приёмника, а при записи – к регистру передатчика. Скорость обмена по SPI определяет блок генератора скорости, который задаёт частоту следования тактовых

импульсов. Для этого предназначены разряды BR0, BR1 и BR2 регистра SPI_CR1. Три разряда предполагают наличие восьми значений скорости. Таким образом, скорость обмена данными по интерфейсу SPI для микроконтроллера STM32 с тактовой частотой 24 МГц может изменяться от 24 МГц/2=12 Мбод до 24 МГц/8=3 Мбод.

Для работы с интерфейсом SPI в микроконтроллере STM32 имеются специальные регистры. Формат этих регистров с названием входящих в них разрядов представлен на рисунке 24.

Сдвиг	Регистр	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0 × 00	SPI_CR1	Резерв																BIDMODE	BIDIOE	CRCEN	CRCNEXT	DFF	RXONLY	SSM	SSI	LSBFIRST	SPE	BR[2:0]			MSTR	CPOL	CPHA
	Исх. значение																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 × 04	SPI_CR2	Резерв																								TXEE	RXNEIE	ERRIE	Резерв		SSDE	TXDMAEN	RXDMAEN
	Исх. значение																									0	0	0			0	0	0
0 × 08	SPI_SR	Резерв																								BSY	OVR	MODE	CRCERR	Резерв	TXE	RXNE	
	Исх. значение																									0	0	0	0		1	0	
0 × 0C	SPI_DR	Резерв																DR[15:0]															
	Исх. значение																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 × 10	SPI_CRCPR	Резерв																CRCPOLY[15:0]															
	Исх. значение																	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
0 × 14	SPI_RXCRCR	Резерв																RXCRC[15:0]															
	Исх. значение																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 × 18	SPI_TXCRRCR	Резерв																TXCRC[15:0]															
	Исх. значение																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Рисунок 24 – Формат регистров SPI

Регистры:

– SPI_CR1 – первый управляющий регистр;

– SPI_CR2 – второй управляющий регистр;

– SPI_SR – регистр статуса;

– SPI_DR – регистр данных;

– SPI_CRCPR – регистр, содержащий полином для вычисления CRC;

– SPI_RXCRCR – регистр, содержащий CRC принятых данных;

– SPI_TXCRCR – регистр, содержащий CRC передаваемых данных.

Некоторые из этих регистров используются для работы в режиме I2S.

Регистр SPI_CR1 является первым управляющим регистром интерфейса SPI. Он имеет вид, представленный на рисунке 25.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDI OE	CRC EN	CRC NEXT	DFF	RX ONLY	SSM	SSI	*LSB FIRST*	SPE	BR [2:0]			MSTR	CPOL	CPHA
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 25 – Регистр SPI_CR1

0. CPHA задаёт фазу тактового сигнала;

1. CPOL устанавливает полярность тактового сигнала;

2. MSTR назначает режим работы интерфейса (0 – ведомый, 1 – ведущий);

5…3. 0BR [2:0] задают скорость обмена (000 – $f_{PCLK}/2$, 001 – $f_{PCLK}/4$, 010 – $f_{PCLK}/8$, 011 – $f_{PCLK}/16$, 100 – $f_{PCLK}/32$, 101 – $f_{PCLK}/64$, 110 – $f_{PCLK}/128$, 111 – $f_{PCLK}/256$);

6. SPE управляет интерфейсом (0 – отключает, 1 – включает);

7. LSBFIRST задаёт направление передачи (0 – младшим разрядом вперёд, 1 –старшим разрядом вперёд);

8. SSI определяет значение NSS при SSM=1;

9. SSM выбирает источник сигнала NSS (0 – с внешнего вывода, 1 – программно от разряда SSI);

10. RX ONLY совместно с битом BIDIMODE определяет направление передачи в однонаправленном режиме;

11. DFF определяет формат данных (0–8 бит, 1–16 бит);

12. CRCNEXT управляет передачей кода CRC (0 – данные, 1 – CRC);

13. CRCEN регулирует аппаратное вычисление CRC (0 – запрещено, 1 – разрешено). Для корректной операции этот бит должен записываться только при отключённом интерфейсе SPI, когда SPE = 0;

14. BIDIOE совместно с битом BIDIMODE управляет двунаправленным режимом работы интерфейса (0 – приём, 1 – передача);

15. BIDIMODE управляет двунаправленным режимом работы интерфейса (0 – двухпроводный однонаправленный режим, 1 – однопроводной двунаправленный режим).

SPI_CR2 является вторым управляющим регистром интерфейса SPI и имеет вид, показанный на рисунке 26.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TXEIE	RXNEIE	ERRIE	Res.	Res.	*SSOE*	TXDMAEN	RXDMAEN
								rw	rw	rw			rw	rw	rw

Рисунок 26 – Регистр SPI_CR2

0. RXDMAEN – запросом DMA для приёмника (0 – запрещает, 1 – разрешает);

1. TXDMAEN – запросом DMA для передатчика (0 – запрещает, 1 – разрешает);

2. SSOE – сигналом NSS в режиме мастера (0 – запрещает, 1 – разрешает);

5. ERRIE – прерыванием в случае ошибки (0–запрещает 1–разрешает);

6. RXNEIE – прерыванием приёма данных (0–запрещает 1–разрешает);

7. TXEIE – управляет прерыванием передачи данных (0–запрещает 1–разрешает);

Регистр статуса SPI_SR имеет вид, показанный на рисунке 27.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								BSY	OVR	MODF	CRC ERR	UDR	CHSIDE	TXE	RXNE
								r	r	r	rc_w0	r	r	r	r

Рисунок 27 – Регистр SPI_SR

0. RXNE устанавливается, если в буфере приёмника есть принятые данные;

1. TXE – устанавливается, если буфер передатчика пуст и готов принять новые данные;

2, 3. – Не используются в SPI;

4. CRCERR устанавливается при ошибке CRC при приёме данных;

5. MODF устанавливается, когда в режиме мастера к сигналу NSS прикладывается низкий потенциал;

6. OVR – флаг переполнения, устанавливается при приёме новых данных, если предыдущие не были прочитаны;

7. BSY – флаг занятости, устанавливается, если интерфейс занят обменом данных или буфер данных передатчика не пустой.

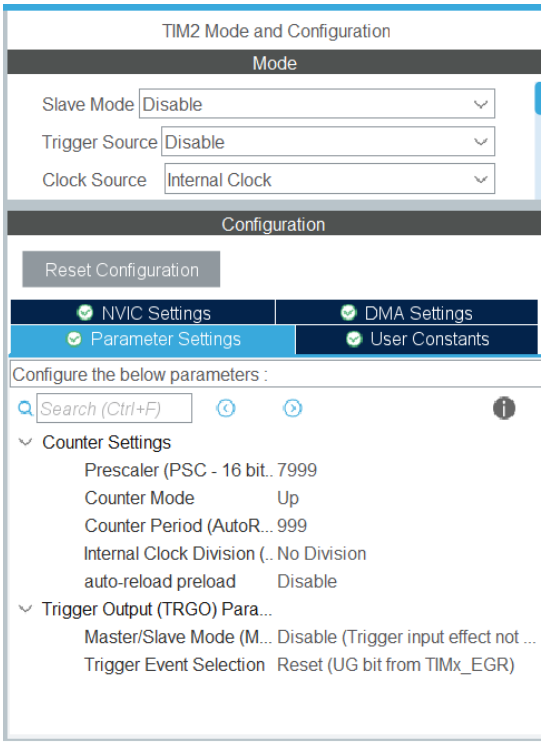
Регистр данных SPI_DR состоит из 16 разрядов данных. В этот регистр данные записываются для передачи и читаются из него при приёме.

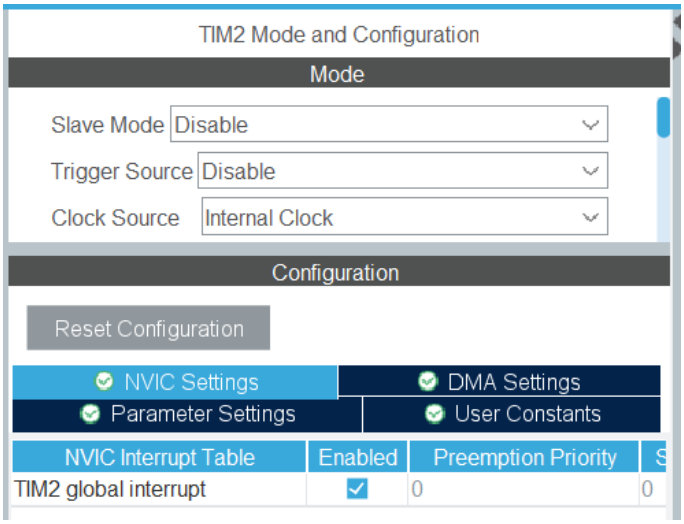
1.2.7 Использование таймера для отсчета времени

Микроконтроллер STM32F103C8T6 имеет в своём составе 16-битных 4 таймера с большим количеством поддерживаемых функций. С помощью любого таймера можно формировать интервалы времени с требуемой длительностью с генерацией прерывания или DMA запроса по окончании интервала. Кроме того, можно формировать одиночные импульсы заданной длительности или периодические импульсы с заданной длительностью и частотой повторения; подсчитывать количество импульсов внешнего сигнала (счётчик может работать в режиме сложения или вычитания); поддерживается режим широтно-импульсной модуляции.

Среди вышеупомянутых 4 таймеров 3 таймера общего назначения (TIM2-TIM4) и один таймер с расширенным функционалом (TIM1). Так как каждый из названных таймеров способен генерировать прерывания по окончании интервала времени, было принято решение взять один из

таймеров общего назначения чтобы уменьшить количество необходимых настроек и соответственно снизить вероятность ошибки или некорректной работы. Таким образом, был выбран TIM2. Он используется для генерации прерываний по окончании интервала жизни временных файлов. Его конфигурация представлена на рисунках 28 и 29.





Рисунки 28,29 – Конфигурация таймера TIM2 для генерации прерываний по истечении временного интервала

Таким образом, таймер имеет предделитель 7999, считает через увеличение числа, считает до 999 с предзагрузкой автоматического сброса. Реализация вышеназванных параметров в коде отображена в листинге 3.

Листинг 3 – Инициализация таймера TIM2

```
static void MX_TIM2_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 7999;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 999;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
```

```
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
}
```

Далее рассмотрим, как именно данные настройки влияют на реальную работу таймера.

Структурная схема таймера представлена на рисунке 30.

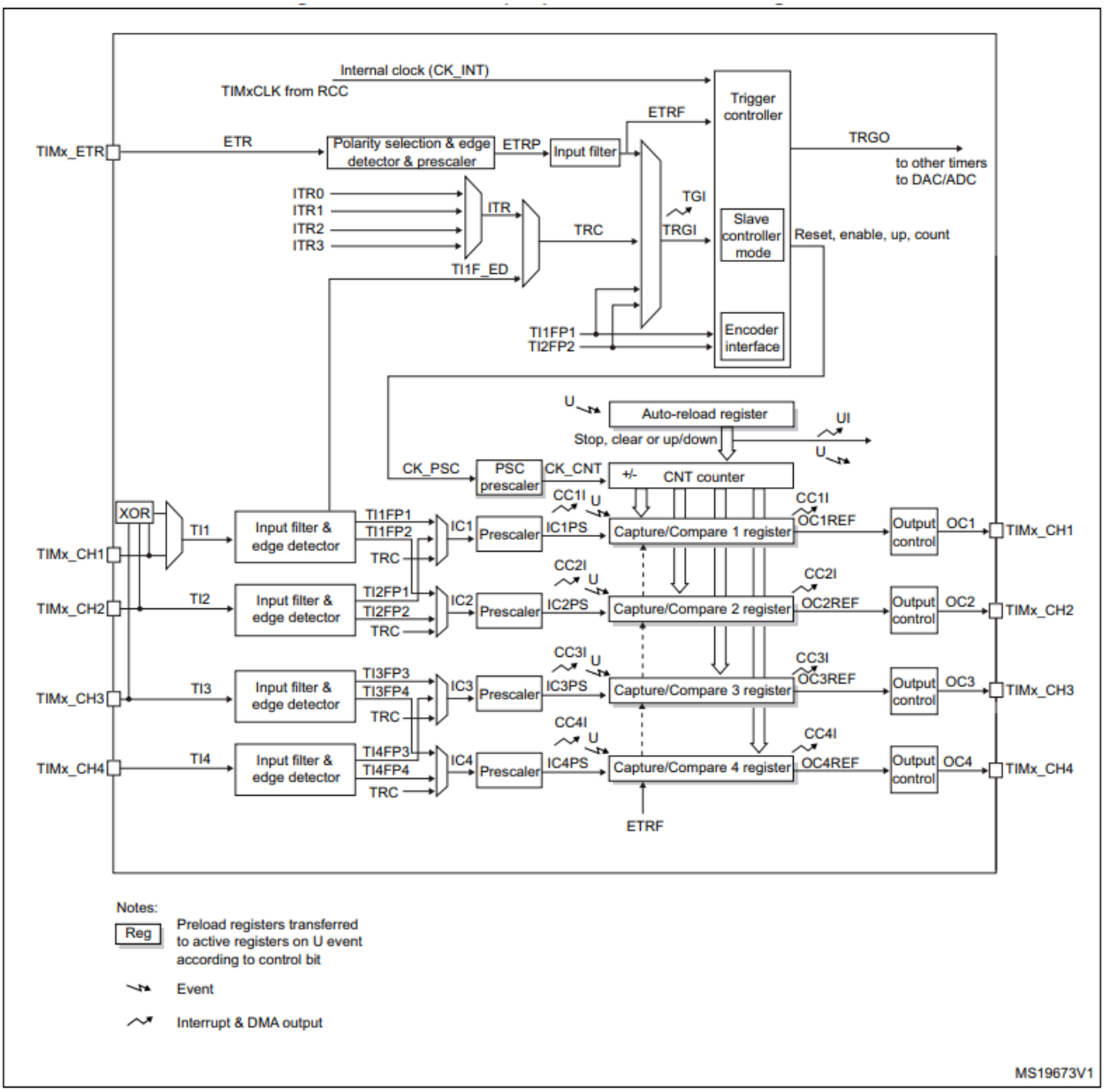


Рисунок 30 – Структурная схема таймера

Таймер непосредственно работает через регистры TIM2_CNT, TIM2_PSC и TIM2_ARR. Они полностью содержат в себе значения текущего счетчика таймера, значение прескейлера и значение автоматической перезагрузки соответственно, и более подробно их рассматривать смысла нет. Прескейлер и автоматическая перезагрузка были выставлены как 7999 и 999 соответственно.

Также таймер имеет множество регистров для настройки режима работы. Я рассмотрю только те, которые необходимо было настроить вручную. Начнем с регистра TIM2_CR1, представленного на рисунке 31.

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 31 – Регистр TIM2_CR1

CKD: Clock division

ARPE: Auto-reload preload enable. Бит для включения режима предзагрузки регистра TIMx_ARR: 0: TIMx_ARR не буферизируется; 1: используется буферизация регистра TIMx_ARR. Когда буферизация включена, новое значение, записанное в регистр, начинает использоваться после очередного события обновления.

CMS: Center-aligned mode selection

DIR: Direction

OPM: One-pulse mode.

URS: Update request source.

UDIS: Update disable.

CEN: Counter enable.

На рисунке 32 изображен регистр TIM2_DIER, в котором нам нужно установить бит UIE (Update Interrupt Enable), чтобы разрешить прерывания.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

Рисунок 32 – Регистр TIM2_DIER

TDE: Trigger DMA request enable.

CC4DE: Capture/Compare 4 DMA request enable.

CC3DE: Capture/Compare 3 DMA request enable.

CC2DE: Capture/Compare 2 DMA request enable.

CC1DE: Capture/Compare 1 DMA request enable.

UDE: Update DMA request enable.

TIE: Trigger interrupt enable.

CC4IE: Capture/Compare 4 interrupt enable.

CC3IE: Capture/Compare 3 interrupt enable.

CC2IE: Capture/Compare 2 interrupt enable.

CC1IE: Capture/Compare 1 interrupt enable.

UIE: Update interrupt enable.

1.2.8 Построение функциональной схемы

На основе всех вышеописанных сведений была спроектирована функциональная схема разрабатываемой системы, показанная на рисунке 33[6, 7].

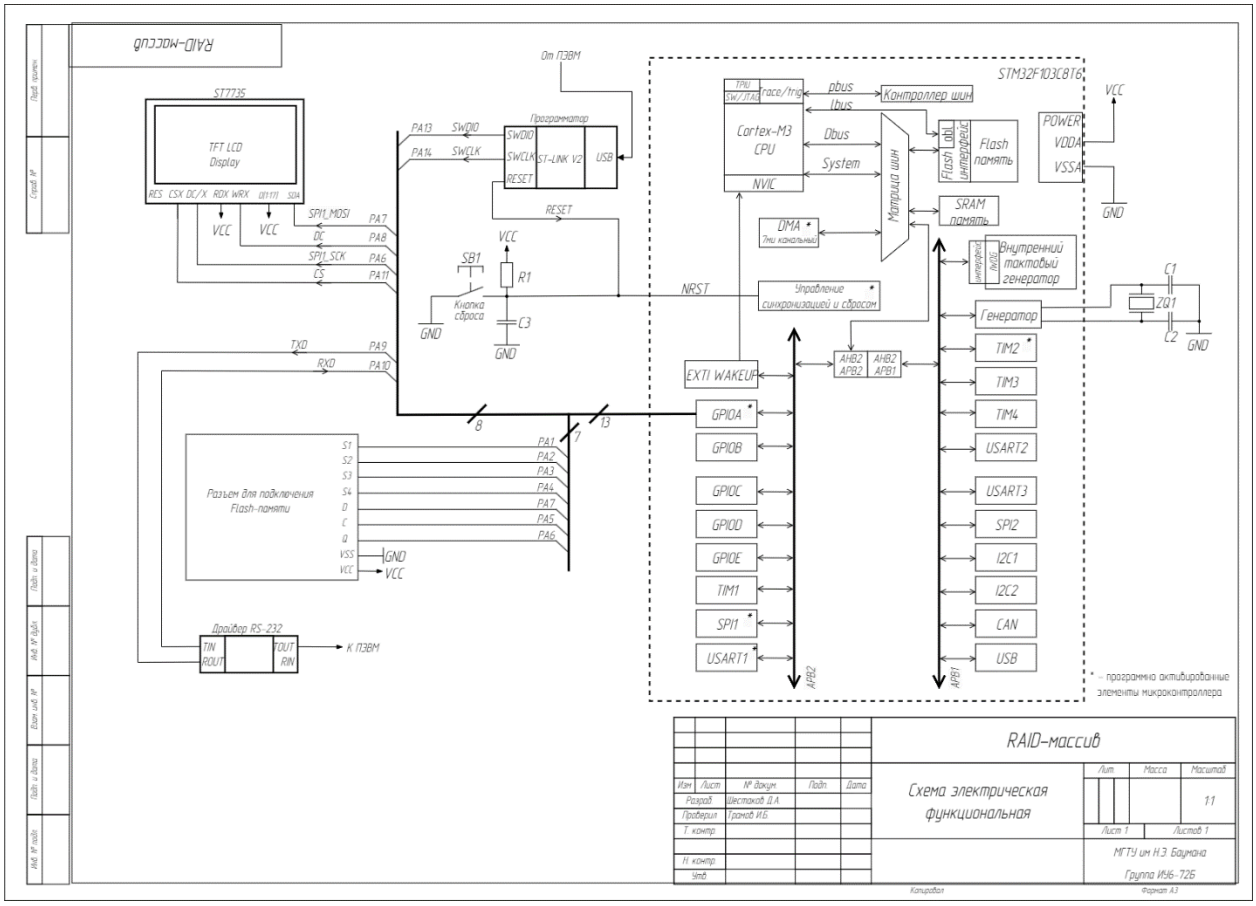


Рисунок 33 – Функциональная схема RAID-массива

1.3 Проектирование принципиальной схемы

1.3.1 Разъем программатора

Для программирования МК используется специальный программатор ST-LINK V2. Подключение программатора осуществляется при помощи портов PA13 и PA14, которые выполняют роль SWDIO и SWCLK соответственно.

Он имеет следующие разъемы для подключения к МК:

– SWCLK – тактовый сигнал;

– SWDIO – для передачи данных;

– RST – сигналом на RST программатор вводит контроллер в режим программирования.

1.3.2 Расчет потребляемой мощности

Потребляемая мощность – это мощность, потребляемая интегральной схемой, которая работает в заданном режиме соответствующего источника питания.

Чтобы рассчитать суммарную мощность, рассчитаем мощность каждого элемента. На все микросхемы подается напряжение +3.3В. Мощность, потребляемая один устройством, в статическом режиме, рассчитывается формулой:

$$P = U * I$$

где U – напряжение питания (В);

I – ток потребления микросхемы (мА).

Также в схеме присутствуют резисторы CF-100. Мощность для резисторов рассчитывается по формуле:

$$P = I^2 * R$$

где R – сопротивление резистора;

I – ток, проходящий через резистор.

Расчет потребляемого напряжения для каждой микросхемы показан в таблице 5.

Таблица 5 – Потребляемая мощность

Микросхема	Ток потребления, мА	Потребляемая мощность, мВт	Количество устройств	Суммарная потребляемая мощность, мВт
STM32F103 C8T6	150	495	1	495
MAX232	10	33	1	33
M45PE16	15	49,5	4	198
ST7735	40	132	1	132
CF-100	-	1	1	1

$P_{суммарная} = P_{STM32F103C8T6} + P_{MAX232} + P_{M45PE16} + P_{ST7735} + P_{CF-100} = 495 + 33 + 198 + 132 + 1 = 859$ мВт

Суммарная потребляемая мощность системы равна 859 мВт = 0,9 Вт.

1.3.3 Построение принципиальной схемы

На основе всех вышеописанных сведений была спроектирована принципиальная схема разрабатываемой системы, показанная на рисунке 34[6, 7].

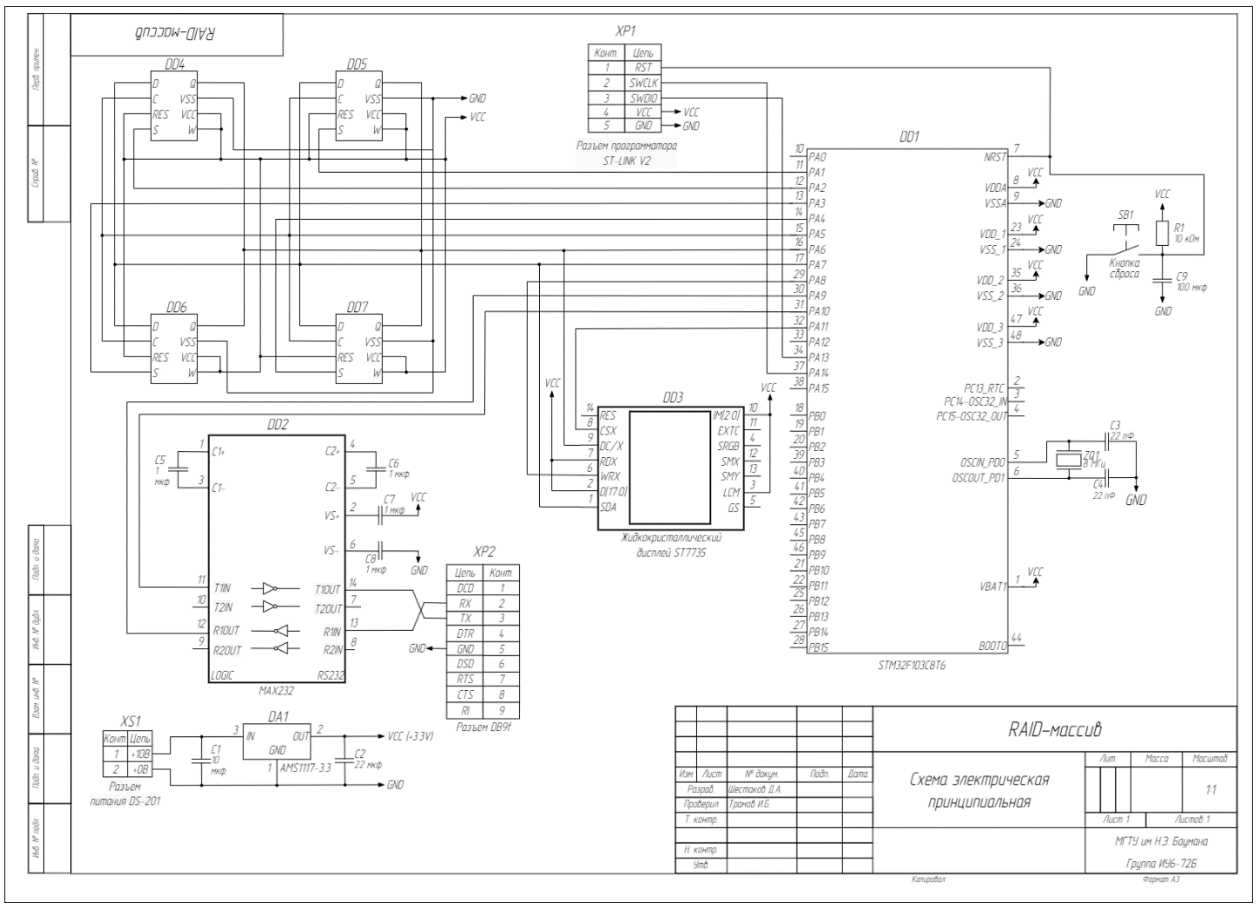


Рисунок 34 – Принципиальная схема

1.4 Алгоритмы работы системы

1.4.1 Общее описание работы программы

Работа начинается с функции main, из которой вызываются все остальные функции. Сначала идет инициализация интерфейсов, портов, таймера TIM2 и инициализация дисплея. Затем выделяется память под переменные, которые хранят значение текущего типа RAID, количество файлов, адреса конца последнего файла на последовательной FLASH-памяти, счетчик для таймера. Затем в бесконечном цикле обрабатывается ввод с ПЭВМ, и выполняются действия, соответствующие введенной команде. Верхнеуровневая схема алгоритма представлена на рисунке 35.

Рисунок 35 – Функция main

1.4.2 Детализация и пояснение основных функций

Рассмотрим более подробно выполнение команд из бесконечного цикла. В зависимости от кода команды, у нас выполняется или запись файлов на FLASH-память, или смена режима хранения RAID (с очисткой всех дисков), или очистка одного выбранного диска, или восстановление выбранного диска. Схема алгоритма выполнения команд представлена на рисунке 36.

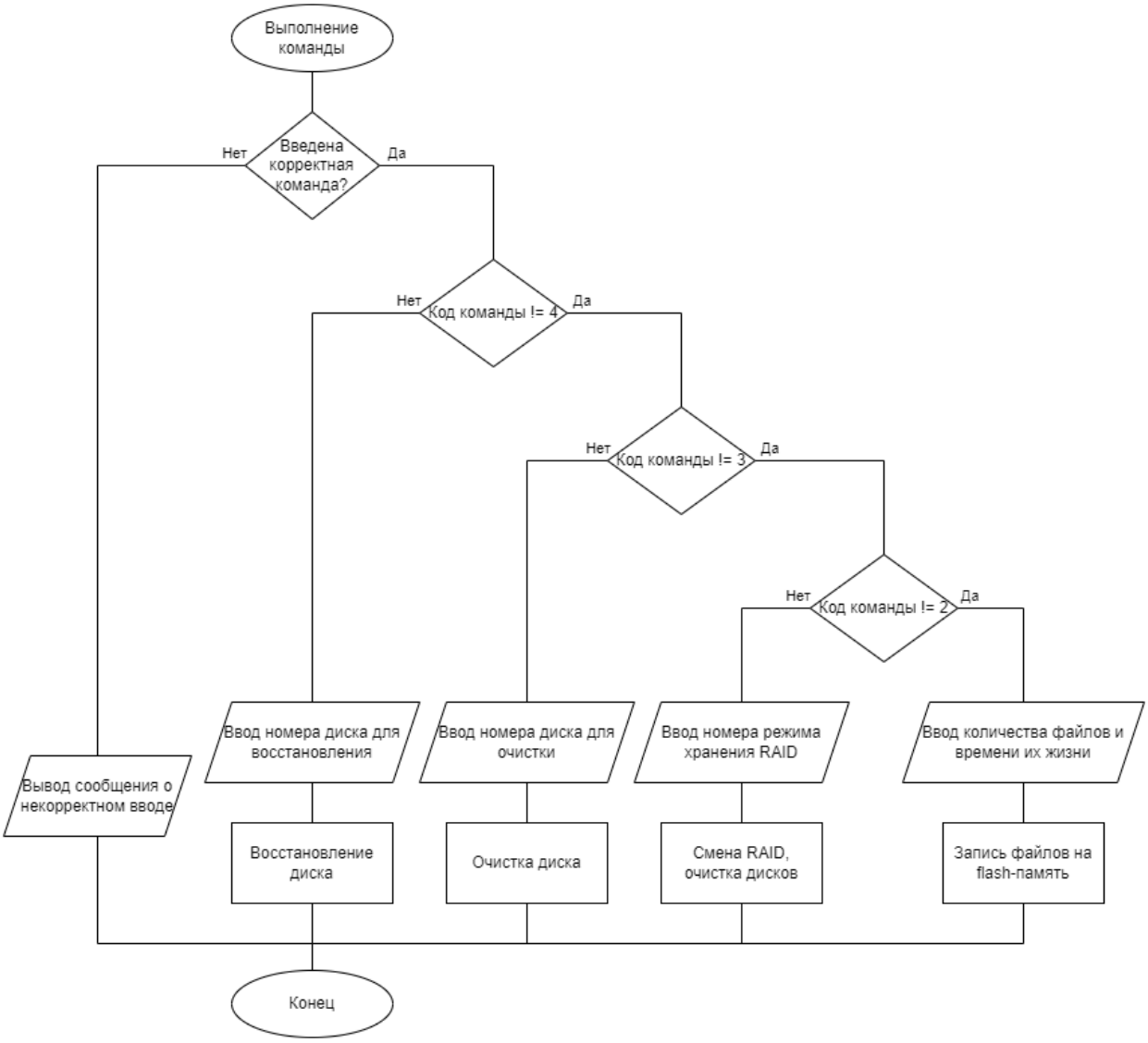


Рисунок 36 – Схема алгоритма функций в основном цикле

Восстановлении диска представляет собой последовательное считывание байтов с трех других дисков и запись XOR-а этих байтов на

восстанавливаемый диск. Алгоритм несложный, поэтому для него схему строить не будем.

Очистка диска представляет собой отправление в цикле адресов секторов с командой их очистки. Для этого алгоритма тоже не будем строить схему.

Наиболее интересным является запись файлов во FLASH-память, поскольку в зависимости от типа хранения RAID, у нас по-разному разбивается и распределяется файл.

Так для RAID 0 мы должны разбить файл на 4 части и записать по одной части на каждый диск.

При RAID 1 у нас файлы полностью записываются на первый и третий диск, пока в них есть место, а потом начинают записывать на второй и четвертый диск.

При RAID 3 файл разбивается на байты, которые последовательно записываются с первого по третий диск, а в четвертый диск записывается XOR этих байтов.

При RAID 4 файл разбивается на 3 части, которые записываются с первого по третий диск, а на четвертый диск записывается XOR этих частей. Поскольку размер файла может быть большим, то формировать блок на четвертый диск придется уже после записи блоков на первые три диска последовательным чтением байтов с этих дисков.

В RAID 5, как и в RAID 4, файл разбивается на 3 части, но блок четности теперь в зависимости от порядкового номера файла может записываться на любой диск.

Стоит отметить, что помимо самих данных, перед их началом будем записывать трехбайтовый адрес того, где начинаются следующие данные, что позволит нам понимать, где какой блок находится.

Еще одна деталь, которую нужно учитывать при записи файлов, заключается в том, что во FLASH-память за раз можно записать только одну

страницу, размер которой 256 байт. При этом, если при записи во FLASH-память вы достигните конца страницы, но не передадите новую инструкцию записи с адресом следующей страницы, то передаваемые байты начнут записываться с начала этой страницы. Оба этих факта учитываются при разработке функций записи.

Схемы алгоритмов записи файлов при разных RAID изображены на рисунках 37 – 41.

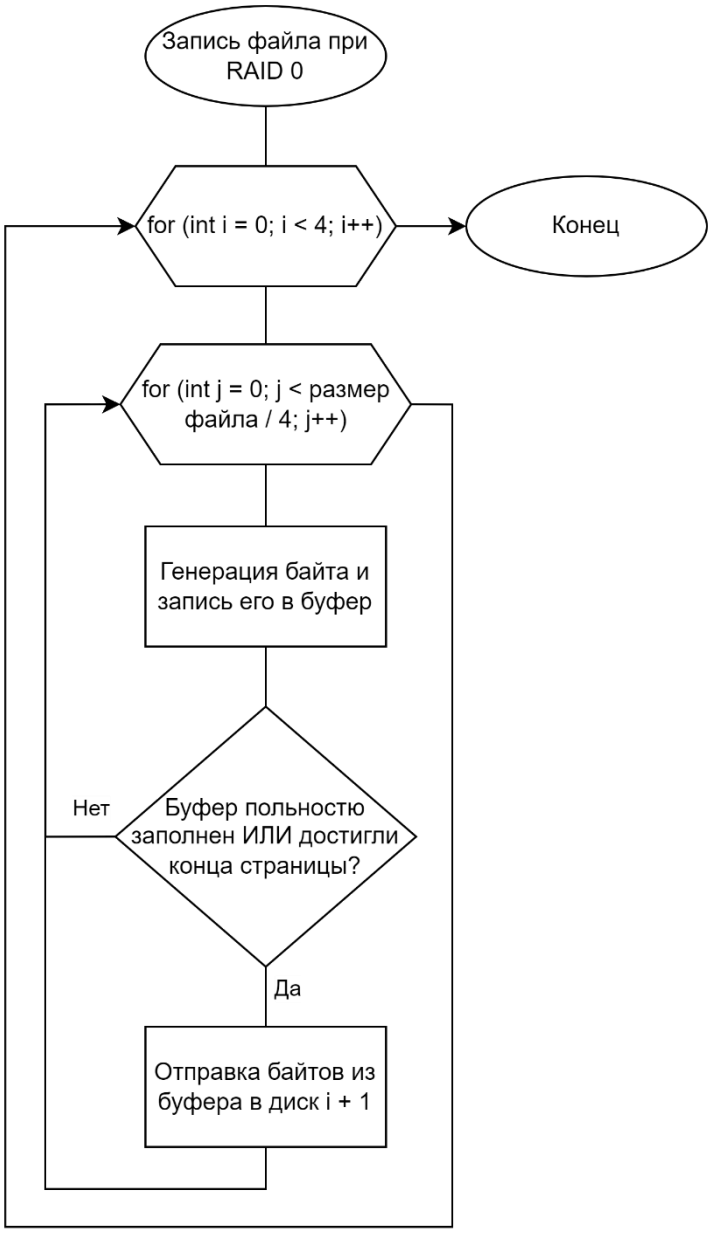


Рисунок 37 – Запись файла при RAID 0

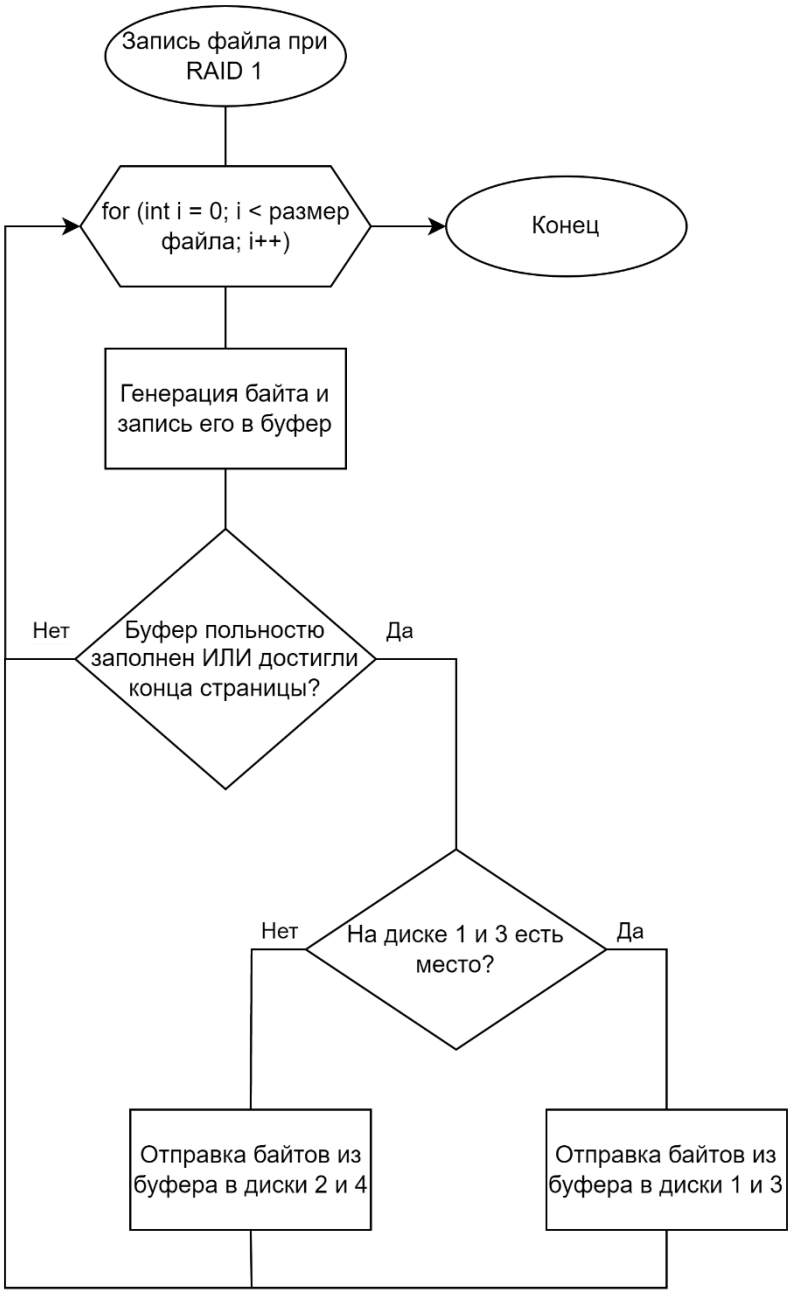


Рисунок 38 – Запись файла при RAID 1

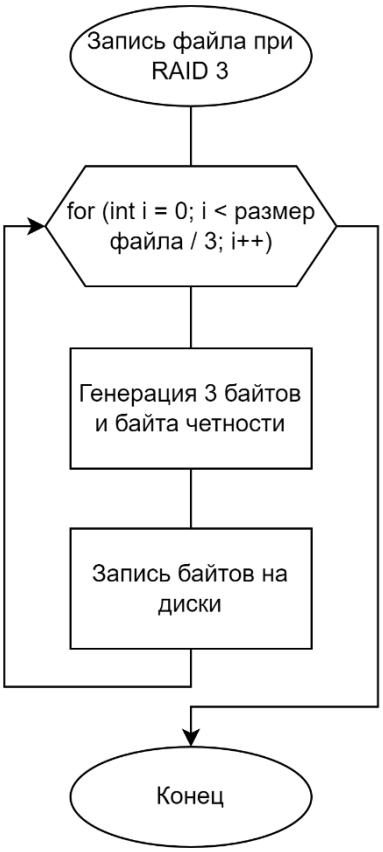


Рисунок 39 – Запись файла при RAID 3

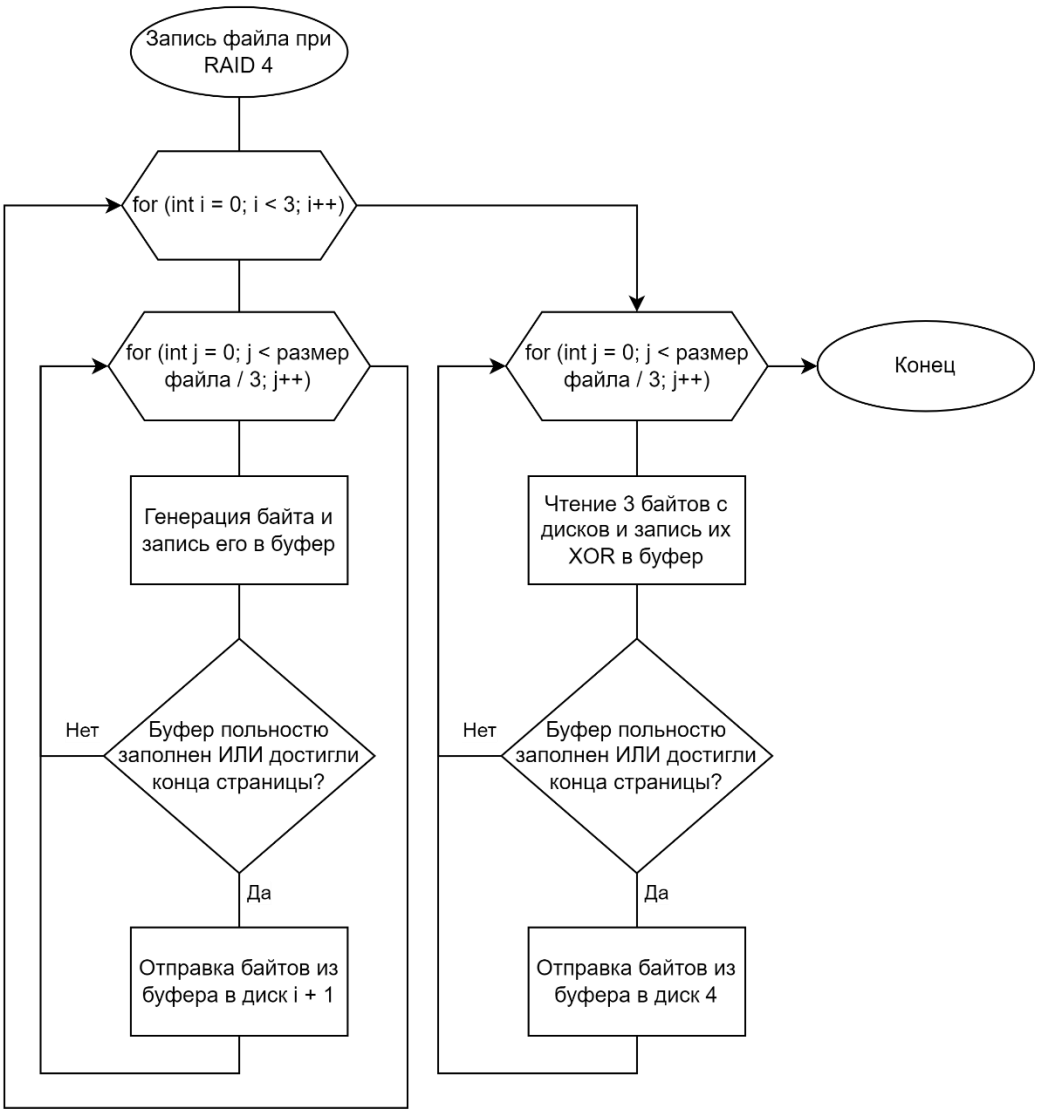


Рисунок 40 – Запись файла при RAID 4

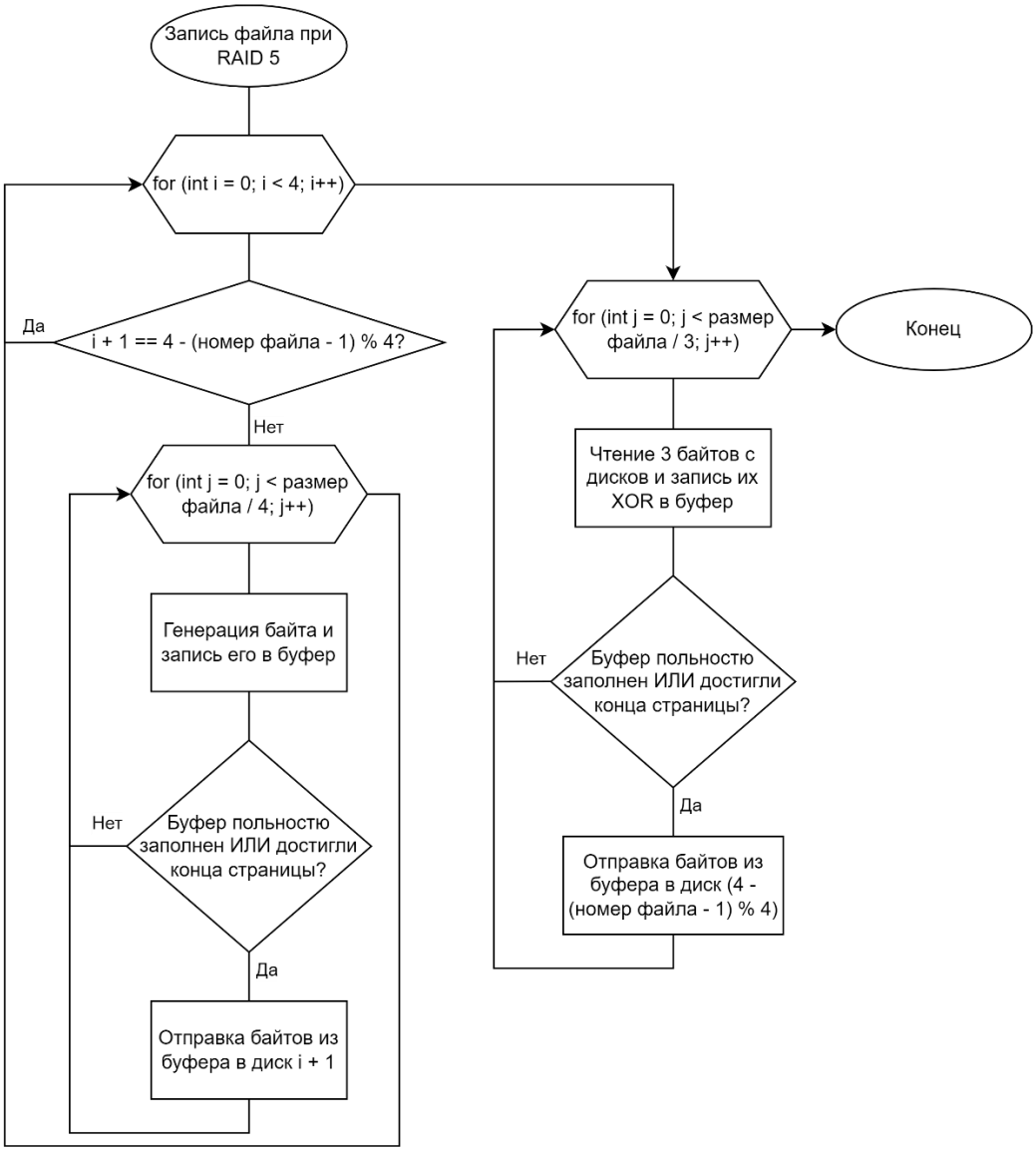


Рисунок 41 – Запись файла при RAID 5

2 Технологическая часть

Для реализации работы RAID-массива была написана программа на языке Си[10], после загруженная в МК. Симуляция проводилась в программе Proteus 8.

2.1 Отладка и тестирование программы

Программа была отлажена с использованием приложения Proteus 8. Это приложение предназначено для выполнения различных видов моделирования аналоговых и цифровых устройств. В ней наглядно было увидеть ввод и вывод на ПЭВМ (виртуальный терминал), увидеть вывод дисплея.

2.2 Симуляция работы системы

Для имитации реальных условий была использована программа Proteus. Схема системы в незапущенном состоянии изображена на рисунке 42.

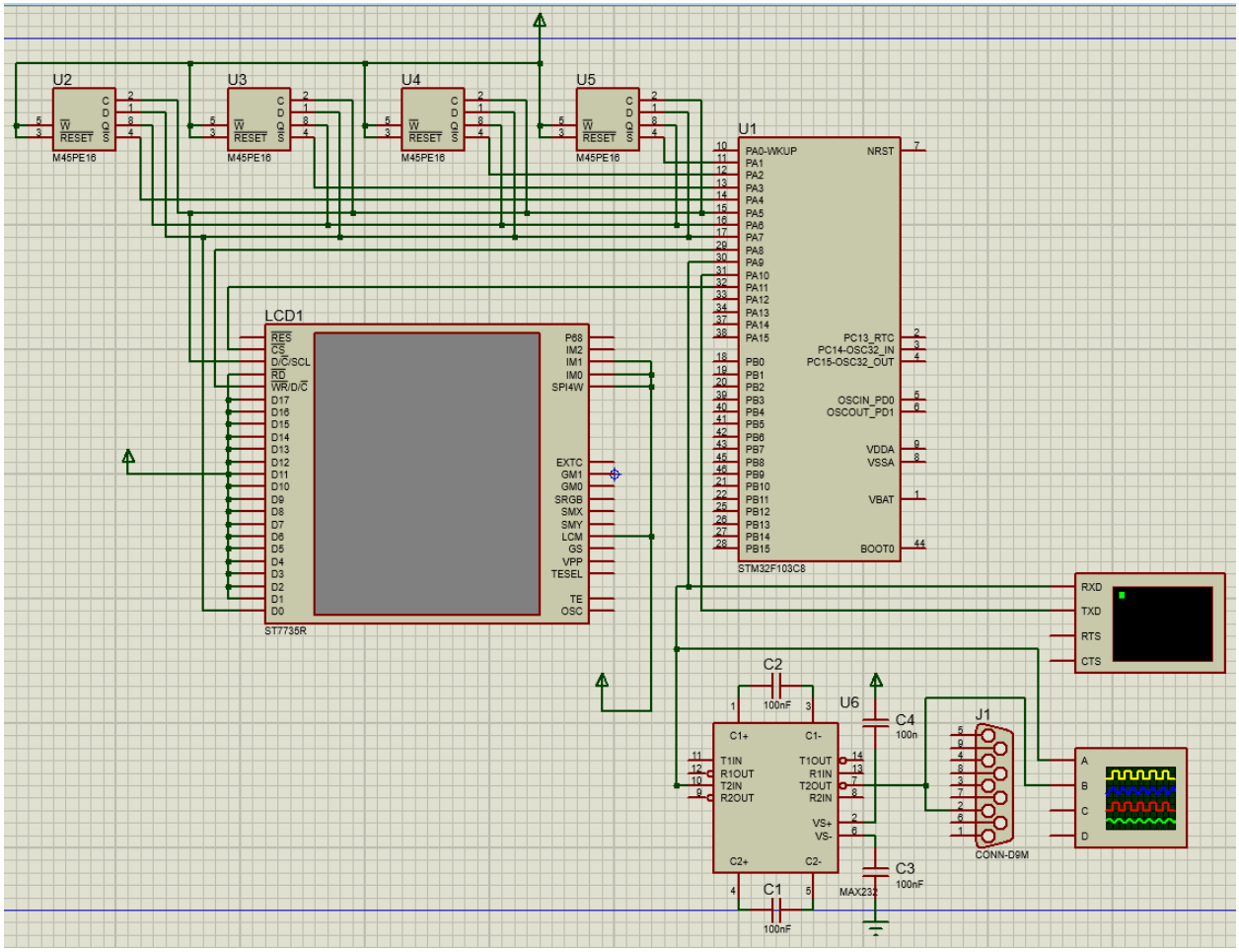


Рисунок 42 – Система в незапущенном состоянии

Модель в proteus отличается от принципиальной схемы отсутствием транзисторов, так как в симуляции подается стабильные 3.3 вольта и нет необходимости в распределении поступающего от портов тока.

Для моделирования ввода данных с ПЭВМ используется инструмент системы – Virtual Terminal. Он позволяет эмулировать простейший терминал, который даёт возможность передавать и получать данные по портам RxD и TxD через интерфейс USART. На рисунке 43 изображена включенная система.

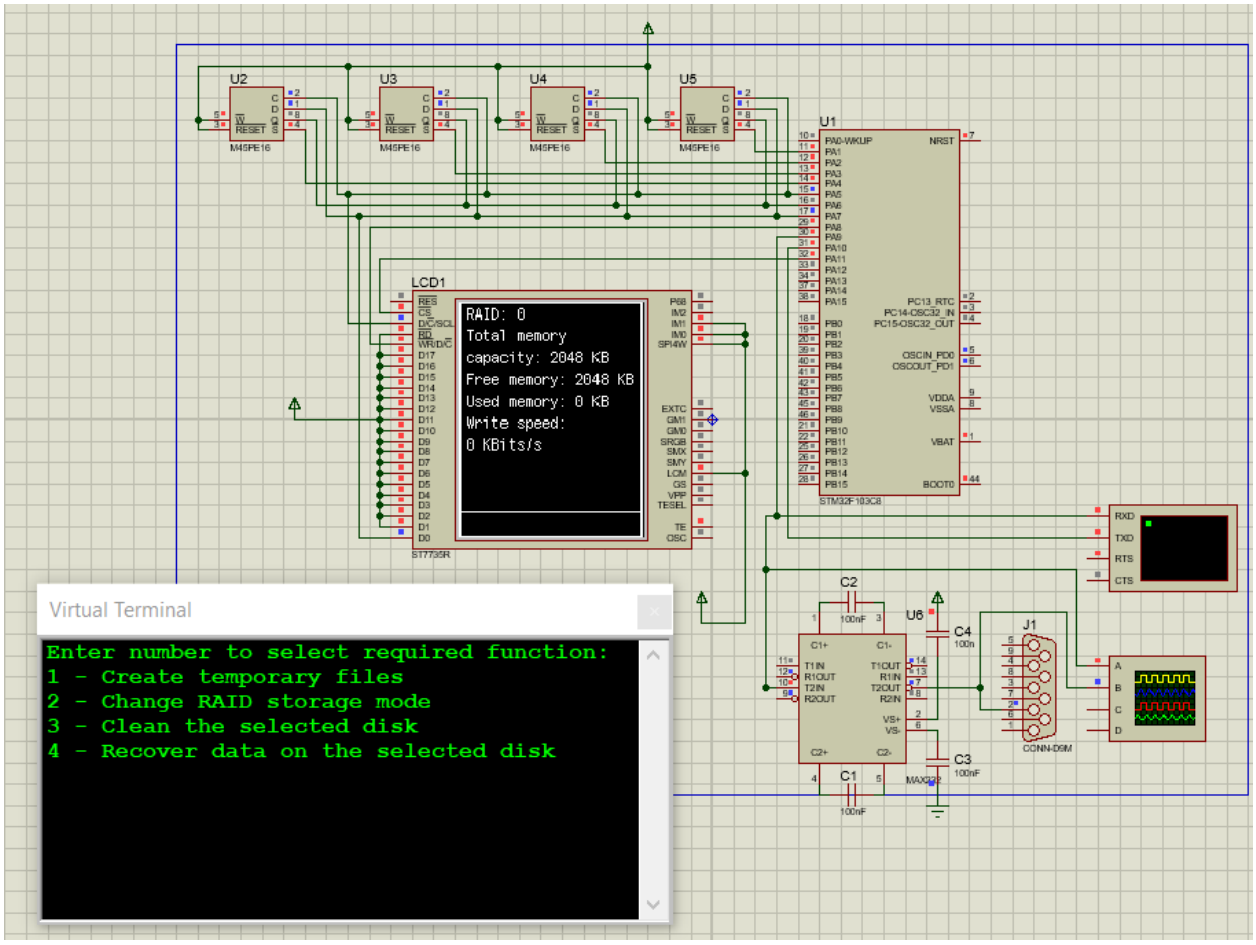


Рисунок 43 – Система в запущенном состоянии

В Virtual Terminal можно ввести цифру от 1 до 4, чтобы выбрать необходимую функцию. Пример использования первой функции показан на рисунке 44.

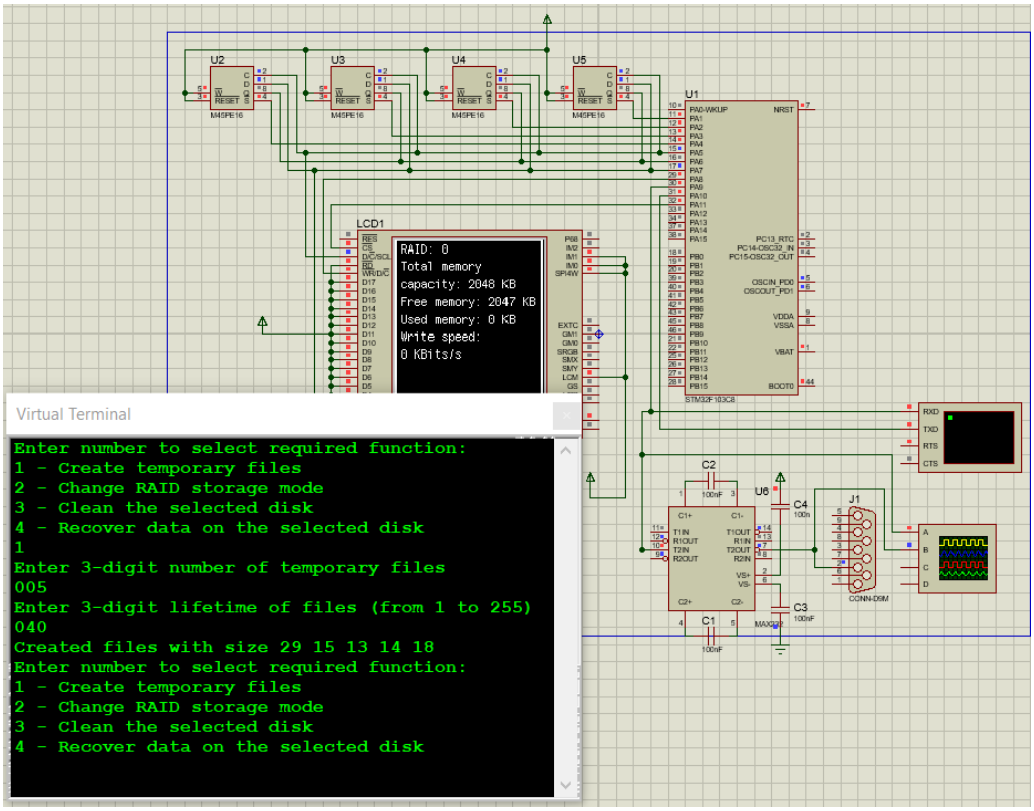


Рисунок 44 – Использование первой функции

По истечении времени жизни файлов, в терминал выведется информация об их удалении. Данная ситуация изображена на рисунке 45.

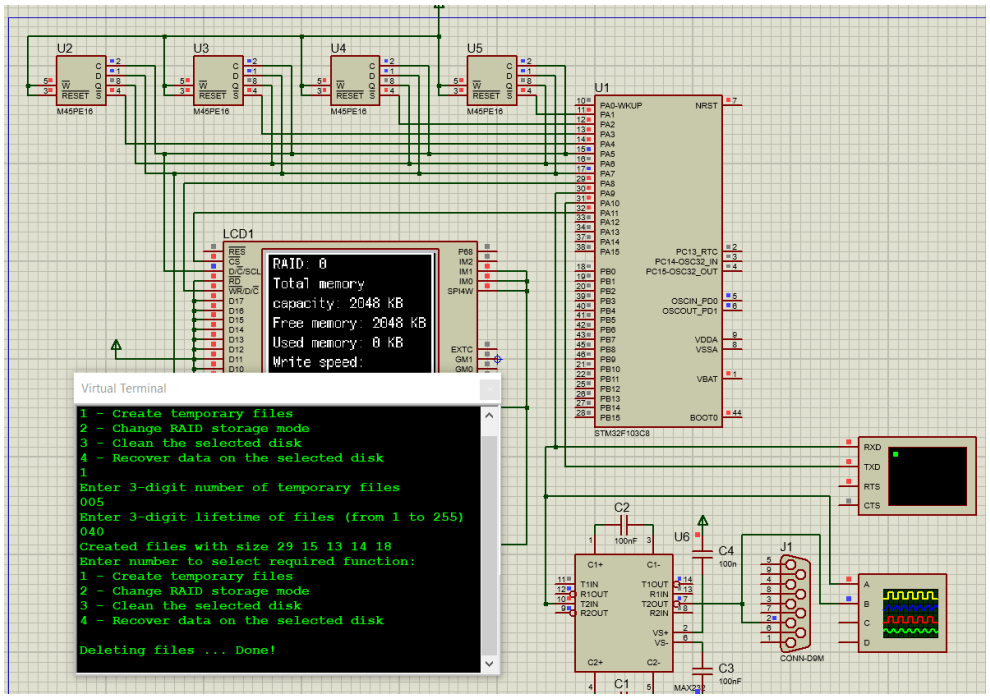


Рисунок 45 – Вывод сообщения об удалении временных файлов

На рисунке 46 изображено использование второй функции.

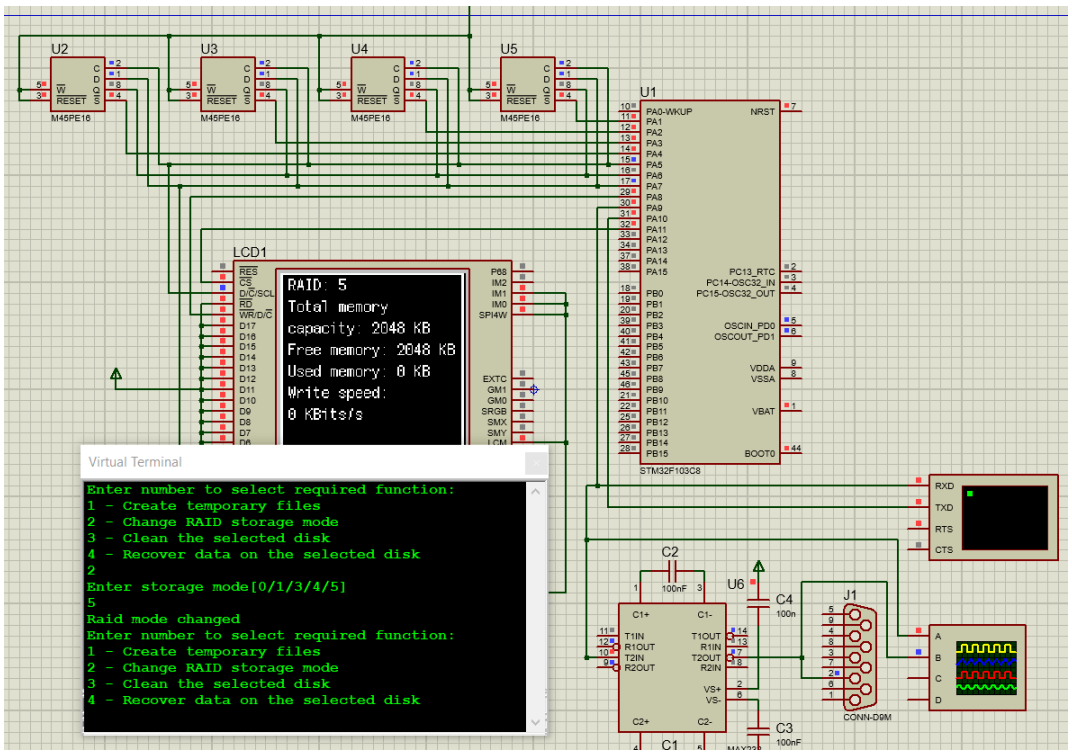


Рисунок 46 – Использование второй функции

На рисунке 47 изображено использование третьей функции.

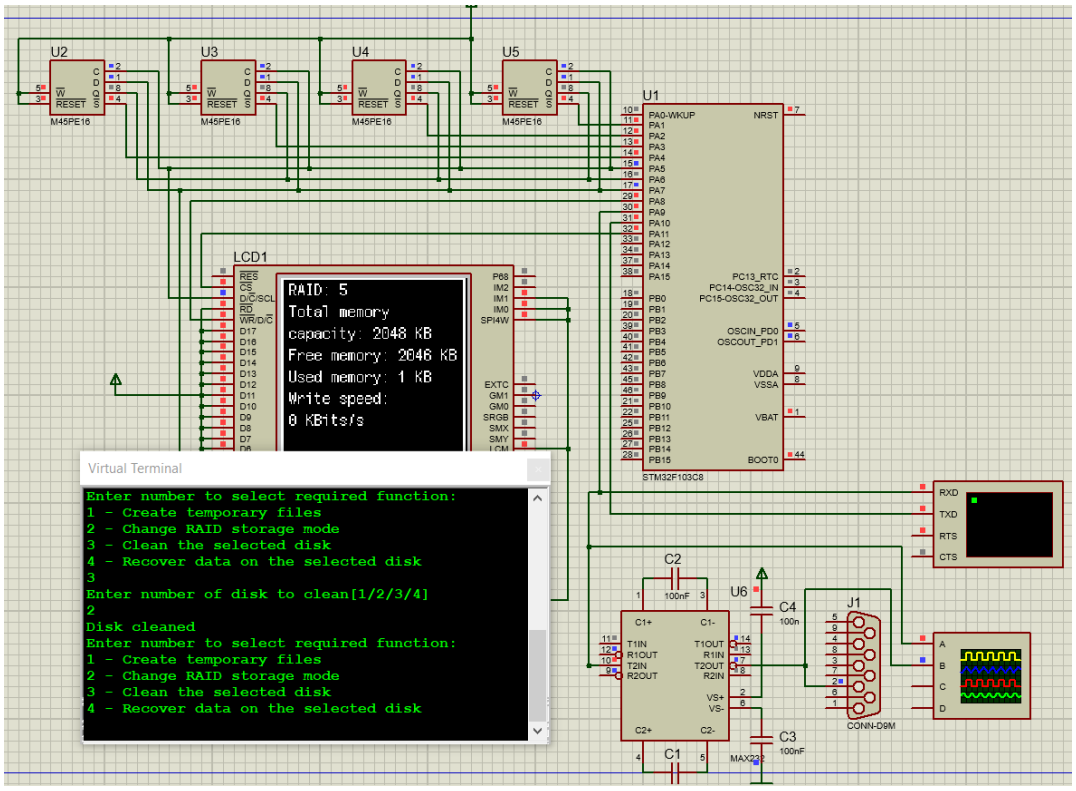


Рисунок 47 – Использование третьей функции

На рисунке 48 изображено использование четвертой функции.

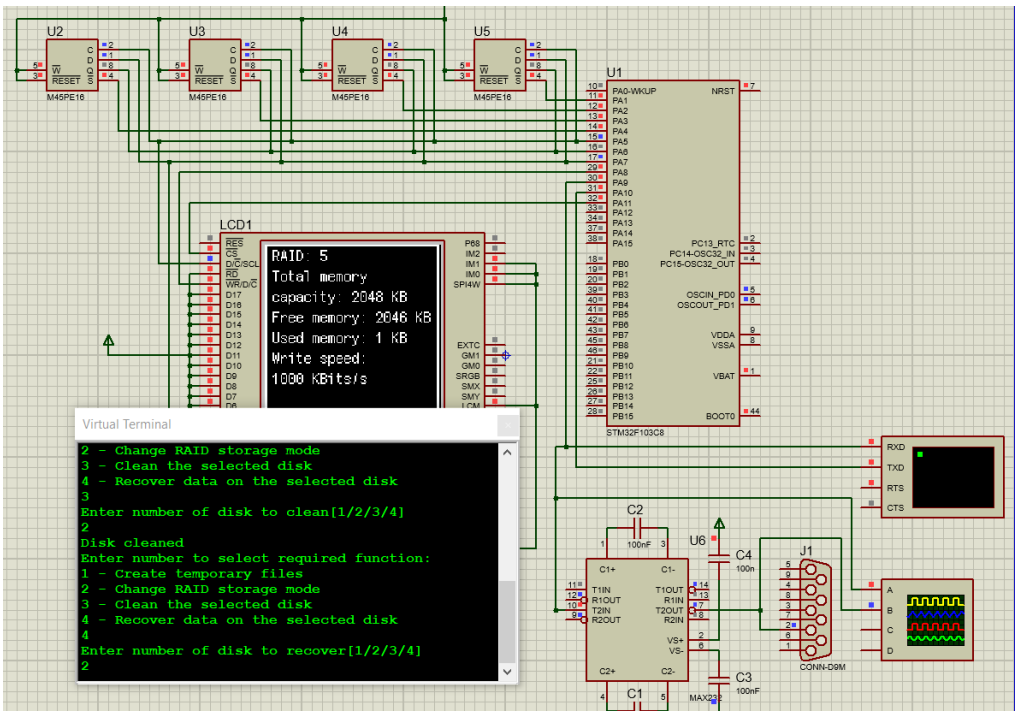


Рисунок 48 – Использование четвертой функции

Рассмотрим передачу по USART с ПЭВМ. Подключим осциллограф к входу на MAX232 и к его выходу (линии 2 и 1 соответственно) и передадим заглавную букву «С». Полученное изображение показано на рисунке 49.

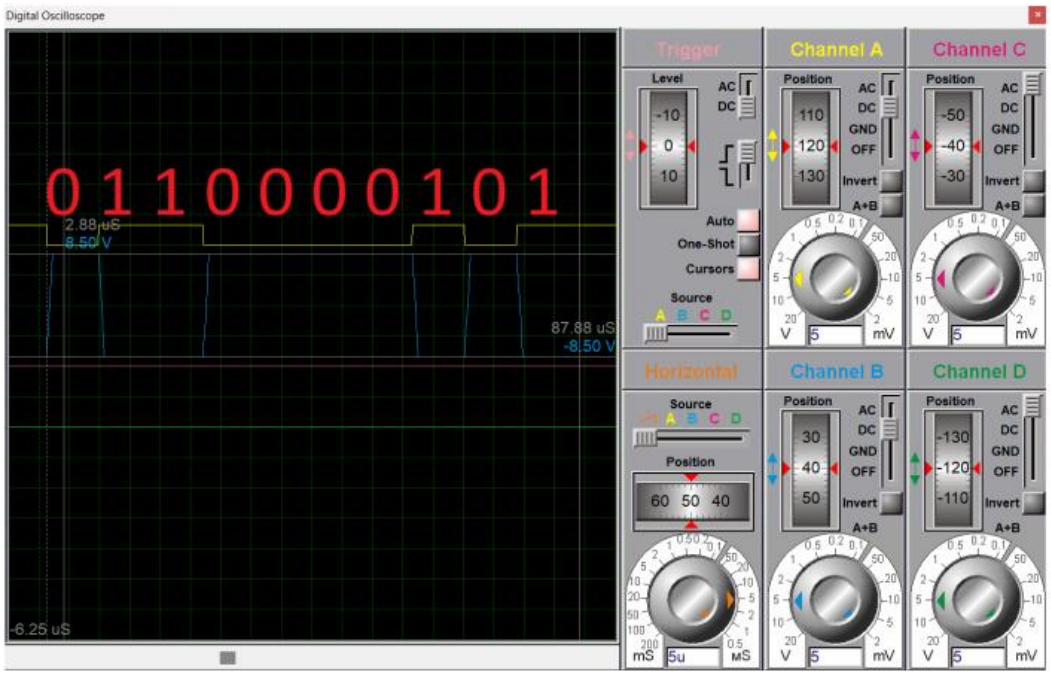


Рисунок 49 – Передача буквы «С» через с ПЭВМ

Заглавная буква «C» имеет в таблице ASCII код 01000011. Этот код передан в обратном порядке и завершен STOP-битом 0. Таким образом, USART также работает корректно.

2.3 Способы программирования МК

После написания и тестирования кода в программе идет этап загрузки файла (с расширением elf – бинарный файл) в микроконтроллер. Это может выполнятся следующими способами [11]:

– через JTAG;

– через SWD.

Выбрана прошивка через SWD так как это простой и популярный метод, с которым уже было знакомство на практике. Программирование МК происходит через программатор и ST-LINKv2, о котором было рассказано в разделе 1.3.1.

В МК передается бинарный файл с расширением “.elf” с скомпилированной программой. Происходит это следующим образом: подается команда RESET через пин NRST. Это используется для сброса микроконтроллера в состояние, готовое к прошивке. Затем через SWCLK идет тактовый сигнал, по которому идет запись программы в микроконтроллер через SWDIO. Этот процесс осуществляется с использованием специальных последовательностей битов (протокол SWD) для передачи команд и данных между ST-LINKv2 и микроконтроллером, обеспечивая правильную последовательность и синхронизацию для успешной прошивки или отладки микроконтроллера STM32.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы был создан проект – RAID-массив с возможностью выбора режима хранения. Система работает на основе МК семейства STM32 – STM32F103C8T6. Устройство разработано в соответствии с ТЗ.

В процессе работы над курсовой работой была разработана схема электрическая функциональная и принципиальная, спецификация и документация к устройству. Исходный код программы, написанный на языке С, отлажен и протестирован при помощи симулятора Proteus 8.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Хартов, В.Я. Микропроцессорные системы: учеб. пособие для студ. учреждений высш. проф. образования, Академия, М., 2014. – 368с.
2. Основные семейства микроконтроллеров [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/ Микроконтроллер#Известные_семейства (дата обращения: 13.12.2023)
3. Документация на STM32F103C8T6 [Электронный ресурс]. – URL: https://www.st.com/resource/en/reference_manual/rm0008-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf (дата обращения: 13.12.2023)
4. Документация на драйвер MAX232 [Электронный ресурс]. – URL: https://www.alldatasheet.com/datasheet-pdf/pdf/73745/MAXIM/MAX7219.html (дата обращения 27.12.2023).
5. Документация на ЖК-дисплей ST7735 [Электронный ресурс]. – URL: https://www.displayfuture.com/Display/datasheet/controller/ST7735.pdf (дата обращения 27.12.2023).
6. ГОСТ 2.710-81 Обозначения буквенно-цифровые в электрических схемах
7. ГОСТ 2.721-74 Обозначения условные графические в схемах. Обозначения общего применения
8. ГОСТ 2.102-68 ЕСКД. Виды и комплектность конструкторских документов
9. ГОСТ 2.105-95 ЕСКД. Текстовые документы
10. Программирование на Си [Электронный ресурс]. – URL: http://www.r-5.org/files/books/computers/languages/c/kr/Brian_Kernighan_Dennis_Ritchie-The_C_Programming_Language-RU.pdf (дата обращения 27.12.2023)

11. Способы программирования stm32 [Электронный ресурс]. – URL: https://portal.tpu.ru/SHARED/t/TORGAEV/academic/Tab4/Posobie3.pdf (дата обращения 27.12.2023)

12. Документация на последовательную FLASH-память M45PE16 [Электронный ресурс]. – URL: https://datasheetspdf.com/datasheet/M45PE16.html (дата обращения 27.12.2023).

Приложение А

Текст программы

Заголовочные файлы

flash.h

```
#ifndef INC_FLASH_H_
#define INC_FLASH_H_

#include "main.h"

extern const uint8_t FLASH_ADDRESS_BYTE_COUNT;
extern const uint8_t FLASH_SECTORS_COUNT;
extern const uint32_t FLASH_ALL_BYTE_COUNT;
extern const uint16_t FLASH_PAGE_SIZE_;
extern const uint8_t FLASH_DISKS_COUNT;

extern SPI_HandleTypeDef FLASH_SPI_PORT;

void FlashWriteEnable(uint16_t flash_pin);
void FlashReadStatusRegister(char* spi_buf, uint16_t flash_pin);
void FlashWrite(char* spi_buf, uint8_t* addr, uint16_t size, uint16_t flash_pin);
void FlashWaitingForEndOfWrite(uint16_t flash_pin);
void FlashRead(char* spi_buf, uint8_t* addr, uint16_t size, uint16_t flash_pin);
void FlashPageErase(uint8_t* addr, uint16_t flash_pin);
void FlashSectorErase(uint8_t* addr, uint16_t flash_pin);
void FlashFullErase(uint16_t flash_pin);

#endif /* INC_FLASH_H_ */
```

raid.h

```
#ifndef INC_RAID_H_
#define INC_RAID_H_

#include <stdint.h>

void RaidCleanDisk(uint8_t disk_number);
void RaidCleanAllDisks();
void RaidFastCleanAllDisks(uint8_t* addr);
void RaidFastCleanDisk(uint8_t* addr, uint8_t disk_number);
void RaidRecoverDisk(uint8_t* addr, uint8_t disk_number);
uint8_t RaidSetAddresses(uint8_t* addr);
void Raid0Write(uint8_t* addr, uint32_t file_size, uint8_t seed);
void Raid1Write(uint8_t* addr, uint32_t file_size, uint8_t seed, uint8_t
disk_number);
void Raid3Write(uint8_t* addr, uint32_t file_size, uint8_t seed);
void Raid4Write(uint8_t* addr, uint32_t file_size, uint8_t seed);
void Raid5Write(uint8_t* addr, uint32_t file_size, uint8_t seed, uint8_t
file_number);

#endif /* INC_RAID_H_ */
```

ST7735.h

```c
#ifndef ST7735_H_
#define ST7735_H_

#include <stdbool.h>
#include <stdlib.h>

#include "main.h"

typedef struct {
    const uint8_t width;
    uint8_t height;
    const uint16_t *data;
} FontDef;

extern FontDef Font_7x10;
extern SPI_HandleTypeDef ST7735_SPI_PORT;

void ST7735_Init();
void ST7735_DrawString(uint16_t x, uint16_t y, const char* str, FontDef font);
void ST7735_FillScreen();
void ST7735_DrawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1);

/* Color definitions */

#define ST7735_BLACK   0x0000
#define ST7735_BLUE    0x001F
#define ST7735_RED     0xF800
#define ST7735_GREEN   0x07E0
#define ST7735_CYAN    0x07FF
#define ST7735_MAGENTA 0xF81F
#define ST7735_YELLOW  0xFFE0
#define ST7735_WHITE   0xFFFF
#define ST7735_RGB(r, g, b) (((r & 0xF8) << 8) | ((g & 0xFC) << 3) | ((b & 0xF8) >>
3))



/* ONLY CONFIG BELOW */

#define ST7735_SPI_PORT hspi1 //hspi1, hspi2, hspi3...

//Port and pin connected signal 'DC' (data or command) ST7735 display
#define ST7735_DC_Pin         GPIO_PIN_8
#define ST7735_DC_GPIO_Port   GPIOA

// WaveShare ST7735S-based 1.8" display, default orientation
#define ST7735_IS_160X128     1
#define ST7735_WIDTH          128
#define ST7735_HEIGHT         160
#define ST7735_XSTART         2
#define ST7735_YSTART         1
#define ST7735_DATA_ROTATION  0

#define ST7735_NOP     0x00
#define ST7735_SWRESET 0x01
#define ST7735_RDDID   0x04
#define ST7735_RDDST   0x09

#define ST7735_SLPIN   0x10
```

```
#define ST7735_SLPOUT  0x11
#define ST7735_PTLON   0x12
#define ST7735_NORON   0x13

#define ST7735_INVOFF  0x20
#define ST7735_INVON   0x21
#define ST7735_DISPOFF 0x28
#define ST7735_DISPON  0x29
#define ST7735_CASET   0x2A
#define ST7735_RASET   0x2B
#define ST7735_RAMWR   0x2C
#define ST7735_RAMRD   0x2E

#define ST7735_PTLAR   0x30
#define ST7735_COLMOD  0x3A
#define ST7735_MADCTL  0x36

#define ST7735_FRMCTR1 0xB1
#define ST7735_FRMCTR2 0xB2
#define ST7735_FRMCTR3 0xB3
#define ST7735_INVCTR  0xB4
#define ST7735_DISSET5 0xB6

#define ST7735_PWCTR1  0xC0
#define ST7735_PWCTR2  0xC1
#define ST7735_PWCTR3  0xC2
#define ST7735_PWCTR4  0xC3
#define ST7735_PWCTR5  0xC4
#define ST7735_VMCTR1  0xC5

#define ST7735_RDID1   0xDA
#define ST7735_RDID2   0xDB
#define ST7735_RDID3   0xDC
#define ST7735_RDID4   0xDD

#define ST7735_PWCTR6  0xFC

#define ST7735_GMCTRP1 0xE0
#define ST7735_GMCTRN1 0xE1

#define DELAY 0x80

/* Ports config */

#define TFT_DC_D()  HAL_GPIO_WritePin(ST7735_DC_GPIO_Port, ST7735_DC_Pin,
GPIO_PIN_SET)
#define TFT_DC_C()  HAL_GPIO_WritePin(ST7735_DC_GPIO_Port, ST7735_DC_Pin,
GPIO_PIN_RESET)

/* Init comands */

static const uint8_t
init_cmds1[] = {                    // Init for 7735R, part 1 (red or green tab)
      15,                           // 15 commands in list:
      ST7735_SWRESET, DELAY,        //  1: Software reset, 0 args, w/delay
      150,                          //     150 ms delay
      ST7735_SLPOUT, DELAY,         //  2: Out of sleep mode, 0 args, w/delay
      255,                          //     500 ms delay
      ST7735_FRMCTR1, 3,            //  3: Frame rate ctrl - normal mode, 3 args:
```

```
      0x01, 0x2C, 0x2D,              //     Rate = fosc/(1x2+40) * (LINE+2C+2D)
      ST7735_FRMCTR2, 3,             //  4: Frame rate control - idle mode, 3 args:
      0x01, 0x2C, 0x2D,              //     Rate = fosc/(1x2+40) * (LINE+2C+2D)
      ST7735_FRMCTR3, 6,             //  5: Frame rate ctrl - partial mode, 6 args:
      0x01, 0x2C, 0x2D,              //     Dot inversion mode
      0x01, 0x2C, 0x2D,              //     Line inversion mode
      ST7735_INVCTR, 1,              //  6: Display inversion ctrl, 1 arg, no delay:
      0x07,                          //     No inversion
      ST7735_PWCTR1, 3, 0xA2,        //  7: Power control, 3 args, no delay:
      0x02,                          //     -4.6V
      0x84,                          //     AUTO mode
      ST7735_PWCTR2, 1,              //  8: Power control, 1 arg, no delay:
      0xC5,                          //     VGH25 = 2.4C VGSEL = -10 VGH = 3 * AVDD
      ST7735_PWCTR3, 2,              //  9: Power control, 2 args, no delay:
      0x0A,                          //     Opamp current small
      0x00,                          //     Boost frequency
      ST7735_PWCTR4, 2,              // 10: Power control, 2 args, no delay:
      0x8A, 0x2A,                    //     BCLK/2, Opamp current small & Medium low
      ST7735_PWCTR5, 2, 0x8A, 0xEE,  // 11: Power control, 2 args, no delay:
      ST7735_VMCTR1, 1, 0x0E,        // 12: Power control, 1 arg, no delay:
      ST7735_INVOFF, 0,              // 13: Don't invert display, no args, no delay
      ST7735_MADCTL, 1,              // 14: Memory access control (directions), 1
arg:
      ST7735_DATA_ROTATION,          //     row addr/col addr, bottom to top refresh
      ST7735_COLMOD, 1,              // 15: set color mode, 1 arg, no delay:
      0x05                                  //     16-bit color
};

#endif /* ST7735_H_ */
```

Исходные файлы

flash.c

```
#include "flash.h"

const uint8_t FLASH_READ = 0b00000011;
const uint8_t FLASH_WRITE = 0b00001010;
const uint8_t FLASH_WRDI = 0b00000100;
const uint8_t FLASH_WREN = 0b00000110;
const uint8_t FLASH_RDSR = 0b00000101;
const uint8_t FLASH_PE = 0b11011011;
const uint8_t FLASH_SE = 0b11011000;
const uint8_t FLASH_WRSR = 0b00000001;

const uint8_t FLASH_ADDRESS_BYTE_COUNT = 3;
const uint8_t FLASH_SECTORS_COUNT = 32;
const uint32_t FLASH_ALL_BYTE_COUNT = 2097152;
const uint16_t FLASH_PAGE_SIZE_ = 256;
const uint8_t FLASH_DISKS_COUNT = 4;

#define FLASH_SPI_PORT    hspi1
#define FLASH_GPIO_PORT   GPIOA

void FlashWriteEnable(uint16_t flash_pin)
{
      HAL_GPIO_WritePin(FLASH_GPIO_PORT, flash_pin, GPIO_PIN_RESET);
      HAL_SPI_Transmit(&FLASH_SPI_PORT, (uint8_t *)&FLASH_WREN, 1, 100);
      HAL_GPIO_WritePin(FLASH_GPIO_PORT, flash_pin, GPIO_PIN_SET);
```

```
}

void FlashReadStatusRegister(char* spi_buf, uint16_t flash_pin)
{
	HAL_GPIO_WritePin(FLASH_GPIO_PORT, flash_pin, GPIO_PIN_RESET);
	HAL_SPI_Transmit(&FLASH_SPI_PORT, (uint8_t *)&FLASH_RDSR, 1, 100);
	HAL_SPI_Receive(&FLASH_SPI_PORT, (uint8_t *)spi_buf, 1, 100);
	HAL_GPIO_WritePin(FLASH_GPIO_PORT, flash_pin, GPIO_PIN_SET);
}

void FlashWaitingForEndOfWrite(uint16_t flash_pin)
{
	uint8_t wip = 1;
	char temp_buf[1];
	while (wip)
	{
	  FlashReadStatusRegister(temp_buf, flash_pin);
	  wip = temp_buf[0] & 0b00000001;
	}
}

void FlashWrite(char* spi_buf, uint8_t* addr, uint16_t size, uint16_t flash_pin)
{
	FlashWaitingForEndOfWrite(flash_pin);
	FlashWriteEnable(flash_pin);
	HAL_GPIO_WritePin(FLASH_GPIO_PORT, flash_pin, GPIO_PIN_RESET);
	HAL_SPI_Transmit(&FLASH_SPI_PORT, (uint8_t *)&FLASH_WRITE, 1, 100);
	HAL_SPI_Transmit(&FLASH_SPI_PORT, (uint8_t *)addr, FLASH_ADDRESS_BYTE_COUNT,
100);
	HAL_SPI_Transmit(&FLASH_SPI_PORT, (uint8_t *)spi_buf, size, 100);
	HAL_GPIO_WritePin(FLASH_GPIO_PORT, flash_pin, GPIO_PIN_SET);
}

void FlashRead(char* spi_buf, uint8_t* addr, uint16_t size, uint16_t flash_pin)
{
	FlashWaitingForEndOfWrite(flash_pin);
	HAL_GPIO_WritePin(FLASH_GPIO_PORT, flash_pin, GPIO_PIN_RESET);
	HAL_SPI_Transmit(&FLASH_SPI_PORT, (uint8_t *)&FLASH_READ, 1, 100);
	HAL_SPI_Transmit(&FLASH_SPI_PORT, (uint8_t *)addr, FLASH_ADDRESS_BYTE_COUNT,
100);
	HAL_SPI_Receive(&FLASH_SPI_PORT, (uint8_t *)spi_buf, size, 100);
	HAL_GPIO_WritePin(FLASH_GPIO_PORT, flash_pin, GPIO_PIN_SET);
}

void FlashPageErase(uint8_t* addr, uint16_t flash_pin)
{
	FlashWaitingForEndOfWrite(flash_pin);
	FlashWriteEnable(flash_pin);
	HAL_GPIO_WritePin(FLASH_GPIO_PORT, flash_pin, GPIO_PIN_RESET);
	HAL_SPI_Transmit(&FLASH_SPI_PORT, (uint8_t *)&FLASH_PE, 1, 100);
	HAL_SPI_Transmit(&FLASH_SPI_PORT, (uint8_t *)addr, FLASH_ADDRESS_BYTE_COUNT,
100);
	HAL_GPIO_WritePin(FLASH_GPIO_PORT, flash_pin, GPIO_PIN_SET);
}

void FlashSectorErase(uint8_t* addr, uint16_t flash_pin)
{
	FlashWaitingForEndOfWrite(flash_pin);
	FlashWriteEnable(flash_pin);
```

```
	HAL_GPIO_WritePin(FLASH_GPIO_PORT, flash_pin, GPIO_PIN_RESET);
	HAL_SPI_Transmit(&FLASH_SPI_PORT, (uint8_t *)&FLASH_SE, 1, 100);
	HAL_SPI_Transmit(&FLASH_SPI_PORT, (uint8_t *)addr, FLASH_ADDRESS_BYTE_COUNT,
100);
	HAL_GPIO_WritePin(FLASH_GPIO_PORT, flash_pin, GPIO_PIN_SET);
}

void FlashFullErase(uint16_t flash_pin)
{
	uint8_t addr[3] = {0x00, 0x00, 0x00};
	for (uint8_t i = 0; i < FLASH_SECTORS_COUNT; i++)
	{
		FlashSectorErase(addr, flash_pin);
		addr[0] += 0x01;
	}
}
```

raid.c

```
#include "raid.h"
#include "flash.h"
#include <stdlib.h>

#include "main.h"

char spi_buf[256];

#define FLASH_PIN_1	GPIO_PIN_1
#define FLASH_PIN_2	GPIO_PIN_2
#define FLASH_PIN_3	GPIO_PIN_3
#define FLASH_PIN_4	GPIO_PIN_4

void RaidCleanDisk(uint8_t disk_number)
{
	switch (disk_number)
	{
		case 1:
			FlashFullErase(FLASH_PIN_1);
			break;
		case 2:
			FlashFullErase(FLASH_PIN_2);
			break;
		case 3:
			FlashFullErase(FLASH_PIN_3);
			break;
		case 4:
			FlashFullErase(FLASH_PIN_4);
			break;
		default:
			break;
	}
}

void RaidCleanAllDisks()
{
	FlashFullErase(FLASH_PIN_1);
	FlashFullErase(FLASH_PIN_2);
	FlashFullErase(FLASH_PIN_3);
	FlashFullErase(FLASH_PIN_4);
```

```
}

void RaidFastCleanAllDisks(uint8_t* addr)
{
	uint8_t tmp_addr[FLASH_ADDRESS_BYTE_COUNT];
	for (uint16_t i = 0; i < FLASH_DISKS_COUNT; i++)
	{
		tmp_addr[0] = 0;
		while (tmp_addr[0] <= addr[i * 3])
		{
			switch (i)
			{
				case 0:
					FlashSectorErase(tmp_addr, FLASH_PIN_1);
					break;
				case 1:
					FlashSectorErase(tmp_addr, FLASH_PIN_2);
					break;
				case 2:
					FlashSectorErase(tmp_addr, FLASH_PIN_3);
					break;
				case 3:
					FlashSectorErase(tmp_addr, FLASH_PIN_4);
					break;
				default:
					break;
			}
			tmp_addr[0]++;
		}
	}
}

void RaidFastCleanDisk(uint8_t* addr, uint8_t disk_number)
{
	uint8_t tmp_addr[FLASH_ADDRESS_BYTE_COUNT];
	tmp_addr[0] = 0;
	tmp_addr[1] = 0;
	tmp_addr[2] = 0;
	while (tmp_addr[0] <= addr[(disk_number - 1) * 3])
	{
		switch (disk_number)
		{
			case 1:
				FlashSectorErase(tmp_addr, FLASH_PIN_1);
				break;
			case 2:
				FlashSectorErase(tmp_addr, FLASH_PIN_2);
				break;
			case 3:
				FlashSectorErase(tmp_addr, FLASH_PIN_3);
				break;
			case 4:
				FlashSectorErase(tmp_addr, FLASH_PIN_4);
				break;
			default:
				break;
		}
		tmp_addr[0]++;
	}
```

```
}

void RaidRecoverDisk(uint8_t* addr, uint8_t disk_number)
{
	uint8_t tmp_disk_number;
	for (uint8_t i = 4; i > 0; i--)
		if (i != disk_number)
		{
			tmp_disk_number = i;
			break;
		}
	uint8_t tmp_addr_read[FLASH_ADDRESS_BYTE_COUNT];
	tmp_addr_read[0] = 0x1F;
	tmp_addr_read[1] = 0xFF;
	tmp_addr_read[2] = 0xFF;
	uint8_t tmp_addr_write[FLASH_ADDRESS_BYTE_COUNT];
	switch (tmp_disk_number)
	{
		case 1:
			FlashRead(spi_buf, tmp_addr_read, FLASH_ADDRESS_BYTE_COUNT,
FLASH_PIN_1);
			break;
		case 2:
			FlashRead(spi_buf, tmp_addr_read, FLASH_ADDRESS_BYTE_COUNT,
FLASH_PIN_2);
			break;
		case 3:
			FlashRead(spi_buf, tmp_addr_read, FLASH_ADDRESS_BYTE_COUNT,
FLASH_PIN_3);
			break;
		case 4:
			FlashRead(spi_buf, tmp_addr_read, FLASH_ADDRESS_BYTE_COUNT,
FLASH_PIN_4);
			break;
		default:
			break;
	}
	for (uint8_t i = 0; i < 2; i++)
	{
		tmp_addr_read[i] = 0;
		tmp_addr_write[i] = 0;
	}
	tmp_addr_read[2] = 2;
	tmp_addr_write[3] = 3;
	uint16_t bytes_can_wr_on_page = FLASH_PAGE_SIZE_;
	uint32_t tmp_condition;
	uint8_t tmp_spi_buf[1];
	uint32_t tmp_number;
	uint8_t k = 3;
	while (spi_buf[0] != 0xFF || spi_buf[1] != 0xFF || spi_buf[2] != 0xFF)
	{
		tmp_condition = ((spi_buf[0] << 16) | (spi_buf[1] << 8) | spi_buf[2]) -
((tmp_addr_write[0] << 16) | (tmp_addr_write[1] << 8) | tmp_addr_write[2]);
		if (bytes_can_wr_on_page < 3)
		{
			Raid0Switch(spi_buf, tmp_addr_write, bytes_can_wr_on_page,
disk_number - 1);
			for (uint8_t l = 0; l < bytes_can_wr_on_page; l++)
			{
```

```
                        spi_buf[0] = spi_buf[1];
                        spi_buf[1] = spi_buf[2];
                    }
                    tmp_number = ((tmp_addr_write[0] << 16) | (tmp_addr_write[1] <<
8) | tmp_addr_write[2]) + bytes_can_wr_on_page;
                    tmp_addr_write[0] = (tmp_number >> 16) & 0xFF;
                    tmp_addr_write[1] = (tmp_number >> 8) & 0xFF;
                    tmp_addr_write[2] = tmp_number & 0xFF;
                    k = FLASH_ADDRESS_BYTE_COUNT - bytes_can_wr_on_page;
                    bytes_can_wr_on_page = FLASH_PAGE_SIZE_;
             }
             else if (bytes_can_wr_on_page == 3)
             {
                    Raid0Switch(spi_buf, tmp_addr_write, bytes_can_wr_on_page,
disk_number - 1);
                    tmp_number = ((tmp_addr_write[0] << 16) | (tmp_addr_write[1] <<
8) | tmp_addr_write[2]) + bytes_can_wr_on_page;
                    tmp_addr_write[0] = (tmp_number >> 16) & 0xFF;
                    tmp_addr_write[1] = (tmp_number >> 8) & 0xFF;
                    tmp_addr_write[2] = tmp_number & 0xFF;
                    bytes_can_wr_on_page = FLASH_PAGE_SIZE_;
                    k = 0;
             }
             else
                    bytes_can_wr_on_page -= 3;
             for (uint8_t i = 0; i < tmp_condition; i++)
             {
                    switch (tmp_disk_number)
                    {
                           case 1:
                                  FlashRead(tmp_spi_buf, tmp_addr_read, 1,
FLASH_PIN_1);
                                  break;
                           case 2:
                                  FlashRead(tmp_spi_buf, tmp_addr_read, 1,
FLASH_PIN_2);
                                  break;
                           case 3:
                                  FlashRead(tmp_spi_buf, tmp_addr_read, 1,
FLASH_PIN_3);
                                  break;
                           case 4:
                                  FlashRead(tmp_spi_buf, tmp_addr_read, 1,
FLASH_PIN_4);
                                  break;
                           default:
                                  break;
                    }
                    spi_buf[k] = tmp_spi_buf[0];
                    for (uint8_t j = 0; j < FLASH_DISKS_COUNT; j++)
                    {
                           if (j + 1 == disk_number || j + 1 == tmp_disk_number)
                                  continue;
                           switch (j + 1)
                           {
                                  case 1:
                                         FlashRead(tmp_spi_buf, tmp_addr_read, 1,
FLASH_PIN_1);
                                         break;
```

```
                                case 2:
                                        FlashRead(tmp_spi_buf, tmp_addr_read, 1,
FLASH_PIN_2);
                                        break;
                                case 3:
                                        FlashRead(tmp_spi_buf, tmp_addr_read, 1,
FLASH_PIN_3);
                                        break;
                                case 4:
                                        FlashRead(tmp_spi_buf, tmp_addr_read, 1,
FLASH_PIN_4);
                                        break;
                                default:
                                        break;
                        }
                        spi_buf[k] = spi_buf[k] ^ tmp_spi_buf[0];
                }
                tmp_number = ((tmp_addr_read[0] << 16) | (tmp_addr_read[1] << 8)
| tmp_addr_read[2]) + 1;
                tmp_addr_read[0] = (tmp_number >> 16) & 0xFF;
                tmp_addr_read[1] = (tmp_number >> 8) & 0xFF;
                tmp_addr_read[2] = tmp_number & 0xFF;
                if (k < FLASH_PAGE_SIZE_ - 1 && bytes_can_wr_on_page != 1)
                {
                        k++;
                        bytes_can_wr_on_page--;
                }
                else
                {
                        Raid0Switch(spi_buf, tmp_addr_write, k + 1, disk_number -
1);
                        tmp_number = ((tmp_addr_write[0] << 16) |
(tmp_addr_write[1] << 8) | tmp_addr_write[2]) + k + 1;
                        tmp_addr_write[0] = (tmp_number >> 16) & 0xFF;
                        tmp_addr_write[1] = (tmp_number >> 8) & 0xFF;
                        tmp_addr_write[2] = tmp_number & 0xFF;
                        k = 0;
                        bytes_can_wr_on_page = FLASH_PAGE_SIZE_ -
tmp_addr_write[2];
                }
        }
        if (k != 0)
        {
                Raid0Switch(spi_buf, tmp_addr_write, k, disk_number - 1);
                tmp_number = ((tmp_addr_write[0] << 16) | (tmp_addr_write[1] <<
8) | tmp_addr_write[2]) + k + 1;
                tmp_addr_write[0] = (tmp_number >> 16) & 0xFF;
                tmp_addr_write[1] = (tmp_number >> 8) & 0xFF;
                tmp_addr_write[2] = tmp_number & 0xFF;
                k = 0;
                bytes_can_wr_on_page = FLASH_PAGE_SIZE_ - tmp_addr_write[2];
        }
        switch (tmp_disk_number)
        {
                case 1:
                        FlashRead(spi_buf, tmp_addr_read,
FLASH_ADDRESS_BYTE_COUNT, FLASH_PIN_1);
                        break;
                case 2:
```

```
                        FlashRead(spi_buf, tmp_addr_read,
FLASH_ADDRESS_BYTE_COUNT, FLASH_PIN_2);
                        break;
                case 3:
                        FlashRead(spi_buf, tmp_addr_read,
FLASH_ADDRESS_BYTE_COUNT, FLASH_PIN_3);
                        break;
                case 4:
                        FlashRead(spi_buf, tmp_addr_read,
FLASH_ADDRESS_BYTE_COUNT, FLASH_PIN_4);
                        break;
                default:
                        break;
            }
            tmp_number = ((tmp_addr_read[0] << 16) | (tmp_addr_read[1] << 8) |
tmp_addr_read[2]) + 1;
            tmp_addr_read[0] = (tmp_number >> 16) & 0xFF;
            tmp_addr_read[1] = (tmp_number >> 8) & 0xFF;
            tmp_addr_read[2] = tmp_number & 0xFF;
      }
      for (uint8_t i = 0; i < 3; i++)
            addr[i] = tmp_addr_write[i];
}

uint8_t RaidSetAddresses(uint8_t* addr)
{
      uint8_t tmp_addr[FLASH_ADDRESS_BYTE_COUNT];
      uint32_t tmp_number;
      uint8_t number_of_files = 0;
      for (uint16_t i = 0; i < FLASH_DISKS_COUNT; i++)
      {
            tmp_addr[0] = 0x1F;
            tmp_addr[1] = 0xFF;
            tmp_addr[2] = 0xFF;
            switch (i)
            {
                case 0:
                        FlashRead(spi_buf, tmp_addr, FLASH_ADDRESS_BYTE_COUNT,
FLASH_PIN_1);
                        break;
                case 1:
                        FlashRead(spi_buf, tmp_addr, FLASH_ADDRESS_BYTE_COUNT,
FLASH_PIN_2);
                        break;
                case 2:
                        FlashRead(spi_buf, tmp_addr, FLASH_ADDRESS_BYTE_COUNT,
FLASH_PIN_3);
                        break;
                case 3:
                        FlashRead(spi_buf, tmp_addr, FLASH_ADDRESS_BYTE_COUNT,
FLASH_PIN_4);
                        break;
                default:
                        break;
            }
            while (spi_buf[0] != 0xFF || spi_buf[1] != 0xFF || spi_buf[2] != 0xFF)
            {
                  number_of_files++;
                  tmp_number = ((spi_buf[0] << 16) | (spi_buf[1] << 8) |
```

```
spi_buf[2]) - 1;
                    tmp_addr[0] = (tmp_number >> 16) & 0xFF;
                    tmp_addr[1] = (tmp_number >> 8) & 0xFF;
                    tmp_addr[2] = tmp_number & 0xFF;
                    switch (i)
                    {
                           case 0:
                                  FlashRead(spi_buf, tmp_addr,
FLASH_ADDRESS_BYTE_COUNT, FLASH_PIN_1);
                                  break;
                           case 1:
                                  FlashRead(spi_buf, tmp_addr,
FLASH_ADDRESS_BYTE_COUNT, FLASH_PIN_2);
                                  break;
                           case 2:
                                  FlashRead(spi_buf, tmp_addr,
FLASH_ADDRESS_BYTE_COUNT, FLASH_PIN_3);
                                  break;
                           case 3:
                                  FlashRead(spi_buf, tmp_addr,
FLASH_ADDRESS_BYTE_COUNT, FLASH_PIN_4);
                                  break;
                           default:
                                  break;
                    }
             }
             if (tmp_addr[0] != 0x1F || tmp_addr[1] != 0xFF || tmp_addr[2] != 0xFF)
                    tmp_number = ((tmp_addr[0] << 16) | (tmp_addr[1] << 8) |
tmp_addr[2]) + 1;
             else
                    tmp_number = 0;
             addr[i * 3] = (tmp_number >> 16) & 0xFF;
             addr[i * 3 + 1] = (tmp_number >> 8) & 0xFF;
             addr[i * 3 + 2] = tmp_number & 0xFF;
      }
      return number_of_files / 4;
}

void Raid0Switch(char* spi_buf, uint8_t* addr, uint16_t size, uint8_t
switch_condition)
{
      switch (switch_condition)
      {
             case 0:
                    FlashWrite(spi_buf, addr, size, FLASH_PIN_1);
                    break;
             case 1:
                    FlashWrite(spi_buf, addr, size, FLASH_PIN_2);
                    break;
             case 2:
                    FlashWrite(spi_buf, addr, size, FLASH_PIN_3);
                    break;
             case 3:
                    FlashWrite(spi_buf, addr, size, FLASH_PIN_4);
                    break;
             default:
                    break;
      }
}
```

```
void Raid0Write(uint8_t* addr, uint32_t file_size, uint8_t seed)
{
	srand(seed);
	uint8_t max = 0xFF;
	uint8_t tmp_addr[FLASH_ADDRESS_BYTE_COUNT];
	uint32_t tmp_number;
	uint32_t tmp_block_size;
	uint8_t j;
	uint16_t bytes_can_wr_on_page;
	for (uint8_t k = 0; k < FLASH_DISKS_COUNT; k++)
	{
		tmp_number = (addr[k * 3] << 16) | (addr[k * 3 + 1] << 8) | addr[k * 3
+ 2];
		tmp_block_size = file_size / FLASH_DISKS_COUNT;
		if (file_size % FLASH_DISKS_COUNT != 0 && k < file_size %
FLASH_DISKS_COUNT)
			tmp_block_size++;
		tmp_number += tmp_block_size + FLASH_ADDRESS_BYTE_COUNT;
		spi_buf[0] = (tmp_number >> 16) & 0xFF;
		spi_buf[1] = (tmp_number >> 8) & 0xFF;
		spi_buf[2] = tmp_number & 0xFF;
		for (uint8_t i = 0; i < FLASH_ADDRESS_BYTE_COUNT; i++)
		{
			tmp_addr[i] = addr[k * 3 + i];
			addr[k * 3 + i] = spi_buf[i];   /
		}
		bytes_can_wr_on_page = FLASH_PAGE_SIZE_ - tmp_addr[2];
		j = 3;
		if (bytes_can_wr_on_page < 3)
		{
			Raid0Switch(spi_buf, tmp_addr, bytes_can_wr_on_page, k);
			for (uint8_t i = 0; i < bytes_can_wr_on_page; i++)
			{
				spi_buf[0] = spi_buf[1];
				spi_buf[1] = spi_buf[2];
			}
			tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) |
tmp_addr[2];
			tmp_number += bytes_can_wr_on_page;
			tmp_addr[0] = (tmp_number >> 16) & 0xFF;
			tmp_addr[1] = (tmp_number >> 8) & 0xFF;
			tmp_addr[2] = tmp_number & 0xFF;
			j = FLASH_ADDRESS_BYTE_COUNT - bytes_can_wr_on_page;
			bytes_can_wr_on_page = FLASH_PAGE_SIZE_;
		}
		else if (bytes_can_wr_on_page == 3)
		{
			Raid0Switch(spi_buf, tmp_addr, bytes_can_wr_on_page, k);
			tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) |
tmp_addr[2];
			tmp_number += bytes_can_wr_on_page;
			tmp_addr[0] = (tmp_number >> 16) & 0xFF;
			tmp_addr[1] = (tmp_number >> 8) & 0xFF;
			tmp_addr[2] = tmp_number & 0xFF;
			bytes_can_wr_on_page = FLASH_PAGE_SIZE_;
			j = 0;
		}
		else
```

```
                bytes_can_wr_on_page -= 3;
            for (uint32_t i = 0; i < tmp_block_size; i++)
            {
                spi_buf[j] = rand() % (max + 1);
                if (j < FLASH_PAGE_SIZE_ - 1 && bytes_can_wr_on_page != 1)
                {
                    j++;
                    bytes_can_wr_on_page--;
                }
                else
                {
                    Raid0Switch(spi_buf, tmp_addr, j + 1, k);
                    tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) |
tmp_addr[2];
                    tmp_number += j + 1;
                    tmp_addr[0] = (tmp_number >> 16) & 0xFF;
                    tmp_addr[1] = (tmp_number >> 8) & 0xFF;
                    tmp_addr[2] = tmp_number & 0xFF;
                    j = 0;
                    bytes_can_wr_on_page = FLASH_PAGE_SIZE_ - tmp_addr[2];
                }
            }
            if (j != 0)
                Raid0Switch(spi_buf, tmp_addr, j, k);
        }
}

void Raid1Switch(char* spi_buf, uint8_t* addr, uint16_t size, uint8_t
switch_condition)
{
        if (switch_condition == 1)
        {
            FlashWrite(spi_buf, addr, size, FLASH_PIN_1);
            FlashWrite(spi_buf, addr, size, FLASH_PIN_3);
        }
        else
        {
            FlashWrite(spi_buf, addr, size, FLASH_PIN_2);
            FlashWrite(spi_buf, addr, size, FLASH_PIN_4);
        }
}

void Raid1Write(uint8_t* addr, uint32_t file_size, uint8_t seed, uint8_t disk_number)
{
        srand(seed);
        uint8_t max = 0xFF;
        uint8_t tmp_addr[FLASH_ADDRESS_BYTE_COUNT];
        uint32_t tmp_number;
        if (disk_number == 1)
            tmp_number = (addr[0] << 16) | (addr[1] << 8) | addr[2];
        else
            tmp_number = (addr[6] << 16) | (addr[7] << 8) | addr[8];
        tmp_number += file_size + FLASH_ADDRESS_BYTE_COUNT;
        spi_buf[0] = (tmp_number >> 16) & 0xFF;
        spi_buf[1] = (tmp_number >> 8) & 0xFF;
        spi_buf[2] = tmp_number & 0xFF;
        if (disk_number == 1)
            for (uint8_t i = 0; i < FLASH_ADDRESS_BYTE_COUNT; i++)
            {
```

```
                        tmp_addr[i] = addr[i];
                        addr[i] = spi_buf[i];
                }
        else
                for (uint8_t i = 0; i < FLASH_ADDRESS_BYTE_COUNT; i++)
                {
                        tmp_addr[i] = addr[6 + i];
                        addr[6 + i] = spi_buf[i];
                }
        uint16_t bytes_can_wr_on_page = FLASH_PAGE_SIZE_ - tmp_addr[2];
        uint8_t j = 3;
        if (bytes_can_wr_on_page < 3)
        {
                Raid1Switch(spi_buf, tmp_addr, bytes_can_wr_on_page, disk_number);
                for (uint8_t i = 0; i < bytes_can_wr_on_page; i++)
                {
                        spi_buf[0] = spi_buf[1];
                        spi_buf[1] = spi_buf[2];
                }
                tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) | tmp_addr[2];
                tmp_number += bytes_can_wr_on_page;
                tmp_addr[0] = (tmp_number >> 16) & 0xFF;
                tmp_addr[1] = (tmp_number >> 8) & 0xFF;
                tmp_addr[2] = tmp_number & 0xFF;
                j = FLASH_ADDRESS_BYTE_COUNT - bytes_can_wr_on_page;
                bytes_can_wr_on_page = FLASH_PAGE_SIZE_;
        }
        else if (bytes_can_wr_on_page == 3)
        {
                Raid1Switch(spi_buf, tmp_addr, bytes_can_wr_on_page, disk_number);
                tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) | tmp_addr[2];
                tmp_number += bytes_can_wr_on_page;
                tmp_addr[0] = (tmp_number >> 16) & 0xFF;
                tmp_addr[1] = (tmp_number >> 8) & 0xFF;
                tmp_addr[2] = tmp_number & 0xFF;
                bytes_can_wr_on_page = FLASH_PAGE_SIZE_;
                j = 0;
        }
        else
                bytes_can_wr_on_page -= 3;
        for (uint32_t i = 0; i < file_size; i++)
        {
                spi_buf[j] = rand() % (max + 1);
                if (j < FLASH_PAGE_SIZE_ - 1 && bytes_can_wr_on_page != 1)
                {
                        j++;
                        bytes_can_wr_on_page--;
                }
                else
                {
                        Raid1Switch(spi_buf, tmp_addr, j + 1, disk_number);
                        tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) |
tmp_addr[2];
                        tmp_number += j + 1;
                        tmp_addr[0] = (tmp_number >> 16) & 0xFF;
                        tmp_addr[1] = (tmp_number >> 8) & 0xFF;
                        tmp_addr[2] = tmp_number & 0xFF;
                        j = 0;
                        bytes_can_wr_on_page = FLASH_PAGE_SIZE_ - tmp_addr[2];
```

```
			}
		}
		if (j != 0)
			Raid1Switch(spi_buf, tmp_addr, j, disk_number);
}

void Raid3Write(uint8_t* addr, uint32_t file_size, uint8_t seed)
{
		srand(seed);
		uint8_t max = 0xFF;
		uint8_t tmp_addr[FLASH_ADDRESS_BYTE_COUNT];
		uint32_t tmp_number[4];
		uint32_t tmp_block_size;
		uint8_t j;
		uint16_t bytes_can_wr_on_page;
		for (uint8_t k = 0; k < FLASH_DISKS_COUNT; k++)
		{
			tmp_number[3] = (addr[k * 3] << 16) | (addr[k * 3 + 1] << 8) | addr[k *
3 + 2];
			tmp_block_size = file_size / (FLASH_DISKS_COUNT - 1);
			if (file_size % (FLASH_DISKS_COUNT - 1) != 0
				tmp_block_size++;
			tmp_number[3] += tmp_block_size + FLASH_ADDRESS_BYTE_COUNT;
			spi_buf[0] = (tmp_number[3] >> 16) & 0xFF;
			spi_buf[1] = (tmp_number[3] >> 8) & 0xFF;
			spi_buf[2] = tmp_number[3] & 0xFF;
			for (uint8_t i = 0; i < FLASH_ADDRESS_BYTE_COUNT; i++)
			{
				tmp_addr[i] = addr[k * 3 + i];
				addr[k * 3 + i] = spi_buf[i];
			}
			bytes_can_wr_on_page = FLASH_PAGE_SIZE_ - tmp_addr[2];
			j = 3;
			if (bytes_can_wr_on_page < 3)
			{
				Raid0Switch(spi_buf, tmp_addr, bytes_can_wr_on_page, k);
				for (uint8_t i = 0; i < bytes_can_wr_on_page; i++)
				{
					spi_buf[0] = spi_buf[1];
					spi_buf[1] = spi_buf[2];
				}
				tmp_number[3] = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) |
tmp_addr[2];
				tmp_number[3] += bytes_can_wr_on_page;
				tmp_addr[0] = (tmp_number[3] >> 16) & 0xFF;
				tmp_addr[1] = (tmp_number[3] >> 8) & 0xFF;
				tmp_addr[2] = tmp_number[3] & 0xFF;
				j = FLASH_ADDRESS_BYTE_COUNT - bytes_can_wr_on_page;
				Raid0Switch(spi_buf, tmp_addr, j, k);
				tmp_number[3] = ((tmp_addr[0] << 16) | (tmp_addr[1] << 8) |
tmp_addr[2]) + j;
				tmp_number[k] = tmp_number[3];
			}
			else
			{
				Raid0Switch(spi_buf, tmp_addr, FLASH_ADDRESS_BYTE_COUNT, k);
				tmp_number[k] = ((tmp_addr[0] << 16) | (tmp_addr[1] << 8) |
tmp_addr[2]) + FLASH_ADDRESS_BYTE_COUNT;
			}
```

```
	}
	char tmp_spi_buf[1];
	uint8_t file_parity = 0;
	if (file_size % (FLASH_DISKS_COUNT - 1) > 0)
		file_parity = 1;
	for (uint32_t i = 0; i < (file_size / (FLASH_DISKS_COUNT - 1)) + file_parity;
i++)
	{
		spi_buf[0] = rand() % (max + 1);
		tmp_spi_buf[0] = spi_buf[0];
		tmp_addr[0] = (tmp_number[0] >> 16) & 0xFF;
		tmp_addr[1] = (tmp_number[0] >> 8) & 0xFF;
		tmp_addr[2] = tmp_number[0] & 0xFF;
		FlashWrite(spi_buf, tmp_addr, 1, FLASH_PIN_1);
		spi_buf[0] = rand() % (max + 1);
		tmp_spi_buf[0] = tmp_spi_buf[0] ^ spi_buf[0];
		tmp_addr[0] = (tmp_number[1] >> 16) & 0xFF;
		tmp_addr[1] = (tmp_number[1] >> 8) & 0xFF;
		tmp_addr[2] = tmp_number[1] & 0xFF;
		FlashWrite(spi_buf, tmp_addr, 1, FLASH_PIN_2);
		spi_buf[0] = rand() % (max + 1);
		tmp_spi_buf[0] = tmp_spi_buf[0] ^ spi_buf[0];
		tmp_addr[0] = (tmp_number[2] >> 16) & 0xFF;
		tmp_addr[1] = (tmp_number[2] >> 8) & 0xFF;
		tmp_addr[2] = tmp_number[2] & 0xFF;
		FlashWrite(spi_buf, tmp_addr, 1, FLASH_PIN_3);
		spi_buf[0] = tmp_spi_buf[0];
		tmp_addr[0] = (tmp_number[3] >> 16) & 0xFF;
		tmp_addr[1] = (tmp_number[3] >> 8) & 0xFF;
		tmp_addr[2] = tmp_number[3] & 0xFF;
		FlashWrite(spi_buf, tmp_addr, 1, FLASH_PIN_4);
		for (j = 0; j < FLASH_DISKS_COUNT; j++)
			tmp_number[j] += 1;
	}
}

void Raid4Write(uint8_t* addr, uint32_t file_size, uint8_t seed)
{
	srand(seed);
	uint8_t max = 0xFF;
	uint8_t tmp_addr[FLASH_ADDRESS_BYTE_COUNT];
	uint32_t tmp_number;
	uint32_t tmp_block_size;
	uint8_t j;
	uint16_t bytes_can_wr_on_page;
	uint32_t tmp_addresses[FLASH_DISKS_COUNT - 1];
	uint8_t tmp_spi_buf[1];
	for (uint8_t k = 0; k < FLASH_DISKS_COUNT - 1; k++)
	{
		tmp_number = (addr[k * 3] << 16) | (addr[k * 3 + 1] << 8) | addr[k * 3
+ 2];
		tmp_number += 2;
		tmp_addresses[k] = tmp_number;
		tmp_number -= 2;
		tmp_block_size = file_size / (FLASH_DISKS_COUNT - 1);
		if (file_size % (FLASH_DISKS_COUNT - 1) != 0)
			tmp_block_size++;
		tmp_number += tmp_block_size + FLASH_ADDRESS_BYTE_COUNT;
		spi_buf[0] = (tmp_number >> 16) & 0xFF;
```

```
            spi_buf[1] = (tmp_number >> 8) & 0xFF;
            spi_buf[2] = tmp_number & 0xFF;
            for (uint8_t i = 0; i < FLASH_ADDRESS_BYTE_COUNT; i++)
            {
                  tmp_addr[i] = addr[k * 3 + i];
                  addr[k * 3 + i] = spi_buf[i];
            }
            bytes_can_wr_on_page = FLASH_PAGE_SIZE_ - tmp_addr[2];
            j = 3;
            if (bytes_can_wr_on_page < 3)
            {
                  Raid0Switch(spi_buf, tmp_addr, bytes_can_wr_on_page, k);
                  for (uint8_t i = 0; i < bytes_can_wr_on_page; i++)
                  {
                        spi_buf[0] = spi_buf[1];
                        spi_buf[1] = spi_buf[2];
                  }
                  tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) |
tmp_addr[2];
                  tmp_number += bytes_can_wr_on_page;
                  tmp_addr[0] = (tmp_number >> 16) & 0xFF;
                  tmp_addr[1] = (tmp_number >> 8) & 0xFF;
                  tmp_addr[2] = tmp_number & 0xFF;
                  j = FLASH_ADDRESS_BYTE_COUNT - bytes_can_wr_on_page;
                  bytes_can_wr_on_page = FLASH_PAGE_SIZE_;
            }
            else if (bytes_can_wr_on_page == 3)
            {
                  Raid0Switch(spi_buf, tmp_addr, bytes_can_wr_on_page, k);
                  tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) |
tmp_addr[2];
                  tmp_number += bytes_can_wr_on_page;
                  tmp_addr[0] = (tmp_number >> 16) & 0xFF;
                  tmp_addr[1] = (tmp_number >> 8) & 0xFF;
                  tmp_addr[2] = tmp_number & 0xFF;
                  bytes_can_wr_on_page = FLASH_PAGE_SIZE_;
                  j = 0;
            }
            else
                  bytes_can_wr_on_page -= 3;
            for (uint32_t i = 0; i < tmp_block_size; i++)
            {
                  spi_buf[j] = rand() % (max + 1);
                  if (j < FLASH_PAGE_SIZE_ - 1 && bytes_can_wr_on_page != 1)
                  {
                        j++;
                        bytes_can_wr_on_page--;
                  }
                  else
                  {
                        Raid0Switch(spi_buf, tmp_addr, j + 1, k);
                        tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) |
tmp_addr[2];
                        tmp_number += j + 1;
                        tmp_addr[0] = (tmp_number >> 16) & 0xFF;
                        tmp_addr[1] = (tmp_number >> 8) & 0xFF;
                        tmp_addr[2] = tmp_number & 0xFF;
                        j = 0;
                        bytes_can_wr_on_page = FLASH_PAGE_SIZE_ - tmp_addr[2];
```

```
                }
        }
        if (j != 0)
                Raid0Switch(spi_buf, tmp_addr, j, k);
}

tmp_number = (addr[9] << 16) | (addr[10] << 8) | addr[11];
tmp_block_size = file_size / (FLASH_DISKS_COUNT - 1);
if (file_size % (FLASH_DISKS_COUNT - 1) != 0)
        tmp_block_size++;
tmp_number += tmp_block_size + FLASH_ADDRESS_BYTE_COUNT;
spi_buf[0] = (tmp_number >> 16) & 0xFF;
spi_buf[1] = (tmp_number >> 8) & 0xFF;
spi_buf[2] = tmp_number & 0xFF;
for (uint8_t i = 0; i < FLASH_ADDRESS_BYTE_COUNT; i++)
{
        tmp_addr[i] = addr[9 + i];
        addr[9 + i] = spi_buf[i];
}
bytes_can_wr_on_page = FLASH_PAGE_SIZE_ - tmp_addr[2];
j = 3;
if (bytes_can_wr_on_page < 3)
{
        Raid0Switch(spi_buf, tmp_addr, bytes_can_wr_on_page, 3);
        for (uint8_t i = 0; i < bytes_can_wr_on_page; i++)
        {
                spi_buf[0] = spi_buf[1];
                spi_buf[1] = spi_buf[2];
        }
        tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) | tmp_addr[2];
        tmp_number += bytes_can_wr_on_page;
        tmp_addr[0] = (tmp_number >> 16) & 0xFF;
        tmp_addr[1] = (tmp_number >> 8) & 0xFF;
        tmp_addr[2] = tmp_number & 0xFF;
        j = FLASH_ADDRESS_BYTE_COUNT - bytes_can_wr_on_page;
        bytes_can_wr_on_page = FLASH_PAGE_SIZE_;
}
else if (bytes_can_wr_on_page == 3)
{
        Raid0Switch(spi_buf, tmp_addr, bytes_can_wr_on_page, 3);
        tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) | tmp_addr[2];
        tmp_number += bytes_can_wr_on_page;
        tmp_addr[0] = (tmp_number >> 16) & 0xFF;
        tmp_addr[1] = (tmp_number >> 8) & 0xFF;
        tmp_addr[2] = tmp_number & 0xFF;
        bytes_can_wr_on_page = FLASH_PAGE_SIZE_;
        j = 0;
}
else
        bytes_can_wr_on_page -= 3;
uint8_t addr1_3[FLASH_ADDRESS_BYTE_COUNT];
for (uint32_t i = 0; i < tmp_block_size; i++)
{
        addr1_3[0] = (tmp_addresses[0] >> 16) & 0xFF;
        addr1_3[1] = (tmp_addresses[0] >> 8) & 0xFF;
        addr1_3[2] = tmp_addresses[0] & 0xFF;
        FlashRead(tmp_spi_buf, addr1_3, 1, FLASH_PIN_1);
        spi_buf[j] = tmp_spi_buf[0];
        addr1_3[0] = (tmp_addresses[1] >> 16) & 0xFF;
```

```
			addr1_3[1] = (tmp_addresses[1] >> 8) & 0xFF;
			addr1_3[2] = tmp_addresses[1] & 0xFF;
			FlashRead(tmp_spi_buf, addr1_3, 1, FLASH_PIN_2);
			spi_buf[j] = spi_buf[j] ^ tmp_spi_buf[0];
			addr1_3[0] = (tmp_addresses[2] >> 16) & 0xFF;
			addr1_3[1] = (tmp_addresses[2] >> 8) & 0xFF;
			addr1_3[2] = tmp_addresses[2] & 0xFF;
			FlashRead(tmp_spi_buf, addr1_3, 1, FLASH_PIN_3);
			spi_buf[j] = spi_buf[j] ^ tmp_spi_buf[0];
			if (j < FLASH_PAGE_SIZE_ - 1 && bytes_can_wr_on_page != 1)
			{
				j++;
				bytes_can_wr_on_page--;
			}
			else
			{
				Raid0Switch(spi_buf, tmp_addr, j + 1, 3);
				tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) |
tmp_addr[2];
				tmp_number += j + 1;
				tmp_addr[0] = (tmp_number >> 16) & 0xFF;
				tmp_addr[1] = (tmp_number >> 8) & 0xFF;
				tmp_addr[2] = tmp_number & 0xFF;
				j = 0;
				bytes_can_wr_on_page = FLASH_PAGE_SIZE_ - tmp_addr[2];
			}
			for (uint8_t k = 0; k < FLASH_DISKS_COUNT - 1; k++)
				tmp_addresses[k]++;
		}
		if (j != 0)
			Raid0Switch(spi_buf, tmp_addr, j, 3);
}

void Raid5Write(uint8_t* addr, uint32_t file_size, uint8_t seed, uint8_t file_number)
{
		srand(seed);
		uint8_t max = 0xFF;
		uint8_t tmp_addr[FLASH_ADDRESS_BYTE_COUNT];
		uint32_t tmp_number;
		uint32_t tmp_block_size;
		uint8_t j;
		uint16_t bytes_can_wr_on_page;
		uint32_t tmp_addresses[FLASH_DISKS_COUNT];
		uint8_t tmp_spi_buf[1];
		for (uint8_t k = 0; k < FLASH_DISKS_COUNT; k++)
		{
			if (k + 1 == 4 - (file_number - 1) % 4)
				continue;
			tmp_number = (addr[k * 3] << 16) | (addr[k * 3 + 1] << 8) | addr[k * 3
+ 2];
			tmp_number += 2;
			tmp_addresses[k] = tmp_number;
			tmp_number -= 2;
			tmp_block_size = file_size / (FLASH_DISKS_COUNT - 1);
			if (file_size % (FLASH_DISKS_COUNT - 1) != 0)
				tmp_block_size++;
			tmp_number += tmp_block_size + FLASH_ADDRESS_BYTE_COUNT;
			spi_buf[0] = (tmp_number >> 16) & 0xFF;
			spi_buf[1] = (tmp_number >> 8) & 0xFF;
```

```
            spi_buf[2] = tmp_number & 0xFF;
            for (uint8_t i = 0; i < FLASH_ADDRESS_BYTE_COUNT; i++)
            {
                  tmp_addr[i] = addr[k * 3 + i];
                  addr[k * 3 + i] = spi_buf[i];
            }
            bytes_can_wr_on_page = FLASH_PAGE_SIZE_ - tmp_addr[2];
            j = 3;
            if (bytes_can_wr_on_page < 3)
            {
                  Raid0Switch(spi_buf, tmp_addr, bytes_can_wr_on_page, k);
                  for (uint8_t i = 0; i < bytes_can_wr_on_page; i++)
                  {
                        spi_buf[0] = spi_buf[1];
                        spi_buf[1] = spi_buf[2];
                  }
                  tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) |
tmp_addr[2];
                  tmp_number += bytes_can_wr_on_page;
                  tmp_addr[0] = (tmp_number >> 16) & 0xFF;
                  tmp_addr[1] = (tmp_number >> 8) & 0xFF;
                  tmp_addr[2] = tmp_number & 0xFF;
                  j = FLASH_ADDRESS_BYTE_COUNT - bytes_can_wr_on_page;
                  bytes_can_wr_on_page = FLASH_PAGE_SIZE_;
            }
            else if (bytes_can_wr_on_page == 3)
            {
                  Raid0Switch(spi_buf, tmp_addr, bytes_can_wr_on_page, k);
                  tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) |
tmp_addr[2];
                  tmp_number += bytes_can_wr_on_page;
                  tmp_addr[0] = (tmp_number >> 16) & 0xFF;
                  tmp_addr[1] = (tmp_number >> 8) & 0xFF;
                  tmp_addr[2] = tmp_number & 0xFF;
                  bytes_can_wr_on_page = FLASH_PAGE_SIZE_;
                  j = 0;
            }
            else
                  bytes_can_wr_on_page -= 3;
            for (uint32_t i = 0; i < tmp_block_size; i++)
            {
                  spi_buf[j] = rand() % (max + 1);
                  if (j < FLASH_PAGE_SIZE_ - 1 && bytes_can_wr_on_page != 1)
                  {
                        j++;
                        bytes_can_wr_on_page--;
                  }
                  else
                  {
                        Raid0Switch(spi_buf, tmp_addr, j + 1, k);
                        tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) |
tmp_addr[2];
                        tmp_number += j + 1;
                        tmp_addr[0] = (tmp_number >> 16) & 0xFF;
                        tmp_addr[1] = (tmp_number >> 8) & 0xFF;
                        tmp_addr[2] = tmp_number & 0xFF;
                        j = 0;
                        bytes_can_wr_on_page = FLASH_PAGE_SIZE_ - tmp_addr[2];
                  }
```

```
            }
            if (j != 0)
                  Raid0Switch(spi_buf, tmp_addr, j, k);
      }
      tmp_block_size = file_size / (FLASH_DISKS_COUNT - 1);
      tmp_number = (addr[(3 - (file_number - 1) % 4) * 3] << 16) | (addr[(3 -
(file_number - 1) % 4) * 3 + 1] << 8) | addr[(3 - (file_number - 1) % 4) * 3 + 2];
      if (file_size % (FLASH_DISKS_COUNT - 1) != 0)
            tmp_block_size++;
      tmp_number += tmp_block_size + FLASH_ADDRESS_BYTE_COUNT;
      spi_buf[0] = (tmp_number >> 16) & 0xFF;
      spi_buf[1] = (tmp_number >> 8) & 0xFF;
      spi_buf[2] = tmp_number & 0xFF;
      for (uint8_t i = 0; i < FLASH_ADDRESS_BYTE_COUNT; i++)
      {
            tmp_addr[i] = addr[(3 - (file_number - 1) % 4) * 3 + i];
            addr[(3 - (file_number - 1) % 4) * 3 + i] = spi_buf[i];
      }
      bytes_can_wr_on_page = FLASH_PAGE_SIZE_ - tmp_addr[2];
      j = 3;
      if (bytes_can_wr_on_page < 3)
      {
            Raid0Switch(spi_buf, tmp_addr, bytes_can_wr_on_page, 3 - (file_number -
1) % 4);
            for (uint8_t i = 0; i < bytes_can_wr_on_page; i++)
            {
                  spi_buf[0] = spi_buf[1];
                  spi_buf[1] = spi_buf[2];
            }
            tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) | tmp_addr[2];
            tmp_number += bytes_can_wr_on_page;
            tmp_addr[0] = (tmp_number >> 16) & 0xFF;
            tmp_addr[1] = (tmp_number >> 8) & 0xFF;
            tmp_addr[2] = tmp_number & 0xFF;
            j = FLASH_ADDRESS_BYTE_COUNT - bytes_can_wr_on_page;
            bytes_can_wr_on_page = FLASH_PAGE_SIZE_;
      }
      else if (bytes_can_wr_on_page == 3)
      {
            Raid0Switch(spi_buf, tmp_addr, bytes_can_wr_on_page, 3 - (file_number -
1) % 4);
            tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) | tmp_addr[2];
            tmp_number += bytes_can_wr_on_page;
            tmp_addr[0] = (tmp_number >> 16) & 0xFF;
            tmp_addr[1] = (tmp_number >> 8) & 0xFF;
            tmp_addr[2] = tmp_number & 0xFF;
            bytes_can_wr_on_page = FLASH_PAGE_SIZE_;
            j = 0;
      }
      else
            bytes_can_wr_on_page -= 3;
      uint8_t addr1_3[FLASH_ADDRESS_BYTE_COUNT];
      for (uint32_t i = 0; i < tmp_block_size; i++)
      {
            Raid5Switch(spi_buf, tmp_spi_buf, addr1_3, tmp_addresses, j, 3 -
(file_number - 1) % 4);
            for (uint8_t k = 0; k < FLASH_DISKS_COUNT; k++)
                  tmp_addresses[k]++;
            if (j < FLASH_PAGE_SIZE_ - 1 && bytes_can_wr_on_page != 1)
```

```
		{
			j++;
			bytes_can_wr_on_page--;
		}
		else
		{
			Raid0Switch(spi_buf, tmp_addr, j + 1, 3 - (file_number - 1) % 4);
			tmp_number = (tmp_addr[0] << 16) | (tmp_addr[1] << 8) |
tmp_addr[2];
			tmp_number += j + 1;
			tmp_addr[0] = (tmp_number >> 16) & 0xFF;
			tmp_addr[1] = (tmp_number >> 8) & 0xFF;
			tmp_addr[2] = tmp_number & 0xFF;
			j = 0;
			bytes_can_wr_on_page = FLASH_PAGE_SIZE_ - tmp_addr[2];
		}
	}
	if (j != 0)
		Raid0Switch(spi_buf, tmp_addr, j, 3 - (file_number - 1) % 4);
}

void Raid5Switch(char* spi_buf, char* tmp_spi_buf, uint8_t* addr1_3, uint32_t*
tmp_addresses, uint8_t j, uint8_t switch_condition)
{
	switch (switch_condition)
	{
		case 0:
			addr1_3[0] = (tmp_addresses[1] >> 16) & 0xFF;
			addr1_3[1] = (tmp_addresses[1] >> 8) & 0xFF;
			addr1_3[2] = tmp_addresses[1] & 0xFF;
			FlashRead(tmp_spi_buf, addr1_3, 1, FLASH_PIN_2);
			spi_buf[j] = tmp_spi_buf[0];
			addr1_3[0] = (tmp_addresses[2] >> 16) & 0xFF;
			addr1_3[1] = (tmp_addresses[2] >> 8) & 0xFF;
			addr1_3[2] = tmp_addresses[2] & 0xFF;
			FlashRead(tmp_spi_buf, addr1_3, 1, FLASH_PIN_3);
			spi_buf[j] = spi_buf[j] ^ tmp_spi_buf[0];
			addr1_3[0] = (tmp_addresses[3] >> 16) & 0xFF;
			addr1_3[1] = (tmp_addresses[3] >> 8) & 0xFF;
			addr1_3[2] = tmp_addresses[3] & 0xFF;
			FlashRead(tmp_spi_buf, addr1_3, 1, FLASH_PIN_4);
			spi_buf[j] = spi_buf[j] ^ tmp_spi_buf[0];
			break;
		case 1:
			addr1_3[0] = (tmp_addresses[0] >> 16) & 0xFF;
			addr1_3[1] = (tmp_addresses[0] >> 8) & 0xFF;
			addr1_3[2] = tmp_addresses[0] & 0xFF;
			FlashRead(tmp_spi_buf, addr1_3, 1, FLASH_PIN_1);
			spi_buf[j] = tmp_spi_buf[0];
			addr1_3[0] = (tmp_addresses[2] >> 16) & 0xFF;
			addr1_3[1] = (tmp_addresses[2] >> 8) & 0xFF;
			addr1_3[2] = tmp_addresses[2] & 0xFF;
			FlashRead(tmp_spi_buf, addr1_3, 1, FLASH_PIN_3);
			spi_buf[j] = spi_buf[j] ^ tmp_spi_buf[0];
			addr1_3[0] = (tmp_addresses[3] >> 16) & 0xFF;
			addr1_3[1] = (tmp_addresses[3] >> 8) & 0xFF;
			addr1_3[2] = tmp_addresses[3] & 0xFF;
			FlashRead(tmp_spi_buf, addr1_3, 1, FLASH_PIN_4);
			spi_buf[j] = spi_buf[j] ^ tmp_spi_buf[0];
```

```
                        break;
                case 2:
                        addr1_3[0] = (tmp_addresses[0] >> 16) & 0xFF;
                        addr1_3[1] = (tmp_addresses[0] >> 8) & 0xFF;
                        addr1_3[2] = tmp_addresses[0] & 0xFF;
                        FlashRead(tmp_spi_buf, addr1_3, 1, FLASH_PIN_1);
                        spi_buf[j] = tmp_spi_buf[0];
                        addr1_3[0] = (tmp_addresses[1] >> 16) & 0xFF;
                        addr1_3[1] = (tmp_addresses[1] >> 8) & 0xFF;
                        addr1_3[2] = tmp_addresses[1] & 0xFF;
                        FlashRead(tmp_spi_buf, addr1_3, 1, FLASH_PIN_2);
                        spi_buf[j] = spi_buf[j] ^ tmp_spi_buf[0];
                        addr1_3[0] = (tmp_addresses[3] >> 16) & 0xFF;
                        addr1_3[1] = (tmp_addresses[3] >> 8) & 0xFF;
                        addr1_3[2] = tmp_addresses[3] & 0xFF;
                        FlashRead(tmp_spi_buf, addr1_3, 1, FLASH_PIN_4);
                        spi_buf[j] = spi_buf[j] ^ tmp_spi_buf[0];
                        break;
                case 3:
                        addr1_3[0] = (tmp_addresses[0] >> 16) & 0xFF;
                        addr1_3[1] = (tmp_addresses[0] >> 8) & 0xFF;
                        addr1_3[2] = tmp_addresses[0] & 0xFF;
                        FlashRead(tmp_spi_buf, addr1_3, 1, FLASH_PIN_1);
                        spi_buf[j] = tmp_spi_buf[0];
                        addr1_3[0] = (tmp_addresses[1] >> 16) & 0xFF;
                        addr1_3[1] = (tmp_addresses[1] >> 8) & 0xFF;
                        addr1_3[2] = tmp_addresses[1] & 0xFF;
                        FlashRead(tmp_spi_buf, addr1_3, 1, FLASH_PIN_2);
                        spi_buf[j] = spi_buf[j] ^ tmp_spi_buf[0];
                        addr1_3[0] = (tmp_addresses[2] >> 16) & 0xFF;
                        addr1_3[1] = (tmp_addresses[2] >> 8) & 0xFF;
                        addr1_3[2] = tmp_addresses[2] & 0xFF;
                        FlashRead(tmp_spi_buf, addr1_3, 1, FLASH_PIN_3);
                        spi_buf[j] = spi_buf[j] ^ tmp_spi_buf[0];
                        break;
                default:
                        break;
        }
}
```

ST7735.c

```
#include "ST7735.h"

uint16_t fcolor = ST7735_WHITE;  // Background color definition
uint16_t bcolor = ST7735_BLACK;  // Font and lines color definition

/* Helpers prototypes */

static void ST7735_Reset();
static void ST7735_WriteCommand(uint8_t cmd);
static void ST7735_WriteData(uint8_t* buff, size_t buff_size);
static void ST7735_ExecuteCommandList(const uint8_t *addr);
static void ST7735_SetAddressWindow(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1);
static void ST7735_WriteChar(uint16_t x, uint16_t y, char ch, FontDef font, uint16_t
color, uint16_t bgcolor);

/* Main functions */
```

```
void ST7735_Init()
{
    ST7735_Reset();
    ST7735_ExecuteCommandList(init_cmds1);
}

void ST7735_DrawString(uint16_t x, uint16_t y, const char* str, FontDef font)
{
    while(*str)
    {
        ST7735_WriteChar(x, y, *str++, font, fcolor, bcolor);
        x += font.width;
    }
}

void ST7735_FillScreen()
{
      ST7735_SetAddressWindow(0, 0, ST7735_WIDTH - 1, ST7735_HEIGHT - 1);

      uint8_t data[] = { bcolor >> 8, bcolor & 0xFF };
      TFT_DC_D();
      for (int y = ST7735_HEIGHT; y >= 0; y--) {
            for (int x = ST7735_WIDTH; x >= 0; x--) {
                  ST7735_WriteData(data, sizeof(data));
            }
      }
      ST7735_DrawLine(0, 143, 128, 143);
}


void ST7735_DrawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1)
{
      ST7735_SetAddressWindow(x0, y0, x1, y0);

      uint8_t data[] = { fcolor >> 8, fcolor & 0xFF };
      TFT_DC_D();
      for (int i = x0; i < x1; ++i) {
            ST7735_WriteData(data, sizeof(data));
      }

      ST7735_SetAddressWindow(x0, y0, x0, y1);
      for (int i = y0; i < y1; ++i) {
            ST7735_WriteData(data, sizeof(data));
      }
}

/* Helpers */

static void ST7735_Reset()
{
  HAL_Delay(20);
}

static void ST7735_WriteCommand(uint8_t cmd)
{
  TFT_DC_C();
  HAL_SPI_Transmit(&ST7735_SPI_PORT, &cmd, sizeof(cmd), HAL_MAX_DELAY);
}
```

```c
static void ST7735_WriteData(uint8_t* buff, size_t buff_size)
{
  TFT_DC_D();
  HAL_SPI_Transmit(&ST7735_SPI_PORT, buff, buff_size, HAL_MAX_DELAY);
}

static void ST7735_ExecuteCommandList(const uint8_t *addr)
{
    uint8_t numCommands, numArgs;
    uint16_t ms;

    numCommands = *addr++;
    while(numCommands--)
    {
      uint8_t cmd = *addr++;
        ST7735_WriteCommand(cmd);

        numArgs = *addr++;
        // If high bit set, delay follows args
        ms = numArgs & DELAY;
        numArgs &= ~DELAY;
        if(numArgs)
        {
            ST7735_WriteData((uint8_t*)addr, numArgs);
            addr += numArgs;
        }

        if(ms)
        {
            ms = *addr++;
            if(ms == 255) ms = 500;
            HAL_Delay(ms);
        }
    }
}

static void ST7735_SetAddressWindow(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1)
{
    // column address set
    ST7735_WriteCommand(ST7735_CASET);
    uint8_t data[] = { 0x00, x0 + ST7735_XSTART, 0x00, x1 + ST7735_XSTART };
    ST7735_WriteData(data, sizeof(data));

    // row address set
    ST7735_WriteCommand(ST7735_RASET);
    data[1] = y0 + ST7735_YSTART;
    data[3] = y1 + ST7735_YSTART;
    ST7735_WriteData(data, sizeof(data));

    // write to RAM
    ST7735_WriteCommand(ST7735_RAMWR);
}

static void ST7735_WriteChar(uint16_t x, uint16_t y, char ch, FontDef font, uint16_t
color, uint16_t bgcolor)
{
    uint32_t i, b, j;

    ST7735_SetAddressWindow(x, y, x+font.width-1, y+font.height-1);
```

```
    for(i = 0; i < font.height; i++)
    {
        b = font.data[(ch - 32) * font.height + i];
        for(j = 0; j < font.width; j++)
        {
            if((b << j) & 0x8000)
            {
                uint8_t data[] = { color >> 8, color & 0xFF };
                ST7735_WriteData(data, sizeof(data));
            }
            else
            {
                uint8_t data[] = { bgcolor >> 8, bgcolor & 0xFF };
                ST7735_WriteData(data, sizeof(data));
            }
        }
    }
}

/* Font definition */

static const uint16_t Font7x10 [] = {
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000,  // sp
      0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000, 0x1000, 0x0000,
0x0000,  // !
      0x2800, 0x2800, 0x2800, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000,  // "
      0x2400, 0x2400, 0x7C00, 0x2400, 0x4800, 0x7C00, 0x4800, 0x4800, 0x0000,
0x0000,  // #
      0x3800, 0x5400, 0x5000, 0x3800, 0x1400, 0x5400, 0x5400, 0x3800, 0x1000,
0x0000,  // $
      0x2000, 0x5400, 0x5800, 0x3000, 0x2800, 0x5400, 0x1400, 0x0800, 0x0000,
0x0000,  // %
      0x1000, 0x2800, 0x2800, 0x1000, 0x3400, 0x4800, 0x4800, 0x3400, 0x0000,
0x0000,  // &
      0x1000, 0x1000, 0x1000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000,  // '
      0x0800, 0x1000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x1000,
0x0800,  // (
      0x2000, 0x1000, 0x0800, 0x0800, 0x0800, 0x0800, 0x0800, 0x0800, 0x1000,
0x2000,  // )
      0x1000, 0x3800, 0x1000, 0x2800, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000,  // *
      0x0000, 0x0000, 0x1000, 0x1000, 0x7C00, 0x1000, 0x1000, 0x0000, 0x0000,
0x0000,  // +
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1000, 0x1000,
0x1000,  // ,
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x3800, 0x0000, 0x0000, 0x0000,
0x0000,  // -
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1000, 0x0000,
0x0000,  // .
      0x0800, 0x0800, 0x1000, 0x1000, 0x1000, 0x1000, 0x2000, 0x2000, 0x0000,
0x0000,  // /
      0x3800, 0x4400, 0x4400, 0x5400, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000,
0x0000,  // 0
      0x1000, 0x3000, 0x5000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000,
0x0000,  // 1
```

```
      0x3800, 0x4400, 0x4400, 0x0400, 0x0800, 0x1000, 0x2000, 0x7C00, 0x0000,
0x0000,  // 2
      0x3800, 0x4400, 0x0400, 0x1800, 0x0400, 0x0400, 0x4400, 0x3800, 0x0000,
0x0000,  // 3
      0x0800, 0x1800, 0x2800, 0x2800, 0x4800, 0x7C00, 0x0800, 0x0800, 0x0000,
0x0000,  // 4
      0x7C00, 0x4000, 0x4000, 0x7800, 0x0400, 0x0400, 0x4400, 0x3800, 0x0000,
0x0000,  // 5
      0x3800, 0x4400, 0x4000, 0x7800, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000,
0x0000,  // 6
      0x7C00, 0x0400, 0x0800, 0x1000, 0x1000, 0x2000, 0x2000, 0x2000, 0x0000,
0x0000,  // 7
      0x3800, 0x4400, 0x4400, 0x3800, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000,
0x0000,  // 8
      0x3800, 0x4400, 0x4400, 0x4400, 0x3C00, 0x0400, 0x4400, 0x3800, 0x0000,
0x0000,  // 9
      0x0000, 0x0000, 0x1000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1000, 0x0000,
0x0000,  // :
      0x0000, 0x0000, 0x0000, 0x1000, 0x0000, 0x0000, 0x0000, 0x1000, 0x1000,
0x1000,  // ;
      0x0000, 0x0000, 0x0C00, 0x3000, 0x4000, 0x3000, 0x0C00, 0x0000, 0x0000,
0x0000,  // <
      0x0000, 0x0000, 0x0000, 0x7C00, 0x0000, 0x7C00, 0x0000, 0x0000, 0x0000,
0x0000,  // =
      0x0000, 0x0000, 0x6000, 0x1800, 0x0400, 0x1800, 0x6000, 0x0000, 0x0000,
0x0000,  // >
      0x3800, 0x4400, 0x0400, 0x0800, 0x1000, 0x1000, 0x0000, 0x1000, 0x0000,
0x0000,  // ?
      0x3800, 0x4400, 0x4C00, 0x5400, 0x5C00, 0x4000, 0x4000, 0x3800, 0x0000,
0x0000,  // @
      0x1000, 0x2800, 0x2800, 0x2800, 0x2800, 0x7C00, 0x4400, 0x4400, 0x0000,
0x0000,  // A
      0x7800, 0x4400, 0x4400, 0x7800, 0x4400, 0x4400, 0x4400, 0x7800, 0x0000,
0x0000,  // B
      0x3800, 0x4400, 0x4000, 0x4000, 0x4000, 0x4000, 0x4400, 0x3800, 0x0000,
0x0000,  // C
      0x7000, 0x4800, 0x4400, 0x4400, 0x4400, 0x4400, 0x4800, 0x7000, 0x0000,
0x0000,  // D
      0x7C00, 0x4000, 0x4000, 0x7C00, 0x4000, 0x4000, 0x4000, 0x7C00, 0x0000,
0x0000,  // E
      0x7C00, 0x4000, 0x4000, 0x7800, 0x4000, 0x4000, 0x4000, 0x4000, 0x0000,
0x0000,  // F
      0x3800, 0x4400, 0x4000, 0x4000, 0x5C00, 0x4400, 0x4400, 0x3800, 0x0000,
0x0000,  // G
      0x4400, 0x4400, 0x4400, 0x7C00, 0x4400, 0x4400, 0x4400, 0x4400, 0x0000,
0x0000,  // H
      0x3800, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x3800, 0x0000,
0x0000,  // I
      0x0400, 0x0400, 0x0400, 0x0400, 0x0400, 0x0400, 0x4400, 0x3800, 0x0000,
0x0000,  // J
      0x4400, 0x4800, 0x5000, 0x6000, 0x5000, 0x4800, 0x4800, 0x4400, 0x0000,
0x0000,  // K
      0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x7C00, 0x0000,
0x0000,  // L
      0x4400, 0x6C00, 0x6C00, 0x5400, 0x4400, 0x4400, 0x4400, 0x4400, 0x0000,
0x0000,  // M
      0x4400, 0x6400, 0x6400, 0x5400, 0x5400, 0x4C00, 0x4C00, 0x4400, 0x0000,
0x0000,  // N
      0x3800, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000,
```

```
0x0000,  // O
      0x7800, 0x4400, 0x4400, 0x4400, 0x7800, 0x4000, 0x4000, 0x4000, 0x0000,
0x0000,  // P
      0x3800, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x5400, 0x3800, 0x0400,
0x0000,  // Q
      0x7800, 0x4400, 0x4400, 0x4400, 0x7800, 0x4800, 0x4800, 0x4400, 0x0000,
0x0000,  // R
      0x3800, 0x4400, 0x4000, 0x3000, 0x0800, 0x0400, 0x4400, 0x3800, 0x0000,
0x0000,  // S
      0x7C00, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000,
0x0000,  // T
      0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000,
0x0000,  // U
      0x4400, 0x4400, 0x4400, 0x2800, 0x2800, 0x2800, 0x1000, 0x1000, 0x0000,
0x0000,  // V
      0x4400, 0x4400, 0x5400, 0x5400, 0x5400, 0x6C00, 0x2800, 0x2800, 0x0000,
0x0000,  // W
      0x4400, 0x2800, 0x2800, 0x1000, 0x1000, 0x2800, 0x2800, 0x4400, 0x0000,
0x0000,  // X
      0x4400, 0x4400, 0x2800, 0x2800, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000,
0x0000,  // Y
      0x7C00, 0x0400, 0x0800, 0x1000, 0x1000, 0x2000, 0x4000, 0x7C00, 0x0000,
0x0000,  // Z
      0x1800, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x1800,  // [
      0x2000, 0x2000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0800, 0x0800, 0x0000,
0x0000,  // /
      0x3000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x3000,  // ]
      0x1000, 0x2800, 0x2800, 0x4400, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000,  // ^
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0xFE00,  // _
      0x2000, 0x1000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000,  // `
      0x0000, 0x0000, 0x3800, 0x4400, 0x3C00, 0x4400, 0x4C00, 0x3400, 0x0000,
0x0000,  // a
      0x4000, 0x4000, 0x5800, 0x6400, 0x4400, 0x4400, 0x6400, 0x5800, 0x0000,
0x0000,  // b
      0x0000, 0x0000, 0x3800, 0x4400, 0x4000, 0x4000, 0x4400, 0x3800, 0x0000,
0x0000,  // c
      0x0400, 0x0400, 0x3400, 0x4C00, 0x4400, 0x4400, 0x4C00, 0x3400, 0x0000,
0x0000,  // d
      0x0000, 0x0000, 0x3800, 0x4400, 0x7C00, 0x4000, 0x4400, 0x3800, 0x0000,
0x0000,  // e
      0x0C00, 0x1000, 0x7C00, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000,
0x0000,  // f
      0x0000, 0x0000, 0x3400, 0x4C00, 0x4400, 0x4400, 0x4C00, 0x3400, 0x0400,
0x7800,  // g
      0x4000, 0x4000, 0x5800, 0x6400, 0x4400, 0x4400, 0x4400, 0x4400, 0x0000,
0x0000,  // h
      0x1000, 0x0000, 0x7000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000,
0x0000,  // i
      0x1000, 0x0000, 0x7000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0xE000,  // j
      0x4000, 0x4000, 0x4800, 0x5000, 0x6000, 0x5000, 0x4800, 0x4400, 0x0000,
0x0000,  // k
      0x7000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000,
0x0000,  // l
```

```
      0x0000, 0x0000, 0x7800, 0x5400, 0x5400, 0x5400, 0x5400, 0x5400, 0x0000,
0x0000,  // m
      0x0000, 0x0000, 0x5800, 0x6400, 0x4400, 0x4400, 0x4400, 0x4400, 0x0000,
0x0000,  // n
      0x0000, 0x0000, 0x3800, 0x4400, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000,
0x0000,  // o
      0x0000, 0x0000, 0x5800, 0x6400, 0x4400, 0x4400, 0x6400, 0x5800, 0x4000,
0x4000,  // p
      0x0000, 0x0000, 0x3400, 0x4C00, 0x4400, 0x4400, 0x4C00, 0x3400, 0x0400,
0x0400,  // q
      0x0000, 0x0000, 0x5800, 0x6400, 0x4000, 0x4000, 0x4000, 0x4000, 0x0000,
0x0000,  // r
      0x0000, 0x0000, 0x3800, 0x4400, 0x3000, 0x0800, 0x4400, 0x3800, 0x0000,
0x0000,  // s
      0x2000, 0x2000, 0x7800, 0x2000, 0x2000, 0x2000, 0x2000, 0x1800, 0x0000,
0x0000,  // t
      0x0000, 0x0000, 0x4400, 0x4400, 0x4400, 0x4400, 0x4C00, 0x3400, 0x0000,
0x0000,  // u
      0x0000, 0x0000, 0x4400, 0x4400, 0x2800, 0x2800, 0x2800, 0x1000, 0x0000,
0x0000,  // v
      0x0000, 0x0000, 0x5400, 0x5400, 0x5400, 0x6C00, 0x2800, 0x2800, 0x0000,
0x0000,  // w
      0x0000, 0x0000, 0x4400, 0x2800, 0x1000, 0x1000, 0x2800, 0x4400, 0x0000,
0x0000,  // x
      0x0000, 0x0000, 0x4400, 0x4400, 0x2800, 0x2800, 0x1000, 0x1000, 0x1000,
0x6000,  // y
      0x0000, 0x0000, 0x7C00, 0x0800, 0x1000, 0x2000, 0x4000, 0x7C00, 0x0000,
0x0000,  // z
      0x1800, 0x1000, 0x1000, 0x1000, 0x2000, 0x2000, 0x1000, 0x1000, 0x1000,
0x1800,  // {
      0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x1000,  // |
      0x3000, 0x1000, 0x1000, 0x1000, 0x0800, 0x0800, 0x1000, 0x1000, 0x1000,
0x3000,  // }
      0x0000, 0x0000, 0x0000, 0x7400, 0x4C00, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000,  // ~
};

FontDef Font_7x10 = { 6,10,Font7x10 };
```

main.c

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
```

```c
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "ST7735.h"
#include "raid.h"
#include "flash.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define UartCommandSize 32U
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
SPI_HandleTypeDef hspi1;

TIM_HandleTypeDef htim2;

UART_HandleTypeDef huart1;

/* USER CODE BEGIN PV */
uint8_t TimerOverflowCount;
uint8_t FileLifetime;
uint8_t NumberOfFiles;
uint8_t CurrentRaid;
uint8_t RaidAddresses[12];
uint32_t FreeSpace;
uint8_t SpeedFlag;
char ScreenBuffer[32];
char UartBuffer[64];
char MSGUnknownCommand[] = "Unknown command\r\n";
char MSGFilesCreation[] = "\r\nCreated files with size ";
char MSGDiskCleaned[] = "\r\nDisk cleaned\r\n";
char MSGRaidAlreadySelected[] = "\r\nThis mode is already selected\r\n";
char MSGRaidModeChanged[] = "\r\nRaid mode changed\r\n";
char MSGInvalidMode[] = "\r\nInvalid mode\r\n";
char MSGInvalidInput[] = "\r\nInvalid input\r\n";
char MSGFunctionSelection0[] = "Enter number to select required function:\r\n";
char MSGFunctionSelection1[] = "1 - Create temporary files\r\n";
char MSGFunctionSelection2[] = "2 - Change RAID storage mode\r\n";
char MSGFunctionSelection3[] = "3 - Clean the selected disk\r\n";
char MSGFunctionSelection4[] = "4 - Recover data on the selected disk\r\n";
char MSGNumberOfFiles[] = "\r\nEnter 3-digit number of temporary files\r\n";
char MSGFilesLifetime[] = "\r\nEnter 3-digit lifetime of files (from 1 to 255)\r\n";
char MSGStorageMode[] = "\r\nEnter storage mode[0/1/3/4/5]\r\n";
char MSGCleanDisk[] = "\r\nEnter number of disk to clean[1/2/3/4]\r\n";
```

```c
char MSGNewLine[] = "\r\n";
char MSGDeletingFiles[] = "\r\nDeleting files ... ";
char MSGDone[] = "Done!\r\n";
char MSGRecoverDisk[] = "\r\nEnter number of disk to recover[1/2/3/4]\r\n";
char MSGDiskRecovered[] = "\r\nDisk recovered\r\n";
char MSGRaidNoRecovery[] = "\r\nCurrent RAID don't support recovery\r\n";
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_SPI1_Init(void);
static void MX_TIM2_Init(void);
/* USER CODE BEGIN PFP */
void ResetScreen();
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART1_UART_Init();
  MX_SPI1_Init();
  MX_TIM2_Init();
  /* USER CODE BEGIN 2 */

//  RaidCleanAllDisks();

  TimerOverflowCount = 0;
```

```
  FileLifetime = 0;
  CurrentRaid = 0;
  SpeedFlag = 0;
  NumberOfFiles = RaidSetAddresses(RaidAddresses);
  uint32_t tmp_number = 0;
  for (uint8_t i = 0; i < 3; i++)
        tmp_number += (RaidAddresses[3 * i] << 16) | (RaidAddresses[3 * i + 1] << 8)
| RaidAddresses[3 * i + 2];
  FreeSpace = 0x200000 - tmp_number;
  HAL_TIM_Base_Stop_IT(&htim2);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_RESET);
  ST7735_Init();
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_SET);
  ResetScreen();
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

        strcpy(UartBuffer, MSGFunctionSelection0);
        HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer), 100);
        strcpy(UartBuffer, MSGFunctionSelection1);
        HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer), 100);
        strcpy(UartBuffer, MSGFunctionSelection2);
        HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer), 100);
        strcpy(UartBuffer, MSGFunctionSelection3);
        HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer), 100);
        strcpy(UartBuffer, MSGFunctionSelection4);
        HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer), 100);

        if (HAL_UART_Receive(&huart1, (uint8_t*)UartBuffer, 1, HAL_MAX_DELAY) ==
HAL_OK)
        {
              uint8_t func_number;
              if (sscanf(UartBuffer, "%hu", &func_number) == 1)
              {
                      uint8_t tmp_num1, tmp_num2;
                      uint8_t for_test;
                        switch (func_number)
                        {
                              case 1:
                                    strcpy(UartBuffer, MSGNumberOfFiles);
                                    HAL_UART_Transmit(&huart1,
(uint8_t*)UartBuffer, strlen(UartBuffer), 100);
                                    if (HAL_UART_Receive(&huart1,
(uint8_t*)UartBuffer, 3, HAL_MAX_DELAY) == HAL_OK)
                                    {
                                          if (sscanf(UartBuffer, "%hu",
&for_test) == 1)
                                          {
                                                strcpy(UartBuffer,
MSGFilesLifetime);
                                                HAL_UART_Transmit(&huart1,
(uint8_t*)UartBuffer, strlen(UartBuffer), 100);
```

```
                                                    if (HAL_UART_Receive(&huart1,
(uint8_t*)UartBuffer, 3, HAL_MAX_DELAY) == HAL_OK)
                                                    {

                                                           if (sscanf(UartBuffer,
"%hu", &tmp_num2) == 1)
                                                           {
                                                                  if (NumberOfFiles
!= 0)
                                                                  {

HAL_TIM_Base_Stop_IT(&htim2);

RaidFastCleanAllDisks(RaidAddresses);
                                                                         FreeSpace =
0x200000;

TimerOverflowCount = 0;

NumberOfFiles = 0;
                                                                         for
(uint8_t i = 0; i < 12; i++)

RaidAddresses[i] = 0;
                                                                  }
                                                                  srand(tmp_num2);
                                                                  uint32_t max =
30;
                                                                  uint32_t min =
10;
                                                                  FileLifetime =
tmp_num2;

strcpy(UartBuffer, MSGFilesCreation);

HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer), HAL_MAX_DELAY);
                                                                  uint32_t
tmp_file_size;
                                                                  SpeedFlag = 1;
                                                                  ResetScreen();
                                                                  for (uint8_t i =
0; i < for_test; i++)
                                                                  {

tmp_file_size = (rand() % (max - min + 1)) + min;

NumberOfFiles++;
                                                                         FreeSpace -
= tmp_file_size;

sprintf(UartBuffer, "%d ", tmp_file_size);

HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer), HAL_MAX_DELAY);
                                                                         switch
(CurrentRaid)
                                                                         {

      case 0:
```

```
                Raid0Write(RaidAddresses, tmp_file_size, tmp_num2);

                break;

        case 1:

                if (0x1FFFFF - ((RaidAddresses[0] << 16) | (RaidAddresses[1] << 8) |
RaidAddresses[2]) < tmp_file_size)

                        Raid1Write(RaidAddresses, tmp_file_size, tmp_num2, 2);

                else

                        Raid1Write(RaidAddresses, tmp_file_size, tmp_num2, 1);

                break;

        case 3:

                Raid3Write(RaidAddresses, tmp_file_size, tmp_num2);

                break;

        case 4:

                Raid4Write(RaidAddresses, tmp_file_size, tmp_num2);

                break;

        case 5:

                Raid5Write(RaidAddresses, tmp_file_size, tmp_num2, NumberOfFiles);

                break;

        default:

                break;
                                                                                }
                                                                        }

strcpy(UartBuffer, MSGNewLine);

HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer), 100);

HAL_TIM_Base_Start_IT(&htim2);
                                                                          SpeedFlag = 0;
                                                                          ResetScreen();
                                                                }
                                                                else
                                                                {
                                                                        strcpy(UartBuffer,
MSGInvalidInput);

        HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer), 100);
                                                                }
                                                        }
                                                        else
                                                        {
```

```
                                                        strcpy(UartBuffer,
MSGInvalidInput);
                                                        HAL_UART_Transmit(&huart1,
(uint8_t*)UartBuffer, strlen(UartBuffer), 100);
                                                }
                                        }
                                        else
                                        {
                                                strcpy(UartBuffer,
MSGInvalidInput);
                                                HAL_UART_Transmit(&huart1,
(uint8_t*)UartBuffer, strlen(UartBuffer), 100);
                                        }
                                }
                                else
                                {
                                        strcpy(UartBuffer, MSGInvalidInput);
                                        HAL_UART_Transmit(&huart1,
(uint8_t*)UartBuffer, strlen(UartBuffer), 100);
                                }
                                break;
                        case 2:
                                strcpy(UartBuffer, MSGStorageMode);
                                HAL_UART_Transmit(&huart1,
(uint8_t*)UartBuffer, strlen(UartBuffer), 100);
                                if (HAL_UART_Receive(&huart1,
(uint8_t*)UartBuffer, 1, HAL_MAX_DELAY) == HAL_OK)
                                {
                                        if (sscanf(UartBuffer, "%hu",
&tmp_num1) == 1)
                                        {
                                                if (CurrentRaid == tmp_num1)
                                                {
                                                        strcpy(UartBuffer,
MSGRaidAlreadySelected);

HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer), HAL_MAX_DELAY);
                                                }
                                                else
                                                {
                                                        if (tmp_num1 != 0 &&
tmp_num1 != 1 && tmp_num1 != 3 && tmp_num1 != 4 && tmp_num1 != 5)
                                                        {

strcpy(UartBuffer, MSGInvalidMode);

HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer), HAL_MAX_DELAY);
                                                        }
                                                        else
                                                        {
                                                                if (NumberOfFiles
!= 0)
                                                                {

HAL_TIM_Base_Stop_IT(&htim2);

TimerOverflowCount = 0;

NumberOfFiles = 0;
```

```
                                                                    FreeSpace =
0x200000;
                                                                    for
(uint8_t i = 0; i < 12; i++)
RaidAddresses[i] = 0;
                                                                }
RaidFastCleanAllDisks(RaidAddresses);
                                                                CurrentRaid =
tmp_num1;
strcpy(UartBuffer, MSGRaidModeChanged);
HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer), HAL_MAX_DELAY);
                                                                ResetScreen();
                                                            }
                                                        }
                                                    }
                                                    else
                                                    {
                                                        strcpy(UartBuffer,
MSGInvalidInput);
                                                        HAL_UART_Transmit(&huart1,
(uint8_t*)UartBuffer, strlen(UartBuffer), 100);
                                                    }
                                                }
                                                else
                                                {
                                                    strcpy(UartBuffer, MSGInvalidInput);
                                                    HAL_UART_Transmit(&huart1,
(uint8_t*)UartBuffer, strlen(UartBuffer), 100);
                                                }
                                                break;
                                            case 3:
                                                strcpy(UartBuffer, MSGCleanDisk);
                                                HAL_UART_Transmit(&huart1,
(uint8_t*)UartBuffer, strlen(UartBuffer), 100);
                                                if (HAL_UART_Receive(&huart1,
(uint8_t*)UartBuffer, 1, HAL_MAX_DELAY) == HAL_OK)
                                                {
                                                    if (sscanf(UartBuffer, "%hu",
&tmp_num1) == 1)
                                                    {
                                                        if (tmp_num1 != 1 && tmp_num1
!= 2 && tmp_num1 != 3 && tmp_num1 != 4)
                                                        {
                                                            strcpy(UartBuffer,
MSGInvalidInput);
HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer), HAL_MAX_DELAY);
                                                        }
                                                        else
                                                        {
RaidFastCleanDisk(RaidAddresses, tmp_num1);
                                                            FreeSpace +=
(RaidAddresses[(tmp_num1 - 1) * 3] << 16) | (RaidAddresses[(tmp_num1 - 1) * 3 + 1] <<
8) | RaidAddresses[(tmp_num1 - 1) * 3 + 2];
```

```
                                                                for (uint8_t i = 0; i <
3; i++)

RaidAddresses[(tmp_num1 - 1) * 3 + i] = 0;
                                                                strcpy(UartBuffer,
MSGDiskCleaned);

HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer), HAL_MAX_DELAY);
                                                                ResetScreen();
                                                          }
                                                    }
                                                    else
                                                    {
                                                          strcpy(UartBuffer,
MSGInvalidInput);
                                                          HAL_UART_Transmit(&huart1,
(uint8_t*)UartBuffer, strlen(UartBuffer), HAL_MAX_DELAY);
                                                    }
                                              }
                                              else
                                              {
                                                    strcpy(UartBuffer, MSGInvalidInput);
                                                    HAL_UART_Transmit(&huart1,
(uint8_t*)UartBuffer, strlen(UartBuffer), HAL_MAX_DELAY);
                                              }
                                              break;
                                        case 4:
                                              if (CurrentRaid == 0 || CurrentRaid == 1)
                                              {
                                                    strcpy(UartBuffer, MSGRaidNoRecovery);
                                                    HAL_UART_Transmit(&huart1,
(uint8_t*)UartBuffer, strlen(UartBuffer), HAL_MAX_DELAY);
                                              }
                                              else
                                              {
                                                    strcpy(UartBuffer, MSGRecoverDisk);
                                                    HAL_UART_Transmit(&huart1,
(uint8_t*)UartBuffer, strlen(UartBuffer), HAL_MAX_DELAY);
                                                    if (HAL_UART_Receive(&huart1,
(uint8_t*)UartBuffer, 1, HAL_MAX_DELAY) == HAL_OK)
                                                    {
                                                          if (sscanf(UartBuffer, "%hu",
&tmp_num1) == 1)
                                                          {
                                                                if (tmp_num1 != 1 &&
tmp_num1 != 2 && tmp_num1 != 3 && tmp_num1 != 4)
                                                                {

strcpy(UartBuffer, MSGInvalidInput);

HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer), HAL_MAX_DELAY);
                                                                }
                                                                else
                                                                {

                                                                      SpeedFlag = 1;
                                                                      ResetScreen();

RaidRecoverDisk(RaidAddresses, tmp_num1);
```

```
                                                                FreeSpace -=
(RaidAddresses[(tmp_num1 - 1) * 3] << 16) | (RaidAddresses[(tmp_num1 - 1) * 3 + 1] <<
8) | RaidAddresses[(tmp_num1 - 1) * 3 + 2];

strcpy(UartBuffer, MSGDiskRecovered);

HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer), HAL_MAX_DELAY);
                                                                SpeedFlag = 0;
                                                                ResetScreen();
                                                        }
                                                }
                                                else
                                                {
                                                        strcpy(UartBuffer,
MSGInvalidInput);
                                                        HAL_UART_Transmit(&huart1,
(uint8_t*)UartBuffer, strlen(UartBuffer), HAL_MAX_DELAY);
                                                }
                                        }
                                        else
                                        {
                                                strcpy(UartBuffer,
MSGInvalidInput);
                                                HAL_UART_Transmit(&huart1,
(uint8_t*)UartBuffer, strlen(UartBuffer), HAL_MAX_DELAY);
                                        }
                                }
                                break;
                        default:
                                strcpy(UartBuffer, MSGInvalidInput);
                                HAL_UART_Transmit(&huart1,
(uint8_t*)UartBuffer, strlen(UartBuffer), HAL_MAX_DELAY);
                                break;
                        }
                }
                else
                {
                        strcpy(UartBuffer, MSGInvalidInput);
                        HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer,
strlen(UartBuffer), HAL_MAX_DELAY);
                }
        }

  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
```

```
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief SPI1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI1_Init(void)
{

  /* USER CODE BEGIN SPI1_Init 0 */

  /* USER CODE END SPI1_Init 0 */

  /* USER CODE BEGIN SPI1_Init 1 */

  /* USER CODE END SPI1_Init 1 */
  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI1_Init 2 */

  /* USER CODE END SPI1_Init 2 */
```

```
}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 7999;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 999;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */

}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */
```

```
  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4
                          |GPIO_PIN_11, GPIO_PIN_SET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);

  /*Configure GPIO pins : PA1 PA2 PA3 PA4
                           PA8 PA11 */
  GPIO_InitStruct.Pin = GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4
                          |GPIO_PIN_8|GPIO_PIN_11;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
```

```
void Timer_OverflowHandler()
{
	TimerOverflowCount++;
	if (TimerOverflowCount == FileLifetime)
	{
		strcpy(UartBuffer, MSGDeletingFiles);
		HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer),
HAL_MAX_DELAY);
		TimerOverflowCount = 0;
		RaidFastCleanAllDisks(RaidAddresses);
		FreeSpace = 0x200000;
		for (uint8_t i = 0; i < 12; i++)
			RaidAddresses[i] = 0;
		HAL_TIM_Base_Stop_IT(&htim2);
		strcpy(UartBuffer, MSGDone);
		HAL_UART_Transmit(&huart1, (uint8_t*)UartBuffer, strlen(UartBuffer),
HAL_MAX_DELAY);
		ResetScreen();
	}
}

void ResetScreen() {
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_RESET);
	ST7735_FillScreen();
	sprintf(ScreenBuffer, "RAID: %d", CurrentRaid);
	ST7735_DrawString(3, 3, ScreenBuffer, Font_7x10);
	sprintf(ScreenBuffer, "Total memory");
	ST7735_DrawString(3, 18, ScreenBuffer, Font_7x10);
	sprintf(ScreenBuffer, "capacity: 2048 KB");
	ST7735_DrawString(3, 33, ScreenBuffer, Font_7x10);
	sprintf(ScreenBuffer, "Free memory: %d KB", FreeSpace / 1024);
	ST7735_DrawString(3, 48, ScreenBuffer, Font_7x10);
	sprintf(ScreenBuffer, "Used memory: %d KB", (0x200000 - FreeSpace) / 1024);
	ST7735_DrawString(3, 63, ScreenBuffer, Font_7x10);
	sprintf(ScreenBuffer, "Write speed:");
	ST7735_DrawString(3, 78, ScreenBuffer, Font_7x10);
	if (SpeedFlag == 0)
		sprintf(ScreenBuffer, "0 KBits/s");
	else
		sprintf(ScreenBuffer, "1000 KBits/s");
	ST7735_DrawString(3, 93, ScreenBuffer, Font_7x10);
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_SET);
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
```

```c
#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Приложение Б

Перечень элементов

На 2 листах