|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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***

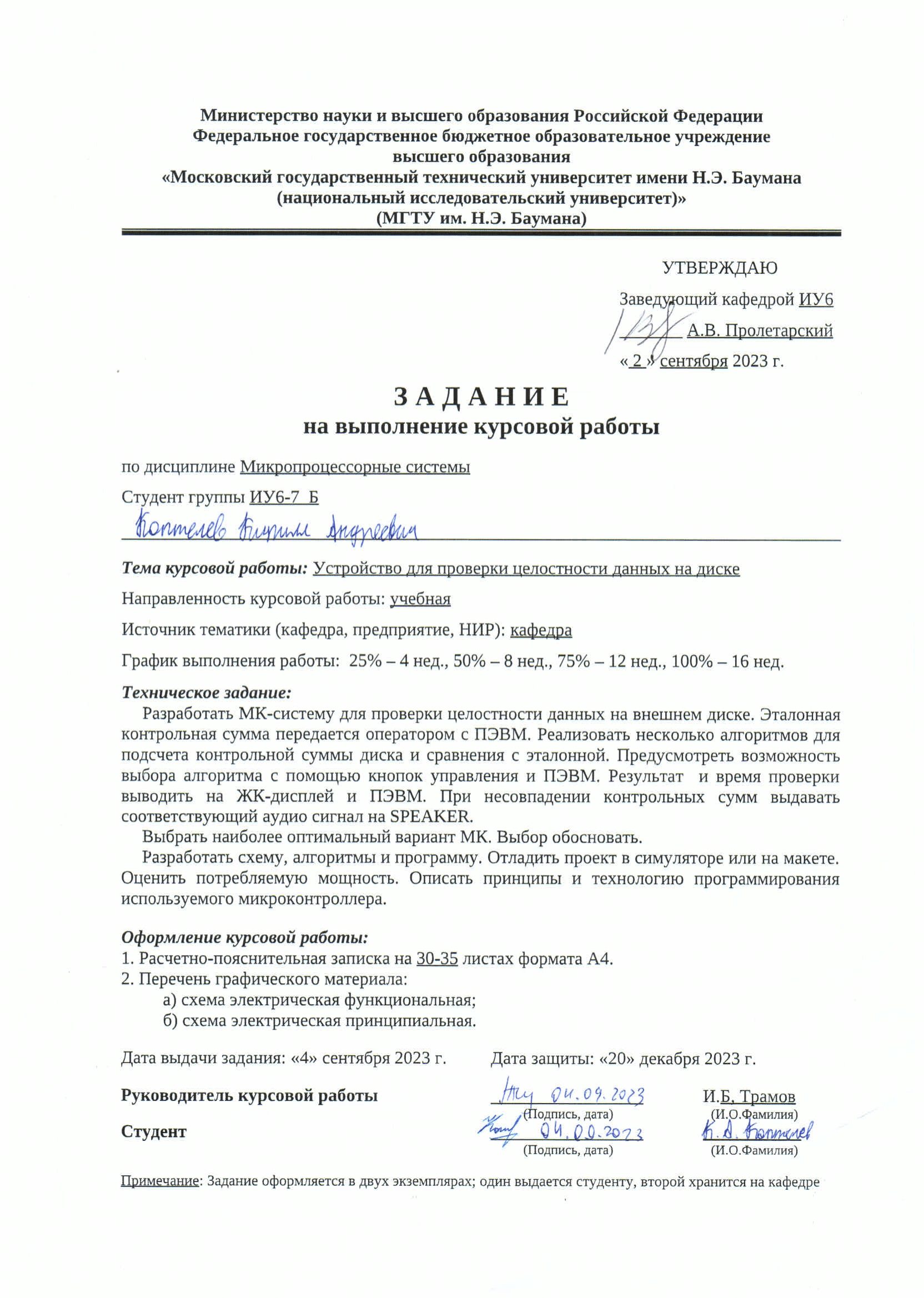
***по дисциплине «Микропроцессорные системы»***

***на тему:***

**Устройство для проверки целостности данных на диске**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент |  |  |  | К.А. Коптелев |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Руководитель |  |  |  | И.Б. Трамов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |

*2023 г.*



**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](#_Toc153222214)

[1 Конструкторская часть 8](#_Toc153222215)

[1.1 Анализ требований и принцип работы системы 8](#_Toc153222216)

[1.2 Проектирование функциональной схемы 10](#_Toc153222217)

[1.2.1 Микроконтроллер STM32F103C8T6 10](#_Toc153222218)

[1.2.1.1 Используемые элементы 17](#_Toc153222219)

[1.2.1.2 Распределение портов 18](#_Toc153222220)

[1.2.1.3 Организация памяти 19](#_Toc153222221)

[1.2.2 Прием данных от ПЭВМ 20](#_Toc153222222)

[1.2.3 Настройка USART для взаимодействия с ПЭВМ 22](#_Toc153222223)

[1.2.4 LCD-дисплей ST7735 30](#_Toc153222224)

[1.2.5 Настройка SPI для взаимодействия с LCD-дисплеем 30](#_Toc153222225)

[1.2.6 Настройка SPI для взаимодействия с сокетом SD-карты. 31](#_Toc153222226)

[1.2.7 Генератор тактовых импульсов и сброс 31](#_Toc153222227)

[1.2.8 Использование таймера для генерации звукового сигнала 31](#_Toc153222228)

[1.2.9 Построение функциональной схемы 36](#_Toc153222229)

[1.3 Проектирование принципиальной схемы 36](#_Toc153222230)

[1.3.1 Разъем программатора 36](#_Toc153222231)

[1.3.2 Расчет сопротивления резисторов 37](#_Toc153222232)

[1.3.3 Расчет потребляемой мощности 38](#_Toc153222233)

[1.4 Алгоритмы работы системы 40](#_Toc153222234)

[1.4.1 Общее описание работы программы 40](#_Toc153222235)

[1.4.2 Детализация и пояснение основных функций 41](#_Toc153222236)

[2 Технологическая часть 44](#_Toc153222237)

[2.1 Отладка и тестирование программы 44](#_Toc153222238)

[2.2 Настройка таймера и работа фаз светофора 45](#_Toc153222239)

[2.3 Симуляция работы системы 46](#_Toc153222240)

[2.4 Способы программирования МК 49](#_Toc153222241)

[ЗАКЛЮЧЕНИЕ 52](#_Toc153222242)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 53](#_Toc153222243)

[Приложение А 54](#_Toc153222244)

[Приложение Б 103](#_Toc153222245)

[Приложение В 104](#_Toc153222246)

**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.

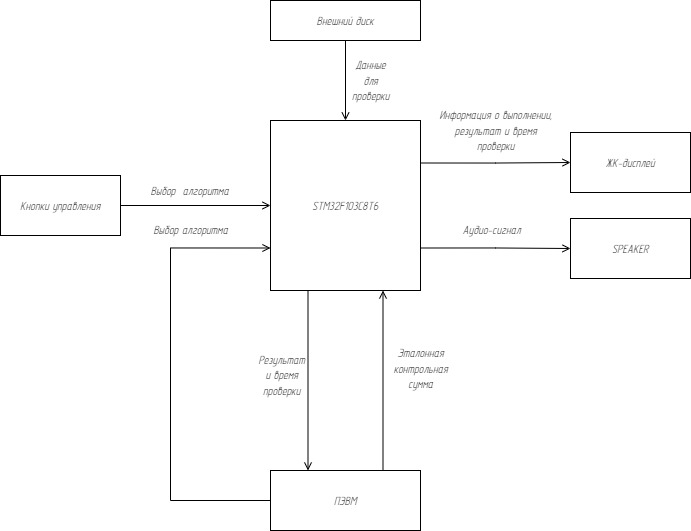


Рисунок 1 – Структурная схема устройства

## 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 [1]:

* 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 инстру-кцию | 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

1. 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

1. Toshiba, имеющие следующие характеристики:

* FLASH до 1,5Mбайт;
* 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 [2].

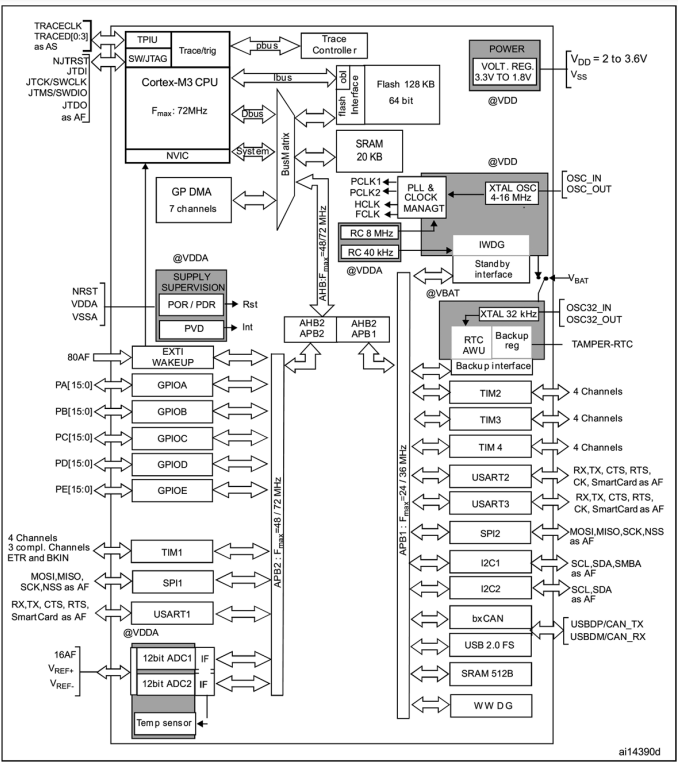


Рисунок 2 – Структурная схема МК STM32F103C8T6

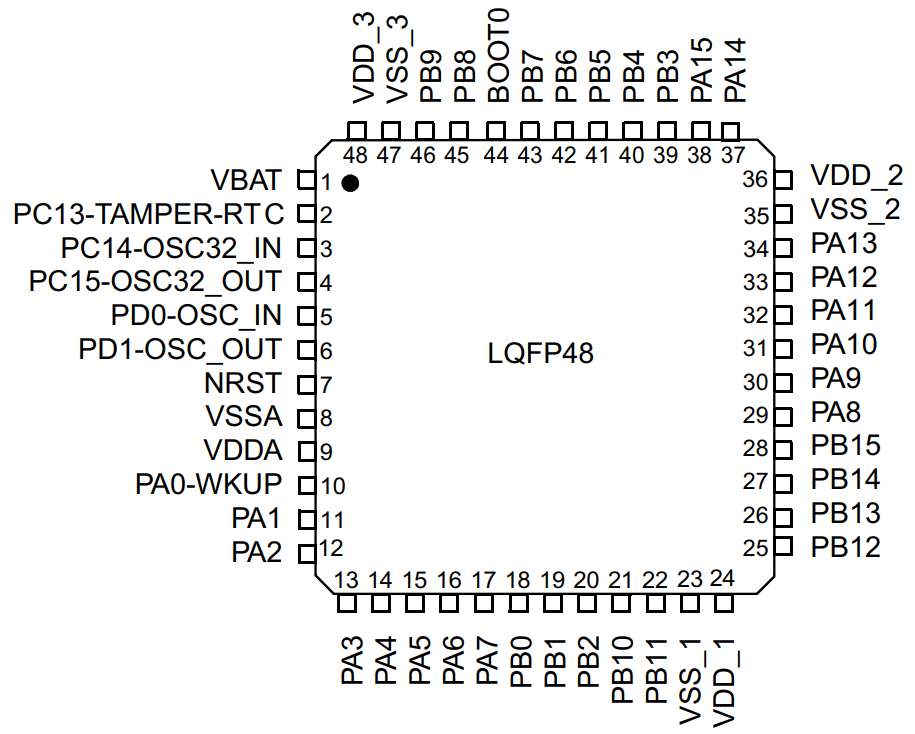
****

Рисунок 3 – УГО МК STM32F103C8T6

Он обладает следующими характеристиками [3]:

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.

1. Память:

* 64 КБ флеш-памяти для программного кода.
* 20 КБ ОЗУ (SRAM) для хранения данных.
* Возможность расширения памяти с использованием внешних устройств.

1. 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.

1. 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.

1. Прочие особенности:

* Встроенные таймеры и счетчики для управления временем и частотой.
* Низкое энергопотребление в режиме ожидания.
* Защита от переполнения стека и ошибок программирования.

1. Программирование и разработка:

* Поддержка различных интегрированных сред разработки (IDE), таких как Keil, STM32CubeIDE, и других.
* Обширная документация, примеры кода и библиотеки для упрощения разработки.

1. Применение:

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

1. Напряжение питания: 2– 3.6В.

### 1.2.1.1 Используемые элементы

Проверить

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 [4].

Порты А, В – назначения каждого из них описано в разделе *1.2.1.2*.

Указатель стека – используется для работы со стеком, при вызове подпрограмм. В коде они присутствуют.

SRAM – статическая память МК, где хранятся объявленные переменные.

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

АЛУ – блок процессора, который под управлением устройства управления служит для выполнения арифметических и логических преобразований над данными.

SREG – регистр состояния, содержит набор флагов, показывающих текущее состояние работы микроконтроллера.

SPI – интерфейс для связи МК с другими внешними устройствами. В проекте используется для прошивки МК, для связи с внешним диском, а также для связи с LCD-дисплеем.

Программный счетчик – используется для указания следующей команды выполняемой программы.

Память Flash – память МК, в котором хранится загруженная в него программа.

Регистры команды – содержит исполняемую в текущий момент (или следующий) команду, то есть команду, адресуемую счетчиком команд.

Декодер – блок, выделяющий код операции и операнды команды, а затем вызывающий микропрограмму, которая исполняет данную команду.

Сигналы управления – синхронизируют обработку данных.

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

Таймеры/счетчики – включает в себя 4 16-ти разрядных таймера TIM1, TIM2, TIM3, TIM4. В проекте используется только один канал TIM2 для генерации ШИМ для динамика.

Управление синхронизацией и сбросом (MCU CTRL. & Timing) – в этом блоке обрабатываются тактовые сигналы и принимается сигнал сброса.

Прерывания – контроллер прерываний обрабатывает внешние прерывания и прерывания от периферийных устройств МК (таймеров, портов ввода/вывода). В проекте используются внешние прерывания для обработки нажатия кнопок и прерывания UART для получения ввода с ПЭВМ.

UART – через этот интерфейс в МК передается информация из ПЭВМ.

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

### 1.2.1.2 Распределение портов

МК STM32F103C8T6 содержит пять портов – А, В, С, D и Е. Опишем назначение тех, что используются в данной системе для её функционирования.

Порт А:

* РА2 – отправка данных по UART на ПЭВМ;
* РА3 – 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:

* РВ5 – пин с внешним прерыванием по нажатии кнопки переключения на следующий алгоритм;
* РВ6 – пин с внешним прерыванием по нажатии кнопки переключения на предыдущий алгоритм;
* РВ7 – пин с внешним прерыванием по нажатии кнопки подтверждения выбора алгоритма;
* РВ8 – пин с внешним прерыванием по нажатии кнопки перезапуска вычислений;
* РВ10 – 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;
* РВ14 – MISO-пин для SPI для сокета SD-карты;

### 1.2.1.3 Организация памяти

Схема организации памяти МК STM32F103C8T6 показана на рисунке 4.

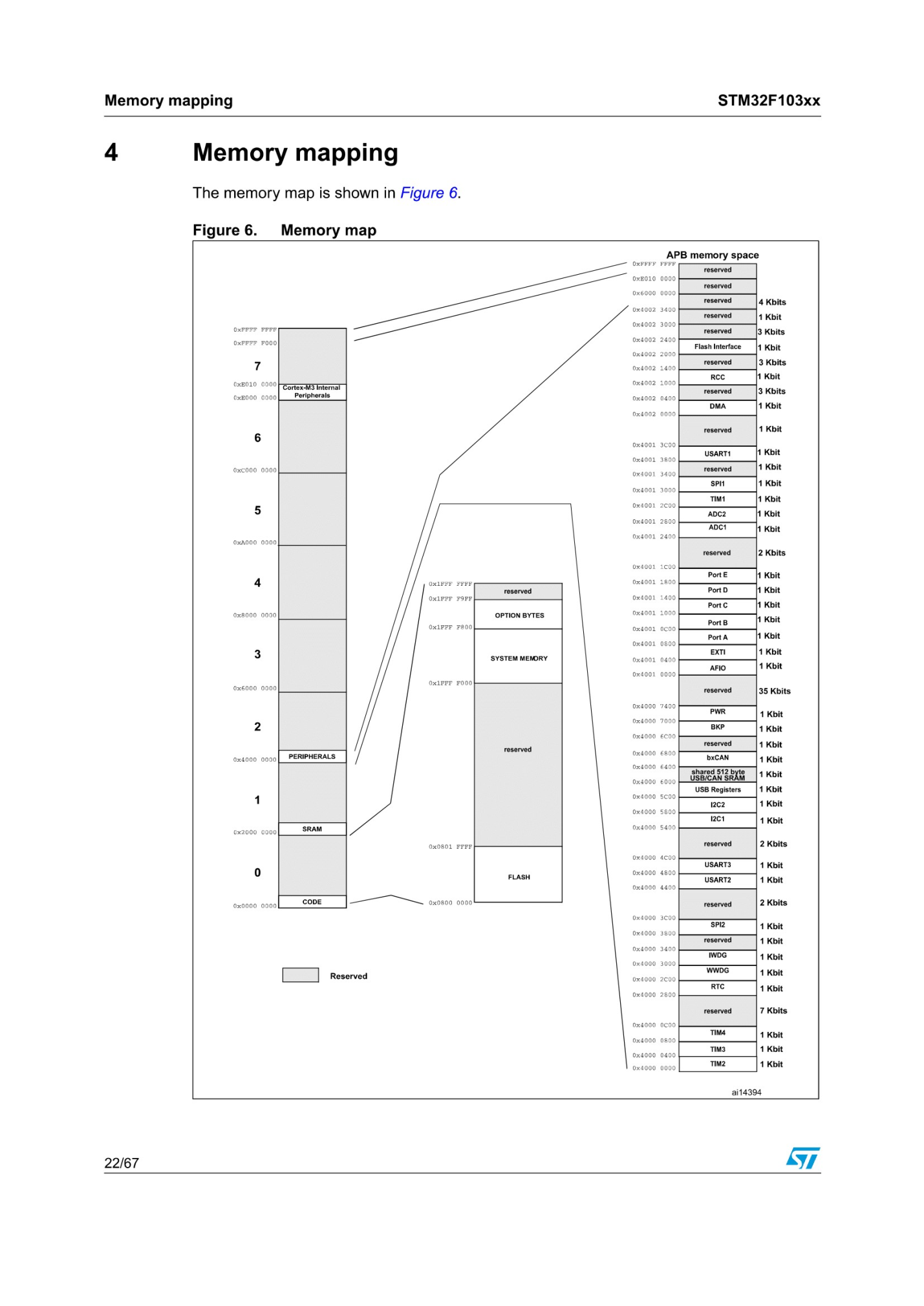


Рисунок 4 – Организация памяти МК STM32F103C8T6

Что-то написать

### 1.2.2 Прием данных от ПЭВМ

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

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

К внешнему устройству MAX232 подключен через разъем DB-9. На схеме условное обозначение – ХР2.

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

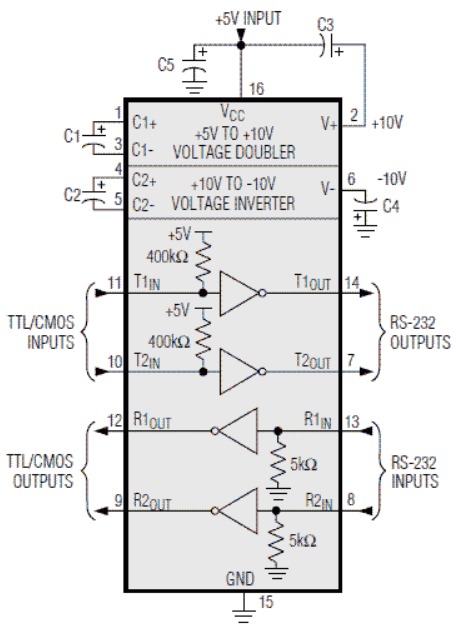


Рисунок 5 – Преобразователь MAX232

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

USART был выбран для использования, так как 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 5 и 6.

|  |  |
| --- | --- |
|  |  |

Рисунки 5,6 – 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 7.

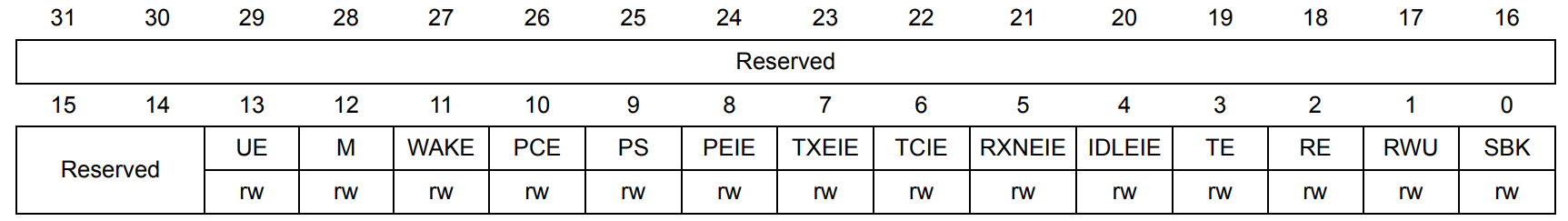


Рисунок 7 – Регистр USART\_CR1

Описание регистра:

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

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

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

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

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

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

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

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

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

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

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

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

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

**SBK:**Send break - отправить Break посылку. Бит может быть установлен и сброшен программно. Его необходимо программно установить в 1 для формирования Break посылки, он будет сброшен аппаратно во время формирования stop-бита в Break фрейме. 0: Break-символ не передаётся; 1: Break-символ будет передан.

Теперь опишем регистр BRR, с помощью которого контролируется скорость передачи данных через USART. Регистр представлен на рисунке 10.

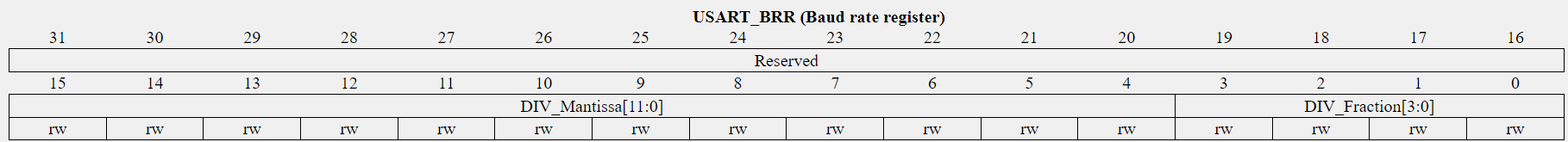


Рисунок 10 – Регистр BRR

**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 11 представлена схема, показывающая, как именно высчитывается скорость передачи.

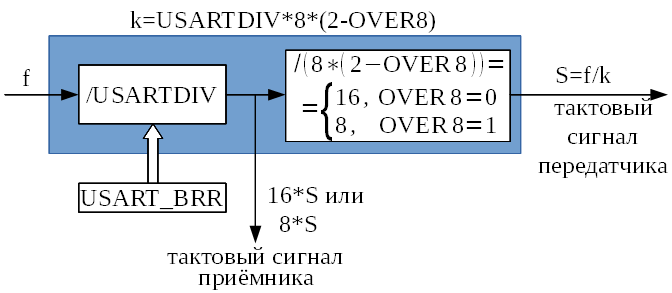


Рисунок 11 – Вычисление скорости приема и передачи

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

Далее рассмотрим USART\_DR – регистр, через который передаются непосредственно данные. Он представлен на рисунке 12.

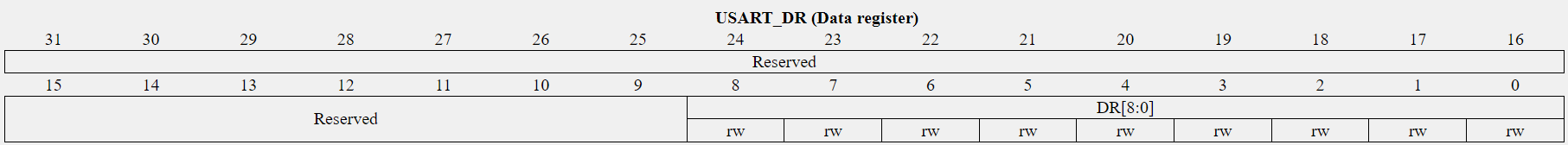


Рисунок 12 – Регистр данных

**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 13.

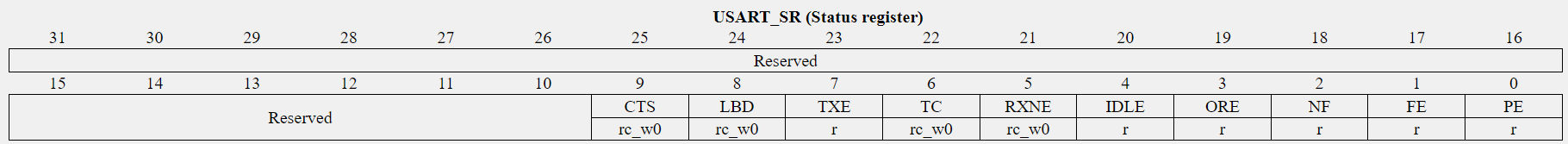


Рисунок 13 – Регистр статуса

**CTS:**CTS flag - флаг изменения состояния nCTS. Устанавливается аппаратно, когда происходит переключение сигнала на входе nCTS. Если установлен бит CTSIE (USART\_CR3.CTSIE==1), то при установке флага генерируется прерывание. Флаг сбрасывается программно записью 0.

**LBD:**LIN break detection flag - флаг приёма посылки Break. Устанавливается аппаратно при обнаружении посылки Break на входе; если установлен бит LBDIE (USART\_CR3.LBDIE==1), то генерируется прерывание. Флаг срсывается программно записью 0.

**TXE:**Transmit data register empty - флаг устанавливается аппаратно, когда содержимое регистра передаваемых данных TDR пересылается в сдвигающий регистр (доступ к TDR осуществляется путём записи в регистр USART\_DR). Если установлен бит TXEIE (USART\_CR1.TXEIE==1), генерируется прерывание. Флаг сбрасывается путём записи в регистр USART\_DR.

**TC:**Transmission complete - флаг завершения передачи, устанавливается аппаратно, если передача фрейма завершена, и флаг TXE установлен (т.е. регистр передаваемых данных пуст, больше нет данных для передачи). Если USART\_CR1.TCIE==1, то при установке флага генерируется прерывание. Флаг сбрасывается программно последовательностью действий: чтение регистра USART\_SR, затем запись в USART\_DR. Также бит может быть сброшен записью в него 0. Примечание. После сброса этот бит установлен.

**RXNE:**Read data register not empty - регистр данных для чтения не пуст. Флаг устанавливается аппаратно, когда содержимое принимающего сдвигающего регистра передаётся в регистр принимаемых данных RDR. Если USART\_CR1.RXNEIE==1, при этом генерируется прерывание. Флаг сбрасывается чтением из регистра USART\_DR. Также бит может быть сброшен записью в него 0.

**IDLE:**IDLE line detected - линия свободна. Флаг устанавливается аппаратно, если обнаружено что линия свободна. Это происходит, если получен целый фрейм единиц. При этом генерируется прерывание, если USART\_CR1.IDLEIE==1. Флаг сбрасывается программно последовательностью действий: чтение регистра USART\_SR с последующим чтением из регистра USART\_DR.

**ORE:**Overrun error - ошибка переполнения. Флаг устанавливается аппаратно, когда слово, полученное в сдвигающей регистр готово к перемещению в регистр принимаемых данных RDR, но RXNE==1 (регистр RDR не пуст, содержит ещё не прочитанные из него принятые USART данные). Если USART\_CR1.RXNEIE==1, то при установке флага генерируется исключение. Флаг сбрасывается программно последовательностью действий: чтение из регистра USART\_SR с последующим чтением из USART\_DR.

**NF:**Noise detected flag - флаг устанавливается аппаратно при обнаружении шума в полученном фрейме. Сбрасывается программно последовательностью действий: чтение из регистра USART\_SR, затем чтение из регистра USART\_DR.

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

**PE:**Parity error - ошибка чётности. Флаг устанавливается аппаратно, когда в принятом фрейме обнаружена ошибка чётности (если контроль чётности включён). Если USART\_CR1.PEIE==1, то генерируется прерывание. Флаг сбрасывается программно последовательностью действий: чтение из регистра USART\_SR, затем чтение либо запись регистра USART\_DR. Перед сбросом флага, программа должна дождаться установки флага RXNE (регистр данных для чтения не пуст).

### 1.2.4 LCD-дисплей ST7735

Почему выбран, полное описание, какие команды принимает, работа в схеме

### 1.2.5 Настройка SPI для взаимодействия с LCD-дисплеем

Описание SPI, регистры, скорость, работа в программе.

### 1.2.6 Настройка SPI для взаимодействия с сокетом SD-карты.

Работа в программе

### 1.2.7 Генератор тактовых импульсов и сброс

Написать про использование внутреннего генератора частоты и про сброс

### 1.2.8 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 13.

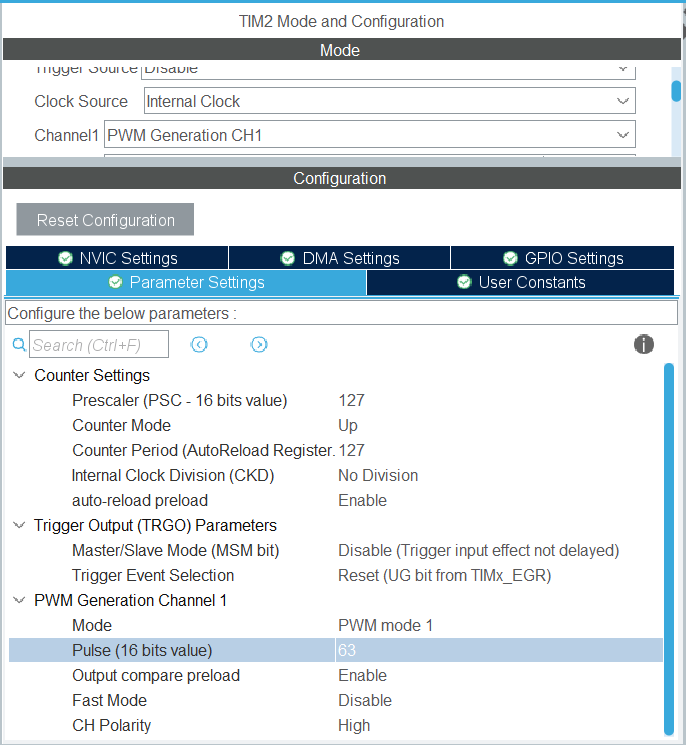


Рисунок 13 – Конфигурация таймера TIM2 для работы в ШИМ-режиме

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

Структурная схема таймера представлена на рисунке 14.

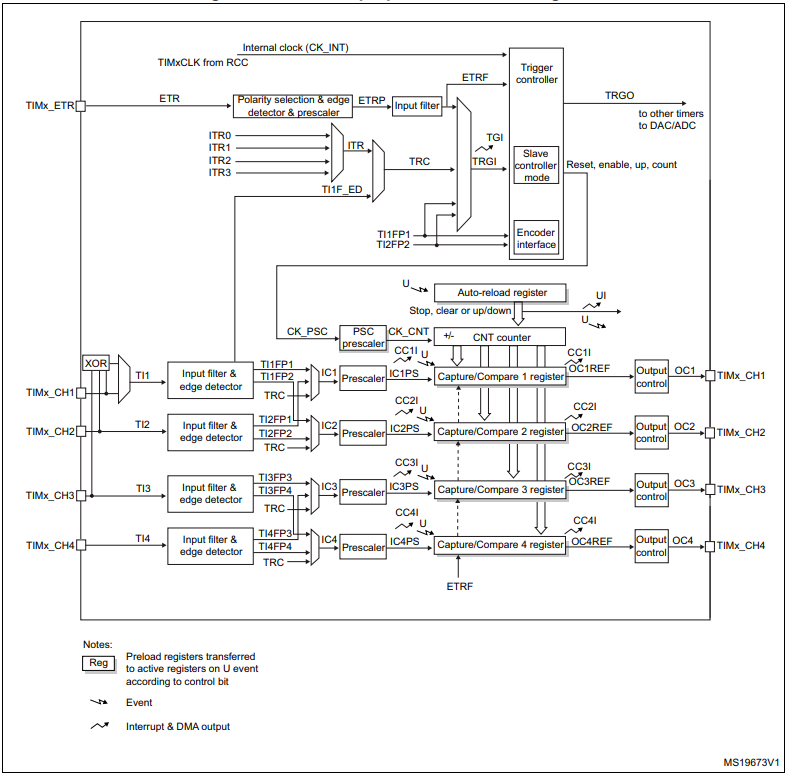


Рисунок 14 – Структурная схема таймера

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 15.

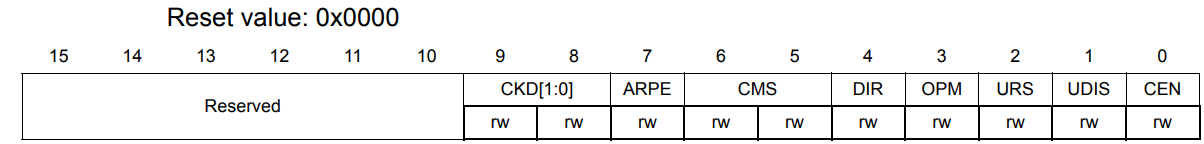


Рисунок 15 – Регистр TIM2\_CR1

**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 16. 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.

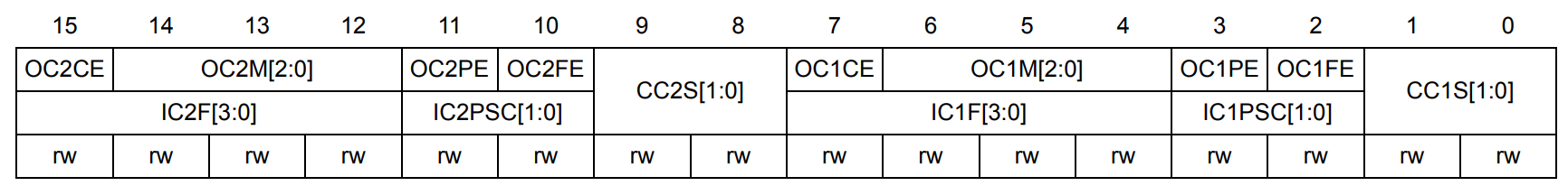


Рисунок 16 – Регистр TIM2\_CR2

**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 вариант – 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).

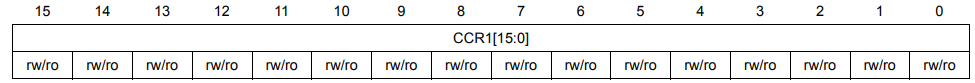
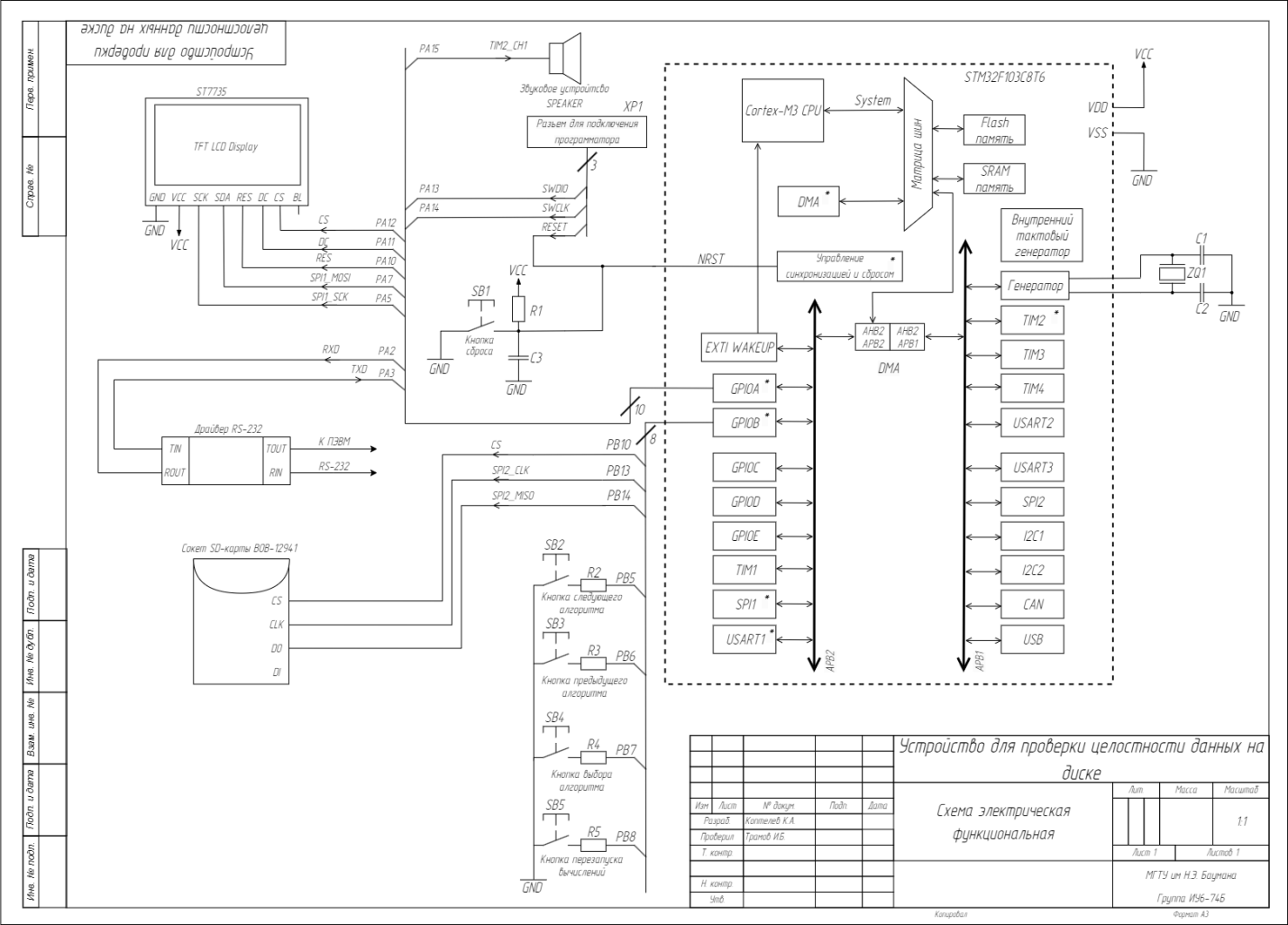


Рисунок 17 – регистр TIM2\_CCR1

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 = AutoReload \* Prescaler \* FreqPWM, тогда для того, чтобы получить FreqPWM примерно равную 500Гц, можно выставить AutoReload и Prescaler как 127. 8000000 / 127 / 127 ~= 500Гц. При этом используется коэффициент заполнения 50% - Для этого возьмем для Pulse примерно половину Prescaler – 63. Именно это значение будет храниться в TIM2\_CCR1.

### 1.2.9 Построение функциональной схемы

На основе всех вышеописанных сведений была спроектирована функциональная схема разрабатываемой системы, показанная на рисунке 18 и в приложении Б [8, 9].

Рисунок 18 – 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 сопротивления резисторов

Рассчитать

Резисторы по своей сути предназначаются, чтобы уменьшить значение тока, подаваемого на устройство. В светофорном блоке резисторы установлены перед светодиодом и со стороны базы транзистора [13].

Сначала определим сопротивление резистора, который предназначен для ограничения величины тока, протекающего через светодиод.

= 105 Ом

где UR – напряжение питания на резисторе,

I – сила протекающего тока.

Ближайшие стандарты – это 100 Ом и 110 Ом. Выбран будет резистор с большим номиналом, то есть 110 Ом – CF-100.

Напряжение на резисторе рассчитывается по формуле:

UR = UK - ΔUVD - ΔUK = 5 - 2,5 - 0,4 = 2,1 В

где UK – напряжение питания;

ΔUVD – напряжение светодиода;

ΔUK – падение напряжение на переходе коллектор-эмиттер.

Ток, протекающий через светодиод, снизился:

= 0,019 А = 19 мА

Для определения резистора со стороны базы, посчитаем ток базы.

= 0,000095 мкА

где – ток коллектора, равный 0,019А;

β – коэффициент усиления биполярного транзистора, для модели 2N2222 равный 200.

= 22105 Ом = 22,1 кОм

Ближайшие стандарты – это 20 кОм и 25 кОм. Выбран будет резистор с большим номиналом, то есть 25 кОм – MO-100.

### 1.3.3 Расчет потребляемой мощности

Рассчитать

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 +5В. Мощность, потребляемая один устройством, в статическом режиме, рассчитывается формулой:

Р = U \* I

где U – напряжение питания (В);

I– ток потребления микросхемы (мА).

Также в схеме присутствуют резисторы и транзисторы. Мощность для резисторов рассчитывается по формуле:

Р = I2 \* R

где R – сопротивление резистора;

I – ток, проходящий через резистор.

Для транзисторов по формуле:

Р = U \* I

где U – напряжение коллектор-эмиттер;

I – ток коллектора (он был рассчитан в разделе *1.3.3*).

Расчет потребляемого напряжения для каждой микросхемы показан в таблице 8.

Таблица 8 – Потребляемая мощность

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Микросхема | Ток потребления, мА | Потребляемая мощность, мВт | Количе-  ство устройств | Суммарная потребляемая мощность, мВт |
| ATmega8515 | 40 | 200 | 1 | 200 |
| MAX232 | 8 | 40 | 1 | 40 |
| К155ЛА1 | 4 | 20 | 1 | 20 |
| К533ЛН1 | 3 | 15 | 1 | 15 |
| К155ЛА8 | 8 | 40 | 5 | 200 |
| L-793SRD-C | 18 | 47 | 12 | 564 |
| L-813GD | 18 | 47 | 12 | 564 |
| L-793YD | 18 | 47 | 4 | 188 |

Также в схеме используются 28 резисторов CF-100 и МО-100 с номиналом 110 и 25 кОм соответственно, 2 резистора CF-25 с номиналом 1 кОм, и 28 транзисторов 2N2222.

Pсуммарная = РATmega8515 + РMAX232 + РК155ЛА1 + РК533ЛН1 + РК155ЛА8 + РL-793SRD-C + РL-813GD + РL-793YD + Ррезисторов + Ртранзисторов = 200 + 40 + 20 + 15 + 200 + 564 + 564 + 188 + 56,5 + 42 = 2189,5 мВт

Суммарная потребляемая мощность системы равна 2189,5 мВт = 2,2 Вт.

**1.3.5 Построение принципиальной схемы**

На основе всех вышеописанных сведений была спроектирована принципиальная схема разрабатываемой системы, показанная на рисунке 19 и в приложении Б [8, 9].

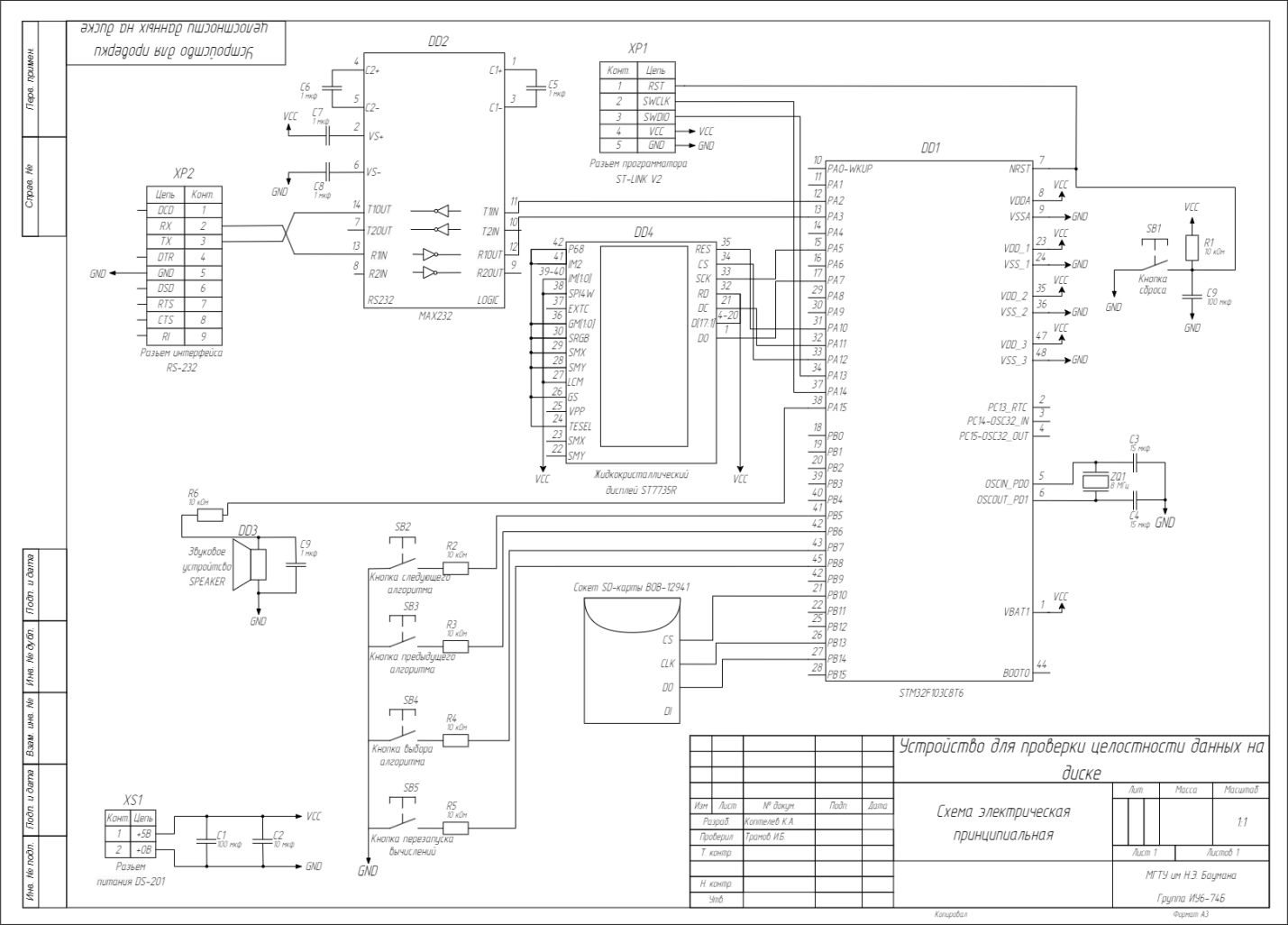


Рисунок 19 – Принципиальная схема

## 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 20.

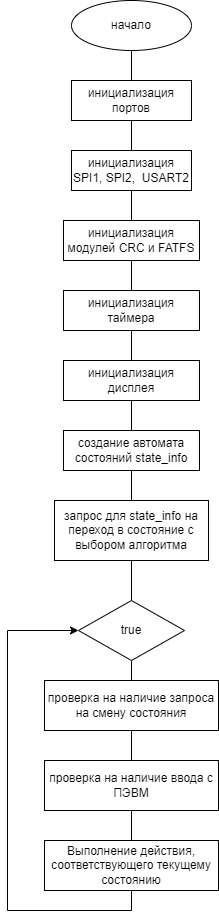


Рисунок 20 – Функция main

### 1.4.2 Детализация и пояснение основных функций

Рассмотрим функции из основного цикла выполнения программы. Схема алгоритма этих функций представлена на рисунке 21.

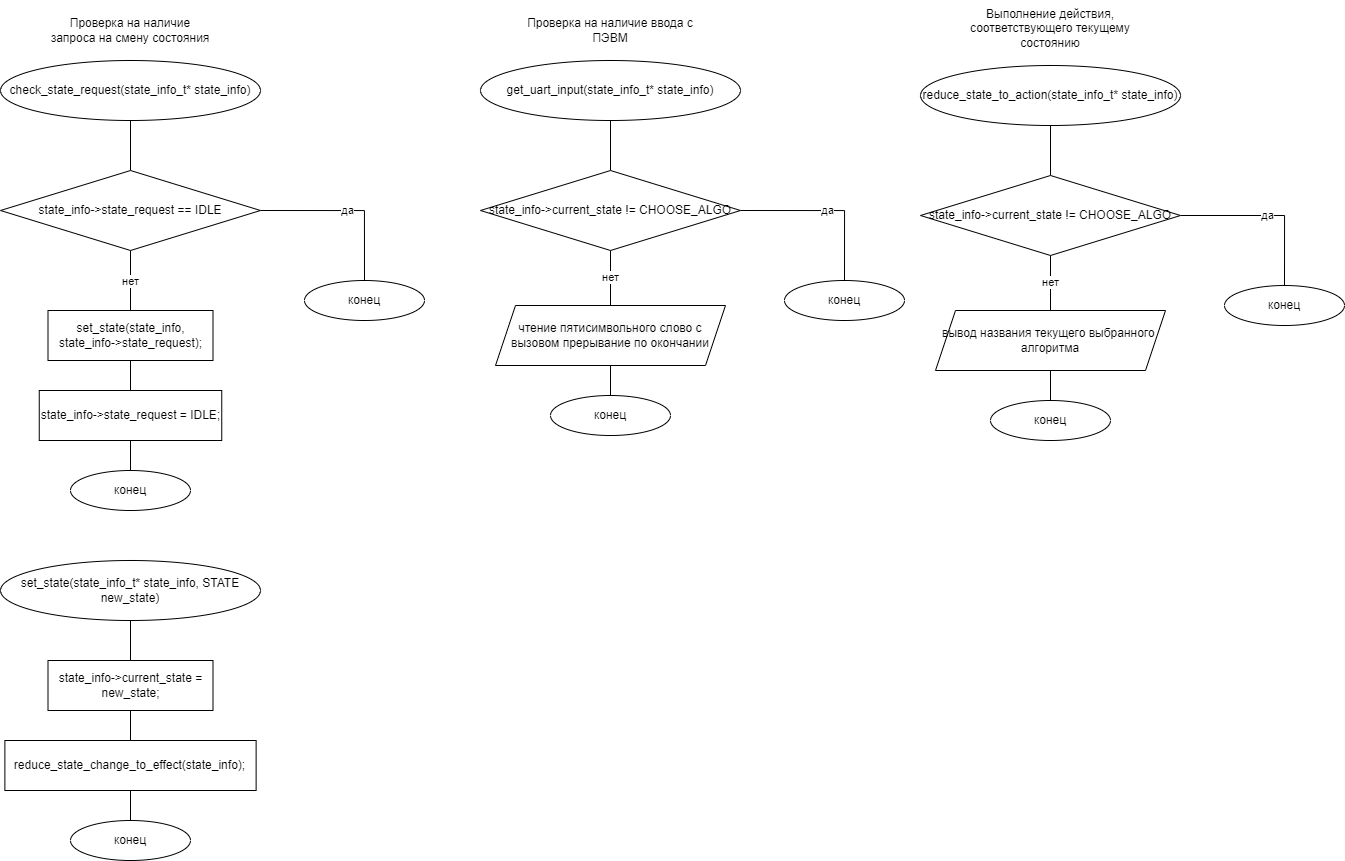


Рисунок 21 – Схема алгоритма функций в основном цикле

Основная логика программы находится в так называемых «эффектах» - одноразовых действиях, не связанными с текущем 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 22.

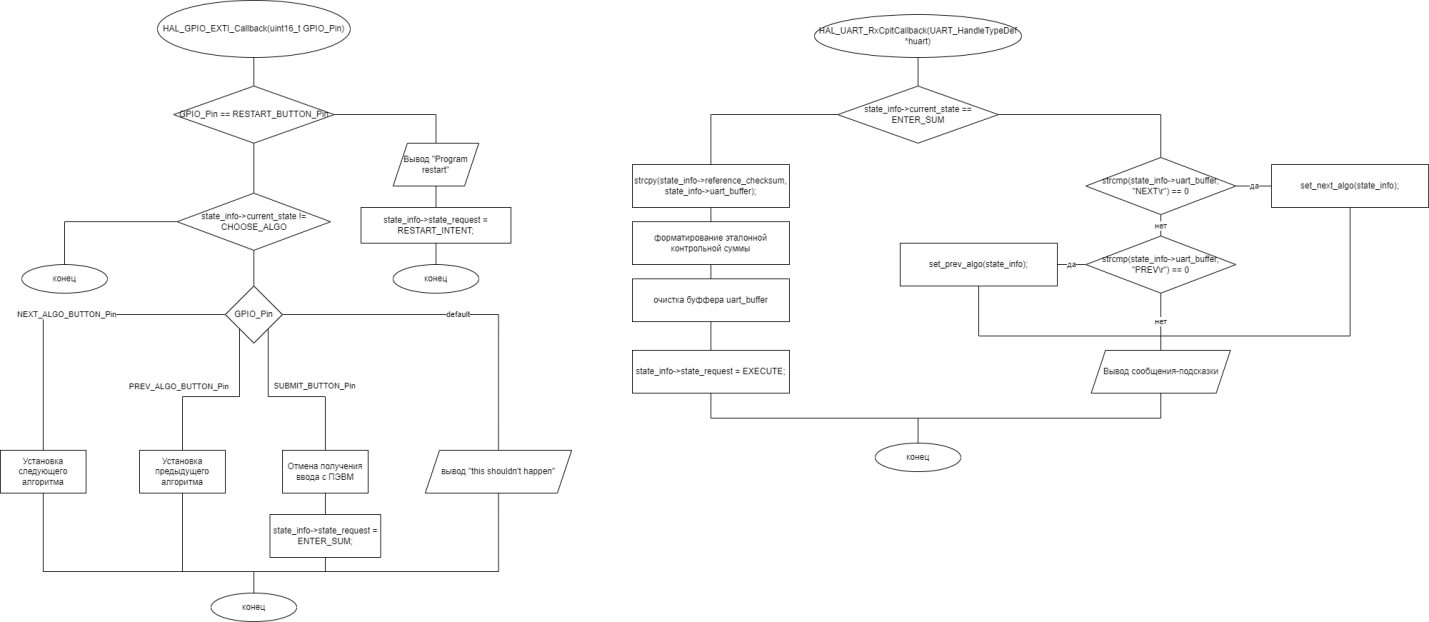


Рисунок 22 – Схема алгоритма работы прерываний

Наконец, рассмотрим само выполнение эффектов. Схема алгоритма работы эффектов представлена на рисунке 23.

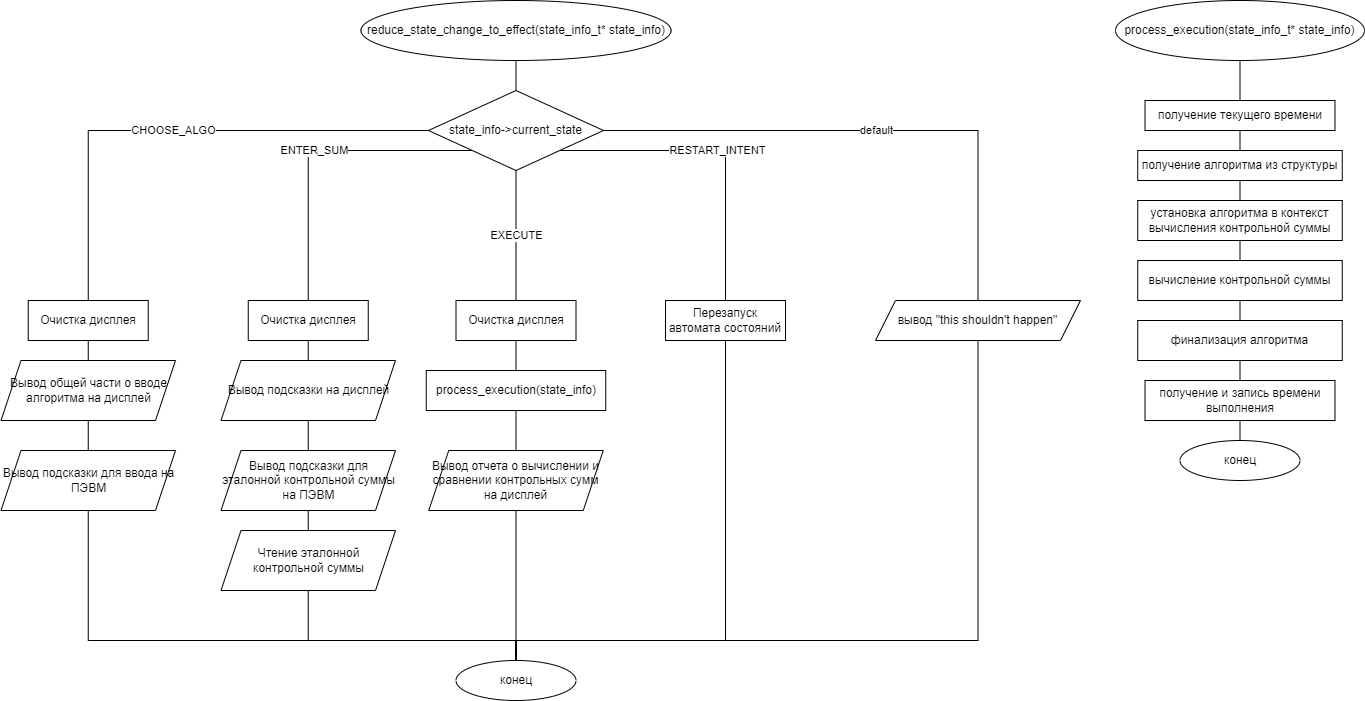


Рисунок 23 – Схема алгоритма работы эффектов

### 

# 2 Технологическая часть

Для реализации работы модели перекрестка была написана программа на языке Си, 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 видно, как ведут себя светодиоды, то есть как МК выполняет заложенный в него заранее написанный алгоритм.

При написании кода были использованы следующие библиотеки:

* avr/io.h – это библиотека ввода/вывода, которая объяснит компилятору какие порты ввода/вывода есть у микроконтроллера, как они обозначены и на что они способны;
* util/delay.h – это библиотека, позволяющая вызывать задержки (delay). При вызове \_delay\_ms в качестве параметра передается время в миллисекундах. В программе используется после чтения данных, перед их выводом, чтобы МК успел обработать полученную информацию;
* stdlib.h – это стандартная библиотека C общего назначения, включающая в себя функции, занимающиеся распределением памяти, управлением процессами, преобразованием и др.;
* util/atomic.h – это библиотека, предоставляющая набор атомарных операций (то есть которые выполнятся либо полностью, либо не выполнятся вообще). Была использована функция ATOMIC\_BLOCK с параметром ATOMIC\_RESTORESTATE (запоминание состояние текущей операции – регистра SREG, и его восстановления после выполнения атомарной операции), чтобы операции выполнялись без возможности их прерывания. Использовалась функция в случаях, когда:
  + увеличивалось системное время;
  + устанавливалось время работы фазы;
  + проверка окончания работы фазы.
* stdbool.h – это библиотека, которая позволяет работать с данными типа bool;

Так же была задействована системная функция ISR – прерывание таймера. Функция вызывается, когда таймер Т0 переполняется. В коде в качестве параметра функции указывается то, с каким прерыванием будет происходить работа – TIMER0\_OVF\_vect, то есть с прерыванием по переполнению таймера T0 [10].

После компиляции создается файл с расширением “.hex”, объем которого равен 5,4 килобайта – столько занимает скомпилированная программа.

По итогу отладки и тестирования, результатом стала функционирующая модель системы перекрестка, работающая в соответствии с ТЗ. Симуляция системы описана в разделе *2.3*.

## 2.2 Настройка таймера и работа фаз светофора

МК работает при частоте 8 МГц, предделитель установлен на 1024, то есть таймер будет переполняться в 1024 раза медленнее, чем работает МК. В написанной программе таймер обновляется при переполнении. Так как максимальное число 255, в таймер Т0 загружается число 248, чтобы до обновления оставалось 8 “тиков”. И с учетом частоты МК и предделителя, таймер будет увеличиваться на ~1 тик в секунду (8 МГц / 1024 = ~8000 и 8000 / 8 = 1000 миллисекунд). Это позволяет удобно отсчитывать время работы фаз системы светофора.

В начале работы системы создаем переменную TimeMS, отвечающую за время внутри системы (в миллисекундах), и обнуляем её. Когда происходит отчет времени какой-либо из фаз, в переменную Timer присваивается текущее время системы и добавляется время работы фазы. И происходит зацикленное сравнение текущего времени системы TimeMS со временем окончания работы фазы Timer до тех пор, пока TimeMS не станет больше Timer.

## 2.3 Симуляция работы системы

Для имитации реальных условий была использована программа Proteus. Схема контроллера модели перекрестка с пешеходной регулировкой показана на рисунке 16. Работа схемы, в данном случае первая фаза – зеленый свет горит для автомобилей на светофорном блоке 2 и 3, а для всех остальных – красный, показана на рисунке 17.

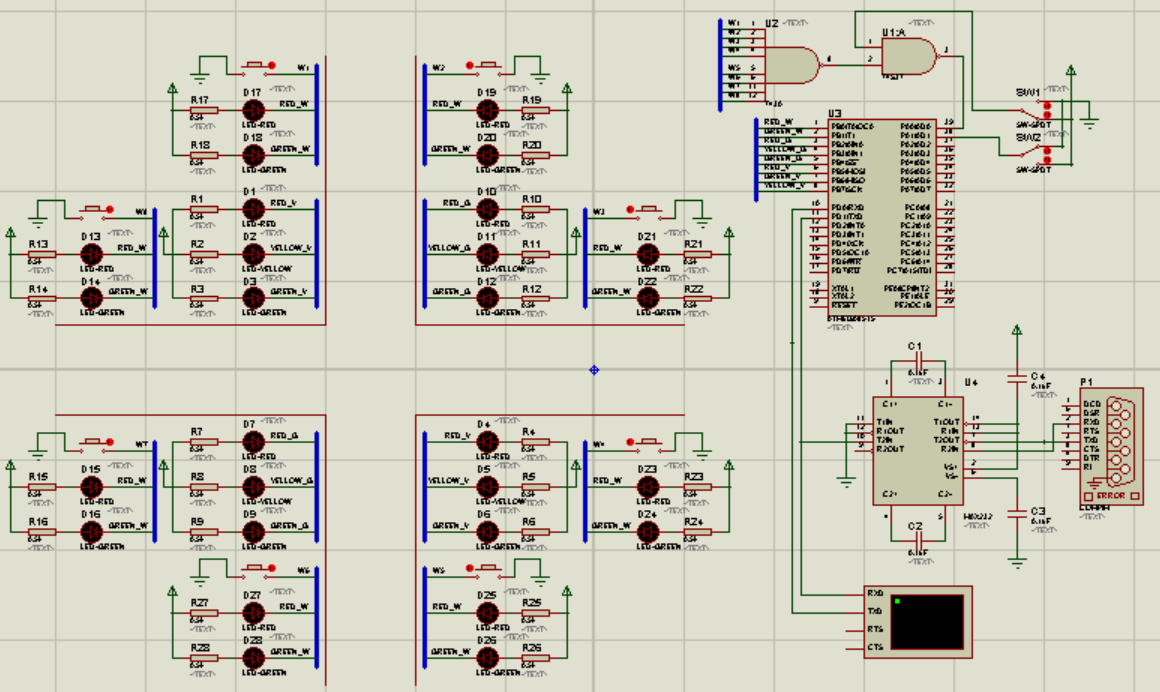


Рисунок 16 – Модель перекрестка

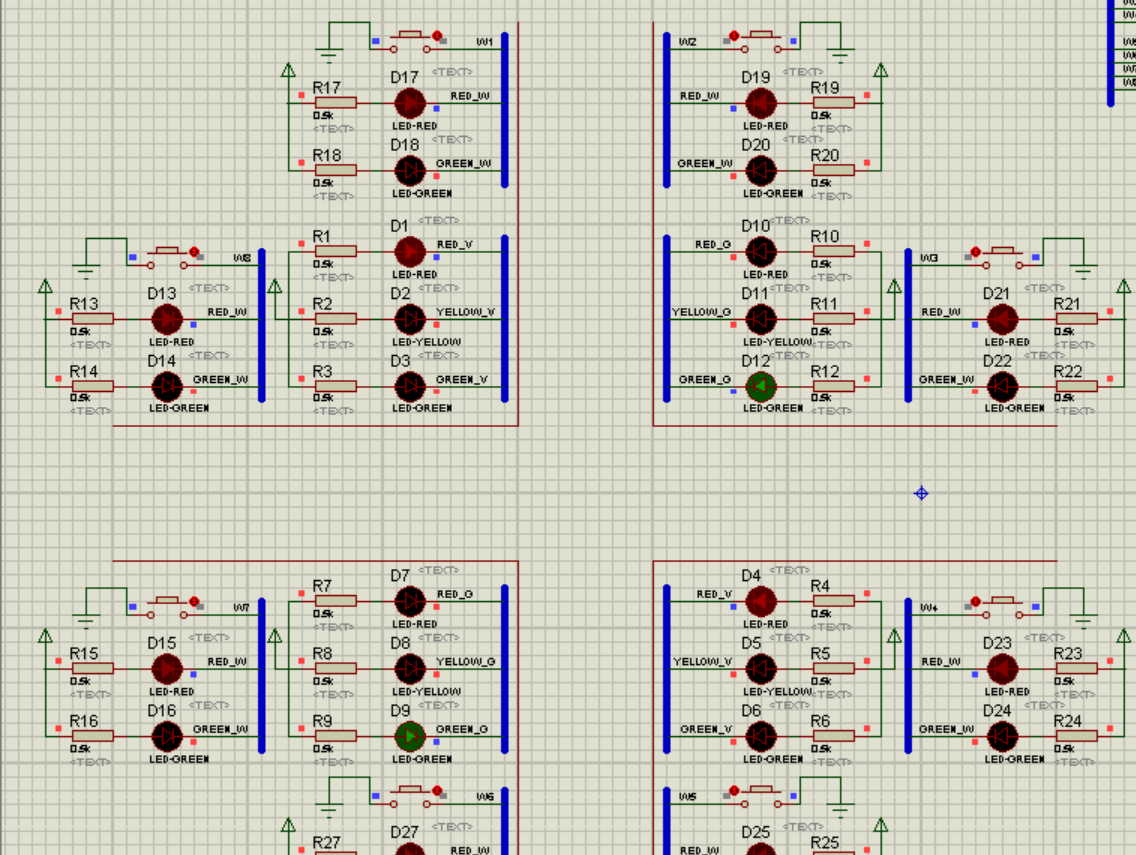


Рисунок 17 – Работа схемы (фаза 1)

Модель в proteus отличается от принципиальной схемы отсутствием транзисторов, так как в симуляции подается стабильные 5 вольт 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 UART.

При запуске системы открывается Virtual Terminal, в который последовательно вводятся 8 чисел, как показано на рисунке 18. После каждого введенного числа, оно передается обратно, для проверки, что передача произошла правильно и выводится сообщение “-ok”, означающее, что передача закончена и можно вводить следующее число.



Рисунок 18 – Ввод чисел

Для проверки передачи снимем показатели с осциллографа, они показаны на рисунке 19 – 20. На рисунке 19 видно передачу цифры 1 в кодировке, соответствующей таблице ASCII – 00110010, которая передается первой при вводе. Ввод начинается с start бита, равного 0 и stop бита, равного 1. Диаграмма, показанная на рисунке 19, которую выдает осциллограф, показывает корректность передаваемых данных.

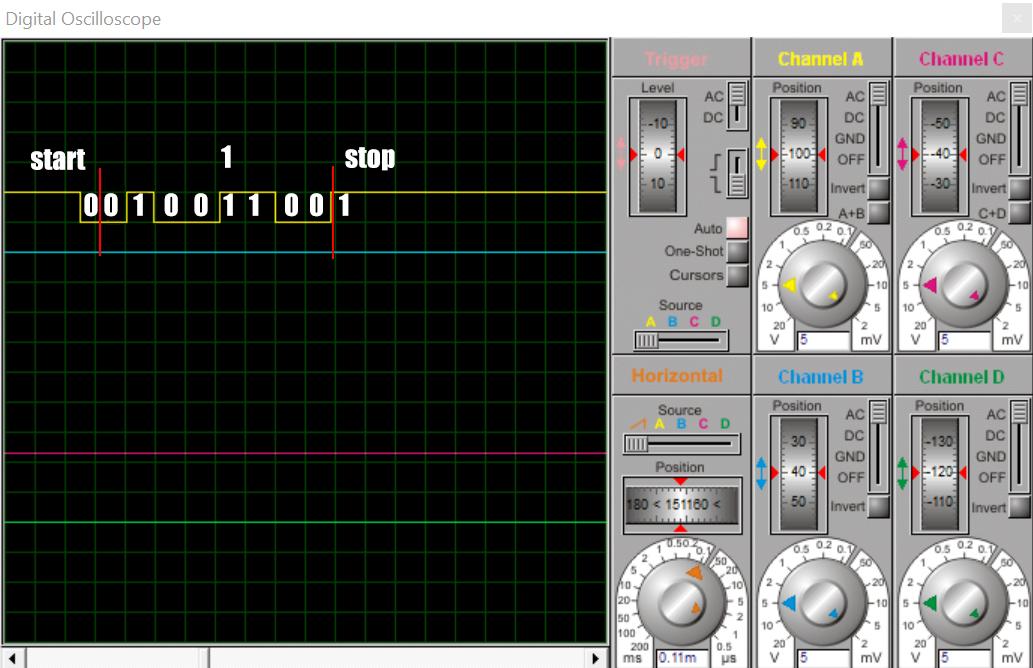


Рисунок 19 – Показатели осциллографа

На рисунке 20 видно передачу данных с МК в MAX232 – синий отрезок и данные, вышедшие из MAX232 – бордовый отрезок. Происходит передача числа 10. Сначала поступает start бит, равный 0, потом 1 – 00110010, а потом 0 – 00110000 по таблице ASCII и заканчивает передачу stop бит, равный 1. Диаграмма, показанная на рисунке 20, которую выдает осциллограф, показывает корректность передаваемых данных.

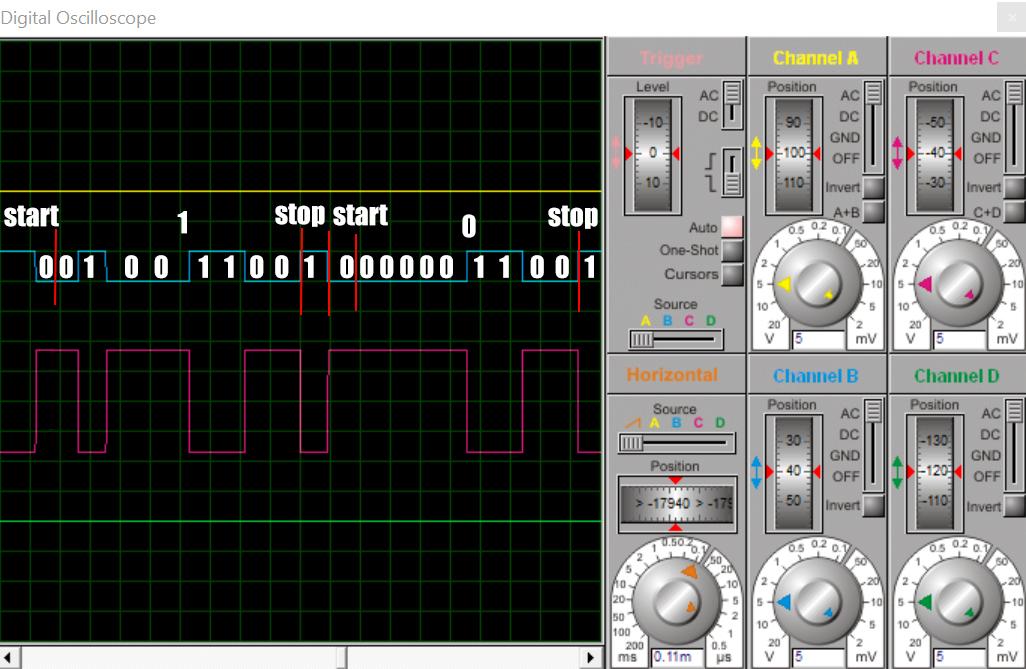


Рисунок 20 – Показатели осциллографа

RS232 основан на TTL-логике, то есть нулевому биту соответствует нулевой уровень напряжения, а единице уровень в +5 В. Стандарт RS232 использует более высокий уровень напряжения, до 15 В, и единице соответствует -15 В, а нулю +15 В. Поэтому выходной – бордовый сигнал выглядит инвертированным относительно входного синего.

## 2.4 Способы программирования МК

После написания и тестирования кода в программе, в который все – это виртуальная модель, идет этап загрузки файла (с расширением hex – бинарный файл) в микроконтроллер. Это может выполнятся следующими способами [12]:

* внутрисхемное программирование (ISP – In-System Programming);
* параллельное высоковольтное программирование;
* через JTAG;
* через Bootloader;
* Pinboard II.

Выбрана прошивка – “внутрисхемное программирование” через канал SPI, так как это простой и популярный метод, с которым уже было знакомство на практике. Программирование МК происходит через программатор и одноименный разъем, о котором было рассказано в разделе *1.3.1*. Визуальное представление показано на рисунке 21.

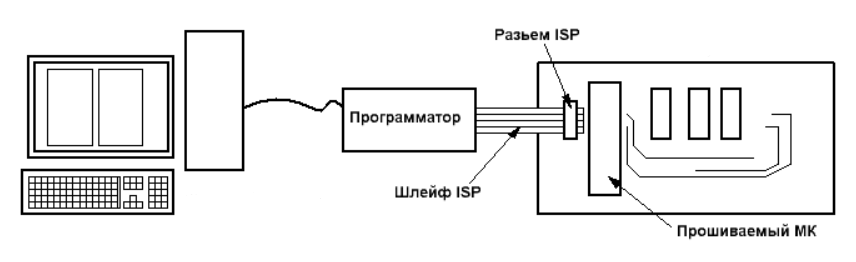


Рисунок 21 – Программирование МК

Прошивка проходит по интерфейсу SPI, для работы программатора нужно 4 контакта и питание (достаточно только земли, чтобы уравнять потенциалы земель программатора и устройства):

* MISO – Master-Input/Slave-Output – данные, идущие от контроллера;
* MOSI – Master-Output/Slave-Input – данные идущие в контроллер;
* SCK – тактовые импульсы интерфейса SPI;
* RESET – сигналом на RESET программатор вводит контроллер в режим программирования;
* GND – земля;
* VCC – питание.

Взаимодействие устройств по интерфейсу SPI требует установки одного из устройств в режим ведущего, а остальных – в режим ведомого. При этом ведущее устройство отвечает за выбор ведомого и инициализацию передачи.

Передача по SPI осуществляется в полнодуплексном режиме, по одному биту за такт в каждую сторону. По возрастающему фронту сигнала SCK ведомое устройство считывает очередной бит с линии MOSI, а по спадающему – выдает следующий бит на линию MISO.

В МК передается бинарный файл с расширением “.hex” 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. Микроконтроллеры 8051, PIC, AVR и ARM: отличия и особенности [Электронный ресурс]. – Режим доступа: <http://digitrode.ru/computing-devices/mcu_cpu/1253-mikrokontrollery-8051-pic-avr-i-arm-otlichiya-i-osobennosti.html> (дата обращения: 13.09.2021)

5. MAX232x Dual EIA-232 Drivers/Receivers datasheet [Электронный ресурс]. – Режим доступа: <https://www.ti.com/lit/ds/symlink/max232.pdf> (дата обращения: 16.10.2021)

8. ГОСТ 2.743-91 ЕСКД ОБОЗНАЧЕНИЯ БУКВЕННО-ЦИФРОВЫЕ В ЭЛЕКТРИЧЕСКИХ СХЕМАХ [Электронный ресурс]. – Режим доступа: <https://docs.cntd.ru/document/1200001985> (дата обращения: 05.10.2021)

9. ГОСТ 2.721-74 ЕСКД ОБОЗНАЧЕНИЯ УСЛОВНЫЕ ГРАФИЧЕСКИЕ В СХЕМАХ [Электронный ресурс]. – Режим доступа: <https://docs.cntd.ru/document/1200007058> (дата обращения: 05.10.2021)

# 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  **#ifdef** ST7735S\_1\_8\_DEFAULT\_ORIENTATION  **#define** ST7735\_IS\_160X128 1  **#define** ST7735\_WIDTH 128  **#define** ST7735\_HEIGHT 160  **#define** ST7735\_XSTART 2  **#define** ST7735\_YSTART 1  **#define** ST7735\_DATA\_ROTATION 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, 0xc1bdceee,  0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,  0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,  0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,  0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,  0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,  0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,  0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,  0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,  0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,  0x289b7ec6, 0xeaa127fa, 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,  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 = 0x1FFF;  /\* 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, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // "  0x0000, 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, 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, 0x0000, 0x7F80, 0x7F80, 0x0000, 0x0000, 0x7F80, 0x7F80, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // =  0x0000, 0x0000, 0x0000, 0x0000, 0x4000, 0x7000, 0x1C00, 0x0700, 0x0180, 0x0700, 0x1C00, 0x7000, 0x4000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // >  0x0000, 0x1F00, 0x3F80, 0x71C0, 0x60C0, 0x00C0, 0x01C0, 0x0380, 0x0700, 0x0E00, 0x0C00, 0x0C00, 0x0000, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000, // ?  0x0000, 0x1E00, 0x3F00, 0x3180, 0x7180, 0x6380, 0x6F80, 0x6D80, 0x6D80, 0x6F80, 0x6780, 0x6000, 0x3200, 0x3E00, 0x1C00, 0x0000, 0x0000, 0x0000, // @  0x0000, 0x0E00, 0x0E00, 0x1B00, 0x1B00, 0x1B00, 0x1B00, 0x3180, 0x3180, 0x3F80, 0x3F80, 0x3180, 0x60C0, 0x60C0, 0x60C0, 0x0000, 0x0000, 0x0000, // A  0x0000, 0x7C00, 0x7E00, 0x6300, 0x6300, 0x6300, 0x6300, 0x7E00, 0x7E00, 0x6300, 0x6180, 0x6180, 0x6380, 0x7F00, 0x7E00, 0x0000, 0x0000, 0x0000, // B  0x0000, 0x1E00, 0x3F00, 0x3180, 0x6180, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6180, 0x3180, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, // C  0x0000, 0x7C00, 0x7F00, 0x6300, 0x6380, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6300, 0x6300, 0x7E00, 0x7C00, 0x0000, 0x0000, 0x0000, // D  0x0000, 0x7F80, 0x7F80, 0x6000, 0x6000, 0x6000, 0x6000, 0x7F00, 0x7F00, 0x6000, 0x6000, 0x6000, 0x6000, 0x7F80, 0x7F80, 0x0000, 0x0000, 0x0000, // E  0x0000, 0x7F80, 0x7F80, 0x6000, 0x6000, 0x6000, 0x6000, 0x7F00, 0x7F00, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x0000, 0x0000, 0x0000, // F  0x0000, 0x1E00, 0x3F00, 0x3180, 0x6180, 0x6000, 0x6000, 0x6000, 0x6380, 0x6380, 0x6180, 0x6180, 0x3180, 0x3F80, 0x1E00, 0x0000, 0x0000, 0x0000, // G  0x0000, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x7F80, 0x7F80, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x0000, 0x0000, 0x0000, // H  0x0000, 0x3F00, 0x3F00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x3F00, 0x3F00, 0x0000, 0x0000, 0x0000, // I  0x0000, 0x0180, 0x0180, 0x0180, 0x0180, 0x0180, 0x0180, 0x0180, 0x0180, 0x0180, 0x6180, 0x6180, 0x7380, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, // J  0x0000, 0x60C0, 0x6180, 0x6300, 0x6600, 0x6600, 0x6C00, 0x7800, 0x7C00, 0x6600, 0x6600, 0x6300, 0x6180, 0x6180, 0x60C0, 0x0000, 0x0000, 0x0000, // K  0x0000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x7F80, 0x7F80, 0x0000, 0x0000, 0x0000, // L  0x0000, 0x71C0, 0x71C0, 0x7BC0, 0x7AC0, 0x6AC0, 0x6AC0, 0x6EC0, 0x64C0, 0x60C0, 0x60C0, 0x60C0, 0x60C0, 0x60C0, 0x60C0, 0x0000, 0x0000, 0x0000, // M  0x0000, 0x7180, 0x7180, 0x7980, 0x7980, 0x7980, 0x6D80, 0x6D80, 0x6D80, 0x6580, 0x6780, 0x6780, 0x6780, 0x6380, 0x6380, 0x0000, 0x0000, 0x0000, // N  0x0000, 0x1E00, 0x3F00, 0x3300, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x3300, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, // O  0x0000, 0x7E00, 0x7F00, 0x6380, 0x6180, 0x6180, 0x6180, 0x6380, 0x7F00, 0x7E00, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x0000, 0x0000, 0x0000, // P  0x0000, 0x1E00, 0x3F00, 0x3300, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6580, 0x6780, 0x3300, 0x3F80, 0x1E40, 0x0000, 0x0000, 0x0000, // Q  0x0000, 0x7E00, 0x7F00, 0x6380, 0x6180, 0x6180, 0x6380, 0x7F00, 0x7E00, 0x6600, 0x6300, 0x6300, 0x6180, 0x6180, 0x60C0, 0x0000, 0x0000, 0x0000, // R  0x0000, 0x0E00, 0x1F00, 0x3180, 0x3180, 0x3000, 0x3800, 0x1E00, 0x0700, 0x0380, 0x6180, 0x6180, 0x3180, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, // S  0x0000, 0xFFC0, 0xFFC0, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000, // T  0x0000, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x7380, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, // U  0x0000, 0x60C0, 0x60C0, 0x60C0, 0x3180, 0x3180, 0x3180, 0x1B00, 0x1B00, 0x1B00, 0x1B00, 0x0E00, 0x0E00, 0x0E00, 0x0400, 0x0000, 0x0000, 0x0000, // V  0x0000, 0xC0C0, 0xC0C0, 0xC0C0, 0xC0C0, 0xC0C0, 0xCCC0, 0x4C80, 0x4C80, 0x5E80, 0x5280, 0x5280, 0x7380, 0x6180, 0x6180, 0x0000, 0x0000, 0x0000, // W  0x0000, 0xC0C0, 0x6080, 0x6180, 0x3300, 0x3B00, 0x1E00, 0x0C00, 0x0C00, 0x1E00, 0x1F00, 0x3B00, 0x7180, 0x6180, 0xC0C0, 0x0000, 0x0000, 0x0000, // X  0x0000, 0xC0C0, 0x6180, 0x6180, 0x3300, 0x3300, 0x1E00, 0x1E00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000, // Y  0x0000, 0x3F80, 0x3F80, 0x0180, 0x0300, 0x0300, 0x0600, 0x0C00, 0x0C00, 0x1800, 0x1800, 0x3000, 0x6000, 0x7F80, 0x7F80, 0x0000, 0x0000, 0x0000, // Z  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, // `  0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1F00, 0x3F80, 0x6180, 0x0180, 0x1F80, 0x3F80, 0x6180, 0x6380, 0x7F80, 0x38C0, 0x0000, 0x0000, 0x0000, // a  0x0000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6E00, 0x7F00, 0x7380, 0x6180, 0x6180, 0x6180, 0x6180, 0x7380, 0x7F00, 0x6E00, 0x0000, 0x0000, 0x0000, // b  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, 0x6180, 0x6180, 0x6180, 0x7380, 0x3F80, 0x1D80, 0x0000, 0x0000, 0x0000, // d  0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1E00, 0x3F00, 0x7300, 0x6180, 0x7F80, 0x7F80, 0x6000, 0x7180, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, // e  0x0000, 0x07C0, 0x0FC0, 0x0C00, 0x0C00, 0x7F80, 0x7F80, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000, // f  0x0000, 0x0000, 0x0000, 0x0000, 0x1D80, 0x3F80, 0x7380, 0x6180, 0x6180, 0x6180, 0x6180, 0x7380, 0x3F80, 0x1D80, 0x0180, 0x6380, 0x7F00, 0x3E00, // g  0x0000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6F00, 0x7F80, 0x7180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x0000, 0x0000, 0x0000, // h  0x0000, 0x0600, 0x0600, 0x0000, 0x0000, 0x3E00, 0x3E00, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0000, 0x0000, 0x0000, // i  0x0600, 0x0600, 0x0000, 0x0000, 0x3E00, 0x3E00, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x4600, 0x7E00, 0x3C00, // j  0x0000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6180, 0x6300, 0x6600, 0x6C00, 0x7C00, 0x7600, 0x6300, 0x6300, 0x6180, 0x60C0, 0x0000, 0x0000, 0x0000, // k  0x0000, 0x3E00, 0x3E00, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0000, 0x0000, 0x0000, // l  0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xDD80, 0xFFC0, 0xCEC0, 0xCCC0, 0xCCC0, 0xCCC0, 0xCCC0, 0xCCC0, 0xCCC0, 0xCCC0, 0x0000, 0x0000, 0x0000, // m  0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x6F00, 0x7F80, 0x7180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x0000, 0x0000, 0x0000, // n  0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1E00, 0x3F00, 0x7380, 0x6180, 0x6180, 0x6180, 0x6180, 0x7380, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000, // o  0x0000, 0x0000, 0x0000, 0x0000, 0x6E00, 0x7F00, 0x7380, 0x6180, 0x6180, 0x6180, 0x6180, 0x7380, 0x7F00, 0x6E00, 0x6000, 0x6000, 0x6000, 0x6000, // p  0x0000, 0x0000, 0x0000, 0x0000, 0x1D80, 0x3F80, 0x7380, 0x6180, 0x6180, 0x6180, 0x6180, 0x7380, 0x3F80, 0x1D80, 0x0180, 0x0180, 0x0180, 0x0180, // q  0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x6700, 0x3F80, 0x3900, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x0000, 0x0000, 0x0000, // r  0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1E00, 0x3F80, 0x6180, 0x6000, 0x7F00, 0x3F80, 0x0180, 0x6180, 0x7F00, 0x1E00, 0x0000, 0x0000, 0x0000, // s  0x0000, 0x0000, 0x0800, 0x1800, 0x1800, 0x7F00, 0x7F00, 0x1800, 0x1800, 0x1800, 0x1800, 0x1800, 0x1800, 0x1F80, 0x0F80, 0x0000, 0x0000, 0x0000, // t  0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6380, 0x7F80, 0x3D80, 0x0000, 0x0000, 0x0000, // u  0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x60C0, 0x3180, 0x3180, 0x3180, 0x1B00, 0x1B00, 0x1B00, 0x0E00, 0x0E00, 0x0600, 0x0000, 0x0000, 0x0000, // v  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, // VGH25 = 2.4C VGSEL = -10 VGH = 3 \* AVDD  ST7735\_PWCTR3, 2, // 9: Power control, 2 args, no delay:  0x0A, // Opamp current small  0x00, // Boost frequency  ST7735\_PWCTR4, 2, // 10: Power control, 2 args, no delay:  0x8A, // 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.с

|  |
| --- |
| **#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\_Pin;  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

Графическая часть

На 2 листах

Электрическая схема функциональная

Электрическая схема принципиальная

# Приложение В

Перечень элементов

На 3 листах