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:

Модуль хэширования данных

Студент	_____ (Группа)	_____ (Подпись, дата)	М.В. Смородина (И.О. Фамилия)
Руководитель		_____ (Подпись, дата)	И.Б. Трамов (И.О. Фамилия)

2023 г.

тут вставлю задание

РЕФЕРАТ

РПЗ 97 страниц, 29 рисунков, 7 таблиц, 11 источников, 2 приложения.

МИКРОКОНТРОЛЛЕР, ХЭШИРОВАНИЕ, ХЭШ-ФУНКЦИЯ, АЛГОРИТМЫ, SHA256, CRC16/CCITT-FALSE, MD5

Объектом разработки является МК-система для хэширования данных.

Цель работы - создание системы хэширования с возможностью выбора алгоритма хэширования, модель устройства и разработка необходимой документации.

Поставленная цель достигается при помощи Proteus 8.13 и STM32CubeIDE.

В процессе работы над курсовым проектом решаются 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.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ, ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

МПС – Микропроцессорные системы.

ЖК-дисплей – жидкокристаллический дисплей.

МК – микроконтроллер.

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

ТЗ – техническое задание.

MD5, CRC – алгоритмы для вычисления контрольной суммы.

SHA256 - алгоритм хэширования.

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

SPI (Serial Peripheral Interface) – интерфейс для взаимодействия МК с внешними устройствами.

УГО – условное графическое обозначение.

ВВЕДЕНИЕ

В данной курсовой работе производится разработка МК-системы хэширования данных.

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, виртуального терминала, необходимого для ввода данных для последующего хэширования, ЖК-дисплея для вывода результатов выполнения операций, ошибок, возникших во время выполнения программы, 3 кнопок, с помощью которых производится управление выбором алгоритма хэширования, 1 кнопки, с помощью которой определяется, осуществляется ввод с телефона или ПЭВМ, преобразователя UART-USB, коннектора USB.

Актуальность разрабатываемой модели хэширования данных состоит в том, что в современном мире любой ресурс обменивается данными с другими ресурсами: например, клиентская часть интернет-приложения с серверной частью или один сервер с другим. В момент передачи злоумышленники могут отловить и прочесть передаваемые данные. Чтобы избежать подобного, данные перед отправлением при помощи специальных алгоритмов превращают в нечитаемый для человека набор символов, а при получении расшифровывают их. Разрабатываемая система позволяет шифровать данные при помощи различных алгоритмов, что позволит безопасно и надежно передавать информацию.

1 Конструкторская часть

1.1 Анализ требований и принцип работы системы

Исходя из требований, изложенных в техническом задании, необходимо осуществить разработку системы, хэширующих данные, вводимые оператором, с возможностью выбора алгоритма хэширования и выбором места осуществления ввода - ПЭВМ или телефон.

Система хэширования данных можно разделить на следующие фазы, представленные в таблице:

Таблица 1 - Фазы работы системы.

Фаза	Описание
Фаза 0	Система приходит в состояние работоспособности
Фаза 1	Система готова принимать данные с ПЭВМ или телефона, на ЖК-дисплей выводится подсказка о том, что система ожидает данные на вход
Фаза 2	Система прочитала вводимые данные, с помощью кнопок NEXT, PREV и CHOOSE система ожидает выбора алгоритма хэширования. Текущий алгоритм высвечивается на ЖК-дисплее
Фаза 3	Выполнение хэширования согласно выбранному алгоритму, подсказки о текущих этапах и результатах выполнения функций выводятся на ЖК-дисплей
Фаза 4	Система вновь готова считывать данные, МК очищает ранее использованные переменные

В системе реализованы 3 алгоритма хэширования - CRC16/CCITT, MD5 и SHA256. Результаты хэширования, полученные после выполнения всех алгоритмов выводятся как на ЖК-дисплей, так и отправляются в ПЭВМ. Однако, из-за того, что каждый алгоритм получает в результате строку разной длины, вывод организован для каждого алгоритма по-разному.

Таблица 2 - Длины полученных в результате захэшированных строк.

Алгоритм	Длина хэшированной строки
CRC16/CCITT	4
MD5	16
SHA256	32

На рисунке 1 представлена структурная схема устройства хэширования данных.

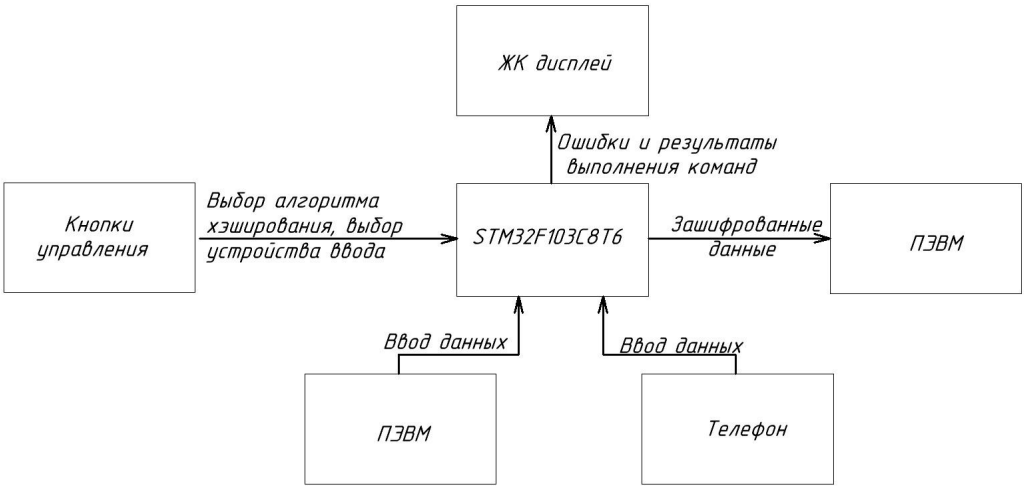


Рисунок 1 - Структурная схема устройства

1.2 Проектирование функциональной схемы

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

1.2.1 Микроконтроллер STM32F103C8T6

В данной курсовой работе используется микроконтроллер STM32F103C8T6. Этот МК принадлежит семейству микроконтроллеров STM32 - семейство 32-битных микроконтроллеров, использующий в качестве ядра ARM, производятся STMicroelecronics. Микроконтроллеры данного семейства группируются в серии в зависимости от конкретного ядра ARM.

Существует несколько основных семейств микроконтроллеров[2]:

- 8051 – это 8-битное семейство МК от компании Intel.
- PIC – это серия МК, разработанная компанией Microchip;
- AVR – это серия МК разработанная компанией Atmel;
- ARM – одним из семейств процессоров на базе архитектуры RISC, разработанным компанией Advanced RISC Machines.

Сравнение этих 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	4 такта на инструкцию	1 такт на инструкцию	1 такт на инструкцию

	инструкцию			
Память	ROM, SRAM, FLASH	SRAM, FLASH	Flash, SRAM, EEPROM	Flash, SDRAM, EEPROM
Энергопотребление	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 быстрой работе алгоритмов, необходимо наличие либо несколько портов UART, либо порта UART и USB для возможности системы принимать данные на вход как с устройства ПЭВМ, так и с телефона.

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

1. STM32, имеющие следующие характеристики:

- Flash-память до 2056 Кбайт;
- RAM до 1,4 Мбайт;
- Максимальная частота ядра до 480 МГц;
- число пинов (ножек) ввода-вывода 16–64;
- самый разнообразный набор периферии

2. NXP, имеющие следующие характеристики:

- FLASH до 2048 Кбайт;
- RAM до 8096 Кбайт;
- Максимальная частота ядра до 360 МГц;
- число пинов ввода-вывода 16-64;
- самый разнообразный набор периферии

3. Toshiba, имеющие следующие характеристики:

- FLASH до 1,5 Мбайт;
- RAM до 514 Кбайт;
- Максимальная частота ядра до 120 МГц;
- самый разнообразный набор периферии

Выберем подсемейство STM32 от STMicroelectronics, так как у них самая активная поддержка сообщества, высокая скорость вычислений, они хорошо масштабируются - на начальном этапе выполнения курсовой работы нельзя точно сказать, какой именно из алгоритмов хэширования будет реализован и сколько памяти этой займет, поэтому возможность переноса кода с одного микроконтроллера на другой без дополнительных трудностей будет плюсом. Кроме того, работа с этим подсемейством микроконтроллеров велась на курсе “Микропроцессорные системы”, что также является плюсом.

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

- наличие как USB, так и нескольких UART портов для осуществления ввода с устройства ПЭВМ и телефона;
- процессорное ядро ARM Cortex-M3 с максимальной тактовой частотой 72 МГц и контроллером прерываний NVIC (Nested vectored interrupt controller);
- возможность переноса кода без дополнительных сложностей;
- 64 Кбайта FLASH-памяти;
- наличие опыта работы с данным 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:

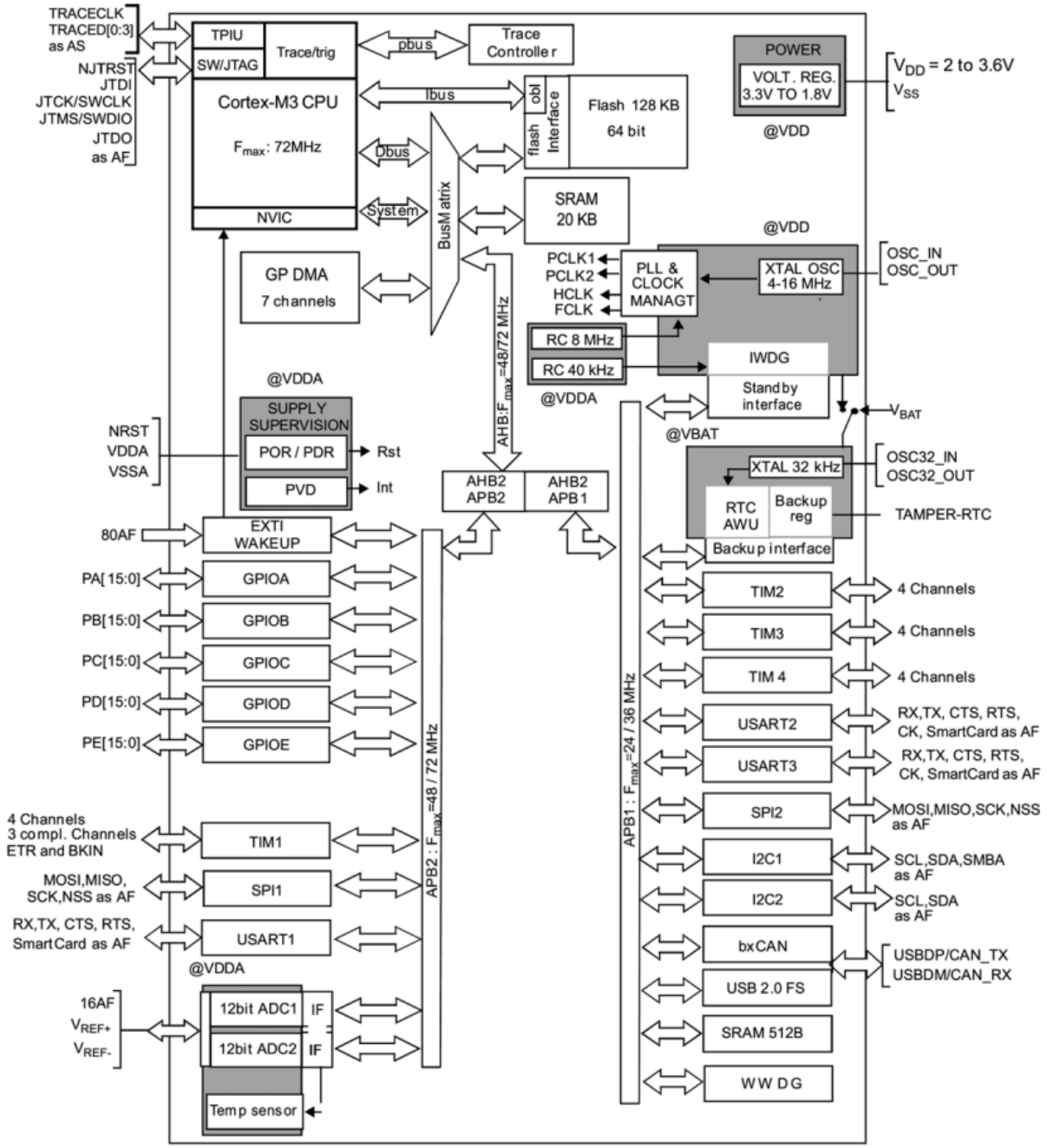


Рисунок 2 - Структурная схема STM32F103C8T6

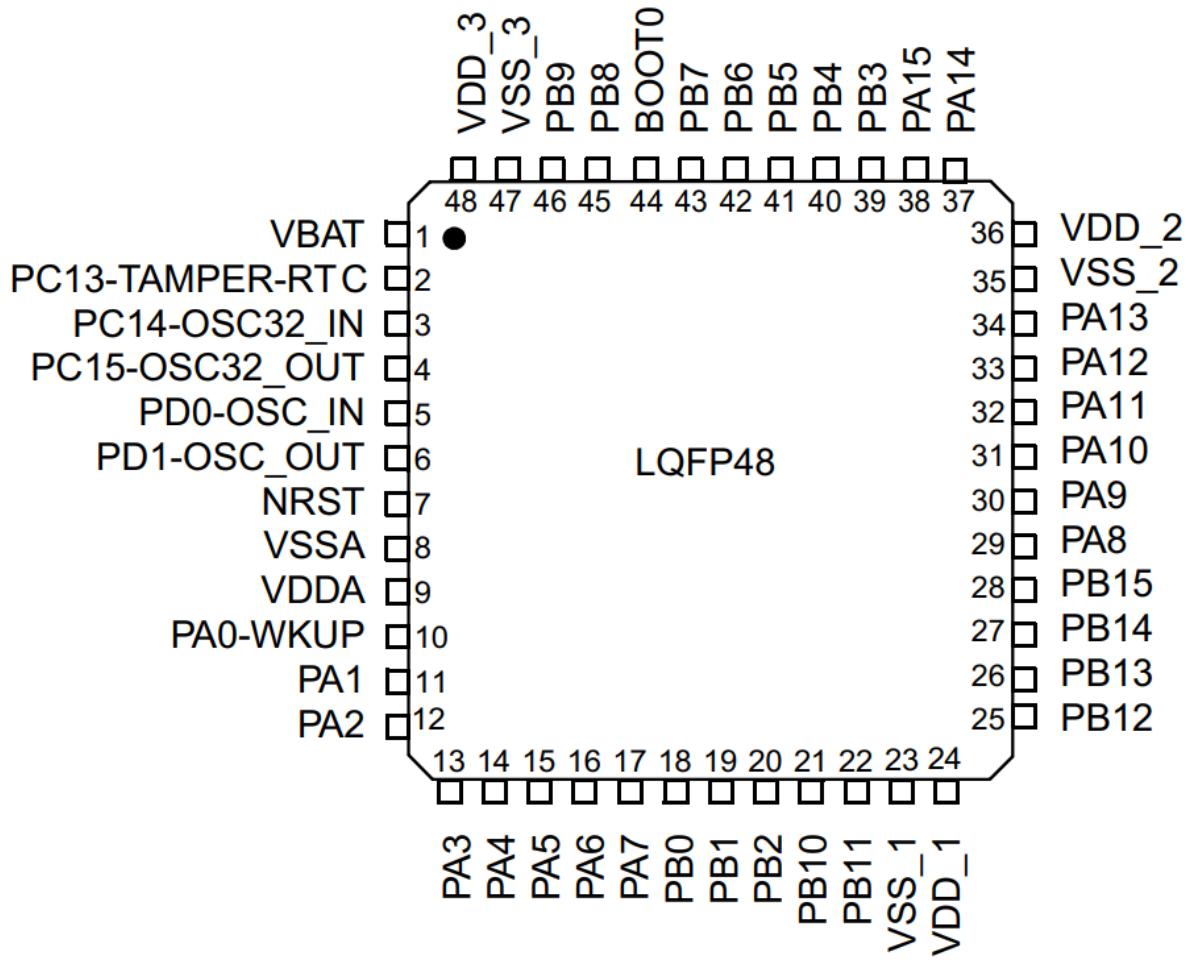


Рисунок 3 – УГО микроконтроллера STM32F103C8T6

Микроконтроллер STM32F103C8T6 включает в себя:

Он обладает следующими характеристиками:

1. Архитектура и производительность:

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

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

2. Память:

- 64 КБ флеш-памяти для программного кода.

- 20 КБ ОЗУ (SRAM) для хранения данных.

- Возможность расширения памяти с использованием внешних устройств.

3. Периферийные устройства:

- Несколько портов GPIO (General-Purpose Input/Output) для подключения и управления внешними устройствами.
- USART, SPI, I2C и другие интерфейсы для обмена данными с внешними устройствами.
- АЦП (аналогово-цифровой преобразователь) для измерения аналоговых сигналов.

4. Интерфейсы и коммуникации:

- USB-интерфейс для обмена данными с компьютером или другими устройствами.
- Возможность работы с различными протоколами связи, такими как CAN (Controller Area Network), Ethernet и другими.

5. Прочие особенности:

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

6. Программирование и разработка:

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

7. Применение:

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

8. Напряжение питания: 2– 3.6В.

1.2.1.1 Используемые элементы

Для работы устройства измерения скорости чтения использованы не все элементы архитектуры МК STM32F103C8T6. Среди использованных элементов и интерфейсов:

- Порты A, B – использованные пины и их назначение описано в пункте 1.2.1.2;
- Указатель стека – играет важную роль в организации стека, используемого для управления вызовами подпрограмм. Указатель стека используется для сохранения адреса возврата и регистров при вызове функций. Это обеспечивает корректный возврат из функций и поддерживает структуру вызовов функций;
- Регистры общего назначения – предназначены для хранения операндов арифметико-логических операций, а также адресов или отдельных компонентов адресов ячеек памяти;

- АЛУ – выполняет арифметические и логические операции, обеспечивает выполнение базовых математических операций и манипуляций с битами;
- Память SRAM – статическая память МК, хранящая объявленные переменные;
- Память Flash – память МК, хранящая загруженную в него программу;
- Программный счетчик – указывает на следующую по исполнению команду;
- Регистры команд – содержит исполняемую в настоящий момент команду(или следующую), то есть команду, адресуемую счетчиком команд;
- Декодер – выделяет код операции и операнды команды и далее вызывает микропрограмму, исполняющую данную команду;
- Сигналы управления – нужны для синхронизации обработки данных;
- Логика программирования – устанавливает логику того, как будет вшита программа в МК;
- Генератор – генератор тактовых импульсов. Необходим для синхронизации работы МК;

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

− Прерывания – обрабатывает внешние прерывания и прерывания периферийных устройств МК (таймеров, портов и т.д.). В устройстве используются прерывания с портов для обработки нажатия кнопок и прерывания UART;

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

− SPI (Serial Peripheral Interface) – интерфейс для связи МК с другими внешними устройствами. В устройстве используется для прошивки МК и вывода данных на жидкокристаллический дисплей;

− Таймеры – МК содержит в себе четыре 16-ти разрядных таймеров (TIM1, TIM2, TIM3, TIM4). В устройстве используется только один канал таймера TIM2 для генерации ШИМ сигнала для динамика.

1.2.1.2 Распределение портов

МК STM32F103C8T6 содержит в себе пять портов – A, B, C, D, E. В устройстве используются порты A и B.

Порт A используется для:

- PA2 - отправка данных по UART на ПЭВМ;
- PA3 - прием данных по UART от ПЭВМ;
- PA5 - кнопка CHOOSE, запускает выполнение текущего алгоритма (название алгоритма отображено на ЖК-дисплее);
- PA6 - кнопка NEXT, увеличивает id текущего алгоритма;

- PA7 - кнопка PREV, уменьшает id текущего алгоритма;
- PA9 - отправка данных по UART через преобразователь UART-USB на телефон;
- PA10 - прием данных по UART через преобразователь UART-USB от телефона;
- PA14 - кнопка COMP/PHONE, выбирает устройство ввода.

Порт B используется для:

- PB10-PB13 - пины используются для передачи данных на ЖК-дисплей;
- PB14 - используется для передачи Enable на ЖК-дисплей;
- PB15 - используется для выбора регистра на ЖК-дисплее.

1.2.1.3 Организация памяти

На рисунке 4 представлена карта памяти МК STM32F103C8T6 [3].

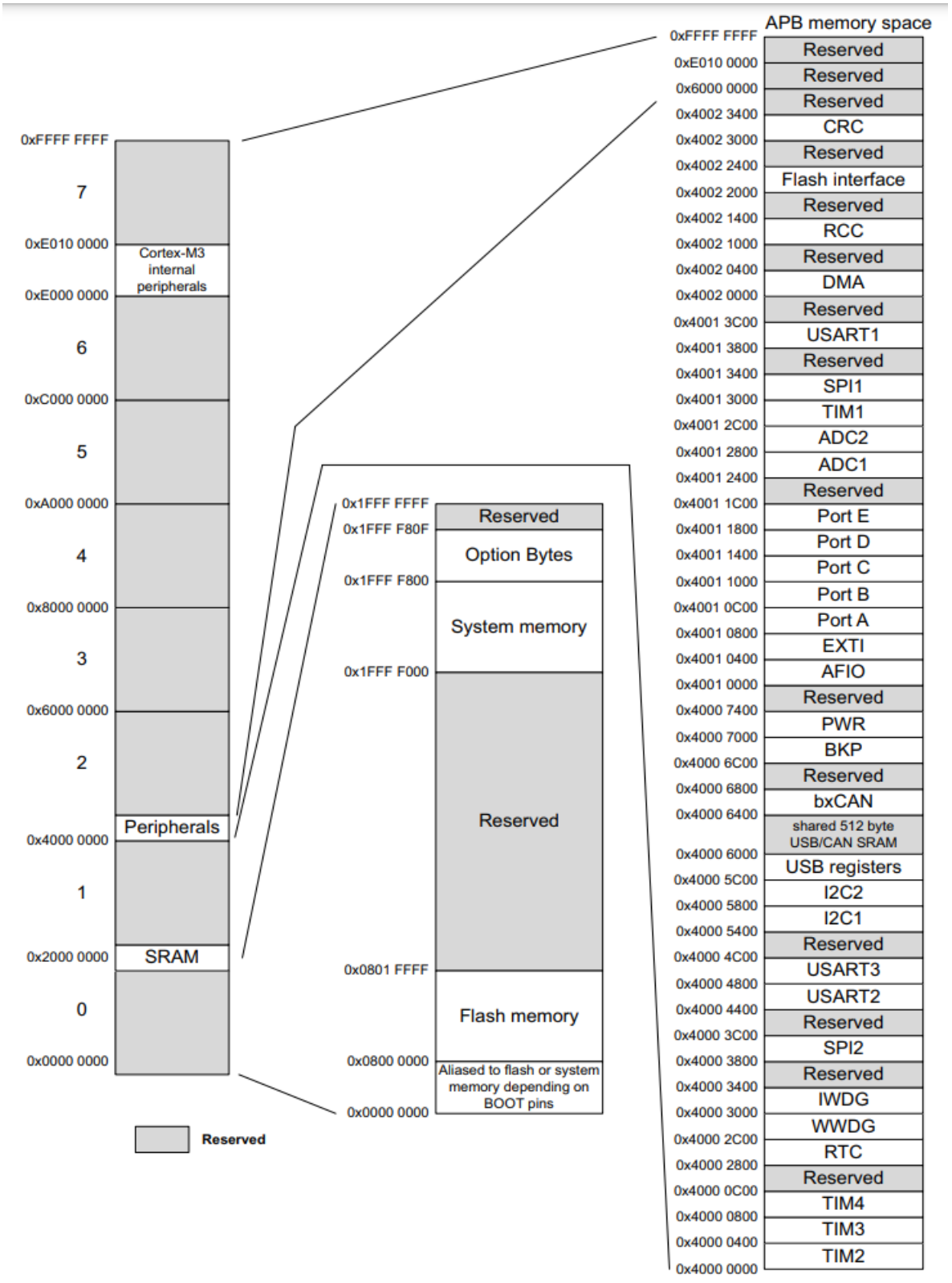


Рисунок 4 – Организация памяти МК STM32F103C8T6

1.2.2 ЖК-дисплей

Для вывода отладочной информации и информации об ошибках был выбран ЖК-дисплей LM016L размером 16x2. У этого дисплея установлен контроллер HD44780.

На рисунке 5 изображен LM016L.

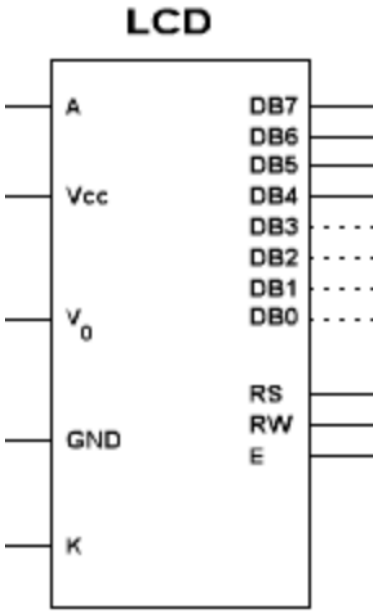


Рисунок 5 - ЖК-дисплей LM016L.

В таблице 4 указаны пины индикатора LM016L [3].

Таблица 4 - пины индикатора LM016L.

№ пина	Название	Описание
1	VSS	Земля
2	VDD	Питание, +5В
3	V_0	Контрастность дисплея
4	RS	Выбор регистра

5	R/W	L - MPU к LCM, H - LCM к MPU
6	E	Enable
7, 8, 9, 10, 11, 12, 13, 14	DB0..DB7	Биты данных от 0 до 7
15	A	Анод LED
16	K	Катод LED

ЖК-дисплей подключен к микроконтроллеру черезу порты: PB-10 - DB7, PB-11 - DB6, PB-12 - DB5, PB-13 - DB4, PB-14 - E, PB-15 - RS.

1.2.3 Выбор вида ввода

Выбор вида ввода представляет собой кнопку (COMP/PHONE), которая отвечает за то, какой порт будет слушаться при состоянии ввода данных. Кнопка подключена к микроконтроллеру через порт PA-14. Нажатая кнопка означает, что ввод должен производится через устройство ПЭВМ с помощью UART, а если кнопка не нажата, то чтение производится с телефона через USB-порт.

1.2.4 Прием данных от ПЭВМ

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

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

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

Внутреннее изображение MAX232 показано на рисунке 6. Назначение пинов описано в таблице 4 [4].

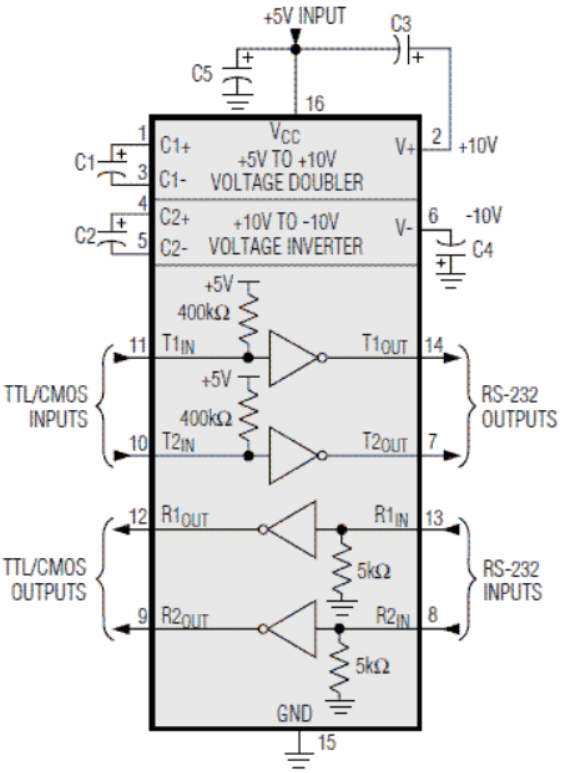


Рисунок 6 – Преобразователь MAX232

В таблице 5 указаны пины преобразователя MAX232.

Таблица 5 - Назначение пинов MAX232

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

Продолжение таблицы 5

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

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

1.2.5 Использование UART при работе с ПЭВМ

Прием данных от ПЭВМ на микроконтроллер осуществляется при помощи модуля UART. В реализации микроконтроллера STM32F103C8T6 имеется 3 интерфейса UART - они одинаковы. На данном микроконтроллере скорость передачи составляет 115200 бод, длина слова составляет 8 бит, включая бит четности и 1 стоповый бит. Передача полнодуплексная. В модуле UART от STM32 существует 7 регистров,

которые его контролируют: USART_SR, USART_DR, USART_BRR, USART_CR1, USART_CR2, USART_GTPR и USART_CR3 [5].

Первый из регистров, USART_SR - регистр статуса. На рисунке 7 изображены его биты.

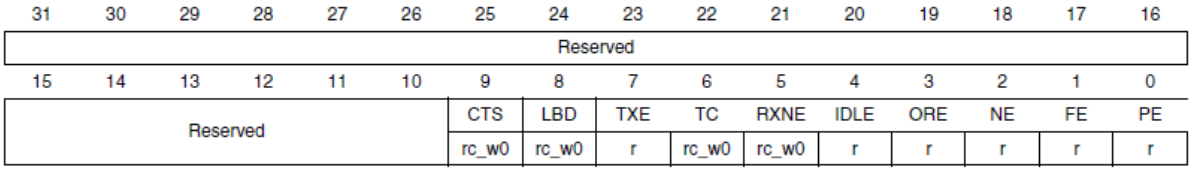


Рисунок 7 - регистр USART_SR

Регистр USART_SR имеет следующие биты:

- TXE. Буферный регистр передатчика пуст. Флаг становится равным 1 после того, как данные перегружаются в регистр сдвига. Флаг устанавливается аппаратно и сбрасывается после записи байта в буферный регистр USART_DR;
- TC. Флаг устанавливается аппаратно и сообщает о том, что передача данных закончена, сдвиговый регистр пуст. Сбрасывается флаг последовательным чтением регистров USART_SR и USART_DR;
- RXNE. Буферный регистр приема не пуст. Флаг сообщает, что в буферном регистре приемника есть данные. Сбрасывается флаг при чтении регистра USART_DR;
- LBD. Выставляется при обнаружении брейка при использовании LIN
- ORE. Ошибка переполнения. Флаг устанавливается в случае, если в приемный буферный регистр поступило новые данные, а предыдущие считаны не были;
- NE. Флаг устанавливается при выделении шума во входном сигнале. Наличие шума определяется как слишком частое переключение входного сигнала;
- FE. Ошибка приема фрейма (кадра). Возникает, когда не был выделен стоп-бит;

- PE. Ошибка паритета. Сигнализирует об ошибке при включенном контроле четности.

На рисунке 8 изображена структура регистра USART_DR.

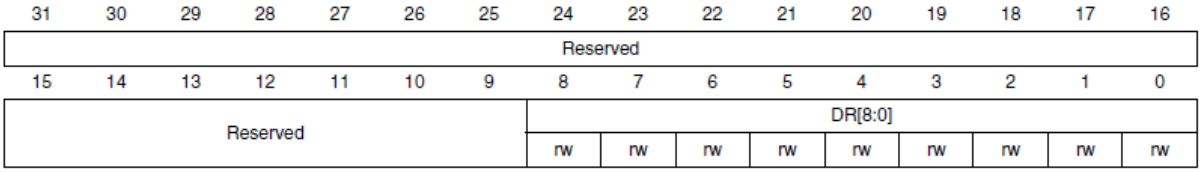


Рисунок 8 - регистр USART_DR.

USART_DR - регистр данных, используется для передачи и чтения данных из буферных регистров передатчика и приемника. На самом деле состоит из 2 регистров: TDR и RDR - регистры данных передатчика и приемника соответственно.

На рисунке 9 изображена структура регистра USART_BRR.

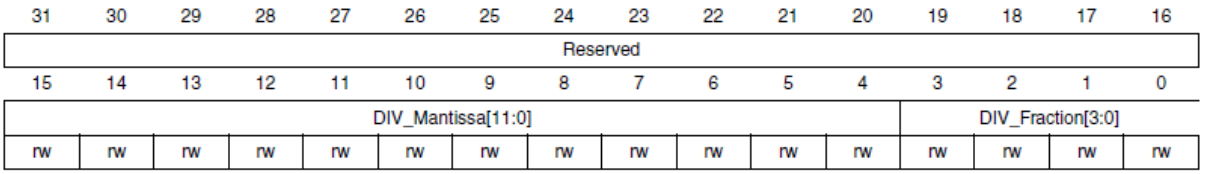


Рисунок 9 - регистр USART_BRR.

USART_BRR - это регистр скорости передачи данных USART. В регистре содержится значение делителя частоты, который определяет скорость обмена данными.

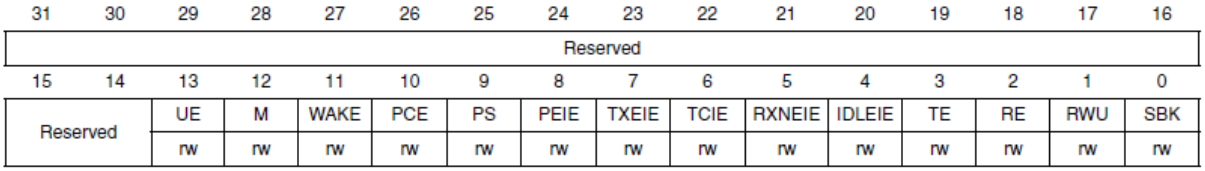
$$BAUD = \frac{F_{ck}}{16 * BRR},$$

где BAUD - скорость обмена данными, в бодах;

F_{ck} - входная частота тактирования UART: PCLK2 для UART1 (используется в приеме данных от ПЭВМ), на шине APB2 и PCLK1 и PCLK3 для UART2 (используется в приеме данных от телефона);

BRR - значение регистра USART_BRR.

На рисунке 10 изображена структура регистра USART_CR1.



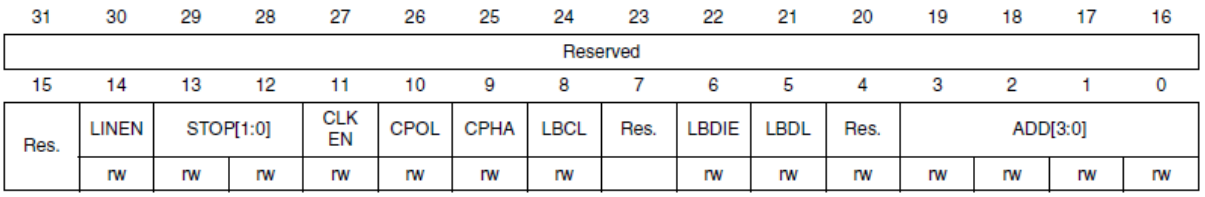
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 10 - структура USART_CR1.

USART_CR1 - регистр контроля. Его биты:

- UE: включить USART - 0 USART выключен, 1 - включен;
- M: длина слова данных. Этот бит определяет длину передаваемых данных. Устанавливается и очищается программно;
- PCE: разрешить контроль четности. Устанавливается и очищается программно;
- PS: выбор типа контроля четности. Этот бит выбирает вариант контроля четности, если установлен бит PCE. Устанавливается и очищается программно;
- TXEIE: разрешить прерывание при опустошении буфера передатчика. Если установлен в 1, то генерируется запрос прерывания USART при установке бита TXE регистра USART_SR;
- TCIE: разрешить прерывания окончания передачи. Если 1, то генерируется запрос прерывания USART при установке флага TC в регистре USART_SR;
- RXNEIE: разрешить прерывание при появлении данных в регистре приемника. Если 1, то генерируется запрос прерывания USART при установке флага RXNE или ORE в регистре USART_SR;
- TE: включить передатчик USART;
- RE: включить приемник USART.

На рисунке 11 изображена структура регистра USART_CR2.



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LINEN	STOP[1:0]		CLK EN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.	ADD[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Рисунок 11 - структура USART_CR2.

USART_CR2 - второй регистр конфигурации. Из важных битов можно выделить STOP - отвечает за количество стоп-бит (00 - 1 стоп-бит, 01 - 0.5 стоп-бит, 10 - 2 стоп-бит, 11 - 1.5 стоп-бит).

Также, у USART еще есть регистры USART_GTPR - регистр, отвечающий за предделитель и USART_CR3 - третий конфигурационный регистр, в котором содержится информация о DMA и SmartCard.

Для приема данных от ПЭВМ используется интерфейс USART1, которому соответствуют порты: PA-9 - TX, PA-10 - RX. Интерфейс находится в асинхронном режиме. Также, на USART1 включены глобальные прерывания.

1.2.6 Прием данных от телефона

Прием данных от телефона осуществляется по USB-коннектору, подключенному к интерфейсу USART2 микроконтроллера STM32F103C8T6 через преобразователь USB-UART FT232RL.

На рисунках 12, 13 изображен преобразователь USB-UART FT232RL и его УГО.

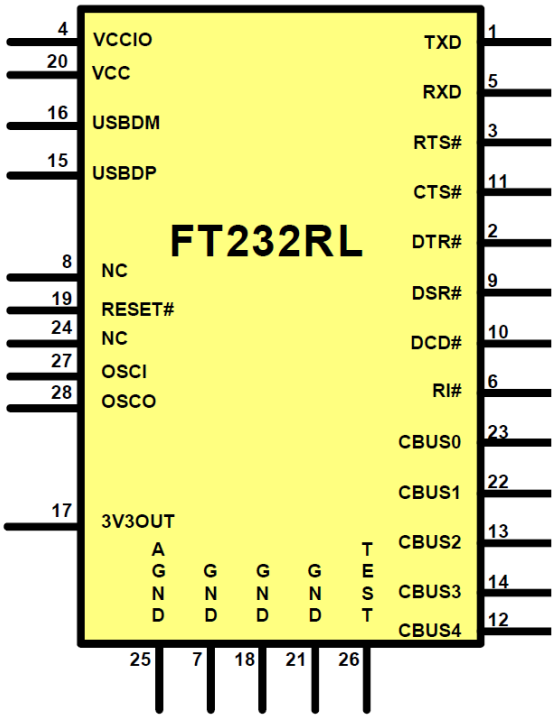


Рисунок 12 - FT232RL.

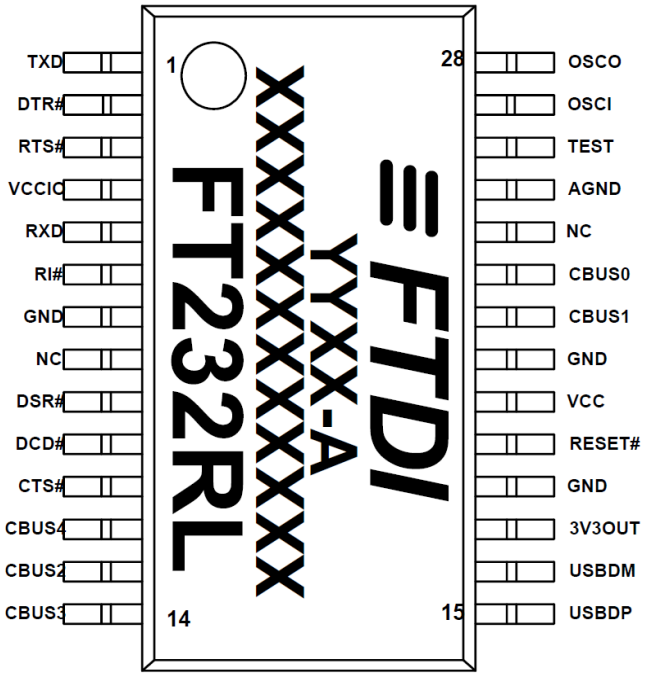


Рисунок 13 - УГО FT232RL.

В таблице 6 дано описание пинов преобразователя FT232RL.

Таблица 6 - пины FT232RL.

№ пина	Название	Тип	Описание
15	USBDP	I/O	положительный сигнал USB данных
16	USBDM	I/O	отрицательный сигнал USB данных
4	VCCIO	PWR	напряжение питание, подключено к источнику питания 5V
7, 18, 21	GND	PWR	земля
17	3V3OUT	O	вывод 3V из интегрированного LDO-регулятора
20	VCC	PWR	напряжение питание, подключено к источнику питания 5V
25	AGND	PWR	Аналог земли устройства для внутреннего счетчика
8, 24	NC	NC	Без внутренней связи
19	RESET	I	RESET
26	TEST	I	Переводит устройство в IC тестовый режим
27	OSCI	I	Ввод осциллографа

28	OSCO	O	Вывод осциллографа
1	TXD	O	Вывод UART
2, 3, 9, 11	DTR#, RTS#, DSR#, CTS#	O	Сигналы рукопожатия
5	RXD	I	Ввод UART
6	RI#	I	Ввод кольцевого управляющего индикатора
10	DCD#	I	Ввод детектора переноса данных
12	CBUS4	O	CBUS только на вывод
13, 14, 22, 23	CBUS2, CBUS3, CBUS1, CBUS0	I/O	CBUS на ввод/вывод

FT232RL подключен к микроконтроллеру STM32F103C8T6 через интерфейс USART2 через порты: PA-2 - TX, PA-3 - RX. Интерфейс находится в асинхронном режиме. Также, на USART1 включены глобальные прерывания.

В качестве USB-коннектора выбран AU-Y1005-R, тип коннектора: USB-A. Данный коннектор был выбран ввиду универсальности данного типа подключения.

1.2.7 Выбор алгоритма хэширования

Выбор используемого алгоритма хэширования осуществляется при помощи 3 кнопок. Первая кнопка (CHOOSE), подключена через порт PA-1, выбирает текущий алгоритм в качестве исполняемого. Вторая кнопка (NEXT), подключенная по порту PA-2, увеличивает id текущего алгоритма на 1. id принимает значения от 0 до 2, если при нажатии второй кнопки id станет превышать 2, то он принудительно устанавливается в 0. Третья кнопка (PREV), подключена через порт PA-3, уменьшает id текущего алгоритма на 1. Если при нажатии на третью кнопку id станет меньше 0, то он устанавливается в 2.

1.2.8 Построение функциональной схемы

На основе всех вышеописанных сведений была спроектирована функциональная схема разрабатываемой системы, показанная на рисунке 14[6, 7].

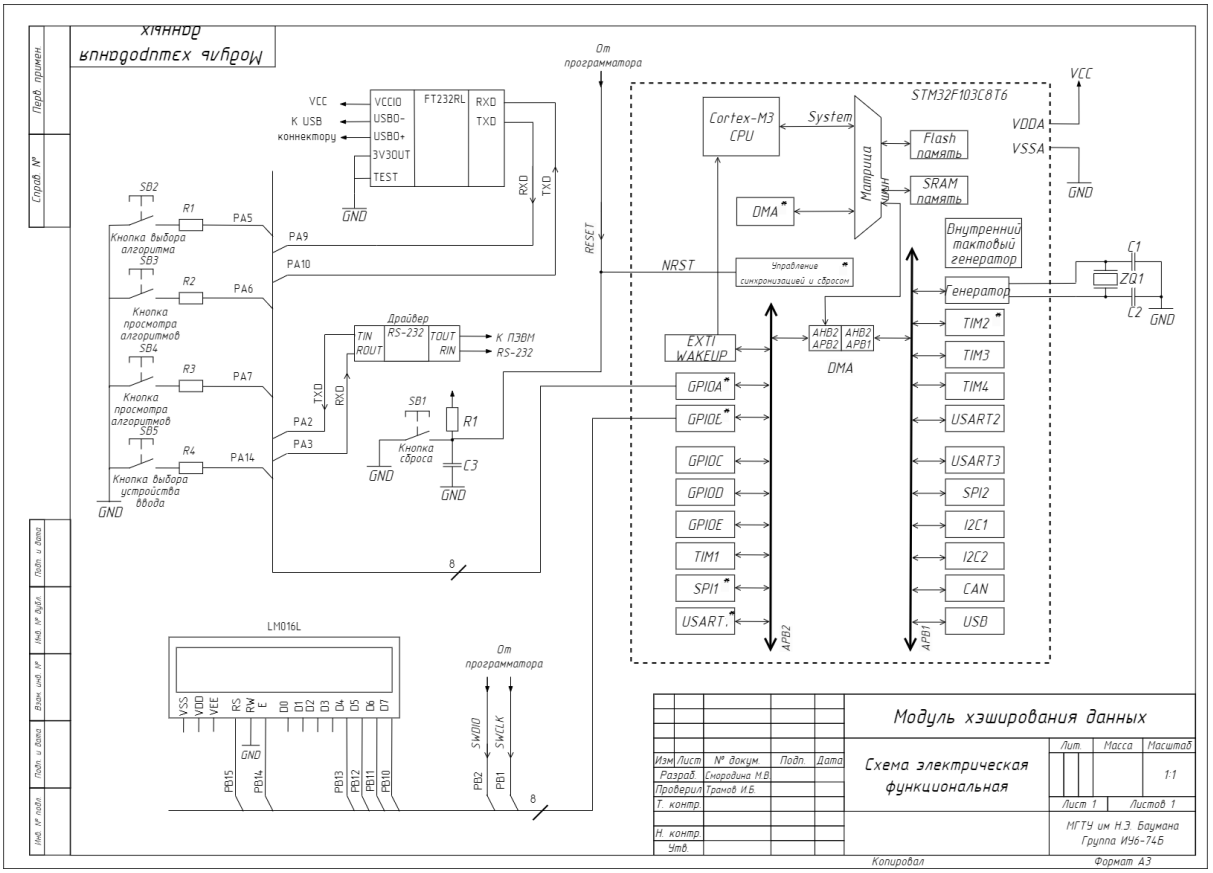


Рисунок 14 - Схема функциональная

1.3 Проектирование принципиальной схемы

В этом разделе приведено принципиальное описание работы системы и проектирование принципиальной схемы.

1.3.1 Разъем программатора

Для программирования МК используется специальный программатор ST-LINK V2. Подключение программатора осуществляется при помощи портов PB1 и PB2, которые выполняют роль SWDIO и SWCLK соответственно.

Он имеет следующие разъемы для подключения к МК:

- SWCLK – тактовый сигнал;
- SWDIO – для передачи данных;

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

1.3.2 Расчет потребляемой мощности

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

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

$$P = U * I$$

где U – напряжение питания (В);

I – ток потребления микросхемы (мА).

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

$$P = I^2 * R$$

где R – сопротивление резистора;

I – ток, проходящий через резистор.

Расчет потребляемого напряжения для каждой микросхемы показан в таблице 7.

Таблица 7 – Потребляемая мощность

Микросхема	Ток потребления, мА	Потребляемая мощность, мВт	Количество устройств	Суммарная потребляемая мощность, мВт
STM32F103 C8T6	150	495	1	495
MAX232	8	26,4	1	26,4
FT232RL	24	500	1	500
LM016L	1	250	1	250
CF-100	10	-	5	9

$P_{суммарная} = P_{STM32F103C8T6} + P_{MAX232} + P_{FT232RL} + P_{LM016L} + P_{CF-100} = 495 + 26,4 + 500 + 250 + 9 = 1280,4$ мВт

Суммарная потребляемая мощность системы равна 1280,4 мВт = 1,3 Вт.

1.3.3 Построение принципиальной схемы

На основе всех вышеописанных сведений была спроектирована принципиальная схема разрабатываемой системы, показанная на рисунке 15[6, 7].

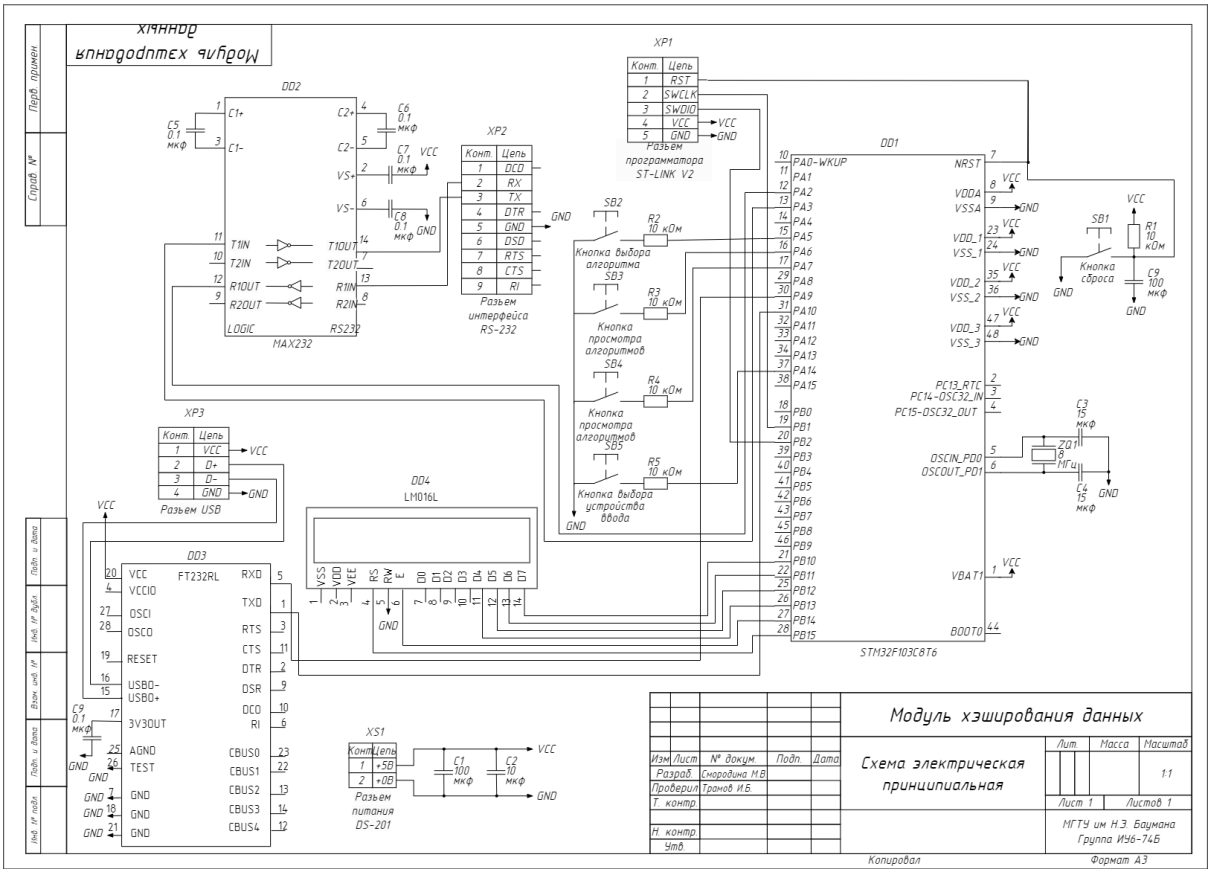


Рисунок 15 - Принципиальная схема

1.4 Алгоритмы работы системы

1.4.1 Функция main

Программа начинает работу с функции main. Управление программой осуществляется с помощью состояний, за которое отвечает extern переменная state.

Первым состоянием является нулевое, состояние ожидания ввода данных. Программа ожидает ввод с UART или USB. После того, как были получены входные данные, состояние переключается в “1”, которое означает состояние ожидания выбора алгоритма хэширования. Выбор алгоритма происходит при увеличении или уменьшении переменной id, где 0 соответствует алгоритму SHA, 1 - CRC, 2 - MD. Программа находится в этом состоянии, пока пользователь не совершит действие, отвечающее за

выбор алгоритма. После этого программа переходит в состояние “2”, а управление передается подпрограмме переключения между алгоритмами.

Каждое состояние сопровождается отладочным выводом о том, что требуется сделать пользователю и о введенных пользователем данных.

На рисунке 16 показана схема алгоритма функции main.

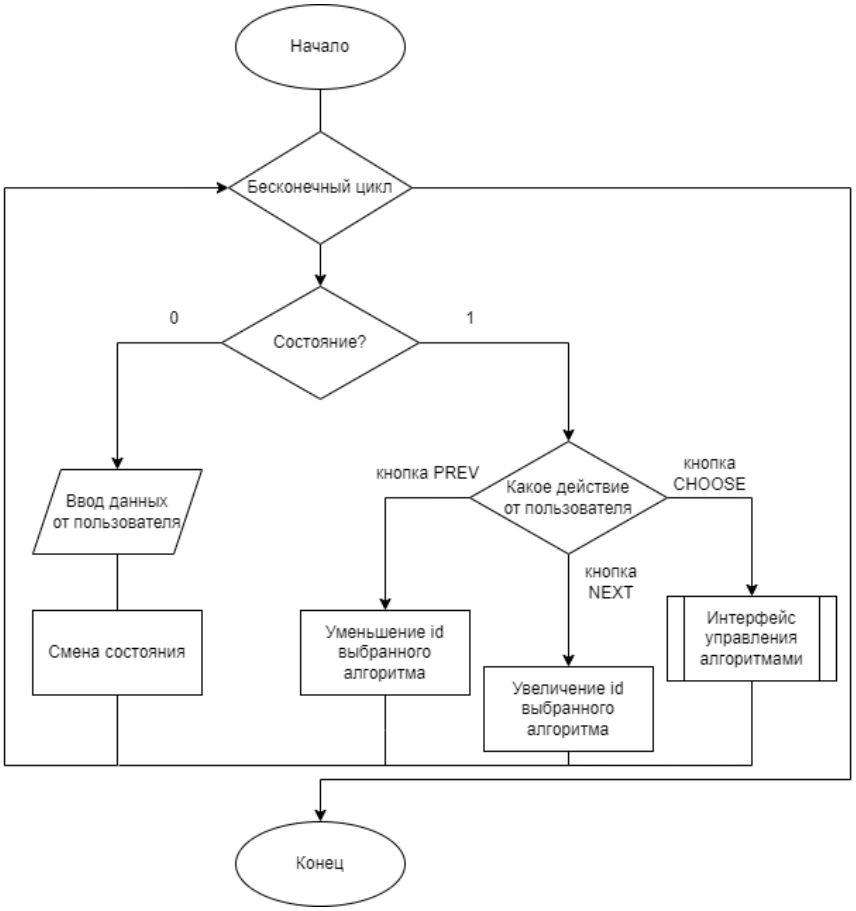


Рисунок 16 - Схема алгоритма main

1.4.2 Подпрограмма переключения между алгоритмами

Когда управление передается подпрограмме переключения между алгоритмами, состояние программы становится “2”, означающее, что программа находится в состоянии выполнения хэширования входных данных.

Сначала пользователю на выход подается информация о выбранном алгоритме хэширования, после чего отображаются стадии выполнения алгоритма: начало хэширования, его завершение, результаты вычислений.

Так как длина хэша в каждом алгоритме разная, то и вывод организован для каждого по-разному, так как не каждый хэш полностью помещается на выбранный ЖК-дисплей размером 16x2.

Для SHA-256 длина хэша составляет 32 символа, поэтому сначала программа выводит на ЖК-дисплей первые 16 символов, 8 в верхней строке, и 8 в нижней, а после ожидания выводит оставшиеся 16. За то, будет текст отображен сверху или снизу ЖК-дисплея отвечает переменная j, равная “0” (сверху) или “1” (снизу), а за отображение переменных по горизонтали отвечает переменная i, принимающая значения от 0 до 16.

Для CRC-16 длина контрольной суммы составляет 4 символа, поэтому осуществляется простой вывод на ЖК-дисплей.

Для MD-5 длина хэша составляет 16 символов, поэтому на верхний ряд осуществляется вывод первых 8 символов хэша, а на нижний - оставшихся 8.

На рисунке 17 представлена схема алгоритма функции do_algorithm.

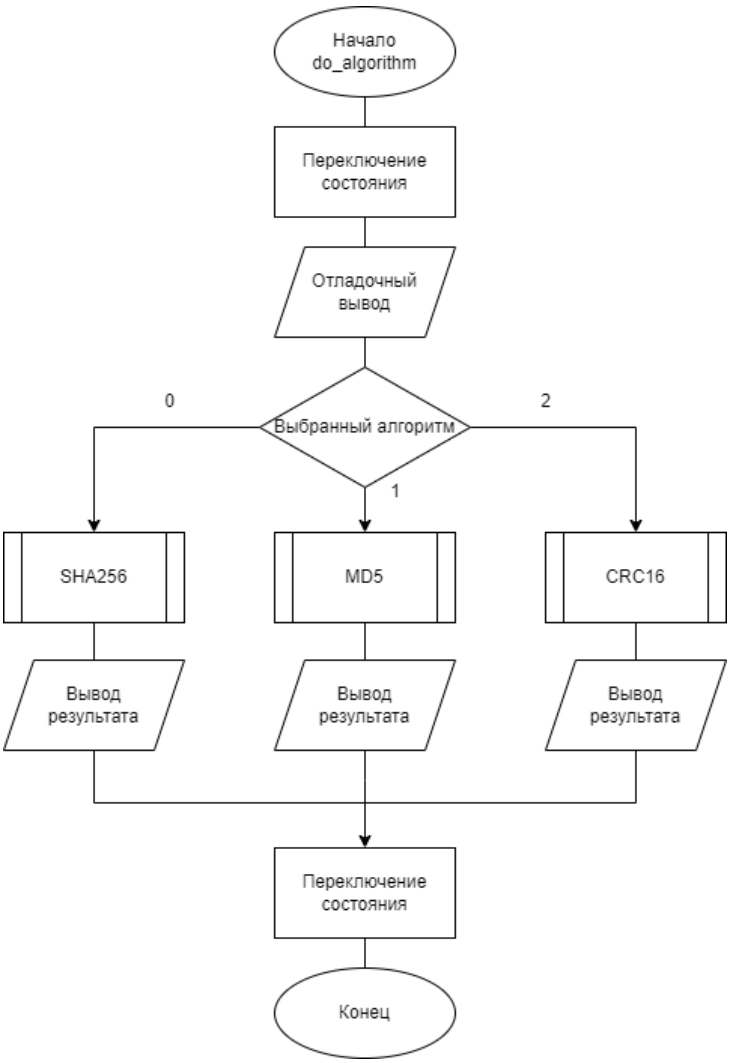


Рисунок 17 - Схема алгоритма do_algorithm

1.4.3 Алгоритм хэширования SHA

SHA (Secure Hash Algorithm, Безопасные алгоритмы хэширования) - это семейство криптографических хэш-функций, преобразующих сообщения произвольной длины в хэш фиксированной длины.

Существуют несколько версий этого семейства:

1 SHA-0 - первая версия алгоритма, разработанная еще в 1993 году;

2 SHA-1 - является исправленной версией SHA-0, 1995 г.;

3 SHA-2 - включает в себя несколько криптографических хэш-функций, различающихся размером ключа (SHA-224, SHA-384,

SHA-256, SHA-512, SHA-512/256 и SHA-512/256), более безопасен, чем SHA-1, применяется до сих пор, 2002 г.;

4 SHA-3 - последняя на данный момент версия, ранее называлась Keccak, размер хэшей на выходе равен SHA-2, но использует принципиально новый алгоритм, 2012 г..

Для реализации была выбрана хэш-функция SHA-256, принадлежащая версии SHA-2.

Хэш-функции SHA-2 в основе используют метод Меркла-Дамгора, использующийся для построения криптографических хэш-функций, когда из сообщения произвольной длины получается хэш фиксированной длины. Для этого, входное сообщение разбивается на блоки равной длины, каждый из которых проходит через преобразования и на выходе получается хэш фиксированной длины.

На рисунке 18 представлен схематичный принцип работы структуры Меркла-Дамгора. Функция f - односторонняя функция сжатия, принимает два входных блока, и преобразует их в один выходной. Алгоритм начинается с поданного на вход начального значения, или вектора инициализации (IV), а на выходе получаем Hash фиксированного размера.

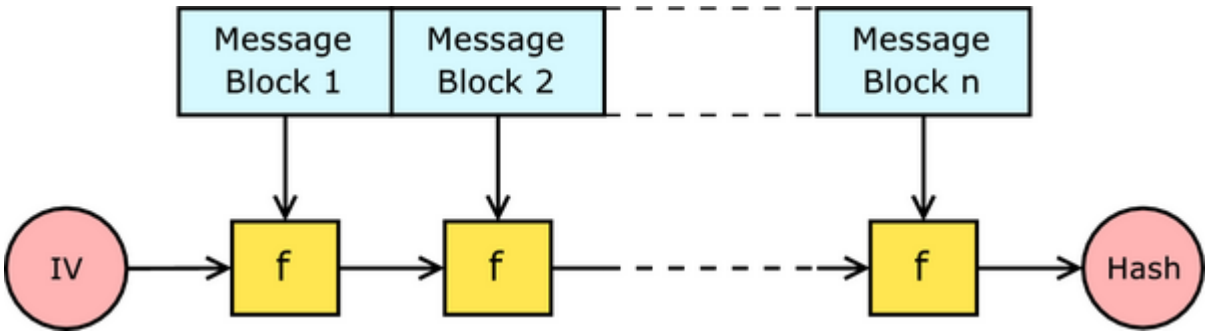


Рисунок 18 - Структура Меркла-Дамгора.

Сам вектор инициализации состоит из 8 констант размером в 32 бита:

- H1 = 0x6A09E667;
- H2 = 0xBB67AE85;

- H3 = 0x3C6EF372;
- H4 = 0xA54FF53A;
- H5 = 0x510E527F;
- H6 = 0x9B05688C;
- H7 = 0x1F83D9AB;
- H8 = 0x5BE0CD19.

Каждая константа высчитывается как первые 32 бита дробной части корней первых 8 простых чисел. Такие числа были выбраны на основе обнаружения “Безопасности Крипто” для того, чтобы добиться высокой степени диффузии и нелинейности в процессе хэширования, так как это позволит уменьшить количество возможных коллизий.

Далее, данные разбиваются на блоки фиксированного размера. В SHA-256 блок составляет 512 бит. Если длины сообщения или его оставшейся части не хватает для заполнения блока, то он заполняется нулями. Так как существует вероятность совпадения полученных хэш-функций, если сообщения различны, но начинаются с одних и тех же символов и оканчиваются нулями, то последний бит заполнителя заменяют на “1”.

Функция сжатия работает по следующему принципу: алгоритм пропускает каждый блок через 64 операции с константами, являющимися первыми 32 битами кубических корней из первых 64 простых чисел. В каждой итерации будут изменяться значения изначальной заданных констант.

$a = H0,\ b = H1,\ c = H2,\ d = H3,\ e = H4,\ f = H5,\ g = H6,\ h = H7$

$\Sigma 0 := (a\ rotr\ 2)\ xor\ (a\ rotr\ 13)\ xor\ (a\ rotr\ 22)$

$Ma := (a\ and\ b)\ xor\ (a\ and\ c)\ xor\ (b\ and\ c)$

$t2 := \Sigma 0 + Ma$

$\Sigma 1 := (e\ rotr\ 6)\ xor\ (e\ rotr\ 11)\ xor\ (e\ rotr\ 25)$

$Ch := (e\ and\ f)\ xor\ ((not\ e)\ and\ g)$

$$t1 := h + \Sigma 1 + Ch + k[i] + w[i]$$

$$h = g,\ g = f,\ f = e,\ e = d + t1,\ d = c,\ c = b,\ b = a,\ a = t1 + t2$$

После 64 итераций цикла, сумма значений переменных в последней итерации складываются со старым вектором инициализации и образуют новый:

$$h0 \mathrel{+}= a, h1 \mathrel{+}= b, h2 \mathrel{+}= c, h3 \mathrel{+}= d, h4 \mathrel{+}= e, h5 \mathrel{+}= f,$$

$$h6 \mathrel{+}= g, h7 \mathrel{+}= h$$

Сумма элементов вектора, полученного в последнем блоке, станет итоговым хэшем:

$$h1 + h2 + h3 + h4 + h5 + h6 + h7 = SHA256$$

На рисунке 19 представлена схема работы алгоритма SHA256.

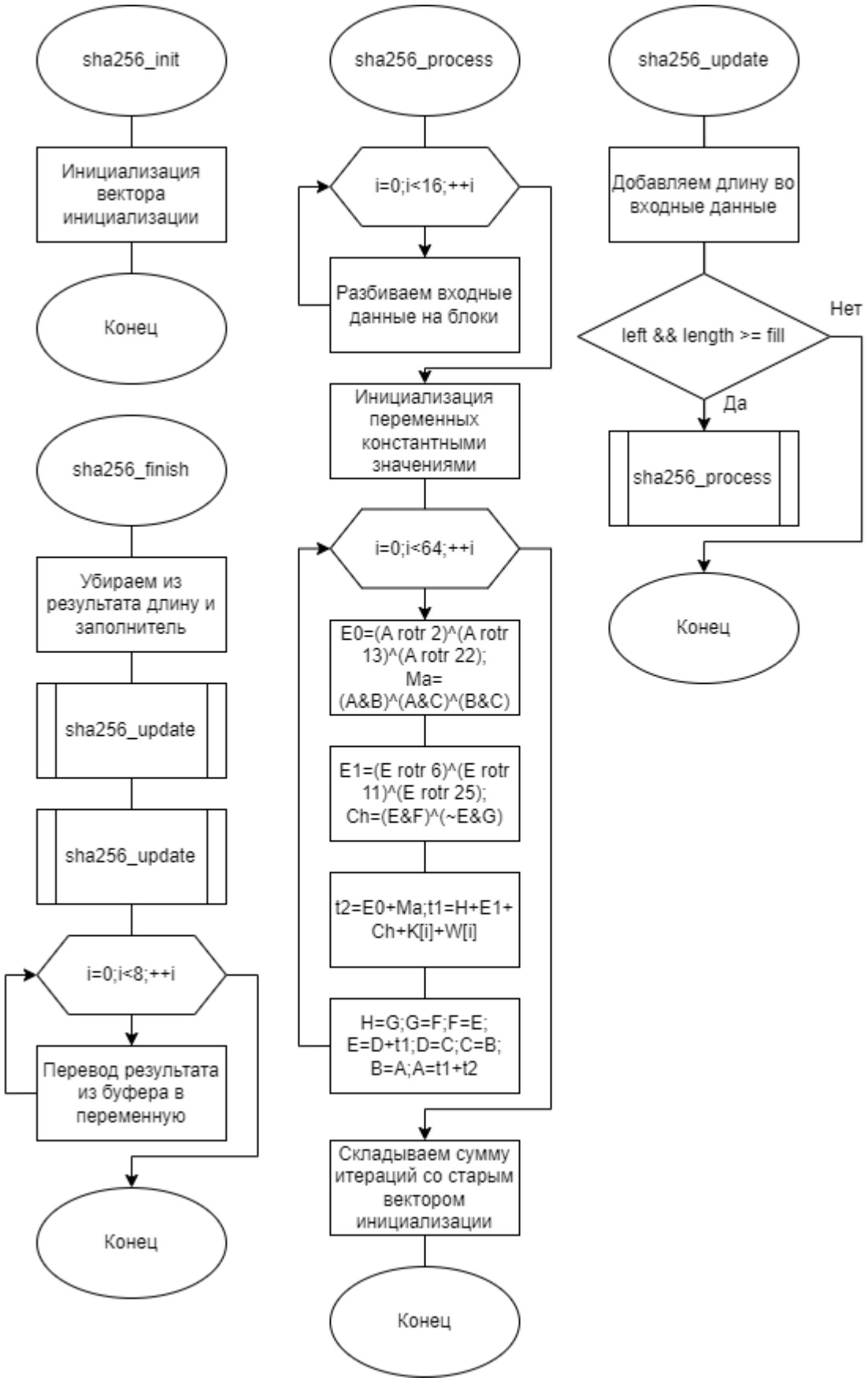


Рисунок 19 - Схема алгоритма SHA256.

Пример:

На вход дана строка hello world. Выходное значение: b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9.

1.4.4 Алгоритм хэширования CRC

CRC (Циклический избыточный код) является алгоритмом нахождения контрольной суммы для последующей проверки целостности данных. Главные преимущества циклических кодов - простота их реализации и обнаружение пакетных ошибок, наиболее распространенных ошибок передачи в каналах связи.

Контрольная сумма - это вычисленное значение по некоторому правилу на основе кодируемого сообщения.

Основой алгоритма CRC служат свойства деления с остатком двоичных многочленов, а само значение CRC представляет из себя остаток от деления многочлена, который является входными данными, на определенный видом CRC порождающий многочлен.

Каждой конечной последовательности битов $a_1, a_2 \dots a_n$ можно взаимно однозначно сопоставить многочлен $\sum_{n=0}^{N-1} a_n x^n$, а количество различных многочленов, меньших степени N равно 2^N. Таким образом, значение контрольной суммы равно:

$$R(x) = P(x)x^N mod\ G(x),$$

где R(x) - полином, представляющий значение CRC, P(x) - полином, соответствующий входным данным, G(x) - порождающий полином, N - степень порождающего полинома.

Основным параметром CRC является порождающий полином. У этого полинома есть степень, которая определяется как количество битов, использующихся для вычисления в алгоритме CRC. Например, 8 степень

используется в 8-битном алгоритме CRC-8. Также, еще одним важным параметром для порождающего полинома является стартовое значение слова.

Общий алгоритм подсчета циклического избыточного кода следующий: из входных данных берется первого слово, размером соответствующее битности выбранного CRC. Если старший бит “1”, то выполняется сдвиг влево на один разряд, а затем производится XOR-операция с порождающим многочленом. Если же старшим битом является “0”, то после сдвига не происходит XOR-операции. После сдвига старший бит удаляется, а на место младшего записывается следующий бит из входных данных. Алгоритм останавливается после того, как был загружен последний бит данных, а получившийся остаток и является контрольной суммой.

Существует множество видов CRC-алгоритма, которые отличаются не только размерностью (CRC-8, CRC-16, CRC-32), но и порождающим полиномом.

Для реализации был выбран алгоритм CRC-16/CCITT, где порождающий полином:

$$x^{16} + x^{12} + x^{5} + 1$$

На рисунке 20 представлена схема работы алгоритма CRC16/CCITT.

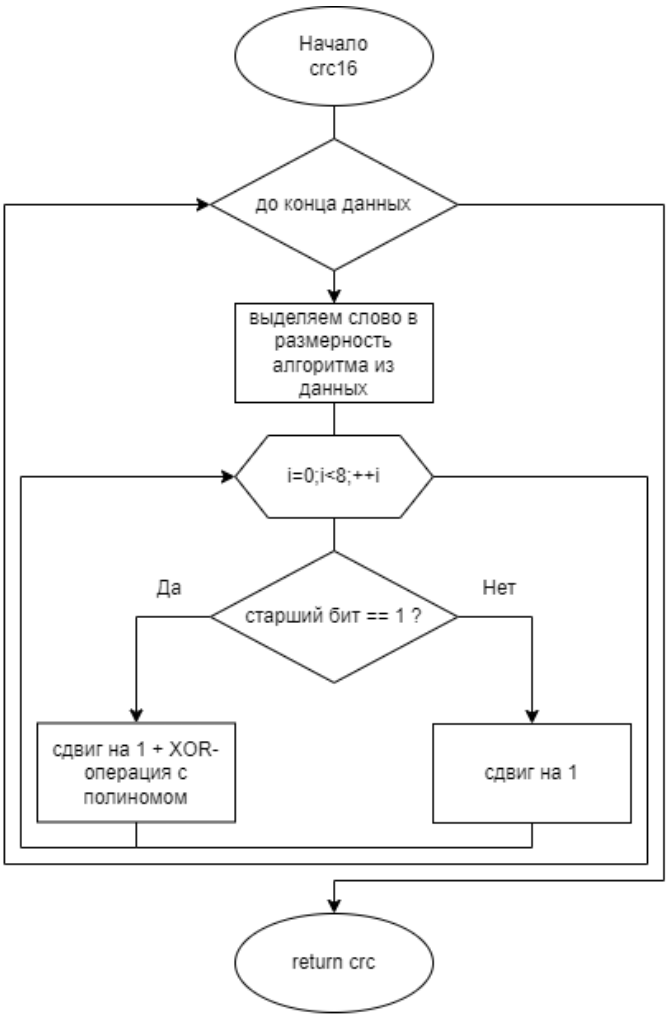


Рисунок 20 - Схема алгоритма CRC16.

Пример:

На вход подана строка: hello world. Значение контрольной суммы: EFEB.

1.4.5 Алгоритм хэширования MD

Для реализации алгоритма хэширования MD была выбрана хэш-функция MD-5.

MD-5 это 128-битная хэш-функция, которая входные данные произвольной длины преобразует в хэш длиной 128-бита. Алгоритм был разработан в 1992 году профессором Рональдом Л. Риверсом, а за основу был взят MD-4.

Алгоритм состоит из 5 шагов:

1. Выравнивание потока;
2. Добавление длины сообщения;
3. Инициализация буфера;
4. Вычисления в цикле;
5. Результат выполнения.

На этапе “Выравнивание потока” в конец сообщения дописывают “1”, а затем некоторое количество нулей, чтобы размер входных данных был сравним с 448 по модулю 512.

На следующем этапе в оставшиеся 64 бита дописывают длину сообщения до выравнивания. Если же длина превосходит $2^{64} - 1$, то записывают только младшие биты. Таким образом, длина потокового сообщения становится кратной 512, а вычисления будут проводиться с массивом данных длиной по 512 бита.

На третьем этапе инициализируются 4 32-битные переменные для хранения промежуточных результатов, а также задаются начальные их значения в 16-ричном виде:

$A = 01\ 23\ 45\ 67;\ //\ 67452301h;$

$B = 89\ AB\ CD\ EF;\ //\ EFCDAB89h;$

$C = FE\ DC\ BA\ 98;\ //\ 98BADCFEh;$

$D = 76\ 54\ 32\ 10.\ //\ 10325476h.$

Далее, производятся вычисления в 4 этапа по 16 раундов для каждого отдельного 512-битного блока данных. Каждый этап состоит из своей функции:

1. $F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$;
2. $F(X, Y, Z) = (X \wedge Z) \vee (\neg Z \wedge Y)$;
3. $F(X, Y, Z) = X \oplus Y \oplus Z$;
4. $F(X, Y, Z) = Y \oplus (\neg Z \vee X)$.

Также, для вычислений необходима специальная таблица констант T[64], вычисляемая как:

$$T[n] = int(2^{32}|sin(n)|)$$

Каждый раунд на каждом этапе вычисляется по формуле:

$$a = b + ((a + Func(b, c, d) + X[k] + T[i]) \lll s),$$

где k - номер 32-битного слова из текущего блока сообщения, $\lll s$ - циклический сдвиг вправо на s бит полученного 32-битного аргумента, s задается отдельно для каждого раунда, Func(b, c, d) - функция для данного этапа.

Далее, заносим в блок данных элемент *n* из массива 512-битных блоков. Сохраняются значения A, B, C и D, оставшиеся после операций над предыдущими блоками (или их начальные значения, если блок первый), а затем суммируем с результатом выполнения предыдущего цикла.

Результатом вычислений находится в буфере ABCD, выводится начиная с младшего байта A и до старшего байта D.

Схема работы алгоритма представлена на рисунках 21 и 22.

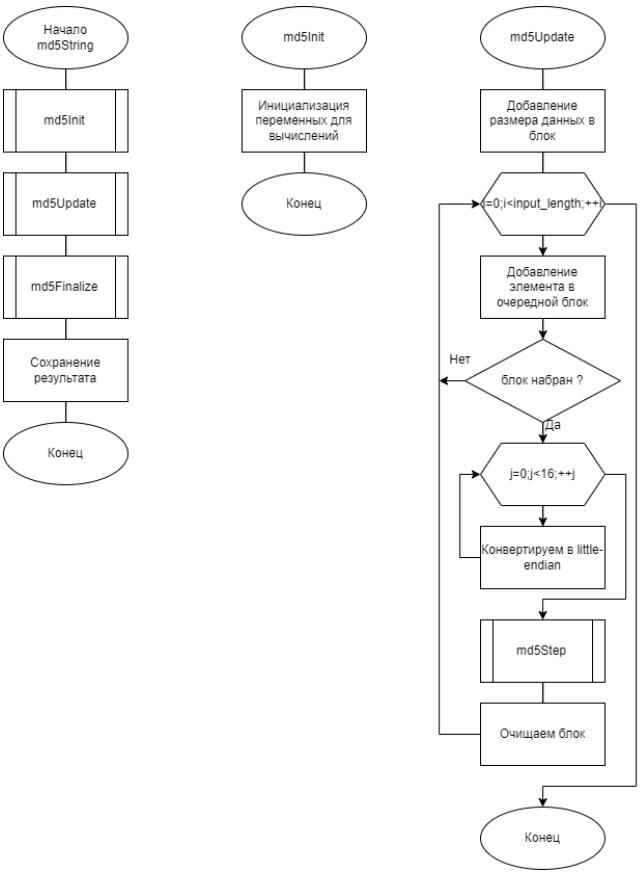


Рисунок 21 - Схема алгоритма MD5, первая часть

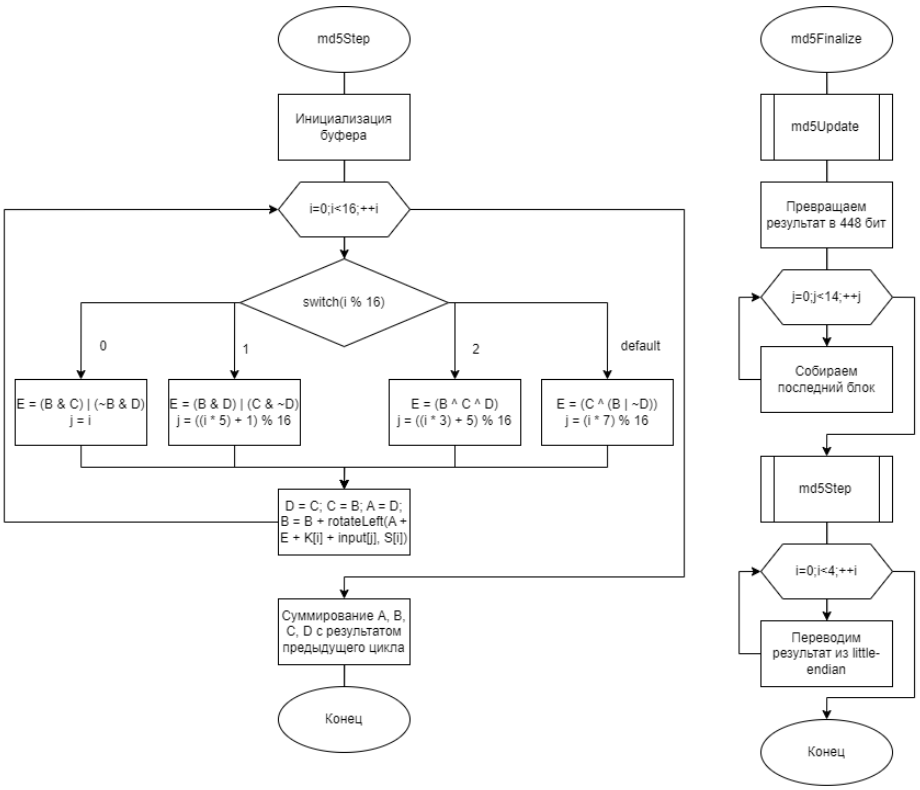


Рисунок 22 - Схема алгоритма MD5, часть вторая.

Пример:

На вход подана строка: hello world. Выходное значение работы алгоритма: 5eb63bbbe01eeed093cb22bb8f5acdc3.

2 Технологическая часть

Для реализации МК-системы хэширования данных была использована среда программирования STM32CubeIDE, TerminalTMBv3 для эмуляции ввода данных с COM-порта. Симуляция проводилась в программе Proteus 8.13, программа написана на языке С[10].

2.1 Отладка и тестирование программы

Отладка программы производилась в среде программирования STM32CubeIDE с помощью программы Proteus 8.13, предназначенной для моделирования аналоговых и цифровых устройств и при помощи терминала TerminalTMBv3 для эмуляции ввода данных с COM-порта вместо ввода данных с телефона по USB. Также, для эмуляции работы USB было скачано расширение Virtual USB, которое подключает виртуальный COM-порт из Proteus как реальный.

При написании кода использовались библиотеки языка Си:

- string.h - библиотека предназначена для работы со строками в языке Си, оттуда были использована функция strlen(), показывающая размер строки;
- stdio.h - из библиотеки была использована функция sprintf() для преобразования вывода результатов выполнения алгоритмов в консоль;
- stdint.h - библиотека для объявления некоторых целочисленных констант и макросов;
- stdlib.h - библиотека для управления выделением памяти.

Для упрощенного взаимодействия с UART модулем микроконтроллера STM32F103C8T6 были использованы HAL-функции: HAL_UART_Receive() для чтения информации и HAL_UART_Trasmit() для вывода информации.

После сборки проекта создается “.hex” файл объемом 54 Кбайт - вес скомпилированной программы.

В результате отладки и тестирования программы была создана МК-система для хэширования вводимых данных с ПЭВМ и телефона, соответствующая требованиям ТЗ.

2.2 Симуляция работы системы

Для симуляции работы МК-системы были использованы программы Proteus 8.13 и TerminalTMBv3. На рисунке 23 показана МК-система для хэширования данных.

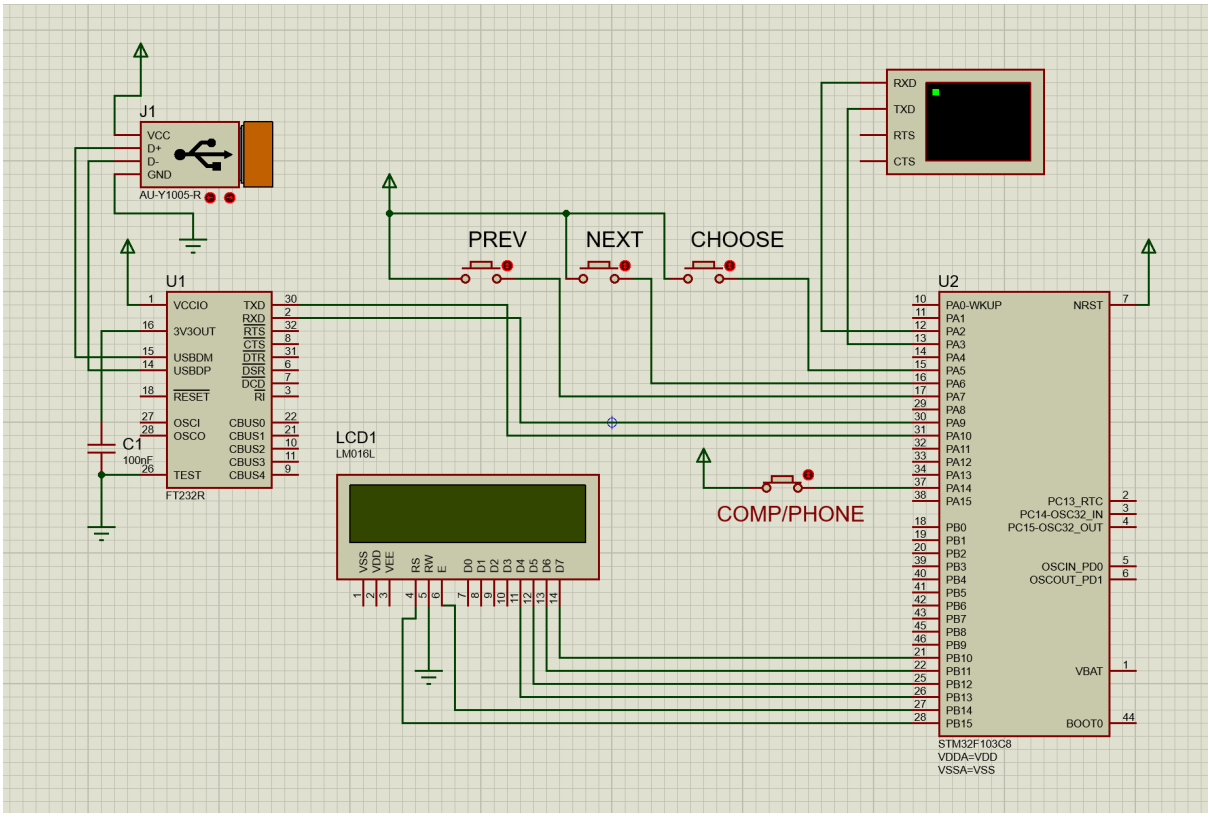


Рисунок 23 - МК-система для хэширования данных.

Для моделирования ввода данных с ПЭВМ использован инструмент системы - Virtual Terminal. Он представляет собой эмуляцию простейшего терминала, позволяющую передавать данные на микроконтроллер по UART через порты RX и TX. На рисунке 24 представлен ввод данных в

терминал и их отображение на ЖК-дисплее, показывающее, что данные дошли до микроконтроллера.

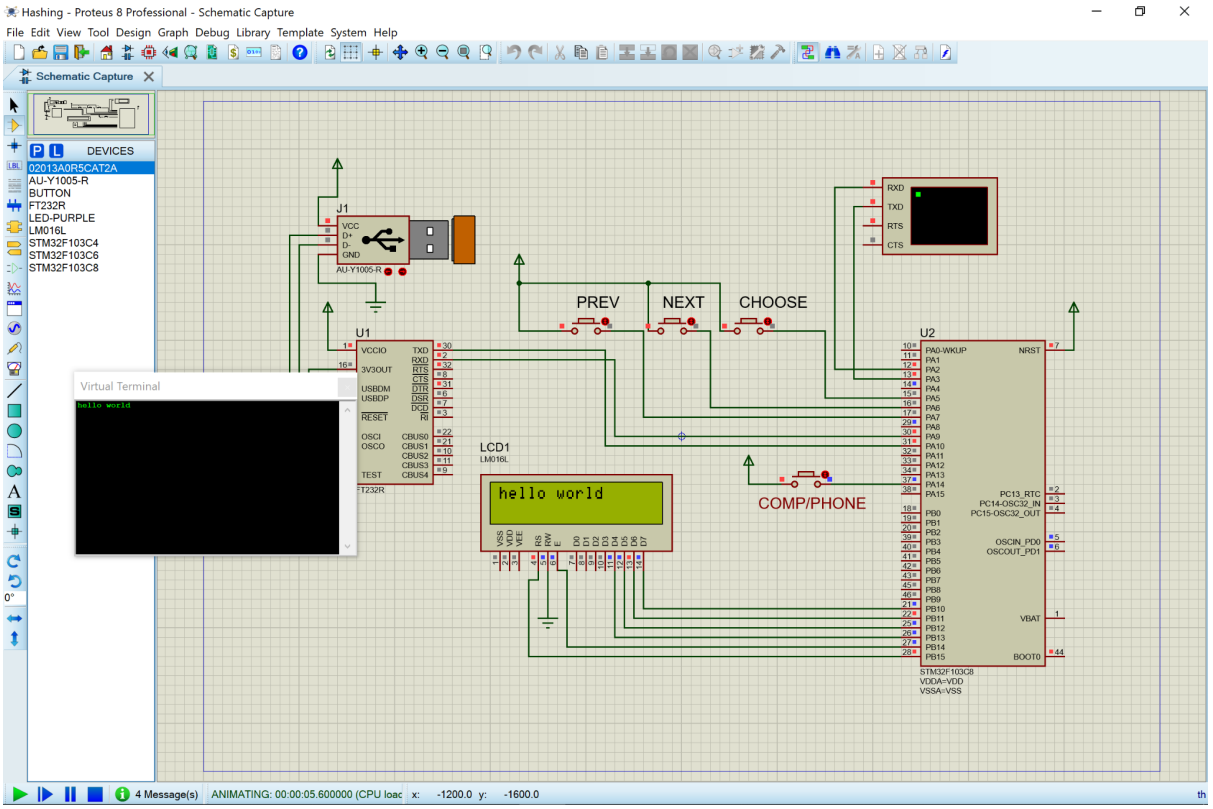


Рисунок 24 - Ввод данных с ПЭВМ через Virtual Terminal.

Для моделирования ввода данных через телефон необходимо эмулировать подключение виртуального USB-коннектора из Proteus к компьютеру, на котором идет запуск программы. Для этого, нужно установить специальное расширение Proteus - Virtual USB Driver. В результате, при запуске программы с USB-коннектором, в Диспетчере устройств компьютера появляется COM-порт данного коннектора - изображено на рисунке 25.

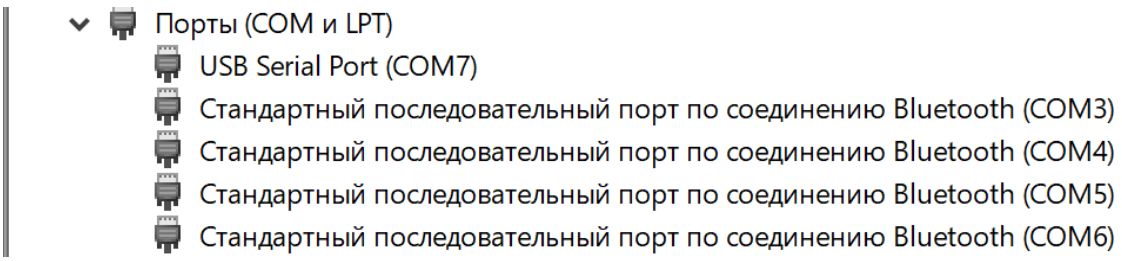


Рисунок 25 - Наличие COM-порта в Диспетчере устройств

Далее, необходимо настроить данный COM-порт как в Диспетчере устройств, так и в программе TerminalTMB в соответствии с настройками микроконтроллера - настройки можно увидеть на рисунках 26 и 28.

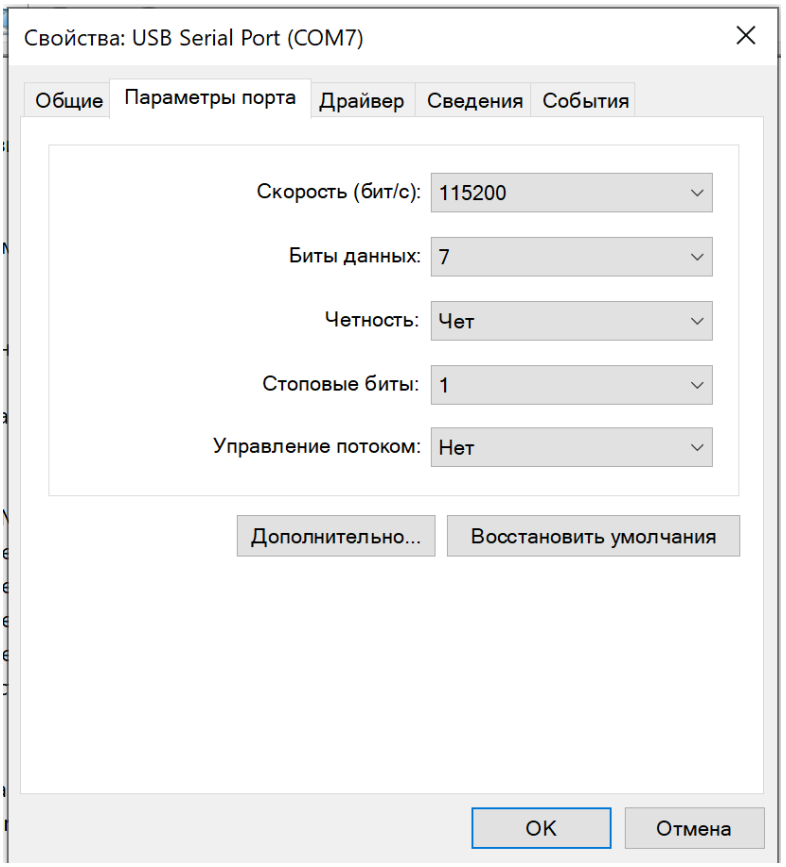


Рисунок 26 - Настроенные параметры порта из Диспетчера устройств

На рисунках 27, 28 и 29 представлена эмуляция ввода данных в МК-систему с телефона.

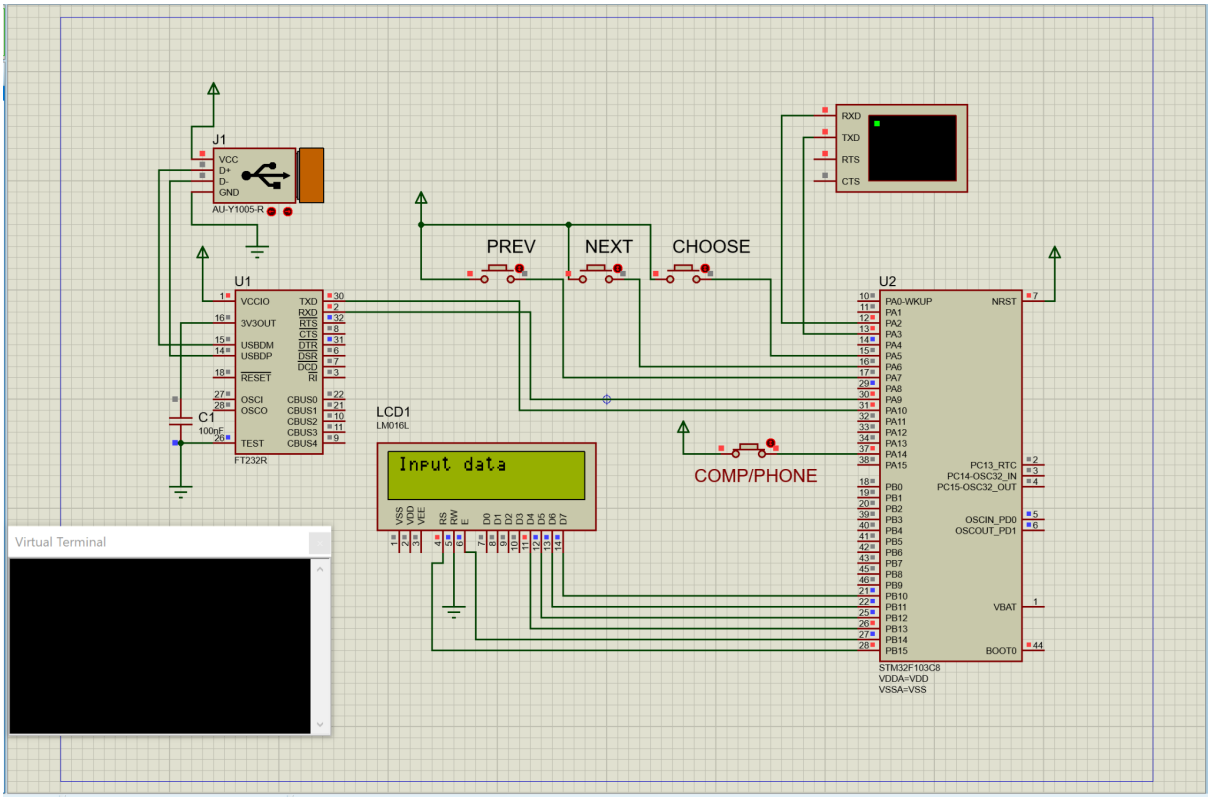


Рисунок 27 - Модель в Proteus, ожидание ввода с USB

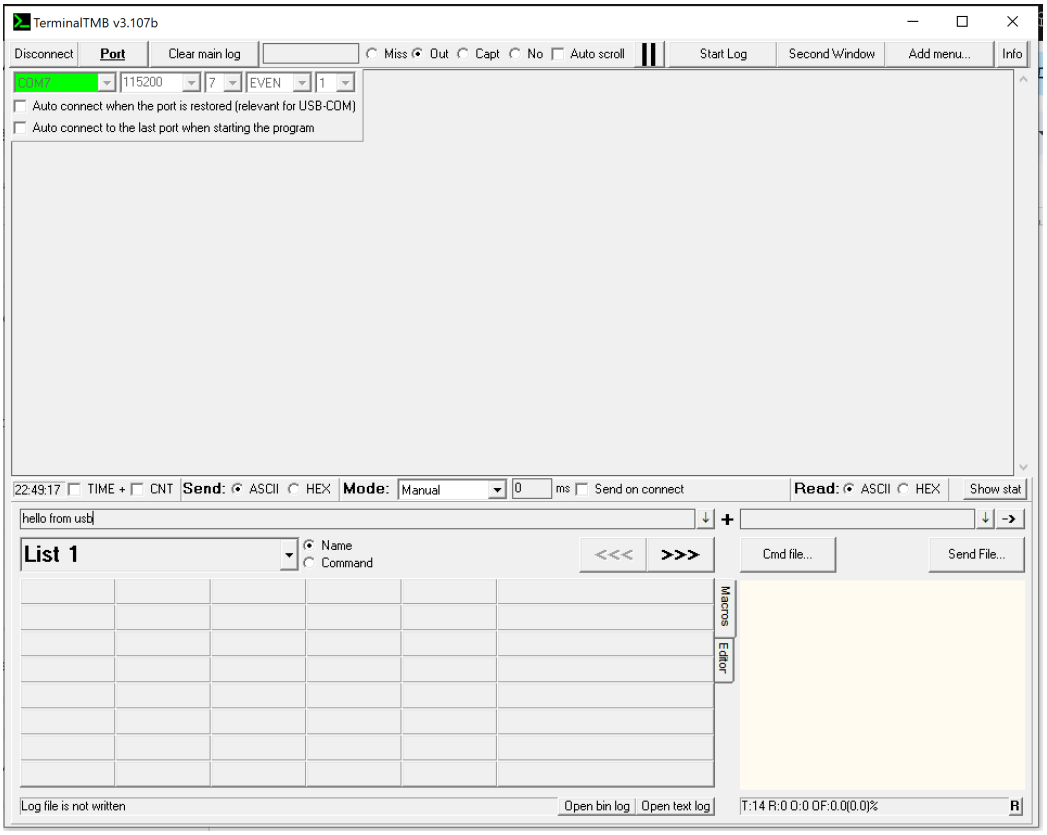


Рисунок 28 - Ввод данных с TerminalTMB, эмулирующего ввод с COM-порта

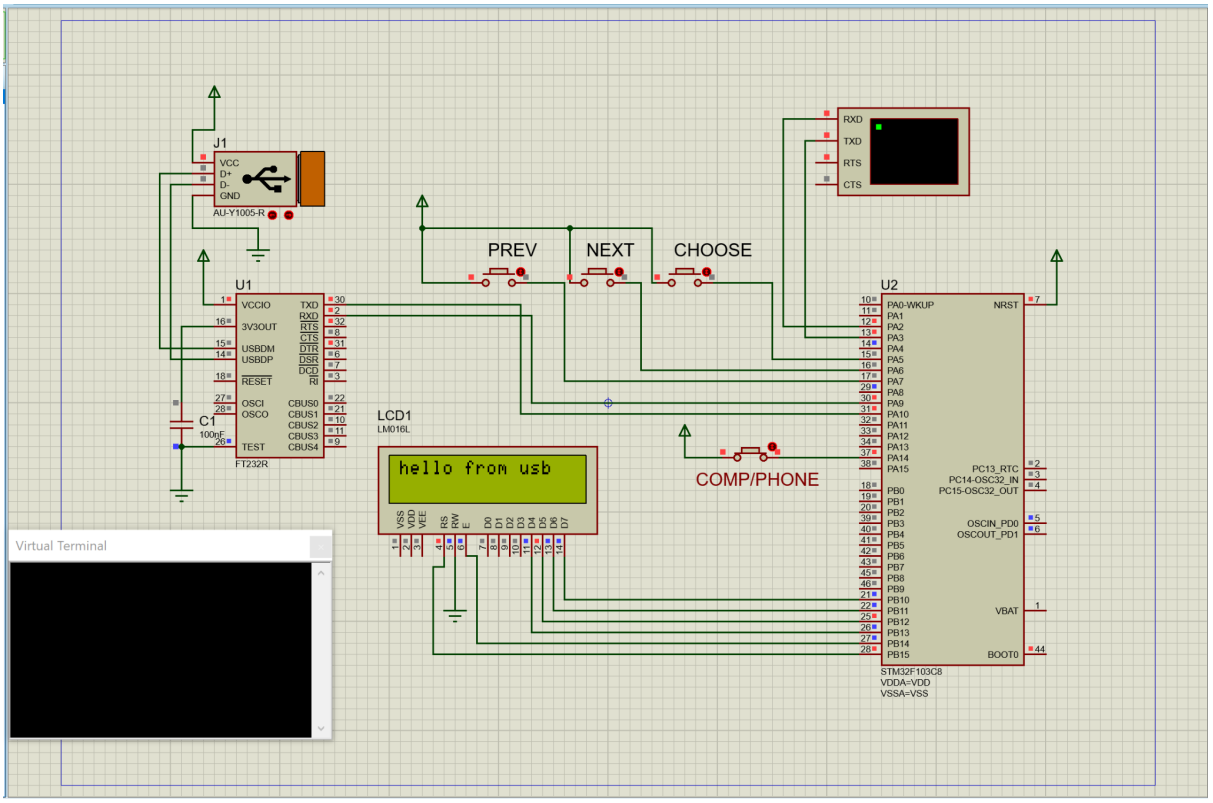


Рисунок 29 - Модель в Proteus, данные от COM-порта получены

2.3 Способы программирования МК

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

- через JTAG;
- через SWD.

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

В МК передается бинарный файл с расширением “.hex” с скомпилированной программой. Происходит это следующим образом:

подается команда RESET через пин NRST. Это используется для сброса микроконтроллера в состояние, готовое к прошивке. Затем через SWCLK идет тактовый сигнал, по которому идет запись программы в микроконтроллер через SWDIO. Этот процесс осуществляется с использованием специальных последовательностей битов (протокол SWD) для передачи команд и данных между ST-LINKv2 и микроконтроллером, обеспечивая правильную последовательность и синхронизацию для успешной прошивки или отладки микроконтроллера STM32.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработана МК-система для хэширования данных, вводимых с ПЭВМ и телефона, тремя алгоритмами на выбор - SHA256, MD5, CRC16/CCITT-FALSE. Система работает на микроконтроллере STM32F103C8, она разработана в соответствии с условиями ТЗ.

Код программы для МК написан на языке С в среде программирования STM32CubeIDE, модель отлажена и протестирована в программе Proteus 8.13.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Хартов, В.Я. Микропроцессорные системы: учеб. пособие для студ. учреждений высш. проф. образования, Академия, М., 2014. – 368с.

2. Основные семейства микроконтроллеров [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Микроконтроллер#Известные_семейства (дата обращения: 13.09.2023)

3. Документация на LM016L [Электронный ресурс]. - URL: https://pdf1.alldatasheetru.com/datasheet-pdf/view/146552/HITACHI/LM016L.html (дата обращения: 16.11.2023)

4. Документация на драйвер MAX232 [Электронный ресурс]. – URL: https://www.alldatasheet.com/datasheet-pdf/pdf/73745/MAXIM/MAX7219.html (дата обращения 27.10.2023).

5. Документация на STM32F103C8T6 [Электронный ресурс]. – URL: https://www.st.com/resource/en/reference_manual/rm0008-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf (дата обращения: 13.09.2023)

6. ГОСТ 2.710-81 Обозначения буквенно-цифровые в электрических схемах

7. ГОСТ 2.721-74 Обозначения условные графические в схемах. Обозначения общего применения.

8. ГОСТ 2.102-68 ЕСКД. Виды и комплектность конструкторских документов

9. ГОСТ 2.105-95 ЕСКД. Текстовые документы

10. Программирование на Си [Электронный ресурс]. – URL: http://www.r-5.org/files/books/computers/languages/c/kr/Brian_Kernighan_Den

nis_Ritchie-The_C_Programming_Language-RU.pdf (дата обращения 27.10.2023)

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

Приложение А

Текст программы

Заголовочные файлы

algorithm_interface.h

```
/*
 * algorithm_interface.h
 *
 *  Created on: Nov 28, 2023
 *      Author: Pizza Delivery
 */

#ifndef INC_ALGORITHM_INTERFACE_H_
#define INC_ALGORITHM_INTERFACE_H_

#include "stm32f1xx_hal.h"
#include <string.h>
#include "lcd_txt.h"

// Алгоритмы
#include "md5.h"
#include "crc16.h"
#include "sha256.h"

extern UART_HandleTypeDef huart2;
extern unsigned char *data;
extern int state;

void do_algorithm(int8_t* data, int algorithm_id);

#endif /* INC_ALGORITHM_INTERFACE_H_ */
```

crc16.h

```
/*
 * crc16.h
 *
 *  Created on: Dec 3, 2023
 *      Author: Pizza Delivery
 */

#ifndef INC_CRC16_H_
```

```
#define INC_CRC16_H_

unsigned short crc16(unsigned char *data, unsigned short len);

#endif /* INC_CRC16_H_ */
```

lcd_txt.h

```
#ifndef		__LCDTXT_H
#define		__LCDTXT_H

#include "stm32f1xx_hal.h"


/*------------- Define LCD Use -----------------*/

/*Note: Comment which not use */

#define LCD16xN //For lcd16x2 or lcd16x4
//#define LCD20xN //For lcd20x4

/*------------- Define For Connection -----------------*/

#define RS_PORT		GPIOB
#define RS_PIN		GPIO_PIN_15

#define EN_PORT		GPIOB
#define EN_PIN		GPIO_PIN_14

#define D7_PORT		GPIOB
#define D7_PIN		GPIO_PIN_10

#define D6_PORT		GPIOB
#define D6_PIN		GPIO_PIN_11

#define D5_PORT		GPIOB
#define D5_PIN		GPIO_PIN_12

#define D4_PORT		GPIOB
#define D4_PIN		GPIO_PIN_13


/*------------ Declaring Private Macro -----------------*/
```

```
#define PIN_LOW(PORT,PIN)
HAL_GPIO_WritePin(PORT,PIN,GPIO_PIN_RESET);
#define PIN_HIGH(PORT,PIN)
HAL_GPIO_WritePin(PORT,PIN,GPIO_PIN_SET);

/*------------ Declaring Function Prototype -------------*/
void lcd_init(void);
void lcd_write(int8_t type, int8_t data);
void lcd_puts(int8_t x, int8_t y, int8_t *string);
void lcd_clear(void);
#endif
```

main.h

```
 /* USER CODE BEGIN Header */
/**

******************************************************************************
***************
  * @file           : main.h
  * @brief          : Header for main.c file.
  *                   This file contains the common defines of the application.

******************************************************************************
***************
  * @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 */

/* Define to prevent recursive inclusion -----------------------------------*/
```

```
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f1xx_hal.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "uart.h"
#include "lcd_txt.h"
#include "algorithm_interface.h"
/* USER CODE END Includes */

/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */

/* USER CODE END ET */

/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */

/* USER CODE END EC */

/* Exported macro ------------------------------------------------------------*/
/* USER CODE BEGIN EM */

/* USER CODE END EM */

/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* USER CODE BEGIN EFP */

/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/

/* USER CODE BEGIN Private defines */
```

```
/* USER CODE END Private defines */

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */
```

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);

void md5String(char *input, uint8_t *result);
void md5File(FILE *file, uint8_t *result);

#endif
```

sha256.h

```
#ifndef _SHA256_H
#define _SHA256_H

#ifndef uint8
#define uint8  unsigned char
#endif
```

```
#ifndef uint32
#define uint32 unsigned long int
#endif

typedef struct
{
   uint32 total[2];
   uint32 state[8];
   uint8 buffer[64];
}
sha256_context;

void sha256_init( sha256_context *ctx );
void sha256_update( sha256_context *ctx, uint8 *input, uint32 length );
void sha256_finish( sha256_context *ctx, uint8 digest[32] );

#endif /* sha256.h */
```

Исходные файлы

algorithm_interface.c

```
/*
 * algorithm_interface.c
 *
 *  Created on: Nov 28, 2023
 *      Author: Pizza Delivery
 */
#include "algorithm_interface.h"

void do_algorithm(int8_t* data, int algorithm_id) {
        // hashing data with chosen algorithm
        state = 2;
        lcd_clear();
        lcd_init();
        switch (algorithm_id) {
                case 0:
                        lcd_init();
                        lcd_puts(0, 0, (int8_t*)"You chose sha256");
                        break;
                case 1:
```

```
                lcd_init();
                lcd_puts(0, 0, (int8_t*)"You chose md5");
                break;
        case 2:
                lcd_init();
                lcd_puts(0, 0, (int8_t*)"You chose crc16");
                break;
        default:
                lcd_init();
                int8_t *error_msg = "Unexpected algorithm chosen";
                lcd_puts(0, 0, (int8_t*) error_msg);
                HAL_Delay(1000);
                break;
    }

    HAL_Delay(1000);
    int8_t *result;
    char *output;
    int j = 0, i = 0, k = 0;
    lcd_init();
    HAL_UART_Transmit_IT(&huart2, (int8_t*)"\r\n\r\nYour data:\r\n",
15);
    HAL_Delay(100);
    HAL_UART_Transmit_IT(&huart2, (int8_t*)data, strlen(data));
    HAL_UART_Transmit_IT(&huart2, (int8_t*)"\r\n", 2);
    switch (algorithm_id) {
            case 0:
                // sha256

                lcd_clear();
                lcd_init();
                lcd_puts(0, 0, (uint8_t*)"Starting sha256");
                sha256_context foo;
                unsigned char hash[32];

                sha256_init(&foo);
                sha256_update(&foo, data, strlen(data));
                sha256_finish(&foo, hash);

                HAL_Delay(1000);
                lcd_clear();
                lcd_puts(0, 0, (uint8_t*)"Result of sha256:");
```

```
                        HAL_UART_Transmit_IT(&huart2,
(int8_t*)"\r\nResult of sha256:\r\n", 24);
                        HAL_Delay(500);

                        lcd_clear();
                        while (k < 32) {
                                if (k == 16) {
                                        HAL_Delay(2000);
                                        lcd_clear();
                                        i = 0;
                                        j = 0;
                                        HAL_UART_Transmit_IT(&huart2,
(int8_t*)"\r\n", 2);
                                }
                                sprintf(output, "%02x", hash[k]);
                                if (i == 16) {
                                        i = 0;
                                        j = 1;
                                }
                                lcd_puts(j, i, (uint8_t*)output);
                                HAL_UART_Transmit_IT(&huart2,
(int8_t*)output, 2);
                                ++k;
                                i += 2;
                        }

                        HAL_Delay(1000);
                        lcd_clear();
                        lcd_puts(0, 0, (uint8_t*)"Finished sha256");
                        HAL_Delay(1000);
                        break;
                case 1:
                        // md5
                        lcd_clear();
                        lcd_init();
                        lcd_puts(0, 0, (uint8_t*)"Starting md5");
                        uint8_t md5_result[16];
                        md5String(data, md5_result);

                        HAL_Delay(1000);
                        lcd_clear();
                        lcd_puts(0, 0, (uint8_t*)"Result of md5:");
```

```
                        HAL_UART_Transmit_IT(&huart2,
(int8_t*)"\r\nResult of md5:\r\n", 20);
                        HAL_Delay(500);

                        lcd_clear();

                        while (k < 16) {
                                sprintf((char*)output, "%02x",
md5_result[k]);
                                if (i == 16) {
                                        i = 0;
                                        j = 1;
                                }

                                lcd_puts(j, i, (uint8_t*)output);
                                HAL_UART_Transmit_IT(&huart2,
(int8_t*)output, 2);
                                k++;
                                i += 2;
                        }

                        HAL_Delay(1000);
                        lcd_clear();
                        lcd_puts(0, 0, (uint8_t*)"Finished md5");
                        break;
                case 2:
                        // crc16

                        lcd_clear();
                        lcd_init();
                        lcd_puts(0, 0, (uint8_t*)"Starting crc16");
                        result = crc16(data, strlen(data));

                        HAL_Delay(1000);
                        lcd_clear();
                        lcd_puts(0, 0, (uint8_t*)"Result of crc16:");
                        HAL_UART_Transmit_IT(&huart2,
(int8_t*)"\r\nResult of crc16:\r\n", 23);
                        HAL_Delay(500);

                        sprintf(output, "%hX", result);
                        HAL_Delay(1000);
```

```
                        lcd_clear();
                        lcd_puts(0, 0, output);
                        HAL_UART_Transmit_IT(&huart2,
(int8_t*)output, 4);

                        HAL_Delay(1000);
                        lcd_clear();
                        lcd_puts(0, 0, (uint8_t*)"Finished crc16");
                        break;
                default:
                        lcd_init();
                        int8_t *error_msg = "Unexpected error in
algorithms\r\n";
                        lcd_puts(0, 0, (int8_t*) error_msg);
                        HAL_Delay(1000);
                        break;
            }
        HAL_UART_Transmit_IT(&huart2, (int8_t*)"\r\n\r\n", 4);
        HAL_Delay(1000);
        state = 0;
        lcd_clear();
}
```

crc16.c

```
/*
  Name  : CRC-16 CCITT
  Poly  : 0x1021    x^16 + x^12 + x^5 + 1
  Init  : 0xFFFF
  Revert: false
  XorOut: 0x0000
  Check : 0x29B1 ("123456789")
  MaxLen: 4095 байт (32767 бит) - обнаружение
    одинарных, двойных, тройных и всех нечетных ошибок
*/
unsigned short crc16(unsigned char *data, unsigned short len)
{
    unsigned short crc = 0xFFFF;
    unsigned char i;

    while (len--)
    {
        crc ^= *data++ << 8;
```

```
        for (i = 0; i < 8; i++)
            crc = crc & 0x8000 ? (crc << 1) ^ 0x1021 : crc << 1;
    }
    return crc;
}
```

lcd_txt.c

```
#include "lcd_txt.h"



/*--------------- Initialize LCD ------------------*/
void lcd_init(void)
{

        HAL_Delay(30);

        PIN_LOW(D4_PORT,D4_PIN);
        PIN_HIGH(D5_PORT,D5_PIN);
        PIN_LOW(D6_PORT,D6_PIN);
        PIN_LOW(D7_PORT,D7_PIN);
        PIN_LOW(RS_PORT,RS_PIN);

        PIN_HIGH(EN_PORT,EN_PIN);
        PIN_LOW(EN_PORT,EN_PIN);

        lcd_write(0,0x28);
        lcd_write(0,0x0c);
        lcd_write(0,0x06);
        lcd_write(0,0x01);
}

/*--------------- Write To LCD ---------------*/
void lcd_write(int8_t type, int8_t data)
{
        HAL_Delay(2);
        if(type)
        {
                PIN_HIGH(RS_PORT,RS_PIN);
        }else
        {
```

```
        PIN_LOW(RS_PORT,RS_PIN);
}

//Send High Nibble
if(data&0x80)
{
        PIN_HIGH(D7_PORT,D7_PIN);
}else
{
        PIN_LOW(D7_PORT,D7_PIN);
}

if(data&0x40)
{
        PIN_HIGH(D6_PORT,D6_PIN);
}else
{
        PIN_LOW(D6_PORT,D6_PIN);
}

if(data&0x20)
{
        PIN_HIGH(D5_PORT,D5_PIN);
}else
{
        PIN_LOW(D5_PORT,D5_PIN);
}

if(data&0x10)
{
        PIN_HIGH(D4_PORT,D4_PIN);
}else
{
        PIN_LOW(D4_PORT,D4_PIN);
}
PIN_HIGH(EN_PORT,EN_PIN);
PIN_LOW(EN_PORT,EN_PIN);


//Send Low Nibble
if(data&0x08)
{
```

```
            PIN_HIGH(D7_PORT,D7_PIN);
        }else
        {
            PIN_LOW(D7_PORT,D7_PIN);
        }

        if(data&0x04)
        {
            PIN_HIGH(D6_PORT,D6_PIN);
        }else
        {
            PIN_LOW(D6_PORT,D6_PIN);
        }

        if(data&0x02)
        {
            PIN_HIGH(D5_PORT,D5_PIN);
        }else
        {
            PIN_LOW(D5_PORT,D5_PIN);
        }

        if(data&0x01)
        {
            PIN_HIGH(D4_PORT,D4_PIN);
        }else
        {
            PIN_LOW(D4_PORT,D4_PIN);
        }
        PIN_HIGH(EN_PORT,EN_PIN);
        PIN_LOW(EN_PORT,EN_PIN);
}

void lcd_puts(int8_t x, int8_t y, int8_t *string)
{
        //Set Cursor Position
        #ifdef LCD16xN    //For LCD16x2 or LCD16x4
        switch(x)
        {
            case 0: //Row 0
                    lcd_write(0,0x80+0x00+y);
                    break;
```

```
                case 1: //Row 1
                        lcd_write(0,0x80+0x40+y);
                        break;
                case 2: //Row 2
                        lcd_write(0,0x80+0x10+y);
                        break;
                case 3: //Row 3
                        lcd_write(0,0x80+0x50+y);
                        break;
        }
        #endif

        #ifdef LCD20xN    //For LCD20x4
        switch(x)
        {
                case 0: //Row 0
                        lcd_write(0,0x80+0x00+y);
                        break;
                case 1: //Row 1
                        lcd_write(0,0x80+0x40+y);
                        break;
                case 2: //Row 2
                        lcd_write(0,0x80+0x14+y);
                        break;
                case 3: //Row 3
                        lcd_write(0,0x80+0x54+y);
                        break;
        }
        #endif

        while(*string)
        {
                lcd_write(1,*string);
                string++;
        }
}
void lcd_clear(void)
{
        lcd_write(0,0x01);
}
```

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"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* 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 ---------------------------------------------------------*/
UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
uint8_t UART1_rxBuffer[26] = {0};
int state = 0; // 0 - input, 1 - choose alg
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_USART1_UART_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
int8_t* chosen_algorithm(int id) {
	switch (id) {
		case 0:
			return "sha256";
		case 1:
			return "md5";
		case 2:
			return "crc16";
		default:
			return "Invalid algorithm";
			break;
	}
}
/* 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_USART2_UART_Init();
  MX_USART1_UART_Init();
  /* USER CODE BEGIN 2 */

//  HAL_UART_Transmit_IT(&huart1, "hi", 2);
  HAL_Delay(1000);
  int8_t *output, *data = NULL;
  int alg_id = 0, not_asked_for_input = 1;
  lcd_init();
  /* USER CODE END 2 */

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

```
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
        /* scrolling through algorithms
                        ids: 0 - sha, 1 - md, 2 - crc
                */
        if (state == 0) {
                if (not_asked_for_input == 1) {
                        lcd_puts(0, 0, "Input data");
                        not_asked_for_input = 0;
                        data = (int8_t*)calloc(1000, sizeof(int8_t));
                }

                // Input place
                switch (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_14)) {
                        case 0: // computer
                                HAL_UART_Receive(&huart2, (int8_t*)data, 1000,
3000);
                        case 1: // phone
                                HAL_UART_Receive_IT(&huart1, (int8_t*)data,
1000);
                }


                if (data && strlen(data) != 0) {
                        lcd_clear();
                        lcd_puts(0, 0, data);
                        HAL_Delay(1000);
                        lcd_clear();
                        state = 1;
                }
        } else if (state == 1) {
                if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_7) == 1) {
                        if (alg_id < 2) {
                                alg_id++;
                        } else {
                                alg_id = 0;
                        }
                        lcd_clear();
                } else if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_6) == 1) {
                        if (alg_id > 0) {
                                alg_id--;
```

```
                        } else {
                                alg_id = 2;
                        }
                        lcd_clear();
                } else if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_5) == 1) {
                        lcd_clear();
                        output = chosen_algorithm(alg_id);
                        lcd_puts(0, 0, (int8_t*) output);
                        lcd_clear();
                        do_algorithm(data, alg_id);
                        not_asked_for_input = 1;
                        data = NULL;
                }

                if (state != 0) {
                        output = chosen_algorithm(alg_id);
                        lcd_puts(0, 0, (int8_t*) output);
                        HAL_Delay(200);
                }
        } else if (state != 2) {
                lcd_clear();
                lcd_init();
                lcd_puts(0, 0, "Unexpected state");
        }
  }
  /* 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 USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
```

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

  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief 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_11|GPIO_PIN_12|GPIO_PIN_13
                    |GPIO_PIN_14|GPIO_PIN_15, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_14, GPIO_PIN_RESET);

  /*Configure GPIO pins : PB10 PB11 PB12 PB13
                         PB14 PB15 */
  GPIO_InitStruct.Pin =
GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13
                    |GPIO_PIN_14|GPIO_PIN_15;
  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 pin : PA14 */
```

```
  GPIO_InitStruct.Pin = GPIO_PIN_14;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

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

/* USER CODE BEGIN 4 */

/* 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 */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
```

```
number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

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 "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;
}
```

```
/*
 * Functions that run the algorithm on the provided input and put the digest
into result.
 * result should be able to store 16 bytes.
 */
void md5String(char *input, uint8_t *result){
   MD5Context ctx;
   md5Init(&ctx);
   md5Update(&ctx, (uint8_t *)input, strlen(input));
   md5Finalize(&ctx);

   memcpy(result, ctx.digest, 16);
}

void md5File(FILE *file, uint8_t *result){
   char *input_buffer = malloc(1024);
   size_t input_size = 0;

   MD5Context ctx;
   md5Init(&ctx);

   while((input_size = fread(input_buffer, 1, 1024, file)) > 0){
      md5Update(&ctx, (uint8_t *)input_buffer, input_size);
   }

   md5Finalize(&ctx);

   free(input_buffer);

   memcpy(result, ctx.digest, 16);
}
```

sha256.c

```
/*
 *  FIPS-180-2 compliant SHA-256 implementation
 *
 *  Copyright (C) 2001-2003  Christophe Devine
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
```

```
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307
USA
 */

#include <string.h>

#include "sha256.h"

#define GET_UINT32(n,b,i)                       \
{                                               \
    (n) = ( (uint32) (b)[(i)    ] << 24 )       \
        | ( (uint32) (b)[(i) + 1] << 16 )       \
        | ( (uint32) (b)[(i) + 2] <<  8 )       \
        | ( (uint32) (b)[(i) + 3]       );      \
}

#define PUT_UINT32(n,b,i)                       \
{                                               \
    (b)[(i)    ] = (uint8) ( (n) >> 24 );       \
    (b)[(i) + 1] = (uint8) ( (n) >> 16 );       \
    (b)[(i) + 2] = (uint8) ( (n) >>  8 );       \
    (b)[(i) + 3] = (uint8) ( (n)       );       \
}

void sha256_init( sha256_context *ctx )
{
    ctx->total[0] = 0;
    ctx->total[1] = 0;

    ctx->state[0] = 0x6A09E667;
    ctx->state[1] = 0xBB67AE85;
    ctx->state[2] = 0x3C6EF372;
    ctx->state[3] = 0xA54FF53A;
```

```
    ctx->state[4] = 0x510E527F;
    ctx->state[5] = 0x9B05688C;
    ctx->state[6] = 0x1F83D9AB;
    ctx->state[7] = 0x5BE0CD19;
}

void sha256_process( sha256_context *ctx, uint8 data[64] )
{
    uint32 temp1, temp2, W[64];
    uint32 A, B, C, D, E, F, G, H;

    for (int i = 0; i < 16; ++i) {
        GET_UINT32( W[i],  data,  i * 4 );
        }

#define  SHR(x,n) ((x & 0xFFFFFFFF) >> n)
#define ROTR(x,n) (SHR(x,n) | (x << (32 - n)))

#define S0(x) (ROTR(x, 7) ^ ROTR(x,18) ^  SHR(x, 3))
#define S1(x) (ROTR(x,17) ^ ROTR(x,19) ^  SHR(x,10))

#define S2(x) (ROTR(x, 2) ^ ROTR(x,13) ^ ROTR(x,22))
#define S3(x) (ROTR(x, 6) ^ ROTR(x,11) ^ ROTR(x,25))

#define F0(x,y,z) ((x & y) | (z & (x | y)))
#define F1(x,y,z) (z ^ (x & (y ^ z)))

#define R(t)                                    \
(                                               \
    W[t] = S1(W[t -  2]) + W[t -  7] +          \
        S0(W[t - 15]) + W[t - 16]               \
)

#define P(a,b,c,d,e,f,g,h,x,K)                  \
{                                               \
    temp1 = h + S3(e) + F1(e,f,g) + K + x;      \
    temp2 = S2(a) + F0(a,b,c);                  \
    d += temp1; h = temp1 + temp2;              \
}

    A = ctx->state[0];
        B = ctx->state[1];
```

```
C = ctx->state[2];
D = ctx->state[3];
E = ctx->state[4];
F = ctx->state[5];
G = ctx->state[6];
H = ctx->state[7];

P( A, B, C, D, E, F, G, H, W[ 0], 0x428A2F98 );
P( H, A, B, C, D, E, F, G, W[ 1], 0x71374491 );
P( G, H, A, B, C, D, E, F, W[ 2], 0xB5C0FBCF );
P( F, G, H, A, B, C, D, E, W[ 3], 0xE9B5DBA5 );
P( E, F, G, H, A, B, C, D, W[ 4], 0x3956C25B );
P( D, E, F, G, H, A, B, C, W[ 5], 0x59F111F1 );
P( C, D, E, F, G, H, A, B, W[ 6], 0x923F82A4 );
P( B, C, D, E, F, G, H, A, W[ 7], 0xAB1C5ED5 );
P( A, B, C, D, E, F, G, H, W[ 8], 0xD807AA98 );
P( H, A, B, C, D, E, F, G, W[ 9], 0x12835B01 );
P( G, H, A, B, C, D, E, F, W[10], 0x243185BE );
P( F, G, H, A, B, C, D, E, W[11], 0x550C7DC3 );
P( E, F, G, H, A, B, C, D, W[12], 0x72BE5D74 );
P( D, E, F, G, H, A, B, C, W[13], 0x80DEB1FE );
P( C, D, E, F, G, H, A, B, W[14], 0x9BDC06A7 );
P( B, C, D, E, F, G, H, A, W[15], 0xC19BF174 );
P( A, B, C, D, E, F, G, H, R(16), 0xE49B69C1 );
P( H, A, B, C, D, E, F, G, R(17), 0xEFBE4786 );
P( G, H, A, B, C, D, E, F, R(18), 0x0FC19DC6 );
P( F, G, H, A, B, C, D, E, R(19), 0x240CA1CC );
P( E, F, G, H, A, B, C, D, R(20), 0x2DE92C6F );
P( D, E, F, G, H, A, B, C, R(21), 0x4A7484AA );
P( C, D, E, F, G, H, A, B, R(22), 0x5CB0A9DC );
P( B, C, D, E, F, G, H, A, R(23), 0x76F988DA );
P( A, B, C, D, E, F, G, H, R(24), 0x983E5152 );
P( H, A, B, C, D, E, F, G, R(25), 0xA831C66D );
P( G, H, A, B, C, D, E, F, R(26), 0xB00327C8 );
P( F, G, H, A, B, C, D, E, R(27), 0xBF597FC7 );
P( E, F, G, H, A, B, C, D, R(28), 0xC6E00BF3 );
P( D, E, F, G, H, A, B, C, R(29), 0xD5A79147 );
P( C, D, E, F, G, H, A, B, R(30), 0x06CA6351 );
P( B, C, D, E, F, G, H, A, R(31), 0x14292967 );
P( A, B, C, D, E, F, G, H, R(32), 0x27B70A85 );
P( H, A, B, C, D, E, F, G, R(33), 0x2E1B2138 );
P( G, H, A, B, C, D, E, F, R(34), 0x4D2C6DFC );
```

```
        P( F, G, H, A, B, C, D, E, R(35), 0x53380D13 );
        P( E, F, G, H, A, B, C, D, R(36), 0x650A7354 );
        P( D, E, F, G, H, A, B, C, R(37), 0x766A0ABB );
        P( C, D, E, F, G, H, A, B, R(38), 0x81C2C92E );
        P( B, C, D, E, F, G, H, A, R(39), 0x92722C85 );
        P( A, B, C, D, E, F, G, H, R(40), 0xA2BFE8A1 );
        P( H, A, B, C, D, E, F, G, R(41), 0xA81A664B );
        P( G, H, A, B, C, D, E, F, R(42), 0xC24B8B70 );
        P( F, G, H, A, B, C, D, E, R(43), 0xC76C51A3 );
        P( E, F, G, H, A, B, C, D, R(44), 0xD192E819 );
        P( D, E, F, G, H, A, B, C, R(45), 0xD6990624 );
        P( C, D, E, F, G, H, A, B, R(46), 0xF40E3585 );
        P( B, C, D, E, F, G, H, A, R(47), 0x106AA070 );
        P( A, B, C, D, E, F, G, H, R(48), 0x19A4C116 );
        P( H, A, B, C, D, E, F, G, R(49), 0x1E376C08 );
        P( G, H, A, B, C, D, E, F, R(50), 0x2748774C );
        P( F, G, H, A, B, C, D, E, R(51), 0x34B0BCB5 );
        P( E, F, G, H, A, B, C, D, R(52), 0x391C0CB3 );
        P( D, E, F, G, H, A, B, C, R(53), 0x4ED8AA4A );
        P( C, D, E, F, G, H, A, B, R(54), 0x5B9CCA4F );
        P( B, C, D, E, F, G, H, A, R(55), 0x682E6FF3 );
        P( A, B, C, D, E, F, G, H, R(56), 0x748F82EE );
        P( H, A, B, C, D, E, F, G, R(57), 0x78A5636F );
        P( G, H, A, B, C, D, E, F, R(58), 0x84C87814 );
        P( F, G, H, A, B, C, D, E, R(59), 0x8CC70208 );
        P( E, F, G, H, A, B, C, D, R(60), 0x90BEFFFA );
        P( D, E, F, G, H, A, B, C, R(61), 0xA4506CEB );
        P( C, D, E, F, G, H, A, B, R(62), 0xBEF9A3F7 );
        P( B, C, D, E, F, G, H, A, R(63), 0xC67178F2 );

        ctx->state[0] += A;
        ctx->state[1] += B;
        ctx->state[2] += C;
        ctx->state[3] += D;
        ctx->state[4] += E;
        ctx->state[5] += F;
        ctx->state[6] += G;
        ctx->state[7] += H;
}

void sha256_update( sha256_context *ctx, uint8 *input, uint32 length )
{
```

```
    uint32 left, fill;

    if( ! length ) return;

    left = ctx->total[0] & 0x3F;
    fill = 64 - left;

    ctx->total[0] += length;
    ctx->total[0] &= 0xFFFFFFFF;

    if( ctx->total[0] < length )
        ctx->total[1]++;

    if( left && length >= fill )
    {
        memcpy( (void *) (ctx->buffer + left),
                (void *) input, fill );
        sha256_process( ctx, ctx->buffer );
        length -= fill;
        input  += fill;
        left = 0;
    }

    while( length >= 64 )
    {
        sha256_process( ctx, input );
        length -= 64;
        input  += 64;
    }

    if( length )
    {
        memcpy( (void *) (ctx->buffer + left),
                (void *) input, length );
    }
}

static uint8 sha256_padding[64] =
{
 0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

void sha256_finish( sha256_context *ctx, uint8 digest[32] )
{
    uint32 last, padn;
    uint32 high, low;
    uint8 msglen[8];

    high = ( ctx->total[0] >> 29 )
         | ( ctx->total[1] <<  3 );
    low  = ( ctx->total[0] <<  3 );

    PUT_UINT32( high, msglen, 0 );
    PUT_UINT32( low,  msglen, 4 );

    last = ctx->total[0] & 0x3F;
    padn = ( last < 56 ) ? ( 56 - last ) : ( 120 - last );

    sha256_update( ctx, sha256_padding, padn );
    sha256_update( ctx, msglen, 8 );

    for (int i = 0; i < 8; ++i) {
        PUT_UINT32( ctx->state[i], digest,  i * 4 );
    }
}
```

Приложение Б

Перечень элементов

На 2 листах