



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 09.03.03 Прикладная информатика

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

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

**Устройство для проверки целостности данных на
диске**

Студент

(Группа)

(Подпись, дата)

К.А. Коптелов

(И.О. Фамилия)

Руководитель

(Подпись, дата)

И.Б. Трамов

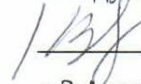
(И.О. Фамилия)

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

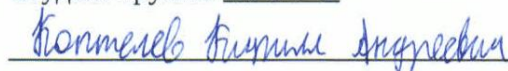
 А.В. Пролетарский

«2» сентября 2023 г.

ЗАДАНИЕ на выполнение курсовой работы

по дисциплине Микропроцессорные системы

Студент группы ИУ6-7 Б



Тема курсовой работы: Устройство для проверки целостности данных на диске

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 проверки целостности данных на внешнем диске. Эталонная контрольная сумма передается оператором с ПЭВМ. Реализовать несколько алгоритмов для подсчета контрольной суммы диска и сравнения с эталонной. Предусмотреть возможность выбора алгоритма с помощью кнопок управления и ПЭВМ. Результат и время проверки выводить на ЖК-дисплей и ПЭВМ. При несовпадении контрольных сумм выдавать соответствующий аудио сигнал на SPEAKER.

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.

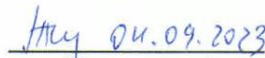
Оформление курсовой работы:

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.

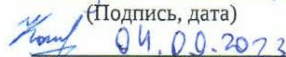
Дата защиты: «20» декабря 2023 г.

Руководитель курсовой работы


(Подпись, дата)

И.Б. Грамов
(И.О.Фамилия)

Студент


(Подпись, дата)

К.А. Коптелев
(И.О.Фамилия)

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

РЕФЕРАТ

РПЗ 48 страниц, 21 рисунок, 8 таблиц, 13 источников, 3 приложения.

МИКРОКОНТРОЛЛЕР, СИСТЕМА, КОНТРОЛЬНАЯ СУММА, ВНЕШНИЙ ДИСК

Объектом разработки является устройство для проверки целостности данных на внешнем диске

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

ВВЕДЕНИЕ.....	7
1 Конструкторская часть	8
1.1 Анализ требований и принцип работы системы	8
1.2 Проектирование функциональной схемы	10
1.2.1 Микроконтроллер STM32F103C8T6.....	10
1.2.1.1 Используемые элементы	17
1.2.1.2 Распределение портов	18
1.2.1.3 Организация памяти	19
1.2.2 Прием данных от ПЭВМ	20
1.2.3 Настройка USART для взаимодействия с ПЭВМ	22
1.2.4 LCD-дисплей ST7735.....	30
1.2.5 Настройка SPI для взаимодействия с LCD-дисплеем	33
1.2.6 Настройка SPI для взаимодействия с сокетом SD-карты.	41
1.2.7 Использование таймера для генерации звукового сигнала ...	43
1.2.8 Построение функциональной схемы	48
1.3 Проектирование принципиальной схемы.....	48
1.3.1 Разъем программатора.....	48
1.3.2 Расчет потребляемой мощности.....	49
1.4 Алгоритмы работы системы	51
1.4.1 Общее описание работы программы	51
1.4.2 Детализация и пояснение основных функций	52
2 Технологическая часть	55
2.1 Отладка и тестирование программы	55

2.2 Симуляция работы системы.....	55
2.3 Способы программирования МК	58
ЗАКЛЮЧЕНИЕ	59
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	60
Приложение А	62
Приложение Б.....	111

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

МК – микроконтроллер.

ТЗ – техническое задание.

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

MD5, CRC – алгоритмы для вычисления контрольной суммы.

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

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

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 устройства для проверки целостности данных на диске.

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, четырех кнопок для управления процессом проверки целостности, кнопки RESET, LED-дисплея для отображения информации о проверке целостности и подсказок пользователю, разъема для SD-карты, которая выполняет роль внешнего диска, динамика для вывода сигнала о несовпадении контрольных сумм, виртуального терминала для симуляции ввода/вывода с ПЭВМ. Также имеется возможность загружать специально созданный образ SD-карты в SD-карту.

Актуальность разрабатываемого устройства для проверки целостности данных на диске заключается в том, что в современном информационном мире, где цифровые данные играют ключевую роль, обеспечение их целостности является критическим аспектом. Эта система предоставляет пользователям возможность быстрой и надежной проверки целостности данных несколькими алгоритмами, что является важным шагом в обеспечении безопасности и надежности хранимой информации.

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 чтение внешнего диска, в данном случае SD-карты, чтение эталонной контрольной суммы с ПЭВМ, вычисление контрольной суммы внешнего диска выбранным алгоритмом и сравнение с эталонной, а также вывод отчета об операции.

Система проверки целостности данных на диске имеет три фазы – выбор алгоритма вычисления контрольной суммы, ввод эталонной контрольной суммы с ПЭВМ и вычисление контрольной суммы с выводом отчета, в котором содержится результат сравнения и время вычисления контрольной суммы.

Таблица 1 – Работа устройства для проверки целостности данных на диске

Этап работы	Описание
Переключение на 1 фазу	Вывод текущего алгоритма на дисплее, предложение ввести одну из фраз с ПЭВМ
Постоянный эффект фазы 1	Вывод текущего алгоритма на дисплее, вывод на ПЭВМ предложения ввести управляющую команду
Переключение на 2 фазу	Вывод на ПЭВМ подсказки для ввода контрольной суммы определенного размера
Постоянный эффект фазы 2	—
Переключение на фазу 3	Вычисление контрольной суммы диска, вывод отчета на LED-дисплей, вывод звука на динамик при несовпадении контрольных сумм
Постоянный эффект фазы 3	—

После выполнения последнего действия у пользователя есть возможность либо нажать на кнопку «Restart», что переключит программу на начальную фазу, при этом SD-карта не будет смонтирована снова, или же нажать кнопку «Reset», тогда выполнение программы будет перезапущено на уровне микроконтроллера.

Также пользователь может нажать любую из этих кнопок в любую из фаз работы программы.

Кроме того, в программе, как и говорилось выше, есть несколько способов выбрать алгоритм – с помощью ПЭВМ или с помощью кнопок. Существует два сценария:

- Пользователь нажимает кнопку NEXT или PREV, тогда текущий алгоритм переключается на следующий или предыдущий соответственно.
- Пользователь вводит в терминал слово «NEXT» или «PREV», тогда текущий алгоритм также переключается на следующий или предыдущий соответственно. При этом ввод любой другой фразы не вызывает никаких действий.

Всего программа поддерживает три алгоритма вычисления контрольной суммы – один поддерживаемый аппаратно и два реализованных вручную. Аппаратно поддерживаемый алгоритм – CRC. Написанные вручную – CRC8 по алгоритму Купмена и MD5. Стоит заметить, что в зависимости от алгоритма контрольная сумма имеет разный размер. При этом контрольная сумма всегда выводится в шестнадцатеричном формате.

Таблица 2 – Доступные в программе алгоритмы

Название алгоритма	Размер контрольной суммы в символах
Аппаратная реализация CRC	8
Реализация CRC8 по алгоритму Купмена	8
MD5	32

После вычисление контрольной суммы она выводится на LED-дисплей, при этом также выводится, совпадает она с эталонной или нет. Если не совпадает, то на динамик в течение секунды выводится звуковой сигнал, обозначающий несовпадение.

Разработанная структурная схема устройства для проверки целостности данных на диске представлена на рисунке 1.

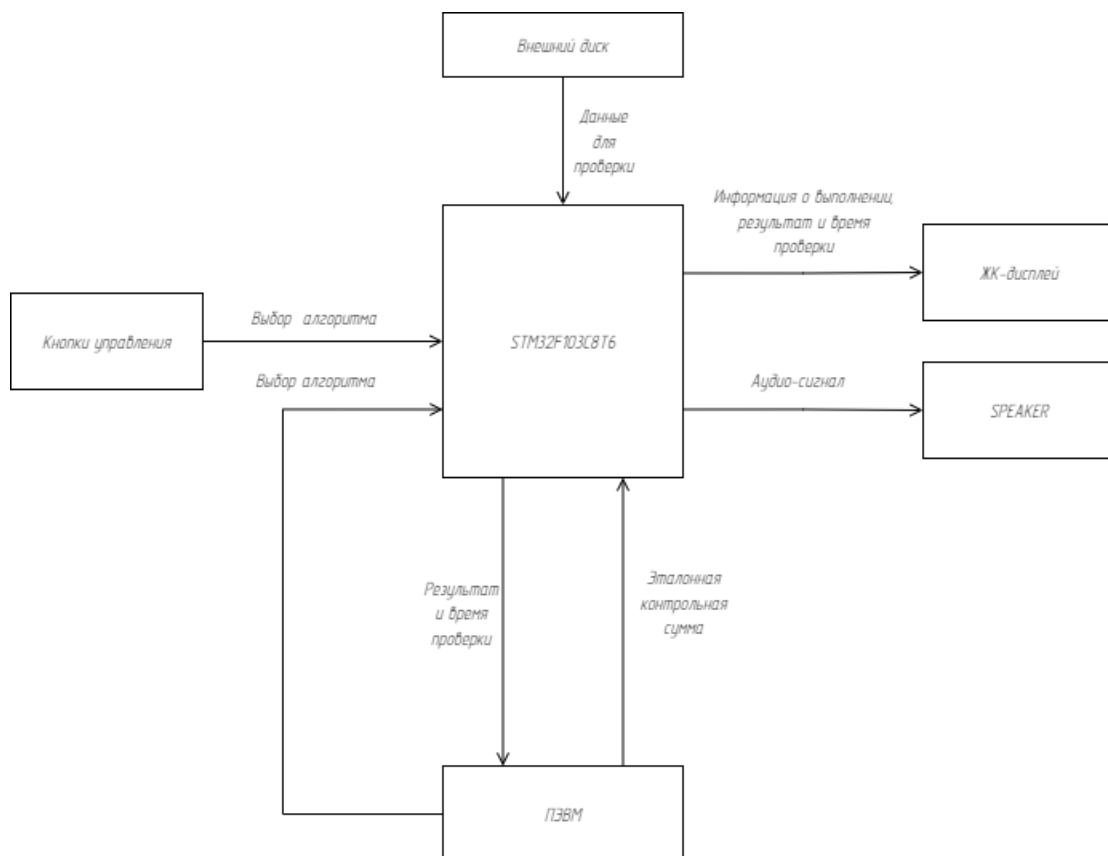


Рисунок 1 – Структурная схема устройства

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 3.

Таблица 3 – 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 инструк- цию	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 SD-картой. Кроме того, для программы потенциально

понадобится большой объем FLASH-памяти, чтобы вместить работу конечного автомата, работу с SD-картой и алгоритмы.

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, например, для взаимодействия с файловой системой FAT на SD-карте. 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 программирования SD-карты и для ЖК-дисплея;
- интерфейс UART для ПЭВМ(виртуального терминала);
- 20 Кбайт RAM;
- 4 таймера, которые могут быть использованы в режиме ШИМ для генерации звукового сигнала;
- 64 Кбайта FLASH-памяти;
- Возможность назначить внешнее прерывание практически на любой PIN;
- Поддержка CRC для вычисления контрольной суммы;
- Поддержка FATFS для файловой системы FAT на SD-карте;
- 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 2 и УГО на рисунке 3 [3].

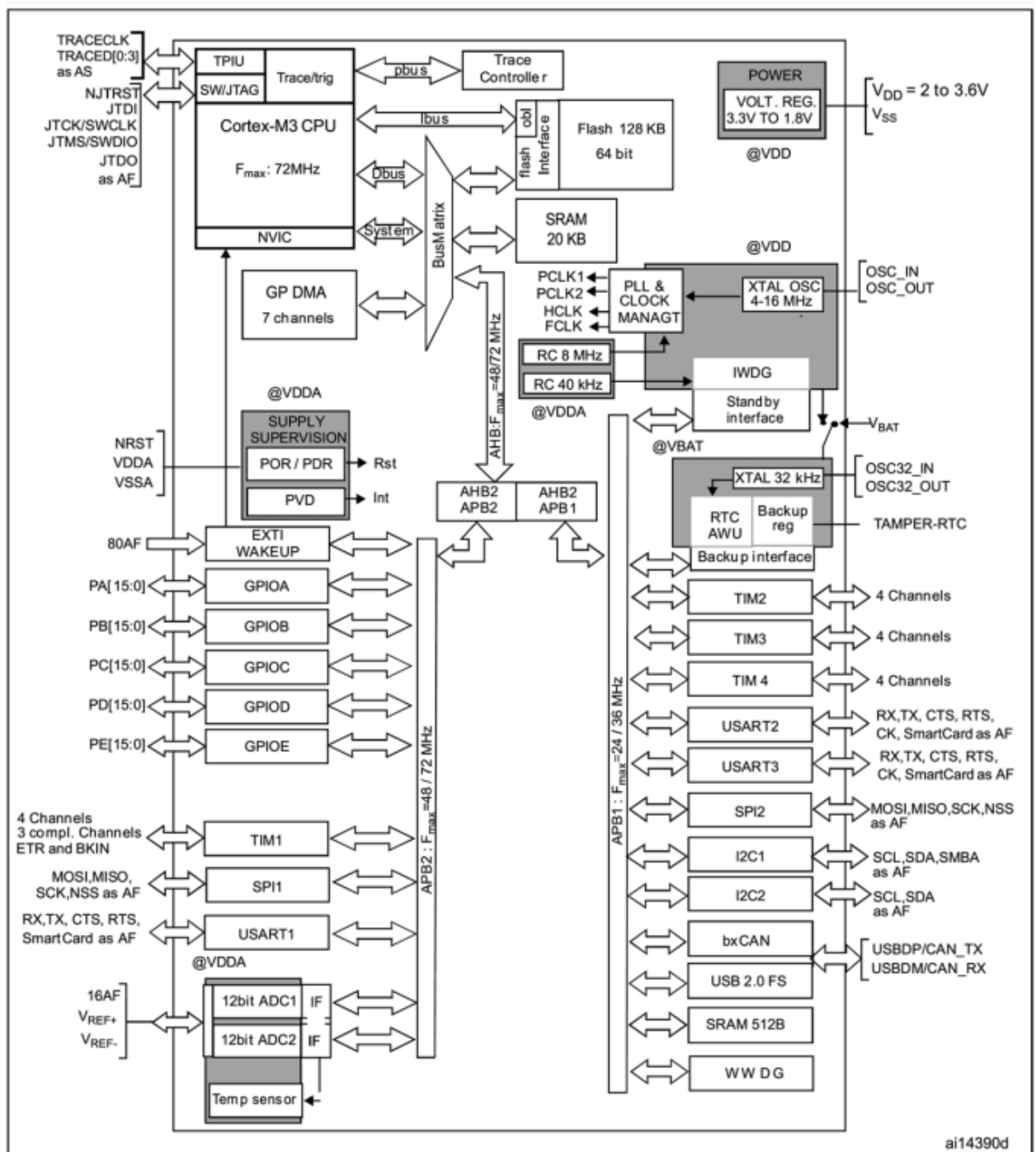


Рисунок 2 – Структурная схема МК STM32F103C8T6

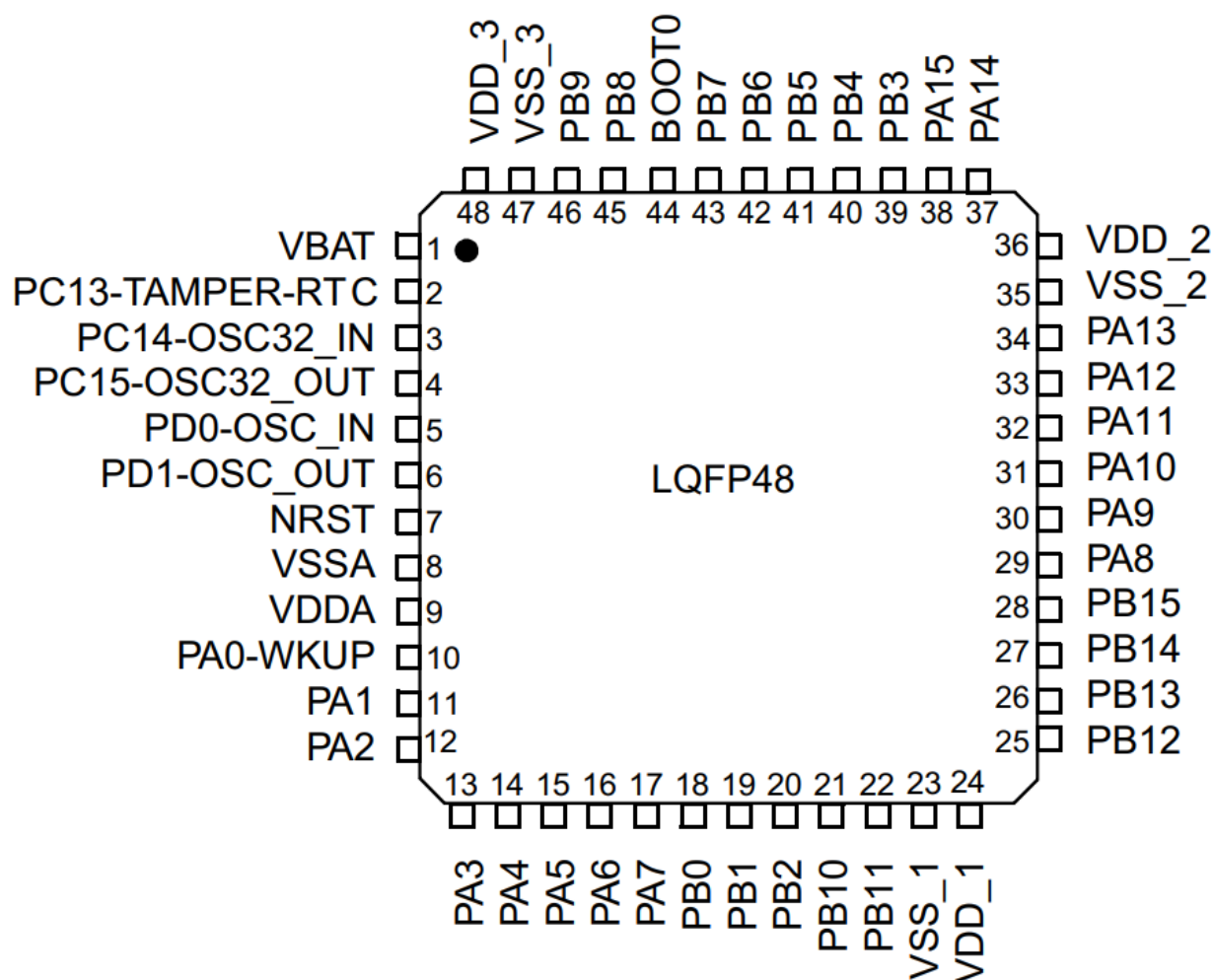


Рисунок 3 – УГО МК STM32F103C8T6

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 устройства проверки целостности данных на диске в МК STM32F103C8T6 задействованы не все элементы его архитектуры. Выделим и опишем те, что используются во время функционирования схемы.

- Порты А, В – 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 исполнению 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 портов для обработки нажатия кнопок и прерывания UART.

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

– SPI (Serial Peripheral Interface) – интерфейс для связи МК с другими внешними устройствами. В устройстве используется для 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 генерации ШИМ сигнала для динамика.

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:

- PA2 – отправка данных по UART на ПЭВМ;
- PA3 – получение данных по UART с ПЭВМ;
- PA5 – тактовый сигнал (SCK) для SPI для LCD-дисплея;
- PA7 – MOSI-пин для SPI для LCD-дисплея;
- PA10 – RES-пин(RESET) для RESET-сигнала для LCD-дисплея;

- PA11 – DC-пин(Data or Command) для передачи данных или команд на LCD-дислей;
- PA12 – CS-пин(Chip Select) для определения активного устройства, взаимодействующего по SPI(в нашем случае это всегда микроконтроллер) с LCD-дисплеем;
- PA15 – выход первого канала таймера TIM2, на котором генерируется ШИМ для динамика.

Порт В:

- PB5 – пин с внешним прерыванием по нажатию кнопки переключения на следующий алгоритм;
- PB6 – пин с внешним прерыванием по нажатию кнопки переключения на предыдущий алгоритм;
- PB7 – пин с внешним прерыванием по нажатию кнопки подтверждения выбора алгоритма;
- PB8 – пин с внешним прерыванием по нажатию кнопки перезапуска вычислений;
- PB10 – CS-пин(Chip Select) для определения активного устройства, взаимодействующего по SPI(в нашем случае это всегда микроконтроллер) с сокетом SD-карты;
- PB13 - тактовый сигнал (SCK) для SPI для сокета SD-карты;
- PB14 – MISO-пин для SPI для сокета SD-карты;

1.2.1.3 Организация памяти

Схема организации памяти МК STM32F103C8T6 показана на рисунке 4.

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

Внутреннее изображение MAX232 показано на рисунке 5. Назначение пинов описано в таблице 4 [4].

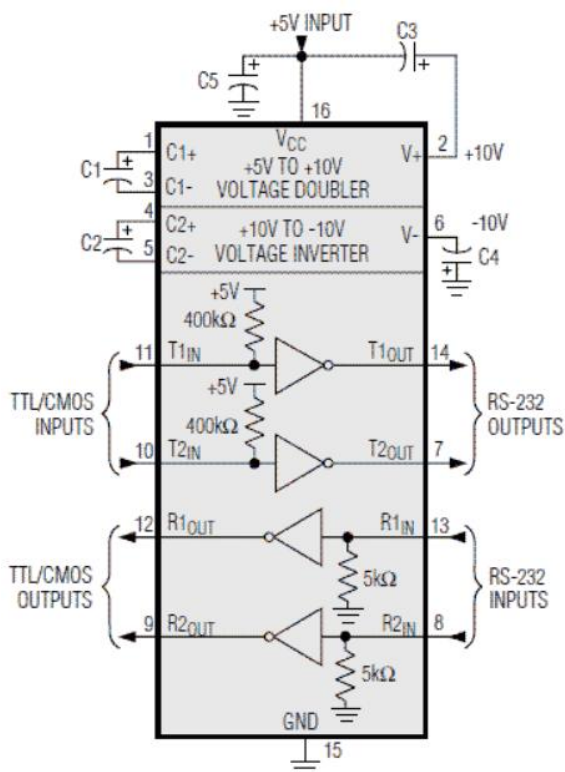


Рисунок 5 – Преобразователь MAX232

Таблица 4 - 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
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.

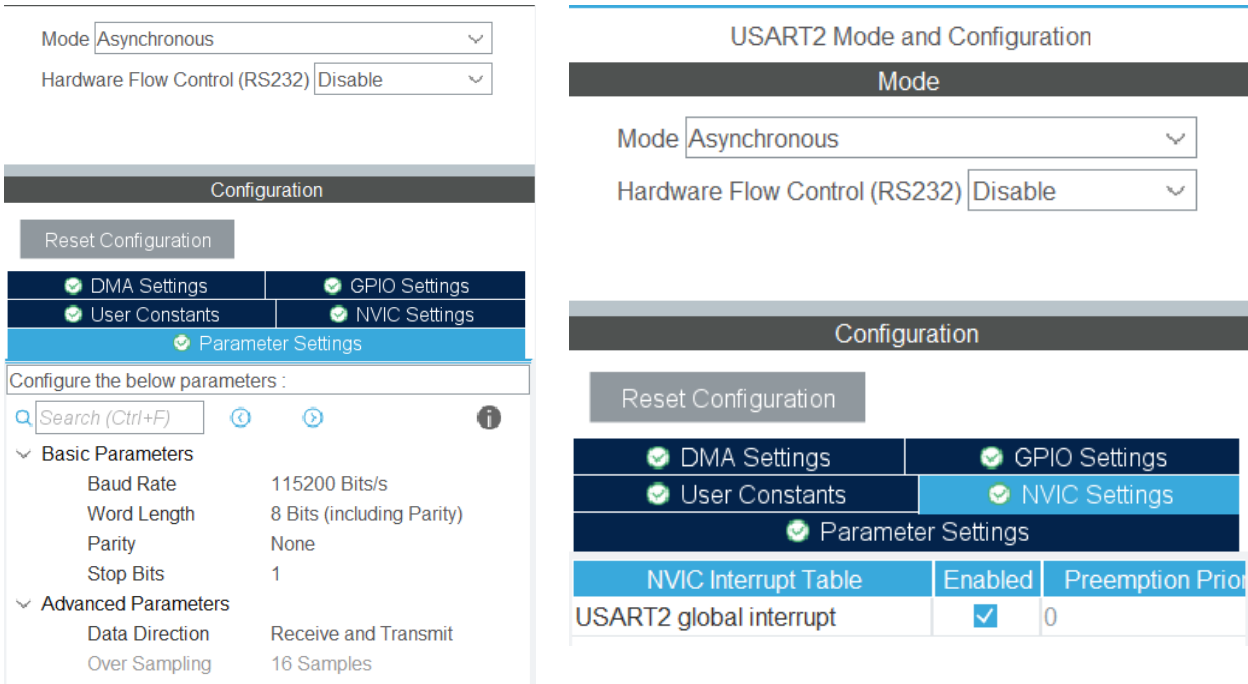
USART может настраиваться на разные скорости передачи данных (баудрейты), количество бит данных, контроль четности, стоповые биты и

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 выбор стоял между ним и I2C – оба интерфейса SPI уже были заняты под более критичные по скорости задачи. USART был выбран как более быстрый интерфейс; кроме того, 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 рисунках 6 и 7.



Рисунки 6,7 – 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. Кроме того, включены прерывания для USART.

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 8.

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	rw

Рисунок 8 – 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 стартового 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 послыску. Бит может быть установлен и сброшен программно. Его необходимо программно установить в 1 для формирования Break послыски, 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 9.

USART_BRR (Baud rate 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
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 9 – 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 10 представлена схема, показывающая, как именно высчитывается скорость передачи.

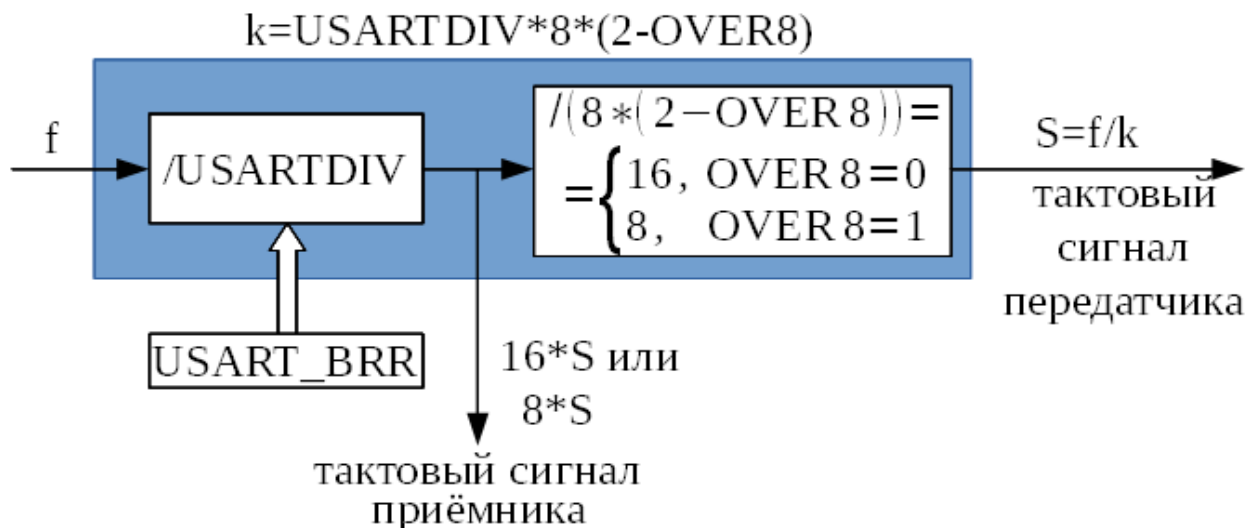


Рисунок 10 – Вычисление скорости приема и передачи

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

Далее рассмотрим USART_DR – регистр, через который передаются непосредственно данные. Он представлен на рисунке 11.

USART_DR (Data register)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								8	7	6	5	4	3	2	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								DR[8:0]							
								rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 11 – 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 12.

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
						re_w0	re_w0	r	re_w0	re_w0	r	r	r	r	r

Рисунок 12 – 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 сбрасывается 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, а объемы текста могут сильно отличаться в зависимости от состояния автомата, необходима поддержка различных цветов и шрифтов.

Размер шрифта 11x18 пикселей на символ является подходящим по читаемости и занимает достаточно места чтобы отобразить большую часть

текста. Однако контрольная сумма, вычисленная с помощью MD5, занимает 32 символа, и при это нежелательно, чтобы она занимала много строк. Для нее был выбран шрифт 7x10.

Для выбора размера экрана проведем расчет в пикселях.

В ширину необходимо максимум 11 символов шрифта 11x18, в высоту 6 строк шрифта 11x18 и две строки 7x10.

Тогда в высоту необходимо $18 * 6 + 2 * 10 = 128$ пикселей, в ширину – $11 * 11 = 121$ пиксель.

Получилось, что достаточный размер экрана 121x128 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 5.

Таблица 5 – 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 однокристалльный 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 6 представлены возможные значения.

Таблица 6 – 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.

1.2.5 Настройка SPI для взаимодействия с LCD-дисплеем

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 13 представлена диаграмма всех перечисленных режимов работы интерфейса SPI.

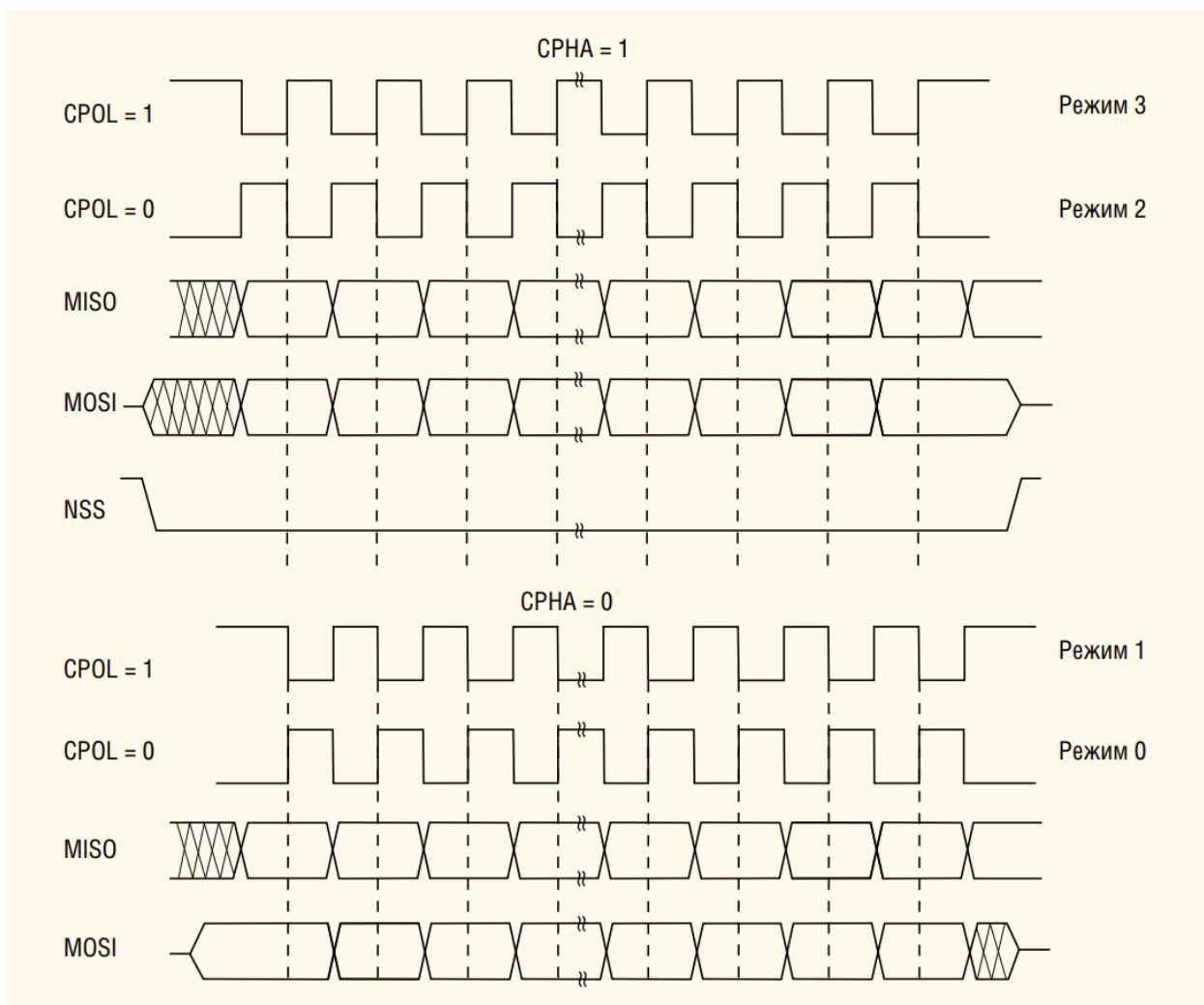


Рисунок 13 – Диаграмма режимов работы интерфейса SPI

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 сокетом SD-карты. Настройка для соединения с LCD-дисплеем показана на рисунке 14.

SPI1 Mode and Configuration

Mode

Mode Transmit Only Master

Hardware NSS Signal Disable

Configuration

Reset Configuration

✔ DMA Settings

✔ GPIO Settings

✔ User Constants

✔ NVIC Settings

✔ Parameter Settings

Configure the below parameters :

🔍

⏪
⏩
i

▼ Basic Parameters

Frame Format
Motorola

Data Size
8 Bits

First Bit
MSB First

▼ Clock Parameters

Prescaler (for Baud ...
2

Baud Rate
4.0 MBits/s

Clock Polarity (CPO..
Low

Clock Phase (CPHA)
1 Edge

▼ Advanced Parameters

CRC Calculation
Disabled

NSS Signal Type
Software

Рисунок 14 – Настройка SPI для взаимодействия с дисплеем

Таким образом, SPI1 работает только в режиме отправки, размер посылки – 8бит, прескейлер – 2, активным уровнем сигнала синхронизации является нулевой потенциал, а запись данных может производиться по фронту импульса данного синхросигнала.

Рассмотрим внутреннюю архитектуру SPI микроконтроллера STM32, которая представлена на рисунке 15.

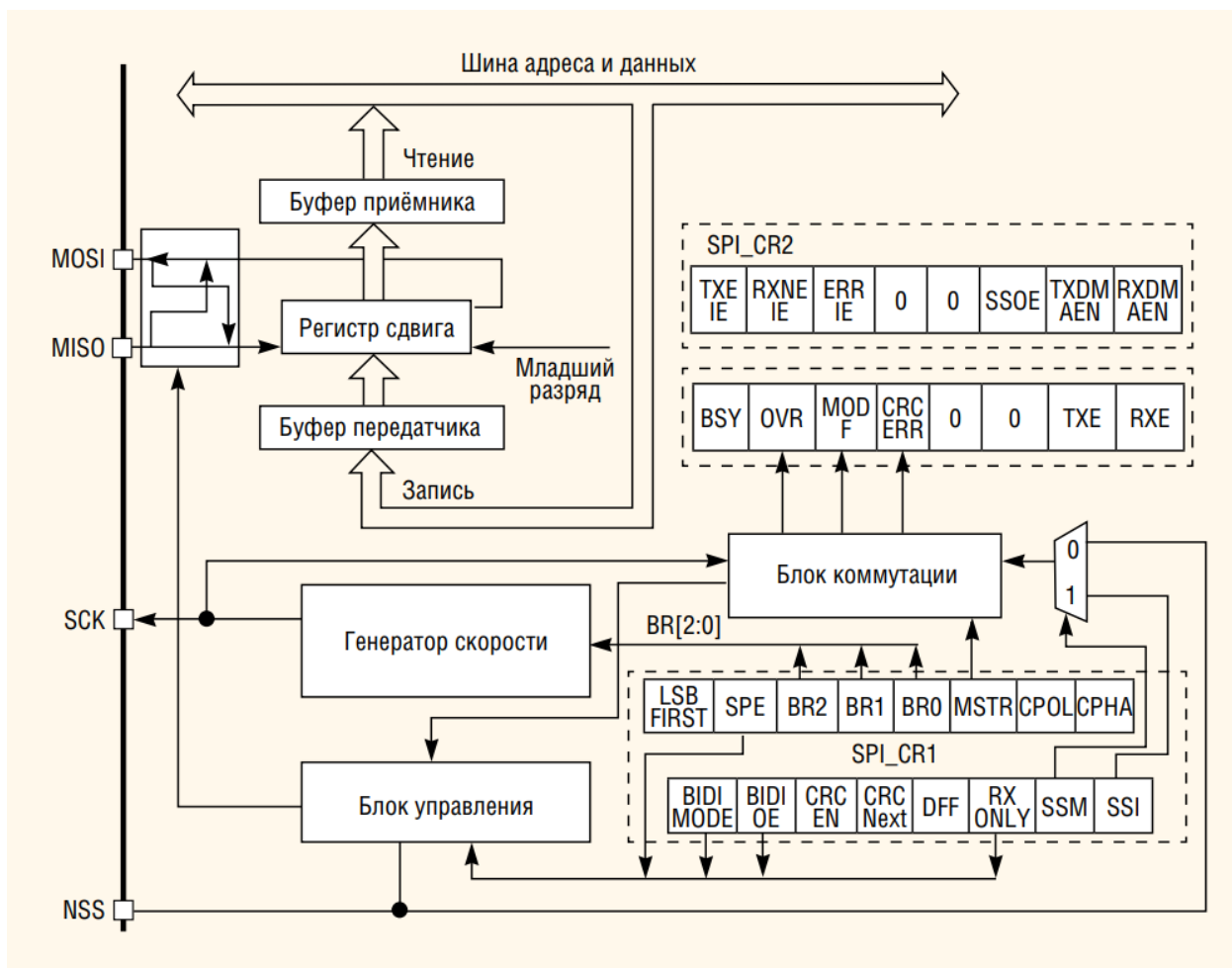


Рисунок 15 – Архитектура SPI МК семейства STM32

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 $24 \text{ МГц}/2=12 \text{ Мбод}$ до $24 \text{ МГц}/8=3 \text{ Мбод}$.

Для работы с интерфейсом SPI в микроконтроллере STM32 имеются специальные регистры. Формат этих регистров с названием входящих в них разрядов представлен на рисунке 16.

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	DFE	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		Резерв	0	0	0	0
0 × 08	SPI_SR	Резерв																							BSY	OVR	MODE	CRCERR		Резерв	TXE	RXNF	
	Исх. значение																								0	0	0	0	Резерв		1	0	Резерв
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	0	0	0
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 16 – 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 17.

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 17 – 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. BR [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 18.

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
								r/w	r/w	r/w			r/w	r/w	r/w

Рисунок 18 – 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 19.

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 19 – 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. – зарезервированы;

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 данными 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.

Рассмотрим

1.2.6 Настройка SPI для взаимодействия с сокетом SD-карты.

Теперь рассмотрим настройку SPI для чтения SD-карты. Был выбран сокет BOB-12941[12], а для подключения был использован SPI2, его конфигурация показана на рисунке 20.

SPI2 Mode and Configuration

Mode

Mode Receive Only Master

Hardware NSS Signal Disable

Configuration

Reset Configuration

✔ DMA Settings

✔ GPIO Settings

✔ User Constants

✔ NVIC Settings

✔ Parameter Settings

Configure the below parameters :

🔍

⏪
⏩
i

▼ Basic Parameters

Frame Format

Motorola

Data Size

8 Bits

First Bit

MSB First

▼ Clock Parameters

Prescaler (for Baud ...

2

Baud Rate

4.0 MBits/s

Clock Polarity (CPO..

Low

Clock Phase (CPHA)

1 Edge

▼ Advanced Parameters

CRC Calculation

Disabled

NSS Signal Type

Software

Рисунок 20 – Конфигурация SPI2

Таким образом, SPI1 работает только в режиме приема, размер послылки – 8бит, прескейлер – 2, активным уровнем сигнала синхронизации является нулевой потенциал, а запись данных может производится по фронту импульса данного синхросигнала.

Был выбран

1.2.7 Использование таймера для генерации звукового сигнала

Микроконтроллеры STM32F103xx имеют в своём составе множество таймеров 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.

В данной системе используется только один из каналов одного таймера общего назначения. Он используется для генерации ШИМ-сигнала для динамика в случае, если эталонная и вычисленная контрольные суммы не совпали. Используется первый канал второго таймера. Его конфигурация представлена на рисунке 21.

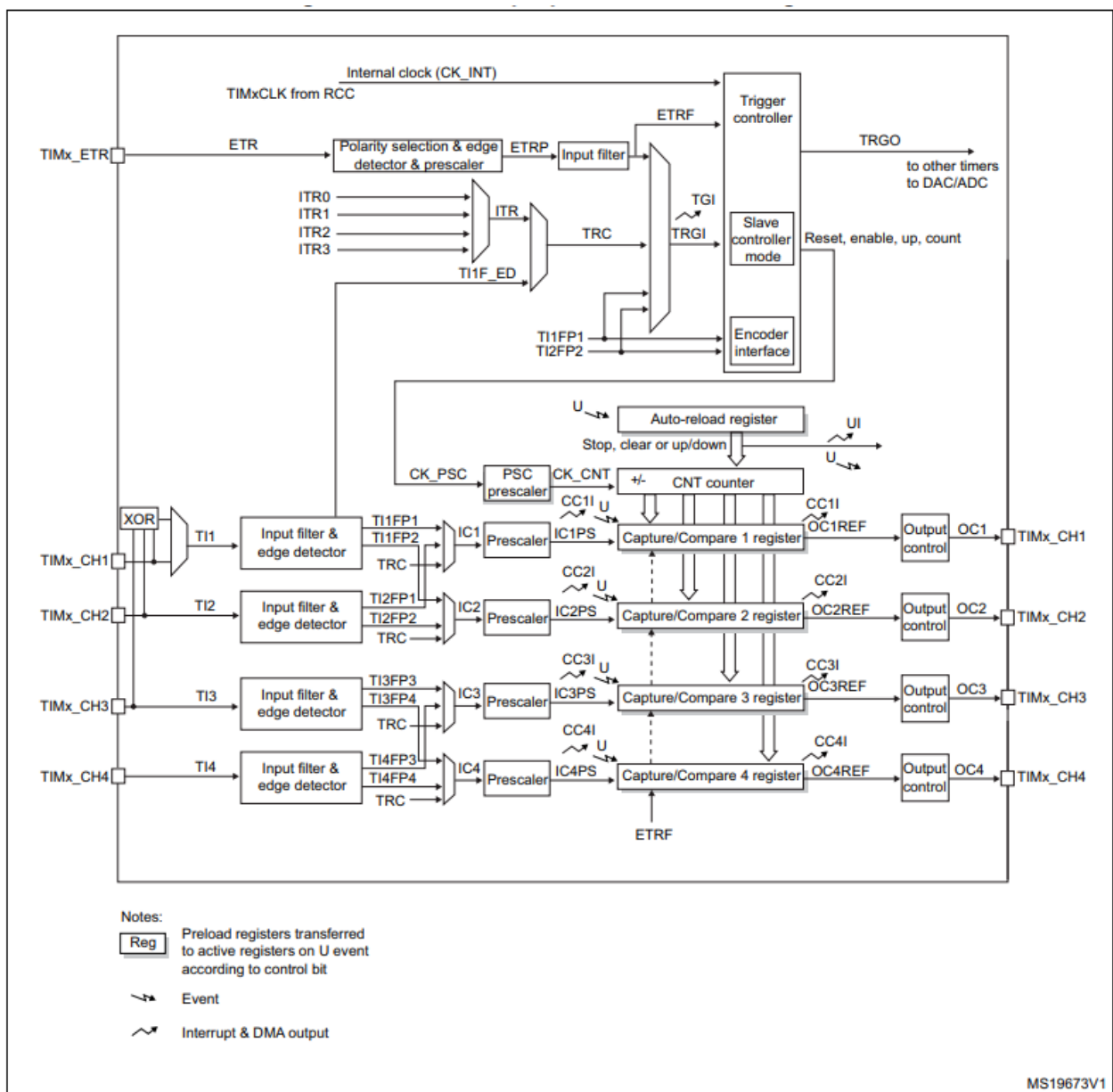


Рисунок 22 – Структурная схема таймера

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 127.

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 23.

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 23 – 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.

Далее рассмотрим TIM2_CCMR1(Compare and capture mode register), который представлен на рисунке 24. Каналы могут быть использованы в режимы захвата(input) и в режиме сравнения(output). При этом одни биты могут иметь разный смысл в зависимости от режима. Для режима ШИМ необходим режим сравнения, поэтому нужно смотреть на верхний ряд битов.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
IC2F[3:0]				IC2PSC[1:0]				IC1F[3:0]			IC1PSC[1:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 24 – Регистр TIM2_CR2

OC2CE: Output compare 2 clear enable

OC2M[2:0]: Output compare 2 mode

OC2PE: Output compare 2 preload enable

OC2FE: Output compare 2 fast enable

CC2S[1:0]: Capture/Compare 2 selection

OC1CE: Output compare 1 clear enable

OC1M: Output compare 1 mode. Эти биты определяют поведение выходного сигнала. Для режима ШИМ существует два варианта – 110 (PWM Mode 1) и 111 (PWM Mode 2). Был выбран PWM Mode 1, который выдает на пин единицу при возрастающем счетчике и 0 при убывающем (PWM Mode 2 действует наоборот).

OC1PE: Output compare 1 preload enable. 0 – предзагрузка отключена; 1 – предзагрузка включена. Для режима ШИМ предзагрузка должна быть включена.

OC1FE: Output compare 1 fast enable

CC1S: Capture/Compare 1 selection

Последний используемый регистр TIM2_CCR1 (Compare and Capture register) представлен на рисунке 25.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro

Рисунок 25 – регистр TIM2_CCR1

Для режима ШИМ была выбрана частота 500Гц, так как это стандартная частота звукового генератора в Proteus 8. Выше перечисленные настройки как раз были сделаны, чтобы ее получить. Частота работы микроконтроллера – 8МГц. $f = \text{AutoReload} * \text{Prescaler} * \text{FreqPWM}$, тогда для того, чтобы получить FreqPWM примерно равную 500Гц, можно выставить AutoReload и Prescaler как 127. $8000000 / 127 / 127 \approx 500\text{Гц}$. При этом используется коэффициент заполнения 50% - Для этого возьмем для Pulse примерно половину Prescaler – 63. Именно это значение будет храниться в TIM2_CCR1.

1.2.8 Построение функциональной схемы

На основе всех вышеописанных сведений была спроектирована функциональная схема разрабатываемой системы, показанная на рисунке 26[6, 7].

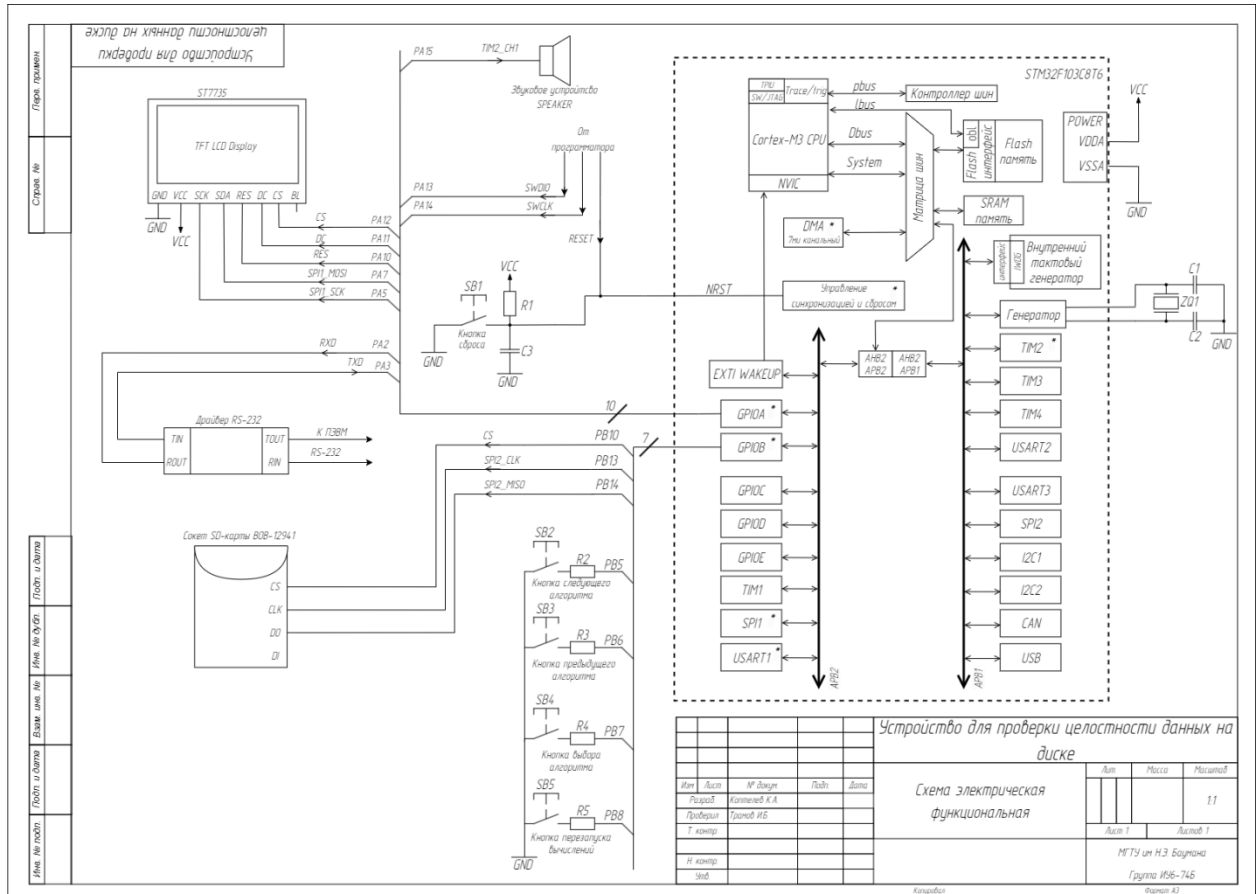


Рисунок 26 – Функциональная схема устройства для проверки целостности данных на диске

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 PB13 и PB14, 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 одним 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 7.

Таблица 7 – Потребляемая мощность

Микросхема	Ток потребления, мА	Потребляемая мощность, мВт	Количество устройств	Суммарная потребляемая мощность, мВт
STM32F103C8T6	150	495	1	495
MAX232	8	26,4	1	26,4
ВОВ-12941	500	1650	1	1650
ST7735	40	132	1	132
CF-100	10	-	6	9

$$P_{\text{суммарная}} = P_{\text{STM32F103C8T6}} + P_{\text{MAX232}} + P_{\text{ВОВ-12941}} + P_{\text{ST7735}} + P_{\text{CF-100}} = 495 + 26,4 + 1650 + 132 + 9 = 2312,4 \text{ мВт}$$

Суммарная потребляемая мощность системы равна 2312,4 мВт = 2,3 Вт.

1.3.3 Построение принципиальной схемы

На основе всех вышеописанных сведений была спроектирована принципиальная схема разрабатываемой системы, показанная на рисунке 27[6, 7].

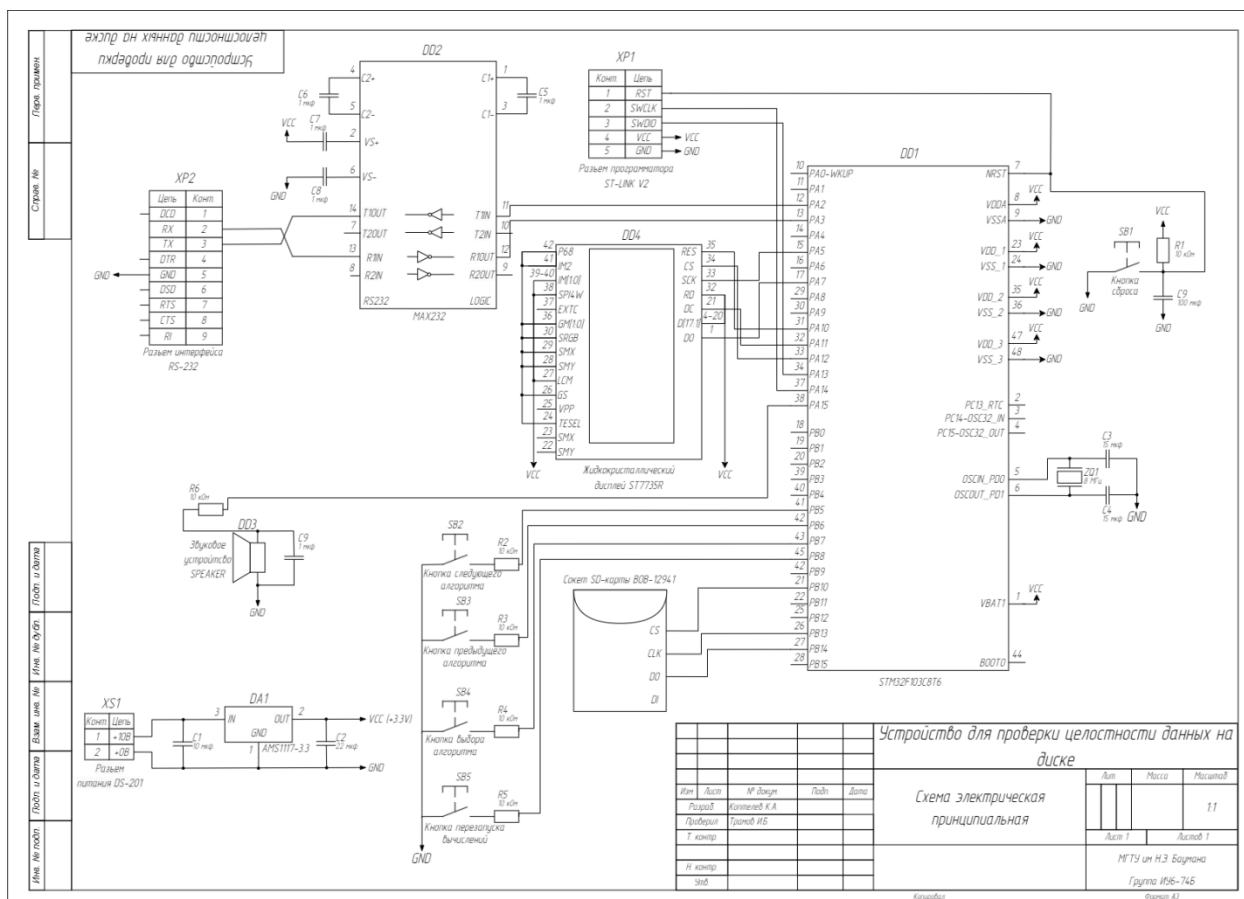


Рисунок 27 – Принципиальная схема

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, модулей FATFS и CRC, таймера `TIM2` и инициализация дисплея. Затем выделяется память под структуру `state_info` – автомата состояний, верхнеуровневой сущности в программе. Подается запрос на смену состояния в самое первое состояние – выбор алгоритма. Затем в бесконечном цикле обрабатывается запрос на смену состояния, затем обрабатывается ввод с ПЭВМ, и наконец производится действие, соответствующее текущему состоянию автомата. Верхнеуровневая схема алгоритма представлена на рисунке 28.

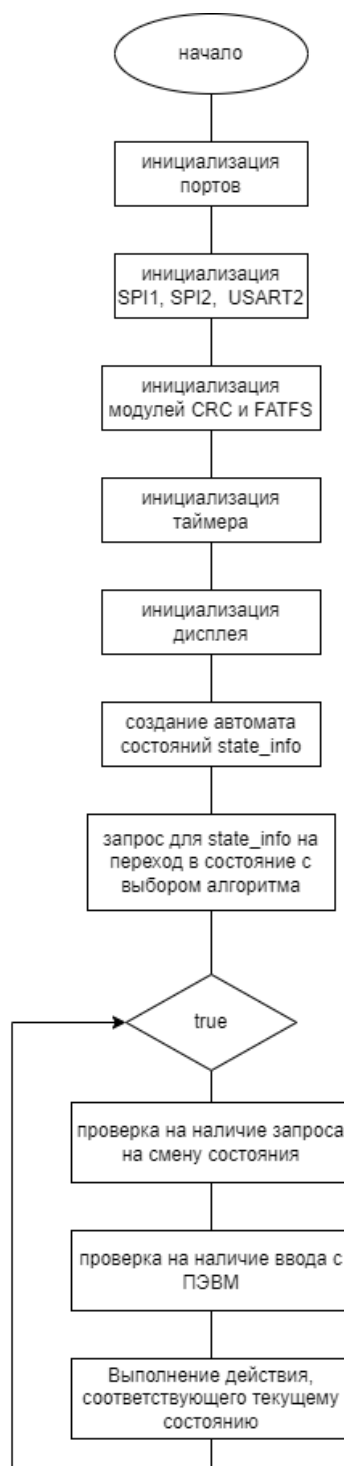


Рисунок 28 – Функция main

1.4.2 Детализация и пояснение основных функций

Рассмотрим функции из основного цикла выполнения программы. Схема алгоритма этих функций представлена на рисунке 29.

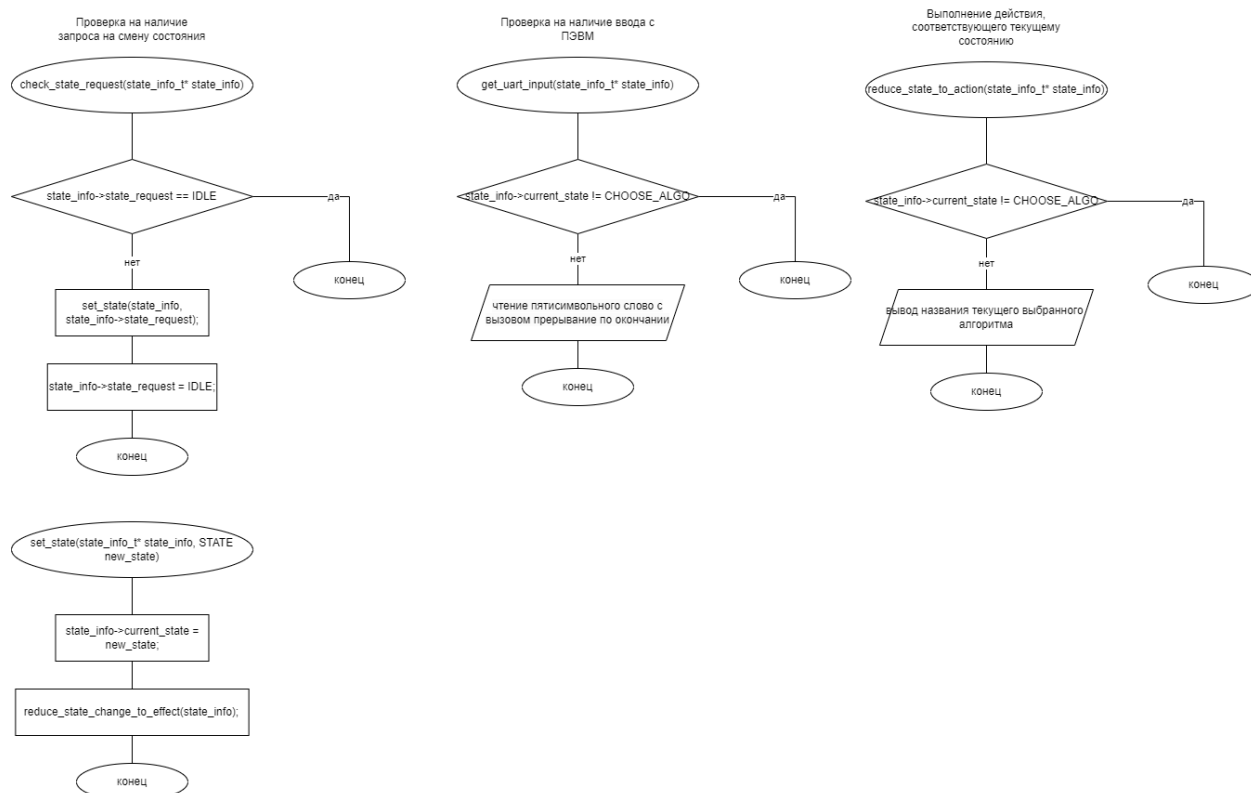


Рисунок 29 – Схема алгоритма функций в основном цикле

Основная логика программы находится в так называемых «эффектах» - одноразовых действиях, не связанными с текущим состоянием автомата, однако способными перевести автомат в другое состояние и «событиях» - действиях, постоянно происходящих во время определенного состояния. Ради эффектов и была создана механика запроса для автомата на переход в следующее состояние, ведь в контексте одного сравнения контрольных сумм контрольная сумма выбирается, вводится и вычисляется только один раз. Запрос на смену состояния работает следующим образом – если запрос не соответствует состоянию по умолчанию, то производится смена состояния на состояние из запроса, при этом смена сопровождается эффектом, а затем запрос на смену состояния выставляется в состояние по умолчанию. В основном эффекты вызываются в прерываниях. Рассмотрим алгоритм работы прерываний на рисунке 30.

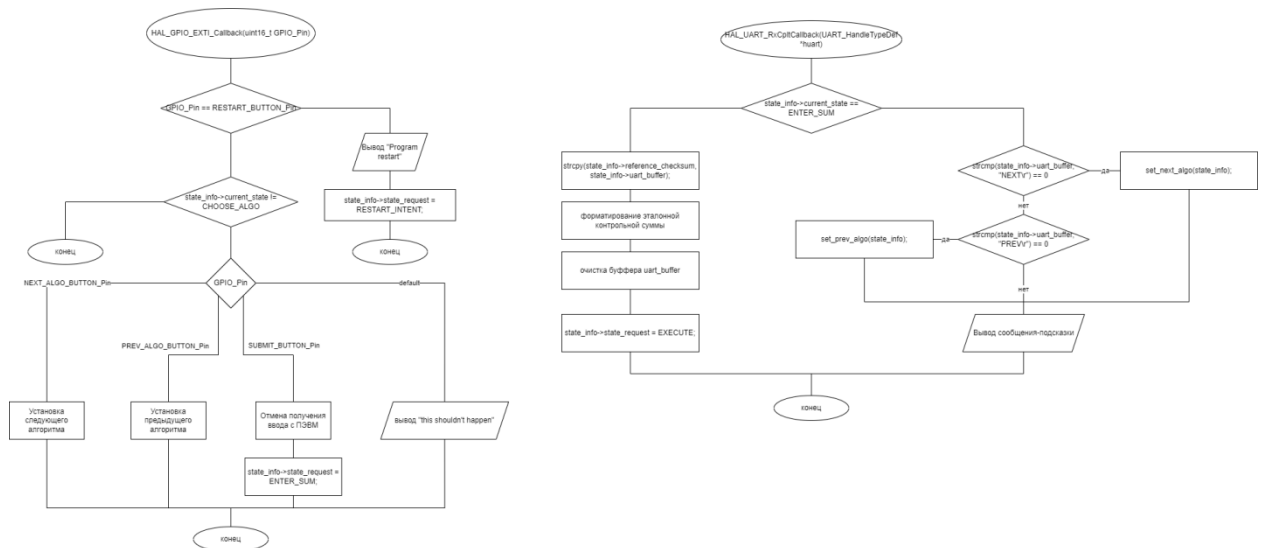


Рисунок 30 – Схема алгоритма работы прерываний

Наконец, рассмотрим само выполнение эффектов. Схема алгоритма работы эффектов представлена на рисунке 31.

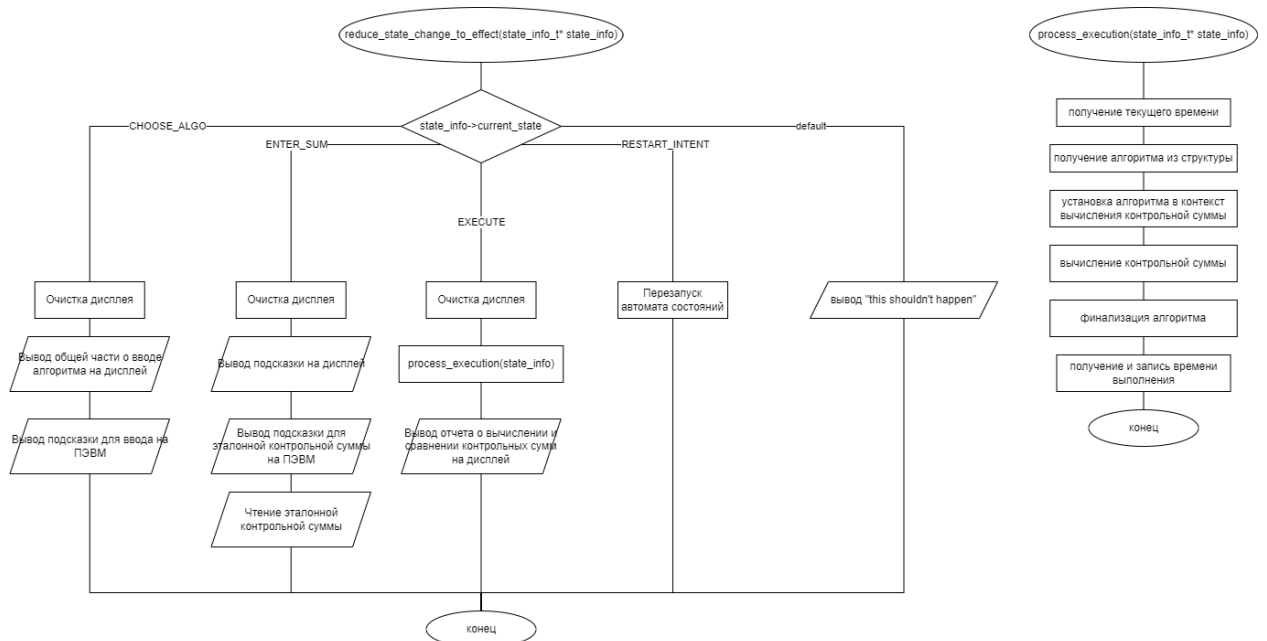


Рисунок 31 – Схема алгоритма работы эффектов

2 Технологическая часть

Для реализации работы устройства для проверки целостности данных на диске 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 и взаимодействовать с кнопками.

2.2 Симуляция работы системы

Для имитации реальных условий была использована программа Proteus. Схема системы в незапущенном состоянии изображена на рисунке 32.

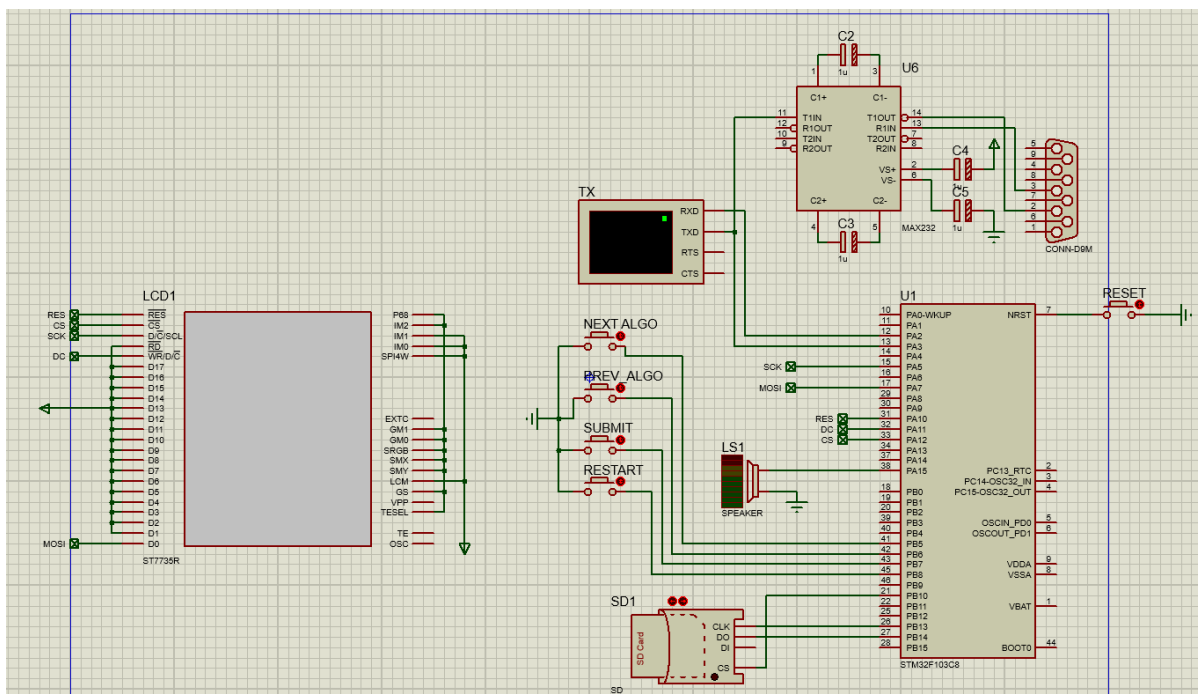


Рисунок 32 – Система в незапущенном состоянии

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. Первое состояние после запуска – состояние выбора контрольной суммы – показано на рисунке 33.

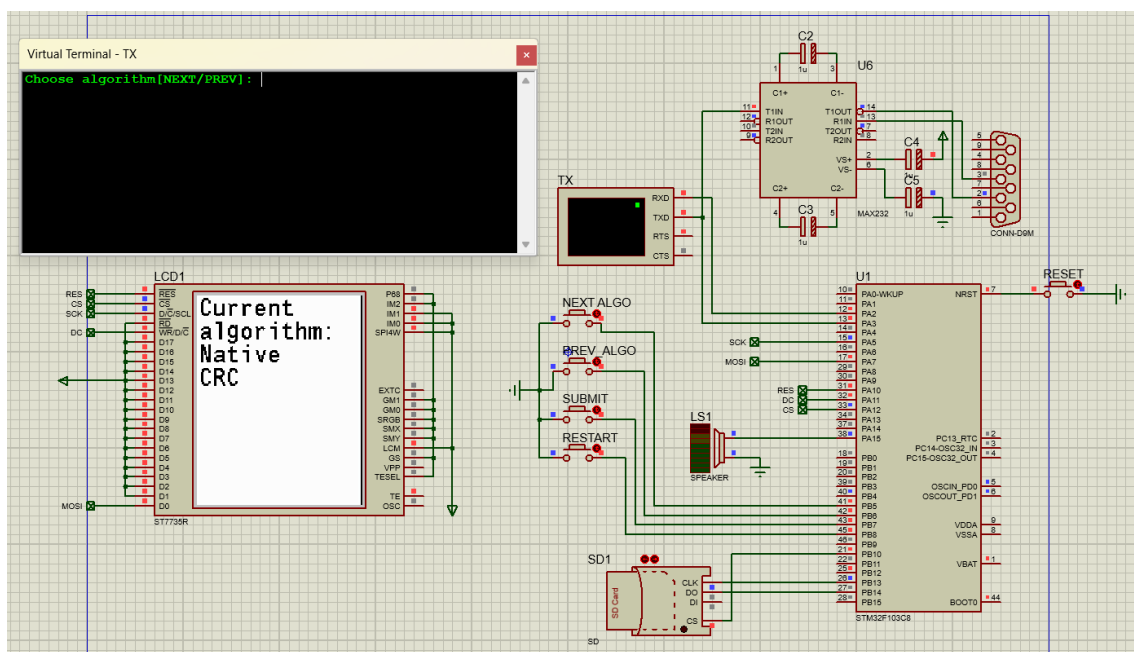


Рисунок 33 – Первое состояние(выбор алгоритма)

В первом состоянии в Virtual Terminal можно вводить «NEXT» или «PREV» для выбора предыдущего или следующего алгоритма соответственно. Пример использования показан на рисунке 34.

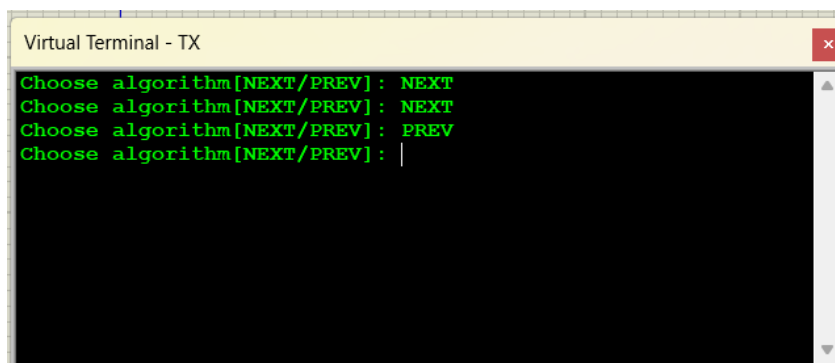


Рисунок 34 – Ввод чисел

Далее в зависимости от выбранного алгоритма вводится 8-значная или 32-значная эталонная контрольная сумма через виртуальный терминал. Это отображено на рисунке 35.

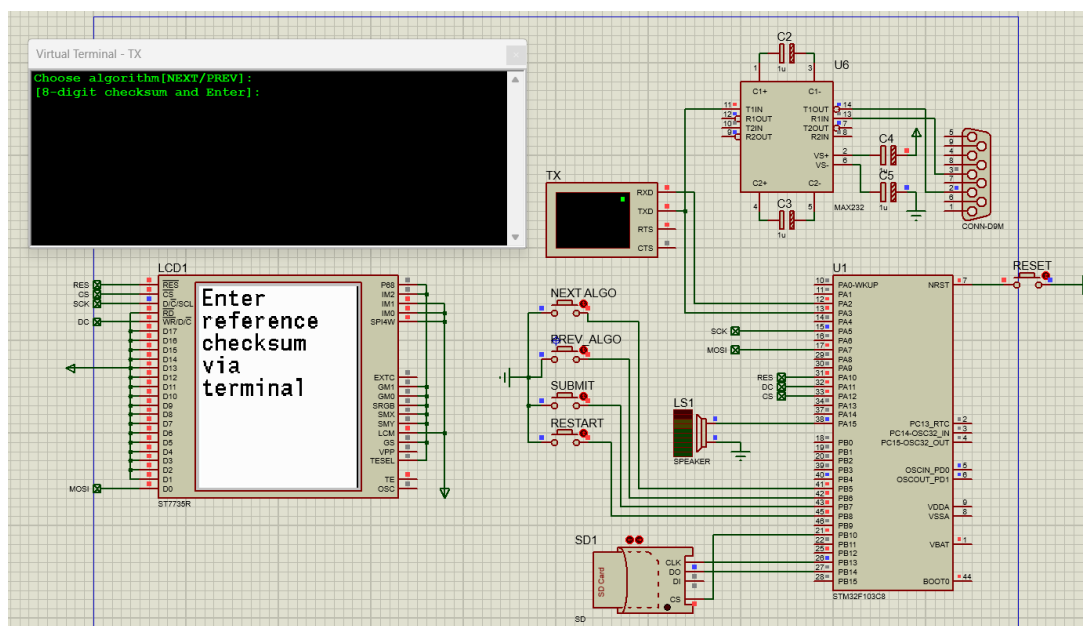


Рисунок 35 – Ввод эталонной контрольной суммы

После ввода эталонной контрольной суммы вычисляется реальная контрольная сумма и на основе сравнения выдается отчет и в случае несовпадения звуковой сигнал. На рисунке 36 представлен неверный ввод и соответственно вывод звукового сигнала на SPEAKER.

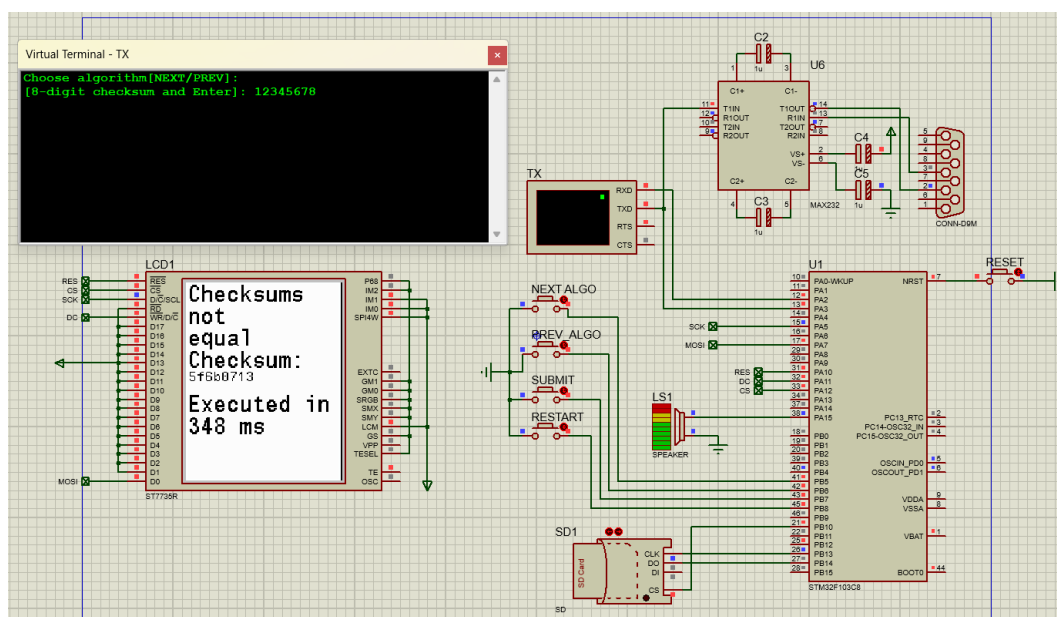


Рисунок 36 – Вывод отчета и звукового сигнала

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 выполняться 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.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы был создан проект – устройство для проверки целостности данных на диске с возможностью выбора алгоритма вычисления контрольной суммы. 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](https://ru.wikipedia.org/wiki/Микроконтроллер#Известные_семейства) (дата обращения: 13.09.2023)
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: 13.09.2023)
4. Документация на драйвер MAX232 [Электронный ресурс]. – URL: <https://www.alldatasheet.com/datasheet-pdf/pdf/73745/MAXIM/MAX7219.html> (дата обращения 27.10.2023).
5. Документация на ЖК-дисплей ST7735 [Электронный ресурс]. – URL: <https://www.displayfuture.com/Display/datasheet/controller/ST7735.pdf> (дата обращения 27.10.2023).
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 27.10.2023)

11. Способы программирования stm32 [Электронный ресурс]. – URL: <https://portal.tpu.ru/SHARED/t/TORGAEV/academic/Tab4/Posobie3.pdf> (дата обращения 27.10.2023)

12. Документация BOB-12941 [Электронный ресурс]. – URL: https://botland.store/index.php?controller=attachment&id_attachment=393 (дата обращения 27.10.2023)

Приложение А

Текст программы

Заголовочные файлы

Algorithms/algorithms.h

```
#ifndef INC_ALGORITHMS_ALGORITHMS_H_
#define INC_ALGORITHMS_ALGORITHMS_H_

#include "Algorithms/crc8.h"
#include "Algorithms/md5.h"
#include "utils.h"

#define MAX_CRC_LEN 32

extern CRC_HandleTypeDef hcrc;

typedef enum ALGORITHM {
    MD5,
    CRC8,
    HAL_CRC
} ALGORITHM;

typedef struct algorithm_ctx {
    ALGORITHM algorithm;
    MD5Context md5ctx;

    uint32_t current_value;
    char result[MAX_CRC_LEN + 1];
} algorithm_ctx_t;

algorithm_ctx_t* new_algorithm_context();
void set_algorithm(algorithm_ctx_t* ctx, ALGORITHM algo);
void algorithm_init(algorithm_ctx_t* ctx, char* buffer);
void algorithm_update(algorithm_ctx_t* ctx, char* buffer);
void algorithm_finalize(algorithm_ctx_t* ctx);

#endif /* INC_ALGORITHMS_ALGORITHMS_H_ */
```

Algorithms/crc8.h

```
#ifndef INC_ALGORITHMS_ALGORITHMS_H_
#define INC_ALGORITHMS_ALGORITHMS_H_

#include "Algorithms/crc8.h"
#include "Algorithms/md5.h"
#include "utils.h"

#define MAX_CRC_LEN 32

extern CRC_HandleTypeDef hcrc;

typedef enum ALGORITHM {
    MD5,
    CRC8,
    HAL_CRC
} ALGORITHM;
```

```

typedef struct algorithm_ctx {
    ALGORITHM algorithm;
    MD5Context md5ctx;

    uint32_t current_value;
    char result[MAX_CRC_LEN + 1];
} algorithm_ctx_t;

algorithm_ctx_t* new_algorithm_context();
void set_algorithm(algorithm_ctx_t* ctx, ALGORITHM algo);
void algorithm_init(algorithm_ctx_t* ctx, char* buffer);
void algorithm_update(algorithm_ctx_t* ctx, char* buffer);
void algorithm_finalize(algorithm_ctx_t* ctx);

#endif /* INC_ALGORITHMS_ALGORITHMS_H */

```

Algorithms/md5.h

```

#ifndef MD5_H
#define MD5_H

#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <stdlib.h>

typedef struct{
    uint64_t size;           // Size of input in bytes
    uint32_t buffer[4];      // Current accumulation of hash
    uint8_t input[64];       // Input to be used in the next step
    uint8_t digest[16];      // Result of algorithm
}MD5Context;

void md5Init(MD5Context *ctx);
void md5Update(MD5Context *ctx, uint8_t *input, size_t input_len);
void md5Finalize(MD5Context *ctx);
void md5Step(uint32_t *buffer, uint32_t *input);

#endif

```

SD-Card/fatfs-sd.h

```

#ifndef __FATFS_SD_H
#define __FATFS_SD_H

#include "fatfs.h"

/* Definitions for MMC/SDC command */
#define CMD0 (0x40+0) /* GO_IDLE_STATE */
#define CMD1 (0x40+1) /* SEND_OP_COND */
#define CMD8 (0x40+8) /* SEND_IF_COND */
#define CMD9 (0x40+9) /* SEND_CSD */
#define CMD10 (0x40+10) /* SEND_CID */
#define CMD12 (0x40+12) /* STOP_TRANSMISSION */
#define CMD16 (0x40+16) /* SET_BLOCKLEN */
#define CMD17 (0x40+17) /* READ_SINGLE_BLOCK */
#define CMD18 (0x40+18) /* READ_MULTIPLE_BLOCK */
#define CMD23 (0x40+23) /* SET_BLOCK_COUNT */
#define CMD24 (0x40+24) /* WRITE_BLOCK */
#define CMD25 (0x40+25) /* WRITE_MULTIPLE_BLOCK */
#define CMD41 (0x40+41) /* SEND_OP_COND (ACMD) */
#define CMD55 (0x40+55) /* APP_CMD */

```

```

#define CMD58      (0x40+58)          /* READ_OCR */

/* MMC card type flags (MMC_GET_TYPE) */
#define CT_MMC      0x01              /* MMC ver 3 */
#define CT_SD1      0x02              /* SD ver 1 */
#define CT_SD2      0x04              /* SD ver 2 */
#define CT_SDC      0x06              /* SD */
#define CT_BLOCK    0x08              /* Block addressing */

/* Functions */
DSTATUS SD_disk_initialize (BYTE pdrv);
DSTATUS SD_disk_status (BYTE pdrv);
DRESULT SD_disk_read (BYTE pdrv, BYTE* buff, DWORD sector, UINT count);
DRESULT SD_disk_write (BYTE pdrv, const BYTE* buff, DWORD sector, UINT
count);
DRESULT SD_disk_ioctl (BYTE pdrv, BYTE cmd, void* buff);

#define SPI_TIMEOUT 100

#endif

```

SD-Card/sd_card_interaction.h

```

#ifndef INC_SD_CARD_INTERACTION_H_
#define INC_SD_CARD_INTERACTION_H_

#include "fatfs_sd.h"
#include "fatfs.h"
#include "utils.h"
#include "Algorithms/algorithms.h"

extern CRC_HandleTypeDef hcrc;

typedef struct sd_card_t {
    FATFS fs; // file system
    FIL file; // file
    FILINFO file_info; // file info

    algorithm_ctx_t* algorithm_ctx;
    bool algorithm_initialized;
} sd_card_t;

sd_card_t* new_sd_card();
void free_sd_card(sd_card_t* sd_card);
void mount_sd_card(sd_card_t* sd_card);
void reset_calculation(sd_card_t* sd_card);
FRESULT calculate_checksum(sd_card_t* sd_card);

#endif /* INC_SD_CARD_INTERACTION_H_ */

```

ST7735/fonts.h

```

#ifndef FONTS_H_
#define FONTS_H_

#include <stdint.h>

typedef struct {
    const uint8_t width;
    uint8_t height;

```



```

        const uint16_t *data;
    } FontDef;

extern FontDef Font_7x10;
extern FontDef Font_11x18;

#endif /* FONTS_H */

```

ST7735/st7735_cfg.h

```

#ifndef ST7735_CFG_H_
#define ST7735_CFG_H_

#include "main.h"
#define ST7735_SPI_PORT hspi1

#define ST7735S_1_8_DEFAULT_ORIENTATION // WaveShare ST7735S-based 1.8"
display, default orientation

//Port and pin connected signal 'RES' (reset) ST7735 display
#ifndef ST7735_RES_Pin
#define ST7735_RES_Pin GPIO_PIN_10
#endif
#ifndef ST7735_RES_GPIO_Port
#define ST7735_RES_GPIO_Port GPIOA
#endif
//Port and pin connected signal 'DC' (data or command) ST7735 display
#ifndef ST7735_DC_Pin
#define ST7735_DC_Pin GPIO_PIN_11
#endif
#ifndef ST7735_DC_GPIO_Port
#define ST7735_DC_GPIO_Port GPIOA
#endif
//Port and pin connected signal 'CS' (chip select) ST7735 display
#ifndef ST7735_CS_Pin
#define ST7735_CS_Pin GPIO_PIN_12
#endif
#ifndef ST7735_CS_GPIO_Port
#define ST7735_CS_GPIO_Port GPIOA
#endif

#endif /* ST7735_CFG_H_ */

```

ST7735/ST7735.h

```

#ifndef ST7735_H_
#define ST7735_H_

#include "fonts.h"
#include "st7735_cfg.h"
#include <stdbool.h>
#include <string.h>

extern SPI_HandleTypeDef ST7735_SPI_PORT;

#define ST7735_MADCTL_MY 0x80
#define ST7735_MADCTL_MX 0x40
#define ST7735_MADCTL_MV 0x20
#define ST7735_MADCTL_RGB 0x00
#define ST7735_MADCTL_BGR 0x08

```

```

// WaveShare ST7735S-based 1.8" display, default orientation
#ifndef ST7735S_1_8_DEFAULT_ORIENTATION
#define ST7735S_1_8_DEFAULT_ORIENTATION 1
#define ST7735_WIDTH 128
#define ST7735_HEIGHT 160
#define ST7735_XSTART 2
#define ST7735_YSTART 1
#define ST7735_DATA_ROTATION ST7735_MADCTL_RGB
#endif //ST7735S_1_8_DEFAULT_ORIENTATION

#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

// 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

void ST7735_Init();
void ST7735_DrawPixel(uint16_t x, uint16_t y, uint16_t color);
void ST7735_DrawString(uint16_t x, uint16_t y, const char* str, FontDef font,
uint16_t color, uint16_t bgcolor);
void ST7735_FillRectangle(uint16_t x, uint16_t y, uint16_t w, uint16_t h,
uint16_t color);
void ST7735_FillScreen(uint16_t color);

#endif /* ST7735_H */

```

State/choosing_state.h

```

#ifndef INC_STATE_CHOOSING_STATE_H
#define INC_STATE_CHOOSING_STATE_H

#include "State/state_info_t.h"
#include "Algorithms/algorithms.h"

#include "ST7735/fonts.h"
#include "ST7735/ST7735.h"

#define ALGORITHMS_COUNT 3
#define SHIFT_WORD_SIZE 5
#define CHOOSE_ALGO_MSG "Choose algorithm[NEXT/PREV]: "

void set_next_algo(state_info_t* state_info);
void set_prev_algo(state_info_t* state_info);
void write_algorithm_message(state_info_t* state_info);
void write_algorithm_name(state_info_t* state_info);
void read_algorithm_shift(state_info_t* state_info);
void get_uart_input(state_info_t* state_info);

#endif /* INC_STATE_CHOOSING_STATE_H */

```

State/entering_state.h

```

#ifndef INC_STATE_ENTERING_STATE_H
#define INC_STATE_ENTERING_STATE_H

#include <State/state_info_t.h>

void read_checksum(state_info_t* state_info);
void print_checksum_helper(state_info_t* state_info);
void format_reference_checksum(state_info_t* state_info);
void write_enter_sum_message(state_info_t* state_info);

#endif /* INC_STATE_ENTERING_STATE_H */

```

State/execution_state.h

```

#ifndef INC_STATE_EXECUTION_STATE_H
#define INC_STATE_EXECUTION_STATE_H

#include "State/state_info_t.h"
#include "Algorithms/algorithms.h"

```

```

extern TIM_HandleTypeDef htim2;

void process_execution(state_info_t* state_info);
char* extract_result(state_info_t* state_info);
void write_checksum_report(state_info_t* state_info);
ALGORITHM get_algo_from_index(int algorithm_index);
void make_error_sound();

#endif /* INC_STATE_EXECUTION_STATE_H */

```

State/state_info_t.h

```

#ifndef INC_STATE_STATE_INFO_T_H
#define INC_STATE_STATE_INFO_T_H

#include "ST7735/fonts.h"
#include "ST7735/ST7735.h"

#include "utils.h"
#include "SD-Card/sd_card_interaction.h"

typedef enum STATE {
    IDLE,
    CHOOSE_ALGO,
    ENTER_SUM,
    EXECUTE,
    RESTART_INTENT
} STATE;

typedef struct state_info_t {
    char output_buffer[DEFAULT_BUFFER_SIZE];

    STATE current_state;
    STATE state_request;
    int algorithm_index;
    uint32_t deltatime;

    int uart_write_ptr;
    char uart_buffer[DEFAULT_BUFFER_SIZE];
    char reference_checksum[MAX_CRC_LEN + 1];

    sd_card_t* sd_card;
} state_info_t;

state_info_t* new_state_info();
void free_state_info(state_info_t* state_info);

#endif /* INC_STATE_STATE_INFO_T_H */

```

State/state.h

```

#ifndef INC_STATE_H
#define INC_STATE_H

#include "State/state_info_t.h"
#include "State/choosing_state.h"
#include "State/entering_state.h"
#include "State/execution_state.h"

void check_state_request(state_info_t* state_info);
void set_state(state_info_t* state_info, STATE new_state);

```

```

void reset_state(state_info_t* state_info);
void reduce_state_to_action(state_info_t* state_info);
void reduce_state_change_to_effect(state_info_t* state_info);

#endif /* INC_STATE_H */

```

utils.h

```

#ifndef INC_UTILS_H_
#define INC_UTILS_H_

#define TERMINAL_LINE_WIDTH 11
#define TERMINAL_LINE_HEIGHT 18
#define DEFAULT_BUFFER_SIZE 100

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <stdint.h>
#include <ctype.h>
#include <stdarg.h>

#include "main.h"

extern UART_HandleTypeDef huart2;

void format_buffer(char* buffer, size_t line_width);
void clear_buffer(char* buffer, size_t size);
void char_array_to_uint32_array(char* src, uint32_t* dest, int len);
void uint32_array_to_char_array(uint32_t* src, char* dest, int len);
void to_lower(char* string);
void print_uart_message(char* format, ...);

#endif /* INC_UTILS_H_ */

```

Исходные файлы

Algorithms/algorithms.c

```

#include "Algorithms/algorithms.h"

algorithm_ctx_t* new_algorithm_context() {
    algorithm_ctx_t* ctx = calloc(1, sizeof(algorithm_ctx_t));
    ctx->current_value = 0;
    clear_buffer(ctx->result, MAX_CRC_LEN + 1);
    return ctx;
}

void set_algorithm(algorithm_ctx_t* ctx, ALGORITHM algo) {
    ctx->algorithm = algo;
}

void format_md5_hash(char* dest, uint8_t* src) {
    for (int i = 0; i < 16; ++i) {
        sprintf(dest + i * 2, "%02x", src[i]);
    }
}

void algorithm_init(algorithm_ctx_t* ctx, char* buffer) {
    switch (ctx->algorithm) {
        case MD5:

```

```

        md5Init(&ctx->md5ctx);
        md5Update(&ctx->md5ctx, (uint8_t*)buffer, DEFAULT_BUFFER_SIZE);
        break;
    case CRC8:
        ctx->current_value = crc8(0, NULL, 0);
        ctx->current_value = crc8(ctx->current_value, (unsigned
char*)buffer, DEFAULT_BUFFER_SIZE);
        break;
    case HAL_CRC:
        HAL_CRC_Calculate(&hcrc, (uint32_t*)buffer, DEFAULT_BUFFER_SIZE);
        break;
    }
}

void algorithm_update(algorithm_ctx_t* ctx, char* buffer) {
    switch (ctx->algorithm) {
    case MD5:
        md5Update(&ctx->md5ctx, (uint8_t*)buffer, DEFAULT_BUFFER_SIZE);
        break;
    case CRC8:
        ctx->current_value = crc8(ctx->current_value, (unsigned
char*)buffer, DEFAULT_BUFFER_SIZE);
        break;
    case HAL_CRC:
        ctx->current_value = HAL_CRC_Accumulate(&hcrc, (uint32_t*)buffer,
DEFAULT_BUFFER_SIZE);
        break;
    }
}

void algorithm_finalize(algorithm_ctx_t* ctx) {
    switch (ctx->algorithm) {
    case MD5:
        md5Finalize(&ctx->md5ctx);
        format_md5_hash(ctx->result, ctx->md5ctx.digest);
        break;
    case CRC8:
        sprintf(ctx->result, "%08lx", ctx->current_value);
        break;
    case HAL_CRC:
        sprintf(ctx->result, "%08lx", ctx->current_value);
        break;
    }
}

```

Algorithms/crc8.c

```

#include "Algorithms/crc8.h"

uint32_t crc8(uint32_t crc, unsigned char const *data, size_t len)
{
    if (data == NULL) {
        return 0;
    }
    crc &= 0xff;
    unsigned char const *end = data + len;
    while (data < end) {
        crc = crc8_table[crc ^ *data++];
    }
    return crc;
}

```

Algorithms/md5.c

```
/*
 * Derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm
 * and modified slightly to be functionally identical but condensed into
 * control structures.
 */

#include "Algorithms/md5.h"

/*
 * Constants defined by the MD5 algorithm
 */
#define A 0x67452301
#define B 0xefcdab89
#define C 0x98badcfe
#define D 0x10325476

static uint32_t S[] = {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12,
17, 22,
                        5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9,
14, 20,
                        4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11,
16, 23,
                        6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10,
15, 21};

static uint32_t K[] = {0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdcee,
0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,
0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,
0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,
0xf61e2562, 0xc40b340, 0x265e5a51, 0xe9b6c7aa,
0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,
0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,
0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,
0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,
0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,
0x289b7ec6, 0xea127fa, 0xd4ef3085, 0x04881d05,
0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,
0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,
0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,
0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,
0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391};

/*
 * Padding used to make the size (in bits) of the input congruent to 448 mod
512
 */
static uint8_t PADDING[] = {0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

/*
 * Bit-manipulation functions defined by the MD5 algorithm
 */
#define F(X, Y, Z) ((X & Y) | (~X & Z))
```

```

#define G(X, Y, Z) ((X & Z) | (Y & ~Z))
#define H(X, Y, Z) (X ^ Y ^ Z)
#define I(X, Y, Z) (Y ^ (X | ~Z))

/*
 * Rotates a 32-bit word left by n bits
 */
uint32_t rotateLeft(uint32_t x, uint32_t n){
    return (x << n) | (x >> (32 - n));
}

/*
 * Initialize a context
 */
void md5Init(MD5Context *ctx){
    ctx->size = (uint64_t)0;

    ctx->buffer[0] = (uint32_t)A;
    ctx->buffer[1] = (uint32_t)B;
    ctx->buffer[2] = (uint32_t)C;
    ctx->buffer[3] = (uint32_t)D;
}

/*
 * Add some amount of input to the context
 *
 * If the input fills out a block of 512 bits, apply the algorithm (md5Step)
 * and save the result in the buffer. Also updates the overall size.
 */
void md5Update(MD5Context *ctx, uint8_t *input_buffer, size_t input_len){
    uint32_t input[16];
    unsigned int offset = ctx->size % 64;
    ctx->size += (uint64_t)input_len;

    // Copy each byte in input_buffer into the next space in our context
    input
    for(unsigned int i = 0; i < input_len; ++i){
        ctx->input[offset++] = (uint8_t)*(input_buffer + i);

        // If we've filled our context input, copy it into our local array
        input

        // then reset the offset to 0 and fill in a new buffer.
        // Every time we fill out a chunk, we run it through the algorithm
        // to enable some back and forth between cpu and i/o
        if(offset % 64 == 0){
            for(unsigned int j = 0; j < 16; ++j){
                // Convert to little-endian
                // The local variable `input` our 512-bit chunk separated
                into 32-bit words
                // we can use in calculations
                input[j] = (uint32_t)(ctx->input[(j * 4) + 3]) << 24 |
                    (uint32_t)(ctx->input[(j * 4) + 2]) << 16 |
                    (uint32_t)(ctx->input[(j * 4) + 1]) << 8 |
                    (uint32_t)(ctx->input[(j * 4)]);
            }
            md5Step(ctx->buffer, input);
            offset = 0;
        }
    }
}

```



```

/*
 * Pad the current input to get to 448 bytes, append the size in bits to the
 * very end,
 * and save the result of the final iteration into digest.
 */
void md5Finalize(MD5Context *ctx){
    uint32_t input[16];
    unsigned int offset = ctx->size % 64;
    unsigned int padding_length = offset < 56 ? 56 - offset : (56 + 64) -
offset;

    // Fill in the padding and undo the changes to size that resulted from
the update
    md5Update(ctx, PADDING, padding_length);
    ctx->size -= (uint64_t)padding_length;

    // Do a final update (internal to this function)
    // Last two 32-bit words are the two halves of the size (converted from
bytes to bits)
    for(unsigned int j = 0; j < 14; ++j){
        input[j] = (uint32_t)(ctx->input[(j * 4) + 3]) << 24 |
            (uint32_t)(ctx->input[(j * 4) + 2]) << 16 |
            (uint32_t)(ctx->input[(j * 4) + 1]) << 8 |
            (uint32_t)(ctx->input[(j * 4)]);
    }
    input[14] = (uint32_t)(ctx->size * 8);
    input[15] = (uint32_t)((ctx->size * 8) >> 32);

    md5Step(ctx->buffer, input);

    // Move the result into digest (convert from little-endian)
    for(unsigned int i = 0; i < 4; ++i){
        ctx->digest[(i * 4) + 0] = (uint8_t)((ctx->buffer[i] & 0x000000FF));
        ctx->digest[(i * 4) + 1] = (uint8_t)((ctx->buffer[i] & 0x0000FF00) >>
8);
        ctx->digest[(i * 4) + 2] = (uint8_t)((ctx->buffer[i] & 0x00FF0000) >>
16);
        ctx->digest[(i * 4) + 3] = (uint8_t)((ctx->buffer[i] & 0xFF000000) >>
24);
    }
}

/*
 * Step on 512 bits of input with the main MD5 algorithm.
 */
void md5Step(uint32_t *buffer, uint32_t *input){
    uint32_t AA = buffer[0];
    uint32_t BB = buffer[1];
    uint32_t CC = buffer[2];
    uint32_t DD = buffer[3];

    uint32_t E;

    unsigned int j;

    for(unsigned int i = 0; i < 64; ++i){
        switch(i / 16){
            case 0:
                E = F(BB, CC, DD);
                j = i;

```

```

        break;
    case 1:
        E = G(BB, CC, DD);
        j = ((i * 5) + 1) % 16;
        break;
    case 2:
        E = H(BB, CC, DD);
        j = ((i * 3) + 5) % 16;
        break;
    default:
        E = I(BB, CC, DD);
        j = (i * 7) % 16;
        break;
}

uint32_t temp = DD;
DD = CC;
CC = BB;
BB = BB + rotateLeft(AA + E + K[i] + input[j], S[i]);
AA = temp;
}

buffer[0] += AA;
buffer[1] += BB;
buffer[2] += CC;
buffer[3] += DD;
}

```

SD-Card/fatfs-sd.c

```

#include <stdbool.h>

#include "stm32f1xx_hal.h"
#include "diskio.h"
#include "SD-Card/fatfs_sd.h"

extern SPI_HandleTypeDef hspi2;
#define HSPI_SDCARD &hspi2
#define SD_CS_PORT GPIOB
#define SD_CS_PIN GPIO_PIN_10

extern volatile uint16_t Timer1, Timer2; /* 1ms
Timer Counter */

static volatile DSTATUS Stat = STA_NOINIT; /* Disk Status */
static uint8_t CardType; /* Type 0:MMC, 1:SDC, 2:Block
addressing */
static uint8_t PowerFlag = 0; /* Power flag */

/*****
 * SPI functions
 *****/

/* slave select */
static void SELECT(void)
{
    HAL_GPIO_WritePin(SD_CS_PORT, SD_CS_PIN, GPIO_PIN_RESET);
    HAL_Delay(1);
}

/* slave deselect */
static void DESELECT(void)

```

```

{
    HAL_GPIO_WritePin(SD_CS_PORT, SD_CS_PIN, GPIO_PIN_SET);
    HAL_Delay(1);
}

/* SPI transmit a byte */
static void SPI_TxByte(uint8_t data)
{
    while(!__HAL_SPI_GET_FLAG(HSPI_SDCARD, SPI_FLAG_TXE));
    HAL_SPI_Transmit(HSPI_SDCARD, &data, 1, SPI_TIMEOUT);
}

/* SPI transmit buffer */
static void SPI_TxBuffer(uint8_t *buffer, uint16_t len)
{
    while(!__HAL_SPI_GET_FLAG(HSPI_SDCARD, SPI_FLAG_TXE));
    HAL_SPI_Transmit(HSPI_SDCARD, buffer, len, SPI_TIMEOUT);
}

/* SPI receive a byte */
static uint8_t SPI_RxByte(void)
{
    uint8_t dummy, data;
    dummy = 0xFF;

    while(!__HAL_SPI_GET_FLAG(HSPI_SDCARD, SPI_FLAG_TXE));
    HAL_SPI_TransmitReceive(HSPI_SDCARD, &dummy, &data, 1, SPI_TIMEOUT);

    return data;
}

/* SPI receive a byte via pointer */
static void SPI_RxBytePtr(uint8_t *buff)
{
    *buff = SPI_RxByte();
}

/*****
 * SD functions
 *****/

/* wait SD ready */
static uint8_t SD_ReadyWait(void)
{
    uint8_t res;

    /* timeout 500ms */
    Timer2 = 500;

    /* if SD goes ready, receives 0xFF */
    do {
        res = SPI_RxByte();
    } while ((res != 0xFF) && Timer2);

    return res;
}

/* power on */
static void SD_PowerOn(void)
{
    uint8_t args[6];

```

```

uint32_t cnt = 0xFFFF;

/* transmit bytes to wake up */
DESELECT();
for(int i = 0; i < 10; i++)
{
    SPI_TxByte(0xFF);
}

/* slave select */
SELECT();

/* make idle state */
args[0] = CMD0;          /* CMD0:GO_IDLE_STATE */
args[1] = 0;
args[2] = 0;
args[3] = 0;
args[4] = 0;
args[5] = 0x95;          /* CRC */

SPI_TxBuffer(args, sizeof(args));

/* wait response */
while ((SPI_RxByte() != 0x01) && cnt)
{
    cnt--;
}

DESELECT();
SPI_TxByte(0xFF);

PowerFlag = 1;
}

/* power off */
static void SD_PowerOff(void)
{
    PowerFlag = 0;
}

/* check power flag */
static uint8_t SD_CheckPower(void)
{
    return PowerFlag;
}

/* receive data block */
static bool SD_RxDataBlock(BYTE *buff, UINT len)
{
    uint8_t token;

    /* timeout 200ms */
    Timer1 = 200;

    /* loop until receive a response or timeout */
    do {
        token = SPI_RxByte();
    } while((token == 0xFF) && Timer1);

    /* invalid response */
    if(token != 0xFE) return false;

```

```

    /* receive data */
    do {
        SPI_RxBytePtr(buff++);
    } while(len--);

    /* discard CRC */
    SPI_RxByte();
    SPI_RxByte();

    return true;
}

/* transmit data block */
#if _USE_WRITE == 1
static bool SD_TxDataBlock(const uint8_t *buff, BYTE token)
{
    uint8_t resp = 0;
    uint8_t i = 0;

    /* wait SD ready */
    if (SD_ReadyWait() != 0xFF) return false;

    /* transmit token */
    SPI_TxByte(token);

    /* if it's not STOP token, transmit data */
    if (token != 0xFD)
    {
        SPI_TxBuffer((uint8_t*)buff, 512);

        /* discard CRC */
        SPI_RxByte();
        SPI_RxByte();

        /* receive response */
        while (i <= 64)
        {
            resp = SPI_RxByte();

            /* transmit 0x05 accepted */
            if ((resp & 0x1F) == 0x05) break;
            i++;
        }

        /* recv buffer clear */
        while (SPI_RxByte() == 0);
    }

    /* transmit 0x05 accepted */
    if ((resp & 0x1F) == 0x05) return true;

    return false;
}
#endif /* _USE_WRITE */

/* transmit command */
static BYTE SD_SendCmd(BYTE cmd, uint32_t arg)
{
    uint8_t crc, res;

```

```

/* wait SD ready */
if (SD_ReadyWait() != 0xFF) return 0xFF;

/* transmit command */
SPI_TxByte(cmd); /* Command */
SPI_TxByte((uint8_t)(arg >> 24)); /* Argument[31..24] */
SPI_TxByte((uint8_t)(arg >> 16)); /* Argument[23..16] */
SPI_TxByte((uint8_t)(arg >> 8)); /* Argument[15..8] */
SPI_TxByte((uint8_t)arg); /* Argument[7..0] */

/* prepare CRC */
if(cmd == CMD0) crc = 0x95; /* CRC for CMD0(0) */
else if(cmd == CMD8) crc = 0x87; /* CRC for CMD8(0x1AA) */
else crc = 1;

/* transmit CRC */
SPI_TxByte(crc);

/* Skip a stuff byte when STOP_TRANSMISSION */
if (cmd == CMD12) SPI_RxByte();

/* receive response */
uint8_t n = 10;
do {
    res = SPI_RxByte();
} while ((res & 0x80) && --n);

return res;
}

/*****
 * user_diskio.c functions
 *****/

/* initialize SD */
DSTATUS SD_disk_initialize(BYTE drv)
{
    uint8_t n, type, ocr[4];

    /* single drive, drv should be 0 */
    if(drv) return STA_NOINIT;

    /* no disk */
    if(Stat & STA_NODISK) return Stat;

    /* power on */
    SD_PowerOn();

    /* slave select */
    SELECT();

    /* check disk type */
    type = 0;

    /* send GO_IDLE_STATE command */
    if (SD_SendCmd(CMD0, 0) == 1)
    {
        /* timeout 1 sec */
        Timer1 = 1000;

        /* SDC V2+ accept CMD8 command, http://elm-

```

```

chan.org/docs/mmc/mmc_e.html */
    if (SD_SendCmd(CMD8, 0x1AA) == 1)
    {
        /* operation condition register */
        for (n = 0; n < 4; n++)
        {
            ocr[n] = SPI_RxByte();
        }

        /* voltage range 2.7-3.6V */
        if (ocr[2] == 0x01 && ocr[3] == 0xAA)
        {
            /* ACMD41 with HCS bit */
            do {
                if (SD_SendCmd(CMD55, 0) <= 1 &&
SD_SendCmd(CMD41, 1UL << 30) == 0) break;
            } while (Timer1);

            /* READ_OCR */
            if (Timer1 && SD_SendCmd(CMD58, 0) == 0)
            {
                /* Check CCS bit */
                for (n = 0; n < 4; n++)
                {
                    ocr[n] = SPI_RxByte();
                }

                /* SDv2 (HC or SC) */
                type = (ocr[0] & 0x40) ? CT_SD2 | CT_BLOCK :
CT_SD2;
            }
        }
        else
        {
            /* SDC V1 or MMC */
            type = (SD_SendCmd(CMD55, 0) <= 1 && SD_SendCmd(CMD41, 0)
<= 1) ? CT_SD1 : CT_MMC;

            do
            {
                if (type == CT_SD1)
                {
                    if (SD_SendCmd(CMD55, 0) <= 1 &&
SD_SendCmd(CMD41, 0) == 0) break; /* ACMD41 */
                }
                else
                {
                    if (SD_SendCmd(CMD1, 0) == 0) break; /* CMD1 */
                }
            } while (Timer1);

            /* SET_BLOCKLEN */
            if (!Timer1 || SD_SendCmd(CMD16, 512) != 0) type = 0;
        }

        CardType = type;

        /* Idle */

```

```

DESELECT();
SPI_RxByte();

/* Clear STA_NOINIT */
if (type)
{
    Stat &= ~STA_NOINIT;
}
else
{
    /* Initialization failed */
    SD_PowerOff();
}

return Stat;
}

/* return disk status */
DSTATUS SD_disk_status(BYTE drv)
{
    if (drv) return STA_NOINIT;
    return Stat;
}

/* read sector */
DRESULT SD_disk_read(BYTE pdrv, BYTE* buff, DWORD sector, UINT count)
{
    /* pdrv should be 0 */
    if (pdrv || !count) return RES_PARERR;

    /* no disk */
    if (Stat & STA_NOINIT) return RES_NOTRDY;

    /* convert to byte address */
    if (!(CardType & CT_SD2)) sector *= 512;

    SELECT();

    if (count == 1)
    {
        /* READ_SINGLE_BLOCK */
        if ((SD_SendCmd(CMD17, sector) == 0) && SD_RxDataBlock(buff,
512)) count = 0;
    }
    else
    {
        /* READ_MULTIPLE_BLOCK */
        if (SD_SendCmd(CMD18, sector) == 0)
        {
            do {
                if (!SD_RxDataBlock(buff, 512)) break;
                buff += 512;
            } while (--count);
            /* STOP TRANSMISSION */
            SD_SendCmd(CMD12, 0);
        }
    }

    /* Idle */
    DESELECT();
    SPI_RxByte();

```



```

        return count ? RES_ERROR : RES_OK;
    }

    /* write sector */
    #if _USE_WRITE == 1
    DRESULT SD_disk_write(BYTE pdrv, const BYTE* buff, DWORD sector, UINT count)
    {
        /* pdrv should be 0 */
        if (pdrv || !count) return RES_PARERR;

        /* no disk */
        if (Stat & STA_NOINIT) return RES_NOTRDY;

        /* write protection */
        if (Stat & STA_PROTECT) return RES_WRPRT;

        /* convert to byte address */
        if (!(CardType & CT_SD2)) sector *= 512;

        SELECT();

        if (count == 1)
        {
            /* WRITE_BLOCK */
            if ((SD_SendCmd(CMD24, sector) == 0) && SD_TxDataBlock(buff,
0xFE))
                count = 0;
        }
        else
        {
            /* WRITE_MULTIPLE_BLOCK */
            if (CardType & CT_SD1)
            {
                SD_SendCmd(CMD55, 0);
                SD_SendCmd(CMD23, count); /* ACMD23 */
            }

            if (SD_SendCmd(CMD25, sector) == 0)
            {
                do {
                    if(!SD_TxDataBlock(buff, 0xFC)) break;
                    buff += 512;
                } while (--count);

                /* STOP_TRAN token */
                if(!SD_TxDataBlock(0, 0xFD))
                {
                    count = 1;
                }
            }
        }

        /* Idle */
        DESELECT();
        SPI_RxByte();

        return count ? RES_ERROR : RES_OK;
    }
#endif /* _USE_WRITE */

```

```

/* ioctl */
DRESULT SD_disk_ioctl(BYTE drv, BYTE ctrl, void *buff)
{
    DRESULT res;
    uint8_t n, csd[16], *ptr = buff;
    WORD csize;

    /* pdrv should be 0 */
    if (drv) return RES_PARERR;
    res = RES_ERROR;

    if (ctrl == CTRL_POWER)
    {
        switch (*ptr)
        {
            case 0:
                SD_PowerOff();          /* Power Off */
                res = RES_OK;
                break;
            case 1:
                SD_PowerOn();           /* Power On */
                res = RES_OK;
                break;
            case 2:
                *(ptr + 1) = SD_CheckPower();
                res = RES_OK;           /* Power Check */
                break;
            default:
                res = RES_PARERR;
        }
    }
    else
    {
        /* no disk */
        if (Stat & STA_NOINIT) return RES_NOTRDY;

        SELECT();

        switch (ctrl)
        {
            case GET_SECTOR_COUNT:
                /* SEND_CSD */
                if ((SD_SendCmd(CMD9, 0) == 0) && SD_RxDataBlock(csd, 16))
                {
                    if ((csd[0] >> 6) == 1)
                    {
                        /* SDC V2 */
                        csize = csd[9] + ((WORD) csd[8] << 8) + 1;
                        *(DWORD*) buff = (DWORD) csize << 10;
                    }
                    else
                    {
                        /* MMC or SDC V1 */
                        n = (csd[5] & 15) + ((csd[10] & 128) >> 7) +
((csd[9] & 3) << 1) + 2;
                        csize = (csd[8] >> 6) + ((WORD) csd[7] << 2) +
((WORD) (csd[6] & 3) << 10) + 1;
                        *(DWORD*) buff = (DWORD) csize << (n - 9);
                    }
                    res = RES_OK;
                }
            }
        }
    }
}

```

```

        break;
    case GET_SECTOR_SIZE:
        *(WORD*) buff = 512;
        res = RES_OK;
        break;
    case CTRL_SYNC:
        if (SD_ReadyWait() == 0xFF) res = RES_OK;
        break;
    case MMC_GET_CSD:
        /* SEND_CSD */
        if (SD_SendCmd(CMD9, 0) == 0 && SD_RxDataBlock(ptr, 16))
res = RES_OK;
        break;
    case MMC_GET_CID:
        /* SEND_CID */
        if (SD_SendCmd(CMD10, 0) == 0 && SD_RxDataBlock(ptr, 16))
res = RES_OK;
        break;
    case MMC_GET_OCR:
        /* READ_OCR */
        if (SD_SendCmd(CMD58, 0) == 0)
        {
            for (n = 0; n < 4; n++)
            {
                *ptr++ = SPI_RxByte();
            }
            res = RES_OK;
        }
    default:
        res = RES_PARERR;
    }

    DESELECT();
    SPI_RxByte();
}

return res;
}

```

SD-Card/sd_card_interaction.c

```

#include "SD-Card/sd_card_interaction.h"

sd_card_t* new_sd_card() {
    sd_card_t* sd_card = calloc(1, sizeof(sd_card_t));
    sd_card->algorithm_ctx = new_algorithm_context();
    sd_card->algorithm_initialized = false;
    return sd_card;
}

void free_sd_card(sd_card_t* sd_card) {
    free(sd_card->algorithm_ctx);
    f_mount(NULL, "", 1);
    free(sd_card);
}

void reset_calculation(sd_card_t* sd_card) {
    sd_card->algorithm_initialized = false;
}

void mount_sd_card(sd_card_t* sd_card) {
    FRESULT fres = f_mount(&sd_card->fs, "/", 1);
}

```

```

    if (fres != FR_OK) {
        print_uart_message("Error in mounting sd_card\r");
        return;
    }

    // print uart message("successful mount...\r");
}

FRESULT hash_next_file(
    char* filename,
    algorithm_ctx_t* ctx,
    void (*algorithm_function)(algorithm_ctx_t*, char*)
) {
    FIL file;
    FRESULT fres = f_open(&file, filename, FA_READ);
    if (fres != FR_OK) {
        print_uart_message("something went wrong with %s\r", filename);
        return fres;
    }

    UINT br;
    char buffer[DEFAULT_BUFFER_SIZE] = {0};
    while (true) {
        f_read(&file, buffer, DEFAULT_BUFFER_SIZE, &br);
        if (br == 0) {
            break;
        }
        // print_uart_message("%s\r", buffer);
        algorithm_function(ctx, buffer);
    }

    f_close(&file);
    return FR_OK;
}

FRESULT calculate_checksum(sd_card_t* sd_card) {
    static char path[DEFAULT_BUFFER_SIZE] = {0};
    DIR directory;

    FRESULT fres = f_opendir(&directory, path);
    if (fres != FR_OK) {
        print_uart_message("\rcannot open dir %s\r", path);
        return fres;
    }

    while (true) {
        fres = f_readdir(&directory, &sd_card->file_info);
        if (fres != FR_OK) {
            print_uart_message("error occurred!\r");
            break;
        }

        if (sd_card->file_info.fname[0] == 0) {
            break;
        }

        if (sd_card->file_info.fattrib & AM_DIR) {
            UINT i = strlen(path);
            sprintf(&path[i], "\\%s", sd_card->file_info.fname);
            fres = calculate_checksum(sd_card);
            if (fres != FR_OK) {

```

```

        print_uart_message("error occurred!");
        return FR_INT_ERR;
    }
    path[i] = 0;
} else { // is file
    char full_name[DEFAULT_BUFFER_SIZE] = {0};
    sprintf(full_name, "%s\\%s", path, sd_card-
>file_info.fname);
    if (sd_card->algorithm_initialized) {
        hash_next_file(full_name, sd_card->algorithm_ctx,
algorithm_update);
    } else {
        hash_next_file(full_name, sd_card->algorithm_ctx,
algorithm_init);
        sd_card->algorithm_initialized = true;
    }
}

f_closedir(&directory);
return FR_OK;
}

```

ST7735/fonts.c

```

/*
 * fonts.c
 *
 */

#include "ST7735/fonts.h"

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, // ~
};

static const uint16_t Font11x18 [] = {
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, //
sp
0x0000, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00,
0x0C00, 0x0C00, 0x0C00, 0x0000, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000, //
!
0x0000, 0x1B00, 0x1B00, 0x1B00, 0x1B00, 0x1B00, 0x1B00, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, //
"
0x0000, 0x1980, 0x1980, 0x1980, 0x1980, 0x1980, 0x7FC0, 0x7FC0, 0x1980, 0x3300,
0x7FC0, 0x7FC0, 0x3300, 0x3300, 0x3300, 0x3300, 0x0000, 0x0000, 0x0000, //
#
0x0000, 0x1E00, 0x3F00, 0x7580, 0x6580, 0x7400, 0x3C00, 0x1E00, 0x0700,
0x0580, 0x6580, 0x6580, 0x7580, 0x3F00, 0x1E00, 0x0400, 0x0400, 0x0000, //
$
0x0000, 0x7000, 0xD800, 0xD840, 0xD8C0, 0xD980, 0x7300, 0x0600, 0x0C00,
0x1B80, 0x36C0, 0x66C0, 0x46C0, 0x06C0, 0x0380, 0x0000, 0x0000, 0x0000, //
%
0x0000, 0x1E00, 0x3F00, 0x3300, 0x3300, 0x3300, 0x1E00, 0x0C00, 0x3CC0,

```



```

0x66C0, 0x6380, 0x6180, 0x6380, 0x3EC0, 0x1C80, 0x0000, 0x0000, 0x0000, //
&
0x0000, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, //
,
0x0080, 0x0100, 0x0300, 0x0600, 0x0600, 0x0400, 0x0C00, 0x0C00, 0x0C00,
0x0C00, 0x0C00, 0x0C00, 0x0400, 0x0600, 0x0600, 0x0300, 0x0100, 0x0080, //
(
0x2000, 0x1000, 0x1800, 0x0C00, 0x0C00, 0x0400, 0x0600, 0x0600, 0x0600,
0x0600, 0x0600, 0x0600, 0x0400, 0x0C00, 0x0C00, 0x1800, 0x1000, 0x2000, //
)
0x0000, 0x0C00, 0x2D00, 0x3F00, 0x1E00, 0x3300, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, //
*
0x0000, 0x0000, 0x0000, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0xFFC0, 0xFFC0,
0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, //
+
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0C00, 0x0C00, 0x0400, 0x0400, 0x0800, //
,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x1E00, 0x1E00, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, //
-
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000, //
.
0x0000, 0x0300, 0x0300, 0x0300, 0x0600, 0x0600, 0x0600, 0x0600, 0x0C00,
0x0C00, 0x0C00, 0x0C00, 0x1800, 0x1800, 0x1800, 0x0000, 0x0000, 0x0000, //
/
0x0000, 0x1E00, 0x3F00, 0x3300, 0x6180, 0x6180, 0x6180, 0x6D80, 0x6D80,
0x6180, 0x6180, 0x6180, 0x3300, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, //
0
0x0000, 0x0600, 0x0E00, 0x1E00, 0x3600, 0x2600, 0x0600, 0x0600, 0x0600,
0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0000, 0x0000, 0x0000, //
1
0x0000, 0x1E00, 0x3F00, 0x7380, 0x6180, 0x6180, 0x0180, 0x0300, 0x0600,
0x0C00, 0x1800, 0x3000, 0x6000, 0x7F80, 0x7F80, 0x0000, 0x0000, 0x0000, //
2
0x0000, 0x1C00, 0x3E00, 0x6300, 0x6300, 0x0300, 0x0E00, 0x0E00, 0x0300,
0x0180, 0x0180, 0x6180, 0x7380, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, //
3
0x0000, 0x0600, 0x0E00, 0x0E00, 0x1E00, 0x1E00, 0x1600, 0x3600, 0x3600,
0x6600, 0x7F80, 0x7F80, 0x0600, 0x0600, 0x0600, 0x0000, 0x0000, 0x0000, //
4
0x0000, 0x7F00, 0x7F00, 0x6000, 0x6000, 0x6000, 0x6E00, 0x7F00, 0x6380,
0x0180, 0x0180, 0x6180, 0x7380, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, //
5
0x0000, 0x1E00, 0x3F00, 0x3380, 0x6180, 0x6000, 0x6E00, 0x7F00, 0x7380,
0x6180, 0x6180, 0x6180, 0x3380, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, //
6
0x0000, 0x7F80, 0x7F80, 0x0180, 0x0300, 0x0300, 0x0600, 0x0600, 0x0C00,
0x0C00, 0x0C00, 0x0800, 0x1800, 0x1800, 0x1800, 0x0000, 0x0000, 0x0000, //
7
0x0000, 0x1E00, 0x3F00, 0x6380, 0x6180, 0x6180, 0x2100, 0x1E00, 0x3F00,
0x6180, 0x6180, 0x6180, 0x6180, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, //
8
0x0000, 0x1E00, 0x3F00, 0x7300, 0x6180, 0x6180, 0x6180, 0x7380, 0x3F80,
0x1D80, 0x0180, 0x6180, 0x7300, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, //
9
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0C00, 0x0C00, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000, //

```

:	0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000, 0x0000, 0x0C00, 0x0C00, 0x0400, 0x0400, 0x0800, //
;	0x0000, 0x0000, 0x0000, 0x0000, 0x0080, 0x0380, 0x0E00, 0x3800, 0x6000, 0x3800, 0x0E00, 0x0380, 0x0080, 0x0000, 0x0000, 0x0000, 0x0000, //
<	0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x7F80, 0x7F80, 0x0000, 0x0000, 0x7F80, 0x7F80, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, //
=	0x0000, 0x0000, 0x0000, 0x0000, 0x4000, 0x7000, 0x1C00, 0x0700, 0x0180, 0x0700, 0x1C00, 0x7000, 0x4000, 0x0000, 0x0000, 0x0000, 0x0000, //
>	0x0000, 0x1F00, 0x3F80, 0x71C0, 0x60C0, 0x00C0, 0x01C0, 0x0380, 0x0700, 0x0E00, 0x0C00, 0x0C00, 0x0000, 0x0C00, 0x0C00, 0x0000, 0x0000, //
?	0x0000, 0x1E00, 0x3F00, 0x3180, 0x7180, 0x6380, 0x6F80, 0x6D80, 0x6D80, 0x6F80, 0x6780, 0x6000, 0x3200, 0x3E00, 0x1C00, 0x0000, 0x0000, //
@	0x0000, 0x0E00, 0x0E00, 0x1B00, 0x1B00, 0x1B00, 0x1B00, 0x3180, 0x3180, 0x3F80, 0x3F80, 0x3180, 0x60C0, 0x60C0, 0x60C0, 0x0000, 0x0000, //
A	0x0000, 0x7C00, 0x7E00, 0x6300, 0x6300, 0x6300, 0x6300, 0x7E00, 0x7E00, 0x6300, 0x6180, 0x6180, 0x6380, 0x7F00, 0x7E00, 0x0000, 0x0000, //
B	0x0000, 0x1E00, 0x3F00, 0x3180, 0x6180, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6180, 0x3180, 0x3F00, 0x1E00, 0x0000, 0x0000, //
C	0x0000, 0x7C00, 0x7F00, 0x6300, 0x6380, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6300, 0x6300, 0x7E00, 0x7C00, 0x0000, 0x0000, //
D	0x0000, 0x7F80, 0x7F80, 0x6000, 0x6000, 0x6000, 0x6000, 0x7F00, 0x7F00, 0x6000, 0x6000, 0x6000, 0x6000, 0x7F80, 0x7F80, 0x0000, 0x0000, //
E	0x0000, 0x7F80, 0x7F80, 0x6000, 0x6000, 0x6000, 0x6000, 0x7F00, 0x7F00, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x0000, 0x0000, //
F	0x0000, 0x1E00, 0x3F00, 0x3180, 0x6180, 0x6000, 0x6000, 0x6000, 0x6380, 0x6380, 0x6180, 0x6180, 0x3180, 0x3F80, 0x1E00, 0x0000, 0x0000, //
G	0x0000, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x7F80, 0x7F80, 0x6180, 0x6180, 0x6180, 0x6180, 0x0000, 0x0000, 0x0000, //
H	0x0000, 0x3F00, 0x3F00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x3F00, 0x3F00, 0x0000, 0x0000, //
I	0x0000, 0x0180, 0x0180, 0x0180, 0x0180, 0x0180, 0x0180, 0x0180, 0x0180, 0x0180, 0x6180, 0x6180, 0x7380, 0x3F00, 0x1E00, 0x0000, 0x0000, //
J	0x0000, 0x60C0, 0x6180, 0x6300, 0x6600, 0x6600, 0x6C00, 0x7800, 0x7C00, 0x6600, 0x6600, 0x6300, 0x6180, 0x6180, 0x60C0, 0x0000, 0x0000, //
K	0x0000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x7F80, 0x7F80, 0x0000, 0x0000, //
L	0x0000, 0x71C0, 0x71C0, 0x7BC0, 0x7AC0, 0x6AC0, 0x6AC0, 0x6EC0, 0x64C0, 0x60C0, 0x60C0, 0x60C0, 0x60C0, 0x60C0, 0x0000, 0x0000, //
M	0x0000, 0x7180, 0x7180, 0x7980, 0x7980, 0x7980, 0x6D80, 0x6D80, 0x6D80, 0x6580, 0x6780, 0x6780, 0x6780, 0x6380, 0x6380, 0x0000, 0x0000, //
N	

O	0x0000, 0x1E00, 0x3F00, 0x3300, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x3300, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, //
P	0x0000, 0x7E00, 0x7F00, 0x6380, 0x6180, 0x6180, 0x6180, 0x6380, 0x7F00, 0x7E00, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x0000, 0x0000, 0x0000, //
Q	0x0000, 0x1E00, 0x3F00, 0x3300, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6580, 0x6780, 0x3300, 0x3F80, 0x1E40, 0x0000, 0x0000, 0x0000, //
R	0x0000, 0x7E00, 0x7F00, 0x6380, 0x6180, 0x6180, 0x6380, 0x7F00, 0x7E00, 0x6600, 0x6300, 0x6300, 0x6180, 0x6180, 0x60C0, 0x0000, 0x0000, 0x0000, //
S	0x0000, 0x0E00, 0x1F00, 0x3180, 0x3180, 0x3000, 0x3800, 0x1E00, 0x0700, 0x0380, 0x6180, 0x6180, 0x3180, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, //
T	0x0000, 0xFFC0, 0xFFC0, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000, //
U	0x0000, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x7380, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, //
V	0x0000, 0x60C0, 0x60C0, 0x60C0, 0x3180, 0x3180, 0x3180, 0x1B00, 0x1B00, 0x1B00, 0x1B00, 0x0E00, 0x0E00, 0x0E00, 0x0400, 0x0000, 0x0000, 0x0000, //
W	0x0000, 0xC0C0, 0xC0C0, 0xC0C0, 0xC0C0, 0xC0C0, 0xC0C0, 0x4C80, 0x4C80, 0x5E80, 0x5280, 0x5280, 0x7380, 0x6180, 0x6180, 0x0000, 0x0000, 0x0000, //
X	0x0000, 0xC0C0, 0x6080, 0x6180, 0x3300, 0x3B00, 0x1E00, 0x0C00, 0x0C00, 0x1E00, 0x1F00, 0x3B00, 0x7180, 0x6180, 0xC0C0, 0x0000, 0x0000, 0x0000, //
Y	0x0000, 0xC0C0, 0x6180, 0x6180, 0x3300, 0x3300, 0x1E00, 0x1E00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000, //
Z	0x0000, 0x3F80, 0x3F80, 0x0180, 0x0300, 0x0300, 0x0600, 0x0C00, 0x0C00, 0x1800, 0x1800, 0x3000, 0x6000, 0x7F80, 0x7F80, 0x0000, 0x0000, 0x0000, //
[0x0F00, 0x0F00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0F00, 0x0F00, //
\ */	0x0000, 0x1800, 0x1800, 0x1800, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0600, 0x0600, 0x0600, 0x0600, 0x0300, 0x0300, 0x0300, 0x0000, 0x0000, 0x0000, /*
]	0x1E00, 0x1E00, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x1E00, 0x1E00, //
^	0x0000, 0x0C00, 0x0C00, 0x1E00, 0x1200, 0x3300, 0x3300, 0x6180, 0x6180, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, //
~	0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xFFE0, 0x0000, //
`	0x0000, 0x3800, 0x1800, 0x0C00, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, //
a	0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1F00, 0x3F80, 0x6180, 0x0180, 0x1F80, 0x3F80, 0x6180, 0x6380, 0x7F80, 0x38C0, 0x0000, 0x0000, 0x0000, //
b	0x0000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6E00, 0x7F00, 0x7380, 0x6180, 0x6180, 0x6180, 0x6180, 0x7380, 0x7F00, 0x6E00, 0x0000, 0x0000, 0x0000, //
	0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1E00, 0x3F00, 0x7380, 0x6180,

	0x6000, 0x6000, 0x6180, 0x7380, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, //
c	0x0000, 0x0180, 0x0180, 0x0180, 0x0180, 0x1D80, 0x3F80, 0x7380, 0x6180, //
d	0x6180, 0x6180, 0x6180, 0x7380, 0x3F80, 0x1D80, 0x0000, 0x0000, 0x0000, //
e	0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1E00, 0x3F00, 0x7300, 0x6180, //
	0x7F80, 0x7F80, 0x6000, 0x7180, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, //
f	0x0000, 0x07C0, 0x0FC0, 0x0C00, 0x0C00, 0x7F80, 0x7F80, 0x0C00, 0x0C00, //
	0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000, //
g	0x0000, 0x0000, 0x0000, 0x0000, 0x1D80, 0x3F80, 0x7380, 0x6180, 0x6180, //
	0x6180, 0x6180, 0x7380, 0x3F80, 0x1D80, 0x0180, 0x6380, 0x7F00, 0x3E00, //
h	0x0000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6F00, 0x7F80, 0x7180, 0x6180, //
	0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x0000, 0x0000, 0x0000, //
i	0x0000, 0x0600, 0x0600, 0x0000, 0x0000, 0x3E00, 0x3E00, 0x0600, 0x0600, //
	0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0000, 0x0000, 0x0000, //
j	0x0600, 0x0600, 0x0000, 0x0000, 0x3E00, 0x3E00, 0x0600, 0x0600, 0x0600, //
	0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x4600, 0x7E00, 0x3C00, //
k	0x0000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6180, 0x6300, 0x6600, 0x6C00, //
	0x7C00, 0x7600, 0x6300, 0x6300, 0x6180, 0x60C0, 0x0000, 0x0000, 0x0000, //
l	0x0000, 0x3E00, 0x3E00, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, //
	0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0000, 0x0000, 0x0000, //
m	0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xDD80, 0xFFC0, 0xCEC0, 0xCCC0, //
	0xCCC0, 0xCCC0, 0xCCC0, 0xCCC0, 0xCCC0, 0xCCC0, 0x0000, 0x0000, 0x0000, //
n	0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x6F00, 0x7F80, 0x7180, 0x6180, //
	0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x0000, 0x0000, 0x0000, //
o	0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1E00, 0x3F00, 0x7380, 0x6180, //
	0x6180, 0x6180, 0x6180, 0x7380, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, //
p	0x0000, 0x0000, 0x0000, 0x0000, 0x6E00, 0x7F00, 0x7380, 0x6180, 0x6180, //
	0x6180, 0x6180, 0x7380, 0x7F00, 0x6E00, 0x6000, 0x6000, 0x6000, 0x6000, //
q	0x0000, 0x0000, 0x0000, 0x0000, 0x1D80, 0x3F80, 0x7380, 0x6180, 0x6180, //
	0x6180, 0x6180, 0x7380, 0x3F80, 0x1D80, 0x0180, 0x0180, 0x0180, 0x0180, //
r	0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x6700, 0x3F80, 0x3900, 0x3000, //
	0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x0000, 0x0000, 0x0000, //
s	0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1E00, 0x3F80, 0x6180, 0x6000, //
	0x7F00, 0x3F80, 0x0180, 0x6180, 0x7F00, 0x1E00, 0x0000, 0x0000, 0x0000, //
t	0x0000, 0x0000, 0x0800, 0x1800, 0x1800, 0x7F00, 0x7F00, 0x1800, 0x1800, //
	0x1800, 0x1800, 0x1800, 0x1800, 0x1F80, 0x0F80, 0x0000, 0x0000, 0x0000, //
u	0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x6180, 0x6180, 0x6180, 0x6180, //
	0x6180, 0x6180, 0x6180, 0x6380, 0x7F80, 0x3D80, 0x0000, 0x0000, 0x0000, //
v	0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x60C0, 0x3180, 0x3180, 0x3180, //
	0x1B00, 0x1B00, 0x1B00, 0x0E00, 0x0E00, 0x0600, 0x0000, 0x0000, 0x0000, //
	0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xDD80, 0xDD80, 0xDD80, 0x5500, //
	0x5500, 0x5500, 0x7700, 0x7700, 0x2200, 0x2200, 0x0000, 0x0000, 0x0000, //

```

w
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x6180, 0x3300, 0x3300, 0x1E00,
0x0C00, 0x0C00, 0x1E00, 0x3300, 0x3300, 0x6180, 0x0000, 0x0000, 0x0000, //
x
0x0000, 0x0000, 0x0000, 0x0000, 0x6180, 0x6180, 0x3180, 0x3300, 0x3300,
0x1B00, 0x1B00, 0x1B00, 0x0E00, 0x0E00, 0x0E00, 0x1C00, 0x7C00, 0x7000, //
y
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x7FC0, 0x7FC0, 0x0180, 0x0300,
0x0600, 0x0C00, 0x1800, 0x3000, 0x7FC0, 0x7FC0, 0x0000, 0x0000, 0x0000, //
z
0x0380, 0x0780, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0E00, 0x1C00,
0x1C00, 0x0E00, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0780, 0x0380, //
{
0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600,
0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, //
|
0x3800, 0x3C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0E00, 0x0700,
0x0700, 0x0E00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x3C00, 0x3800, //
}
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x3880, 0x7F80,
0x4700, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, //
~
};

FontDef Font_7x10 = {7,10,Font7x10};
FontDef Font_11x18 = {11,18,Font11x18};

```

ST7735/ST7735.c

```

#include "ST7735/ST7735.h"
#include "stdlib.h"

#define TFT_CS_H() HAL_GPIO_WritePin(ST7735_CS_GPIO_Port, ST7735_CS_Pin,
GPIO_PIN_SET)
#define TFT_CS_L() HAL_GPIO_WritePin(ST7735_CS_GPIO_Port, ST7735_CS_Pin,
GPIO_PIN_RESET)
#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)
#define TFT_RES_H() HAL_GPIO_WritePin(ST7735_RES_GPIO_Port, ST7735_RES_Pin,
GPIO_PIN_SET)
#define TFT_RES_L() HAL_GPIO_WritePin(ST7735_RES_GPIO_Port, ST7735_RES_Pin,
GPIO_PIN_RESET)

#define ST7735_COLOR565(r, g, b) (((r & 0xF8) << 8) | ((g & 0xFC) << 3) | ((b
& 0xF8) >> 3))
#define SWAP_INT16_T(a, b) { int16_t t = a; a = b; b = t; }
#define DELAY 0x80

static int16_t _height = ST7735_HEIGHT, _width = ST7735_WIDTH;
static uint8_t _xstart = ST7735_XSTART, _ystart = ST7735_YSTART;

// based on Adafruit ST7735 library for Arduino
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,          // 7: Power control, 3 args, no delay:
0xA2,
0x02,                       // -4.6V
0x84,                       // AUTO mode
ST7735_PWCTR2, 1,          // 8: Power control, 1 arg, no delay:
0xC5,                       // VG25 = 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,                       // BCLK/2, Opamp current small & Medium
low
0x2A,
ST7735_PWCTR5, 2,          // 11: Power control, 2 args, no delay:
0x8A, 0xEE,
ST7735_VMCTR1, 1,          // 12: Power control, 1 arg, no delay:
0x0E,
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

static void ST7735_GPIO_Init(void);
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);

static void ST7735_GPIO_Init(void)
{
}

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

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

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 + _xstart, 0x00, x1 + _xstart };
    ST7735_WriteData(data, sizeof(data));

    // row address set
    ST7735_WriteCommand(ST7735_RASET);
    data[1] = y0 + _ystart;
    data[3] = y1 + _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));
            }
        }
    }
}

void ST7735_Init()
{
    ST7735_GPIO_Init();
    TFT_CS_L();
    ST7735_Reset();
    ST7735_ExecuteCommandList(init_cmds1);
    TFT_CS_H();
}

void ST7735_DrawPixel(uint16_t x, uint16_t y, uint16_t color)
{
    if((x >= _width) || (y >= _height))
        return;

    TFT_CS_L();

    ST7735_SetAddressWindow(x, y, x+1, y+1);
    uint8_t data[] = { color >> 8, color & 0xFF };
    ST7735_WriteData(data, sizeof(data));

    TFT_CS_H();
}

void ST7735_DrawString(uint16_t x, uint16_t y, const char* str, FontDef font,
uint16_t color, uint16_t bgcolor)
{
    TFT_CS_L();

    while(*str)
    {
        if(x + font.width >= _width)
        {
            x = 0;
            y += font.height;
            if(y + font.height >= _height)
            {
                break;
            }
        }

        if(*str == ' ')
        {
            // skip spaces in the beginning of the new line
            str++;
        }
    }
}

```



```

        continue;
    }
}

ST7735_WriteChar(x, y, *str, font, color, bgcolor);
x += font.width;
str++;
}
TFT_CS_H();
}

void ST7735_FillRectangle(uint16_t x, uint16_t y, uint16_t w, uint16_t h,
uint16_t color)
{
    // clipping
    if((x >= _width) || (y >= _height)) return;
    if((x + w - 1) >= _width) w = _width - x;
    if((y + h - 1) >= _height) h = _height - y;

    TFT_CS_L();
    ST7735_SetAddressWindow(x, y, x+w-1, y+h-1);

    uint8_t data[] = { color >> 8, color & 0xFF };
    TFT_DC_D();
    for(y = h; y > 0; y--)
    {
        for(x = w; x > 0; x--)
        {
            HAL_SPI_Transmit(&ST7735_SPI_PORT, data, sizeof(data),
HAL_MAX_DELAY);
        }
    }
    TFT_CS_H();
}

void ST7735_FillScreen(uint16_t color)
{
    ST7735_FillRectangle(0, 0, _width, _height, color);
}

```

State/choosing_state.c

```

#include <State/choosing_state.h>

static const char* algorithms[] = {
    "Native CRC",
    "Software CRC",
    "Software MD5"
};

void set_next_algo(state_info_t* state_info) {
    state_info->algorithm_index += 1;
    state_info->algorithm_index %= ALGORITHMS_COUNT;
}

void set_prev_algo(state_info_t* state_info) {
    state_info->algorithm_index -= 1;
    if (state_info->algorithm_index == -1) {
        state_info->algorithm_index = ALGORITHMS_COUNT - 1;
    }
}

```

```

void write_algorithm_message(state_info_t* state_info) {
    sprintf(state_info->output_buffer, "Current algorithm:");
    format_buffer(state_info->output_buffer, TERMINAL_LINE_WIDTH);
    ST7735_DrawString(0, 0, state_info->output_buffer, Font_11x18,
ST7735_BLACK, ST7735_WHITE);
}

void write_algorithm_name(state_info_t* state_info) {
    uint16_t y = TERMINAL_LINE_HEIGHT * 2;
    sprintf(state_info->output_buffer, algorithms[state_info-
>algorithm_index]);
    format_buffer(state_info->output_buffer, TERMINAL_LINE_WIDTH);
    ST7735_DrawString(0, y, state_info->output_buffer, Font_11x18,
ST7735_BLACK, ST7735_WHITE);
}

void read_algorithm_shift(state_info_t* state_info) {
    HAL_UART_Receive_IT(
        &huart2,
        (uint8_t*)state_info->uart_buffer,
        SHIFT_WORD_SIZE
    );
}

void get_uart_input(state_info_t* state_info) {
    if (state_info->current_state != CHOOSE_ALGO) {
        return;
    }

    read_algorithm_shift(state_info);
}

```

State/entering_state.c

```

#include <State/entering_state.h>

void read_checksum(state_info_t* state_info) {
    uint16_t bytes_to_read = (state_info->algorithm_index == 2 ? 32 : 8) +
1; // for \r
    HAL_UART_Receive_IT(
        &huart2,
        (uint8_t*)state_info->uart_buffer,
        bytes_to_read
    );
}

void print_checksum_helper(state_info_t* state_info) {
    uint16_t bytes_to_read = (state_info->algorithm_index == 2 ? 32 : 8);
    print_uart_message(
        "\r[%u-digit checksum and Enter]: ",
        bytes_to_read
    );
}

void format_reference_checksum(state_info_t* state_info) {
    char* carry = strchr(state_info->reference_checksum, '\r');
    if (carry == NULL) {
        return;
    }

    *carry = '\0';
}

```

```

void write_enter_sum_message(state_info_t* state_info) {
    sprintf(state_info->output_buffer, "Enter reference checksum via
terminal");
    format_buffer(state_info->output_buffer, TERMINAL_LINE_WIDTH);
    ST7735_DrawString(0, 0, state_info->output_buffer, Font_11x18,
ST7735_BLACK, ST7735_WHITE);
}

```

State/execution_state.c

```

#include <State/execution_state.h>

void process_execution(state_info_t* state_info) {
    uint32_t start = HAL_GetTick();
    ALGORITHM algo = get_algo_from_index(state_info->algorithm_index);
    set_algorithm(state_info->sd_card->algorithm_ctx, algo);
    calculate_checksum(state_info->sd_card);
    algorithm_finalize(state_info->sd_card->algorithm_ctx);
    state_info->deltatime = HAL_GetTick() - start;
}

char* extract_result(state_info_t* state_info) {
    return state_info->sd_card->algorithm_ctx->result;
}

void write_checksum_report(state_info_t* state_info) {
    char* result = extract_result(state_info);
    bool is_match = strcmp(state_info->reference_checksum,
extract_result(state_info)) == 0;

    sprintf(state_info->output_buffer, "Checksums %sequal",      is_match ?
"" : "not ");
    format_buffer(state_info->output_buffer, TERMINAL_LINE_WIDTH);
    ST7735_DrawString(0, 0, state_info->output_buffer, Font_11x18,
ST7735_BLACK, ST7735_WHITE);

    uint16_t y1 = TERMINAL_LINE_HEIGHT * (is_match ? 2 : 3);
    ST7735_DrawString(0, y1, "Checksum:", Font_11x18, ST7735_BLACK,
ST7735_WHITE);
    ST7735_DrawString(0, y1 + TERMINAL_LINE_HEIGHT, result, Font_7x10,
ST7735_BLACK, ST7735_WHITE);

    uint16_t y2 = y1 + TERMINAL_LINE_HEIGHT * 2;
    clear_buffer(state_info->output_buffer, DEFAULT_BUFFER_SIZE);
    sprintf(state_info->output_buffer, "Executed in %lu ms", state_info-
>deltatime);
    ST7735_DrawString(0, y2, state_info->output_buffer, Font_11x18,
ST7735_BLACK, ST7735_WHITE);

    if (!is_match) {
        make_error_sound();
    }
}

ALGORITHM get_algo_from_index(int algorithm_index) {
    switch (algorithm_index) {
    case 0:
        return HAL_CRC;
    case 1:
        return CRC8;
    case 2:

```

```

        return MD5;
    default:
        print_uart_message("this shouldn't happen\r");
        return HAL_CRC;
    }
}

void make_error_sound() {
    HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
    HAL_Delay(1000);
    HAL_TIM_PWM_Stop(&htim2, TIM_CHANNEL_1);
}

```

State/state_info_t.c

```

#include "State/state_info_t.h"

state_info_t* new_state_info() {
    state_info_t* state_info = calloc(1, sizeof(state_info_t));

    clear_buffer(state_info->output_buffer, DEFAULT_BUFFER_SIZE);

    state_info->algorithm_index = 0;
    state_info->current_state = IDLE;
    state_info->state_request = IDLE;
    state_info->deltatime = 0;

    state_info->uart_write_ptr = 0;
    clear_buffer(state_info->uart_buffer, DEFAULT_BUFFER_SIZE);
    clear_buffer(state_info->reference_checksum, MAX_CRC_LEN + 1);

    state_info->sd_card = new_sd_card();
    mount_sd_card(state_info->sd_card);

    return state_info;
}

void free_state_info(state_info_t* state_info) {
    free_sd_card(state_info->sd_card);
    free(state_info);
}

```

State/state.c

```

#include "State/state.h"

void set_state(state_info_t* state_info, STATE new_state) {
    state_info->current_state = new_state;
    reduce_state_change_to_effect(state_info);
}

void reset_state(state_info_t* state_info) {
    set_state(state_info, CHOOSE_ALGO);
    reset_calculation(state_info->sd_card);
    state_info->algorithm_index = 0;
}

void check_state_request(state_info_t* state_info) {
    if (state_info->state_request == IDLE) {
        return;
    }
}

```

```

        set_state(state_info, state_info->state_request);
        state_info->state_request = IDLE;
    }

    void reduce_state_to_action(state_info_t* state_info) {
        if (state_info->current_state != CHOOSE_ALGO) {
            return;
        }

        write_algorithm_name(state_info);
    }

    void reduce_state_change_to_effect(state_info_t* state_info) {
        switch (state_info->current_state) {
            case CHOOSE_ALGO:
                ST7735_FillScreen(ST7735_WHITE);
                clear_buffer(state_info->output_buffer, DEFAULT_BUFFER_SIZE);
                write_algorithm_message(state_info);
                print_uart_message("%s", CHOOSE_ALGO_MSG);
                break;
            case ENTER_SUM:
                ST7735_FillScreen(ST7735_WHITE);
                clear_buffer(state_info->output_buffer, DEFAULT_BUFFER_SIZE);
                write_enter_sum_message(state_info);
                print_checksum_helper(state_info);
                read_checksum(state_info);
                break;
            case EXECUTE:
                ST7735_FillScreen(ST7735_WHITE);
                clear_buffer(state_info->output_buffer, DEFAULT_BUFFER_SIZE);
                process_execution(state_info);
                write_checksum_report(state_info);
                break;
            case RESTART_INTENT:
                reset_state(state_info);
                break;
            default:
                print_uart_message("this shouldn't happen\r");
                break;
        }
    }
}

```

utils.c

```

#include "utils.h"

void format_buffer(char* buffer, size_t line_width) {
    char res[DEFAULT_BUFFER_SIZE] = {'\0'};

    size_t shift = 0;
    size_t len = strlen(buffer);
    for (size_t i = 0; i < len; ++i) {
        if (buffer[i] != ' ') {
            res[i + shift] = buffer[i];
            continue;
        }

        size_t spaces_amount = line_width - ((i + shift) % line_width);
        for (size_t j = 0; j < spaces_amount; ++j) {
            res[i + j + shift] = ' ';
        }
        shift += spaces_amount - 1;
    }
}

```

```

    }

    strcpy(buffer, res);
}

void clear_buffer(char* buffer, size_t size) {
    memset(buffer, '\0', size);
}

void char_array_to_uint32_array(char* src, uint32_t* dest, int len) {
    for (int i = 0; i < len; ++i) {
        dest[i] = (uint32_t)src[i];
    }
}

void uint32_array_to_char_array(uint32_t* src, char* dest, int len) {
    for (int i = 0; i < len; ++i) {
        dest[i] = (char)src[i];
    }
}

void to_lower(char* string) {
    size_t len = strlen(string);
    for (size_t i = 0; i < len; ++i) {
        string[i] = tolower(string[i]);
    }
}

void print_uart_message(char* format, ...) {
    va_list args;
    char res[DEFAULT_BUFFER_SIZE]={0};
    va_start(args, format);
    vsprintf(res, format, args);
    va_end(args);

    HAL_UART_Transmit(&huart2, (uint8_t*)res, strlen(res), 200);
}

```

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.
 *
 */

```

```

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

/* Private includes -----
*/
/* USER CODE BEGIN Includes */
#include "State/state.h"
/* USER CODE END Includes */

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

/* USER CODE END PTD */

/* Private define -----
*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

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

/* USER CODE END PM */

/* Private variables -----
*/
CRC_HandleTypeDef hcrc;

SPI_HandleTypeDef hspi1;
SPI_HandleTypeDef hspi2;

TIM_HandleTypeDef htim2;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
state_info_t* state_info = NULL;
/* USER CODE END PV */

/* Private function prototypes -----
*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_SPI1_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_SPI2_Init(void);
static void MX_CRC_Init(void);
static void MX_TIM2_Init(void);
/* USER CODE BEGIN PFP */
/* 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_SPI1_Init();
    MX_USART2_UART_Init();
    MX_FATFS_Init();
    MX_SPI2_Init();
    MX_CRC_Init();
    MX_TIM2_Init();
    /* USER CODE BEGIN 2 */
    ST7735_Init();

    state_info = new_state_info();
    state_info->state_request = CHOOSE_ALGO;
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {
        check_state_request(state_info);
        get_uart_input(state_info);
        reduce_state_to_action(state_info);
        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
    }
    free_state_info(state_info);
    /* 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 CRC Initialization Function
 * @param None
 * @retval None
 */
static void MX_CRC_Init(void)
{
    /* USER CODE BEGIN CRC_Init 0 */

    /* USER CODE END CRC_Init 0 */

    /* USER CODE BEGIN CRC_Init 1 */

    /* USER CODE END CRC_Init 1 */
    hcrc.Instance = CRC;
    if (HAL_CRC_Init(&hcrc) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN CRC_Init 2 */

    /* USER CODE END CRC_Init 2 */
}

```

```

}

/**
 * @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 SPI2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_SPI2_Init(void)
{
    /* USER CODE BEGIN SPI2_Init 0 */

    /* USER CODE END SPI2_Init 0 */

    /* USER CODE BEGIN SPI2_Init 1 */

    /* USER CODE END SPI2_Init 1 */
    /* SPI2 parameter configuration*/
    hspi2.Instance = SPI2;
    hspi2.Init.Mode = SPI_MODE_MASTER;
    hspi2.Init.Direction = SPI_DIRECTION_2LINES_RXONLY;
    hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
    hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;

```

```

hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
hspi2.Init.NSS = SPI_NSS_SOFT;
hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
hspi2.Init.CRCPolynomial = 10;
if (HAL_SPI_Init(&hspi2) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN SPI2_Init 2 */

/* USER CODE END SPI2_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};
    TIM_OC_InitTypeDef sConfigOC = {0};

    /* USER CODE BEGIN TIM2_Init 1 */

    /* USER CODE END TIM2_Init 1 */
    htim2.Instance = TIM2;
    htim2.Init.Prescaler = 127;
    htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim2.Init.Period = 127;
    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();
    }
    if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
    {
        Error_Handler();
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
    if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) !=
        HAL_OK)
    {
        Error_Handler();
    }

```

```

    }
    sConfigOC.OCMode = TIM_OCMode_PWM1;
    sConfigOC.Pulse = 63;
    sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
    sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
    if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN TIM2_Init 2 */

    /* USER CODE END TIM2_Init 2 */
    HAL_TIM_MspPostInit(&htim2);

}

/**
 * @brief USART2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART2_UART_Init(void)
{
    /* USER CODE BEGIN USART2_Init 0 */

    /* USER CODE END USART2_Init 0 */

    /* USER CODE BEGIN USART2_Init 1 */

    /* USER CODE END USART2_Init 1 */
    huart2.Instance = USART2;
    huart2.Init.BaudRate = 115200;
    huart2.Init.WordLength = UART_WORDLENGTH_8B;
    huart2.Init.StopBits = UART_STOPBITS_1;
    huart2.Init.Parity = UART_PARITY_NONE;
    huart2.Init.Mode = UART_MODE_TX_RX;
    huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart2.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart2) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN USART2_Init 2 */

    /* USER CODE END USART2_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_GPIOA_CLK_ENABLE();
__HAL_RCC_GPIOB_CLK_ENABLE();

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10|GPIO_PIN_4, GPIO_PIN_RESET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12,
GPIO_PIN_RESET);

/*Configure GPIO pins : PB10 PB4 */
GPIO_InitStruct.Pin = GPIO_PIN_10|GPIO_PIN_4;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pins : PA10 PA11 PA12 */
GPIO_InitStruct.Pin = GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12;
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);

/*Configure GPIO pins : NEXT_ALGO_BUTTON_Pin PREV_ALGO_BUTTON_Pin
SUBMIT_BUTTON_Pin RESTART_BUTTON_Pin */
GPIO_InitStruct.Pin =
NEXT_ALGO_BUTTON_Pin|PREV_ALGO_BUTTON_Pin|SUBMIT_BUTTON_Pin|RESTART_BUTTON_Pi
n;
GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
GPIO_InitStruct.Pull = GPIO_PULLUP;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/* EXTI interrupt init*/
HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

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

/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == RESTART_BUTTON_Pin) {
        print_uart_message("Program restart\r");
        state_info->state_request = RESTART_INTENT;
        return;
    }

    if (state_info->current_state != CHOOSE_ALGO) {
        return;
    }

    switch (GPIO_Pin) {
        case NEXT_ALGO_BUTTON_Pin:
            set_next_algo(state_info);
            break;
        case PREV_ALGO_BUTTON_Pin:
            set_prev_algo(state_info);
            break;
    }
}

```

```

    case SUBMIT_BUTTON_Pin:
        HAL_UART_AbortReceive_IT(&huart2);
        state_info->state_request = ENTER_SUM;
        break;
    default:
        print_uart_message("this shouldn't happen\r");
        break;
}

}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
    if (state_info->current_state == ENTER_SUM) {
        strcpy(state_info->reference_checksum, state_info->uart_buffer);
        format_reference_checksum(state_info);
        clear_buffer(state_info->uart_buffer, DEFAULT_BUFFER_SIZE);
        state_info->state_request = EXECUTE;
    } else if (state_info->current_state == CHOOSE_ALGO) {
        if (strcmp(state_info->uart_buffer, "NEXT\r") == 0) {
            set_next_algo(state_info);
        } else if (strcmp(state_info->uart_buffer, "PREV\r") == 0) {
            set_prev_algo(state_info);
        }
        print_uart_message(CHOOSE_ALGO_MSG);
    }
}

/* 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 */
}

```

Приложение Б

Перечень элементов

На 1 листе